Appearing in *IEEE International Conference on Field-Programmable Technology* (FPT 2004), December 6–8, 2004

A Greedy Algorithm for Tolerating Defective Crosspoints in NanoPLA Design

Helia Naeimi
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125
<helia@caltech.edu>

André DeHon
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125
<andre@cs.caltech.edu>

Abstract

Recent developments suggest both plausible fabrication techniques and viable architectures for building sublithographic Programmable Logic Arrays using molecular-scale wires and switches. Designs at this scale will see much higher defect rates than in conventional lithography. However, these defects need not be an impediment to programmable logic design as this scale. We introduce a strategy for tolerating defective crosspoints and develop a linear-time, greedy algorithm for mapping PLA logic around crosspoint defects. We note that P-term fanin must be bounded to guarantee low overhead mapping and develop analytical guidelines for bounding fanin. We further quantify analytical and empirical mapping overhead rates. Including fanin bounding, our greedy mapping algorithm maps a large set of benchmark designs with 13% average overhead for random junction defect rates as high as 20%.

1. Introduction

Recent work shows how to build nanoscale Programmable Logic Arrays (nanoPLAs) using the bottom-up synthesis techniques being developed by physical chemists [1] [2] [3]. With these bottom-up techniques, it is possible to build features (*e.g.* wires and programmable junctions) without relying on lithography. As such, these techniques provide a path to continue the advance of field-programmable technology beyond the end of the traditional, lithographic roadmap (*e.g.* [4]).

Nonetheless, nanoscale features, both in the sublithographic and lithographic arenas, come with a new set of challenges. Notably, as devices become smaller, they are constructed from fewer and fewer atoms and molecules. Since individual atoms behave statistically, this means we have higher variance in the shape and makeup of our devices, and a higher likelihood that devices are simply unusable. Designs at this scale **must** be defect tolerant. This, and other aspects of sublithographic assembly techniques, suggest that all devices we build at these scales will be reconfigurable.

Hewlett-Packard has recently demonstrated an 8×8 crossbar using molecular switches at the crosspoints [5]. In the HP crossbar, they observed that 85% of the crosspoint junctions were programmable (15% were defective). The HP crossbar is an early laboratory prototype, and we expect these defect rates to decrease. Nonetheless, we are unlikely to achieve 100% crosspoint yield at this scale using these kinds of bottom-up, statistical fabrication techniques. If defects are randomly distributed, at a 15% crosspoint defect rate, essentially every row and column in a 100×100 crosspoint array will contain a defective junction.

With the techniques in this paper, we show that nanoPLA arrays with a 20% crosspoint defect rate are still usable with modest (13% including fanin bounding) overhead. That is, despite the fact that no rows or columns are free of defective junctions, we can still make use of more than 90% of the nanowires. Snider *et al.* have also looked at defect tolerant mapping using a similar defect model and shown that a 4-bit microprocessor can tolerate defect rates up to 20% [6].

This defect mapping must be applied on a per-array basis. That is, each nanoPLA will have a unique defect pattern. Since nanoPLAs are a few microns tall and 10–20 microns wide [2], we can easily have millions of these nanoPLAs on a modest die. Consequently, it is important that we minimize the time required to map around defects. To this end, we introduce a linear-time, greedy mapping algorithm for assigning logical P-terms to physical nanowires avoiding defective junctions in a fabricated nanoPLA.

Novel contributions of this work include:

- Formulation of defective crosspoint mapping problem for nanoPLAs
- Introduction of simple, greedy algorithm for linear-time mapping around defects
- Analytical estimates on mapping times

- Analytical identification of bounds on P-term fanin driven by array size and defect rate
- Empirical and analytical characterization of mapping overhead for our proposed algorithm

In the next section, we review the emerging, bottom-up fabrication techniques for nanowires and crosspoints and the architectural building blocks for restoration and nanoscale addressing. We then review the nanoPLA architecture (Section 3). In Section 4, we introduce our defect model. Section 5 formulates the problem and introduce the basic idea for the solution. Section 6 reviews exact algorithms to solve the identified mapping problem and develops our linear-time heuristic algorithms. In Section 7, we analyze the algorithms based on expected case behavior and derive bounds for input fanin (Section 8). Section 9 provides experimental results which ground and confirm the analysis.

2. Substrate

Nanowire We can grow nanowires to controlled dimensions on the nanometer scale using seed catalysts to define their diameter. Nanowires with diameters down to 3nm have been demonstrated [7]. With suitable doping, conduction through nanowires can be controlled by an applied electrical field like Field-Effect Transistors [8]. Techniques have been demonstrated to align a set of nanowires into a single orientation, close pack them, and transfer them onto a surface. This step can be repeated and rotated by 90 degrees so that we get multiple layers of nanowires [9].

Programmable Crosspoints Over the past few years, many technologies have been demonstrated for molecular-scale memories. So far, they all seem to have: (1) resistance which changes significantly between ON and OFF states, (2) the ability to be made rectifying, and (3) the ability to turn the device ON or OFF by applying a voltage differential across the junction. UCLA and HP have demonstrated a number of molecules which exhibit hysteresis [10].HP has demonstrated an 8×8 programmable crossbar and observed that they could force an order of magnitude resistance difference between ON and OFF state junctions [5].

Restoring Crosspoint Programmable diode crosspoints in a crossbar array give us a programmable OR array (See Section 3 for more detail). Diodes alone do not give us cascadable logic. To achieve restoration, these programmable diode stages can be followed by dedicated, nonprogrammable restoring stages. The restoring stages can also provide selective inversion. DeHon and Wilson describe how to build a nanoscale nonprogrammable restoring stage using a stochastic assembly of nanowires with doping profiles [2] [11].

Lithographic-scale Address Decoder The pitch of the nanowires can be much smaller than our lithographic patterning. We will be using the crosspoint programmability to configure logic functions into our nanoscale devices. In order to do this, we need a way to selectively place a defined voltage on a single row and column wire in order to set the state of the crosspoint. By constructing nanowires with doping profiles on their ends [11] [12], we can give each nanowire an address (See left end of Figure 1(a)). The dimensions of the address bit control regions can be set to the lithographic pitch so that a set of crossed, lithographic wires can be used to address a single nanowire. Detailed information of this addressing scheme can be found in [12] and [2].

3. NanoPLA Architecture

NanoPLAs, like conventional PLAs, consist of two programmable NOR planes (Figure 1(a)). Each of the NOR planes consists of two arrays: logic array and buffer/inverter array.

The logic array is the programmable part of each NOR plane. Its junctions are the bistable crosspoints described in Section 2. The logic array implements the OR function of its inputs, which is why the outputs of this array are called OR-terms. Each of the connected junctions behaves like a diode, and each OR-term is the wired OR logic of its inputs. The output of each OR-term is pulled down weakly. If any of the inputs is high, then it pulls up the OR-term output (Figure 1(b)).

The two states of the logic array junctions are: 1) connected via a PN junction, 2) disconnected. If an input participates in an OR function, the junction of that input and the OR-term nanowire representing that function will be programmed "closed"; the junction will be left "open" when the input is not in the OR function. The junctions are initially in the "open" state. To program a junction "closed" a high voltage difference is applied to the nanowires that cross the junction. To change the junction state back to "open", we apply the opposite voltage polarity by switching the place of the low voltage and high voltage [5].

The second part of the NOR plane is the buffer/inverter array. This array restores the input signals using the restoring, nonprogrammable junctions (Section 2). The OR-term signals can be selectively inverted or buffered in this array. So the result of the NOR plane is either a NOR function or an OR function [2].

The restored outputs of the top NOR plane can be the inputs of the bottom NOR plane; and vice versa; *e.g.* a 4-level logic can be implemented by rotating the signals through the two NOR planes for 2 rounds [2].

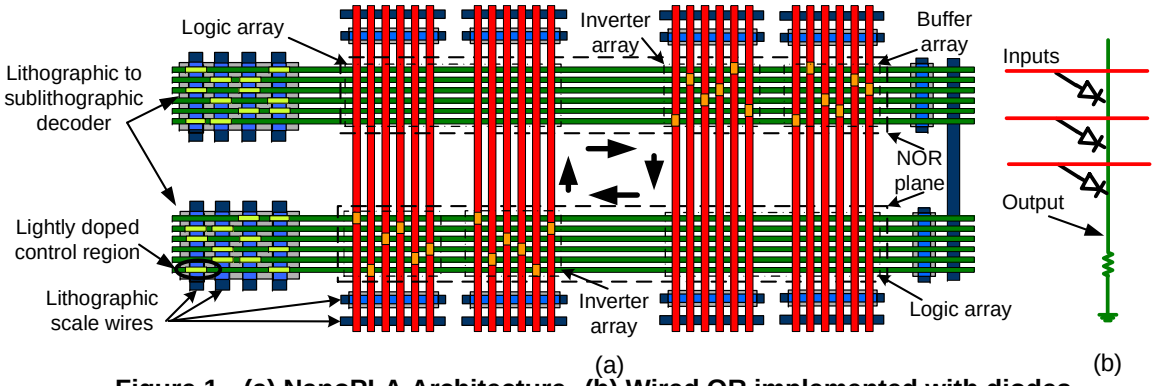


Figure 1. (a) NanoPLA Architecture. (b) Wired OR implemented with diodes.

4. Defect Model

In this section we discuss possible defects in the nanoPLAs and the defect model used in this paper. The two more probable defects that we focus on here are: 1) Defects in programmable crosspoints, 2) Defects in nanowires. The defective nanowires can be easily detected with the procedure suggested in [2]. The time required to test the nanowires of each array is linear in the code space size of the stochastic address decoder. The defect models are broken nanowires, stuck-at-0 and stuck-at-1.

Defects in programmable crosspoints are due to the structure of the junctions, which is a sandwich of bistable molecules between two layers of nanowires. In each crosspoint there are only a few molecules. For example, nanowires of width 5nm having cross sectional area of 25nm^2 can hold about 18 molecules [5]. (In [5] they have different active area size. Therefore here the number of molecules is scaled accordingly.) The programmability of a crosspoint comes from the bistable attribute of the molecules located in the crosspoint area. If there are too few molecules at the crosspoint then the junction may never be able to be programmed "closed", or the "closed" state may have higher resistance than the designed threshold chosen for correct operation and timing of the PLA.

We abstract this into a simple crosspoint defect model. Crosspoints will be in one of two states:

- **programmable** – crosspoint can be programmed into both a "closed" state and an "open" state.
- **non-programmable** – crosspoint cannot be programmed into an adequate "closed" state, but can be set into a suitable "open" state.

Crosspoints which cannot be programmed into a suitable "open" state will result in the entire horizontal and vertical nanowires being unusable. We treat these as nanowire defects rather than junction defects. Based on the physical model suggested above and discussion with physical scientists, we expect these defects which "short" horizontal and vertical nanowires to be much less likely and, consequently, believe it is reasonable to treat them as wire defects.

5. Problem Statement

5.1. Overview

To implement a specific circuit on a nanoPLA, we program up the logic arrays. This means that each OR function of a design will be mapped to an OR-term nanowire.

For clarity, we define the following terminology. The *logical inputs* are the set of inputs to the OR functions. The logical inputs includes the primary inputs of the nanoPLA and the signals that are fed back from the other NOR plane. In each OR function the set of logical inputs that participate in the OR function is called *ON-inputs* and those that do not participate are called *OFF-inputs*.

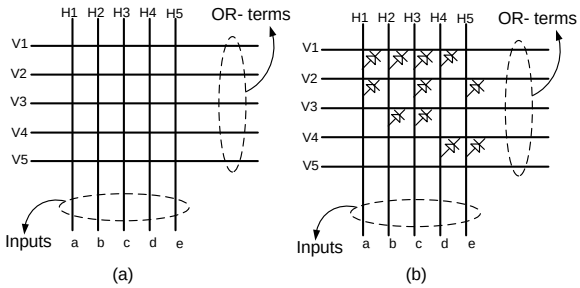


Figure 2. (a) A logic array of a nanoPLA. (b) Programmed logic array.

Henceforth we assume that the input nanowires of logic arrays are previously assigned to the logical inputs; and order of the logical inputs is preserved. This assumption lets us use the same programming process for all the inputs irrespective of whether they are primary inputs or intermediate signals. Intermediate signals do not have full freedom in placement because they are the OR-terms of the previous logic array and they may already be programmed and assigned to fixed location.

To map each OR function to an OR-term nanowire, the crosspoints of the OR-term nanowire associated with the ON-inputs of the OR function are programmed "closed", and crosspoints of OFF-inputs are left "open". Figure 2 shows an example of mapping four OR functions, $f_1 = a + b + c + d$,

$f_2 = a + c + e$, $f_3 = b + c$, and $f_4 = d + e$, with logical inputs, a, b, c, d, and e. The logic array inputs a to e are assigned to input nanowires $H1$ to $H5$, respectively. In the case like Figure 2(b) where there is no defect in the array, each OR function can be mapped to any nanowire. Here OR functions 1 to 4 are mapped to nanowires $V1$ to $V4$ respectively.

5.2. Challenge

Logic arrays may contain defective junctions that cannot be programmed closed, as described in Section 4. An OR function can be assigned to a physical OR-term nanowire if and only if each of the ON-inputs of the OR function has a corresponding programmable junction on the physical OR-term nanowire.

If a logic array of a nanoPLA has defective junctions as marked in Figure 4(a), then the OR function $a+c+e$ cannot be assigned to nanowires $w1$ or $w2$ because junctions $(w1, c)$, $(w2, c)$ and $(w2, e)$ are non-programmable, but it can be assigned to nanowires $w3$, $w4$, and $w5$. Although the nanowires $w1$ and $w2$ cannot implement the OR function $a + c + e$ they are still useful for some other OR functions such as $b + d$.

In spite of having defective junctions in a nanowire, some OR functions can be successfully mapped to that nanowire. The challenge is to find an assignment of the OR functions to the OR-term nanowires. Our key question is: *How do we perform this assignment with a small number of spare nanowires and in reasonable running time?*

5.3. Idea

In each OR function there are always some OFF-inputs, *i.e.* some of the junctions will always be left open. If there is a nanowire with defective junctions only at a subset of those positions, then this defective nanowire can be successfully assigned to the OR function.

Let F be the set of OR functions and W be the set of physical OR-term nanowires. The problem is finding an assignment of OR functions to the nanowires. This problem can be formally stated as finding a *bipartite matching* from the set F to the set W.

Definition of Bipartite Matching In a bipartite graph $G(V_1, V_2, E)$, the set $M \subset E$ is a *matching* from V_1 to V_2 if and only if the following conditions hold:

$\underset{u \in V_1}{\forall}$, exists exactly one $v \in V_2$, s.t. $(u, v) \in M$.
and

$\underset{v \in V_2}{\forall}$, exists at most one $u \in V_1$, s.t. $(u, v) \in M$.
Here $V_1 = F$ and $V_2 = W$ and E is defined below.

```
    \* General heuristic matching algorithm *\
1   While F is not empty
2       Choose a node f_i ∈ F
3       While ((f_i is not matched) and
          (W has non-visited by f_i vertex) )
4           Choose a node w_j ∈ W
5           If (f_i, w_j) ∈ E
6               Mark(f_i, w_j) as match,
7               Remove f_i from F and w_j from W
8           Else
9               Set w_j visited by f_i
10      EndWhile
11  EndWhile
```

(a)

```
    \* The algorithm used in this paper*\
1   Order the elements in F in decreasing order of c_i
2   While F is not empty
3       Choose the first f_i ∈ F
4       While (f_i is not matched) and
          (W has non-visited by f_i vertex)
5           Choose a random w_j ∈ W
6           If (∀_{k,I_{i,k}=1}, (J_{j,k} == 1)) \* try programming
                                  all the c_i's crosspoints*\
7               Mark(f_i, w_j) as match,
8               Remove f_i from F and w_j from W
9           Else
10              Set w_j visited by f_i
11      EndWhile
12  EndWhile
```

(b)

Figure 3. The algorithm frameworks.

5.4. Formal Problem Statement

Let $f_0, f_1, ..., f_{|F|-1}$ be the OR functions, F, and $w_0, w_1, ..., w_{|W|-1}$ be the OR-term nanowires, W. If the number of inputs is N, then for all $f_i \in F$, $f_i = (I_{i,0}, I_{i,1}, ..., I_{i,N-1})$, where $I_{i,j}$ is 1 if input j of OR function f_i is ON and 0 if OFF. Similarly for all $w_i \in W$, $w_i = (J_{i,0}, J_{i,1}, ..., J_{i,N-1})$, where $J_{i,k}$ has value 1 if the corresponding crosspoint is programmable and 0 if non-programmable.

$G(F, W, E)$ is a directed bipartite graph. For every f_i in F and w_j in W, $(f_i, w_j) \in E$ if and only if:

$$\underset{0 \le k \le N-1}{\forall} (I_{i,k} \le J_{j,k}) \qquad (1)$$

Every matching of size $|F|$ on this bipartite graph is a valid assignment of the OR functions to the OR-term nanowires, because it finds an assignment for all of the OR functions in F. Figure 4(b) shows a bipartite graph $G(F, W, E)$. Set F is the set of OR functions in Figure 2, and set W is the set of nanowires in the nanoPLA of Figure 4(a). Figure 4(c) shows one possible matching.

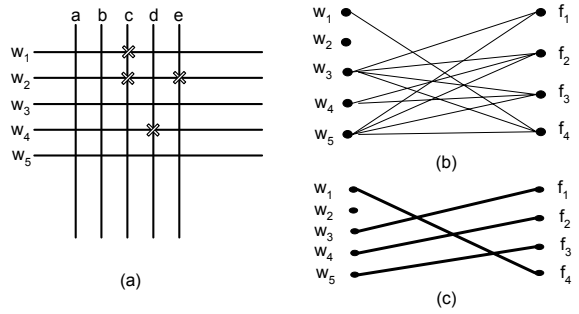


Figure 4. (a) The crosses show defective junctions. (b) The graph of the OR-term nanowire of part (a) and OR-functions of Figure 2. (c) One possible assignment.

6. Algorithm

6.1. Graph Construction

To build the graph $G(F, W, E)$ the first step is to find the nodes in each of the sets F and W. Each OR function in the design is a node in F. Marking ON and OFF inputs of each OR function, (*i.e.* the values of $I_{i,k}$) $f_i \in F$ and all the inputs k, $0 \leq k \leq N-1$, takes $O(|F| \cdot N)$ computing operations. To find the defect configuration of each OR-term nanowire (*i.e.* the value of each $J_{j,k}$), one should check the programmability of all the junctions of each OR-term nanowire. This takes $(N \cdot |W|)$ *programming and test* operations.

To make the set E, the condition (1) will be checked for each pair of (f_i, w_j). Checking it once takes N computing operations, and checking it over all of the pairs takes $O(N \cdot |F| \cdot |W|)$ computing operations. So the total time complexity of the graph construction is $(N \cdot |W|)$ *programming and test* operations and $O(N \cdot |F| \cdot |W|)$ computing operations.

6.2. Exact Algorithm

For now assume that W is large enough so that there exists a maximum matching of size $|F|$. Later, in Section 7, we calculate how large W should be in practice.

There are a number of exact algorithms to solve the maximum bipartite matching problem, such as the algorithm based on Ford-Fulkerson maximum flow network algorithm [13] with time complexity $O(|V| \cdot |E|)$ and Hopcroft-Karp [14] with time complexity $O\left(\sqrt{|V|} \cdot |E|\right)$. In our graph $|V| = |F| + |W|$ and $|E| = O(|F| \cdot |W|)$, which makes the total time complexity $O\left(\sqrt{(|F| + |W|)} \cdot |F| \cdot |W|\right)$ computing operations.

The time complexity of all of these matching algorithm will be dominated by the time complexity of graph construction of Section 6.1. Therefore to reduce the total time complexity, we suggest an approach that reduces the *program and test* operations of graph construction as well as the computing operations.

6.3. Greedy Heuristic Algorithm

There are heuristic algorithms that, with high probability, and small time complexity find the maximum matching. A general heuristic algorithm is shown in Figure 3(a).

We distinguish the different heuristic algorithms by the way they choose the nodes in lines 2 and 4 of Figure 3(a). One way is to choose both f and w randomly. Another way is to choose each of them in increasing order of node degree. A combination of the above is another option. We obtain our best results by choosing the least degree f from F and choosing w randomly.

Here we show how we can eliminate the need to actually build the graph $G(F, W, E)$. There are two points in the algorithm that are dependent on graph G: 1) Choosing f_i's based on their degrees G, 2) Line 5 of Figure 3(a) that checks the matching condition by checking the existence of the edge (f_i, w_j).

To select OR-terms based on least degree, we would need to sort F. Instead of sorting f_i's of F based on their degree, the nodes can be sorted based on the *expected value* of their degree. Let P_J be the probability that a junction is programmable, and c_i be the number of ON-inputs in the OR function f_i. The probability that $(f_i, w_j) \in E$ is ${P_J}^{c_i}$. This is the probability that the OR function f_i can be assigned to the nanowire w_j. So the *expected value* of node degree of f_i is $({P_J}^{c_i} \cdot |W|)$. Ordering F based on the *expected value* of node degrees is the same as ordering it based on the value of c_i. This means there is no need to build the graph for sorting purpose.

To test the condition of line 5 of Figure 3(a), in the case that there is no graph, we need to program and test every single nanowire that is picked up to be assigned to each OR function f_i. The time complexity of mapping and testing is $O(c_i)$ for each OR function f_i. In order to have time complexity of $O(c_i)$ instead of $O(N)$ the $I_{i,k}$'s need to be stored efficiently (sparsely). Hence by paying this cost there is no longer a need to build the graph $G(F, W, E)$, and the total time complexity is only due to the algorithm of finding a matching (Figure 3(b)).

7. Analysis

7.1. Running Time Complexity

We first compute the worst-case time complexity. As explained above, line 6 of the algorithm in Figure 3(b) takes $O(c_i)$ *program and test* operations. The maximum number of iterations of the line 4 loop

is the total number of unmatched nanowires which is $|W|-i$. The line 2 loop runs exactly for $|F|$ iterations in order to map each of the OR functions. So the total number of *program and test* operations in the worst-case is $O(\sum_{i=0}^{i=|F|-1}((|W|-i)\cdot c_i))$. It can be written as $O\left(|F|\cdot|W|\cdot c_M\right)$ when c_M is the maximum of c_i's. In Section 8 we show how to bound the size of c_M, without scaling $|F|$ by more than a small constant factor. Sorting F in the first line of Figure 3(b) takes $O(|F|\log(|F|))$ computing operations. So assuming $|F| \approx |W| \approx N$, our greedy algorithm takes N^2 *program and test* operations and $O(N\log(N))$ computing operations, while the exact approach takes N^2 *program and test* and $O(N^3)$ computing operations.

On average the number of iterations will be smaller than this. Let m_i be the number of iterations that it takes to find a match for OR function f_i. If we want the expected value of the matching for f_i in m_i nanowires to be 1 then:

$$\begin{aligned}&\mathrm{E}(\text{Number of matching in } \ m_i) = 1\\ &m_i \cdot P_J^{c_i} = 1 \Rightarrow m_i = P_J^{-c_i}\end{aligned} \tag{2}$$

So the average number of iterations of the line 4 loop is $P_J^{-c_i}$ for each f_i. and the total number of operations in the average case is:

$$O\left(\sum_{i=0}^{i=|F|-1}\left(P_J^{-c_i}\cdot c_i\right)\right) \tag{3}$$

and replacing c_i's with c_M: $O\left(|F|\cdot\left(P_J^{-c_M}\cdot c_M\right)\right)$

If the value of c_M is small, which it is after bounding fanin sizes, then the greedy algorithm takes $O(|F|)$ *program and test* and $O(|F|\log(|F|))$ computing operations on average, while the time complexity of the graph construction in the exact approach is N^2 *program and test* operations and $O(N^3)$ computing operations. Figure 6(b) shows the number of iterations to map each design ($|F|\cdot P_J^{-c_M}$) for bounded and unbounded c, which are nearly linear graphs.

7.2. Area Overhead Estimation

Here we compute how large W should be in practice. In the average case as shown before, if the size of the unmatched set of nanowires when matching the ith OR function is at least $P_J^{-c_i}$ then the *expected value* of finding a match in this set is 1. Therefore

$$\underset{0\le i\le|F|-1}{\forall}\left(P_J^{-c_i}\le|W|-i\right) \tag{4}$$

defines a lower bound on the size of W.

Remember that in our algorithm the set of nanowires that f_i can choose from, is of size $(|W|-i)$. Therefore the probability of successfully assigning f_i to a nanowire is $1-(1-{P_J}^{c_i})^{|W|-i}$ Hence the probability of successfully mapping all the OR functions is:

$$\prod_{i=0}^{|F|-1}\left(1-(1-{P_J}^{c_i})^{|W|-i}\right) \tag{5}$$

Let Y be the yield of mapping designs to nanoPLA. Then the following inequality gives a tighter lower bound on the size of W:

$$\prod_{i=0}^{|F|-1}\left(1-(1-{P_J}^{c_i})^{|W|-i}\right) > Y \tag{6}$$

8. Bounded Fanin

We show the effect of bounding the size of c_i's with an example from the IWLS93 benchmark [15]. In this example $|F| = 1186$, $c_M = 772$, and $P_J = 0.95$. The lower bound on $|W|$ for mapping a single nanowire with $c_i = c_M = 772$ from Equation (4), is:

$$(0.95)^{-772} \le W \Rightarrow 10^{17} \le W$$

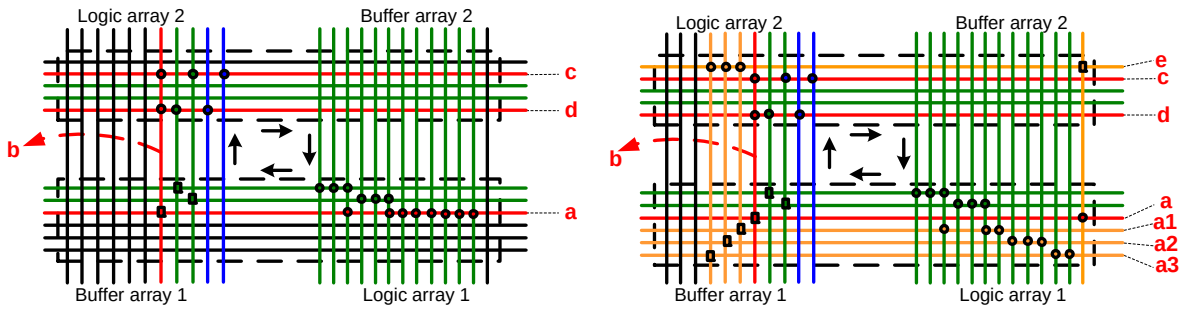
If we decompose this OR function to 8 OR functions, such that 7 of them have $c_i = 100$, and 1 has $c_i = 72$, then the lower bound on $|W|$ to map all of these OR functions is:

$$\begin{aligned}&Max\left(\underset{0\le i\le 6}{\forall}\left((0.95)^{-100}+i\right),\left((0.95)^{-72}+7\right)\right)\\ &\Rightarrow 173 \le W\end{aligned}$$

Applying Equation (3) we also see that bounding the fanin improves the mapping running time from 10^{20} to 10^5 *program and test* operations.

Figure 5 shows how an OR function with c=8, will be decomposed into OR functions with $c \le 3$. Figure 6(a) shows which OR functions in each design need to be divided to smaller fanin OR functions if the size of $|W|$ is desired to be $|F|$ or $1000\times|F|$. The x-axis in this graph is the number of OR functions in each design, *i.e.* $|F|$. Each point on the x-axis is dedicated to a single design with $|F|$ equal to the value of x at that point. For example the highlighted yellow diamonds show the c_i value of all the OR functions of a design with $|F| = 1186$. The curves show the estimation in Equation (4) of the maximum size of c_i if $|W| = |F|$ or $|W| = 1000\times|F|$. Assuming $|W| = |F|$ and using Equation (2) and (4) the value of the maximum c_i's on the lower curve result from ${P_J}^{-c_M} < |F|$, and further we can estimate the lower bound on c_M by $c_M < -\log_{P_J}|F|$. Similarly the lower bounds for c_i's related to the case when $|W| = 1000\times|F|$ will be $-\log_{P_J}(1,000\cdot|F|)$.

The graphs of Figure 6(a) show that the number of OR functions with large ON input set is relatively small, and we also observed they cause very long running time and large area overhead in mapping. This suggests we should bound the size of c_M with $-\log_{P_J}|F|$ to get the ratio of 1 for $|W|/|F|$ on average. Since the size of c_i's is bounded, in order to sort



In (a) the OR function *a* has c=8. In (b) it is divided into 3 OR functions *a1*, *a2* and *a3*. They are OR-ed together in logic array 2 and make signal *e*, which is the same as the logic of the original *a* OR function. The OR function *e* is rotated to logic array 1 and then to buffer array 1, which was *a*'s original position. Two logic levels of delay are added to the OR function *a*. If the size of ON inputs of signal e is more than c_M then the decomposition process will be repeated for signal e. For an OR function with c ON inputs, the decomposition process happens $\log_{c_M}(c)$ times.

Figure 5. (a) The original design, (b) design with c value bounded by $c_M = 3$.

F in the first line of the algorithm in Figure 3(b) we can use a radix sort with time complexity of $O(|F|)$, bringing the total computation time of our greedy algorithm to $O(|F|)$.

9. Experimental Results

The mapping algorithm is tested over three different benchmarks: 1) Selected elements of datapath (See [2]), 2) Small examples from IWLS93 benchmark suit [15], 3) PLA book examples [16]. For statistical purposes each benchmark is mapped 100 times. The designs of these benchmarks have been first synthesized to multilevel logic and rotated through two NOR planes of a nanoPLA [2].

Figure 6(b) shows the graphs for estimation of total number of iteration to map OR functions of a design for bounded and unbounded c and also the simulation results for bounded c. Figure 6(b) shows that the total number of iterations is generally linear in the number of OR functions, $|F|$, and well matched with the calculation in Equation (3). Figure 6(c) shows the average area overhead ratio over all the benchmark set designs. Bounding the fanin scales the number of OR functions by an average factor of 1.11 for a defect rate (P_J) of 0.20. The additional average factor of 1.02 is incurred after physically mapping these OR functions onto nanowires. This brings the total average overhead factor to 1.13. The total number of *program and test* operations to map each design, are plotted in Figure 6(d). The blue line shows the size of the OR functions. The slope of the pink graph is close to the slope of the blue line that is one. This indicates that the total number of *program and test* operations are linear to the size of the OR functions as explained in Section 7.1.

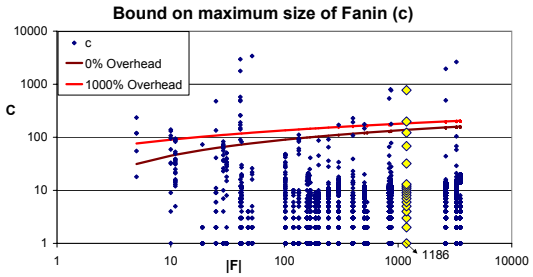
The area overhead of this greedy algorithm is compared with an exact matching algorithm for a 4×4 multiplier that is implemented in two logic planes. The first plane has 697 OR functions and 33 inputs and the other one has 25 OR functions and 697 inputs. In Figure 7(a) area overhead of each of the planes is plotted for both greedy and exact algorithm. In Figure 7(b) the ratio of the total area of the exact algorithm over the total area of the greedy algorithm is plotted. This shows that our greedy algorithm is within a few percent of optimal on average for modest fault rates.

10. Summary

A plausible architecture for nanoPLA design is suggested in [2]. The defect rate of different fabrication processes is unknown but expected to be on the order of a few defects per 100 junctions. This suggests searching for an efficient programming operation that tolerates the defective junctions. In this paper we compare the exact matching algorithm with a suggested greedy algorithm. Assuming that $|F| \approx |W| \approx N$, the time complexity of our algorithm is, $O(N)$ *program and test* operations and $O(N \log(N))$ computing operations, while the time complexity of the exact algorithm plus graph construction is N^2 *program and test* operations and $O(N^3)$ computing operations. We also showed that it is necessary to bound the fanin size in order to achieve reasonable running time and area overhead for matching. After bounding the fanin, the time complexity of our algorithm will be $O(N)$ computing and *program and test* operations. Including bounding the fanin and mapping, our algorithm can tolerate defect rates as high as 20% with an average overhead factor of less than 13%.

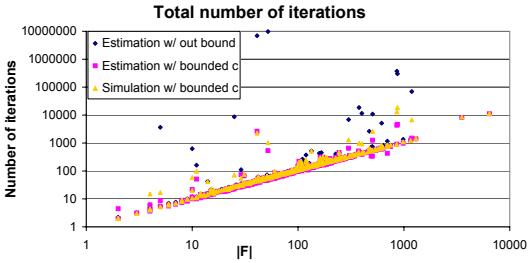
11. Acknowledgments

This research was funded in part by the DARPA Moletronics program under grant ONR N00014-01-0651 and N00014-04-1-0591.



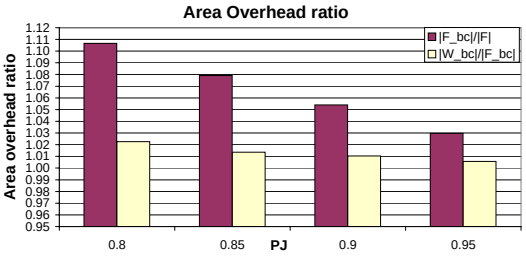
The dots show the c_i's of each OR functions of each design. The curves show two maximum sizes of c for each design.

(a)



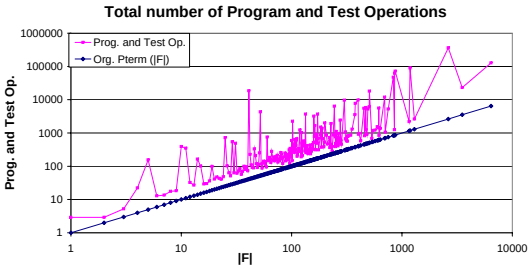
Number of iterations to map the whole design.

(b)



The average ratio of $\frac{|F\_bounded\_c|}{|F|}$ and $\frac{|W\_bounded\_c|}{|F|}$.

(c)



The average number of *program and test* operations to map each design.

(d)

Figure 6. $P_J = 0.95$**, and** c **is bounded by** $-\log_{P_J}|F|$ **in (b).**

12. References

[1] S. C. Goldstein and M. Budiu, "NanoFabrics: Spatial Computing Using Molecular Electronics," in *ISCA*, June 2001, pp. 178–189.

[2] A. DeHon and M. J. Wilson, "Nanowire-Based Sublithographic Programmable Logic Arrays," in *FPGA*, February 2004, pp. 123–132.

[3] Y. Luo, P. Collier, J. O. Jeppesen, K. A. Nielsen, E. Delonno, G. Ho, J. Perkins, H.-R. Tseng, T. Yamamoto, J. F. Stoddart, and J. R. Heath, "Two-Dimensional Molecular Electronics Circuits," *ChemPhysChem*, vol. 3, no. 6, pp. 519–525, 2002.

[4] "International Technology Roadmap for Semiconductors," <http://public.itrs.net/>, 2003.

[5] Y. Chen, G.-Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams, "Nanoscale Molecular-Switch Crossbar Circuits," *Nanotechnology*, vol. 14, pp. 462–468, 2003.

[6] G. Snider, P. Kuekes, and R. S. Williams, "CMOS-like Logic in Defective, Nanoscale Crossbars," *Nanotechnology*, vol. 15, pp. 881–891, June 2004.

[7] Y. Cui, L. J. Lauhon, M. S. Gudiksen, J. Wang, and C. M. Lieber, "Diameter-Controlled Synthesis of Single Crystal Silicon Nanowires," *Applied Physics Letters*, vol. 78, no. 15, pp. 2214–2216, 2001.

[8] Y. Huang, X. Duan, Y. Cui, L. Lauhon, K. Kim, and C. M. Lieber, "Logic Gates and Computation from Assembled Nanowire Building Blocks," *Science*, vol. 294, pp. 1313–1317, 2001.

[9] D. Whang, S. Jin, and C. M. Lieber, "Nanolithography Using Hierarchically Assembled Nanowire Masks," *Nanoletters*, vol. 3, no. 7, pp. 951–954, July 9 2003.

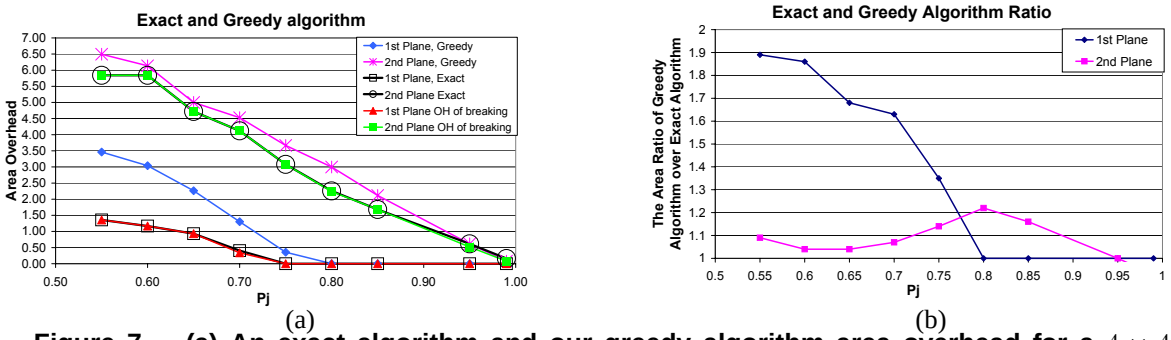
[10] C. Collier, G. Mattersteig, E. Wong, Y. Luo, K. Beverly, J. Sampaio, F. Raymo, J. Stoddart, and J. Heath, "A [2]Catenane-Based Solid State Reconfigurable Switch," *Science*, vol. 289, pp. 1172–1175, 2000.

[11] M. S. Gudiksen, L. J. Lauhon, J. Wang, D. C. Smith, and C. M. Lieber, "Growth of Nanowire Superlattice Structures for Nanoscale Photonics and Electronics," *Nature*, vol. 415, pp. 617–620, February 7 2002.

[12] A. DeHon, P. Lincoln, and J. Savage, "Stochastic Assembly of Sublithographic Nanoscale Interfaces," *IEEE Transactions on Nanotechnology*, vol. 2, no. 3, pp. 165–174, 2003.

[13] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. MIT Press, 1990.

[14] J. E. Hopcroft and R. M. Karp, "An $n^{2.5}$ Algorithm for Maximum Matching in Bipartite Graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.



(a) (b)

Figure 7. (a) An exact algorithm and our greedy algorithm area overhead for a 4×4 multiplier. The area over $|F|$ is plotted. (b) $\frac{|W_{greedy}|}{|W_{exact}|}$.

[15] K. McElvain, "LGSynth93 Benchmark Set: Version 4.0," online <http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth93/doc/iwls93.ps>, May 1993.

[16] U. C. Group, "Espresso Examples," Online <ftp://ic.eecs.berkeley.edu/pub/Espresso/espresso-book-examples.tar.gz>, June 1993.