DESIGN, FAULT STUDIES, AND PERFORMANCE ANALYSIS FOR
NANOMAGNET LOGIC PLAS AND OTHER NANOMAGNET LOGIC
ARCHITECTURES

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Michael S. Crocker

_________________________________________
Xiaobo Sharon Hu, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
July 2011
DESIGN, FAULT STUDIES, AND PERFORMANCE ANALYSIS FOR
NANOMAGNET LOGIC PLAS AND OTHER NANOMAGNET LOGIC
ARCHITECTURES

Abstract
by
Michael S. Crocker

In order to continue the performance and scaling trends that we have come
to expect from Moore’s Law, many emergent computational models, devices, and
technologies are actively being studied to either replace or augment CMOS tech-
nology. Nanomagnet Logic (NML) is one such alternative. NML is a device archi-
tecture that utilizes the magnetization of nano-scale magnets to perform logical
operations. NML has been experimentally demonstrated and operates at room
temperature.

We present an NML Programmable Logic Array (PLA) based on a previously
proposed reprogrammable Quantum-dot Cellular Automata PLA design. We also
discuss the fabrication and simulation validation of the circuit structures unique
to the NML PLA. We present a fault model for NML PLAs, and provide an im-
proved technique for mapping logic to the NML PLA. We present area, energy, and
delay estimates for the NML PLA, compare the area of NML PLAs to other re-
programmable nanotechnologies, and analyze how architectural-level redundancy
will affect performance and defect tolerance in NML PLAs.

Because the nanomagnets are non-volatile, as data flows through a circuit, it
is inherently pipelined. This feature makes NML an excellent fit for systolic architectures, which could enable low-power, high-throughput systems that can address a variety of application-level tasks. When considering possible NML systolic systems, the underlying systolic clocking scheme affects both architectural design and performance. We explore these issues in the context of two NML designs for simplified convolution, both of which focus on different dataflow patterns. We also study N-bit NML adders and multipliers in the context of high-throughput NML systems. We compare parallel and serial NML designs and compare NML to CMOS in terms of delay and energy. We also explore possible performance enhancement techniques for NML in an effort to make NML as low energy as possible.
FIGURES

2.1 Schematics of the fundamental structures needed to build QCA circuits: (a) basic device, (b) wire, (c) logic gate, (d) inverter, and (e) crossover. ................................. 10

2.2 Cartoon representations of a wire segment (a), a majority gate (b), and a crossover (c). Wire segments have been experimentally demonstrated (d) as have majority gates (e). Crossovers function correctly in simulation (f). ................................. 12

2.3 (a) Cartoon representations of how magnets are nulled by a hard axis magnetic field. (i) Magnets are initially in an antiferromagnetically-ordered state, and (ii) as the magnetic clocking field increases, the magnetization of the devices turns toward the horizontal, overcoming the energy barrier between states, so that (iii) when the field is removed the magnets can settle into a new antiferromagnetically-ordered state. (b) Cartoon drawing of the clocking wire structure underneath the layer of the magnets. As is indicated by the inset text at the top of the figure, all parallel NML clocking wires would require insulating oxides between the wires (which is not shown graphically). ................................. 13

2.4 Illustration of 3-phase and 2-phase clocking schemes. .............. 16

2.5 (a) QCA AND Plane Cell Schematic, (b) AND Plane Cell Layout, (c) QCA OR Plane Cell Schematic, (d) AND Plane Cell Regions. Note in (a) and (c) that the circle with a dot inside indicates an AND gate, and the circle with a plus sign inside indicates an OR gate. ........... 17

2.6 A QCA PLA that implements the Boolean functions ABC+BCD and ABC+AD ........................................ 18

3.1 (a) NML AND plane cell schematic with vertical implicant wire, (b) NML AND plane cell magnet layout with vertical implicant wire, (c) NML AND plane cell schematic with zig-zag implicant wire, (d) NML AND plane cell magnet layout with zig-zag implicant wire. 21
3.2 (a) Two and four magnet vertical wires (with helper cells); (b) A plot of the angle of magnetization versus applied field for 2 and 4 magnet vertical wires. The results indicate that lower fields are required to put the devices in the shorter line into a metastable state.

3.3 Examples of a 2x2 AND Plane array using: (a) vertical AND plane cell schematic and the (b) zig-zag AND plane cell schematic. Arrows show the propagation of literal and implicant signals though the array structure.

3.4 (a) An example 3x3 AND plane array and 3x3 OR plane array to illustrate a timing problem that arises if the AND and OR planes are placed adjacent to each other. (b) NML OR plane cell schematic similar to the zig-zag AND plane cell, but with AND and OR gates switched. (c) An example 3x3 AND plane array and 3x3 OR plane array to illustrate the routing overhead needed if the AND and OR planes are oriented the same way.

3.5 (a) OR plane cell schematic that utilizes zig-zag propagation of the implicant signal; (b) OR plane cell magnet layout; (c) An example NML PLA structure programmed to implement the Boolean functions ABC+BCD and ABC+AD.

3.6 An example NML PLA system of 5 PLAs with NML interconnect.

3.7 Illustration of how a signal can be routed through the NML-based interconnect between two levels of NML PLAs. An output function signal exits a PLA (top left) and enters the interconnection network. By programming cells in the network, the signal can be routed down the interconnect structure and then into the next PLA (bottom right).

3.8 (a) Illustration of the operation necessary to program the select bit in an AND plane cell. The signals must be driven backwards through the cell, from right to left. (b) Schematic of the magnet ensemble simulated to determine if required functionality is achievable.

3.9 (a) Magnet layout of the simulated 1:2 fanout structure (magnets are numbered); (b) time evolution of magnets propagating the signal vertically (magnets 4,5,6), and (c) time evolution of magnets propagating the signal horizontally (magnets 4,7,8,9).
3.10 (a) The effects of coercivity are shown by plotting the hysteresis loop for magnets of different aspect ratios. The longer magnet (dashed line) requires a larger applied field to switch its state than the shorter magnet requires (solid line). Such a magnet could be used to save the binary value in the select bit wire of the PLA cell. (b) A possible select bit structure: a wire of 6 magnets with helper magnets on the left end. The magnets are numbered for easier reference. (c) The time evolution of magnets in the select bit structure, showing how a signal can be driven into structure from an input on the right. (d) The time evolution of magnets in the select bit structure, showing how the binary value programmed into the wire can be used multiple times, since the magnets are not changed with an applied clocking field.

4.1 Possible defects: (a) stray charge, (b) shifted device, (c) rotated device, (d) missing device, (e) misshapen device.

4.2 Fault probability and yield versus device defect rate (DDR) with the cutoff defect rate shown as a vertical line. The PLA AND plane has 20 columns and 20 rows.

4.3 This plot shows how increasing the input size in the AND plane reduces the allowed device defect rate (DDR) for the same yield. The AND plane has 10 rows.

4.4 This graph shows the number of actual PLA rows required to implement a certain number of implicants as a function of the device defect rate (DDR). The AND plane has 20 columns.

4.5 (a) Experimentally realized magnets can have slanted edges. (b) This SEM image shows wires made using EBL. Proximity effects and other variations have led to bulging between the two wires. The lithography design called for 25nm between wires, but the fabricated wires are much closer. Also, note that the vertical wire bulges most significantly where it is adjacent to the horizontal wire.

4.6 $H_{\text{clock}}$ vs. $M_y$ for a 60x90nm magnet with no fabrication variation and a 60x90nm magnet with a “slanted” edge. $H_{\text{clock}}$ required to null the slanted magnet is greater than that for the perfect magnet.

4.7 $H_{\text{clock}}$ vs. $M_y$ for the 4th magnet in a 7-cell line. A larger $H_{\text{clock}}$ is required to facilitate a $\downarrow$ to $\uparrow$ transition than a $\uparrow$ to $\downarrow$ transition when compared to a wire with no missing magnetic material.

4.8 With more missing material, it becomes more difficult to null the nanomagnet as evidenced by the y-component of the defective magnet’s magnetization.
4.9 If $H_{\text{clock}}$ is applied to a magnet with missing magnetic material from right- to-left, a $\uparrow$ to $\downarrow$ transition becomes more difficult.

4.10 (a) $M_y$ vs. $H_{\text{clock}}$ for a 60x90nm magnet and a 65x90nm magnet. As the aspect ratio decreases, the magnet is easier to null. (b) Fields generated by various magnets.

4.11 $M_y$ as a function of $H_{\text{clock}}$ and $M_y$ as a function of $H_{\text{local}}$ in the presence of a $1.5 \times 10^4$ A/m biasing field. Note that the field seen at the field near the surface of the magnet is greater than the applied field. This will be an important measure when considering why a circuit does or does not show logically correct behavior.

4.12 Local field vs. $M_y$ for magnets in a wire. Note that magnet needs to see a similar local field to null but at the expense of a larger global field.

4.13 A 2x2 section of the AND plane of an NML PLA showing three faults.

5.1 Logic Graph Representation. The logical functions $S = WX + XZ$ and $T = WX + WY'$ are represented in graph form.

5.2 (a) A defective and redundant nanowire crossbar structure. The AND plane is 5x5 cells, and the OR plane is 3x5 cells. There are 5 faults (2 stuck open (X), 1 stuck closed (square) and 2 broken nanowires). (b) Crossbar Structure Graph Representation. Vertices and edges have been removed based on the location and types of PLA crossbar faults.

5.3 A defective and redundant QCA PLA structure. The AND plane is 5x5 cells, and the OR plane is 3x5 cells. There are 5 faulty cells.

5.4 (a) The QSG corresponding to the QCA PLA structure in Fig. 5.3. The solid edges represent PLA cells that can be programmed into either logic or wire mode (Type 1), while the dashed edges represent the cells that are stuck in logic mode (Type 2), and dotted edges represent cells that are stuck in wire mode (Type 3). (b) The CLG corresponding to the LG in Fig. 5.1. The dashed edges show the logic to be mapped (Type 2), while the gray edges show which connections need to pass signals (Type 3).

5.5 (a) A defective and redundant QCA PLA structure that includes cells that are faulty but only affect part of a row or column. (b) The corresponding QSG. Vertices and edges have been removed based on the location and types of PLA cell faults, including partial line faults.
5.6 A plot of PLA yield versus EDR for mapping 5 benchmarks to a (150,4000,100) PLA using QCAPM method with partial line faults considered ......................................................... 104

5.7 Curves obtained from mapping the alu4 benchmark onto different systems of PLAs. As the alu4 logic is broken up into smaller parts and mapped onto smaller PLAs, the system can be mapped even with higher EDRs. ......................................................... 110

7.1 (a) Design W1 and (b) Design W2 from [45]. ......................... 130

7.2 Systolic cell for the (a) uni-directional and (b) bi-directional convolution designs. ......................................................... 131

7.3 Systolic array with 4 cells for the uni-directional design. Each clocking wire would be as wide as one cell, and would extend from top to bottom across the input wires and the height of the cell. The inputs enter the design through a shift register on the left. ........ 133

7.4 The arrangement of the nanomagnets in the simulation that demonstrated a single signal of data traveling in two directions in the same clocking zone. Note the horizontal helper magnets next to the magnets in the vertical portion of the wire to assist in nulling the magnets. ......................................................... 135

7.5 A plot of the changes in the magnetization angles of the nanomagnets in the simulation demonstrating a signal traveling in two directions in the same clock zone. ......................... 136

7.6 (a) XOR-gate based full adder, with a majority gate for the carryout signal, and (b) Majority-gate based full adder with 2 inverters, and (c) black-box full adder. ......................................................... 138

7.7 4-bit parallel adder with interconnect wires used to connect full adders and to route I/O. It can be extended to an adder of any size. 139

7.8 N-bit parallel adder with I/O at each full adder. This design can be extended to an adder of any size. ................................. 140

7.9 (a) Serial Adder design which is just the XOR-gate based Full Adder with a feedback loop, and (b) black-box serial adder. ............... 141

7.10 Image of the feedback loop simulation setup including trapezoidal magnets needed for bi-directional dataflow with a 2-phase clock. The division of the design into 3 clock zones is shown by vertical black lines. The magnets are numbered for reference. The loop is designed to move signals in a clockwise direction. ............. 142
7.11 Plot of magnetization angle for the magnets labeled 1-9 in the simulation setup image. The magnets switch correctly over the course of all 3 clock cycles as the 2 binary signals travel around the feedback loop.

7.12 NML Parallel multiplier design. There are 2 major parts to the design: (1) **AND** gate array to compute the partial products, and (2) full adder array to sum up partial products.

7.13 Serial multiplier made from many serial adders, one for each bit width. This particular example is for a 4-bit multiplier. For each serial adder there is also an **AND** gate for computing the partial products.

7.14 Parallel convolution circuit (2-bit multiply and 4-bit add)

7.15 Serial convolution circuit (4-bit multiply and 8-bit add)

8.1 Systolic cell magnet layout (a) for the uni-directional design, and (b) for the bi-directional design.

8.2 Area comparison of both designs, keeping the number of inputs (N) constant, while varying the number of systolic cells (K).

8.3 Delay comparison of both designs for \( A = 100 \), keeping the number of systolic cells (K) constant, while varying the number of inputs (N).

8.4 Energy comparison of both designs for \( A = 100 \), keeping the input size (N) constant, while varying the number of systolic cells (K).

8.5 Area-Energy-Delay comparison of both designs for \( A = 100 \), keeping the number of inputs (N) constant, while varying the array size (K).

8.6 Nanomagnet layout of the 4-bit parallel adder design. This layout is based on the parallel adder design that can place inputs and outputs throughout (see Fig. 7.8).

8.7 (a) Serial adder design repeated for comparison, and (b) serial adder layout based on the **XOR** full adder design and the NML feedback loop.

8.8 Comparison of NML adders both in terms of energy and delay per addition operation.

8.9 Parallel multiplier layout for 2-bit inputs.
8.10 Parallel multiplier layout for 4-bit inputs. The main parts of the multiplier have been indicated by the black parallelograms. This was done to help with calculating the area requirements for NML parallel multipliers with larger input sizes. 168

8.11 Serial multiplier layout 171

8.12 Comparison of NML multipliers in terms of energy and delay per multiplication operation. 172

8.13 A plot of energy and delay for CMOS and NML implementations of the bi-directional systolic cell for different technology characteristics. EDP is shown with diagonal dashed lines in units of fJ-ns. 175

8.14 Plot of voltage/current vs. energy for NML and CMOS adders 178

8.15 Plot of voltage/current vs. energy for NML and CMOS multipliers 179

8.16 Serial convolution layout with a 4-bit multiplier and a 12-bit adder. 181

8.17 Serial adder layout that uses multiple layers of nanomagnets 184
### TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>CONNECTING DEFECT TO FAULTS</td>
<td>48</td>
</tr>
<tr>
<td>4.2</td>
<td>NUMBER OF QCA DEVICES ($C_x$) SUSCEPTIBLE TO FAULTS</td>
<td>50</td>
</tr>
<tr>
<td>4.3</td>
<td>EFFECTS OF DEVICE FAULTS ON AND PLANE PLA CELL OPERATION. IF THE OPERATION OF THE PLA CELL IS UNAFFECTED, THERE IS A '.', LOGIC MODE IS INDICATED BY 'L', WIRE MODE IS INDICATED BY 'W', AND A FULLY FAULTY PLA CELL BY 'F'.</td>
<td>59</td>
</tr>
<tr>
<td>4.4</td>
<td>EFFECTS OF DEVICE FAULTS ON AND PLANE PLA CELL OPERATION. IF THE OPERATION OF THE PLA CELL IS UNAFFECTED, THERE IS A '.', LOGIC MODE IS INDICATED BY 'L', WIRE MODE IS INDICATED BY 'W', A FULLY FAULTY PLA CELL THAT CAUSES ENTIRE LINE FAULTS IS INDICATED BY 'F', WHILE A PARTIAL LINE FAULT IS INDICATED BY 'P'.</td>
<td>61</td>
</tr>
<tr>
<td>5.1</td>
<td>PLA YIELD FOR MAPPING THE EXAMPLE FUNCTIONS TO AN (8,6,4) PLA</td>
<td>97</td>
</tr>
<tr>
<td>5.2</td>
<td>PLA YIELD FOR MAPPING THE 3-BIT ADDER FUNCTIONS TO A (20,40,10) FAULTY PLA STRUCTURE USING A CUTOFF OF 5 SECONDS</td>
<td>98</td>
</tr>
<tr>
<td>5.3</td>
<td>RUNTIME COMPARISON OF THE TWO MAPPING METHODS FOR SUCCESSFUL MATCHINGS OF THE 3-BIT ADDER</td>
<td>100</td>
</tr>
<tr>
<td>5.4</td>
<td>SIZES FOR SELECTED BENCHMARKS</td>
<td>102</td>
</tr>
<tr>
<td>5.5</td>
<td>PLA YIELD FOR MAPPING 5 BENCHMARKS TO A (150,4000,100) PLA USING QCAPM METHOD WITHOUT AND WITH PARTIAL LINE FAULTS CONSIDERED</td>
<td>103</td>
</tr>
<tr>
<td>5.6</td>
<td>FAULT RATE CUTOFF POINT AND BENCHMARK CHARACTERISTICS FOR ALL TESTED BENCHMARKS</td>
<td>107</td>
</tr>
<tr>
<td>6.1</td>
<td>PERFORMANCE PARAMETERS FOR NML PLAS</td>
<td>113</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would like to acknowledge the support of this work by the Nanoelectronics Research Initiative (NRI) through the Midwest Institute for Nanoelectronics Discovery (MIND). This work was also supported in part by NSF grants CCF-0702705, CCF06-21990, CCF-0541324, and CCR-02-10153.
CHAPTER 1:
INTRODUCTION

Many emerging computational models, devices, and technologies are actively being explored to either replace or supplement CMOS technology in order to continue the performance scaling trends that have driven the computer industry over the last few decades. CMOS has been the foundation for the remarkable performance scaling trends in information processing, but scaling is becoming more difficult because of limitations in lithographic resolution and increases in static power dissipation and fabrication cost. In this dissertation, we will consider how logic operations that are performed by field-coupled, nano-scale magnets might be used to create computer circuits that can continue performance scaling. This device architecture is now commonly referred to as Nanomagnet Logic (NML). However, work on this technology has also been referred to as Magnetic Quantum-dot Cellular Automata (MQCA). NML is the term used in the latest edition of the ITRS roadmap.

In NML, lithographically defined nano-scale magnets are arranged to form circuits. Magnetic polarizations represent binary information, and logical operations are achieved by manipulating clock-driven nearest-neighbor magnetic interactions. The device-level scheme for NML is based on quantum-dot cellular automata (QCA), but its operation is substantially different from electrostatic QCA. One of the biggest differences is how the clock facilitates operation. In elec-
trosstatic QCA, the clocking field activates the QCA devices so that they can take on a new polarization and binary state, while, for NML, the clock removes state to allow for new information to propagate. NML wires behave differently when running horizontally compared to NML wires that run vertically, and the direction of the magnetic clocking fields needs to be carefully aligned with the horizontal axis of the NML devices. Also, integration of NML circuits with conventional CMOS logic is actively being studied through the use of magnetic tunnel junctions [48].

Relative to other emerging technologies, NML is mature with many experimentally validated circuit components. “Wires,” majority gates, non-majority 2-input AND and OR gates, inverters, and fanout have all been experimentally demonstrated [42, 55, 68], and operate at room temperature. Devices have also been cycled on chip [8]. It has been estimated that if $10^{10}$ magnets switch $10^8$ times per second, the magnets themselves would only dissipate about 0.1 W [19]. When the drive circuitry is included, work in [25, 52] predicts that NML circuits could provide performance wins over state-of-the-art, low power CMOS when considering metrics such as energy and energy-delay product. Devices can scale and remain non-volatile provided their size/shape remains above the superparamagnetic limit. However, binary state in nanomagnets with feature sizes below the superparamagnetic limit could be stable for around 1 ms [7] – long enough to perform logical operations. The potential for low-power NML circuits is an exciting possibility, and suitable architectures that best match the performance characteristics of NML are of great interest.

For devices that are not intended to be a direct replacement for a traditional MOSFET, the study of suitable architectures is very important. NML devices and circuits, which operate through the interaction of device magnetization and
magnetic fringing fields, behave very differently compared to CMOS devices and circuits. However, both NML and CMOS implement Boolean logic allowing for NML to take advantage of the circuit and architectural developments in transistor-based computational technologies. While lessons learned about transistor-based circuits and architectures can be applied to NML-based designs, there are many unique characteristics about this magnetic technology that make direct conversions from CMOS-based designs difficult. In order to make use of well studied transistor-based designs and approaches, clever work-arounds are required when designing for NML. This includes different approaches to using logic gates, dealing with wire crossings, playing with signal flow directionality, and separate handling of vertical and horizontal wires. Therefore, this work studies how two specific architectural approaches can be applied to NML technology.

First, programmable logic arrays (PLAs) are explored, both because of their flexibility in implementing arbitrary logic and for ease of adding redundancy for defect tolerance. Second, NML systolic architectures will be explored as an option for low-energy, high-throughput performance. Reprogrammable architectures like PLAs and field-programmable gate arrays (FPGAs) have been studied for emerging technologies that require redundancy [23, 61, 62, 64]. Systolic architectures have been studied for high-throughput applications [35, 73] and for emerging technologies [5].

The first architecture that we present is an NML implementation of a previously proposed reprogrammable QCA PLA design [17, 39]. The QCA PLA was initially explored as a reprogrammable architecture designed to enhance defect tolerance at the architectural level for circuits and systems with nanometer feature sizes and/or that may be self-assembled. To implement the PLA with
nanomagnets, some design changes are proposed to the generic QCA PLA design in [39] to create a better mapping to NML. These changes affect how data flows through the PLA architecture and how programmability is achieved. Fabrication and simulation validation of the circuit structures unique to the NML PLA are also presented. This includes programmability in each PLA cell and backward gate operation in the PLA during programming mode. NML defects are studied and the results are presented in a fault model for NML-based PLAs. All of these results are combined to provide a means for evaluating design alternatives for an NML PLA “cell” and for a complete NML PLA array which is scalable to any size.

The next important aspect of studying NML PLAs is to provide performance estimates. For this task, we used the combinational circuits from the Toronto 20 Benchmark Suite\(^1\). By mapping these benchmarks to NML PLAs, we were able to calculate area, energy, and delay estimates for the NML-based PLA design. These performance estimates were compared to performance estimates for CMOS FPGAs and hybrid nanowire-CMOS FPGAs [62]. As suggested in [22], we also studied how systems of PLAs might improve performance by breaking up the benchmarks into smaller parts and mapping them onto many smaller PLAs. We further analyze the performance and defect tolerance benefits of an NML PLA with redundant rows and columns. In our analysis, we assume that fabrication variations and defects unique to NML will be present. Results are presented in terms of the maximum tolerated defect rates for certain benchmarks mapped to NML PLAs.

In cases where NML defect rates are sufficiently small, however, redundant

\(^1\)The Toronto 20 Suite has been used to evaluate the performance of other reprogrammable nanotechnologies including CMOS FPGAs and CMOS/nanowire hybrid reprogrammable architectures [62] [64].
rows and columns are not helpful and the PLA architecture leads to performance degradation. Therefore, we continue by considering alternative architectures that are more suitable for maximizing throughput and minimizing the energy of NML systems. We focus on pipelined systolic architectures for NML. NML devices that are above the superparamagnetic limit are non-volatile and can be used for both logic and memory. This non-volatility feature, combined with the target NML clocking scheme, results in circuits that are inherently pipelined. This means that a pipelined architecture can be constructed for NML without the need for registers – as the logic devices themselves retain state between clock cycles. This avoids the overhead associated with pipelining, while gaining the throughput performance benefits of pipelining. Also, the compute-bound applications commonly implemented on systolic systems should lead to well-filled pipelines. A natural application to study in this vein is convolution, which can be implemented in NML using different circuit-level approaches.

While studying systolic architectures for NML, we considered how the NML clocking scheme affects a system’s performance, and also how it impacts what designs are possible by drawing from past work on systolic architectures [45] and NML circuit design. One study that we performed considered an NML systolic cell designed to perform a simplified 1-bit convolution from the standpoint of two different dataflow patterns. This was the first step in demonstrating that local feedback is possible with implementable NML clock structures. We also studied convolution designs for larger input sizes. We created both parallel and serial NML designs for adders, multipliers, and convolver circuits. The serial designs took advantage bi-directional dataflow through the use of NML feedback loops, which is presented here and demonstrated in simulation. NML systolic cells and
related circuits (such as adders and multipliers) are compared to CMOS in terms of energy and delay. In future, many application-specific comparisons could be made between NML and CMOS for different systolic applications (such as pattern matching) based on our work with adders, multipliers, and convolution.

The work presented in this dissertation provides detailed and realistic performance estimations to show whether or not NML will ultimately offer performance wins over other technologies at the systems-level, both for reprogrammable and systolic architectures.

1.1 Accomplishments

The following is a list of the accomplishments achieved by the work given in this dissertation. Each of these is a new result that provides a significant contribution to the field.

1. A previously proposed QCA PLA architecture has been modified to create a working NML-based PLA, including demonstration of all required circuit components and design of all necessary structures.

2. A unique study of NML defects and faults has been completed which shows how misshapen magnets can lead to stuck-at faults in NML circuits.

3. Studies of NML stuck-at faults were coupled with QCA PLA designs to create a fault model which formalized the faulty behaviors possible in NML PLA cells as a result of NML defects.

4. A graph-based PLA mapping method was improved, through the use of edge attributes, to increase the likelihood of finding successful mappings in the case of defective PLAs.
5. The design for a simplified 1-bit NML convoler has been given which could be converted into an efficient systolic array for other low-power, high-throughput computational applications.

6. An NML feedback loop has been validated through physical-level simulation, which allows for compact NML serial circuits to be created.

7. Parallel and serial N-bit NML adder and multiplier designs (including designs that use multi-layer NML) have been presented as a starting point for creating more complex N-bit NML systolic designs.

1.2 Organization

The dissertation is organized as follows. We begin in Chapter 2 by discussing the basics of QCA and NML and giving an overview of a previously proposed QCA-based PLA design. In Chapter 3, we present an NML-based PLA design and address the functionality issues that arise, including organization of the array structure and validation of the unique circuit components needed for the NML PLA. A fault model for QCA-based PLAs, including NML-based PLAs, is presented in Chapter 4. We review mapping techniques for nanowire crossbars in Chapter 5 and then leverage this work to create our improved mapping methodology for NML-based PLAs. In Chapter 6, we present performance metrics for NML PLAs, and use that data to compare with other reprogrammable nanotechnologies. In Chapter 7, we explore NML systolic architectures by looking at pipelined adders, multipliers, and convolution circuits, including a novel NML feedback circuit component that is part of the serial NML designs. Performance metrics of the designs that contribute to the NML systolic architecture study are reported in Chapter 8. Finally, a summary of this work and conclusions about the
feasibility of these NML architectures is given in Chapter 9 as well as a discussion of future prospects for NML.
2.1 QCA Basics

The initial description of a QCA device called for encoding binary numbers into devices that have a bi-stable charge configuration. A QCA device would consist of 2 or 4 “charge containers” (e.g. quantum dots) and 1 or 2 excess charges, respectively. One configuration of charge represents a binary ’1’ and the other a binary ’0’ (Fig. 2.1a) [47]. Logical operations and data movement are accomplished via Coulomb (or nearest-neighbor) interactions. QCA devices interact because the charge configuration of one device alters the charge configuration of the next device. In a magnetic implementation of QCA, charge configurations are replaced with magnetic polarizations.

Figs. 2.1b-e illustrate the building blocks that would be used to construct QCA circuits [67]. A QCA wire (Fig. 2.1b) is just a line of QCA devices. The wire is driven at the input by a device with a fixed/held polarization. The majority gate (Fig. 2.1c) implements the logic function \(AB + BC + AC\). The output device assumes the polarization of the majority of the 3 input devices [47]. By setting one input of a majority gate to a logic ‘0’ or ‘1’, the gate will execute an AND or OR function respectively. An inverter can be easily built with QCA devices (Fig. 2.1d). QCA wires with different orientations (Fig. 2.1e) can theoretically
The basic logic building blocks for NML circuits. A wire (Fig. 2.2b) is just a line of magnets that are antiferromagnetically coupled with each other. This structure has been experimentally demonstrated (Fig. 2.2i) and operates at room temperature [8,16].

2.2 Nanomagnetic Logic Basics

In a magnetic implementation of QCA, charge configurations are replaced with magnetic polarizations of single domain magnets. Fig. 2.2 illustrates three important building blocks for NML circuits. A wire (Fig. 2.2b) is just a line of magnets that are antiferromagnetically coupled with each other. This structure has been experimentally demonstrated (Fig. 2.2i) and operates at room temperature [8,16]. The basic logic gate (Fig. 2.2b) in NML is based on the majority voting function – where the output takes on the logical value associated with the majority of the 3 input devices (indicated by A, B, and C in the figure). By setting one input
of a majority gate to a logic ‘0’ or ‘1’, the gate will execute an **AND** or **OR** function, respectively. Note that in NML, the gate is actually an **inverting** majority gate. This structure has also been experimentally demonstrated and operates at room temperature (see Fig. 2.2a, [42]). Structures have also been devised to cross two signals in the plane by leveraging magnets shaped so that they can represent two bits of information simultaneously (Fig. 2.2c) [52] – as the diamond-shaped magnets have no preferred magnetization and are readily influenced by the logical state of its neighbors. This structure has not yet been experimentally demonstrated, but no exotic shapes are required and the design has been verified via micro-magnetic simulation [52].

### 2.3 Clock Structure

Regardless of implementation, a circuit or system made from QCA devices requires a clock structure that takes the form of lithographically defined, conducting metal wires [37]. A clock is required to maintain state for electrostatic QCA, and to remove state for magnetic QCA. For nanomagnets, a current in the clock wires produces a magnetic field that turns the magnetic moments of all magnets horizontally into a neutral logic state against the preferred magnetic anisotropic directions and removes remnant magnetizations from a previous computation. When the field is removed, the nanomagnets relax into a new, antiferromagnetically ordered ground state in response to a new input. For detailed descriptions on the clocking structure, refer to [18, 52].

A circuit or system made from NML devices requires a clock structure that can produce the magnetic fields needed to modulate the energy barriers between the magnetization states of NML devices. When a magnetic clocking field is applied
Figure 2.2. Cartoon representations of a wire segment (a), a majority gate (b), and a crossover (c). Wire segments have been experimentally demonstrated (d) as have majority gates (e). Crossovers function correctly in simulation (f).

Figure 2.2. Cartoon representations of a wire segment (a), a majority gate (b), and a crossover (c). Wire segments have been experimentally demonstrated (d) as have majority gates (e). Crossovers function correctly in simulation (f).
against the preferred shape anisotropy of a group of magnets, they will each enter a metastable state (or a nulled state). As the clocking field is removed, the magnets relax into a state in accordance with a new input (see Fig. 2.3(a)).

One method for providing this functionality on-chip is to use lithographically defined, conducting metal wires clad with ferromagnetic material (like word and bit lines in field MRAM) [52]. Magnets would be placed on the surface of the conducting wires which produce the magnetic field necessary to bias NML devices along their hard axes (see Fig. 2.3(b)). To increase the field strength applied to the magnets on top of the clocking wires, a magnetic cladding would be placed around 3 sides of the clock wires. In addition, a layer of insulating oxide would be
required to separate each parallel copper wire and its cladding from neighboring wires and cladding.

2.3.1 2,3,4-Phase Clocking and Dataflow

Larger NML circuits will not always fit on top of a single clock wire, and multiple clocking zones are necessary to control the entire circuit. How magnets at the boundaries of clocking zones interact with neighboring magnets in different clocking zones will affect how data moves in an NML circuit. More specifically, each clock wire – corresponding to a different clocking zone – needs to be activated in an appropriate sequence to properly control dataflow in the NML elements that reside on top. Different patterns of clock wire activation have been explored for use in NML circuits \[5, 25\], including 2, 3, and 4-phase clocking schemes.

Any NML clocking scheme should be implemented so that magnets in the same clocking zone only see the intended inputs from a previous clock zone as strong drivers. If instead, magnets that are settling into a new state are strongly influenced from the input and output side, then those magnets will be driven from two directions – which is undesirable.

For 3-phase clocking patterns, it is easy to establish the direction of dataflow and protect against back propagation by exciting clock lines in a particular order. In a 3-phase clocking scheme, the pattern of clock activation repeats every 3 zones (see Fig. 2.4(a)). The magnets in Zone 1 are not clocked and serve as a strong driver for Zone 2, while the magnets in Zone 3 are nulled by a clocking field such that their magnetization states are nearly horizontal (i.e. hard-axis biased) and will not act as a strong driver for those in Zone 2. The magnets in Zone 2 can settle correctly into a new state, only influenced by the one strong driver from the...
left (Zone 1). (A 4-phase clock is similar to a 3-phase clock in that the direction of dataflow is determined by the pattern of clock wire activations, however the pattern repeats every 4 zones instead of every 3.)

In a 2-phase clocking scheme, the direction of dataflow is not defined by the order of clocking zone activation, because, for magnets in zones ready to settle into a new state, both neighboring zones are unclocked (see Fig. 2.4(b)). As such, fringing fields from magnets at the end of Zone 1 and the beginning of Zone 3 could drive the devices in Zone 2 from opposite directions. One way to avoid this situation is to place trapezoid-shaped magnets at the clock boundaries to help define the direction of dataflow [25]. A trapezoid magnet will concentrate magnetic flux more in one direction (longer edge side), and less in the other direction (shorter edge size). The location and orientation of a trapezoid magnet near the clock boundaries can enable a 2-phase clocking scheme and avoid back propagation. However, according to initial simulation results [25], a higher clock field strength (and thus more energy) is needed to successfully propagate data along an NML wire with a 2-phase clock.

2.4 QCA PLA Cell and Array Structure

Due to the unique operation of QCA majority gates, the QCA PLA discussed in [39] uses **AND** and **OR** logic. A schematic of one PLA cell for the **AND** plane is shown in Fig. 2.5a. In this dissertation, we will refer to the structures at the crosspoints of the PLA as “PLA cells” and the QCA building blocks as “QCA devices.” This structure contains a reprogrammable **select bit** (denoted by “Select” or “S”) and two majority gates – one configured to act as an **AND** gate and the other configured to function as an **OR** gate. Referring to the layout in Fig. 2.5b:
if S=0, \((\text{Implicant Out}) = (\text{Literal In}) \bullet (\text{Implicant In})\)

if S=1, \((\text{Implicant Out}) = (\text{Implicant In})\)

Thus, if S=0, the PLA cell acts as an AND gate (logic mode), and if S=1, the PLA cell will act as a wire (wire mode). The ability to conditionally set each select bit makes the PLA reprogrammable.

In the OR plane, the position of the AND and OR gates in one “cell” is reversed (Fig. 2.5c). The select bit should be set to 1 for logic mode and 0 for wire mode:

if S=1, \((\text{Function Out}) = (\text{Implicant In}) + (\text{Function In})\)

if S=0, \((\text{Function Out}) = (\text{Function In})\)

By leveraging the structures just discussed, it is relatively easy to construct the logic required for a PLA of arbitrary size. Fig. 2.6 shows how the “cell” construct can be used to make entire AND and OR planes. The select bits are set to evaluate two Boolean functions with their binary values in the inset triangles.

For a magnetic implementation of QCA, the PLA design will have some slight differences, but the basic idea will remain the same. There will still be two gates
Figure 2.5. (a) QCA AND Plane Cell Schematic, (b) AND Plane Cell Layout, (c) QCA OR Plane Cell Schematic, (d) AND Plane Cell Regions. Note in (a) and (c) that the circle with a dot inside indicates an AND gate, and the circle with a plus sign inside indicates an OR gate.
Figure 2.6. A QCA PLA that implements the Boolean functions
ABC+BCD and ABC+AD

and a select bit for each PLA cell. The design of the NML PLA will be discussed
in detail in Chapter 3.
CHAPTER 3:

NML PLA ARCHITECTURE

Previous work on QCA-based PLAs was intended to be independent of the QCA implementation [17, 39], although it favors electrostatic implementations. We continue the discussion of QCA PLAs assuming a specific magnetic implementation of QCA. While the basic idea behind the PLA will be the same, the design and dataflow will be quite different. Earlier QCA PLAs were based on PLA “cells” having two functional modes, and those cells could be arranged into arrays of any size. Once an array of PLA cells is created, each cell can be programmed into one of two modes to implement any Boolean functions. A QCA PLA made from nanomagnets can be created with the same requirements, but the layout of the nanomagnet devices needs to be different.

In order to create an NML PLA cell, there are differences that must be taken into consideration. First, vertical and horizontal wires created with electrostatic QCA devices are very similar in operation. Vertical wires made with nanomagnets are very different from horizontal NML wires. This means that the NML PLA cell design must use horizontal and vertical wires differently. Second, the clocking field used to drive QCA circuits is significantly different between electrostatic and magnetic implementations. For electrostatic QCA PLA implementations, a diagonal clocking mechanism was suggested [17] (although not the first time a diagonal clocking scheme was proposed for QCA) so that one diagonal direction
could be used for programming the PLA cells, and the other diagonal direction could be used for PLA operation and logic evaluation. Because the electric clocking field used in electrostatic implementations of QCA runs perpendicular to the plane of QCA devices, the layout of the parallel clocking wires could be arranged in any direction. On the other hand, the magnetic clocking field needed to null the nanomagnets must run through the plane of the QCA devices and must be perpendicular to the easy axis of the rectangular nanomagnets. Because of these requirements for NML, the NML PLA cell must be designed so that it can be programmed and operated with clocking that drives the data either left-to-right or right-to-left. A diagonal clock will not work because it will not null the nanomagnets properly.

Using the QCA PLA design described in Sec. 2.4 [39] as a starting point, for this work, we created a logically equivalent NML PLA design. The PLA cell design that we have created for NML still maintains the bi-directional functionality of past QCA PLA designs. In this section we will discuss (a) specific modifications made to the baseline PLA design to accommodate NML dataflow, and (b) introduce NML specific structures that will enable a logic function to be mapped to many smaller, interconnected PLAs. The NML PLA design, and the use of multiple, smaller PLAs form the basis for our performance analysis in Chapter 6.

3.1 AND Plane PLA Cell Design

In an attempt to create an efficient, NML-specific PLA, we analyzed two possible layouts for an AND plane cell. The first design, referred to as the vertical design, was simply a direct mapping of the QCA PLA cell layout in [39] to a design with NML devices. The schematic is illustrated in Fig. 3.1a, and the nano-
Figure 3.1. (a) NML AND plane cell schematic with vertical implicant wire, (b) NML AND plane cell magnet layout with vertical implicant wire, (c) NML AND plane cell schematic with zig-zag implicant wire, (d) NML AND plane cell magnet layout with zig-zag implicant wire.

A magnet layout is shown in Fig. 3.1b. As in Fig. 2.5a, in Fig. 3.1a, the circle with a dot inside indicates an AND gate, and the circle with a plus sign inside indicates an OR gate. In the second design, referred to as the zig-zag design, the implicant wire is laid out in a “staircase” fashion, from top-to-bottom and from left-to-right (Fig. 3.1c). An important feature of this second design is the shorter vertical wire (we highlight the utility of this design feature below). In terms of NML devices, this design is longer in the x-dimension and shorter in the y-dimension (Fig. 3.1d).

Both designs of the NML AND plane cells have the same logical functionality as the QCA-based AND plane cell. We considered both designs for implementation of an NML PLA structure. The main differences between the designs are:

- Cell size
- AND plane arrangement
- Clocking pattern
- Use of vertical NML wires
To help determine which approach is a better design alternative, we first present a subset of physical-level, micro-magnetic simulations of vertical NML wires of varying lengths (i.e., numbers of nanomagnets). As an example, we discuss a ferromagnetically ordered line consisting of 2 magnets and another consisting of 4 magnets (see Fig. 3.2a).

The net objective of the simulations is to determine what external field strength is needed to place the magnets in a ferromagnetically ordered line into a metastable, or “nulled,” state so that the line can be correctly re-evaluated with a new input. We wanted to determine if the longer lines would require higher external fields and hence more energy. As seen in Fig. 3.2b, the 4 magnet line requires an external field that is about 35% greater than the 2 magnet line (32 mT vs. 18 mT)
and would realistically require more energy\(^1\). We also compared the area and the number of magnets in each design. The vertical cell design has 32 nanomagnets and has a 9x12 magnet footprint. The zig-zag cell design requires 40 magnets and has a 7x18 magnet footprint. Thus, the area difference between designs is about 14\% (with the zig-zag being larger).

Finally, we considered how the cells would be arranged into an array to compose an AND plane, and how this might affect the clocking structure design. The vertical AND plane cell can be arranged so that the neighboring cells are placed directly next to and above/below each cell. This leads to a rectangular AND plane array (as shown in Fig. 3.3a). The zig-zag AND plane cell can be arranged with neighboring cells directly next to each other, but neighbors above and below are offset by half the width of the cell (see Fig. 3.3b). This leads to an AND plane with a “staircase” or parallelogram shape.

From the perspective of layout, the rectangular design (Fig. 3.3a) is more desirable. However, from the perspective of clocking, the “staircase” design in Fig. 3.3b can be more advantageous. The rectangular AND plane array created with the vertical AND cell has many cells arranged in a single column. If the entire column was controlled by the same clock wire (see Fig. 3.3a), then many magnets would be simultaneously placed into a metastable state. This would lead to a very large critical path that would (a) grow with the size of the PLA (see annotation in Fig. 3.3a) and (b) be more susceptible to fabrication errors, thermal fluctuations, etc. [11, 53]. To avoid excessively long critical paths, subgroups of PLA cells in the same column would need to be clocked separately and any required drive circuitry would become more complex.

---

\(^1\)Physically, the reason for this is stronger coupling between devices. In the longer line, more magnets exert \(H_y\) biases on their top and bottom neighbors – further reinforcing previous state.
Figure 3.3. Examples of a 2x2 AND Plane array using: (a) vertical AND plane cell schematic and the (b) zig-zag AND plane cell schematic. Arrows show the propagation of literal and implicant signals though the array structure.

Alternatively, an AND plane made from the zig-zag AND cells would have each row offset from the previous row – resulting in critical path lengths that are much shorter. As a signal propagates from left to right and top to bottom, it would be controlled by different clock wires as seen in Fig. 3.3b. Moreover, a vertical clocking wire with a width of one cell could span the entire height of the PLA, significantly simplifying the clocking structure.

With an eye toward manufacturability, a design based on the zig-zag AND plane cell is a better choice. The size differences between it and the vertical cell are small, the shorter, vertical wires that this design requires should reduce the energy overhead from the clock, and the clock structure itself should be easier to build for the zig-zag cell based design.
3.2 Putting the AND and OR Planes Together

The QCA PLA in [39] uses an OR plane cell design that is almost identical to the AND cell. The only difference is that the AND and OR gates are switched, and the cell is rotated 90 degrees (see Fig. 2.5a versus Fig. 2.5b). This can be done with the NML PLA as well.

However, if we were to form an OR plane cell by starting with the zig-zag AND plane cell design, switching the gates, and rotating the cell by 90 degrees, it would be difficult to connect the two planes together into a single PLA structure (see Fig. 3.4a). Because the cells have different widths and heights, and because the planes have a staircase shape, the outputs from the AND plane do not line up with inputs to the OR plane.

It would be quite difficult to resolve the timing issues that such an AND–OR plane concatenation would impose. Specifically, each output signal from a zig-zag cell-based AND plane arrives at the bottom of the plane sequentially (i.e., not at the same time – see Fig. 3.3b). However, the OR plane design in [39] requires that all inputs arrive simultaneously. Moreover, as seen in Fig. 3.4a, a direct design translation would lead to the same (excessively long) vertical critical paths that we earlier worked to avoid.

Another design is to switch the position of the AND and OR gates, without rotating the cell (see Fig. 3.4b). However, this design requires significant interconnection overhead. To move outputs from the AND plane to inputs in the OR plane, the vertical outputs from the AND plane would have to drive anti-ferromagnetically ordered lines (horizontally) into the OR plane (see Fig. 3.4c). This represents a significant interconnection overhead where no computation is performed.

Given the above considerations, we redesigned the OR plane cell so that an OR
Figure 3.4.  (a) An example 3x3 AND plane array and 3x3 OR plane array to illustrate a timing problem that arises if the AND and OR planes are placed adjacent to each other.  (b) NML OR plane cell schematic similar to the zig-zag AND plane cell, but with AND and OR gates switched.  (c)  An example 3x3 AND plane array and 3x3 OR plane array to illustrate the routing overhead needed if the AND and OR planes are oriented the same way.
plane array could easily be placed next to an AND plane array, allowing data to flow through the entire PLA more efficiently. The new design for the NML OR plane cell is shown in Fig. 3.5a-b. This redesigned OR plane cell utilizes zig-zag data propagation, with the Implicant In signal moving from top to bottom in a cell that is wider than it is tall. This is very much like the zig-zag AND plane cell, as opposed to the other OR cells we have considered, which are either taller than they are wide or have the implicant signal moving from left to right. The implicant signal fans out into two signals; one which leads down to the next cell through a wire crossing, and the other which is input into the AND gate adjacent to the programmable bit. This is done so that it can be conditionally ORed with the Function In signal, which moves left to right across the cell. The result is an OR plane that produces output functions that move horizontally across the plane and are available on the right side of the array. This design is similar in size to the zig-zag AND plane cell (18x8 magnets vs. 18x7 magnets), allows the AND and OR planes to be adjacent (no interconnect), and avoids long vertical critical paths in the OR plane.

By leveraging the structures just discussed, it is relatively easy to construct the logic required for a PLA of arbitrary size. In each plane, rows of adjacent PLA cells are created, and each row is offset from the row immediately above it by half the width of one cell. An example of an NML PLA structure with both planes appears in Fig. 3.5c. As in Fig. 2.6, this NML PLA structure is programmed to implement the Boolean functions ABC+BCD and ABC+AD. The literals would enter the array from the left, and pass through each cell from left-to-right. Inside each cell, the logical operation of the gates would produce implicants from these literals. The implicant wires would carry each implicant signal down and to the
right in a staircase pattern. Therefore, dataflow within each AND plane PLA cell (and the entire AND plane as a whole) would be top-left to bottom-right. The same is true for the implicant and function signals in the OR plane. Thus the dataflow for the whole NML PLA occurs in a zig-zag path through the staircase-shaped structure.

3.3 Programmability

Any layout must still enable/support PLA cell programmability. The PLA design present in Fig. 3.5 fully supports the bi-directional dataflow used in the previous QCA PLA design [39]. If we change the order in which clock wires are activated, we will change the order in which NML devices are placed into a metastable state. Thus, if we initially set the state of magnets on the right side of the design to serve as inputs, we can drive signals into the select bits given the design in Fig. 3.5c to achieve PLA cell programmability. In Sec. refssec:Backwards, we show that it is possible to drive signals backwards through the PLA cells so that those signals can reach the select bits. We also show how it is possible to drive a signal into the select bit so that a binary value is saved there, and we show how it is possible to drive that saved signal out of the select bit without erasing it so that it can be used to evaluate the PLA (see Sec. 3.5.3). As for the pattern of inputs required to program the PLA, a detailed description of the programming procedure is given in [39].

3.4 Interconnect

In certain PLA designs, it is desirable to map logic to systems of smaller PLAs. This approach has been considered for reprogrammable crossbar PLAs
Figure 3.5. (a) OR plane cell schematic that utilizes zig-zag propagation of the implicant signal; (b) OR plane cell magnet layout; (c) An example NML PLA structure programmed to implement the Boolean functions ABC+BCD and ABC+AD.
and improved both defect tolerance and performance [21]. To investigate the applicability of systems of PLAs in the NML paradigm, we have developed a methodology to support the construction of such a system.

Specifically, we arrange groups of PLAs into levels and connect one level to another via an interconnection network. To provide this functionality, we decided to utilize the PLA cells we created for logic to implement an NML-based programmable interconnect. The interconnect between the different levels of PLAs can be programmed to route signals using the same process as programming for logic, with the major difference being that only one cell should be programmed to logic mode in each row (meaning that, at most, only one signal will be input or output from the interconnect from each row). Successful interconnect solutions would be found using a mapping process similar to how successful PLA logic mappings are found. The interconnect would also take on a staircase pattern just like the PLA. An example system consisting of 5 PLAs and one interconnection array is shown in Fig. 3.6.

It is important to differentiate between rows of the interconnection network that accept signals, and rows that let signals leave the network. This can be achieved by using both AND and OR plane cells. Output signals from a previous level of PLAs enter the left side of the interconnection network into a row composed of AND plane cells. These cells receive the interconnect signals much as the AND plane cells in the PLA receive literals. By programming one of the AND plane cells to logic mode, the interconnect signal on that row will also move down the interconnect array in a staircase fashion. From this point, moving down the array, the signal passes through PLA cells programmed to wire mode so that it is not affected by other signals. The signal is also available to be output from the network. For a
signal to be output, there needs to be a row of OR plane cells. If the OR plane cell in the same column is programmed to logic mode, then the signal will also move horizontally out of the interconnect into a PLA in the next level. This is illustrated in Fig. 3.7. Note that in a system composed of NML PLAs and NML interconnect, the AND and OR plane cells must be the same width and height so that they align properly.

This NML-based interconnection network affects performance – manifested as an area, delay, and energy overhead. As no logical operations are performed in the interconnect, we want to make this overhead as small as possible. In the simplest implementation (see Fig. 3.6), the height of the interconnect is determined by the height of all PLAs in the previous level of logic, and the width is determined by the number of signals that must be routed between levels. We explore how the interconnect requirement for different benchmarks affects performance in Sec. 6.2.2. For the benchmarks that required NML interconnect, the area and

Figure 3.6. An example NML PLA system of 5 PLAs with NML interconnect.
Figure 3.7. Illustration of how a signal can be routed through the NML-based interconnect between two levels of NML PLAs. An output function signal exits a PLA (top left) and enters the interconnection network. By programming cells in the network, the signal can be routed down the interconnect structure and then into the next PLA (bottom right).

Energy overhead was prohibitively large, although breaking up a large PLA into smaller PLAs with interconnect improved delays in some cases (see Table 6.7).

3.5 Component Validation

An important part of the design of the NML PLA is ensuring that the nanomagnet layout will operate as we intend post fabrication. The core components of the NML PLA (i.e., AND and OR plane cells) are based on circuit elements that have been demonstrated either experimentally or via physical-level simulation –
e.g., wires, logic gates, and wire crossings. Simulation of the entire PLA structure using the physics-level OOMMF simulation tool would be prohibitively time consuming as it would contain thousands of magnets for implementing even a small benchmark. Even for simulation of just one PLA cell (just over 40 nanomagnets), it would still be a large simulation. In addition to the magnets, we would also have to include helper magnets, a wire crossing, multiple clocking zones, and neighboring boundary-condition circuits into the simulation. Such a simulation would take several days or weeks. With dozens of iterations to determine the optimal setup, the entire undertaking would be very time consuming. Instead, core components demonstrated in isolation can be tested in such a way that it is reasonable to assume that they will work collectively (e.g., as was done in [21]).

That said, the NML PLA in Fig. 3.6 ultimately requires two specific types of dataflow that have not yet been considered experimentally or via physical-level simulation:

1. Driving signals backward through majority gates during programming mode.

2. The ability to route a signal to a programmable select bit (during programming mode) – and then have that bit drive a gate while the PLA is being used to evaluate a given set of inputs.

To ensure that these operations are in fact feasible, we consider both via physical-level simulation.
3.5.1 Evaluation Environment

The Object-Oriented MicroMagnetic Framework (OOMMF) simulation tool (developed by NIST) \cite{26} was used for the NML simulations discussed here.\footnote{OOMMF is widely used and there is excellent correlation between simulation and experimental results \cite{49,71}.} The simulation parameters discussed here apply to all OOMMF simulations discussed in Sec. 3.5.2 and Sec. 3.5.3. In those simulations, devices were assumed to be supermalloy (79% Ni-15% Fe-5% Mo), with 60 × 90 nm$^2$ dimensions and 30 nm thicknesses. NML structures with supermalloy magnets are regularly made by our research group and representative structures can be seen in Fig. 2.2. In our simulations, magnets were spaced 12 nm apart horizontally and 24 nm apart vertically. Helper islands were also placed next to each magnet’s left and right edge in the case of vertical magnet arrangements – such that the easy axes of each helper island is parallel to the direction of an applied clocking field (see Fig. 3.2a and Fig. 3.9a). We do not provide a full discussion on the utility of helper islands, but they essentially generate a static bias along the hard axis of each magnet in the ferromagnetically ordered line. As magnetic field vectors add, the net external field (and energy) from the clock can be reduced (see \cite{53} for additional discussion). (Helper cells are not shown in Figs. 3.1b,d and Fig. 3.5b for clarity.) Magnets with dimensions of 60 × 90 nm$^2$ are stable at room temperature and behave predictably during switching according to many OOMMF simulations we have run.

3.5.2 Backwards Operation of Majority Gates

During programming mode, the select bits of each cell in the PLA array must be set to a binary ‘1’ or ‘0’ to ensure that the PLA computes a desired logical function. Per \cite{39}, to initially configure a PLA, signals must be brought from the
2. ACKNOWLEDGMENTS

Michael Crocker, X. Sharon Hu, and Michael Niemier
University of Notre Dame
Notre Dame, IN 46556, USA
Email: mcrocker,shu,mniemier@nd.edu

Categories and Subject Descriptors: B.6.2 [Hardware]: Logic Design—
Function In
Implicant In
S

1. BACKGROUND

Defects and Faults in QCA-Based PLAs

Quantum-dot Cellular Automata, Defects, and Faults yield. Within this context, we introd-
el, and discuss how expected defects

Select Bit Value

Literal In

B

Avoiding Defects

The authors gratefully acknowledge the support of this work

C

Select Bit Value

D

AND Plane

OR Plane

ABC + AD

ABC + BCD

0

1

0 1

1

1 0 0

0

00

1

11

10

0

Fig. 1. Alternate NML PLA Cell Design

S

Implicant In

(ii)

Implicant Out

Binary '0'

Must bring logic '0' to OR gate input; can treat AND gate as
1:2 “fanout” structure (fixed ‘0’ input does not interfere).

A.

1:2 “fanout” structure (fixed ‘0’ input does not interfere).

(ii)

Select Bit Value

(i)

Fixed input

To OR gate

To Implicant In

B.

Fixed input

To OR gate

Fixed input

Select Bit Value

To Implicant In

Fig. 3.8. (a) Illustration of the operation necessary to program the
select bit in an AND plane cell. The signals must be driven backwards
through the cell, from right to left. (b) Schematic of the magnet
ensemble simulated to determine if required functionality is achievable.

edge of the PLA to each PLA cell, and then to the individual select bits. Referring
to Fig. 3.5, signals propagate from right to left during this programming process.
Since data flows from left to right during operational mode, we must ensure that
the gates can operate in the opposite direction.

As an example, consider the AND plane PLA cell in Fig. 3.8a. To program the
select bit in an AND plane cell, the desired value must be brought (unchanged)
through the OR gate (see (i) in Fig. 3.8a). The only way to accomplish this is to
OR a binary ‘0’ with the desired select bit value. To accomplish this, the second
input to the OR gate must be a logic ‘0’ (see Fig. 3.8a(ii)).

Physical-level simulation results indicate this functionality is in fact achievable
with the following procedure: the wire labeled Implicant Out from the AND cell
now serves as an input – and is set to a logic ‘0’. The desired select bit value
can be placed on the line labeled Select Bit Value (see Fig. 3.8a-i). When a clock
field is applied to the magnet ensemble, fringing fields from the Select Bit Value

35
magnet initiates antiferromagnetic coupling on the Literal In line. Subsequently, the select bit signal propagates into the OR gate, and propagates across the wire to the next PLA cell. Additionally, the majority-based AND gate essentially doubles as a 1:2 fanout structure. The logic ‘0’ input at Implicant Out propagates up toward Implicant In and to the other OR gate input (see Fig. 3.8a-ii). The hard coded logic ‘0’ input (which makes the majority gate function as an AND gate) should not interfere with this operation as it only reinforces the binary ‘0’ value that must propagate to the second OR gate input. Moreover, the logic ‘0’ that propagates up to Implicant In serves as an input to Implicant Out of the next PLA cell. This same procedure can be utilized for OR plane cells in a similar fashion, except that the OR gate doubles as a 1:2 fanout structure for a binary ‘1’ signal, which propagates to the AND gate and allows a select bit signal to pass through the AND gate to the select bit.

In order to test the 1:2 fanout structure, we considered the AND plane cell and simulated the lower right portion of the cell (Fig. 3.8b), which includes a majority gate programmed to be an AND gate and three wires leading from it. During PLA evaluation, this structure is used as an AND gate with 2 inputs and 1 output. However, during programming, this gate is instead used as a 1:2 fanout structure, with 1 input and 2 outputs. We wished to show through simulation that a binary ‘0’ can be propagated from right to left through the gate so that it is fanout both up and to the left. The simulated design (see Fig. 3.9a) had 9 magnets – one of which (magnet 1) is fixed in a binary ‘0’ state by a horizontal magnet below it – as well as many helper islands next to the magnets in the vertical wires and at the boundaries.

In our simulations, magnets were initialized into a metastable (nulled) state,
Figure 3.9. (a) Magnet layout of the simulated 1:2 fanout structure (magnets are numbered), (b) time evolution of magnets propagating the signal vertically (magnets 4,5,6), and (c) time evolution of magnets propagating the signal horizontally (magnets 4,7,8,9).
and a field of 30 mT was applied to the simulated structure. As the applied clock field decreases in magnitude, the fringing fields from the input (magnet 2) sets the state of its immediate neighbor (which in turn sets the state of its neighbor, etc.). Simulation results are illustrated in Fig. 3.9b-c. Here magnet 1 is set to a binary ‘0’ by the horizontal magnet just below to create an AND gate, and magnet 2 is set to a binary ‘0’ using an external magnetic field so that it can act as an input to the fanout structure. Fig. 3.9b shows that the ferromagnetically coupled vertical wire (magnets 4, 5, and 6) successfully propagates the binary ‘0’ (negative magnetization angle). Fig. 3.9c shows that the antiferromagnetically coupled horizontal wire (magnets 4, 7, 8, and 9) successfully propagates the signal (alternating negative and positive magnetization angles). Specifically, magnet 4 takes on a negative magnetization angle from the influences of magnets 2 and 3, and magnet 1 reinforces that state. Then magnet 4 influences the other two fanout wires to propagate the correct binary value. Note that the horizontal magnet used to hard code a majority gate input does not adversely affect the propagation of a logic ‘0’.

3.5.3 Routable Programmability

Programmability of NML gates has been demonstrated experimentally in [69]. This is achieved by selectively changing magnet shapes or aspect ratios. By increasing a magnet’s length along its easy axis, its coercivity in that direction increases. As a magnet’s length increases, higher external fields are required to facilitate a state transition (see Fig. 3.10a). Thus, if we make one magnet longer, and the external field applied is (a) sufficient to set the state of the smaller inputs, (b) insufficient to switch the longer input, and (c) sufficient to place the
other magnets in the gate into a metastable state, then the longer input can retain its initial magnetization state even after the entire gate switching process is complete.

While the state of the programmed bit in Fig. 2.2c is set with external fields of varying magnitude, “routable programmability” could be achieved by creating anti-ferromagnetically ordered lines of magnets with different aspect ratios – as lines made from higher aspect ratio magnets should require higher magnetic fields to facilitate switching than lines of lower aspect ratio magnets. Post-programming re-evaluation would occur with lower external fields. This approach could be applied to the NML PLA to provide a circuit component that would operate as the reprogrammable select bit.

Another way to create a programmable bit is to use an NML wire with helper magnets on the left end of the line. This setup is presented in Fig. 3.10b as a 6 magnet line with 2 helper magnets on the left end. We wish to show that by using this simple structure, a binary value can be driven into the line from the right, and that subsequent clocking of the structure from left to right leads to that same binary value being driven out of the wire. This behavior is desired for the select bit in the NML PLA, because it must repeatedly output the same binary value during PLA evaluation. Therefore, we simulated this 6 magnet structure using OOMMF.

Fig. 3.10c shows the time evolution of each magnet during the simulation that demonstrates driving a signal into the wire. When trying to set the state of the programmable bit, magnet 1 acts as the input and drives the wire from right to left. As the clocking field is applied, magnets 2 and 3 begin to switch into new states based on the magnetization of the input magnet (magnet 1). However, magnets
Figure 3.10.  (a) The effects of coercivity are shown by plotting the hysteresis loop for magnets of different aspect ratios. The longer magnet (dashed line) requires a larger applied field to switch its state than the shorter magnet requires (solid line). Such a magnet could be used to save the binary value in the select bit wire of the PLA cell. (b) A possible select bit structure: a wire of 6 magnets with helper magnets on the left end. The magnets are numbered for easier reference. (c) The time evolution of magnets in the select bit structure, showing how a signal can be driven into structure from an input on the right. (d) The time evolution of magnets in the select bit structure, showing how the binary value programmed into the wire can be used multiple times, since the magnets are not changed with an applied clocking field.
4-6 retain their old state while the field is applied. Later in the simulation, the
clocking field is removed allowing all magnets (2-6) to relax into the correct state
based on the state of magnet 1. This shows that the binary state held on the
NML wire can be programmed with a new signal brought in from the right side.
Note that the plot in Fig. 3.10c shows a long period of time between 28 and 42 ns
where the field is applied but no switching occurs. Because of the way in which
this simulation was run, the time in between is unnecessary and the switching
activity occurs mostly at the beginning and the end of the plot.

Fig. 3.10d shows the time evolution of the same 6 magnets in a simulation
that demonstrates how the binary value held on the wire is retained even after a
clocking field is applied. During PLA evaluation, magnet 6 acts as the driver to
the wire, since it feels no significant vertical influence from the helper magnets. A
clocking field is applied to the wire, and the magnetizations of the magnets in the
wire are turned closer to the horizontal, but are never switched. This wire can be
used any number of times to evaluate the PLA cell using the same binary value of
the select bit (until it is programmed to a different binary value). This approach
for creating a select bit wire could be combined with a higher aspect ratio magnet
to take advantage of both approaches.

To summarize, past experimental work [69] has shown that by using higher
coercivity magnets, selective programmability is possible in NML circuits. Pro-
grammability is also possible using a standard NML wire with helper magnets
on the left end, thereby taking advantage of the right-to-left data propagation
during programming and the left-to-right data propagation during evaluation. By
leveraging either or both techniques, it is possible to implement a reprogrammable
select bit for the NML PLA.
With backward gate propagation demonstrated (using AND and OR gates as fanout under certain conditions) and routable programmability demonstrated, combined with the demonstrated basic NML circuit elements (wires, gates, wire crossings, fanout), all the pieces are now in place for an NML PLA.

3.6 Summary

In this chapter we give an NML PLA design based on the QCA PLA design. Creation of the NML PLA includes layouts of the NML PLA cells to satisfy NML circuit designs concerns. One result is the NML-specific AND and OR plane cell designs that allow for a tight array organization when the cells are joined together. Also, a means of providing NML interconnect between NML PLA arrays is provided. Finally, we have validated through simulation two unique NML circuits which are required for a working NML PLA. The first is validation that the NML gates in the PLA cell can be operated backwards (from right to left) to allow for binary signals to reach the reprogrammable bits when passing through the PLA structure during Programming Mode. Second, the select bit wire itself has been validated to show that a binary signal can be saved to the wire (perhaps requiring a stronger NML magnetic clock field during programming), and that the signal which is saved in the select bit wire can be driven out each clock cycle to allow the programmable bit to do its job during Operational Mode.
CHAPTER 4:
DEFECTS AND FAULTS

The QCA PLA is a reconfigurable architecture intended to provide “defect avoidance” through redundancy and routing around defective PLA cells. It is important to study the defects possible in QCA circuits as the information gained from these studies will be necessary to determine how these defects will affect the behavior of QCA PLAs. In addition, we would like to determine what types of defects lead to what types of faulty behavior for different implementations of QCA. Similar studies for QCA devices and circuits have been performed in previous work [32, 51, 66] but not for the QCA PLA. We start by developing a fault model for QCA PLAs, and then expand that fault model for different implementations of QCA, especially Magnetic QCA. From there, we want to know what levels of redundancy will be necessary to allow QCA PLAs to successfully implement certain Boolean logic functions even in the presence of defects at different rates. Previous work has investigated the inherent reliability of QCA circuits including PLAs to determine what defect rates can be tolerated without redundancy [28, 29]. Also, N-modular redundancy has been investigated for QCA circuits as a means of providing defect and fault tolerance [30, 31]. By the end of this chapter, we will have presented a comprehensive defect and fault study for NML and NML PLAs. This information will be used as the starting point for creating a mapping scheme
for NML PLAs, which will be discussed in Chapter 5. Mapping to redundant QCA PLAs will provide a means for studying defect “avoidance.”

For MOSFET PLAs, the crosspoint fault model [63] was developed as a model that better encompasses the faulty behaviors specific to PLAs as an extension of the stuck-at fault model. The crosspoint model considers how defective or misplaced transistors cause the Boolean logic implemented by a PLA to change. There are four crosspoint faults: growth, shrinkage, appearance, and disappearance. However, the crosspoint model is not sufficient for many nano-scale PLAs, as there are nano-scale defects and faults not present in MOSFET PLAs. Therefore, we will present a fault model for QCA-based PLAs that includes additional PLA faults.

4.1 Modeling QCA Faults

In this section, we discuss defects and the consequent faulty behavior that comes from the defects. Here, a “defect” is defined as a deviation of a QCA device from the ideal device in terms of shape, orientation, location, etc., and is severe enough to cause an error. For example, a QCA device shifted out of place could be a defect. A defect can be thought of as a permanent physical error. A “fault” is the incorrect logical behavior in a QCA circuit as a result of a defective device. For example, a QCA defect could produce a stuck-at-0 fault in a QCA wire causing that wire to only output a binary ‘0’ no matter which binary value is input to the wire. A fault can be thought of as a functional error that might or might not occur again, although, if it is caused by a defect, it is likely that the fault would persist. In this section, we first discuss possible defects and faults for different QCA implementations, then we quantify the fault probabilities based on
defect rates, and then we examine the effects that those defects and faults can have on the QCA PLA structure.

4.1.1 QCA Defects and Faults

We first introduce the unique types of faults in QCA devices and then discuss what kinds of defects cause such faults. Due to its different operating principles, QCA devices experience additional types of faults besides the traditional MOSFET faults in the stuck-at fault model. QCA devices experience the stuck-at fault (which is equivalent to the MOSFET stuck-at fault), and two other types of unique faults. The first is the non-deterministic fault (ND) where the output of a circuit may vary during run time for the same inputs. The second is the inversion fault, where the output of a circuit becomes the inverted value of the desired output.

The ND fault occurs primarily in electrostatic implementations. Looking at molecular QCA, the output of a circuit has a probability of being ‘1’ or ‘0’, which is based on the Boltzmann distribution of electrons in the QCA devices. Circuits that operate properly have a high probability to output one value and a low probability to output the other. However, if a circuit is unstable, it will have probabilities approaching 50% for outputs of both “1” and “0”. For these types of circuits, small fluctuations in the environment surrounding the circuit (such as a change in temperature) can cause it to behave differently each time the circuit is used. Simulations show that a defect-free wire circuit that operates at a high probability can become unstable with a single missing QCA device. The ND fault captures this unpredictable behavior of a QCA circuit.

The inversion fault can occur in both electrostatic and magnetic implementa-
tions. It is often instigated by device misalignment. The misalignment can cause a QCA device to take on the opposite polarization of its neighbor. This phenomenon is exploited to our advantage in the inverter design (Fig. 2.1), where neighboring QCA devices are placed next to each other in a diagonal direction to achieve inversion. However, an undesirable misalignment during fabrication can cause an unwanted inversion on a wire. This misbehavior is categorized as an inversion fault.

Defects in each QCA implementation may cause any one of the three faults (stuck-at, ND, and inversion) discussed above. Proper association of defects to faults helps in studying fault probabilities and yield. By examining experimental data and performing simulations at the physical level [26, 39, 56], we have matched defects to faults for two QCA implementations. In Table 4.1, we summarize the faults that are induced by five typical defect types, i.e., shifts, rotations, missing, stray charges, and misshapenness. While variations can occur, Table 4.1 represents the “common case.”

There are five primary defects that we consider. For all implementations of electrostatic QCA, stray charges (α) can affect whether or not an individual QCA device retains its correct value (see Fig. 4.1a). Device shifts (β) and rotations (γ) occur when the position or the orientation of the QCA device deviates from the ideal (Fig. 4.1b and Fig. 4.1c respectively). Each of these defects can cause QCA devices to interact with one another in ways they normally would not. For example, a shifted nanomagnet (i.e. due to lithographic variations) might cause part of a wire to switch before it normally should. Similarly, a shifted QCA molecule may cause a signal to be inverted, as discussed above.

Missing devices (δ), shown in Fig. 4.1d, cause nearest neighbor interactions to
be weaker, as interactions are distance dependent. Such defects could be more frequent in the molecular implementation because of self-assembly. Finally, if a device deviates from its ideal shape (ε), as shown in Fig. 4.1b, it may interact with its neighbors incorrectly. Devices realized by lithography (e.g., nanomagnets) are especially susceptible to this type of defect.

We now look at how the faults discussed above affect the logical behavior of the PLA cell design in [39]. The probability of a PLA cell exhibiting faulty behavior depends on the number of QCA devices in it that could cause the PLA cell to fail for a given QCA fault type. We use the variable $C_x$ to represent the number of failure points for each fault type $x$. In other words, $C_x$ represents the total number of sources susceptible to fault-causing defects in a PLA cell. As seen in Fig. 2.1b, there are 19 QCA devices in each PLA cell, and each is a potential point of failure. In the worst case, $C_x = 19$ for all fault types, and any fault that occurs in a PLA cell causes the entire row or column to fail. However, such a
worst case is too pessimistic. We consider a more realistic analysis of fault sources in the PLA cell based on the observation that certain regions of the PLA cell can withstand a fault and still operate properly. We use the PLA **AND** plane as an example (as the **OR** plane can be treated in much the same way). To facilitate this analysis, we divide the **AND** plane schematic into eight regions according to the basic functionalities of the QCA devices (see Fig. 2.1d).

By examining how each of the three possible faults would logically affect these regions, we were able to find cases in which a fault does not make the PLA cell inoperable. For example, if a stuck-at-1 fault occurs in the Select Wire region of a PLA cell that needs to be programmed into logic mode, the fault does not adversely affect the behavior, since the select bit still outputs a “1” as required. Also, a PLA cell operating incorrectly does not necessarily ruin the operation of an entire row.

Therefore, in a more realistic case, some of the susceptible sources can be discounted, as they do not lead to incorrect operations. We have determined

<table>
<thead>
<tr>
<th>Molecular</th>
<th>Magnetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defect</td>
<td>Fault</td>
</tr>
<tr>
<td>shifts</td>
<td>inversion</td>
</tr>
<tr>
<td>rotations</td>
<td>inversion</td>
</tr>
<tr>
<td>missing</td>
<td>ND</td>
</tr>
<tr>
<td>stray charge</td>
<td>stuck at</td>
</tr>
</tbody>
</table>
which of the PLA cell regions are susceptible to each fault type and which are not. This leads to different values of $C_x$ for each fault type. The reduced values of $C_x$ for the AND plane, based on the more realistic assumptions, can be found in Table 4.2. The 8 regions are listed in the first column. The number of QCA devices in each region is given in the second column. For each of the 3 fault types, the number of devices causing the faults are given in columns 3 to 7. Note that the stuck-at fault type column is the union of the stuck-at-1 and stuck-at-0 columns. Each region susceptible to the fault type is marked by putting the number of QCA devices into the table.

As an example, consider the ND fault (third column). Every region of the PLA cell except the Literal Wire region is susceptible to this fault because the PLA cell can be programmed to wire mode such that the fault in the Literal Wire does not affect the Implicant Wire. Since the PLA cell can withstand a fault in the Literal Wire, the entry in the table for that region is empty. For each fault, the numbers of cells are added to obtain the final $C_x$ value.

While the more realistic case above reduces the value of $C_x$ for each fault type, the considerations are still somewhat pessimistic. There are some techniques that we have not considered in our assumptions. For example, device-level redundancy [32, 56] can be used to make the PLA cell more defect resistant in molecular QCA, possibly preventing defects from causing faults. Also, we only considered one way to program around faults, by setting the affected PLA cell to wire mode. It is possible that some inversion faults could be fixed by reordering the input literals or re-writing the logic functions. Such techniques could significantly reduce fault probabilities and improve yield. We omit further discussion on this, and leave it for future work.
TABLE 4.2

NUMBER OF QCA DEVICES ($C_x$) SUSCEPTIBLE TO FAULTS

<table>
<thead>
<tr>
<th>QCA Device Location</th>
<th>Device Count</th>
<th>AND Plane Row Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>AND wire</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OR wire</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>literal wire</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>implicant wire</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>select wire</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>middle wire</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>AND gate</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OR gate</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>total devices ($C_x$)</td>
<td>19</td>
<td>16</td>
</tr>
</tbody>
</table>
4.1.2 Quantifying Defects and Faults

It is well recognized that defect rates will increase when dealing with devices that have nanometer feature sizes. For nano-scale QCA implementations, almost every QCA device is defective in that it deviates from its ideal shape, location or orientation to some degree. For example, in a lithography-based implementation of magnetic QCA, none of the nanomagnets have exactly the ideal rounded rectangle shape (see Fig. 4.1e). For molecular QCA, all devices could be slightly misaligned. Fortunately, both experiments and simulations indicate that fabrication variations must reach a certain severity before a defect occurs.

For the sake of quantifying defects and showing quantitatively how QCA device defect rates correspond to PLA cell “fault rates,” we have defined defect rates \( r_x \) for each of the 5 defect types \( x \) as shown in Fig. 4.1. The defect rate \( r_x \) is the probability that a defect of type \( x \) occurs at a certain QCA device. For the defect types discussed in Sec. 4.1.1 the device defect rates are defined as follows:

\[
\begin{align*}
    r_\alpha &= P(\text{a stray charge causes a defect per } nm^2) \\
    r_\beta &= P(\text{a shifted QCA device causes a defect}) \\
    r_\gamma &= P(\text{a missing QCA device causes a defect}) \\
    r_\delta &= P(\text{a rotated QCA device causes a defect}) \\
    r_\epsilon &= P(\text{a misshaped QCA device causes a defect})
\end{align*}
\]

where \( P \) denotes the probability. For the stray charge defect, we assume that the distribution of the stray charges is uniform.

In order to study the capability of QCA-based PLAs to implement arbitrary logic even in the presence of QCA defects, we use \( f_x \) to represent the probability of a fault occurring in a PLA cell due to defect type \( x \). If the device defect rate of defect type \( x \) is known, \( f_x \) can be readily computed. Consider the “device shift”
(β) defect, which, according to Table 4.1, most commonly leads to an inversion fault. Given the number of QCA devices susceptible to inversion faults in a PLA cell \(C_{\text{inversion}}\), the PLA cell fault probability can be calculated using Eq. 4.1:

\[
f_\beta = 1 - (1 - r_\beta)^{C_{\text{inversion}}}
\]

The computation of \(f_\gamma\), \(f_\delta\) and \(f_\epsilon\) can be done in the same way using the corresponding \(C_x\) value from Table 4.2. For stray charge defects, we compute the fault probability as follows:

\[
f_\alpha = 1 - (1 - A \ast r_\alpha)^{C_{\text{stuckat}}}
\]

where \(A\) is the effective area covered by a PLA cell. It is not sufficient to simply use the area of the QCA devices in a PLA cell, since a stray charge in the vicinity of the cell could still cause the PLA to malfunction. Furthermore, as there may be different sources of stray charges, the effective area and defect rate for each kind of charge may need to be evaluated separately, leading to a different fault probability. For this study, we assume that all defects types are independent. The overall PLA cell fault probability can then be evaluated as

\[
F_{\text{cell}} = 1 - \Pi_{x \in \text{all fault types}} (1 - f_x)
\]

Computing the probability of failure for each row and column in a PLA is straightforward once the fault probability for a PLA cell is known. It is the probability that all PLA cells in that row (or column) are fault free. This can be calculated using Eq. 4.4 where \(N\) is the number of PLA cells in the row (or column). From the fault probability calculations discussed here, it is clear that
the device defect rate, the $C_x$ values, and the number of PLA cells in each row and column of the PLA can significantly impact the PLA yield.

$$F_{\text{row}} = 1 - (1 - F_{\text{cell}})^N \quad (4.4)$$

### 4.1.3 Yield Trends

In this section, we discuss the behavior of the PLA yield for different device defect rates and PLA sizes. As in other PLA yield analysis work [23], we assume that redundant rows and columns may be used. Calculating the QCA PLA yield is straightforward using the fault probabilities discussed in Sec. 4.1.2. The yield of each plane in the PLA can be determined using an M-choose-N calculation based on the row and column fault probabilities. The yield of the whole PLA is the product of the yields of both the AND and OR planes.

We first examine the dependency of the fault probability and yield on the device defect rate. The fault probability for a single PLA cell grows exponentially as the device defect rate increases (Eq. 4.1 and 4.2). For different types of defects and also for PLA cells in the AND and OR planes, the growth rates can be different depending on the $C_x$ values in Table 4.2. The fault probability for a row or column grows even faster as defect rates increase due to the multiplicative effect of multiple PLA cells. Consequently, the yield of a PLA falls off quickly as the defect rate increases.

Fig. 4.2 illustrates the above behaviors with three different curves. Here, misshape defects ($\epsilon$) are considered. This defect type corresponds to the stuck-at fault type ($C_{\text{stuckat}} = 11$). In Fig. 4.2, the x-axis is the misshape device defect rate ($\epsilon$), while the y-axis shows both the fault probability and yield. The fast
growth of the fault probability can be seen from the dot-dash line depicting $f_\varepsilon$ for one cell. The rapid decline in the AND-plane yield can be seen from the curve drawn as a sequence of dots. In this case, each PLA row has 20 PLA cells. After the defect rate passes a certain threshold, the yield quickly drops off. To quantify this phenomenon, we define the “cutoff defect rate” to be the highest defect rate that gives a yield of \( \geq 90\% \), represented by a vertical line.

Defect rate is not the only factor that has a significant effect on yield. The number of inputs and outputs in each plane must also be considered. Fig. 4.3 shows how the dependency of the AND-plane yield on the defect rate changes as
the number of inputs changes while the number of rows (i.e., the output of the AND-plane) is fixed. Here, the misshapen defect is considered, the x-axis indicates changing device defect rate, and the y-axis shows the yield. The three yield curves correspond to 2000, 200, and 20 input columns from left to right, respectively. As the number of inputs increases, the yield curve shifts to the left and the cutoff defect rate decreases. In fact, the cutoff defect rate appears to be a linear function of the number of inputs.

Given the number of inputs, implicants, and outputs of the intended logic
functions, the designer must select the actual size of the PLA (i.e., the number of rows and columns) in order to ensure a predetermined yield. The selection is critically dependent on the device defect rates. The fault probability and yield analysis discussed above facilitates the selection process. We illustrate this with Fig. 4.4 which plots the number of rows needed to achieve 90% yield as a function of the defect rate. Again, misshape defects are considered. Three different curves are depicted corresponding to 5, 10 and 20 required implicants from bottom to top, respectively. As the defect rate increases, the number of required rows also increases. The curves begin to grow rapidly as the defect rate approaches the cutoff point. The curves in Fig. 4.4 are determined using an N-choose-M calculation where N is the number of rows that need to be fault free out of M total available rows (some of which could be faulty).

4.1.4 Refined PLA Fault Model

The information given in Table 4.2 provides a good starting point to estimate the effects of each fault type on the operation of a QCA-based PLA cell. However, there are only two possibilities for the PLA cell – faulty or not faulty. As it turns out, a fabricated QCA PLA cell can be faulty with one of three different operational faults. In the worst case the QCA PLA cell is totally unusable and thus it causes the entire row or column to be unusable as well. This is a fully faulty QCA PLA cell, and it cannot be programmed into wire or logic mode successfully. It is also possible for a QCA PLA cell to be partially faulty. In such a case, the PLA cell can only be programmed into wire mode or logic mode, but not both. A partially faulty PLA cell is either called stuck-in-wire-mode or stuck-in-logic-mode, depending on the faulty behavior of the PLA cell.
Figure 4.4. This graph shows the number of actual PLA rows required to implement a certain number of implicants as a function of the device defect rate (DDR). The AND plane has 20 columns.

To continue the work in the previous section, and in order to provide a better approximation of the effects of each fault on the operation of an entire PLA, we reanalyzed the AND plane cell that had been divided into eight different operational regions (See Fig. 2.5d). We focused on the operation of the AND plane PLA cell for each region and fault type. Table 4.2 only indicated if a device fault would lead to a fully faulty PLA cell, or if it would not lead to a fully faulty PLA cell. Instead of two possibilities, we instead considered which of four possible states the PLA cell would be in if a fault occurred. Considering all fault types, a PLA cell would either be fully faulty, stuck-in-wire-mode, stuck-in-logic-mode, or fully

57
operational. The new results are summarized in Table 4.3. This information can be used later to assist with creating a fault map and will be necessary to perform mapping tests on randomly defective PLAs. For now, this work remains largely implementation independent, however it should be a useful starting point for work that investigates logic mapping for QCA PLAs of a specific implementation.

4.1.5 Special Case Faults

Further study has led to an additional consideration in the context of PLA cell faults. In most cases, if a PLA cell is fully faulty, it often causes the entire row in the AND plane or the entire column in the OR plane to be unusable as well. For example, if there is a stuck-at-0 fault in the implicant wire of an AND plane PLA cell, then no matter what the inputs are, the AND gates along the corresponding implicant row will always evaluate to ‘0’. Therefore, the entire row is faulty and cannot be used to map logic. There are, however, cases of fully faulty PLA cells that do not necessarily lead to a failure in the entire row or column. For example, if there is a stuck-at-1 fault in the implicant wire of an AND plane PLA cell, all previous evaluation will be lost (since it will always evaluate to ‘1’), but any evaluation after the fault will not be affected. In this second case, the affected PLA cell is fully faulty, but later PLA cells are still usable.

To state this in another way, some device faults lead to PLA cells that are fully faulty. Fully faulty PLA cells cannot be programmed to either logic or wire mode. Even though all instances of fully faulty PLA cells cannot be utilized in a PLA, some fully faulty PLA cells cause an entire line to be faulty, while others only cause part of a line to be faulty. Therefore it is possible to improve the fault table to include partial salvage of a row or column for some fully faulty PLA cells,
TABLE 4.3


<table>
<thead>
<tr>
<th>QCA Device Location</th>
<th>AND Plane Row Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>AND wire</td>
<td>F</td>
</tr>
<tr>
<td>OR wire</td>
<td>F</td>
</tr>
<tr>
<td>literal wire</td>
<td>W</td>
</tr>
<tr>
<td>implicant wire</td>
<td>F</td>
</tr>
<tr>
<td>select wire</td>
<td>F</td>
</tr>
<tr>
<td>middle wire</td>
<td>F</td>
</tr>
<tr>
<td>AND gate</td>
<td>F</td>
</tr>
<tr>
<td>OR gate</td>
<td>F</td>
</tr>
</tbody>
</table>
depending on the type and location of the device faults. These changes are shown in Table 4.4. Fully faulty PLA cell faults that only cause partial line faults are indicated with a ‘P’ in the table. We call these partial-line PLA cell faults.

In Table 4.3, stuck-in-wire-mode and stuck-in-logic-mode PLA cell faults were introduced. These were added to the table in places that were previously considered not faulty. This makes sense since PLA cells that are stuck in wire or logic mode can still be used to map logic. In Table 4.4, however, the newly introduced fault type takes the place of faults previously considered fully faulty. Again this makes sense, since partial-line faults are fully faulty in themselves. There are three cases of device faults in a PLA cell that can lead to partial-line faults. For AND Plane cells, stuck-at-1 and inversion faults in the implicant line, as well as stuck-at-1 faults at the AND gate can lead to partial-line faults. Again, it is important to note that this analysis is intended to be implementation independent. When considering a specific implementation of QCA, the fault model should be refined to fit that implementation.

The inclusion of this PLA cell fault type gives us a more complete connection between defects and PLA cell faults. It allows for a better understanding of faulty behavior, and should lead to a better utilization of all available resources in a faulty QCA-based PLA structure once the details are incorporated into the mapping process. Sec. 5.2.2 shows how this new PLA cell fault type will affect the representation of PLA structures used for mapping logical functions.

4.2 Process Variations and Defects in Nanomagnets

In this section, we begin to examine defects and faults specific NML. NML devices are simply three-dimensional blocks made from one layer of a magnetic
TABLE 4.4

EFFECTS OF DEVICE FAULTS ON AND PLANE PLA CELL OPERATION. IF THE OPERATION OF THE PLA CELL IS UNAFFECTED, THERE IS A '.', LOGIC MODE IS INDICATED BY 'L', WIRE MODE IS INDICATED BY 'W', A FULLY FAULTY PLA CELL THAT CAUSES ENTIRE LINE FAULTS IS INDICATED BY 'F', WHILE A PARTIAL LINE FAULT IS INDICATED BY 'P'.

<table>
<thead>
<tr>
<th>QCA Device Location</th>
<th>AND Plane Row Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>AND wire</td>
<td>F</td>
</tr>
<tr>
<td>OR wire</td>
<td>F</td>
</tr>
<tr>
<td>literal wire</td>
<td>W</td>
</tr>
<tr>
<td>implicant wire</td>
<td>F</td>
</tr>
<tr>
<td>select wire</td>
<td>F</td>
</tr>
<tr>
<td>middle wire</td>
<td>F</td>
</tr>
<tr>
<td>AND gate</td>
<td>F</td>
</tr>
<tr>
<td>OR gate</td>
<td>F</td>
</tr>
</tbody>
</table>
material (e.g., permalloy or supermalloy). Conventional, imprint, or electron-beam lithography (EBL) are all candidate fabrication mechanisms. EBL was used to make the nanomagnets for the NML majority gate and wire experiments discussed in [42]. Although EBL is well accepted for patterning at the nanometer regime, it can introduce many variations that may effect the functionality of an NML-based circuit: (a) Over-exposure and under-exposure could lead to incorrect shapes. (b) Irregularities in the EBL gun can lead to “shot noise” that can cause rough edges. (c) The electrons that make up the beam are subject to scattering effects that can cause exposure of the resist outside of the desired shapes [41]. This can lead to bulging around the edges. (d) If the spacing between shapes is small, the resist in the gap may fail completely during development, leading to the merger of two or more shapes. The thin resist walls can also bend, creating nanomagnets with bulges and indentations [40]. (e) With liftoff, a pattern is made in resist and metal is evaporated over it. When the resist is dissolved, the metal that previously resided on the top surface is washed away, or lifted off. However, the deposited material can clump, and the liftoff process might remove a large chunk from the edges of the rounded rectangle shapes.

Scanning Electron Microscopy (SEM) images of nanomagnet lines from [43] (see Fig. 4.5(a)) illustrate some of these variations. While it is difficult to get precise information about the entire 3D nanomagnet shape from this top-down view, the images indicate that the edges of the nanomagnets are missing magnetic material. The SEM image in Fig. 4.5(b) shows a horizontal wire next to a vertical wire. The magnetic material bulges between the two wires. Similar behavior has been observed for NML nanomagnets that are placed close together.

Here, we consider how missing magnetic material, bulges and “shifts”, and
edge roughness affect how well a magnet represents and propagates binary state. Important to this discussion is how the magnetization (binary state) associated with a previous computation is “removed,” by nulling the nanomagnets. This is essential as it allows the nanomagnets that make up NML circuit elements to be re-evaluated with new inputs. Fabrication variations can make this process more difficult – which can in turn lead to faulty behavior and undesirable logical states. We leverage micro-magnetic simulations of supermalloy nanomagnets (the material used in [16]) performed with the OOMMF simulation suite [26] to study these issues. As in [20], we assumed a saturation magnetization of $8.0 \times 10^5$ A/m, an exchange stiffness constant of $1.05 \times 10^{-11}$ J/m, an anisotropy constant $K_1$ of $3$ J/m$^3$, and the default damping constant of 0.5. Each simulation stage (when the magnitude or direction of the applied field changes) was considered complete when the maximum $|\frac{d \mathbf{m}}{dt}|$ dropped below a preset number of degrees per nanosecond.
We note that OOMMF is widely used and there is excellent correlation between simulation and experimental results (see [49, 71] for examples).

4.2.1 A Base Case

Before looking at defective systems, we first consider a simulation of a non-defective magnet to provide a basis for comparison – a 60x90x30nm magnet (as in [52]), in isolation, where the y-component of magnetization was initially strongly negative (see bottom inset in Fig. 4.6). We then applied an external magnetic field ($H_{\text{clock}}$) along the magnet’s hard axis that continuously increased in magnitude to simulate the effects of a clock. A $1.5 \times 10^4$ A/m biasing field was simultaneously applied along the magnet’s easy axis. The biasing field is necessary as, after we null the magnet ($M_y = 0$), we want to “tip” it to the opposite polarization. Without a biasing field, the magnet would randomly tip up or down as the field applied along its hard axis was removed. This simulation determines the external field required to null this magnet given this local bias.

The “No Defects” curve in Fig. 4.6 illustrates how the y-component of magnetization changes as a function of the magnitude of the external field. As the magnitude of the external field ($H_{\text{clock}}$) increases, the magnitude of its y-component of magnetization decreases – eventually reaching 0 when the external field is approximately $0.5 \times 10^5$ A/m. The nanomagnet is now nulled and is tipped up by the biasing field. Note that if the magnitude of $H_{\text{clock}}$ continues to increase, it will work to keep the magnet nulled (see the “tail” of the “No Defects” curve). This explains why after the magnet initially changes its polarization, the y-component of its magnetization trends back toward 0 – the external field can be too strong and works to re-null the magnet. When $H_{\text{clock}}$ returns to 0, the y-component of
Figure 4.6. $H_{\text{clock}}$ vs. $M_y$ for a 60x90nm magnet with no fabrication variation and a 60x90nm magnet with a “slanted” edge. $H_{\text{clock}}$ required to null the slanted magnet is greater than that for the perfect magnet.

magnetization is strongly positive. After the magnet has changed polarizations, we again applied an external field along the magnet’s hard axis and a biasing field of the same magnitude (but in the opposite direction) to return the magnet to its original state. The direction of the nulling field should not impact the magnitude of the field required to null the magnet. For a magnet with no fabrication variations, this is in fact the case as seen in Fig. 4.6.

4.2.2 Missing Magnetic Material

Now consider a simulation similar to the one above, but this time with magnetic material removed from the top-right of the magnet – a possible fabrication
Figure 4.7. $H_{\text{clock}}$ vs. $M_y$ for the 4th magnet in a 7-cell line. A larger $H_{\text{clock}}$ is required to facilitate a ↓ to ↑ transition than a ↑ to ↓ transition when compared to a wire with no missing magnetic material.
variation as seen in Fig. 4.5(a). While the same biasing field was used in this simulation, the external field was applied in the same direction for both state transitions. These simulation results are illustrated in Fig. 4.6. We consider the down-to-up transition first (see middle inset in Fig. 4.6). Note that a stronger external field is required to null this magnet (approximately $0.6 \times 10^5$ A/m instead of $0.5 \times 10^5$ A/m). Magnetic moments tend to align along a magnet’s edge. In this simulation, the placement of the slant and the direction of the applied external field help to reinforce the initial downward polarization ($\downarrow$). For this same reason, the up-to-down transition (see top inset in Fig. 4.6) can be accomplished when the magnitude of $H_{\text{clock}}$ is lower (approximately $0.4 \times 10^5$ A/m). (Only the first portion of this curve is shown – i.e. until the magnet is nulled – to improve graph readability.)

We now consider the above results in the context of a simple circuit – in this case a line of 7, 60x90x30nm magnets spaced 16nm apart (see inset at lower left in Fig. 4.7). The line was initialized to a logically correct, antiferromagnetically coupled ground state and the y-component of magnetization in the first magnet was switched in order to simulate a new input to this line. $H_{\text{clock}}$ was then applied along the magnets’ hard axes and increased from 0 A/m to approximately $4.0 \times 10^4$ A/m in approximately $1.5 \times 10^4$ A/m increments. The field was then allowed to relax back to 0 A/m in the same manner.

The results of three simulations are summarized in Fig. 4.7. We plot the magnitude of $H_{\text{clock}}$ versus $M_y$ for the 4th magnet in the line – as $M_y$ represents binary state. We first considered a wire with no defects. In this plot, one can see

---

$4.0 \times 10^4$ A/m is the magnitude of $H_{\text{clock}}$ that a clock wire in [52] might produce assuming a current density of $10^7$ A/cm$^2$ – and was considered to be to a practical upper limit. Realistically, placing nanomagnets closer to one another can increase coupling and lower the magnitude of $H_{\text{clock}}$ required to null the magnets.
that $H_{\text{clock}}$ was approximately $2.0 \times 10^4$ A/m when the 4th magnet was nulled.
When a misshapen magnet is introduced into the line (insets show magnets 3, 4, and 5), as before, the external field required to null the misshapen magnet can increase or decrease depending on the magnet’s initial state. As seen from the inset at the top left of Fig. 4.7, the field required to null the defective magnet is actually about 20% lower if magnets 3 and 4 are initially ↓ and ↑. However, if magnets 3 and 4 are initially ↑ and ↓, the magnitude of $H_{\text{clock}}$ must be about 14% higher in order to “reuse” this system to correctly evaluate a new input.

The above simulations were repeated with magnets that became progressively more misshapen. These results are summarized in Fig. 4.8. As fabrication variation gets worse, the misshapen magnet is in fact never nulled. While $H_{\text{clock}}$ begins to cause the 4th magnet’s y-component of magnetization to trend toward 0, the missing magnetic material/slant helps to re-enforce its initial state. Moreover, as the amount of magnetic material missing from the edge of the 4th magnet increases, it becomes more difficult to null when $H_{\text{clock}}$ has a similar magnitude. After $H_{\text{clock}}$ reaches $4.0 \times 10^4$ A/m and relaxes back to 0 A/m, the 4th magnet settles back into its old (and now logically incorrect) polarization.

This suggests that we can leverage the external drive circuitry to help tolerate the effects of fabrication variation. Increasing the magnitude of $H_{\text{clock}}$ might increase the likelihood that the nanomagnets will behave in a logically correct manner. A stronger external field makes it easier to remove the state associated with a previous computation. However, more reliability comes at the expense of an increase in system energy. If $H_{\text{clock}}$ is never strong enough to remove the state associated with the previous computation, missing magnetic material can induce stuck at faults. That said, whether or not a wire (for example) is stuck at 0 or is
Figure 4.8. With more missing material, it becomes more difficult to null the nanomagnet as evidenced by the $y$-component of the defective magnet’s magnetization.
Figure 4.9. If $H_{\text{clock}}$ is applied to a magnet with missing magnetic material from right- to-left, a ↑ to ↓ transition becomes more difficult.
stuck at 1 can depend on (a) where on the magnet material is missing from, and (b) what polarization state a group of magnets was initially in. For the example discussed above, if the misshapen magnet initially has a positive component of $M_y$, it can easily make a $\uparrow$ to $\downarrow$ transition. However, a $\downarrow$ to $\uparrow$ transition is more difficult suggesting that a “stuck-at-down” is most likely.

The simulations discussed thus far have assumed that $H_{\text{clock}}$ is applied unidirectionally. However, as noted in [52], $H_{\text{clock}}$ could also be sinusoidal. This could be advantageous from the perspective of minimizing overall system energy as the clock generation circuitry could be derived from the designs for adiabatic clocks [12, 74]. Moreover, as seen in Fig. 4.6 for a magnet with no fabrication variations, the magnitude of $H_{\text{clock}}$ (not the direction) is of most importance when working to null a given magnet. However, if the magnet is missing magnetic material, this is not necessarily the case.

In Fig. 4.9, we illustrate the results of 2 simulations designed to determine how an external parameter – the direction of $H_{\text{clock}}$ – might affect the logical correctness of some physical structure – a configuration of nanomagnets with fabrication variation. A setup is used that is similar to that in Fig. 4.6. The first half of this simulation is essentially identical to the $\downarrow$ to $\uparrow$ transition captured in Fig. 4.6. However, when considering the $\uparrow$ to $\downarrow$ transition here, the external field was this time applied from right-to-left. If $H_{\text{clock}}$ is applied in this manner, the magnitude of the external field must increase ($0.6 \times 10^5 A/m$ vs. $0.4 \times 10^5 A/m$) to support the $\uparrow$ to $\downarrow$ transition.

These results are significant in NML circuit design. Consider a line of 3 nanomagnets where the middle magnet has missing magnetic material (i.e. as in Fig. 4.9). If the initial state of this line is $\uparrow\downarrow\uparrow$ and $H_{\text{clock}}$ is applied from left-
to-right, the third magnet will be more likely to end up in a “stuck at up” state per the discussion above. However, if the initial state of the line is $\downarrow\uparrow\downarrow$ and $H_{\text{clock}}$ is applied from right-to-left (with data still flowing from left-to-right), the third magnet is likely to end up in a “stuck at down” state. Essentially, a stuck at fault can occur, but it may change between stuck-at-up or stuck-at-down. *The stuck-at value is determined by the previous state of the magnets, the location of missing material, the direction of $H_{\text{clock}}$, and can change based on what inputs are applied.*

4.2.3 Bulging Nanomagnets

Above we looked at how missing edges might affect a magnet’s logical state and dataflow. We now consider what might happen if magnets “bulge.” Because bulging usually occurs where lithographically defined shapes are adjacent and close to each other, most nanomagnets in a wire will see similar bulging from neighbors on their left and right. To consider the effects of fabrication variations that manifest themselves as bulges, we simulated lines of 60x90x30nm, 65x90x30nm, and 70x90x30nm magnets. For sufficiently strong nulling fields, both up and down inputs were successfully propagated down each of the aforementioned lines. However, it is more revealing to examine how bulges affect the quality of a signal’s binary state by considering an M-H plot analogous to Fig. 4.6.

Specifically, Fig. 4.10 plots the y-component of magnetization against $H_{\text{clock}}$. (Again, a $1.5 \times 10^4$ A/m bias is used.) As one can see, the magnet with a smaller aspect ratio is actually easier to null. For example, the magnitude of $H_{\text{clock}}$ when $M_y = 0$ for the 65x90nm magnet is lower than that for the 60x90nm magnet. This would seemingly indicate that the lower aspect ratio magnets resulted from bulging are an asset as the current density in a clock wire could decrease. While
in some sense this is true, there are consequences when moving toward magnets with smaller aspect ratios.

When studying a line or “wire” of nanomagnets, as a magnet’s aspect ratio decreases, the magnitude of the y-component of magnetization for each individual magnet in the line decreases as well. Magnets tended to couple with one another in the x-direction more readily. From the perspective of binary logic, this represents a “weaker” 1 or 0. While one might look at Fig. 4.10(a) and note that when $H_{\text{clock}}$ was 0, $M_y$ was strong, this is only because the global bias field was applied continuously throughout this simulation. To better quantify the y-component of magnetization as a function of aspect ratio, we initialized 60x90, 65x90, and 70x90nm magnets in isolation such that $M_y$ was strongly negative and then let each magnet relax to a ground state with no applied biasing field.
We report the average Y component of their magnetization in the presence of zero bias (normalized to ±1) in Fig. 4.10(b). We averaged $M_y$ over all of the simulation mesh points and also averaged $H_y$ 4nm away from the magnet. As the aspect ratio decreases, the magnitude of the y-component of the field below the magnet decreases. This makes sense as the magnetization in the x-direction increases as aspect ratio decreases. Logically, as aspect ratio decreases, the y-component of magnetization decreases – resulting in a potentially “weaker” 0 or 1 and a “weaker” driver due to a lower biasing field. A strong component of $M_y$ is desirable as it can help facilitate other circuit structures (i.e. the driver to a crossover structure).

4.2.4 Shifted Nanomagnets

We now consider what might happen if the space between adjacent nanomagnets in a line is non-uniform. This will bring 2 nanomagnets closer together and move two others farther apart as seen in the insets of Fig. 4.12. (With no variations, each nanomagnet would be 16nm away from its neighbors.) In these simulations, we will consider $M_y$ as a function of the x-component of the local magnetic field seen in the gap near the surface of a nanomagnet (see inset in Fig. 4.11). This measurement ($H_{x-local}$) is more insightful as it also incorporates the contribution of the magnetic material itself. One way to interpret this data is that in order to reuse a magnet to evaluate new inputs, the 60x90x30nm magnet needs to “see” this value of $H_{x-local}$. Depending on the size and shape of the surrounding magnetic material, a given value of $H_{x-local}$ may be generated with a higher or lower external field. (Note the difference in Fig. 4.11)

We leverage $H_{local}$ to consider how spacing variations between nanomagnets
Figure 4.11. $M_y$ as a function of $H_{\text{clock}}$ and $M_y$ as a function of $H_{\text{local}}$ in the presence of a $1.5 \times 10^4$ A/m biasing field. Note that the field seen at the field near the surface of the magnet is greater than the applied field. This will be an important measure when considering why a circuit does or does not show logically correct behavior.
Figure 4.12. Local field vs. $M_y$ for magnets in a wire. Note that magnet needs to see a similar local field to null but at the expense of a larger global field.
might affect dataflow in a 7 cell line with no missing magnetic material. Simulation results are illustrated in Fig. 4.12 where we plot $M_y$ of the 5th magnet from the left versus the local field 8nm to the left of the 5th magnet. We begin by examining the 16nm curve – which is essentially a wire with no spacing variation between the magnets. Note that when the magnet is nulled, $H_{x-local} = 2.8 \times 10^5$ A/m. We next examine the curve for the simulation with a 20nm spacing between the 4th and 5th magnets. Again, the 4th magnet is successfully nulled and that the wire functions correctly. However, while $H_{local}$ or $H_{x-local}$ is nearly identical to the 16nm case, the corresponding value of $H_{clock}$ when magnet 5 was nulled was greater than the magnitude of $H_{clock}$ for the 16nm case – indicating that a stronger clock/more energy is needed to null magnet 5 due to weaker coupling with the adjacent magnetic material. Looking at the curves for 24 and 28nm spacing between the 4th and 5th magnets, the magnitude of $H_{clock}$ is not sufficient to null all of the magnets in a line and a stuck-at state ensues.

4.2.5 Edge Roughness and Magnetic Material

A magnet’s edge roughness and the magnetic material used to make it can also affect circuit functionality. While we have not given all the details here, we do highlight two observations obtained from simulation results. First, as a magnet’s edge roughness increases, the magnitude of $H_{clock}$ required to null it generally increases as well (when compared to simulation results described by Figs. 4.6 and 4.9). Second, simulation results indicate that magnets made with a harder magnetic material (i.e. permalloy instead of supermalloy) may be more difficult to null, but also provide a stronger, local biasing field. Depending on the type of defect, sometimes a stronger local biasing field can allow fault free operation with
a lower nulling field (as an individual magnet can provide more local control over nearby devices).

4.3 Fault Detection Scheme

In order to utilize the fault tolerance potential in NML-based PLAs, faults must first be detected so that faulty resources can be avoided and fault-free resources can be utilized. A basic fault detection scheme was proposed in [39]. The basic idea of fault detection given in [39] for QCA-based PLAs is hierarchical in nature. First, the long wires that run across the entire PLA should be tested for faults. Each column and row of the PLA will be tested for wire faults. Once these high-level tests are complete, the individual PLA cells can be tested for faults by examining all PLA cells in the same column or row until the entire PLA has been explored. For the purposes of illustration, we have given an example showing how fault detection might work in a small section of an NML PLA.

In Fig. 4.13, a small section of the AND plane of a NML PLA is shown. Wires leading into and out of this section are marked by the letters A through H. Three possible locations of faults are indicated by letters X through Z. Suppose that a stuck-at-1 fault exists at location X. This can be detected by driving a binary ‘1’ and ‘0’ into wires A and B, and then examining the wire state at C and D. This process can test all horizontal wires in the AND plane in parallel. To test the staircase wires that flow from top-left to bottom-right, each PLA cell should be programmed into wire mode. Then, both binary values can be driven into wires E and F, and the result observed at G and H. These steps would detect a fault at location Y, for example.

To detect faults inside the PLA cells (like at location Z) a few more steps are
needed. All the PLA cells in one row can be tested at the same time. All PLA cells should be programmed to wire mode, except the cells in a single row, which should be programmed to logic mode. To find a fault at location Z, the second row of PLA cells should be programmed into logic mode. Then, a binary ‘1’ should be driven into wires E and F while at the same time driving both ‘1’ and ‘0’ into wire B. Wires G and H should be observed to check for the correct result values. If there was a fault at Z, the values seen at H would be incorrect for one or more of the inputs given at B.

While this is a working fault detection scheme, the analysis done here does not
fully explain how finding faults in one location will affect fault detection in other locations. We hope to devise a more general and complete fault detection scheme in our future work.

4.4 Defect Tolerance

Reconfigurable logic offers another way to avoid faults via architectural-level redundancy. For example, the PLA structure in [39] can be expanded to include more rows and columns such that defective crosspoints and/or interconnect can be avoided – increasing the probability that the desired set of logic functions can be mapped onto the faulty PLA. However, for NML, a larger PLA not only means a larger chip area, but also more/longer clock wires to operate the logic and interconnect. Redundancy in an NML PLA provides a way to trade power consumption and area for fault tolerance.

Consider the yield versus defect rate study in Sec. 4.1.3. This study indicates that a yield of 90% is possible given defect rate of $10^{-3}$ and 10% redundancy. However, if the defect rate increases to $10^{-2}$, 400% redundancy is required. As seen in Sec. 4.2, increasing the magnitude of $H_{\text{clock}}$ provides another level of flexibility to tolerate defects. According to [52], the magnitude of $H_{\text{clock}}$ increases linearly with the current density and current in a clock wire. However, the power increases with the square of the current. Therefore, doubling the magnitude of $H_{\text{clock}}$ implies that current will double while the power will quadruple. Thus, from the standpoint of performance and logical correctness, it is an interesting optimization problem to determine the most effective usage of the above mechanisms for defect tolerance. Together these techniques could allow for tolerance of a higher defect rate than
can be achieved by either individually. However, one technique might be sufficient to provide the required defect tolerance with the smallest increase in power.

To further quantify the tradeoffs discussed above, we worked with the ALU4 benchmark from the Toronto 20 benchmark suite [9]. Assume that the defect rate in a given NML PLA is normally $5 \times 10^{-4}$ with a clocking field of $2.0 \times 10^4$ A/m, and an increase in the clocking field to $2.4 \times 10^4$ A/m leads to a lower defect rate of $2.5 \times 10^{-4}$. With these assumptions in mind, we mapped the ALU4 benchmark to a PLA structure. For the higher defect rate ($5 \times 10^{-4}$), the area needed to successfully map the ALU4 benchmark to the PLA with a yield of 96% is approximately 68,500 $\mu m^2$. For the lower defect rate ($2.5 \times 10^{-4}$), a PLA with an area of only 24,500 $\mu m^2$ is sufficient to map the benchmark with 96% yield. These area estimates are based on 60x90nm magnets with 20nm spacings and wires crossings as described in [52].

To estimate the power consumption, the clocking wire design given in [52] is adopted, where the wire has a width of 2 $\mu m$ and a length of 4 $\mu m$ for an area of 8 $\mu m^2$. This means that the larger PLA will require 8,563 wires while the smaller PLA will require 3,063 wires. However, for the smaller PLA, the clocking field needs to be increased by 20% from $2.0 \times 10^4$ A/m to $2.4 \times 10^4$ A/m. This 20% increase in field strength means a 20% increase in current, and a 44% increase in power consumption. Therefore, if the power consumed by one clocking wire region is $P$, then the power consumption for the larger PLA with the lower clocking field is $8563P$. The power consumption for the smaller PLA with the higher clocking field is $4411P$. In this case, a 96% yield can be achieved with less power consumption if $H_{clock}$ is increased so that a smaller PLA can be used to map the logic. We are continuing to study such tradeoffs in our future work.
4.5 Summary

In this chapter we have given a detailed summary of how QCA defects (including NML) can lead to faulty behaviors in general QCA circuits and in the QCA PLA cells. Past work has done a good job of providing insightful work relating to electrostatic-based QCA defects. Our work builds on that to provide an additional in-depth study of NML defects to fill in the gaps. From the NML defect study, we have built up a fault model for QCA PLAs that captures how the different types of QCA faults (including stuck-at, non-deterministic, and inversion faults) affect the operation of the QCA PLA cell (stuck-in-logic, stuck-in-wire, and full faulty PLA cells with and without partial line faults). We have quantified the QCA defects and PLA faults as part of a yield study, and have laid the groundwork for mapping studies for QCA PLAs. Also, a fault detection scheme is presented that shows how PLA cell faults can be discovered inside a PLA array.
CHAPTER 5:

PLA MAPPING

In terms of redundancy and resource allocation in programmable logic, there was a great deal of research effort put into MOSFET PLA repair in the 1980s and early 1990s. The idea of mapping to PLAs was not as developed. For the repair process, redundant rows were introduced to replace already-programmed rows that were defective. There were a few research efforts that did look at fault detection and made strides towards full mapping. In [24], a field-programmable PLA (FPLA) was designed with fixed inputs and outputs, but the implicant terms had full arrangement flexibility. Using a bipartite graph representation of the FPLA, a matching algorithm could be used to determine successful mappings of the desired Boolean functions. Beyond that, most of the mapping and resource allocation work moved on to FPGAs. However, the popularity of re-programmable PLAs is increasing again in the area of emerging nanotechnologies. It is here that mapping research will be applied to PLAs.

Among the advantages of a reconfigurable PLA architectures is the ability to map any Boolean functions to the PLA that will fit in the rows and columns of the structure, adding redundancy if some of the resources are defective. To do this, some sort of resource allocation method must be used to map arbitrary logic to the PLA. We started creating a QCA PLA mapping method by studying a technique that was previously applied to nanowire crossbar PLAs, and then adapted it to
work with QCA-based PLAs. This technique works for PLAs with and without defects. We performed many mapping tests on thousands of randomly generated PLA structures using many differently sized benchmarks. For these tests, we created PLA arrays that contained randomly assigned PLA cell faults based on the QCA PLA fault model studies performed in Sec. 4.1. The frequency of occurrence of these faulty PLA cells comes from a given **effective defect rate (EDR)**, which is defined in Sec. 5.3. We performed the mapping tests on PLA structure with many different EDRs to determine the maximum EDR for which the benchmark could be successfully mapped to the PLA structure at least 90% of the time. We found that in mapping to defective PLAs larger benchmarks would require lower EDRs to be successfully mapped as compared to smaller benchmarks. One natural extension of this idea was that by breaking up large Boolean functions into many smaller functions, the entire benchmark could be successfully mapped onto systems of smaller PLAs at higher rates.

5.1 Existing Mapping Techniques

In this section, we first briefly review existing PLA mapping research, and then discuss one specific PLA mapping approach in more detail as it forms the basis of our mapping work for QCA PLAs.

In [61], a resource allocation method was proposed for nanowire crossbar PLAs. This nanowire crossbar architecture was similar to the one discussed in [36]. At a high level, this approach consisted of two major steps: (i) model the given Boolean function and the PLA structure as two graphs, and (ii) determine a graph monomorphism matching between the two graphs. This approach allowed for rearrangement flexibility of inputs, outputs, and implicant terms, resulting in more
Figure 5.1. Logic Graph Representation. The logical functions $S = WX + XZ$ and $T = WX + WY'$ are represented in graph form.

possible mappings and higher yields when compared to the mapping algorithm given in [24]. However, more rearrangement flexibility leads to a more computationally intensive mapping algorithm. While other research projects attempted to reduce computation time through different heuristics in the mapping algorithm [58], our goal is to improve the model to identify more possible mappings. We review the work from [61] in more detail below since it is closely related to our work.

The graphs used to model the sum-of-product Boolean functions and the PLA structure both take the form of back-to-back bipartite graphs. More specifically, there are three layers of vertices $S_1$, $S_2$, and $S_3$. Edges are directed and only allowed from vertices in $S_1$ to $S_2$ and from $S_2$ to $S_3$.

For the graph representation of Boolean functions, which will be referred to as the Logic Graph (LG), vertices in $S_1$ correspond to input literals, vertices in $S_2$ correspond to implicants, and vertices in $S_3$ will correspond to output functions. Edges between the vertices in $S_1$ and $S_2$ represent the combination of literals into
implicants using the AND Boolean operation. Edges between vertices in $S_2$ and $S_3$ represent the combination of implicants into functions using the OR Boolean operation. As an example LG, consider the two following sum-of-product functions: $S = WX + XZ$ and $T = WX + WY'$. The graph representation of these two logic functions is shown in Fig. 5.1. For the representation of Boolean functions, there are no extra edges or vertices beyond the minimum needed to represent the logic.

The graph for the nanowire crossbar-based PLA structure, referred to as the **Crossbar Structure Graph (CSG)**, captures both redundant nanowires and defects in the model. To create a CSG from a PLA structure, information must first be obtained from a fault-detection procedure. If there are no defects in the nanowire crossbar structure, the representative back-to-back bipartite graph would have vertices in $S_1$ equal to the inputs, vertices in $S_2$ equal to implicant rows, and vertices in $S_3$ equal to outputs in the PLA. In addition, the graph would be a complete bipartite graph.

However, large collections of nanowires are unlikely to have zero defects. There are three main defects that affect the nanowire crossbar: broken nanowires, stuck-open crosspoints, and stuck-closed crosspoints. Since each defect causes certain resources to be faulty, they must be reflected in the CSG. As an example, Fig. 5.2a illustrates a defective PLA with 5 inputs, 5 implicant rows, and 3 outputs. There are two broken nanowires shown as broken lines. There are two stuck-open crosspoints indicated by an ‘X’ for each. Finally, there is one stuck-closed crosspoint shown as a dark square.

To create a CSG that represents a defective PLA structure, a complete bipartite graph is initially used but certain rules are applied to remove edges and vertices
Defects and Faults in QCA-Based PLAs

Michael Crocker, X. Sharon Hu, and Michael Niemier

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
Email: {mcrocker, shu, mniemier}@nd.edu

Defect tolerance will be critical in any system with nanoscale feature sizes. This paper examines some fundamental aspects of defect tolerance for a reconfigurable system based on Quantum-dot Cellular Automata (QCA). We analyze a novel, QCA-based, Programmable Logic Array (PLA) structure, develop an implementation independent fault model, and discuss how expected defects and faults might affect yield. Within this context, we introduce techniques for mapping Boolean logic functions to a defective QCA-based PLA. Simulation results show that our new mapping techniques can achieve higher yields than existing techniques.

Categories and Subject Descriptors: B.6.2 [Hardware]: Logic Design—Reliability and Testing, Redundant Design

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Nanotechnology, Quantum-dot Cellular Automata, Defects, Faults, Logic Mapping

1. BACKGROUND

![Figure 5.2. (a) A defective and redundant nanowire crossbar structure. The AND plane is 5x5 cells, and the OR plane is 3x5 cells. There are 5 faults (2 stuck open (X), 1 stuck closed (square) and 2 broken nanowires). (b) Crossbar Structure Graph Representation. Vertices and edges have been removed based on the location and types of PLA crossbar faults.](image)

based on the defect pattern. Vertices in the graph correspond to nanowires in the crossbar, while edges between vertices correspond to crosspoints in the crossbar. For broken nanowires, the corresponding vertex must be removed (and thus all adjacent edges). For nanowire crosspoints that have a stuck-open defect, the corresponding edge must be removed from the graph. While there is no discussion of stuck-closed defects in [61], there is mention of these defects in [36] along with a scheme that can be used to modify the structure graph. All nanowires that are connected through a stuck-closed crosspoint are removed from the graph representation and are instead used to route a single signal. For the sake of the mapping, both vertices and all adjacent edges are removed from the structure graph. Fig. 5.2b shows the CSG for the PLA structure in Fig. 5.2a.
Once the graph for the defective nanowire crossbar structure has been generated, a mapping can be achieved by finding a monomorphism between the graphs \(^{61}\). We refer to this mapping method as the Crossbar Based PLA Mapping method (CBPM method). This method includes both the generation of the structure graph from the known defects and the use of monomorphism to find a matching. According to \(^{61}\), the authors used an implementation of monomorphism matching discussed in \(^{15}\).

5.2 Mapping to Defective QCA PLAs

In Sec. 5.1, we discussed a mapping procedure that converted the mapping problem into a graph matching problem. In this section, we present a new approach to mapping. The new approach exploits the unique features of the faults in QCA PLAs to construct new graph models for the mapping problem, and hence allows for a more versatile usage of defective crosspoints as compared to the graph matching approach proposed for nanowire crossbar PLAs \(^{61}\). Higher yields can be achieved by this new approach as will be shown in Sec. 5.3.

5.2.1 Yield-Increasing Mapping for QCA PLAs

Although the CBPM method discussed in Sec. 5.1 is easy to implement and quite efficient in terms of finding mappings (to be shown later), it can be rather pessimistic in terms of yield. This is due to the fact that QCA PLA cells are just as likely to become stuck in logic mode as in wire mode – but a much larger number of edges would be removed from the CSG for a stuck-in-logic fault than for a stuck-in-wire fault. Such removals could significantly hurt the yield compared to what could be achieved by a mapping that treats stuck-in-logic faults more intelligently.
In this section, we introduce a new method of dealing with stuck-in-logic faults, which leads to PLA mappings with higher yields.

Before introducing our new method, let us first review the implications of the edges in the LG and CSG. An edge in an LG represents the existence of a logic function (AND or OR). An edge in the CSG represents the capability of a programmable PLA cell being programmed into a logic gate. If this edge is matched to an edge in the LG, the respective PLA cell is programmed into a logic gate. Otherwise, the PLA cell is programmed into a simple wire. If a PLA cell is stuck at logic, it could still be used properly if this cell “happens” to be selected to function as a gate. Therefore, if we could distinguish the edges in a PLA structure graph between wire mode only, logic mode only, or both modes, it would be possible to match these edges in a more intelligent manner.

We now introduce our new graph model. We use a **QCA-based PLA Structure Graph (QSG)** to represent a QCA PLA. The vertices in a QSG are derived the same way as those in a CSG. That is, literal wires correspond to the first layer of vertices, implicant wires correspond to the second layer of vertices, and output wires correspond to the third layer of vertices. Between any two vertices in the adjacent layers, there may exist an edge of one of the following three types:

- **Type 1:** The PLA cell is fault free
- **Type 2:** The PLA cell is stuck in logic mode
- **Type 3:** The PLA cell is stuck in wire mode

Fig. 5.4 is the QSG for the PLA structure given in Fig. 5.3. Due to the two completely unusable PLA cells, vertices $H$ and $P$ and their adjacent edges are missing from the QSG. The two stuck at wire PLA cells give the two dotted edges
Figure 5.3. A defective and redundant QCA PLA structure. The AND plane is 5x5 cells, and the OR plane is 3x5 cells. There are 5 faulty cells.

while the one stuck at logic PLA cell results in the one dashed edge. Compared with the CSG in Fig. 5.2b, the QSG has more edges and vertices.

To leverage the graph monomorphism algorithm, we also modify the way that a Boolean logic function is modeled. Instead of modeling only the logic functions as edges in an LG, we model both the logic functions and the signal passages. Specifically, we introduce a new graph called the **Complete Logic Graph (CLG)**.

Similar to the LG, the CLG is a back-to-back bipartite graph where the first layer vertices represent the literals in the given function, the second layer vertices represent the implicants, and the third layer vertices represent the functions. Different from an LG, the CLG is a complete back-to-back bipartite graph. A subset of the edges correspond exactly to the edges in the LG and are called Type 2 edges, while the rest of the edges are called Type 3 edges. The CLG corresponding to the example LG in Fig. 5.1 is shown in Fig. 5.4b. Type 2 edges are shown as dashed edges while Type 3 edges are shown as dotted edges.

Mapping a given set of Boolean logic functions to a QCA PLA structure can
Figure 5.4. (a) The QSG corresponding to the QCA PLA structure in Fig. 5.3. The solid edges represent PLA cells that can be programmed into either logic or wire mode (Type 1), while the dashed edges represent the cells that are stuck in logic mode (Type 2), and dotted edges represent cells that are stuck in wire mode (Type 3). (b) The CLG corresponding to the LG in Fig. 5.1. The dashed edges show the logic to be mapped (Type 2), while the gray edges show which connections need to pass signals (Type 3).
now be solved by employing a monomorphism matching algorithm for attributed relational graphs (ARG). In particular, the edge types are treated as attributes such that Type 2 edges can be matched to either Type 1 or Type 2 edges while Type 3 edges can be matched to either Type 1 or Type 3 edges. By incorporating these attributes, we do not prematurely eliminate certain edges and hence increase the probability of finding a matching between the CLG and QSG. There exist both exact and approximate algorithms for ARG monomorphism matching (e.g. [15, 33]) which can be readily applied to solve our PLA mapping problem.

We would like to point out that our new QSG/CLG based PLA mapping technique (referred to as QCA-based PLA Mapping method (QCAPM method)) does lead to more complex graphs having to be matched. This could significantly increase the computation time for solving the ARG monomorphism problem for large problem instances. For such cases, approximate algorithms should be used instead of exact ones. We will use experimental data to illustrate the effectiveness and efficiency of our new QCAPM approach.

5.2.2 Special Case Graph Representation

As discussed in Sec. 4.1.5, there are different kinds of fully faulty PLA cell faults. All fully faulty PLA cells are unusable by themselves because they cannot be programmed into either logic or wire mode. These cells cannot contribute to the programming of a QCA PLA to generate a set of logical functions. In most cases, a fully faulty PLA cell will also sabotage the entire row or column it lies on, making the entire line unusable. This reduces the redundancy in the PLA, since more than just one PLA cell is lost. However, some fully faulty PLA cells are not as damaging to the row or column it lies on. These partial-line PLA cell
faults are themselves unusable, but PLA cells later in the array can be used to map logic. New rules for generating a structure graph are required.

In Sec. 5.2.1 the rules are given for generating a QSG. The rule for a fully faulty PLA cell requires the removal of a node and all adjacent edges. The consequences are severe for a fully faulty PLA cell, as compared to stuck-in-wire-mode and stuck-in-logic-mode faults. For a partial line fault, only the edges corresponding to the faulty cell and the cells that came before are removed. The node corresponding to that row or column remains, and the edges that correspond to later PLA cells are not removed. This helps maintain higher levels of redundancy in the QCA PLA as fewer resources are lost before the mapping process begins.

To illustrate the different rules for graph generation, the example from Fig. 5.3 is repeated in Fig. 5.5a. Two of the fully faulty PLA cells from the original example are replaced with partial line faults to show the difference. A new QSG is created with these new rules in mind (See Fig. 5.5b). The resulting graph has more nodes and edges thanks to the extra consideration given to PLA cells with partial line faults.

Looking at Fig. 5.5b, there are a few things that stand out when compared to Fig. 5.3b. While there are still three edges that have special attributes as a result of the stuck-in-logic-mode and stuck-in-wire-mode faults, one would quickly notice that in Fig. 5.5b all nodes in the complete bipartite graph are still present. As a result of the partial-line faults, a few edges are missing, but the nodes ‘H’ and ‘P’ remain useful. This is much better than losing all adjacent edges, thus leading to more possible mappings.
Figure 5.5. (a) A defective and redundant QCA PLA structure that includes cells that are faulty but only affect part of a row or column. (b) The corresponding QSG. Vertices and edges have been removed based on the location and types of PLA cell faults, including partial line faults.

5.3 Evaluation

In Sec. 4.1 we discussed the fault model for the QCA PLA, and in Sec. 5.2 we discussed the approach for mapping Boolean functions to a defective PLA structure. We now use a set of experiments to evaluate the different PLA mapping methods, i.e., the original CBPM method and the new QCAPM method. This will involve generating random defective PLA structures, converting those into graph representations, and then performing graph matchings between the PLA graphs and the graphs for various benchmarks. We have implemented both mapping methods in C++, along with a tool to generate PLAs with randomly distributed faults. The actual mapping was achieved by leveraging the VFLib Graph Matching Library [15]. The benchmarks that we consider range from small examples to larger functions from the Toronto 20 benchmark set [9].
The results presented in Sec. 5.3.1 focused on two important metrics in comparing the two mapping methods: the CBPM method and the QCAPM method. We examined the mapping success rate (yield) and the runtime efficiency for both methods. Two benchmarks were used to compare the two mapping methods. These benchmarks were mapped to many randomly generated defective PLA structures.

The experiments presented in the Sec. 5.3.2 focus on finding at what defect rates certain benchmarks can still be mapped to defective PLAs structures using the QCAPM Method. The defect rate that is used to generate the defective PLA structures is called the effective defect rate (EDR). The EDR is defined as the rate at which a QCA circuit – in this case the PLA cell – exhibits faulty behavior because of a defective device in that circuit. It corresponds to the PLA cell fault rate described in Sec. 4.1.2 which, if the device defect rate were known could be calculated using an equation similar to Eq. 4.1. Random PLA structures were generated so that various benchmarks can be mapped to them. Also, the randomly generated structures were created by using the fault model information from Sec. 4.1.5. This was done by assuming an even distribution of faulty PLA cells, but determining the type of faulty PLA cell by considering the PLA cell fault information from Table 4.4. The tests will be used to determine how much yield improvement can be achieved by including the partial line faults as part of the QCAPM method. Five different benchmarks of varying sizes were used in the mapping tests to show how yield is affected by benchmark size. Four of the benchmarks used come from the Toronto 20 benchmark set.
5.3.1 Yield and Runtime Comparisons

The first sets of experiments that we ran were aimed at comparing the yields possible for the two mapping methods. As a simple example, we first utilized the Boolean functions in Fig. 5.1. To implement these two Boolean functions, four literals and three implicants are required. Therefore, we decided to map the functions to a defective PLA with 8 inputs, 6 rows, and 4 outputs, which can be written as a PLA of size (8,6,4), and which has 48 crosspoint cells in the AND Plane and 24 crosspoints in the OR Plane. To perform these tests, we made a simple software tool that would generate many randomly defective PLA structures, and then converted those structures into graph representations to be used with the graph matching software. The procedure we used for generating the random structures was very simple. Given some EDR, each PLA cell had a certain chance of being assigned one of four possible conditions (either fully faulty, stuck-in-wire-mode, stuck-in-logic-mode, or not faulty). For these tests, each PLA cell fault type occurred at the same rate. We created 1,000 PLA structures with random faults, and attempted to map the functions to the test structures. PLA cell fault rates of 0%, 1%, 5%, and 10% were used during the generation of the defective PLAs. Mapping success rates are presented in Table 5.1.

The yield results were obtained using an exhaustive search of matchings between the representative structure and function graphs. Both the CBPM and the QCAPM methods found successful mappings for the entire set of 1,000 sample PLAs when the EDR was set at 0% and at 1%. Since there are only 2 Boolean functions, 3 implicants, and 4 literals to map onto the PLA structure, it is unlikely that having only 1% faulty cells out of 72 will lead to an unsuccessful matching.
At a 5% EDR, both methods exhibited at least a few cases out of 1,000 tests without a successful matching, and even more so at a 10% EDR.

The results show that the QCAPM method is more effective in providing higher yields than the CBPM method. Even at 10% EDR, the QCAPM method had a yield over 94% while the CBPM mapping method had a much lower yield (63%). We expected this, since our approach can utilize crosspoints that are stuck in wire and logic mode more effectively than the other approach. The CBPM method also saw a drop in yield more quickly as the EDR increased.

In general, real-world Boolean functions are much larger than the example considered above. Since the exact graph monomorphism algorithm used in our tool suite is exponential in time complexity [15], as the benchmark increases in size, the mapping process simply takes too much time. Therefore, we have implemented a runtime cutoff to stop searching for a successful matching after a certain period of time. This is a common step taken when trying to map onto reprogrammable logic structures. A runtime cutoff or an iteration cutoff has been used when mapping to other nanotechnologies [38, 58, 62]. Stopping the search early can
TABLE 5.2

PLA YIELD FOR MAPPING THE 3-BIT ADDER FUNCTIONS TO A (20,40,10) FAULTY PLA STRUCTURE USING A CUTOFF OF 5 SECONDS

<table>
<thead>
<tr>
<th>EDR</th>
<th>0%</th>
<th>1%</th>
<th>2%</th>
<th>3%</th>
<th>4%</th>
<th>5%</th>
<th>6%</th>
<th>7%</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBPM</td>
<td>100%</td>
<td>74.5%</td>
<td>22.0%</td>
<td>3.0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>QCAPM</td>
<td>100%</td>
<td>100%</td>
<td>94.0%</td>
<td>69.5%</td>
<td>37.5%</td>
<td>9.5%</td>
<td>3.5%</td>
<td>0%</td>
</tr>
</tbody>
</table>

reduce the number of successful matchings, however, only a small number of tests should incorrectly fail because the cutoff was too small.

To test these two methods on a larger benchmark, we used the Boolean functions for a 3-bit adder expressed in the sum-of-product form. There are 6 literals (12 if the inverted literals are included) and 31 implicant terms required to produce the 4 bits of output needed for the 3-bit adder. We attempted to map the 3-bit adder to 200 PLA structures of size (20,40,10) with random faulty PLA cells. We used a cutoff time of 5 seconds. The yield results are shown in Table 5.2. These results indicate that the QCAPM method is similarly effective in providing higher yields as in our test on a smaller benchmark.

The trends seen for the small example are still visible for this larger example up until the EDR reached 7%. At that point, all methods had a 0% yield. It is interesting to note that the largest yield reduction occurred at different places and at different rates for each method. The CBPM method dropped off most significantly between 1% and 3% EDR, while the QCAPM method dropped off.
most between 2% and 5% EDR. In addition, the yield for the QCAPM method did not decrease at the same rate as the CBPM method. It took an EDR change of 3% to reduce the QCAPM method from a yield of about 70% down to around 3%. On the other hand, the CBPM method went from a 75% yield to a 3% yield with only a 2% increase in EDR.

As compared to the results given in Table 5.1, the yield for the same EDR is lower for both mapping methods for the 3-bit adder. This is likely due to the PLA structure size. For the smaller example functions, the PLA size was twice that of the requirements for the functions. That is, there were only 4 literals, 3 implicants, and 2 functions trying to map onto a PLA with 8 inputs, 6 rows, and 4 outputs. In the case of the 3-bit adder, we tested a structure that was larger and had less redundancy. There were only 20 inputs for 12 literals, 40 rows for 31 implicants, and 10 outputs for 4 functions. By adding extra redundancy, we would see an increase in yield in all cases.

While the QCAPM method produces better yields with the same defective PLA structures, there is a cost for this improvement. When we ran the 3-bit adder tests used to generate Table 5.2, we also tracked the runtime of each mapping method. For tests that could not find a successful mapping, the run times were all the same, since the cutoff was the same for each. However, for tests that did find a successful mapping, there were more interesting runtime results.

The CBPM method had shorter run times than the QCAPM method on successful tests. This meant that if there was a relatively easy solution to the mapping problem, the CBPM method would find that solution faster. On the other hand, the QCAPM method would find a mapping more often. The runtime averages for successful tests are shown in Table 5.3. On average, the CBPM method seems
TABLE 5.3

RUNTIME COMPARISON OF THE TWO MAPPING METHODS
FOR SUCCESSFUL MATCHINGS OF THE 3-BIT ADDER

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>1%</th>
<th>2%</th>
<th>3%</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBPM</td>
<td>98 µsec</td>
<td>94 µsec</td>
<td>85 µsec</td>
<td>89 µsec</td>
</tr>
<tr>
<td>QCAPM</td>
<td>126 µsec</td>
<td>142 µsec</td>
<td>145 µsec</td>
<td>146 µsec</td>
</tr>
</tbody>
</table>

to be about 1.5 to 2 times times faster than the QCAPM in finding successful mappings.

5.3.2 Fault Distribution and Larger Benchmarks

In Sec. 5.3.1 we compared two different mapping methods by considering both the mapping success rate and mapping runtime over the course of many random tests. The even distribution of PLA fault types was sufficient for comparing different mapping methods, however it does not correspond very well to the expected frequency of occurrence for PLA cell faults. To improve our approach, we wanted to generate defective PLA structures using known correlations between defects and PLA cell faults so that our mapping tests would provide a better picture of what we might see in fabricated PLAs, especially in terms of the distribution of PLA cell fault types. The most complete approach would involve starting at device defects and working up to PLA cell faults. At this time, however, we do not have sufficient information about device fault rates for QCA circuits of any implementation. It is possible, however, to assign faults at the PLA cell level,
while improving the fault distribution in the randomly generated PLA structures. Specifically, we used the information in Table 4.4 to give each PLA cell one of five possible conditions (either fully faulty with full line faults, fully faulty with partial line faults, stuck-in-wire-mode, stuck-in-logic-mode, or not faulty). Assigning faults to the PLA cells using the information in Table 4.4 would lead to different rates for each PLA cell fault type according to the relationships between defects and cell faults.

In addition to providing a better PLA cell fault distribution, we also investigated how the partial line faults would affect the mapping results. We generated random PLA structures for a fixed size of 150 inputs, 4000 rows, and 100 outputs at various PLA cell fault rates (EDRs) using the data from Table 4.4. We then used our graph matching tool to map five different benchmarks of varying sizes (see Table 5.4) to these randomly generated structures. These tests were done once while not considering the partial line faults discussed in Sec. 4.1.5, and then again but this time considering the partial line faults. We expect that there should be at least as many successful mappings, if not more, when the partial line faults are considered while using the QCAPM method for similar tests. The results of these tests can be found in Table 5.5. The 3-bit adder benchmark was used again, along with the alu4, apex4, ex1010, and seq benchmarks from the Toronto 20 benchmark suite. The complexity of each benchmark is shown in Table 5.4 by indicating the number of literals, implicants, and functions.

In every case, either the two tests resulted in the same yield, or the tests that considered partial line faults (‘with’ columns) resulted in higher yields. In some instances the difference was significant. Good examples include the add3
TABLE 5.4

SIZES FOR SELECTED BENCHMARKS

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Literals</th>
<th>Implicants</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>add3</td>
<td>6</td>
<td>31</td>
<td>4</td>
</tr>
<tr>
<td>alu4</td>
<td>14</td>
<td>952</td>
<td>8</td>
</tr>
<tr>
<td>apex4</td>
<td>9</td>
<td>441</td>
<td>19</td>
</tr>
<tr>
<td>ex1010</td>
<td>10</td>
<td>1024</td>
<td>10</td>
</tr>
<tr>
<td>seq</td>
<td>41</td>
<td>1066</td>
<td>35</td>
</tr>
<tr>
<td>k2</td>
<td>45</td>
<td>212</td>
<td>45</td>
</tr>
</tbody>
</table>

benchmark at 0.002 fault rate and a few of the results for the apex4 benchmark. In most cases, however, the difference was small between the two cases. There is a critical range of EDRs for each benchmark in which the yield goes from 100% yield down to 0% yield with only a small increase in EDR. It seems that the partial line fault consideration is most beneficial in these regions. This could be useful in terms of real world circuits depending on many practical considerations associated with a specific implementation of QCA. Considering partial line faults is another way to attempt to squeeze more yield out of the QCA PLA.

It is useful to examine the trends in terms of the way in which the yield drops off for a given benchmark. Also, the size of the benchmark is important to consider. In Sec. 4.1.3 we gave a first order yield analysis, without actually conducting the mapping process for QCA-based PLAs. The calculations showed that there should be a relatively sharp ‘cutoff point’ where the yield of a PLA
**TABLE 5.5**

PLA YIELD FOR MAPPING 5 BENCHMARKS TO A (150,4000,100) PLA USING QCAPM METHOD WITHOUT AND WITH PARTIAL LINE FAULTS CONSIDERED

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>add3</th>
<th>alu4</th>
<th>ex1010</th>
<th>apex4</th>
<th>seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDR</td>
<td>w/o</td>
<td>with</td>
<td>w/o</td>
<td>with</td>
<td>w/o</td>
</tr>
<tr>
<td>0.005</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>0.003</td>
<td>0%</td>
<td>2%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>0.002</td>
<td>15%</td>
<td>52%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>0.001</td>
<td>99%</td>
<td>100%</td>
<td>88%</td>
<td>90%</td>
<td>65%</td>
</tr>
<tr>
<td>0.0009</td>
<td>100%</td>
<td>100%</td>
<td>98%</td>
<td>99%</td>
<td>93%</td>
</tr>
<tr>
<td>0.0008</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>99%</td>
</tr>
<tr>
<td>0.0005</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>0.0004</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>0.0003</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
would drop significantly with only a small increase in device defect rate and, by consequence, EDR. The calculations also showed that as the PLAs increased in size, the cutoff point would shift towards lower defect rates, indicating that small PLAs would be able to tolerate higher defect rates while large PLAs would only tolerate lower defect rates. In Fig. 5.6, we plotted the results from Table 5.5 (with partial line faults considered) expecting to see similar trends. The plot uses a logarithmic scale for the EDR on x-axis, and a linear scale for the yield on the y-axis.

From Fig. 5.6, the first clear trend that can be seen is the steep slope of each curve. This corresponds very well to the cutoff point discussed in Sec. 4.1.3. For each benchmark, there is a small range of EDRs that correspond to a quick
decrease in yield. The cutoff point is different for each benchmark, but they all exhibit the same behavior.

In addition, the benchmark curves make for a nice comparison. In order to compare the benchmarks, we generated defective PLAs with a consistent size of 150 inputs, 4000 rows, and 100 outputs. A PLA of size (150,4000,100) is large enough to accommodate many of the benchmarks we expected to use from the Toronto 20 benchmark set, while still providing significant levels of redundancy. Even though all five benchmarks were mapped to a defective PLA structure with the same size, each benchmark is different in size, leading to curves that are shifted from each other. The add3 benchmark has 12 literals (regular and inverted), 31 implicants, and 4 functions, making it the smallest (12,31,4). This benchmark is most easily mapped to the defective PLA structure because of its small size. The alu4 benchmark is of size (28,952,8). The ex1010 benchmark is (20,1024,10). The apex4 benchmark is (18,441,19). The seq benchmark is (82,1066,35). These five benchmarks vary in size enough to provide the nicely spaced curves seen in Fig. 5.6, with the smaller ones towards the right (able to tolerate higher EDRs) and the larger ones towards the left (only able to tolerate lower EDRs). While there is some variation here in terms of size and cutoff point (the apex4 benchmark seems to be out of order in terms of benchmark size), the benchmarks seem to follow the trend in general, and the variation may be attributed to the detailed compositions of the implicant and output terms.

There are many factors that could affect the mapping success rate for various benchmarks mapped onto defective PLA structures. The first factor is size (see Table 5.4 for the size of all benchmarks mentioned here). Overall, it seems that this is the most important factor. However, a benchmark is composed of a cer-
tain number of literals, implicants, and functions, and each affects the mapping in different ways. For example, the apex4 benchmark has fewer literals and implicants than the alu4 benchmark, but more functions. Results show that the alu4 benchmark can be mapped successfully more often than the apex4 benchmark. In looking at the necessary logical connections required for both benchmarks, apex4 is more dense, meaning that more of the PLA cells need to be programmed to logic mode to accommodate the logic functions of the benchmark. The alu4 benchmark has a logic density of 24%, while the apex4 benchmark has a density of 33%. On the other hand, the ex1010 has a logic density of 61%, but still shows more successful mappings than the apex4 benchmark. We have summarized the characteristics of each benchmark in Table 5.6. Also included in this table is an additional benchmark (k2), which has a high number of inputs and outputs, but a low number of implicants. Results show that the k2 benchmark has a lower cutoff point than any of the other benchmarks that we tested, and thus the lowest rate for successful mappings, despite having only 212 implicants. All of this seems to indicate that the number of inputs, implicants, outputs, and the logic density are all parameters that affect mapping quality for the different benchmarks. A systematic study would be needed in order to quantify the importance of each parameter, which we would like to study in future work.

5.4 PLA Systems

In order to find ways to increase yield for QCA PLAs, we began looking at ways to create systems of PLAs. While studying PLA yield, our calculations suggested that smaller PLA structures would be able to tolerate higher defect rates than larger PLA structures could tolerate. After performing many mapping

106
TABLE 5.6

FAULT RATE CUTOFF POINT AND BENCHMARK CHARACTERISTICS FOR ALL TESTED BENCHMARKS

<table>
<thead>
<tr>
<th>Name</th>
<th>add3</th>
<th>alu4</th>
<th>ex1010</th>
<th>apex4</th>
<th>seq</th>
<th>k2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>(12,31,4)</td>
<td>(28,952,8)</td>
<td>(20,1024,10)</td>
<td>(18,441,19)</td>
<td>(82,1066,35)</td>
<td>(90,212,45)</td>
</tr>
<tr>
<td>Density</td>
<td>29%</td>
<td>24%</td>
<td>61%</td>
<td>33%</td>
<td>11%</td>
<td>10%</td>
</tr>
<tr>
<td>Cutoff</td>
<td>0.0012</td>
<td>0.001</td>
<td>0.00091</td>
<td>0.00058</td>
<td>0.00041</td>
<td>0.00032</td>
</tr>
</tbody>
</table>

tests, we found that smaller benchmarks successfully mapped to PLAs at higher defect rates, while larger benchmarks did not, even with the same proportion of redundant row and columns as compared to the benchmark size. While not conclusive, this certainly suggested that smaller is better, and that breaking up a logical benchmark in smaller parts and mapping those parts onto many smaller PLAs should provide an increase in yield. Therefore, we set out to design systems of QCA PLAs, and performed mapping tests that would demonstrate that smaller is better. To break the benchmark logic into smaller parts, we utilized a tool call PLAmap, which is part of the RASP package and SIS toolset [14, 59]. This tool has been used to map benchmarks onto systems of nanowire crossbar PLAs [22].

5.4.1 PLA Size Study

In order to more efficiently use QCA PLAs, we wanted to determine if there was an optimal PLA size. Trying to find this optimal PLA size closely corresponds to finding an optimal Boolean function size, since logic conversion tools would be
used to break benchmark logic down into a certain size to fit onto a redundant PLA structure. There is a tradeoff going on between the size of the redundant PLA and the ability to logically break a benchmark into smaller parts. As Boolean functions and PLAs get bigger, the chance of finding a successful mapping of that logic onto the PLA decreases for the same defect rate. Therefore, it is better to break the logic down into smaller Boolean functions so that they fit onto smaller defective but redundant PLAs. On the other hand, Boolean functions and PLAs that are very small start to become inefficient because the logic cannot easily be broken down into implicants that only have a few literals and functions that only have a few implicants. To take this to an extreme, a 1x1 PLA is too small to be useful for mapping any Boolean function. There is another factor that can push the optimal PLA size larger. In order for a system of PLAs to work together, they need to be connected. As PLAs get smaller, there need to be more of them to map the same logic. At some point the overhead for interconnect will exceed the efficiency gained by using smaller PLAs.

We performed some basic tests to confirm the idea that smaller is better. We used the SIS tool PLAmap to break up the alu4 benchmark into 7 and 14 PLAs. Then we mapped the alu4 benchmark logic onto systems of 1, 7, and 14 PLAs using different levels of redundancy. The mapping results, which did not consider interconnect, are plotted in Fig. 5.7. They show how mapping the three different alu4 logical breakdowns can tolerate different EDRs as a function of total PLA size over all PLAs in the system. In this case, PLA size is simply the number of PLA cells needed in all PLAs structures. We found that for the same number of PLA cells, using more PLAs to map the benchmark corresponded to a higher tolerated EDR. This can be seen by drawing a vertical line that touches all three
curves. The curve for the system of 14 PLAs will Intersect the vertical line higher on the graph for the same number of PLA cells used, indicating that it can map all logic successfully at higher EDRs.

5.5 Summary

In this chapter, we have taken a PLA mapping approach which was initially proposed for nanowire crossbar PLAs and improved it for QCA PLAs so that it can find successful mappings in more cases. This improvement has come as a result of trying to find ways to use faulty QCA PLA cells as part of the mapping procedure. In the past, many faulty PLA resources were lost due to limitations in the mapping approach. We have reduced this limitation by using edge attributes in the graph representations of the Boolean logic and the defective PLA structures. This improved mapping method allowed us to show increased rates of successful mappings to defective PLAs for many different benchmarks.
Figure 5.7. Curves obtained from mapping the alu4 benchmark onto different systems of PLAs. As the alu4 logic is broken up into smaller parts and mapped onto smaller PLAs, the system can be mapped even with higher EDRs.
CHAPTER 6:

PLA PERFORMANCE

Performance data are important to any emerging technology so that it can be compared to other technologies. In many cases, one nanotechnology will be quite different than another nanotechnology, and there needs to be some common points for comparison, since the devices and circuits will not be directly comparable. Commonly used metrics include area, energy, delay, and many combinations of those three including area-delay product, energy-delay product, or even area-energy-delay product. For computer architectures, these metrics must often be compared at the system or application level, again because the technologies are often very different and can really only be compared at the highest levels. First we made estimates about future NML circuits and systems and determined the likely area, energy, and delay for the NML PLA cells. In order to compare NML to other reprogrammable technologies that do not have PLA “cells,” we extrapolated our results to many common benchmarks. We choose to use the Toronto 20 [9] suite of benchmarks because they have been used to evaluate performance for other nanotechnologies such as CMOS FPGAs, nanowire PLAs, and nanowire FPGAs [22, 62, 64].

In Sec. 6.2 we provide area, energy, and delay performance estimates for NML PLAs, and compare the area of an NML PLA to other reprogrammable nanotech-
nologies\(^5\). To calculate the aforementioned performance metrics, we need to determine the number of PLA cells required to implement a given logic function or benchmark. This can be obtained from a successful mapping of each benchmark. To perform these mappings, we follow the approach described in Chapter 5.

6.1 Parameters That Influence Performance and Assumptions

The design parameters that impact each metric are summarized in Table 6.1.

Area calculations for the NML PLA make assumptions about nanomagnet footprint size and PLA cell size. For the performance metrics calculated in this chapter, we assume a fabrication process that can define nanomagnets 40 nm wide by 60 nm high with 10 nm spacings horizontally and 15 nm spacings vertically. This gives a magnet footprint of 50 nm by 75 nm or 3,750 nm\(^2\). In our lab, this is achieved with e-beam lithography. For larger production, it is unknown at this time how such devices would be produced. Because of the simple single-layer requirement for the NML devices, it is possible that photolithography could be used for the NML clock wires\(^6\) and something like imprint lithography could be used for the NML devices (much like hybrid CMOS-nanowire systems\(^7\)). Another reason for choosing this magnet size is that it provides an initial comparison point for other nanotechnologies, specifically 45 nm CMOS FPGAs and 30 nm nanowire/45 nm CMOS hybrid FPGAs. While it might not be a completely fair comparison among the three technologies, it provides a starting point that can be improved on in the future, and does allow for a comparison

\(^5\)Comparing area makes the most sense given the available data in the literature for reprogrammable designs based on emerging technologies. Other comparisons would be incomplete or misleading. Energy estimates are difficult to compare consistently with other charge-based technologies, and NML circuits will have larger delays.
### TABLE 6.1

**PERFORMANCE PARAMETERS FOR NML PLAS**

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Area</strong></td>
<td>Magnet Footprint</td>
<td>Area of magnet plus spacings in $nm^2$</td>
</tr>
<tr>
<td></td>
<td>PLA Cell Size</td>
<td>Size of PLA cell in $nm^2$</td>
</tr>
<tr>
<td></td>
<td>PLA Size</td>
<td># of PLA cells from mapping solution</td>
</tr>
<tr>
<td><strong>Delay</strong></td>
<td>Switching Time</td>
<td>Time to switch magnet in $ps$</td>
</tr>
<tr>
<td></td>
<td>Critical Path</td>
<td># of magnets on critical path</td>
</tr>
<tr>
<td></td>
<td>Clock Period</td>
<td>Time between clock activations in $ns$</td>
</tr>
<tr>
<td></td>
<td>Zones per Cell</td>
<td># of clock zones across PLA cell</td>
</tr>
<tr>
<td></td>
<td>PLA Width</td>
<td># of clock zones across entire PLA</td>
</tr>
<tr>
<td><strong>Magnet Energy</strong></td>
<td>Switching Energy</td>
<td>Energy dissipated by switching a magnet</td>
</tr>
<tr>
<td></td>
<td>Magnets per Cell</td>
<td># of magnets in each PLA cell</td>
</tr>
<tr>
<td></td>
<td>PLA Size</td>
<td># of PLA cells from mapping solution</td>
</tr>
<tr>
<td><strong>Clock Energy</strong></td>
<td>Power per Wire</td>
<td>Power consumed by clock wire in $nW$</td>
</tr>
<tr>
<td></td>
<td>Clock Period</td>
<td>Time between clock activations in $ns$</td>
</tr>
<tr>
<td></td>
<td>Wires per Cell</td>
<td># of clock wires per PLA cell</td>
</tr>
<tr>
<td></td>
<td>PLA Size</td>
<td># of PLA cells from mapping solution</td>
</tr>
</tbody>
</table>
of the "high-end" for each technology. We calculate the size of the PLA cell by looking at the layout of both types of PLA cells given in Fig. 3.1d and Fig. 3.5b. The AND plane cell is 18 nanomagnets wide and 7 nanomagnets tall, while the OR plane cell is 18 nanomagnets wide and 8 nanomagnets tall. Using the nanomagnet footprint dimensions, we calculated that the AND plane cell would be 900 nm by 525 nm or 472,500 nm\(^2\) and the OR plane cell would be 900 nm by 600 nm or 540,000 nm\(^2\).

Clocking wires to produce the necessary nulling magnetic fields can be placed under the nanomagnets that make up each NML PLA cell. These wires take the form of long metal rectangles running from top to bottom in the design. To calculate the delay for a PLA, we made assumptions about clocking zones inside the PLA cells and about the critical path in these zones. Considering the PLA cell designs in Fig. 3.1d and Fig. 3.5b, we decided to divide both cells vertically down the middle, producing 2 clock zones horizontally per cell. The longest critical path in a clock zone is 16 nanomagnets. Assuming a 200ps switching time per magnet \[^{34}\], the critical path delay would be 3.2 ns, leading to a clock period of 6.4 ns.

To estimate magnet switching energy, we first need to determine the number of magnets in each PLA cell. The layouts in Fig. 3.1d and Fig. 3.5b suggest that about 40 magnets are required for the AND plane cell, and 44 magnets are required for the OR plane cell. We assume 100\(kT\) of energy dissipation per magnet switching event (with T=300 K). Also, we assume that every nanomagnet switches every clock cycle. This is probably a pessimistic assumption, but provides an upper bound on the power dissipation for the nanomagnets\(^1\).

\(^1\)For example, if the input to an antiferromagnetically ordered line does not change, then the magnetic state of the devices in the line will not change either. However, the same line will still be subjected to a clocking field and there will be some state change in the devices as the field is applied and removed.
To determine clocking energy, we define a clock wire geometry and specify the current that it will be subjected to in order to provide a given clocking field strength. Here we assume that the clock wire will be 450 nm wide – or half as wide as a PLA cell, as there are two horizontal zones per cell. Also, we assume that the clock wire will be 600 nm thick. We assume a 4-phase clock, and assume that the clocking wires need to produce a 5mT field [25]. Using Maxwell simulations, we determined that to produce a 5mT field with a 450 nm wide clock wire (400 nm of copper and 25 nm of ferromagnetic yoke cladding plus oxide on each side) requires a current of 2.2mA. With these assumptions, we can calculate the resistance and power consumption for the clock wire using the approach from [52]. Using the clock wire cross-sectional area, we can calculate the resistance of the copper clock wire per unit length. To simplify the calculation, we amortized the long clock wires by dividing them into “segments” that are each 2.1 microns long. Each 2.1-micron-long wire segment has a resistance of 0.1575 Ohms, leading to a power of 508.2 nW per segment. The total power for a given PLA is the power per segment multiplied by the number of segments that would make up the long wires that run across the entire PLA structure. The total energy for the evaluation of an entire PLA structure is the power for all wire “segments” and all magnets switching multiplied by the NML clock period.

6.2 Technology Comparison

6.2.1 Single NML PLA

For our first set of NML PLA performance estimates, we mapped the Boolean functions for each benchmark to a single NML PLA structure. This was done for 9 combinational benchmarks from the Toronto 20 suite (see Table 6.2 for the size
of each benchmark in terms of the number of literals, implicants, and functions). Using the mapping results to get a PLA size requirement for each benchmark, and using the assumptions stated in Sec. 6.1, we calculated the area, energy, and delay for each. These results are shown in Table 6.3. The PLA sizes are given in the form (A,B,C), where A is the number of input rows, B is the number of columns, and C is the number of output rows. We also compared the area of NML PLAs to CMOS FPGAs [62] and FPNI hybrid nanowire FPGAs [62], as shown in Table 6.4. All results consider no defects.

The results indicated that benchmarks implemented on NML PLAs have long
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>PLA Size</th>
<th>Area ($\mu m^2$)</th>
<th>Energy ($pJ$)</th>
<th>Delay ($ns$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>(28,952,8)</td>
<td>16,708</td>
<td>65.3</td>
<td>3,102</td>
</tr>
<tr>
<td>apex2</td>
<td>(78,1075,3)</td>
<td>41,361</td>
<td>161.6</td>
<td>3,568</td>
</tr>
<tr>
<td>apex4</td>
<td>(18,441,19)</td>
<td>8,275</td>
<td>32.3</td>
<td>1,469</td>
</tr>
<tr>
<td>ex1010</td>
<td>(20,1024,10)</td>
<td>15,206</td>
<td>59.4</td>
<td>3,323</td>
</tr>
<tr>
<td>ex5p</td>
<td>(16,256,63)</td>
<td>10,644</td>
<td>41.6</td>
<td>944</td>
</tr>
<tr>
<td>misex3</td>
<td>(28,1287,14)</td>
<td>26,757</td>
<td>104.5</td>
<td>4,184</td>
</tr>
<tr>
<td>pdc</td>
<td>(32,2192,40)</td>
<td>80,490</td>
<td>314.4</td>
<td>7,128</td>
</tr>
<tr>
<td>seq</td>
<td>(82,1066,35)</td>
<td>61,450</td>
<td>240.1</td>
<td>3,597</td>
</tr>
<tr>
<td>spla</td>
<td>(32,2172,46)</td>
<td>86,793</td>
<td>339.0</td>
<td>7,074</td>
</tr>
</tbody>
</table>
# TABLE 6.4

## AREA COMPARISON AMONG NANOTECHNOLOGIES FOR SELECTED TORONTO 20 BENCHMARKS IN UNITS OF $\mu m^2$

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CMOS FPGAs 45nm</th>
<th>NML PLAs 40\times60nm</th>
<th>FPNI FPGAs 30nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>137,700</td>
<td>16,708</td>
<td>17,513</td>
</tr>
<tr>
<td>apex2</td>
<td>166,050</td>
<td>41,361</td>
<td>18,983</td>
</tr>
<tr>
<td>apex4</td>
<td>414,619</td>
<td>8,275</td>
<td>13,457</td>
</tr>
<tr>
<td>ex1010</td>
<td>391,331</td>
<td>15,206</td>
<td>41,252</td>
</tr>
<tr>
<td>ex5p</td>
<td>100,238</td>
<td>10,644</td>
<td>11,050</td>
</tr>
<tr>
<td>misex3</td>
<td>124,538</td>
<td>26,757</td>
<td>14,750</td>
</tr>
<tr>
<td>pdc</td>
<td>369,056</td>
<td>80,490</td>
<td>48,153</td>
</tr>
<tr>
<td>seq</td>
<td>151,369</td>
<td>61,450</td>
<td>17,513</td>
</tr>
<tr>
<td>spla</td>
<td>326,025</td>
<td>86,793</td>
<td>43,493</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>2,180,926</strong></td>
<td><strong>347,684</strong></td>
<td><strong>226,164</strong></td>
</tr>
</tbody>
</table>
delays, on the order of a few microseconds. These same benchmarks have delays in the range of 5-20 nanoseconds when implemented on the other reprogrammable nanotechnologies we considered [62]. Energy for these benchmarks implemented on NML PLAs is on the order of a few picojoules per operation. In terms of area, NML PLAs are competitive with other reprogrammable nanotechnologies [62]. Across all the benchmarks, NML PLAs require six times less than the total area required by CMOS FPGAs, and the NML PLAs require an area that is only 54% larger than FPNI FPGAs. The delays for the NML PLAs are much larger than the delays for CMOS and FPNI FPGAs (an average of 7.87 ns for FPNI, 6.28 ns for CMOS, and 3,821 ns for NML), but the results shown in Table 6.3 assume that the benchmark is evaluated in the PLA only once, leading to a delay for the entire width of the PLA structure. If logic evaluations can be pipelined through the PLA structure, then a new result after the first can be obtained after each clock period of 6.4ns.

6.2.2 Systems of PLAs

In order to find ways to increase performance (and possibly defect tolerance) for NML PLAs, we began looking at ways to create systems of PLAs. To break the benchmark logic into smaller parts, we utilized the PLAmap tool, which is part of the RASP package and SIS toolset [14, 59]. This tool has been used to map benchmarks onto systems of nanowire crossbar PLAs [21].

We started by doing a very basic test to look at the potential that systems of PLAs have as compared to a single, large PLA. We used PLAmap to break the benchmarks into smaller functional units. In some cases, PLAmap was able to find a mapping that allowed some functions to be implemented independently
on many smaller PLAs. This breakdown has only one level of logic, and avoids the need for interconnect between PLAs. In other cases, PLAmap optimized the logic by breaking the benchmark into multiple dependent levels of logic, which would require interconnect between different levels. We took the results from PLAmap for each benchmark, and determined the optimal arrangement of the logical benchmark to be mapped onto a system of PLAs.

Considering just the size of the PLA logic (we ignore interconnect overhead for now), we calculated the performance for those benchmarks implemented on systems of many PLAs. A total of 6 out of the 9 combinational benchmarks could be implemented on systems of PLAs without interconnect. However, for 3 of the benchmarks (alu4, apex2, and seq), PLAmap could not find a mapping using only one level of logic in the parameter space we tested. In the cases of two or more levels of logic, an interconnect layer, like that in Fig. 3.6 would be needed but was not included here. These performance estimates are shown in Table 6.5.

Since, PLAmap did not take into consideration the interconnect performance overhead for alu4, apex2, and seq, to investigate the interconnect overhead, we determined how much interconnect was needed to connect the PLAs in the three multi-level logic cases. Besides the interconnect overhead, the calculations also assumed that the AND plane cell is as tall as the OR plane cell, to allow for correct alignment of the PLAs with the interconnect (as discussed in Sec. 3.4). Performance results for alu4, apex2, and seq with interconnect overhead are summarized in Table 6.6. The results indicate that adding NML-based interconnect results in a large overhead, and scales poorly. In all three cases, the interconnect area and energy overhead is larger than the area and energy required to implement the logic. The interconnect also adds a delay overhead.
TABLE 6.5

PERFORMANCE NUMBERS FOR COMBINATIONAL TORONTO
20 BENCHMARKS MAPPED ONTO SYSTEMS OF MANY SMALL
NML PLAS (INTERCONNECT OVERHEAD NOT CONSIDERED)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>PLA Size (A,B,C)</th>
<th># of PLAs</th>
<th>Depth</th>
<th>Area ($\mu m^2$)</th>
<th>Energy (pJ)</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>(16,32,8)</td>
<td>36</td>
<td>2</td>
<td>13,686</td>
<td>53.5</td>
<td>278</td>
</tr>
<tr>
<td>apex2</td>
<td>(24,16,8)</td>
<td>124</td>
<td>3</td>
<td>31,069</td>
<td>121.4</td>
<td>302</td>
</tr>
<tr>
<td>apex4</td>
<td>(16,192,4)</td>
<td>5</td>
<td>1</td>
<td>9,331</td>
<td>36.5</td>
<td>645</td>
</tr>
<tr>
<td>ex1010</td>
<td>(16,128,12)</td>
<td>10</td>
<td>1</td>
<td>17,971</td>
<td>70.2</td>
<td>453</td>
</tr>
<tr>
<td>ex5p</td>
<td>(8,64,16)</td>
<td>4</td>
<td>1</td>
<td>3,180</td>
<td>12.4</td>
<td>242</td>
</tr>
<tr>
<td>misex3</td>
<td>(16,256,4)</td>
<td>4</td>
<td>1</td>
<td>9,953</td>
<td>38.9</td>
<td>850</td>
</tr>
<tr>
<td>pdc</td>
<td>(16,128,12)</td>
<td>4</td>
<td>1</td>
<td>7,188</td>
<td>28.1</td>
<td>453</td>
</tr>
<tr>
<td>seq</td>
<td>(24,64,4)</td>
<td>49</td>
<td>2</td>
<td>42,336</td>
<td>165.4</td>
<td>496</td>
</tr>
<tr>
<td>spla</td>
<td>(16,128,16)</td>
<td>4</td>
<td>1</td>
<td>8,294</td>
<td>32.4</td>
<td>459</td>
</tr>
</tbody>
</table>
**TABLE 6.6**

PERFORMANCE NUMBERS FOR THE THREE COMBINATIONAL TORONTO 20 BENCHMARKS THAT REQUIRE INTERCONNECT BETWEEN MULTIPLE LEVELS OF LOGIC WHEN MAPPED TO MANY PLAS

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of Signals</th>
<th># of PLAs</th>
<th>Area $(\mu m^2)$</th>
<th>Energy $(pJ)$</th>
<th>Delay $(ns)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Level 1-2</td>
<td>Level 2-3</td>
<td>L1 L2 L3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>alu4</td>
<td>70</td>
<td>0</td>
<td>29 7 0</td>
<td>41,239</td>
<td>161.0</td>
</tr>
<tr>
<td>apex2</td>
<td>292</td>
<td>118</td>
<td>105 16 3</td>
<td>572,244</td>
<td>2,235.0</td>
</tr>
<tr>
<td>seq</td>
<td>175</td>
<td>0</td>
<td>41 8 0</td>
<td>155,902</td>
<td>608.8</td>
</tr>
</tbody>
</table>

We took the data for NML PLAs and systems of NML PLAs summarized in Tables 6.3 - 6.6 and put all results, side by side, in Table 6.7. Performance overheads due to interconnect are included for the alu4, apex2, and seq benchmarks.

We found that by decomposing benchmark logic into smaller parts with PLAmap, there is often a reduction in the area requirements to implement the logic. However, if the breakdown requires NML interconnect, these improvements are lost to the interconnect overhead. In 4 cases (ex5p, misex3, pdc, and spla), less area was required to implement the logic as systems of NML PLAs, while the other 5 required less area when implemented as a single NML PLA. Across all benchmarks, systems of PLAs usually had smaller delays (apex2 being the only exception). In some cases, the delay is more than ten times smaller. While further investigations might find ways to reduce the interconnect overhead, it seems that NML-based in-
TABLE 6.7

PERFORMANCE COMPARISON BETWEEN SINGLE PLA IMPLEMENTATION AND SYSTEMS OF PLAS FOR COMBINATIONAL TORONTO 20 BENCHMARKS (INTERCONNECT PERFORMANCE OVERHEADS INCLUDED)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Single PLA</th>
<th></th>
<th></th>
<th>Many PLAs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area ($\mu m^2$)</td>
<td>Energy (pJ)</td>
<td>Delay (ns)</td>
<td>Area ($\mu m^2$)</td>
<td>Energy (pJ)</td>
<td>Delay (ns)</td>
</tr>
<tr>
<td>alu4</td>
<td>16,708</td>
<td>65.3</td>
<td>3,102</td>
<td>41,239</td>
<td>161.0</td>
<td>1,579</td>
</tr>
<tr>
<td>apex2</td>
<td>41,361</td>
<td>161.6</td>
<td>3,568</td>
<td>572,244</td>
<td>2,235.0</td>
<td>6,942</td>
</tr>
<tr>
<td>apex4</td>
<td>8,275</td>
<td>32.3</td>
<td>1,469</td>
<td>9,331</td>
<td>36.5</td>
<td>645</td>
</tr>
<tr>
<td>ex1010</td>
<td>15,206</td>
<td>59.4</td>
<td>3,323</td>
<td>17,971</td>
<td>70.2</td>
<td>453</td>
</tr>
<tr>
<td>ex5p</td>
<td>10,644</td>
<td>41.6</td>
<td>944</td>
<td>3,180</td>
<td>12.4</td>
<td>242</td>
</tr>
<tr>
<td>misex3</td>
<td>26,757</td>
<td>104.5</td>
<td>4,184</td>
<td>9,953</td>
<td>38.9</td>
<td>850</td>
</tr>
<tr>
<td>pdc</td>
<td>80,490</td>
<td>314.4</td>
<td>7,128</td>
<td>7,188</td>
<td>28.1</td>
<td>453</td>
</tr>
<tr>
<td>seq</td>
<td>61,450</td>
<td>240.1</td>
<td>3,597</td>
<td>155,902</td>
<td>608.8</td>
<td>2,850</td>
</tr>
<tr>
<td>spla</td>
<td>86,793</td>
<td>339.0</td>
<td>7,074</td>
<td>8,294</td>
<td>32.4</td>
<td>459</td>
</tr>
</tbody>
</table>
terconnect with systems of PLAs is, in general, too costly in terms of performance to be a viable option.

We took the best area result from Table 6.7 for each benchmark, and compared them to CMOS FPGAs and FPNI hybrid nanowire FPGAs, as shown in Table 6.8. The results indicate that NML PLAs can implement 7 out of 9 of the benchmarks in a smaller area than FPNI FPGAs, and NML PLAs require less area over all benchmarks. The delays for NML PLAs are much reduced if the benchmarks are decomposed, but they are still larger than for CMOS and FPNI FPGAs (an average of 7.87 ns for FPNI, 6.28 ns for CMOS, and 1,608 ns for NML).

6.3 PLAs With Redundancy

Our previous performance evaluation of NML PLAs assumed the structures would be defect free. This was done to compare fairly with other technologies that provided performance figures for defect free structures in addition to other metrics. However, an important reason for investigating reprogrammable architectures like NML PLAs was to allow for working logic even in the presence of defects by using architectural redundancy and defect avoidance. Therefore, a performance analysis would not be complete without looking at how redundancy would affect the area, energy, and delay of NML PLAs, and also determining what defect rates the PLA can operate under.

To allow for consistent comparisons, we mapped the combinational benchmarks from Sec. 6.2.1 to the same NML PLA with 150 inputs columns, 4000 term rows, and 100 output columns (150,4000,100). A PLA of this size was chosen as it provides enough rows and columns to accommodate all the benchmarks we considered plus extra rows and columns to provide redundancy in the case of defects.
### TABLE 6.8

**AREA COMPARISON AMONG NANOTECHNOLOGIES FOR SELECTED COMBINATIONAL TORONTO 20 BENCHMARKS IN UNITS OF $\mu m^2$**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CMOS FPGAs 45nm</th>
<th>NML PLAs 40x60nm</th>
<th>FPNI FPGAs 30nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>137,700</td>
<td>16,708</td>
<td>17,513</td>
</tr>
<tr>
<td>apex2</td>
<td>166,050</td>
<td>41,361</td>
<td>18,983</td>
</tr>
<tr>
<td>apex4</td>
<td>414,619</td>
<td>8,275</td>
<td>13,457</td>
</tr>
<tr>
<td>ex1010</td>
<td>391,331</td>
<td>15,206</td>
<td>41,252</td>
</tr>
<tr>
<td>ex5p</td>
<td>100,238</td>
<td>3,180</td>
<td>11,050</td>
</tr>
<tr>
<td>misex3</td>
<td>124,538</td>
<td>9,953</td>
<td>14,750</td>
</tr>
<tr>
<td>pdc</td>
<td>369,056</td>
<td>7,188</td>
<td>48,153</td>
</tr>
<tr>
<td>seq</td>
<td>151,369</td>
<td>61,450</td>
<td>17,513</td>
</tr>
<tr>
<td>spla</td>
<td>326,025</td>
<td>8,294</td>
<td>43,493</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>2,180,926</strong></td>
<td><strong>171,615</strong></td>
<td><strong>226,164</strong></td>
</tr>
</tbody>
</table>
TABLE 6.9

TOLERATED DEFECT RATE FOR COMBINATIONAL TORONTO
20 BENCHMARKS MAPPED ONTO DEFECTIVE BUT
REDUNDANT NML PLAS OF SIZE (150,4000,100)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>“Tolerated” EDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>$1.0 \times 10^{-3}$</td>
</tr>
<tr>
<td>apex2</td>
<td>$1.1 \times 10^{-3}$</td>
</tr>
<tr>
<td>apex4</td>
<td>$5.8 \times 10^{-4}$</td>
</tr>
<tr>
<td>ex1010</td>
<td>$9.1 \times 10^{-4}$</td>
</tr>
<tr>
<td>ex5p</td>
<td>$2.7 \times 10^{-4}$</td>
</tr>
<tr>
<td>misex3</td>
<td>$6.9 \times 10^{-4}$</td>
</tr>
<tr>
<td>pdc</td>
<td>$3.2 \times 10^{-4}$</td>
</tr>
<tr>
<td>seq</td>
<td>$4.1 \times 10^{-4}$</td>
</tr>
<tr>
<td>spla</td>
<td>$3.1 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

(see Table 6.2 to see the sizes of the benchmarks). We performed 1,000 mapping tests for each benchmark at many different EDRs (see Sec. 5.3 for a discussion on EDR) to find the maximum rate that allows 90% mapping success. The results are summarized in Table 6.9. Since all benchmarks are mapped to the same sized NML PLA structure, the performance is the same for all. The (150,4000,100) PLA would take up an area of 499,500 $\mu m^2$, consume 1,951.3 $pJ$ of energy per evaluation, and would produce a result every 13,198 $ns$ without pipelining.

The mapping results show that each benchmark mapped to the same sized
PLA has a different maximum “tolerated” EDR. These range from $2.7 \times 10^{-4}$ (lowest rate tolerated) to $1.1 \times 10^{-3}$ (highest rate tolerated). Also, the large, redundant PLA structure results in areas and energies that are approximately 6 to 60 times larger than for the defect free PLA mappings (6.3), and delays that are approximately 2 to 15 times larger.

The maximum EDR corresponds to a “cutoff point” as described in Sec. 4.1.3. The cutoff point, practically speaking, means that for small increases in the EDR (relative to the cutoff rate), the yield would quickly drop to 0%, and for small decreases in the EDR, the yield would quickly rise to 100%. This means that each benchmarks should have high yields at any EDR smaller than the cutoff EDR given in the corresponding row of Table 6.9. To illustrate this, we mapped all the benchmarks to PLAs which had only a few rows and columns of redundancy and which had an EDR of $10^{-4}$. The result is that all benchmarks had 90-100% yield from the mapping tests at that rate. Said another way, if the EDR was $10^{-4}$, all benchmarks could be mapped with a high yield to an NML PLA with only a small amount of redundancy, and area/energy/delay performance would be much better than for the (150,4000,100) PLA implementation.
CHAPTER 7:

NML SYSTOLIC ARCHITECTURES

In the case of NML PLAs, the biggest advantages are defect tolerance and the ability to implement arbitrary logic. However, the flexibility of implementing arbitrary logic with PLAs comes at the cost of extra wires and gates to create each PLA cell. In order to achieve the best possible performance, NML needs to take full advantage of its best characteristics. NML has great potential for low energy computing, and, since the biggest shortcoming with NML PLAs was delay, NML should take advantage of pipelining to provide many computation results much more quickly. Based on experimental results, we are hopeful that NML circuits can be fabricated with low enough fault rates to allow for non-redundant architectural designs. We considered other applications for which NML would be well suited, and found that systolic architectures that implement large amounts of data processing would be an excellent choice.

NML which uses a clocking approach that divides circuits into many clocking zones is inherently pipelined. The non-volatile nature of the NML devices allows the magnets in a zone to save state while no clocking field is present, which can later be used to drive the magnets in the next zones. This means that any circuits we design are forced to be pipelined (although we can set the size of each zone) but they also receive all benefits of pipelining without any extra device overhead. For a large system, there will be a large number of pipeline stages.
We have concluded that one good way to use this characteristic of the technology is to create a high-throughput, pipelined system that can take advantage of a deep pipeline by processing large amounts of data. Systolic architectures fit this description.

As for specific systolic applications, we felt that convolution was a good place to start. We considered both 1-bit and N-bit convolution, which led us to also consider N-bit adders and multipliers. In terms of implementing N-bit circuits, we present both parallel and serial designs. Through simulation, we studied NML feedback circuit elements that would be useful for producing bi-directional dataflow for certain systolic designs and for serial circuits. We also decided that convolution was a good choice to study, because other systolic applications, such as pattern matching and discrete mathematical transforms, are implemented in a similar fashion. In the future, we hope that our NML convolution designs can be modified and extended to other high throughput applications.

7.1 Systolic Dataflow Alternatives

A significant amount of research has explored various systolic architectural designs for many different compute-bound applications. We have considered previous work to find a design strategy for systolic arrays and systolic cells that is well suited for NML’s dataflow characteristics. We want to use NML to implement a simplified convolution\(^1\) function using a systolic array with a simple cell design. The implemented circuit performs 1-bit multiplication (i.e. AND) between

\(^1\)Full 1-bit convolution requires that a sum be accumulated from all 1-bit multiplication operations. While each multiplication result is 1-bit, the sum could be multiple bits. For implementation of complete convolution, adders and multiple wires would be required for each systolic cell. For the 1-bit study, we are mostly concerned with systolic dataflow, so we simplified the application requirement, only ORing the products.
an input stream of bits, and a series of weight bits programmed into the systolic array. Then the results are ORed together. While this design is simple, it provides a starting point to compare designs. In addition, this design could be transformed into pattern matching, for example, by using different gates (XOR-OR or XNOR-AND).

Two systolic designs, given as “Design W1” and “Design W2” in [45], are promising choices for NML implementation of systolic arrays because they are pipelined, have no global interconnect, and require only two inputs. Still, challenges exist when considering an actual NML physical implementation. Design W1 requires that data flow in two opposite directions through the systolic array at the same time (see Fig. 7.1a). This is a possible limitation for NML because of the specific clocking schemes dictated by NML circuits. Design W2 (see Fig. 7.1b) requires that the two signals propagating through the systolic array travel at two different speeds, which could also be difficult for NML clocking.

Figure 7.1. (a) Design W1 and (b) Design W2 from [45].
7.1.1 Unidirectional Design

We considered the W2 Design from [45] as a starting point for creating a unidirectional NML systolic array. One way to achieve propagation of two signals through the systolic array at different speeds would be to delay one signal by using a feedback loop in each systolic cell. We considered this possibility for our NML design, and discovered that a feedback loop would require extra area overhead that would hurt efficiency. Also, the feedback loop would require bi-directional dataflow within each cell, going against the intent of a uni-directional design. Because of these complications we instead designed our own uni-directional NML systolic cell. In this new design (Fig. 7.2a) there are two inputs. The ‘Yin’ signal comes from the previous systolic cell in the array, while the ‘Xin’ signal comes from an input wire. A programmed bit value, indicated by ‘W’ is ANDed with the ‘X’ input, and combined with the cumulative ‘Y’ using an OR operation. The ORed result is then passed on to the next cell in the systolic array. This cell can be replicated to create a linear array of $K$ cells that can compute the simplified convolution for a sequence of $K$ weights and a stream of $N$ input bits.
This cell design is simple, can be used in a systolic array of any length, and operates with data flowing in one direction (from left to right). A simple 3-phase clocking scheme is sufficient to drive the NML circuit in a pipelined fashion, with each clocking wire having the same width as the systolic cell. However, the systolic array requires that there be one input wire for each cell, meaning that there needs to be $K$ inputs per cycle and $K$ horizontal wires. (The $K$ inputs can be reduced to only one input by using a shift register, but the $K$ wires are unavoidable.) These wires can be costly in terms of area and power, especially for a large $K$. An illustration of the systolic array is shown in Fig. 7.3.

Note that the long wires leading into the array are routed in a staircase fashion. This staircase wire structure is employed for long NML wires to avoid long vertical critical paths in the same clocking zone. If these long wires were instead routed as one long horizontal wire and one long vertical wire, then the last wire would have a tall column of magnets leading into the last systolic cell from above. If all the magnets in that column were simultaneously placed into a metastable state, it would lead to a long critical path (which increases with the size of the systolic array) and be more susceptible to fabrication errors, thermal fluctuations, etc. 11 53.

7.1.2 Bidirectional Design

Considering previous work on systolic architectures, we utilized Design W1 proposed in 45 as a starting point for creating a more efficient NML systolic cell (Fig. 7.1a). As discussed briefly in Sec. 7.1, the biggest challenge to creating an NML implementation of this design is the bi-directional dataflow requirement. One way to address this challenge is to use a 2-phase clocking scheme.
Figure 7.3. Systolic array with 4 cells for the uni-directional design. Each clocking wire would be as wide as one cell, and would extend from top to bottom across the input wires and the height of the cell. The inputs enter the design through a shift register on the left.
The bi-directional dataflow functionality for this design was first studied using physical level simulation. We simulated a 2-phase clock and a wire that “turned back” in the same clock zone. This arrangement is shown in Fig. 7.4.

This arrangement of 9 magnets provides a simple test to demonstrate that a signal can propagate in two directions by looping back in the same clock zone. Magnet 1 serves as the signal input (see Fig. 7.4), while magnets 2-8 are in an antiferromagnetically ordered state that does not correspond to the state of the input magnet. Then a clocking field is applied to magnets 2-8. Finally, the clocking field is released, and the magnets are allowed to relax into a new state. Fig. 7.5 illustrates how the angle of magnetization of each device changes in time – and all magnets do switch in the proper order.

Having demonstrated that bi-directional dataflow can be achieved with a 2-phase clock, we adapted Design W1 for NML (see Fig. 7.2b). This systolic processing element – unlike the uni-directional design discussed in Sec. 7.1.1 – can be duplicated and arranged into arrays to perform simplified convolution without the need for $K$ input wires. While we did not simulate the entire systolic cell design, each component in the cell (wires, gates, and fanout) has been demonstrated experimentally.

7.2 NML Adders and Multipliers

In order to explore the advantages of NML systolic architectures more comprehensively, larger N-bit designs must be considered. A 1-bit simplified convolution circuit is useful for exploring dataflow possibilities in a simple case, and there are applications that exploit systolic systems for low-bit convolution and mathematical transformations \([10, 13\). However, more versatile high-throughput applications
Figure 7.4. The arrangement of the nanomagnets in the simulation that demonstrated a single signal of data traveling in two directions in the same clocking zone. Note the horizontal helper magnets next to the magnets in the vertical portion of the wire to assist in nulling the magnets.
Figure 7.5. A plot of the changes in the magnetization angles of the nanomagnets in the simulation demonstrating a signal traveling in two directions in the same clock zone.
require N-bit convolution. Reaching to higher bit width components requires more complicated local routing, which could be difficult for NML circuits. Such issues will be studied here. In addition, an important circuit element for serial NML circuits, a complete feedback loop, will be studied as well.

Two different NML full adder designs will be considered. From there, larger N-bit adders and multipliers can be created that contain many full adders each. In addition, the carry signals of the full adder can be linked with a feedback loop to create serial adders and multipliers. All of these will be considered for NML implementation.

7.2.1 Full Adder

The full adder is a critical component in larger NML systolic designs. Many instances of this circuit element will be used in N-bit adders and multipliers, so it should be as compact and efficient as possible. In [27, 52], an QCA/NML full adder design was proposed that used only majority gates and inverters. However, a more compact full adder design is possible with the use of an implicit NML XOR gate which has a size similar to a shape-based NML AND and OR gate [70].

The full adder designs are shown in Fig. 7.6. They are presented in a schematic-like way that uses shapes (circles, rectangles, and triangles) to represent the NML logic elements and lines to represent NML wires. XOR gates are shown as a plus sign surrounded by a circle inside a circle (similar to the symbol for XOR in Boolean algebra), while majority gates are shown as a square with “MAJ” inside, and inverters are shown as a dark triangle connected to a dark circle. Fig. 7.6a is the XOR-based full adder design, Fig. 7.6b is the majority-based design, and Fig. 7.6c is the black-box view of the full adder.
7.2.2 Parallel Adder

Using an NML full adder design to create an N-bit parallel adder (ripple-carry adder) is straightforward enough in terms of how the full adders are connected to each other. However, bringing the two N-bit inputs to the adder while routing the output leads to a series of wire crossing points, which, in NML, needs to be addressed carefully.

Schematically, an N-bit adder requires wires connecting a previous full adder’s CarryOut signal with the next full adder’s CarryIn signal. Also, wires need to bring the A and B inputs to each full adder. Finally, the Sum output signals from each full adder need to be routed to the final output using wires. To satisfy all these requirements, it is possible to connect the carry signals without wire crossings, but the output Sum signals will cross either the A or B signals. Fig. 7.7 shows an example of a 4-bit parallel adder.

The N-bit adder is created by placing N full adders into a horizontal array so that the Cout signal from each is input into the Cin of the next full adder. For
the most efficient design in NML, the two other inputs to each full adder \( A \) and \( B \) should be the top and bottom inputs so that they can be brought to the adder without crossing the wire connecting the carry signals. The \( A \), \( B \), and \( C_{\text{out}} \) inputs are interchangeable. Therefore, the full adder design in Fig. 7.6 can still be used. This \( N \)-bit NML parallel adder can be expanded to a parallel adder of any size by including more full adders.

Depending on how NML I/O works, there could be overhead in bringing inputs to and taking outputs from this \( N \)-bit parallel adder. Increasing to more bits would require expanding the wire crossing structure for more input and output signals. The same pattern of NML wires and crossings can be used. Overall, this design of an \( N \)-bit NML adder has a significant local routing overhead. However, it is also a pipelined adder that can produce a new addition result after each NML clock cycle.

It might be possible, however, to bring inputs and take outputs from the NML layer out to the CMOS layer through magnetic tunnel junction (MTJ) I/O devices
Figure 7.8. N-bit parallel adder with I/O at each full adder. This design can be extended to an adder of any size.

[48]. If this device can be incorporated into NML logic at select places, it might be possible to give and take each input and output from the parallel adder at the boundaries of the full adders. Fig. 7.8 shows the 4-bit adder without interconnect.

7.2.3 Serial Adder

Another approach to designing an N-bit adder using NML full adders is to combine a full adder with a feedback loop, turning it into a serial adder. Instead of using more area to implement N-bits of addition, a single full adder can produce an N-bit result after N cycles of the NML clock, giving the result one bit at a time. This removes the need for wires to connect full adders and wire crossings for overlapping input and output wires (although wire crossings would still be part of the full adder design). The design is given in Fig. 7.9.

The serial N-bit adder is just a full adder with a feedback loop connecting the carry signals. We assume the XOR-based NML full adder design. Fig. 7.9a shows the schematic-like design. There are 3 required wire crossings, and the $C_{in}$ and $C_{out}$ signals have been connected together. Fig. 7.9b shows the black-box serial adder.
In creating a serial NML adder, area efficiency is at its best, since no extra wires or wire crossings are necessary. The design is very compact, but does require more time to produce each addition result. In Sec. 8.3.1 we will study which is more energy efficient, the parallel or the serial NML adder.

7.2.4 NML Feedback Loop

In Sec. 7.1.2 we talked about using 2-phase clocking to achieve bi-directional dataflow in our proposed 1-bit convolution systolic cell. Now, at the N-bit level, we would like to utilize 2-phase clocking to get bi-directional dataflow in a serial adder design to allow for a tight feedback loop. Our idea was to create two horizontal NML wires running in parallel to each other that are connected by vertical NML wires at each end. These wires would span 3 clock zones so, under a 2-phase clocking scheme, this circuit would act like a 2-cycle feedback loop.
Figure 7.10. Image of the feedback loop simulation setup including trapezoidal magnets needed for bi-directional dataflow with a 2-phase clock. The division of the design into 3 clock zones is shown by vertical black lines. The magnets are numbered for reference. The loop is designed to move signals in a clockwise direction.

(If such a feedback loop could fit into a space spanning only 2 clock zones, then the feedback loop would bring the signal back every cycle, rather than every 2 cycles.) We created a nanomagnet layout for the feedback loop design and tested it though time-evolution in the OOMMF simulator to demonstrate the cycling of signals around the loop. The design layout is shown in Fig. 7.10.

We simulated the feedback loop for 3 full clock cycles starting with opposite binary signals in each half of the loop. The trapezoids are positioned and oriented so that the loop moves signals in a clockwise direction. The magnets in Fig. 7.10 are numbered to help with explanations. We started the simulation by setting magnet 1 to a binary ‘0’ and setting magnet 11 to a binary ‘1’. Then we drove
the entire circuit using a 2-phase clock across all 3 clock zones. We collected and plotted the magnetization angle of magnets 1-9, and the curves, which show the change in magnetization angle over time, are shown in Fig. 7.11.

The curves are split into two groups. The upper plot shows magnets 1-5, which are all part of the upper wire in the 2nd clock zone, and the lower plot shows magnets 6-9, which are part of the 3rd clock group. The first group of curves shows magnet 1 switching from down to up to down (binary ‘0’ to ‘1’ to ‘0’), and,
since magnets 1-5 are all antiferromagnetically coupled, they switch up-down-up or down-up-down accordingly over the course of the 3 cycles that are simulated. These curves show that magnets 1-5 switch correctly as the two opposite binary signals pass around the loop. The second group of curves show the same successful switching as the signals pass around the loop, however, magnets 6-9 are coupled differently. Magnet 6 switches to a magnetization state opposite of what magnets 7-9 switch to. This is correct, since magnets 7-9 are part of a vertical NML wire. These simulation results show that a feedback loop is possible with NML using 2-phase clocking.

7.2.5 Parallel Multiplier

The next important component of a complete NML convolution circuit is an N-bit multiplier.

We propose a parallel NML multiplier organized like a ripple carry array multiplier. Such a design requires two parts. First, there needs to be an array of AND gates to produce the partial products and a series of full adders to sum up the partial products. There are many ways to create a parallel multiplier using Boolean circuits. However, an NML implementation would do best to have as few crossings as possible. One way to achieve this is to use a ripple carry array of full adders. This approach is much like unrolling a shift-and-add multiplier. The design for a 2-bit multiplier is shown in Fig. 7.12.

This approach for making an NML multiplier is somewhat area inefficient. There are a lot of components required to make this multiplier, and many NML wires with crossings needed to connect the components. The local interconnect requirement grows quickly for larger input sizes. However, all the components of
Defects and Faults in QCA-Based PLAs
Michael Crocker, X. Sharon Hu, and Michael Niemier
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
Email: {mcrocker,shu,mniemier}@nd.edu

Defect tolerance will be critical in any system with nano-scale feature sizes. This paper examines some fundamental aspects of defect tolerance for a reconfigurable system based on Quantum-dot Cellular Automata (QCA). We analyze a novel, QCA-based, Programmable Logic Array (PLA) structure, develop an implementation independent fault model, and discuss how expected defects and faults might affect yield. Within this context, we introduce techniques for mapping Boolean logic functions to a defective QCA-based PLA. Simulation results show that our new mapping techniques can achieve higher yields than existing techniques.

Figure 1. Figure to be extracted

Categories and Subject Descriptors: B.6.2 [Hardware]: Logic Design—Reliability and Testing, Redundant Design

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Nanotechnology, Quantum-dot Cellular Automata, Defects, Faults, Logic Mapping

1. BACKGROUND

Figure 7.12. NML Parallel multiplier design. There are 2 major parts to the design: (1) AND gate array to compute the partial products, and (2) full adder array to sum up partial products.

7.2.6 Serial Multiplier

A serial NML multiplier could be better in terms of area and energy compared to a parallel NML multiplier because it would be smaller in area and require only small interconnect overheads. However, each N-bit multiply result would...
Defects and Faults in QCA-Based PLAs

Michael Crocker, X. Sharon Hu, and Michael Niemier

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
Email: \{mcrocker, shu, mniemier\}@nd.edu

Defect tolerance will be critical in any system with nano-scale feature sizes. This paper examines some fundamental aspects of defect tolerance for a reconfigurable system based on Quantum-dot Cellular Automata (QCA). We analyze a novel, QCA-based, Programmable Logic Array (PLA) structure, develop an implementation independent fault model, and discuss how expected defects and faults might affect yield. Within this context, we introduce techniques for mapping Boolean logic functions to a defective QCA-based PLA. Simulation results show that our new mapping techniques can achieve higher yields than existing techniques.

Categories and Subject Descriptors: B.6.2 [Hardware]: Logic Design—Reliability and Testing, Redundant Design

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Nanotechnology, Quantum-dot Cellular Automata, Defects, Faults, Logic Mapping

1. BACKGROUND

Figure 7.13. Serial multiplier made from many serial adders, one for each bit width. This particular example is for a 4-bit multiplier. For each serial adder there is also an AND gate for computing the partial products.

take longer to arrive, as it would only generate one bit per NML clock cycle. In addition, there would also be some wasted computations as all resources in the multiplier would be used each clock cycle, even if not performing computation useful to the multiplication result.

A serial approach to an NML N-bit multiplier would reduce the area needed to implement the multiplier while also simplifying the local routing to almost nothing. Overall, this should create a more efficient design, especially for larger input sizes. However, the area saving is traded for increased delay to see each multiply result completed. Instead of getting a result every NML clock cycle, each result would be complete in 2N cycles. Such a design is shown in Fig. 7.13.

The N-bit serial multiplier is composed of N serial adders and N AND gates to produce the multiplication partial products. A certain pattern is required for bringing input bits to the multiplier. Each bit of the B input can stay in place throughout the multiplication, but each bit of the A input needs arrive at each locations marked “A” in Fig. 7.13 on consecutive cycles from least significant to...
most significant followed by $N$ zeros. For the 4-bit example, the 8-bit multiply result is output after 8 cycles at “P” in Fig. 7.13.

This serial multiplier design, which trades area for delay, will be part of a performance analysis in Sec. 8.4.2 to determine which NML multiplier design is the most energy efficient.

7.3 NML Convolution

Using the NML $N$-bit adders and NML $N$-bit multipliers, NML convolution circuits can be created at many combinations of input and output sizes. However, there are some considerations beyond just plugging the parts together. Again, NML wires to connect the circuits can be tricky. Additionally, there are timing concerns in terms of the arrival of inputs, and instead of a second input to the multiplier, there are programmed weights that need to be placed adjacent to the multiplier. To perform convolution, there needs to be an input vector $X$ and a weight vector $W$. The output is a vector $Y$ that contains the sum of many multiplication products depending on the size of the input and weight vectors. If the $X$ vector is of length $n$ and there are $m$ weights, then the output vector will be of length $k$, where $k = n - m + 1$. Each value of the output vector $Y$ can be calculated as shown in Eq. 7.1.

$$y_i = w_1 \times x_i + w_2 \times x_{i+1} + ... + w_m \times x_{i+m-1} \quad (7.1)$$

Basically, there are many multiplies and adds that need to be performed in a simple pattern to complete the computation. Already, we have presented NML adder and multiplier designs. Therefore, the other issues that remain for NML
convolution circuits are deciding what size of multipliers and adders to use and figuring out the best way to connect the multiplier and the adder. Once a convolution cell is created using one adder and one multiplier, a convolution with a kernel of any size (any number of weights) can be made by creating an array of convolution cells of the necessary size (equal to the number of weights).

7.3.1 Parallel Convolution

Parallel convolution cells can be made from parallel adders and parallel multipliers. The multiplier takes two N-bit inputs, while the adder is connected to the multiplier’s outputs by NML wires. If the multiplier takes N-bit inputs, then the adder should take at least 2N-bit inputs. An example parallel convolution design is shown in Fig. 7.14 with a 2-bit multiplier and a 4-bit adder. This can be scaled up to many combinations of multiplier and adder.

Considering the parallel convolution design in Fig. 7.14, one thing that stands out is that there are rather long NML wires that connect the output of the multiplier to the input of the adder. The routing of inputs and outputs for the adder look more like the adder presented in Fig. 7.7 which has many NML wires and wire crossings as part of the adder design. For larger input sizes, the parallel convolution circuit would suffer from the same inefficiencies as the parallel multiplier. Therefore, we considered a serial convolution circuit as well.

7.3.2 Serial Convolution

A serial convolution circuit can be created in the same way as a parallel one, by joining a serial multiplier and a serial adder of N-bit and 2N-bit size, respectively. An example serial convolution design is shown in Fig. 7.15 with a 4-bit multiplier.
and an 8-bit adder. As was the case with the multipliers, the wire routing for serial convolution is much simpler than for parallel convolution.
Defects and Faults in QCA-Based PLAs

Michael Crocker, X. Sharon Hu, and Michael Niemier

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
Email: {mcrocker, shu, mniemier}@nd.edu

Defect tolerance will be critical in any system with nano-scale feature sizes. This paper examines some fundamental aspects of defect tolerance for a reconfigurable system based on Quantum-dot Cellular Automata (QCA). We analyze a novel, QCA-based, Programmable Logic Array (PLA) structure, develop an implementation independent fault model, and discuss how expected defects and faults might affect yield. Within this context, we introduce techniques for mapping Boolean logic functions to a defective QCA-based PLA. Simulation results show that our new mapping techniques can achieve higher yields than existing techniques.

Categories and Subject Descriptors: B.6.2 [Hardware]: Logic Design—Reliability and Testing, Redundant Design

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Nanotechnology, Quantum-dot Cellular Automata, Defects, Faults, Logic Mapping

Figure 7.15. Serial convolution circuit (4-bit multiply and 8-bit add)
CHAPTER 8:
SYSTOLIC PERFORMANCE STUDY

In this chapter, performance comparisons will be made between different approaches to NML designs, and between NML and CMOS designs for adders, multipliers, and systolic convolution. We will consider nanomagnet layouts for the designs given in Chapter 7, which will allow for performance estimates in terms of area, delay, and energy. The designs given here have considered timing for both the NML clocking requirements and the architectural requirements. For example, to satisfy the clock timing requirements for the serial designs, we needed to consider how the 2-phase clocking could be used to make the feedback loop work in conjunction with the NML logic gates and wires. An example of satisfying the architectural timing requirements is how we designed the serial multiplier design to keep all $B$ input bits in place near the $\text{AND}$ gates as the $A$ input bits move through the multiplier one after the other. Such considerations were made for all layouts, including the N-bit systolic design and the multi-layer NML serial adder design.

8.1 Performance Estimate Assumptions

For the different NML performance estimation studies reported in this chapter, there were some common assumptions made about the technology.
First, NML devices were assumed to be 40x60x20 nm$^3$ in size with 10 nm of horizontal spacing and 15 nm of vertical spacing between devices. This magnet size was chosen because it represents the smallest devices that can be fabricated in our lab. The dimensions of the magnets affect both the area of designs made from those devices and the switching energy per device.

Second, the equations used to calculate clock wire power and energy are the same in all the studies, and it is based on the performance calculation approach used in [52]. The resistance of the clock wire is determined by the length ($l$), width ($w$), and thickness ($t$) of the wire as well as the conductivity constant for copper, since we assume copper wires (see Eq. 8.1). For the resistivity of copper, we assume 18 Ω$\cdot$nm. Once resistance has been calculated, the power consumption of the clock wire can be calculated from the current using Eq. 8.2, where $D$ is the duty cycle of the clock signal. For a 3-phase clock, $D = 2/3$, and for a 2-phase clock, $D = 1/2$. Finally, the energy consumption is just the delay multiplied by the power. While the equations are the same in all studies, the values used for $l$, $w$, and $t$ are different, and the specific values used will be mentioned in the section.

$$R = \frac{\sigma \cdot l}{w \cdot t}$$ (8.1)

$$R = D \cdot I^2 \cdot R$$ (8.2)

Another common assumption has to do with the field strengths required to operate certain NML circuits. For NML circuits with uni-directional dataflow using a 3-phase clock scheme, we assume that a 5 mT clocking field is required. However, for any NML circuits that use bi-directional dataflow using a 2-phase clocking
scheme and trapezoid magnets at the boundaries of clock zones, we assume that a 25 mT clocking field is required (although we do examine the possibility of lowering this requirement through further improvements in bi-directional circuits). These field strength assumptions come from work done in [25]. The consequence of these differences is that the designs that require bi-directional dataflow also require higher currents on the clock wires to produce the high magnetic clocking fields.

Also worth mentioning is how we calculate total energy consumption. In one study we consider both magnet switching energy dissipation and clock wire energy dissipation. In that study we just assume that each magnet switching event dissipates energy equal to $100kT$. However, in all our studies, the clock energy always dwarfs the magnet switching energy. Therefore, in some studies, we drop the magnet switching energy calculation.

In our performance studies, delay is not as consistently determined. In some studies we calculate a critical path delay based on an average per-magnet switching time, which is usually around 200 ps per magnet. However, in other studies, we just assume a 9 ns NML clock cycle time based on the results of [25]. The specific assumptions will be given in each section.

In our comparison of parallel and serial NML N-bit adders and multipliers, we normalize our energy and delay results. The base design that these results are normalized to is the NML full adder. For a full adder, we calculated the delay based on a 9 ns cycle time and a 3-phase clocking scheme, and we calculated energy based on a 1.4 mA wire current (to produce a 5 mT clock field) on clock wires that are 300 nm width and 900 nm thick (this 3:1 aspect ratio for clock wires
is based on what is possible to fabricate with today’s technology). The layout of
the full adder is shown in Fig. 8.6 as full adders make up the parallel adder layout.

In addition, the possibility of using a high permeability material around the
NML devices can lead to reductions in the current required on the clock wires. As
discussed in [25], this material can allow the same clock wire current to produce
higher strength magnetic fields in the plane of the nanomagnets and allow a lower
wire current to produce the same strength magnetic fields. This material can have
different relative permeabilities anywhere from $\mu_r = 3$ up to $\mu_r = 12$, which would
allow reductions of current wire currents by a factor of 3 up to a factor of 12. Such
material enhancements are considered in some of the studies in this chapter.

8.2 1-bit Systolic Convolution Comparison

The two systolic designs described in Sec. 7.1 are very similar in terms of the
cell layout. Both designs assume 2-input, shape-based AND and OR gates and a
reprogrammable bit (W). The nanomagnet layouts of both cell designs are also
very similar. A schematic of the uni-directional cell design appears in Fig. 8.1(a),
and the bi-directional design in Fig. 8.1(b).

While there are some similarities between the two designs, there are also im-
portant differences. As mentioned earlier, one difference is that the uni-directional
array requires $K$ horizontal wires leading into the array from a $K$-bit shift reg-
ister, while the bi-directional array does not. Another difference is that the uni-
directional design is controlled by a 3-phase clock, and can operate with lower
clocking fields (approx. 5mT), while the bi-directional design is controlled by a 2-
phase clock, and could need higher clock fields (approx. 25mT), and a higher drive
current [25]. Also, the bi-directional design requires bubbles in the pipeline every
Figure 8.1. Systolic cell magnet layout (a) for the uni-directional design, and (b) for the bi-directional design.

### TABLE 8.1

**IMPORTANT NML PERFORMANCE PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magnet Footprint</td>
<td>3750</td>
<td>$nm^2$</td>
<td>$F_M$</td>
</tr>
<tr>
<td>Clock Period</td>
<td>3.6</td>
<td>$ns$</td>
<td>$P$</td>
</tr>
<tr>
<td>Magnet Switching Energy</td>
<td>0.414</td>
<td>$aJ$</td>
<td>$100kT$</td>
</tr>
<tr>
<td>Current Required (5mT)</td>
<td>1.2</td>
<td>$mA$</td>
<td>$I_u$</td>
</tr>
<tr>
<td>Current Required (25mT)</td>
<td>5.55</td>
<td>$mA$</td>
<td>$I_b$</td>
</tr>
<tr>
<td>Clock Wire Resistance</td>
<td>56.3</td>
<td>$m\Omega$</td>
<td>$R$</td>
</tr>
<tr>
<td>Cell Footprint</td>
<td>0.09375</td>
<td>$\mu m^2$</td>
<td>$F_C$</td>
</tr>
</tbody>
</table>
other cycle – so it takes roughly two times longer for all \( N \) inputs to propagate through the array.

In order to do a full comparison of both designs, we consider area, energy, and delay. To quantify the aforementioned metrics, we will first define basic parameters. We assume magnets that are 40x60x20\( \text{nm}^3 \) in size with a 10\( \text{nm} \) horizontal and 15\( \text{nm} \) vertical spacing leading to a magnet footprint of 50\( \text{nm} \) * 75\( \text{nm} \) = 3750\( \text{nm}^2 \). Each cell is 5 magnets wide and 5 magnets tall, so each cell is 250\( \text{nm} \) wide and 375\( \text{nm} \) long. We also assume that the clocking wires will be 600\( \text{nm} \) thick – a realistic fabrication target since lines with similar feature sizes are used in field MRAM. By using 2D Maxwell simulations [1], we can estimate the currents needed for generating the magnetic clocking field strengths required by the two designs. Finally, we assume that for both cells the critical path has a delay of 1.8\( \text{ns} \), based on 9 magnets on the critical path and 200\( \text{ps} \) switching time per magnet [34]. This translates to a clock period (\( P \)) of approximately 3.6\( \text{ns} \) for both designs. Parameters are summarized in Table 8.1.

Parametrized expressions for area and delay – in terms of \( K \) (the number of systolic cells in the array), \( N \) (the size of the input vector in bits), and \( A \) (the number of input vectors) – are given below. Area for the uni-directional design (Eq. 8.3) is calculated using the magnet footprint area multiplied by three terms. The first term (25\( K \)) corresponds to the \( K \) systolic cells in the array 5 magnets wide by 5 magnets tall. The second term (12\( K \)) is for the shift register, each bit-slice of which is 4 magnets wide by 3 magnets tall. Finally, the area of the staircase-shaped wires leading into the array is calculated using half the area of the bounding rectangle. The rectangle is 5\( K \) magnets wide (width of the array) and 2 * (\( K − 1 \)) magnets tall, since there are \( K − 1 \) wires stacked up, with each
wire taking up vertical space equal to the height of 2 magnets. The area for the bi-directional design is simply given by the area of $K$ cells (Eq. 8.4).

The total delay for the uni-directional design (Eq. 8.5) is calculated as follows: First, it takes $K$ cycles to fill the shift register with input bits, then it takes $K$ cycles to bring the first output signal across the systolic array, and finally, the $M = N - K + 1$ outputs are produced in $M$ consecutive cycles. The delay for the bi-directional design is given by Eq. 8.6. $K$ cycles are required to bring the first input bit across the array. An additional $K$ cycles are needed to produce the first output. Finally, the remaining $(M-1)$ outputs are each produced every two cycles. The delay expressions can be simplified in that $A$ can be multiplied by the $N$ term in both equations to calculate the delay for $A$ consecutive input vectors. This is important for the uni-directional design, since the $K + 1$ term is the delay overhead to fill the pipeline, and only needs to be accounted for with the first input vector.

\[
Area_u = F_M \times [25K + 12K + (2(K - 1) \times 5K)/2] = F_M \times (5K^2 + 32K) \tag{8.3}
\]

\[
Area_b = F_M \times 25K \tag{8.4}
\]

\[
Delay_u = P \times (K + K + M) = P \times (N + K + 1) \tag{8.5}
\]

\[=> P \times (A \times N + K + 1)\]

\[
Delay_b = P \times [K + K + 2(M - 1)] = P \times 2N \tag{8.6}
\]

\[=> P \times 2A \times N\]

The number of inputs ($N$) does not affect area, but is very important for
Figure 8.2. Area comparison of both designs, keeping the number of inputs (N) constant, while varying the number of systolic cells (K).

calculating delay. Fig. 8.2 plots area of both designs as a function of K, and Fig. 8.3 plots delay as a function of N. The area curves show that the bi-directional design scales linearly vs. K, while the uni-directional design scales quadratically with K. For large systolic arrays (large K), the bi-directional design is more area efficient. The delay curves show that both designs scale linearly with N, but that the bi-directional delay increases faster by a factor of two.

We next considered the energy of each design. There are two main sources of energy dissipation for NML circuits: the magnets themselves and the clocking wires that control the magnets. The energy dissipated by the clocking wires tends to dwarf the energy dissipation by the magnets [19], but we included both to be complete. The basic idea is to multiply the switching energy for one magnet (100kT) by the number of magnets in each design and the number of clock cycles during operation. The number of cycles (#Cyc) can be determined by dividing
Figure 8.3. Delay comparison of both designs for $A = 100$, keeping the number of systolic cells ($K$) constant, while varying the number of inputs ($N$).

the total time for each design by the clock period. This appears in the magnet energy equation for the uni-directional design (Eq. 8.7) as the term ($\text{Delay}_u/P$). Eq. 8.7 also needs terms for the number of magnets in the entire design. There are 12 magnets in each cell, 6 magnets in each bit-slice of the shift register, and 5 magnets for each wire segment.

For the bi-directional design, the magnet energy calculation is much simpler (See Eq. 8.8). The clocking energy can be calculated by multiplying the power consumed by each clocking wire with the total delay, and then multiplying that by the number of clocking wires. The term ($\text{Area}/F_C$), which is the total area of one design divided by the area of one clock wire, corresponds to the number of clock wires in the design. Eq. 8.9 is for the uni-directional design, and Eq. 8.10 is for the bi-directional design. The coefficient in each expression depends on the
number of clock phases (and is obviously higher for the 3-phase clock). Total energy is simply the sum of the magnet and clocking energies.

\[
ME_u = 100kT \times [12K + 6K + (5K^2)/2] \times \#Cyc
\]

\[
= 100kT \times (18K + 2.5K^2) \times (Delay_u/P)
\]

\[
ME_b = 100kT \times 14K \times (Delay_b/P)
\]

\[
CE_u = (2/3)I_u^2 \times R \times Delay_u \times (Area_u/F_C)
\]

\[
CE_b = (1/2)I_b^2 \times R \times Delay_b \times (Area_b/F_C)
\]

Using the above expressions, we compared the designs in terms of total energy. Fig. 8.4 illustrates energy as a function of the size of the systolic array (K). The bi-directional design at first increases at a greater rate (due to the higher current requirements). However, the clocking energy is also impacted by the number of the clocking lines, which is related to the area. When the systolic array becomes large, the energy for uni-directional design grows quadratically and passes the bi-directional design even though the energy per wire is less.

We end our discussion by comparing the area-energy-delay product (AEDP) of both designs. Fig. 8.5 shows the results of that analysis as the number of systolic cells (K) is increased. The results again indicate that for large systolic arrays, the bi-directional design performs best.
Figure 8.4. Energy comparison of both designs for $A = 100$, keeping the input size ($N$) constant, while varying the number of systolic cells ($K$).

Figure 8.5. Area-Energy-Delay comparison of both designs for $A = 100$, keeping the number of inputs ($N$) constant, while varying the array size ($K$).
8.3 Parallel Versus Serial

In this section, we will consider which approach to designing circuits is best suited to NML for high-throughput applications: parallel or serial. Just from a theoretical standpoint, serial designs trade an increase in delay for a decrease in area compared to parallel designs. However, the most important metric for NML is energy. For any computational task, there are a certain number of logical operations that are required to complete the task. By choosing certain approaches, the total performance cost (area, energy, delay) can be higher or lower. In addition, arranging the circuit components into a specific design can change the performance as well. There are overheads associated with any design that add to the minimum requirements for the desired computation. For NML, often times there is a significant overhead associated with interconnect whether local or global. By looking at locally connected systolic architectures, we hope to avoid large global interconnect, however, even connecting local components can change the NML layout and affect performance.

For this study, we focus on addition and multiplication, as these operations form the basis for systolic convolution. We have already considered a very compact systolic cell designed to perform a simplified 1-bit convolution in order to determine the best dataflow pattern for NML systolic architectures. Now we will consider the general case of N-bit convolution to determine if parallel or serial is better for NML systolic architectures. The comparison will start by looking at serial N-bit adders compared to parallel N-bit adders. The second part of the comparison will look at N-bit serial and parallel multipliers. The comparisons will be based on the designs from Chapter 7.
Defect tolerance will be critical in any system with nano-scale feature sizes. This paper examines some fundamental aspects of defect tolerance for a reconfigurable system based on Quantum-dot Cellular Automata (QCA). We analyze a novel, QCA-based, Programmable Logic Array (PLA) structure, develop an implementation independent fault model, and discuss how expected defects and faults might affect yield. Within this context, we introduce techniques for mapping Boolean logic functions to a defective QCA-based PLA. Simulation results show that our new mapping techniques can achieve higher yields than existing techniques.

8.3.1 NML Adder Comparison

Here we compared parallel and serial NML designs for N-bit adders. We wanted to see how the two designs compared for increasing bit widths, all the way up to 32-bit add. In order to complete the comparison, we laid out the adder designs using basic NML circuit components (wires, gates, fanout, crossings) to get the specifications we need to calculate the performance estimates. The nanomagnet layouts were done for bit widths of 4, 8, 12, 16, 24, and 32. The parallel adder layout is shown in Fig. 8.6, and the serial adder layout is shown in Fig. 8.7(b).

The parallel adder layout shows what the adder looks like for an 4-bit design. In this case, 4 full adders are need. As the input size increases, more full adders are needed (more area and energy), and the pipeline gets deeper. However, a new result can be obtained each cycle due to pipelining. This layout can easily be scaled up to a parallel adder of any size.

The serial adder layout shows what the adder looks like, in general, for an
Defects and Faults in QCA-Based PLAs

Michael Crocker, X. Sharon Hu, and Michael Niemier
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
Email: {mcrocker, shu, mniemier}@nd.edu

Defect tolerance will be critical in any system with nano-scale feature sizes. This paper examines some fundamental aspects of defect tolerance for a reconfigurable system based on Quantum-dot Cellular Automata (QCA). We analyze a novel, QCA-based, Programmable Logic Array (PLA) structure, develop an implementation independent fault model, and discuss how expected defects and faults might affect yield. Within this context, we introduce techniques for mapping Boolean logic functions to a defective QCA-based PLA. Simulation results show that our new mapping techniques can achieve higher yields than existing techniques.

Categories and Subject Descriptors: B.6.2 [Hardware]: Logic Design—Reliability and Testing, Redundant Design

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Nanotechnology, Quantum-dot Cellular Automata, Defects, Faults, Logic Mapping

1. BACKGROUND

Figure 8.7. (a) Serial adder design repeated for comparison, and (b) serial adder layout based on the XOR full adder design and the NML feedback loop.
N-bit design. Only one full adder with a feedback loop is required to implement a serial adder of any size. The only thing that changes with increased input size is the number of cycles required to produce the N-bit sum.

We calculated the delay and energy consumption for both the serial and the parallel NML adder designs. For the parallel design, we assumed a clock wire current that could provide a 5 mT clocking field. For the serial design, we assumed 3 different currents that could produce a 5, 10, and 25 mT clocking field, since simulation results indicated that the feedback loop could require higher fields. The calculations included overheads for filling the pipeline, and assumed the serial design interleaved two adds at the same time because of the 2-cycle feedback loop. Both performance calculations were completed assuming that 1,000 adds were performed in succession so that the pipeline could be filled. Then the results for total delay and energy were divided by 1,000 to get the average per add. All results are normalized to the energy and delay performance for a full adder as discussed in Sec. 8.1. The results are plotted in Fig. 8.8.

The curves for delay are not surprising, as the parallel adder provides each add on average in a shorter time than the serial adder does. The energy curves show that the parallel adder uses less energy per add (requiring only a 5 mT field) than the serial adder uses in the cases of higher required fields (10 and 25 mT). In fact, at 25 mT, the energy for the serial adder is off the charts. However, if the serial design could be operated with a 5 mT clocking field, it would require less energy per add.
Figure 8.8. Comparison of NML adders both in terms of energy and delay per addition operation.
Defects and Faults in QCA-Based PLAs

Michael Crocker, X. Sharon Hu, and Michael Niemier

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA

Email: \{mcrocker, shu, mniemier\}@nd.edu

Defect tolerance will be critical in any system with nano-scale feature sizes. This paper examines some fundamental aspects of defect tolerance for a reconfigurable system based on Quantum-dot Cellular Automata (QCA). We analyze a novel, QCA-based, Programmable Logic Array (PLA) structure, develop an implementation independent fault model, and discuss how expected defects and faults might affect yield. Within this context, we introduce techniques for mapping Boolean logic functions to a defective QCA-based PLA. Simulation results show that our new mapping techniques can achieve higher yields than existing techniques.

Categories and Subject Descriptors: B.6.2 [Hardware]: Logic Design—Reliability and Testing, Redundant Design

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Nanotechnology, Quantum-dot Cellular Automata, Defects, Faults, Logic Mapping

1. BACKGROUND

8.3.2 NML Multiplier Comparison

Here we compared parallel and serial NML designs for N-bit multipliers. We wanted to see how the two designs compared for increasing bit widths, all the way up to a 16-bit multiply. In order to complete the comparison, we laid out the multiplier designs using basic NML circuit components (wires, gates, fanout, crossings) to get the specifications we need to calculate the performance estimates. The parallel multiplier layout is shown in Fig. 8.9 and the serial multiplier layout is shown in Fig. 8.11.

The parallel multiplier design requires $N^2$ AND gates to compute all the partial products and $(N - 1) \times N$ full adders to sum up the partial products based on bit significance. In addition to these components, there are many NML wires to connect all these components. For the 2-bit parallel multiplier example in Fig. 8.9...
there are not many wires, and no wire crossings are needed beyond the ones that are part of the AND gate array and the 3 needed per full adder. However, as the inputs to the multiplier increase in size, the arrays will get bigger and the wires will get longer and the routing more complex. Because of this, we did not do complete nanomagnet layouts of parallel multipliers for larger than 4-bit inputs. The 4-bit parallel multiplier design is shown in Fig. 8.10.

Looking at how the design increases in area going from 2-bit to 4-bit, we created 2 equations that estimate the area of larger parallel multiplier designs. This was done so that comparisons could be made to 8, 12, and 16-bit parallel multipliers more easily. One equation determines the width of an NML parallel multiplier in units of clock zones. The second equation determines the height of
the multiplier in terms of the number of magnets vertically. Both equations are in terms of the input size $N$.

\[
Width = [N \times N] + [N \times N + (N - 2) \times N + N/2] = 3N^2 - 3N/2
\]  
(8.11)

\[
Height = [N \times (N + 1) \times 2] + [N \times 2] = 2N^2 + 4N
\]  
(8.12)

Eq. 8.11, which estimates the width of the multiplier design, was determined in two parts. The first part of the equation has to do with the horizontal space used up by the array of AND gates used to compute the partial products. For the 2-bit multiplier, there are about 2 clock zones per column of AND gates, which contain the required NML gates, crossings, and wires. For the 4-bit multiplier, there are about 4 clock zones per column of AND gates. As the size of the AND gate array increases, more results need to be routed through the array to the right side of the array, requiring more space between each AND gate as the number of input bits increases. So for the 2-bit case, there are 2 columns of AND gates, each requiring about 2 clock zones of horizontal space. For the 4-bit case, there are 4 columns of AND gates, each requiring about 4 clock zones of horizontal space. For 2-bits, we estimate $2 \times 2$ total clock zones, and for 4-bits, we estimate $4 \times 4$ total clock zones. This trend should continue for N-bits, so we introduced the $[N \times N]$ term to Eq. 8.11.

For the second term in Eq. 8.11, the horizontal space that the adder array requires can be determined by the number of columns of full adders and an increasing interconnect requirement due to more partial product signals to route. For the 2-bit multiplier, the full adders are tightly packed and only require 2 clock
zones of horizontal space. There is only one column of full adders, so no additional horizontal space is required for more columns. Finally, we estimate 1 extra clock zone of horizontal space for routing in the 2-bit case. For the 4-bit multiplier, the full adders are not as tightly packed, so they average about 4 clock zones of horizontal space per full adder. There are 2 additional columns of full adders in the design, each taking up 4 more zones. Finally, there are 2 more clock zones for extra interconnect. That leads to the $N \times N + (N - 2) \times N + N/2$ term in Eq. 8.11. Note that the second line of Eq. 8.11 is the simplified form.

Eq. 8.12 was also determined in two parts. We determined that the vertical space required for the multiplier was a function of the input size, because there are $N$ rows of AND gates in the array. As the design increases in size, the vertical space between AND gates also increases as more signals are being routed through the AND gate array. There are 3 wires vertically for each AND gate row in the 2-bit case, and 5 wires vertically for each row in the 4-bit case. The vertical separation between wires is twice the height of a magnet plus twice the vertical spacing between magnets. The first term of Eq. 8.12 ($N \times (N + 1) \times 2$) comes from the vertical wire requirement of $N + 1$ for each of the $N$ rows of AND gates multiplied by 2 vertical magnets per vertical wire separation. The second term in Eq. 8.12 ($2N$) comes from the $N$-bit B input at the bottom of the array, which requires $N$ rows of wires.

These equations give the approximate size of the 2-bit and 4-bit NML multipliers. At $N = 2$, the width is $3 \times 2^2 - 3 \times 2/2 = 9$, and the height is $2 \times 2^2 + 4 \times 2 = 16$. These two values are very close to the 2-bit layout which is 9 clock wires wide and 17 magnets tall. At $N = 4$, the width is $3 \times 4^2 - 3 \times 4/2 = 42$, and the height is $2 \times 4^2 + 4 \times 4 = 48$. These two values are very close to the 4-bit layout which
is 42 clock zones wide and 47 magnets tall. We will use these equations when completing the parallel versus serial performance comparison to determine area requirements for parallel multipliers designs that have 8, 12, and 16 bit inputs.

The serial multiplier design shown in Fig. 8.11 is simpler to layout as compared to the parallel multiplier design. There only need to be $N$ serial adders connected together, and $N$ AND gates to supply partial products to the line of serial adders. The design can be scaled up to larger input sizes easily and without many extra NML wires.

Now that we have given layouts for the serial and parallel NML multipliers, we can do a performance comparison. We considered multipliers with input sizes from 2-bit up to 16-bit. A plot of energy and delay per multiply for both designs is shown in Fig. 8.12, similar to how the adder comparison was presented.

We calculated the delay and energy consumption for both the serial and the parallel NML multiplier designs. Again, one wire current was used for the parallel design while three different currents were used for the serial design corresponding
Figure 8.12. Comparison of NML multipliers in terms of energy and delay per multiplication operation.
to 5, 10, and 25 mT clocking fields. The calculation included overheads for filling the pipeline, and assumed the serial design interleaved two multiplies at the same time. Both performance calculations were completed assuming that 1,000 multiplies were performed. Then the results for both delay and energy were divided by 1,000 to get the average per multiply. All results are normalized to the performance of one NML full adder as discussed in Sec. 8.1. The results are plotted in Fig. 8.12.

As expected, the parallel multiplier averages a smaller delay per multiply. However, because of the rapid growth in area for the parallel multiplier, the serial adder is more energy efficient for multiplier designs for larger input sizes at 10 and 5 mT. For smaller input sizes, the parallel multiplier designs are more energy efficient. And, the serial multiplier, if it needs 25 mT to operate correctly, will still not best the parallel design, even for a 16-bit multiply.

8.4 Technology Comparison: NML vs. CMOS

While it is important to compare NML designs and determine the most efficient designs possible in NML, it is also interesting and necessary to compare the best NML designs to CMOS in order to provide insight into the possible usefulness of the technology compared to the current standard.

8.4.1 Simplified 1-bit Convolution

In order to compare NML to CMOS, we consider a single, bi-directional systolic cell as the common element. (Any performance results from one cell of the bidirectional design implemented on either technology could be multiplied out to get results for a systolic array of any size K implementing the simplified 1-bit
convolution. We did not examine how the design changes for n-bit convolution.)

From the NML cell design, we created an equivalent, pipelined CMOS design made up of 2 NAND gates, 1 inverter, and 3 flip-flops (needed for the programmable bit and 2 pipeline registers). Then we took that CMOS design and performed HSPICE simulations to determine energy and critical path delay. For the CMOS transistors we used 90nm, 45nm, and 22nm node HSPICE models from [2]. These include low power (LP) and high performance (HP) CMOS models at the 45nm node and the LP CMOS model for the 22nm node.

We used all these models because it is hard to find a fair comparison point when comparing CMOS to NML. By sweeping a large technology space for CMOS, we feel that we can make a thorough comparison. (Note that NML devices can be fabricated in a simple one-layer process on top of the clocking wires, possibly using imprint lithography combined with photolithography for the NML clocking wires.) For the sake of a most fair comparison, consider NML versus the 22nm LP CMOS model.

For the CMOS implementation, a number of voltage levels were tested (0.6-1.2V) for each HSPICE model. The energy and delay results are plotted in Fig. 8.13 as a logarithmic plot of delay and energy. There are also three NML datasets in the figure for different clocking currents (1.2, 2.3, and 5.55mA). These three curves show the range of possible performance improvements for the NML design if clocking wire current can be reduced (See [25] for possible approaches). We also incorporated improvements in the average magnet switching time into the three NML datasets. The energy per nanomagnet switching event is 200ps per magnet ([34]), but we consider switching times in the range of 50-300ps in increments of 50ps in Fig. 8.13. In almost all cases, except for a few of the 22nm LP CMOS
Figure 8.13. A plot of energy and delay for CMOS and NML implementations of the bi-directional systolic cell for different technology characteristics. EDP is shown with diagonal dashed lines in units of fJ-ns.

datapoints, energy is lower for all of the given NML datapoints. The middle of the NML grouping is lower energy than all of the CMOS datapoints.

8.4.2 N-bit Addition and Multiplication

Scaling up to larger sized adders and multipliers is the first step in reaching toward application-level comparison between NML and other technologies. It is especially difficult to do a comparison between transistor-based technologies and
a technology that is based on coupled magnetic interactions. However, it should be possible to do a fair comparison starting with N-bit adders, and then moving on to N-bit multipliers and then convolution with different sized kernels. Such a study should provide a nice starting point for future application-level comparisons between NML and other technologies that look at other computations like pattern matching and other mathematical transforms.

First is the comparison of adders. Because NML is aimed at low-energy applications, it is worth looking at sub-threshold CMOS as a practical comparison point. In [44], the authors look at both parallel and serial CMOS adders in terms of energy and delay. The main focus of [44] is to show how adder designs for low-energy operation should favor lower leakage power for sensors with long off times, but the performance result provide useful CMOS numbers for adder comparisons. We have taken the adder designs for NML and compared them to CMOS based on the results in [44]. The points of comparison are shown in Table 8.2 and graphed in Fig. 8.14.

The data Table 8.2 and Fig. 8.14 report delay and energy per add for parallel NML, serial NML, and serial sub-threshold CMOS. For CMOS, different results are presented for different voltages, while for NML, different results are presented for different currents. The serial adder considers 5, 10, and 25 mT operation and $\mu_r = 1, 3, 12$, while the parallel adder considers 5 mT operation and $\mu_r = 1, 3, 12$.

CMOS adders are more energy efficient than most of the the NML multipliers presented here for $\mu_r = 1$. However, serial multipliers operating at 5 mT field strength is slightly lower energy than the lowest energy CMOS adder shown in Table 8.2. However, for $\mu_r = 3$, NML serial multiplies are more energy efficient for operating fields of 5 and 10 mT. At $\mu_r = 12$, NML multipliers operating at any
**TABLE 8.2**

ENERGY AND DELAY PERFORMANCE FOR NML AND CMOS
32-BIT ADDERS

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Delay per add</th>
<th>Energy per add</th>
</tr>
</thead>
<tbody>
<tr>
<td>22nm CMOS Serial 0.20V</td>
<td>1000.0 ns</td>
<td>50 fJ</td>
</tr>
<tr>
<td>22nm CMOS Serial 0.35V</td>
<td>100.0 ns</td>
<td>60 fJ</td>
</tr>
<tr>
<td>22nm CMOS Serial 0.45V</td>
<td>15.0 ns</td>
<td>100 fJ</td>
</tr>
<tr>
<td>22nm CMOS Serial 0.80V</td>
<td>3.5 ns</td>
<td>500 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 6.80 mA</td>
<td>288.0 ns</td>
<td>1099.0 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 2.30 mA</td>
<td>288.0 ns</td>
<td>126.0 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 0.57 mA</td>
<td>288.0 ns</td>
<td>7.7 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 2.80 mA</td>
<td>288.0 ns</td>
<td>77.0 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 0.93 mA</td>
<td>288.0 ns</td>
<td>20.6 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 0.23 mA</td>
<td>288.0 ns</td>
<td>1.3 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 1.40 mA</td>
<td>288.0 ns</td>
<td>46.6 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 0.47 mA</td>
<td>288.0 ns</td>
<td>5.9 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 0.12 mA</td>
<td>288.0 ns</td>
<td>0.3 fJ</td>
</tr>
<tr>
<td>40x60nm NML Parallel 5mT 1.40 mA</td>
<td>9.3 ns</td>
<td>64.0 fJ</td>
</tr>
<tr>
<td>40x60nm NML Parallel 5mT 0.47 mA</td>
<td>9.3 ns</td>
<td>7.2 fJ</td>
</tr>
<tr>
<td>40x60nm NML Parallel 5mT 0.12 mA</td>
<td>9.3 ns</td>
<td>0.5 fJ</td>
</tr>
</tbody>
</table>
Figure 8.14. Plot of voltage/current vs. energy for NML and CMOS adders

Field strength between 5 and 25 mT require less energy as compared to CMOS. Parallel NML adders are lower energy than CMOS for $\mu_r = 3, 12$.

For multipliers the comparison approach is similar. In [6], the authors have optimized a parallel 16-bit CMOS multiplier design to be low energy by improving the compressor circuits. We compare that multiplier to our own NML-based multiplier designs. The data points are reported in Table 8.3 and shown graphically in Fig. 8.15. Both the table and the figure report delay and energy per multiply for parallel NML, serial NML, and parallel sub-threshold CMOS. For CMOS, different results are presented for different voltages, while for NML, different results
Figure 8.15. Plot of voltage/current vs. energy for NML and CMOS multipliers

are presented for different currents. The serial adder considers 5, 10, and 25 mT operation and $\mu_r = 1, 3, 12$, while the parallel adder considers 5 mT operation and $\mu_r = 1, 3, 12$.

CMOS multiplies are more energy efficient than NML multipliers presented here for $\mu_r = 1$. However, for $\mu_r = 3$, NML serial multiplies are more energy efficient for operating fields of 5 and 10 mT. At $\mu_r = 12$, NML serial multipliers operating at any field strength between 5 and 25 mT require less energy as compared to CMOS.
TABLE 8.3

ENERGY AND DELAY PERFORMANCE FOR NML AND CMOS
16-BIT MULTIPLIERS

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Delay per multiply</th>
<th>Energy per multiply</th>
</tr>
</thead>
<tbody>
<tr>
<td>45nm CMOS Parallel 0.6V</td>
<td>2.40 ns</td>
<td>900 fJ</td>
</tr>
<tr>
<td>45nm CMOS Parallel 0.7V</td>
<td>1.45 ns</td>
<td>1200 fJ</td>
</tr>
<tr>
<td>45nm CMOS Parallel 0.8V</td>
<td>1.10 ns</td>
<td>1700 fJ</td>
</tr>
<tr>
<td>45nm CMOS Parallel 0.9V</td>
<td>0.75 ns</td>
<td>2300 fJ</td>
</tr>
<tr>
<td>45nm CMOS Parallel 1.0V</td>
<td>0.69 ns</td>
<td>3100 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 6.80 mA</td>
<td>288.0 ns</td>
<td>24470 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 2.30 mA</td>
<td>288.0 ns</td>
<td>2799 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 0.57 mA</td>
<td>288.0 ns</td>
<td>172 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 2.80 mA</td>
<td>288.0 ns</td>
<td>4149 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 0.93 mA</td>
<td>288.0 ns</td>
<td>458 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 0.23 mA</td>
<td>288.0 ns</td>
<td>28 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 1.40 mA</td>
<td>288.0 ns</td>
<td>1037 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 0.47 mA</td>
<td>288.0 ns</td>
<td>117 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 0.12 mA</td>
<td>288.0 ns</td>
<td>8 fJ</td>
</tr>
<tr>
<td>40x60nm NML Parallel 5mT 1.40 mA</td>
<td>11.2 ns</td>
<td>23567 fJ</td>
</tr>
<tr>
<td>40x60nm NML Parallel 5mT 0.47 mA</td>
<td>11.2 ns</td>
<td>2656 fJ</td>
</tr>
<tr>
<td>40x60nm NML Parallel 5mT 0.12 mA</td>
<td>11.2 ns</td>
<td>173 fJ</td>
</tr>
</tbody>
</table>
Defect tolerance will be critical in any system with nano-scale feature sizes. This paper examines some fundamental aspects of defect tolerance for a reconfigurable system based on Quantum-dot Cellular Automata (QCA). We analyze a novel, QCA-based, Programmable Logic Array (PLA) structure, develop an implementation independent fault model, and discuss how expected defects and faults might affect yield. Within this context, we introduce techniques for mapping Boolean logic functions to a defective QCA-based PLA. Simulation results show that our new mapping techniques can achieve higher yields than existing techniques.

Categories and Subject Descriptors: B.6.2 [Hardware]: Logic Design—Reliability and Testing, Redundant Design

General Terms: Design, Reliability, Verification

Additional Key Words and Phrases: Nanotechnology, Quantum-dot Cellular Automata, Defects, Faults, Logic Mapping

1. BACKGROUND

2. ACKNOWLEDGMENTS
The authors gratefully acknowledge the support of this work by the Nanoelectronics Research Initiative (NRI) through the Midwest Institute for Nanoelectronics Discovery (MIND). This work was also supported in part by NSF grants CCF-0702705, CCF-06-21990, CCF-0541324, and CCR-02-10153.

8.4.3 N-bit Convolution

One reason for looking at N-bit adders and multipliers for NML was to build up to an NML-based convolution design and show that it can provide performance improvements for applications that use convolution. In this section, we will consider a layout for serial convolution and give energy and delay performance estimates.

By combining a serial multiplier and a serial adder, we can construct a systolic convolution cell. This cell can be combined with many other cells to produce a circuit that can perform convolution using any number of weights. The layout for an example convolution cell with a 4-bit serial multiplier and a 12-bit serial adder is shown in Fig. 8.16.

In terms of performance numbers for convolution designs implemented on other technologies, there are many that consider low-energy design or even report energy consumption in the literature. In addition, many convolution designs in the literature are for 2D convolution arrays often targeted at image processing. For comparison’s sake, it is possible to implement a 2D convolution array with a 1D
convolution array as detailed in [46], however, such an approach greatly increases the number of convolution cells required for the specific implementation. For example, [65] details common parameters for 2D convolution implemented in CMOS or in hybrid nanowire-CMOS technologies. These designs considered convolution kernel sizes from 3x3 up to 15x15 (corresponding to 9 or 225 convolution weights in a 2D array) and for inputs of sizes from 128x128 up to 1024x1024.

Implementing a 3x3 kernel and a 128x128 input vector, for example, using a 1D NML convolution system would require adding many more ‘0’ weights to the 9 weights already in the 2D kernel. In total, there would need to be 259 weights and thus 259 convolution cells to implement the 2D convolution on a 1D array. For a 15x15 kernel and a 1024x1024 input vector, the 1D implementation would need 176,517,300 convolution cells. From an energy perspective for NML-based convolution systems, this would not be an efficient design.

However, we did complete performance calculations for a serial 1D NML convolution system with 9 weights and 16,384 input values. We assumed that each input value was 8-bit, which would require an 8-bit multiplier and a 20-bit adder to create of the convolution cell. Such a serial convolver would be able to complete the convolution computation in 2.95 ms using a total of 28.3 nJ of energy assuming middle of the road serial NML adders and multipliers at 10mT/\(u_r\) = 3, which corresponds to a clock wire current of 0.93mA.

8.4.4 Multi-layer NML

The energy and delay numbers calculated for NML in the technology comparison for adders and multipliers indicates that NML would require more energy compared to CMOS in some cases and less energy compared to CMOS in other
cases. However, in all cases (parallel and serial NML 32-bit adders and 16-bit multipliers) had higher energy requirements per operation compared to CMOS when $\mu_r = 1$. In order to enhance NML circuits so that they can operate with lower energy requirements in more situations, certain improvements need to be made. While lowering the current required to make NML work is constantly being studied, we are also interested in making more compact NML circuit designs.

In an attempt to improve area efficiency, we wanted to find another way to cross NML wires without using the co-planar crossing. This is possible with multi-layer NML circuits. Rather than using NML the crossing circuit with the 'X'-shaped arrangement, we propose instead to have many layers of NML magnets. These layers would be separated by isolation layers, but also connected by stacked nanomagnets that allow signal transmission in the third dimension (z direction). Such wires are different from vertical NML wires. With this approach, NML signals can be passed on horizontal wires, vertical wires, and wires that join the layers. Such an approach was presented in [54]. It is worth mentioning that the current approach to NML clocking would still be able to supply an appropriate magnetic field to the layers in a multi-layer NML circuit. Current carrying wires can provide the necessary magnetic field strength to all magnets that are within 200 nm of the wire surface based on Maxwell simulation reported in [54].

We have considered an improved serial adder design that takes advantage of multi-layer NML to create a more compact layout. The new serial adder covers a smaller area. The layout for this serial adder is given in Fig. 8.17. The more compact serial adder design could be used in a more compact serial multiplier circuit as well.

We used this new design to calculate improved energy requirements for NML
Figure 8.17. Serial adder layout that uses multiple layers of nanomagnets
TABLE 8.4

ENERGY AND DELAY PERFORMANCE FOR MULTI-LAYER NML
AND CMOS 32-BIT ADDERS

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Delay per add</th>
<th>Energy per add</th>
</tr>
</thead>
<tbody>
<tr>
<td>22nm CMOS Serial 0.20V</td>
<td>1000.0 ns</td>
<td>50 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 6.80 mA</td>
<td>288.0 ns</td>
<td>466.1 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 2.30 mA</td>
<td>288.0 ns</td>
<td>53.3 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 0.57 mA</td>
<td>288.0 ns</td>
<td>3.3 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 2.80 mA</td>
<td>288.0 ns</td>
<td>32.7 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 0.93 mA</td>
<td>288.0 ns</td>
<td>8.7 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 0.23 mA</td>
<td>288.0 ns</td>
<td>0.5 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 1.40 mA</td>
<td>288.0 ns</td>
<td>19.8 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 0.47 mA</td>
<td>288.0 ns</td>
<td>2.5 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 0.12 mA</td>
<td>288.0 ns</td>
<td>0.1 fJ</td>
</tr>
</tbody>
</table>

serial adders and multipliers. These new performance numbers also reflect the use of high permeability materials, just as the performance estimates in Sec. 8.4.2 did. The new adder performance has been included in Table 8.4. The new multiplier performance can be seen in Table 8.5. These tables only show performance numbers for a serial adders and multipliers using multi-layer NML and the lowest energy CMOS adder.

With the improvements seen in area efficiency due to using multiple layers of
TABLE 8.5

ENERGY AND DELAY PERFORMANCE FOR MULTI-LAYER NML AND CMOS 16-BIT MULTIPLIERS

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Delay per multiply</th>
<th>Energy per multiply</th>
</tr>
</thead>
<tbody>
<tr>
<td>45nm CMOS Parallel 0.6V</td>
<td>2.40 ns</td>
<td>900 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 6.80 mA</td>
<td>288.0 ns</td>
<td>17945 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 2.30 mA</td>
<td>288.0 ns</td>
<td>2053 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 25mT 0.57 mA</td>
<td>288.0 ns</td>
<td>126 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 2.80 mA</td>
<td>288.0 ns</td>
<td>3043 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 0.93 mA</td>
<td>288.0 ns</td>
<td>336 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 10mT 0.23 mA</td>
<td>288.0 ns</td>
<td>21 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 1.40 mA</td>
<td>288.0 ns</td>
<td>761 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 0.47 mA</td>
<td>288.0 ns</td>
<td>86 fJ</td>
</tr>
<tr>
<td>40x60nm NML Serial 5mT 0.12 mA</td>
<td>288.0 ns</td>
<td>6 fJ</td>
</tr>
</tbody>
</table>
nanomagnets, NML serial adders could out-perform CMOS in terms of energy for all instances, except for $\mu_r = 1, 3$ with 25 mT operation. For multipliers, NML is better than CMOS for operation at 5 mT, for operation at 10mT at $\mu_r = 3, 12$, and for operation at 25mT at $\mu_r = 12$. If multi-layer NML is realized and high permeability materials are shown to work in NML systems, then NML circuits of many types, especially for high-throughput applications such as convolution, mathematical transforms, and pattern matching, would be very promising for low energy operation. In the best cases, the energy improvement could be over 100 times better.
CHAPTER 9:

CONCLUSIONS

In this dissertation, circuit designs for two major architectural categories have been proposed to show the potential of NML technology in the areas of defect tolerance and low-energy computation. The NML PLA design and supporting methodology (such as fault model and logic mapping) presented here provide a powerful guidebook for making reprogrammable PLAs for NML. The same goes for the designs and system-organization ideas presented for NML high-throughput systolic architectures. While there are challenges for NML to overcome, this work provides a solid basis for showing how NML can provide performance wins against other computational nanotechnologies for certain applications.

Given the many desirable properties of NML, we have proposed an NML PLA design that aims at achieving area and energy efficiency. All the basic components of our proposed design have been either validated experimentally or via simulation. Starting from a simple fault model, we have built up a body of work that has considered device-level defects and faults, and extrapolated those into a fault model for QCA PLA cells, including NML PLA cells. Using that model, we have created an improved mapping method that can better avoid defects, and continued that work in an attempt to find more ways to increase defect avoidance efficiency for NML PLAs. This includes careful control of the magnetic clocking fields, systems of small PLAs, and redundant PLAs. Through our mapping tests
and performance calculations for defect-free circuits, we have seen that the NML PLA competes very well – in terms of area – with other reprogrammable nanotechnologies with similar feature sizes. However, NML does not compete well with other computational technologies when it comes to delay. While there are not many comparison points available in the literature, we do believe that NML PLAs should provide a low-energy solution for reprogrammable logic.

From the standpoint of the NML PLA’s defect tolerance, our benchmark mapping suggests that redundancy is most beneficial when the defect rate for the NML PLA cell (see EDR discussion in Sec. 5.3) is in the range of $10^{-3}$ to $10^{-4}$. If the defect rate is lower (i.e. $< 10^{-5}$), then the need for a reprogrammable architecture is diminished. If the defect rate is too high (i.e. $> 10^{-3}$), then redundancy added into the NML PLA is not enough to compensate for the defects (except for the smallest of benchmarks).

If defect rates are acceptably low, a reprogrammable architecture may not be the best system-level architecture for NML. After looking at performance comparisons for NML PLAs, we became very interested to see what sorts of architectures and applications would allow NML technology to be utilized to its fullest. We decided that systolic NML architectures could provide the best overall performance wins as compared to state of the art CMOS. NML circuits have shown great promise in areas that are important to systolic architectures (low power, pipelining, high throughput, and small area). In this vein, we have investigated compute-bound applications that map to systolic architectures. We explored two different systolic designs for NML, each one utilizing a different NML clocking scheme. The uni-directional design using a 3-phase clocking scheme suffered from a performance penalty in area and energy due to the long wires leading into the
systolic array. The bi-directional systolic design using a 2-phase clocking scheme proved to have better performance, in spite of the higher clock field requirements needed for two-way dataflow, which adversely affected clocking energy. In addition, the bi-directional design has a lot of room for performance improvement, which can be achieved through reduction of clock wire current, and reduction in the switching time for an ensemble of magnets in a circuit. Performance estimates for simplified 1-bit convolution indicate that NML has the potential to outperform low-power CMOS in terms of energy and energy-delay product for high-throughput applications.

We also considered larger systolic designs, starting with NML adders and multipliers. While the 2-phase clocking approach with bi-directional dataflow was initially used to create a more efficient method for bringing inputs to the simplified convolution circuit, we also used the 2-phase clocking approach to create serial NML designs with the aid of a feedback loop. We validated an NML feedback loop circuit component design through simulation. The NML serial designs avoided the interconnect overheads of parallel designs, however, NML serial designs suffered from high currents on the NML clocking wires. For both the parallel and serial NML designs, we explored ways to help NML perform much better than CMOS in terms of energy consumption by creating compact designs with multi-layer NML and using high-permeability materials to reduce the current required on the NML clocking wires. These solutions are currently being considered for new NML designs and approaches. With the combination of the NML designs presented here and the efficiency improvements, NML systolic circuits could be a factor of 10 to 100x lower energy than state-of-the-art, sub-threshold CMOS for high-throughput, pipelined, systolic applications.
9.1 Accomplishments

The following is a list of the accomplishments achieved by the work given in this dissertation. Each of these is a new result that provides a significant contribution to the field.

1. A previously proposed QCA PLA architecture has been modified to create a working NML-based PLA, including demonstration of all required circuit components and design of all necessary structures.

2. A unique study of NML defects and faults has been completed which shows how misshapen magnets can lead to stuck-at faults in NML circuits.

3. Studies of NML stuck-at faults were coupled with QCA PLA designs to create a fault model which formalized the faulty behaviors possible in NML PLA cells as a result of NML defects.

4. A graph-based PLA mapping method was improved, through the use of edge attributes, to increase the likelihood of finding successful mappings in the case of defective PLAs.

5. The design for a simplified 1-bit NML convoler has been given which could be converted into an efficient systolic array for other low-power, high-throughput computational applications.

6. An NML feedback loop has been validated through physical-level simulation, which allows for compact NML serial circuits to be created.

7. Parallel and serial N-bit NML adder and multiplier designs (including designs that use multi-layer NML) have been presented as a starting point for creating more complex N-bit NML systolic designs.
9.2 Future Work

All performance metrics (area, delay, and energy) have room for improvement through nanomagnet scaling. We assumed 40x60nm magnets, however, with improved lithography, it could be possible to create magnets with smaller feature sizes down to (or even below) the superparamagnetic limit, which is in the range of about 20nm. Also, it could be possible with other advances in the technology to find additional ways for producing the necessary clocking field with less current, leading to lower energy operation.

For NML PLAs, there are some interesting threads of research that could be pursued as future work. First, some sort of hybrid NML-CMOS PLA system could be investigated to improve area, delay, and energy performance. Such a hybrid system could also allow for such PLAs to be able to work in the presence of higher defect rates compared to NML-only PLAs. Second, NML PLAs could take advantage of multi-layer NML to make a more compact NML PLA cell. These is one crossing and some long wires in the PLA cell design that could be eliminated if multi-level NML could be utilized. Third, the tolerated defect rates reported in this dissertation were based on an EDR that corresponded to the defect rate of the PLA cell. It would be worth investigating how this EDR corresponds to an NML device defect rate and how the defect tolerance from architectural redundancy compares to other forms of defect tolerance studied for NML and QCA in general. Finally, the programming procedure for QCA PLAs could be combined with the defect detection procedure discussed in this dissertation to create a combined procedure that would give concise test vectors for evaluating the location of defects in an efficient manner while also giving the necessary information for quick programming of the PLA structure in the presence of defects.
Relating to the systolic work presented in this dissertation, there is future work that would be useful to improve knowledge about high-throughput, low-power NML systems. First, there are many other NML designs (besides the full adder) which could take advantage of multi-layer NML to improve the compactness of those designs. Designs which have a large amount of interconnect and a large number of co-planar crossings would benefit the most. This would be especially true of the NML parallel adder and multiplier. Second, there were many ways in which the NML adder and multiplier designs had to satisfy timing requirement for both the NML clocking scheme and for the architectural dataflow. A more extensive study on how satisfying both timing requirements could be applied to more general NML circuits should be investigated. Finally, there are many high-throughput applications that could be studied for implementation with NML circuits. Looking at pattern matching, cryptography, signal processing, and other applications which use mathematical transforms would be very much worth investigating to see if certain application spaces are well suited to NML’s best characteristics.
BIBLIOGRAPHY


