# Nanowire Reconfigurable Crossbar 구조를 위한 결함 회피형 로직 재할당 방식의 분석과 총 비용에 따른 최적화 방안

## (Cost-Driven Optimization of Defect-Avoidant Logic Mapping Strategies for Nanowire Reconfigurable Crossbar Architecture)

이 종 석 † 최 민 수 ††
(Jongseok Lee) (Minsu Choi)

**요 약** Photolithography 통합 시대의 끝이 빠르게 다가옴에 따라, 최근에는 새로운 나노 스케일의 소재와 집적 방식에 기반을 둔 수많은 나노 스케일 장치와 시스템이 나타나고 있다. 특히 nanowire crossbar 구조를 이용한 다양한 reconfigurable architecture 들이 보고되고 있다. 하지만 아쉽게도 나노 스케일의 구성 요소를 이용한 이러한 고집적 시스템은 생산 단계에서 발생하는 각종 물리적 결함과 오차에 취약하며 따라서 결함에 대한 관용성 즉 defecttolerance는 nanowire reconfigurable crossbar 시스템에 있어 해결해야 할 가장 중대한 문제 중 하나라 할 수 있다. 이에 본 논문에서는 nanowire reconfigurable crossbar 시스템 상에서 사용되어질 수 있는 세 가지의 결함 회피형(defectavoidant) 로직 재할당 알고리듬을 설명하고 다양한 방식으로 평가하였다. 이에 더불어 로직 재할당시에 발생하는 비용과 이로 인해 얻어지는 repair performance를 계량적으로 상호 분석하여 최적화된 repair 방식을 찾아내는 새로운 방안을 소개하였다. 이어 다양한 파라메터들을 이용한 시뮬레이션 결과를 제시함으로써 새로 소개된 cost-driven repair 최적화 방식을 검증하였다.

키워드 : Nanoelectronics, nanowire crossbar, reconfiguration system, 결함 회피, 로직재할당, cost-driven optimization

*Abstract* As the end of photolithographic integration era is approaching fast, numerous nanoscale devices and systems based on novel nanoscale materials and assembly techniques are recently emerging. Notably, various reconfigurable architectures with considerable promise have been proposed based on nanowire crossbar structure as the primitive building block. Unfortunately, high-density sys-tems consisting of nanometer-scale elements are likely to have numerous physical imperfections and variations. Therefore, defect-tolerance is considered as one of the most exigent challenges in nanowire crossbar systems. In this work, three different defect-avoidant logic mapping algorithms to circumvent defective crosspoints in nanowire reconfigurable crossbar systems are evaluated in terms of various performance metrics. Then, a novel method to find the most cost-effective repair solution is demonstrated by considering all major repair parameters and quantitatively estimating the performance and cost-effectiveness of each algorithm. Extensive parametric simulation results are reported to compare overall repair costs of the repair algorithms under consideration and to validate the cost-driven repair optimization technique.

· 이 논문은 2010학년도 우석대학교 교내학술연구비 지원에 의하여 연구되었음

† 종신회원 : 우석대학교 컴퓨터교육과 교수
jong1007@nate.com
†† 정 회 원 : Missouri University of Science and Technology Dept. of Electrical & Computer Engineering professor
choim@mst.edu

논문접수 : 2010년 2월 27일
심사완료 : 2010년 8월 9일

Copyright©2010 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 시스템 및 이론 제37권 제5호(2010.10)

Key words : Nanoelectronics, Nanowire crossbar, Reconfigurable system, Defect-tolerance, Logic mapping, Cost-driven optimization

## 1. Introduction

The end of photolithography as the driver for Moore's Law is predicted within seven to twelve years [1] and various nanotechnologies are emerging that are expected to continue the technological revolution in integrated circuits and systems [2]. Recently, numerous nanoscale logic devices have been proposed based on nanoscale components such as carbon nanotubes (CNTs) and silicon nanowires (SiNWs); computing architectures are also being proposed using them as primitive building blocks. Unlike conventional CMOS (Complementary Metal-Oxide Semiconductor), chemically-assembled nanoscale components (such as CNTs and SiNWs) are unlikely to be used to construct complex aperiodic structures due to nondeterministic nature of nanoscale assembly [3-5].

One of the most promising computational nanotechnologies is the crossbar-based archi-tecture [6-13], a two-dimensional array (nanoarray) formed by the intersection of two orthogonal sets of parallel and uniformly-spaced nanometer-sized wires, such as carbon nanotubes (CNTs) and silicon nanowires (SiNWs). Experiments have shown that such nanoscale wires can be aligned to construct an array with nanometer-scale spacing using a form of directed self-assembly the formed crosspoints of nanoscale wires can be used as programmable diodes, memory cells or FETs (Field-Effect Transistors) [14-18].

Nanoscale crossbar systems offer both an opportunity and a challenge. The opportunity is to achieve ultra-high density which has never been achieved by photolithography (a density of $10^{11}$ crosspoints per $cm^2$ has been achieved [14]). The challenge is to make them defect tolerant, since high-density systems consisting of nanometer-scale elements assembled in a bottom-up manner are likely to have many imperfections (much higher raw fabrication defect densities, as high as 10%, are expected [19-21]). A computing or storage system designed on conventional defect basis and top-down lithographic manufacturing would not be practical [22]. Ultra-high density fabrication could potentially be very inexpensive if researchers can actualize a chemical self-assembly, but such a circuit would require laborious testing, repair and reconfiguration processes, implying significant overhead costs. So, finding the most cost-effective post-fabrication logic mapping solution is desired.

In this work, inherently-defective nanoscale crossbar systems are considered to be repaired by defect-avoidant logic mapping. Newly assembled nanowire crossbars are to be tested to locate defective crosspoints. Such defective crosspoints cannot be programmed to "closed" state (i.e., also known as the ON-state); therefore, should be avoided when a netlist is mapped onto them. Three simple repair algorithms are considered and analyzed in terms of repair performance and overhead costs in this work. In such logic mapping procedure with defect-avoidance, 100% netlist coverage should be achieved while minimizing overall overhead costs such as unused reconfigurable resources, logic mapping algorithm execution time and the number of reprogrammed switching crosspoints. To address this problem, a novel cost-driven repair optimization technique has been proposed and validated through extensive parametric simulations. In the proposed technique, all design and algorithm parameters are considered at the same time to find the optimized repair solution among various repair solution candidates. Full coverage of the given netlist is achieved by the selected optimal repair solution while guaranteeing the lowest possible overhead costs.

This article is organized as follows: In Section II, all aspects of nanowire reconfigurable crossbar architectures (such as bottom-up assembly, devices and architectures) are summa-rized and reviewed. Then, the reconfigurable crossbar system repair problem is formally defined in Section III. Three different logic mapping algorithms with defect-avoidance are discussed in Section IV. The proposed cost-driven repair optimization technique and simu-

lation results are given in Section V. Then, concluding remarks are made in Section IX.

## 2. Preliminaries & Review

### 2.1 Crossbar-Based Nanoscale Devices

CNTs and SiNWs are the most promising building blocks for nanoscale computing sys-tems. Unfortunately, synthesis of such nanowires and high-density integration of devices and systems based on nanowires are fully different from conventional top-down lithographic fabrication techniques, because such nanowires must be synthesized first, then assembled into functional devices and systems in a bottom-up manner [23]. Fig. 1 illustrates hierarchical patterning of 1D and 2D nanowire crossbar structures by fluidic flow method [23].

One or more crosspoints can be grouped together to form a memory or logic device. Electro-mechanical switching devices using suspended nanotubes are proposed in [6]. The NT-NT junction is bistable with an energy barrier between the two resistive states (i.e., GΩ to 100 KΩ). Thus, diode logic can be realized in nanoscale.

Doped SiNWs exhibit FET (Field Effect Transistor) behavior [17]. That is, oxide can be grown over the SiNW to prevent direct electrical contact of a crossed conductor. The electrical field of one wire can then be used to gate the other wire by locally evacuating a region of the doped SiNW of carriers to prevent conduction. In other words, as the gate voltage is changed, conductance increases or decreases between the source and drain. CNTs also demonstrate FET behavior [24-27] and can be also used as memory devices [6,28] or interconnects [29].

Molecular resonant tunneling diodes, often called negative differential resistors (NDRs), have also been realized [30-35]. Devices with useful Peak-to-valley ratio have been measured at room temperature [32]. These molecules can be used to as the basis of logic families [36,37] or as the core of a molecular latch which also provides signal restoration and I/O isolation [4].

Molecular switches and memories are also proposed in [38,39]. Organic molecules exist which have two mechanically distinct parts, such as a ring and a rod or interlocking rings. Applying a programming voltage across the molecule adds or subtracts an electron (oxidation-reduction), shifting the ring and changing the molecule's conductivity. It functions as a non-volatile programmable molecular switch. Used between a metal wire and an n-type silicon nanowire, the junction acts as a programmable diode, making an addressable memory array.

### 2.2 Nanowire Crossbar Architectures

Using the nanoscale switching, configurable OR planes can be assembled, with connected wires acting as low-resistance p-n-junctions and distant wires isolated by high resistance (see Fig. 2) [40]. In Fig. 2, two logic functions are implemented by configuring nanoscale switches at crosspoints: *out1 = in1 OR in3* and *out2 = in1 OR in2*. Similarly, configurable NOR planes can be assembled. Since {OR, NOR} is a complete logic set, any digital

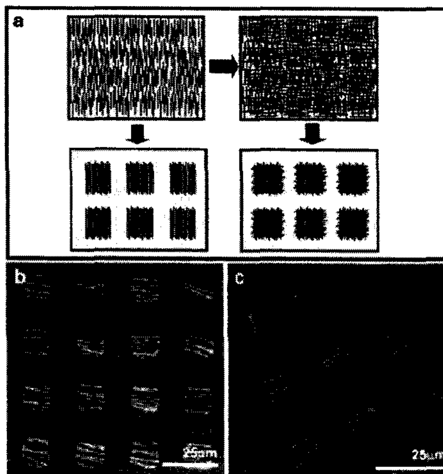


Fig. 1 (a) hierarchical patterning by fluidic flow and LB method to form: (b) 1D parallel bar structure, (c) 2D crossbar structure [23]

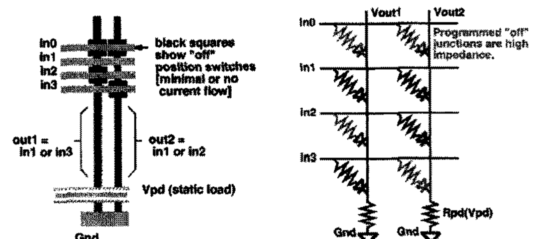


Fig. 2 Nanowire-based programmable diode OR array [40]

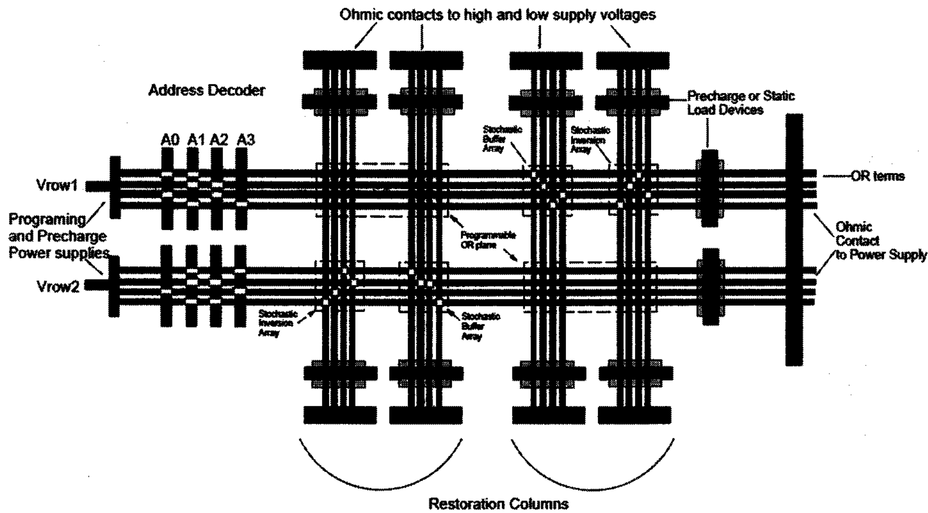


Fig. 3 An illustration of NanoPLA Architecture [41]

logic circuits can be implemented, if sufficiently interconnected OR and NOR planes are given.

In [41], DeHon et. al. have proposed a method to build sublithographic PLAs (Pro-grammable Logic Arrays) using nanowires (NWs) and to interconnect PLAs to form large arrays. In this architecture, the PLAs are built upon programmable crosspoint diodes and by using lithographic scale address decoder that can be used to address individual nanowires. Also, by using some semi-static structure and app-lying a sequence of timing control signals, the PLAs can perform buffer and inverter functions as well as global clock control.

There is another nanowire crossbar architecture called the NanoFabric. The NanoFabric architecture has nanoscale crossbars and supporting microscale components and facilitates directed nanoscale self-assembly paradigm. There are a few key elements in the architecture as follows: *Nanoblock:* The archi-tecture of NanoFabric consists of array of inter-connected nanoblocks and switch blocks. The logic block in this architecture is similar to configurable logic block in FPGA. Nanoblock is based on a mole-cular logic array (MLA). There is a reconfigurable switch at each intersection of MLA in series with diode. Diode-resistor logic is used to perform logical operations. Both signals and their complemented signals are produced to make a complete logic set. Logic values are restored using molecular latches, and *Switchblock:* A switch block is also recon-figurable and serves to connect wire segments of adjacent NanoBlocks. The configuration of the switch block determines the direction of data flow between the logic blocks.

## 3. Nanoscale Reconfigurable Crossbar Repair Problem

Due to imperfections and variations in nanoscale manufacturing, defect densities as high as 10% is anticipated in nanowire crossbars. Nanowire crossbar architectures share common characteristics - they support nanoscale manufacturing paradigm via simple homogeneous periodic structures and reconfigura-bility for post-fabrication design mapping. Thus, such defects should be located when tested and avoided when the given design is mapped. In this section, a general model for nanoscale crossbar systems will be proposed and the defect avoidance logic mapping problem will be formally defined based on the proposed model.

The programmable diode crossbar structure (namely, logic block) supports flexible utiliza-tion of its cross-points through reconfiguration, even though the defect rate is as high as 10%. The internal lines in the logic blocks are completely interchangeable and the switch block can provide the flexible connec-tions between inputs and outputs signals of adjacent logic blocks. It is possible to utilize such flexibility

and reconfigurability to get around defective crosspoints. The term *"repair"* used in this paper refers to the defect-avoidant logic mapping procedure in which defective crosspoints are located and avoided when the given logic is programmed.

A few testing methods for crossbar architectures have been reported in literature, including [5] and [21]. The basic idea is to use an external tester to test a certain area of the crossbar chip under test. Then program that tested area to an internal tester which can be used to test the rest of the given DUT (Device Under Test). Except using the external tester, all the rest testing can be therefore viewed as BIST (Built-In Self Test). Also, this method could be done in parallel so that better test speed can be achieved.

Likely defects in nanowire crossbar systems are also anticipated to be different from ones in CMOS systems. Such nanowire crossbar specific defects are often categorized as: 1) defects in programmable crosspoints, and 2) defects in nanowires [42]. Nanowires with short or break can be easily tested out and all crosspoints fall into those nanowires simply marked as unusable. Physically, defects in programmable crosspoints are due to the structure of the junctions, which are bistable molecules between two layers of nanowires. Reprogrammability of a crosspoint comes from the bistable property of the molecules located in the crosspoint area. If there are not enough molecules at a certain crosspoint then that junction may not be able to be programmed to a "closed" state, or the "closed" state may have higher resistance than the threshold from speculation which enable the whole system operate properly. If the crosspoint cannot be programmed to "open" status which means the two crossing nanowires are always connected, like a short occurred in those two nanowires, we should treat these as nanowire defects rather than junction defects. Those crosspoints which cannot be programmed into a "closed" state, but can be programmed into a "open" state are referred to as the *non-programmable* crosspoints. Although the non-programmable crosspoints are defective, they do not affect the other crosspoints in the rows and columns associated with them.

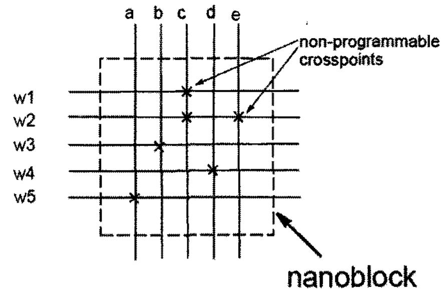


Fig. 4 Nanoscale crossbar structure with defective crosspoints

For example, in Fig. 4, nanowire $w3$ only has one non-programmable crosspoint at the junction with nanowire $b$. The junctions with nanowire $a$, $c$, $d$ and $e$ are still defect-free; therefore, programmable. So, if we happen to have a function $f = a + c + d + e$ needed to be mapped (i.e., note $b$ is not used in this function), nanowire $w3$ is still good to be matched and the output will not be affected by the non-programmable crosspoint. As clearly shown in this example, defective crosspoints can be avoided while the given logic functions are successfully mapped if certain conditions are met.

To map the given physical design onto the reconfigurable crossbar system, the logic synthesizer should generate a netlist which allocates some of the nanowires as inputs, some as outputs and also indicate which crosspoints need to be set to "closed" state. In this work, $M \times M$ matrix $F$ is used to represent the set of functions that are needed to program the given reconfigurable crossbar system. In $F$, columns represent input terms and rows represent OR functions based on the input terms. If the node value is 1, this means the corresponding crosspoint is needed to be programmed to "closed" state and the crosspoint that should be programmed to "closed" state is called "on-input". If the node value is 0, the crosspoint should be left as "open" state. Because of the inherent reconfigurability of the crossbar architecture, the order of rows and columns can be rearranged if coupled switch blocks are reconfigured, accordingly.

After testing, a defect map which indicates the locations of the defective crosspoints can be constructed. Another $N \times N$ matrix $D$ is to represent

the defect map. Both $F$ and $D$ are assumed to be symmetric for the simplicity in numerical analysis in this paper. For the location which represents a non-programmable crosspoint, 1 is allocated to indicate the defect. Otherwise, 0 is allocated to indicate the corresponding crosspoint location is programmable. An OR function from $F$ (i.e., one row from $F$) can be assigned to a physical nanowire row if and only if each of the on-inputs of the OR function has a corresponding non-defective crosspoint on the physical nanowire.

Even though some nanowires have defective crosspoints, some OR functions may still be successfully mapped to them, if on-inputs do not fall into non-programmable crosspoints. The challenge is to find a successful mapping while minimizing overhead costs induced by the defect-avoidant logic mapping procedure. In the following section, three simple defect-avoidant logic mapping algorithms under consideration will be discussed.

## 4. Defect-Avoidant Logic Mapping Algorithms Under Consideration

For the given nanowire crossbar repair problem, row-wise (or column-wise) repair algo-rithm is trivial, since the greedy algorithm always results in the best solution. However, for the two-dimensional repair problem which is similar to two-dimensional memory repair problem, only the brute-force algorithm can guarantee the optimal solution, simply because it is an NP-complete problem [43-47]. So, faster algorithms that can be used to find sub-optimal solution at reasonable overhead are usually pursued. For the given reconfigurable crossbar repair problem, the following three repair algorithms are extensively evaluated:

1) *Row-wise matching algorithm (Algorithm #1)*: $F$ and $D$ are sorted in descending order and ascending order of $n_{ri}$, respectively. The idea is to arrange OR functions in $F$ in decreasing order of $n_{ri}$ and nanowire rows in $D$ in increasing order of $n_{ri}$. Then, row-wise greedy matching is performed. For each OR function in $F$ is selected from top to bottom. Then, sequential search on nanowire rows in $D$ is performed to find a successful matching. This procedure is repeated until all OR functions in $F$ are successfully mapped to nanowire rows while avoiding defective crosspoints that cannot be programmed to "closed" state. A detailed pseudo code of this algorithm is shown in Fig. 5.

```
   \* Row-wise matching algorithm *\
1  Sort F in descending order of n_ri and D in ascend-
   ing order of n_ri
2  While F is not empty
3    Select the first OR function from F
4    While (f_i is not matched) and (W has non-visited)
5      Select the first unused row in D
6      If defect-avoiding matching is possible
7        Mark both as matched
8        Remove them from F and D
9      Else
10       Set the selected row visited
11   EndWhile
12 EndWhile
```

Fig. 5 Pseudo code for row-wise matching algorithm (namely, Algorithm #1). Both $F$ and $D$ are sorted and each OR function in $F$ is mapped onto $D$ while avoiding defective crosspoints

2) *Column-matching-first algorithm (Algorithm #2)*: In this algorithm, the order of nanowire columns are rearranged first so that the possibility of successful mapping could be improved. The idea is to arrange nanowire columns in $D$ so that the nanowire column with larger $n_{ci}$ to the physical column with smaller number of defects (i.e., smaller $n_{ci}$). In this way, the possibility of successful mapping will be increased significantly. So, in this algorithm, nanowire columns are rearranged and mapped to function input columns. Then, the row-wise mapping algorithm is invoked to map individual OR functions. A detailed pseudo code of this algorithm is shown in Fig. 6.

3) *Redundant column-matching-first algorithm (Algorithm #3)*: If $M > N$, there should be unused nanowire columns can be used as redundancy. The switching block allocated to arrange input terms to the logic block can be rearranged to assign an input term to more than one nanowire columns. In that case, these unused columns can be utilized as redundancy and more than one nanowire columns can be configured to represent the same input term. So, if more than one of the crosspoint(s) that represent the same input term is programmable, then successful mapping

```
\* Column-matching-first algorithm *\
1  While (F has unvisited column)
2    Find the column w/ max # of 1s in F
3    Find the column w/ min # of 1s in D
4    Match two selected columns
5    Mark two selected columns as visited
6  EndWhile
7  Invoke row-wise greedy matching algorithm
```

Fig. 6 Pseudo code for column-matching-first algorithm (namely, Algorithm #2) . Columns of $D$ are rearranged to increase the chance of successful matching in row-wise mapping

```
\* Redundant column-matching-first algorithm *\
1  While (F has unvisited column)
2    Find the column w/ max # of 1s in F
3    Find the column w/ min # of 1s in D
4    Match two selected columns
5    Mark two selected columns as visited
6  EndWhile
7  Mark all columns in F as unvisited
8  While (D has unused column)
9    Find the column w/ max # of 1s in F
10   Find the column w/ min # of 1s in D
11   Match two selected columns
12   Mark two selected columns as visited
13 EndWhile
14 Invoke row-wise greedy matching algorithm
```

Fig. 7 Pseudo code for redundant column-matching-first algorithm (namely, Algorithm #3). Columns with excessive defective crosspoints are simply screened out first

is still possible. Thus, this redundancy utilization may further increase the probability of successful mapping. A detailed pseudo code of this algorithm is shown in Fig. 7.

Parametric simulators of the described algorithms have been implemented using MatLab. The following notations will be used throughout this work:

- $n_{ri}$: the number of 1s in the $i_{th}$ row vector from $F$ or $D$.
- $n_{ci}$: the number of 1s in the $i_{th}$ column vector from $F$ or $D$.
- $r_i$: the $i_{th}$ row in matrix.
- $c_i$: the $i_{th}$ column in matrix.
- $M$: the size of matrix $F$ is $M \times M$ which is also the size of the given function set.
- $N$: the size of matrix $D$ is $N \times N$ which is also the size of the physical array.
- $P_j$: the probability of the given crosspoint to be functional.
- $E_m$: the possibility of having rowi from $F$ has a matching in $D$.

The following assumptions were used throughout the simulation:

- A defect rate of 10% is used. When a $N \times N$ matrix $D$ is constructed, each crosspoint has 10% probability to be 1.
- A function usage rate, $P_f$ is used to generate the function matrix $F$; meaning that each crosspoint has $P_f$ probability to be 1.
- It is also assumed that $N \geq M$.

Two performance metrics for the nanowire crossbar repair algorithms were proposed in [48] and are defined as follows:

- *Utilization*: The number of utilized nanowires in $D$ which have been successfully mapped to OR functions in $F$ divided by the physical array size $N$.
- *Coverage*: The number of OR functions which have been successfully mapped to physical array divided by the given function size $M$.

Considering the $F$ and $D$ are randomly generated following the predetermined probabilities (i.e., $P_f$ and $P_j$), the simulation result may show some fluctuations. In order to capture steady-state results, each simulation routine was iterated for 100 times and the average values were collected for each data point. Coverage and utilization plots with respect to different values of $N$ in $X$ axis were reported in [48]. As shown in these plots, algorithms #2 and #3 (i.e., 2D repair algorithms) have significant performance benefits over the algorithm #1 (i.e., 1D repair algorithm). Thus, it can be concluded that both the column rearrangement technique and the redundant column utilization technique are effective in improving the overall repair performance.

## 5. Cost-Driven Repair Optimization

### 5.1 Algorithm Execution Time Overhead

From the preliminary discussion in the previous section, it is possible to observe the advantages that 2D algorithms have over 1D in terms of *Utilization* and *Coverage*. However, 2D algorithms are more computationally complex than the 1D one, which means it is likely to spend more time to run 2D algorithms. The time overhead is a significant

cost factor in manufacturing, so it should be considered as an overhead cost along with the other repair overhead costs.

Firstly, let us analyze the *Algorithm #1*. As shown in Fig. 5, the algorithm could be decomposed in two segments. The first segment is the sorting process. The second segment is the sequential matching process. In order to sort the rows in the order of $n_{ri}$, $n_{ri}$ should be counted first. For the function matrix $F$, we need $O(M)$ steps to count $n_{ri}$. For a typical quick sort algorithm, $O(M \times \log M)$ steps are needed to get the sorting done. To get the number of steps needed to complete the matching process, the approach described in [42] can be used. Suppose $m_i$ is the number of iterations needed to find a matching of the $i_{th}$ row from $F$ in $D$. Probability of the matching can be calculated which is denoted as $E_m$. So, we have $m_i \times P_j^{c_i} = E_m$. Therefore, $m_i = E_m \times P_j^{c_i}$, which is the average number of iterations that one row need to find a matching. So, the total number of steps on average is:

$$O\left(\sum_{i=1}^{i=M}\left(E_m \times P_j^{-c_i} \times c_i\right)\right) \quad (1)$$

Note equation (1) are adopted from the work by [40]. Considering the $D$ is also need to be sorted, the number of overall steps for 1D algorithm need include the $D$ sorting part $O(N \times \log N)$. Among the $O(M)$, $O(N)$, $O(M \times \log M)$, $O(N \times \log N)$ and equation (1), the dominant complexity of the algorithm is therefore from equation (1).

Then let us consider *Algorithm #2*. As shown in Fig. 6, the difference between this algorithm and 1D algorithm exists in the column rearrangement process. So, following the similar method, the complexity of *Algorithm #2* is also dominated by equation (1).

For *Algorithm #3*, an additional redundant column rearrangement process is needed, so the dominant part is expressed in equation (1).

Thus, in terms of time complexity, those three algorithms are relatively similar. But in reality, the running times of those algorithms are still quite different. The software running time of each algorithm was measured by MatLab tool and shown in Fig. 8. The computing hardware used for this analysis has a Pentium IV 2.4Ghz processor and a 512MB RAM. In different industry/hadware environments, if we can correctly select coefficient, the actual running time captured by Matlab can reflect the real time overhead of the implementations. A method to estimate the time overhead coefficient will be discussed later in this section.

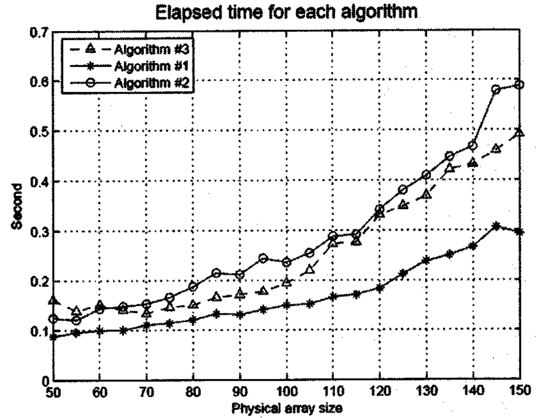


Fig. 8 Elapsed execution time curves for three different repair algorithms [48]

### 5.2 Unused Area Overhead

In practice, after the completion of design, the size of the OR function set $M$ is determined. The manufacturing process would define the real array size $N$ with a certain variation. It is possible to have the array size $N$ to be greater or equal to $M$. Different array size $N$ may cause different *Utilization* and unused area overhead. So the first task is to find the area overhead to implement the given function set with a certain size $M$ for different physical array size $N$. In order to achieve 100% *Coverage*, one or more logic blocks are needed to map the given function set $F$, completely.

For example, if the physical array size is the same as given function set size (i.e., $N = M = 50$), *Coverage* is around 32%. So, at least 4 logic blocks are needed to fully implement $F$. If $N$ is increased to 150, *Coverage* is 100% so one 150 × 150 logic block is enough to fully implement the given $F$. The number of crosspoints used in these two cases can be also calculated to be considered. The first case is 50 × 50 × 4 = 10; 000 and the second case is 150 × 150 × 1 = 22; 500. There are $M^2$ crosspoints are used by implementing the function set $F$, which should be deducted from the total number of cross-

points when calculating the wasted area. So, the first design candidate, $N = M = 50$, is better than the second one in terms of area overhead since less crosspoints are unused. The same method can be used to find the area overhead for each $N$ value, so that the optimal physical array size can be found. A mathematical model can be established to calculate the lowest area overhead as possible.

First, the number of arrays needed for a certain combination of $M$ and $N$ should be found. The objective of this search is to find the average number of arrays needed, which can be defined as: $N_a = \lceil (Coverage)^{-1} \rceil$.

$N_a$ can be used as a starting point and can be increased or decreased until 100% *Coverage* and smallest $N_a$ are achieved simultaneously. The detailed process is described as follows. We first try to match $F$ to $N_a$ physical arrays initially. If 100% coverage is achieved then we decrease the $N_a$ by 1 otherwise we increase the $N_a$ by 1 until we obtain 100% coverage. In this way, we can get the $N_f$ as the minimum number of logic blocks needed to fully implement $F$. Another simulation results are shown in Fig. 9, in which the minimum number of logic blocks required to fully cover the given $F$ for each algorithm is shown as a function of $N$. $N_f$ decreases dramatically at the beginning and slowly decreases as $N$ is increased. *Algorithm #3* has much better performance than the other two. *Algorithm #2* is slightly better than *Algorithm #1*.

Then, we calculate the *Utilization* again after 100% coverage achieved. Then, we can calculate the unused area by the equation shown as $N_f \times N^2 - M^2$.

More simulation results are shown in Fig. 10.

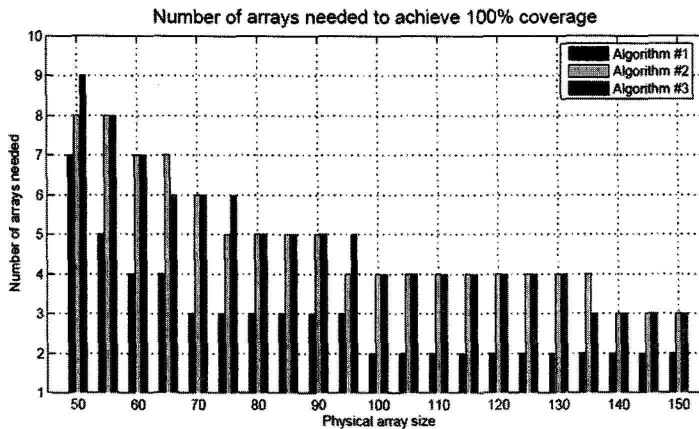


Fig. 9 Number of logic arrays needed to implement the given function set $F$ vs. physical array size $N$

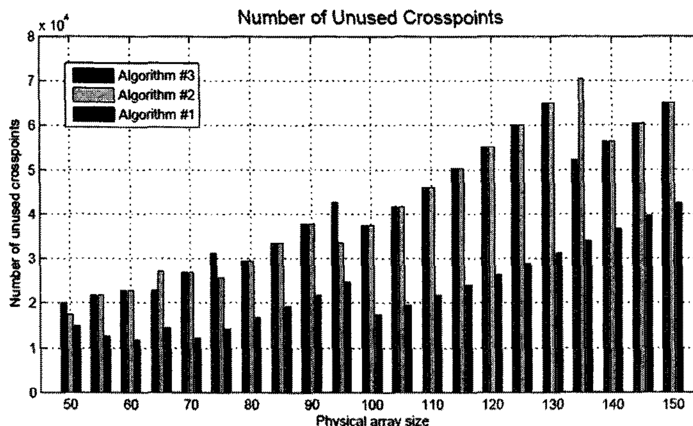


Fig. 10 The total number of unused crosspoints

From the simulation results, the lowest area overhead can be achieved at N = 60 by *Algorithm #3* and $N$ = 50 by the other two algorithms. Among those three algorithms, *Algorithm #3* has the best performance in terms of the area overhead.

## 6. Programmability Comparison

Probability of successful mapping the given logic function using the given partially de-fective nanowire crossbar is referred to as the *Programmability*. If the given crossbar has no defective crosspoint and enough number of programmable crosspoints to realize the given logic function, 100% programmability will result. Programmability is mainly determined by two major factors - defect rate and performance of the repair algorithm. Programmability decreases as the defect rate increases, since more number of crosspoints become unusable. Also, if the given repair algorithm performs better, then the probability of finding a successful circumvention of defective crosspoints increases.

In order to evaluate three given repair algorithms in terms of programmability, a series of parametric simulations have been conducted. As a benchmark suite of logic functions, we have selected logic functions for threshold gates of Null Convention Logic (NCL) [49]. NCL is one of the emerging asynchronous logic paradigms where delay-insensitive computing is possible; therefore, considerably promising when applied to nanoscale reconfigurable hardware since all timing-related failure modes can be inherently addressed [50].

Null Conventional Logic integrates data and control into a single signal. The two states, DATA and NULL are used for achieving local synchronization and handshaking-based I/O control. Primitive logic gates in NCL are referred to as the threshold (TH) gates with hysteresis. There are 27 TH gates in NCL and they directly implement all Sum-of-Product (SOP) logic functions with 4 or less variables. Each NCL TH gate has a threshold term for logic operation a hysteresis term for state-holding behavior. For example, TH23 gate can be expressed in a Boolean expression of $F = AB + BC + AC + (A + B + C)F'$ where $AB + BC + AC$ term is for the threshold behavior (i.e., it describes conditions to assert the output) and $(A + B + C)F'$ term is for the hysteresis behavior (i.e., once asserted, the output maintains the value of 1 until all input wires are de-asserted).

Aforementioned repair algorithms have been tested with various parameter sets to obtain parametric simulation data. Each of the results describes the variation of programmability of various TH gate macros. Six representative TH gates with various complexities, TH12 ($Z = A + B$), TH24 ($Z = AB + BC + CD + AD + BD + CD$), TH34 ($Z = ABC + ABD + BCD + ACD$), TH34w2 ($Z = AB + BC + AD + BCD$), TH44w322 ($Z = AD + BC$), TH54w322 ($Z = AB + AC + BCD$) have been used in the simulations for testing their programmability at different defect rates on crossbar's with various dimensions.

Fig. 11 illustrates the variation of programmability of the aforementioned six threshold gates with crossbar's of various dimensions 10% defect rate using Algorithm #1. Careful observation of this figure suggests us that the gates with lesser number of crosspoints, which depend on the number of AND terms in their expression, have higher programmability as the redundancy increases (e.g., a TH12 gate has high programmability compared to the TH24). This method assures a profitable yield while manufacturing simple circuits at low crossbar defect rates. It will ensure lesser cost and faster mapping process rather than using the other complex

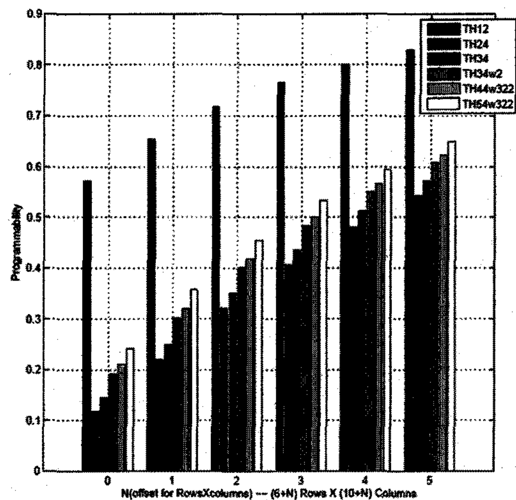


Fig. 11 Programmability of various logic functions at 10% defect rate using Algorithm #1

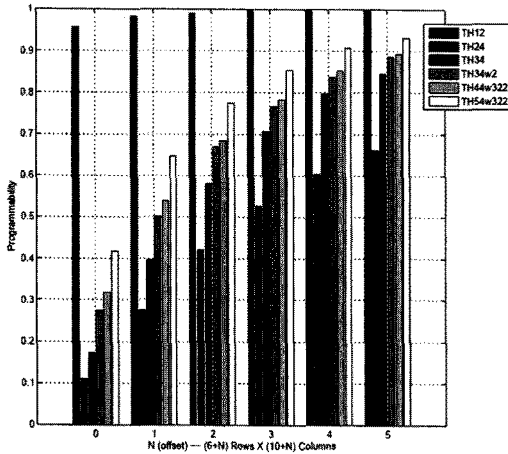


Fig. 12 Programmability of various logic functions at 10% defect rate using Algorithm #2

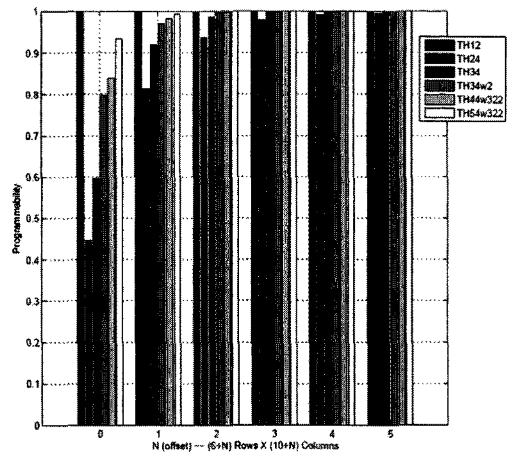


Fig. 13 Programmability of various logic functions at 10% defect rate using Algorithm #3

techniques. We can also use this technique if manufacturers are able to manufacture crossbars with lesser defect rate and also can tolerate the cost of higher redundancy overhead.

Fig. 12 shows the variation of programmability with change in crossbar dimensions at 10defect rate using Algorithm #2. Results shown in this figure infer that this process is better than the initially shown Algorithm #1 based on the programmability at similar defect densities. This can be used as an immediate upgrade if the manufacturer is not willing to tolerate high redundancy at similar defect rates. This method can be considered as a compromise between the Algorithm #1 and #3 which are at the two opposite extremes.

Fig. 13 illustrates the variation of programmability with change in crossbar dimensions at 10%, defect rate. Shown results indicate an improvement in programmability when compared to two previously discussed algorithms. Algorithm #3 is considered to be an annexure for Algorithm #2 which has a considerable increase in programmability at similar defect rates. The probable difference between the two approaches would be increase in time complexity which accounts for increased programmability in case of the latter approach.

## 7. Overall Repair Cost Analysis

As shown in the previous section, considered repair algorithms can achieve certain level of programmability by circumventing defective crosspoints. However, such repair performance comes with unwanted cost factors including repair time overhead and area overhead. Therefore, it is desirable to factor in both repair performance factors and repair overhead factors in the same numerical cost optimization model to find the optimal repair solution. The selected optimal repair solution has full logic mapping coverage (i.e., 100% programmability) and the minimum overhead costs. The similar cost optimization approach proposed in [48,51] can be followed to balance the repair performance and the overhead costs.

The cost of area overhead can be calculated as follows: first, the normalized cost (e.g., in unit cost) is estimated per one crossbar array, spent on the fabrication process including the material cost, machine usage cost, storage and transportation cost for the material, labor cost, etc. For a mature fabrication plant, this cost can be empirically measured and estimated by the planning department. Secondly, the fabrication cost for a single crossbar array is divided by the total number of crosspoints in the logic blocks of the crossbar so that the unit cost for each crosspoint can be calculated. Let us denote as the coefficient for the unit area cost. Finally, the overall unused area overhead cost can be calculated by multiplying the unit cost for a crosspoint by the number of wasted crosspoints.

Time overhead cost can be estimated as follows:

first, the ratio on the practical imple-mentation time over simulation time from our simulator is found. This could be done if we have the operational frequency of testers, the machine touch-down time, the number of I/O pins on the interface device, etc. Secondly, for the manufacturing process, we need to find the cost of machine usage, power consumption and so on per unit time. Let us denote as the cost coefficient per unit manufacturing time multiplied by. Then we can multiply the simulation time by to get the normalized cost of time overhead.

For each cost factor, as long as the ratios among the cost parameters, and are the same, the final cost curve will show the same result comparatively. Now we can add up those cost factors to obtain the overall repair cost (i.e., $Cost_{overall}$ = α × (*# of unused crosspoints*) + β × (*repair algorithm running time*)) [48].

## 8. Parametric Repair Cost Comparison

Extensive parametric simulation has been conducted to validate the proposed cost-driven repair optimization model, in which a set of production parameters, = 1 and = 500 were chosen arbitrarily and used. Simulation results are shown in Fig. 14.

From the results, it can be observed that if we choose *Algorithm #3* and the physical logic block size of 60 × 60, we will be able to achieve the lowest overall cost, while successfully implementing the given function set *F*. In order to simulate

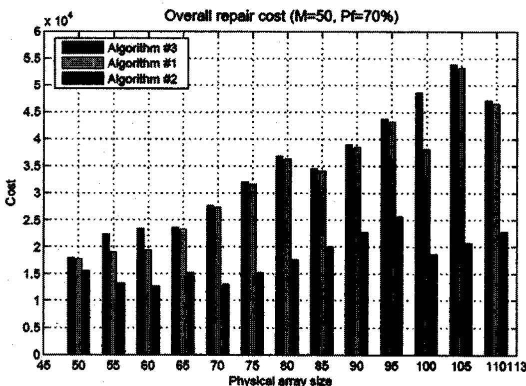


Fig. 14 Overall normalized repair costs of three different algorithms for $M$ = 50 and $P_f$ = 70% case

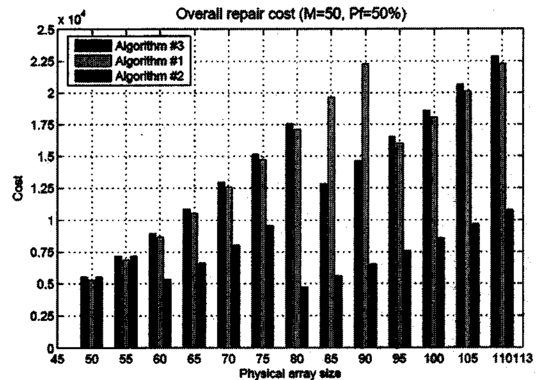


Fig. 15 Overall normalized repair costs for $M$ = 50 and $P_f$ = 50% case

different production environments, different and values can be assumed and a different result (i.e., the most cost-effective repair solution) can be obtained.

Besides these cost coefficients, the property of the given function set *F* affects the overall cost as well. The simulation results shown in Fig. 14 are based on a function set *F* with $P_f$ = 70%. However, some other physical designs may not need such a high usage rate of the crosspoints. For example, let us consider another design with $P_f$ = 50%. It is easier to find a matching for an OR function with lower usage rate, if the defect rate remains the same. To get a comparative result, another parametric simulation was conducted with the same cost coefficients = 1 and = 500. Simulation results are shown in Fig. 15 and it can be concluded that the most cost-effective repair solution can be achieved by *Algorithm #3* when the physical array size of 80 × 80 is used. For the other two algorithms, the optimal physical array size is 50 × 50. Compared with result in Fig. 14, the cost for each algorithm drops dramatically (i.e., approximately 50%). Therefore, the function usage rate $P_f$ also plays as a significant factor in determining the overall cost. In addition to the cost coefficients and function usage rate $P_f$, let us consider the impact of the function set size *M* on the overall repair cost. Aforementioned simulation results are based on *M* = 50. In different design environments, the value of M could be different. *M* = 70 and $P_f$ = 50% and the same cost parameters = 1 and = 500 have been used to conduct another

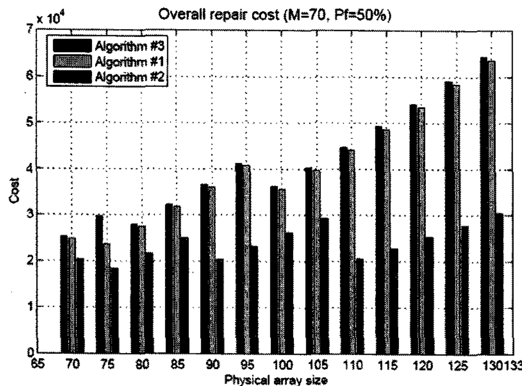


Fig. 16 Overall normalized repair costs for $M$ = 70 and $P_f$ = 50% case

parametric simulation to compare. Results are shown in Fig. 16 and *Algorithm #1* and *#3* have the optimal physical array size at 75 × 75 but *Algorithm #2* has the optimal physical array size at 70 × 70.

Compared with the results shown in Fig. 15, although the function usage rate $P_f$s are the same (i.e., 50%), the overall normalized cost shown in Fig. 16 is much higher. So, it can be observed that the function set size $M$ also plays a very important role.

## 9. Conclusion

Recent trend and issues in nanowire reconfigurable crossbar architectures based on bottom-up assembly paradigm are discussed. For the emerging nanoscale crossbar-based systems, higher defect densities are anticipated due to nondeterministic nature of nanoscale bottom-up assembly paradigm. This indicates that effective and efficient testing and logic mapping methods are needed to locate and tolerate such defects. Considering the defects in nanoscale wires can be easily screened out by testing, effectively avoiding the defective crosspoints in cost-effective manner is important. Thus, three different repair algorithms have been evaluated to tolerate the *non-programmable crosspoints* in this work. From the simulation results, 2D algorithms show significantly better performance over 1D algorithm in terms of *Utilization*, *Coverage* and *Programmability*, but the performance benefits come with higher repair overhead costs. The area and time cost factors are identified as major overhead in logic mapping-based repair. Thus, such cost factors were parameterized and have been considered to evaluate the overall repair cost of each repair algorithm. Extensive parametric simulation results are shown to compare three nanowire crossbar repair algorithms and a novel method to find the optimal repair solution has been also demonstrated.

## References

[1] International Technology Roadmap for Semiconductors, "International Technology Roadmap for Semiconductors (ITRS) 2004," http://public.itrs.net, 2004.

[2] G. Bourianoff, "The future of nanocomputing," *IEEE Computer*, vol.38, no.8, pp.44-53, August 2003.

[3] S. C. Goldstein and M. Budiu, "Nanofabrics: spatial computing using molecular nanoelectronics," in *Proc. 28th Int. Symp. Computer Architecture*, 2001, pp.178-189, 2001.

[4] S. C. Goldstein and D. Rosewater, "Digital logic using molecular electronics," *Int. Solid-State Circuits Conf.*, p.125, 2002.

[5] M. Mishra and S. Goldstein, "Scalable defect tolerance for molecular electronics," *Workshop Non-Silicon Computation (NSC-1)*, p.78, 2002.

[6] T. Rueckes, K. Kim, E. Joselevich, G. Y. Tseng, C. L. Cheung and C. M. Lieber, "Carbon nanotube based nonvolatile random access memory for molecular computing," *Science*, vol.289, pp.94-97, 2000.

[7] P. J. Kuekes, J. R. Heath, and R. S. Williams, "Molecular wire crossbar memory," U.S. Patent 6 128 214, October 2000.

[8] Y. Chen, G. 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*, no.14, pp.462-468, 2003.

[9] M. Ziegler and M. Stan, "Design and analysis of crossbar circuits for molecular nanoelectronics," *IEEE Conf. Nanotechnology (IEEE-NANO)*, pp.323-327, 2002.

[10] P. J. Kuekes, J. R. Heath and R. S. Williams, "Molecular-wire crossbar interconnect (MWCI) for signal routing and communications," U.S. Patent 6 314 019, November 2001.

[11] P. J. Kuekes and R. S. Williams, "Demultimplexer for a molecular wire crossbar network (MWCN DEMUX)," U.S. Patent 6 256 767, July 2001.

[12] M. Ziegler and M. Stan, "A case for CMOS/nano co-design," *Int. Conf. Computer Aided Design (ICCAD)*, pp.348-352, 2002.

[13] M. Stan, "A scaling scenario for nanoelectronic technologies," *Georgia Tech Conf. Nanoscience and Nanotechnology*, p.103, 2001.

[14] N. Melosh, A. Boukai, F. Diana, B. Gerardot, A. Badolato, P. Petroff and J. Heath, "Ultrahigh-density nanowire lattices and circuits," *Science*, vol.300, p.112-115, April 2003.

[15] S. R. Nicewarner-Pena, S. Raina, G. P. Goodrich, N. V. Fedoroff and C. D. Keating, "Hybridization and extension of Au nanoparticle-bound oligonucleotides," *Journal of American Chem. Soc.*, vol.124, pp.7314-7323, 2002.

[16] Y. Huang, X. Duan, Q. Wei and C. M. Lieber, "Directed assemble of one-dimensional nanostructures into functional networks," *Science*, vol.291, pp.630-633, January 2001.

[17] 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.

[18] Y. Cui and C. M. Lieber, "Functional nanoscale electronic devices assembled using silicon nanowire building blocks," *Science*, vol.291, pp.851-853, February 2001.

[19] J. Huang, M. B. Tahoori and F. Lombardi, "On the defect tolerance of nano-scale two-dimensional crossbars," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp.96-104, October 2004.

[20] M. Jacome, C. He, G. de Veciana, and S. Bijansky, "Defect tolerant probabilistic design paradigm for nanotechnolo-gies," *IEEE/ACM Design Automation Conference (DAC)*, pp.1-6, 2004.

[21] M. Tehranipoor, "Defect Tolerance for Molecular Electronics-Based NanoFabrics Using Built-In Self-Test Procedure," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Submitted for publication.

[22] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams, "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol.280, pp.1716-1721, 1998.

[23] D. Whang, S. Jin and C. M. Lieber, "Large-Scale Hierarchical Organization of Nanowires for Functional Nanosystems," *Japanese Journal of Applied Physics*, vol.43, no.7B, 2004.

[24] A. Bachtold, P. Hadley, T. Nakanishi and C. Dekker, "Logic circuits with carbon nanotube transistors," *Science*, vol.294, pp.1317-1320, November 2001.

[25] V. Derycke, R. Martel, J. Appenzeller and Ph. Avouris, "Carbon nanotube inter- and intramolecular logic gates," *Nano Letter*, vol.1, no.9, pp.453-456, 2001.

[26] C. Soh, C. Quate, C. Morpurgo, C. Marcus, C. Kong and C. Dai, "Integrated nanotube circuits: controlled growth and ohmic contacting of single-walled carbon nanotubes," *Appled Physics Letter*, vol.75, no.5, pp.627-629, 1999.

[27] S. J. Trans, A. R. M. Verschueren and C. Dekker, "Room-temperature transistor based on a single carbon nanotube," *Nature*, vol.393, pp.49-51, May 1998.

[28] H. Finkelstein, P. M. Asbeck and S. Esener, "Architecture and analysis of a self-assembled 3D array of carbon nanotubes and molecular memories," *IEEE Conference on Nanotechnology*, pp.12-14, August 2003.

[29] C. Dekker, "Carbon nanotubes as molecular quantum wires," *Physics Today*, pp.22-28, May 1999.

[30] M. Ziegler, G. Rose and M. Stan, "A universal device model for nanoelectronic circuit simulation," *IEEE Conf. Nanotechnology (IEEE-NANO)*, pp.83-88, 2002.

[31] M. Bhattacharya and P. Mazumder, "Augmentation of SPICE for simulation of circuits containing resonant tunneling diodes," *IEEE Trans. Computer-Aided Design*, vol.20, pp.39-50, January 2001.

[32] J. Chen, W. Wang, M. A. Reed, M. Rawlett, D. W. Price and J. M. Tour, "Room-Temperature Negative Differential Resistance in Nanoscale Molecular Junctions," *Appl. Phys. Lett.*, vol.77, p.1224, 2000.

[33] J. Chen, M. A. Reed, A. M. Rawlett and J. M. Tour, "Large on-off ratios and negative differential resistance in a molecular electronic device," *Science*, vol.286, pp.1550-1552, November 1999.

[34] R. H. Mathews, J. P. Sage, T. C. L. G. Sollner, S. D. Calawa, C. L. Chang-Lee Chen, L. J. Mahoney, P. A. Maki and K. M. Molvar, "A new RTD-FET logic family," *Proc. IEEE*, vol.87, pp.596-605, April 1999.

[35] Y. S. Yu, Y. I. Jung, J. H. Park, S. W. Hwang and D. Ahn, "Simulation of single-electron/CMOS hybrid circuits using SPICE macromodeling," *J. Korean Phys. Soc.*, vol.20, no.35, pp.991-994, 1999.

[36] P. Franzon and D. Nackashi, "Moletronics: a circuit design perspective," *Int. Conf. SPIE Smart Electronics and MEMS*, vol.4236, pp.80-88, 2000.

[37] J. C. Ellenbogen and J. C. Love, "Architectures for molecular electronic computers. I. Logic structures and an adder designed from molecular electronic diodes," *Proc. IEEE*, vol.88, pp.386-426, March 2000.

[38] C. J. Christian, J. Amsinck, D. P. David, P. Nackashi, N. H. Neil, H. D. Spigna and P. D. Franzon, "Electrically accessible molecular memories," *IEEE J. Nanotechnol.*, submitted for publication.

[39] R. M. Metzger et al, "Unimolecular electrical rectification in hexadecylquinolinium tricyanoquinodimethanide," *J. Amer. Chem. Soc.*, vol.119,

pp.10455-10466, 1997.
[40] A. DeHon, "Array-Based Architecture for FET-Based, Nanoscale Electronics," *IEEE Transactions on Nanotechnology*, vol.2, no.1, pp.23-32, March 2003.
[41] A. Dehon, M. J. Wilson, "Nanowire-Based Sublithographic Programmable Logic Arrays," *FPGA'04*, Monterey, CA, February, 2004.
[42] H. Naeimi and A. DeHon, "A greedy algorithm for tolerating defective crosspoints in nanoPLA design," *IEEE International Conference on Field-Programmable Technology*, pp.49-56, 2004.
[43] H. C. Liang, W. C. Ho and M. C. Cheng, "Identify unrepairability to speed-up spare allocation for repairing memories," *IEEE Transactions on Reliability*, vol.54, no.2, pp.358-365, June 2005.
[44] J. F. Li, J. C. Yeh, R. F. Huang and C. W. Wu, "A Built-In Self-Repair Design for RAMs With 2-D Redundancy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.13, no.6, pp.742-745, June 2005.
[45] C. T. Huang and C. F. Wu, J. F. Li and C. W. Wu, "Built-in redundancy analysis for memory yield improvement," *IEEE Transactions on Reliability*, vol.52, no.4, pp.386-399, December 2003.
[46] M. Choi and N. Park, "Dynamic yield analysis and enhancement of FPGA reconfigurable memory systems," *IEEE Transactions on Instrumentation and Measurement*, vol.51, no.6, pp.1300-1311, December 2002.
[47] W. K. Huang, Y.-N. Shen and F. Lombardi, "New approaches for the repairs of memories with redundancy by row/column deletion for yield enhancement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.9, no.3, pp.323-328, Mar 1990.
[48] Y. Yallambalase and M. Choi, "Cost-driven repair optimization of reconfigurable nanowire crossbar systems with clustered defects," *Journal of Systems Architecture*, vol.54, no.8, pp.729-741, 2008.
[49] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," *Integration, The VLSI Journal*, vol.37, no.3, pp.135-165, 2004.
[50] R. Bonam, S. Chaudhary, Y. Yellambalase and M. Choi, "Clock-Free Nanowire Crossbar Architecture based on Null Convention Logic (NCL)," *7th IEEE International Conference on Nanotechnology (IEEE-Nano)*, Apr 2007.
[51] S. Zhang, M. Choi, N. Park and F. Lombardi "Cost-Driven Optimization of Fault Coverage in Combined Built-In Self-Test/Automated Test Equipment Testing," *IMTC 04*, 2004.

**이 종 석**

1988년 서울대학교 계산통계학과 졸업(이학사). 1990년 서울대학교 대학원 계산통계학과 졸업(이학석사). 2001년 서울대학교 대학원 전기컴퓨터공학부 졸업(공학박사). 1993년 3월~2006년 2월 우석대학교 컴퓨터공학과 부교수. 2006년 3월~현재 우석대학교 컴퓨터교육과 교수. 관심분야는 객체지향소프트웨어 공학, 소프트웨어 메트릭, Fault Tolerance, Testing, Quality Assurance, Reliability Modeling and Analysis

**최 민 수**

1995, 1998, 2002년 미 Oklahoma State University Computer Science과 졸업(학사, 석사, 박사). 2002년~2008년 미 Missouri University of Science & Technology, Electrical & Computer Engineering과 조교수. 2008년~현재 미 Missouri University of Science & Technology, Electrical & Computer Engineering과 부교수. 관심분야는 Computer Architecture & VLSI, Nanoelectronics, Embedded Systems, Fault Tolerance, Testing, Quality Assurance, Reliability Modeling and Analysis, Configurable Computing, Parallel & Distributed Systems and Dependable Instrumentation & Measurement