DESIGNING DIGITAL SYSTEMS IN QUANTUM CELLULAR AUTOMATA

A Thesis

Submitted to the Graduate School

of the University of Notre Dame

in Partial Fulfillment of the Requirements

for the Degree of

Masters of Science in Computer Science and Engineering

by

Michael Thaddeus Niemier, B.S.

Peter M. Kogge, Director

Department of Computer Science and Engineering

Notre Dame, Indiana

January 2004

DESIGNING DIGITAL SYSTEMS IN QUANTUM CELLULAR AUTOMATA

Abstract

by

Michael Thaddeus Niemier

The Quantum Cellular Automata (QCA) is currently being investigated as an alternative to CMOS VLSI. While some simple logical circuits and devices have been studied, little if any work has been done in considering the architecture for systems of QCA devices. This work presents one of the first such efforts when considering systems of QCA devices. Namely, a simple but complete processor dataflow has been designed exclusively in QCA. Additionally, techniques for floorplanning and simulating circuits have also been developed. Size projections for the dataflow designed are striking, as the design has the potential to be 900 times smaller than an end of the CMOS curve equivalent. Basic QCA device physics, floorplanning techniques, actual designs, simulation techniques, and size and power projections are discussed.

For mom and dad.

What you have taught me can best be summed up with this quote from Paulo Coelho's "The Alchemist", "Everyone on earth has a treasure that awaits him, his heart said. We, people's hearts, seldom say much about those treasures, because people no longer want to go in search of them. We speak of them only to children. Later, we simply let life proceed in its own direction, toward its own fate. But, unfortunately, very few follow the path laid out for them – the path to their destinies, and to happiness. Most people see the world as a threatening place, and, because they do, the word turns out, indeed, to be a threatening place."

Thanks for being the exception.

Your love and support has helped me get this far and undoubtedly will help me along the rest of the way.

CONTENTS

FIGURES

ACKNOWLEDGEMENTS

I would like to give special thanks to my advisor Dr. Peter Kogge for allowing me to pursue this project. The possibilities seem endless and remember, with nanoelectronics, “there’s plenty of room at the bottom”!

Thanks also to Dr. Craig Lent and Dr. Wolfgang Porod for many useful discussions and opportunities.

I would also like to thank the National Science Foundation for providing a fellowship and funding for this work. Additionally, thanks to the University of Notre Dame for providing an Arthur J. Schmidtt Presidential Fellowship.

To all of my friends I thank you for your assistance and support. In particular, I would like to thank Jason Keith, Shannon Kuntz, and Richard Murphy for productive (and sometimes unproductive) discussions. I would also like to thank undergraduates Michael Kontz and Walter Tuholski for their contributions to this project. Thanks also to Michael Macedonia for assistance with LaTeX to properly format this entire document!

Finally, I would like to thank Ferris Bueller for reminding me that, “Life moves pretty fast. If you don’t stop and look around once in awhile, you could miss it.” And, Jackie Robinson for reminding me that, “A life is not important except for the impact it has on other lives.”

CHAPTER 1

INTRODUCTION

1.1 An Introduction to the Problem

In 1965, Gordon Moore predicted that the number of transistors that could be integrated into a single die would grow exponentially with time. Moore's law has governed microprocessor manufacturing processes, and consequently microprocessor performance ever since. However, recent studies indicate that during the next two decades, the laws of nature will begin to govern microprocessor design and fabrication.

Today many integrated circuits are manufactured at 0.25-0.33 micron processes. As device sizes decrease to an order of 0.05 microns (a technology that is currently unrealizable), physical limitations of conventional electronics including power consumption, interconnect, and lithography will become increasingly difficult to surmount [10]. In fact, studies indicate that as early as 2010, the physical limits of transistor sizing may be reached [1]. Thus, it may not be possible to continue the norms of doubling the number of devices in a microprocessor every two years and doubling the clock rate every three years. Consequently to maintain trends of increasing microprocessor performance, other technologies should be studied.

1.2 An (Alternative) Solution

As an alternative to CMOS-VLSI, researchers have proposed an approach to computing with quantum dots, the quantum cellular automata (QCA). First proposed in 1994, unlike conventional computers in which information is transferred from one place to another by means of electrical current, QCA transfers information by propagating a polarization state [12, 11].

QCA is based upon the encoding of binary information in the charge configuration within quantum dot cells. Computational power is provided by the Coulombic interaction between QCA cells. No current flows between cells and no power or information is delivered to individual internal cells. The local interconnections between cells are provided by the physics of cell-to-cell interaction due to the rearrangement of electron positions [12].

While there is still much work to be done, early experimental results indicate that QCA may be an extremely viable alternative to CMOS. QCA cells and a simple QCA logical device have been successfully fabricated and tested [3]. However, the actual design of many of the circuits and devices required for a QCA microprocessor have not yet even been considered. What is required is a methodology for constructing and designing QCA circuits that are essential for a design such as a microprocessor. Furthermore, understanding how microprocessor components should be built/designed, should assist in the design of QCA physical devices. In short, what is required is a reference QCA architecture and the design tools to manipulate and analyze it.

1.3 Developing the Solution

In an effort to successfully develop a viable, understandable, and usable QCA architecture, the following four tasks have been accomplished: (1.) The first microprocessor dataflow has been designed completely with QCA devices. (2.) Floorplanning techniques have been developed to efficiently design and layout QCA circuits to allow for the fastest possible clock rate and circuits with the minimum required area.

(3.) A library of design rules for QCA circuits has been built. (4.) A simulator for QCA program has been written. This allows a QCA design or architecture to be constructed and simulated in a easy and efficient manner.

These four accomplishments provide an excellent starting point for future QCA designs and prove that QCA circuits can function logically (the physical realization must still be determined) and implement equivalent versions of CMOS circuits. Also, resulting designs can be and have been used to calculate area models/density gains and will later be used to calculate power and clock rate estimates and models.

1.4 Previous Work / QCA Design

Prior to this research, little work has been done in considering the architecture for systems of QCA devices. Basic logical devices and an adder have been designed by Lent, et al [12]. Such devices were simulated and verified with a program called AQUINAS (more in Chapter 5).

Memory has been studied by Terry Fountain, et al. at the University College of London and a complex SRAM cell has undergone successful simulation. Additionally, a simple shift register has also been constructed and simulated [6]. Both of these design schematics take advantage of an architecture developed by Fountain, et al. called the SQUARES architecture. The SQUARES architecture essentially consists of cells that are 5 QCA cells wide and 5 QCA cells high. A library of various QCA functional devices (see Chapter 2 for logic device types) such as a majority gate was then built up (with each device "housed" in a square) and use to construct the various schematics. While resulting in successful simulations, the drawback to the SQUARES architecture was that designs using it had less than optimal density.

It should be mentioned that the development of the SQUARES architecture stemmed from a perceived problem called the "time-delay problem". It was believed that in order for a QCA logical gate to switch successfully, all inputs to it had to arrive at the device cell at *exactly the same time*. However, this does not appear to be the case. More will be said about the functionality of QCA logical gates in Chapters 2 and 5.

Again, with these research efforts, by-in- large only QCA *devices* were considered, not the systems of devices and their interactions that are absolutely necessary for QCA to be considered a viable replacement to CMOS circuits.

1.5 A Summary of the Current Work

Initial work on the QCA architecture was spent understanding how QCA cells worked physically and understanding the few existing QCA logical circuit designs (i.e. the adder). To become more familiar with the new paradigms of the technology, other QCA components, such as an XOR gate and a multiplexor using the QCA logical device – the majority gate, were designed and studied. In doing so, it was discovered that for some circuits/devices a QCA version could only be constructed by implementing the direct logical equation (i.e. XOR = A'B + AB'). However, for circuits such as the adder, simplified versions could be constructed with QCA majority gates. (More will be said about this in Chapter 2).

It was then determined that it would be extremely valuable to create a program that could translate a schematic containing conventional Boolean logic gates/equations into a schematic consisting entirely of QCA majority gates/majority gate logic. Mentor Graphics' AutoLogic II was chosen to accomplish this task. It allows a schematic created with general library components to be mapped to a specific technology provided that a library for that technology exists. The goal was to create such a QCA library with the hope that, once completed, this tool would take as input any conventional schematic, Boolean equation, or VHDL code and generate its minimized equivalent in QCA. Additionally, the possibility of having AutoLogic II perform some initial routing of QCA ”wire”, cells, and gates was considered.

However, as development of the QCA AutoLogic II library continued and an understanding of QCA device physics was enhanced, two extremely important realizations were made. First, a complete set of QCA design rules that were essential for a complete and thorough CAD program had not yet been fully developed. Second, it was discovered that AutoLogic II could not satisfactorily handle several of the QCA design requirements that had been encountered. While AutoLogic II could translate the logic for a QCA circuit design (from CMOS to QCA), making allowances for specific design layout requirements proved to be much more difficult. For this and similar reasons, attention was focused on more hand-crafted designs that would allow QCA design issues to be encountered first-hand and would allow for the development of specific design rules.

As QCA is being investigated as an alternative to CMOS, an ultimate goal should be to build complete microcomputers from QCA cells. With this thought in mind, it was determined that a simple microprocessor should be constructed by hand (in the same manner that the first Intel 8086 processor was constructed). The processor of choice, Simple 12 (see Chapter 2 for more information), was advantageous for multiple reasons. Most importantly, while the processor was simple enough to be designed by hand, it still contained the basic elements that are part of any microprocessor (i.e. arithmetic and logic units, registers, latches, etc.). Hence, solutions to the difficulties encountered and overcome in this design will apply to even more complex systems and processors and will form our desired design rule library.

The design process began by performing a layout of the Simple 12 ALU. The first-cut of this design was completed largely by translating the logic of an existing transistor version of the ALU to an equivalent QCA representation. Problems encountered during this design process were largely related to floorplanning. An extensive study of floorplanning was conducted and several viable floorplans for QCA circuits were developed. Finally, QCA logical circuits were overlaid on floorplans that were designed. While performing these "hand-crafted" designs, a library of design rules was constructed.

Initial designs/layouts were completed in Mentor Graphics' Design Architect using symbols to represent QCA cells. While this was an extremely easy-to-use layout tool, it provided no means for simulating designs for logical correctness. To solve this problem, a tool for laying-out and simulating large QCA designs was written. This simulator allows cells, wires, logical devices, etc. to be placed on a grid like structure to form a specific circuit. Design rules were compiled and form the engine of the simulator which is used to test the circuit for logical correctness.

These design tools were then used to simulate and reanalyze existing design schematics. Not only did this provide a concrete verification of the logical correctness of a Simple 12 dataflow, but it also assisted in determining places for design optimization – particularly with regard to minimizing the longest path/wire. The simulator was also used to design and explore other circuits that would be needed for a complex system such as a microprocessor (i.e. state machines).

1.6 A Thesis Map

Chapter 2 of this thesis will provide the necessary background about QCA physical devices, QCA logical devices, the QCA clock, and the Simple 12 microprocessor. Essentially, it will discuss QCA from a logic designers point of view. Basic devices such as wires and logic gates will be illustrated and explained. Additionally, a basic description of how a single QCA device functions will also be included. Details about the how the QCA clock functions and the Simple 12 microprocessor – the processor for which a dataflow was designed in QCA – will also be included. Chapter 3 will discuss dataflow driven floorplanning for QCA circuits and Chapter 4 will show how the floorplans discussed in Chapter 3 apply to a real design. Chapter 5 will discuss the development of the QCA simulator/layout tool. Design rules will be discussed in detail here. Chapter 6 will provide density comparisons of QCA designs versus CMOS designs and will also address power and clock rate concerns. Finally, Chapter 7 will conclude with a plan for future work.

CHAPTER 2

A BACKGROUND IN QCA DEVICES, THE QCA CLOCK, AND THE SIMPLE 12 MICROPROCESSOR

This chapter will provide the background material needed for a full and complete discussion of the work to be presented in this thesis. It will begin with a discussion of the QCA device. This discussion will then extend to logical circuits that are constructed from the basic QCA device. Then, a discussion on how QCA devices are "clocked" will ensue. Finally, the chapter will conclude with background material for the Simple 12 microprocessor that will be constructed from QCA devices.

2.1 QCA Device Background

QCA cells perform computation by interacting Coulombically with neighboring cells to influence each other's polarization. In the following subsections we review some simple, yet essential, QCA logical devices: a majority gate, QCA "wires", and more complex combinations of QCA cells.

2.1.1 The Basic QCA Device

A high-level diagram of a four-dot QCA cell appears in Figure 2.1. Four quantum dots are positioned to form a square. Quantum dots are small semi-conductor or metal islands with a diameter that is small enough to make their charging energy greater than k_BT (where k_B is Boltzmann's constant and T is the operating temperature). (In the future, they will shrink to regions within specially designed molecules.) If this is the case, they will trap individual charge barriers [11, 12].

Exactly two mobile electrons are loaded in the cell and can move to different quantum dots in the QCA cell by means of electron tunneling. Tunneling paths are represented by the lines connecting the quantum dots in 2.1. Coulombic repulsion will cause the electrons to occupy only the corners of the QCA cell resulting in two specific polarizations (see below). This figure represents places where the electrons are as far as possible from each other without escaping the confines of the cell.

Electron tunneling is assumed to be completely controllable by potential barriers (that would exist underneath the cell) that can be raised and lowered between adjacent QCA cells by means of capacitive plates.

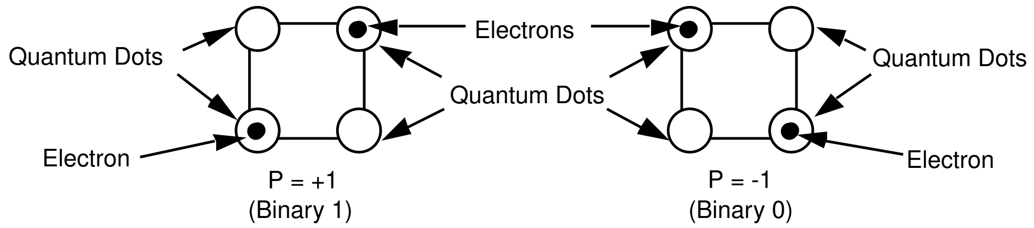

Figure 2.1. QCA cell polarizations and representations of binary 1 and binary 0.

For an isolated cell there are two energetically minimal equivalent arrangements of the two electrons in the QCA cell, denoted cell polarization P = +1 and cell polarization P = -1. Cell polarization P = +1 represents a binary 1 while cell polarization P = -1 represents a binary 0. This concept is also illustrated graphically in Figure 2.1.

It is also worth noting that there is an *unpolarized state* (which will be discussed in later chapters) as well. In an unpolarized state, interdot potential barriers are lowered which reduces the confinement of the electrons on the individual quantum dots. Consequently, the cells exhibit little or no polarization and the two-electron wave functions have delocalized across the cell [8].

2.1.2 The Basic QCA Logical Device – The Majority Gate

The fundamental QCA logical circuit is the three-input majority gate that appears in Figure 2.2 [12]. Computation is performed with the majority gate by driving the device cell (cell 4 in the figure) to its lowest energy state. This happens when it assumes the polarization of the majority of the three input cells. We define an input cell simply as one that is changed by a signal that is propagating in a direction that is toward the device cell. The device cell will always assume the majority polarization because it is this polarization where electron repulsion between the electrons in the three input cells and the device cell will be at a minimum.

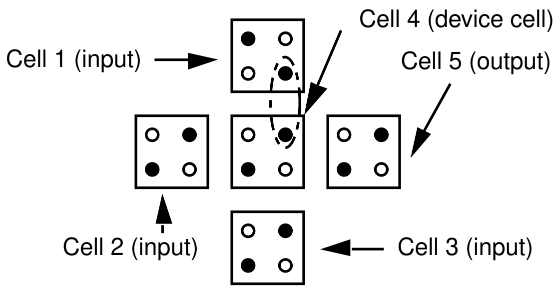

Figure 2.2. The fundamental QCA logical device - the majority gate.

To understand how the device cell reaches its lowest energy state (and hence P=+1 in Figure 2.2), consider the Coulombic interaction between cells 1 and 4, cells 2 and 4, and cells 3 and 4. Coulombic interaction between electrons in cells 1 and 4 would normally result in cell 4 changing its polarization because of electron repulsion (assuming cell 1 is an input cell). However, cells 2 and 3 also influence the polarization of cell 4 and have polarization P=+1. Consequently, because the majority of the cells influencing the device cell have polarization P=+1, it too will also assume this polarization because the forces of Coulombic interaction are stronger for it than for P=-1.

2.1.3 A Straight "90-Degree" QCA Wire

Figure 2.4 illustrates how a binary value propagates down the length of a QCA "wire"[12] . In this figure, the wire is a horizontal row of QCA cells. The binary signal propagates from left-to-right because of the Coulombic interactions between cells. (See Figure 2.3)

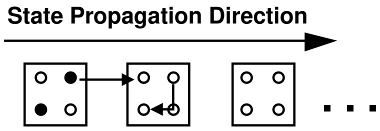

Figure 2.3. Interaction between 2 cells.

In Figure 2.4, cell 1 has polarization P = -1 and cell 2 has polarization P = +1. (Again, we assume that charges in cell 1 are trapped in polarization P= - 1 but those in cells 2-9 are not. Because of this, there is no danger that the wire could "reverse directions" and have a polarization propagate in the direction from which it came). A binary 0 (from polarization P = -1) will propagate down the length of the wire because of the Coulombic interactions between cells. Initially, the electron repulsion caused by Coulombic interaction between cell 1 and cell 2 will cause cell 2 to change polarizations. Then, the electron repulsion between cell 2 and cell 3 will cause cell 3 to change polarizations. This process will continue down the length of the QCA "wire".

2.1.4 A Straight "45-Degree" QCA Wire

A QCA wire can also be comprised of cells oriented at 45-degrees as opposed to the 90-degree orientation discussed above [12]. With the 45-degree orientation, as

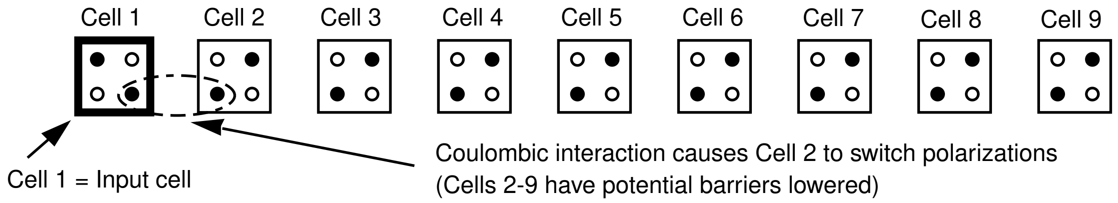

Figure 2.4. A QCA "wire"

the binary value propagates down the length of the wire, it alternates between polarization P = +1 and polarization P = -1. A complemented or uncomplemented value can be ripped off the wire by placing a ripper cell at the proper location and considering the direction of signal propagation (this is explained in detail in the Design Rules section of Chapter 5). The significant advantage of the 45-degree wire is that both a transmitted value and its complement can be obtained from a wire without the use of an explicit inverter! An illustration of a value being transmitted on a 45-degree wire and an example of ripping off a value from that wire appears in Figure 2.5 a and Figure 2.5 b.

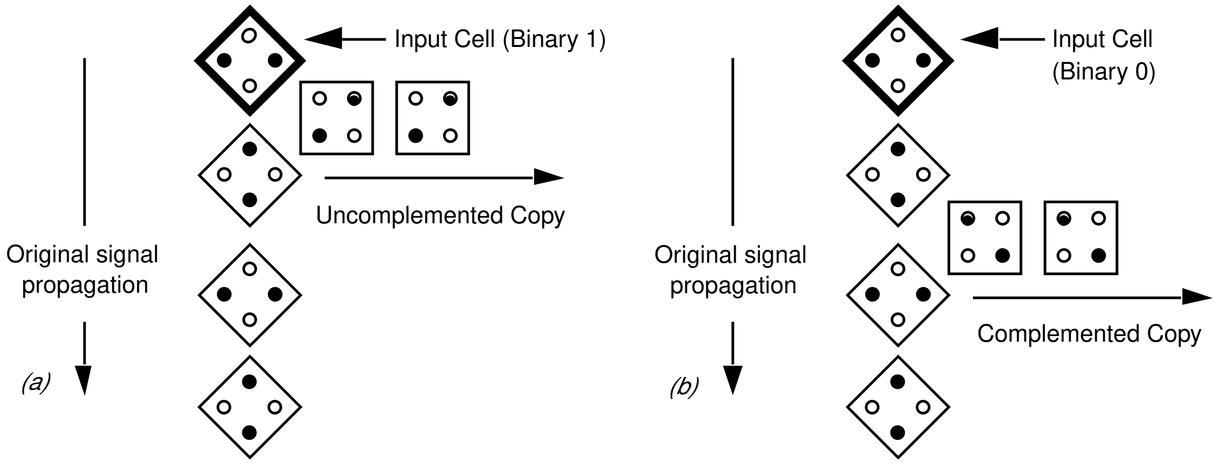

Figure 2.5. (a) Ripping off a Binary 1; (b) Ripping of a Binary 0.

2.1.5 An Off-Center "90-Degree" QCA Wire

Also, QCA cells do not have to be in a perfectly straight line to transmit binary signals correctly. Cells with a 90-degree orientation can be placed next to one another, but off center, and a binary value will still be transmitted successfully as depicted in Figure 2.6 [12].

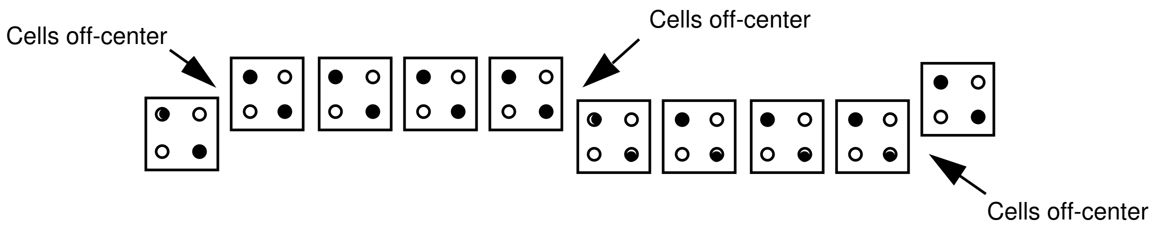

Figure 2.6. A nonrectangular binary wire.

However, there is a restriction on this. Consider the cases illustrated in Figure 2.7:

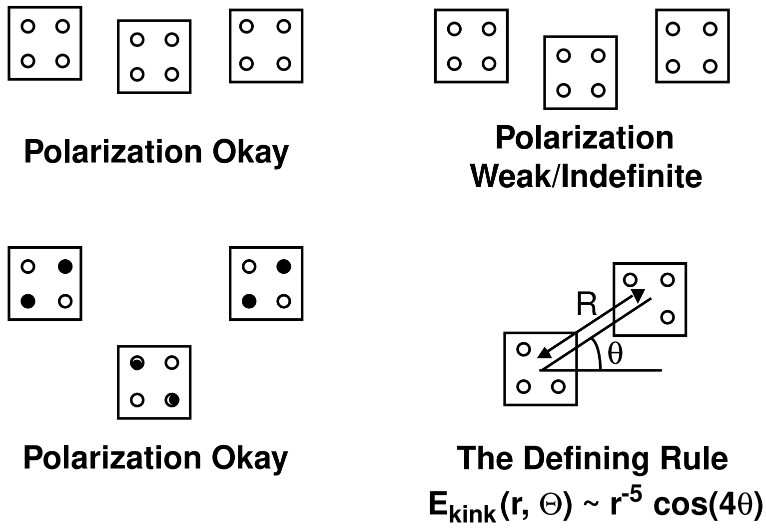

Figure 2.7. Off-center wire issues.

In the first row of this figure, there is off-center 90-degree wire labeled "Polarization Okay" and another labeled "Polarization Weak/Indefinite". If the two quantum dots of the middle cell are below the center lines of its neighboring cells then the polarization will be weak/indefinite. If not, the value will be transmitted successfully.

In the first figure of the second row of Figure 2.7, the "middle" QCA cell is entirely below both "neighboring" cells. In this case, the middle cell's polarization will be different than its two neighbors (thus, it has the function of an inverter).

Finally, the second figure of the second row of Figure 2.7 dictates the amount of "off-centeredness" thought possible. Its behavior is influenced by equation 2.1.

$$E_{kink}(r, \Theta) \simeq \frac{1}{r^5} cos(4\theta) \tag{2.1}$$

E_{kink} refers to the amount of energy that would be required for a successful switch. Thus, it is governed by the distance and angle constraints of equation 2.1.

2.1.6 QCA Wires Crossing in the Plane

Finally, QCA wires possess the unique property that they are able to cross in the plane without the destruction of the value being transmitted on either wire. However, this property holds only if the QCA wires are of different orientations (i.e. one wire is a 45-degree wire and the other is a 90-degree wire) and is shown in Figure 2.8 [12].

2.1.7 A Simple QCA Circuit

To implement more complicated logical functions, a subset of simple logical gates is required. For example, it would be impossible to implement a multiplexor, decoder, or adder in QCA without a logical AND gate, OR gate, or inverter. It has been demonstrated that a value's complement can be obtained simply by ripping it off a 45-degree wire at the proper location. Implementing the logical AND and OR functions is also quite simple.

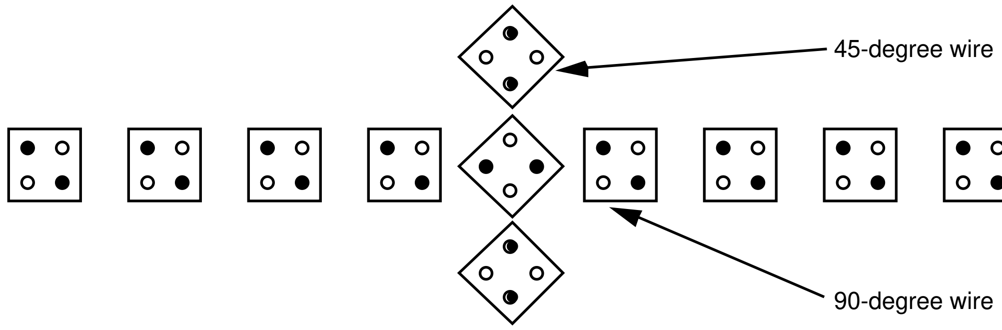

Figure 2.8. Two wires crossing in the plane.

The logical function for the majority gate is:

$$Y = AB + BC + AC \tag{2.2}$$

The AND function can be implemented by setting one value (A, B, or C) in equation 2.2 to a logical 0. Similarly, the OR function can be implemented by setting one value (A, B, or C) in equation 2.2 to a logical 1. This results in the equations:

$$AND = AB + B(0) + A(0) = AB \tag{2.3}$$

$$OR = AB + B(1) + A(1) = A + B \tag{2.4}$$

It is worth noting that because this property exists (i.e. the ability to generate the AND and OR functions) and given the fact that it is possible to obtain the inverse of a signal value, the QCA logic set is *functionally complete* meaning that any logical circuit can be generated with QCA devices.

More complex logical circuits (such as the multiplexor in Figure 2.9) can then be constructed from at least AND and OR gates if not clever combinations of majority gates. QCA cells labeled anchored in Figure 2.9 have their electron polarization frozen to successfully implement AND and OR functions.

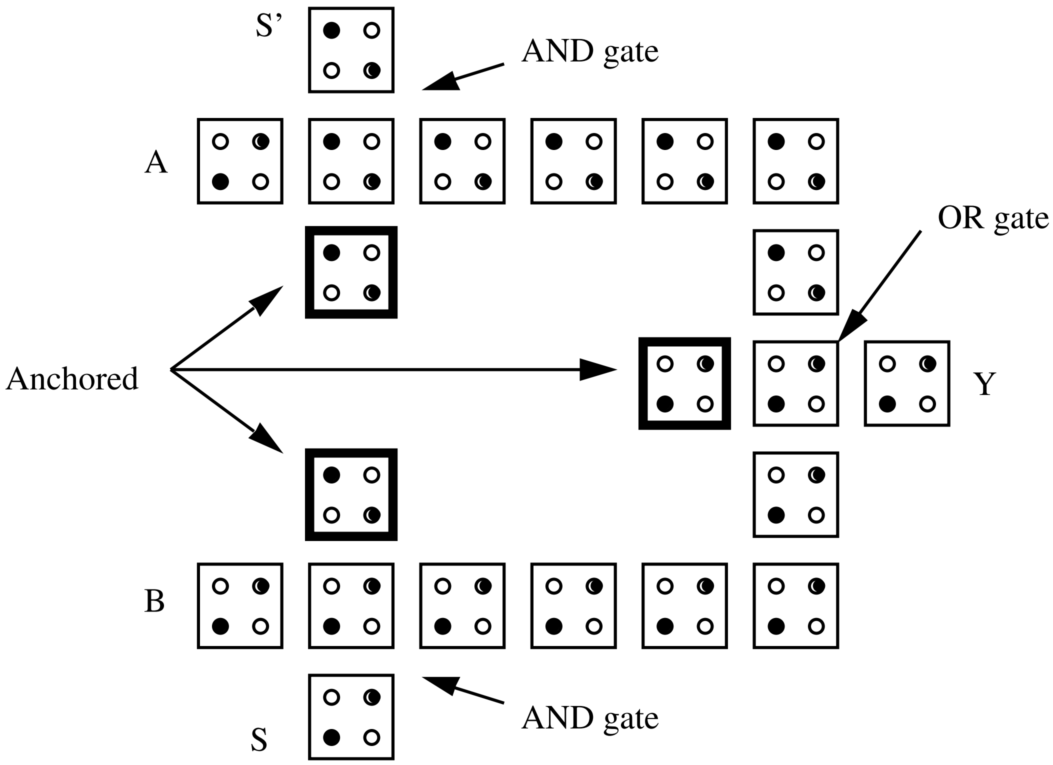

Figure 2.9. A 2x1 QCA multiplexor with logical equation: Y = AS’ + BS.

2.2 The QCA Clock

This subsection will explain and discuss how the QCA clock works. Unlike the standard CMOS clock, the QCA clock has more than a high and a low phase. The phases of the QCA clock and examples are discussed below.

2.2.1 The Basics

The clock in QCA is multi-phased. Individual QCA cells are not timed separately. The wiring required to clock each cell individually could easily overwhelm the simplification won by the inherent local interconnectivity of the QCA architecture [8]. However, an array of QCA cells can be divided into subarrays that offer the advantage of multi-phase clocking and pipelining. For each subarray, a single potential modulates the inter-dot barriers in all of the cells in a given array [8].

This clocking scheme allows one subarray to perform a certain calculation, have its state frozen by the raising of its interdot barriers, and have the output of that subarray act as the input to a successor array (i.e. clocking subarray 1 can act as input to clocking subarray 2). During the calculation phase, the successor array is kept in an unpolarized state so it does not influence the calculation. Each of the four clocking subarrays corresponds to one of four different clocking phases. Neighboring subarrays concurrently receive neighboring clocking phases [8].

Finally, it is important to reiterate and stress what exactly is meant when referring to the QCA "clock". As mentioned above, the QCA clock has more than a high and a low phase. Additionally, it is not a "signal" with four different phases. Rather, it can be said that the clock changes phase when the potential barriers that affect a group of QCA cells (referred to as a *clocking zone*) are raised or lowered or remain raised or lowered (thus accounting for the four clock phases). Furthermore, all of the cells within a clocking zone obviously are in the same phase. It is said that one clock cycle occurs when a given clocking zone cycles through the four different clock phases. What exactly the "clock" does is to trap one set of cells in a specific polarization which in turn allows other cells to make appropriate changes. More will be said about this in the next subsectixon.

2.2.2 How it Actually Works

During the first clock phase, the *switch phase*, QCA cells begin unpolarized and their interdot potential barriers are low. The barriers are then raised during this phase and the QCA cells become polarized according to the state of their driver (i.e. their input cell). It is in this clock phase that the actual computation (or switching) occurs. By the end of this clock phase, barriers are high enough to suppress any electron tunneling and cell states are fixed. During the second clock phase, the *hold* *phase*, barriers are held high so the outputs of the subarray can be used as inputs to the next stage. In the third clock phase, the *release phase*, barriers are lowered and cells are allowed to relax to an unpolarized state. Finally, during the fourth clock phase, the *relax phase*, cell barriers remain lowered and cells remain in an unpolarized state [8]. The four clock phases are illustrated in two different ways in Figure 2.10 and Figure 2.11 while an example of a value being transmitted on a QCA wire is illustrated in Figure 2.12.

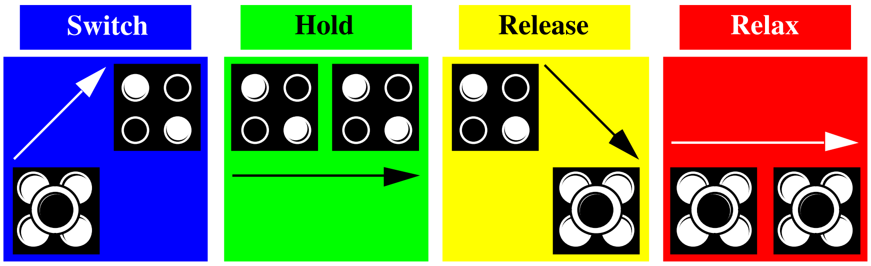

Figure 2.10. The four phases of the QCA clock.

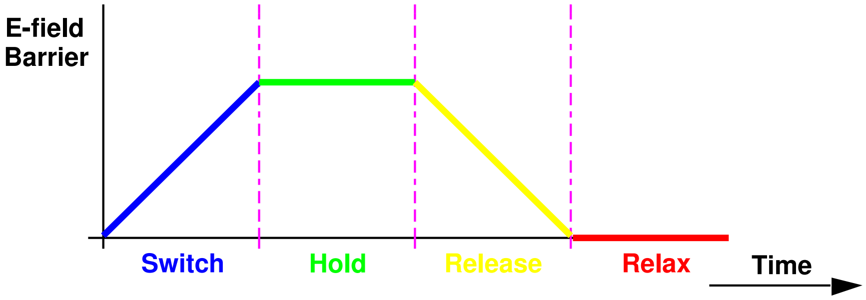

Figure 2.11. The four phases of the QCA clock (an alternative expression).

Figure 2.12 represents a 5 cell segment of QCA wire with each region representing a cell. Figure 2.12 essentially has four significant parts to it. First, the figure is divided into 5 vertically shaded regions with the label "clocking zone x" appearing in each region. Second, essentially 5 representations of the horizontal QCA wire are illustrated in Figure 2.12 and the state of the wire is shown at 5 different time

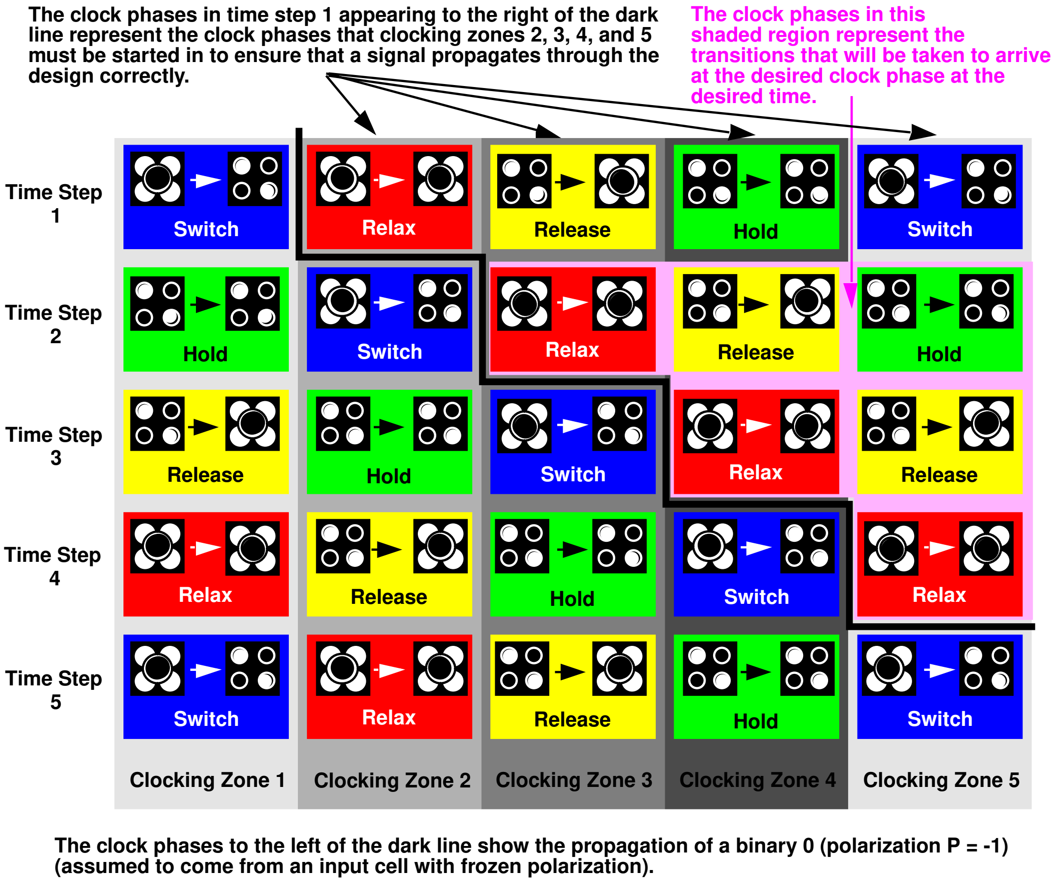

Figure 2.12. An example of QCA clock transitions.

steps. Third, the state transitions for cells that make up the wire are illustrated for each time step and are based on what clocking zone the particular cell is a part of. Fourth, this figure is divided into 2 parts by a thick black line. Only cells to the left of the black line will have a meaningful change of state during a given time step. Nevertheless, cells to the right of the black line still "exist" as they are part of the wire.

They also illustrate that clocking zones must be "initialized". What is meant by this? Clocking zones must traverse through the four phases as follows. From switch, the zone transitions to hold. From hold, the zone transitions to release. From release, the zone transitions to relax. Finally, from relax, the zone transitions back to switch. Such a transition order is important because if cells in one clocking zone are in the hold phase, cells in an adjacent zone should be in the switch phase – with the cells in the clocking zone that is in the hold phase acting as inputs to cells in the clocking zone that is in the switch phase. In Figure 2.12, in time step 2, this in fact the situation. However, to ensure that during time step 2, the cells in clocking zone 2 are in the switch phase, it must be started in the relax phase. Thus, when the zones change phases after the first time step, zone 1 will go to hold while zone 2 will go to switch.

Assuming that there is a frozen input cell with polarization P=+1 (binary 1) to the left of this wire, a value would propagate down the length of a wire as follows: Cells immediately to the left of the input cell (clocking zone 1) would begin in the switch phase (in time step 1). As mentioned earlier, in the switch phase, the potential barriers for the zone would be low. During this phase, they would be raised and the cells would become polarized according to the state of their driver (in this case, the input cell with polarization P=+1).

In time step 2, clocking zone 1 would transition to the hold phase while clocking zone 2 would transition to the switch phase. The barriers of clocking zone 1 are held high and cell polarizations and states are frozen. Clocking zone 1 serves as the input to clocking zone 2 (in the switch phase) and the cells in clocking zone 2 are polarized according to the states of the cells in clocking zone 1.

In time step 3, clocking zone 1 would transition to the release phase, clocking zone 2 to the hold phase, and clocking zone 3 to the switch phase. Clocking zones 2 and 3 would interact in the exact same manner in time step 3 that clocking zones 1 and 2 did in time step 2. However, in time step 3, the cells in clocking zone 1 will enter the release phase. Here, potential barriers are lowered and the cells are allowed to relax to an unpolarized and neutral state. This is done so that cells in clocking zone 1 will be allowed to obtain a new value for transmission on the "wire".

In time step 4, clocking zone 1 would transition to the relax phase, clocking zone 2 to the release phase, clocking zone 3 to the hold phase, and clocking zone 4 to the switch phase.

The effects of the release, hold, and switch phases have been explained in detail for previous time steps. However, the purpose of the relax phase warrants some further commentary. It would appear that this clock phase is not really necessary. Why not simply proceed from release back to switch? After all, the cells in the release phase have been unpolarized and have no state. The relax phase is necessary because, as mentioned earlier, this clock phase sequencing is done so that the subarray does not influence the next calculation (i.e. a switch clocking phase follows the relaxed clocking phase but if a switch clocking phase were to directly follow a release clocking phase, the switched clocking phase could affect the QCA polarizations of the release clocking phase).

In time step 5, clocking zone 1 returns to the first clock phase (switch) and repolarizes. A new value could now be transmitted down this QCA wire. The other clocking zones make the usual transitions discussed above.

At this point and time it is worth mentioning that there is some inherent pipelining built into the QCA technology. After every 4 time steps, it is possible to put a new value onto a QCA wire.

2.3 Simple 12

While there is still much work to be done, early results indicate that QCA is a very viable alternative to CMOS. QCA cells and a QCA majority gate have been fabricated and tested successfully. However, the actual design of many of the circuits and devices required for a QCA microprocessor have not yet even been considered. What is required is a methodology for constructing and designing the QCA circuits that are essential for a design such as a microprocessor. Furthermore, understanding how circuits are built should assist in the actual design of the devices themselves. It will also serve to open a discussion about architectural issues of QCA and other nanotechnologies.

As a means for generating the QCA architecture an obvious first step is to translate existing CMOS designs directly into QCA majority gate logic. However, while such a translation is possible, the nature of QCA devices will require an architecture that is radically different from conventional CMOS.

The inherent pipelining associated with QCA and the logical device and clocking methodology discussed above are only several of the ways that QCA designs differ from conventional CMOS designs. To develop a library of design rules and hence the QCA architecture, we are designing and simulating a custom design of a microprocessor called Simple 12 entirely in QCA. The advantages of choosing Simple 12 are three fold. First, the processor IS simple. Simple 12 has 12-bit data words, an 8-bit addressable memory, and uses minimal hardware. Consequently, much of the physical layout can be performed by hand. Second, an actual processor will be designed with an instruction set that includes arithmetic instructions, loads, stores, and jumps. Therefore, solutions to the difficulties encountered in this design will apply to even more complex systems of custom and synthesized logic. Third, we have completed and fabricated a two micron CMOS Simple 12. Thus, it will be possible to make comparisons to an existing design in a technology on which we are trying to improve.

2.3.1 The Simple 12 Dataflow

A high-level block diagram of Simple 12 appears in Figure 2.13. Again, although simple, it exhibits almost all of the major attributes of a more complex design. As mentioned, the design includes three registers, address and data buses, feedback paths, and a memory interface.

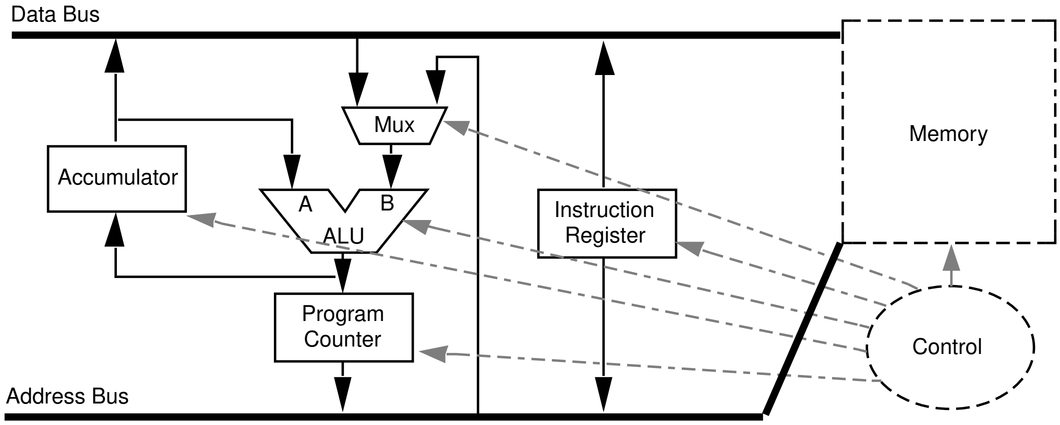

Figure 2.13. The Simple 12 datapath.

A sample instruction might be executed as follows: The program counter (PC) supplies the address of the instruction to memory. While the instruction is being fetched, the PC can be incremented by 1 using the sole ALU. Next, using data from the Accumulator and the Data Bus (which could have data from the Instruction Register for instance), the ALU will perform an operation and store the result in the proper location (i.e. Accumulator, memory, etc.).

2.3.2 The Simple 12 Instruction Set

Simple 12 has 4 basic classes of instructions. The Jump class will change the value of PC. Memory access class instructions will load and store information from memory. Operand class instructions execute logical and arithmetic operations. Finally, the reserved class of instructions consists of opcodes not used in the original design. The Simple 12 Instruction set appears in Table 2.1.

Table 2.1. The Simple 12 Instruction Set.

Opcode	Mnemonic	Register Transfer Language
0000	JMP X	PC <– X
0001	JN X	if $A < 0$ then PC <– X else PC++
0010	JZ X	if $A = 0$ then PC <– X else PC++
0100	LOAD X	A <– M(X) || PC++
0101	STORE X	M(X) <– A || PC++
1000	AND X	A <– A AND M(X) || PC++
1001	OR X	A <– A OR M(X) || PC++
1010	ADD X	A <– A + M(X) || PC++
1011	SUB X	A <– A − M(X) || PC++

2.3.3 Functions of the Simple 12 ALU

To successfully execute the above instructions, the Simple 12 ALU must be able to generate the following outputs (where A and B are inputs into the ALU): A+B, A-B, A AND B, A OR B, B, B+1, and 0. To generate these outputs several control signals are needed that serve as inputs to the intermediate signal generation logic of the ALU. They are summarized below.

The *ZeroA* signal is used to perform the B+1 operation. Specifically, the A input must be set to a logical 0. If this is desired, this signal is set low and ANDed with the A input. In all other cases, the signal should be a logical 1/high.

Logic/Adder is used to control the multiplexor that selects between the output of the arithmetic unit and the logic unit.

B-Invert is used to perform the A-B operation. Specifically, the signal is used to generate the inverse of B if it is required so that twos complement addition can be performed. It is the input to an XOR gate along with B. This signal also controls the multiplexor of the logic unit to select between A AND B and A OR B.

CHAPTER 3

DATAFLOW DRIVEN CLOCKING FLOORPLANS

This chapter will discuss floorplanning. In particular, it will answer the question of how one arranges QCA cells to perform logical and useful computation within the constaints of clocking zones and the QCA clock. First-cut designs and floorplans will be illustrated and discussed. Problems that exist within first-cut designs will be identified and solutions will be proposed.

3.1 A First-Cut of the Simple 12 ALU

The QCA ALU was largely designed by translating the logic of the transistor version of the ALU to an equivalent QCA representation. Essentially, equivalent QCA majority gate representations of the transistor logic were determined and implemented. The only difference between the QCA ALU and the transistor ALU is that the QCA ALU did not use a mirror adder, but the full adder designed by Lent, et al [8] (the CMOS design used a full adder).

The design of the ALU can essentially be broken down into three blocks. One block represents the adder, another block represents the portion of the ALU that performs operations such as AND and OR (the logic unit), and the last block contains logic for intermediate ALU signal generation. Each block will be discussed in a separate subsection below.

3.1.1 The Adder Unit

As was mentioned above, the QCA adder does not use the mirror adder that was included in the Simple 12 CMOS design, but rather a full adder designed by Lent, et al [8]. A majority gate schematic of the full adder appears in Figure 3.1. It can easily be seen that by using majority gates, the adder that is produced is significantly different from a "normal" or conventional full adder. This majority gate single-bit full adder (first-cut) requires five majority gates and three inverters and appears in Figure 3.2. As can be seen in Figure 3.2, the data lines for inputs A, B, and C are 45-degree wires. Thus, the need for explicit inverters is eliminated (note: values are not necessarily ripped off in the correct places from these 45-degree wires. These design rules were not determined until later and were in fact determined from this first cut design. If a values complement must be ripped off, it is indicated by the use of an inverter. If the original value is desired, a buffer symbol is used.) The 5 majority gates are marked with dots in Figure 3.2.

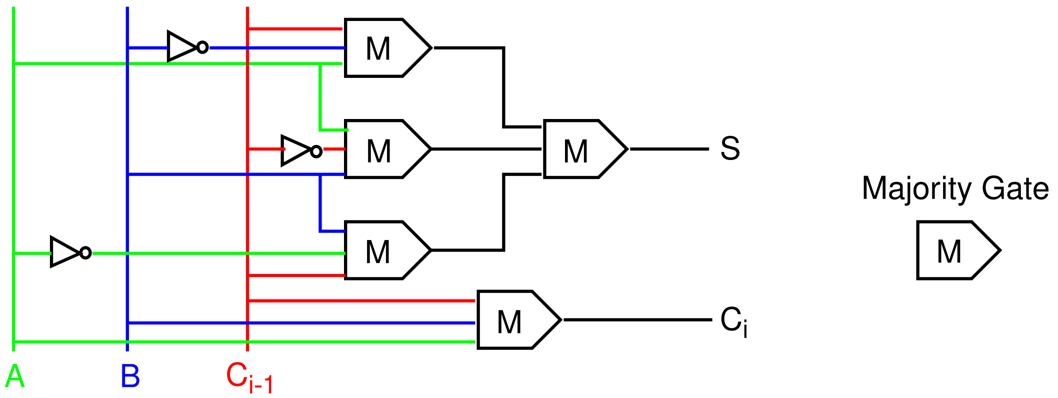

Figure 3.1. A block diagram of the adder used in the QCA Simple 12 ALU.

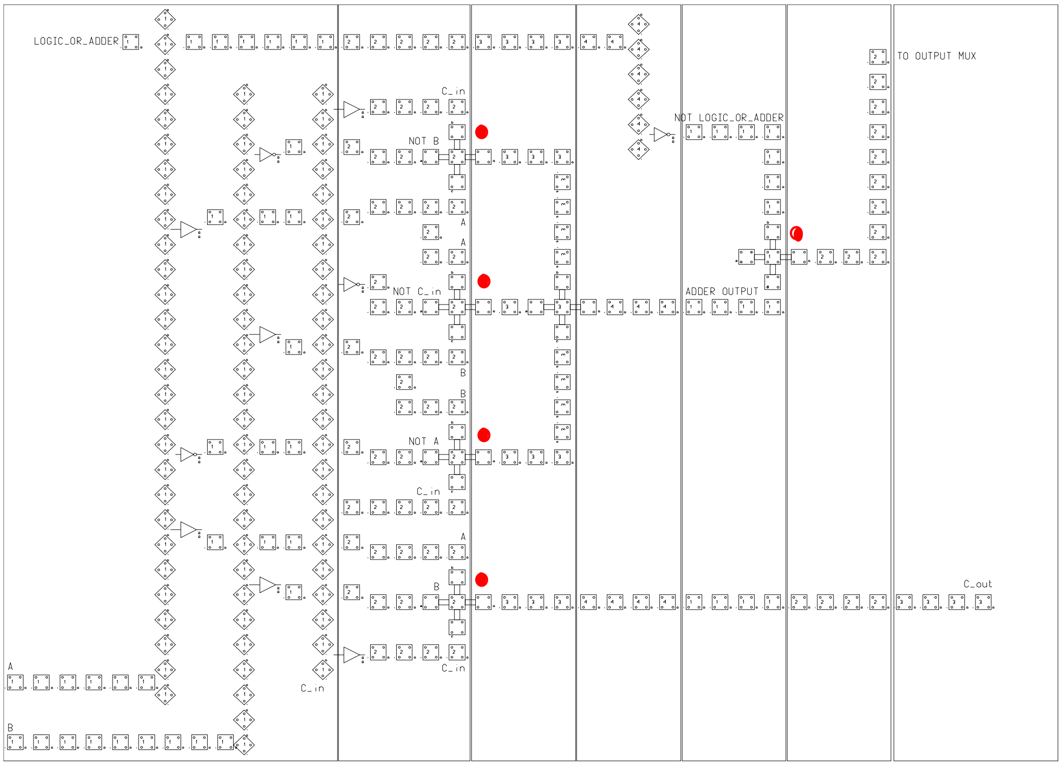

Figure 3.2. A first-cut of the adder for the QCA Simple 12 ALU.

3.1.2 The Logic Unit

To successfully execute the complete Simple 12 instruction set, the ALU must be able to generate the following outputs: A+B, A-B, A AND B, A OR B, B, B+1, and 0. The logic unit of the ALU will generate the outputs: A AND B, A OR B, B, and 0. (The output of the logic unit is then multiplexed with the output of the adder unit and one output from the ALU is generated). The logic unit consists only of a majority gate with an input cell anchored so that it performs the AND operation, a majority gate with an input cell anchored so that it performs the OR operation, and a 2x1 multiplexor to select between the output of the AND and OR gate. A first-cut of the schematic of the logic unit for the QCA Simple 12 ALU appears in Figure 3.3.

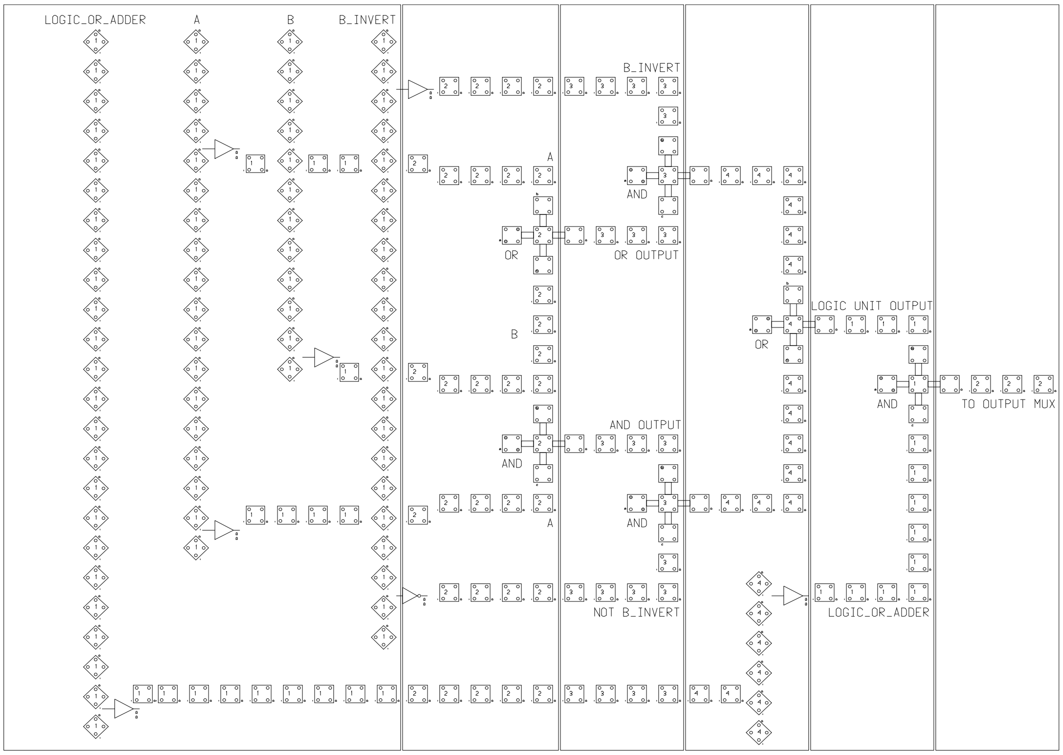

Figure 3.3. A first-cut of the logic unit for the QCA Simple 12 ALU.

3.1.3 The Intermediate Signal Generation Unit

In Section 3.1.2, it was indicated that the logic unit had to generate the following outputs: A AND B, A OR B, B, and 0. One mechanism for generating B would be to OR every bit of B with a logical 0. However, to perform this operation, the other input to the logic unit, A, must be set to 0. In this case, the intermediate signal generation logic will perform such an operation by setting the ZeroA signal low and ANDing it with the A input. Thus, the new "A" input will automatically be a 0. Similarly, a method for generating the 0 output would be to AND any B input with a logical 0. The A input can be zeroed as mentioned above and ANDed with B generating a 0 for any B input. Finally, it should be mentioned that the intermediate signal generation unit is also used to assist with adder operations – particularly A-B.

This unit will generate the complement of B if a subtraction operation is desired by XORing B with a control signal B-Invert. This will allow twos complement addition to be performed. The intermediate signal generation schematic (first-cut) for the QCA Simple 12 ALU appears in Figure 3.4.

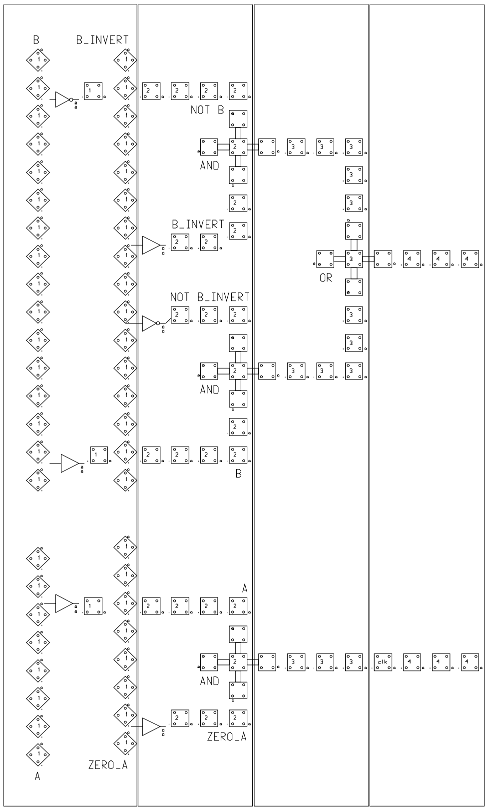

Figure 3.4. A first-cut of the intermediate signal generation unit for the QCA Simple 12 ALU.

3.1.4 The Final Product

In Figure 3.5, the 3 parts of the Simple 12 ALU – adder, logic unit, and intermediate signal generation logic – are joined to form the first-cut of the Simple 12 ALU. Problems that exist with this design will be discussed in the next subsection and solutions to such problems will be discussed in the subsection after that.

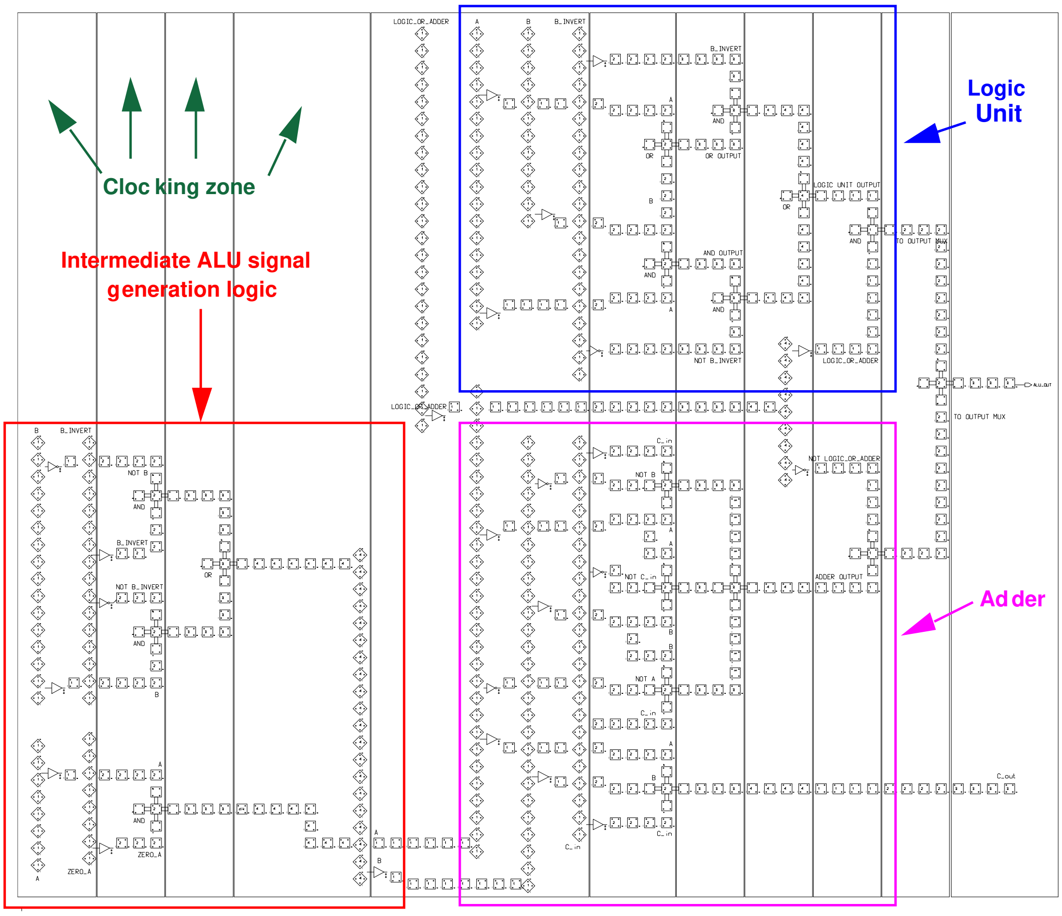

Figure 3.5. 1st cut of the QCA Simple 12 ALU.

3.2 Problems with the First-Cut of the Simple 12 ALU

The first-cut of the Simple 12 ALU has 5 significant problems. First, wire lengths vary from extremely short (i.e. 4 QCA cells in length) to extremely long (i.e. 36 QCA cells in length). Second, the clocking zones in this first-cut have non-uniform widths. Third, some clocking zones contain a very large number of QCA cells while others only contain a few. Fourth, there is no means for generating feedback in this design. Fifth, and finally, there is a large amount of white-space/wasted area in this design. Why the above 5 characteristics are problems will be discussed in the subsections below. Additionally, the problems are illustrated in Figure 3.6.

These problems (and more importantly solutions to them) must be considered when designing future circuits.

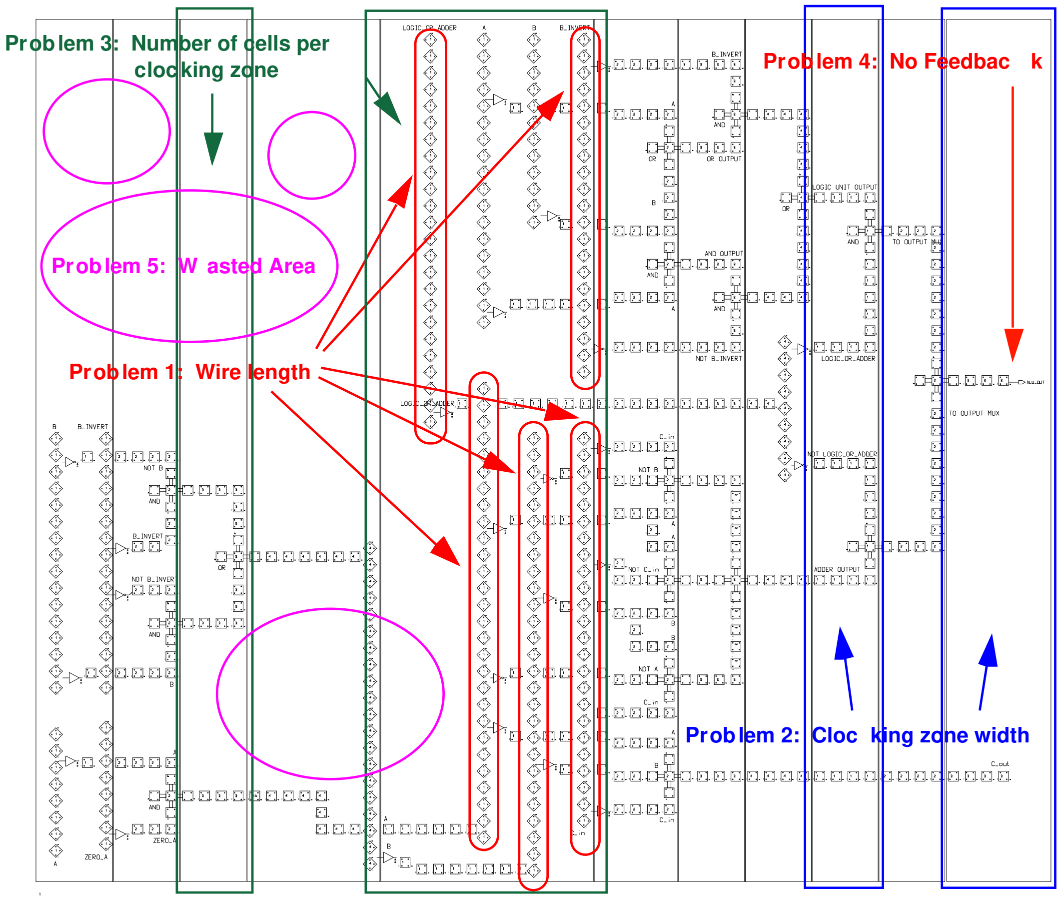

Figure 3.6. 1st cut of the QCA Simple 12 ALU.

3.2.1 Wire Length

When generating designs in QCA, a significant effort should be made to keep the length of a wire within a given clocking zone to a minimum. There are two very important reasons to do this. First, as wire length grows, the probability that a QCA cell will switch successfully decreases in proportion to the distance a particular cell is from a frozen input at the beginning of the "wire" [8]. Consequently, for shorter wires, there is a higher probability that all cells making up the wire will switch successfully. Additionally, wire length will determine the clock rate – or in other words, the rate at which clocking zones can change clock phases. This is so because, before a given zone can change phase, every cell within the zone must make appropriate polarization changes. Obviously, the longer the wire, the longer the time for a signal to propagate down the length of it.

3.2.2 Clocking Zone Width

Like wire length, there are also two important reasons for keeping clocking zone width to a minimum. The first reason centers around the desire to keep wire lengths at a minimum. If clocking zone widths are narrow, it will force the designer to keep wire lengths small (at least in one dimension). For example, in most of the clocking zones in Figure 3.6, a horizontal wire is composed of no more than 5 cells. A second reason centers around uniformity. A conscious effort has been made to make circuits that have been designed in QCA as uniform as possible. This was done to hopefully increase the manufacturability of the circuits and designs if and when that time comes.

3.2.3 Number of QCA Cells per Clocking Phase

As can easily be seen in Figure 3.6, there is a large disparity between the number of QCA cells in some clocking zones when compared to the number of QCA cells in other clocking zones (i.e. in the clocking zone at the far right of Figure 3.6 there are only 8 cells in the zone while in the clocking zone in the middle of Figure 3.6 there are 211 in the zone).

If too many cells are included in a single clocking zone, the clock rate could deteriorate (simply because the time for all cells to make there required transitions will most likely increase). However, more importantly, for arrays of cells on the order of 10^3 (i.e. 10^3 cells per clocking zone), there will be a tendency for the system to settle into an excited state rather than a ground state – and a cell is in a ground state when it has a definitive polarization, and hence logical value. This occurs because of thermodynamic effects.

It is worthwhile to include a short discussion of what exactly *thermodynamic effects* are. Such effects were first described by Lent, et al. If thermal fluctuations excite an array of cells above its ground state, i.e. so that the cell does not have a definite polarization, wrong answers can appear at the outputs of a circuit. To be robust, the excitation energy must be well above $k_B T$. It can be determined from calculations that a maximum operating temperature for cells depends in part on the size of a cell. As cell sizes decrease, the energy separations between states increase and higher temperature operation is possible [8]

Additionally, consider a linear array of N cells acting as a wire transmitting a logical 1. The ground state for such a configuration would be all of the cells obtaining the same polarization as that of the input (or driving cell). The first excited state of this array will consist of the first m cells polarized in a representative binary 1 state and N-m cells in the binary 0 state. This excitation energy of this state (E_k) is the energy of introducing a "kink" in the polarization. The energy is independent of where the kink occurs (i.e. the exact value of m). As the array N becomes larger, the kink energy E_k remains the same. But, the entropy of this excited state increases (as there are more ways to make a "mistake" in a larger array). When the array size reaches a certain size, the free energy of the mistake state becomes lower than the free energy of the correct state. A complete analysis reveals that the maximum number of cells in a single array is given by $e^{E_k/k_B T}$. This again requires the excitation energies to be significantly larger than $k_B T$. Finally, the kink energy increases as the system is scaled to smaller sizes [8].

3.2.4 Lack of Feedback

A significant "logical" problem with the design appearing in Figure 3.6 is the complete absence of physical feedback – namely, a value generated as output from the circuit has no means for traveling back to the input. Physical feedback is all but essential in most useful microprocessors and finite state machines. (As a simple example, consider a register. Any processor should be capable of writing to a register and using a register as input. Thus, a path should exist from the output of a dataflow, to a register, and back to the input of that dataflow. Without feedback, this would be impossible.) Furthermore, as seen in Figure 9, the Simple 12 dataflow requires some form of physical feedback. However, in Figure 3.6 data flows only in one direction. Some method of allowing the output of a given circuit to be used again at the input must be generated.

3.2.5 Wasted Area

A final problem with the design appearing in Figure 3.6 is that it is not very space efficient (examples of wasted area are illustrated with circles). A significant cause of wasted space in this design comes as a result of the intermediate signal generation logic. Both the logic and adder unit require the A and B inputs to be changed to perform certain operations. Consequently, the intermediate signals must be generated before inputs reach the logic or adder unit. In the first-cut design, data flows only in one direction (to the right). Therefore, the intermediate signal generation logic must precede the logic and adder units. However, despite its importance in precedence, the intermediate signal generation logic is actually two very simple circuits. As a result, not much area is required. Still, because it comes before the other two units, space is wasted.

3.3 Floorplanning

This subsection will discuss methods for solving the 5 problems discussed in the previous subsection. Many of the solutions stem from a specific arrangement of clocking zones onto which a QCA circuit is overlayed. For this reason, this section is entitled "Floorplanning".

3.3.1 Trapezoidal Clocking

As discussed in section 3.2.5, a significant problem with the first-cut design is wasted area in the design. In particular, this wasted area comes from logic required to generate intermediate ALU data. To remedy this particular problem, the new technique of "trapezoidal clocking" will be introduced. In Figure 3.7, QCA logic to generate intermediate ALU data is not placed in front of the computational logic but rather, below it. Instead of leaving large gaps – or areas with no logic (like those appearing in Figure 3.6), "trapezoids" containing computational logic and intermediate signal generation logic can be fit together to minimize wasted area. Thus, data will flow in two different directions. It should be noted that the dotted lines in Figure 3.7 represent clocking zone boundaries. Thus, the computational and intermediate signal generation logic would still be divided up into clocking zones as depicted in Figure 3.6.

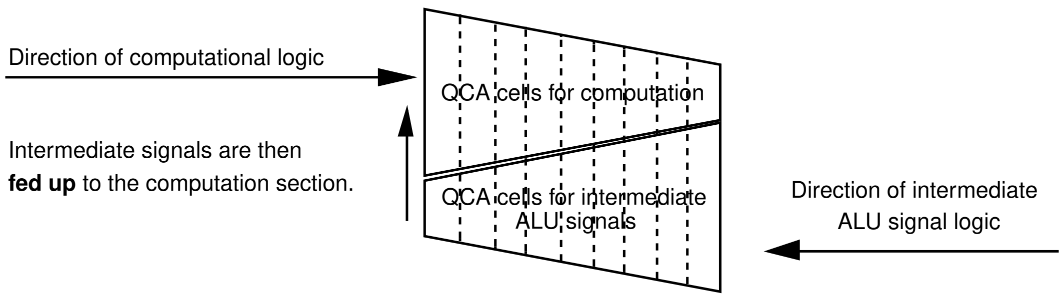

Figure 3.7. A description of trapezoidal clocking.

It is also worth noting that QCA inherently lends itself to such a ”trapezoidal” structure. In QCA circuits, an output is usually generated from a single gate or wire. If it is a logical gate from which the output comes, that gate usually performed a computation by using inputs generated from 2 or 3 other logical gates. This process most likely continues ”backwards” until some inputs are reached. In this way, it is not unlike a ”tournament bracket” in which there are an initial n slots and after a certain number of m stages, a single slot remains. As illustrated in Figure 3.8 (and in the first cut designs appearing in the figures of section 3.2), QCA circuits have a very similar form. By allowing data to flow in two directions (as shown in Figure 3.7) and by carefully fitting ”trapezoids” together it would seem very possible that very dense and compact QCA circuits could be generated.

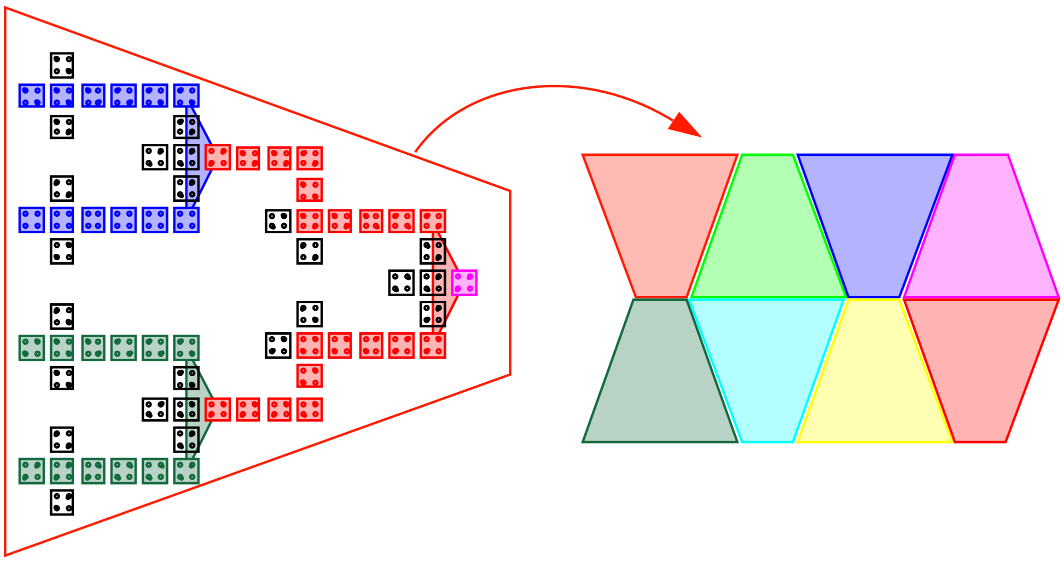

Figure 3.8. A QCA ”tournament bracket” and potential for very dense circuits.

3.3.2 Feedback and Trapezoidal Clocking

Trapezoidal clocking does not only provide a means for minimizing total area. It can also be used to implement a feedback path in QCA circuits. In Figure 3.9, the four clocking phases are each given a number (1, 2, 3, and 4) and a color shade. These correspond to the four different clock phases that were discussed in Section 2.2.2. and illustrated in Figures 7 and 8. If the top "trapezoid" is computational logic, data can be fed back to the input (assumed to be in clocking zone 1 at the far left) after "switching" in clocking zone 1 at the far right. White arrows illustrate the feedback path through the numbered clocking zones. It can easily be seen that the clocking phases are traversed in the proper order (i.e. in the order 1, 2, 3, 4 – and so that the required clock phases are always adjacent to one another to allow for correct signal propagation). Furthermore, a signal can start at a given point and a path exists to return to that point – the definition of feedback.

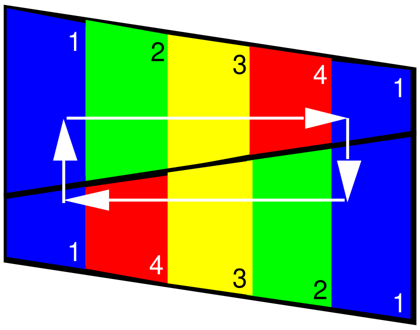

Figure 3.9. A trapezoidal clocking floorplan with clocking zones.

3.3.3 A Universal Clocking Cell

The next question to be asked is whether or not the clocking zone arrangement illustrated in Figure 3.9 can be extended to allow efficient and easy wire routing. Thus, can the clocking zones be arranged or tiled so that there are multiple "wire" loops and "wire" crossings and still allow feedback? Such a floorplan is necessary because for designs and components (such as the QCA Simple 12 ALU) multiple control and data input and output wires will have to be included in the design. The ALU also requires some feedback mechanism. Furthermore, a standardized clocking floorplan would provide a start for a new design, as a way to run wire and generate feedback would already exist. Such a pattern is possible and is illustrated in Figure 3.10. As seen in Figure 3.10, several different loops can be generated that cross (recall that 45 and 90-degree QCA wires can cross in the plane with no interference of either value being transmitted on either wire) and do not violate the condition that the clocking zones must be traversed in the order 1, 2, 3, 4.

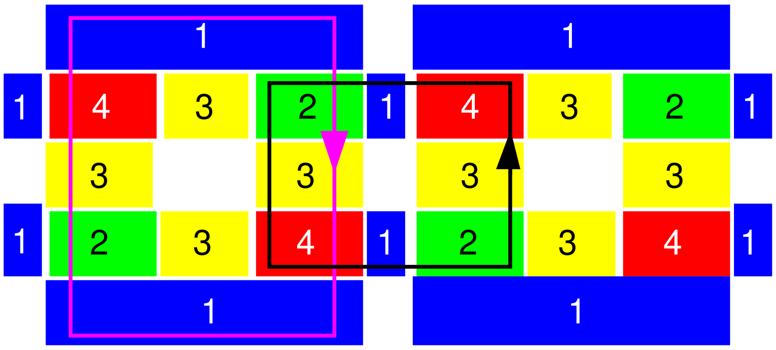

Figure 3.10. The universal clocking cell.

3.3.4 Universal Clocking Floorplans and Data and Control Routing

The pattern that appears in Figure 3.10 can be tiled to form a universal clocking floorplan (UCF). The UCF allows large designs to be constructed and allows extremely complicated paths of QCA wires to be constructed without violating the condition that the clocking zones must be traversed in the order 1, 2, 3, 4 (as illustrated in Figure 3.11). Also, some potential and feasible wire paths are also included. (Note: the pattern in Figure 3.10 is rotated 90 degrees in Figure 3.11).

The UCF solves the problem of physical feedback in QCA circuitry for large designs. Also, it provides a mechanism for tracing complicated paths while still traversing the clocking zones in the proper order. Furthermore, it provides an extremely efficient manner to run data signals and control signals (Figure 3.12).

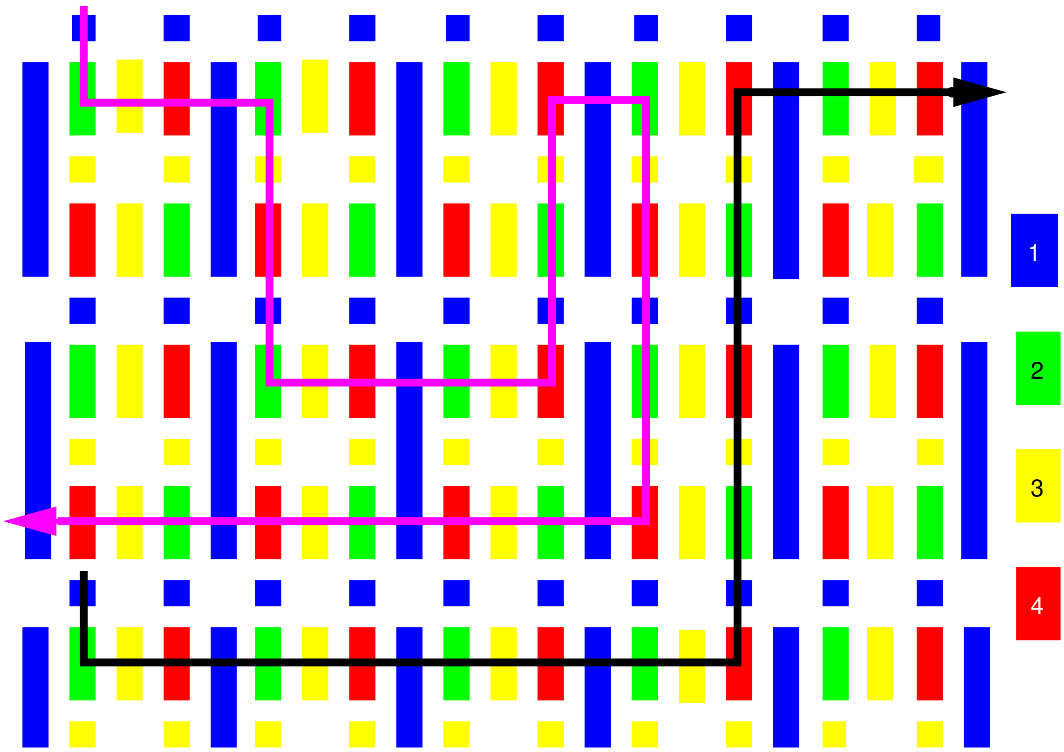

Figure 3.11. The universal clocking floorplan.

Data signals can be run horizontally while control signals are run vertically (or vice versa). This is directly analogous to CMOS VLSI where one layer of metal is run in the vertical direction and used to transmit data signals or control signals, while another layer of metal is run perpendicular to the first and used to transmit data or control signals.

3.4 A Few Final Floorplanning Comments

It should be noted that in most cases, the exact universal floorplan appearing in Figure 3.11f will not be used for every design. Specific circuits may require slight variations of it (i.e. slightly wider or taller clocking zones, etc.). However, what the UCF does provide is a means for starting any design. It also provides fundamental

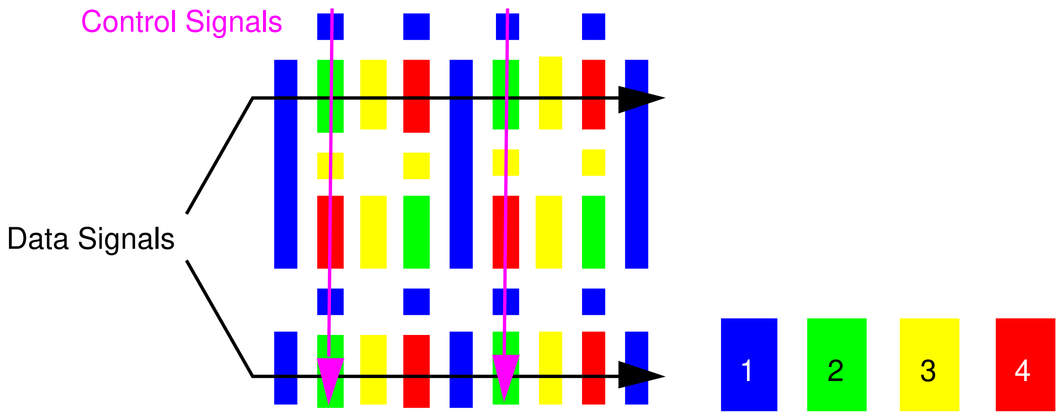

Figure 3.12. A Universal Clocking Floorplan with data and control signal routing.

mechanisms for routing control and data signals as well as a means for generating feedback.

CHAPTER 4

ACTUAL DESIGNS

This chapter will begin by discussing the floorplanning techniques developed to solve the design problems discussed in Chapter 3 – particularly, how they effect "real designs". While the "second-cut" of the QCA Simple 12 dataflow has now undergone some significant optimizations and revisions, feedback is still not present in the design. While methods have been discussed in Chapter 3 for generating feedback in QCA circuits, no circuits with feedback have yet been described. Several circuits involving feedback will be discussed and described in this chapter including simple wires, latches, and registers. After concepts needed for QCA registers and latches have been introduced, specific latch/register requirements for the QCA Simple 12 will be mentioned. These latches/registers will then be integrated into the overall design for the existing QCA Simple 12 dataflow. Finally, a brief discussion of state machines and state machine logic will be presented – particularly as to how it relates to the idea of the control unit for the QCA Simple 12.

4.1 "Second-Cut" Designs

Using floorplanning techniques developed and discussed in Chapter 3, the first-cut design of the QCA Simple 12 ALU was reworked to solve the problems of long wire length, inconsistent clocking zone width, the large disparity of QCA cells in wires across some clocking zones, the lack of physical feedback, and wasted area in the design. The design first created to address some of these problems appears in Figure 4.1. As one can see in the figure, this "second-cut" design does not address all of the problems listed above. For instance, there is still a lack of physical feedback and wasted area in the design. Additionally, there are also still a few cases of "long" wires. However, the clocking zone widths have been made more uniform, the number of cells in a given clocking zone have been reduced, and wire lengths have been shortened.

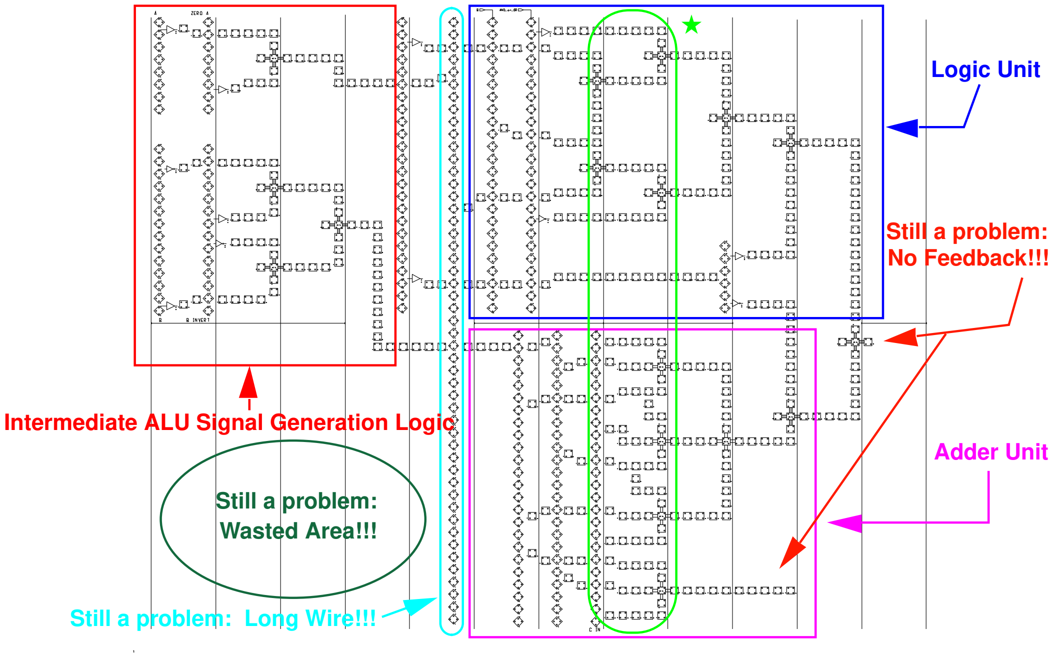

Figure 4.1. A "second-cut" design of the QCA Simple 12 ALU.

It is worth mentioning that one potential solution employed in this "second-cut" design to help reduce the number of cells per clocking zone really has nothing to do with innovative floorplanning (i.e. arranging/tiling clocking zones in a specific order). In this "second-cut" design, some clocking zones are simply divided in half to reduce the number of cells in the given zone. Adjacent zones can still have the same phase but are reduced in size to reduce the thermodynamic effects of having too many cells in a given zone (see $\star$). (It should be noted that future designs do not specifically employ this feature as the number of cells in our design are not numerous enough for thermodynamic effects to be a concern.)

As illustrated in Figure 4.1, three significant problems still exist. In an attempt to solve the remaining problems, another design was generated and appears in Figure 4.2. As seen in Figure 4.2, the problems of wasted area and long wire lengths have all but been eliminated. The method used to eliminate wasted area was to simply duplicate portions of the intermediate signal generation logic – in particular the logic needed to zero the A input. This logic was simply placed in front of the adder unit and the logic unit. As this logic only involves an AND gate, the additional QCA cells required were minimal. Interestingly, this duplication of logic actually solves two problems at the same time. The duplication of logic eliminates the need for long "routing wires" to carry signals to various portions of the ALU. This not only saves area, but also reduces wire length!

It should be mentioned that there is no need to duplicate the logic that will change the B input signal for various ALU operations. The B signal only needs to be altered if the output from the adder unit is desired. Consequently, the "normal" B input can be fed to the logic unit.

4.2 Feedback and its Applications – The Remaining Problem

The two second-cut designs have addressed four of the five design issues associated with the first-cut design of Chapter 3. Wire lengths have been reduced, clocking zone widths have been made uniform, the number of cells in a given QCA clocking zone have been reduced to a reasonable number, and wasted area has been eliminated. However, physical feedback does still not exist in any of the first or second-cut designs. This is not to say that it is impossible to generate feedback in QCA circuits.

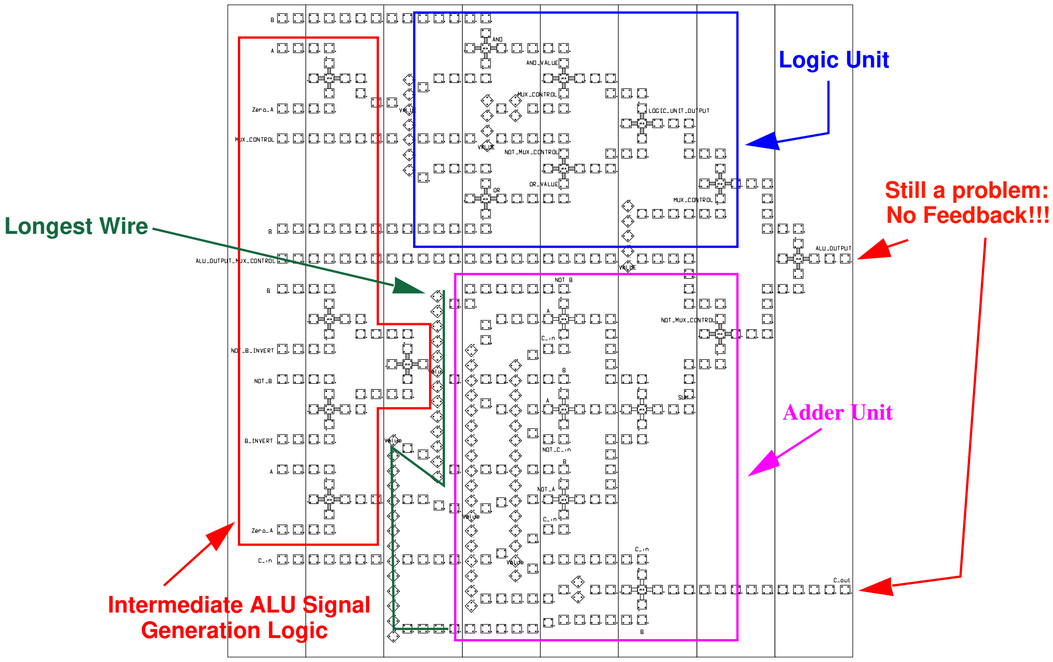

Figure 4.2. Another ”second-cut” design of the QCA Simple 12 ALU.

Floorplanning techniques discussed in Chapter 3 clearly indicate that feedback is possible. However, it has not yet been incorporated into any of the designs.

Feedback is most important as it allows a value to be stored for a given length of time – and is thus essential for registers and latches. A discussion of registers/latches in QCA will follow shortly. However, first a simple example of a QCA feedback circuit will be provided.

4.2.1 A Simple Feedback Example

Figure 4.3 shows a portion of the QCA Simple 12 ALU (part of the logic unit and intermediate signal generation logic) that uses the Chapter 3 floorplanning technique illustrated in Figure 18 to feed a value from the output of the ALU back to the input. While no registers or latches are implemented, this figure serves to demonstrate that feedback is possible in an actual circuit.

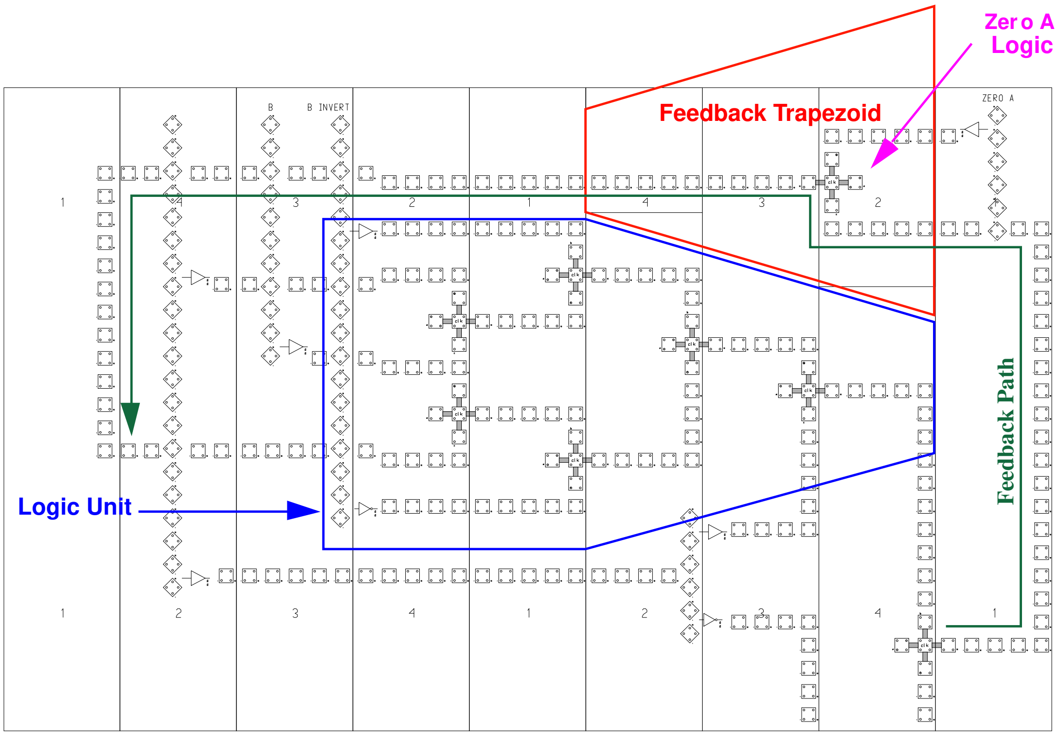

Figure 4.3. A simple example of feedback in a QCA circuit schematic.

There are two other interesting things about this design that are also worthy of mention. First, a portion of this design provides the first illustration of trapezoidal clocking in an actual QCA circuit schematic (it is highlighted in Figure 4.3). Second, in this schematic, part of the intermediate signal generation logic is placed "directly after" the output of the ALU – or more specifically, above it in the "feedback trapezoid". Thus, the A input to the ALU is zeroed on the way back to the input of the ALU. Essentially, useful computation is being performed "in wire"! While this technique is not used in many of the designs to follow, it is a valuable technique that can be used to save area and perform useful computation during signal transport.

4.2.2 An Introduction to Registers and Latches

Given that the movement of binary data in QCA is ballistic, what if an arbitrary delay is required for data being sent to a computational unit? (Note: ballistic data refers to the fact that a bit of binary QCA data will not stay on a wire indefinitely. When the clocking zone that the particular bit of data is in changes phase, there is a real possibility that the given bit could be lost if it is not copied) For example, at times a mechanism will be necessary to preserve a given binary value until other binary inputs arrive at the device cell of a majority gate. How can a "preservation" delay be added?

A common method for storing a bit of binary data is to use a flip-flop. A "generic" flip-flop usually has a 1-bit data signal input, a clock signal input, a reset signal input, a write-enable signal input, a 1-bit data signal output, and a complemented 1-bit data signal output. For different combinations of the clock, reset, and write-enable input signals, the binary value stored in the flip-flop will remain the same or be overwritten. Obviously, a similar functioning QCA device is required. Figure 4.4 illustrates the basic requirements for and design of a QCA "flip-flop".

The design essentially consists of a 2 x 1 multiplexor with intrinsic latching resulting from spreading the circuit over multiple sets of clocking zones. The two data inputs to the multiplexor in Figure 4.4 are a 1-bit data input and a 1-bit input from a feedback/delay path. A multiplexor control signal selects between the two inputs and selects either the old value stored in a feedback loop or a new value from some circuit. Thus, the device has the same functionality as a latch. A new value is sent back to some input and stored in the feedback loop or the new value is discarded and the old value remains in the feedback loop.

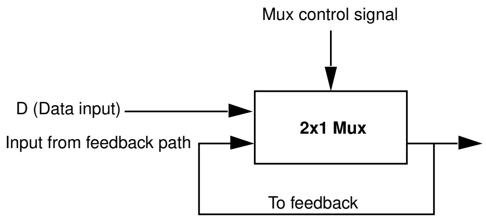

Figure 4.4. A block diagram for a QCA latch.

An actual schematic depicting the design appearing in Figure 4.4 is shown in Figure 4.5. Like the simple feedback example in Figure 4.3, processing is essentially being done "in wire" here as well. How? Well, the output of the ALU in Figure 4.5 will take a certain number of n clock cycles to arrive back at the input to the ALU. If logic gates or devices are included within the clocking zones that a signal passes through to arrive back at the input, n clocking zones will still be required. Thus, essentially, the register logic/operations are performed for "free" and do not necessarily add to the critical path of a circuit.

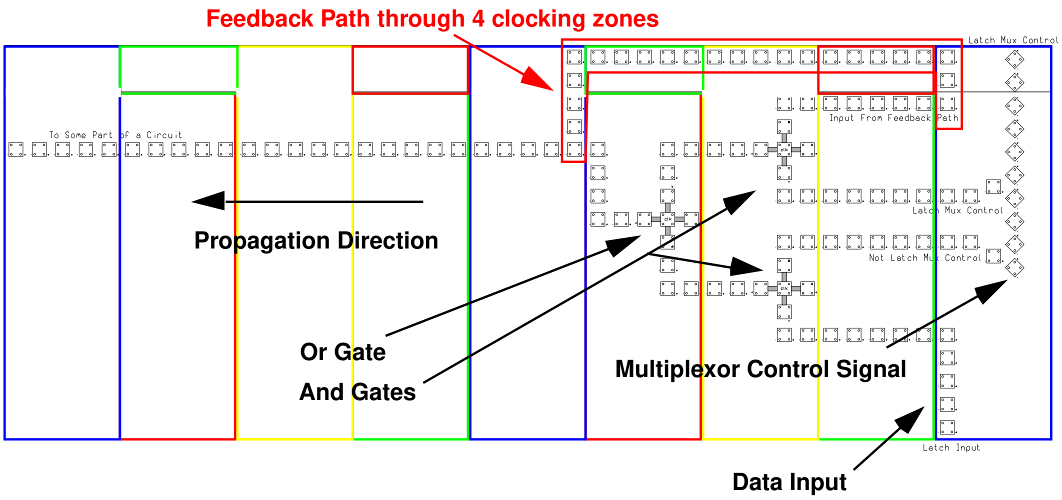

Figure 4.5. A schematic for a QCA latch.

So, just how does a multiplexor function as a latch/register? As mentioned above, intrinsic latching results from spreading part of the circuit over a set of clocking zones. This "feedback path"/intrinsic latching is highlighted in Figure 4.5. Essentially, the output of the multiplexor is fed back to the input of some circuit and back to one of the inputs of the multiplexor. The output of the multiplexor arrives back at the input of the multiplexor by traversing a wire that spans at least four clocking zones. What this really means is that this feedback path can provide an arbitrary delay of n clocks (as each of the four clocking zones will traverse through the four clock phases during the signal traversal on the wire). A register/latch will provide the exact same functionality – namely, a delay of some arbitrary number of clock cycles.

4.3 Putting Some Pieces Together

The latch design depicted in Figure 4.5 is exactly what is needed for the QCA Simple 12. A latch for the Accumulator and the PC as well as a multiplexor to select between the PC and the data bus have been added to the design in Figure 4.2. The new design appears in Figure 4.6.

4.4 Interconnect

The design in Figure 4.6 represents a complete one bit dataflow for the QCA Simple 12 (the Instruction register is considered to be part of the control flow). However, designs larger than one bit must be generated. Consequently, issues of interconnect must be addressed. In particular, how do you chain multiple bits of a dataflow together to generate something akin to a ripple carry adder?

In CMOS VLSI, the solution to this problem is to have the carry-out bit of the nth dataflow bit slice be at the same level as that of the carry-in bit of the (n+1)th dataflow bit slice (i.e. the carry-out bit of the *n*th dataflow bit slice and the carry-in bit of the (*n+1*)th bit slice will form a perfectly "straight wire"/be

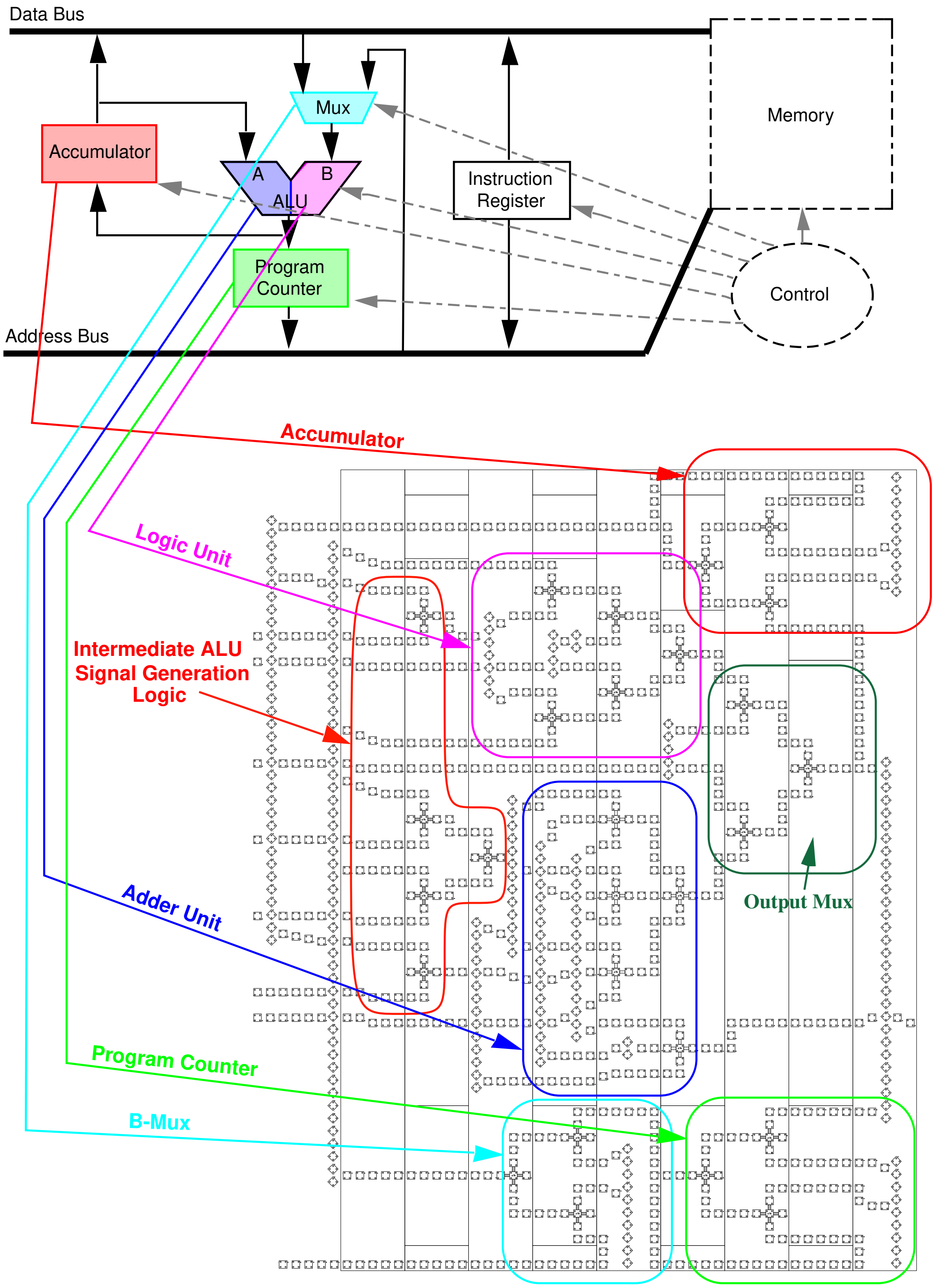

Figure 4.6. A complete 1-bit *data*flow of the QCA Simple 12.

logically connected when placed adjacent to one another). Thus, the carry-out bit will simply turn into the carry-in bit for the next bit slice over. The solution is the same for a QCA circuit. Two bit slices of the QCA Simple 12 ALU that are linked together are shown in Figure 4.7.

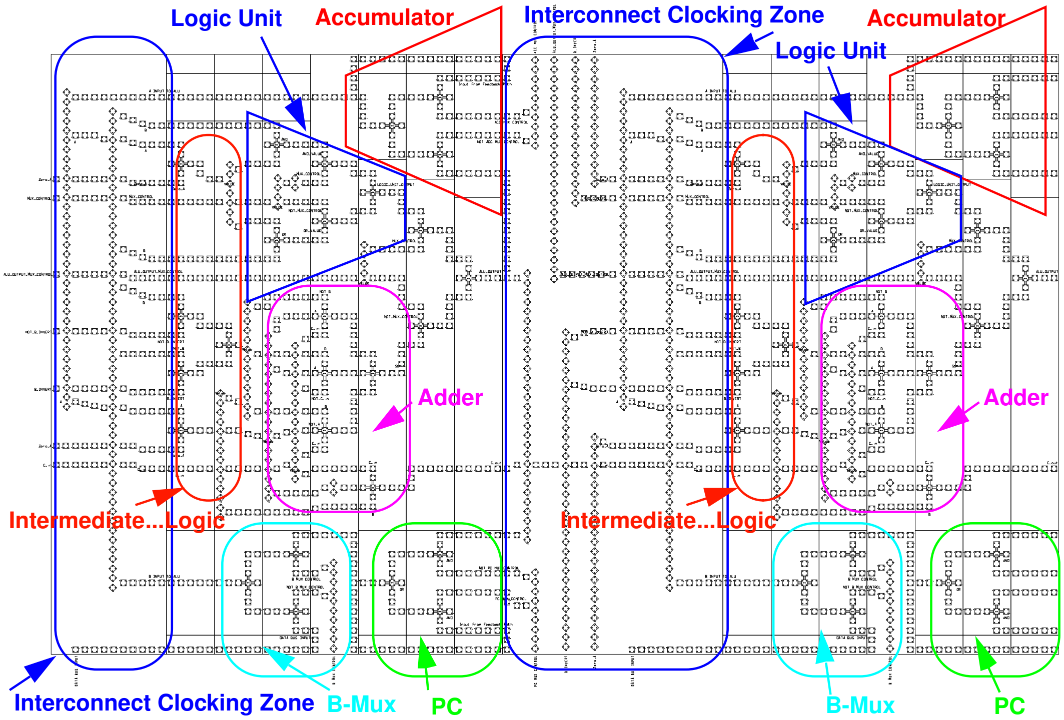

Figure 4.7. A 2-bit QCA Simple 12 ALU with registers and interconnect.

Upon examining Figure 4.7, perhaps the most notable features of the design are the wide clocking zones that precede the logic, adder, and intermediate signal generation logic before each bit. These clocking zones seemingly incorporate many of the design issues that were said to be undesirable in Chapter 3 and eliminated in other designs in this chapter. For instance, the "interconnect" clocking zones have a different width from all of the other clocking zones in the design. Additionally, there is a very large number of QCA cells within each "interconnect" clocking zone. Finally, wire lengths are extremely long in each of the "interconnect" zones. So, what affect does each of the three issues listed above really have on this design. Each will be addressed in the following subsections.

4.4.1 Interconnect Clocking Zone Width and Wire Length

While clocking zone width was labeled an "undesirable" in Chapter 3, this does not mean that "wider" clocking zones are "unfunctional". An attempt was made to limit clocking zone width for two reasons. First, it would reduce wire length. As mentioned in Chapter 3, wire lengths should be as short as possible to ensure that all cells in a wire switch successfully, and to help generate the shortest possible clock period (the clock rate will be governed by the time it takes for all cells in the longest critical path to finish switching).

However, in the interconnect clocking zones, clocking zone width really has no effect on wire length. Instead wires are necessarily long because between bit slices, control and input signals must enter from the top and bottom of the design. For example, the input from the data bus is required by both the logic unit and the adder unit. However, it enters the ALU via the B-Mux which is at the bottom of this design. Consequently, a long wire must be used to transfer this input to the adder unit and to the logic unit at the top of this design. This is similar for other input signals and feedback signals.

So, what effect does this have on the functionality of this design? The main concern with regard to wire length is to make sure that all cells in the wire do in fact switch and that the signal does not "degrade" sufficiently so that a cell gets stuck in a metastable state. (Note: A metastable state is one in which a local energy minimum is reached, but not the true ground state of the system – i.e. a definite polarization in this case resulting from a logical computation [8]). The problem of ideal wire lengths is still being studied by the technologists. Therefore an attempt is being made to keep wire lengths as short as possible (within reason). The other issue regarding wire length is how it affects clock rate. Because the longest wire in this design appears in the interconnect clocking zone, it will effectively set the critical path and have a large influence on the clock rate. In other words, the clock rate will have to be slow enough to ensure that all cells in a given zone actually switch during the switch clocking phase before the clocking zone changes to a hold phase.

It is definitely worth mentioning that having interconnect govern a given design's clock rate is not exclusive to QCA. In CMOS VLSI designs, interconnect can have a significant effect on clock rate.

Finally, there are two potential solutions to the problem of wire length. One solution would be to route various control and data signals from the top and bottom of the design (this is in fact done in the design of Figure 4.7 in several cases). This would help shorten wire lengths. However, in the case of an input coming from the B-Mux of the QCA Simple 12, this would not help solve the problem of wire length (as the B-Mux is only at the bottom of the design).

Another solution might be to add small clocking zones stacked vertically between bits. An example is depicted in Figure 4.8. This would allow longer wires to be "shortened" as they would be spread out over different clocking zones. However, this might very well pose a routing and timing nightmare when considering the fact that all inputs must arrive to both the logic and adder unit at the same time. If a signal was routed up from the B-Mux to the logic unit of the ALU, it would have to pass through several additional clocking zones which would ensure that the B input arrived at the logic unit and the adder unit at different times.

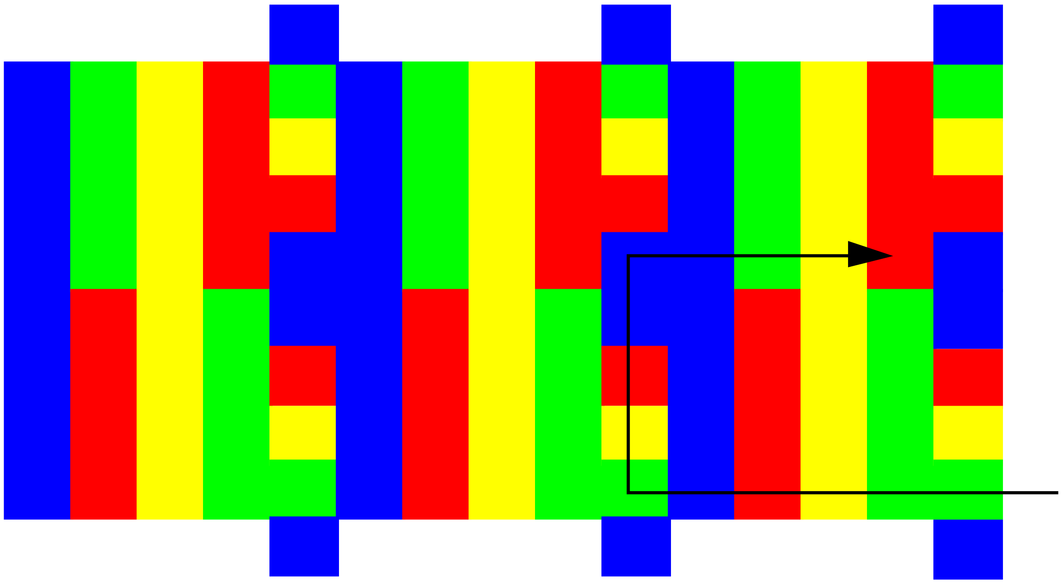

Figure 4.8. Stacked Clocking Zones.

4.4.2 The Number of QCA Cells per Interconnect Clocking Zone

The number of QCA cells in the interconnect clocking zones is not a significant problem at all. For thermodynamic effects to cause signal degradation, there must be on the order of 10^3 cells per clocking zone. There is no where near this many in the design of Figure 4.7.

4.5 State Machines

To conclude this chapter, a brief discussion of state machines – particularly as how they apply to control logic generation – will be presented. First, a simple QCA state machine will be presented. Then a brief discussion of the state machine logic required for the QCA Simple 12 will be presented. This discussion will include state transition diagrams and offer possibilities for routing control signals from the state machine/control logic to the Simple 12 dataflow.

4.5.1 A Simple State Machine

A picture of a very simple "one-hot" state machine appears in Figure 4.9 (in a "one-hot" state machine, a 1-bit storage device – i.e. register or latch – exists for each state; if the machine is currently in state k, the kth storage device will contain a logical "1" while all other flip flops will contain a logical "0"). The QCA representation of this state machine also appears in Figure 4.9. As with the Simple 12 dataflow, registers/latches are implemented intrinsically by spreading out wire over four clocking zones. The state bit can then be ripped out of the appropriate clocking zone to serve as input to the control logic.

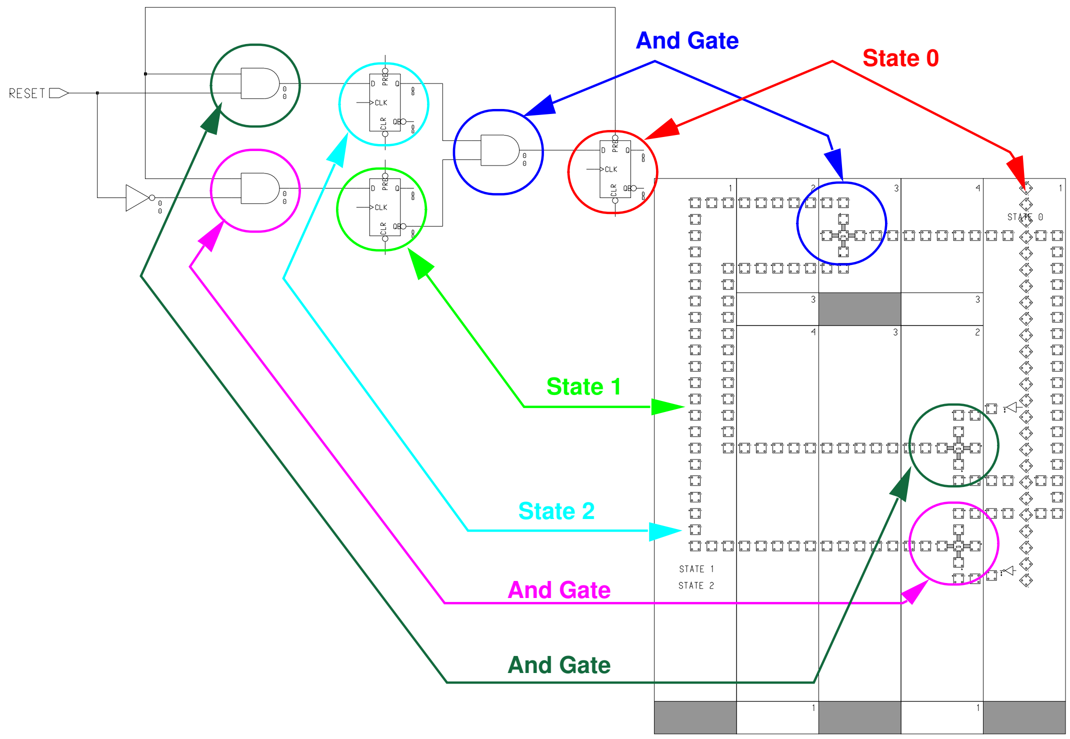

Figure 4.9. A simple QCA "one-hot" state machine.

4.5.2 Requirements for a QCA State Machine

To successfully execute all of the instructions in the Simple 12 instruction set, 11 control signals must be generated. These are summarized in the subsections below:

Accumulator Mux Control

Controls whether or not the value in the accumulator will be saved or will be overwritten by selecting the input from the feedback path or the input from the ALU output.

PC Mux Control

Controls whether or not the value in the PC will be saved or will be overwritten by selecting the input from the feedback path or the input from the ALU output.

B-Mux Control

Controls whether or not the PC or a value from the Data Bus is used as the ”B” input to the ALU.

B-Invert

This signal is used in performing the A-B operation. Specifically, it is used to generate the complement of B by XORing it with the B input. The B-Invert signal also controls the multiplexor that selects between the AND and OR portion of the logic unit.

Carry-In

This initial Carry-in bit (i.e. the 0th bit) must be set to ”1” in certain circumstances (i.e. generating the ALU output B+1 to increment the PC by 1).

Zero-A

This control signal is used to 0 the A input to perform the B+1 and 0 operations.

Logic-Or-Adder

This control signal selects between the output of the logic unit and the output of the adder unit to determine the final output of the ALU.

Memory Write Enable

This signal controls whether or not a value sent to memory will actually be written to memory (not a concern for the dataflow designs above).

A-Or-Memory

Controls whether or not the accumulator or memory data is placed on the data bus (not a concern for the dataflow designs above).

IR Mux Control

Controls whether or not the value in the IR will be saved or will be overwritten by selecting the input from the feedback path or the input from the ALU output (not a concern for the dataflow designs above).

PC-Or-IR

Controls which device writes to the address bus (not a concern for the dataflow design above).

The control signals generated will be based on what the current control state is and what the opcode of the instruction being executed is. State transitions are summarized below in Figure 4.10:

Essentially, this simple state machine can be implemented as a "one-hot" state machine. This information (i.e. a state is represented as either a 0 or 1) and the 4

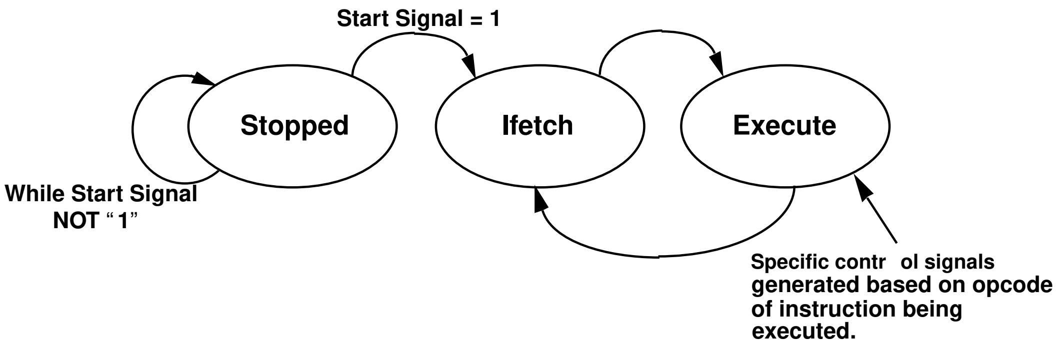

Figure 4.10. State Transition Diagram for Simple 12.

bit opcode of the current instruction can then act as inputs to some control logic block to generate the required control signals for the QCA Simple 12 dataflow.

4.5.3 Control Signal Routing

Finally, routing control signals to the actual dataflow in a QCA design should be briefly discussed. The most difficult part of control signal routing will be (not surprisingly) timing. This problem does not explicitly evolve from getting various control signals to multiple bits. In fact, control signal wires should be able run along the top and bottom of the Simple 12 design. Thus, they will arrive at bits slices along with other signals such as a Carry-in signal from a previous bit slice. However, given the fact that some control signals "start" in the middle of the design (i.e. the B-Mux control signal enters the design in a clocking zone that is "in the middle" of it), synchronization will be a difficult task. This will be an extensive focus of future research.

It is also worth answering (or proposing an answer to) the following question: namely, in a one-hot design composed of QCA cells, how does one assure that all bits of the QCA "registers" switch at the same time. Additionally, how does one assure that all control signals switch at the same time. This problem is analogous to clock skew in CMOS VLSI. The simplest answer to this question is clocking zone alignment. Particularly, properly aligning clocking zones and routing wire through clocking zones should solve this "problem". Nevertheless, it will receive extensive study in future work.

CHAPTER 5

SIMULATORS AND SIMULATIONS

This chapter will discuss QCA simulators and simulations. It will begin with the very brief history of QCA simulators in particular noting what, given existing simulation programs, we hope to achieve with our simulator. Next, a very brief overview of our existing simulator will be provided. After this subsection, the architectural simulation rules that comprise the engine of the simulator as well as their development will be discussed. Additional subsections will provide more details concerning data structures used, how values propagate and change, etc. Another subsection will discuss simple propagation simulators and compare them to a version of the simulator that allows clocking zones to be added. Next, using clocking zones in the QCA simulator will be discussed. A section concerning special difficulties encountered when designing this simulator will follow. Finally, the chapter will conclude with some "next steps" to be taken with the QCA simulator.

5.1 The VERY Brief History of QCA Simulators

The first simulator written for QCA devices was called AQUINAS. AQUINAS was written by the University of Notre Dame's Electrical Engineering Department and modeled the interactions between small systems/groups of QCA cells by actually solving the Schrodinger equation to follow electron movement. Such a simulation certainly provided a correct result when used to verify the logical correctness of a given design. However, the computation time for even a small circuit was extraordinary (on the order of 2^n).

In an attempt to devise a simple logical simulator, a group at the University College of London developed a simple "spreadsheet simulator" in Microsoft Excel. The goal of the spreadsheet simulator was to use information derived from AQUINAS and QCA theory (i.e. how groups of QCA cells interact and can be used for logical computation) to create a set of simple rules to verify the logical correctness of a given QCA design. The spreadsheet simulator achieved its goal but it was extremely difficult to create and change large designs (like that of Figure 4.7) with it.

What was desired and needed was a "happy medium", namely, a tested engine based on QCA theory that could verify the logical correctness of a given design with a user-friendly interface with reasonable simulation speed. This need gave rise to the Quantum Logic Based Engine Rules Tool (Q-BERT) (written in C++).

5.2 An Introduction to the Q-BERT Interface and Engine

As evidenced by discussions in Chapters 3 and 4 of this paper, QCA designs are somewhat rigid. What this means is that QCA cells must be in a fairly straight, vertical or horizontal line to form a wire, majority gate, etc. (The only exception is an off-center wire and in the designs of Chapter 4 where cells are only off-center by half of one cell). Additionally, when ripping a value off-of or onto a 45-degree wire, a 90-degree cell must be placed exactly between two 45-degree cells. Because of such constraints, a grid structure was devised as a means for "laying out" QCA cells. Examples appear in Figure 5.1.

It is worth nothing that the lines connecting the "X's" in Figure 5.1 do not represent any particular physical device. They are simply added to give the illusion of a "wire" and to help indicate the "direction" of wires.

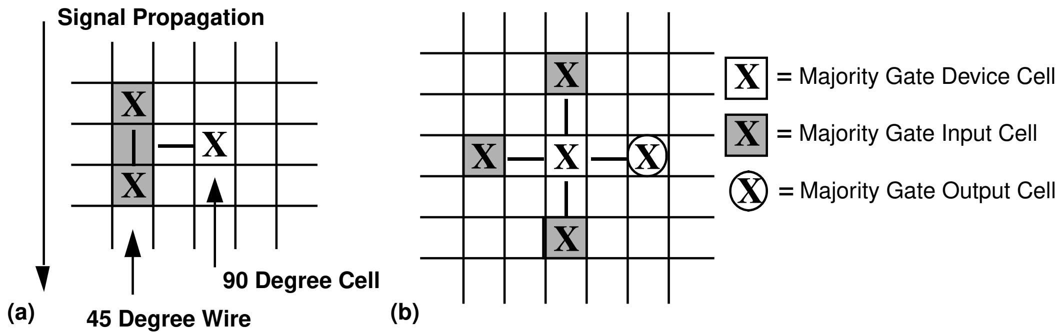

Figure 5.1. (a.) A graphical illustration of ripping a value off of a 45-degree wire to a 90-degree cell; (b.) A graphical illustration of potential cases of a majority gate input cell interacting with a majority gate cell.

Different types of QCA devices (i.e. majority gates, 45-degree wires, 90-degree cells, etc.) can be added to the grid (and hence the design) by highlighting a grid square and pressing the appropriate device button. For example, to add a majority gate, the user should highlight the square where the device cell should be and then press the majority-gate button. A majority gate will then be added. Devices are represented by coloring cells appropriately (i.e. a 90-degree cell is represented by a dark blue 'X' while the cells that are part of the majority gate are red). Additionally, some coloring is done to help the user identify devices. For example, Figure 5.1 a represents a value being ripped off of a 45-degree wire. A ripper cell is not colored dark blue, but rather another color to help the user identify what actually happens when these QCA cells interact.

It is worth noting that in terms of the engine, the colored ripper cell would not be treated as such, but rather just as a 90-degree cell. This is done to simplify interaction rules and will be discussed further in the next subsection.

It is also worth noting that the GUI can be and is used to display the results of a simulation. For instance, after a design has been entered, it can be simulated.

During simulation all X's will change to 0s or 1s based on the interaction rules for types of QCA cells. A screen shot of part of Q-BERT's GUI appears in Figure 5.2.

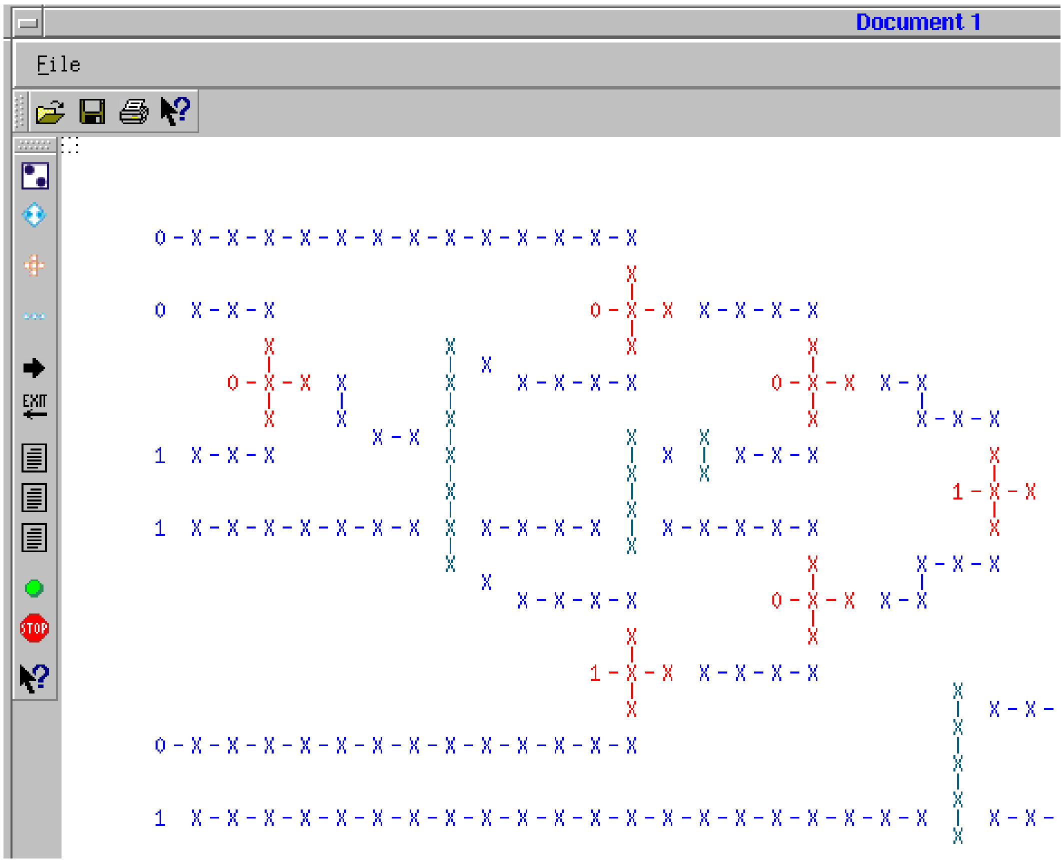

Figure 5.2. A screenshot of the Q-BERT GUI before simulation.

5.3 Q-BERT's Engine – for a Simple, Propagation Based Simulator

To verify the logical correctness of early designs and to ease the development of the QCA architectural simulation rules, a simple propagation simulator was first developed. Such a simulator does not take into affect the effects of clocking zones in the design. A simulation simply starts at the inputs of the design. Each input cell is placed on a queue and is assigned timestep 0. Cells adjacent to the input cells are compared with the design/interaction rules of the engine. If a design rule indicates that the two cells can interact (i.e. a 90-degree cell can interact with an adjacent 90-degree cell), then the adjacent cell is placed on the queue. When cells that were initially on the queue are processed, the timestep will be incremented and the cells placed on the queue during the previous timestep will be tested for possible interactions. This process will continue until the queue is empty. It gives the user the appearance that all changes during a given timestep occur simultaneously. As mentioned, the next thing that will be added to the simulator is the ability to simulate with clocking zones. Thus, the engine will be applied to cells within a given zone and a small amount of code will be added to handle interactions between zones when clock phases indicate that such interactions should take place. More details about the clocked engine appear in Section 5.6.

Initially, a significant reason for developing the simple QCA propagation simulator (as opposed to directly developing the clocked simulator) was to solidify existing knowledge about QCA device interactions. However, a secondary (and perhaps more important) goal quickly became developing efficient methods to implement QCA architectural simulation rules in code. As mentioned above, QCA devices are presented to the user via the GUI as 90-degree cells, 45-degree cells, majority gates, AND gates, OR gates, rippers, etc. However, if the Q-BERT simulation engine were to treat each cell interaction as a very specific type (i.e. a ripper cell interacts with a 45-degree wire or a ripper cell interacts with a 90-degree cell), the engine's complexity would be enormous.

Given this realization, it was determined that the data should be presented to the user in one way (i.e. so he or she is able to identify specific devices) but to the engine in another way. Thus, a data structure was set up so that every device in a given QCA design/schematic was treated as either a 90-degree, 45-degree, or device cell by the engine. This resulted in only eight architectural simulation rules that are summarized in the next section.

5.4 Architectural Simulation Rules

Each of the eight possible interaction scenarios is detailed in a different subsection. They are discussed in terms of what must happen in the engine during a simulation. It is worth noting at this time that all of the "rules" presented here have been verified in every manner possible at the current time. Many were derived from papers referenced in Chapter 2. Additionally, these rules were verified when the full adder designed by Lent et al [8] was thoroughly tested using the AQUINAS simulator. Only "design rules" appearing in Section 5.4.4 were not explicitly stated in a paper. However, they evolve from simple electron positioning and must take on the polarization specified in Section 5.4.4 because of Coulombic interaction. It is expected that future physical experiments will be needed to complete verification.

5.4.1 A 90-Degree Cell Interacting with a 90-Degree Cell

All possible situations of this case are represented by Figure 5.3. If a 90-degree cell changes phase, locations 1, 2, 3, and 4 must be checked for existing 90-degree cells. If a 90-degree cell exists in any or all of these four locations, it will get the data associated with the cell that changed with one exception. If a cell in location 1, 2, 3, or 4 changed in the timestep just before cell $\star$ did, then it will not change. Why? Because cell $\star$ changed in response to this change.

5.4.2 A 45-Degree Cell Interacting with a 45-Degree Cell

This case is identical to the case 5.4.1. However, cells will get the complement of the value associated with the cell that changed as adjacent 45-degree cells have alternate polarizations.

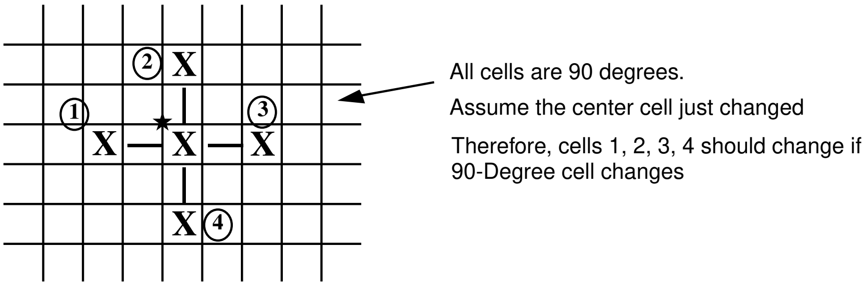

Figure 5.3. A graphical illustration of potential straight-adjacent 90-degree cell interactions.

5.4.3 A 90-Degree Cell Interacting with an Off-center 90-Degree Cell

All possible situations of this case are represented by Figure 5.4.

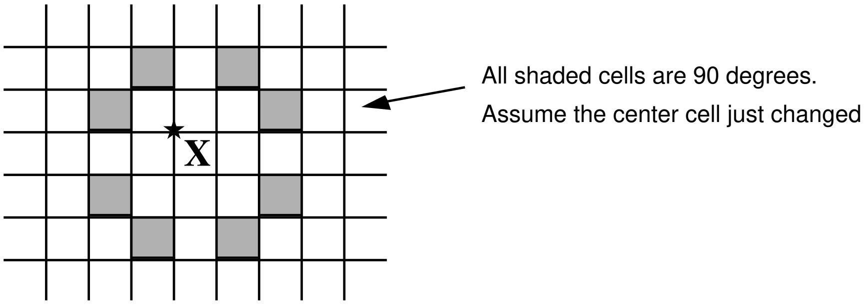

Figure 5.4. A graphical illustration of potential off-center 90-degree cell interactions.

If a 90-degree cell changes, shaded locations must be checked for existing 90-degree cells. If a 90-degree cell exists in any or all of these locations, it will get the data associated with the cell that changed with one exception. If a cell in a shaded location changed in the timestep just before cell $\star$ did, then it will not change. Why? Because cell $\star$ changed in response to this change.

5.4.4 A 90-Degree Cell Getting a Value from a 45-Degree Wire

Two possible situations are illustrated in Figure 5.5. In Figure 5.5 a, cell "o" will receive the complement of the data that is associated with cell * (just like the "next"

45-degree cell of the 45-degree wire). If the signal propagation along the 45-degree wire was in the opposite direction (i.e. in the "up" direction), cell "o" would receive the data associated with cell * (Note: **not** the complement. This is because of electron positions within cells). Because this is a rather complicated case, every possible interaction between a 45-degree wire and a 90-degree cell is illustrated in detail in Figure 5.6.

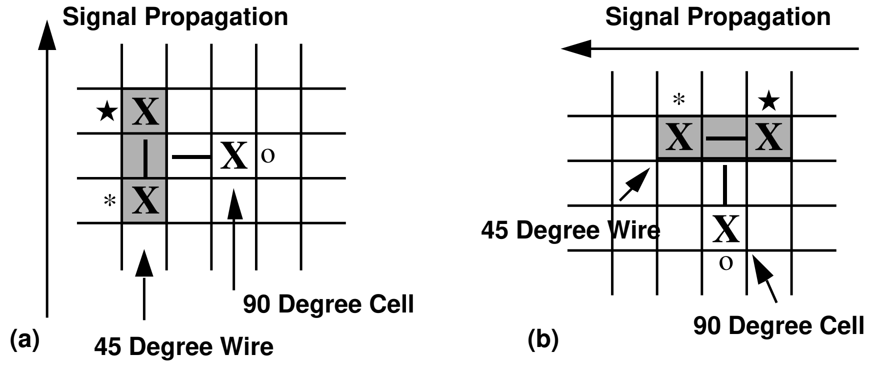

Figure 5.5. A graphical illustration of ripping a value off of a 45-degree wire to a 90-degree cell.

5.4.5 An Input Cell of a Majority Gate Interacting with a Device Cell of a Majority Gate

This case is *nearly* identical to the case 5.4.1. However, unlike a simple cell in a 90-degree wire, the cell that can be influenced by the cell that changed (i.e. another 90-degree cell) does not just receive the data associated with the cell that changed. Here, the majority gate device cell should get the majority of the cells that surround it. In the simple propagation simulator, the simulator will simply wait until all inputs to arrive for the device cell to change. However, in future versions of the simulator additional cell states may be introduced in an effort to mimic the lack of definitiveness in a device cell if all three inputs have not arrived. Still, it is worth noting that the simple propagation simulator will nearly emulate the functionality

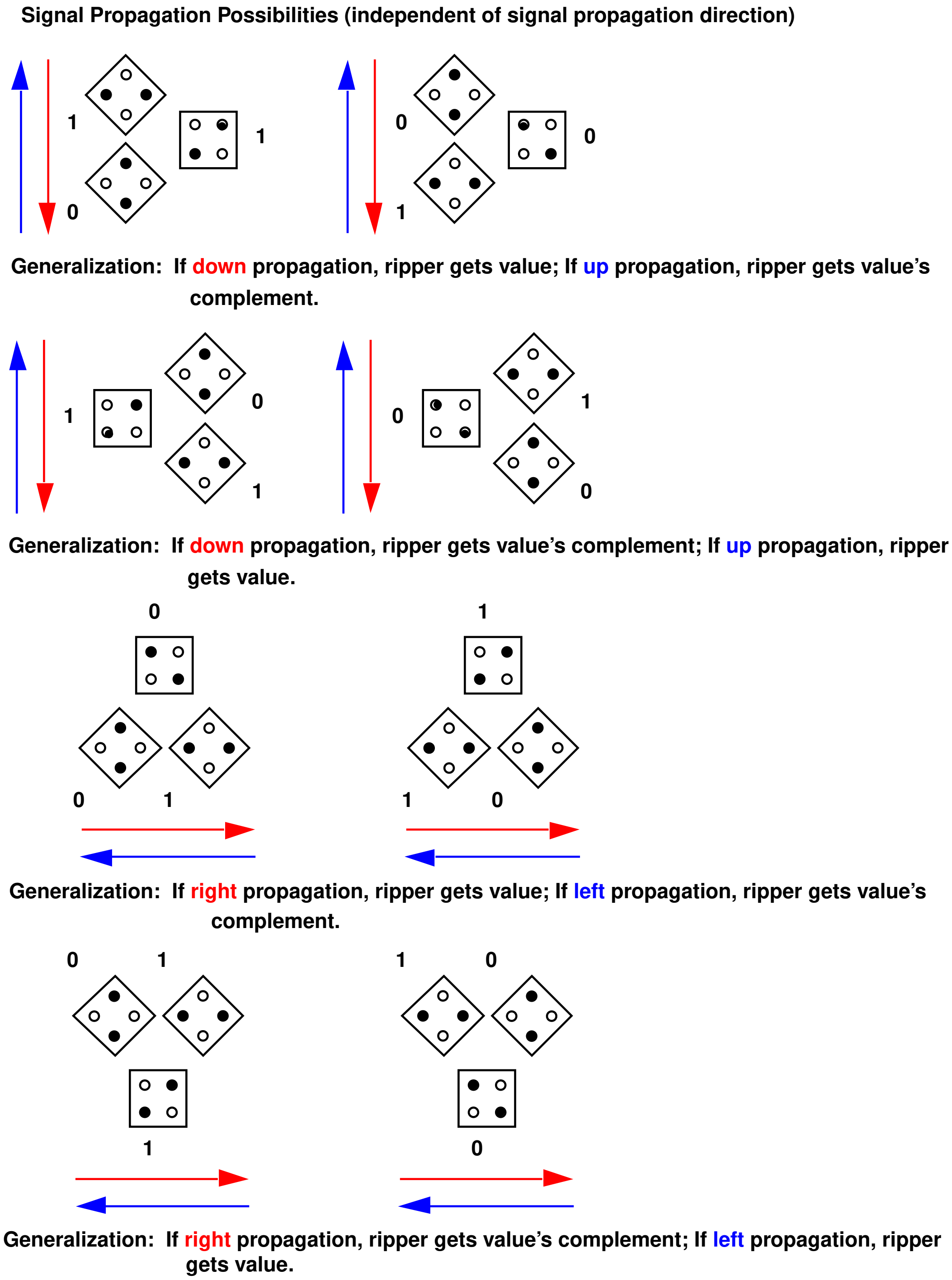

Figure 5.6. Possible 45-degree wire and 90-degree cell interactions.

of a design and simulator with clocking zones as a majority gate device cell usually is present just after the start of a clocking zone border. Consequently, all input wire lengths are nearly identical.

5.4.6 A Device Cell of a Majority Gate Interacting with a 90-Degree Cell

This case is identical to the case 5.4.1. The device cell will simply give its data to the 90-degree output cell of the majority gate.

5.4.7 A Crossover

One possible case is illustrated in Figure 5.7. In Figure 5.7, if cell H changes, a check is made to see if a cell of a 45-degree wire is directly in line with it. If it is, a check must be made for a 90-cell (cell *), on the other side of it. If cell * does exist, then it will receive the data associated with cell ⋆. Cell * will then be put on the queue.

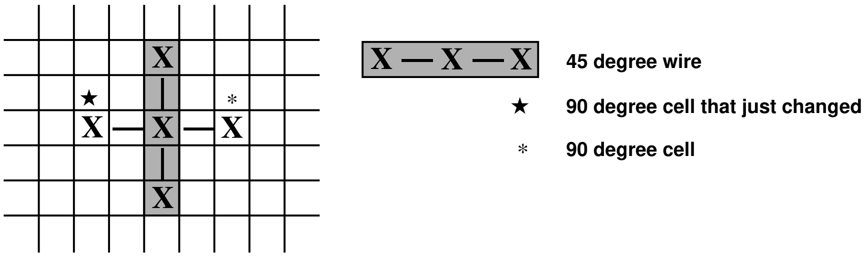

Figure 5.7. Situation for a crossover.

5.4.8 Ripping a Value from a 90-Degree Cell to a 45-Degree Cell

All cases are illustrated in Figure 5.8. In all cases, cell * will get the data associated with cell "o" and cell ⋆ will get the complement of this data.

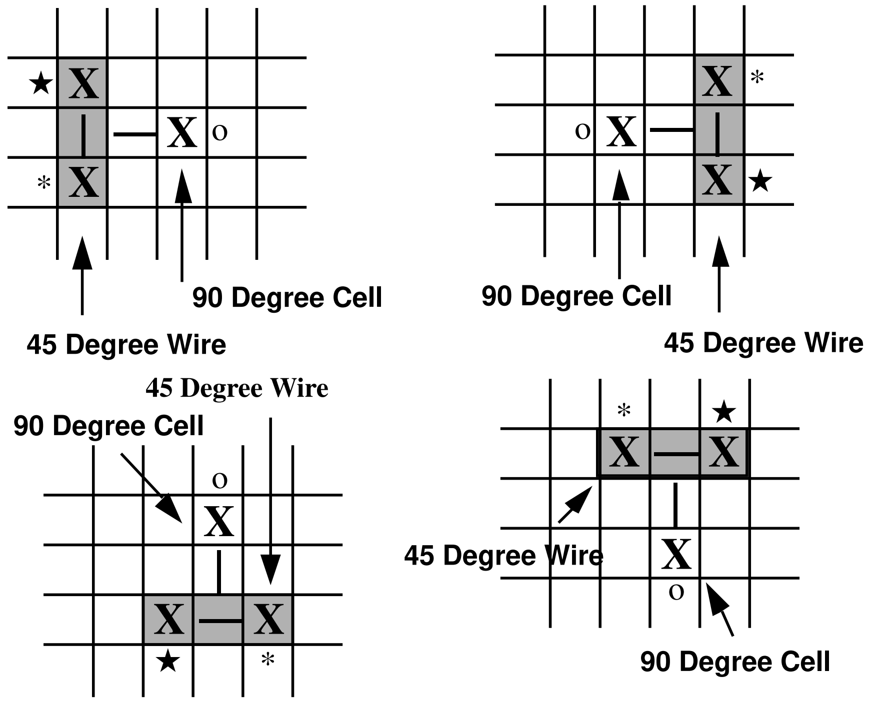

Figure 5.8. Situation for a crossover.

5.5 Details of Q-BERT's Engine

As mentioned in section 5.2, a grid structure was chosen as a base for representing QCA designs. This was done in an effort to promote designs that possessed some degree of rigidness. In other words, the grid structure forced the user to create "straight" wires, rip off values from wires in the proper location, etc. Because of this grid like structure, arrays were chosen as data structures to store data required by the engine (and the GUI as well). The most important data structures will be discussed in subsections below. Then, how propagation/simulation occurs will be discussed in the context of these data structures.

5.5.1 The Color Array

This array stores the "colors" of all of the cells on the grid. Essentially, it has one purpose – to allow cells displayed on the GUI grid to be colored appropriately in an effort to identify device types. Thus, if a particular cell is supposed to function as a ripper, it will be identified as such in this array. When the rendering of the grid is done for the GUI, information from this array will be used to color the cell appropriately.

5.5.2 The Device Matrix

The device matrix is similar to the color array. However, the device matrix is used by the engine, not the GUI. Thus, if a particular cell is supposed to function as a ripper, it will be identified as a 90-degree cell (recall that for the engine, all QCA cells in a given design are classified only as 45-degree, 90-degree, or device cells).

5.5.3 The Contents Array

This array holds a string representation of the data for every cell on the grid. It is not used for any computation or the engine, but rather just holds information (i.e. whether or not a cell is currently in a state equivalent of a binary 0 or 1 or garbage data) so that it can easily be rendered in the grid structure of the GUI.

5.5.4 The Changable Array

This array stores information about whether or not a given cell is "changable". What does this mean? Well, physically, certain cells can have their polarizations "frozen". Consequently, they will never change state. For instance, this is how an AND gate or an OR gate is created. One of the input cells of a majority gate is simply frozen so that it always has a polarization corresponding to a binary 0 or a binary 1 respectively. (It is worth noting that physically, this could be accomplished by creating a QCA cell with only two quantum dots to form the required polarization (illustrated in Figure 5.9. It is also worth noting that input cells also fall into this "unchangable" category). However, this information must be conveyed to the engine of the simulator (i.e. it would be less than ideal if the engine caused a cell with a frozen polarization to change). Thus, when a cell is added to a schematic, and its polarization should be unchangable, a flag in this array will be raised.

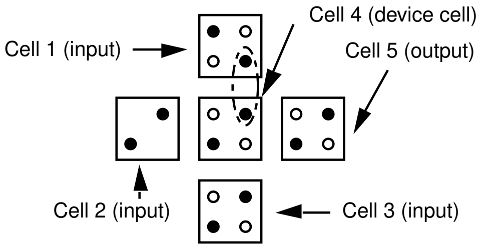

Figure 5.9. An example of a "dedicated" QCA cell with a majority gate (hence the majority gate is an OR gate)

5.5.5 The Data Array

This array stores logical information about the design (i.e. if a cell has a polarization P=+1, P=-1, or no polarization). It is used first during a simulation to update the logical values of QCA cells in a schematic, and second when a cell is initially added to the design to indicate if it has a frozen polarization or no polarization.

5.5.6 The Timestamp Array

As discussed in Section 5.3, timestamps are used to help convey to the user what cells change during what time. The timestamp array simply stores timestamps for each cell in the design. In particular, they also provide some degree of simultinaety – i.e. two parallel wires should have matching cells that will change at identical times. However, physically (with regards to the engine) the cells making up the wire will change at different times. Timestamps assist in representing parallelism. Finally, as mentioned, timestamps are used to make sure signals propagate in the right direction and don't "backtrack". The timestamp array simply stores timestamps for each cell in the design.

5.5.7 The Majority Gate Count Array and The Majority Gate Device Array

The functionality of these two arrays can best be described by explaining exactly how majority gates might function in a real design. Figure 5.10 offers two potential ways in which a majority gate might appear in a given design:

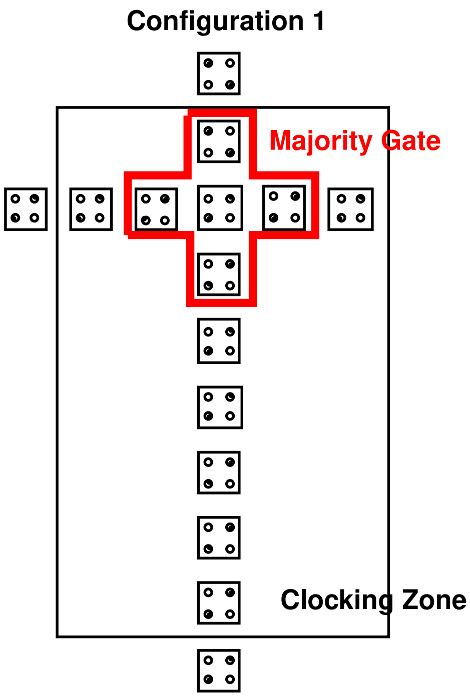

Figure 5.10. Potential logic gate configurations.

Seemingly, there should be nothing wrong with Configuration 1 of Figure 5.10. However, every logical gate in the designs of Chapter 4 is set up with Configuration 2. Why is this? Well, with Configuration 1, the three inputs to the majority gate will reach the device cell at "radically" different times. If this were to occur, there is a real possibility that the majority gate will just be treated as a "wire". In other words, when the signal entering from the top of Configuration 1 reaches the majority/logic gate device cell, the signal would simply fan out in three directions. Not only might similar things occur with other input signals (i.e. from the left and bottom), but eventually, signals might even "collide" as they will be coming from different directions.

However, with Configuration 2, inputs to the majority/logic gate are essentially "frozen" in a hold clocking zone. While the "wires" that act as inputs to the device cell of the adjacent majority gate are still not of equal length, two of them are and more importantly, they are extremely short. After conversations with researchers in the University of Notre Dame Electrical Engineering Department, it was concluded that such a configuration would function reliably as a majority/logic gate. The sufficiently short (and almost equal in length) wires will in fact settle into definite polarizations and the device cell will function as intended.

So, how should this be simulated? While it would be feasible to simulate a "bounce" or unintended fan out from a device cell, it is really not worth the effort to do this. The lengths of the three wires that act as inputs to the device cell of a majority or logic gate are one cell, three cells, and three cells. Thus, initially, the first cell of each wire would change polarization (based on "input" cells from the adjacent clocking zone in the hold phase). Next, the second cell in each of the three cell wires would change while the one cell wire would affect the device cell. Then, the third cell in each of the three cell wires would change while the device cell might affect its adjacent cells (including the output cell of the gate and the third cell of the three cell wires). Thus, there is a most "bounceback" on one cell of a wire. Instead of logically simulating this, two arrays were created to count the number of inputs that arrive at a given majority gate. When all three inputs arrive, the device cell changes. Logically, the device cell gets the exact value that it should.

5.5.8 Putting it all Together

While the functionality of the engine was discussed briefly in Section 5.3, more details will be provided here using, in particular, the context of these data structures.

At the start of a simulation, all of the input cells will be placed on a queue, with a delimiter placed after the initial entries. Also, the timestamps of every cell in the design will be initialized to some negative number (except the input cells which will be initialized to have a timestamp of 0). The input cells will be used to check for potential interactions using the architectural simulation rules of Section 5.4. If the design rules indicate that cells are properly aligned for an interaction (using data from the device matrix), the timestamps of the cells involved will also be compared. If the timestamp of the cell from the queue is greater than the timestamp of the cell involved in the possible interaction, the interaction will take place.

Based upon the type of interaction that occurred, the data array will be updated accordingly and the contents array will be changed to reflect the change in polarization of a given cell in order to convey this information to the user. Additionally, the cell that just changed will be placed on the queue (following the delimiter initially placed on the queue). Finally, the timestamp of the cell that just changed will be changed to the next timestamp. For example, if an input cell had a timestamp of 0, then the cell that changed would get a timestamp of 1. Thus, it is impossible for a signal to travel "backwards" because of the timestamp restriction.

When all of the cells on the queue preceding a delimiter have been checked against the architectural simulation rules, the timestamp will be updated and the next set of cells on the queue will be checked against the architectural simulation rules. This process will continue until the queue is empty signaling the end of the simulation.

5.6 A "Clocked" Simulator"

A simple propagation simulator is certainly valuable for verifying the logical correctness of a design. However, should a complex QCA circuit (such as the designs in Chapter 4) actually be implemented without clocking zones, the probability of it functioning correctly is essentially zero. Issues such as wire length and majority/logic gate race conditions would almost assuredly prevent the circuit from working properly. With this thought in mind, it was obvious that Q-BERT should not simply verify the logical correctness of a design – it should also mimic the way a circuit with clocking zones would actually function. Consequently, it was determined that a means for adding clocking zones to a schematic should exist.

5.6.1 Adding Clocking Zones

It should quickly be mentioned exactly how clocking zones are added to a schematic. Essentially, the simulator has the ability to define only rectangular clocking zones. Thus, to add a clocking zone, the user should highlight the grid square that he or she wants to the be upper left-hand corner of a given zone. Then the user should press the "Add Clocking Zone" button. This will launch a dialog box that will allow the user to select the initial phase that the clocking zone should be in (i.e. switch, hold, release, or relax). When this phase is selected, the dialog box will disappear and the user only needs to click the grid square where he or she wants the lower right-hand corner of the clocking zone to be. While the fact that only rectangular clocking zones may be added to the design may seem restrictive, it is actually extremely advantageous as will be seen.

5.6.2 The “Hold” Situation”

An interesting problem is how hold clocking zones should be simulated. In the hold phase, inter-dot potential barriers are held high so the outputs of a subarray (or the border cells of a clocking zone) can be used as inputs to the next (or adjacent) clocking zone which should be in the switch phase. However, a problem stems from the fact that at startup, a clocking zone in the hold phase is adjacent to a clocking zone in the release phase and a clocking zone in the switch phase. As a result, there is never any time where cells in the hold phase are ”driven”. This raises the question of whether or not the cells in the hold phase would have any polarization at all.

After conversations with individuals in the University of Notre Dame Electrical Engineering Department, it was determined that at startup, cells in the hold phase would indeed have a definite polarization (after all, potential barriers are high and electrons must be in definite positions). Nevertheless, exactly what polarization each cell in a hold zone would have would be unknown and random. Thus, a situation in Figure 5.11 would be entirely possible.

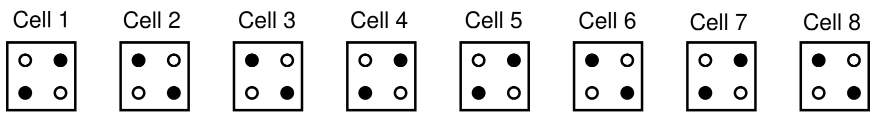

Figure 5.11. A potential QCA ”wire” in the hold phase at startup

The fact that cells in a hold clocking zone have some initial random polarization is both good and bad. It is good because if cells in a hold phase initially had no polarization, we would essentially be dealing with three-state logic. In other words, a hold clocking zone would be adjacent to a switch clocking zone but there would be no definite polarization to drive the switch clocking zone! Granted, the driving cells in the hold clocking zone are random – and thus a circuit must be cleared and initialized at startup. However, this is not any different than any CMOS VLSI circuit. The negative side to random values in a hold clocking zone at startup stems from simulation difficulties. Essentially, all border cells of hold clocking zones must be added to the queue when the engine is invoked. More will be said about how this is accomplished in a following subsection.

5.6.3 Clocking Data Structures

The addition of clocking zones to Q-BERT also requires the addition of several data structures to support them. The three most important data structures are the clocking zone array and the hold clocking zone array(s).

The clocking zone array simply stores a value for every cell in the grid that corresponds to the state of the clocking zone that a particular cell is in. Thus, at startup, the clocking zone array entry for a given cell will correspond to the phase initially assigned to that zone. Similarly, as a simulation progresses and clocking zones change phase, this array will be updated to store this information.

The hold clocking zone arrays are required to store the initial positions of the hold clocking zones (i.e. the upper left-hand corner's and lower right hand corner's coordinates are stored). These are required because when the engine is invoked, border cells of a hold clocking zone that are adjacent to border cells of a switch clocking zone must be placed on the queue to check for potential cell-to-cell interactions. Thus, there must be some means for determining where the hold clocking zones are. When a hold clocking zone is added, its upper left-hand corner coordinates and lower right-hand coordinates are added to this array. Then, a nested for loop can simply use these coordinates to check for the existence of border cells in the zone.

This hold clocking zone "situation" is the reason that only rectangular clocking zones are allowed. If random clocking zone shapes were allowed, it would be *extremely* difficult to determine where border cells were, and storing the upper left-hand corner and lower right-hand corner of the zone would no longer work for determining border cells (see Figure 5.12 a). However, by constructing clocking zones from rectangular blocks, nonrectangular clocking zones can still be created and the left-hand corner/right-hand corner method can still be used to determine border cells (see Figure 5.12 b).

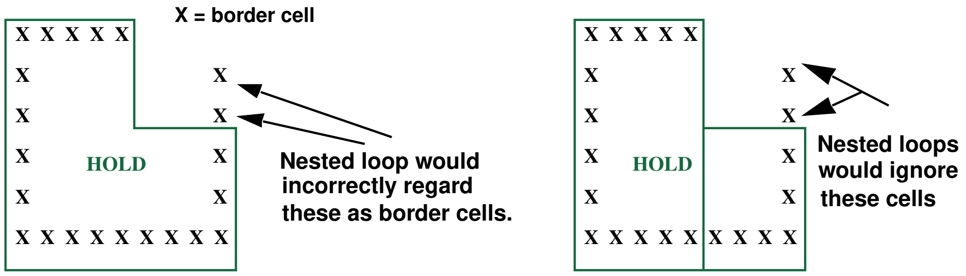

Figure 5.12. (a.) A hold clocking zone constructed from nonrectangular elements; (b.) A hold clocking zone constructed from rectangular elements

5.7 Q-BERT's Engine – for a Clocked Simulator

This section will explain how Q-BERT's engine works for a clocked simulator as well as what changes were made to the engine to successfully implement clocking.

5.7.1 Startup

For the simple propagation simulator, it was discussed in Section 5.3 that, at startup, each input cell was placed on a queue and assigned timestamp 0. However, with clocked designs, border cells of hold clocking zones that border switch clocking zones must be considered. In particular, if they have a cell that the cell of the hold clocking zone can interact with they must also be placed on the initial queue. This is accomplished as follows:

When the engine is invoked, as before, all input cells are placed on the initial queue. Then, using the coordinates of the hold clocking zones, every cell on the border of a hold clocking zone is placed on another queue. The engine design rules checks are then invoked on the cells placed on this other queue. As before, this logic simply checks for any potential interactions with adjacent cells (according the architectural simulation rules of Section 5.4). If a potential interaction(s) exists, then the clocking zone of the found cell is also examined. If this found cell is in a switch clocking zone, then the cell of the hold clocking zone is added to the initial queue. When all border cells in hold clocking zones have been examined and the initial queue has been updated accordingly, the engine will again be invoked but this time to simulate the entire design.

5.7.2 The Release Problem and its Consequences

The most difficult part of simulating a design with clocking zones stems from what occurs during the release clocking phase. When a clocking zone is in a release phase, its inter-dot potential barriers are lowered and the cells of that zone relax from a polarized state to an unpolarized state. Thus, transitions such as 0-to-unpolarized and 1-to-unpolarized must also be considered. Essentially, this adds an extra state to our engine propagation logic.

That being said, there are two potential ways to assure that such transitions take place properly. The first method would involve storing the coordinates of every clocking zone in a given design (in a data structure similar to that used for hold clocking zones) as well as the phase that the clocking zone is in. During each clock transition, the following would occur: Hold clocking zones would be unaffected (after all, cells in the hold phase do not change polarizations). Cells in the relax clocking zone would also be unaffected (after all, cells in the relax phase also do not change polarizations). Cells in clocking zones that are in the release phase would simply change from whatever value/polarization that they had to no polarization. This could be accomplished simply by means of a nested for loop. Cells in clocking zones that are in the switch phase would simply have the engine invoked upon them as normal using cells that are on the queue. When the queue becomes empty, clock phases can be advanced. Then, startup code would have to be reexecuted to place cells in the hold phase adjacent to cell in the switch phase on the queue. After every four clock transitions, input cells would also be placed on the queue (because clocking zones adjacent to them would be in the switch phase).

Another method for solving the release transition problem would be to have the engine handle it. In other words, if the engine detected two cells that could interact with the cell that most recently changed being adjacent to another cell in the release phase, then the cell in the release would change just as a cell in a switch phase that is adjacent to a cell in the hold phase would. However, in this case, the cell in the release phase would change to an unpolarized state. While this method avoids storing the coordinates of all of the clocking zones, it also creates a logic nightmare. Cases would have to exist for every possible clocking zone border interaction to determine the proper course of action. Additionally, cells' timestamps would be updated at improper times. Thus, while the overhead of the first method is slightly larger than that of this second method, the first method will result in a much more "stable" and risk-free simulation. Also, the first method should be faster as every cell in the design is not being examined during each iteration.

5.8 Future Simulator Improvements

It is worth stressing that Q-BERT is a work in progress. Current and future work with it will center around two axes. First, efforts are currently underway to make the simulator more user friendly. Something similar to Mentor Graphics Design Architect is desired. In particular, at present, there is no way for the user to move components of the design around. Thus, once a gate or a wire is added, for it to be moved, it must be deleted and redrawn in its new and desired location. Second, future work will be directed into giving the user the option of having the engine/simulator define the clocking zones. All that would be required of the user would be to specify certain locations between cells where he or she would like a clocking zone border to be. The engine/simulator would then "draw" the rest of the zone.

CHAPTER 6

SIZE COMPARISONS

This chapter will discuss dimensions that are associated with QCA cells. Area estimates will be given for our designs. Additionally, the area of a QCA design will be compared against an equivalent circuit implemented in CMOS VLSI. Density gains will then be calculated and estimated

6.1 QCA Dimensions

It is now worthwhile to consider the dimensions associated with the QCA technology. Given current projections for early fabrication runs, the expected distance between quantum dots is 10 nm as shown in Figure 6.1. Furthermore, the diameter of a quantum dot is also 10 nm. The distance between centers of adjacent cells is on the order of 42 nm as there must be a slightly larger separation between electrons of neighboring cells [5].

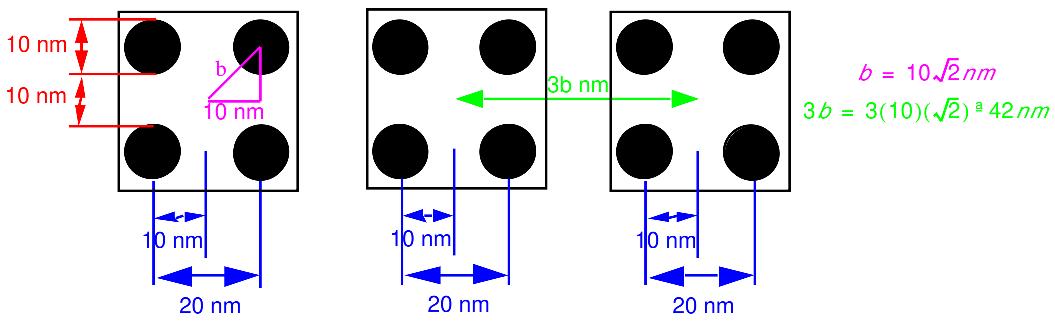

Figure 6.1. Assumed dimensions associated with QCA cells (standard).

Additionally, scientists are currently developing means for scaling the size of a QCA cell by a factor of 10. Such a dot implementation would be accomplished by using chemical molecules to form various QCA cells. Figure 6.2 reflects these size numbers.

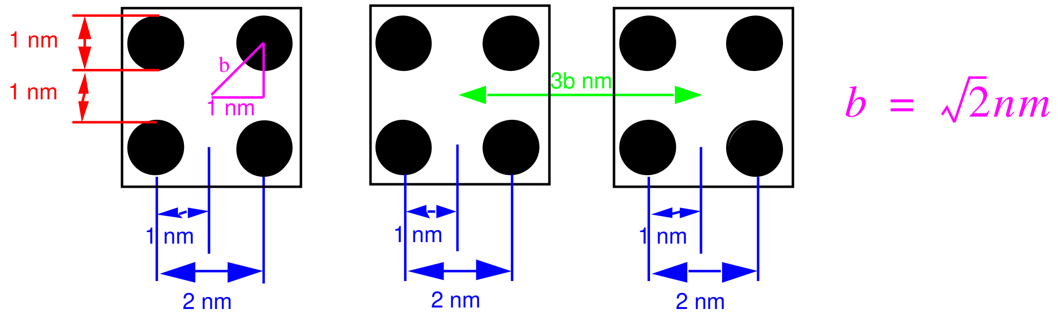

Figure 6.2. Assumed dimensions associated with QCA cells (molecular).

6.2 Density Comparisons

At present, designs for the complete dataflow of Simple 12 have been completed (see especially Figure 27 and Figure 28). Additionally, a CMOS VLSI design of the entire Simple 12 processor exists (using 2 micron design rules). The dimensions for the CMOS equivalent of the QCA design were taken and compared to the dimensions of the QCA circuit (essentially, this designed contained a 12-bit dataflow consisting of the accumulator, b-mux, ALU, and program counter). The results are summarized below in Table 6.1.

Note that in Table 6.1, no dimensions appear for the 2 micron CMOS VLSI design of the Simple 12 dataflow. Obviously, few if any circuits (especially microprocessors) are fabricated using a 2 micron process. Also, we are considering QCA as a potential replacement for CMOS. Thus, it is more worthwhile to look at processes toward the end of the CMOS curve (i.e. 0.05 and 0.07 micron processes). As a result, the 2 micron numbers were scaled to given an approximate area for a 12-bit Simple 12

Table 6.1. QCA Density Gains over Equivalent CMOS Designs.

Technology	Length	Width	Area	Density Gain
CMOS (0.07 μ)	29.8 μm	45.5 μm	1356.1 μm^2	N/A
CMOS (0.05 μ)	32.5 μm	21.3 μm	692.3 μm^2	N/A
QCA ("conventional")	28.1 μm	2.7 μm	75.9 μm^2	17.8 (0.07 μ) / 9.1 (0.05 μ)
QCA ("molecular")	2.81 μm	0.27 μm	7.6 μm^2	1787 (0.07 μ) / 912 (0.05 μ)

dataflow for a 0.05 and a 0.07 micron CMOS process and it is these numbers that appear in Table 6.1.

When comparing the projected densities for a 12-bit CMOS Simple 12 dataflow against both "conventional" QCA numbers (those of Figure 44) and "molecular" QCA numbers (those of Figure 45), it was concluded that QCA has the potential to offer substantial density gains. For instance, when comparing the "conventional" QCA design against a 0.05 or 0.07 micron process CMOS design, calculations project that a QCA design would be an order of magnitude more dense. Furthermore, when comparing the "molecular" QCA design against the 0.05 and 0.07 micron process CMOS designs, calculations project that a QCA design results in a three orders of magnitude density gain. Thus, while there is undoubtedly some error associated with scaling CMOS designs and with the final numbers of QCA devices, these errors are certainly within a three order of magnitude density gain.

6.3 Odds and Ends

It is worth noting that all of the designs discussed in this section (i.e. those appearing in Figure 27 and 28) and the CMOS designs have been simulated and verified for logical correctness (using simple hand-checking and Q-BERT). All are capable of performing the operations needed to successfully execute the Simple 12 instruction set. Thus, if constructed, there is no reason to believe that these designs would not function correctly.

6.4 A QCA "Roadmap"

Finally, it would be appropriate to discuss the current benefits, limitations, and projections for actual QCA devices.

6.4.1 Limitations

According to the Technology Roadmap for Nanoelectronics [2], there are two main problems that must be overcome to successfully implement a QCA circuit. They are the need for individual adjustment of each cell and the limits of operating temperature. Individual adjustment of each cell is currently required because of fabrication tolerances, the presence of stray charges, and need for $4N + 2$ excess electrons in each cell (limits performance degradation). To make cell adjustment possible, leads are required to load onto the quantum dots the exact and required number of excess electrons. By attaching leads to cells, straightforward lateral branching from a chain of cells (a basic feature needed to create logic gates) is prevented [2].

Operating temperature limitations are more fundamental in nature [2]. Operating temperature limits stem from the weakness of the dipole interactions between cells which much be significant larger than kT. The current state of the art in QCA devices is the majority gate. This gate was constructed based on metal-insulator tunnel junctions which operate at only a few millikelvin [4]. Structures at the molecular level will be needed to approach room temperature operation [2].

6.4.2 Destinations

In order for QCA cells to operate at close to room temperature (and overcome the two significant limitations discussed above), small cell dimensions on the order only a few nanometers are required [2]. Possible implementations of such cells include single bistable molecules. These molecules, when in the absence of an external electric field, will have some valence electrons that can localize in two different groups of bonds with the same probability. This is analogous to four-dot QCA cells where, the presence of an external electric dipole field would make one of the two possible electron configurations energetically favorable [2].

Problems would undoubtedly exist with arranging the molecules in structured arrays on some substrate. But, if this problem can be overcome, the problem of fault tolerance will also be solved. Why? Molecules are intrinsically precise. Also, bound electrons would be sufficiently localized to act as “excess” electrons without the need for external leads [2].

On remaining issue would still be stray charges unless it becomes possible to develop substrates virtually free of them [2].

This subsection concludes with some projections from the Technology Roadmap for Nanoelectronics for the next 6 and 12 years [2]. These are summarized below in Table 6.2.

Table 6.2. Projections for QCA in various benchmarks.

Benchmark	2006	2012
Feature size	2 nm	10 nm
No. of devices	4	10^6
Circuit speed	10 kHz	100 GHz
Events / chip / s	4 10^4	10^{14}
Power supply, V_{dd}	0.1 mV	0.1 mV
Power dissipation	1 pW (excluding cooling)	n/A
Temperature	4 K	4 K

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This section will summarize the results from our work with floorplanning for QCA circuits, applying developed floorplans to actual – and useful – designs, and the development of tools to simulate and design QCA circuits. Additionally, based on the results presented here and the rest of the work presented in this thesis, areas of future work will be presented and discussed in detail.

7.1 Oh the Places We've Gone

This research effort began in an effort to investigate the potential of a nanotechnology (QCA) as being a viable alternative to CMOS VLSI – particularly with regard to logic design issues. When work first began, only designs for a memory cell, a shift register, a simple adder, and a few other basic logical devices existed. However, as a result of this work, the following ideas and designs have been developed:

To begin, floorplanning techniques have been designed that not only allow designs to be efficiently constructed with QCA cells, but also exploit the four phase QCA clock. The floorplans that have been developed not only created designs that are efficient in area usage, but they also offer inherent pipelining and the potential for multithreading. Additionally, the floorplans were applied to the design of the dataflow of an actual microprocessor that was successfully implemented in QCA. The development of Simple 12 allowed design issues to be encountered first hand.

Namely, solutions for potential roadblocks such as latch/register implementation, ALU design, means for feedback, etc. were discovered. Most importantly, these solutions can be applied and scaled to more complicated designs.

Designs were not simply constructed and implemented in schematic form only. A simulator and design architect tool were written so that a means for verifying the logical correctness of a design (other than simply checking it by hand) would exist. The Simple 12 dataflow was then simulated with this design and was verified for logical correctness. The development of the simulator also helped develop a family of design rules (and design rules of thumb) for dataflows that include:

- The idea of dividing clocking zones in half that are logically in the same phase to minimize thermodynamic effects from having too many QCA cells in one clocking zone.

- Duplicating portions of intermediate signal generation logic to help eliminate wasted area and "long" wires.

- Using trapezoidal clocking to eliminate wasted area and to make denser circuits.

- Performing computation for "free" in "wire".

- Creating intrinsic latching by spreading a "wire" over four or more clocking zones.

So, how has research into floorplanning techniques for QCA designs, the actual development of QCA designs, and work with a QCA simulator paid off with regard to the original question? Namely, can circuits such as a microprocessor be constructed in QCA and what benefits will such a design give over CMOS? Well, as seen in Chapter 6, in the worst case (non-molecular QCA implementation), QCA offers an almost order of magnitude density gain over an equivalent CMOS design (at the end of the CMOS curve). In the best case (molecular QCA implementation), QCA offers a density gain of 900 over an equivalent CMOS design.

Finally, it would be appropriate to consider the physics of the QCA device itself. A meaningful metric to consider is the power-delay-product (PDP). The power-delay-product can be considered a quality measure for a switching device as it is simply a measure of the energy consumed by a gate per switching event. A graph of the PDP for various technologies and where QCA would fit into this mix appears below in Figure 7.1 [9]. As one can see, the PDP for QCA is well below that of any other technology.

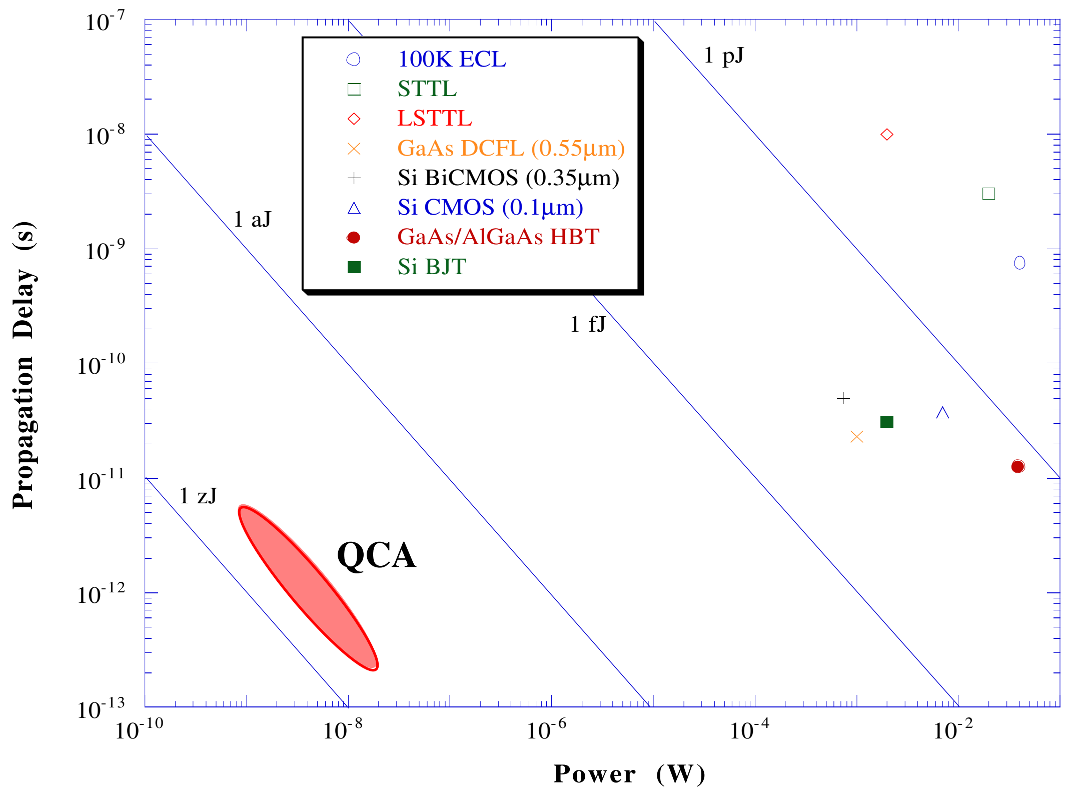

Figure 7.1. The power-delay-product for QCA and other technologies.

Given these amazing numbers, further research into the QCA technology and architecture is certainly warranted. Potential areas for future exploration and research are outlined in the next subsection.

7.2 The Future

Based upon the results discussed above, several topics have been identified as sources for future work and research. First and foremost, Simple 12 must be completed. This involves developing the control and state machine logic for it as well as studying how it might work with memory. Also, ways to improve routing must be developed. Potential problems stem from the existence of "long" wires in designs. Furthermore, an efficient and "safe" method of interconnect (i.e. one that avoids as much as possible the design problems of Chapter 3) must be considered. Additionally, novel architectural techniques should be considered. These include but are not limited to multithreading. Finally, some technology issues must be considered. One of the most important areas for investigation might be how a clocking zone would physically be implemented. Each potential area of future work is discussed in a subsection below (starting from the "bottom up").

7.2.1 Technology Issues

There are many opportunities for research when considering the "lowest levels" of QCA design. For example, means for actually implementing the clocking zones that are an integral part of all of the designs in this thesis must be developed and pursued. In particular, a method must be found that assures that there is a sharp "edge" between two clocking zones. Also, research efforts should be directed into determining how a specific logical value (i.e a binary 1 or 0) is actually "read-in" by the design or "read-out" by somebody or something in the outside world.

Additionally, specific methods and tools for developing power, area, and speed models for a given design should be researched and developed. Along with this, the effects of devices with "defects" and how they affect an overall design should be considered. Finally, while a rather thorough set of design rules for QCA circuits has been developed and presented here, work should continue to form a complete set of design rules that will apply to all possible QCA circuits (i.e. design rules for memory devices and control and state machine logic should be developed and enhanced).

7.2.2 Logic Design

Without a doubt, developing the specific control and state machine logic for Simple 12 will be where significant effort and research is next directed. It might actually be more appropriate to characterize this effort as a mix of research and design however as some of the necessities to accomplish this task are already in place. For example, at present, a sophisticated simulator exists for verifying the logical correctness of a given design. Additionally, some work has been done with simple QCA state machines. Also, complete state transition diagrams and the resulting control signal output for each state and opcode have already been determined. However, a mechanism to place and route the required logic will also have to be developed.

Based on the discussions of Section 4.4 (interconnect in the Simple 12 dataflow) and the discussion of the need for control signal routing techniques above, work in the immediate future will have an extensive focus on methods for efficient interconnect and routing. Obviously, floorplans are required to properly route and time signals from the control logic block (as well as from memory). However, more importantly, floorplanning techniques must be studied further in an effort to eliminate long wires and crowded clocking zones that currently result from the interconnect between two bit slices of the Simple 12 dataflow.

One potential solution discussed in Section 4.4.1 (Interconnect Clocking Zone Width and Wire Length) involves stacking small clocking zones vertically between bits (see Figure 29) in an effort to "break-up" long wires. However, such a solution would appear to add an otherwise unnecessary delay to a given design. Additionally, all of the routing wires would have to fanout from the clocking zone in the middle of Figure 29 so that all of the inputs to the dataflow were synchronized. While this is possible, it may become something of a routing nightmare. Nevertheless, this as well as other solutions will be explored.

Methods for developing both simple and complex memory structures must also be created and enhanced. As seen in Chapter 4, current designs use a means of intrinsic latching to store a value in a "register" for some indefinite period of time. Such a device seems to work quite well for a circuit such as the Simple 12 dataflow and would seemingly scale to larger and more complex designs as well. However, more complex memory devices must also be developed. For instance, some type of main memory structure (possibly analogous to an SRAM) will be needed. Such a structure will have to overcome and solve the timing issues sure to be associated with it – in particular those related to connecting memory associated with a cache, for example, to the actual dataflow.

Finally, research should focus on developing reconfigurable QCA logic arrays. With such arrays, the structure of a single chip is a repetition of some more basic multi-device macro. This in turn can be programmed either after manufacture or at power-up time to perform different logic functions. Creation of such design points would allow real device developers to focus on a small number of multi-device combinations, which if feasible, could be replicated into real systems.

7.2.3 Architecture

Because of the inherent pipelining created by the clocking zones of a QCA design, QCA is an ideal candidate for multithreading. In an ideal situation (i.e. assuming that all control signal and data routing timing and delay problems can be worked out successfully), new data bits for a separate instruction stream could enter the dataflow every time a clocking zone bordering the inputs recycles back to the switch phase (or in other words it goes through each of its four phases). This would result in a remarkable throughput.

QCA may also be an ideal candidate for other alternative architectures as well – a combinator-based PIM like architecture for example. In a combinator system, the key functions implemented are simple ones that only re-arrange their operands. In fact, it has been shown that only two simple functions are needed to compute any computable function [7]. Given the natural pipelined flow of information in a QCA wire, it appears that many of the basic combinator functions can be implemented simply in the access logic next to memory, with no degredation in speed as data moves to its ultimate destination.

Finally, it is worth mentioning that architectural techniques described above can be applied not only to general purpose processing, but special purpose processing as well. The possibilities of applying the QCA technology to special purpose processing (such as digital signal processors) will also be explored.

7.2.4 Design Automation Tools

Finally, as discussed in Section 5.9, work will also be done to enhance design automation tools. Research and design efforts in the immediate future (currently underway) will focus on improving the "user-friendliness" of Q-BERT. Other future work in the area of QCA design automation will center around letting the engine/simulator define clocking zones. Ideally, the user would simply indicate where specific clocking zone boundaries would occur and the simulator/engine would construct their borders. Finally, the possibility of writing a program that would convert conventional boolean logic (i.e. a schematic containing AND gates, OR gates, NAND gates, XOR gates, etc.) to a schematic consisting only of optimized majority gate logic should be explored.

List of References

[1] In *The National Technology Roadmap for Semiconductors*. Semiconductor Industry Association, 1997.

[2] In *Technology Roadmap for Nanoelectronics*. European Commission IST Programme Future and Emerging Technologies, Microelectronics Advanced Research Initiative, 1999.

[3] I. Amlani. Digital logic gate using quantum-dot cellular automata. *Science*, 284:289, 1999.

[4] I. Amlani, A.O. Orlov, G.L. Snider, and C.S. Lent. Experimental demonstration of a binary wire for quantum-dot cellular automata. *Appl. Phys. Lett.*, 72:2179, 1999.

[5] G. H. Bernstein, G. Bazan, M. Chen, C. S. Lent, J. L. Merz, A. O. Orlov, W. Porod, G. L. Snider, and P. D. Tougaw. Practical issues in the realization of quantum-dot cellular automata. *Superlattices and Microstructures*, 20:447–459, 1996.

[6] D. Berzon and T. Fountain. Unpublished.

[7] P. M. Kogge. *The Architecture of Symbolic Computers*. McGraw-Hill Series in Supercomputing and Parallel Processing), New York, 1990.

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

[9] Courtesy of Dr. Craig Lent of the University of Notre Dame Electrical Engineering Department.

[10] J.M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall Electronics, New Jersey, 1996.

[11] C.G. Smith. Computation without current. *Science*, 284:274, 1999.

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