A Potentially Implementable FPGA for Quantum Dot Cellular Automata

Michael Thaddeus Niemier
University of Notre Dame
Dept. of Comp. Sci. and Eng.
Notre Dame, IN 46556, USA
mniemier@nd.edu

Arun Francis Rodrigues
University of Notre Dame
Dept. of Comp. Sci. and Eng.
Notre Dame, IN 46556, USA
arodrig6@nd.edu

Peter M. Kogge
University of Notre Dame
Dept. of Comp. Sci. and Eng.
Notre Dame, IN 46556, USA
kogge@wizard.cse.nd.edu

Abstract

While still relatively "new", the quantum-dot cellular automata (QCA) appears to be able to provide many of the properties and functionalities that have made CMOS successful over the past several decades. Early experiments have demonstrated and realized most, if not all, of the "fundamentals" needed for a computational circuit – devices, logic gates, wires, etc. This study introduces the beginning of a next step in experimental work: designing a computationally useful – yet simple and fabricatable circuit for QCA. The design target is a QCA Field Programmable Gate Array.

1. Introduction

One alternative to silicon/CMOS is an approach to computing with quantum dots called the quantum-dot cellular automata (QCA). First proposed in 1993 by Lent, et. al and fabricated in 1997, an idealized QCA cell/device can be viewed as a set of four change containers or "dots", positioned at the corners of a square [7], [8]. The cells contain two extra mobile electrons which can quantum mechanically tunnel between dots but, by design, cannot tunnel between cells. The dots can be realized in several different ways – electrostatically formed quantum dots in a semiconductor, small metallic islands connected by tunnel junctions, or redox centers in a molecule. The barriers between dots should be high enough so that charge can move only by tunneling and is therefore localized in the dots and not in the connectors. The configuration of charge within the cell is quantified by cell polarization, which can vary between P = -1, representing a binary "0", and P = +1, representing a binary "1". This is illustrated in Figure 1.

Unlike conventional logic in which information is transferred from one place to another by means of electrical current, QCA is at its base self-latching where information is stored at each device by the positions of single electrons and logic functions are performed not by electron flow, but rather by Coulombic interactions between cells. The result

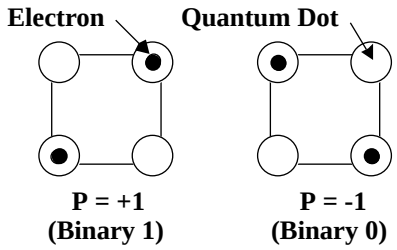

Figure 1. QCA cell polarizations.

is a technology where even the interconnect is made out of the same self-latching devices as the logic functions, bringing "pipelining" down well below the level of even a simple logic gate [15].

While other work [10] has addressed "custom" designs in QCA regarding the complexity of a complete microprocessor, in the near term it is appropriate to focus on a design that is not the fastest or densest circuit with respect to "custom" designs, but rather one which would allow for early implementation experiments. The target discussed in this paper is a QCA Field Programmable Gate Array (FPGA), where simplicity and regularity, coupled with the potential for self-assembly, are at the heart of the design. Because of the inherent differences in models between CMOS and QCA, such a design target represents a fertile source of design problems including the choice of logic complexity to provide in an individual logic block node, interconnection topology and mechanization, and programming.

These topics will all be discussed within this paper but we will begin with a discussion in Section 2 of what has been accomplished experimentally with QCAs. Next, in Section 3 some basic and necessary properties of FPGAs and how they relate to QCA will be introduced. Then, in Section 4 specific obstacles to generating a QCA FPGA will be discussed. In Section 5 the basic logic block for the QCA FPGA will be described, in Section 6 the interconnection scheme for the QCA FPGA will be explained, and in Section 7 resulting schematics will be presented. Finally, we will conclude in Section 8.

2. Experimental QCA

In the introduction we indicated that the goal of this work was to develop a QCA circuit for near term implementation experiments. Thus, we feel it is important to stress that QCA has moved beyond the realm of theory and to detail what actual devices and circuits have been constructed and what experiments have been conducted. Generically speaking, experimental QCA work can fall under one of two categories – work with prototype metal dot systems and work with molecules targeted for a QCA implementation.

Researchers at the University of Notre Dame have demonstrated the first QCA cell, showing that the position of a single electron can control the position of a second single electron [11]. This demonstrated the basic feature of the QCA paradigm – that information can be coded in the configuration of charge. In these experiments, aluminum islands act as the dots, coupled by aluminum oxide tunnel junctions [1]. Using this metal tunnel-junction technology, a majority gate – the basic logical device in QCA – was experimentally demonstrated. In a majority gate, three inputs are applied to a four-dot QCA cell, and the majority vote of the inputs determines the polarization of an output cell. This demonstrates the fundamental QCA logic function which is actually the logic equation: $AB + BC + AC$ with A, B, and C being inputs. (A functionally complete logic set can be derived from this device by permanently keeping one of the inputs to the majority gate in a binary 0 or a binary 1 polarization – which results in AND/OR gate equivalents, and because the means to invert a QCA signal exists.) A QCA binary wire has also been fabricated with this technology and essentially consists of a linear array of QCA cells [14]. Coulombic interaction between cells makes nearby cells align in the same state/polarization.

Finally, clocked QCA cells have been investigated. Clocking groups of QCA cells enables complete control of the direction of information flow and as was eluded to earlier, QCA shift registers become the paradigm for moving information from one place to another. Most recently, clocked QCA switching has been demonstrated in a three-dot cell [12]. In these clocked metal-dot cells, the dots are completely isolated electrically from any source of current. Unlike single electron transistors, these cells function with no current flowing through them, even in the switching phase. They are therefore particularly appropriate prototypes for molecular QCA cells.

Before discussing experimental work with molecular QCA cells, it is important to briefly enumerate two other important benefits of clocked QCA cells. First, the binary information stored in a QCA cell can be held by the clock signal and copied to neighboring cells. Addressable memory cell arrays have been designed and simulated and a single bit QCA memory has been demonstrated experimentally. Second, when one device causes another to switch, there is inevitable loss of energy to the environment through irreversible processes (phonons, plasmons, molecular vibrations, etc.). For signals to continue to propagate, this lost signal energy must be restored. In conventional transistors, this energy comes from the power supply. In clocked QCA circuits, this energy comes from the clock itself. QCA circuits can therefore exhibit true signal power gain. This has been shown theoretically and has recently been observed experimentally [13]. Much more will be said about the specifics of the QCA clock – particularly how it relates to circuit and system design – later on in this paper.

However, we will first conclude this section with a short discussion on molecular QCA. Creating QCA cells composed of single molecules holds enormous promise for realizing the ultimate limits of electronic device miniaturization and integration [6]. In contrast to present metal-dot cells, the small size of molecules means that Coulomb energies are much larger, so room temperature operation is possible. In addition, the power requirements and heat dissipation of QCA are low enough that high-density molecular logic circuits and memory are feasible. In contrast to lithographic device fabrication techniques which always introduce variations in device characteristics, each molecular cell can be made exactly identical using chemical synthesis. Combining molecular QCA with nanoelectronics and molecular/biomolecular sensors opens new possibilities for an integrated molecular technology. Currently, work is underway to develop target models and substrates [3], [2], [5].

Thus, the basic building blocks for computational devices have all been theoretically studied and in most cases experimentally verified. The next logical step is to combine computational building blocks into a realizable computational circuit. The best match between useful computation and design simplicity is a QCA FPGA.

3. An FPGA Design Target

Generically, an FPGA is a collection of functionally complete logic elements that are arranged in some interconnection framework. A typical CMOS interconnection scheme involves signals entering an FPGA circuit via some input buffers which are then transferred to horizontal wires. These horizontal wires cross with vertical wires throughout the FPGA and programmable connections can usually be made at crossings to facilitate data routing. Another common interconnection scheme is *direct interconnection*. For example, often in FPGAs, paths exist for connections from a logic block to any or some of its North, South, East or West neighboring logic blocks [4]. Duplicating such a scheme in QCA would obviously be quite simple – QCA cells could just be used to hard-wire these connections. However, direct interconnect is only effective if it is feasible to make appropriate function assignments to adjacent logic cells. As designs become more complex, this becomes more difficult

to do. One common solution to this problem are dedicated long-line interconnect wires (i.e. wires that could bypass logic blocks to move a signal to another part of the FPGA). How these techniques might map to a QCA FPGA will be the major focus of the next section.

4. FPGAs in QCA

While long wires work well in CMOS, the nature of the clock makes them a much more difficult task in QCA. Unlike the standard CMOS clock, the QCA clock is not a signal with a high or low phase. Rather, the clock changes phases when potential barriers that affect a group of QCA cells (a *clocking zone*) pass through four clock phases: *switch* (unpolarized QCA cells are driven by some input and change state), *hold* (QCA cells are held in some definite polarization – i.e. some binary state), *release* (QCA cells lose their polarization), and *relax* (QCA cells remain unpolarized). One clock cycle occurs when a given clocking zone has cycled through all four clock phases and the potential barriers that implement the clock could actually be generated by CMOS circuitry. The net effect of this scheme is that even on a simple wire, data is latched as it moves from one clocking zone to the next (analogous to a shift register). An example of a 1-bit value propagating down a five cell QCA wire is illustrated in Figure 2.

It is this inherent self-latching that forms the heart of the problem for creating dedicated long-line interconnect wires in a QCA FPGA. Information being transmitted on the longer wire will be "pipelined" and its transmission will not be instantaneous (unlike electron flow in a metal wire in CMOS). Thus, coordinating the arrival times for input signals to a given function block becomes a much more difficult task. The obvious alternative to this problem would simply be placing all of the cells for a long-line interconnect wire in fewer or just a single clocking zone. However, this introduces problems of signal reliability (as a QCA wire grows in length the probability that all cells will switch successfully decreases) [9], and introduces design irregularity. Nevertheless, while these problems should be solvable, others loom.

Perhaps the most useful feature in a CMOS FPGA is the mechanism for general purpose interconnect. One common/generic means for this is a grid of metal lines that resides between function blocks. These form a switching matrix which permits several distinct interconnection options. The junction where a horizontal and vertical wire cross usually consists of a network of pass transistors (see Figure 3). For example, one common configuration would let a signal entering the junction from any given direction make a left turn, right turn, or continue straight ahead. To accomplish this, six pass transistors are required [4]. The unique feature of a pass transistor is that it essentially allows current (i.e. information) to flow between *a* and *b* in either

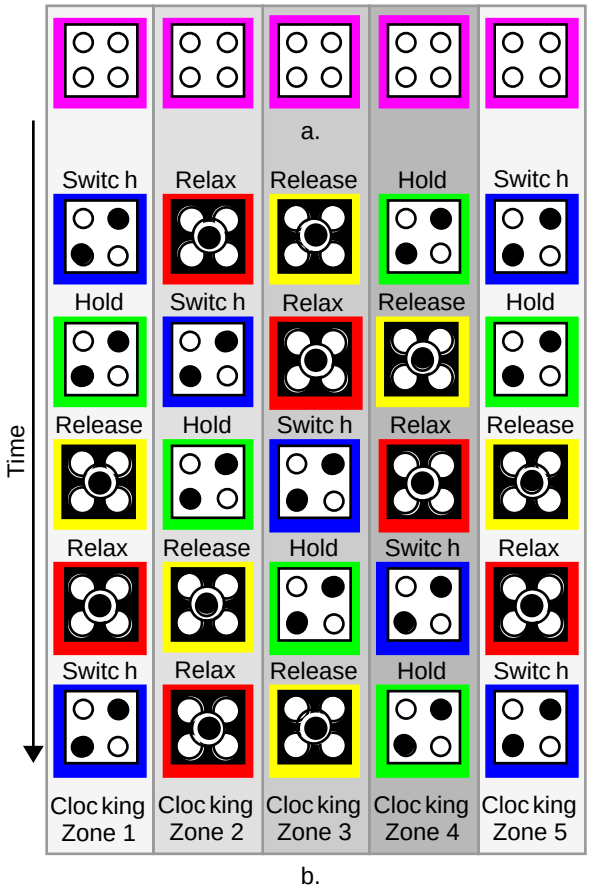

Figure 2. Part (a.) shows a physical 5-cell wire while part (b.) shows a value propagating down the wire.

direction. However, in QCA information is not moved by electron *flow* but rather by Coulombic interaction between electrons in the quantum dots. Because *nearness* between QCA cells is required to move information from *a* to *b* there is no obvious way to create the equivalent of a pass transistor (either bi- or uni-directional) using only QCA devices. This makes generating a switching matrix for a QCA FPGA much more difficult (although not impossible).

To understand how the equivalent of at least a unidirectional QCA pass transistor or switch might be implemented, its worthwhile to consider the exact purpose of the *relax* clock phase. Without it, QCA cells in the *switch* phase could be driven from two different directions (i.e. from cells with a definite polarization in the adjacent *hold* phase and cells with an initial polization in the adjacent *release* phase). The *relax* phase acts as a buffer to ensure that this does not occur. Thus, the *relax* phase has the effect of "removing" a group of QCA cells from a given design. Using this idea, routing could be accomplished by using the clock to selectively "turn off" groups of QCA cells to create switches.

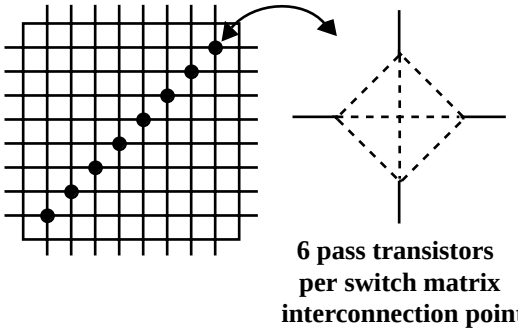

Figure 3. An example of a "CMOS" switching matrix and the pass transistors needed to create it (dashed lines).

A similar mechanism could be used to program logic or routing cells. However, instead of "turning off" groups of QCA cells by keeping them in the *relax* phase, binary values could be stored by keeping clocking zones in the *hold* phase.

This technique will be used to develop an interconnection mechanism for a QCA FPGA array. A complete schematic and programming examples will be developed in the next 3 sections.

5. The Basic Logic Block

When studying what elements should make up the logic block of a QCA FPGA, 3 things were considered: what functionality was absolutely needed, how would logic elements in the block be programmed, and was it simple enough to potentially fabricate. Logic block designs that were considered included a single NAND or NOR gate, a majority gate that could be programmed to act as an AND or OR gate, and a logic block with some form of memory.

The latter two options – a sea of programmable AND or OR gates or a logic block with some simple combinational logic and storage capacity – would easily result in the most "functional and useful" FPGA. For instance, a sea of 2-input AND and OR gates would be ideal for implementing sum-of-products or product-of-sums logic equations. And, FPGA logic-blocks with some kind of built-in storage elements would obviously assist in implementing a simple microprocessor or finite state machine. However, both of these techniques also have disadvantages. AND and OR logic only is not functionally complete as some means for conditional inversion is also necessary. Things are further complicated given that AND and OR logic cannot be programmed to implement an inverter. Adding a specific inverter logic block would complicate our supposedly "simple" design, and using a multiplexor to conditionally select an inverted input would overwhelm the basic logic block in terms of complexity (as will be seen in the next section). Also, additional problems arise as the AND/OR cells and the memory elements would have to be conditionally programmed. In this case, the proposed *data routing* clocking scheme would have to be extended to route, store, and refresh a programming signal (for the AND/OR gates and memory elements).

These issues were deemed too complex for a first attempt at a fabricatable design and a majority gate configured to act as a NAND gate was chosen as the logic block. This design was considered advantageous as it would be simple to build, data routing would handle all "programming" of the cells, and circuits composed entirely of NAND gates are functionally complete. This scheme also avoids the need to generate a signal path for configuring majority gates to provide selective functionality and to store majority gate programming bits.

6. Interconnect

Interconnection networks for traditional CMOS FPGAs utilize SRAM memory to configure the pass transistors which regulate the flow of information between programmed logic elements. However, in QCA there are no direct counterparts to SRAM or pass transistors, and, as was just discussed, to keep fabrication complexity to a minimum the logic elements are static. Fabrication concerns also dictate that routing elements be simple and modular. A number of designs were analyzed to overcome these limitations.

Perhaps the most direct "translation" of CMOS routing to QCA would be an array of programmed multiplexors. In QCA, a simple 2-to-1 multiplexor or 1-to-2 selector can be built from three majority gates. Combining these basic units together can form an any-to-any interconnection network of arbitrary size. However, this network is plagued by a number of problems. The most pressing concern is that the control signals for the multiplexors would have to be stored and carried to the network. The lack of a QCA flip-flop equivalent makes this storage an expensive prospect and points to another weakness - that of complexity. With a minimal routing element being comprised of at least 6 majority gates, it is already several times more complex than the proposed logic elements. When compared against other routing elements to be discussed, the simple multiplexor performs poorly with regard to total area measurements (see Figure 4).

It is possible to devise a more efficient routing mechanism by taking advantage of the QCA clock scheme. FPGA routing networks can be seen as presenting a set of possible connections. By programming a routing network, we select a subset of those connections matching the connections we actually need. As was discussed, in QCA signals are transmitted via an inherent "pipeline" created by the four phase clock. If we devise a network of QCA wires which presents connections between all of our logic elements, we can program this network by simply not applying clocking

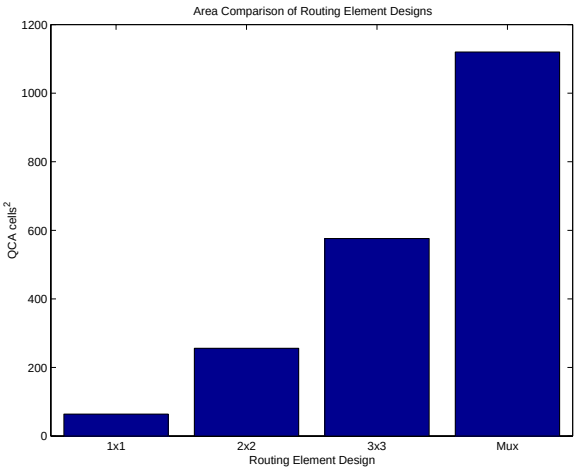

Figure 4. Multiplexor based designs quickly "scale out of range".

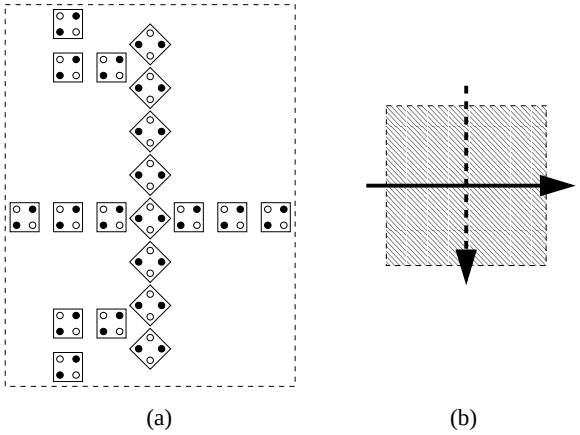

Figure 5. A simple 1 clock zone routing element (a) allows two signals to cross (b), but is ultimately ineffective as a routing element. (Also, note that when a 90-degree QCA cells is placed between 2 45-degree QCA cells, it serves to act as a "ripper", transferring a value to the 45-degree wire).

fields to the unwanted connections until the only remaining connections are those which we actually need. Because the clock fields could be generated by traditional CMOS circuitry this programming can be easily carried out. As was mentioned, cells in unclocked zones are left in the *release* phase and remain unpolarized, thus they do not influence any adjoining cells. An additional benefit of programming by the clock is that multiple adjacent routing elements (and clocking zones) can be joined into one "larger" clock zone, allowing a signal to propagate over several routing elements in one cycle.

The simplest routing element would be to cross two QCA wires in a single clock zone (see Figure 5). (In QCA, with one wire comprised of cells oriented at 90-degrees and another comprised of cells oriented at 45-degrees, it would be possible for the wires to cross without interfering with either signal on either wire and achieve two layer routing. However, without any mechanism to "jump" from the 45 to the 90-degree wires, signals in this simple element would not be able to "turn corners" - making it useless as a routing element, but a useful basis of comparison.

The problem of "turning corners" can be addressed by having a routing element comprised of 4 clock regions in a 2 by 2 pattern (see Figure 6(a)). Signals are again carried by perpendicular 45 and 90-degree wires, and an additional wire connects the two. This additional wire is placed in the upper right clock region. If this region is left unclocked, signals will only propagate along the perpendicular 45 and 90-degree wires (see Figure 6(b)). However, if this region is clocked, a signal can propagate from one of the perpendicular wires to the other, allowing a signal to fan out. The draw back to this configuration is that a signal which is switched from one perpendicular wire to another continues to propagate along both wires. If left unchecked, this signal will interfere with signals traveling in adjacent blocks.

One way to stop signals from "colliding" is to expand the basic routing element to a 3x3 clock region element (see Figure 7a). In addition to the benefits of the 2x2 element, the 3x3 allows us even greater flexibility in routing (see Figure 7b). Additionally, we avoid a drawback of the 2x2 element; namely regions can remain unclocked to provide a buffer against adjacent signals from adjacent routing elements (see Figure 6c). The drawback of the 3x3 design is the 125% expansion in area. A reasonable compromise is a 3x2 routing element (see Figure 8). It allows a high degree of density while still providing good flexibility in routing.

The routing elements discussed here are modular and can be easily connected with logic elements to form the complete FPGA (see Figure 9). Placing and routing for these FPGAs can be accomplished by a number of mechanisms already used for traditional CMOS as the organization and characteristics of the routing elements easily lends itself to a number of two layer grid routing techniques. The ratio and pattern by which logic elements are embedded in the interconnect fabric can be adjusted for optimal channel depth.

7. A QCA FPGA Schematic

Now that a logic block and interconnection framework for our QCA FPGA have been established and linked, we can turn our attention to actually programming it. A fundamental part of any microprocessor is an ALU which invariably uses (and needs!) an adder (previous custom designs and architectural studies for instance used an adder). [10]. With this in mind, the logic for a full adder was converted

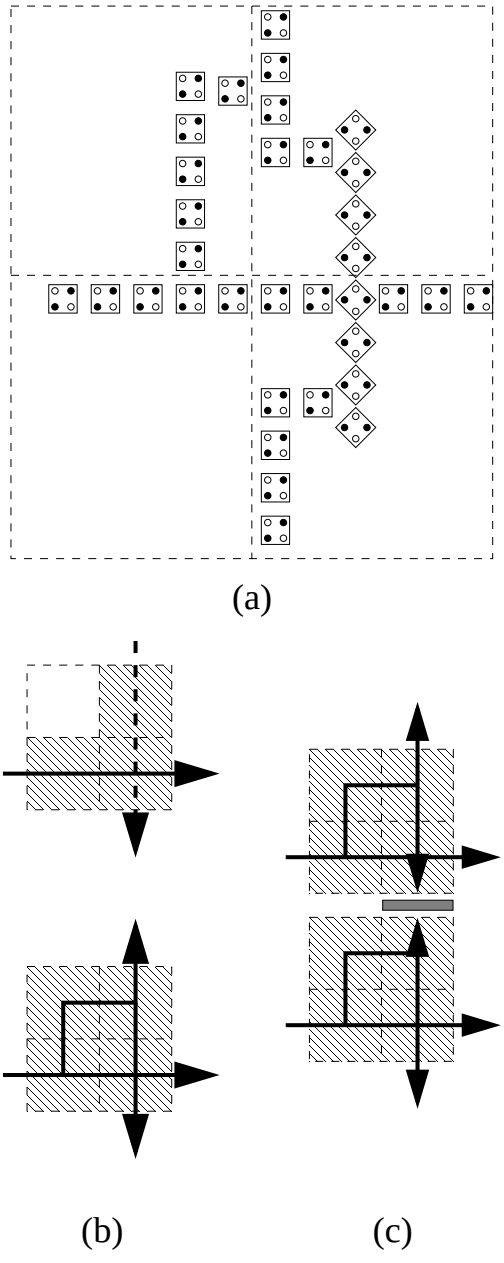

Figure 6. A 2x2 zone routing element (a) allows two layer routing (b), but signals can interfere with adjacent routing elements (c).

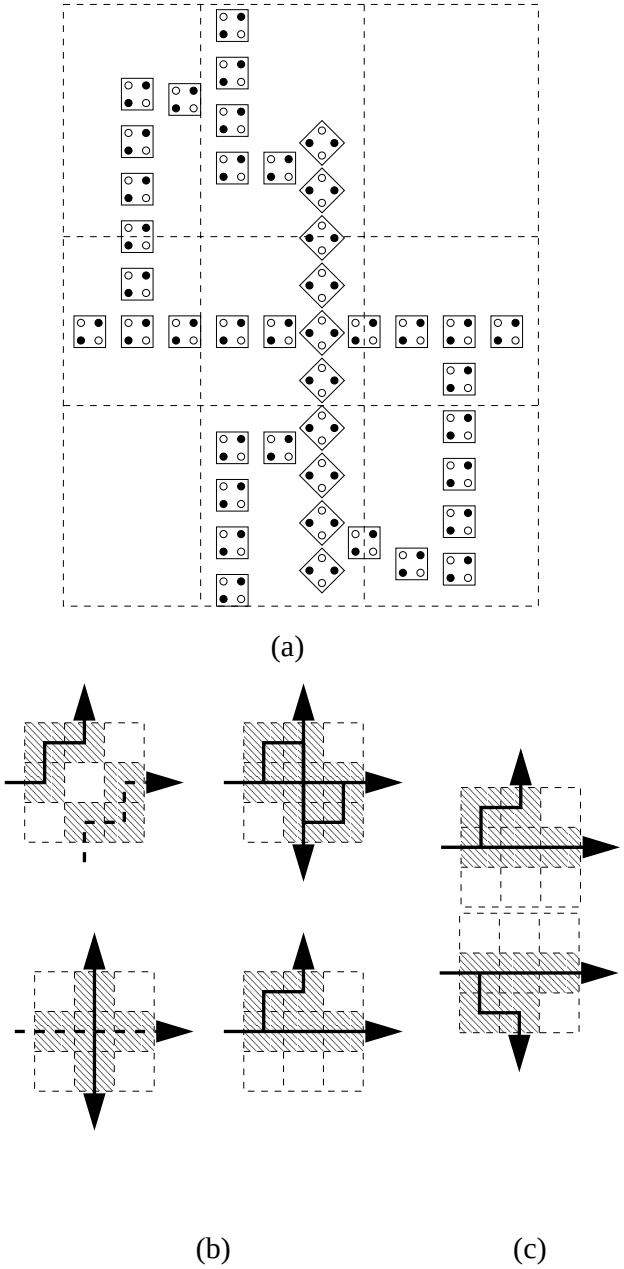

Figure 7. A 3x3 zone routing element (a) allows flexible routing (b), and provides protection against interference(c).

to NAND logic (see Figure 10) and was then "placed" in the QCA NAND-based FPGA (see Figure 11). In Figure 11 circles represent logic blocks and squares represent paths in different clocking zones. Routing paths are always a multiple of four squares (or "clocking zones") to correspond to the four phases of the QCA clock and the FPGA logic block routing was performed by hand to achieve an optimal density. (Note that the numbers on these two schematics "match up" the gates between these two designs and in these figures only the sum-bit logic is included.)

Finally, it is important to point out one additional place and routing feature of the NAND-based adder implemented in the FPGA. Earlier in Section 4 we indicated that as information is transferred from point *a* to point *b* in QCA, it is inherently "pipelined" because of the nature of the QCA clock. A consequence of this was increased difficulty in coordinating the arrival times for a set of input signals to a

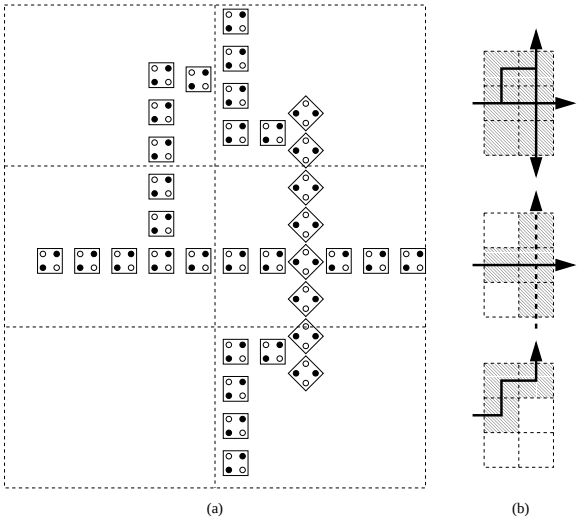

Figure 8. A 3x2 zone routing element (a) allows flexible routing (b), at a reasonable size

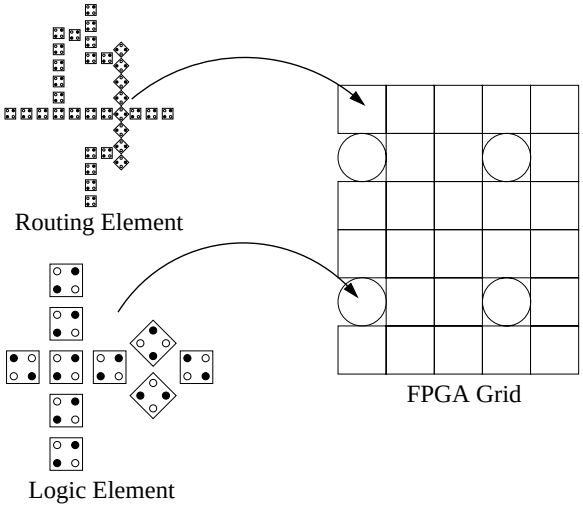

Figure 9. Routing and logic elements combine to form the FPGA

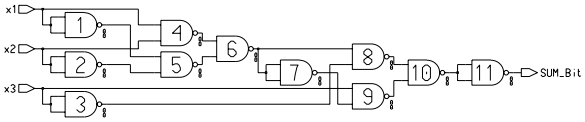

Figure 10. The sum logic for an all-NAND-based adder.

logic gate. Upon examing Figure 11, there are places where the pipelined paths through the squares/clocking zones are not equal. However, as alluded to in Section 6, an additional benefit of programming by the clock is that multiple adjacent routing elements can be joined into one clock zone, allowing a signal to propagate over several routing elements

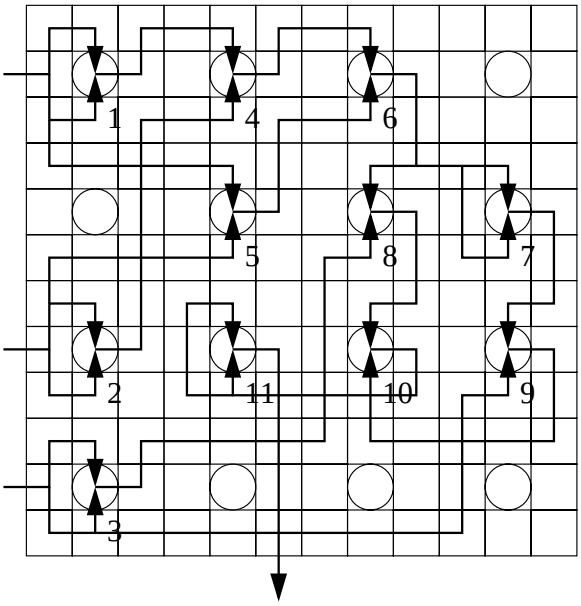

Figure 11. How the NAND gates would potentially map into a QCA FPGA (sum-bit only).

in one cycle. If these times are coordinated correctly, input signals will arrive at their respective logic gates simulatneously.

8. Conclusions and Future Work

To conclude, we have succeeded in designing the first QCA-based FPGA. Most importantly, the circuit is simple and regular and it, or its components, are potential candidates for future experimental work. Additionally, we have developed schemes to potentially implement switches in QCA which broadly expands its base of useful applications. We have also illustrated that even though this FPGA has primitive logic blocks, it is possible to place, route, and program more sophisticated circuits with it.

Now, admittingly, logical completeness is not the only requirement for an FPGA. To be useful, it must also have the ability to store state. The lack of a direct QCA equivalent to a flip-flop makes this difficult (especially with primitive logic blocks), however, it is possible to store state. Data can be stored by creating a QCA wire loop. It should be possible to construct such loops from routing elements, thus turning extra network capacity into extra storage capacity. Again, the design itself will remain simple with an eye toward implementation issues.

Finally, the authors would like to acknowledge the National Science Foundation, the Notre Dame Center for Nanoelectronics, and especially Dr. Craig Lent.

References

[1] I. Amlani, A. Orlov, G. Toth, G. Bernstein, G. Lent, and G. Snider. Digital logic gates using quantum-dot cellular automata. *Science*, 284:289–291, 1999.

[2] A. Aviram. Molecules for memory, logic, and amplification. *Journal of the American Chemical Society*, 110(17):5687–5692, 1988.

[3] K. Demadis, C. Hartshorn, and T. Meyer. The localized-to-delocalized transition in mixed-valance chemistry. *Chem. Rev.*, 101:2655–2685, 2001.

[4] J. Hayes. *Designing with FPGAs and CPLDs*. PTR Prentice Hall, New Jersey, 1994.

[5] N. Hush, A. Wong, G. Bacskay, and J. Reimers. Electron and energy-transfer through bridged systems .6. molecular switches – the critical-field in electric-field activated bistable molecules. *Journal of the American Chemical Society*, 112(11):4192–4197, 1990.

[6] C. Lent. Molecular electronics: Bypassing the transistor paradigm. *Science*, 288:1597–1599, 2000.

[7] C. Lent, P. Tougaw, and W. Porod. Bistable saturation in coupled quantum dots for quantum cellular automata. *Appl. Phys. Lett.*, 62:714, 1993.

[8] C. Lent, P. Tougaw, W. Porod, and G. Bernstein. Quantum cellular automata. *Nanotechnology*, 4:49–57, 1993.

[9] C. S. Lent and P. D. Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85:541, 1997.

[10] M. Niemier and P. Kogge. Exploring and exploiting wire-level pipelining in emerging technolgies. In *Proc. of the 28th International Symposium of Computer Architecture*, pages 166–177, 2001.

[11] A. Orlov, I. Amlani, G. Bernstein, C. Lent, and G. Snider. Realization of a functional cell for quantum-dot cellular automata. *Science*, 277:928–930, 1997.

[12] A. Orlov, I. Amlani, R. Kummamuru, R. Rajagopal, G. Toth, C. Lent, G. Bernstein, and G. Snider. Experimental demonstration of clocked single-electron switching in quantum-dot cellular automata. *Applied Physics Letters*, 85:2977–2984, 2000.

[13] A. Orlov, I. Amlani, R. Kummamuru, R. Rajagopal, G. Toth, J. Timler, C. Lent, G. Bernstein, and G. Snider. unpublished.

[14] A. Orlov, I. Amlani, C. Lent, G. Bernstein, and G. Snider. Experimental demonstration of a binary wire for quantum-dot cellular automata. *Appl. Phys. Lett.*, 74:2875–77, 1999.

[15] P. Tougaw and C. Lent. Logical devices implemented using quantum cellular automata. *Journal of Applied Physics*, 75:1818–1825, 1994.