Digital Architectures for Hybrid CMOS/Nanodevice Circuits

A Dissertation Presented

by

Dmitri B. Strukov

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

August 2006

Stony Brook University

The Graduate School

Dmitri B. Strukov

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Konstantin K. Likharev
Distinguished Professor, Department of Physics and Astronomy
Advisor

Yuanyuan Yang
Professor, Department of Electrical and Computer Engineering
Co-Advisor

Alex Doboli
Associate Professor, Department of Electrical and Computer Engineering
Dissertation Chair

Ridha Kamoua
Associate Professor, Department of Electrical and Computer Engineering

James Lukens
Professor, Department of Physics and Astronomy

This dissertation is accepted by the Graduate School.

Dean of the Graduate School

Abstract of the Dissertation

Digital Architectures for Hybrid CMOS/Nanodevice Circuits

by

Dmitri B. Strukov

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

2006

This dissertation describes architectures of digital memories and reconfigurable Boolean logic circuits for the prospective hybrid CMOS/nanowire/nanodevice ("CMOL") technology. The basic idea of CMOL circuits is to combine the advantages of CMOS technology (including its flexibility and high fabrication yield) with those of molecular-scale nanodevices. Two-terminal nanodevices would be naturally incorporated into nanowire crossbar fabric, enabling very high function density at acceptable fabrication costs. In order to overcome the CMOS/nanodevice interface problem, in CMOL circuits the interface is provided by sharp-tipped pins that are distributed all over the circuit area, on top of the CMOS stack.

The most straightforward possible application of CMOL circuits is terabit-scale "resistive" memories, in which nanodevices (e.g., single molecules) would be used as single-bit memory cells, while the semiconductor subsystem would perform all the peripheral (input/output, coding/ decoding, line driving, and sense amplification) functions. Using bad-bit exclusion and error-correcting codes synergistically we show that CMOL memories with a nano/CMOS pitch ratio close to 1/3 may overcome purely

semiconductor memories in useful density if the fraction of bad nanodevices is below $\sim 15\%$, even for the 30 ns upper bound on the total access time. As the nanotechnology matures, and the pitch ratio approaches an order of magnitude, the CMOL memories may be far superior to the densest semiconductor memories by providing, e.g., 1 Tbit/cm^2 density even for the plausible defect fraction of 2%.

Even greater defect tolerance (about 20% for 99% circuit yield) can be achieved in uniform a cell-FPGA-like CMOL circuits. In such circuits, two-terminal nanodevices provide programmable diode functionality for logic circuit operation, and allow circuit mapping and reconfiguration around defective nanodevices, while CMOS subsystem is used for signal restoration and latching. The cell-based architecture is based on a uniform CMOL fabric of "tiles", while each tile consists of 12 four-transistor basic cells and one latch cell. To evaluate the potential performance of CMOL FPGA we have developed a completely custom design automation tools. Using these tools we have successfully mapped on CMOL FPGA the well known Toronto 20 benchmark circuits and estimated their performance. The results have shown that, in addition to high defect tolerance, CMOL FPGA circuits may have extremely high density (more than two orders of magnitude higher that that of usual CMOS FPGA with the same CMOS design rules) while operating at higher speed at acceptable power consumption.

Dedicated to my parents.

Contents

List of Figures **x**

List of Tables **xi**

Acknowledgements **xii**

1 Introduction **1**
1.1 Motivation 1
1.2 Prior Work 3
1.2.1 Devices 3
1.2.2 Circuits 7
1.2.3 Architectures 9

2 CMOL Approach and Hardware Models **14**
2.1 Concept 14
2.2 CMOL Cousins 16
2.3 Performance Model 20
2.3.1 Nanodevices 20
2.3.2 Nanowires 21
2.4 Defect Model 22

3 CMOL Memory **24**
3.1 Architecture and Operation 24
3.2 Defect Tolerance Calculation 27
3.3 Area Calculation 29
3.3.1 Total Area and Capacity 29
3.3.2 Crossbar Arrays 31
3.3.3 Decoders 31
3.3.4 Control Circuitry 32
3.3.5 ECC Decoders 32
3.4 Speed and Power 33
3.4.1 ECC Decoder 33

3.4.2 Intrablock Delay . . . 34
3.4.3 Interblock Delay . . . 35
3.4.4 Example . . . 35
3.4.5 Power . . . 36
3.5 Optimization . . . 36
3.6 Results . . . 38

4 CMOL Boolean Logic **41**
4.1 Hardware Architecture . . . 41
4.1.1 One-Cell Fabric . . . 41
4.1.2 Two-Cell Fabric . . . 45
4.2 Design Automation . . . 46
4.2.1 General Flow . . . 46
4.2.2 Technology Mapping and Circuit Processing . . . 48
4.2.3 Placement . . . 49
4.2.4 Global Routing . . . 50
4.2.5 Defect Tolerance: Defective Cell Avoidance . . . 54
4.2.6 Defect Tolerance: Detailed Routing around Defective Nanodevices . . . 54
4.3 Performance Calculation . . . 60
4.3.1 Area . . . 60
4.3.2 Delay . . . 61
4.3.3 Power . . . 63
4.3.4 Optimization . . . 63
4.3.5 Simplification . . . 65
4.4 Results . . . 66

5 Discussion **75**
5.1 Conclusions . . . 75
5.1.1 CMOL Extension of Resistive Memory . . . 75
5.1.2 CMOL Logic Circuits . . . 76
5.2 Main Challenges and Possible Future Work . . . 78

A BCH Decoder **83**
A.1 Introduction . . . 83
A.2 Model . . . 84
A.2.1 General Structure . . . 84
A.2.2 Syndrome Calculation . . . 84
A.2.3 Finding Error-Location Polynomial with Berlekamp-Massey Algorithm . . . 85
A.2.4 Finding Error Location Numbers and Correction . . . 87

A.3 Results and Discussion 89

B Prior Work on CMOL FPGA circuits **92**
B.1 Architecture and Reconfiguration Algorithm 92
B.2 32-bit Kogge-Stone Adder 92
B.3 A Full Crossbar 94
B.4 Results 95

C CMOL FPGA CAD **101**
C.1 Structure 101
C.2 Command Line Options 103

Bibliography **117**

List of Figures

1.1 Two-terminal latching switch 4
1.2 The concept of "floating electrodes" 6
1.3 Crossbar array structure 7
1.4 Equivalent circuits of crossbar memory array 10

2.1 Generic CMOL circuits 15
2.2 CMOS-to-nano interface yield 16
2.3 A plausible CMOL fabrication process flow 17
2.4 CMOS-to-nano interface yield 19
2.5 Comparison of CMOL and HP's FPNI circuits 20
2.6 Nanowire capacitance in a crossbar as a function of nanowire half-pitch 21

3.1 Memory top-level architecture 25
3.2 Example of addressing nanodevices in a memory block 26
3.3 Possible structure of CMOS relay cell 28
3.4 BCH decoder delay and area vs. specific code parameters 30
3.5 Assumed structure of peripheral circuits 31
3.6 Equivalent circuit for the readout operation 33
3.7 Total chip area and its components of one useful memory cell as functions of CMOL memory array size 37
3.8 Redundant cells and mapping table area overheads 38
3.9 Total chip area of one useful memory cell as a function of bad bit fraction 39

4.1 One-cell CMOL FPGA fabric 42
4.2 CMOL FPGA cells 43
4.3 Logic and routing primitives in CMOL FPGA circuits 44
4.4 Two-cell CMOL FPGA fabric 45
4.5 Tile connectivity domain 47
4.6 CMOL FPGA deisgn flow 48
4.7 Example of preprocessing step 49
4.8 Wiring cost function calculation example 51
4.9 Pseudo-code of the global routing algorithm 51
4.10 Pseudo-code of the global routing subroutines 53

4.11 Example of first stage of the global routing 55
4.12 Example of second stage of the global routing 56
4.13 Pseudo-code of the detail routing algorithm 57
4.14 Example of a circuit fragment reconfiguration around "stuck-on-open" defective nanodevice 58
4.15 Equivalent circuit of a CMOL logic stage 61
4.16 Example of optimization results 64
4.17 Example of defect-free placement for dsip.blif circuit 71
4.18 Example of global routing for dsip.blif circuit 72
4.19 Example of CMOL mapping with a presence of defective cells 73
4.20 Example of CMOL mapping with a presence of bad (stuck-on-open) nanodevices 74

5.1 Comparison of CMOL FPGA and HP's FPNI logic architectures . . . 78
5.2 A possible structure of large-scale CMOL FPGA system 81
5.3 Boolean logic based on nanodevices with NDR characteristics 82

A.1 Syndrom calculation 85
A.2 Finding error-location polynomial with Berlekamp-Massey algorithm . 86
A.3 Finding error location numbers and correction 88
A.4 BCH decoder area and delay as a functions of the code performance . 90

B.1 One-cell CMOL FPGA fabric with additional 45° crossbar tilt 93
B.2 The 32-bit Kogge-Stone adder 94
B.3 Mapping of the 32-bit Kogge-Stone adder on CMOL FPGA fabric . . 97
B.4 Logic depth of considered circuits 98
B.5 Full crossbar 98
B.6 Defect tolerance results 99
B.7 Small fragment of the adder after reconfiguration 100

C.1 General structure of CMOL FPGA CAD 102
C.2 Options compatibility 103

List of Tables

3.1 Optimal parameters of CMOL memory 40

4.1 Performance results for Toronto 20 benchmark set mapped on CMOL fabric with no defects 67
4.2 Performance results of the Toronto 20 benchmark set mapped on CMOL fabric in a presence of defective cells 68
4.3 Defect rate of bad (stuck-on-open) nanodevices requirement to achieve 90% and 99% final yield 69

A.1 Complexity of key operations in BCH decoder 89
A.2 Complexity of proposed BCH decoder 89

Acknowledgements

I would not have been able to complete my PhD without the aid and support of many people over the past six and half years. First of all, I owe eternal gratitude towards my advisor, Konstantin Likharev. It is because of his tremendous experience and brainpower combined with exceptional personal qualities he became more a mentor for me than a professor. I will always see his outstanding work ethic, hard work, support, scholarship, as well as his never fading enthusiasm and true professionalism as an example I should aspire.

Over the years, I have enjoyed the fruitful discussions of issues related to my thesis work with our collaborators and research colleagues: Iris Bahar, Jacob Barhen, Valeriu Beiu, Robert Brayton, Sat Chatterjee, Deming Chen, Shamik Das, Andre DeHon, Neil Di Spigna, Dan Hammerstrom, Ramesh Karri, Alexandr Khitun, Ralf Koetter, Phil Kuekes, Guy Lemieux, Alan Mishchenko, Csaba Moritz, Kundan Nepal, Alex Orailoglu, Garrett Rose, Greg Snider, Mircea Stan, Stan Williams, Tong Zhang, and Nikolai Zhitenev.

I am indebted to Pernille Jensen and Debbie Kloppenburg for all the instances in which their assistance helped me along the way. I also thank my former and current officemates, Yusuf Kinkhabwala and Jung Hoon Lee. They each helped make my time in the PhD program more fun and interesting.

Last but not the least I would like to thank my friends and family members for their support during the long years of my academic career. It is their understanding, distraction, encouragement, and help that gave me the strength to finish this work.

Chapter 1

Introduction

1.1 Motivation

The phenomenal success of semiconductor electronics during the past three decades dwelled on scaling down of Si complementary metal-oxide-semiconductor (CMOS) technology, in particular field-effect transistors (MOSFETs), and the resulting increase in density of logic and memory chips. The most authoritative industrial forecast, the International Technology Roadmap for Semiconductors (ITRS) [1] predicts that this exponential ("Moore-Law") progress of silicon MOSFETs and integrated circuits will continue at least for the next 15 years. By the end of this period, devices with 10-nm minimum features (transistor gate length) should become commercially available.

The prospects to continue the Moore Law with current VLSI paradigm, based on a combination of lithographic patterning, CMOS circuits, and Boolean logic, beyond the 10 nm frontier are more uncertain [36, 70]. The main reason is that at gate length beyond 10 nm, the sensitivity of parameters (most importantly, the voltage threshold) of MOSFETs to inevitable fabrication spreads grows exponentially. As a result, the gate length should be controlled with a few-amstrong accuracy, far beyond even the long-term projections of the semiconductor industry [1]. For example, for the most promising double gate silicon-on-insulator (SOI) MOSFETs the definition accuracy of 5-nm-long gate channel should be better than 0.2 nm in order to keep fluctuations of the voltage threshold below reasonable value of 50 mV [70], i.e. much smaller than ITRS projected value of 0.5 nm [1]. Even if such accuracy could be technically implemented using sophisticated patterning technologies, this would send the fabrication facilities costs (growing exponentially even now) skyrocketing, and lead to the end of the Moore's Law some time during the next decade.

Similar problems with scaling await today's mainstream memory technologies when their feature sizes will approach the 10-nm-scale regime. Indeed, the basic cell (holding one bit of information) of todays mainstream memories, like static and dy-

namic random access memories (denoted as SRAM and DRAM, correspondingly), as well as those of relatively new but already commercialized technologies like ferroelectric, magnetic, and structural phase transition memories, needs at least one transistor and hence will run into the aforementioned limitation in the future. Moreover, the scaling of DRAM is problematic already now (i.e., for 90 nm half-pitch CMOS technology node) because of more severe problem with the capacitance scaling [77]. Similarly, the scaling of Flash memories may stop much earlier, somewhere close to 45 nm CMOS technology node [97]. The problem is that reducing dimensions of the Flash memory cell requires corresponding scaling down of the tunnel oxide thickness, but too thin tunnel oxide will cause leaking of the charge from the floating gate.

The main alternative nanodevice concept, single-electronics [68, 70], offers some potential advantages over CMOS, including a broader choice of possible materials. Unfortunately, for room-temperature operation the minimum features of these devices (single-electron islands) should be below $\sim$1 nm [68]. Since the relative accuracy of their definition has to be between 10 and 20%, the absolute fabrication accuracy should be of the order of 0.1 nm, again far too small for the current and realistically envisioned lithographic techniques.

Fortunately, a critical dimensions of devices can be controlled much more accurately via some other techniques, e.g., film deposition. Even more attractive would be a "bottom-up" approach with the smallest active devices formed in a special way ensuring their fundamental reproducibility. The most straightforward example of such device is a specially designed and chemically synthesized molecule, implementing single electron transistor.

However, integrated circuits consisting of molecular devices alone are hardly viable, because of limited device functionality. For example, the voltage gain of a 1-nm-scale transistor, based on any known physical effect (e.g., the field effect, quantum interference, or single-electron charging), can hardly exceed one, i.e. the level necessary for sustaining the operation of virtually any active digital circuit. This is why the only plausible way toward high-performance nanoelectronic circuits is to integrate nanodevices, and the connecting nanowires, with CMOS circuits whose (relatively large) field-effect transistors would provide the necessary additional functionality, in particular high voltage gain.

The novel hybrid technology paradigm will certainly require rethinking of the current circuit architectures. Therefore, the purpose of this PhD work was to investigate plausible architectures for hybrid CMOS/nanodevice circuits. In the reminder of this chapter we will review nanoscale devices suitable for hybrid circuits and then discuss the most important unique circuit- and architectural-level challenges as well as some of the proposed solutions. One of the most promising hybrid concepts, dubbed "CMOL", which was used as primary technology for all our architectures will be discussed in the next chapter. The rest of the Dissertation is devoted to the original

results for CMOL-based digital memories (Chapter 3) and reconfigurable Boolean-logic circuits (Chapter 4). The thesis is concluded (Chapter 5) with a summary of the major results and possible future directions. Finally, Appendix A discusses in details the model of a key component of a memory circuitry, BCH decoder, Appendix B presents results of our preliminary work on CMOL FPGA circuits, while Appendix C describes the custom design automation tool which was developed to enable evaluation of Boolean CMOL logic circuits.

1.2 Prior Work

1.2.1 Devices

The first critical issue in the development of semiconductor/nanodevice hybrids is making a proper choice in the trade-off between nanodevice simplicity and functionality. On one hand, simple molecule-based nanodevices (like the octanedithiols [126]), which may provide nonlinear but monotonic $I - V$ curves with no hysteresis (i.e. no internal memory), are hardly sufficient for highly functional integrated circuits, because a semiconductor memory subsystem would hardly be able to store enough data for processing by more numerous nanodevices. On the other hand, very complex molecular devices (like a long DNA strand [89]) may have numerous configurations that can be, as a matter of principle, used for information storage. However, such molecules are typically very "soft", so that thermal fluctuations at room temperature (that is probably the only option for broad electronics applications) may lead to uncontrollable switches between their internal states, making reliable information storage and usage difficult, if not totally impossible.

Moreover, so far there are only practical solutions for fabricating two-terminal devices, because they may have just one critical dimension (distance between the electrodes) which may be readily controlled by, e.g. film deposition or oxidation rate. Equally, chemically-directed self-assembly of two-terminal devices would be immeasurably simpler than the multi-terminal ones. This is why many realistic proposals of hybrid circuits are based on two-terminal "latching switches" or "programmable diodes" (see, e.g., Refs. 26, 27, 29, 40, 79 as well as circuits desribed in this Dissertation [108–115], and also recent reviews [23,28,30,58,65,73,76,105]).[1] The functionality of such devices is illustrated on Fig. 1.1a. At low applied voltages, the device behaves

[1] As it will be shown later in this work the diode-like characteristic is necessary for the operation of the hybrid memory circuits and is helpful for the proposed logic circuits. However, simple programmable resistance switches could be enough for, e.g., nanoelectronic neuromorphic networks [63, 64, 72, 122], programmable interconnect hybrid CMOS/nanodevice architectures [37, 102], as well as Goto-pair-based circuit architectures [59, 99, 100]. (The latter group, however, runs into serious architectural problems, which are discussed at the end of this chapter.)

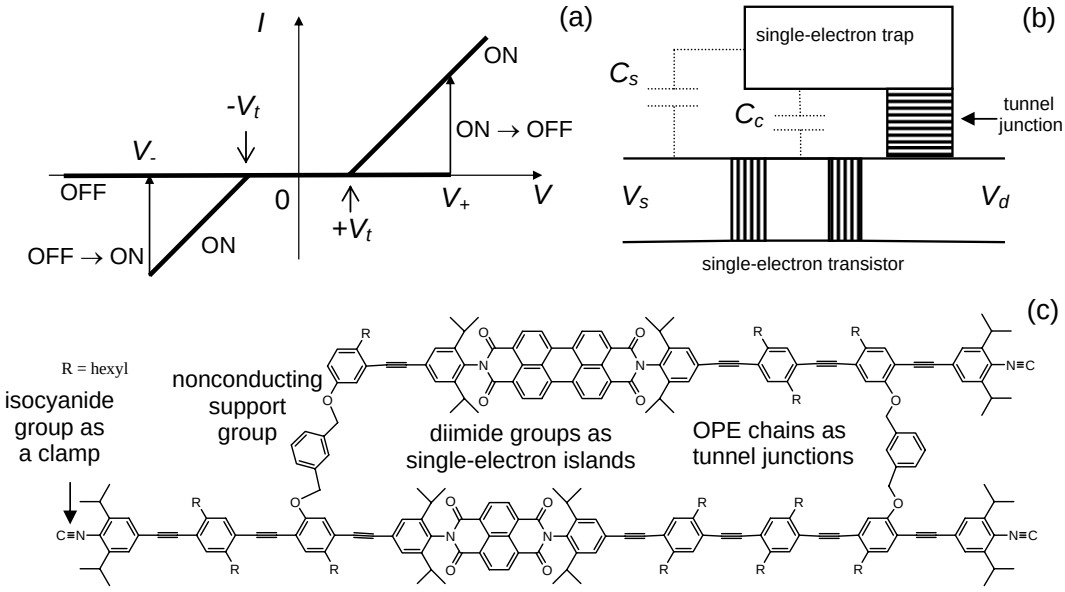

Figure 1.1: Two-terminal latching switch: (a) $I-V$ curve (schematically), (b) single-electron device schematics [35], and (c) a possible molecular implementation of the device (courtesy A. Mayr).

as a usual diode, but a higher voltage may switch it between low-resistive (ON) and high-resistive (OFF) states.

Numerous devices with a similar functionality have been already demonstrated using several materials, notably including amorphous metal-oxide films (see, e.g. Refs. 9,19), relatively thick organic films (both with [13,75] and without [61,96] embedded metallic clusters), self-assembled monolayers (SAM) of molecules [22,66,127], and thin chalcogenide layers [21, 60]. The physics of the ON-OFF switching in these devices is still a matter of substantial discussion, with the reversible filament formation most probable for organic systems, and trapped electric charge accumulation looking like the most plausible candidate for amorphous oxide films. In both interpretations, the conductance is due to some random active conducting centers (filaments or hopping percolation paths) separated by distances of the order of a few nanometers. In order to be reproducible, the device should have a large number of such centers. This is why the extension of the excellent reproducibility demonstrated for such statistical devices with a lateral size larger than 100 nm [19] to the most interesting range, i.e. below 10 nm, presents a challenge.

The problem may be addressed using uniform self-assembled monolayers of specially designed molecules [72] implementing binary single-electron latching switches [35]. Such switch may be readily implemented, for example, as a combination of

two single-electron devices: a "transistor" and a "trap" (Fig. 1.1b).[2] If the applied drain-to-source voltage $V = V_d - V_s$ is low, the trap island in equilibrium has no extra electrons ($n = 0$), and its net electric charge $Q = -ne$ is zero. As a result, the transistor is in the virtually closed (OFF) state, and source and drain are essentially disconnected. If V is increased beyond a certain threshold value V_+, its electrostatic effect on the trap island potential (via capacitance C_s) leads to tunneling of an additional electron into the trap island: $n \to 1$. This change of trap charge affects, through the coupling capacitance C_c, the potential of the transistor island, and suppresses the Coulomb blockade threshold to a value well below V_+. As a result, the transistor, whose tunnel barriers should be thinner than that of the trap, is turned into ON state in which the device connects the source and drain with a finite resistance $R_{\rm ON}$. (Thus, the trap island plays the role similar to that of the floating gate in the usual nonvolatile semiconductor memories [14].) If the applied voltage stays above V_+, this connected state is sustained indefinitely; however, if V remains low for a long time, the thermal fluctuations will eventually kick the trapped electron out, and the transistor will get closed, disconnecting the electrodes. This ON $\to$ OFF switching may be forced to happen much faster by making the applied voltage V sufficiently negative, $V \approx V_-$.

Figure 1.1c shows a possible molecular implementation of the device shown in Fig. 1.1b. Here two different diimide acceptor groups play the role of single-electron islands, while short oligo-ethynylenephenylene (OPE) chains are used as tunnel barriers. The chains are terminated by isocyanide-group "clamps" ("alligator clips") that should enable self-assembly of the molecule across a gap between two metallic electrodes.

A major challenge for such molecular devices is the reproducibility of the interface between the monolayer and the second (top) metallic electrode, because of the trend of the metallic atoms to diffuse inside the molecules during the electrode deposition [129], and the difficulty in ensuring a unique position of the molecule relative to the electrodes, and hence a unique structure and transport properties of molecular-to-electrode interfaces. Recent very encouraging results towards the solution of the first problem have been obtained using an intermediate layer of a conducting polymer [6].

The latter problem can be solved, e.g., by attaching relatively large "floating electrodes" (large acceptor groups or metallic clusters - see Fig. 1.2) to the molecular device. If the characteristic internal resistance $R_{\rm ON}$ of such a molecule is much higher than the range of possible values of molecule/electrode resistances R_i, and the floating electrode capacitances are much higher than those of the internal single-electron

[2] Metal-based, low-temperature prototypes of such switches, with multi-hour state retention times, have been implemented and successfully tested experimentally, with electron trapping times beyond 12 hours [34, 50, 70]. However, so far molecular implementations have been only demonstrated (see, e.g., Refs. 55, 88) for the main component of the device, the single-electron transistor [70], rather than for the latching switch as the whole.

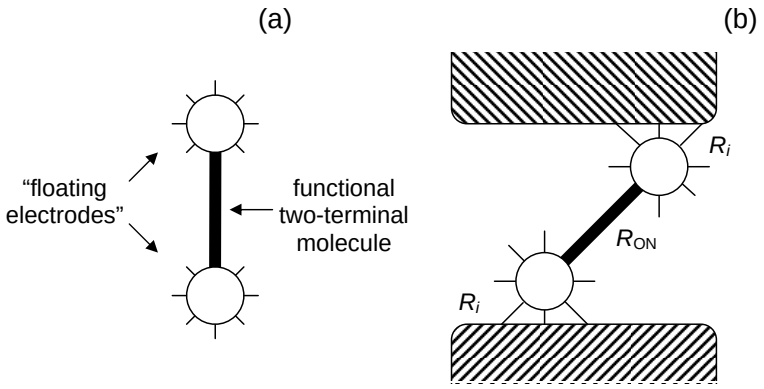

Figure 1.2: A molecule with "floating electrodes" (a) before and (b) after its self-assembly on "real electrodes", e.g., metallic nanowires (schematically).

islands, then the transport through the system will be determined by $R_{\rm ON}$ and hence be reproducible.[3]

Another possible way toward high yield is to form a SAM on the surface of the lower nanowire level, and only than deposit and pattern the top layer. Such approach has already given rather reproducible results (in the nanopore geometry) for simple, short molecules [126]. The apparent problem here is that each crosspoint device would have several parallel devices even if the nanowire width is scaled down to a few nanometers, and this number may be somewhat different from one crosspoint device to the other. However, the CMOL circuits discussed in Chapters 3, 4 can function properly even in this case.

Finally, the potentially enormous density of nanodevices can hardly be used without individual contacts to each of them. This is why the fabrication of wires with nanometer-scale cross-section is another central problem of nanoelectronics. The currently available photolithography methods, and even their rationally envisioned extensions, will hardly be able to provide such resolution. Several alternative techniques, like the direct e-beam writing and scanning-probe manipulation can provide a nm-scale resolution, but their throughput is forbiddingly low for VLSI fabrication. Self-growing nanometer-scale-wide structures like carbon nanotubes or semiconductor nanowires can hardly be used to solve the wiring problem, mostly because these structures (in contrast with the nanodevices that have been discussed above) do not have means for reliable placement on the lower integrated circuit layers with the necessary

[3]Actually, this approach to interfaces is very much parallel to that accepted de facto in semiconductor electronics. Indeed, despite decades of research, properties of silicon-to-metal interfaces (in particular, the Fermi level pinning due to surface traps) are still neither completely understood nor fully predictable. This is why in most semiconductor circuit technologies, metal-semiconductor junctions are used only as passive Ohmic contacts, while active devices are built around much better explored $p-n$ junctions formed inside the semiconductor.

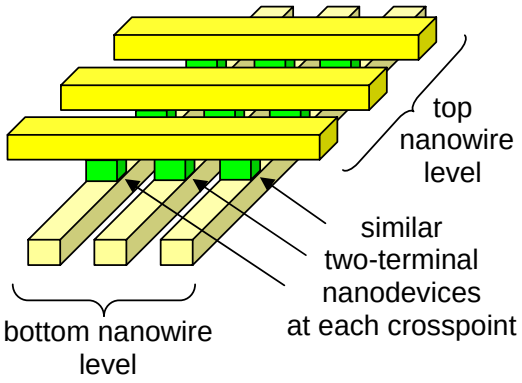

Figure 1.3: Crossbar array structure.

(a-few-nm) accuracy. Fortunately, there are several new patterning methods, notably nanoimprint [42, 120, 124] and interference lithography [15, 103], which may provide much higher resolution than the standard photolithography. Indeed, the layers of parallel nanowires with a nano half-pitch $F_{\rm nano} = 17$ nm have already been demonstrated [51], and there are good prospects for the half-pitch reduction to 3 nm or so in the next decade [42, 120, 124]. (The scaling of the pitch below 3 nm value would be not practical because of the quantum mechanical tunneling between nanowires.)

1.2.2 Circuits

The novel device and patterning technologies may allow to extend microelectronics into the few-nm range. However, they impose a number of challenges and limitation for integrated circuit design.

• **Defect Tolerance** - Perhaps, the main challenge faced by the hybrid circuits might be the requirement of very high defect tolerance. Indeed, it is natural to expect that at the initial stage of development of all nanodevices, their fabrication yield for $F_{\rm nano} < 30$ nm will be considerably below 100%, and, for $F_{\rm nano} \sim 3$ nm, will possibly never approach this limit closer than a few percent. This number can be compared with at most 10^{-8}% of bad transistors for the mature CMOS technology [1].

• **Circuit Regularity** - Nanoimprint and interference lithography cannot be used for the fabrication of arbitrary integrated circuits, in particular because they lack adequate layer alignment accuracy ("overlay"). This means that the nanowire layers should not require precise alignment with each other. The remedy to this problem can be a very regular "crossbar" nanowire structures [43] with two layers of similar wires perpendicular to those of the other layer (Fig. 1.3). On one hand, such structures are ideal for the integration of two-terminal nanodevices which can be sandwiched, e.g. by self-assembly or film deposition, in between two layers of nanowires. On the other hand, if all nanodevices are functionally similar to each

other, the relative position of one nanowire layer with respect to the other is not important. Not surprisingly, virtually all proposals for digital CMOS/nanodevice hybrids, most importantly including memories [8,20,29,110,112,113,127] and Boolean logic circuits [26,27,40,41,79,99–102,109,111,114,115,125,130], are based on crossbar structures (see also reviews of such circuits in Refs. 23, 28, 30, 58, 73, 105).[4]

• **Micro-to-Nano Interface** - The lack of alignment accuracy of novel patterning technologies also results in much harder problem of building CMOS-to-nanowire interfaces. In fact, the interface should enable the CMOS subsystem, with a relatively crude device pitch $2\beta F_{\rm CMOS}$ (where $\beta \sim 1$ is the ratio of the CMOS cell size to the wiring period and $F_{\rm CMOS}$ is a CMOS half-pitch), to address each wire separated from the next neighbors by a much smaller distance $F_{\rm nano}$.

Several solutions to this problem, which had been suggested earlier, seems to be not very efficient. In particular, almost all of the proposed interfaces are based on statistical formation of semiconductor-nanowire field-effect transistors gated by CMOS wires [31, 39, 56, 57] and can only provide a limited (address-decoding-type) connectivity, which might present a problem for sustaining sufficient data flow in and out of the nanoscale subsystem. Moreover, such demux-based interfaces presents architectural challenges since they are both needed for configuration of the nanodevices, as well as transferring data between CMOS and nano subsystems. Even more importantly, the technology of ordering chemically synthesized semiconductor nanowires into highly ordered parallel arrays has not been developed, and there is probably no any promising idea that may allow such assembly.

A more interesting approach was discussed in Ref. 130 (see also Refs. 105 and 17). It is based on a cut of the ends of nanowires of a parallel-wire array, along a line that forms a small angle $\alpha = \arctan(F_{\rm nano}/F_{\rm CMOS})$ with the wire direction. As a result of the cut, the ends of adjacent nanowires stick out by distances (along the wire direction) differing by $2F_{\rm CMOS}$, and may be contacted individually by the similarly cut CMOS wires. Unfortunately, the latter (CMOS) cut has to be precisely aligned with the former (nanowire) one, and it is not clear from Ref. 130 how exactly such a feat might be accomplished using available patterning techniques.

Finally, the idea of achieving CMOS-to-nano interface without any "overlay" alignment using precisely angled cuts, suggested recently [33], is very similar to the CMOL concept proposed earlier by K. Likharev [70, 71] and will be discussed in the next chapter.

[4] Another, not less exciting, application of the crossbar nanoelectronic hybrids, mixed-signal neuromorphic networks [63, 64, 72, 122], is out of the scope of this work.

1.2.3 Architectures

A) Memories

The most straightforward application of crossbar CMOS/nanodevice hybrids is in memory circuits - see, e.g., theoretical proposals [29, 74] and the first experimental demonstrations [20,127]. Note that such circuits can be thought of as an extension of more general "crossbar" or "resistive" memory species. In particular, it includes very promising crossbar memories with CMOS-scale wires [21], which have a potential to be the densest among memories based on the conventional photolithography-based technologies. This is why our discussion of these circuit (as well as the results in Chapter 3 of this thesis) is somewhat relevant for a much wider types of memories.

In crossbar memories, nanodevices are used as a single-bit memory cells, while the semiconductor transistor subsystem performs all the peripheral (input/output, coding/decoding, line driving, and sense amplification) functions that require relatively smaller number of devices (scaling as $N^{1/2}$, where N is the memory size in bits). If area overhead associated with periphery circuits is negligible then the footprint of the crossbar memories can be as small as $(2F_{\rm nano})^2$, which might result in the unprecedented density in excess of 1 Terabit/cm^2 at the end of the hybrid technology roadmap (for $F_{\rm nano} = 3$ nm), i.e. three orders of magnitude higher than that in existing semiconductor memory chips. [5]

The basic operation of crossbar memories can be explained using simplified equivalent circuits shown on Fig. 1.4. In the low-resistive state presenting binary 1, the nanodevice is essentially a diode, so that the application of voltage $V_{\rm t} < V_{\rm READ} < V_+$ to one (say, horizontal) nanowire leading to the memory cell gives a substantial current injection into the second wire (Fig. 1.4a). This current pulls up voltage $V_{\rm out}$ which can now be read out by a sense amplifier. The diode property to have low current at voltages above $-V_{\rm t}$ prevents parasitic currents which might be induced in other state-1 cells by the output voltage - see the red line in Fig. 1.4a. On the other hand, it is easy to show that memory arrays with purely linear (resistive) nanodevices do not scale well and hardly practical [104].

In state 0 (which presents binary zero) the crosspoint current is very small, giving a nominally negligible contribution to output signals at readout. In order to switch the cell into state 1, the two nanowires leading to the device are fed by voltages $\pm V_{\rm WRITE}$ (Fig. 1.4b), with $V_{\rm WRITE} < V_+ < 2V_{\rm WRITE}$. (The left inequality ensures that this operation does not disturb the state of "semi-selected" devices contacting just one of the biased nanowires.) The write 0 operation is performed similarly using the reciprocal switching with threshold V_- (Fig. 1.1). It is evident from Figs. 1.4a, b

[5] Here, we do not include in our comparison the data storage systems (such as hard disk drives, etc.) which cannot be used for bit-addressable memories because of their very large (millisecond-scale) access time.

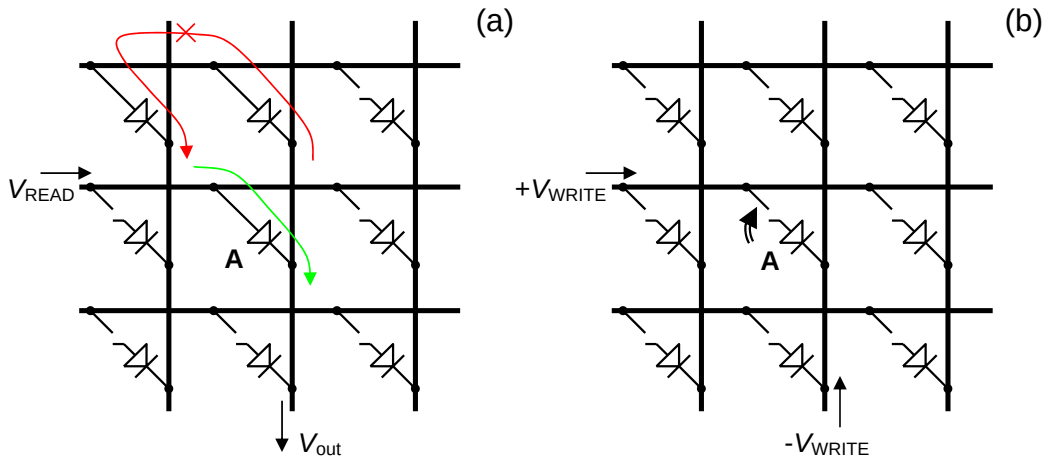

Figure 1.4: Equivalent circuits of the crossbar memory array showing (a) read and (b) write operations for one of the cells (marked A). On panel (a), green arrow shows the useful readout current, while red arrow shows the parasitic current to the wrong output wire, which is prevented by the nonlinearity of the $I-V$ curve of device A (if the output voltage is not too high, $V_{\rm out} < V_{\rm t}$).

that the read and write operations may be performed simultaneously with all cells of one row.[6]

The two main approach for fighting errors in semiconductor memory technology is reconfiguration, i.e. the replacement of memory array lines (rows or columns) containing bad cells by spare lines [18, 90]. The effectiveness of the replacement depends on how good its algorithm is [18, 46]. The Exhaustive Search approach (trying all possible combinations) finds the best repair solution, though it is not practicable because of the exponentially large execution time. A more acceptable choice is the "Repair Most" method that allows a simple hardware implementation and an execution time scaling linearly with the number of bits. In this approach, the number of defects in each line of a memory block (matrix) is counted, and the lines having the largest number of defects are replaced with a spare lines.

For a larger fraction of bad bits, better results may be achieved [29, 106, 116] by combining the bad line exclusion with ECC techniques. The simulation results for application of such technique for crossbar hybrid memories [29, 116] have shown that defect tolerance up to $\sim 10\%$ may be achieved using very powerful ECC, e.g. Reed-Solomon and Bose-Chaudhuri-Hocquenghem (BCH) codes [11]. Unfortunately,

[6] Actually, only one of the "write 0" and "write 1" operations can be performed simultaneously with all cells. Because of the opposite polarity of the necessary voltages across nanodevices for these two operations, the complete write may be implemented in two steps, e.g., first writing 0s and then writing 1s.

in those works, the contributions of the circuits implementing these codes to the memory access time (which for some codes may be extremely large) and the total memory area have not been estimated. Also the account of the finite leakage current through nominally closed crosspoints (which was neglected in Ref. 29) may change the memory scaling rather substantially [8].

Section 3 describes our own approach to terabit-scale defect tolerant CMOL-based nanoelectronic memories. In our first paper we have proposed the concept of CMOL memories based on continuous-nanowire structure [110], which would have quite low $\sim 50\%$ interface yield. Moreover, since only light ECC were employed in early version of CMOL memories, like Hamming codes [11], the defect tolerance even with a synergy of Exhaustive search reconfiguration was not very encouraging, e.g., 2% of defects allowed to achieve a 10-fold increase in density with respect to the defect-free pure semiconductor densest memories. This is why this thesis describes an advanced version of CMOL memories [112, 113]. First of all, the defect tolerance of improved version of CMOL memories is boosted by using a more aggressive codes. Secondly, their segmented structure allows for virtually 100% interface yield (for latter point, see, e.g. Section 2.1). Finally note, that the presented analysis of CMOL memories is more detailed than that of other groups, since it includes a possible tradeoff between density, defect tolerance, and speed performance accounting for all most important overheads from memory circuitry.

B) Logic Circuits

The practical techniques for high defect tolerance in digital (Boolean) logic is less obvious. In the usual custom logic circuits the location of a defective gate from outside is hardly possible, while spreading around additional logic gates (e.g., providing von Neumann's majority multiplexing [123]) for error detection and correction becomes very inefficient for fairly low fraction q of defective devices. For example, even the recently improved von Neumann's scheme requires a 10-fold redundancy for q as low as $\sim 10^{-5}$ and a 100-fold redundancy for $q \approx 3 \times 10^{-3}$ [93].

This is why the most significant previously published proposals for the implementation of logic circuits using CMOL-like hybrid structures had been based on reconfigurable regular structures like the field-programmable gate arrays (FPGA). Before this work, two FPGA varieties had been analyzed, one based on look-up tables (LUT) and another one using programmable-logic arrays (PLA).

In the former case, all possible values of an m-bit Boolean function of n binary operands are kept in m memory arrays, of size $2^n \times 1$ each. (For $m = 1$, and some representative applications the best resource utilization is achieved with n close to 4 [5], while the famous reconfigurable computer Teramac [43] is using LUT blocks with $n = 6$ and $m = 2$.) The main problem with this approach is that the memory arrays of the LUTs based on realistic molecular devices cannot provide address de-

coding and output signal sensing (recovery). This means that those functions should be implemented in the CMOS subsystem, and the corresponding overhead may be estimated using our results discussed in the previous section. Using the results for from Chapter 3 one can show that for the memory array with $2^6 \times 2$ bits, performing the function of a Teramac's LUT block, and for a realistic ratio $F_{\rm CMOS}/F_{\rm nano} = 10$ the area overhead would be above four orders of magnitude (!), and would even loose the density (and hence performance) competition to a purely-CMOS circuit performing the same function. On the other hand, increasing the memory array size to the optimum is not an option, because the LUT performance scales (approximately) only as a log of its capacity [5].

The PLA approach is based on the fact that an arbitrary Boolean function can be re-written in the canonical form, i.e. in the two-level logical representation. As a result, it may be implemented as a connection of two crossbar arrays, for example one performing the AND, and another the OR function [105]. The first problem with the application of this approach to the CMOS/nanodevice hybrids is the same as in the case of LUT's: the optimum size of the PLA crossbars is finite, and typically small [54], so that the CMOS overhead is extremely large. Moreover, any PLA logic built with diode-like nanodevices faces an additional problem of high power consumption. In contrast with LUT arrays, where it is possible to have current only through one nanodevice at a time, in PLA arrays the fraction of open devices is of the order of one half [23]. Let us estimate the static power dissipated by such an array. The specific capacitance of a wire in an integrated circuit is always of the order of 2×10^{-10} F/m (for justification, see, e.g., Fig. 2.6.). With $F_{\rm nano} = 3$ nm, this number shows that in order to make the RC time constant of the nanowire below than, or of the order of the logic delay in modern CMOS circuits ($\sim10^{-10}$s), the ON resistance $R_{\rm ON}$ of a molecular device has to be below $\sim 7 \times 10^7$ ohms. For reliable operation of single-electron transistor (and apparently any other active electronic nanodevice) at temperature T, the scale $V_{\rm ON}$ of voltage $V = V_{\rm s} - V_{\rm d}$ across it has to be at least $10k_BT$ [70]. For room temperature this gives $V_{\rm ON} > 0.25$ Volt, so that static power dissipation per one open device, $P_{\rm ON} = V_{\rm ON}^2/R_{\rm ON}$ is close to 10 nW. With the open device density of $0.5/(2F_{\rm nano})^2 \approx 10^{12}$ cm^{-2}, this creates a power dissipation density of at least 10 kW/cm^2, much higher than the current and prospective technologies allow to manage [1].

As a matter of principle, power consumption may be reduced by using dynamic logic, but this approach requires more complex nanodevices. For example, Refs. 26–28 describe a dynamic-mode PLA-like structure (with improved functional density via wrapped logic mapping) using several types of molecular-scale devices, most importantly including field-effect transistors which are formed at crosspoints of two nanowires. In such transistor, one (semiconductor) nanowire would serve as a drain/channel/source structure, while the perpendicular nanowire would play the role of

the gate. Unfortunately, such circuits would fail because of the same fundamental physical reason that provides the fundamental limitation to the Moore's Law: any semiconductor MOSFET with a-few-nm-long channel is irreproducible because of exponential dependence of the threshold voltage on the transistor dimensions [118].[7] Similar problems are likely to prevent hybrid circuits described in Refs. 101, 125 from scaling down beyond 10-nm range, since they are based on nanoscale FETs.

Finally, the last significant category of suggested crossbar hybrids includes circuits based on Goto-pair logic [105]. In particular, Refs. 99, 100 describe an architecture where Goto-pair logic is implemented with two-terminal resistive crossbar latches [59]. The main architectural challenge of this approach is due to the fact that nanodevice bistability is employed during Goto pair operation. [8] Since the assumed nanodevices have no third state, and hence, cannot be enabled or disabled, it is unclear how to mapping a particular circuit on such architectures. (Having a third state is probably not very practical since multi-state devices are not very reliable.)

Moreover, the use of bistability in the circuit operation is rather impractical due to the relation between the retention time and the switching speed in the crossbar latches. In order to be useful for most electronics applications, the latches should be switched very fast (in a few picoseconds in order to compete with advanced MOSFETs), but retain their internal state for the time necessary to complete the calculation (ideally, for a few years, though several hours may be acceptable in some cases). This means that the change of the applied voltage by the factor of two (the difference between the fully selected and semi-selected crosspoints of a crossbar) should change the switching rate by at least 16 orders of magnitude. However, even the most favorable physical process we are aware of (the quantum-mechanical tunneling through high-quality dielectric layers like the thermally-grown SiO_2) may only produce, at these conditions, the rate changes below 10 orders of magnitude, even if uncomfortably high voltages of the order of 12 V are used [14].

In Section 4 we present an alternative approach to Boolean logic circuits based on CMOL concept [109, 111, 114], that is closed to the so-called cell-based FPGA [91]. We will show that such circuits can have much better area-delay performance as compared to purely CMOS FGPAs, and at the same time provide very high defect tolerance.

[7] In principle, this problem can be alleviated by making the width of nanowires in one dimension comparable with that of lithographically defined wires [27]. However, that also means that such hybrid circuits cannot take full advantage (only in one dimension) of nanodevice nanometer-scale footprints.

[8] As a reminder, in all discussed crossbar circuits above, as well as in our approach for Boolean logic (Chapter 4), the state of nanodevices remains unchanged during circuit operation.

Chapter 2

CMOL Approach and Hardware Models

2.1 Concept

Figure 2.1 shows the so-called CMOL approach [70, 71, 73] to the interface problem. The difference between this approach (based on earlier work on the so-called "InBar" neuromorphic networks [72, 122]), and the suggestions discussed in Section 1.2.2 is that in CMOL the CMOS-to-nanowire interface is provided by pins distributed all over the circuit area. In the generic CMOL circuit (Fig. 2.1), pins of each type (contacting the bottom and top nanowire levels) are located on a square lattice of period $2\beta F_{\rm CMOS}$. Relative to these arrays, the nanowire crossbar is turned by a (typically, small) angle α which is found as (Fig. 2.1c):

$$\alpha = \arctan\frac{1}{a} = \arcsin\frac{F_{\rm nano}}{\beta F_{\rm CMOS}} \ll 1, \tag{2.1}$$

where a is a (typically, large) integer. Such tilt ensures that a shift by one nanowire (e.g., from the second wire from the left to the third one in Fig. 2.1c) corresponds to the shift from one interface pin to the next one (in the next row of similar pins), while a shift by a nanowires leads to the next pin in the same row. This trick enables individual addressing of each nanowire even at $F_{\rm nano} \ll \beta F_{\rm CMOS}$. For example, the selection of CMOS cells 1 and 2 (Fig. 2.1c) enables contacts to the nanowires leading to the left one of the two nanodevices shown on that panel. Now, if we keep selecting cell 1, and instead of cell 2 select cell 2' (using the next CMOS wiring row), we contact the nanowires going to the right nanodevice instead.

It is also clear that a shift of the nanowire/nanodevice subsystem by one nanowiring pitch with respect to the CMOS base does not affect the circuit properties. Moreover, a straightforward analysis of CMOL interface (Fig. 2.2) shows that at an optimal shape of the interface pins (for example, when top radius of both upper and lower level interface pins, the nanowire width and nanowire spacing are all equal), even a complete lack of alignment of these two subsystems leads to a theoretical

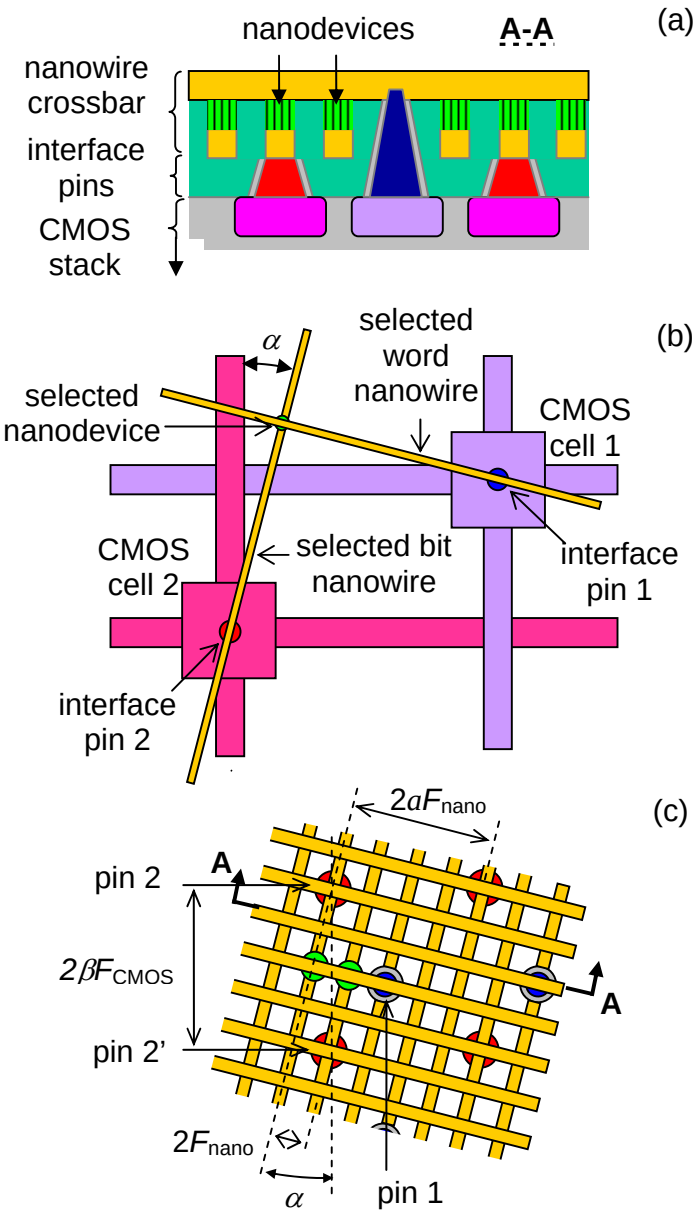

Figure 2.1: The generic CMOL circuit: (a) a schematic side view; (b) a schematic top view showing the idea of addressing a particular nanodevice via a pair of CMOS cells and interface pins, and (c) a zoom-in top view on the circuit near several adjacent interface pins. On panel (b), only the activated CMOS lines and nanowires are shown, while panel (c) shows only two devices. (In reality, similar nanodevices are formed at all nanowire crosspoints.) Also disguised on panel (c) are CMOS cells and wiring.

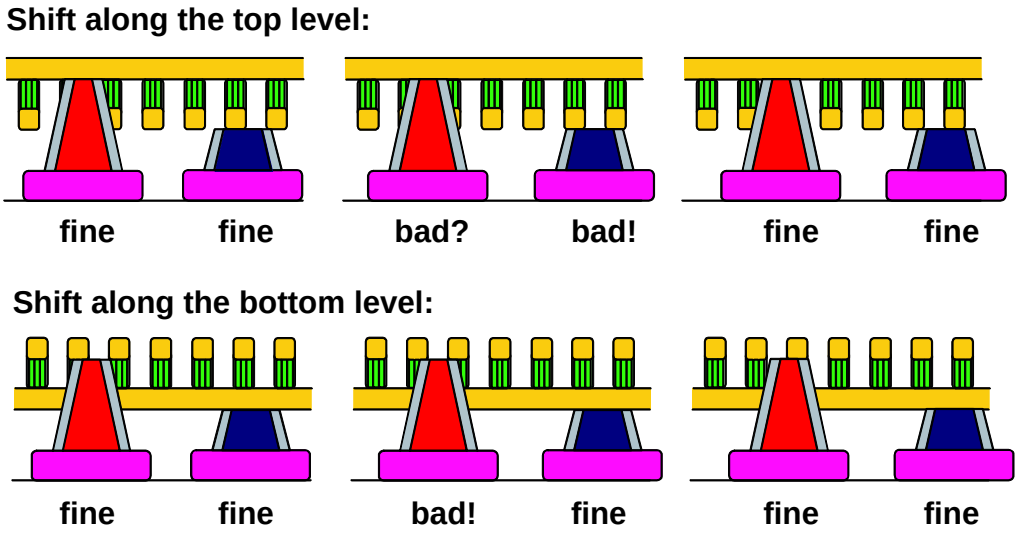

Figure 2.2: The idea of 100% CMOS-to-nano interface yield without any overlay alignment.

interface yield of 100%. (Note that the last statement is only true for the latest version of CMOL [112, 114] in which pin, going to the upper nanowire level, intentionally interrupts a lower layer wire - see Fig. 2.1.) Even if the interface yield will be less than 100%, it may be acceptable, taking into account that the cost of the nanosystem fabrication, including the chemically-directed assembly of molecular devices may be rather low, especially in the context of an unparalleled density of active devices in CMOL circuits.

Figure 2.3 shows plausible fabrication steps required to create described CMOS-to-nano interface and nanowire crossbar with intermediate nanodevice layer.

2.2 CMOL Cousins

More recently, at least two approaches to the interface between CMOS and nano subsystems, very similar to CMOL, have been proposed . In Ref. 33, interface between nano and CMOS wires is supposed to be formed by exposing portions of CMOS wires with precisely angled cut in the insulator layer (Fig. 2.4). The key point in this proposal is that the interface yield can be up to 100% without any overlay alignment between nano and CMOS layers if the vertical gap $w_{\rm gap}$ between CMOS openings and its height are exactly equal to nanowire width $w_{\rm nano}$ and nanowire spacing $s_{\rm space}$, correspondingly. Clearly, the idea behind it is the same as that discussed in previous section, if one replace CMOS area openings with CMOS pillars.

The advantage over CMOL approach is that the cut is much easier to imple-

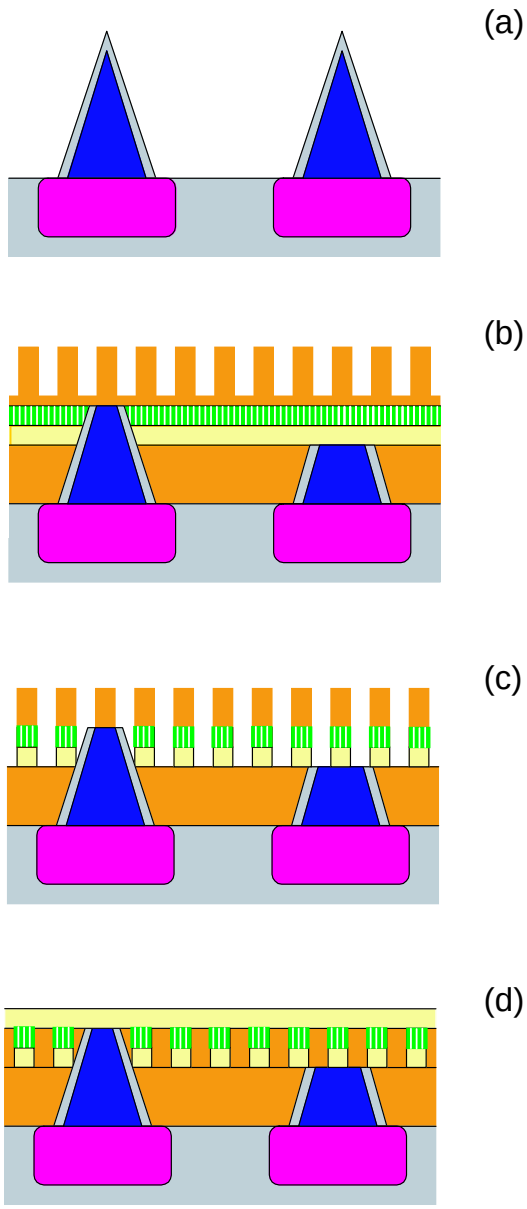

Figure 2.3: A plausible CMOL fabrication process flow (crudely): (a) Formation of pins on top of CMOS layer (e.g, using technique used for fabrication of the tips in field-emission arrays - see Ref. 49) and covering them with the insulator (oxidation), (b) deposition of a base resist layer, etching lower layer pins, deposition of metal and device layers (SAM or any film of a material with a properties discussed in Sec. 1.1), and transferring nanowire pattern on the upper resist layer, (c) etching device and metal layers to form low level nanowires, and (d) deposition of supporting resist, patterning upper level nanowires.

ment than the pins. On the other hand, this approach has also rather substantial disadvantages:

- The interface density is more than twice lower that the maximum possible one $1/(2F_{\rm CMOS} \times 2F_{\rm nano})$. More specifically, if $w_{\rm CMOS}$, $s_{\rm CMOS}$, and $w_{\rm cut}$, are CMOS wire width, spacing and the width of the cut, respectively, the following equations should be satisfied [33]:

$$w_{\rm nano} = w_{\rm gap} = s_{\rm CMOS}/\tan\alpha - w_{\rm cut}/\sin\alpha, \tag{2.2}$$

$$s_{\rm nano} = w_{\rm CMOS}/\tan\alpha + w_{\rm cut}/\tan\alpha, \tag{2.3}$$

  where angle α of the cut can be found similarly to Eq. (2.1) as

$$\tan\alpha = \frac{w_{\rm nano} + s_{\rm nano}}{w_{\rm CMOS} + s_{\rm CMOS}}. \tag{2.4}$$

  Assuming that, as in our previous section, the minimum nano- and micro-scale feature sizes are $F_{\rm nano}$ and $F_{\rm CMOS}$, correspondingly, it is clear from Eqs. (2.2, 2.3, 2.4) that picking minimum spacing and width for both nanowires and CMOS wires is not possible. The largest density of the interface can be found assuming that $w_{\rm nano} = w_{\rm cut} = F_{\rm nano}$, $w_{\rm CMOS} = F_{\rm CMOS}$, and optimizing $s_{\rm CMOS}$ and $s_{\rm nano}$. For the most interesting case $F_{\rm CMOS}/F_{\rm nano} \gg 1$, and hence solving Eqs. (2.3) and (2.4) together gives

$$s_{\rm CMOS} \gtrsim 2F_{\rm nano}F_{\rm CMOS}/(s_{\rm nano} - F_{\rm nano}). \tag{2.5}$$

  Therefore, the maximum density of the interface is roughly equal to $1/(2F_{\rm CMOS} \times 4F_{\rm nano})$, i.e. about 50% of CMOL density, since optimal values are $s_{\rm CMOS} = F_{\rm CMOS}$, $s_{\rm nano} = 3F_{\rm nano}$.

- The proposed interface is peripheral since the suggested technique only feasible for interfacing one layer of nanowires at a time. Hence, it may be used on the crossbar periphery rather than distributed over all the area as CMOL. As a result, the implmentation of logic circuits in this technology is hardly feasible (cf. Chapter 4).

The area interface without nanometer-scale pins is suggested recently in HP's FPNI circuits [102]. According to the authors, such FPNI circuits is a generalization of the CMOL FPGA approach, allowing for simpler fabrication and more conservative process parameters. More specifically, authors indicate that the sharply-pointed interface pins with nanometer-scale top radii present a fabrication challenge and at the initial stage it is easier to replace them with CMOS-scale pins. For such change

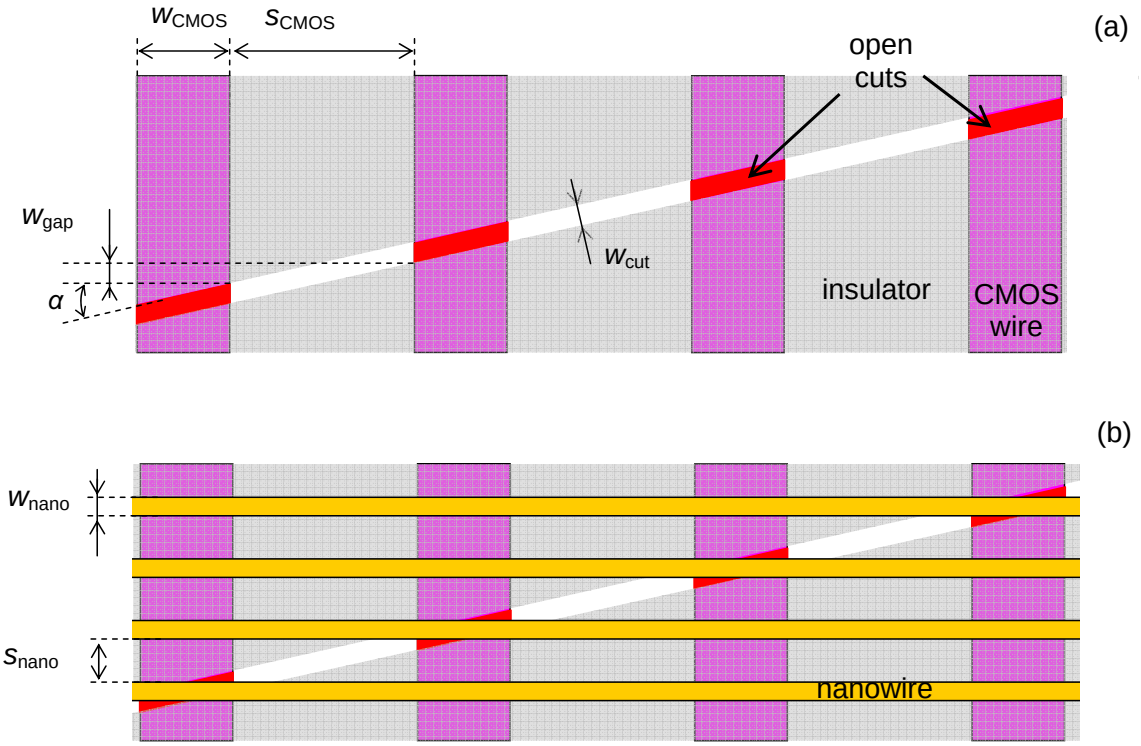

Figure 2.4: The idea of 100% CMOS-to-nano interface yield without any overlay alignment [33].

the nanowire crossbar requires CMOS-scale alignment with respect to CMOS subsystem and will be much sparser than the original used in CMOL (Fig. 2.5). Another feature that simplifies fabrication of FPNI is the fact that nanodevices are used only as programmable resistance switches. The downside of FPNI approach is that more functionality is transferred in CMOS subsystem and together with sparser nanowire crossbar, as it will be shown later in Chapter 5, the areal density of such circuits is substantially lower than that of CMOL-based ones.

This is why, these recently suggested CMOL varieties might be seen as just an intermediate steps towards real CMOL circuits which are superior in characteristics, though more challenging for fabrication.

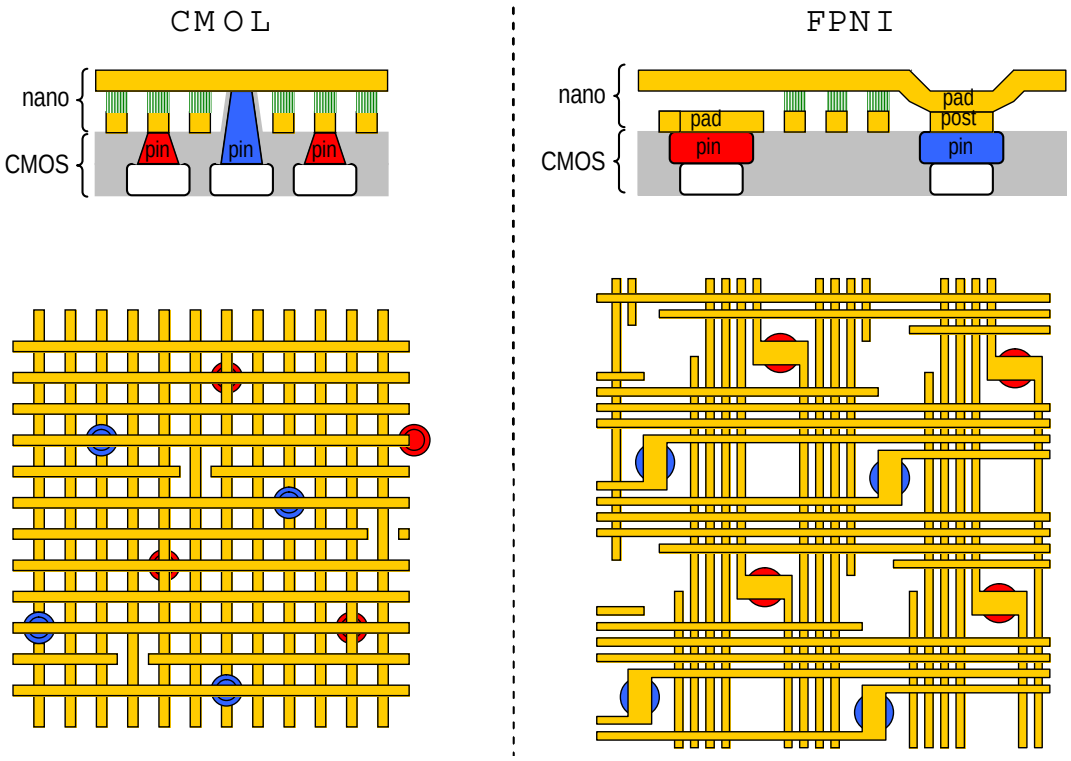

Figure 2.5: Comparison of CMOL and HP's FPNI circuits (adapted from Ref. 102).

2.3 Performance Model

2.3.1 Nanodevices

We have assumed that each crosspoint programmable diode is implemented as a parallel connection of D single-electron latching switches [35,72], so that the resistance of the single crosspoint nanodevice is $R_{\rm ON}/D$ and $R_{\rm OFF}/D$ in the ON and OFF states, correspondingly. (Estimates for the devices based on electron trapping in random localized states are in the same ballpark, because the basic physics of their operation [98] is very close to that of single-electron devices.) Based on the experimental data for self-assembled monolayers (see, e.g., Ref. 128), the footprint of a single molecule may be estimated as 0.25 nm^2; so for D we have used the following value:

$$D \approx \frac{(F_{\rm nano})^2}{0.25\ {\rm nm}^2}. \tag{2.6}$$

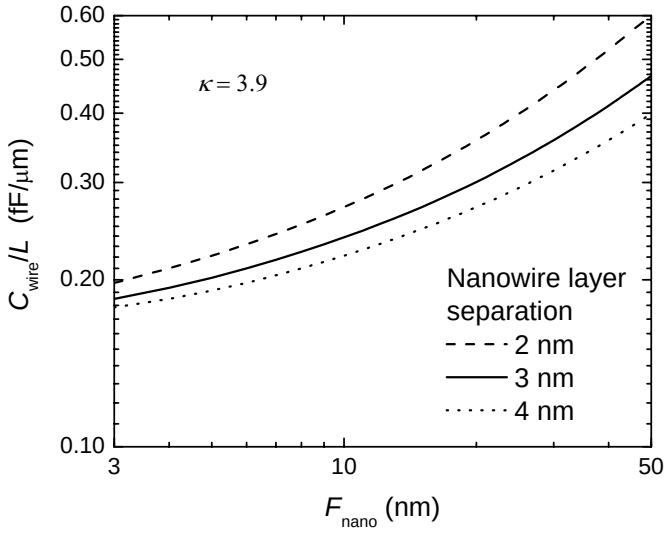

Figure 2.6: Specific capacitance of a nanowire with $F_{\rm nano} \times F_{\rm nano}$ cross-section, in a crossbar with several values of interlayer spacing (for dielectric constant $\kappa = 3.9$).

Resistances $R_{\rm ON}$ and $R_{\rm OFF}$ are related by the equation for the second-order quantum effect, elastic co-tunneling [48]:

$$R_{\rm OFF}/R_{\rm ON} = R_{\rm ON}/R_{\rm Q}, \tag{2.7}$$

where $R_{\rm Q} \equiv \hbar/e^2 \approx 4.1$ kΩ is the quantum unit of resistance.

The thermally-induced suppression of $R_{\rm OFF}$ resulting from the classical thermal activation [38], i.e.

$$\frac{R_{\rm OFF}}{R_{\rm ON}} = \cosh^2 \frac{V_{\rm READ}}{2k_{\rm B}T}, \tag{2.8}$$

where $k_{\rm B}$ is a Boltzman constant and T is a (room) temperature, is typically negligible.

2.3.2 Nanowires

Specific capacitance $C_{\rm wire}/L$ of nanowires has been calculated using the well-known FASTCAP code [83] for the crossbar structure (Fig. 2.1a, c) in which both the width and the thickness of the nanowire, as well as the horizontal distance between the wires, were assumed to be all equal to $F_{\rm nano}$, while the vertical distance between two layers was varied from 2 to 4 nm. The insulator between and around the wires is assumed to have a dielectric constant of 3.9 (corresponding to SiO_2); the use of a low-κ dielectric would give the corresponding increase of the circuit operation speed cited below. The result of the calculation is shown in Fig. 2.6. In particular, Fig. 2.6 shows that the capacitance of nanowires per unit length for the most interesting

range of $F_{\rm nano}$ is about 0.2 fF/μm.

In order to calculate the specific resistance $R_{\rm wire}/L$ of a metallic nanowire with the assumed square-shaped cross-section $F_{\rm nano} \times F_{\rm nano}$, the usual formula $\rho/(F_{\rm nano})^2$ has to be generalized to include the increase of resistivity ρ due to possible diffusive surface scattering of electrons. (This effect becomes substantial when $F_{\rm nano}$ is decreased below the electron mean free path λ due to scattering on phonons.) A reasonable approximation for ρ is given by the Matthiessen rule [52] in the form

$$\rho \approx \rho_0 \times (1 + \lambda/F_{\rm nano}), \tag{2.9}$$

where ρ_0 is the table (bulk) resistivity of a pure metal. In our calculations we have assumed values $\rho_0 = 2$ $\mu\Omega$-cm and $\lambda = 10$ nm which are typical for good metals at room temperature.

2.4 Defect Model

It is believed that the most numerous and hence the most significant types of "hard" (fabrication-induced) faults will be

- **"Stuck-on-open" defects in nanodevices -** Such defects corresponds to permanently disconnected crosspoints. We will assume that stuck-on-open defects are uniformly distributed with probability q. (Note, that any clustering of defects would be much easier to cope with, e.g., via reconfiguration.)

This assumption is justified by recent experimental works (see, e.g., Ref. 20). It is important, therefore, for an architecture to provide first of all the defect tolerance with respect to these kind faults. This is why only these kind of defects were taken into account in most of the hybrid circuit papers [32, 101].

While stuck-on-open defects will be considered both in CMOL memories (Chapter 3) and CMOL Boolean logic circuits (Chapter 4), we will also simulate the defect tolerance with respect to other types of faults for the latter case. The other, less important types of defects, might include:

- **Defective nano-to-CMOS interface pins -** The technology of formation of sharply pointed pins (e.g., etching down from a larger cylindrical structure or growing on the substrate) is stochastic in nature. This means that the locations of the end points of the pins may deviate from precisely defined ones. Alternatively, a defective interface can be resulted from a too small overlapping area between the surface of a pin and nanowire.

- **Broken or shortened nanowires -** For example these might be due to some imperfections in nanowire mold.

- **Defective CMOS circuitry -** Such faults may be simply resulted from defects in CMOS circuitry, in particular, due to narrow voltage threshold margins of MOSFETs.
- **"Stuck-on-close" defects in nanodevices -** This type of defects corresponds to permanently shorted nanowire crosspoints.

Chapter 3

CMOL Memory

3.1 Architecture and Operation

Figure 3.1a shows the assumed general structure of the CMOL memory. Essentially, it is similar to that of the conventional memories, i.e. it is a rectangular array of L crosspoint memory banks ("blocks"), so that during a single operation, a particular row of CMOL blocks is accessed with the help of block address decoders. In contrast, the block architecture (Fig. 3.1b) is specific for the CMOL interface which allows the placement of CMOS "relay" cells under the nanowire crossbar. These cells are controlled by CMOS-level decoders, four per each block (Fig. 3.1b). At each elementary operation, one pair of block decoders (shown in magenta in Fig. 3.1b, as well as in Figs. 3.2 and 3.3 below) addresses one vertical and one horizontal CMOS line, and thus selects a certain relay cell at their crosspoint. This cell (Fig. 3.3) applies the data signal to a "red" interface pin contacting a bottom-layer nanowire. The other pair of decoders (shown in violet in Figs. 3.1b, 3.2 and 3.3) selects a set of different relay cells which provide similar biasing of the corresponding top level nanowires through "blue" pins. These nanowires may now address all crosspoint nanodevices (memory cells) of a particular nanowire segment. Thus the four decoders of the block, working together, can provide every memory cell of the segment with voltages necessary for the read and write operations.

The remaining circuitry shown in Fig. 3.1b, i.e. CMOS-based mapping table and address control circuits, is needed to convert the logical (external) addresses, which are fed to the CMOL blocks, into internal addresses of memory cells inside the block. In particular, the mapping table converts the logical address of the segment (which is the same for all selected blocks) into a pair of block-specific physical addresses, A_{col1} and A_{row1}, and CMOS-implemented decoders activate the corresponding CMOS-level lines.

Figure 3.2 shows the low-level structure of the CMOL memory for a particular (unrealistically small) values of the block size and the main topological parameter of

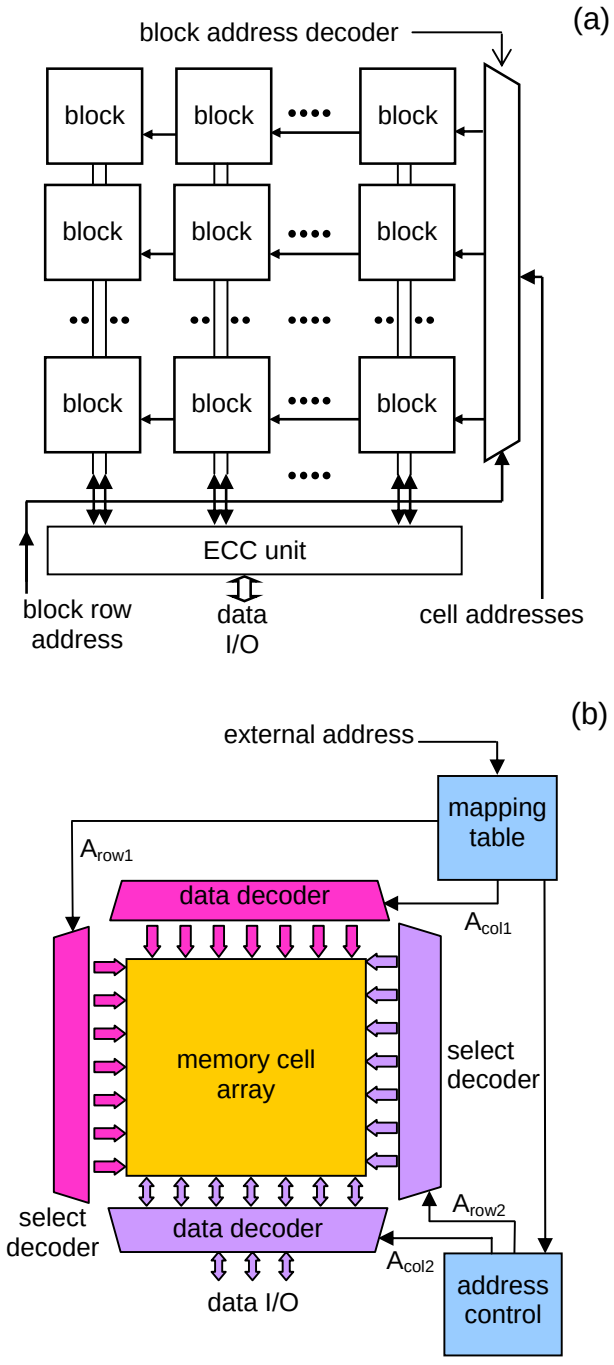

Figure 3.1: CMOL memory structure considered in this work: (a) global and (b) block architectures.

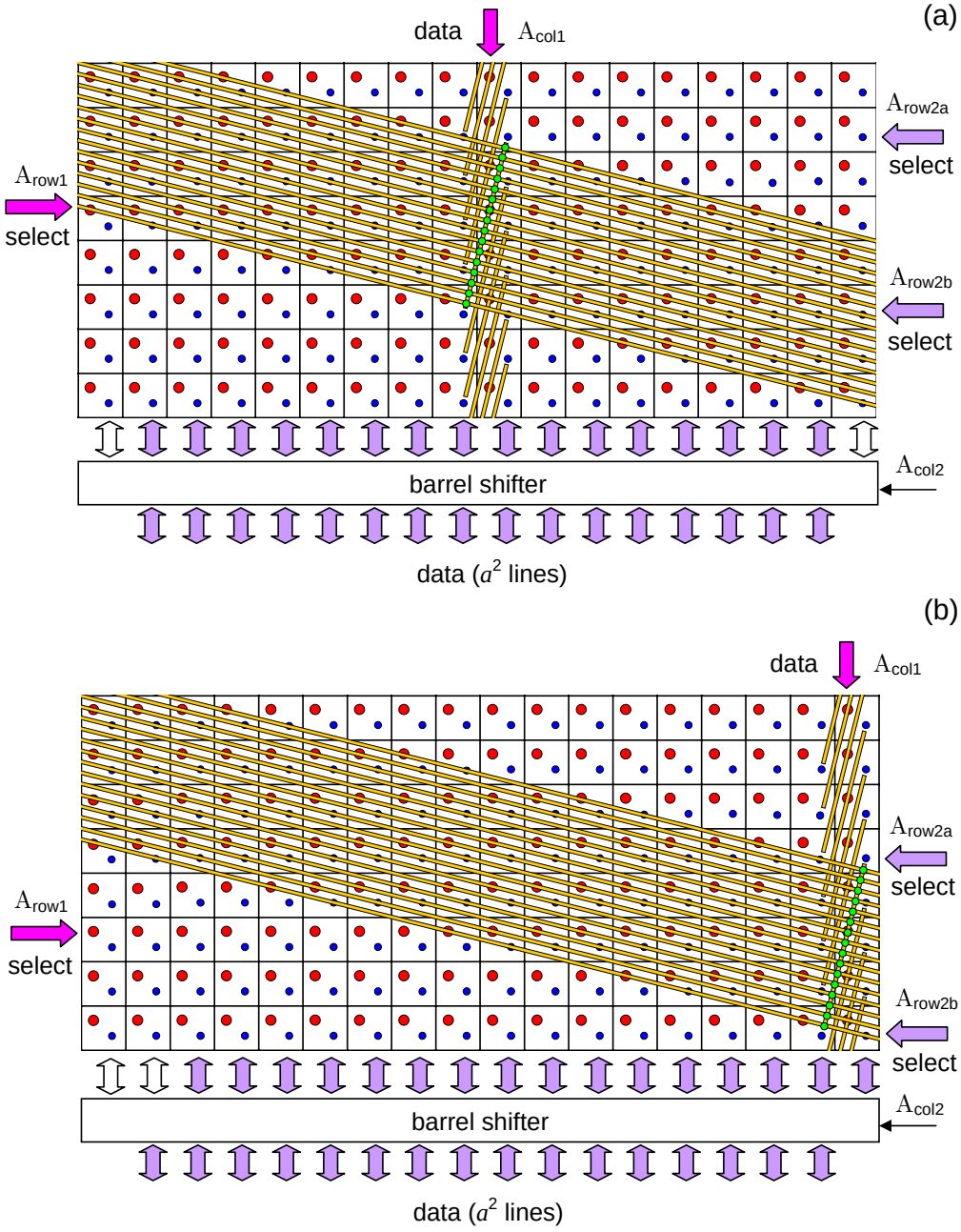

Figure 3.2: CMOL block architecture: Addressing of an interior column of nanowire segments (for $a = 4$). The figure shows only one (selected) column of the segments, the crosspoint nanodevices connected to one (selected) segment, and continuous top-level nanowires connected to these nanodevices. (In reality, the nanowires of both layers fill all the array plane, with nanodevices at each crosspoint.) The block arrows indicate the location of CMOS lines activated at addressing the shown nanodevices.

CMOL, $a = 4$. The top-level nanowires (here shown quasi-horizontal) stretch over the whole block, but the low-level (nearly-vertical) nanowires are naturally cut into segments of equal length. An elementary analysis of the CMOL geometry (Fig. 2.1) shows that each nanowire segment stretches over a CMOS cells and contacts a^2 (in Fig. 3.2, sixteen) crosspoint nanodevices.

Signals A_{col1} and A_{row1} are applied to CMOS wires, feeding the "red" lines of the corresponding CMOS-implemented relay cells (Fig. 3.3). By opening all pass transistors of the row, A_{row1} selects a specific "red" pin of column A_{col1}, so that the data A_{col1} are fed only to a specific nanowire segment contacting a^2 crosspoint nanodevices. In parallel, addresses A_{col1} and A_{row1} are sent to the CMOS-based address control circuitry to generate another pair of physical addresses A_{row2} and A_{col2}. Signal A_{row2} opens the "blue"-pin pass transistors in relay cells of a row, and thus connects each of a^2 quasi-horizontal nanowires of the top layer to a specific CMOS lines (shown purple), thus enabling a read or write operation.

For large CMOL arrays (say, a square array with W relay cells on a side, with $W \gg a^2$), most nanowire segments are well inside the array. In order to address such segments, A_{row2} might just reproduce A_{row1}. The problem with such scheme is that it would allow addressing only $W(W - a^2)$ internal nanowire segments, of the total number W^2. In order to decrease the associated loss of Wa^4 memory cells (of the total W^2a^2 relay cells in the block), we prefer to address simultaneously two lines:

$$A_{row2a} = A_{row1} + a/2, \tag{3.1}$$

$$A_{row2b} = A_{row1} - a/2. \tag{3.2}$$

Such addressing scheme (Fig. 3.2) reduces the loss to just Wa^3 memory cells per block. (For the case shown in Fig. 3.2, the lost cells are located in two rectangular $W \times a/2$ areas on the top and at the bottom of the array.) The associated area penalty is negligible in most cases (see the analysis below).

Note that of W output CMOS lines (purple arrows on the bottom of each panel of Fig. 3.2), at each operation only a^2 lines are connected to nanodevices of the selected fragment. The selection of these useful lines (or a part of them, see Sec. 3.2 below) and their connection to the system output are provided by the data decoder controlled by signal A_{col2} (Fig. 3.1b). The necessary circuit is rather simple (essentially, a barrel shifter) and may be readily implemented in the CMOS subsystem.

3.2 Defect Tolerance Calculation

We have calculated the tolerance of our memories to "hard" (fabrication-induced) defects; however, in future, it is certainly desirable to extend the analysis to other defects, see, e.g., Section 2.4. Here, in Chapter 5 we only briefly discuss the possible

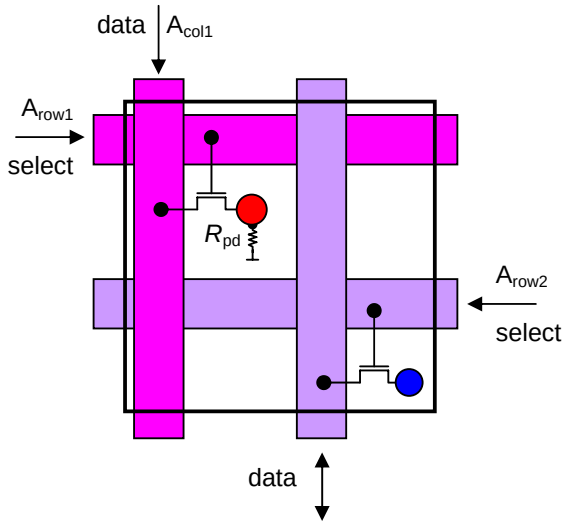

Figure 3.3: Possible structure of CMOS relay cell. Red and blue points indicate the corresponding interface pins.

effect of such defects on the performance of the memory.

In the synergetic approach of combining the memory array reconfiguration with ECC [106], memory cells are divided into fragments of certain size ("granularity"). Each of these fragments is tested using ECC circuitry, and those of them which may not be ECC-corrected are excluded from operation. (For that, the addresses of good fragments are written into the mapping table, see Fig. 3.1b). If the fraction q of bad bits is large, the large granularity of exclusion is impracticable, due to the exponential growth of the number of necessary redundant resources. (In particular, this was one of the reasons of the relatively poor defect tolerance achieved in our previous work [110], where only global nanowires might be excluded.) On the other hand, fine granularity requires an unacceptably large mapping table. In this work we use a new, more flexible approach when the granularity of exclusion is not related to the physical structure of the memory array. This means that the data fragment length, equal to g nanowire segments (i. e. ga^2 memory cells) may be either smaller or larger than the one segment (which has a^2 memory cells).

In the former case ($g < 1$), the fragment is physically placed on a part of one nanowire segment. This can be easily implemented by keeping an additional "displacement" address inside the mapping table, and an additional circuitry which multiplies the displacement by ga^2 and adds the result to A_{col2}. In the latter case ($g > 1$), the fragment is physically placed (at the same intrablock address) into g nanowire segments of g adjacent blocks of the same block row -see Fig. 3.1a. (Their location inside the same block would result in a larger block size, and hence in larger global nanowire

capacitance and delay time - see Sec. 3.4.) This requires the address mapping table to be shared between g neighboring blocks.

For a binary ECC with length n and the information length k, which can fix up to t errors in any of the n bits, the probability $P_{\rm f}$ to fix a fragment is

$$P_{\rm f} = \left[\sum_{i=0}^{t} \binom{n}{i} q^i (1-q)^{n-i} \right]^{ga^2/n} . \tag{3.3}$$

Here n is assumed to divide ga^2 exactly. (If this is not so, the bits in the reminder cannot be used and are wasted.) If each block has b spare and m useful fragments (with the sum $m + b = W^2$), the probability $P_{\rm sb}$ for a "superblock" (consisting of g adjacent blocks which share data fragments) to be fully functional can be calculated as

$$P_{\rm sb} = \sum_{i=0}^{b} \binom{m+b}{i} (1-P_{\rm f})^i P_{\rm f}^{m+b-i}, \tag{3.4}$$

and the yield Y of the total memory is

$$Y = P_{\rm sb}^{L/g}. \tag{3.5}$$

In order to characterize the defect tolerance, we fix Y at a certain level (usually, 90%), and numerically optimize the granularity (fragment length g) and ECC parameters n and k to calculate the maximum manageable fraction q of bad memory cells. Since for realistic parameters (in particular, $W^2 \gg 1$), $P_{\rm f}$ drops from nearly one to zero extremely fast at a certain value $q \ll 1$, the block size (and hence the number L of blocks at a fixed total memory size) and Y virtually do not affect these results.

3.3 Area Calculation

3.3.1 Total Area and Capacity

In Ref. 110 we have showed that for realistic block dimensions the area of block decoders is negligible, therefore the total memory area may be calculated as

$$\begin{aligned} A = L \times A_{\rm block} = L \times (&A_{\rm array} + A_{\rm cell\ decoder} + A_{\rm drive/sense} \\ &+ A_{\rm control}/g + A_{\rm mapping\ table}/g), \end{aligned} \tag{3.6}$$

while useful capacity of the memory is

$$N = \frac{L}{g} mk \left\lfloor \frac{ga^2}{n} \right\rfloor , \tag{3.7}$$

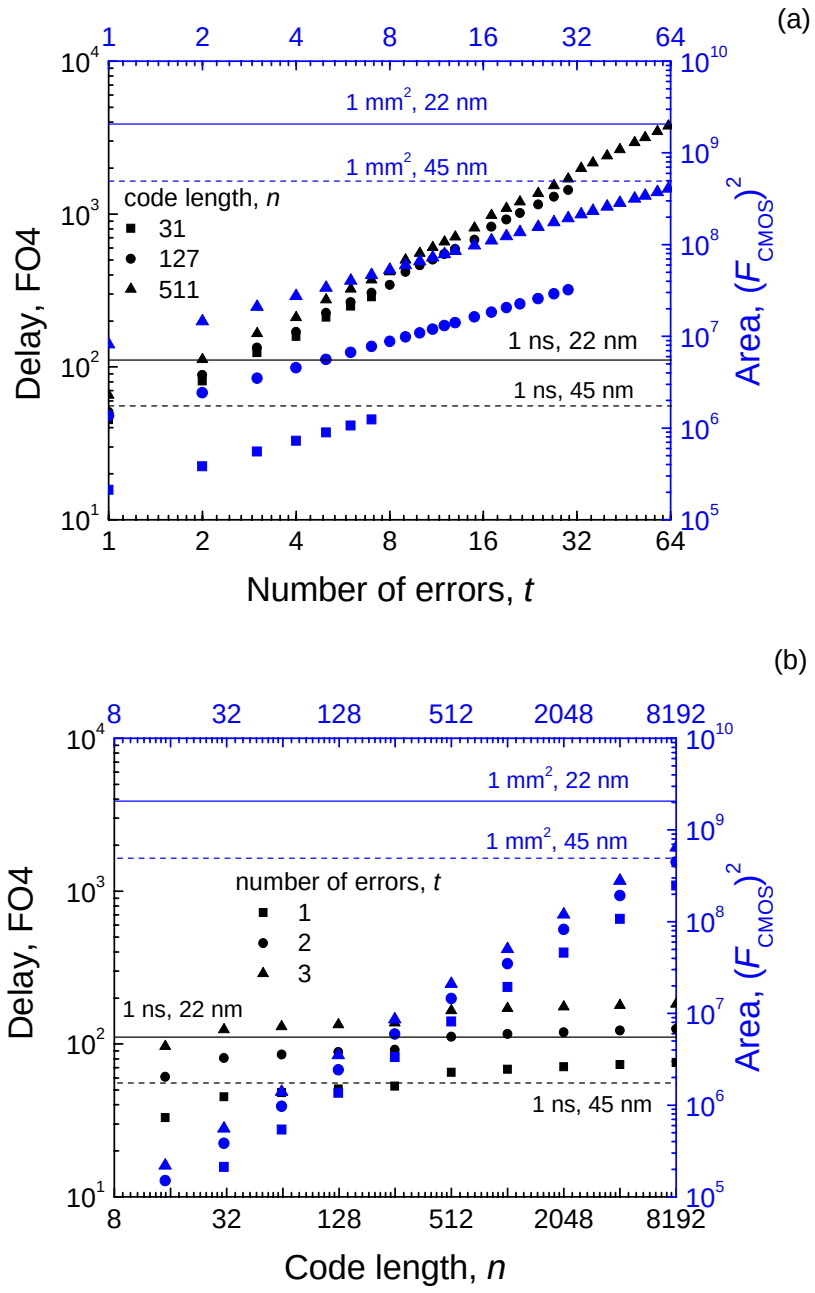

Figure 3.4: Delay (black lines) and area (blue lines) of a bit-parallel decoder for binary BCH codes as functions of the number of errors t the code can correct for several values of code length n. The delay is measured in delays of a standard CMOS fan-out-of-four inverter, while the area in the CMOS half-pitch units. For convenience, horizontal lines show the delay of 1 ns and the area of 1 mm^2, for the 22 nm and 45 nm ITRS technology nodes (solid and dashed lines, respectively).

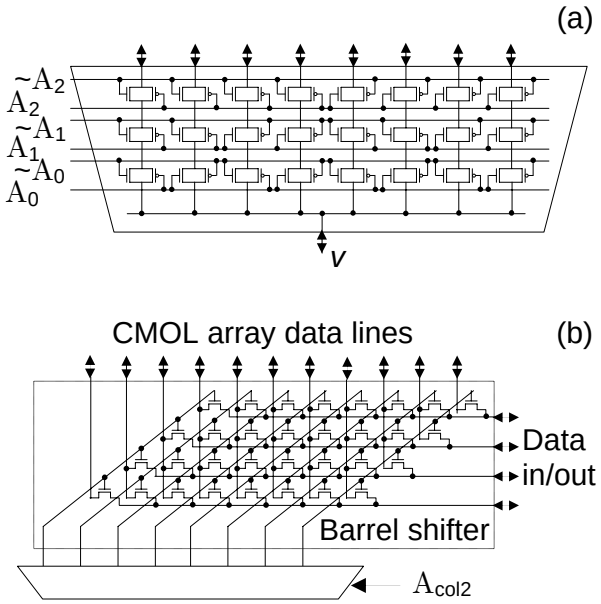

Figure 3.5: The assumed structure of peripheral circuits: (a) cell decoder, and (b) barrel shift decoder.

where m is the number of good data fragments per block, L/g is the number of "superblocks", and ga^2 is the number of memory cells in one fragment - see Sec. 3.2 above. The ratio $k/n < 1$ reflects the area loss due to the ECC. The memory density may be characterized by the total area per one useful cell, i.e. the ratio A/N, which may be conveniently expressed in the units of $(F_{\rm CMOS})^2$.

3.3.2 Crossbar Arrays

The area of a CMOL array of $W \times W$ relay cells is simply

$$A_{\rm array} = W^2 \times (2\beta F_{\rm CMOS})^2. \tag{3.8}$$

For the CMOS relay cell area, we have used an estimate of $(5F_{\rm CMOS})^2$, resulting in $\beta \geq 1.6$. We believe this is a fair estimate for the cell shown in Figure 3.3 implemented in a style similar to the usual NAND flash memory cell [14]. If this estimate seems too optimistic, note that it does not affect the results in the most realistic case $m+b \gg a^2$.

3.3.3 Decoders

Our decoders are essentially the pass-gate-based multiplexers (Fig. 3.5a), with all CMOS transistors of minimum width. We assume that one pass gate, consisting of

one nMOS and one pMOS transistors, can be placed in the area of $4F_{\rm CMOS} \times 2\beta F_{\rm CMOS}$, i.e. matching the CMOS pitch of the memory array. In this case the height of the 2^W-input decoder is simply $2\log_2 W \times 2F_{\rm CMOS}$. Similarly, we have assumed that the height of the a^2-out-of-W barrel shifter implementation (Fig. 3.5b) is equal to $(2\log_2 W + a^2) \times 2F_{\rm CMOS}$.

Considering a 6-transistor latch-style implementation of sense amplifiers [91] with $A_{\rm sense} = 100 \times (F_{\rm CMOS})^2$, and 2-transistor drive buffers (inverters) with $A_{\rm drive} = 25 \times (F_{\rm CMOS})^2$, the increase in the linear size of the crossbar array in the horizontal direction, due to decoders, is

$$\Delta w_1 = [12\log_2 W + 2A_{\rm drive}/(2\beta F_{\rm CMOS})] \times F_{\rm CMOS}. \tag{3.9}$$

Similarly, the array increase in the vertical dimension due to one decoder and one barrel shifter is

$$\Delta w_2 = [8\log_2 W + 2a^2 + (2A_{\rm drive} + A_{\rm sense})/(2\beta F_{\rm CMOS})] \times F_{\rm CMOS}. \tag{3.10}$$

3.3.4 Control Circuitry

The control circuitry consists of two adders of length $\log_2 W$ (bits). Since it is relatively short, we assume the ripple carry implementation with the area of a 1-bit full adder equal to $2000 \times (F_{\rm CMOS})^2$ [112], so that

$$A_{\rm control} = 4000\log_2 W \times (F_{\rm CMOS})^2. \tag{3.11}$$

Control circuitry may be placed into the corner areas between the adjacent cell decoders; therefore the area contribution from the first four terms in Eq. (3.6) may be calculated as

$$\begin{aligned} A_{\rm array} + A_{\rm cell\ decoder} + A_{\rm drive/sense} + A_{\rm control}/g = \\ (W \times 2\beta F_{\rm CMOS} + \Delta w_1)(W \times 2\beta F_{\rm CMOS} + \Delta w_2) \\ + \max(0, A_{\rm control}/g - \Delta w_1 \Delta w_2). \end{aligned} \tag{3.12}$$

The area of mapping table implemented in CMOS is assumed to be

$$A_{\rm mapping\ table} = 2m\log_2 W \times (2F_{\rm CMOS})^2. \tag{3.13}$$

3.3.5 ECC Decoders

Earlier, we have studied the area and delay tradeoffs of fast bit-parallel decoding for a popular class of multiple error-correcting linear codes - BCH codes [107, 112]. (For example, the Reed-Solomon codes [11], which are a subclass of the BCH family, are broadly used in today's flash memories and storage devices [90].) While BCH ECC

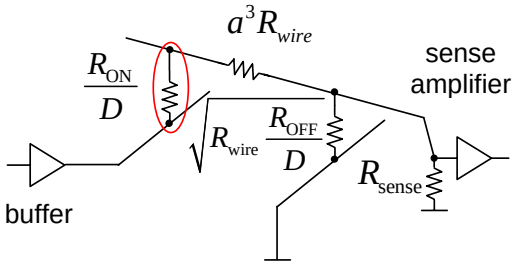

Figure 3.6: Equivalent circuit for the readout operation.

may not be the absolutely best option for our particular problem, they are among the most efficient short codes (with $n \leq 512$), and, e.g., certainly much more powerful than the Hamming codes assumed in our previous work [110]. Our model of a fast bit-parallel decoder for BCH codes is presented in Appendix A. Its analysis shows, in particular, that the decoder area, in the $(F_{\rm CMOS})^2$ units, may be estimated as

$$\begin{aligned} A_{\rm BCH} \approx 125nt(\log_2 n)^2 + 40n(\log_2 n)^2 + 250nt\log_2 n \\ +190n\log_2 n + 1200t(\log_2 n)^2 + 300t^2\log_2 n. \end{aligned} \tag{3.14}$$

Figure 3.4 shows several examples of application of this formula. The results indicate that the decoder area (even for this completely parallel, i.e., most spacious, implementation) is negligible in comparison with that of memory arrays, so that it will not be considered in the optimization procedure described below.

3.4 Speed and Power

3.4.1 ECC Decoder

The BCH ECC decoder delay is also described in the Appendix A. In the units of the standard CMOS fan-out-of-four delay, it may be expressed as

$$\begin{aligned} \tau_{\rm BCH} \approx 0.7t\log_2 n + 0.7t + 8t\log_2\log_2 n + 3.6t\log_2 t \\ +2.5\log_2\log_2 n + 1.8\log_2 t + 1.8. \end{aligned} \tag{3.15}$$

Figure 3.4 shows several results obtained using this formula. We will now show that the delays of read and write operations[1] in the crossbar arrays with metallic nanowires,

[1]The actual write delay can be affected by the fundamental relationship between the retention time and write/erase speed in two-terminal devices. For the usual uniform barriers, the ratio of these two time constants can hardly exceed 9 orders of magnitude, and only for the (so far, hypothetical) structures with optimized barrier profile [69] may be increased to ~ 17 orders of magnitude. Unless such advanced barriers have been implemented, a crossbar memory with a-few-nanosecond

as well as those of moving data between the blocks, are much shorter than that of ECC decoding, expressed by Eq. (3.15). Indeed, let us first estimate the delays inside the blocks.

3.4.2 Intrablock Delay

In order to speed up the intrablock latency, it is beneficial to reduce the signal swing of sense amplifier's input voltage $V_{\rm swing}$, by decreasing the $R_{\rm sense}$. The voltage swing can be found from the equivalent circuit shown in Fig. 3.6. Here we have assumed that all unselected nanowire segments are pulled to the ground through resistance $(R_{\rm wire}R_{\rm OFF}/D)^{1/2}$ which is the input resistance of a semi-infinite ladder formed by the wire resistances $R_{\rm wire}$ of the length $2F_{\rm nano}$ and crosspoint nanodevices in OFF state $R_{\rm OFF}/D$. Note that the pulldown resistance $R_{\rm pd}$ shown in Fig. 3.3 may be neglected here since it is much less than $R_{\rm OFF}/D$. Also, in the equivalent circuit we neglect the resistance of bottom layer nanowires, which is at most $a^2R_{\rm wire}/2$, i. e. much smaller than the worst-case resistance $a^3R_{\rm wire}$ of the top level (quasi-horizontal) nanowire.

For all reasonable parameter sets, it is possible to pick a realistic value of $R_{\rm ON}$ such that the following conditions are satisfied:

$$R_{\rm ON}/D \ll a^3R_{\rm wire}, \tag{3.16}$$

$$R_{\rm sense} \ll a^3R_{\rm wire}, \tag{3.17}$$

$$R_{\rm sense} \ll (R_{\rm wire}R_{\rm OFF}/D)^{1/2}. \tag{3.18}$$

In this case, all the formulas describing the equivalent circuit may be significantly simplified. In particular, the voltage swing is simply

$$V_{\rm swing} \approx R_{\rm sense}/(a^3R_{\rm wire}) \times V_{\rm READ}. \tag{3.19}$$

The limit to the reduction of $V_{\rm swing}$, is set up by the requirement for the swing $V_{\rm swing}$ to be larger than the possible total noise swing $\Delta V_{\rm N}$. In general, the equivalent circuit shown in Fig. 3.6 allows one to calculate the total noise r.m.s. voltage fluctuation $V_{\rm N}$ contributed by the shot noise on the nanodevices in their ON and OFF states, and the thermal (Johnson-Nyquist) noise on the nanowires and $R_{\rm sense}$. However, the inequalities (3.16)-(3.18) ensure that $V_{\rm N}$ is dominated by the thermal noise of $R_{\rm sense}$, so that

$$V_{\rm N} \approx [k_{\rm B}T/(2C_{\rm wire} + C_{\rm CMOS})]^{1/2}, \tag{3.20}$$

write/erase time may need periodic refresh similar to that used in DRAM. Alternatively, when microsecond-scale write time is acceptable, the retention time may be above 10 years, the industrial standard for non-volatile operation.

where $C_{\rm wire}$ is the capacitance of the global quasi-horizontal nano-wire, of length $Wa \times 2F_{\rm nano}$, while $C_{\rm CMOS}$ is a capacitance of the CMOS data line of the same length (see the violet data lines in Fig. 3.2). The typical value of $C_{\rm CMOS}$ for the considered CMOS technology nodes is about 0.1 fF/μm. The doubling of $C_{\rm wire}$ in Eq. (3.20) is due to the fact that two CMOS select lines have to be activated for a read operation (Fig. 3.2) and therefore two quasi horizontal nanowire lines will be charged.

Assuming Gaussian distribution of the noise and very strict requirement for the bit error rate to be below $q_{\rm gate} = 10^{-23}$ (corresponding for example, to a mean time between failures of at least 10^8 hours [1] at the very aggressive aggregate memory bandwidth of 1Tbit/s), the maximum swing $\Delta V_{\rm N}$ of this noise, calculated from the equation

$$1 - {\rm erf}(\Delta V_{\rm N}/\sqrt{2}V_{\rm N}) = 2q_{\rm gate}, \tag{3.21}$$

is equal to $10V_{\rm N}$.[2]

The full intrablock latency may be estimated as

$$\tau_{\rm intrablock} \approx R_{\rm sense}(2C_{\rm wire} + C_{\rm CMOS}). \tag{3.22}$$

3.4.3 Interblock Delay

This delay can be reduced by adding repeaters on the periphery of each block, so that the delay is

$$\tau_{\rm interblock} \approx \sqrt{L}R_{\rm CMOS}C_{\rm CMOS}/2, \tag{3.23}$$

where resistance $R_{\rm CMOS}$ is, similarly to $C_{\rm CMOS}$, proportional to the linear block size. Here we assume that the area overhead to the repeaters is insignificant.

3.4.4 Example

Now let us consider a particular, but typical case $W = 256$, $F_{\rm CMOS} = 45$ nm and $F_{\rm nano} = 4.5$ nm (and hence $a = 16$). In this case the nanowire resistance $R_{\rm wire}$ turns out to be about 14 Ω, $a^3R_{\rm wire} \approx 57$ KΩ, while $C_{\rm wire} = 7.4$ fF. By using nanodevices with $R_{\rm ON} = 400$ KΩ $\cong 10^2R_{\rm Q}$, and therefore $R_{\rm OFF} \cong 40$ GΩ, the resistances of the crosspoint devices are, respectively, $R_{\rm ON}/D \approx 5$ KΩ, and $R_{\rm OFF}/D \approx 0.5$ GΩ, since for the considered value of $F_{\rm nano}$ the packing factor D is about 80. Using typical CMOS value $V_{\rm READ} = 1$ V [1], the smallest possible $R_{\rm sense}$ may be found from Eqs. (3.19) and (3.20) to equal 3 KΩ, i.e. a level much lower than the parallel resistance of the semi-infinite ladder $(R_{\rm wire}R_{\rm OFF}/D)^{1/2} \approx 85$ KΩ. It is easy to check that all conditions (3.16)-(3.18) are satisfied, so that our approximate formulas are indeed valid. The

[2]Note that the additional erroneous factor of 2 in the same equation in Ref. 109 resulted in the underestimated performance of CMOL logic circuits.

corresponding intrablock delay in this case is about 55 ps, i.e. much smaller than that of the ECC decoding and hence can be neglected. (Even in case of $F_{\rm CMOS} = 22$ nm and $F_{\rm nano} = 2.2$ nm the intrablock delay, which is linearly proportional to the $R_{\rm wire}$, is only 400 ps.)

Similarly, the interblock delay (Eq. 3.23) can be neglected since it is below 1 ns even for the "worst" case of $F_{\rm CMOS} = 22$ nm. Indeed, for CMOS metal 1 layer lines, the typical specific resistance is about 40 Ω/μm [1], i.e. for the block size $W = 256$, the CMOS wire resistance $R_{\rm CMOS}$ is approximately 700 Ω. Assuming $F_{\rm nano} = 2.2$ nm and $N = 1$ Tbit (which gives $\sqrt{L}$ of the order of 1000), the corresponding interblock delay is about 0.65 ns, i.e. also much smaller than ECC decoding delay.

Due to this fact, our results are rather insensitive to the hardware assumptions made above.

3.4.5 Power

The assumptions and formulas given above allow also a calculation of power consumption in all components of the memory. Such calculations show that the power consumption is well below the ITRS-specified limits [1]. Indeed, the consumption is dominated by the static power dissipated in nanowires connected to the nanodevices in ON state. Since there might be at most a^2 such nanowires (with the average resistance $a^3 R_{\rm wire}/2$) in a block, the corresponding power for the whole memory can be estimated as $\sqrt{L} \times 2(V_{\rm READ})^2/(aR_{\rm wire})$, giving the power density well below 1 W/cm^2 for all cases we have studied. Note, however, that local overheating is still possible and should be carefully evaluated in future. At such analysis, one may consider an option of a modest increase of $R_{\rm ON}$ and/or decrease of $V_{\rm READ}$, since the resulting increase of the block latency would not affect the total memory access time significantly.

3.5 Optimization

Requiring that the total yield Y described by Eq. (3.5) is fixed at a certain level, we can use the area model of the Section 3.3 to calculate the total chip area A necessary to achieve a certain useful bit capacity N, and hence the area per useful bit, A/N. The last number, normalized to the CMOS half-pitch area,

$$\mathsf{a} \equiv \frac{A}{N(F_{\rm CMOS})^2}, \tag{3.24}$$

is a very convenient figure of merit that depends only on the ratio $F_{\rm CMOS}/F_{\rm nano}$ rather than on the absolute parameters of the fabrication technology.

We first investigate how a depends on the block size W. Figure 3.7 shows the results of this calculation for fixed granularity and ECC code parameters. Not sur-

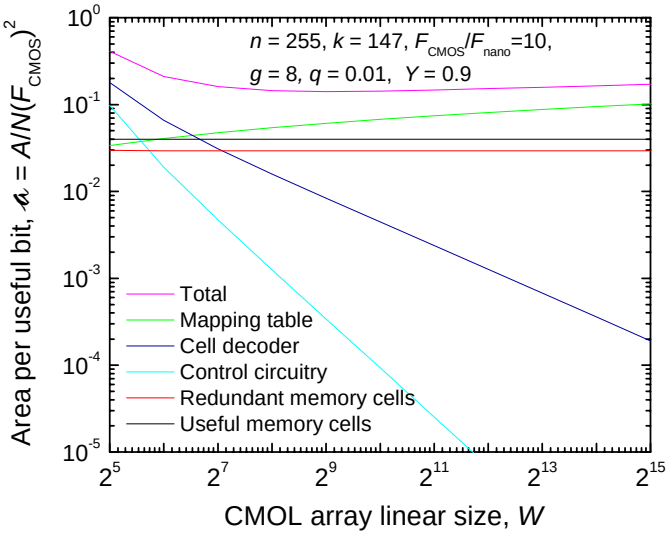

Figure 3.7: The total chip area per one useful memory cell, and its components (all in the units of $(F_{\rm CMOS})^2$), as functions of CMOL array size W (at fixed total memory size N).

prisingly, the exponential explosion of redundancy at $W \to \infty$, pertinent to the global crossbar memory architecture [110], disappears in the considered defect tolerance approach, since the granularity of exclusion does not depend on the block size.

Concerning other contributions to A, for large arrays (in the case shown in Fig. 3.7, for $W > 100$), the most important contributor is the mapping table whose area grows logarithmically with W - see Eq. (3.13). This overhead can be reduced dramatically by choosing appropriate granularity parameter g - see below. The next most important overhead is that of the cell address decoders, which does not allow the reduction of W below $\sim 2^7$. Because of that, in our calculations we have used the fixed value $W = 2^8 = 256$. This is very convenient for the most interesting case $F_{\rm CMOS}/F_{\rm nano} = 10$, because in this case $W = a^2 = 256$, and there is no need in the barrel shifter (Fig. 3.2). Moreover, in this case, and $g \geq 1$, all CMOS data wires of each array are fully used at each operation, resulting in the maximum possible throughput.

Let us now return to the mapping table overhead. Figure 3.8 shows this overhead in comparison with that due to the redundant cells, as a function of data fragment size (granularity). As Eq. (3.13) shows, when the granularity is increased, the mapping table shrinks (because the memory has fewer fragments). On the other hand, the redundancy overhead is skyrocketing, because the use of heavy BCH codes, which are necessary to handle such long fragments, is very expensive in terms of decoding latency. Only if the fraction of bad bits is very low, this increase of the redundant

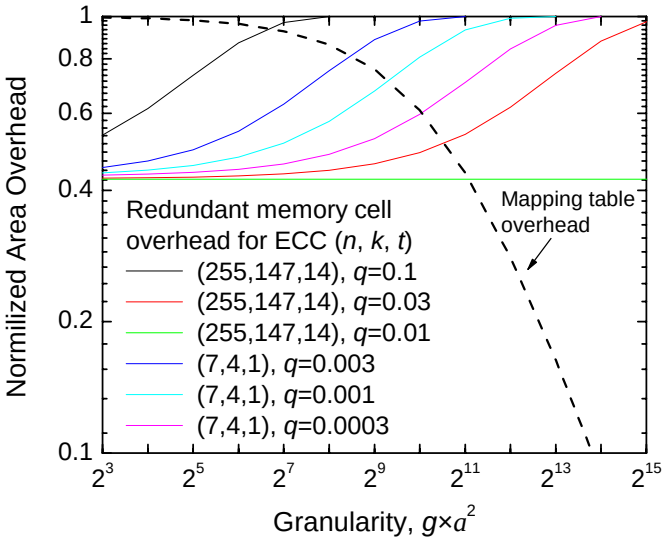

Figure 3.8: The normalized memory area overheads due to the redundant memory cells and the mapping tables. Each absolute overhead $A_{\rm x}$ is normalized as $A_{\rm x}/(A_{\rm x} + A_{\rm useful})$, where $A_{\rm useful} = 4N(F_{\rm nano})^2$ is the area of useful memory cells.

cell number (in order to get fixed yield Y) is deferred to very high data fragment size.

3.6 Results

Figure 3.9 presents typical final results[3] of our optimization procedure, carried out for several values of the total access time (for our parameters, dominated by the ECC decoding time). The cusps on the curves are due to sudden changes of discrete parameters (ga^2, n and k) for which the largest memory density is achieved. These changes are clearly visible in Table 3.1 which provides details of the optimization results for the particular case $\tau = 10$ ns and $F_{\rm CMOS}/F_{\rm nano} = 10$. One can see that as the fraction q of bad memory cells is increased, the granularity ga^2 has to be decreased in order to sustain acceptable probability of ECC-correctable data fragments. (Only for very high q the fragment length ga^2 becomes less than the nanowire segment size a^2, i.e. the fragment may be stored in a single block.) As a result, optimal error correcting codes become shorter, i.e. n and k decrease.

In particular, Fig. 3.9 shows that CMOL memories may become denser than purely CMOS ones at the fraction of bad bit devices as high as ~ 15 if the latency

[3]Though formally the results depend on the total memory size N and yield Y, they are rather insensitive to these parameters in the range of our interest ($N \approx 10^{12}$ bits, $Y \approx 90\%$). As Fig. 3.9 shows, the required memory access time τ also has a marginal effect on density, provided that τ is not too small.

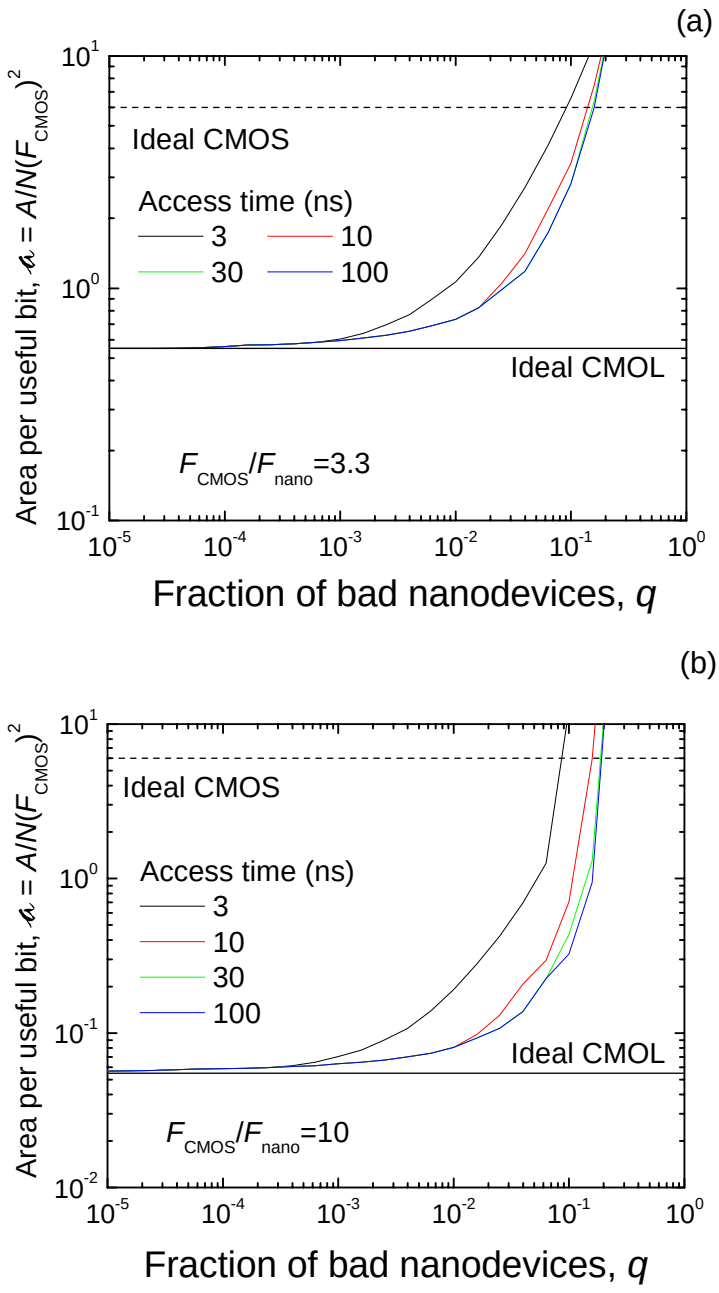

Figure 3.9: The total chip area per one useful memory cell, as a function of the bad bit fraction q, for several values of the memory access time and two typical values of the $F_{\rm CMOS}/F_{\rm nano}$ ratio. The horizontal lines indicate the area for "perfect" CMOS and CMOL memories. In the latter case, this line shows our results for negligible q, while for the former case we use the ITRS data [1] for the densest semiconductor (flash) memories.

Fraction of bad bits, q	Redundant segments per block, b	Useful segments per block, m	Granularity, ga^2	ECC useful bits, k	ECC total bits, n	ECC correctable errors, t	ECC latency, τ (ns)	Area per useful bit, $\mathscr{a}$
0.00001	136	61063	131072	239	255	2	1.70	0.057
0.00003	1108	60091	131072	239	255	2	1.70	0.058
0.00010	116	61083	131072	231	255	3	2.55	0.059
0.00032	1389	59810	65536	231	255	3	2.55	0.06
0.00100	271	60928	131072	215	255	5	4.27	0.063
0.00316	724	60475	131072	199	255	7	5.77	0.069
0.01000	2433	58766	32768	179	255	10	8.71	0.081
0.03162	9266	51933	8192	57	127	11	9.36	0.156
0.10000	33698	27501	1024	16	63	11	9.19	0.707

Table 3.1: Optimal parameters of CMOL memory for the case $F_{\rm CMOS} = 45$ nm, $F_{\rm nano} = 4.5$ nm, $Y = 90\%$, $N = 1$ Tb, for several values of the fraction q of bad crosspoint devices.

requirement is not too small (i.e. > 10 nm) for both considered cases of pitch ratio. On the other hand, to reach 5x and 10x advantage in density such fraction of bad bits should be below 5% and 2% for $F_{\rm CMOS}/F_{\rm nano} = 3.3$ and 10 pitch ratios, correspondingly.

The results in this work are much better than those obtained in our earlier work [110], where, for example a 5x density advantage could only be reached (for the same $F_{\rm CMOS}/F_{\rm nano} = 3.3$ ratio) at $q \cong 3\%$ using the impracticably long Exhaustive Search algorithm, while for the realistic Repair Most algorithm, $q_{\rm max}$ was as low as $\sim 10^{-3}$. This improvement is due mostly to the use of more advanced error correcting codes.

Note, however, that our optimistic results for the memory speed are based on the fundamental physical limitations for the crosspoint nanodevice parameters, in particular, $R_{\rm ON}$. For the currently implemented programmable diodes, the picture is somewhat different. For example, for the simple and reproducible CuO_x devices [19], scaled down to $F_{\rm nano} = 3$ nm, the effective value of $R_{\rm ON}/D$ would be ~ 2 MΩ, resulting in intrablock latency of about 50 ns. This means that our results (Fig. 3.9) would degrade only slightly. On the other hand, for the demonstrated reproducible molecular monolayers [6], typical $R_{\rm ON}/D$ of a similarly scaled crosspoint device would be in the GΩ range, so that the memory speed would be much lower. Nevertheless, a considerable progress of the improvement of molecular programmable diodes during the next few years may be readily anticipated.

Chapter 4

CMOL Boolean Logic

4.1 Hardware Architecture

4.1.1 One-Cell Fabric

We have studied two varieties of CMOL FPGA fabrics [109,111,114]. The architecture of the simplest variety, one-cell fabric [109], is very convenient for elaborating the concept and basic properties of CMOL FPGA, though it cannot be used for sequential circuit design. On the other hand, a two-cell fabric [111,114], which is a generalization of the single cell structure, can be used for mapping arbitrary circuits, and all analysis in subsequent sections we will be given for a such variety of CMOL FPGAs. (The discussion of our early results for one-cell CMOL FPGA circuits can be found in Appendix B.)

Figure 4.1 shows a fragment of one-cell CMOL FPGA fabric. Essentially, it is a uniform structure which is built by replicating "basic" cells with an area $A = (2\beta F_{\rm CMOS})^2$. In this case, the angle α is given by the generic formula for CMOL, i.e. $\tan\alpha \equiv 1/a$ (Eq. 2.1), where a is an integer defining the range of cell interaction (Fig. 4.1).[1] For fixed fabrication technology parameters $F_{\rm CMOS}$, $F_{\rm nano}$, and $\beta_{\rm min}$, the lower bound on a is given by inequality:

$$a^2 > (\beta_{\rm min} F_{\rm CMOS}/F_{\rm nano})^2 - 1. \tag{4.1}$$

Each basic cell (Fig. 4.2a) consists of an inverter and two pass transistors that serve two pins (one of each type) serving as the cell input and output, respectively. During the configuration stage, all inverters are disabled by an appropriate choice of global voltages $V_{\rm DD}$ and $V_{\rm gnd}$ (Fig. 4.2a), and testing and setting of all nanodevices is carried out absolutely similarly to memory write operation described in the Sections 3.1 (see also Section **??** and Figure 2.1).

[1] Note that even though the nanowire crossbar in Ref. 109 was rotated by the additional 45° angle, which was convenient for manual mapping, it does not affect the performance results.

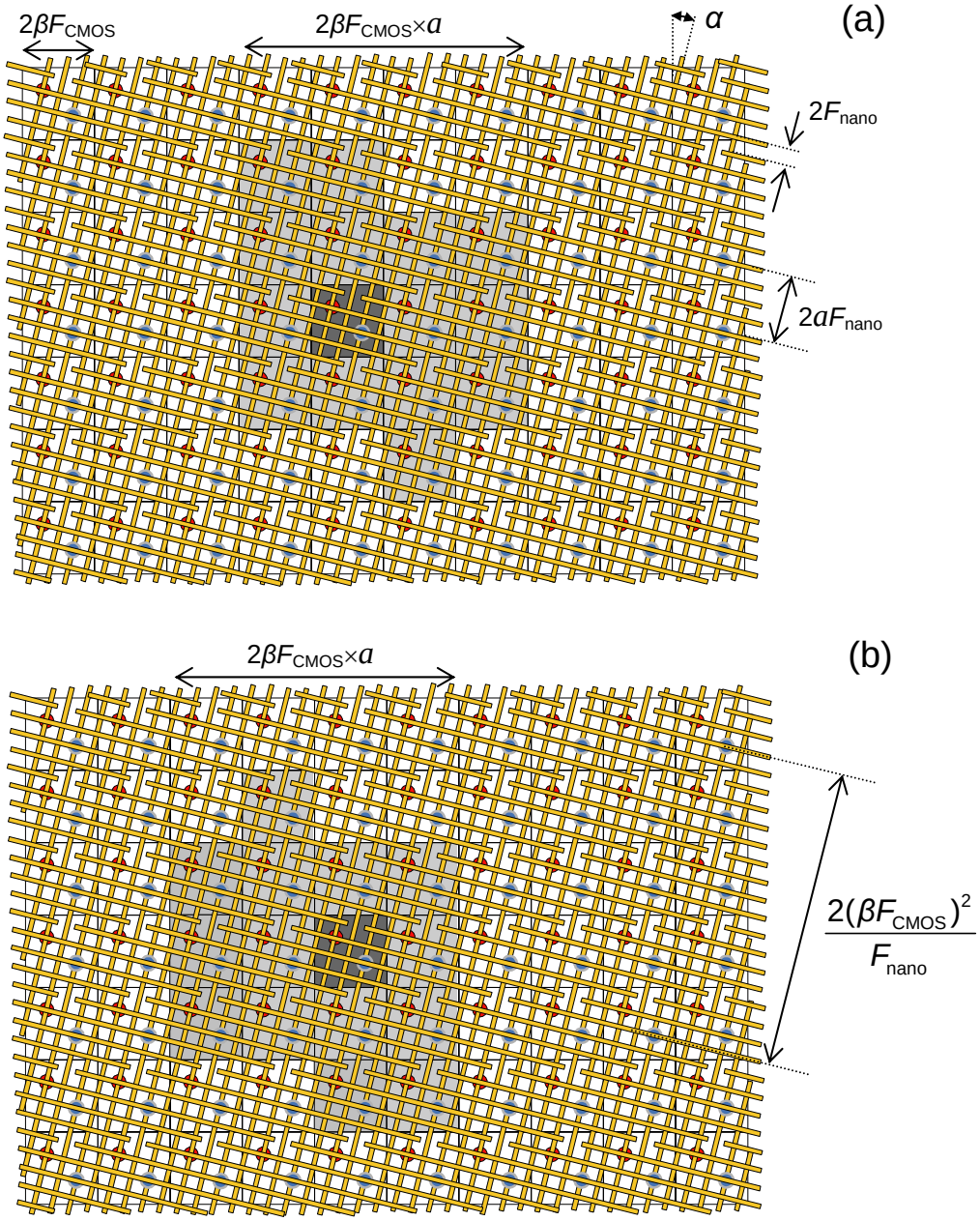

Figure 4.1: The fragment of one-cell CMOL FPGA fabric for the particular case $a = 4$. In panel (a), output pins of $M = a^2 - 2 = 14$ cells (which form so-called input cell connectivity domain) painted light-gray may be connected to the input pin of a specific cell (shown dark-gray) via a pin-nanowire-nanodevice-nanowire-pin links. Similarly, panel (b) shows cells (painted light gray) whose inputs may be connected directly to the output pin of a specific cell (called output connectivity domain).

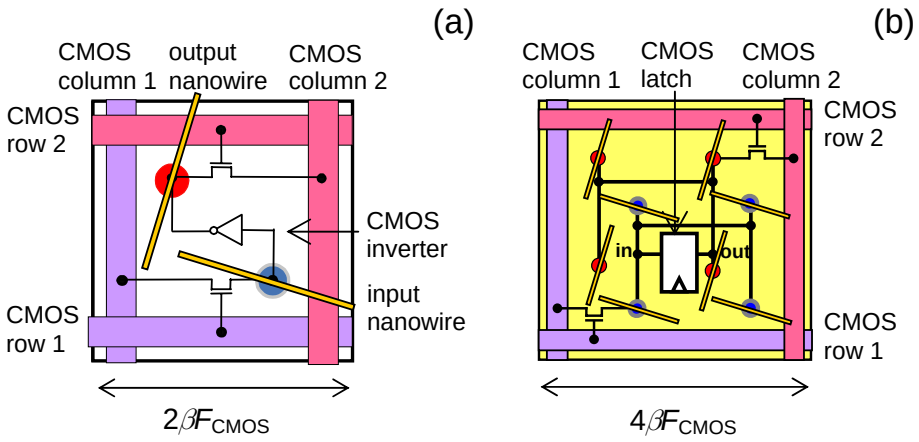

Figure 4.2: CMOL FPGA cell structures: (a) the basic cell and (b) the latch cell. For the sake of clarity both panels shows only nanowires which are contacted by interface pins of the given cells.

In contrast with CMOL memories, nanowires in upper layer are also fabricated with small breaks repeated with period $L = 2(\beta F_{\rm CMOS})^2/F_{\rm nano}$. With this arrangement, each nanowire segment is connected to exactly one interface pin.[2] As a result, each input or output of a basic cell can be connected through a pin-nanowire-nanodevice-nanowire-pin link to each of

$$M = a^2 - 2 \tag{4.2}$$

other cells located within a square-shaped "cell connectivity domain" around the initial cell - see Fig. 4.1. (For infinitesimal gaps, M would equal $a^2 - 1$, but for a more feasible gap width of the order of $2F_{\rm nano}$, the connectivity domain is by one cell smaller.) Note that in reality both input and output cell connectivity domains would be much larger than those shown in Fig. 4.1 for practical values of $a > 10$ and have the same roughly square shape (with some protrusions of the cells on the perimeter of the domain). This fact, as it will be shown in Sec. 4.2, simplifies the design automation for CMOL FPGA circuits.

When the configuration stage has been completed, the pass transistors are used

[2]The best performance is achieved if the pin contacts the wire fragment in its middle, and our analysis has been carried out with this assumption. Since lower layer nanowire segments are cut by upper layer pins, a connection exactly in a center is easily achievable, i.e. by locating upper level pins correspondingly. For upper layer pins, a similar trick can be done, if upper layer nanowire breaks are provided by features of the smae lithographic mask that defines interface pin positions. Also note, that a modest misalignment of the pin and the breaks (by $\sim F_{\rm CMOS}$) reduces the circuit performance only by a small factor of the order of $1/\beta \ll 1$.

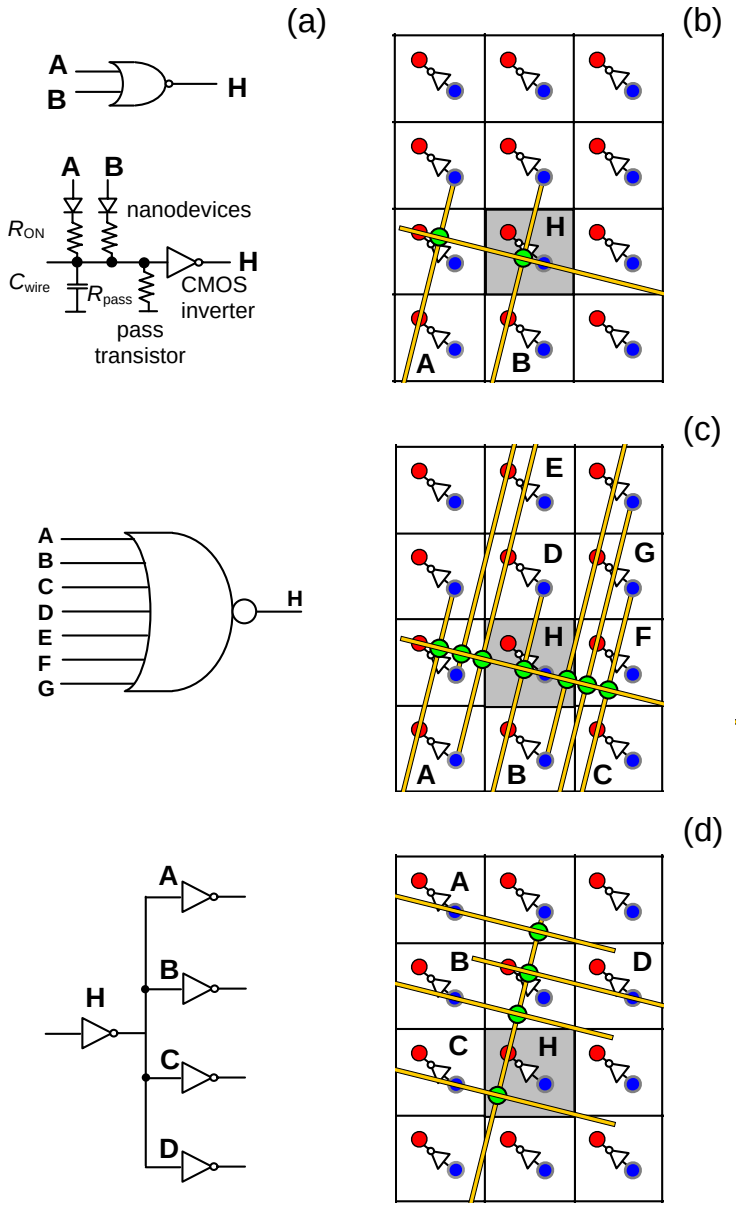

Figure 4.3: Logic and routing primitives in CMOL FPGA circuits: (a) equivalent circuit of fan-in-two NOR gate, (b) its physical implementation in CMOL, (c) the example of 7-input NOR gate and (d) the example of fan-out of signal to four cells. Note that only several (shown) nanodevices on the input nanowires in panels (b), (c), and output nanowire in panel (d) of cell H are set to the ON state, while others (not shown) are set to the OFF state. Also, for the sake of clarity panels (b), (c), and (d) shows only the nanowires used for the gate and the broadcast.

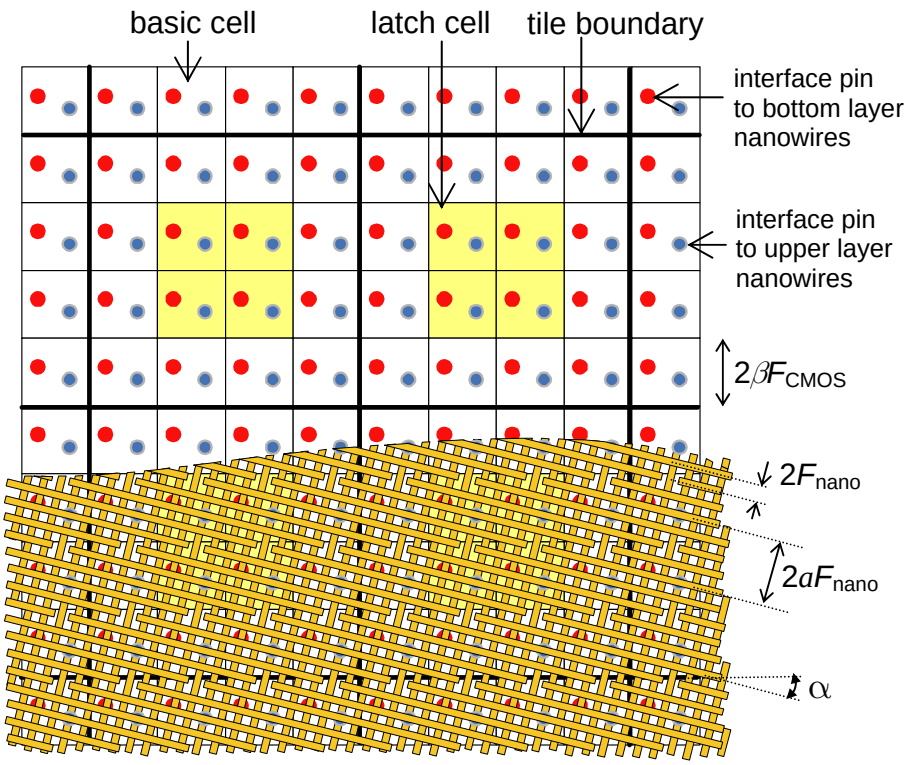

Figure 4.4: A fragment of two-cell CMOL FPGA fabric for the particular case $a = 4$.

as pull-down resistors, while the nanodevices set into ON (low-resistive) state are used as pull-up resistors. Together with CMOS inverters, these components may be used to form the basic "wired-NOR" gates (Fig. 4.3). For example, if only the two nanodevices shown in a Fig. 4.3b are in the ON state, while all other latching switches connected to the input nanowire of cell H are in the OFF (high resistance) state, then cell H calculates NOR function of signals A and B. Clearly, the gates with high fan-in (Fig. 4.3c) and fan-out or broadcast (Fig. 4.3d) may be readily formed as well by turning ON the corresponding latching switches. Having these primitives is sufficient to implement any Boolean function, as well as to perform routing, providing that the hardware resources are sufficient.

4.1.2 Two-Cell Fabric

A genuine optimization of CMOL FPGA circuit architectures would require a completely new set of CAD tools, whose development is a challenging task. At this preliminary stage, our choice was instead to get as much leverage as possible from the existing ideas and algorithms used for mapping and architecture exploration of semiconductor logic, in particular, from the design automation algorithms for island-

type CMOS FPGAs [10].

In order to use such design automation algorithms, we have restricted our design to a specific, simple two-cell-species CMOL fabric. The fabric is a uniform mesh of square-shaped "tiles" (Fig. 4.4). Each tile consists of a shell of T basic cells (Fig. 4.2a) surrounding a single "latch" cell (Fig. 4.2b). The latter cell is just a level-sensitive latch implemented in the CMOS subsystem, connected to 8 interface pins, plus two pass transistors used for circuit configuration. Note that all four pins of each (either input or output) group are always connected, so that the nanowires they contact always carry the same signal. This means that at configuration, groups of four nanodevices sitting on these wires may be turned on or off only together. A simple analysis shows that this does not impose any restrictions on the CMOL FPGA fabric functionality.

CMOS layout estimates assuming a compact layout from, e.g., Ref. 44 have shown that the latch cell requires an area approximately four times larger than that of the basic cell. As a result, for this analysis we have accepted $T = 12$, so that the total tile area is $T + 4 = 16 = 4 \times 4$ basic cells (Fig. 4.4). This provides a latch/logic resource ratio comparable to those of the conventional FPGAs. In fact, the 4-input parity function (the worst-case Boolean function of 4 inputs) can be implemented with 14 four-input NOR gates, while an average 4-input Boolean function requires much less (6 to 8) of such gates. Hence each CMOL tile is crudely similar in functionality to the basic logic element consisting of a four-input LUT and one latch [10].

4.2 Design Automation

4.2.1 General Flow

The convenience of the proposed two-cell CMOL FPGA structure is that, from the design point of view, the CMOL tile can be treated in the same way as that of the island-type CMOS FPGA.

Let us first introduce a very useful concept of "tile connectivity domain" which makes routing of CMOL FPGA circuits similar to CMOS FPGA ones. Similarly to the cell connectivity domain, the tile connectivity domain of a given tile is defined as such fabric fragment that any cell within it can be connected to any cell of the initial tile directly, i.e. via one pin-nanowire-nanodevice-nanowire-pin link (Fig. 4.5). Just as for cell connectivity domains all tile connectivity domains are similar and have square shape. (Note that we assume that input and output tile connectivity domain are the same.) The linear size $\mathbb{A}$ of the tile connectivity domain for the assumed tile size $T = 16$ can be found as

$$\mathbb{A} = 2\lfloor a/8 \rfloor - 1. \tag{4.3}$$

For instance, Fig. 4.5 shows a tile connectivity domain for the case $\mathbb{A} = 5$. (In more

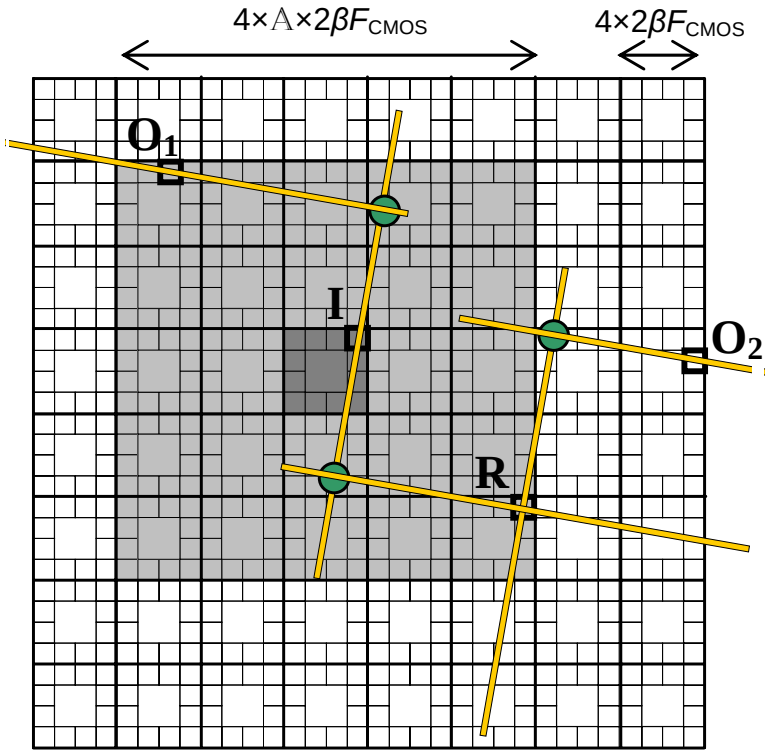

Figure 4.5: Tile connectivity domain: Any cell of the central tile (shown dark gray) can be connected with any cell in the tile connectivity domain (shown light gray) via one pin-nanowire-nanodevice-nanowire-pin link (e.g., cells I and O_1). Cells outside of each other's tile connectivity domain (e.g., I and O_2) can be connected with additional routing inverters (e.g., R). Note that nanowire width and nanodevice size are boosted for clarity. For example, for the considered CMOL parameters, 1600 crosspoint nanodevices may fit on one basic cell area.

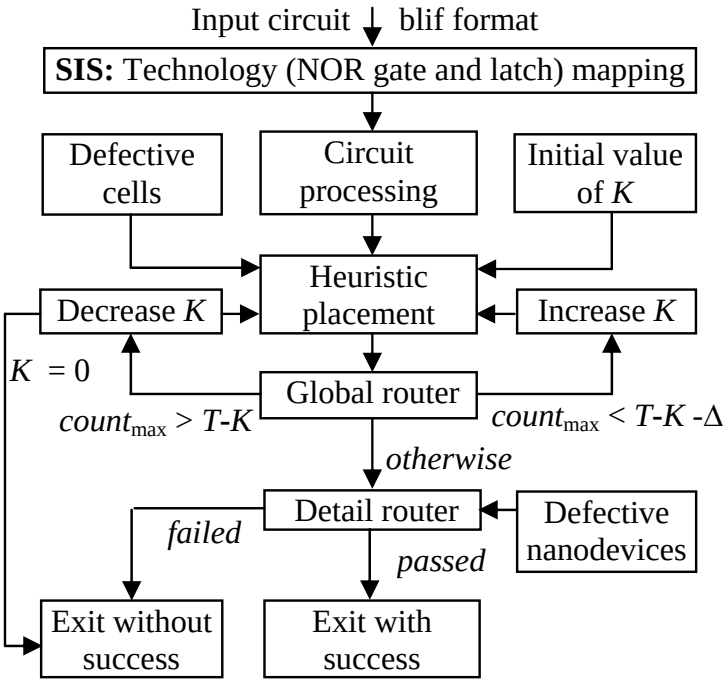

Figure 4.6: CMOL FPGA design flow used in this work.

realistic cases $a = \beta F_{\rm CMOS}/F_{\rm nano} \approx 40$, i.e. $\mathbb{A} \approx 9$.)

The main idea of the proposed design flow for CMOL FPGAs (Fig. 4.6) is to reserve some number of basic cells $(T-K)$ inside each tile for routing purposes, while use the rest (K) cells for logic during the placement step. The placer tries to put gates into such locations (with maximum one latch and K NOR gates per tile) so that their interconnect is local or, equivalently, is within tile connectivity domain of each other. At the global routing step, idle cells inside each tile are used to interconnect global connections. If there is a congestion after the global routing step, i.e. the number of requested basic cells during routing $count_{\rm max}$ is larger than the actual number of idle cells $T - K - \Delta$ (here Δ is parameter which allows to trade-off the number of iterations with the mapping quality), then we decrease K and repeat the flow again until there is no congestion.

4.2.2 Technology Mapping and Circuit Processing

More specifically, every circuit is first mapped into the network of NOR gates (so far with a rather artificial maximum fan-in of 7) and latches, using the SIS package [95]. After that, at the circuit processing step, all inverters (1-input NOR gates) are removed from the circuit (Fig. 4.7). (For the considered benchmark set, such inverters comprise about 28% of the total number of gates on the average.) The idea

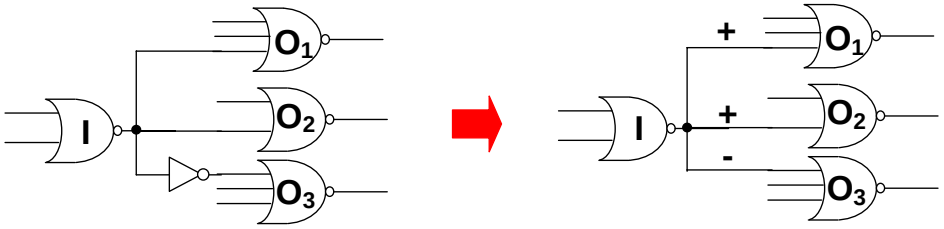

Figure 4.7: Example of preprocessing step. At preprocessing step the inverter between gates I and O_3 is removed and polarity of the connection between these gates is set to negative.

behind the processing step is that inverters which are synthesized by SIS are not different logically from the routing inverters and can be more efficiently (in terms of performance and CAD running time) used during the global routing step (as opposed to the situation when they are treated as logic gates and have fixed positions after the placement step). Instead of the removed inverters, every connection a net[3] is assigned a "polarity" property, which is just an additional information for a placer and global router to specify whether an odd or even number of routing inverters should be used when interconnecting nets. A negative polarity (shown with "-" sign on figures and representing by integer 1) assumes that the signal should be inverted, while the positive polarity (shown with "+" sign and represented by 0) means no inversion. Note that even if two gates, connected with the negative polarity net, are located in the tile connectivity domain of each other, the connection should be made with at least one (or odd number in general) routing inverter(s).

4.2.3 Placement

The placement algorithm follows very closely the heuristic one used in the VPR tool [10, 78]. It is based on a simulated annealing algorithm with some additional techniques employed for a faster and better solution. According to our simulation results, the best area and delay in mapping circuits to CMOL FPGA can be achieved using the timing-driven placement algorithm with $\lambda = 0.5$, the criticality exponent equal to 1, the time-analysis interval of about 10000 successful swap/moves, and with similar temperature and $D_{\rm lim}$ schedules. (For a detailed explanation of these parameters and the algorithm, see Ref. 78, in particular Fig. 2.)

In this algorithm, the wiring cost is calculated as a sum of costs for all connections in the circuit, while the cost of a single connection (Hop) reflects the number of hops

[3]Here by "net" we denote the interconnect between gates running the same signals, while "connection" always assume interconnect between one pair of gates. A net may be comprised of more than one connection. For example the net on Fig. 4.7 consists of three connections, i.e. I to O_1, I to O_2, and I to O_3.

(routing inverters) needed between the tiles having the input and output gates of the connection with the account of polarity. The latter is calculated via *SimpleHop* function, which returns the number of hops between two tiles without accounting for polarity property), i.e.

$$SimpleHop(tile_{\text{in}}, tile_{\text{out}}) = \left\lfloor \frac{2\max\left(|x_{\text{in}} - x_{\text{out}}|, |y_{\text{in}} - y_{\text{out}}|\right) - 1}{\mathbb{A} - 1} \right\rfloor, \tag{4.4}$$

$$Hop(tile_{\text{in}}, tile_{\text{out}}, polarity) = 2\left\lceil \frac{SimpleHop(tile_{\text{in}}, tile_{\text{out}}) + polarity}{2} \right\rceil - polarity, \tag{4.5}$$

$$WiringCost = \sum_{i=1}^{N_{\text{connections}}} Hop[tile_{\text{in}}(i), tile_{\text{out}}(i), polarity(i)]. \tag{4.6}$$

Here x, y is the location of a tiles with input and output gates of the considered connection, and $N_{\text{connections}}$ is the total number of connections in the circuit.

The timing cost was calculated according to the Eqs. (6-8) of Ref. 78, using the slack (normalized with respect to inverter delay), also obtained through Eqs. (4), (5) of that work. Using cost functions, a placement algorithm tries to put connected gates close to each other, ideally within each other's tile connectivity domains.

Note that, in contrast with typical CMOS FPGA placement tools [10], the swapping and moving is performed with gates rather than clusters of gates. Even though the clustering helps to perform the placement step faster, it also has disadvantages. First of all, it is not obvious how to pack the gates into clusters (tiles), since any gate connection within the tile connectivity domain (as opposed to connections within a tile in most of CMOS FPGAs) is virtually free. (This is because the nanodevice utilization is rather small, see, e.g. final results in table 4.1.) On the contrary, placement of two gates as far as connectivity domain allows, might help to reduce the number of routing inverters. Second, clustering is rather impracticable if each tile has different number of basic cells, which can be due to the fact that some of the cells are defective (for the discussion of this point see Section 4.2.5).

Figure 4.8 gives an example of wiring cost calculation for $\mathbb{A} = 5$. In this example, one input (I) and three output (O_1, O_2, and O_3) gates are located in tiles with positions (1, 1), (6, 10), (10, 6), and (7, 1), correspondingly. Hence, according to Eqs. (4.4, 4.5, 4.6) the wiring cost function is $WiringCost = 4 + 4 + 3 = 11$.

4.2.4 Global Routing

The next operation, global routing, which is needed to connect the gates which are not within tile connectivity domains of each other, performed in two stages (Fig. 4.9). At the first stage the algorithm connects global nets by allocating the routing inverters,

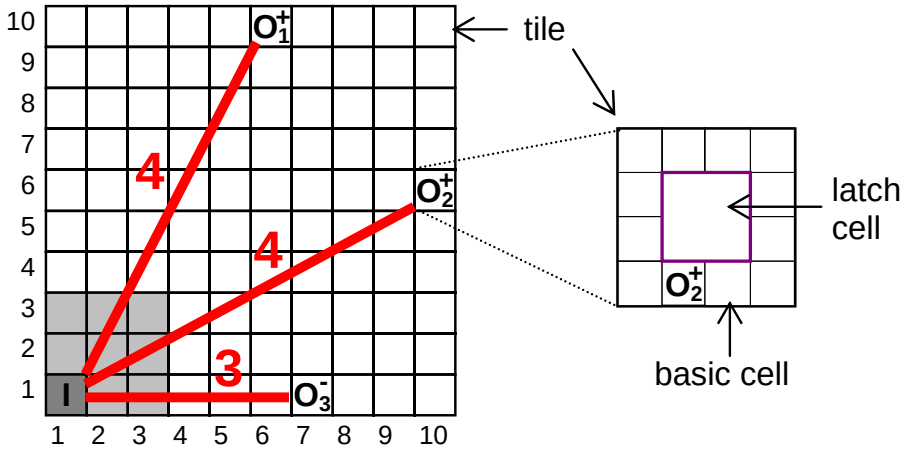

Figure 4.8: Wiring cost function calculation example. Assuming the linear size of the tile connectivity domain $\mathbb{A} = 5$, the wiring cost of the circuit mapping with connections between gate I and gates O_1, O_2 and O_3 is 11.The numbers shown in red are the contributions of specific connections to the cost function. In this particular example, these numbers also give parameters $Hop_{\min}$ used in the global routing procedure (see Sec. 4.2.4 below). Note, that the actual location of gates inside the tile does not matter at this stage.

INPUT:
A) mapping of gates to tiles
B) map of defective cells (***def***)
START:
1: Generate ***input_netlist*** from input mapping
2: Initialize counter (***count***) of unused basic cells for each tile and adjusting accordingly to ***def***
3: Create ***sorted_netlist*** by sorting ***input_netlist*** by the number of outputs (largest first) while sorting entries with the same number of outputs by the cost function (lowest first)
4: For each net (***curnet***) from the ***sorted_netlist***
5: RouteNetGreedy(***input***, ***outputlist***), where ***input***, ***outputlist*** are the coordinates of an input and output gates of ***curnet***, respectively
6: If there are congestions (for some tiles ***count*** > *T* - *N*) {
7: For each tile (***curtile***) {
8: For each ***curgate*** from ***curtile*** {
9: If (***curgate*** from ***curtile*** is inverter and fanout = 1) {
10: Remove ***curgate*** and all consecutive inverters with (fanout =1) connected to ***curgate***'s input and output
11: If (RouteNetExhaustively(***curnet***) is false)
12: Exit without success
13: }}}} Exit with success

Figure 4.9: A top-level pseudo code for global routing algorithm.

so far without any tile capacity limitations. In this step, each connection of the net is routed with the smallest possible number of inverters. Moreover, if the net has more than one output, the algorithm tries to minimize the number of routing inverters by sharing them among connections of the same net. Since this problem is equivalent to finding the shortest-path Steiner tree [47], which is exponentially hard, our algorithm is heuristic.

The algorithm, formally presented in Fig. 4.10, is close to the so-called RSA heuristic algorithm [92]. It is based on the recursive function (*RouteNetGreedy*) which, in a single iteration, finds the quasi-optimal position for the routing inverter in the tile connectivity domain of the input gate (***input***) for the given set of output gates (***outputlist***).

The algorithm can be better explained using the example shown in Fig. 4.11. Let us consider the case $\mathbb{A} = 5$ and suppose that the algorithm needs to route input gate I to three output gates O_1, O_2, and O_3 (corresponding to the tiles colored yellow and cyan, respectively. At first, for each pair of ***input*** and output gates from ***outputlist***, the algorithm determines the minimal number $Hop_{\min}$ of inverters required for routing, taking into account the polarity of each connection. Then *RouteNetGreedy* function ranks all tiles of the ***input*** tile connectivity domain (step I in Fig. 4.11). The rank of a tile shows how many output gates from ***outputlist*** can be routed to the ***input*** gate with the minimal path, i.e. with $Hop_{\min}$ inverters, using a routing cell in the given tile. In the new iteration, the routing cell tile location (R_1), chosen "greedily" among the tiles with maximum rank, is considered as a new ***input*** tile, while the set of output gates, which contributes to the rank (e.g., for R_3 in step III of Fig. 4.11, gates O_1 and O_2) becomes new ***outputlist***, etc. Once these outputs have been routed (step IV in Fig. 4.11) they are not considered during the ranking of the rest of output gates, e.g., gate O_3 (step V in Fig. 4.11).

Note, that it is possible to have congestion after the first stage, i.e. the number of routing cells assigned to a tile may be larger than the one physically available. However the algorithm at the first stage tries to avoid such congestion by keeping an occupation counter of the total number of routing cells (*count*) requested by the algorithm in each tile. At the start of the routing procedure, the counter value is set to zero. If for a certain iteration there are several tiles with the same rank, the preference is given to the tiles with the least utilized routing cells. Also, routing nets in a specific order, i.e. nets having fewer outputs and larger cost last, helps assigning routing cells more evenly throughout the tile array.

If there are still congestions after the first stage of the algorithm (Fig. 4.12), they are dealt with at the second stage, using recursive function *RouteNetExhaustively* (Fig. 4.10). The idea is to reroute exhaustively the nets whose routing inverters are located in the congested tiles. To make this stage more powerful, we perform slack analysis of the congested nets, and (provided that the critical path stays the same)

```
Function RouteNetGreedy (input, outputlist)
1:  For each tile (curtile) from input's connectivity domain {
      curtileoutputcount = 0
      curtileoutputlist = Ø
2:    For each output (curoutput) from outputlist  {
3:      If(SimpleHop(curtile, curoutput) + 1 =
          Hop(input, curoutput, curoutput's polarity) )  {
4:          curtileoutputcount ++
5:          Add curoutput to curtileoutputlist  }
6:      Among tiles with largest curtileoutputcount choose the one
        which has the largest count and increment count for this tile
7:      For each output (curoutput) from curtileoutputlist
8:          If(Hop(curtile, curoutput, curoutput 's polarity)=0)
9:              Delete curoutput from curtileoutputlist
10:     If(curtileoutputlist != Ø)
11:         RouteNetGreedy(curtile, curtileoutputlist)
12:     outputlist = outputlist - curtileoutputlist
13:     If(outputlist != Ø)
14:         RouteNetGreedy(input, outputlist) }

Function RouteNetExhaustively (curnet)
1:  Calculate slack for curnet
2:  For curhops = Hop(curnet's input, curnet's output, curnet's polarity)
    to slack steps of 2 {
3:    Find all tiles (tilelist) which might be used to route curnet using curhops
      routing inverters
4:      For each tile (curtile)  from tilelist
5:          For hops = SimpleHop(curnet's output, curtile) to
            slack - SimpleHop(curnet's input, curtile)
6:              RankTile(curtile, hops, curnet's output, count)
7:    Choose tile (besttile) in curnet's input tile domain with the largest rank
8:    If(rank != 0) {
9:      Route curnet  by traversing tiles using bestpath  starting from besttile
10:     Return true
11: }} Return false

Function RankTile(inputtile, hops, outputtile)
1:    Let tiledomain  be tile domain of inputtile
2:    Eliminate all tiles (curtile) from tiledomain such that
      SimpleHop(curtile, outputtile) ≥ hops
3:    For each tile (curtile) from tiledomain {
4:      If(rank (curtile, hops -1) is undefined) {
5:          If(hops =1)  {
6:            rank (curtile, hops -1) = count (curtile)
7:            bestpath (curtile, hops -1) = outputtile
8:          } Else RankTile(curtile, hops -1, outputtile) }
9:    Choose tile (besttile) from tiledomain with largest rank (hops -1)
10:   If(rank (besttile, hops -1) = 0 or count (inputtile) = 0 ) {
11:     rank (inputtile, hops) = 0
12:   Else {
13:     bestpath (inputtile, hops) = besttile
14:     rank (inputtile, hops) = rank (besttile, hops -1) + count (inputtile) }
15:   Return rank (inputtile, hops)

Function Hop (tile1, tile2, polarity)
1:  hops =  SimpleHop (tile1, tile2)
2:  If((hops is even and polarity is odd) or (hops is odd and polarity is even)) {
3:    Return hops + 1
4:  } Else Return hops

Function SimpleHop (tile1, tile2)
1:  Let x1, y1 and x2, y2 be tile1 and tile2 coordinates, respectively
2:  Return floor(2*max(abs(x1-x2),abs(y1-y2)-1)/(A-1))
```

Figure 4.10: A pseudo code for the global routing subroutines.

allow for more flexible rerouting with the number of routing inverters larger than its minimum.

Finally, after finding the largest K enabling successful global routing, all circuits are functionally verified. This is done by first converting the mapped circuit into the BLIF format [95], and then comparing them with the original circuit with the help of the ABC verification tool [3].

4.2.5 Defect Tolerance: Defective Cell Avoidance

What would happen if CMOS cells whose pins are connected to a broken or shorted (input and/or output) nanowires are defective? In general, it would be still possible to use cells with broken nanowires, especially if the break happens close to the naturally made gap. Similarly, cells with shortened nanowires might be used for routing purposes only. However, it would also mean that the cell connectivity domains would be different from one cell to the other, complicating the design automation tools.

Moreover, to keep the tools simple we may also assume that any stuck-on-close-type defect results in a defective cell. Obviously, this is very inefficient way of coping with such defects. However, such approach might be used if the number of stuck-on-close defects is very small (i.e. much less than one defect per CMOS cell).

This is why all of the defects in the second group discussed in Section 2.4, i.e., broken or shorted nanowires, stuck-on-close-type nanodevices, defective nano-to-CMOS interface, and defective CMOS circuitry (in particular CMOS inverter and pull down pass transistor), may be considered equivalent to defective cells.

The proposed placement and global routing algorithms can be easily modified to provide defect tolerant mapping with respect to faulty cells. Indeed, since there is no clustering step, the only requirement is that the initial (random) placement and also any gate moves during annealing procedure should be performed with the account of the actual number of defect-free cells in a tile (which will be different from one tile to the other). Similarly, during the global routing, instead of having the *count* parameters initialized to zero before the first stage of the algorithm it is set to some positive number reflecting how many basic cells are defective in a particular tile.

4.2.6 Defect Tolerance: Detailed Routing around Defective Nanodevices

Up to that point in the design flow, the position of the gates inside each tile was not important. These locations are specified at the final step of the flow - detail routing (Fig. 4.6). In the first part of this detail routing step the position of the gates is assigned greedily inside each tile, assuming perfect CMOL fabric without any stuck-on-open defects. (In the case if there is indeed no such defects this would be the end

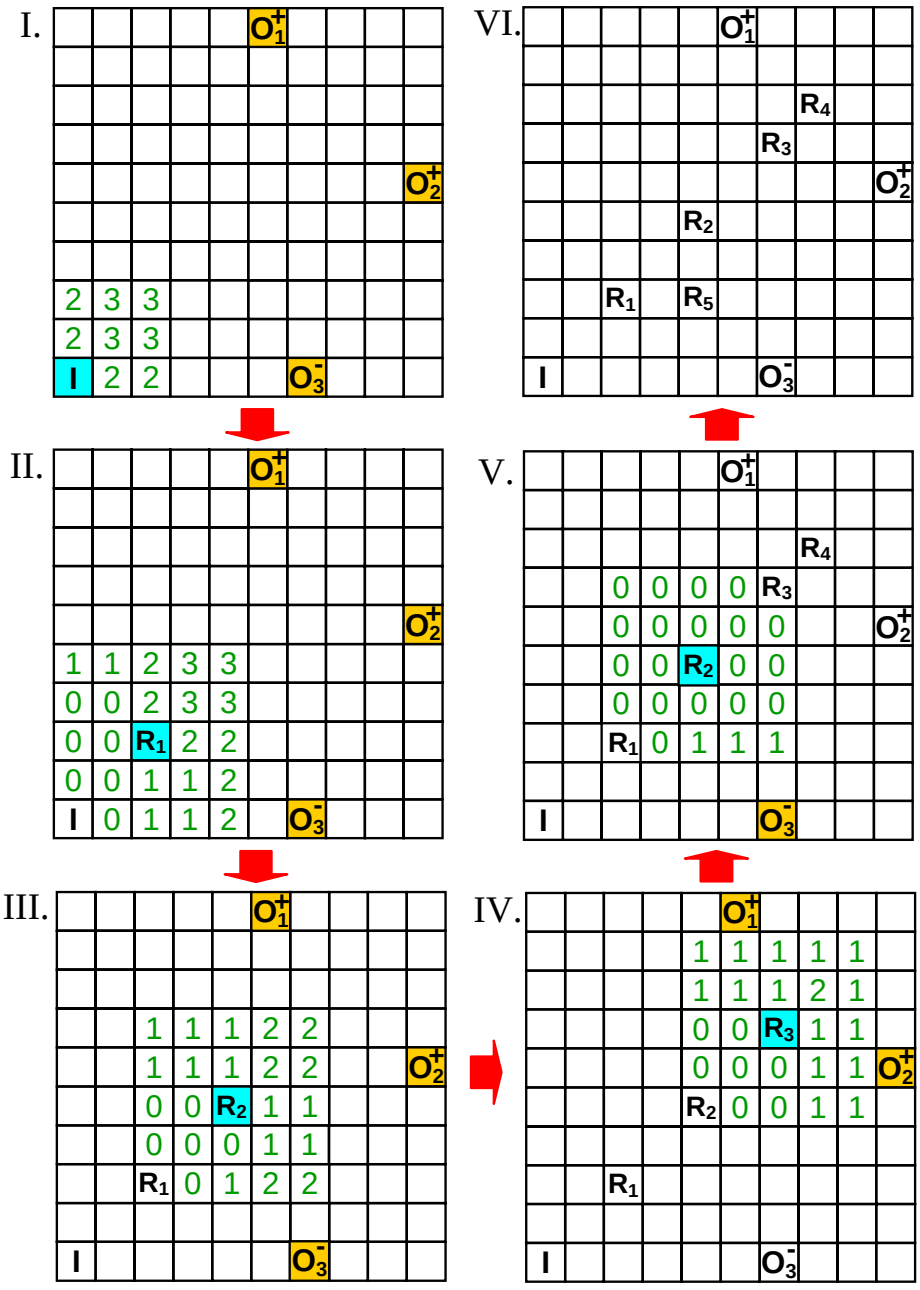

Figure 4.11: Example of first stage of single net global routing for the case $\mathbb{A} = 5$.

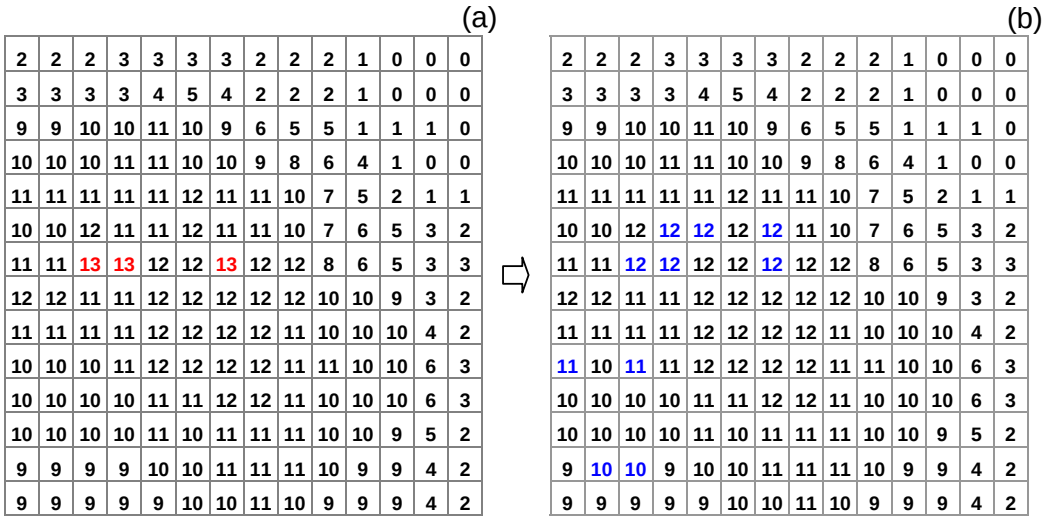

Figure 4.12: Example of second stage of the global routing for s298.blif circuit: tile capacity (a) after stage 1 and (b) after stage 2. On panel (a) the tiles with capacity numbers shown in red requested more resources than physically available. These congestion are rerouted on stage 2 using resources from tiles with numbers shown in blue.

of the design automation flow, i.e. successful mapping of the circuit.) In the second part of detail routing step, the gates are relocated to produce defect free mapping, using the algorithm formally presented in Figure 4.13 [108].

The algorithm is based on sequential attempts to move each gate from a basic cell with bad input or/and output connections to a new basic cell, while keeping its input and output gates in fixed positions. At the start of the algorithm each gate except latches are allowed to move to a different position.[4] (Note that according to the CMOL FPGA topology shown in Figs. 4.1, 4.4, in each position the basic cell uses the different set of nanodevices.)

At such move, the gate may be swapped with another one (whose position was not fixed previously by the algorithm), provided that all connections of the swapped gates can be realized with the CMOL fabric. In order to implement this idea, we first calculate the "repair region" of the gate, where it could be moved if there were no other cells around; this region is just the overlap of the cell connectivity domains of all its input and output cells. For example, for the circuit shown in Fig. 4.14a, gate A can be moved to any cell of the repair region painted pink in Fig. 4.14b, which is the intersection of the cell connectivity domains of its output and input gates 1 and 4. If some cell of the repair region is already occupied by another gate and this cell

[4]The tile location of the gate is not important in the detail routing, i.e. any gate can be moved outside tile boundaries as long as gate's interconnects are physically possible.

```
INPUT:
A) design_list  (list of gates mapped onto a perfect CMOL fabric,
    with entries holding positions of the initial cell and its input and output gates
B) defect_pattern (locations of defective bits; in our work simulated randomly)
C) Width and height of CMOL array, and parameter a

START:
1: Sort gates in design_list by probability (sorted_design_list)
2:   Take the next cell (cur_gate) from sorted_design_list
3:   If cur_gate is not the end of the list {
4:      If some of the connections from cur_gate to its input and output gates
        with fixed position are defective for defect_pattern {
5:          Create a list (cur_candidate_list) through the following sequence of steps:
                (i) Based on location of inputs and outputs of the cur_gate, create the list
                    of cells ("repair region") where cur_gate could be moved, if no other
                    gates were involved
                (ii) In this list eliminate all the cells occupied by other gates which cannot
                     be swapped with cur_cell
                (iii) Sort the cur_candidate_list by the connection length penalty F
6:          Take the next cell (candidate_cell) from cur_candidate_list
7:          If candidate_cell is not the end of the list {
8:              If connections from candidate_cell to cur_gate's input and output gates
                    with fixed position are defect-free for defect_pattern  {
9:                  Move cur_gate into candidate_cell, exchanging it with the gate
                    (if any) that occupied the cell
10:             } Else { Go to step 8 }
11:         } Else { Exit without success }
12:     } Else {
13:         Fix position for cur_cell
14:         Update sorted_design_list
15:         Go to step 2 }
16: } Else { Exit with success }
```

Figure 4.13: Pseudo-code of the detail routing algorithm used for CMOL FPGA reconfiguration around bad stuck-on-open nanodevices.

has not yet been processed (fixed) by the algorithm, e.g. gate B (Fig. 4.14c), then a similar region is calculated for that gate as well. (For example, in Fig. 4.14c the repair region for gate B is the intersection of the cell connectivity domains of gates 2, 3, and 4.) If the original gate lies in that new repair region, then these two gates can be swapped keeping the circuit functional (provided that all the original gate's connections to the fixed gates are good).

If there are several cells in the initial gate's repair domain (i. e. several positions this gate may be moved to), higher priority is assigned to positions providing smaller interconnect length. More exactly, for each position we calculate the penalty function

$$F = \sum_i [(\Delta x_i)^2 + (\Delta y_i)^2]^f. \tag{4.7}$$

where x and y are the horizontal and vertical coordinates of each cell, and f is an empirically selected exponent. (We have got the best results for $f = 2$.) The summation in Eq. (4.7) is over all potential interconnects (excluding those between original gate

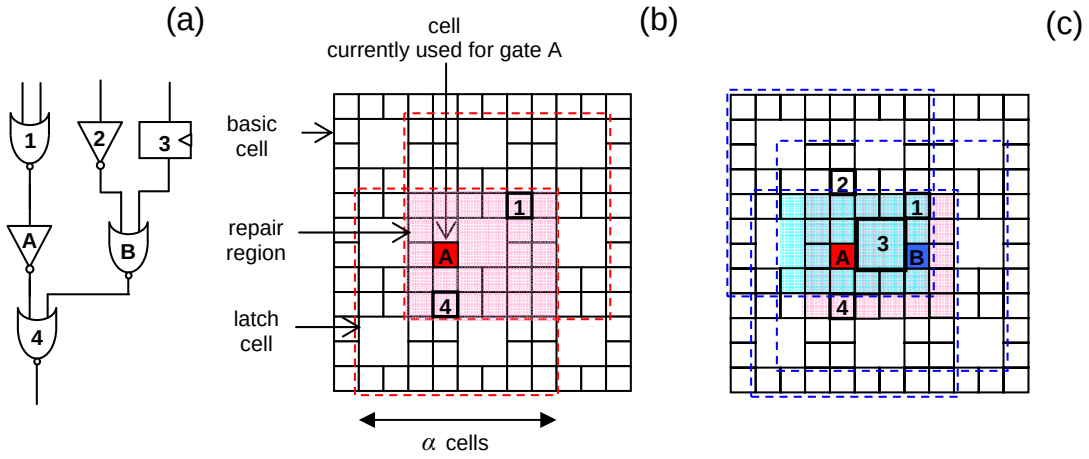

Figure 4.14: Example of a circuit fragment reconfiguration: (a) Circuit whose gate A is to be relocated, because at least one of its connections (with its either input gate 1 or output gate 4) is faulty. (b) The "repair region" of gate A (painted pink) is the intersect of the cell connectivity domains (shown by dashed lines) of its input and output gate cells. (c) If a cell of the "repair region" of A already houses another gate B, the repair domain of the latter cell (painted light blue) is also calculated. Since in this case A is within the repair domain of B, these gates may be swapped, connection quality permitting. For clarity, in this figure $a = 8$; optimal values of a are typically larger (see below). Also note that since for a latch cell any of the input or output pins can be used, the linear size of cell connectivity domain is larger by one cell.

and gates with fixed positions); if the move requires a cell swap, interconnections of both cells are counted. For example, in Fig. 4.14c five connections (from gate A to 1 and 4, and from gate B to 2, 3, and 4) give contributions to this sum, providing that all of these gates have not been processed by the algorithm.[5] (Typically, though not always, this rule gives higher priority to gate moving into an initially empty cell.)

After the list of all possible moving options has been compiled, they are checked, in the order of increasing penalty F, for defective interconnects. The first met option with all good connections is implemented. The case when there are no possible moving options with good connections is considered a reconfiguration failure.

The order in which the gates are processed is determined by the probability of finding a good position. Crudely, the reason for such trick is that repair domains get smaller as the algorithm proceeds due to more gates been fixed. Because of that the processing of gates with a smaller repair domains earlier in the algorithm will create more chances for such gates to find defect free location.

[5]Even though the position of lath cell (gate 3) is fixed by default, the gate B will be moved later in the algorithm and it is important to include the cost of B to gate 3 connection too.

More specifically, at first all the gates are sorted (with the gate having smallest probability appearing first in the list) according to the specific probability P_{gate}, which for a gate i located in basic cell with coordinates (x, y) (denoted as $i_{x,y}$) can be found as:

$$P_{\text{gate}}(i_{x,y}) = 1 - \prod_{\zeta,\xi \in \Omega(x,y)} Q(i_{\zeta,\xi}, x, y). \tag{4.8}$$

Here Ω is a set of cells (with the horizontal and vertical coordinates denoted by ζ and ξ, correspondingly) of the "repair region" for gate i calculated exactly as described above, while $Q = 1 - P$ is probability of failure of moving gate i to the cell at (ζ,ξ). The latter can be found as

$$Q(i_{\zeta,\xi}, x, y) = \begin{cases} Q_{\text{MOVE}}(i_{\zeta,\xi}), & \text{if cell } (\zeta,\xi) \text{ is empty or } x = \zeta, y = \xi, \\ Q_{\text{SWAP}}(i_{\zeta,\xi}, j_{x,y}), & \text{if cell } (\zeta,\xi) \text{ is occuppied by gate } j, \end{cases} \tag{4.9}$$

where $Q_{\text{MOVE}}(i_{\zeta,\xi})$ reflects the probability probability of finding all (C) connections to input and output gates free of defects for the original gate i at cell position (ζ,ξ). Similarly, $Q_{\text{SWAP}}(i_{\zeta,\xi}, j_{x,y})$ is the similar probability of failure of finding defect free connections for both gate i at cell (ζ,ξ) and the swapped gate j at cell (x,y). More exactly, these probabilities are calculated as:

$$Q_{\text{MOVE}}(i_{\zeta,\xi}) = 1 - P_{\text{MOVE}}(i_{\zeta,\xi}) = 1 - (1-q)^C \times \prod_{t=1}^{N_{\text{latch}}} (1 - q^{Overlap(\zeta,\xi,t)}), \tag{4.10}$$

$$Q_{\text{SWAP}}(i_{\zeta,\xi}, j_{x,y}) = 1 - P_{\text{MOVE}}(i_{\zeta,\xi}) P_{\text{MOVE}}(j_{x,y}). \tag{4.11}$$

The essence of Eq. 4.10 is that, if gate i is connected to a latch, we assume that at least one connection (out of four at most) should be defect free. (N_{latch} is the total number of latches the considered gate is connected to.) The *Overlap* function returns the integer (from 1 to 4) corresponding to how many inputs or outputs of a particular latch can be reached (i.e., in a cell connectivity domain) from a given cell.

After the gate is processed and its defect-free location is found, it is marked as fixed and the specific probability of the affected gates (including those which could be potentially swapped with the given gate and also it's input and output gates) is accordingly adjusted.

Adjusting the probability of (at most M) affected gates and placing them to the corresponding position in the sorted list would require to perform $M \log(N)$ (where N is the number of gates in the circuit) compare operations in the worst case. This means that the total complexity of the algorithm is of the order of $NM \log(N)$. For example, for a billion-gate circuit and $M \sim 1600$ this means about $\sim 10^{14}$ operations or about 30-hour runtime for a 1 GHz processor, which might not be practical. To reduce the complexity, instead of classical sorting of the list, in which for any two gates i and j

(with position in the list $Index_i$ and $Index_j$, correspondingly), $Index_i < Index_j$ only if $P_{\text{gate}}(i) < P_{\text{gate}}(j)$, we implement quasi-sorting for which $Index_i < Index_j$ only if there exists a value of $p \in P_d$ such that $P_{\text{gate}}(i) < p < P_{\text{gate}}(j)$, where P_d is a finite set of real numbers between 0 and 1. (For example, for our simulations we used a set of $P_d = [0, 0.01, 0.02, ..., 0.99, 1]$.) Using the quasi-sorting with such P_d instead of the classical one virtually does not affect the quality of the detail routing. On the other hand, it allows to reduce the complexity of detailed routing algorithm to just NM substantially, dropping the runtime to about an 1 hour for the example discussed above. (The runtime for the circuits considered below is not a problem since the largest simulated circuits had less than 3×10^4 gates.)

An approximate analysis of this reconfiguration algorithm shows that most reconfiguration failures come from the longest initial connections, corresponding to the very periphery of the cell connectivity domains. This is why from the point of view of defect tolerance it is beneficial to carry out the initial design for artificially confined connectivity domains. Such confined placement may be already provided by discreteness of the tile connectivity domain $\mathbb{A}$ for a certain parameters of a. For example, for the cases considered in this work, the linear size of the cell connectivity domain $a = 40$ results in $\mathbb{A} = 9$, according to the Eq. (4.3). This means that during placement and global routing the cells connectivity domain with the average effective value of $a' = 36$ was used. However, the worst case for this value could be still $a = 40$, because initial greedy mapping of cells inside each tile is not optimal and there is some non-trivial probability that two connected cells can be placed initially in the opposite corners of corresponding tiles.

4.3 Performance Calculation

4.3.1 Area

We will show below that the typical current through molecular devices in the ON state is of the order of 1 μA. With a saturation current density of 1 mA/μm, typical for the long term CMOS projections [1], such current may be provided with a MOSFET channel as narrow as 1 nm. Hence, we can safely assume that all four transistors of the basic cell are of the minimum width. (Note, that even the minimum-width CMOS inverter in the cell can provide a very large (> 20) fan-out without any latency degradation.) Using SCMOS design rules [82], we estimate the smallest basic cell area to be about $A_{\text{cell}} = 64(F_{\text{CMOS}})^2$, i.e., $\beta_{min} \approx 4$. The additional area overhead associated with auxiliary circuitry such as clock buffers, peripheral logic for reconfiguration, etc. has not been taken into account at this stage, but is probably negligible[6].

[6]For example, using results from CMOL memories (Chapter 3), the overhead associated with peripheral logic for reconfiguration for 50×50 CMOL FPGA array of tiles should be less than 20%,

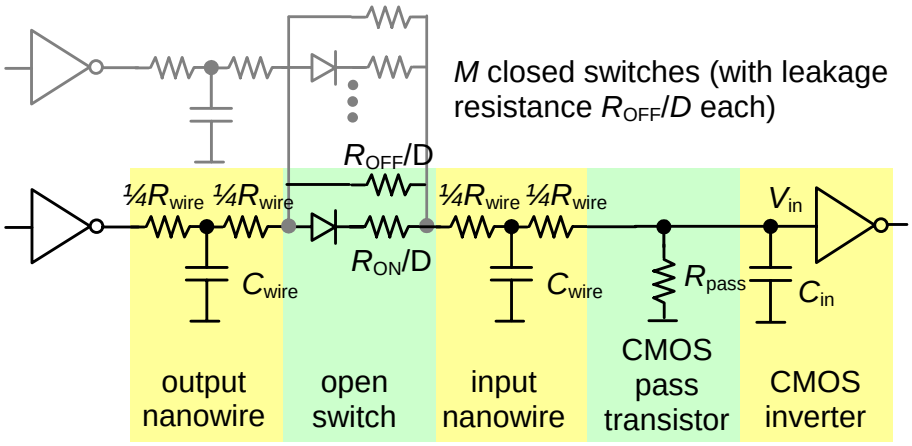

Figure 4.15: The equivalent circuit of a CMOL logic stage.

4.3.2 Delay

In order to speed up the CMOL FPGA circuit, it is beneficial to reduce the signal swing of CMOS inverter's input voltage $V_{\rm in}$ by decreasing the effective parallel resistance $R_{\rm par}$ defined as (Fig. 4.15)

$$\frac{1}{R_{\rm par}} \equiv \frac{1}{R_{\rm pass}} + \frac{M}{R_{\rm OFF}/D}. \tag{4.12}$$

Just like in the memory circuits (Section 3.4.2) the limit to this reduction is set up by the requirement for the swing $V_{\rm in}$ to be larger than the possible total noise swing at the inverter input. The most important components of the noise are the thermal fluctuations, shot noise produced by the nanodevices, and digital noise of other gates.

At $M \gg 1$, the thermal and shot noises are typically Gaussian, with the r.m.s. value

$$V_{\rm T} = [k_{\rm B}(T + T_{\rm ef})/C_{\rm wire}]^{1/2}, \tag{4.13}$$

where $T_{\rm ef}$ is the effective temperature contributed by the shot noise. Since such $V_{\rm in}$ is much less than $V_{\rm DD}$ (see below) the effective temperature can be found as

$$T_{\rm ef} = \frac{eV_{\rm in}}{2k_{\rm B}} \coth \frac{eV_{\rm DD}}{2k_{\rm B}T}. \tag{4.14}$$

Also note, that while $C_{\rm wire}$ corresponds always corresponds to the full nanowire segment length $L = 2\beta^2 F_{\rm CMOS}^2/F_{\rm nano}$, only a part of the fragment (from the crosspoint

while the shift-register reconfiguration circuitry which is discussed in Ref. 109 (see also Fig. 5.2b), will make this overhead even much smaller.

nanodevice to the interface pin) contributes to its resistance $R_{\rm wire}$.[7] In order to keep our estimates on the conservative side, we have assumed the worst configuration case when the length of this part is largest ($L/2$).

With the very strict requirement for the bit error rate to be below $q_{\rm gate} = 10^{-28}$ (corresponding, for example, to a mean time between failures of at least 10,000 hours [1] for a CMOL FPGA chip with as many as 10^{10} gates operating with a 0.1-ns clock cycle), the maximum swing $\Delta V_{\rm T}$ of this noise, calculated from the equation Eq. (3.21), is close to $12V_{\rm T}$.

The digital noise is created mostly by coupling of output signals of M other gates (with swing equal to $V_{\rm DD}$ each) through the M parallel resistances of latching switches turned OFF (Fig. 4.15). Though for $M \gg 1$ the statistics of this noise is usually also close to Gaussian, one cannot exclude the possibility of strong correlation of signals processed by neighboring gates. To play it safe, we have assumed the worst case scenario when all digital noise sources are fully correlated, resulting in the maximum swing $MV_{\rm DD}/(R_{\rm OFF}/D)$ of the current flowing to the inverter input.

Summing these two noise contributions, we get the following condition on $V_{\rm in}$:

$$V_{\rm in} > \Delta V_{\rm T} + MV_{\rm DD}\frac{R_{\rm pass}}{R_{\rm OFF}/D}, \tag{4.15}$$

where the simplification is due to the fact that for all considered cases $R_{\rm pass} \ll R_{\rm OFF}/(DM)$, i.e. $R_{\rm par} \approx R_{\rm pass}$, and $V_{\rm DD} \ll V_{\rm in}$.

Indeed, with the parameters considered below, this condition allows to reduce $V_{\rm in}$ well below 100 mV, i.e. make it much lower than $V_{\rm DD}$, which is in the range 0.2-0.3 V. This means that the CMOL FPGA circuit speed is limited by the relatively slow recharging of a few-fF "input" (post-latch) nanowire capacitance $C_{\rm wire}$ shunted by a relatively low parallel resistance $R_{\rm pass}$ given by Eq. (4.12), through a much higher series resistance $R_{\rm ser} \sim R_{\rm ON}/D + 2R_{\rm wire}$. This is why the full equivalent circuit of one logic stage (Fig. 4.15) yields the elementary formula for the signal delay per logic stage:[8]

$$\tau_0 \approx \log(2I)R_{\rm pass}C_{\rm wire}, \tag{4.16}$$

where I is the gate fan-in, while the necessary value of $R_{\rm pass}$ may be calculated as

$$R_{\rm pass} = V_{\rm in}/DI_{\rm ON}. \tag{4.17}$$

[7] Note, that because the performance of CMOL FPGA circuits is limited by the static power consumption (see Section 4.3.3 below), it is possible to use gradual sloping for the clock signals; hence Miller effect [91] can be neglected.

[8] The output dynamic impedance of the CMOL inverter and its input capacitance $C_{\rm in}$ give negligible contribution to τ_0. For example, $C_{\rm in}$ of the 22 nm minimum-width inverter is of the order of 0.02 fF, i.e., much less than $C_{\rm wire}$.

The ON current of the nanodevice should be generally calculated from the $I-V$ curve (Fig. 1.1a), with D parallel nanodevices connected in series with the Ohmic resistance $R_{\rm wire}$, driven by voltage $V_{\rm DD}$. However, since the only lower bound on the suppressed Coulomb blockade threshold $V_{\rm t}$ is to be larger than $V_{\rm in}$ (in order to prevent current leakage through ON-state nanodevices fed by 0-level output of CMOS inverters), $V_{\rm t}$ may be substantially less that $V_{\rm DD}$. Hence, we may consider the nanodevice $I-V$ curve linear, and find $I_{\rm ON}$ as

$$DI_{\rm ON} \approx \frac{V_{\rm DD}}{R_{\rm ser}} = \frac{V_{\rm DD}}{R_{\rm ON}/D + 2R_{\rm wire}}. \tag{4.18}$$

4.3.3 Power

The average total power consumption of a CMOL gate may be estimated as a sum of the static power $P_{\rm ON}$ due to currents $I_{\rm ON}$, static power $P_{\rm leak}$ due to current leakage through nanodevices in their OFF state, and dynamic power $P_{\rm dyn}$ due to recharging of nanowire capacitances.[9] The above estimate $V_{\rm in} \ll V_{\rm DD}$ allows to calculate these contributions using simple formulas

$$P_{\rm ON} \approx \frac{N_{\rm dev}V_{\rm DD}^2}{2R_{\rm ser}}, \tag{4.19}$$

$$P_{\rm leak} = \frac{MV_{\rm DD}^2}{2R_{\rm OFF}/D}, \tag{4.20}$$

$$P_{\rm dyn} = \frac{C_{\rm wire}V_{\rm DD}^2}{4\tau}, \tag{4.21}$$

where τ is critical path circuit delay, $N_{\rm dev}$ is an average number of nanodevices in the ON state per cell. The factors 1/2 reflect the natural assumption that on average there is an equal number of CMOS inverters with Boolean 1 and 0; the dynamic power has an additional factor 1/2 describing the energy loss at capacitance recharging.

4.3.4 Optimization

In general, the best performance results of CMOL FPGA circuits can be achieved via optimization of pitch ratio $F_{\rm CMOS}/F_{\rm nano}$ (assuming that either $F_{\rm CMOS}$ or $F_{\rm nano}$ is fixed) and supply voltage $V_{\rm DD}$. Indeed, making CMOS-to-nano pitch ratio larger increases the nanowire segment capacitance and hence the delay, see, e.g., Eq. (4.16). On the other hand, making such ratio smaller increases the number of routing inverters

[9] It is easy to show that the power consumption due to the leakage current through CMOS inverters is negligible.

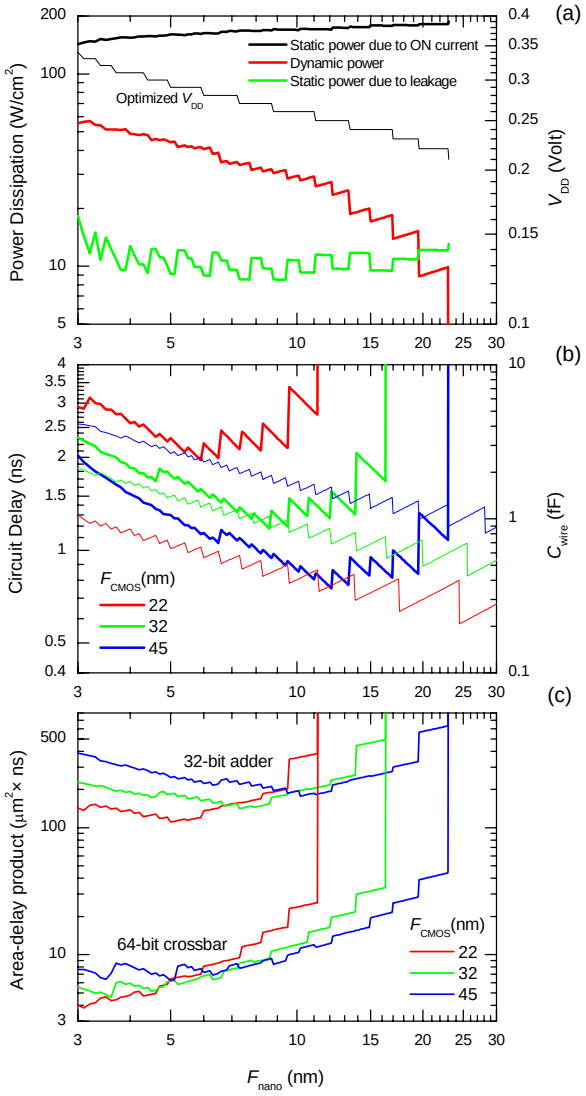

Figure 4.16: The example CMOL FPGA optimization results for two simple circuits studied earlier in Ref. 109 (see also Appendix B): (a) three components of the total power (fixed at 200 W/cm^2), and the optimum value of the power supply voltage $V_{\rm DD}$, for the 32-bit adder with $F_{\rm CMOS} = 45$ nm; (b) nanowire segment capacitance (thin lines) and the total logic delay of the circuit (bold lines); and (c) area-delay product $A\tau$ of the two CMOL FPGA circuits under analysis, for three ITRS long-term CMOS technology nodes. The (formal) jump of the $A\tau$ product to infinity at some $(F_{\rm nano})_{\rm max}$ reflects the fact that our procedure of initial circuit mapping may only be implemented for $F_{\rm nano}$ below this value - see Fig. 9 from Ref. 109 and its discussion. The finite sharp jumps of the curves are due to the transfers between adjacent integer values of a that would satisfy Eq. (2.1) and provide the smallest $\beta > \beta_{\rm min} = 4$.

required to map circuits successfully. The latter effect results in increase in logic depth of the circuit and hence the logic delay.

The optimization of supply voltage V_{DD} is more complicated. With the assumption that the total power $P = P_{ON} + P_{leak} + P_{dyn}$ per unit area fixed at the level $P/A = 200$ W/cm^2 planned by the ITRS [1] for the next decade, the reduction of V_{DD} makes possible to choose smaller value of R_{ON} and hence reduce the circuit delay. However, reduction of R_{ON} also results in corresponding scaling down of R_{OFF} (Eq. 2.7). The latter cannot be too small, mainly due to increase of the digital noise (Eq. 4.15).

Earlier we have performed an optimization of both pitch ratio and V_{DD} for a two simple circuits - 32-bit Kogge Stone adder and 64-bit full crossbar (Fig. 4.16) mapped on one-cell CMOL FPGA fabric [109] (see also Appendix B). In order to do this, for each pair of F_{CMOS} and F_{nano} (and hence for parameters, a, a', L, C_{wire}, and R_{wire} calculated as described above), we had varied V_{DD}, each time adjusting the ratio R_{ON}/D (and hence the product DI_{ON} calculated from Eq. (4.18)) so that the total power calculated from Eqs. (4.21, 4.20, 4.19) equaled the specified level. At this procedure, R_{OFF} was adjusted from elastic co-tunneling effect using Eq. (2.7).

4.3.5 Simplification

The results from Fig. 4.16 show that CMOL parameters $F_{CMOS} = 45$ nm, $F_{nano} = 4.5$ nm, which seems technologically plausible at the initial stage of CMOL technology development [67], and $V_{DD} = 0.3$ are in the ballpark of optimization. With these parameters fixed the performance model of CMOL FPGA ciruits can be greatly simplified.

Indeed, even taking into account the additional diffusive scattering at nanowire surface (Sec. 2.3.2), the estimated resistance between the center and the end of a nanowire fragment, of the length $(\beta F_{CMOS})^2/F_{nano} = 7.2$ μm, is about 20 KΩ. Such resistance is negligible, because it is connected in series with that of a crosspoint device (Fig. 4.15), which is an order of magnitude larger, even in the ON state - see below. [10] With the wire capacitance per unit length to be close to 0.2 fF/μm (Sec. 2.3.2), capacitance C_{wire} of the full nanowire fragment is about 3 fF.

Moreover, Fig. 4.16 shows that the most important component of the power consumption is due to the static power through nanodevices in the ON state, i.e. given by Eq. (4.19). Therefore, the smallest acceptable resistance R_{ON} can be found

[10] The considered resistances of the nanodevices in the CMOL FPGA is much higher as compared to those of CMOL memory, due to much larger number of nanodevices set to the ON state.

from[11]

$$R_{\rm ON} = \frac{DN_{\rm dev}(V_{\rm DD})^2}{2A_{\rm cell}p_{\rm max}}. \tag{4.22}$$

Due to relatively large value of $R_{\rm ON}$ we can also the digital noise can be also neglected in Eq. (4.15), so that for the considered parameters $T_{\rm ef} \approx 250K$ and the voltage swing on the input of the CMOS inverter is

$$V_{\rm in} \geq \Delta V_{\rm T} = 12\sqrt{k_{\rm B}(T + T_{ef})/C_{\rm wire}} \approx 20 \text{ mV}. \tag{4.23}$$

Finally, in the considered case we could use the following simplified formula for gate delay:

$$\tau \approx \ln(2I) \times (C_{\rm wire}R_{\rm ON}/D) \times (V_{\rm in}/V_{\rm DD}). \tag{4.24}$$

4.4 Results

We have applied our methods to analyze possible CMOL FPGA implementation of the Toronto 20 benchmark circuit set [2]. Using the completely custom design automation flow described in Sec. 4.2 we have first mapped the circuits on the two-cell CMOL FPGA fabric. For example, Fig. 4.17a shows the initial (random) placement of circuit, while Fig. 4.17a the final placement for dsip.blif circuit mapped on the (17+2)×(17+2) tile array with no defects. (Here the additional layer of tiles at the array periphery is used exclusively for I/O functions. The cells from these peripheral tiles are functionally similar to input and output pads and cannot be configured to NOR gates.) The global routing step, i.e. allocation of routing inverters for the gates which are not within cell connectivity domain of each other, is shown on Fig. 4.18. (As a reminder, in CMOL FPGA hardware each of the straight lines shown on Fig. 4.18b actually consists of two mutually perpendicular nanowires, connected with nanodevices.)

The largest value of the average nanodevice utilization factors among all circuits of the set has turned out to be about 1.6 nanodevices turned ON per basic cell (Table 4.1). Plugging $N_{\rm dev}$ into Eq. (4.22), we find that $R_{\rm ON} = 22$ MΩ[12] and the ON resistance of a crosspoint nanodevice is $R_{\rm ON}/D = 280$ KΩ (using $D \approx 80$ obtained with Eq. (2.6)). These values justify the simplifications described in the previous sections.

[11] It is easy to check that for the considered parameters, in particular $V_{\rm DD} = 0.3$V, the ratio of resistances in the OFF and ON states provided by the second-order quantum effect of elastic cotunneling given by Eq. (2.7) is always less than the maximum value of this ratio, which is limited by the classical thermal activation given by Eq. (2.8).

[12] Such resistances, well above $R_{\rm Q}$, may be readily implemented in molecular electronics - see, e.g., Ref. 121.

Circuit	CMOS FPGA F_{CMOS} = 45 nm					CMOL FPGA F_{CMOS} = 45 nm, F_{nano} = 4.5 nm, max fanin = 7						Comparison		
	Depth	LUTs	Linear size (tiles)	Area (μm^2)	Delay (ns)	Depth	Linear size (tiles)	K	Nano-devices	Area (μm^2)	Delay (ns)	A_{CMOS}/A_{CMOL}	$A_{nanoPLA}/A_{CMOL}$	A_{FPNI}/A_{CMOL}
alu4	7	1274	19 × 19	137700	5.1	23	19 × 19	7	5468	749	1.7	184	0.37	6.71
apex2	8	1602	21 × 21	166050	6.0	26	20 × 20	7	6241	830	2.1	200	3.41	6.56
apex4	6	1147	34 × 34	414619	5.5	19	16 × 16	7	4203	531	1.5	781	0.73	7.27
bigkey	3	1810	22 × 22	193388	3.1	20	18 × 18	9	6589	672	0.9	288	2.25	-
clma	16	6779	42 × 42	623194	13.1	78	55 × 55	4	33772	6272	4.2	99	1.74	3.56
des	6	1263	19 × 19	148331	4.2	28	22 × 22	7	6955	1004	1.8	148	3.51	-
diffeq	14	987	16 × 16	100238	6.0	73	20 × 20	9	6279	830	3.7	121	3.27	6.56
dsip	3	1362	19 ×19	148331	3.2	26	17 × 17	9	5392	600	1.1	247	2.25	-
elliptic	18	2142	24 × 24	213638	8.6	81	34 × 34	7	16403	2399	4.9	89	3.12	5.20
ex1010	8	4050	33 × 33	391331	9.0	43	29 × 29	6	13540	1745	2.0	224	0.56	6.78
ex5p	7	950	16 × 16	100238	5.1	27	16 × 16	6	3551	531	1.7	189	0.30	5.97
frisc	23	2320	25 × 25	230850	11.3	114	35 × 35	6	16393	2542	6.8	91	4.36	4.91
misex3	7	1178	18 × 18	124538	5.3	24	17 × 17	7	4798	600	1.3	208	0.93	7.06
pdc	9	3901	32 × 32	369056	9.6	54	41 × 41	4	21804	3488	2.7	106	0.22	3.96
s298	15	1682	21 × 21	166050	10.7	45	15 × 15	7	5891	467	3.5	356	2.36	12.61
s38417	11	4773	36 × 36	462713	7.3	52	55 × 55	4	32430	6277	3.0	74	1.84	3.85
s38584	9	4422	35 × 35	438413	4.8	64	45 × 45	5	27360	4202	3.0	104	-	4.53
seq	7	1427	20 × 20	151369	5.4	23	21 × 21	6	6003	915	1.7	165	1.63	5.95
spla	8	3331	30 × 30	326025	7.3	40	38 × 38	4	18562	2996	2.7	109	0.12	4.17
tseng	13	781	14 × 14	78469	6.3	75	20 × 20	8	5610	830	4.4	95	3.57	6.06

Table 4.1: Performance results for Toronto 20 benchmark set mapped on two-cell CMOL fabric with no defects.

According to Eq. (4.24), the delay of a 1-input NOR gate turns out to be about 40 ps.[13] The full delay of the considered circuits was calculated from the critical path, which had been found after circuit placement and global routing.

Table 4.1 summarizes the performance results for the benchmark circuits mapped on CMOL FPGA without any defects. Note that in contrast with earlier nanoelectronics work, the results for different circuits are obtained for the CMOL FPGA fabric with exactly the same operating conditions and physical structure for all the circuits, thus enabling a fair comparison with CMOS FPGA. For this comparison, the same benchmark circuits have been synthesized into cluster-based island-type logic block architecture [10]. This was done with T-VPack and VPR tools using the architecture designed for the optimal area-delay product, specifically the cluster size of 4, 4-input LUTs [5], and the VPR's default architecture file (4x4lut-sanitized.arch) with technology parameters corresponding to the 0.3 μm CMOS process. We had first found

[13]Due to correction in the Eq. (3.21) the delay is almost twice smaller than that assumed in Refs. 109, 111, 114.

Circuit	No defects		10% defective cells		30% defective cells	
	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)
alu4	749	1.7	915	1.6	1745	2.0
apex2	830	2.1	1004	2.0	1297	2.1
apex4	531	1.5	600	1.5	915	1.5
bigkey	672	0.9	749	0.9	1098	1.0
des	1004	1.8	1098	1.8	1403	1.8
diffeq	830	3.7	830	3.7	1195	3.7
dsip	600	1.1	672	1.1	915	1.1
elliptic	2399	4.9	3488	5.0	11362	5.8
ex1010	1745	2.0	1994	2.0	2996	2.2
ex5p	531	1.7	600	1.7	749	1.7
frisc	2542	6.8	2841	6.8	5187	7.1
misex3	600	1.3	749	1.4	1004	1.4
pdc	3488	2.7	4982	3.0	9594	3.9
s298	467	3.5	531	3.6	749	3.7
s38417	6277	3.0	6980	3.2	9038	3.3
s38584	4202	3.0	4781	3.0	6050	3.0
seq	915	1.7	1004	1.7	1513	1.8
spla	2996	2.7	4390	3.0	8499	4.0
tseng	830	4.4	830	4.4	1098	4.4

Table 4.2: The performance results of the Toronto 20 benchmark set mapped on two-cell CMOL fabric (with $F_{\rm CMOS} = 45$nm, $F_{\rm nano} = 4.5$nm, and $I_{\max} = 7$) in a presence of defective cells.

the worst case segment width required to route every circuit successfully, which has turned out to be 70 for pdc.blif circuit. Then, using an architecture with such segment width we have mapped and routed all circuits, and then extracted their delay and area (for the optimistic case of buffer sharing). Assuming very optimistic $1/s$ scaling for the delay and assuming the area of the minimum-width transistor to be $25(F_{\rm CMOS})^2$, we have obtained the results shown in the left part of Table 4.1. (As a sanity check, the delays before scaling are close to those obtained in Ref. 10 for CMOS FPGA with a similar architecture.)

In general, presence of defective cells is logically equivalent (for $M \ggg 1$) to the reduction of connectivity domain size, since any cell can now be connected directly with a smaller number of "good" cells on average. The decrease of connectivity domain, in turns, results in the increase of the circuit's area and potentially critical path delay due to larger number of routing inverters for longer global wires. Our earlier results for 32-bit Kogge-Stone adder (see Fig. 9 of Ref. 109) indicate that there is certain value of the connectivity domain size below which the depth of the

Circuit	Defect probability, q	
	90%	99%
alu4	0.12	0.07
apex2	0.07	0.05
apex4	0.13	0.1
bigkey	0.15	0.1
clma	0.02	0.01
des	0.06	0.02
diffeq	0.12	0.04
dsip	0.07	0.03
elliptic	0.06	0.05
ex1010	0.06	0.03
ex5p	0.16	0.1
frisc	0.04	0.03
misex3	0.09	0.02
pdc	0.03	0.01
s298	0.16	0.09
s38417	0.02	0.01
s38584	0.14	0.1
seq	0.09	0.05
spla	0.04	0.03
tseng	0.16	0.09

Table 4.3: The maximum allowable defect rate of bad (stuck-on-open) nanodevices to achieve 90% and 99% final yield.

circuit starts to increase exponentially fast until the mapping becomes impossible. This means that as long as the connectivity domain size is not approaching this limit the mapping can be physically implemented. In other words, the reasonable (not very large) number of bad cells should not prohibit from the successful mapping, albeit increasing the area and delay.

Table 4.2 shows results of the performance for the considered circuits in the presence of defective cells, proving our observation. To obtain these results we performed large number of trials (more than 100) for each circuit, each time with a new map of defective cells, finding minimum value of K, and hence minimum area and delay. The defective cells have been generated randomly with a probability q_{cell}. For example, Fig. 4.19 shows successful mapping for dsip.blif circuit on the CMOL FPGA fabric with $q_{\text{cell}} = 0.3$.

In particular, Table 4.2 shows that the average swelling of the circuit area is rather limited: only about 20% and 80% for, respectively, $q_{\text{cell}} = 0.1$ and $q_{\text{cell}} = 0.3$. This means that faulty interface pins, nanowires and/or CMOS circuitry can be very effectively tolerated. On the other hand, the tolerance to stuck-on-close crosspoint defects is rather low (equivalent to about 0.02% of defective nanodevices for $q_{\text{cell}} = 0.3$), so that some other defect tolerance mechanism should be used to reduce the effects of such faults.

Finally, Table 4.3 present our preliminary simulation results for the defect tolerance with respect to stuck-on-open kind of defects (in this simulations we assumed that there is no defective cells). These results were obtain by Monte Carlo simulations of the detail routing algorithm using our group's supercomputer cluster *Njal* (http://njal.physics. sunysb.edu/). For each initially placed and globally routed circuit, the algorithm has been run 1000 times with a randomly chosen set of defects, formed with the same probability q.

The results are much worse than that for 32-bit Kogge-Stone adder we simulated earlier (see Fig. B.6). For example, at realistic parameter $a = 40$ circuits may have a fabrication yield above 99% at the average fraction of bad stuck-on-open nanodevices 5%. On the other hand, similar yield (99%) could be achieved with up to 20% defective nanodevices in CMOL-mapped adder. One reason for such decrease in defect tolerance is that the simulated circuits have much higher fan-in and fan-out. However, most important factor is rather our naive mapping of gates inside the tiles in the first stage of detail routing algorithm. We expect that by moderately limiting fan-in and fan-out, and smarter mapping in detail routing algorithm the defect tolerance can be boosted up to the one achieved in our earlier work, i.e. to about 20% defective nanodevices.

Such high defect tolerance should not be surprising because only a small percent of nanodevices (about 0.1% of the total on average) is utilized. Such huge redundancy can be efficiently exploited by the detail routing algorithm, which allows to pick completely deferent set of nanodevices by moving positions of the gates.

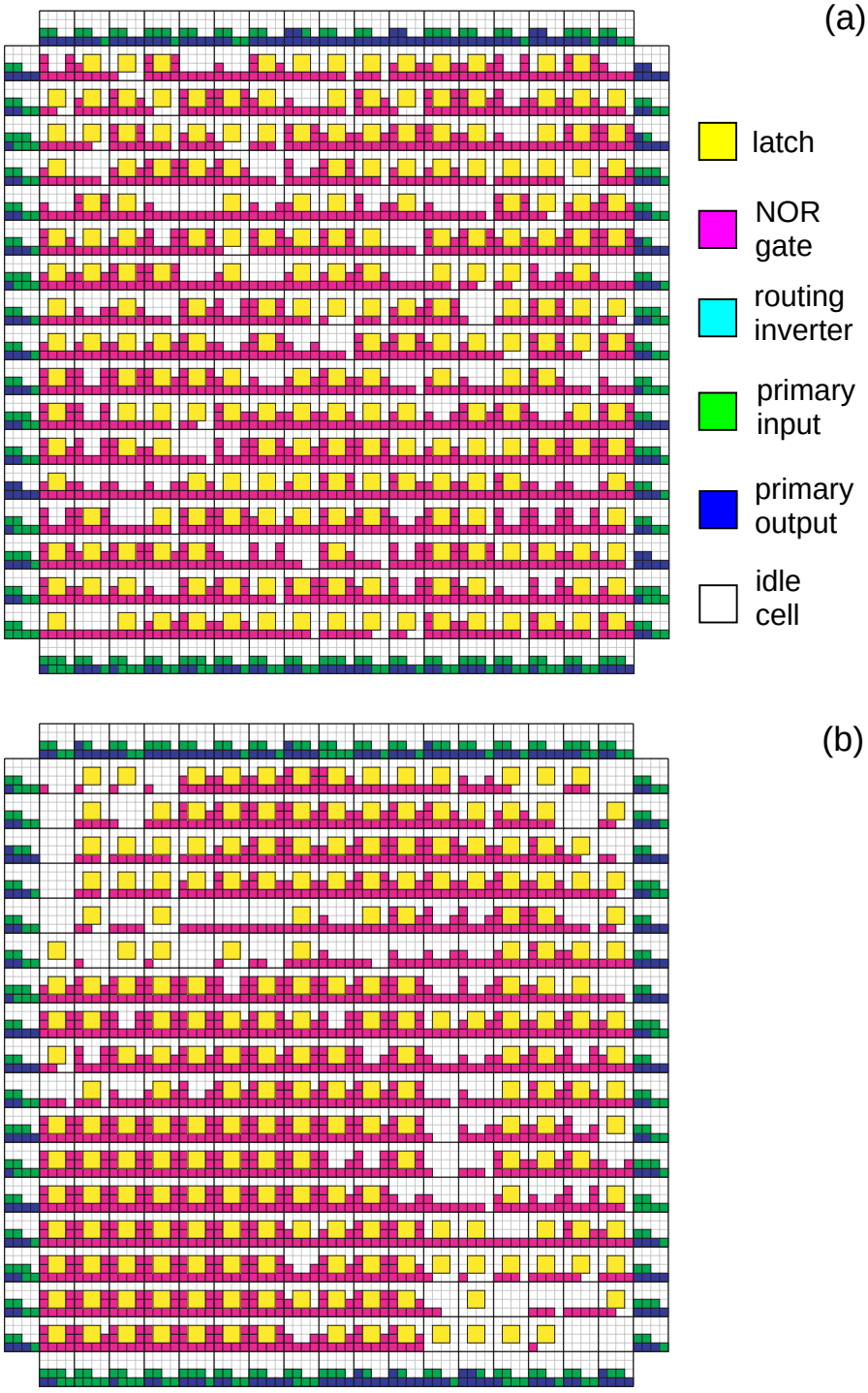

Figure 4.17: Example of dsip.blif circuit mapped on the (17+2)×(17+2) tile array with no defects: (a) initial (random) and (b) final placement. Note that not all of the mapped gates can be connected directly on that stage.

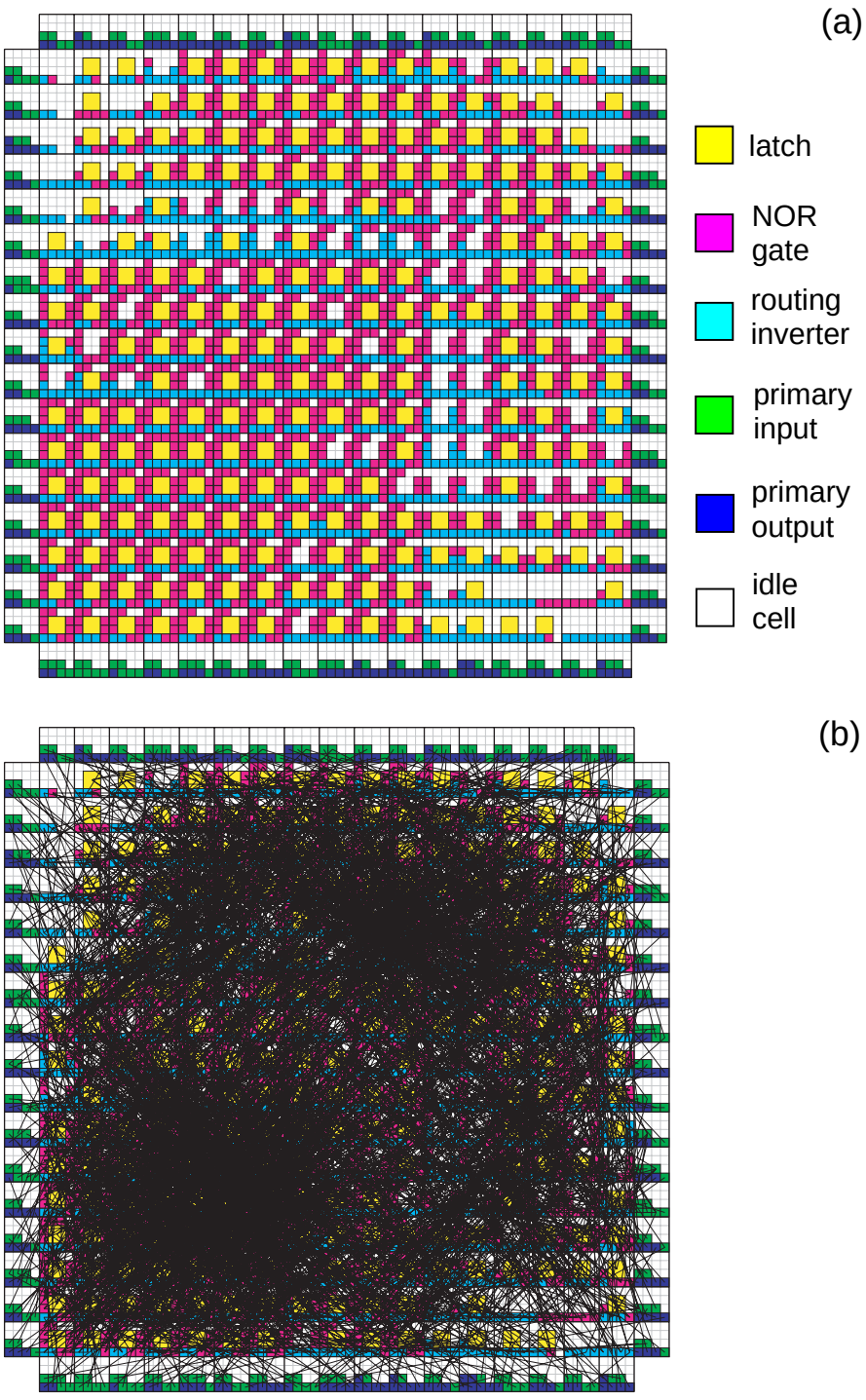

Figure 4.18: Example of dsip.blif circuit global routing mapped on the (17+2)×(17+2) tile array with no defects: (a) without and (b) with gate connections (shown as the straight lines). With no defects any greedy mapping inside the tile (e.g., shown on this figure) can be physically implemented.

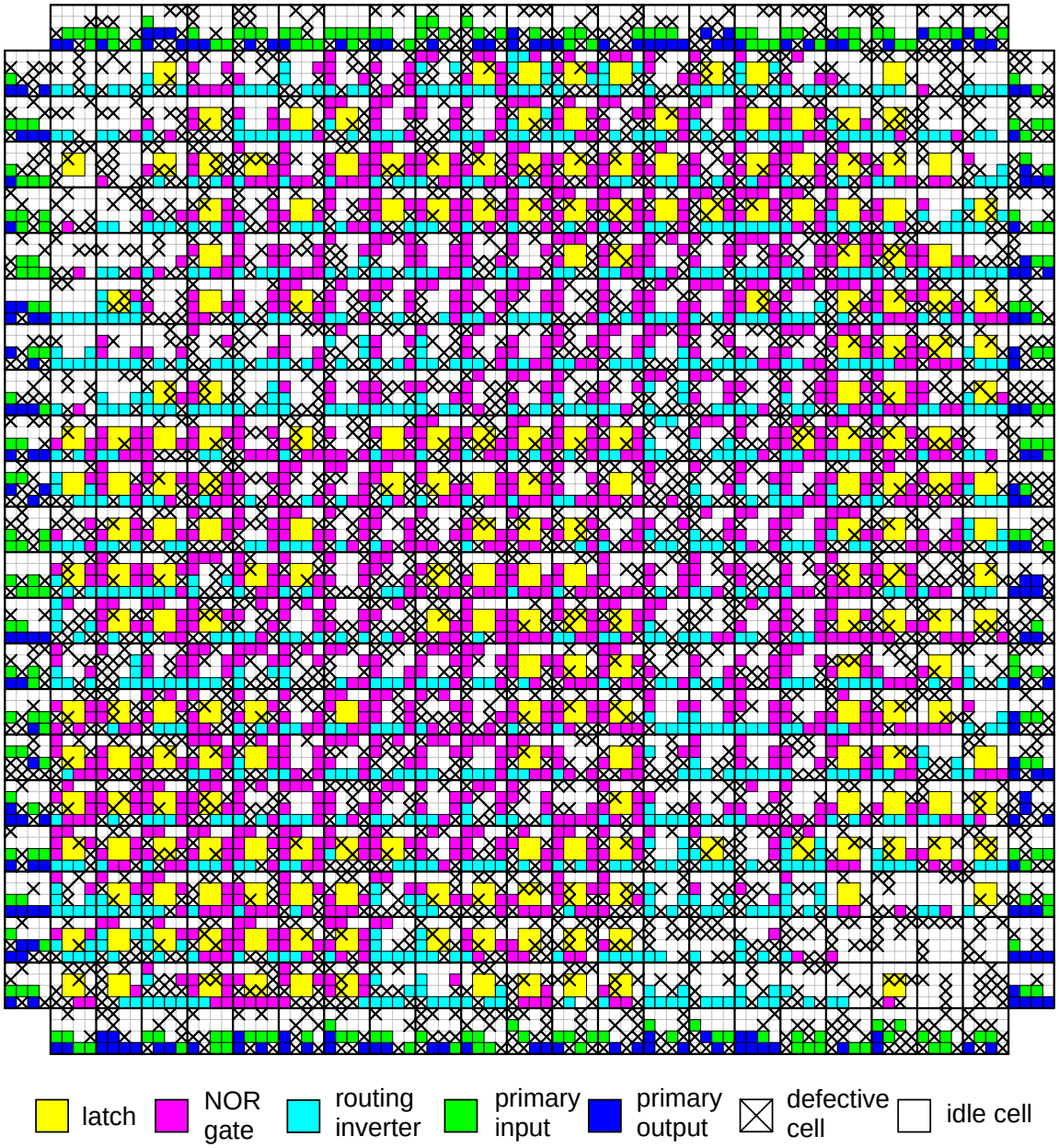

Figure 4.19: Example of CMOL mapping with a presence of defective cells: dsip.blif circuit of the Toronto 20 set, mapped on the (21+2)×(21+2) tile array with 30% defective cells.

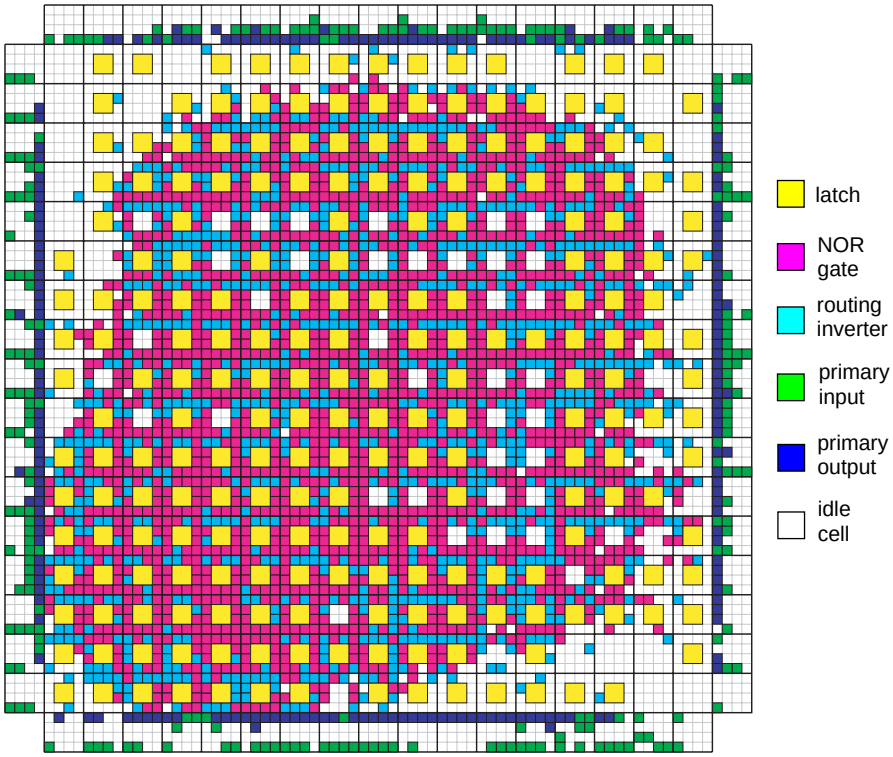

Figure 4.20: Example of CMOL mapping with a presence of defective cells: dsip.blif circuit of the Toronto 20 set, mapped on the (21+2)×(21+2) tile array with 30% defective nanodevices.

Chapter 5

Discussion

5.1 Conclusions

Simulation results presented in this work clearly show that CMOL-based digital circuits may continue the performance scaling of microelectronics well beyond the limits of currently dominating CMOS technology.

5.1.1 CMOL Extension of Resistive Memory

The most natural area of CMOL circuit application is the extension of resistive memories to the sub-$F_{\rm CMOS}$ region. Fig. 3.9a shows that even at the initial stage of the CMOL technology development (when the ratio $F_{\rm CMOS}/F_{\rm nano}$ is of the order of 3), the defect tolerance and density of CMOL memories may be quite impressive. For example, if the required latency is not too small (say, 10 ns or higher), CMOL memories may become denser than CMOS-based memories at the fraction q of bad devices as high as $\sim 15\%$, and at the (quite realistic) value $q = 5\%$ provide a nearly five-fold density edge ($a \cong 1.2$ instead of 6). For $F_{\rm nano} = 15$ nm this would mean useful density $1/a$ of ~ 30 Gbits/cm^2, i.e. the level which CMOS technology may be able to reach in the very end of scaling [1], if ever.

Moreover, as the CMOL technology matures, the $F_{\rm CMOS}/F_{\rm nano}$ ratio may approach an order of magnitude (say, $F_{\rm CMOS} = 32$ nm, $F_{\rm nano} = 3$ nm [67]), and the CMOL memory superiority may be quite spectacular. Indeed, as Fig. 3.9b shows, for the defect fraction $q = 2\%$ (which looks quite plausible), the cell area factor a may become as low as 0.1, implying the dimensional density as high as 1 Tbit/cm^2, far beyond the most optimistic projections for CMOS technology [1].

For the latter design point, the estimated memory throughput is equally impressive (helped by the fact that this pitch ratio optimizes the throughput, since all CMOS data lines are being utilized). Indeed, even assuming a BCH decoder without pipelining, the memory bandwidth can be as high as 30 Tb/s for a 1 cm^2 chip. (For reaching this maximum, the BCH decoder has to be replicated for each column of

CMOL blocks, but even in this case the corresponding area overhead is less than 20% - Fig. A.4).

One reservation has been made concerning our defect-tolerance results for CMOL memories. Indeed, our analysis has been limited to the crosspoint device defects equivalent to the stuck-at-open faults, while in practice defects of other types are also possible (see, e.g. Section 2.4). In this context, it is worth mentioning that both breaks of the lower-layer nanowires and defective pins leading to these nanowires can be very efficiently excluded by reconfiguration around them, using the same techniques employed for coping with stuck-on-open-type defects. The exclusion of defects in the upper layer nanowires requires some modifications of the hardware. For example, by increasing the size of array it is possible to access nanodevices through several pins contacting the same nanowire simultaneously. (This would require a somewhat more complicated address control circuitry, but the area of these circuits would still give a negligible contribution to the total area.)

In contrast, the worst-case stuck-on-close-type defects (resulting in a very low crosspoint resistance) can be overcome by exclusion of both nanowires leading to the defective crosspoint. Preliminary estimates show that such exclusion incurs high area overhead, so that our current architecture is not very efficient for dealing with these defects. On the other hand, if a stuck-on-close defect has a resistance comparable with the nominal value of $R_{\rm ON}$, addressing of several adjacent nanodevices may still be possible. An analysis of this problem, which is similar to finding acceptable margins for variations of $R_{\rm OFF}$, $R_{\rm ON}$, and $R_{\rm wire}$, is one of our future goals.

Let us repeat that though our calculations have been carried out for CMOL memories with their segmented nanowire structure (Fig. 3.2), we have found the contribution from the nanowire recharging time to the total access time negligible.[1] This is why our final results are actually valid also for crosspoint memories with global blocks and peripheral nano/CMOS interfaces, though they seem much harder for the practical implementation than the area-distributed CMOL interface.

5.1.2 CMOL Logic Circuits

Even more spectacular performance estimates are obtained for reconfigurable, FPGA-like CMOL circuits. Table 4.1 shows very clearly that CMOL FPGA circuits may be much denser than the purely CMOS FPGA ones fabricated with the same CMOS design rules. The benchmark circuit area for CMOL FPGA also favorably compares with that implemented using the nanoPLA concept [27], taking into account the fact that the latter results had been calculated assuming a smaller nanowire half-pitch

[1] This would not be true for the hypothetical (and hardly plausible) semiconductor-wire or molecular-wire crossbars in which high nanowire resistance may seriously affect the memory access time.

$F_{\rm nano} = 2.5$ nm.[2] Concerning the speed performance, the delays calculated in this work for all benchmark circuits are slightly better than those of their CMOS FPGA counterparts.

It is safe to expect that the improvement in area will be even larger if CMOL FPGA is used for much larger circuits, because the area of CMOS FPGA is always determined by the worst-case routing requirement. On the other hand, a distinctive feature of the CMOL FPGA fabric suggested in this work is that the same resources, basic cells, are used to perform both logic and interconnect functions.[3] Using the proposed CAD flow, the resources can be allocated flexibly according to the specific logic-to-routing ratio of the circuit. For example, in order to synthesize the relatively large pdc.blif circuit, only about 30% of the cells have been allocated for logic operation, while this number is about 75% for the smaller dsip.blif (Table 4.1).

Another evident resource for improvement the results in Table 4.1 is the optimization of $F_{\rm nano}$ and $V_{\rm DD}$, just like described in Section 4.3.4. Next, an optimization of the maximum fan-in for each circuit may also give substantial results. For example, the area of the s298.blif circuit would be one half its current size, and its pre-mapped depth by 30% lower, if the maximum fan-in was 16 (rather than the maximum value of 7 which we used for this work).

The defect tolerance of CMOL FPGA circuits is even more impressive than that of CMOL memories. First of all, Table 4.2 shows that the performance of CMOL FPGA is only slightly affected if the number of defective cells is not large. For example, the presence of 5% defective nanowires and 5% defective interface pins (corresponding to 10% of defective cells in the worst case) only increases the area of circuits by 10%, while virtually does not change the delay. Even more importantly, the defect tolerance with respect to stuck-on-open-type nanodevices is very high. Table 4.3 shows that 99% yield can be achieved with up to 20% of bad nanodevices. On the other hand, similar to CMOL memories, the situation with stuck-on-close-type defects is not very optimistic. The current defect tolerance mechanism only practical for tolerating no more than $\sim 0.02\%$ stuck-on-close nanodevices.[4]

[2] Also, the nanoPLA results might be worse if all circuits had been mapped on a hardware fabric of fixed geometry, as we did for the CMOL architecture. Finally, the fabrication technology assumed in the nanoPLA architecture is much more demanding than CMOL, requiring (besides programmable diodes at crosspoints) nanoscale field-effect transistors, which are inherently irreproducible [73] (see Section 1.2.3 for a more detailed discussion of this point).

[3] For CMOS technology, similar ideas have been developed in several works. For example, Triptych FPGA architecture [12] is based on the universal cell which is used both for routing and logic operations. The authors of Refs. 25, 119 suggested to avoid the worst-case routing limitation in CMOS FPGA during mapping by deliberately underusing logic resources. However, both in Triptych FPGA (in which the universal cell has physically different CMOS hardware for routing and logic operations), and in the latter approach the CMOS circuitry utilization suffers. On the contrary, in CMOL FPGA, the CMOS subsystem utilization may be close to 100%.

[4] Trying to avoid stuck-on-close-type defective nanodevices could be one of the major conclusions

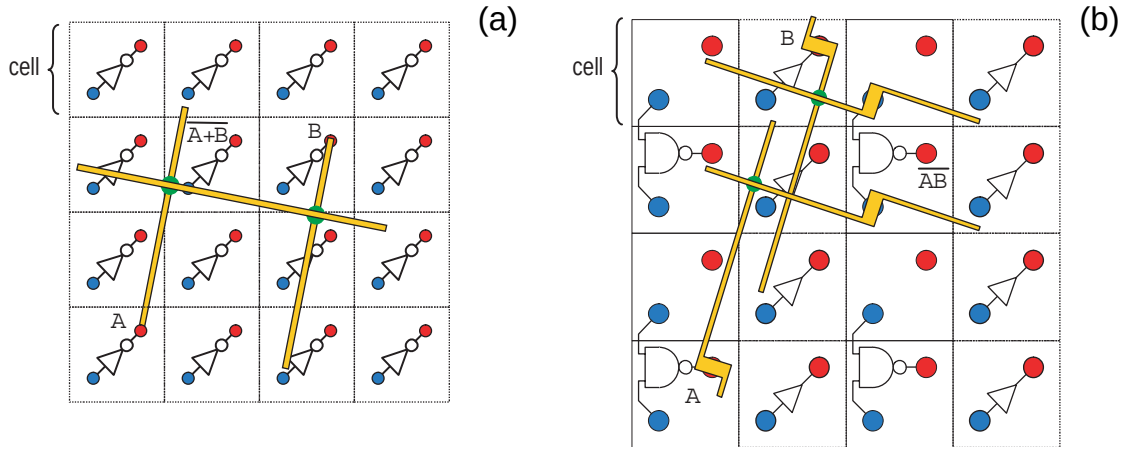

Figure 5.1: Comparison of (a) CMOL FPGA and (b) HP's FPNI logic architectures (adapted from Ref. 102).

Our performance results for CMOL FPGA architecture has become an inspiration for HP's very recent suggestion of the so-called field-programmable nanowire interconnect (FPNI) circuits [102] - see, e.g., Fig. 5.1 showing logic architecture and also Sec. 2.2 and Fig. 2.5 for the discussion of the FPNI topology. Since this idea trades off some performance for simplicity in FPNI circuit fabrication, it is not surprising that the area-delay metric of FPNI circuits is at least about 5 times worse as compared to that of CMOL FPGA (Table 4.1). This is why FPNI circuits may be thought about as an intermediate step towards CMOL FPGA circuits which are superior in performance, but more challenging from the fabrication perspective.

To summarize, we believe that the results presented in this work show a possible substantial impact of the circuit transfer from CMOS to CMOL technology.

5.2 Main Challenges and Possible Future Work

In the previous section we have mentioned in brief all the immediate work which should be done to improve and extend the analysis of considered CMOL memories and CMOL FPGA circuits. Let's us now address main general challenges on the road toward CMOL technology, and possible directions of a long-term research work.

It is quite obvious that the hardest challenge on the way to realization of CMOL-based and any other crossbar hybrids with truly nanoscale dimensions (i.e., with $F_{\rm nano} < 10$ nm) is high fabrication yield of nanodevices, in particular (as this work has shown) low fraction of stuck-on-close-type defects. If no sufficient progress is

for device engineers. For example, it might be possible to burn such nanodevice by running very large current and turning it into stuck-on-open-type one.

made in that direction in a future, this issue alone may prevent the development of the hybrid crossbar circuits.

From the author's point of view, the second most important work towards implementation of considered circuits is the demonstration of high yield CMOS-to-nano CMOL interfaces. Even though all the necessary components of such interfaces have been demonstrated separately, a demonstration of a complete functional interface would help to attract electronic industry attention. Moreover, only semiconductor pins with nanometer-sharp tips have been demonstrated, while the technology for fabricating metallic pins, which have superior properties and could potentially simplify the whole CMOL process flow, still has to be developed.

Obviously, CMOL FPGA circuits, considered in this work, is just a small subset of all possible CMOL circuit architectures. The choice of our particular CMOL FPGA architecture was influenced in part by the conventional CMOS FPGA structures. (Thus similarly has enabled the use of standard approaches in design automation.) It may be possible to improve performance of CMOL circuits in a future by developing more original architectures, as well as using new ideas in circuit mapping. To this end, at least several directions are worth trying. First of all, all nanowire segments in the current approach have the same length. On the other hand, simple analysis shows that by arranging interface pins in a particular order it is possible to obtain a CMOL structure with different length nanowires. By matching the nanowire length distribution to the one of the real-world circuits (which always follow Poisson distribution [24]) routing might be performed much more efficiently. Moreover, it would be natural to add another level of wiring in CMOS. The latter could be used for global interconnects as well as some very high fan-out nets. (An analysis of considered benchmark circuits have shown that there are usually few such nets in each circuit.)

Next, since the logic-to-latch-gates ratio varies from one circuit to another (e.g., about half of the circuits in the Toronto benchmark do not have any latches at all), for some circuits a lot of CMOS area is wasted because of the fixed dedicated hardware resources. In principle, signal latching can be implemented even in one-cell fabric by connecting two basic cells (with inverters) in a loop. The downside of this approach is that the clock signals must be supplied either through the nano-subsystem or specially designated cells. The former option is rather impractical since multiple hops through nano-subsystem will most probably result in high clock jitter and hence asynchronous operations would be more preferable. The latter may be a more attractive option, though more investigation is certainly needed.

We believe that the performance of the considered CMOL FPGA architecture can be greatly improved using more advanced mapping algorithms. For example, in a current design automation flow we allocate routing resources evenly among the tiles of the array. The actual demand of routing resources usually varies, with the center

tiles typically used more heavily. Finding the best routing resource area profile for a particular circuit can be done during placement, e.g., by simply tracking the density of global connections going through the corresponding tile. (Note that a similar trick is not possible in conventional CMOS FPGA due to the fixed dedicated hardware for routing and logic operations.)

Additional studies of design issues related to large-scale CMOL FPGA systems is also very desirable. For instance, it is clear that each CMOL FPGA chip will have a unique pattern of defects, and therefore unique mapping of circuits, i.e. "per chip compilation" instead of "per application circuit" for conventional CMOS FPGA. This means that CMOL FPGA chips should also include a small memory to keep the locations of the defects and most probably a simple CMOS microprocessor performing configuration around such defects. An evaluation of the required area and corresponding delays of such additional circuitry, as well as related issues like test strategies etc., might be a very useful exercise [108].

Concerning the compilation time challenge, note that the detail routing around stuck-on-open-type defects in the considered design automation flow is an incremental step, and therefore can be performed very fast after more time-consuming generic (common to all CMOL FPGA chips) placement and global routing. (In particular, this is the reason for the inferior defect tolerance in comparison to that of FPNI circuits [102]). On the other hand, the avoidance of defective cells is performed during placement and global routing algorithm. Hence one possible work direction would be to make defective cells avoidance also incrementally, integrating it into the detailed routing algorithm.

In addition, large-scale CMOL FPGA circuits would certainly require the development of the system level architecture, e.g., interconnect hierarchies using small CMOL FPGA arrays. For example, Fig. 5.2 shows a possible hierarchical computing structures functionally similar to Teramac's PLASMA chip [7, 43]. For more details on such effort, see Ref. 109.

Finally, there are at least two very interesting original future directions. The first one is developing CMOL FPGA-like circuit architectures for digital signal processing (DSP). In fact, most of the image processing tasks exhibits high degree of parallelism and their structure is usually regular with relatively short interconnections, hence making CMOL FPGA circuits a very attractive platform for their implementation. Indeed, running such tasks on conventional microprocessors would be very slow for runtime applications (e.g., face recognition). The other option, state-of-the-art CMOS DSP implementations can deliver potentially very fast and area efficient solutions; however the ever increasing nonrecurrent engineering costs could make it less practical for future CMOS technology. On the other hand, as this work demonstrated, CMOL FPGA circuits are intrinsically reconfigurable and have a much better performance as compared to purely CMOS FPGA ones. Our preliminary results for three-cell based

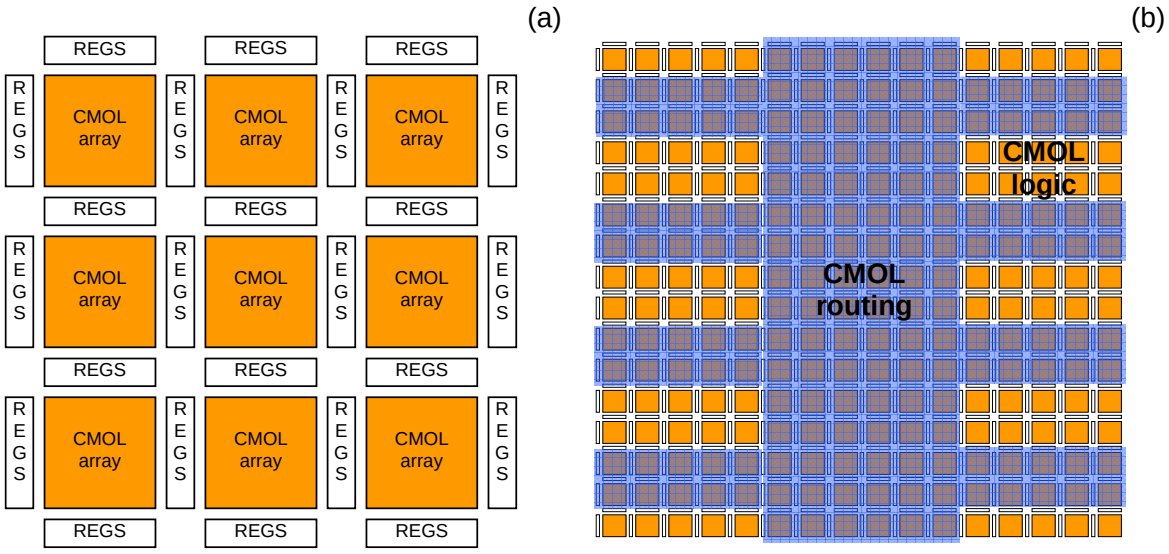

Figure 5.2: (a) A macro-array of one-cell CMOL FPGA arrays interleaved with CMOS registers, and (b) its use for implementation of the PLASMA chip architecture [7]. Note that a particular partition of resources shown on panel (a) is not fixed and can be changed adjusted according to specific application.

field-programmable CMOL DSP architecture have shown that image convolution, one of the most important operations in that image processing, can be implemented very efficiently, while allowing high defect tolerance [115]. For example, a 12-bit image convolution using a 32×32 window size is estimated to take about 100 μs on a $\sim$ 3.5 cm^2 CMOL DSP chip (with $F_{\rm nano} = 3.2$ nm and $F_{\rm CMOS} = 32$ nm) consuming about 200 W/cm^2. On the other hand, the similar task would take about 100 ms for a CMOS microprocessor implemented with the same CMOS design rules and consuming similar power.

Perhaps, even more interesting is exploring CMOL FPGA-like circuit architectures based on nanodevices with negative differential resistance (NDR). For example, Figure 5.3 shows a concept of such circuits (proposed by K. Likharev) with the basic logic gates implemented similar to Goto pairs [105]. (Note that, in contrast to some previous proposals [59, 100], nanodevices in Fig. 5.3 are not changing states during circuit operation.) Clearly, the fabrication of NDR-type nanodevices is even more challenging than that considered in this work. However, it is also obvious that the use of such nanodevices can even further scale up the performance of CMOL FPGA circuits.

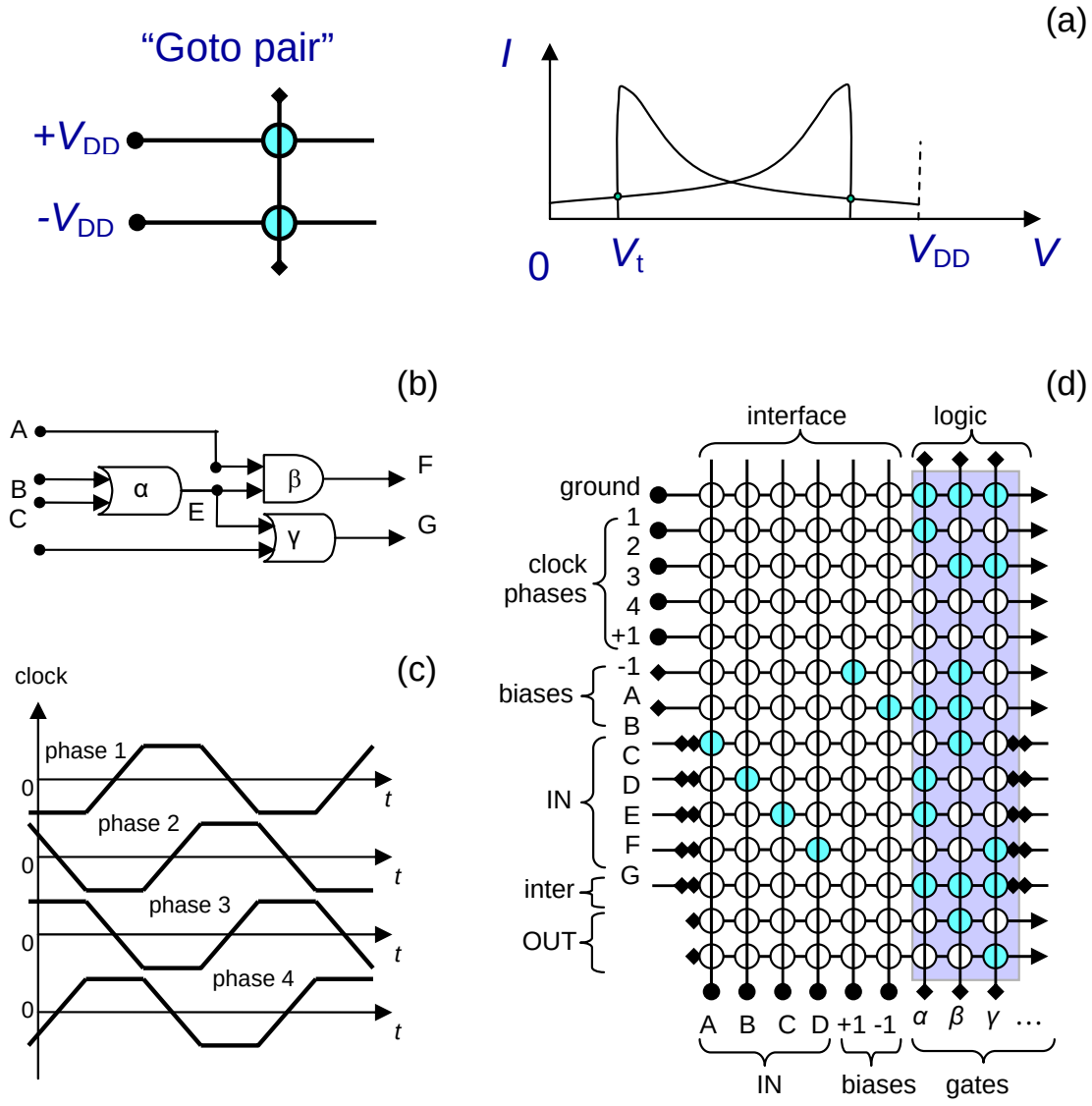

Figure 5.3: Boolean logic based on nanodevices with NDR characteristics (courtesy K. Likharev): (a) the basic idea of a crossbar Goto pair logic (for the detail explanation on how such gate operates, see, e.g., Ref. 105) and an example of simple circuit including (b) circuit schematic, (c) clocking scheme, and (d) mapping on a crossbar structure.

Appendix A

BCH Decoder

A.1 Introduction

The error-correcting codes are being used extensively in virtually all commercial memory and storage devices to tolerate both transient faults and static errors. Typically, light ECCs (e.g., Hamming codes) are used for fast memories, like SRAMs and DRAMs, and more powerful ECCs (e.g., Reed Solomon codes) are used in flash memories and magnetic/optical storages devices. Since the latter have intrinsically slow access time much of the research was done for optimizing the throughput of the ECC decoders rather than their latency [62,94]. On the contrary, the access time of hybrid nanoelectronic memories, e.g. based on CMOL technology (Section 3.4) can be very small, i.e. in the nanosecond range, the fact that makes them suitable for random-access operations. Hence, it is advantageous to use both high-speed and powerful ECCs which can cope with high defect rates presented by this new technology.

Low-latency decoding can be achieved by parallelizing the decoding algorithm. One obvious limitation for parallelization is the increase of the ECC decoder area. For example, standard array decoding [11], i.e. keeping the memory with leader cosets, would be the fastest technique for decoding linear block codes, however the area of such decoder grows exponentially with the number of errors the code can fix. Therefore, to avoid unpractical solutions, we are interested both in latency and area dependence on error-correcting capability of the code. Similar tradeoffs analysis was carried out for various high throughput ECC decoders [81, 84].

A fast bit-parallel finite field arithmetic (finite field arithmetic is required in many decoding algorithms) was already heavily studied in the context of cryptography [85, 86]. In this paper we use some of these results to study the area and latency tradeoffs of the fast decoders for binary BCH codes [11].[1] (The binary codes are

[1]Though BCH codes are perhaps not the absolutely best option for nanoelectronic memories, they are the most efficient ones among short (less than 1024 bits long) codes. Currently, we are investigating a more natural codes, e.g., Euclidean geometry and LDPC ones [11,62], and also codes

better suited for our defect model with errors uniformly distributed throughout the memory matrix. Such model is adequate for molecular-scale memories and was widely used in other research papers [29, 32, 101].)

A.2 Model

A.2.1 General Structure

For simplicity, let us consider primitive BCH codes, with the code length on the finite Galois field GF(2^m) is $n = 2^m - 1$. A plausible implementation of the fast decoding for binary BCH codes requires three major steps [11, 62]:

- Step 1: Syndrome evaluation;
- Step 2: Finding the error-location polynomial using the Berlekamp-Massey iterative algorithm; and
- Step 3: Finding error-location numbers with subsequent error-correction.

  Let us estimate the area and delay contributions of the circuits of each step.

A.2.2 Syndrome Calculation

In a completely bit-parallel implementation each bit of a syndrome $\boldsymbol{S}$ can be obtained by a separate XOR tree with inputs taken from the received code vector [62]:

$$\boldsymbol{S} = (S_1, S_2, ..., S_{2t}) = \boldsymbol{r}\boldsymbol{H}^{\mathrm{T}} = \boldsymbol{r} \begin{pmatrix} 1 & 1 & . & 1 \\ (\alpha) & (\alpha^2) & . & (\alpha^{2t}) \\ . & . & . & . \\ (\alpha)^{n-1} & (\alpha^2)^{n-1} & . & (\alpha^{2t})^{n-1} \end{pmatrix}. \tag{A.1}$$

Here $\boldsymbol{r}$ is a received code word of length n, $\mathbf{H}$ is a parity matrix, t is the maximum number of errors the code can fix, and α is a primitive element in GF(2^m), i.e. each column in the matrix in the Eq. (A.1) can be rewritten as an m binary columns. On the average, binary columns of $\boldsymbol{H}$ are half filled with ones while in the worst case there is a column with almost all ones. Hence the syndrome calculation circuit will be comprised of total $2tm$ XOR trees with the average depth $\log_2(n/2)$ and the worst depth of $\log_2 n$ (Fig. A.1), i.e.

$$\tau_1 = \tau_{\mathrm{add}}(n) = \lceil \log_2(n) \rceil \times \tau_{\mathrm{XOR}}, \tag{A.2}$$

which take advantage of the asymmetry of the error model.

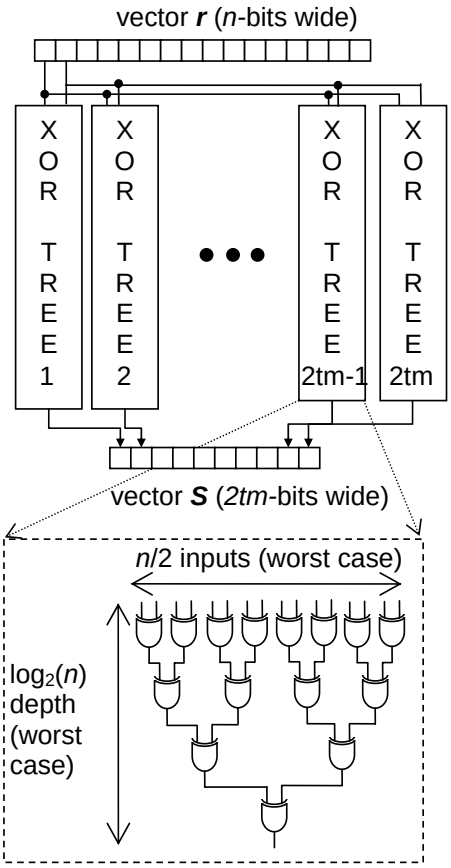

Figure A.1: Syndrome calculation.

$$A_1 = 2tm \times A_{\text{add}}(n/2) = tnm \times A_{\text{XOR}}. \tag{A.3}$$

Here by $\tau_{\text{add}}(n)$ and $A_{\text{add}}(n)$ we denote, correspondingly, the delay and area of 1-bit parallel addition in finite field (XOR tree) of n elements. Similarly, τ_{XOR} and A_{XOR} (and also τ_{OR} and A_{OR} which are used later in text) denote the delay and area of Boolean logic 2-input XOR (OR) gate, respectively.

A.2.3 Finding Error-Location Polynomial with Berlekamp-Massey Algorithm

Since the algorithm is based on a recursive procedure it cannot be parallelized completely. However, it is possible to speed up the computation of each iteration. Two most time-consuming operations in this step are calculating discrepancy d (element of the field GF(2^m)) and adjusting error location polynomial $\sigma(X)$ if necessary [62],

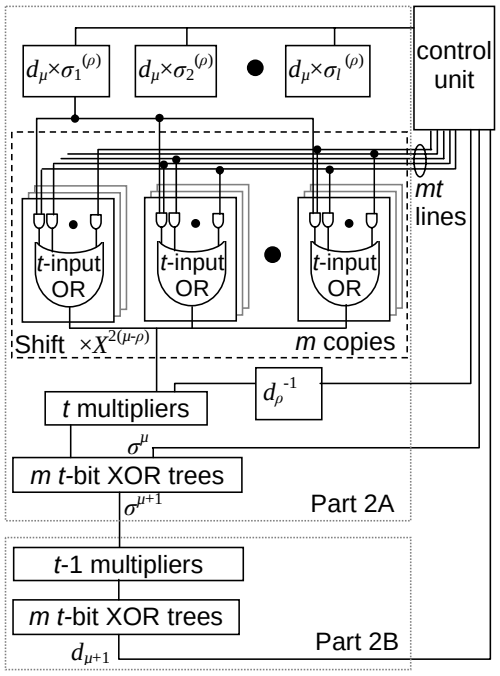

Figure A.2: Finding error-location polynomial with Berlekamp-Massey algorithm.

e.g., for the $\mu + 1$ step ($0 \leq \mu \leq t - 1$):

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{2(\mu-\rho)} \sigma^{(\rho)}(X), \tag{A.4}$$

$$d_{\mu+1} = S_{2\mu+3} + \sigma_1^{(\mu+1)} S_{2\mu+2} + \sigma_2^{(\mu+1)} S_{2\mu+1} + ... + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{2\mu+3-l_{\mu+1}}. \tag{A.5}$$

Here $\rho \leq \mu$ and $l \leq \mu$ are indices specific to the implementation of the algorithm [62]. For the worst case, one needs to update $\sigma_\mu(X)$ in each iteration μ, while the degree of $\sigma_\mu(X)$ exactly equal to μ. Hence the calculation of $\sigma_\mu(X)$ in early iterations can be done with smaller circuit area and (slightly) faster time. However, it also implies different hardware for each iteration. Instead, we will assume the scheme shown in Fig. A.2 where the hardware is shared among the iterations and the latency is the same in all iterations, being dictated by the operations on $\sigma_\mu(X)$ with the maximum degree t.

Implementation of Eq. (A.4) (see part 2A on Figure A.2) involves one inversion in GF(2^m), $2t$ multiplications in GF(2^m), m additions of t bits and multiplying by the indeterminate $X^{2(\mu-\rho)}$ (i.e. a shift operation). Fast inversion can be done directly as a hard-wired LUT table for small m, so that the complexity of such operation is roughly [85]:

$$\tau_{\text{inv}} = \lceil \log_2(m-1) \rceil \times \tau_{\text{AND}} + (m-1) \times \tau_{\text{OR}}, \tag{A.6}$$

$$A_{\text{inv}} = m^2 n/4 \times A_{\text{AND}} + mn/4 \times A_{\text{OR}}. \tag{A.7}$$

For small values of m (less than 10) the Mastrovito multiplier [80] ensures fast efficient implementation comparable with the other approaches [85]. The complexity of such multiplier is

$$\tau_{\text{mult}} = \tau_{\text{AND}} + 2\lceil \log_2 m \rceil \times \tau_{\text{XOR}}, \tag{A.8}$$

$$A_{\text{mult}} = m^2 \times A_{\text{XOR}} + m^2 \times A_{\text{AND}}. \tag{A.9}$$

Complexity for the fast shift operation (Fig. A.2) is

$$\tau_{\text{shift}} = \tau_{\text{AND}} + \lceil \log_2 m \rceil \times \tau_{\text{OR}}, \tag{A.10}$$

$$A_{\text{shift}} = mt^2 \times A_{\text{AND}} + mt^2 \times A_{\text{OR}}. \tag{A.11}$$

Thus, the complexity of the part 2A during one iteration is

$$\begin{aligned} \tau_{2\text{A}} =& \tau_{\text{inv}} + \tau_{\text{mult}} + \tau_{\text{add}} = (1 + \lceil \log_2(m-1) \rceil) \times \tau_{\text{AND}} + \\ & (2\lceil \log_2 m \rceil + \lceil \log_2 t \rceil) \times \tau_{\text{XOR}} + (m-1) \times \tau_{\text{OR}}, \end{aligned} \tag{A.12}$$

$$\begin{aligned} A_{2\text{A}} =& A_{\text{inv}} + 2t \times A_{\text{mult}} + m \times A_{\text{add}} = \\ & (2mt^2 + 2m^2 t + m^2 n/4) \times A_{\text{AND}} + \\ & (m^2 t + mt) \times A_{\text{XOR}} + (m^2 t + mn/4) \times A_{\text{OR}}. \end{aligned} \tag{A.13}$$

Here it is assumed that inversion is slower operation than the first multiplication and shift, and hence it is included in the critical path delay calculation. Finally, assuming that part 2B is implemented with $(t-1)$ multiplications and additions, and neglecting other circuitry (e.g., a control unit), the total complexity of the Step 2 is

$$\begin{aligned} \tau_2 =& t(2 + \lceil \log_2(m-1) \rceil) \times \tau_{\text{AND}} + \\ & t(4\lceil \log_2 m \rceil + 2\lceil \log_2 t \rceil) \times \tau_{\text{XOR}} + t(m-1) \times \tau_{\text{OR}}, \end{aligned} \tag{A.14}$$

$$\begin{aligned} A_2 =& (2mt^2 + 3m^2 t + m^2 n/4 - m^2) \times A_{\text{AND}} + \\ & (3m^2 t + 2mt - m^2) \times A_{\text{XOR}} + (m^2 t + mn/4) \times A_{\text{OR}}. \end{aligned} \tag{A.15}$$

A.2.4 Finding Error Location Numbers and Correction

The simple substitution which is performed in parallel for all 2^m elements is the fastest (though rather area-expensive) way to find the error location numbers. The test circuit for checking whether some element from GF(2^m) is a root of error location polynomial $\sigma(X)$ requires a multiplication by a constant (Fig. A.3) which has a complexity [80, 85]:

$$\tau_{\text{const}} = \lceil \log_2 m \rceil \times \tau_{\text{XOR}}, \tag{A.16}$$

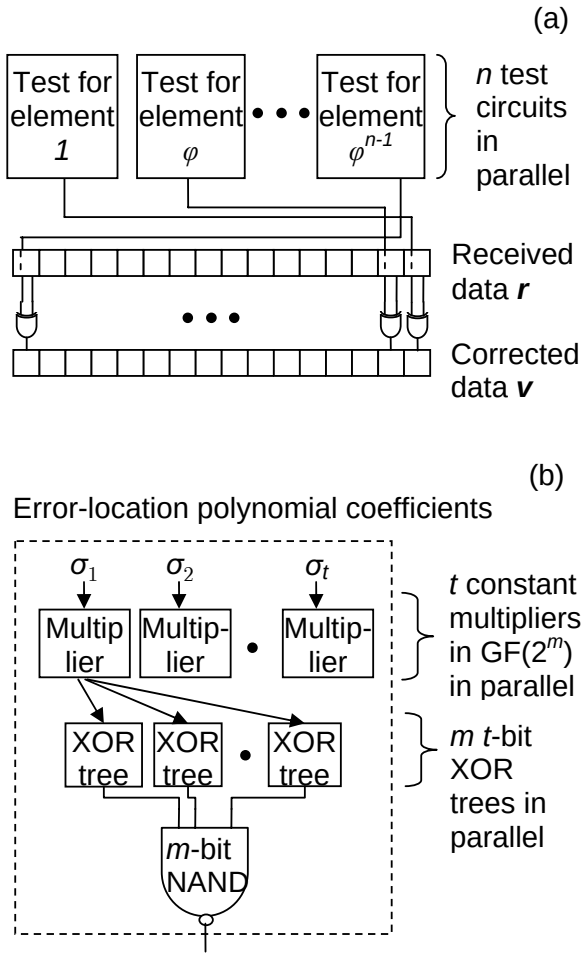

Figure A.3: (a) Finding error location numbers and (b) error correction.

$$A_{\text{const}} = (m^2/2 - m) \times A_{\text{XOR}}. \tag{A.17}$$

Hence the test circuit can be implemented with

$$\begin{aligned} \tau_{\text{test}} =& \tau_{\text{const}} + \tau_{\text{add}} + \lceil \log_2 m \rceil) \times \tau_{\text{AND}} = \\ & (\lceil \log_2 m \rceil + \lceil \log_2 t \rceil) \times \tau_{\text{XOR}} \lceil \log_2 m \rceil) \times \tau_{\text{AND}}, \end{aligned} \tag{A.18}$$

$$A_{\text{test}} = t \times A_{\text{const}} + m \times A_{\text{add}} + m \times A_{\text{AND}} = tm^2/2 \times A_{\text{XOR}} + m \times A_{\text{AND}}, \tag{A.19}$$

so that the total complexity of the Step 3 is

$$\tau_3 = \tau_{\text{test}} + \tau_{\text{XOR}} = \lceil \log_2 m \rceil) \times \tau_{\text{AND}} + (1 + \lceil \log_2 m \rceil + \lceil \log_2 t \rceil) \times \tau_{\text{XOR}}, \tag{A.20}$$

$$A_3 = n \times A_{\text{test}} + m \times A_{\text{XOR}} = (tm^2 n/2 + n) \times A_{\text{XOR}} + nm \times A_{\text{AND}}. \tag{A.21}$$

Operation in GF(2^m)	Area			Critical Path Delay		
	OR	AND	XOR	OR	AND	XOR
Addition (n elements)	0	0	mn	0	0	$\lceil \log_2 n \rceil$
Multiplication	0	m^2	m^2	0	1	$2\lceil \log_2 m \rceil$
Constant Multiplication	0	0	$m^2/2$-m	0	0	$\lceil \log_2 m \rceil$
Inversion	$mn/4$	$m^2n/4$	0	m-1	$\lceil \log_2(m\text{-}1) \rceil$	0
Multiplication by indeterminate of polynomial with degree t	mt^2	mt^2	0	$\lceil \log_2 m \rceil$	1	0

Table A.1: Complexity of key operations in BCH decoder

Operation	Area			Critical Path Delay		
	OR	AND	XOR	OR	AND	XOR
Step 1	0	0	tnm	0	0	$\lceil \log_2 n \rceil$
Step 2	$mt^2 + mn/4$	$3m^2t + 2mt^2 + m^2n/4 - m^2$	$3m^2t + 2tm - m^2$	$t(m\text{-}1)$	$2t + t\lceil \log_2(m\text{-}1) \rceil$	$4t\lceil \log_2 m \rceil + 2t\lceil \log_2 t \rceil$
Step 3	0	mn	$m^2tn/2 + n$	0	$\lceil \log_2 m \rceil$	$1+\lceil \log_2 m \rceil + \lceil \log_2 t \rceil$
Total	$mt^2 + mn/4$	$3m^2t + 2mt^2 + m^2n/4 + mn - m^2$	$tnm + 3m^2t + 2tm + m^2tn/2 + n - m^2$	$t(m\text{-}1)$	$2t + t\lceil \log_2(m\text{-}1) \rceil + \lceil \log_2 m \rceil$	$1+\lceil \log_2 n \rceil + (4t+1)\lceil \log_2 m \rceil + (2t+1)\lceil \log_2 t \rceil$

Table A.2: Complexity of proposed BCH decoder

A.3 Results and Discussion

Table A.1 outlines the complexity of key operations used in BCH decoder model, while Table A.2 presents our main results. In particular, the latter shows that for large values of n the area of the decoder scales approximately as nm^2t and mostly dominated by the circuitry in Step 3. On the other hand, most of the delay, which is roughly proportional to mt, comes from Step 2.

As expected the full delay is certainly much better than that of bit-serial BCH decoder [11]. Moreover, Table A.2 shows that it might be possible to reduce the area-delay product without significant area or delay degradation by balancing performance among all steps. For example, since the delay of the Step 3 is negligible its area might be reduced by implementing a slower algorithm.

It is more convenient to express the decoder complexity in technology-independent units such as CMOS half-pitch F_{CMOS} and fanout-of-4 inverter delay τ_{FO4}. To simplify the analysis let us assume the static CMOS gate implementation. Using SCMOS rules [91], the areas of the static minimum-size 2-input OR (6-transistor), 2-input AND (6-transistor) and 2-input XOR (10-transistor) gates are roughly equal to 150, 150 and 250 $(F_{\mathrm{CMOS}})^2$, while their delays (assuming that each gate drives only one copy of itself) are about 2/3, 2/3 and 9/5, respectively [117]. Therefore, adding

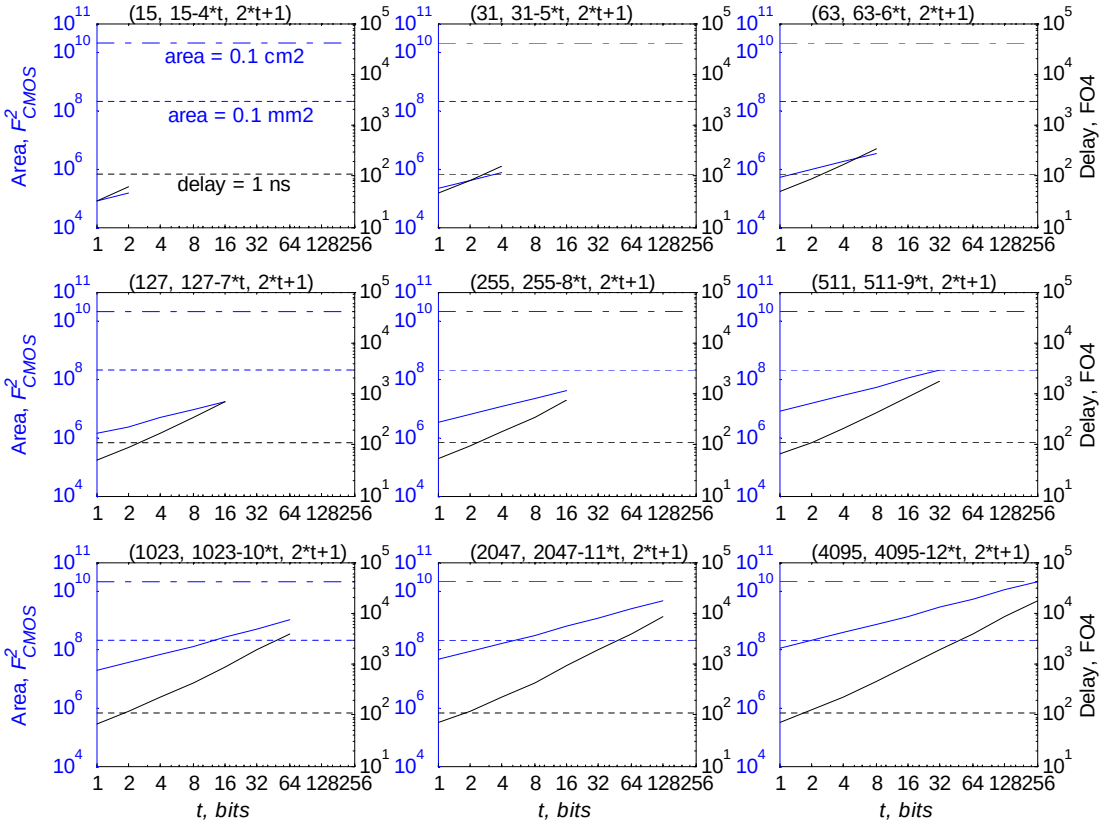

Figure A.4: The area and delay as a functions of the maximum number of errors t the BCH code can correct for several values of n. For convenience, horizontal lines show the delay of 1 ns, and the areas of 0.1 cm^2 and 0.1 mm^2, for the 22 nm ITRS technology [1] nodes.

contributions from each step, and droping negligibly small terms, the full area and latency of the BCH decoder can be described by

$$\begin{aligned} A_{\text{BCH}} \approx\ & 125nt(\log_2 n)^2 + 40n(\log_2 n)^2 + 250nt\log_2 n \\ & +190n\log_2 n + 1200t(\log_2 n)^2 + 300t^2\log_2 n. \end{aligned} \tag{A.22}$$

$$\begin{aligned} \tau_{\text{BCH}} \approx\ & 0.7t\log_2 n + 0.7t + 8t\log_2\log_2 n + 3.6t\log_2 t \\ & +2.5\log_2\log_2 n + 1.8\log_2 t + 1.8. \end{aligned} \tag{A.23}$$

Figure A.4 shows the dependence of the area and delay versus the code efficiency, i.e. the maximum number of errors t which can be decoded from the minimum distance, for several codes of different block length. We assumed here that each code has mt parity-check bits and the minimum distance of the code is equal to $2t + 1$.

(For large values of n and t the number of parity bits can be somewhat smaller [11]; however it does not affect the results on Figure A.4.) To appreciate the results for a $F_{\mathrm{CMOS}} = 22$ nm CMOS technology node ($\tau_{\mathrm{FO4}} = 9$ ps) Figure A.4 also shows the delay of 1 ns, and also the areas of 0.1 cm^2 and 0.1 mm^2, which may be a reasonable chip's real estate for the stand alone and embedded memories.

It is worth mentioning that the results in Eqs. A.23, A.22, and also in Figure A.4, are rather crude because:

(i) We do not account for wire delay. The contribution of these delays can be rather significant, especially for deep submicron CMOS nodes [45], i.e. for the cases we are mostly interested. Moreover, in the first two steps there are few places with a large fan-out on the critical path, which would require extra stages for an optimal design.

(ii) Our area estimates are based on the minimum-width gates. The optimization of delay will certainly require some gates to be larger than the minimum size and hence the area will be somewhat larger.

(iii) The circuitry can be optimized to reduce delay and possibly area by using faster circuit families. For example, static XOR gate is about 30% slower than the fastest implementation [16]. Moreover, the latency may be further reduced by balancing the delay of each stage [117] in the circuit.

The first issue is probably the most important and our estimates are likely to be on the optimistic side. Nevertheless, even such crude analysis might be enough to estimate performance overheads of ECC decoders in the prospective memory technologies with high defect rates.

Appendix B

Prior Work on CMOL FPGA circuits

B.1 Architecture and Reconfiguration Algorithm

In our first work on CMOL FPGA circuits [109] we have considered one-cell fabric similar to the one shown on Fig. 4.1 but with additional 45° angle (Fig. B.1). More exactly, the angle α and the dimensionless factor β was chosen to satisfy two requirements:

$$\cos\alpha = \frac{rF_{\text{nano}}}{\beta F_{\text{CMOS}}}, \qquad \sin\alpha = \frac{(r-1)F_{\text{nano}}}{\beta F_{\text{CMOS}}}, \tag{B.1}$$

where r is a positive integer number, which, similar to parameter a given by Eq 2.1, defines the range of cell interactions. Due to additional crossbar angle parameters a and r are roughly related as

$$r = \sqrt{2}a. \tag{B.2}$$

Also note that the number of cells M in a cell connectivity domain is given by

$$M = 2r(r-1) - 1. \tag{B.3}$$

In our early work, the placement and global routing steps were done manually. On the other hand we have used almost the same algorithm described in Sec. 4.2.6 for detail routing to reconfigure the circuits around defective stuck-on-open nanodevices. In fact, the only difference in the algorithm in Ref. 109 is that we have processed gates in rather naive order, i.e., starting from those located (after placement) at one edge of the fabric and finishing with those at the other end.

B.2 32-bit Kogge-Stone Adder

As the first example, we have consider the CMOL FPGA implementation of an integer, parallel-prefix adder which is one of the key digital logic circuits in digital design - see, e.g., Ref. [91]. Among such adders, the Kogge-Stone adder [53] has the

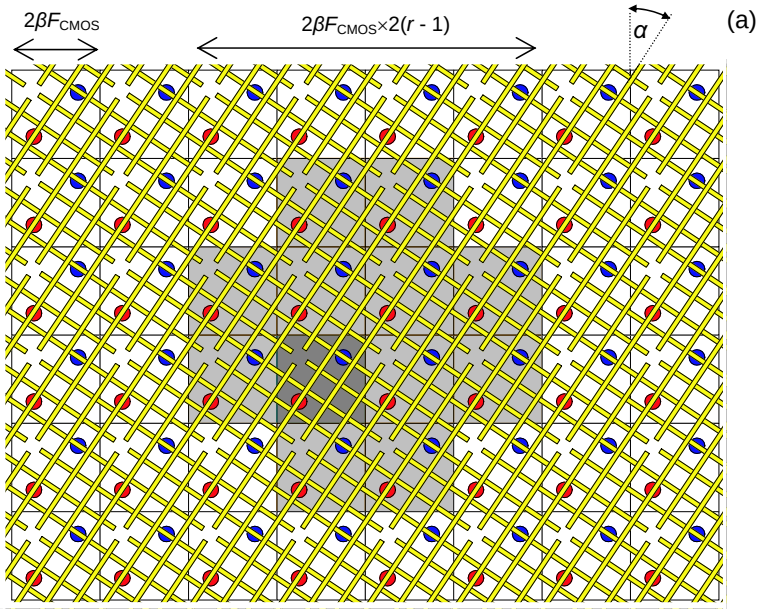

Figure B.1: One-cell CMOL FPGA fabric with additional 45° crossbar tilt. On this figure, $M = 2r(r-1) - 1$ CMOS (basic) cells painted light-gray (in the shown case, $r = 3$, $M = 11$ form the "cell connectivity domain" for the input pin of the cell painted dark-gray. (The output cell connectivity domain has as many cells.) Note that per CMOS cell border there are r nanowires of one orientation and $(r-1)$ of the perpendicular orientation.

most regular structure (Fig. B.2a,b) and therefore manual mapping was not that time consuming. To perform such mapping, first, the 32-bit adder circuitry has been converted into a netlist of fan-in-two NOR gates (Fig. B.2c) and then mapped onto a rectangular CMOL block (Fig. B.3), with interleaved inputs A[31:0] and B[31:0] on the top side and outputs S[31:0] and Cout on the bottom side. (For simplicity, C_{in} is assumed to be always "0"). The mapping procedure was first performed for one bit slice and then repeated for the rest of the circuit, for several values of the cell connectivity domain radius $r' \leq r$. For example, Fig. B.3a shows the map for $r' = 10$. For this case the cell connectivity domain's diagonal has $2(r'-1) = 18$ cells; however in the last (fourth) logic stage the signal vector G (generate) has to span over 32 cells in the horizontal direction. To implement such connections, two additional inverters have been added in the design in each bit slice. Also, assuming that the inputs to the adder are provided by CMOS lines, the broadcast of the input signal vectors A and B (Fig. B.2b) has been avoided by adding another logic level (Fig. B.2c).

For this particular value of r', the final "logic depth" (the number of logic levels in the critical path) is 21, the number to be compared to 13 levels for the conventional

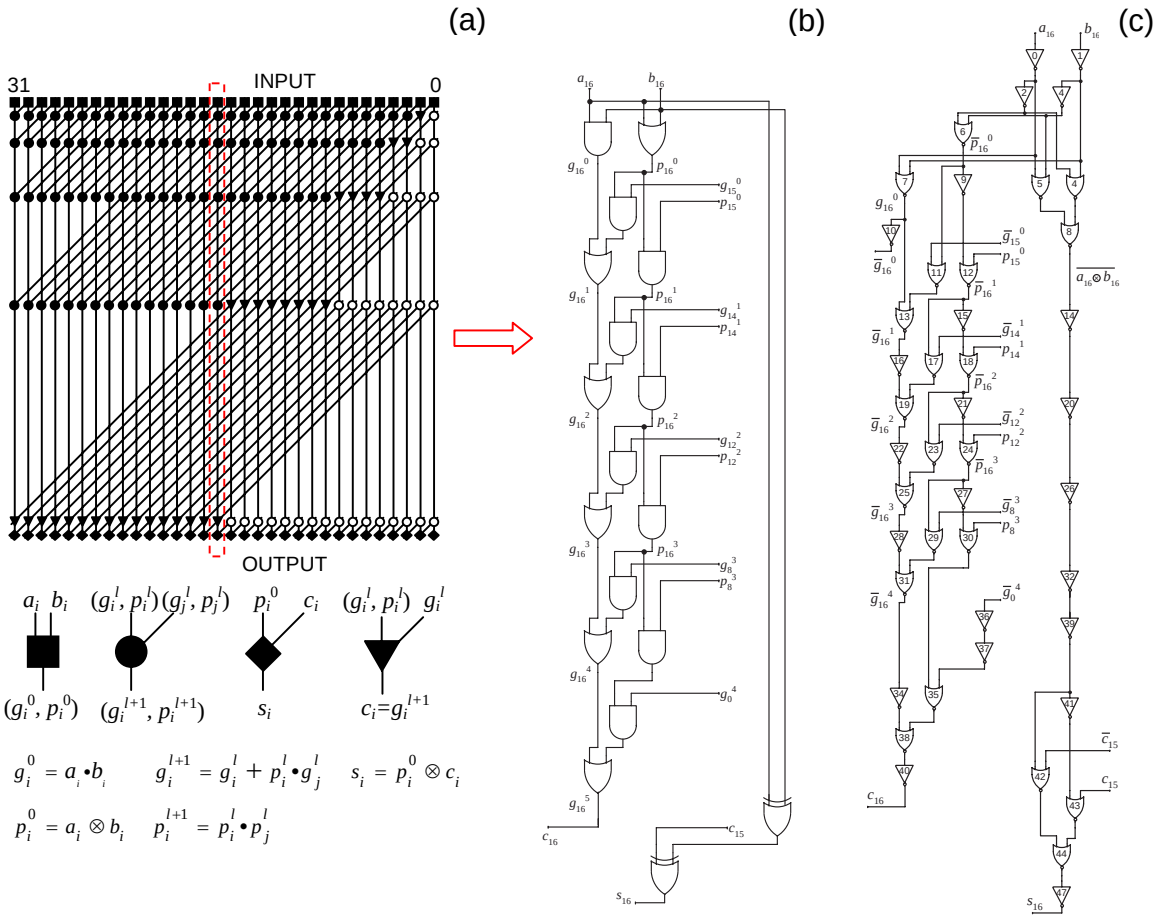

Figure B.2: (a) The 32-bit Kogge-Stone adder and ((b),(c)) its single (16th) bit slice implemented with: (b) AND, OR, and XOR gates, and (c) NOR gates only.

implementation, and 7 levels for the implementation with [4:1] LUTs. Figure B.4 shows the depth as a function of r'. Smaller values of r' result in larger depth and hence the larger total number of CMOS cells, up to the point $r' = r_{\min}$, at which the layout becomes impossible. However, a reduction of r' is beneficial for defect tolerance - see Section B.4.

B.3 A Full Crossbar

Routing resources are a very important part of the conventional FPGAs, as well as more exotic reconfigurable systems such as the Teramac computer [43]. This is why as our second case we have chosen the fully-connected crossbar (Fig. B.5a). For this circuit even the initial mapping on a rectangular CMOL array (with gates working as

simple inverters) may be readily automated, for example using the simple "greedy" algorithm. In this procedure, the I/O pairs to be connected are mapped onto the array one-by-one. Each pair is first assigned a perfect-world Manhattan route, using the vertical rows of the input and output cells and some horizontal row (Fig. B.5b). The algorithm checks that the vertical fragments of various routes do not overlap, while uniformly distributing their horizontal fragments among the array rows.

Then, to create an actual path for each I/O pair, the algorithm tries to allocate cells which are closest to the perfect route and are within each other's connectivity radius. (Of course, the cells used in mapping of the previously routed pairs cannot be used again.) Just as for the other cases, an artificial reduction of the connectivity to radius $r' \leq r$ (at the initial mapping only) improves the final defect tolerance.

In order to use this (or any other) routing algorithm practically, one needs to select the vertical size m of the CMOL array first (Fig. B.5a). In general, we are interested in the smallest value of m, because this leads to the smallest area and logic depth of the crossbar. Such value can be calculated considering the worst possible combinations of the I/O pairs, which result in the largest aggregate data flow (n routes) across the middle cross-section S of the rectangular array (Fig. B.5c). Since CMOL fabric has $(r-1)$ nanowires passing over each CMOS cell in the least favorable direction (Fig. B.1a), and only $(r'-1)$ of them are used at the constrained-radius mapping, there are only $m(r'-1)$ nanowires overall to serve the critical cross-section S. This is why the crossbar height should satisfy the condition $m(r'-1) \geq n$. Moreover, in our simple "greedy" algorithm, nanowires of the same critical cross-section may be used to provide vertical transport of (in the worst case) n routes, so that a more strict condition should be satisfied: $m(r'-1) \geq n+m$, i.e. $m \geq n/(r'-2)$. Finally, including two input and output rows, the minimal crossbar height is $m_{\min} = \lceil n/(r'-2) \rceil + 2$.

From here, the maximum logic depth d (the number of cell-to-cell hops) of the crossbar may be calculated as $\lceil (n+m)/(r'-2)$, because the longest route has the length of $(n+m)$ cells and each cell-to-cell hop allows to move along this route by $(r'-2)$ cells in the worst (left-to-right) case. The resulting dependence of the logic depth of a 64-bit crossbar on r' is shown in Fig. B.4; it is substantially smaller than the depth of the 32-bit adder which has the same total input vector width (32 + 32 = 64).

B.4 Results

We have performed defect tolerance analysis using Monte Carlo simulation technique. As in our latest work the defects were randomly generated with the same probability q. For some (randomly chosen) successful reconfiguration runs the final layout has been functionally simulated to verify the correctness of the design. This was achieved by first saving the layout in the BLIF format and then converting it into the structural

VHDL code with the help of the SIS package [95]. The verification has been fully successful.

Figure B.3b shows the final connection map of the same adder as in Fig. B.3a ($r' = 10$), after a typical successful reconfiguration with $r = 12$ and $q = 0.5$, while Fig. B.7 shows the layout of a small fragment of this circuit, with defective nanodevices marked black. We were very much impressed how resilient the circuit was, retaining full functionality after the reconfiguration around as much as 50% of bad devices. Actually, the defect tolerance could be even higher if we allowed the input and output cells of the adder to be moved (as can be done at a joint reconfiguration of several functional units) or processing gates in a specific order (Sec. 4.2.6).

Figure B.6 shows the fault tolerance of the adder and full crossbar as a function of r and r'. If we choose not to confine the initial mapping additionally (i.e., take $r' = r$), the circuit becomes more defect tolerant as r is increased. (With a fixed CMOS technology, $F_{\rm CMOS}$, this requires scaling down the nanowire and nanodevice half-pitch $F_{\rm nano}$.) If r, i.e. fabrication technology, is fixed, the defect tolerance still may be improved remarkably by taking just a slightly lower r'. The practical limit for this reduction is imposed by the explosive growth of the logic depth at $r' \to r_{\rm min}$ (Fig. B.4), as the corresponding performance degradation. The results show, for example, that at realistic parameters ($r = 12$, $r' = 10$) circuits may have a fabrication yield above 99% at the fraction of bad nanodevices as high as ~25% (Fig. B.6).

The performance model considered for that work was exactly the the as described in Sec. 4.3. The optimization results for the two considered circuits are shown in Fig. 4.16. For example, for the apparently realistic values $F_{\rm CMOS} = 32$ nm and $F_{\rm nano} = 8$ nm, the 32-bit CMOL FPGA adder could have an area about 110 μm^2 and the total logic delay 1.3 ns, at acceptable power dissipation. (Note that the corrected delay would be almost twice smaller due to erroneous factor in noise swing calculations in Ref. 109.)

In order to compare these numbers with purely CMOS FPGAs, we have used the Xilinx ISE WebPack package (see http://www.xilinx.com) to simulate the similar 32-bit Kogge-Stone adder for the commercially available 90nm Xilinx Spartan-3 technology. (The basic unit of such FPGA is a slice consisting of two 4-input LUTs.) The total delay of the adder, excluding the pin-to-slice propagation delay, has turned out to be about 5.1 ns. Assuming the $1/s$ delay scaling [91], this corresponds to 1.7 ns for the 32-nm technology. The circuit area of the CMOS circuit could be calculated from the known number of its cells ("tiles"), equal to 139, and the tile area estimate of approximately 2,100 μm^2, which follows from the data cited in Ref. 87. With the usual $1/s^2$ area scaling, for $F_{\rm CMOS} = 32$ nm this gives the 280 μm^2 tile area, i.e. a total adder area of about 39,000 μm^2. Thus the delay-area product would be about 70,000 ns-μm^2, i.e. about 500 times larger than in CMOL FPGA with the same $F_{\rm CMOS}$.

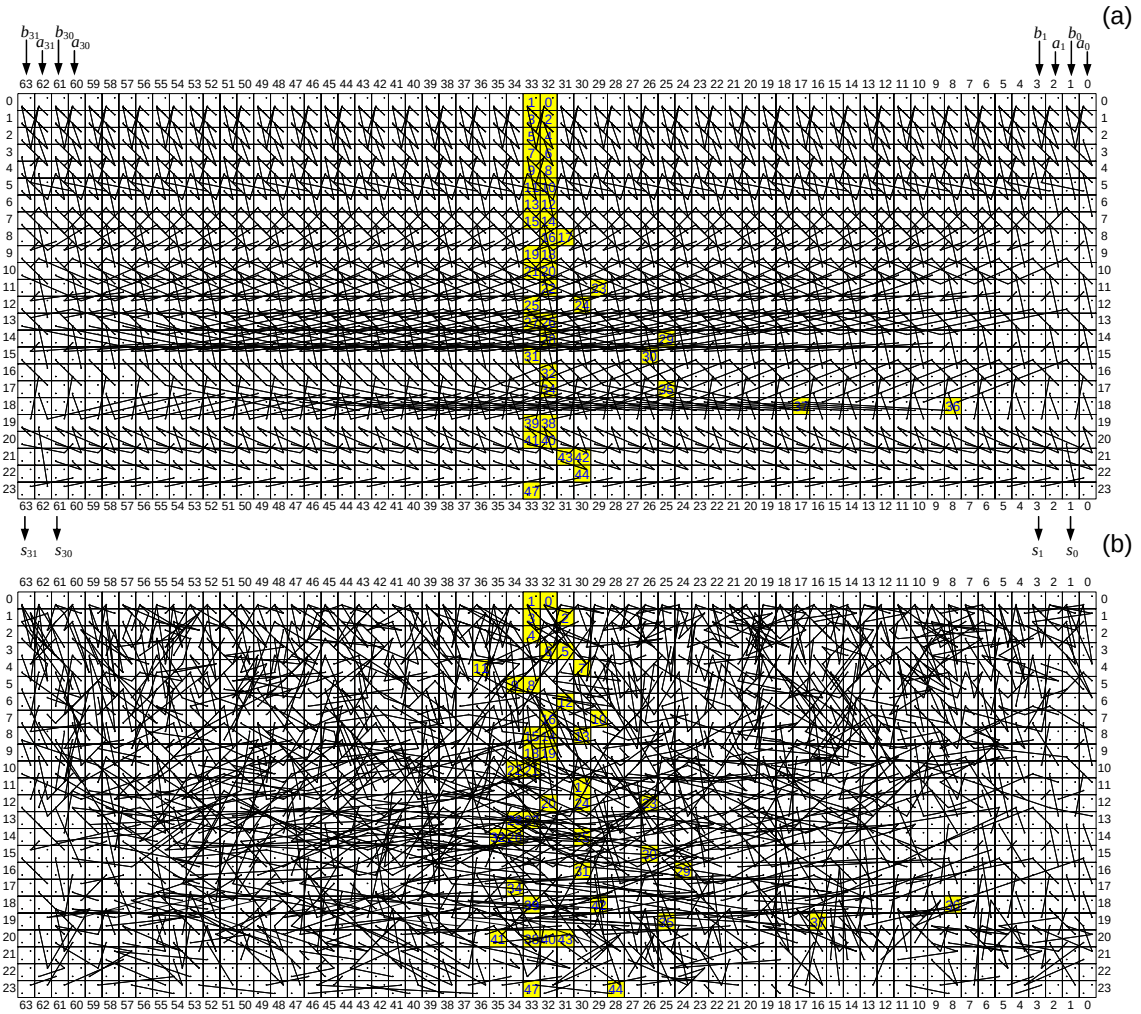

Figure B.3: Mapping of the 32-bit Kogge-Stone adder on CMOL FPGA fabric with $r' = 10$: (a) the corresponding initial map of cell connections, and (b) the connection map after the successful reconfiguration on the circuit around as many as 50% of bad stuck-on-open-type nanodevices (for $r = 12$). Gates of the 16th bit sllice (see the dashed line in Figure **??**a are painted yellow and numbered in accordance with Figure **??**c.

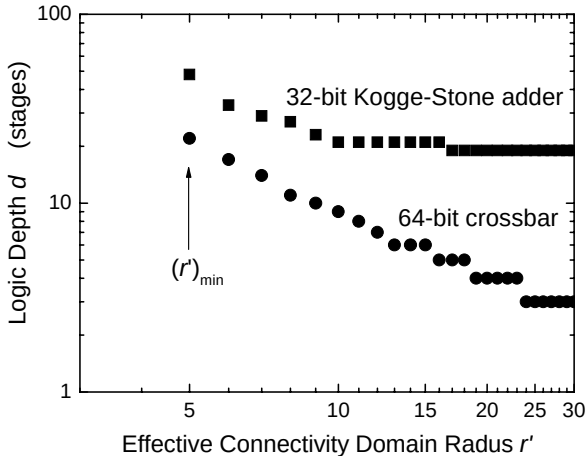

Figure B.4: Logic depth (critical path length in this case) of the two studied circuits as a function of the effective cell connectivity radius r'.

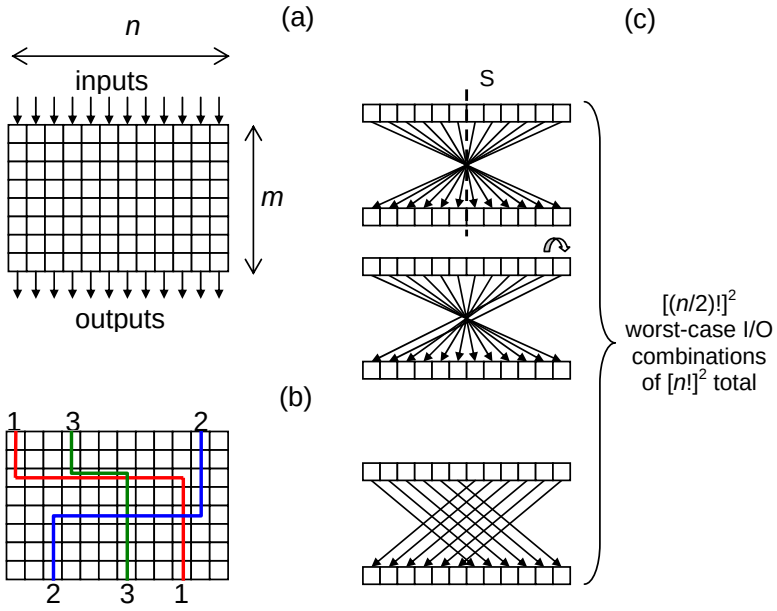

Figure B.5: Full crossbar: (a) general configuration of the CMOL fabric, (b) perfect Manhattan routes used by the "greedy" algorithm, and (c) the family of worst-case I/O pairs.

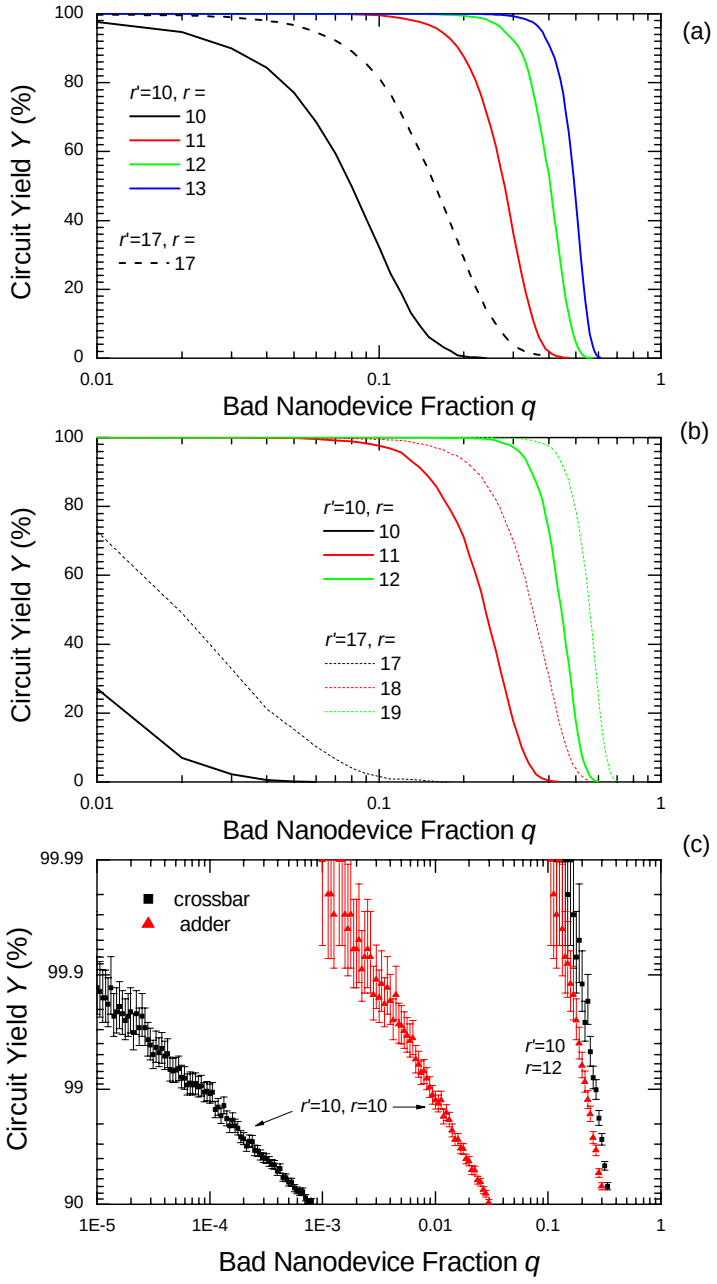

Figure B.6: The final (post-reconfiguration) defect tolerance of (a, c) the 32-bit Kogge-Stone adder and (b, c) the 64-bit full crossbar for several values of r and r'. Panel (c) shows the defect tolerance of the circuits on the log scale which makes visible the results for the most interesting (high) values of yield. This panel is for the same values of r and r' which have been used for Fig. B.3.

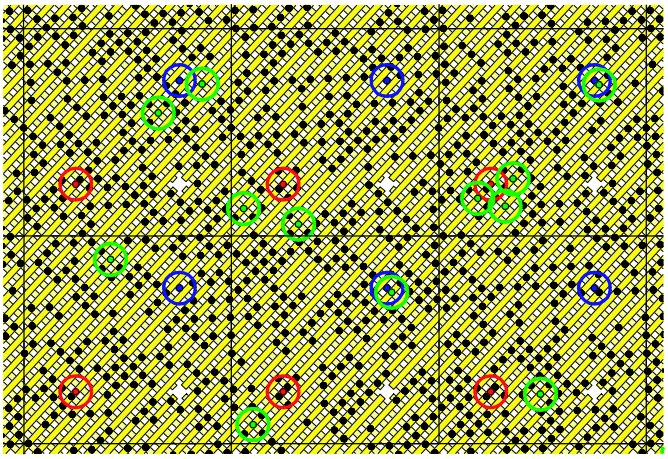

Figure B.7: A small fragment of the adder after the same reconfiguration as in Fig. B.3b. Bad nanodevices (50% of the total number) are shown black, good used devices green, unused devices are not shown, for clarity. Colored circles are only a help for the eye, showing the location of interface pins (red and blue points) and used nanodevices. Thin vertical and horizontal lines show CMOS cell borders.

Appendix C

CMOL FPGA CAD

C.1 Structure

CMOL FPGA CAD software [4] is an interactive tool for mapping sequential circuits on two-cell CMOL FPGA fabric (for the detail architecture of such circuits, see, e.g., Chapter 4 and also Refs. 111, 114). This tool combines the simulated annealing placement, linear-time (with respect to number of gates in the circuit) global and detail routing. CMOL FPGA CAD is meant to provide a quick performance and defect tolerance assessment for CMOL FPGA circuits and should be easily expandable to include other CMOL-like circuit architectures.

Given a circuit in flat BLIF format [95] the full run of the program first performs a quasi-optimal placement and then routing with or without defects (Fig. C.1) using algorithms described in Section 4.2 of this Dissertation. The input BLIF file should contain only NOR gates, latches, and buffers; though, the latter are always removed in the preprocessing step. (Such technology mapping can be, e.g. accomplished with SIS [95] or ABC [3] tools.) The maximum allowed gate fanin and fanout in BLIF circuits should be less than the number of cells in a connectivity domain $M = (F_{\rm CMOS}/F_{\rm nano})^2 - 1$.

The other acceptable inputs to the tool include the map of cell and nanodevice defects, as well as placement, global routing, and detail routing files. Moreover, specifying one of the latter three files changes the point of entry to the flow, as shown in Fig. C.1. This can be used, e.g. to run Monte Carlo simulations enabling the defect tolerance simulations with respect to stuck-on-open-type defective nanodevices. Instead of performing placement and global routing for each trial in the simulation, one can use the same preexisting global routing file. The program flow in this example would be limited to just generation of map of bad nanodevices and then running detail routing algorithm.

The easiest way to create these additional files is to use the tool to generate them (Fig. C.1). If the step is successful, the CMOL FPGA CAD will generate the

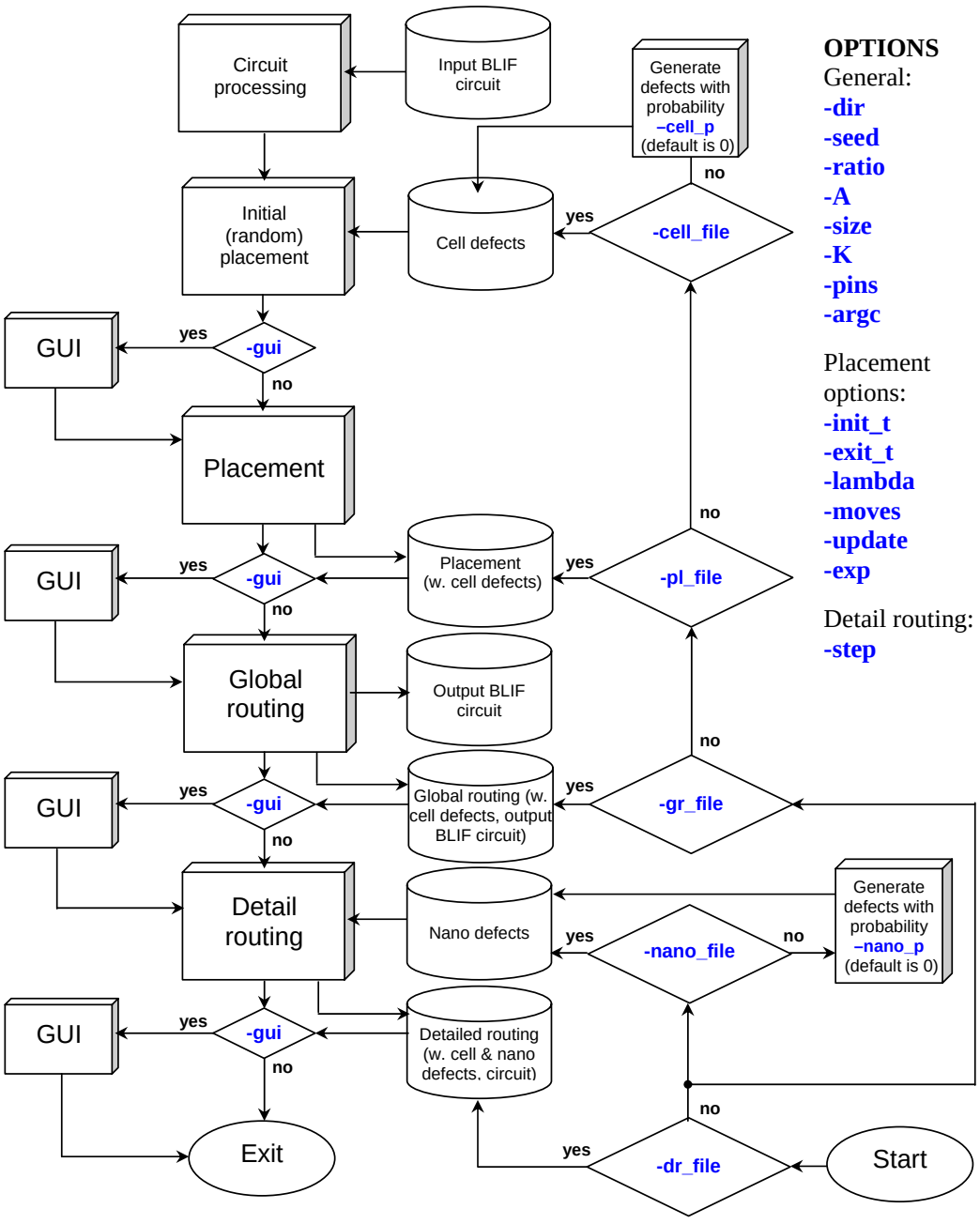

Figure C.1: General structure of CMOL FPGA CAD. All possible input and output files are shown with the cylinder-shaped boxes.

command line options	argc	blif	gui	dir	seed	ratio	A	size	K	pins	init_t	exit_t	lambda	moves	update	exp	step	cell_p	nano_p	cell_file	nano_file	pl_file	gr_file	dr_file
argc	-	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
blif	O	-	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	X	X
gui	O	O	-	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
dir	O	O	O	-	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
seed	O	O	O	O	-	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	X
ratio	O	O	O	O	O	-	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	X
A	O	O	O	O	O	O	-	O	O	O	O	O	O	O	O	O	O	O	O	O	O	X	X	X
size	O	O	O	O	O	O	O	-	X	O	O	O	O	O	O	O	O	O	O	O	O	X	X	X
K	O	O	O	O	O	O	O	X	-	O	O	O	O	O	O	O	O	O	O	O	O	X	X	X
pins	O	O	O	O	O	O	O	O	O	-	O	O	O	O	O	O	O	O	O	O	O	X	X	X
init_t	O	O	O	O	O	O	O	O	O	O	-	O	O	O	O	O	O	O	O	O	O	X	X	X
exit_t	O	O	O	O	O	O	O	O	O	O	O	-	O	O	O	O	O	O	O	O	O	X	X	X
lambda	O	O	O	O	O	O	O	O	O	O	O	O	-	O	O	O	O	O	O	O	O	X	X	X
moves	O	O	O	O	O	O	O	O	O	O	O	O	O	-	O	O	O	O	O	O	O	X	X	X
update	O	O	O	O	O	O	O	O	O	O	O	O	O	O	-	O	O	O	O	O	O	X	X	X
exp	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	-	O	O	O	O	O	X	X	X
step	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	-	O	O	O	O	O	O	X
cell_p	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	-	O	X	O	X	X	X
nano_p	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	-	O	X	O	O	X
cell_file	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	X	O	-	O	X	X	X
nano_file	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	X	O	-	O	O	X
pl_file	O	O	O	O	O	O	X	X	X	X	X	X	X	X	X	X	O	X	O	X	O	-	X	X
gr_file	O	X	O	O	O	O	X	X	X	X	X	X	X	X	X	X	O	X	O	X	O	X	-	X
dr_file	O	X	O	O	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	-

Figure C.2: Options compatibility. "O" means that two corresponding options can appear together, while "X" prohibits their simultaneous usage.

corresponding file automatically. To simplify usage, placement, global routing, and detail routing files contain the data from all previous steps.

The successfully mapped circuit can be functionally compared with initial one, e.g., using ABC tool [3]. To make it possible, tool generates automatically post-mapping BLIF circuit. Note that even though all the unconnected (floating) primary inputs or outputs are deleted in the preprocessing steps they are added back to the output BLIF circuit.

C.2 Command Line Options

a) Usage

To map circuit "*input.blif*" on CMOL FPGA fabric with default parameters in Linux or Windows systems simply run

cmolcad -blif *input.blif*

For other specific cases use the options below. Options may appear in any order, however not all options can be used simultaneously (Fig. C.2).

b) General Options

-A <int>: Linear size of a tile connectivity domain. Default is 9. Only odd numbers are allowed. Note that A could be made deliberately smaller (for better defect tolerance with respect to stuck-on-open-type nanodevices) than the maximum linear size of tile connectivity domain obtained from CMOL parameters $F_{\rm CMOS}$ and $F_{\rm nano}$ (see Eq. 4.3).

-K <int>: The number of basic cells (K) per tile allocated for logic gates. Default is 6. This option is ignored if array size option (**-size**) is specified. In the latter case, the total number of basic cells allocated for the logic gates in the whole CMOL FPGA array is exactly equal to the number of gates (excluding inverters) in the circuit. Moreover, such cells will be evenly distributed among tiles, but the actual number K maybe different (at most by one) from one tile to the other.

-pins <int>: Maximum number of input/output cells ($pin_{\rm max}$) which are available during mapping in each peripheral tile. Default is 4. The maximum number for this value is 16.

-ratio <int>: The ratio of CMOS and nano pitches (i.e. $F_{\rm CMOS}/F_{\rm nano}$). Default is 10. This ratio determines the linear size of the cell connectivity domain $a = \beta F_{\rm CMOS}/F_{\rm nano}$ which is used for the detail routing step.

-seed <int>: The random number generator seed (must be a negative integer number). This value can be used to change behavior of heuristic algorithm and defect generator routines.

-gui: Run program with GUI. Note that currently GUI is supported only under Linux system.

-argc <int>: The number of arguments passed to **cmolcad**. This option can be used when **cmolcad** with its arguments happens to be an argument itself for some execution script, e.g., parallel job launcher **mpiexec**.

-cell_p <float>: The probability of cells (including cells which comprise latch cells and input/output cells) being defective. Default is 0. The cells are distributed uniformly throughout the array. Note that the latch cell is considered operational as long as it has at least one not defective pair of input-output pins (or equivalently at least one cell out of four total).

-nano_p <float>: The probability of nanodevice being (stuck-on-open) defective. The nanodevices are distributed uniformly throughout the array. Default is 0.

-size <int>: The linear size of array ($Size$) measured in tiles, not including the peripheral tiles (which are used solely for input/output purposes). In case if this option is not specified, the array size is determined automatically from options **-K**, **-pins**, **-cell_p** and the circuit statistics. More specifically, if **-cell_p** is set to 0 it is equal to

$$Size = \lceil \max[(N_{\text{in}} + N_{\text{out}})/(4.0 pin_{\max}), (N_{\text{NOR}}/K))^{1/2}, (N_{\text{latch}})^{1/2}] \rceil, \quad \text{(C.1)}$$

where N_{in}, N_{out}, N_{NOR}, and N_{latch} are the number of primary inputs, primary outputs, NOR gates, and latches in the considered circuit, correspondingly. Otherwise, when **-cell_p** is not equal to 0, the array size will be the smallest possible one fitting the circuitry while having $12 - K$ routing cells in each tile on average.

-dir <char>: Target folder for all generated output files.

-cell_file <char>: The name of the file with cell defect map. If this option is specified then option **-cell_p** will be ignored. Note that the file is allowed to have defect map for a larger CMOL FPGA array size than the current one in the program. However, defect map for smaller than the current array size will result in error.

-nano_file <char>: The name of the file with nanodevice defect map. If this option is specified then option **-nano_p** will be ignored.

-pl_file <char>: The name of the file with placement. The placement file also contains cell defect map, so that option **-cell_file** will be ignored.

-gr_file <char>: The name of the file with global routing. The global routing file also contains the final circuit netlist in BLIF format and cell defect map, so that option **-cell_file** and **-blif** options will be ignored.

-dr_file <char>: The name of the file with detail routing. The detail routing file also contains the final circuit netlist in BLIF format, cell defect map, and nanodefect map, so that options **-cell_file**, **-nano_file**, **-blif** will be ignored.

c) Placement Options

-init_t <float>: The initial temperature for the simulated annealing placement. Currently, the default value, calculated similarly to Ref. 10, is hardly optimal so that choosing a proper initial temperature may improve placement rather dramatically.

-exit_t <float>: The final temperature for the simulated annealing placement. The default value, calculated similarly to Ref. 10, is so far not optimal.

-lambda <float>: Tradeoff variable distributing weight between timing and wiring cost. Default is 0.5. For more details, see Ref. 78.

-moves <int>: The exponent (*moves*) defining the number of moves to be attempted before temperature update during annealing. The actual number of moves is calculated as $(N_{\mathrm{NOR}} + N_{\mathrm{latch}} + N_{\mathrm{in}} + N_{\mathrm{out}})^{moves}$. Default is 10.

-update <int>: The interval (in terms of the number of accepted moves or swaps) for the time analysis, i.e. updating the slack for each gate. Default is 10000. For more details, see Ref. 78.

-exp <int>: Criticality exponent used in timing cost calculation. Default is 3. For more details, see Ref. 78.

e) Detail Routing Options

-step <int>: The reciprocal of this parameter defines the bucket size for quasi-sorting algorithm in detail routing step. Default is 100. For example, for the default value the size of the bucket is 0.01 and the total number of buckets is 99, since the probability of finding defect free location ranges from 0 to 1. (For more details, see, e.g., Section 4.2.6.)

Bibliography

[1] *International Technology Roadmap for Semiconductors. 2005 Edition.* available online at http://public.itrs.net/.

[2] *FPGA place-and-route challenge.* 1999. Available online at http://www.eecg.toronto.edu/∼vaughn/challenge/ challenge.html/.

[3] *ABC: A System for Sequential Synthesis and Verification.* 2005. available online at http://www.eecs.berkeley.edu/∼alanmi/abc/.

[4] *CMOL FPGA CAD: An interactive tool for mapping sequential circuits on CMOL FPGA.* 2006. available online at http://rsfq1.physics.sunysb.edu/∼likharev/cmolfpgacad/.

[5] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Trans. VLSI*, 12(3):288–298, 2004.

[6] H. B. Akkerman, P. W. M. Blom, D. M. de Leeuw, and B. de Boer. Towards molecular electronics with large-area molecular junctions. *Nature*, 441(7089):69–72, 2006.

[7] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes, G. Snider, and L. Albertson. Plasma: An FPGA for million gate systems. In *FPGA*, pages 10–16, 1996.

[8] C. J. Amsinck, N. H. Di Spigna, D. P. Nackashi, and P. D. Franzon. Scaling constraints in nanoelectronic random-access memories. *Nanotechnology*, 16(10):2251–2260, 2005.

[9] I. G. Baek, D. C. Kim, M. J. Lee, H.-J. Kim, E. K. Yim, M. S. Lee, J. E. Lee, S. E. Ahn, S. Seo, J. H. Lee, J. C. Park, Y. K. Cha, S. O. Park, H. S. Kim, I. K. Yoo, U.-I. Chung, J. T. Moon, and B. I. Ryu. Multi-layer cross-point binary-oxide resistive memory (oxram) for post-nand storage applications. *IEDM Tech. Digest*, page 31.4, 2005.

[10] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for deep-submicron FPGAs*. Kluwer Int. Series in Eng. and Comp. Science 497. Kluwer Academic, Boston; London, 1999.

[11] R. E. Blahut. *Algebraic Codes for Data Transmission*. Cambridge University Press, Cambridge, 2003.

[12] G. Borriello, C. Ebeling, S. A. Hauck, and S. Burns. The Triptych FPGA architecture. *IEEE Trans. VLSI*, 3(4):491–501, 1995.

[13] L. D. Bozano, B. W. Kean, V. R. Deline, J. R. Salem, and J. C. Scott. Mechanism for bistability in organic memory elements. *Appl. Phys. Lett.*, 84(4):607–609, 2004.

[14] W. D. Brown and J. Brewer. *Nonvolatile semiconductor memory technology : a comprehensive guide to understanding and to using NVSM devices*. IEEE Press series on microelectronic systems. IEEE Press, New York, 1998.

[15] S. R. J. Brueck. There are no fundamental limits to optical lithography. In *International Trends in Applied Optics*, pages 85–109. SPIE Press, Bellingham, WA, 2002.

[16] H. T. Bui, A. K. Al-Sheraidah, A., and Y. Wang. New 4-transistor XOR and XNOR designs. In *Proc. AP-ASIC-2000*, pages 25–28, 2000.

[17] G. F. Cerofolini. Search for realistic limits to computation. ii. the technological side. *Appl. Phys. A-Mater. Sci. Process.*, 2006. in press.

[18] K. Chakraborty and P. Mazumder. *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories*. Prentice Hall, Upper Saddle River, NJ, 2002.

[19] A. Chen, S. Haddad, Y.-C. Wu, T.-N. Fang, Z. Lan, S. Avanzino, S. Pangrle, M. Buynoski, M. Rathor, W. Cai, N. Tripsas, C. Bill, M. VanBuskirk, and M. Taguchi. Non-volatile resistive switching for advanced memory applications. *IEDM Tech. Digest*, page 31.3, 2005.

[20] Y. Chen, G. Y. Jung, D. A. A. Ohlberg, X. M. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14(4):462–468, 2003.

[21] Y.-C. Chen, C. F. Chen, C. T. Chen, J. Y. Yu, S. Wu, S. L. Lung, R. Liu, and C.-Y. Lu. An access-transistor-free (0T/1R) non-volatile resistance random access memory (RRAM) using a novel threshold switching, self-rectifying chalcogenide device. *IEDM Tech. Digest*, page 37.4.1, 2003.

[22] C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams, and J. R. Heath. Electronically configurable molecular-based logic gates. *Science*, 285(5426):391–394, 1999.

[23] S. Das, G. Rose, M. M. Ziegler, C. A. Picconatto, and J. E. Ellenbogen. Architecture and simulations for nanoprocessor systems integrated on the molecular scale. In G. Cuniberti, G. Fagas, and K. Richter, editors, *Introducing Molecular Electronics*, pages 479–515. Springer, Berlin, 2005.

[24] J. A. Davis, V. K. De, and J. D. Meindl. A stochastic wire-length distribution for gigascale integration (gsi) - part i: Derivation and validation. *IEEE Trans. Electron Devices*, 45(3):580–589, 1998.

[25] A. DeHon. Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% LUT utilization). In *Proc. of FPGA*, pages 69–78, Monterey, CA, Febrary 1999.

[26] A. DeHon. Array-based architecture for fet-based, nanoscale electronics. *IEEE Trans. Nanotechnol.*, 2(1):23–32, 2003.

[27] A. DeHon. Design of programmable interconnect for sublithographic programmable logic arrays. In *Proc. of FPGA*, pages 127–137, Monterey, CA, Febrary 2005.

[28] A. DeHon. Nanowire-based programmable architectures. *ACM J. Emerg. Technol. Comput. Syst.*, 1(2):109–162, 2005.

[29] A. DeHon, S. C. Goldstein, P. J. Kuekes, and P. Lincoln. Nonphotolithographic nanoscale memory density prospects. *IEEE Trans. Nanotechnol.*, 4(2):215–228, 2005.

[30] A. DeHon and K. Likharev. Hybrid CMOS/nanoelectronic digital circuits: Devices, architectures, and design automation. In *Proc. of ICCAD*, pages 375–382, 2005.

[31] A. DeHon, P. Lincoln, and J. E. Savage. Stochastic assembly of sublithographic nanoscale interfaces. *IEEE Trans. Nanotechnol.*, 2(3):165–174, 2003.

[32] A. DeHon and H. Naeimi. Seven strategies for tolerating highly defective fabrication. *IEEE Des. Test Comput.*, 22(4):306–315, 2005.

[33] N. H. Di Spigna, D. P. Nackashi, C. J. Amsinck, S. R. Sonkusale, and P. Franzon. Deterministic nanowire fanout and interconnect without any critical translation alignment. *IEEE Trans. Nanotechnol.*, 5(4):356–361, 2006.

[34] P. D. Dresselhaus, L. Ji, S. Y. Han, J. E. Lukens, and K. K. Likharev. Measurement of single-electron lifetimes in a multijunction trap. *Phys. Rev. Lett.*, 72(20):3226–3229, 1994.

[35] S. Fölling, Ö. Türel, and K. K. Likharev. Single-electron latching switches as nanoscale synapses. In *Proc. of IJCNN'01*, pages 216–221, Mount Royal, NY, 2001. Int. Neural Network Soc.

[36] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H. S. P. Wong. Device scaling limits of Si MOSFETs and their application dependencies. *Proc. IEEE*, 89(3):259–288, 2001.

[37] A. A. Gayasen, N. Vijaykrishnan, and M. J. Irwin. Exploring technology alternatives for nano-scale FPGA interconnects. In *Proc. of DAC*, pages 921–926, June 2005.

[38] L. I. Glazman and K. A. Matveev. Residual quantum conductivity under coulomb-blockade conditions. *Jetp Letters*, 51(8):484–487, 1990.

[39] B. Gojman, E. Rachlin, and J. E. Savage. Evaluation of design strategies for stochastically assembled nanoarrays. *ACM J. Emerg. Technol. Comput. Syst.*, 1(2):73–108, 2005.

[40] S. C. Goldstein and M. Budiu. NanoFabrics: Spatial computing using molecular electronics. In *Proc. of Int. Symp. on Computer Architectures*, pages 178–189, Göteborg, Sweden, June 2001.

[41] S. C. Goldstein and D. Rosewater. Digital logic using molecular electronics. In *Proc. of IEEE Int. Solid-State Circuits Conf.*, page 12.5, San Francisco, CA, February 2002.

[42] L. J. Guo. Recent progress in nanoimprint technology and its applications. *J. Phys. D-Appl. Phys.*, 37(11):R123–R141, 2004.

[43] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280(5370):1716–1721, 1998.

[44] S. Heo, R. Krashinsky, and K. Asanović. Activity-sensitive flip-flop and latch selection for reduced energy. In *Proc. of Conf. on Advanced Research in VLSI*, Salt Lake City, UT, March 2001.

[45] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proc. IEEE*, 89(4):490–504, 2001.

[46] C. T. Huang, C. F. Wu, J. F. Li, and C. W. Wu. Built-in redundancy analysis for memory yield improvement. *IEEE Trans. Reliab.*, 52(4):386–399, 2003.

[47] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. Annals of discrete mathematics; 53. North-Holland, Amsterdam; London, 1992.

[48] G.-L. Ingold and Y. V. Nazarov. Charge tunneling rates in ultrasmall junctions. In H. Grabert and M. H. Devoret, editors, *Single Charge Tunneling*, volume 294, pages 21–107. Plenum Press, New York, 1992.

[49] K. L. Jensen. Field emitter arrays for plasma and microwave source applications. *Physics of Plasmas*, 6(5):2241–2253, 1999.

[50] L. Ji, P. D. Dresselhaus, S. Y. Han, K. Lin, W. Zheng, and J. E. Lukens. Fabrication and characterization of single-electron transistors and traps. *J. Vac. Sci. Technol. B*, 12(6):3619–3622, 1994.

[51] G.-Y. Jung, E. Johnston-Halperin, W. Wu, Z. Yu, S.-Y. Wang, W. Tong, Z. Li, J. E. Green, B. A. Sheriff, A. Boukai, Y. Bunimovich, J. R. Heath, and R. S. Williams. Circuit fabrication at 17 nm half-pitch by nanoimprint lithography. *Nano Lett.*, 2(3):351–354, 2006.

[52] C. Kittel. *Introduction to solid state physics*. John Wiley & Sons Inc., Hoboken, N.J., 2005.

[53] P. M. Kogge and H. S. Stone. Parallel algorithm for efficient solution of a general class of recurrence equations. *IEEE Trans. Comput.*, C-22(8):786–793, 1973.

[54] J. Kouloheris and A. E. Gamal. PLA-based FPA versus cell granularity. In *Proc. of Custom Integrated Circuits Conf.*, pages 4.3.1–4, Boston, MA, May 1992.

[55] S. Kubatkin, A. Danilov, M. Hjort, J. Cornil, J. L. Bredas, N. Stuhr-Hansen, P. Hedegard, and T. Bjornholm. Single-electron transistor of a single organic molecule with access to several redox states. *Nature*, 425(6959):698–701, 2003.

[56] P. J. Kuekes, W. Robinett, R. M. Roth, G. Seroussi, G. S. Snider, and R. S. Williams. Resistor-logic demultiplexers for nanoelectronics based on constant-weight codes. *Nanotechnology*, 17(4):1052–1061, 2006.

[57] P. J. Kuekes, W. Robinett, G. Seroussi, and R. S. Williams. Defect-tolerant demultiplexers for nano-electronics constructed from error-correcting codes. *App. Phys. A-Mater. Sci. Process.*, 80(6):1161–1164, 2005.

[58] P. J. Kuekes, G. S. Snider, and R. S. Williams. Crossbar nanocomputers. *Sci. Am.*, 293(5):72–80, 2005.

[59] P. J. Kuekes, D. R. Stewart, and R. S. Williams. The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits. *J. Appl. Phys.*, 97(3):034301, 2005.

[60] M. Kund, G. Beitel, C.-U. Pinnow, T. Röhr, J. Schumann, R. Symanczyk, K.-D. Ufert, and G. Müller. Conductive bridging ram (cbram). *IEDM Tech. Digest*, page 31.5, 2005.

[61] Y. S. Lai, C. H. Tu, D. L. Kwong, and J. S. Chen. Bistable resistance switching of poly(n-vinylcarbazole) films for nonvolatile memory applications. *Appl. Phys. Lett.*, 87(12):122101, 2005.

[62] H. Lee. High-speed VLSI architecture for parallel Reed-Solomon decoder. *IEEE Trans. VLSI*, 11(2):288–294, 2003.

[63] J. H. Lee and K. K. Likharev. CMOL CrossNets as pattern classifiers. In *Proc. of Int. Work-Conf. on Artificial Neural Networks*, pages 446–454, Barcelona, Spain, June 2005.

[64] J. H. Lee and K. K. Likharev. Defect-tolerant nanoelectronic pattern classifiers. *submitted to Int. J. of Circuit Theory and Applications*, 2006. available online at http://rsfq1.physics.sunysb.edu/∼likharev/nanno/IJCTA06.pdf.

[65] J. H. Lee, X. Ma, D. B. Strukov, and K. K. Likharev. CMOL. In *Proc. of NANOARCH*, pages 3.9–3.16, Palm Springs, CA, May 2005.

[66] C. Li, D. H. Zhang, X. L. Liu, S. Han, T. Tang, C. W. Zhou, W. Fan, J. Koehne, J. Han, M. Meyyappan, A. M. Rawlett, D. W. Price, and J. M. Tour. Fabrication approach for molecular memory arrays. *App. Phys. Lett.*, 82(4):645–647, 2003.

[67] K. K. Likharev. CMOL technology: Possible roadmap. document in preparation.

[68] K. K. Likharev. Single-electron devices and their applications. *Proc. IEEE*, 87(4):606–632, 1999.

[69] K. K. Likharev. Riding the crest of a new wave in memory. *IEEE Circuits Devices*, 16(4):16–21, 2000.

[70] K. K. Likharev. Electronics below 10 nm. In *Nano and Giga Challenges in Microelectronics*, pages 27–68. Elsevier, Amsterdam, 2003.

[71] K. K. Likharev. CMOL: A silicon-based bottom-up approach to nanoelectronics. *Interface*, 14:43–46, 2005.

[72] K. K. Likharev, A. Mayr, I. Muckra, and Ö. Türel. CrossNets - High-performance neuromorphic architectures for CMOL circuits. *Ann. NY Acad. Sci.*, 1006:146–163, 2003.

[73] K. K. Likharev and D. B. Strukov. CMOL: Devices, circuits, and architectures. In G. Cuniberti, G. Fagas, and K. Richter, editors, *Introducing Molecular Electronics*, pages 447–478. Springer, Berlin, 2005.

[74] R. J. Luyken and F. Hofmann. Concepts for hybrid CMOS-molecular non-volatile memories. *Nanotechnology*, 14(2):273–276, 2003.

[75] L. P. Ma, J. Liu, and Y. Yang. Organic electrical bistable devices and rewritable memory cells. *Appl. Phys. Lett.*, 80(16):2997–2999, 2002.

[76] X. Ma, D. B. Strukov, J. H. Lee, and K. K. Likharev. Afterlife for silicon: CMOL circuit architectures. In *Proc. of IEEE Conf. on Nanotechnol.*, pages 175–178, Nagoya, Japan, July 2005.

[77] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens. Challenges and future directions for the scaling of dynamic random-access memory (dram). *IBM J. Res. Dev.*, 46(2-3):187–212, 2002.

[78] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *Proc. of FPGA*, pages 203–213, New York, NY, USA, 2000.

[79] M. Masoumi, F. Raissi, M. Ahmadian, and P. Keshavarzi. Design and evaluation of basic standard encryption algorithm modules using nanosized complementary metal-oxide-semiconductor-molecular circuits. *Nanotechnology*, 17(1):89–99, 2006.

[80] E. D. Mastrovito. *VLSI architectures for computation in Galois fields*. Linkoping University, Linkoping, Sweden, 1995. PhD thesis.

[81] T. K. Matsushima, T. Matsushima, and S. Hirasawa. Parallel encoder and decoder architecture for cyclic codes. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E79A(9):1313–1323, 1996.

[82] C. Mead and L. Conway. *Introduction to VLSI systems*. Addison-Wesley, Reading, MA; London, 1980.

[83] K. Nabors and J. White. Fastcap - a multipole accelerated 3-d capacitance extraction program. *IEEE Trans. Comput-Aided Des. Integr. Circuits Syst.*, 10(11):1447–1459, 1991.

[84] M. A. Neifeld and J. D. Hayes. Error-correction schemes for volume optical memories. *Applied Optics*, 34(35):8183–8191, 1995.

[85] C. Paar. *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields*. Univ. of Essen, Essen, Germany, 1995. PhD thesis, Inst. for Experimental Math.

[86] C. Paar, P. Fleischmann, and P. Soria-Rodriguez. Fast arithmetic for public-key algorithms in Galois fields with composite exponents. *IEEE Trans. Comput.*, 48(10):1025–1034, 1999.

[87] K. Padalia, R. Fung, M. Bourgeault, A. Egier, and J. Rose. Automatic transistor and physical design of FPGA tiles from an architectural specification. In *Proc. of FPGA*, pages 164–172, 2003.

[88] J. Park, A. N. Pasupathy, J. I. Goldsmith, C. Chang, Y. Yaish, J. R. Petta, M. Rinkoski, J. P. Sethna, H. D. Abruna, P. L. McEuen, and D. C. Ralph. Coulomb blockade and the Kondo effect in single-atom transistors. *Nature*, 417(6890):722–725, 2002.

[89] D. Porath. DNA-based devices. In G. Cuniberti, G. Fagas, and K. Richter, editors, *Introducing Molecular Electronics*, pages 411–446. Springer, Berlin, 2005.

[90] B. Prince. *Semiconductor Memories: A Handbook of Design, Manufacture, and Application*. Wiley, Chichester, 2nd edition, 1991.

[91] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Pearson Education, Upper Saddle River, NJ, 2nd edition, 2003.

[92] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor. The rectilinear steiner arborescence problem. *Algorithmica*, 7(2-3):277–288, 1992.

[93] S. Roy and V. Beiu. Majority multiplexing-economical redundant fault-tolerant designs for nanoarchitectures. *IEEE Trans. Nanotechnol.*, 4(4):441–451, 2005.

[94] D. V. Sarwate and N. R. Shanbhag. High-speed architectures for Reed-Solomon decoders. *IEEE Trans. VLSI*, 9(5):641–655, 2001.

[95] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, 1992.

[96] R. Sezi, A. Walter, R. Engl, A. Maltenberger, J. Schumann, M. Kund, and C. Dehm. Organic materials for high-density non-volatile memory applications. *IEDM Tech. Digest*, page 10.2.1, 2003.

[97] M. She. *Semiconductor Flash Memory Scaling*. Univ. of California, Berkeley, 2003. PhD thesis.

[98] J. G. Simmons and Verderbe.Rr. New conduction and reversible memory phenomena in thin insulating films. *Proc. of the Royal Society of London Series a-Mathematical and Physical Sciences*, 301(1464):77–&, 1967.

[99] G. Snider. Computing with hysteretic resistor crossbars. *App. Phys. A-Mater. Sci. Process.*, 80(6):1165–1172, 2005.

[100] G. Snider, P. Kuekes, T. Hogg, and R. S. Williams. Nanoelectronic architectures. *App. Phys. A-Mater. Sci. Process.*, 80(6):1183–1195, 2005.

[101] G. Snider, P. Kuekes, and R. S. Williams. CMOS-like logic in defective, nanoscale crossbars. *Nanotechnology*, 15(8):881–891, 2004.

[102] G. Snider and R. S. Williams. Nano/CMOS architectures using field-programmable nanowire interconnect. preprint.

[103] H. H. Solak, C. David, J. Gobrecht, V. Golovkina, F. Cerrina, S. O. Kim, and P. F. Nealey. Sub-50 nm period patterns with EUV interference lithography. *Microelectron. Eng.*, 67-8:56–62, 2003.

[104] P. P. Sotiriadis. Information capacity of nanowire crossbar switching networks. *IEEE Trans. Inf. Theory*, 52(7):3019–3032, 2006.

[105] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler. Molecular electronics: From devices and interconnect to circuits and architecture. *Proc. IEEE*, 91(11):1940–1957, 2003.

[106] C. H. Stapper and H. S. Lee. Synergistic fault-tolerance for memory chips. *IEEE Trans. Comput.*, 41(9):1078–1087, 1992.

[107] D. B. Strukov. The area and latency tradeoffs of binary bit-parallel bch decoders for the prospective nanoelectronic memories. In *Proc. Asilomar Conf. on Signals, Systems and Computers*, Asilomar, CA, 2006. in print.

[108] D. B. Strukov. Defect tolerance in digital CMOL circuits. 2006. in preparation.

[109] D. B. Strukov and K. K. Likharev. CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, 16(6):888–900, 2005.

[110] D. B. Strukov and K. K. Likharev. Prospects for terabit-scale nanoelectronic memories. *Nanotechnology*, 16:137–148, 2005.

[111] D. B. Strukov and K. K. Likharev. CMOL FPGA circuits. In *Proc. of Int. Conf. on Computer Design, CDES'2006*, pages 213–219, Las Vegas, NE, June 2006.

[112] D. B. Strukov and K. K. Likharev. Defect tolerant architectures for nanoelectronic crossbar memories. *J. Nanosci. Nanotechnol.*, 2006. in press.

[113] D. B. Strukov and K. K. Likharev. Defect-tolerant CMOL memories. In *Proc. of NANOARCH*, pages 1–8, Boston, MA, June 2006.

[114] D. B. Strukov and K. K. Likharev. A reconfigurable architecture for hybrid CMOS/Nanodevice circuits. In *Proc. of FPGA*, pages 131–140, New York, NY, 2006. ACM Press.

[115] D. B. Strukov and K. K. Likharev. A reconfigurable CMOL DSP circuit. 2006. in preparation.

[116] F. Sun and T. Zhang. Two fault tolerant design approaches for hybrid CMOS/nanodevice digital memories. In *Proc. of NANOARCH*, pages 9–16, Boston, MA, June 2006.

[117] I. E. Sutherland, R. F. Sproull, and D. Harris. *Logical effort: Designing fast CMOS circuits*. Morgan Kaufmann Publishers, San Francisco, CA; London, 1999.

[118] V. A. Sverdlov, T. J. Walls, and K. K. Likharev. Nanoscale silicon mosfets: A theoretical study. *IEEE Trans. Electron Devices*, 50(9):1926–1933, 2003.

[119] M. Tom and G. Lemieux. Logic block clustering of large designs for channel-width constrained FPGAs. In *Proc. of DAC*, pages 726–731, San Diego, CA, June 2005.

[120] C. M. S. Torres, S. Zankovych, J. Seekamp, A. P. Kam, C. C. Cedeno, T. Hoffmann, J. Ahopelto, F. Reuther, K. Pfeiffer, G. Bleidiessel, G. Gruetzner, M. V. Maximov, and B. Heidari. Nanoimprint lithography: An alternative nanofabrication approach. *Mater. Sci. Eng. C-Biomimetic Supramol. Syst.*, 23(1-2):23–31, 2003.

[121] J. M. Tour. *Molecular Electronics: Commercial Insights, Chemistry, Devices, Architecture and Programming*. World Scientific, Singapore; River Edge, NJ, 2003.

[122] O. Türel, J. H. Lee, X. L. Ma, and K. K. Likharev. Neuromorphic architectures for nanoetectronic circuits. *Int. J. Circ. Theory App.*, 32(5):277–302, 2004.

[123] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In G. Cuniberti, G. Fagas, and K. Richter, editors, *Automata Studies*, pages 329–378. Princeton Univeristy Press, Princeton, NJ, 1956.

[124] D. J. Wagner and A. H. Jayatissa. Nanoimprint lithography: Review of aspects and applications. In W. Y. Lai, L. E. Ocola, and S. Pau, editors, *Nanofabrication: Technologies, Devices, and Applications II*, volume 6002, page 60020R. SPIE, 2005.

[125] T. Wang, Z. Qi, and C. A. Moritz. Opportunities and challenges in application-tuned circuits and architectures based on nanodevicess. In *Proc. of Conf. on Computing Frontiers*, pages 503 – 511, Italy, April 2004.

[126] W. Wang, T. Lee, and M. Reed. Intrinsic electronic conduction mechanisms in self-assembled monolayers. In G. Cuniberti, G. Fagas, and K. Richter, editors, *Introducing Molecular Electronics*, pages 275–300. Springer, Berlin, 2005.

[127] W. Wu, G. Y. Jung, D. L. Olynick, J. Straznicky, Z. Li, X. Li, D. A. A. Ohlberg, Y. Chen, S. Y. Wang, J. A. Liddle, W. M. Tong, and R. S. Williams. One-kilobit cross-bar molecular memory circuits at 30-nm half-pitch fabricated by nanoimprint lithography. *App. Phys. A-Mater. Sci. Process.*, 80(6):1173–1178, 2005.

[128] N. B. Zhitenev, W. Jiang, A. Evbe, Z. Bao, E. Garfunkel, D. M. Tennant, and R. Cirelli. Control of topography, stres, and diffusion of molecule-metal interface. *Nanotechnology*, 17(5):1272–1277, 2006.

[129] N. B. Zhitenev, H. Meng, and Z. Bao. Conductance of small molecular junctions. *Phys. Rev. Lett.*, 88(22):226801, 2002.

[130] M. M. Ziegler and M. R. Stan. CMOS/nano co-design for crossbar-based molecular electronic systems. *IEEE Trans. Nanotechnol.*, 2(4):217–230, 2003.