Defects and Faults in QCA-Based PLAs

MICHAEL CROCKER, X. SHARON HU, and MICHAEL NIEMIER
University of Notre Dame, Notre Dame, IN

Defect tolerance will be critical in any system with nanoscale feature sizes. This article 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 [**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

ACM Reference Format:

Crocker, M., Hu, X. S., and Niemier, M. 2009. Defects and faults in QCA-based PLAs. ACM J. Emerg. Technol. Comput. Syst. 5, 2, Article 8 (July 2009), 27 pages.
DOI = 10.1145/1543438.1543441 http://doi.acm.org/10.1145/1543438.1543441

1. INTRODUCTION

At present, there are a number of research efforts that have focused on different devices that might either replace or augment CMOS technology such that the performance scaling trends that we have seen for the last 30 years—and expect to see for the next 10–15 years—might continue beyond the year 2020. The work presented here looks at the Quantum-dot Cellular Automata (QCA) device architecture—and more specifically a reconfigurable Programmable Logic Array (PLA) realized with QCA devices. We consider how faults will affect the

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, CCF06-21990, CCF-0541324, and CCR-02-10153.

Authors' address: Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46656; email: {mcrocker,shu,mniemier}@nd.edu.

DOI 10.1145/1543438.1543441 http://doi.acm.org/10.1145/1543438.1543441

functionality of a reconfigurable PLA and propose mapping techniques to improve overall yield.

QCA accomplishes logical operations and moves data via nearest-neighbor interactions rather than with electric current flow. A given implementation could potentially lead to fast and/or low power circuits with nanometer feature sizes. Different implementations are being researched and include devices based on a metal-dot structure [Amlani et al. 1999], individual molecules [Qi et al. 2003], and semiconductor technology [Mitic et al. 2006]. A magnetic implementation of QCA is also possible. For nanomagnet-based QCA (MQCA), wires, gates, and inverters, operating at room temperature, have all been experimentally realized and verified. It has been estimated that if 10^{10} nanomagnets are adiabatically switched 10^8 times each second, they should only dissipate approximately 0.1W of power [Imre et al. 2006]. Nanomagnets with feature sizes above the superparamagnetic limit are also nonvolatile.

Of course, it is also well recognized that any circuit with nanometer feature sizes will almost certainly be more defective than what we have come to expect from previous generations of CMOS-based circuits. Largely for this reason, many research groups studying emerging technologies have proposed specific reconfigurable logic structures and investigated means to map Boolean functions to redundant, reconfigurable architectures [DeHon and Wilson 2004; Snider et al. 2005; Strukov and Likharev 2005]. A PLA structure for QCA that uses AND-OR logic was introduced in Hu et al. [2006], while QCA-specific defect and fault modeling was studied in Crocker et al. [2007].

The QCA PLA is a promising, regular, and reprogrammable structure that can utilize redundancy to facilitate usable circuits with nanometer feature sizes. The design is compact and easily extensible. Here, we present a fault model that considers the unique effects of QCA defects for many implementations. We analyze these faults and their impacts on the functionality of PLA cells. We then devise a graph-based approach to mapping Boolean logic functions onto random defective PLA structures. Then we perform many mapping tests to reveal the yield trends that a QCA PLA might exhibit.

The article is organized as follows. We begin in Section 2 by discussing the basics of the QCA device architecture and the QCA-based PLA design. A fault model for QCA-based PLAs is presented in Section 3. We review mapping techniques for nanowire crossbars in Section 4 as we leverage this work in our proposed mapping methodology. In Section 5 we present a new methodology for mapping logic functions to the QCA-based PLA design. We present our results in Section 6 and discuss future work in Section 7.

2. BACKGROUND

2.1 QCA Basics

The initial description of a QCA device called for encoding binary numbers into devices that have a bistable charge configuration. A QCA device would consist of 2 or 4 “charge containers” (i.e., quantum dots) and 1 or 2 excess charges respectively. One configuration of charge represents a binary 1 and the other

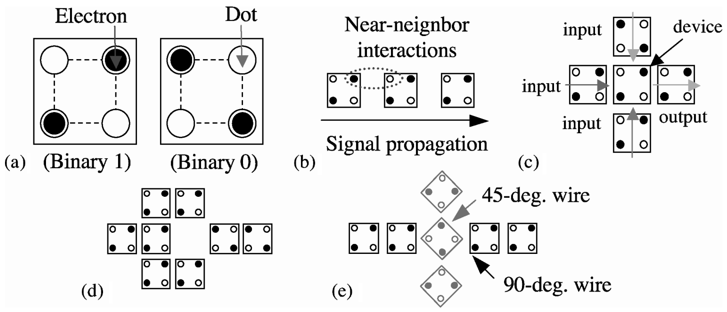

Fig. 1. Schematics of the fundamental structures needed to build QCA circuits: (a) basic device, (b) wire, (c) logic gate, (d) inverter, and (e) crossover.

a binary 0 (Figure 1(a)) [Lent and Tougaw 1997]. 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.

Figures 1b–e illustrate the building blocks that would be used to construct QCA circuits [Tougaw and Lent 1994]. A QCA wire (Figure 1(b)) 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 (Figure 1(c)) implements the logic function $AB + BC + AC$. The output device assumes the polarization of the majority of the 3 input devices [Lent and Tougaw 1997]. 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 (Figure 1(d)). QCA wires with different orientations (Figure 1(e)) can theoretically cross in the plane without destroying the binary value on either wire.

Regardless of implementation, a circuit or system made from QCA devices will require a clock structure that will take the form of lithographically defined, conducting metal wires [Hennessy and Lent 2001]. The clock is required to maintain state for electrostatic QCA, and to remove state for magnetic QCA. For detailed descriptions on the clocking structure, readers can refer to Crocker et al. [2008].

2.2 Reprogrammable PLA Cells and Array Structure

Reconfigurable structures are usually created to be flexible in the logic that can be implemented and to provide redundancy and fault tolerance. Because of the high defect and fault rates expected at the nanoscale, many nanotechnologies have considered reprogrammable structures like PLAs or FPGAs. This is true for QCA circuits as well. A QCA PLA design was first detailed in Hu et al. [2006] for the purpose of improving fault tolerance, especially since it could be used in conjunction with device-level fault tolerance strategies that had already been proposed for QCA circuits. In addition, the regular, repeated design of a PLA array was much preferable to custom logic in cases where a self-assembly process was involved in fabrication, such as for molecular QCA circuits. While there are other implementations of QCA that use lithography, the fault tolerance and flexibility that comes from the reprogrammable design is still important.

Traditional MOSFET PLAs have been made from NAND or NOR logic. While such a design is possible using QCA devices, other designs were explored. One choice would be to create a PLA that utilizes majority logic, since the basic gate in QCA is a majority gate. Other work has been done on majority-majority PLAs [Manem et al. 2008]. Our initial investigation into a majority-based QCA PLA found that such a design would be adversely affected by interconnect. For QCA-based circuits, we try to avoid intersections as (a) all nearest neighbor interactions are in-plane, and (b) wire crossings are difficult to fabricate in some implementations of QCA. While wire crossings are unavoidable in a PLA array, we wanted to reduce them as much as possible in the design. Unfortunately, a PLA design that uses majority logic would require far too many wire crossings to be efficient. However, QCA majority gates can be programmed to AND and OR gates without requiring any extra signal routing. Implementing NAND and NOR functionality would require more QCA inverters, therefore the QCA PLA discussed in Hu et al. [2006] uses AND and OR logic.

A schematic of one PLA cell for the AND plane is shown in Figure 2(a). For this article 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 Figure 2(b):

if S=0, *(Implicant Out)* = *(Literal In)* • *(Implicant In)*
if S=1, *(Implicant Out)* = *(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 (Figure 2(c)). The select bit should be set to 1 for *logic* mode and 0 for *wire* mode:

if S=1, *(Function Out)* = *(Implicant In)* + *(Function In)*
if S=0, *(Function Out)* = *(Function In)*

By leveraging the structures just discussed, it is relatively easy to construct the logic required for a PLA of arbitrary size. Figure 3 shows how the "cell" construct can be used to make entire AND and OR planes. Select bits are set to perform two generic Boolean functions with their binary values in the inset triangles.

2.3 Faults and the QCA PLA Design

Crocker et al. [2007] explored the impacts of faults on system-level functionality by employing the well known stuck-at fault model. It showed how stuck-at faults would logically affect different parts of the PLA. Besides the faults similar to conventional PLA structures such as broken literal wires, QCA PLAs may have some unique fault types. For example, if a stuck-at-1 fault occurs in the Select Wire region of a PLA cell (see Figures 2(a) and (d)) that needs to be programmed

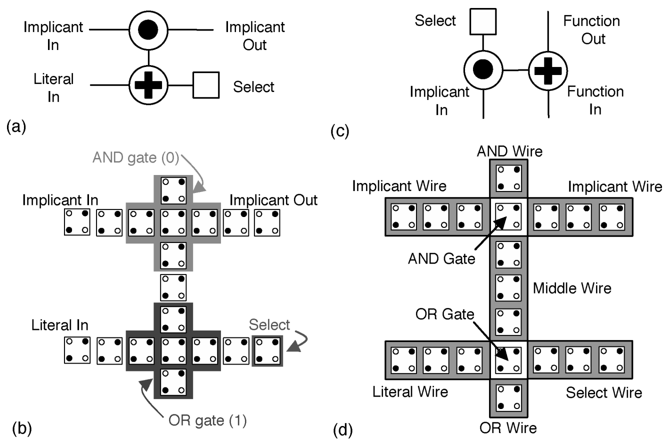

Fig. 2. (a) QCA AND Plane Cell Schematic, (b) AND Plane Cell Layout, (c) QCA OR Plane Cell Schematic, (d) AND Plane Cell Regions.

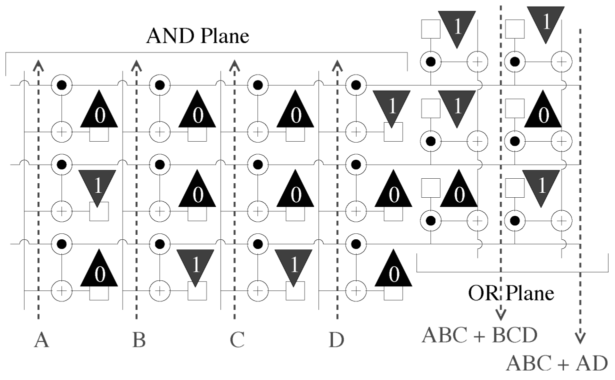

Fig. 3. A PLA that implements the Boolean functions ABC+BCD and ABC+AD.

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. The logic implemented in the PLA can take on many patterns, allowing for the use of faulty PLA cells. Similarly, if a stuck-at-0 fault occurs on the Literal Wire, the PLA cell can be programmed to wire mode, and it would still be useful for generating an implicant.

2.4 Fault Detection

In order to utilize the fault tolerance potential in QCA-based PLAs, faults must first be detected so that faulty resources can be avoided and operational resources can be utilized. A basic fault detection scheme was proposed in Hu et al. [2006]. The basic idea of fault detection given 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 this high-level test is complete, the individual PLA cells can

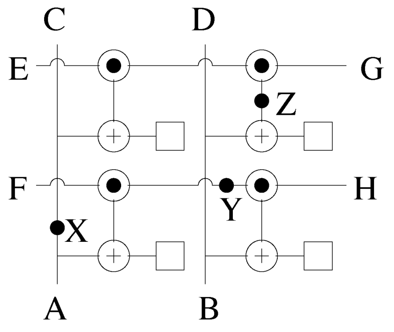

Fig. 4. A 2x2 section of the AND plane of a PLA showing three faults.

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 a QCA PLA.

In Figure 4, a small section of the AND plane of a QCA 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 vertical wires in the AND plane in parallel. To test the row wires, 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 column can be tested at the same time. All PLA cells should be programmed to wire mode, except the cells in a single column, which should be programmed to logic mode. To find a fault at location Z, the second column of PLA cell 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 wire G would be incorrect for one or more of the inputs given at B.

While this is a working fault detection scheme, improvements could be made to make it more efficient. In addition, the analysis done here does not fully explain how finding faults in one location will affect fault detection in other locations. However, a more detailed discussion of detection schemes is beyond the scope of this article. We are considering this in our future work.

2.5 Related Work

We are not aware of any defect and yield studies specifically for QCA-based PLAs. However, a PLA fault model has been adopted in the context of nanowire crossbars [DeHon and Wilson 2004]. For MOSFET PLAs, the crosspoint fault model was developed as a more accurate model for PLA faults [Somenzi and Gai 1986]. 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. The crosspoint model is not sufficient for a QCA PLA, as there are faults that are not present in a MOSFET PLA.

3. FAULT MODELING

In this section, we discuss defects and the consequent faults. Here, a "defect" is defined as a deviation of a QCA device from the ideal device in terms of shape, orientation, location, etc. A defective QCA device does not necessarily exhibit the wrong logical behavior. Defect tolerance is the ability for a device to operate properly even though it is not perfect. A "fault" occurs when a defect is severe enough to cause incorrect logical behavior in a QCA device. We first discuss possible defects and faults for different QCA implementations, then examine the effects that those defects and faults can have on the QCA PLA cell design and the PLA structure as a whole

3.1 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. QCA-based circuits can experience more types of faults than traditional MOSFET devices. Besides the stuck-at fault (which is equivalent to the MOSFET stuck-at fault), QCA devices have two other types of faults. The first is the nondeterministic 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 of QCA. This fault has some similarities to floating node faults in CMOS, because the logical value of the fault will change from cycle to cycle even with the same imputs. However, the QCA ND fault is unique because the unpredictable value created by the fault is determined by factors such as operating temperature or small flcuations in the surrounding electromatnetic fields. Using molecular QCA as an example, 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 [Niemier et al. 2006]. 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 easily occur in electrostatic implementations of QCA, have have been seen on rare occasions in magnetic implementations. 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 (Figure 1(d)), where neighboring QCA

Table I. Connecting Defects to Faults

Molecular		Magnetic	
Defect	Fault	Defect	Fault
stray charge shifts/rotations missing	stuck at inversion ND	misshape shifts	stuck at/ND inversion/stuck at/ND

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 device faults (stuck-at, ND, and inversion) discussed above. Proper association of defects to faults helps in studying the fault probabilities and yield. By examining experimental data and performing simulations at the physical level [Hu et al. 2006; Niemier et al. 2006; Donahue and Porter], we have matched defects to faults for two QCA implementations. In Table I, we summarize the faults that are induced by five typical defect types, that is, shifts, rotations, missing, stray charges and misshapenness. While variations can occur, Table I represents the "common case." For magnetic implementations of QCA, there is more ambiguity about how defects connect to faults.

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 Figure 5(a)). (There are no magnetic monopoles.) Device shifts (β) and rotations (γ) occur when the position or the orientation of the QCA device deviates from the ideal (Figure 5(b) and Figure 5(c) respectively). Each of these defects can cause QCA devices to interact with one another in ways they normally would not. For example, variation in the spacing between magnets—for example, a shift—may cause the magnets in one part of a wire to switch before they otherwise should. By switching out of order, the circuit would exhibit a stuck-at behavior. Similarly, a shifted QCA molecule may cause a signal to be inverted as discussed above.

Missing devices (δ), shown in Figure 5(d), cause nearest neighbor interactions to be weaker, as interactions are distance dependent. Such defects could be more frequent in the molecular implementation because it would have to rely on self-assembly as a fabrication mechanism. Finally, if a device deviates from its ideal shape (ϵ), as shown in Figure 5(e), it may interact with its neighbors incorrectly. Devices realized by lithography (e.g., nanomagnets) are especially susceptible to this type of defect. Devices with the wrong shape will lead to stuck-at faults.

3.2 PLA Cell Faults

We now look at how the defects and faults we have discussed affect the logical behavior of the PLA design in Hu et al. [2006]. We methodically examined the 8 general regions of the QCA PLA cell (Figure 2(d)) for each of the possible device fault types. As mentioned before, defective QCA devices can exhibit stuck-at-1, stuck-at-0, ND, or inversion faults. Depending on what region these faults occur

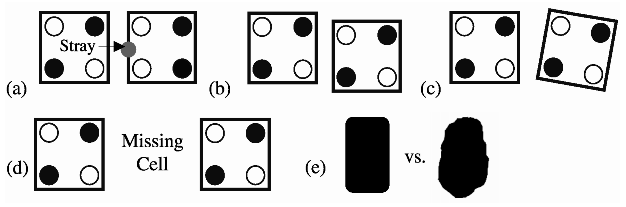

Fig. 5. Possible defects: (a) stray charge, (b) shifted device, (c) rotated device, (d) missing device, (e) misshapen device.

in, the logical behavior of the PLA cell can be aversely affected. Also, depending on the severity of the fault in the PLA cell, neighboring cells could be affected as well. In the worst case, one faulty QCA PLA cell could cause an entire row or column to fail. However, such a worst case is too pessimistic to be realistic.

Instead, we consider a more realistic analysis of fault probabilities 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 Figure 2(d)).

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 the 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. The logic implemented in the PLA can take on many patterns, allowing for the use of faulty PLA cells. For example, if a stuck-at-0 fault occurs on the Literal Wire, the PLA cell can be programmed to wire mode, and it would still be useful for generating an implicant.

Therefore, in a more realistic case, combinations of device faults and fault locations can be discounted, as they do not lead to incorrect operations. We have determined which of the PLA cell regions are susceptible to each fault and which are not. A very similar analysis was done in Crocker et al. [2007], which tried to estimate the number of devices in a QCA PLA cell that could be affected by faults. This analysis was part of an attempt to perform a preliminary yield analysis. For our purposes in this paper, we are more interested in the connection between device faults and PLA cell faults. This information is shown in Table II. The 8 regions are listed in the first column. The 4 fault types are given in columns 2 to 5. Each region susceptible to the fault type is marked by putting an F into the table.

As an example, consider the ND fault (second 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.

Table II. Effects on QCA PLA Cells Based on the Location and Type of QCA Device Faults

QCA Device Location	AND Plane Row Failures			
	ND	invert	SA1	SA0
AND wire	F			
OR wire	F			
literal wire		F		
implicant wire	F	F	F	F
select wire	F			
middle wire	F	F		F
AND gate	F		F	F
OR gate	F			F

3.3 Refined PLA Fault Model

The information given in Table II 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 will be discussed later in the section on logic mapping, 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 called either stuck-in-wire-mode or stuck-in-logic-mode, depending on the faulty behavior of the PLA cell.

To continue the work in the previous section, and in order to provide a better approximation of the effects of each defect/fault on the operation of an entire PLA, we reanalyzed the AND plane cell that had been divided into eight different operational regions (See Figure 2(d)). We focused on the operation of the AND plane PLA cell for each region and device fault type. Table II 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 now considered which of four possible states the PLA cell would be in if the device fault occurred. With any device fault, the PLA cell would either be fully faulty, stuck-in-wire-mode, stuck-in-logic-mode, or fully operational. The new results are summarized in Table III.

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 works 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.

3.4 Special Case Faults

Further study has led to an additional consideration in the context of device faults and 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

Table III. 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'

QCA Device Location	AND Plane Row Failures			
	ND	invert	SA1	SA0
AND wire	F	.	.	.
OR wire	F	.	.	.
literal wire	W	F	W	W
implicant wire	F	F	F	F
select wire	F	.	W	L
middle wire	F	F	W	F
AND gate	F	.	F	F
OR gate	F	.	.	F

faulty 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, depending on the type and location of the device faults. These changes are shown in Table IV. 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 III, 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 IV, 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.

Table IV. 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'

QCA Device Location	AND Plane Row Failures			
	ND	invert	SA1	SA0
AND wire	F	.	.	.
OR wire	F	.	.	.
literal wire	W	F	W	W
implicant wire	F	P	P	F
select wire	F	.	W	L
middle wire	F	F	W	F
AND gate	F	.	P	F
OR gate	F	.	.	F

The inclusion of this PLA cell fault type, gives us a more complete connection between device faults 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. Section 5.2 shows how this new PLA cell fault type will affect the representation of PLA structures used for mapping logical functions.

4. EXISTING MAPPING TECHNIQUES

In this section, we first briefly review existing PLA fault tolerance 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 terms of redundancy, 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 programmed rows that were defective [Wey 1988]. There were a few research efforts that did look at fault detection and made strides towards full mapping. In Demjanenko and Upadhyaya [1990], 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 reprogrammable PLAs is increasing again in the area of emerging nanotechnologies.

We are aware of one fault study specifically for QCA PLAs [Crocker et al. 2007]. This work provides a fault model, and gives a brief yield study. However, the yield analysis does not consider the mapping of benchmarks. Instead, a simplification is made in order to get first-order yield numbers for PLAs of a general size. A more complete nanoscale PLA fault model has been adopted in the context of nanowire crossbars [DeHon and Wilson 2004]. While not specific

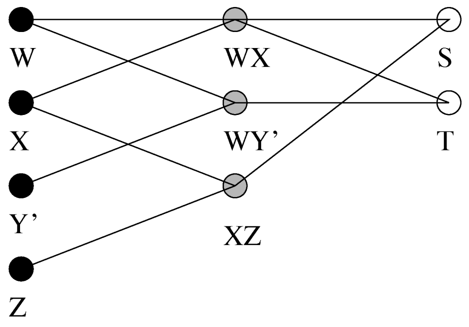

Fig. 6. Logic Graph Representation. The logical functions $S = WX + XZ$ and $T = WX + WY'$ are represented in graph form.

to QCA, we can leverage PLA-based mapping techniques developed for other emerging technologies.

In Snider et al. [2005], a resource allocation method is proposed for nanowire crossbar PLAs. This nanowire crossbar architecture is similar to the one discussed in Haselman and Hauck [2008]. At a high level, this approach consists 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 allows for rearrangement flexibility of inputs, outputs, and implicant terms, resulting in more possible mappings and higher yields when compared to the mapping algorithm given in Demjanenko and Upadhyaya [1990]. However, more rearrangement flexibility leads to a more computationally intensive mapping algorithm. While other research projects attempt to reduce computation time through different heuristics in the mapping algorithm [Rao et al. 2006], our goal is to improve the model to identify more possible mappings. We review the work from Snider et al. [2005] 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 Figure 6. 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

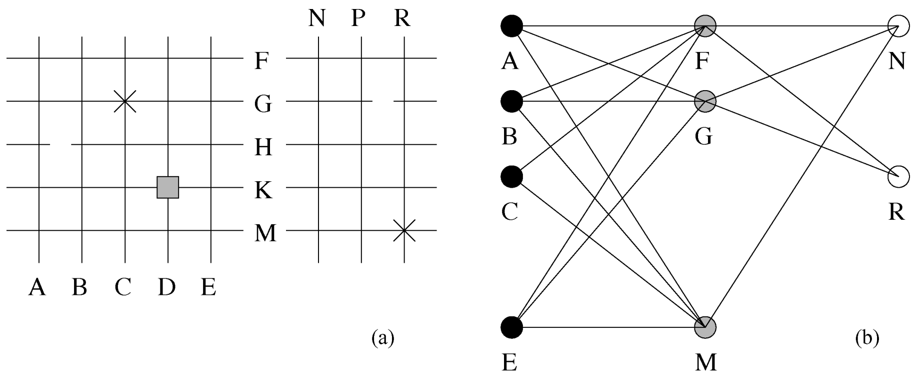

Fig. 7. (a) A defective and redundant nanowire crossbar structure. The AND plane is 5×5 cells, and the OR plane is 3×5 cells. There are 5 faults. (b) Crossbar Structure Graph Representation. Vertices and edges have been removed based on the location and types of PLA cell faults.

defects in the model. To create a CSG from a fabricated PLA structure, information must first be obtained from a fault-detection procedure, which is future work. 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 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, Figure 7(a) 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 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 Snider et al. [2005], there is mention of these defects in Haselman and Hauck [2008] 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. Figure 7(b) shows the CSG for the PLA structure in Figure 7(a).

Once the graph for the defective nanowire crossbar structure has been generated, a mapping can be achieved by finding a monomorphism between the graphs [Snider et al. 2005]. 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. Snider et al. [2005], used an implementation of monomorphism matching discussed in Cordella et al. [2004].

5. MAPPING TO FAULTY QCA PLAS

In Section 4, 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 faulty crosspoints as compared to the graph matching approach proposed for nanowire crossbar PLAs [Snider et al. 2005]. Much higher yields can be achieved by this new approach as will be shown in Section 6.

5.1 Yield-Increasing Mapping for QCA PLAs

Although the CBPM method discussed in Section 4 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 defect tolerance and 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.

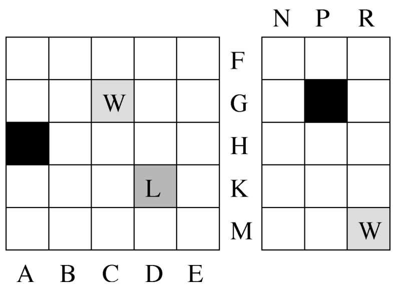

Fig. 8. A defective and redundant QCA PLA structure. The AND plane is 5x5 cells, and the OR plane is 3x5 cells. There are 5 faulty cells.

Figure 9(a) is the QSG for the PLA structure given in Figure 8. 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 while the one stuck at logic PLA cell results in the one dashed edge. Compared with the CSG in Figure 7(b), 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 Figure 6 is shown in Figure 9(b). 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 now be solved by employing a monomorphism matching algorithm for attributed relational graphs (ARG). In particular, the edges 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 [Cordella et al. 2004; El-Sonbaty and Ismail 1998] which can be readily applied to solve our PLA mapping problem. Elaborating on the inner working of such algorithms is beyond the scope of this article.

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

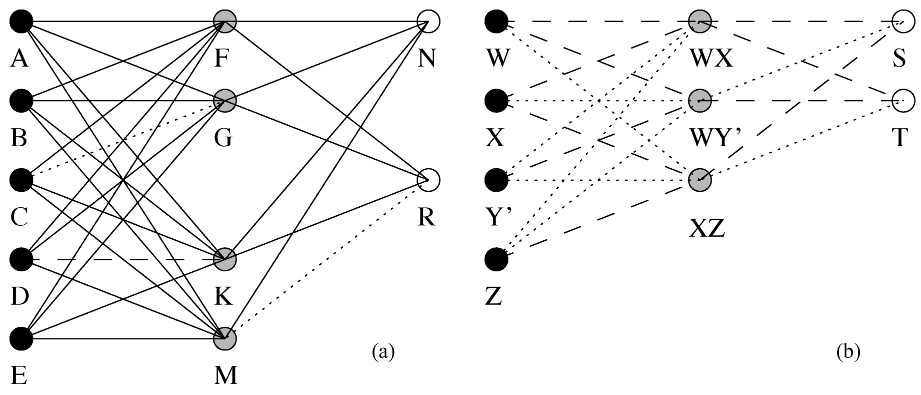

Fig. 9. (a) The QSG corresponding to the QCA PLA structure in Figure 8. The solid edges represent PLA cells that can be programmed into either logic or wire mode (Type 1), while the dashed edges represent the cells that are stuck in logic mode (Type 2), and dotted edges represent cells that are stuck in wire mode (Type 3). (b) The CLG corresponding to the LG in Figure 6. The dashed edges show the logic to be mapped (Type 2), while the gray edges show which connections need to pass signals (Type 3).

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 Special Case Graph Representation

As discussed in Section 3.4, 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 Section 5.1, 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 Figure 8 is repeated in Figure 10(a). Two of the fully faulty PLA cells from the original example are replaced with partial line faults to show the difference.

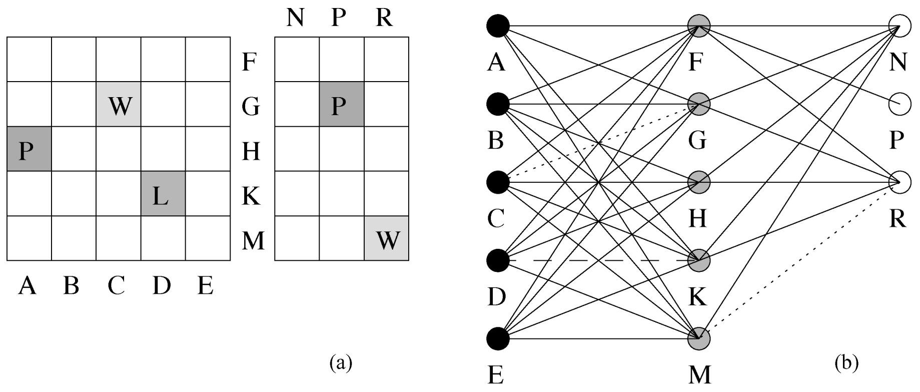

Fig. 10. (a) A defective and redundant QCA PLA structure that includes cells that are faulty but only affect part of a row or column. (b) The corresponding QSG. Vertices and edges have been removed based on the location and types of PLA cell faults, including partial line faults.

A new QSG is created with these new rules in mind (see Figure 10(b)). The resulting graph has more nodes and edges thanks to the extra consideration given to PLA cells with partial line faults.

Looking at Figure 10(b), there are a few things that stand out when compared to Figure 8(b). 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 Figure 10(b) 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.

6. EVALUATION

In the previous sections we discussed the fault model for the QCA PLA and 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 discussed in this paper, that is, 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 [Cordella et al. 2004]. The benchmarks that we consider range from small examples to larger functions from the Toronto 20 benchmark set.

The results presented in the first section focus on two important metrics in comparing the two mapping methods: the CBPM method and the QCAPM method. We examine the potential yield for QCA-based PLAs and runtime efficiency for both methods. Three benchmarks of different sizes were used to compare the two mapping methods. These benchmarks were mapped to many randomly generated simple PLA structures.

Table V. PLA Yield for Mapping the Two Functions from Figure 6 to an (8,6,4) PLA

Fault Rate	1%	5%	10%
CBPM	100%	92.3%	63.0%
QCAPM	100%	99.4%	94.5%

The experiments presented in the second section consider defect distributions beginning at the device level instead of at the PLA cell level and examine yield improvements by using the more detailed graph model. 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 Section 3. This was done by assuming an even distribution of device-level faults, and then applying the correlations between device and cell faults from Table IV to determine the distribution of cell faults. The tests will be use to determine how much yield improvement can be achieved by including the partial line faults. 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.

6.1 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 Figure 6. 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 random 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 fault rate, each PLA cell was given one of four possible conditions (either fully faulty, stuck-in-wire-mode, stuck-in-logic-mode, or not faulty). 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 1%, 5%, and 10% were used during the generation of the faulty PLAs. All fault types had equal distributions in the generation of these structures. Results are presented in Table V.

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 fault rate was set 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

Table VI. PLA Yield for Mapping the 3-bit Adder Functions to a (20,40,10) Faulty PLA Structure. A Cutoff of 5 Seconds was Used.

Fault Rate	1%	2%	3%	4%	5%	6%	7%
CBPM	74.5%	22.0%	3.0%	0%	0%	0%	0%
QCAPM	100%	94.0%	69.5%	37.5%	9.5%	3.5%	0%

matching. At a 5% fault rate, both methods exhibited at least a few cases out of 1,000 tests without a successful matching, and even more so at a 10% fault rate.

The results show that the QCAPM method is more effective in providing higher yields than the CBPM method. Even at 10% faults, 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 crosspoint that are stuck in wire and logic mode more effectively than the other approaches. The two crossbar-based methods see a drop in yield more quickly as the fault rate increases.

In general, real-world Boolean functions are much larger than the example considered previously. Since the exact graph monomorphism algorithm used in our tool suite is exponential in time complexity [Cordella et al. 2004], 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 [Snider and Williams 2007; Rao et al. 2006; Hogg and Snider 2006]. Stopping the search early can 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 12 inputs 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 with random faults. We used a cutoff time of 5 seconds. The yield results are illustrated in Table VI. 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 fault rate 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% faults, while the QCAPM method dropped off most between 2% and 5%. In addition, the yield for the QCAPM method did not decrease at the same rate as the CBPM method. It took a fault rate *change* (not fault rate) of 3% to reduce the QCAPM method from a yield of 95% down to around 10%. On the other hand, the CBPM method went from a 75% yield to around a 5% yield with only a 2% increase in fault rate.

Table VII. Runtime Comparison of the Two Mapping Methods for Successful Matchings of the 3-bit Adder

Fault Rate	1%	2%	3%
CBPM	94 μsec	85 μsec	89 μsec
QCAPM	142 μsec	145 μsec	146 μsec

As compared to the results given in Table V, the yield for the same fault rate is lower for all methods for the 3-bit adder. This is more 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 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 clearly produces better yields with the same faulty PLA structures, there is a cost for this improvement. When we ran the 3-bit adder tests used to generate Table VI, 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 VII. On average, the CBPM method seems to be about 1.5 to 2 times times faster than the QCAPM in finding successful mappings.

6.2 Fault Distribution and Larger Benchmarks

In Section 6.1, we compared two different mapping methods by considering both the mapping success rate and mapping runtime over the course of many random tests. Though the fault distribution adopted was sufficient for comparing different mapping methods, it does not correspond very well to the expected occurance frequency of PLA cell faults. To improve our approach, we wanted to generate faulty PLA structures using known correlations between device faults 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 faults 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 to assign device faults at this level. It is possible, however, to assign faults at the PLA cell level, while improving the fault distribution in the

Table VIII. PLA Yield for Mapping 5 Benchmarks to a (150,4000,100) PLA Using QCAPM Method Without and With Partial Line Faults Considered

Benchmark	add3		alu4		ex1010		apex4		seq	
Size	(12,31,4)		(28,952,8)		(20,1024,10)		(18,441,19)		(82,1066,35)	
Fault Rate	w/o	with	w/o	with	w/o	with	w/o	with	w/o	with
0.005	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
0.003	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%
0.002	15%	52%	0%	0%	0%	0%	0%	0%	0%	0%
0.001	99%	100%	88%	90%	65%	68%	8%	15%	0%	0%
0.0009	100%	100%	98%	99%	93%	93%	19%	24%	0%	0%
0.0008	100%	100%	100%	100%	99%	99%	52%	61%	0%	0%
0.0005	100%	100%	100%	100%	100%	100%	100%	100%	38%	39%
0.0004	100%	100%	100%	100%	100%	100%	100%	100%	93%	93%
0.0003	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

randomly generated PLA structures. Specifically, we used the information in Table IV 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 IV would lead to different rates for each PLA cell fault type according to the relationships between device faults and cell faults.

In addition to providing a better 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 using the fault distribution data from Table IV. We then used our graph matching tool to map five different benchmarks of varying sizes to these randomly generated structures. These tests were done once while not considering the partial line faults discussed in Section 3.4, 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 VIII. 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 included by showing the number of literals, implicants, and functions in the form (lit,imp,func).

In every case, either the two tests resulted in the same yield rate, or the tests that considered partial line faults ("with" columns) resulted in higher yield rates. In some instances the difference was significant. Good examples include the add3 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 PLA cell fault rates for each benchmark in which the yield goes from 100% yield down to 0% yield with only a small increase in cell fault rate. It seems that the partial line fault consideration is most

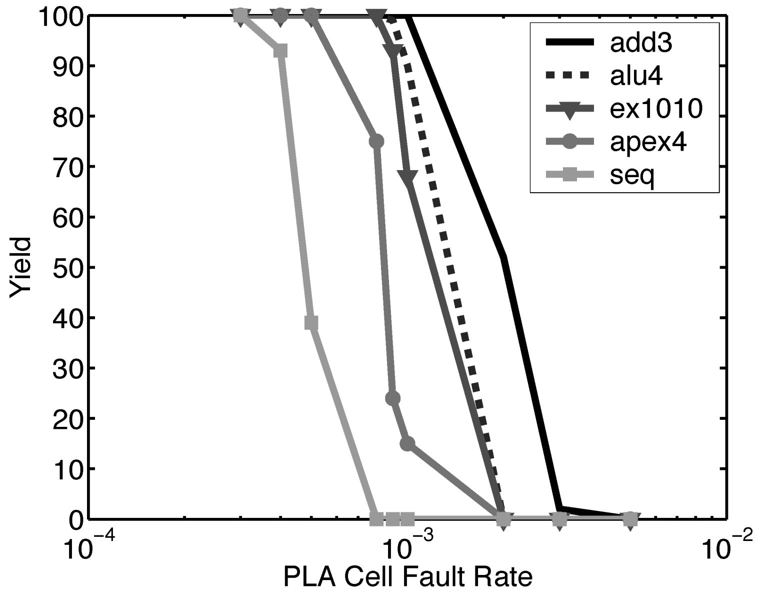

Fig. 11. A plot of PLA yield vs. PLA cell fault rate for mapping 5 benchmarks to a (150,4000,100) PLA using QCAPM method without and with partial line faults considered.

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 Crocker et al. [2007], a first-order yield analysis was provided, 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 would drop significantly with only a small increase in fault rate. The calculations also showed that as the PLAs increased in size, the cutoff point would shift towards lower fault rates, indicating that small PLAs would be able to tolerate higher fault rates while large PLAs would only tolerate lower fault rates. We plotted the results from Table VIII (with partial line faults considered) expecting to see similar trends. The plot uses a logarithmic scale for the PLA cell fault rate on x-axis, and a linear scale for the yield on the y-axis.

From Figure 11, 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 [Crocker et al. 2007]. For each benchmark, there is a small range of fault rates that correspond to a quick decrease in yield. The cutoff point is different for each benchmark, but they all exhibit the same behavior. These results confirm the expectations calculated in Crocker et al. [2007].

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

Table IX. Fault Rate Cutoff Point and Benchmark Characteristics for All Tested Benchmarks

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

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 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 Figure 11, with the smaller ones towards the right (able to tolerate higher fault rates) and the larger ones towards the left (only able to tolerate lower fault rates). 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 product and output terms.

There are many factors that could affect the mapping success rate for various benchmarks mapped onto defective PLA structures. While our results indicate that size has the most impact, a benchmark is composed of a certain 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. 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 IX. 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. Other benchmarks analyzed (not reported here due to space constraints) are generally consistent with the above results. A systematic study would be needed in order to quantify the importance of each parameter, which we would like to study in future work.

7. CONCLUSIONS AND FUTURE WORK

In the area of reconfigurable nanotechnology, much effort has focused on nanowire or nanotube PLAs using diode logic. Defect studies, fault models, routing algorithms, and performance calculations have been obtained from the mapping of logical benchmarks. In QCA research, similar studies have been done. From reconfigurable designs [Hu et al. 2006] and defect studies [Momenzadeh et al. 2005; Ottavi et al. 2005; Niemier et al. 2006] up to fault modeling [Huang et al. 2004; Crocker et al. 2007] and this work on mapping, QCA is proving to be an exciting emerging technology with several implementation alternatives that target many applications.

We have presented a fault model unique to QCA and have applied it to a redundant PLA design. This fault model considers explicit connections between defect types, device fault types, and PLA cell fault types. Based on the fault study, we have proposed a new graph model for mapping Boolean functions to faulty PLAs. Yield results discussed in Section 6 show that higher yields are possible with the improved graph representation and QCAPM method. Further refinements are possible by considering partial line faults, and this can lead to even better yields, especially for PLAs with cell fault rates in the critical region around the cutoff point.

There is still much work to be done, especially for QCA PLAs. We plan on using various tools and benchmarks to look at how QCA PLAs compare to other technologies in terms of area, delay, and power. While work has been done to consider general power consumption for magnetic QCA [Niemier et al. 2007], it was not specific to reconfigurable logic. The QCA PLA reconfigurable architecture is an excellent vehicle for logic mapping, and we intend to utilize our mapping work to look at logic density, area requirements, and power consumption for certain logic benchmarks. While this work attempts to remain implementation independent, any performance data would have to be implementation specific. In that vein, we also plan to explore methodologies to lower the EDR. For example, physical-level simulation has shown that one can effectively remove faults in MQCA circuit constructs by using a stronger clocking field [Niemier et al. 2008], which is also applicable to MQCA PLAs. This suggests that we can reduce or eliminate faulty behavior at the expense of increased energy, as the applied field strength is determined by the amount of current in a clock wire [Niemier et al. 2007]. More systematic research in this direction will be conducted.

Another area of research that we are exploring involves systems of PLAs for mapping logical benchmarks. Instead of using one large PLA to map the Boolean functions, logic synthesis could be used to break up the logic so that it could fit onto many smaller PLAs. Since smaller logic and smaller PLAs can tolerate higher fault rates and still maintain acceptable yield, such an approach could improve fault tolerance. On the other hand, systems of PLAs would require interconnect, and that interconnect would need to be analyzed for fault tolerance and yield. Work in this direction would need to consider how systems of many smaller PLAs would compare to a large PLA in terms of area, yield, and other metrics. We are also working to develop an efficient and effective fault detection process for the QCA PLA, and for systems of PLAs with interconnect. This work has been drawing from a previous detection scheme proposed in Hu et al. [2006]. We are working to expand and improve this scheme.

REFERENCES

AMLANI, I., ORLOV, A., TOTH, G., BERNSTEIN, G., LENT, C., AND SNIDER, G. 1999. Digital logic gate using quantum-dot cellular automata. *Science 284* 5412, 289–291.

CORDELLA, L., FOGGIA, P. SANSONE, C., AND VENTO, M. 2004. A (sub)graph isomorphism algorithm for matching large graphs. *Trans. Patt. Anal. Mach. Intell. 26,* 10, 1367–1372.

CROCKER, M., HU, X. S., AND NIEMIER, M. 2007. Fault models and yield analysis for QCA-based PLAs. In *Proceeding of the International Symposium on FPL*, 435–440.

CROCKER, M., NIEMIER, M., HU, X. S., AND LIEBERMAN, M. 2008. Molecular QCA design with chemically reasonable constraints. *ACM J. Emerg. Tech. Comput. Syst. 4*.

DEHON, A. AND WILSON, M. 2004. Nano-wire based sublithographic programmable logic arrays. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 123–132.

DEMJANENKO, M. AND UPADHYAYA, S. J. 1990. Yield enhancement of field-programmable logic array by inherent component redundancy. *IEEE Trans. Comput.-Aid. Des. 9,* 8, 876–884.

DONAHUE, M. AND PORTER, D. OOMMF user's guide, version 1.0, Interagency Report NISTIR 6367. http://math.nist.gov/oommf.

EL-SONBATY, Y. AND ISMAIL, M. A. 1998. A new algorithm for subgraph optimal isomorphism. *Pattern Recog. 31*, 205–218.

HASELMAN, M. AND HAUCK, S. 2008. The future of integrated circuits: A survey of nano-electronics. *Proc. IEEE*.

HENNESSY, K. AND LENT, C. 2001. Clocking of molecular quantum-dot cellular automata. *J. Vac. Sci. Tech. B 19(5)*, 1752–55.

HOGG, T. AND SNIDER, G. 2006. Defect-tolerant adder circuits with nanoscale crossbars. *IEEE Trans. Nanotech. 5,* 2.

HU, X. S., CROCKER, M., NIEMIER, M., YAN, M., AND BERNSTEIN, G. 2006. PLAs in quantum-dot cellular automata. In *Proceeding of the International Symposium on VLSI*.

HUANG, J., MOMENZADEH, M., TAHOORI, M., AND LOMBARDI, F. 2004. Design and characterization of an and-or-inverter (AOI) gate for QCA implementation. *ACM Great Lakes Symposium on VLSI*. 426–29.

IMRE, A., CSABA, G., JI, L., ORLOV, A., BERNSTEIN, G., AND POROD, W. January 13, 2006. Majority logic gate for Magnetic Quantum-dot Cellular Automata. *Science 311*, 5758, 205–208.

LENT, C. AND TOUGAW, P. 1997. A device architecture for computing with quantum dots. *Proc. IEEE 85*, 541.

MANEM, H., PALIWODA, P. C., AND ROSE, G. S. 2008. A hybrid CMOS/nano FPGA architectire build from programmable majority logic arrays. *Great Lakes Symposium on VLSI*. 249–254.

MITIC, M., CASSIDY, M., PETERSSON, K., STARRETT, R., GAUJA, E., BRENNER, R., CLARK, R., DZURAK, A., YANG, C., AND JAMIESON, D. July 5, 2006. Demonstration of a silicon-based Quantum Cellular Automata cell. *Appl. Phys. Lett. 89*, x.

MOMENZADEH, M., JING, H., TAHOORI, M., AND LOMBARDI, F. 2005. On the evaluation of scaling of QCA devices in the presence of defects at manufacturing. *IEEE T. NanoTech. 4*, 6, 740–743.

NIEMIER, M., ALAM, M., HU, X. S., BERNSTEIN, G., POROD, W., PUTNEY, M., AND DEANGELIS, J. 2007. Clocking structures and power analysis for nanomagnet-based logic devices. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'07)*, 26–31.

NIEMIER, M., CROCKER, M., AND HU, X. S. 2008. Clocking structures and power analysis for nanomagnet-based logic devices. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*.

NIEMIER, M., CROCKER, M., HU, X. S., AND LIEBERMAN, M. 2006. Using CAD to shape experiments in molecular QCA. In *Proceedings of the International Conference on Computer-Aided Design*. 907–914.

OTTAVI, M., MOMENZADEH, M., AND LOMBARDI, F. 2005. Modeling QCA defects at molecular-level in combinational circuits. In *Proceedings of the IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, 208–216.

QI, H., SHARMA, S., LI, Z., SNIDER, G., ORLOV, A., LENT, C., AND FEHLNER, T. 2003. Molecular Quantum Cellular Automata cells. electric field driven switching of a silicon surface bound array of vertically oriented two-dot molecular quantum cellular automata. *J. Amer. Chem. Soc. 125*, 15250–15259.

RAO, W., ORAILOGLU, A., AND KARRI, R. 2006. Topology aware mapping of logic functions onto nanowire-based crossbar architectures. In *Proceedings of the IEEE/ACM Design Automation Conference*.

SNIDER, G., KUEKES, P. J., HOGG, T., AND WILLIAMS, R. S. 2005. Nanoelectronic architectures. *Appl. Phys. A 80*, 1183–1196.

SNIDER, G. AND WILLIAMS, R. S. 2007. Nano/cmos architectures using a field-programmable nanowire interconnect. *Nanotech. 18*, 035204.

SOMENZI, F. AND GAI, S. May 1986. Fault detection in programmable logic arrays. *Proc. IEEE 74,* 5, 655–667.

STRUKOV, D. AND LIKHAREV, K. 2005. CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotech. 16*, 888–900.

TOUGAW, P. AND LENT, C. 1994. Logical devices implemented using quantum cellular automata. *J. App. Phys. 75*, 1818.

WEY, C. L. 1988. On yield consideration for the design of redundant programmable logic arrays. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst. 7*, 528–535.

Received October 2008; revised January 2009, March 2009; accepted March 2009 by Iris Bahar