2010

Implementation of multi-CLB designs using quantum-dot cellular automata

Chia-Ching Tung

Implementation of Multi-CLB Designs Using Quantum-dot Cellular Automata

by

Chia-Ching Tung

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science
in Electrical Engineering

Supervised by

Assistant Professor Dr. Eric Peskin
Department of Electrical and Microelectronic Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
February 2010

Approved by:

Dr. Eric Peskin, Assistant Professor
Thesis Advisor, Department of Electrical and Microelectronic Engineering

Dr. Dorin Patru, Assistant Professor
Committee Member, Department of Electrical and Microelectronic Engineering

Dr. Dhireesha Kudithipudi, Assistant Professor
Committee Member, Department of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

Implementation of Multi-CLB Designs Using QCA

I, Chia-Ching Tung, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Chia-Ching Tung

Date

Acknowledgments

I am grateful for my parents' unconditional support for me to finish this thesis.

Abstract

Implementation of Multi-CLB Designs Using Quantum-dot Cellular Automata

Chia-Ching Tung

Supervising Professor: Dr. Eric Peskin

CMOS scaling is currently facing a technological barrier. Novel technologies are being proposed to keep up with the need for computation power and speed. One of the proposed ideas is the *quantum-dot cellular automata* (QCA) technology. QCA uses quantum mechanical effects in the device at the molecular scale. QCA systems have the potential for low power, high density, and regularity.

This thesis studies QCA devices and uses those devices to build a simple *field programmable gate array* (FPGA). The FPGA is a combination of multiple *configure logical blocks* (CLBs) tiled together. Most previous work on this area has focused on fixed logic and programmable interconnect. In contrast, the work at the Rochester Institute of Technology (RIT) has designed and simulated a *configurable logic block* (CLB) based on *look-up tables* (LUTs). This thesis presents a simple FPGA that consists of multiple copies of the CLB created by the RIT group. The FPGA is configured to emulate a ripple-carry adder and a bit-serial multiplier. The latency and throughput of both functions are analyzed. We employ a multilevel approach to design specification and simulation. QCADesigner software is used for layout and simulation of an individual CLB. For the FPGA, the high-level HDLQ Verilog library is used. This hybrid approach provides a high degree of confidence in reasonable simulation time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As of 2009, the transistor density or the number of transistors on an *integrated circuit* (IC) chip is approaching a limit. The semiconductor industry is still able to fabricate devices exponentially smaller according to Moore's law [25], but as transistor sizes are shrinking to the nanometer scale, it becomes extremely difficult to continue packing more and more computational power into our microprocessors. In the near future, the operation of transistors will be dominated by the physics of the quantum world. Conventional *complementary metal-oxide-semiconductor* (CMOS) transistor is facing its physical limitations [8, 9] related to power dissipation, interconnects, and fabrication. Research on a novel technology to replace the current technology is necessary. The new method has to take advantage of quantum effects and nanoscale physics in order to keep up with the ever growing need for computation power and speed.

1.1 Motivation

Currently, CMOS is the mainstream technology for fabricating chips. As more transistors are being put into the processors, the issues related to doping regions, oxide thickness, diffusion barriers, power dissipation, and leakage current *etc.* become difficult to scale. All these issues call into question the feasibility of CMOS in the future.

There is much ongoing research to find a way to replace the standard CMOS technology. *Quantum-dot cellular automata* (QCA) [18], *carbon nanotube transistors* (CNT) [3], *silicon nanowires* [12], *etc.* are all possible technologies. The international technology roadmap for semiconductors

(ITRS) [1] describes the difficulty that the current semiconductor technology will face. The biggest concern that the CMOS is facing is the difficulty on setting more and more devices on the chips. Approximately at 25 nm technology the fabrication process might fail to achieve this goal [8]. Quantum mechanical effects have an important role in device performance and induce uncertainty. QCA is an emerging nanotechnology that provides a revolutionary approach to substitute for the traditional CMOS technology for *very-large-scale integration* (VLSI) fabrication. QCA technology addresses the inevitable nanoscale level issues, such as interaction between devices and operation of devices. The interaction between devices at the nanoscale level is a roadblock on further scaling of the CMOS devices. QCA also has the following potential advantages: low power consumption, higher device density, and implementation on the nanoscale level.

The current ongoing research in the field of QCA has two directions. One direction is to find a method for stable fabrication of QCA devices. In order to compete with CMOS technology, the device must have low cost, be able to operate at room temperature, and be able to handle high clock frequencies. Some significant examples of such research are [4, 10, 21, 23, 24]. Three types of QCA fabrication process have been proposed, *metal-dot QCA* [21], *molecular QCA* [23], and *magnetic QCA* [5]. However, there has not been a successful attempt to create a testable circuit of molecular QCA, and only small circuits have been fabricated by metal-dot QCA, and magnetic QCA.

The other direction of research is on the *nanoarchitecture* level [34], which is the primary focus of this thesis. This area of research has focuses on using QCA technology for circuit implementation. Taskin *et al.* suggest that the general nanoarchitecture design is able to be studied by modeling QCA dots at a higher level of abstraction [34]. The high-level design is independent of the manufacturing technology. Unlike power dissipation, area of the device, and operating clock frequency, the architecture level does not require hardware manufactured devices to perform experimental testing.

Field programmable gate arrays (FPGAs) [6] are reconfigurable logic devices. FPGAs can implement any combinational or sequential functions. FPGAs consist of programmable logic or programmable interconnect or

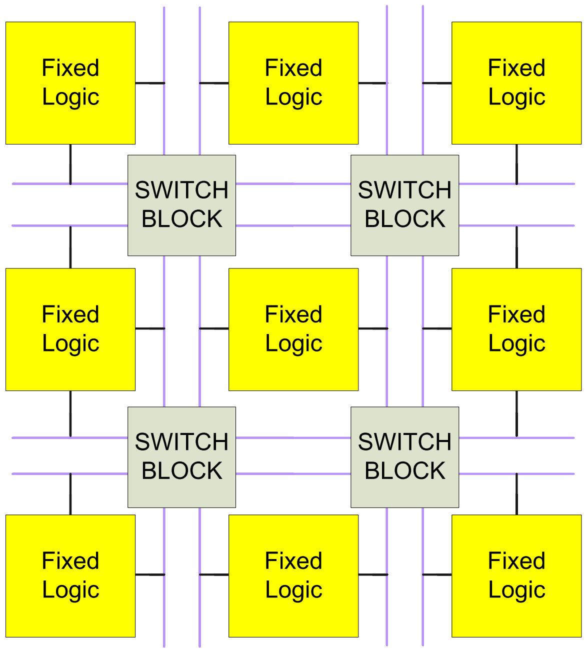


Figure 1.1: Fixed logic FPGA.

both. The logic blocks have the ability to compute complex functions of given inputs. The outputs of the logic blocks are connected and wired together. FPGAs have a regular pattern in design, which may facilitate self-assembly when fabricated in nanotechnology. FPGAs can emulate many applications. If an FPGA can be built using QCA technology, it would allow many applications to run on QCA. Therefore, creating an FPGA using QCA technology is an interesting application of QCA.

FPGAs mainly consist of an array of multiplexers, *look-up tables* (LUTs), memory cells, and special hardware blocks. As mentioned in the previous paragraph, FPGAs depend on programmable logic, programmable interconnect, or both [7,46]. FPGAs with fixed logic consist of a sea of fixed logic elements interconnected by a mesh of wires. The programmability is made at the point of intersection of the wires. Figure 1.1 shows a general example of fixed-logic-type FPGA. The switch blocks decide the signal routings based on the users' configurations. Figure 1.2 shows an FPGA with programmable logic. The inputs and the outputs on each side of the block are connections to the next nearest block. Each block is a *configurable logic*

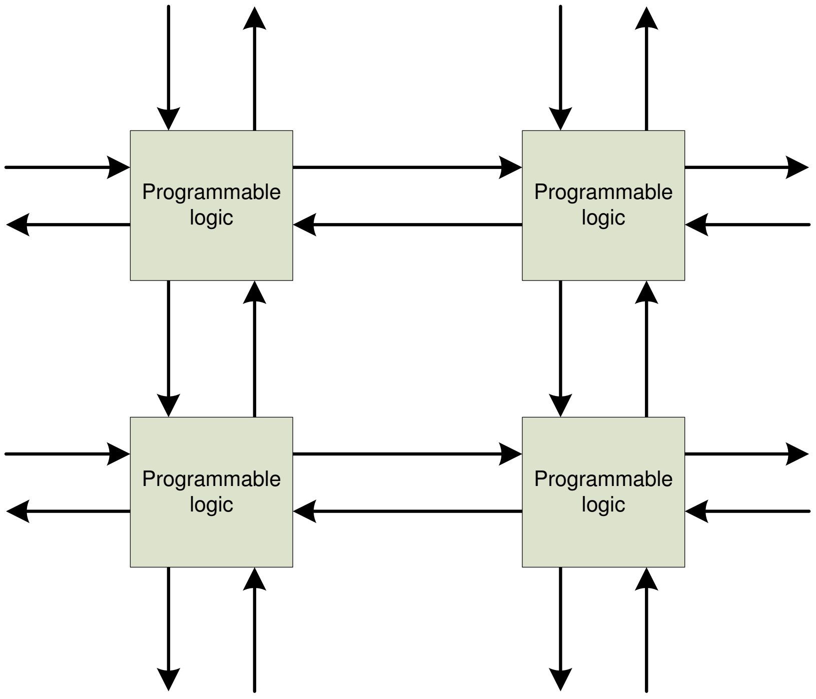


Figure 1.2: FPGA with programmable logic.

block (CLB) with fixed connections to the neighboring CLBs. Most commercial FPGAs consist of both types of circuits. The configurable logic blocks are linked by programmable interconnect. Programmable interconnects are switching blocks that can be programmed to give various connections between the logic blocks for different functions. Such an arrangement is shown in Figure 1.3. The black lines in Figure 1.3 represent the interconnections between the CLB and the switch blocks.

This thesis focuses on the programmable-logic method. This method achieves the goal through the use of LUTs. The LUT consists of a 16-bit memory and two 2-to-4 decoders. It is capable of serial write and parallel read operations with low read latency.

1.2 Previous Work

Previous work on QCA-based FPGAs has focused on programmable interconnect [2,15,26,27,39]. Niemier *et al.* [26,27] present a QCA-based FPGA

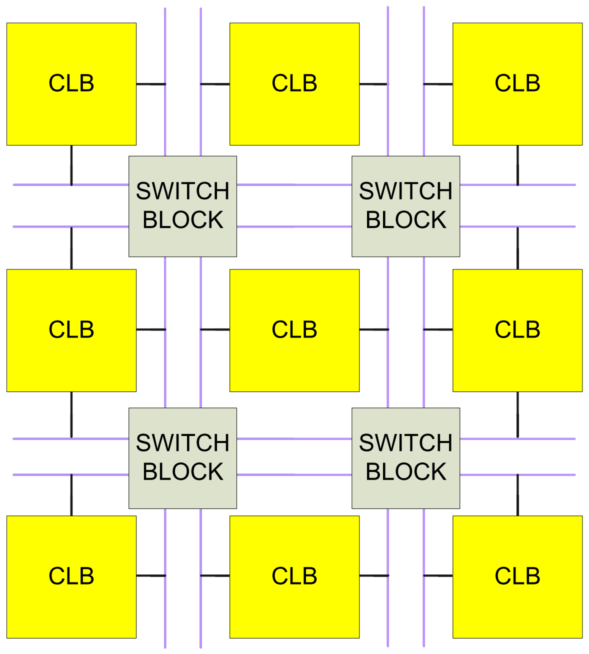


Figure 1.3: FPGA with programmable logic and programmable interconnect.

based on programmable interconnect. In [27], authors develop an interconnect design with flexible routing. The interconnects between each block are operated by QCA cells. If the connection is to be established, the clock signal is applied to the interconnect to make the QCA cells operate. When the clock is not given to the QCA cells, the connection is not established, and the cells act as open circuits. The logic block in [26,27] consists of a NAND gate. The size of logic blocks for data processing are equal to the size of the interconnect blocks. This allows the shape of the FPGA to be more compact and organized. These articles discuss the use of programmable interconnect with fixed logic, which is the FPGA architecture represented in Figure 1.1.

Amiri *et al.* [2] created a tree of multiplexers to be the fixed logic cell for the FPGA. The fixed logic presented has six inputs and one output. Two of the six inputs are the control signals that decide which inputs will be routed to the output. The module can emulate any two-input or three-input logic function such as NAND, AND, OR, and NOR. The authors believe their proposed circuit could operate at terahertz frequencies. Similar to [27], the directions of all input signals are controlled by one or two control signals.

Jazbec *et al.* [15] designed a *programmable switch matrix* (PSM) as the programmable interconnect in the FPGA. The PSM allows the logic blocks of the FPGA to be interconnected when needed. The PSM acts as a controller in the system to decide the direction and the destination of all the signals.

In contrast to the above articles, Lantz and Peskin [16] explored the method of building FPGA by programmable logic. A novel CLB design for FPGAs implementation in QCA has been presented. The CLB is built by using four *look-up tables* (LUTs), each of them corresponding to one of the four directions: north, south, east, and west. Each LUT is equipped with a decoder and 16-bit memory blocks. Each LUT is capable of taking four inputs and producing one output. The complete CLB is formed by joining four LUTs together. The CLB has separate inputs for data and for LUT reconfiguration that may be used independently. Therefore, it is possible to reconfigure the memory of the LUT while data is being processed. This CLB has four inputs and four outputs that can also combine with other CLBs or stand alone to perform digital functions. A disadvantage of the design is its high latency. The read operation requires 30 clock cycles. An FPGA is made up of an array of CLB blocks. Each CLB is capable of implementing an independent function. The output of the CLBs can be combined to implement a complex function of the input variables. The signals will traverse multiple CLBs. Therefore, the latency will accumulate and become an important issue. Optimizing to reduce the latency and area of the design is necessary.

Rungta [32] presents an improved version of CLB from work [16]. The latency is reduced from 30 clock cycles down to 16 clock cycles. The throughput performance of one read per cycle is still maintained from [16]. In contrast with [16], the memory cells of the LUT are arranged in a two-dimensional array. This gives a rectangular LUT. The result of this is that a given input signal traverses fewer clock zones before it reaches the input of the logic block. The interconnect requires the most latency, but the area structure of the design can be more compact, which also helps to improve the latency and input signal routing issues. Unlike Lantz and Peskin's proposed CLB with a tree-based structure which results in a large unused area, Rungta [32] designed the CLB with linear row and column decoders like

CMOS memories to avoid unused area. The FPGA studied in this thesis will be an array of multiple copies of the CLB presented by [32, 36].

1.3 Contributions

In contrast to [2, 15, 26, 27, 39]. This thesis presents an FPGA based on configurable *logic*. In contrast to [16, 32], this thesis creates a small FPGA consisting of *multiple* CLBs.

1.3.1 Proposed Design

This thesis presents the design and simulation of a CLB-based FPGA using QCA technology, and applications along with the FPGA. The FPGA in the thesis is considered to be the composition of *configurable logic blocks* (CLBs). The FPGA architecture presented in the thesis is formed through tiling multiple CLBs together with nearest-neighbor connections only. The primary inputs and outputs of the FPGA are on the outer perimeter of the FPGA. Unlike other types of QCA-based FPGAs, which focus on programmable interconnect [15, 27], the presented FPGA is formed using programmable logic. The FPGA can be designed in any size and for different purposes. Each CLB consists of four LUTs and has four pairs of inputs and outputs associated with it. QCADesigner [41] is used for layout and simulation of the design. However, when the layout design requires a significant amount of time to simulate, another tool is needed to reduce the simulation time. HDLQ [29] is a Verilog library, which is also used for simulation purposes. The results from both QCADesigner and HDLQ match each other. A *ripple-carry adder* (RCA), a *bit-serial multiplier* (BSM), and glitchless reconfiguration are demonstrated on the proposed FPGA. The proposed FPGA has not been fabricated. However, the simulation results match the expected results.

1.3.2 Applications

Every device needs to be tested by applications to ensure its reliability. The applications designed for this simple FPGA are the following: 4-bit *ripple carry adder* (RCA), 4-*bit serial multiplier* (BSM), and *partial reconfiguration*. To speed up simulation, *HDLQ* [29] is used. Each level of the design drawn in QCADesigner is also modeled and re-simulated using HDLQ Verilog library. For levels up to the CLB, the layout is simulated in QCADesigner and the HDLQ model is simulated using ModelSim. The results match bit for bit and cycle by cycle. The designs with multiple CLBs are only simulated using the HDLQ model in ModelSim. For each application, the results match the expected results.

1.4 Organization

This thesis is organized as follows: Chapter 2 provides the QCA background for an understanding of QCA technology. It also includes a brief discussion of various types of implementation techniques of QCA devices. Chapter 3 presents the architecture of the completed FPGA. It also describes the necessary components embedded within it. Chapter 4 presents simulated results for the applications of the FPGA. Chapter 5 concludes this thesis and discusses directions for future work.

Chapter 2

QCA Background

The concept of *quantum cellular-dot automata* (QCA) was introduced by Tougaw and Lent [18] in 1993. They proposed this new technique to be an alternative method for fabricating electronic devices. The concept was very theoretical in the beginning. Many mathematical equations were used to prove its feasibility and strengths. In the following decade, this topic drew more attention and improved in both design and fabrication.

QCA technology provides a computation method which is different from that used in traditional transistor-based circuits. This section explains the basic operation of QCA technology and its associated components, such as a cell, wire, majority gate, and inverter. The layout designs of the QCA circuits are the combination of all the mentioned components. More information can be obtained in [18, 22, 41]. Mainly, there are two categories of current QCA research, *physical implementation* and *nanoarchitecture*. This thesis is focused on the nanoarchitecture. The logic design is independent of the implementation technique. Once the implementation becomes more stable for fabrication, the design can be set into a real hardware device.

2.1 Basic QCA Operation

This section describes the basic principles of QCA operation, starting from a single QCA cell, and on to a QCA wire, and then logic gates. Finally, the basic clocking scheme is also described.

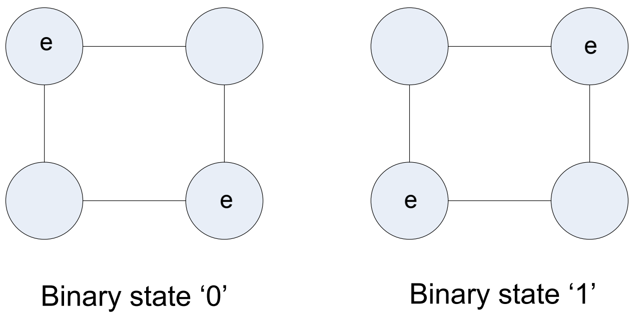


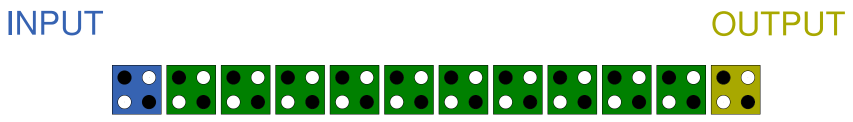
Figure 2.1: Binary interpretation of the states of a QCA cell.

2.1.1 QCA Cell

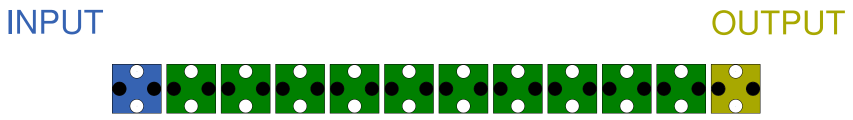
QCA circuits are made up of QCA *cells* [20]. Generally, each QCA cell contains four electron wells and two electrons. The electron wells are held at a low potential and are coupled to each other by tunnel junctions. The interpretation of the states of a QCA cell is dependent on the position of the electrons in the electron wells. The electron position is calculated by a Schrödinger equation through a quantum-mechanical Hamiltonian [17]. The electrons repel each other to opposite corners of the cell. This gives two stable configurations — one for each diagonal. One diagonal is used to represent a binary '0.' The other diagonal is used to represent '1.' Figure 2.1 shows the two stable configurations and their binary interpretation.

2.1.2 QCA Wire

The QCA cells can be placed next to each other to form a chain of QCA cells. This chain is called a QCA *wire*. Figure 2.2(a) shows an example of a QCA wire carrying a logic '0.' Each cell is polarized to "-1" due to Coulombic interaction. It can be seen in the first cell that the occupied dots in the cell distance themselves from the other occupied dots in the cell. The second cell reproduces the logical value from the first cell and passes this on to the next one. This process is called a *domino effect*, in which all cells adopt the same polarization. Figure 2.2(b) shows the same wire carrying a logic '1.'

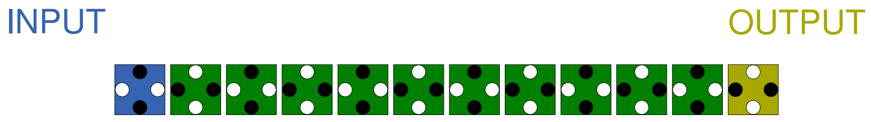


(a) Output = Input = ‘0.’

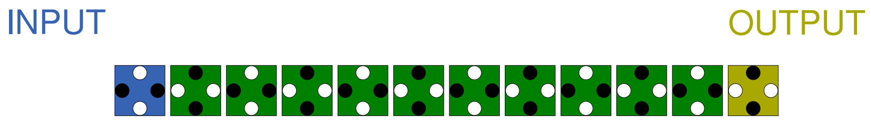


(b) Output = Input = ‘1.’

Figure 2.2: QCA normal wire.



(a) Even cells ‘0,’ and odd cells ‘1.’



(b) Even cells ‘1,’ and odd cells ‘0.’

Figure 2.3: QCA inversion chain.

Besides the normal QCA wire shown in Figure 2.2, there is another type of QCA wire called *inversion chain*. Figure 2.3 demonstrates the two stable configurations of the QCA inversion chain with (a) logic ‘0’ and (b) logic ‘1’ inputs. All the cells in the inversion chain are rotated 45 degrees with respect to the normal QCA cells. A QCA inversion chain consists of an array of 45-degree orientated cell. In the inversion chain, the neighboring cells are aligned in opposite polarization for stable configuration. As the information is being transmitted, the polarizations alternate between ‘1’ and ‘-1’. This means that the QCA inversion chain can be considered as a serial chain of inverters. A normal QCA cell can be tapped off the inversion to obtain buffered or inverted outputs.

2.1.3 Wire Crossing

The feasibility of multi-layer layout in QCA technology has not yet been proven. However, co-planar wire crossing may be feasible in QCA. As

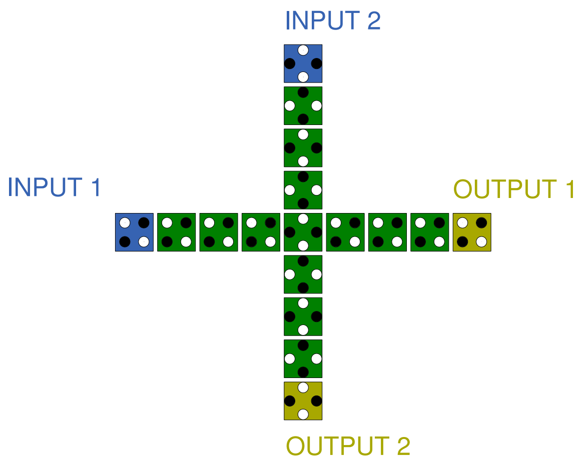


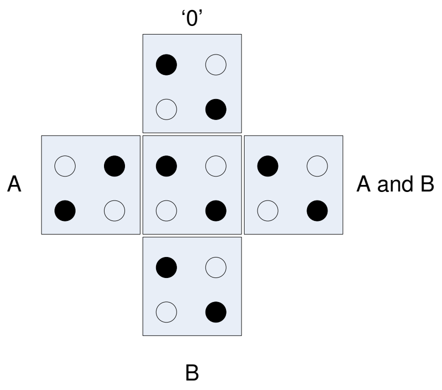
Figure 2.4: Wire crossing.

mentioned in Section 2.1.2, there are two types of QCA wires, normal wire and inversion chain. It is possible to have wire crossing between a normal wire and an inversion chain, because the signals will not interfere with each other. The cell at the intersection must be an inverted cell. Figure 2.4 shows a wire crossing between a vertical inversion chain and a horizontal normal wire.

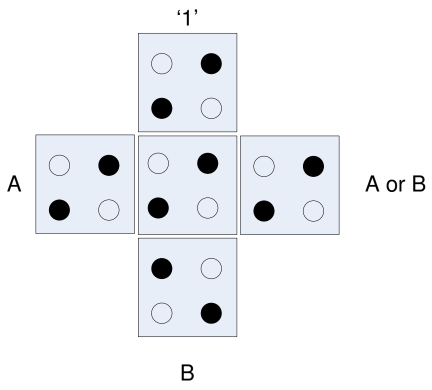
2.1.4 Majority Gate and Inverter Gate

One of the major components in QCA technology is the 3-input *majority gate*. The Boolean function of the majority gate is $F = AB + BC + AC$. If two of the inputs are fixed at ‘0’, the output will be ‘0.’ If two of the inputs are fixed at ‘1,’ the output will be ‘1.’ The majority gate can implement a 2-input AND or a 2-input OR gate by setting one of the inputs to constant ‘0’ or ‘1.’ Figure 2.5 (a) shows the majority gate configured as a 2-input AND gate, and Figure 2.5 (b) shows the 2-input OR gate. Table 2.1 shows the configuration needed to implement a 2-input AND and a 2-input OR gate.

Another important component is the *inverter*. Figure 2.6 shows QCA schematic of the inverter. The function of the inverter is to change the input value from either logic ‘0’ to logic ‘1’ or from logic ‘1’ to logic ‘0.’ In an inverter gate setup, the input wire is set as a two-prong fork. The output wire is aligned vertically between the two prongs of the fork on the edge.



(a) AND gate



(b) OR gate

Figure 2.5: QCA majority gate.

Table 2.1: Truth table for majority gate.

(a) with $C = 0$, $F = A \wedge B$

C	B	A	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1

(b) with $C = 1$, $F = A \vee B$

C	B	A	F
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

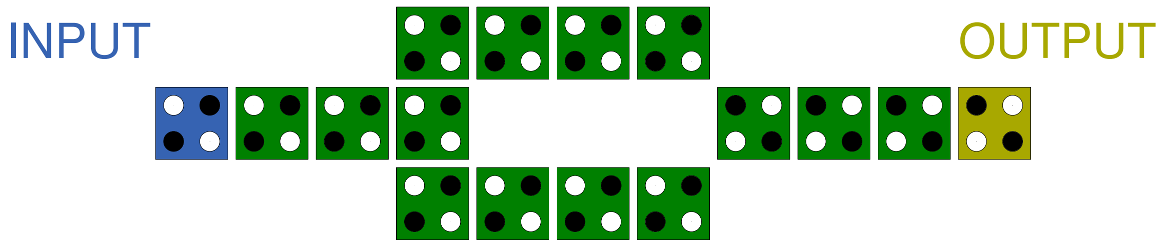


Figure 2.6: QCA inverter.

The logic value is passed to the edge of the two-prong fork. The output is driven by the diagonally opposite force from the two-prong fork input. The logic value of output is then forced to be opposite to that of the input by the Coulombic interaction.

2.1.5 QCA Clock

The electron in the electron well requires high potential energy to tunnel through the junction. This potential energy is provided by the QCA clock. In this thesis, a four-phase clock system is adopted. The electrons are allowed to move between the wells when the clock is high, because the potential barrier is low. On the other hand, the electron is trapped in the well when the clock is low due to the raised potential barrier. Electrons will then assume the polarization of their neighbors. QCA circuits adopt a four-clock-phase technique to power the circuits.

CMOS clocking is implemented adiabatically to reduce the power dissipation. Typically, four-phase clocking is used for QCA logic synchronization. The four phases are *switch*, *hold*, *release*, and *null*. The clock wires are embedded directly under the physical layer of QCA cells. In the *hold* state where the clock is low, the QCA cell is holding the current position of the electrons. The rising edge of the clock is the *release* state, where electrons tunnel down to the bottom of the QCA cell due to the electrons attraction. When the clock is high, the QCA cell enters the *null* state. At the falling edge of the clock, QCA enters the *switch* state where the electrons are repelled from the clock plane and tunnel to other positions. Figure 2.7 shows the clock mechanism for four phase clocking. It is a shifting movement over time. In the figure, s stands for switch phase. h stands for hold phase. r

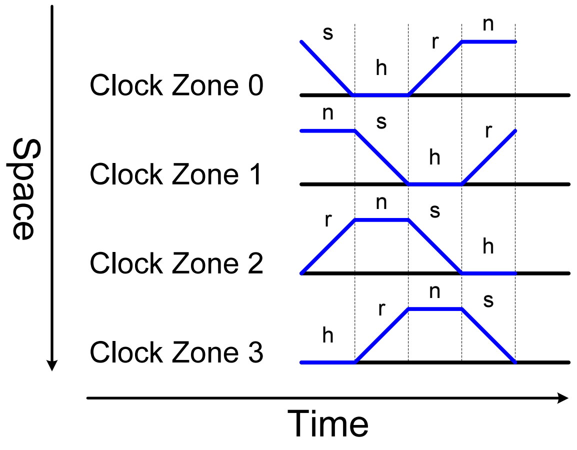


Figure 2.7: Clock zones and phases.

stands for release phase. Finally, n stands for null phase.

The four clock zones are zone 3, zone 2, zone 1, and zone 0. Figure 2.7 shows the clock zone signals behavior. In a QCA circuit, each cell must be assigned to a clock zone. Although all the cells will experience the four phases of the clock over time, each cell still needs to be given a specific clock zone in the design. The arrangement of clock zones determines the direction of information flow. The direction of information flow is from zone 0 to zone 1 to zone 2 to zone 3, then back to zone 0 of the next clock period. The clock signal in all QCA-based circuits has to be a global signal where all cells share the same clock. Unlike CMOS technology, the clocking of QCA is implemented underneath the layer upon which QCA cells are placed [34].

2.1.6 Design Rules

Certain restrictions must be met for correct QCA operation. This section describes some of these conditions. I believe these conditions are fundamental to QCA. However, these conditions were observed during QCADesigner simulation. These rules have not been tested in a fabricated hardware device.

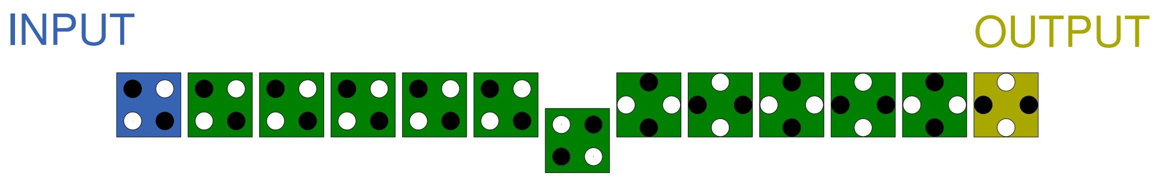


Figure 2.8: Normal wire connects to inversion chain.

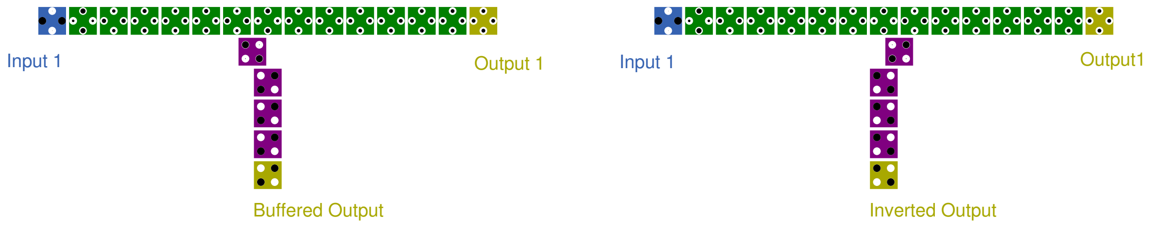


Figure 2.9: Inversion chain tapping off.

Length of Clock Zones If there are more than 10 cells in the single clock zone, the signal seems to be inaccurate. On the other hand, a minimum of two cells in a single clock zone is required. This minimum of two is sufficient for QCADesigner. However, for ease of fabrication, it may be necessary to group more cells in each clock zone [37].

Normal and Inversion Chain Connection In the design process, connections might be required between a normal wire and an inversion chain. The connecting cell needs to be translated by half a cell width in order for the desired signal to transfer properly. Figure 2.8 shows an example of how such a connection can be established.

Inversion Chain Taps Off An inversion chain is a special type of QCA wire. Each cell within the inversion chain takes an opposite polarization from its previous cell. Therefore, a buffered or an inverted signal can be obtained from inversion chain, depending on the location of tapping off. Figure 2.9 demonstrates examples of such behavior. A normal cell shifted by half a cell width to meet with the inversion chain. The location of the tap determines whether the output is buffered or inverted.

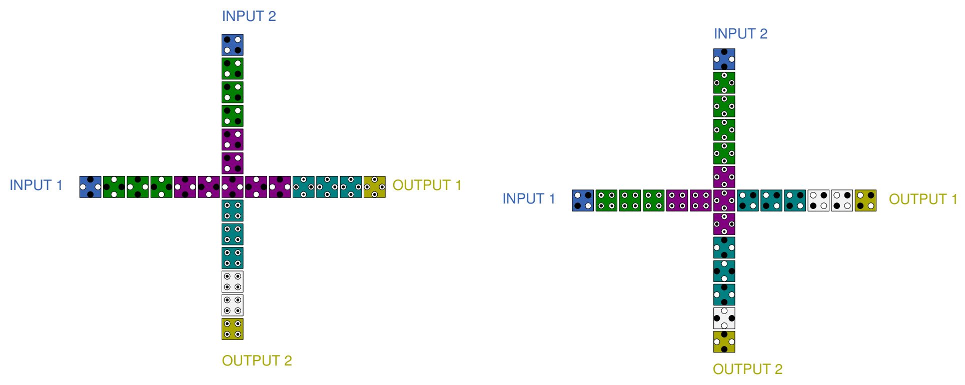


Figure 2.10: QCA wire crossing between normal and inversion.

Normal Wire Crossing Inversion Chain Wire crossing between normal wires and inversion chains is complex. During design process, a couple of behaviors were explored during the simulation. Some restrictions were developed to ensure the correct simulation.

1. The intersection cell should be an inversion chain cell.
2. A minimum of two cells of the normal wire before the intersection should have the same clock phase as the inversion cell at the intersection.
3. A minimum of one additional inversion cell on each side of the intersection must be the same clock phase. Figure 2.10 illustrates these minimal phase concepts.
4. The normal chain should move to the next clock phase after crossing the intersection.

2.2 QCADesigner

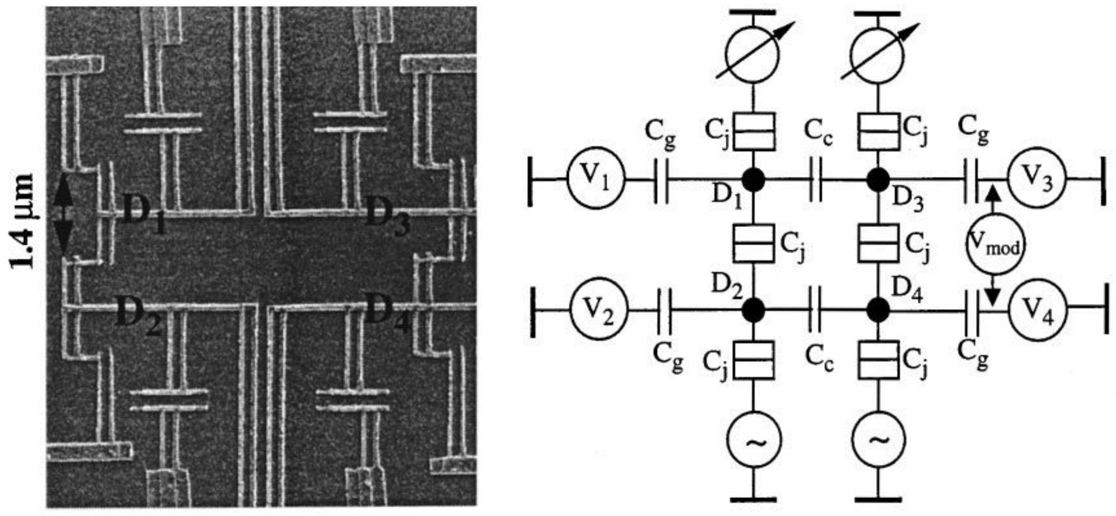
QCADesigner is an easy and useful program to design and simulate QCA circuits. QCADesigner is not just a switch-level simulator. It simulates QCA using the quantum mechanics of QCA. It has a friendly *graphic user interface* (GUI). QCA cell and wire are already a tool icon on the GUI. Once the cell is placed in the schematic, it is easy to change its clock zone or the rotation of the cell. By double clicking on the cell, the user can name the cell to be either input or output. To simulate the circuit, there are two steps to do. The first step is to set up the *simulation engine*, and the second step is to set up the *simulation type*. Simulation engine has two choices, *coherence vector* and *bistable*. This thesis uses the bistable simulation engine for all the proposed designs. The simulation type is to allow the user to set up the input test vectors. User can add in more test cycles in the table. The more cycles there are, the more boxes there are for clicking. Checked boxes represent an input of '1', and unchecked boxes represent input of '0.' Once all the set up is completed, choose *start simulation* from the simulation menu on top to start the process. There are more details and a user guide from [40]. However, there are still certain limitations of QCADesigner.

2.2.1 Limitations

All the proposed designs are simulated in QCADesigner version 2.0.3. The simulating computer is equipped with 3 GB of memory and a Pentium(R) 4 3.2 GHz CPU and works under Windows XP professional with service pack 3. The time required to simulate for one CLB is approximately 12 hours. The total cells used are 22,558. The shape of the design was constructed to be a rectangle with area of 48.41 μm^2, but the occupied area is only 7.31 μm^2. This is the largest scale design possible for simulation. When tiled multi-CLB together and perform simulation in QCADesigner, the error message always occurs to state that "out of memory." During the design process, there are certain things that were explored in the simulation such as number of cells in the clock zones.

2.3 QCA Implementation

Three types of QCA implementation have been proposed. The first one is called *metal dot junction QCA* [4, 21], the second type is called *molecular QCA* [23, 24], and the last type is called *magnetic QCA* [10]. Each implementation method has its own advantages and disadvantages and yet none



(a) (b)

Figure 2.11: Metal dot QCA cell: (a) SEM image, and (b) schematic diagram. Reproduced from [28] with permission of the publisher.

has been completely developed.

2.3.1 Metal Junction QCA

Metal junction QCA [4, 21] was the first fabrication technique developed to demonstrate the concept of QCA. It was not intended to compete with the current CMOS technology in the sense of speed and practicality. The basic idea of metal dot QCA is to build quantum dots using aluminum islands. The cell size is approximately 60 nm by 60 nm, with junction capacitance of 400 aF [4]. The method has the advantages of an easier fabrication process, reliability, and ease of modeling and analyzing. However, it has one major drawback, which is the operating temperature. The prototype only operates at $10°K$ or below. The required quantum-mechanical effects only happen at this operating temperature. Metal dot QCA is meant as a proof-of-concept implementation. Figure 2.11 shows a *scanning electron microscope* (SEM) image of the metal dot QCA and its diagram.

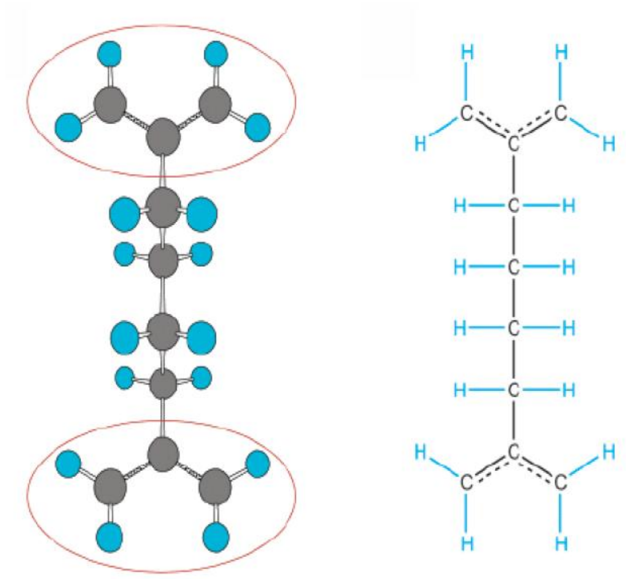


Figure 2.12: Two views of a molecule as a QCA cell. Reproduced from [19] with permission of the publisher.

2.3.2 Molecular QCA

To address the issues of metal junction QCA, *molecular QCA* was proposed [23, 24]. In *molecular QCA*, each device is built by molecules. The basic concept of molecular QCA is that each molecular QCA cell consists of a pair of identical allyl groups as shown in Figure 2.12. The molecule shown in Figure 2.12 is also known as a 1, 4-diallyl butane radical cation. This is formed by two allyl groups connected by a butyl bridge in between. This molecule is neutral on one end and the other end behaves as a cation.

This molecule has an extra hole or electron that allows the quantum tunneling effect needed by QCA to happen. If an electrical field is placed near one end of the molecule, it can create either a repelling or attracting force. It has been calculated that the molecule in Figure 2.12 has nonlinear switching characteristics, which make it an ideal switch. When the molecules are placed next to each other with a distance of seven angstroms, the electrostatic interaction will cause the holes to be at opposite ends, which makes the propagation of the electron feasible to create the state of the QCA cell. Figure 2.13 shows the different states of the molecular QCA. Part (a) is a +1 state, part (b) is a non-ideal state which is not needed, and part (c) is a -1

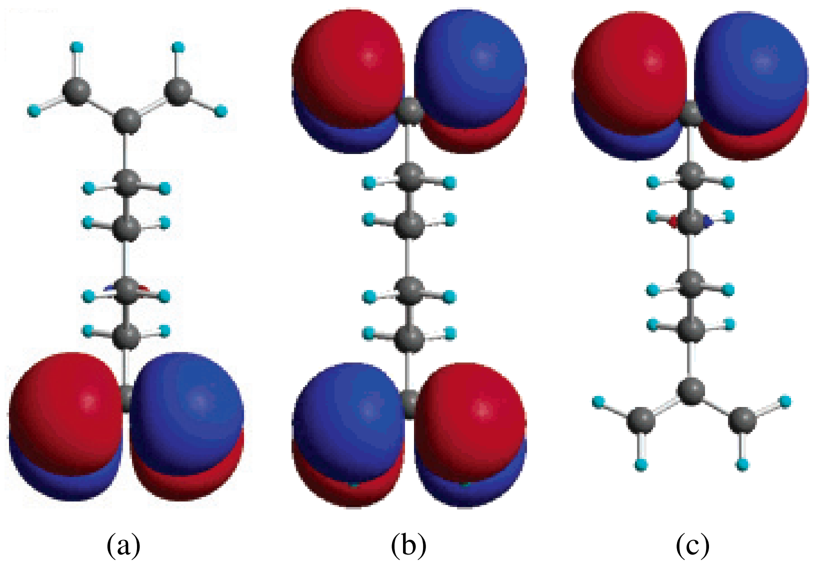


Figure 2.13: Different states of a molecular QCA cell: (a) +1 state, (b) non-ideal state, and (c) -1 state. Reproduced from [19] with permission of the publisher.

state. At this scale, the required quantum-mechanical effects can happen at room temperature.

Molecular QCA is believed to have the following advantages: high density, high clock frequency from the gigahertz range to the terahertz range, low power consumption, low power loss. An individual molecular QCA cell has been demonstrated. However, no complete circuit using molecular QCA has yet been demonstrated [43].

2.3.3 Magnetic QCA

A *magnetic QCA* (MQCA) cell consists of a single circular nanodot [10, 13, 14, 30]. A magnetic *Supermalloy* (mainly Ni) [10] is used to create the nanodot. Each nanodot is 110 nm in diameter and is 10 nm in thickness. Nanodots are placed 20 nm apart on a straight line. The basic operation is to use an oscillating field on the dot to have it point to a certain direction to represent the binary value. Magnetic QCA cell is capable of operating at room temperature. Other advantages include high density and low power loss. The operating frequency is low (in the MHz range) when

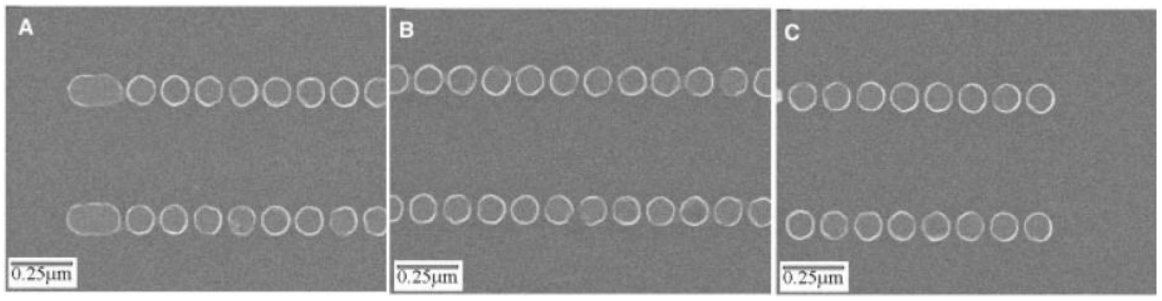


Figure 2.14: SEM image of a fabricated MQCA network. Reproduced from [10] with permission of the publisher.

compared to CMOS [4]. A NOT gate and a majority gate have been demonstrated [11, 14]. Figure 2.14 shows a SEM image of a fabricated magnetic QCA network.

2.4 QCA Concept Relative to the Thesis

QCA is a nanotechnology, and it depends on quantum-mechanical effects, in particular, quantum-tunneling. However, the use of QCA does not imply quantum computing. Quantum computing depends on the superposition of the states. The circuits implemented in Section 2.1 are standard logic gates. Those gates are built based on the QCA concept. The output of each gate is either ‘0’ or ‘1’ at any given clock cycle. The circuits built from these gates do not depend on superposition. Hence, this is not a quantum computing method.

It is suggested that QCA has potential for low power dissipation compared to CMOS technology. This fact is based on the theory that QCA interactions rely on the position of electrons within each cell. CMOS technology relies on the current flow through the wires and devices. However, another issue also needs to be considered for the overall power performance, which is the clock power. In CMOS technology, the clock is only needed for sequential elements in synchronous design. In QCA, even wires and gates must be clocked. These clocks are embedded underneath each QCA cell in the implementation. This means all cells will require additional power for data transmission. Without actual hardware testing, it is hard to determine which technology has better overall power performance. However, it is one key idea to keep in mind for future work.

Chapter 3

Architectures

This chapter describes the hierarchical architecture of the design presented in this thesis. The layout of the design is shown in QCADesigner. The description flows in a top-down format. The FPGA is explained first, and second the CLB, next the LUT, and finally the memory component.

3.1 FPGA

The proposed architecture for the FPGA is a cellular array of CLBs tiled together. Figure 3.1 demonstrates an example of the proposed FPGA. The number of CLBs tiled together determines the size of the applications that can be performed on the FPGA. Each side of a CLB block has a single input and a single output. The outputs of each CLB are connected to the adjacent CLB's input for data communication purposes. Besides the input and output lines shown on the figure, there are also some configuration bits used to configure the CLB memory on the right side of the FPGA (not shown on the figure) and the output of those bits are on the left side of the FPGA.

The CLBs are designed so that they can be set up next to each other (as in Figure 3.1) to form a simple FPGA. With the setup of the configuration bits, each CLB can be programmed differently. The memory configuration process is described in Section 3.4.

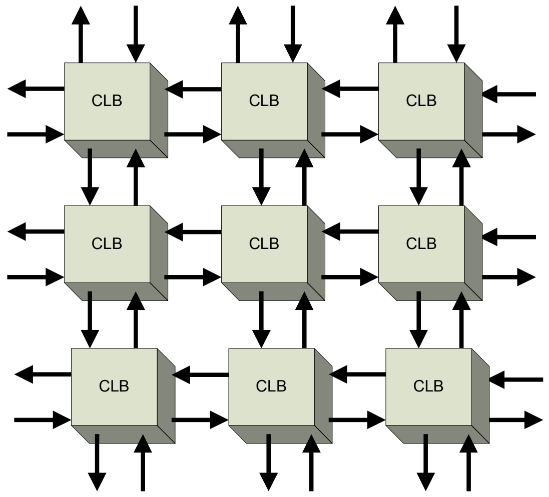


Figure 3.1: Architecture for QCA FPGA.

3.2 CLB

The CLB is composed of four *look-up tables* (LUTs). The LUTs are placed in a 2×2 array to form the CLB. Figure 3.2 shows the block diagram of the CLB. It has four data inputs: n, s, e, w, and four data outputs: N, S, E, W. There are 16 additional configuration inputs to the CLB, which are not shown on the Figure 3.2, but will be explained in Section 3.4. The four data inputs come from the four corners of the CLB and travel to the center. The signals are then evenly distributed to the four LUTs. Each input line must incur the same number of clock cycles of latency to ensure a throughput of one read operation per clock cycle. The CLB was first developed by [16], and then optimized by [32]. It takes seven clock cycles for the data to travel from n, s, e, and w to the LUTs. Then each LUT takes nine clock cycles to produce an output signal. The CLB has a total latency of 16 clock cycles.

As mentioned above, the CLB is made up of four LUTs. They are identical. However, to connect the input and output lines, the LUTs are flipped with respect to each other and arranged as shown in Figure 3.2. It is important to make sure that the outputs and inputs are lined up, so that multiple CLBs can be tiled together without extra wiring. Figure 3.3 shows the layout design of the CLB. It is drawn and simulated in QCADesigner [41]. A

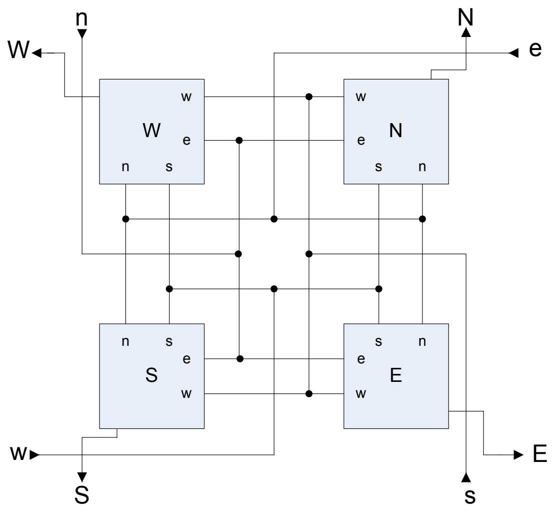


Figure 3.2: Block diagram of CLB.

Verilog model using HDLQ [29] is separately developed. The completed simulations in both programs are shown in Chapter 4.

3.3 LUT

The LUT is made up of three major components: a decoder, a 16-bit memory, and an output circuit. Figure 3.4 (a) shows the block diagram of the completed LUT. The number next to each signal on the figure indicates the clock cycles of latency relative to the initial inputs applied to the decoder. As the numbers on Figure 3.4 (a) indicate, the inputs are coming from the top left corner. All the latencies are with respect to the arrival of the input signals. The vertical arrows represent the column decoder rails. The only exception is the most right vertical arrows that represent the vertical OR chain outputs. The horizontal arrows are shorthand for the row decoder rails, the data input rails, the read/write enable rails and the OR chain rails. Those signal rails will be explained in Section 3.3.1. The clock cycle information on the arrows applies to row and column decoder inputs. Those numbers indicate the clock cycles required by an address input to arrive at a specific unit memory location. No matter what path is chosen to travel for the signal, the delay remains the same, with exactly one clock cycle delay

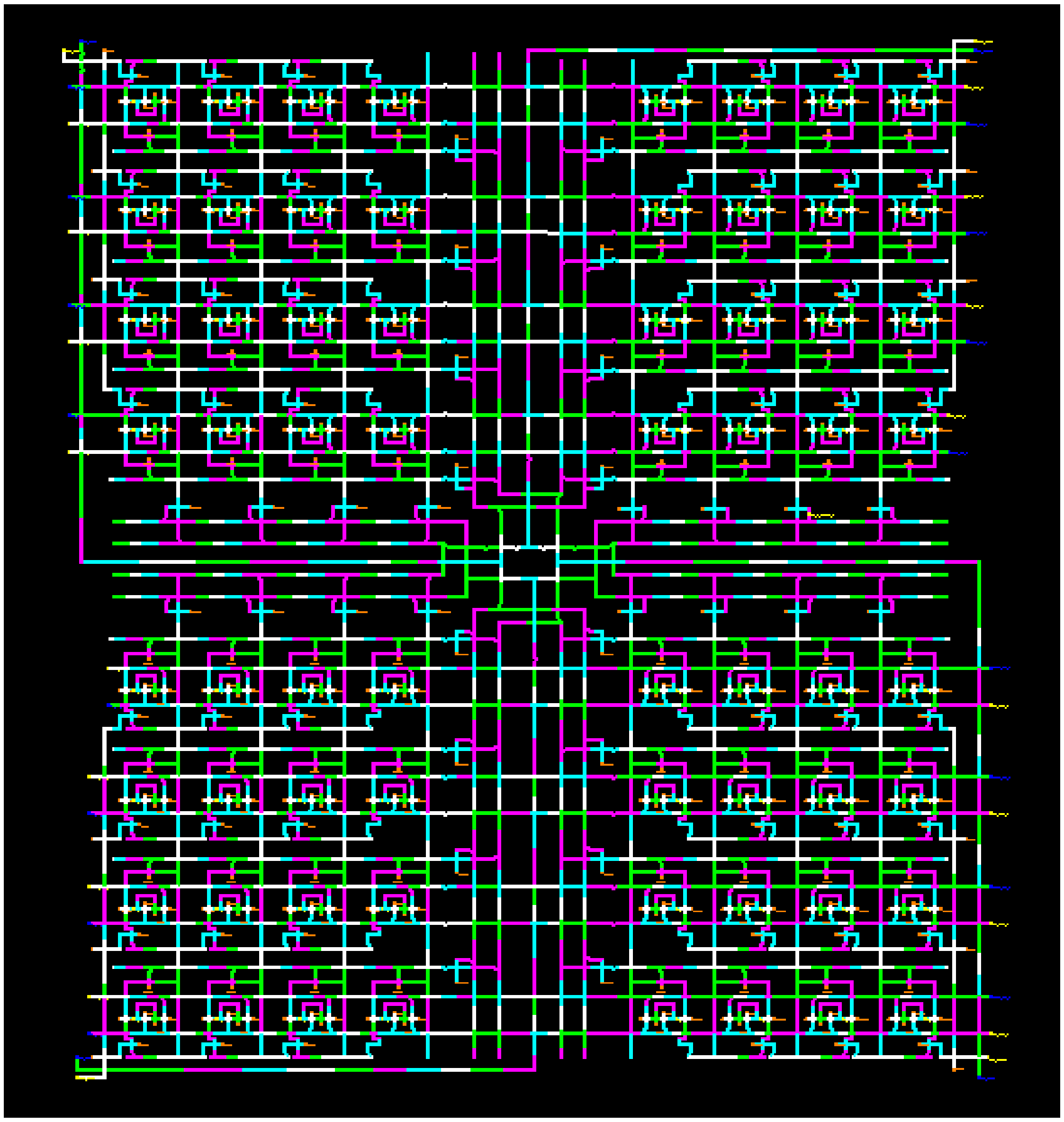


Figure 3.3: CLB layout in QCADesigner [41].

between each cell. This enables a throughput of one read per cycle. The total latency from the decoded address to arrive at a valid output is nine clock cycles.

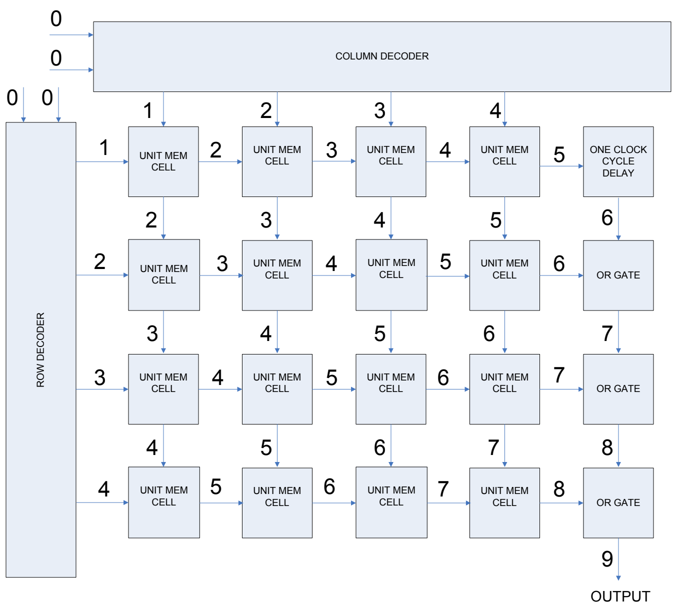
At the bottom right of the LUT, there is the output of the LUT. It is the OR of the enabled outputs of all 16 memory cells. The decoder ensures that only one memory cell is enabled. Thus, the output of the entire LUT is equal to the value stored in the enabled cell.

The decoder used in the LUT is similar to the traditional CMOS memory design, which has separate row and column decoders. This decoder is capable of decoding four inputs to 16 outputs that matches with the 16-memory block in the design. Figure 3.5 shows the schematic and layout of the 2-to-4 decoder. The row and column decoders are identical to each other. The only difference is the rotation. The outputs of the row and column decoders are ANDed together to form the enable signals inside the memory blocks.

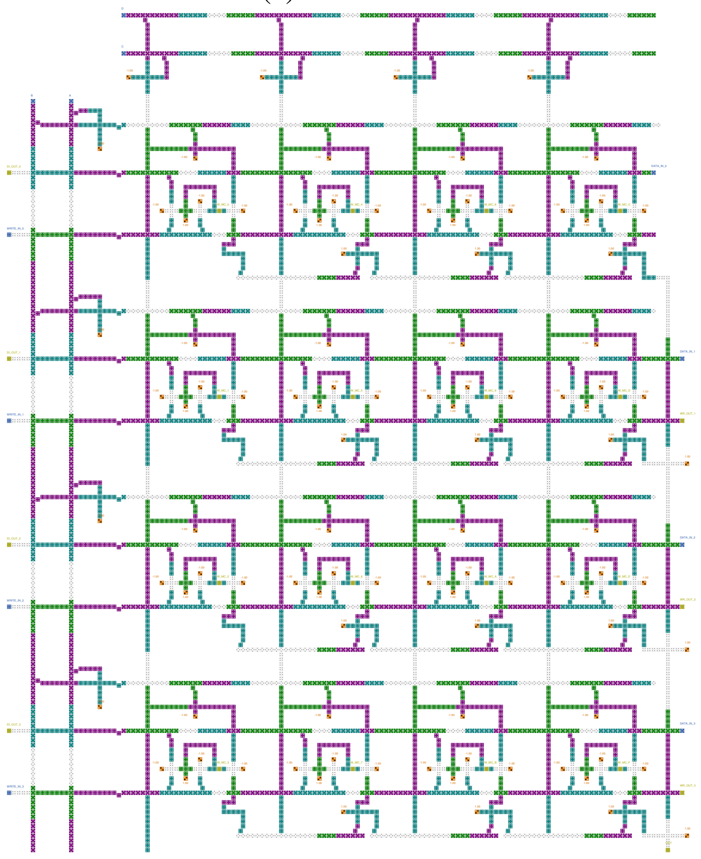
Figure 3.4 (b) shows the schematic of the LUT. The decoders are located on the top and left side. The 16 cells in the middle are arranged in a 4×4 array. The bottom right is the output circuitry that combines all the 16 memory outputs. To configure the LUT, there are four input lines for the memory entering from the right side, and there are also four input lines for a Read/Write (RW) signal entering from the left side. The device has a read latency of nine clock cycles and a throughput of one read operation per clock cycle. It is designed in a way that it is possible to rotate it and to connect it to another LUT. Arranging four LUTs in a 2×2 array gives the CLB shown in Figure 3.2.

3.3.1 Unit Memory

The 16 memory blocks in the LUT are made up of 16 copies of the unit memory cell. Figure 3.6 shows the schematic and layout of the unit memory design. The clock zones on Figure 3.6 (a) are calculated precisely in order for DATA_IN (DI) to match with RW_IN (RW) and configure the unit memory to hold the desired value. The time required for unit memory to store one value and pass data to the next memory cell is exactly one clock

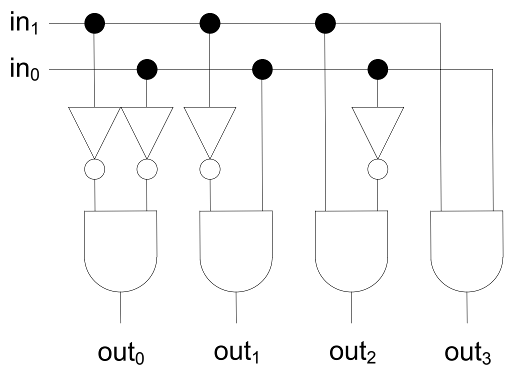


(a) Schematic.

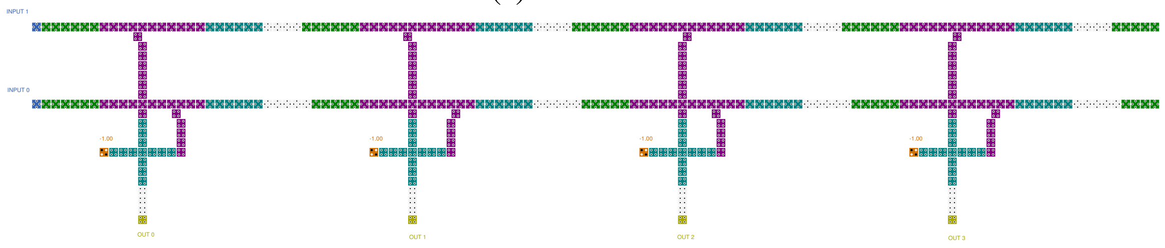


(b) Layout in QCADesigner [41]

Figure 3.4: LUT.



(a) Schematic.



(b) Layout in QCADesigner [41]

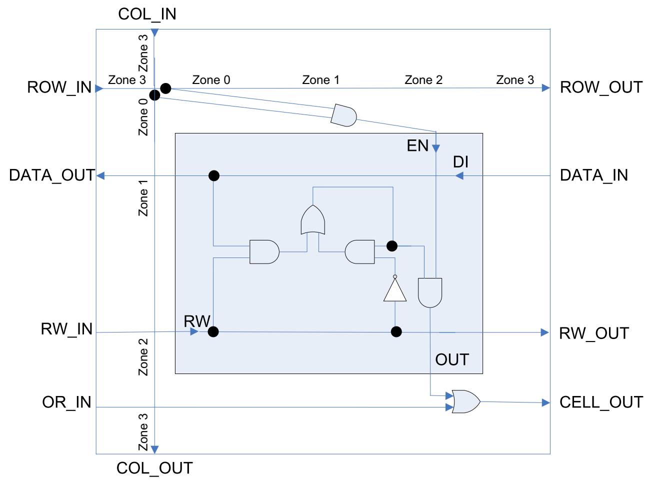
Figure 3.5: Decoder.

cycle. One more important idea in the design is the way that memory is configured. Memory is only written when the RW signal encounters a pulse on the DI signal. Otherwise, the memory is not being configured and remains unchanged. The configuration process is discussed in Section 3.4.

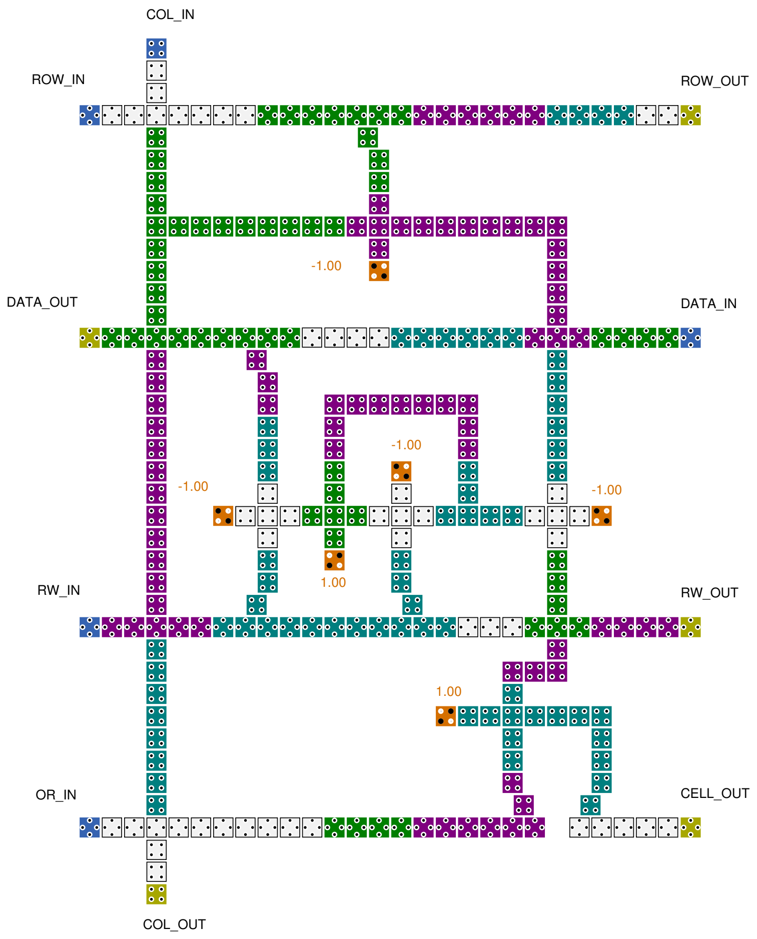
3.3.2 Output Circuitry

The output circuitry merges the outputs from all the memory cells such that the enabled cell determines the output of the LUT. QCA does not have tristate buffers. Therefore, the use of logic for selecting a single output from the memory cells is required. The proposed LUT used a series chain of OR gates to achieve one read per clock cycle. The decoder and the output circuitry are arranged in a way that the input information and the output information flow in the same direction as each other. The proposed design ensures a fully pipelined operation. The design has similarity to that of [42]. However, the throughput frequency is unclear from [42].

The output circuitry is split into five OR chains shown in Figure 3.7.



(a) Schematic.



(b) Layout in QCADesigner.

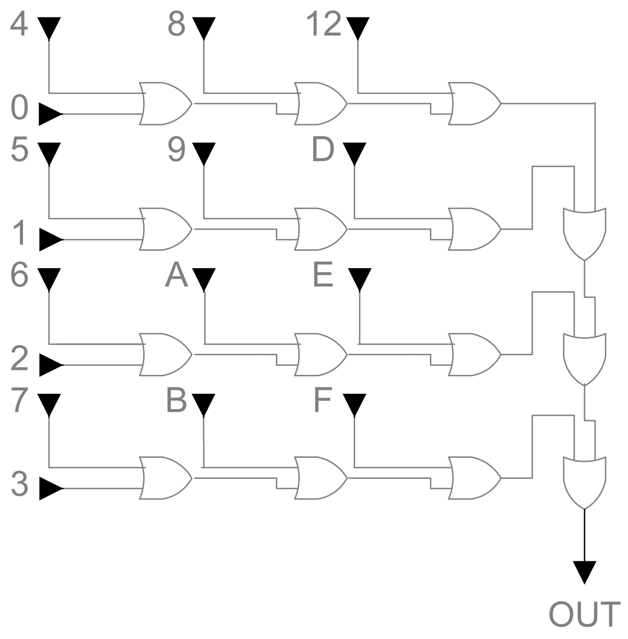
Figure 3.6: Unit memory cell.

Four OR chains are running horizontally, and one is running vertically. The horizontal chains combine the value stored in the memory within a row, and the vertical chain merges the values from four horizontal chains. Each OR chain is made up of three gates, and it is executed in parallel with the row and column decoders. This two-stage OR gate circuitry and the row and column decoders are designed specifically for selecting a single memory cell out of the 16-bits memory of the LUT. In Figure 3.7, besides the three OR gates on the vertical (rightmost chain), all other OR gates are embedded within the unit memory cell of Figure 3.6.

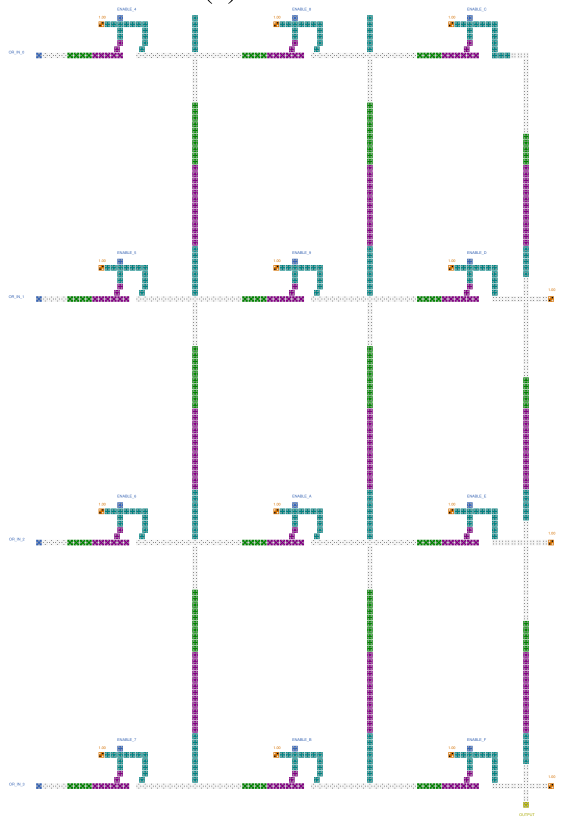
Figure 3.7 (b) shows the layout of the output circuitry. The latency of the OR chain is one clock cycle for a single OR operation. This is important because the row and column decoders also have the same latency to ensure a fully pipelined structure for the LUT. The delay along the OR chain matches the delay along the array of Figure 3.4 (a) for a throughput of one read operation per clock cycle.

3.4 Configuration

The method to configure the memory of the CLB and FPGA is the same. As described earlier, the FPGA is a combination of multiple CLB blocks tiled together. Therefore, the memory configuration bits are also tiled together. Thus, the row of memory cells across the entire FPGA forms the fundamental unit of reconfiguration of the proposed FPGA. Reconfiguring any part of the row requires reconfiguring the whole row of memory, but each row of memory can be reconfigured independently. This is similar to the Xilinx FPGA architecture in which the frame is the fundamental unit of reconfiguration. The difference is that Xilinx has a column memory, and the proposed FPGA reconfiguration is a row. Figure 3.8 shows all the inputs and outputs of the CLB, including those used for configuration. On the right side of the CLB are the memory inputs, and on the left side are the Read/Write signal inputs. The memory values come in from the right, and RW signals come in from the left. The memory is configured when both signals meet. Therefore, it is required to calculate the precise timing and sequence necessary to configure the desired function.



(a) Schematic.



(b) Layout.

Figure 3.7: Output circuitry.

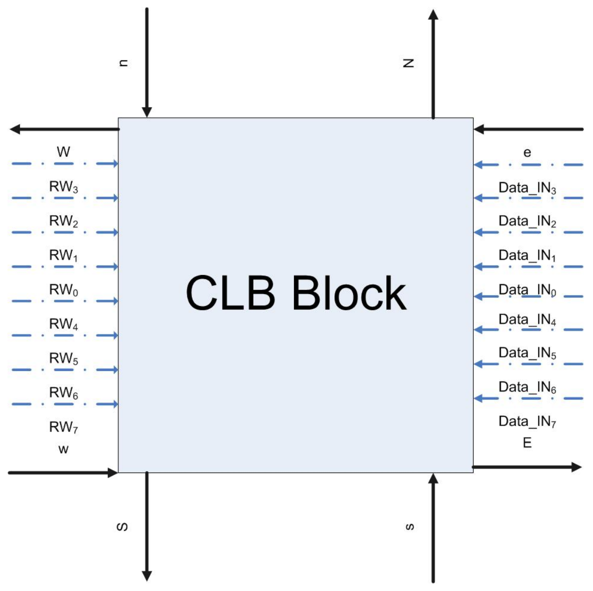


Figure 3.8: A single CLB.

A CLB consists of four LUTs. Each LUT has 16 memory cells, which means a complete CLB has 64 bits of memory. Figure 3.9 shows how the memory is allocated in the CLB. Each DATA_IN signal forms a pair with its matching RW signal. Each such pair controls an entire row of memory. Based on this type of configuration, to set up a desired function, a truth table must be written first. In Figure 3.9, the memory is divided into four sections. The top right section is associated with the N output. The top left section is associated with the W output. The bottom right section is associated with the E output. Finally, the bottom left section is associated with the S output. The input to the CLB block is in a (e, w, s, n) pattern. For example, when an input of 0111 is given to the CLB, the CLB must return the values stored in the location of 7 in all four sections. Equations (3.1) and (3.2) show the full adder logical behavior.

$$s_i = a_i \oplus b_i \oplus c_i \tag{3.1}$$

$$c_{i+1} = a_i b_i + b_i c_i + a_i c_i \tag{3.2}$$

Table 3.1 shows a truth table of a full adder. The first row represents the functional logical name, and the second row represents the corresponding

W				N			
15	11	7	3	3	7	11	15
14	10	6	2	2	6	10	14
13	9	5	1	1	5	9	13
12	8	4	0	0	4	8	12
12	8	4	0	0	4	8	12
13	9	5	1	1	5	9	13
14	10	6	2	2	6	10	14
15	11	7	3	3	7	11	15
S				E			

Figure 3.9: Position of each address within the LUT of the CLB.

port of the CLB. It is simple to match the truth table with memory configuration. This table is set so that it increases its value from 0 to 15 in the binary system. The addresses in Table 3.1 correspond to the locations in Figure 3.9. In this example, N and E are all meaningless (*don't care*) values. Therefore, the memory associated with them, the top right and bottom right sections, should be set to '0.' s_i and c_{i+1} correspond to the bottom left and the top left sections. Those memory units must be configured as shown in Tables 3.2 and 3.3. Based on the setup and the memory configuration input from Figure 3.8, each DATA_IN controls a row of memory. Table 3.4 shows the inputs needed to configure the CLB into a full adder. Between each valued input, there must be a delay cycle to match the RW signals. There are eight of memory units in each row, so eight valid inputs are required to configure the memory unit. The first eight cycles from Table 3.4 configure the left part of the memory in the CLB. The last eight cycles configure the right part of the memory in the CLB. In the table, X indicates a *don't care* bit. It is just because this full adder example does not require the right part of memory to function. When users design their own functions, they will have to figure out the memory input for them. RW signals travel in the opposite direction from that of the $DATA\_IN$ signals. Each RW signal must have the input sequence of 0000000001000000. This sequence starts the RW pulse with the correct timing to meet each bit of the $DATA\_IN$ at the appropriate cell. Once the full adder is configured, the top left memory contains $EE88$ in hex and bottom left contains 9966 in hex, and the other two memory units contain 0000 in hex.

Table 3.1: Full adder truth table.

Address	Inputs				Outputs			
	c_i		b_i	a_i	c_{i+1}	s_i		
	e	w	s	n	W	S	N	E
0	0	0	0	0	0	0	X	X
1	0	0	0	1	0	1	X	X
2	0	0	1	0	0	1	X	X
3	0	0	1	1	1	0	X	X
4	0	1	0	0	0	0	X	X
5	0	1	0	1	0	1	X	X
6	0	1	1	0	0	1	X	X
7	0	1	1	1	1	0	X	X
8	1	0	0	0	0	1	X	X
9	1	0	0	1	1	0	X	X
10	1	0	1	0	1	0	X	X
11	1	0	1	1	1	1	X	X
12	1	1	0	0	0	1	X	X
13	1	1	0	1	1	0	X	X
14	1	1	1	0	1	0	X	X
15	1	1	1	1	1	1	X	X

Table 3.2: s_i memory configuration.

1	1	0	0
0	0	1	1
0	0	1	1
1	1	0	0

Table 3.3: c_{i+1} memory configuration.

1	1	1	1
1	1	0	0
1	1	0	0
0	0	0	0

Table 3.4: Input for configuration.

Input clock cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$DATA\_IN_3$	1	-	1	-	1	-	1	-	X	-	X	-	X	-	X	-
$DATA\_IN_2$	1	-	1	-	0	-	0	-	X	-	X	-	X	-	X	-
$DATA\_IN_1$	1	-	1	-	0	-	0	-	X	-	X	-	X	-	X	-
$DATA\_IN_0$	0	-	0	-	0	-	0	-	X	-	X	-	X	-	X	-
$DATA\_IN_4$	1	-	1	-	0	-	0	-	X	-	X	-	X	-	X	-
$DATA\_IN_5$	0	-	0	-	1	-	1	-	X	-	X	-	X	-	X	-
$DATA\_IN_6$	0	-	0	-	1	-	1	-	X	-	X	-	X	-	X	-
$DATA\_IN_7$	1	-	1	-	0	-	0	-	X	-	X	-	X	-	X	-

Chapter 4

Simulation and Analysis

Two different simulators, QCADesigner [41] and HDLQ [29] with *ModelSim*, are used for the simulation. The layout was first drawn and simulated using QCADesigner. However, at the one-CLB level, the simulation time requires more than 12 hours. To reduce the significant amount of simulation time, HDLQ with ModelSim is used for designs with more than one CLB. All the architectures below the CLB level are rebuilt in HDLQ and simulated for verification. The results from both QCADesigner and HDLQ match each other. However, HDLQ is possible for large-scale simulation.

QCADesigner is an open-source software package that many QCA researchers have been using. However, QCADesigner simulator (V.2.0.3) was last updated at 2005. Although the tool allows simulating of multi-layer QCA, not enough evidence is provided to support the feasibility of such design. Therefore, all the architectures presented in this thesis use a single layer. HDLQ [29] is a Verilog library for behavioral simulation of QCA architectures. Unlike QCADesigner, which models QCA at the physical level with quantum-mechanics, HDLQ models QCA circuits by assuming an ideal logical model of QCA operation. This is important for a fast prototyping of complex QCA circuit. The complex QCA circuits are composed of different logic blocks. For example, *wire*, *fanout*, *inverter*, and *majority voter* blocks are provided in the HDLQ library. The unit memory, LUT, CLB, and FPGA are formed as structural interconnections of these blocks. In each level up to the CLB, the simulation results match those of QCADesigner.

The CLB of Figure 3.3 is simulated in QCADesigner version 2.0.3 using the *bistable simulation engine* with a *radius of effect* of 50 nm. The total

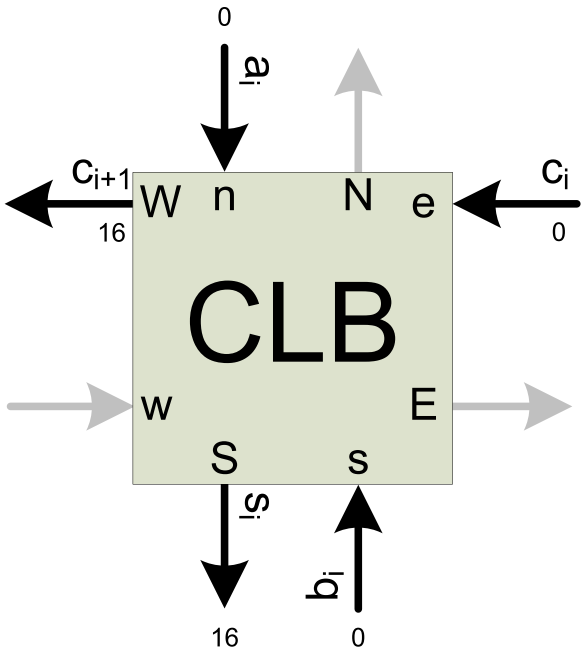


Figure 4.1: Full adder symbol.

number of samples is 300,000. All remaining parameters use the default values provided by QCADesigner.

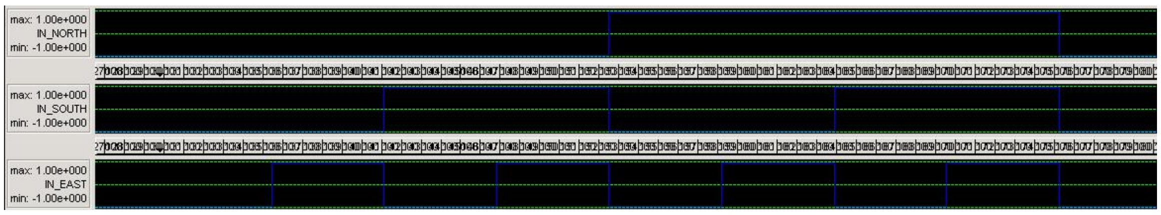
4.1 Full Adder

In the first step of design verification, the 1-bit *full adder* (FA) is simulated using QCADesigner. The FA implements Equations (3.1) and (3.2). Figure 4.1 shows the schematic view of the FA. Figure 3.3 shows the layout. In this circuit, inputs and outputs are given, as shown in Figure 4.1. Table 4.1 shows a general input and output relationship. Our CLB has a latency of 16 clock cycles, which means for any given input vector to the system, the corresponding output is expected after 16 clock cycles. As for this full adder testing, only three inputs, a_i, b_i, and c_i, are given. To run an exhaustive test, eight test vectors are given to the system.

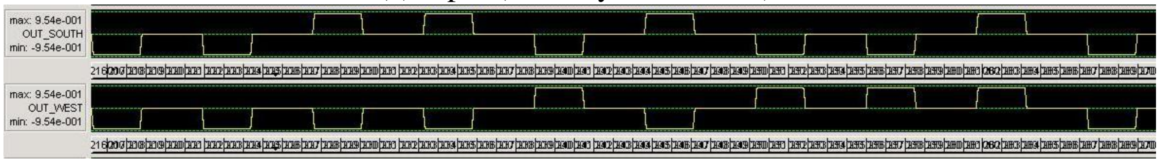
Figure 4.2 shows the output of the full adder only under the QCADesigner simulator. The latency of the system is 16 clock cycles. Figure 4.2 (a) shows the cycles from 19 to 27, which represent the input to the FA, and Figure 4.2 (b) shows the cycles from 25 to 43, which represent the output of

Table 4.1: Full adder result.

	Inputs					Outputs	
Clock Cycle	c_i	b_i	a_i		Clock Cycle	c_{i+1}	s_i
20	0	0	0	$\rightarrow$	36	0	0
21	0	0	1	$\rightarrow$	37	0	1
22	0	1	0	$\rightarrow$	38	0	1
23	0	1	1	$\rightarrow$	39	1	0
24	1	0	0	$\rightarrow$	40	0	1
25	1	0	1	$\rightarrow$	41	1	0
26	1	1	0	$\rightarrow$	42	1	0
27	1	1	1	$\rightarrow$	43	1	1



(a) Input (From cycle 19 to 27).



(b) Output (cycle 35 to 43).

Figure 4.2: FA simulation in QCADesigner.

the FA. IN_NORTH, IN_SOUTH, and IN_EAST are the three inputs: a_i, b_i, and c_i. OUT_SOUTH is s_i, and OUT_WEST is c_{i+1}.

Figure 4.3 demonstrates the input and output waveform in HDLQ. There is a signal named zone1 in the figure. It represents the clock provided to the cells in clock zone 1 of the QCA system. Only one clock is shown here because the output signal changes its status based on this clock. On Figure 4.3, each cycle is 100 ns with the exception of the first cycle being 75 ns to match the clock cycles in QCADesigner. These times are arbitrary. The actual clock period needed for correct operation would depend on the physical QCA implementation used. $data0\_in0$ and $data0\_in1$ are the two input line used to configure the memory of the LUT. in_a, in_b, and in_c are the three inputs of the FA. Comparing Figure 4.2 and Figure 4.3, the same results are obtained. Therefore, to reduce the simulation time for the

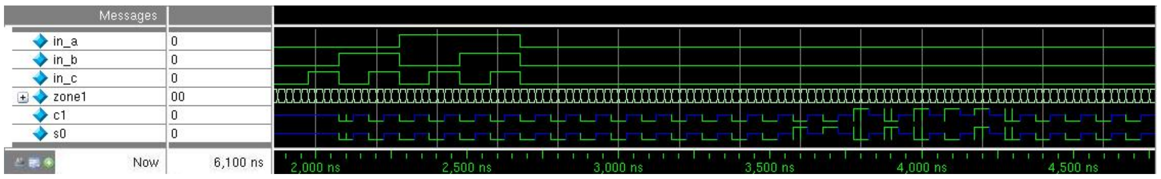


Figure 4.3: Simulation result of full adder in HDLQ.

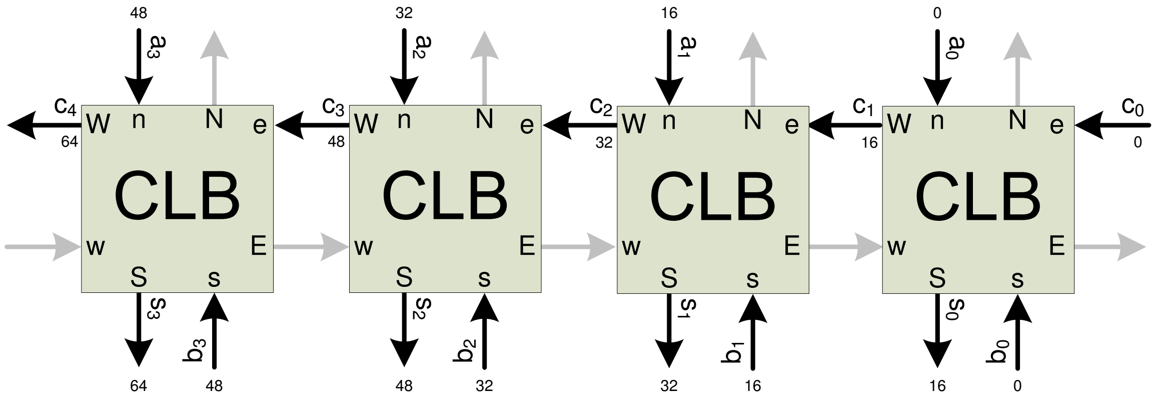


Figure 4.4: RCA block diagram.

larger-scale design, HDLQ is used to replace QCADesigner.

4.2 Ripple Carry Adder

A 4-bit *ripple-carry adder* (RCA) is constructed by tiling four 1-bit full adders together. Figure 4.4 shows the block diagram of the design. The HDLQ Verilog model is simulated using ModelSim. To achieve a 4-bit plus 4-bit RCA, the FPGA is made up of four CLBs tiled together horizontally. For two 4-bit numbers, there are nine inputs to the system and five outputs from the system. Figure 4.4 shows how CLBs are tiled together to form the 4-bit RCA. CLB_0 is the least significant block. It takes three inputs, c_0, a_0, and b_0, to produce two outputs, c_1, and s_0. The behavior of the RCA is calculated based on Equations (3.1) and (3.2), from which the expected output from each bit can be calculated.

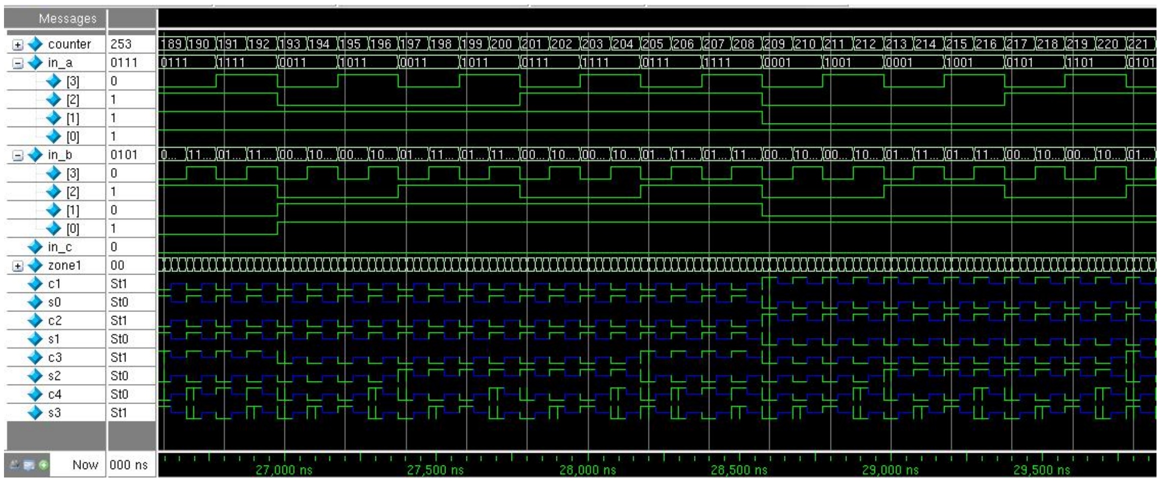
As shown in Figure 4.4, CLB_0 produces s_0 and c_1 for CLB_1. CLB_1 follows the same behavior for CLB_2 and then CLB_3. The final five output bits are c_4, s_3, s_2, s_1, and s_0. The result is simulated under HDLQ, using an exhaustive test set of all 512 input combinations. Figure 4.4 depicts four CLBs configured to implement the RCA. Within each shaded CLB, the inputs are $n, s, e, w,$ and the outputs are $N, S, E, W.$ The function is then determined by the memory configuration. The gray arrows in Figure 4.4 represent inputs to and outputs from the CLBs that are not used by the RCA design.

The number next to each signal on the figure indicates the clock cycles of latency relative to the initial inputs applied to the least significant bit. For a given test vector, if a_0, b_0 and c_0 are all applied at clock cycle t, then any signal marked with the number n in the diagram occurs at clock cycle $t + n$. For an input signal, this means that the user must apply that input at the indicated cycle. For an output, this means that the output must be read at that cycle.

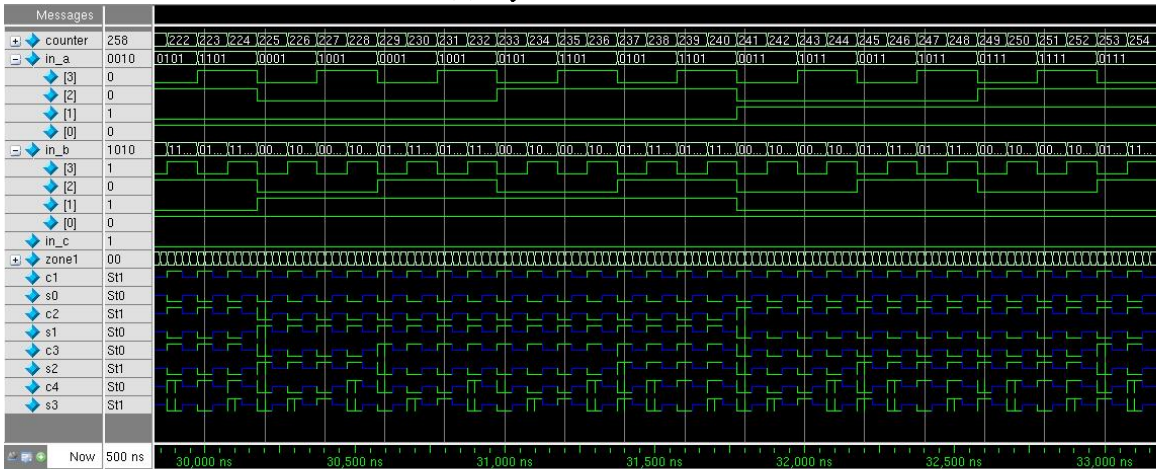
The latency on the critical path from the least-significant inputs to the most significant outputs is 64 clock cycles long. However, the adder is fully pipelined. Note that all the inputs to a given CLB arrive with the same latency. The least significant bits for the next pair of addends can be applied in cycle $t + 1$. Thus, a throughput of one addition per clock cycle can be maintained. Figure 4.5 shows the partial waveform result from HDLQ.

Here is an example of the addition process. If $0111 + 1110$ are given to the RCA, Table 4.2 shows the input and output sequences of the RCA. The inputs need to be staggered. That is why 0111 and 1110 are split and given to the RCA in four different clock cycles, and there are also four outputs associated with each given input after 16 clock cycles. The Xs in the table are the do not care bits. Table 4.2 only shows one addition process. However, it does not mean that the clock cycles in between are idle. Those clock cycles in between are also calculating other sets of addition process.

Calculating the exact number of cycles for a signal to arrive at certain time seems like a *wave pipeline*. Wave pipelining is to send in a steam of signals from different directions and have them converge at the expected location and time. The main difference is the clock. Wave pipelines have no clock, but QCA circuits do have clocks. However, since QCA has clock signals everywhere, it feels like the clock becomes “time.” Signals are given at certain “times” and expected it arrive the desired locations at certain “times.”



(a) Cycle 189 to 221.



(b) Cycle 222 to 254.

Figure 4.5: Partial 4-bit RCA waveform result form HDLQ.

Table 4.2: RCA result.

Clock Cycle	Input Bit	Carry Bit	Output Bit
190	$a_0 = 1, b_0 = 0, c_0 = 0$		
206	$a_1 = 1, b_1 = 1, c_0 = 0$	$c_1 = 0$	$s_0 = 1$
222	$a_2 = 1, b_2 = 1, c_0 = 0$	$c_2 = 1$	$s_1 = 0$
238	$a_3 = 0, b_3 = 1, c_0 = 0$	$c_3 = 1$	$s_2 = 1$
254		$c_4 = 1$	$s_3 = 0$

In this sense, a QCA circuit operates as a wave pipeline. In the test bench design, all the inputs are pre-designed and sent into the system like a wave of signals. The complete simulation waveform of the RCA can be found in Appendix B.

To run an exhaustive test on the 4-bit RCA requires a long list of vectors. To avoid mistakes in the test bench, a spreadsheet is used to compute the inputs and expected outputs. After the results are constructed, each input is staggered according to the CLB latencies. The table is saved into a text file. This file contains the input to the system and also the expected output from the system in each cycle. The test bench is able to read from this file, so it knows in which cycle what inputs are given to the RCA and what outputs are expected. The test bench compares the actual outputs from the RCA with the expected outputs from the text file. If there is mismatch from the comparing process, the mismatch cycle is reported on the screen.

4.3 Bit-Serial Multiplier

Another application for the proposed FPGA is a *bit-serial multiplier* (BSM). The BSM is capable of taking two 4-bit numbers and multiplying them together. Figure 4.6 shows the block diagram of the BSM. $A = \langle a_3, a_2, a_1, a_0 \rangle$ is the multiplicand and is input to the system in parallel. X_{IN} represents one bit of the multiplier from $X = \langle x_3, x_2, x_1, x_0 \rangle$ and is input serially every other cycle. Each input is delayed for one clock cycle when it passes through the AND gate to the FA block. However, for the proposed FPGA, the latency between each CLB is 16 cycles. This increases the serial input to the system from every other cycle to every 32 cycles, and the inputs are also delayed from one cycle to 16 cycles. To produce one 8-bit product from two 4-bit factors, 256 cycles are needed. The design is tested under HDLQ with ModelSim. The test bench was written to compare the simulation output to the expected output from the file. To run an exhaustive test, 65,536 test vectors are needed. The simulation results match to the expected result. Therefore, the proposed FPGA is proven to handle BSM. Partial waveform results are shown in Appendix C.

BSM is chosen to be one of the test applications because of the delays in the design. The proposed CLB is a heavy pipelined structure. Initially, the array multiplier seems to be a good application for the proposed CLB because of the pipelined block structure. However, it turns out that the signal cannot always arrive at desired time because different paths between the same two points can have different latencies. The internal delay makes the array multiplier hard to implement using the proposed FPGA. Instead BSM has a lot of internal signal delays, which fits to the delay of the proposed CLB. 12 CLBs in an array format of 3×4 are used to implement the BSM. Each CLB in the middle row is configured to be a 1-bit full adder. Each CLB in the top row is configured as an AND gate for ANDing a_i and X_{IN}. In Figure 4.6, there are many delay boxes shown on the schematic. Those delays can be taken care by the proposed CLB internal delay. Each CLB in the bottom row is configured to be a feedback loops.

4.4 Glitchless Reconfiguration

Partial reconfiguration [33, 45] is a process to configure part of an FPGA while the whole FPGA is still operating. Partial reconfiguration is interesting in many areas. Patterson [31] applied the partial reconfiguration technique to the *data encryption standard* (DES) algorithm. DES is a private key algorithm that is used to encrypt or decrypt the information transferred. The FPGA implementation uses a circuit that is specialized for a particular key to improve performance. Patterson uses partial reconfiguration to replace the key-specific portion of the circuit leaving the rest unchanged.

Even in FPGAs that support partial reconfiguration, there is typically some minimum unit of reconfiguration, such that reconfiguring any part of the unit requires reconfiguring the entire unit. In the case of the proposed

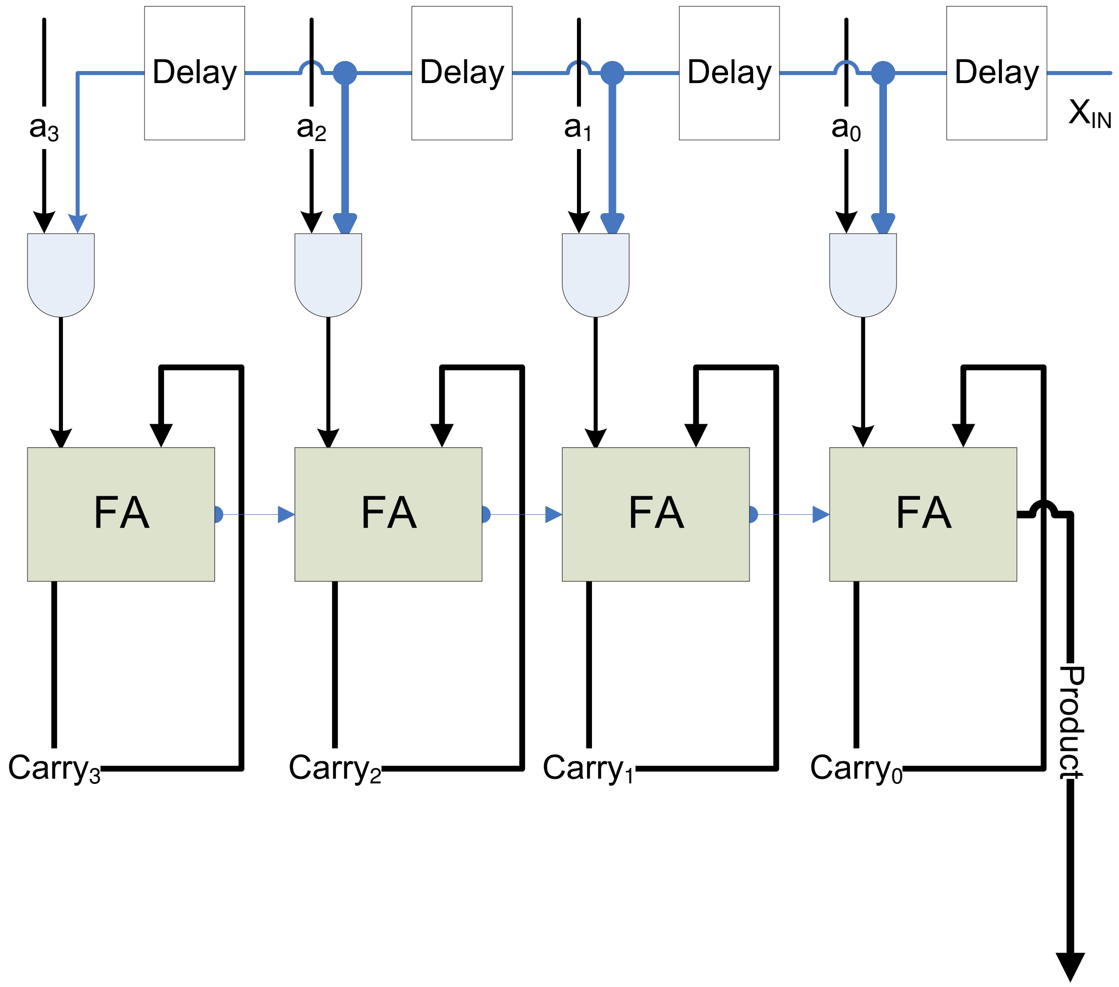


Figure 4.6: BSM block diagram.

FPGA, this unit consists of one row of configuration bits. Therefore, partially reconfiguring a region that does not span the full width of the device requires reconfiguring areas on either side of the target region. In these areas, the new configuration bits are equal to the old configuration bits. If the user circuits in the side region are to keep operating uninterrupted during partial reconfiguration, it is important that this operation does not cause a glitch in any configuration values in the side regions. An FPGA is said to support *glitchless reconfiguration* if, when a bit holds the same value before and after reconfiguration, the act of reconfiguration is guaranteed not to cause a glitch [33].

Figure 4.7 shows the result of glitchless reconfiguration from QCADesigner. It uses the same example of a 1-bit full adder. Table 4.3 shows the inputs and memory results with clock cycles. $Data\_IN_{3...0}$ and $Data\_IN_{7...4}$ are used to configure the memory. It needs 17 cycles to configure the 8×8 memory of the proposed CLB. Cycle 1 to 17 from Table 4.3 shows the vectors to configure the memory to emulate the CLB. The address inputs are given when the configuration process is completed. However, to observe the reconfiguration process, $Data\_IN_{3...0}$ and $Data\_IN_{7...4}$ are given the same configuration bits once again, while the input vectors are also given to the system. Figure 4.7 shows the simulation waveform from QCADesigner. Table 4.4 shows the output result of this simulation. The adder performs its function uninterrupted while re-programming of memory is occurring. The results in Table 4.3 prove that the memory contents do not change even when the re-programming process is happening. It is clear that the results validate the original assumption. Therefore, the proposed CLB is capable of handling glitchless reconfiguration. The complete waveforms are shown in Appendix D.

Table 4.3: Glitchless reconfiguration.

Cycle	$Data\_IN_{3...0}$	$Data\_IN_{7...4}$	RW_IN	Address input	N_{MC}	E_{MC}
1	E	9	0	0	zzzz	zzzz
2	-	-	0	0	zzzz	zzzz
3	E	9	0	0	zzzz	zzzz
4	-	-	0	0	zzzz	zzzz
5	8	6	0	0	zzzz	zzzz
6	-	-	0	0	zzzz	zzzz
7	8	6	0	0	zzzz	zzzz
8	-	-	0	0	zzzz	zzzz
9	-	-	0	0	zzzz	zzzz
10	-	-	1	0	zzzz	zzzz
11	0	0	0	0	zzzz	zzzz
12	-	-	0	0	Ezzz	9zzz
13	0	0	0	0	EEzz	99zz
14	-	-	0	0	EE8z	996z
15	0	0	0	0	EE88	9966
16	-	-	0	0	EE88	9966
17	0	0	0	0	EE88	9966
18	-	-	0	0	EE88	9966
19	-	-	0	0	EE88	9966
20	E	9	0	0	EE88	9966
21	-	-	0	1	EE88	9966
22	E	9	0	2	EE88	9966
23	-	-	0	3	EE88	9966
24	8	6	0	4	EE88	9966
25	-	-	0	5	EE88	9966
26	8	6	0	6	EE88	9966
27	-	-	0	7	EE88	9966
28	-	-	0	0	EE88	9966
29	-	-	1	1	EE88	9966
30	-	-	0	2	EE88	9966
31	0	0	0	3	EE88	9966
32	-	-	0	4	EE88	9966
33	0	0	0	5	EE88	9966
34	-	-	0	6	EE88	9966
35	-	-	0	7	EE88	9966

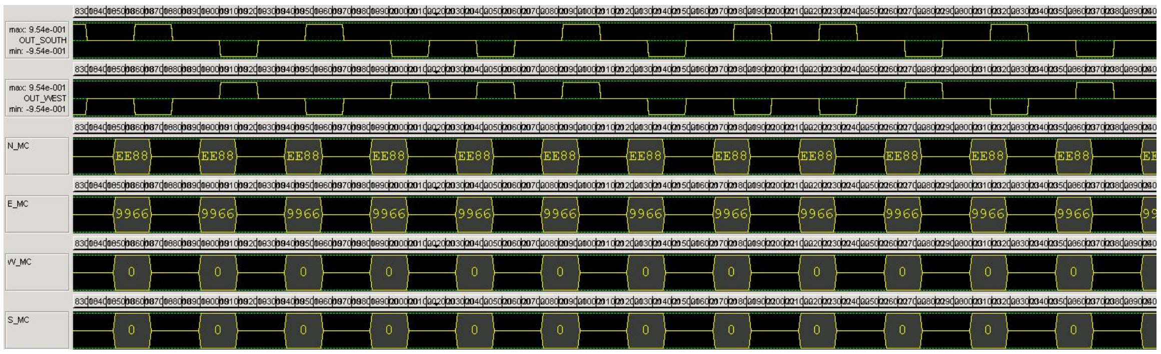


Figure 4.7: Simulation result of glitchless reconfiguration.

Table 4.4: Full adder result during partial reconfiguration.

	Inputs					Outputs	
Clock Cycle	c_i	b_i	a_i		Clock Cycle	c_{i+1}	s_i
20	0	0	0	$\rightarrow$	36	0	0
21	0	0	1	$\rightarrow$	37	0	1
22	0	1	0	$\rightarrow$	38	0	1
23	0	1	1	$\rightarrow$	39	1	0
24	1	0	0	$\rightarrow$	40	0	1
25	1	0	1	$\rightarrow$	41	1	0
26	1	1	0	$\rightarrow$	42	1	0
27	1	1	1	$\rightarrow$	43	1	1
28	0	0	0	$\rightarrow$	44	0	0
29	0	0	1	$\rightarrow$	45	0	1
30	0	1	0	$\rightarrow$	46	0	1
31	0	1	1	$\rightarrow$	47	1	0
32	1	0	0	$\rightarrow$	48	0	1
33	1	0	1	$\rightarrow$	49	1	0
34	1	1	0	$\rightarrow$	50	1	0
35	1	1	1	$\rightarrow$	51	1	1

Chapter 5

Conclusions and Future Work

This thesis presents the design, layout and successful simulation of a LUT-based CLB in QCADesigner and HDLQ. A collection of four CLBs and another collection of 12 CLBs acting as a very simple FPGA are also simulated in HDLQ. Previous work on FPGAs in QCA has focused on programmable interconnect. In contrast, this thesis presents what we believe to be the first QCA design to use multiple CLBs consisting of *look-up tables* (LUTs).

QCA depends on the use of clocks throughout. Even logic gates and wires are clocked. Thus, even simple operations incur multiple clock cycles of latency. This forms a fundamental design challenge in QCA. To achieve high throughput, QCA designs must be heavily pipelined. The LUTs presented here adopt a two-dimensional memory structure with separate row and column decoders inspired by CMOS memories. These are carefully designed such that inputs and outputs all flow in the same direction and delays are matched. This allows the LUTs to maintain a throughput of one transaction per clock cycle. The proposed CLB has just over half the area and incurs just over half the latency of the CLB of [16].

We have demonstrated a sample application of a four-bit ripple-carry adder (four-CLB) and a four-bit bit serial multiplier (12-CLB) running on the simple FPGA. Although the latency of the critical path is 64 clock cycles, the adder maintains a throughput of one sum per clock cycle. This is achieved by applying the same strategy as that used to design the LUT.

The CLB unit is tested for glitchless reconfiguration in both QCADesigner and HDLQ. The simulation results prove that the proposed FPGA has this property. Our design supports this feature due to the way that the memory is configured [16, 36]. Although the reconfiguration bits are input serially, they are not shifted from cell to cell. Instead, each bit only changes the value of its destination cell.

Once more realistic applications are implemented, we will be in a better position to compare the effectiveness of configurable logic to that of configurable routing in the context of QCA. The FPGA presented here relies entirely on configurable logic. Most commercial FPGAs use both configurable logic and configurable routing. Possible future work could investigate combining our CLB with configurable routing of existing QCA designs.

Currently, all the applications presented here rely on hand-mapping for all the memory allocation, which takes time and has a high possibility of error. To reduce the error and time, an *electronic-design automation* (EDA) tool must be developed to efficiently map applications to our design.

An issue in the proposed FPGA design is that the function cannot always have matched latency from every signal. QCA is a pipelined technology. Hence, everything must be pipelined. The CLB presented in this thesis maintains a throughput of one transaction per clock cycle. However, in a user design, different paths might cross different numbers of CLBs before converging. Such mismatch delays interfere with pipelining. This issue has been studied in other fields [35, 44]. In those papers, the proposed FPGA is considered as *fixed-frequency FPGA*. They proposed an idea to solve this issue by adding a programmable delay at the input or output of each CLB to balance the path delays. Further research can apply the techniques of [35, 44] to the CLB presented in this thesis.

The CLB itself can be further optimized. Grouping the cells into larger, more regular clock zones will facilitate robust fabrication [38]. The current design has many small, irregular clock zones. These clock zones result in more power consumption and also require more latency for data transmission. Furthermore, when it comes to the fabrication point of view, the small, irregular clock zones create a certain degree of difficulty. We will also investigate a multi-layer version without coplanar wire crossings. Fabricating a prototype is an important next step.

Bibliography

[1] Alan Allan, Don Edenfeld, William H. Joyner, Jr., Andrew B. Kahng, Mike Rodgers, and Yervant Zorian. 2001 technology roadmap for semiconductors. *IEEE Computer*, 35(1):42–53, 2002.

[2] M. A. Amiri, M. Mahdavi, and S. Mirzakuchaki. QCA implementation of a mux-based FPGA CLB. In *Proceedings of the International Conference on Nanoscience and Nanotechnology (ICONN 2008)*, pages 141–144, February 2008.

[3] Adrian Bachtold, Peter Hadley, Takeshi Nakanishi, and Cees Dekker. Logic circuits with carbon nanotube transistors. *Science*, 294(5545):1317–1320, 2001.

[4] Gary H. Bernstein. Quantum-dot cellular automata: computing by field polarization. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pages 268–273. ACM, 2003.

[5] Gary H. Bernstein, Alexandra Imre, V. Metlushko, Alexei O. Orlov, L. Zhou, L. Ji, György Csaba, and Wolfgang Porod. Magnetic QCA systems. *Microelectronics Journal*, 36(7):619–624, 2005.

[6] Stephen Brown and Jonathan Rose. FPGA and CPLD architectures: A tutorial. *IEEE Des. Test. Comput.*, 13(2):42–57, 1996.

[7] Stephen Brown and Zvonko Vranesic. *Fundamentals of Digital Logic with VHDL Design*. McGraw-Hill Higher Education, third edition, 2009.

[8] B. H. Calhoun, Yu Cao, Xin Li, Ken Mai, L. T. Pileggi, R. A. Rutenbar, and K. L. Shepard. Digital circuit design challenges and opportunities in the era of nanoscale CMOS. *Proc. IEEE*, 96(2):343–365, February 2008.

[9] Tze-Chiang Chen. Where CMOS is going: trendy hype vs. real technology. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC 2006)*, pages 1–18, February 2006.

[10] R. P. Cowburn and M. E. Welland. Room temperature magnetic quantum cellular automata. *Science*, 287(5457):1466–1468, 2000.

[11] R.P. Cowburn. Digital nanomagnetic logic. In *Device Research Conference*, pages 111–114, June 2003.

[12] Yi Cui and Charles M. Lieber. Functional nanoscale electronic devices assembled using silicon nanowire building blocks. *Science*, 291(5505):851–853, 2001.

[13] S. Anisul Haque, Masahiko Yamamoto, Ryoichi Nakatani, and Yasushi Endo. Magnetic logic gate for binary computing. *Science and Technology of Advanced Materials*, 5(1-2):79 – 82, 2004. 21st Century COE Program, Osaka University.

[14] A. Imre, G. Csaba, L. Ji, A. Orlov, G. H. Bernstein, and W. Porod. Majority logic gate for magnetic quantum-dot cellular automata. *Science*, 311(5758):205–208, 2006.

[15] A. Jazbec, N. Zimic, I. L. Bajec, P. Pečar, and M. Mraz. Quantum-dot field programmable gate array: enhanced routing. In *Proceedings of the 2006 Conference on Optoelectronic and Microelectronic Materials and Devices*, pages 121–124, December 2006.

[16] Timothy D. Lantz and Eric R. Peskin. A QCA implementation of a configurable logic block for an FPGA. In *Proceedings of the Third International Conference on Reconfigurable Computing and FPGAs (ReConFig 2006)*, pages 132–141, September 2006.

[17] C. S. Lent and P. D. Tougaw. A device architecture for computing with quantum dots. *Proc. IEEE*, 85(4):541–557, April 1997.

[18] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein. Quantum cellular automata. *Nanotechnology*, 4(1):49–57, January 1993.

[19] Craig S. Lent, Beth Isaksen, and Marya Lieberman. Molecular quantum-dot cellular automata. *Journal of the American Chemical Society*, 125(4):1056–1063, January 2003.

[20] Craig S. Lent, P. Douglas Tougaw, and Wolfgang Porod. Bistable saturation in coupled quantum dots for quantum cellular automata. *Applied Physics Letters*, 62(7):714–716, 1993.

[21] Mo Liu and C. S. Lent. High-speed metallic quantum-dot cellular automata. In *Proceedings of the Third IEEE Conference on Nanotechnology (IEEE-NANO 2003)*, volume 2, pages 465–468, August 2003.

[22] Fabrizio Lombardi, Jing Huang, Xiaojun Ma, Mariam Momenzadeh, Marco Ottavi, Luca Schiano, and Vamsi Vankamamidi. *Design and Test of Digital Circuits by Quantum-Dot Cellular Automata*. Artech House, 2008.

[23] Yuhui Lu and C. S. Lent. Theoretical study of molecular quantum dot cellular automata. In *Proceedings of the 10th International Workshop on Computational Electronics (IWCE-10)*, pages 118–119, October 2004.

[24] Yuhui Lu, Mo Liu, and Craig Lent. Molecular quantum-dot cellular automata: From molecular structure to circuit dynamics. *Journal of Applied Physics*, 102(3):034311, 2007.

[25] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114 ff., April 1965.

[26] Michael Thaddeus Niemier and Peter M. Kogge. The '4-diamond circuit' - a minimally complex nano-scale computational building block in QCA. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2004)*, pages 3–10. IEEE Computer Society, February 2004.

[27] Michael Thaddeus Niemier, Arun Francis Rodrigues, and Peter M. Kogge. A potentially implementable FPGA for quantum dot cellular automata. In *Proceedings of the First Workshop on Non-Silicon Computation (NSC-1)*, pages 38–45, 2002.

[28] Alexei O. Orlov, Islamshah Amlani, Geza Toth, Craig S. Lent, Gary H. Bernstein, and Gregory L. Snider. Correlated electron transport in coupled metal double dots. *Applied Physics Letters*, 73(19):2787–2789, 1998.

[29] Marco Ottavi, Luca Schiano, Fabrizio Lombardi, and Douglas Tougaw. HDLQ: A HDL environment for QCA design. *J. Emerg. Technol. Comput. Syst.*, 2(4):243–261, 2006.

[30] M. C. B. Parish and M. Forshaw. Physical constraints on magnetic quantum cellular automata. *Applied Physics Letters*, 83(10):2046–2048, 2003.

[31] Cameron Patterson. High performance DES encryption in Virtex(tm) FPGAs using Jbits(tm). In *FCCM '00: Proceedings of the 2000*

IEEE Symposium on Field-Programmable Custom Computing Machines, page 113, Washington, DC, USA, 2000. IEEE Computer Society.

[32] Ruchi B. Rungta. Programmable logic device in QCA. Master's thesis, Rochester Institute of Technology, May 2009. Graduate paper.

[33] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght. Modular dynamic reconfiguration in Virtex FPGAs. *IEE Proceedings - Computers and Digital Techniques*, 153(3):157–164, May 2006.

[34] Baris Taskin, Andy Chiu, Jonathan Salkind, and Daniel Venutolo. A shift-register-based QCA memory architecture. *J. Emerg. Technol. Comput. Syst.*, 5(1):1–18, 2009.

[35] William Tsu, Kip Macy, Atul Joshi, Randy Huang, Norman Walker, Tony Tung, Omid Rowhani, Varghese George, John Wawrzynek, and André DeHon. HSRA: high-speed, hierarchical synchronous reconfigurable array. In *FPGA '99: Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 125–134, New York, NY, USA, 1999. ACM.

[36] Chia-Ching Tung, Ruchi B. Rungta, and Eric R. Peskin. Simulation of a QCA-based CLB and a multi-CLB application. In *Proceedings of the 2009 International Conference on Field-Programmable Technology (FPT'09)*, pages 62–69. IEEE, December 2009.

[37] V. Vankamamidi, M. Ottavi, and F. Lombardi. Tile-based design of a serial memory in QCA. In *GLSVLSI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI*, pages 201–206, New York, NY, USA, 2005. ACM.

[38] V. Vankamamidi, M. Ottavi, and F. Lombardi. Tile-based design of a serial memory in QCA. In *GLSVSLI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI*, pages 201–206, New York, NY, USA, 2005. ACM.

[39] Vamsi Vankamamidi, Marco Ottavi, and Fabrizio Lombardi. Two-dimensional schemes for clocking/timing of QCA circuits. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 27(1):34–44, 2008.

[40] K. Walus. QCADesigner — Microsystems and Nanotechnology Group (MiNa). http://www.mina.ubc.ca/qcadesigner, June 2009.

[41] K. Walus, T. J. Dysart, G. A. Jullien, and R. A. Budiman. QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.*, 3(1):26–31, March 2004.

[42] K. Walus, A. Vetteth, G. A. Jullien, and V. S. Dimitrov. RAM design using quantum-dot cellular automata. In *Technical Proceedings of the 2003 Nanotechnology Conference and Trade Show*, volume 2, pages 160–163, 2003.

[43] Konrad Walus, Faizal Karim, and André Ivanov. Architecture for an external input into a molecular QCA circuit. *Journal of Computational Electronics*, 8(1):35–42, March 2009.

[44] Nicholas Weaver, John Hauser, and John Wawrzynek. The SFRA: a corner-turn FPGA architecture. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 3–12, New York, NY, USA, 2004. ACM.

[45] Michael J. Wirthlin and Brad L. Hutchings. Improving functional density using run-time circuit reconfiguration. *IEEE Trans. VLSI Syst.*, 6(2):247–256, 1998.

[46] Wayne Wolf. *FPGA-Based System Design*. Modern Semiconductor Design. Prentice Hall, 2004.

Appendix A

Full Adder Waveform Simulation

This appendix shows the completed simulation waveforms of the full adder in both QCADesigner and HDLQ. The waveforms show the complete simulation from configuring the memory unit, and then test vectors are shifted in. Figure A.1 to Figure A.6 are the waveforms in QCADesigner. The signals OUT_SOUTH and OUT_WEST are the outputs of the full adder, s_i and c_{i+1}. $Data\_Bus\_1$ and $Data\_Bus\_2$ are used to configure the CLB into desired function. In this case, the desired function is the full adder. N_MC, E_MC, W_MC, and S_MC are the memory contents that determine the function of the CLB. IN_NORTH, IN_SOUTH, and IN_EAST are the test vectors for full adder, a_i, b_i, and c_i. The test vectors are not given to the full adder until the memory is completely configured into desired function. Once the test vectors are applied for testing, the corresponding outputs are expected after 16 cycles.

Figure A.7 to Figure A.9 are the waveforms of full adder in HDLQ with ModelSim. $data0\_in0$ and $data0\_in1$ are used to configure the CLB as a full adder. in_a, in_b, and in_c are the test vector inputs, a_i, b_i, and c_i. Zone 1 is the clock when the outputs are triggered. $MC0\_east$, $MC0\_west$, $MC0\_south$, and $MC0\_north$ are the memory contents that determine the function of the CLB. $c1$ and $s0$ are the simulating outputs, s_i and c_{i+1}.

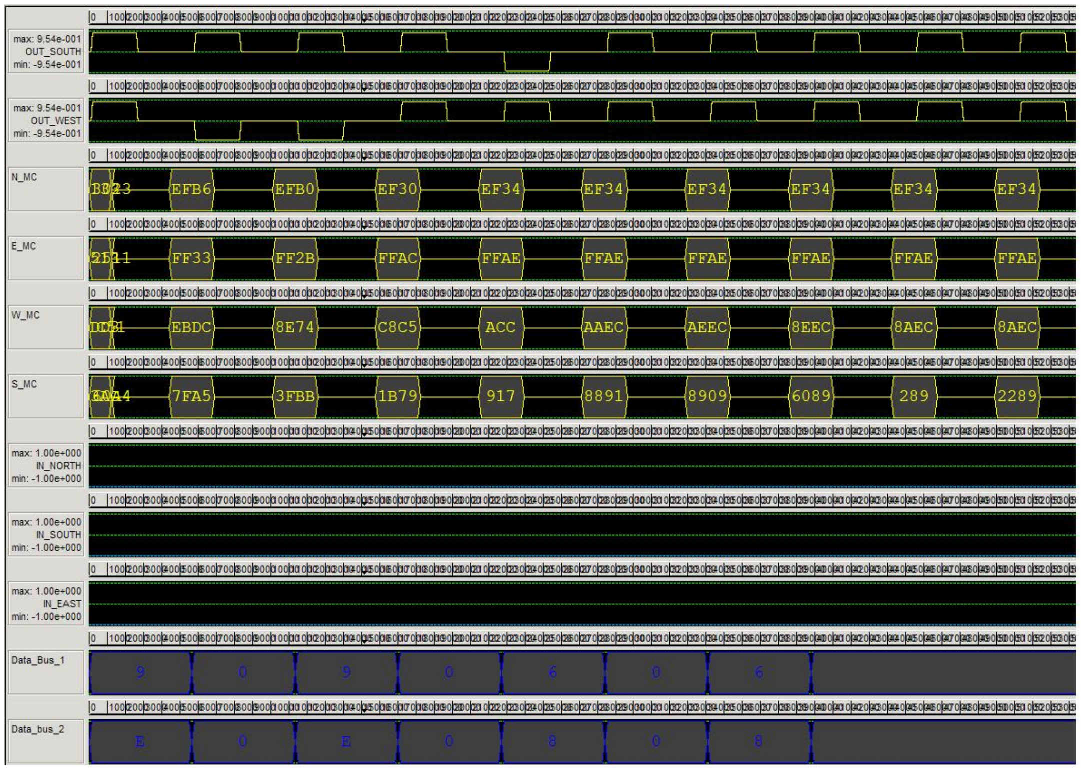


Figure A.1: Full adder waveform in QCADesigner 1 of 6.

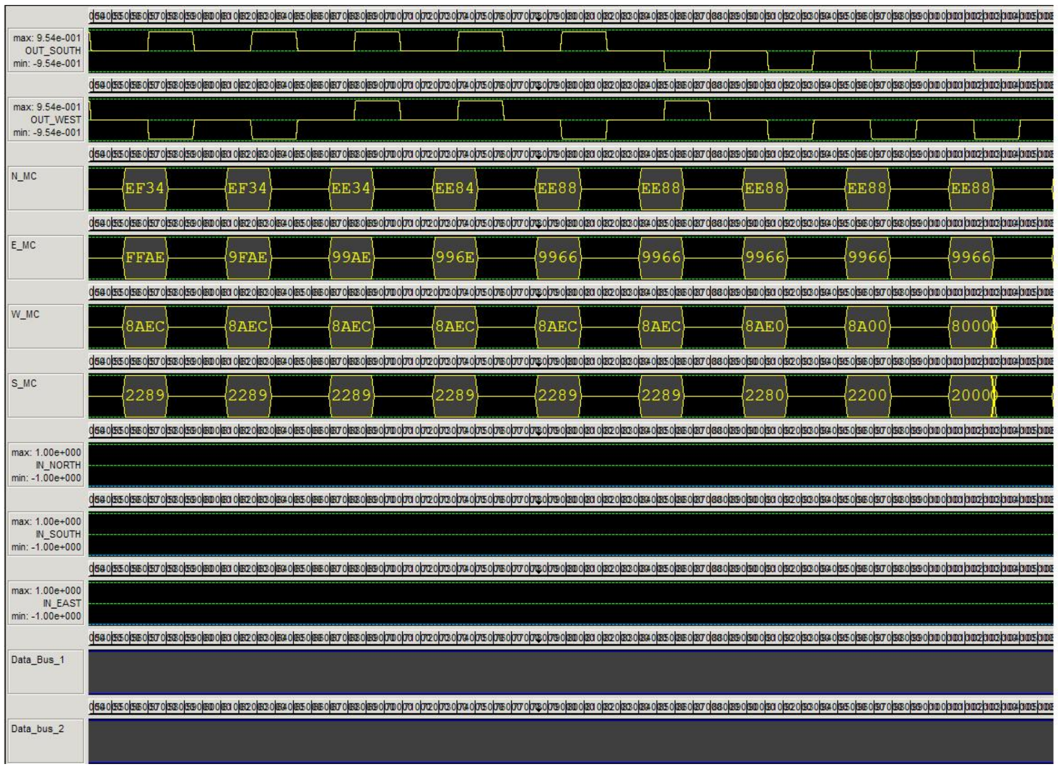


Figure A.2: Full adder waveform in QCADesigner 2 of 6.

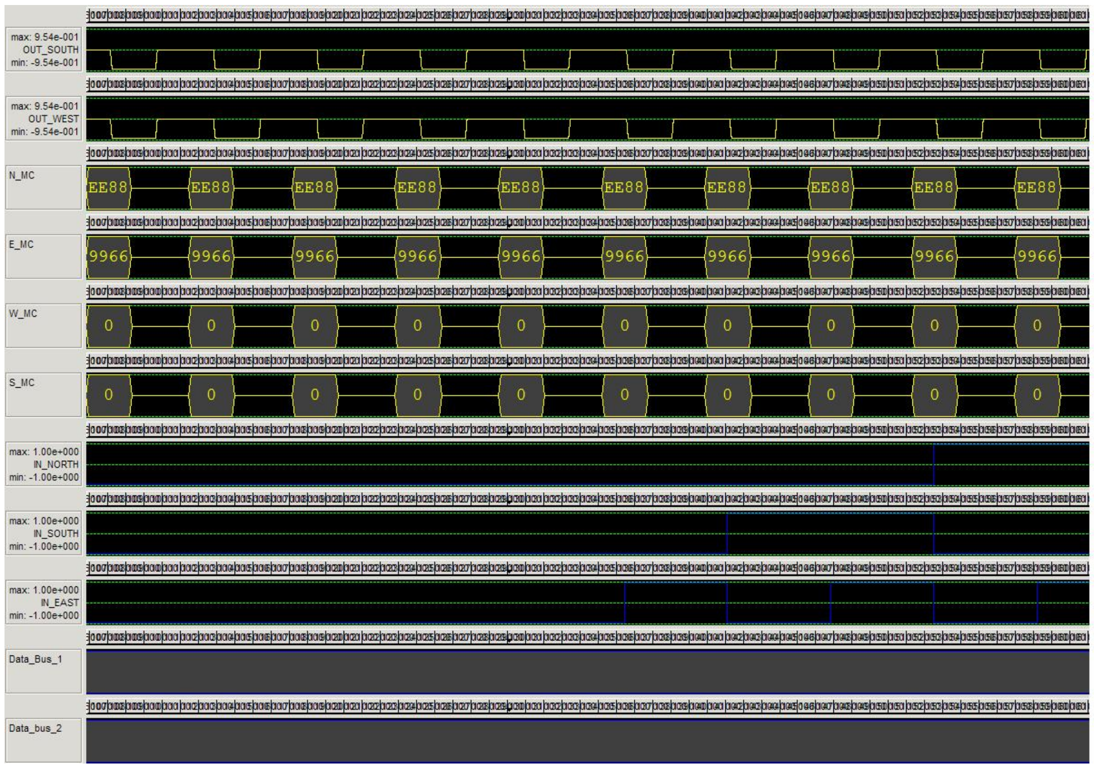


Figure A.3: Full adder waveform in QCADesigner 3 of 6.

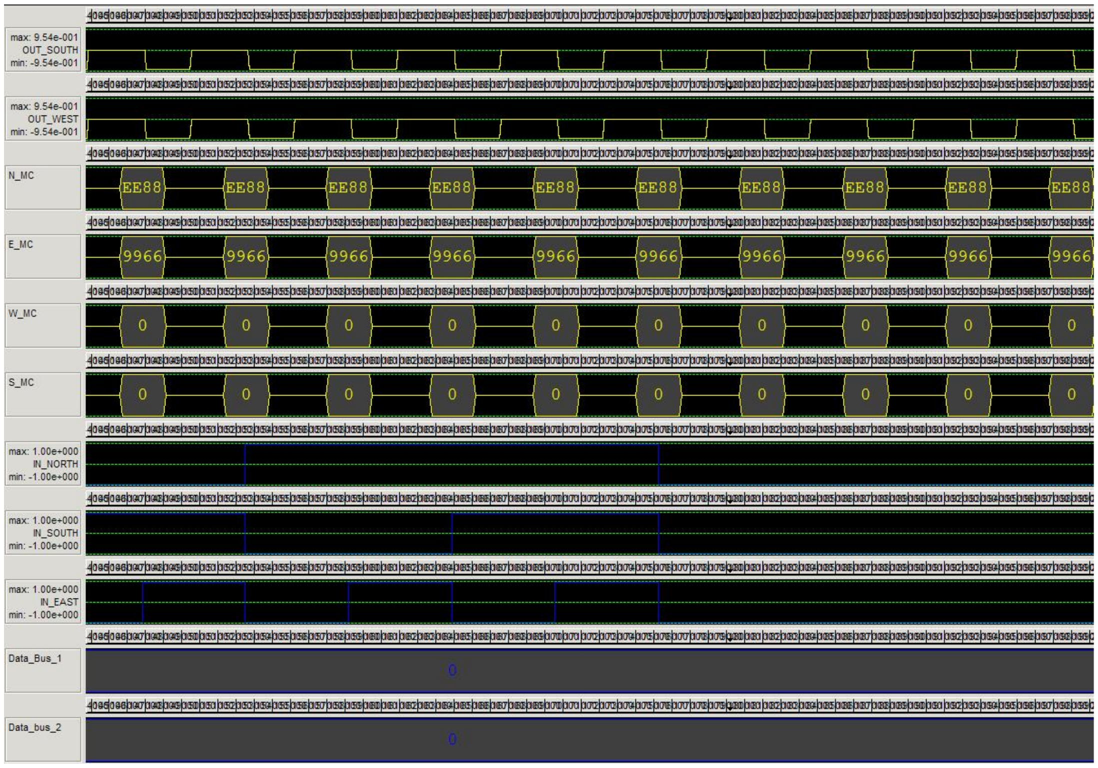


Figure A.4: Full adder waveform in QCADesigner 4 of 6.

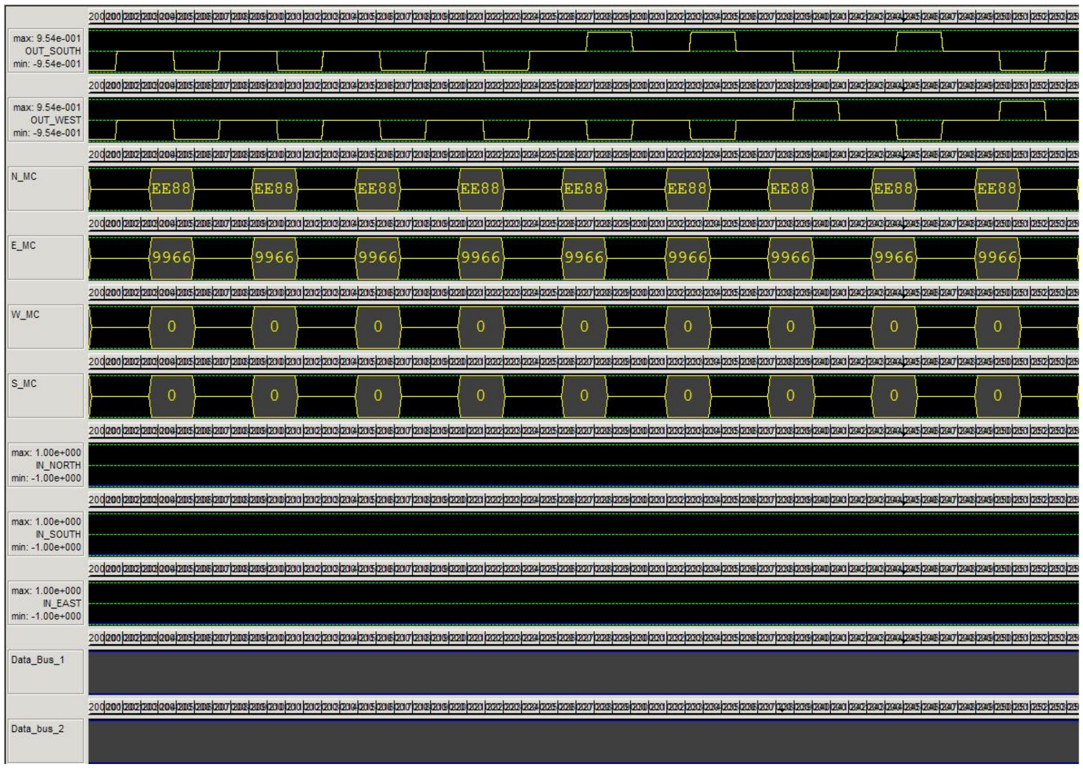


Figure A.5: Full adder waveform in QCADesigner 5 of 6.

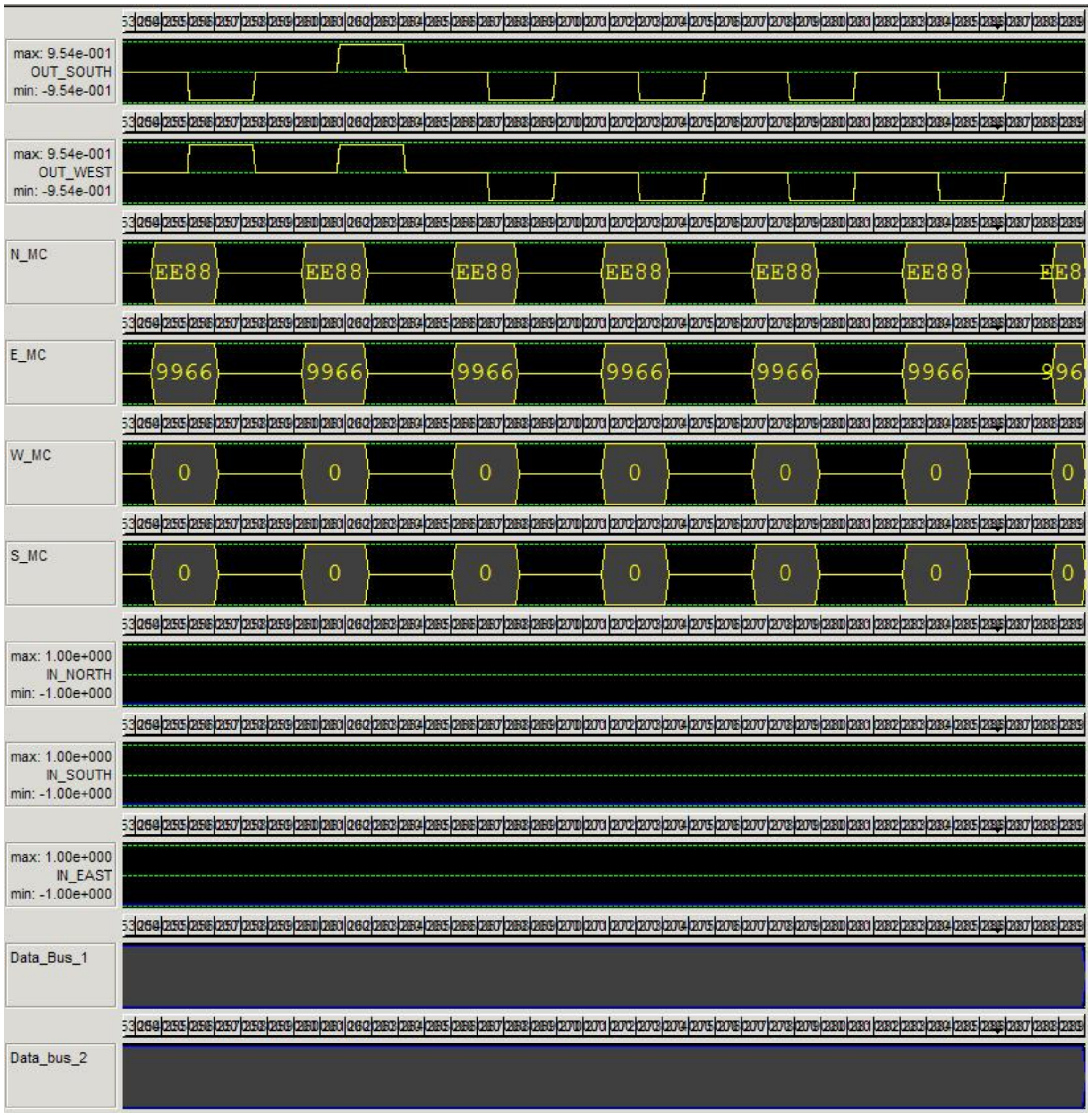


Figure A.6: Full adder waveform in QCADesigner 6 of 6.

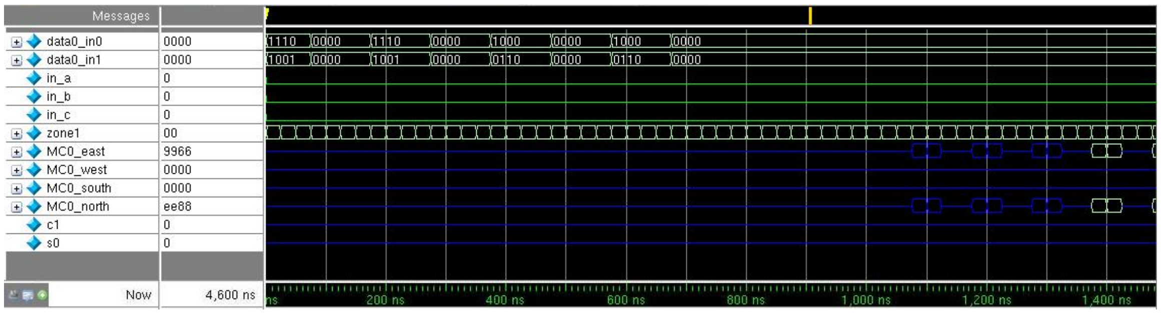


Figure A.7: Full adder waveform in HDLQ 1 of 3.

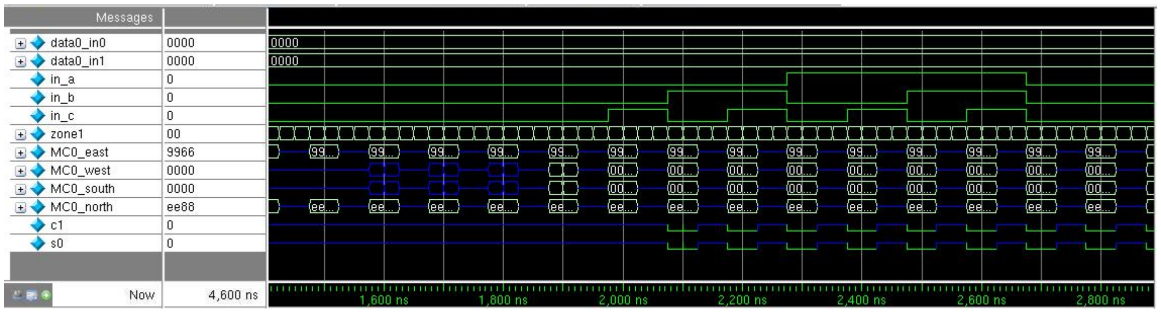


Figure A.8: Full adder waveform in HDLQ 2 of 3.

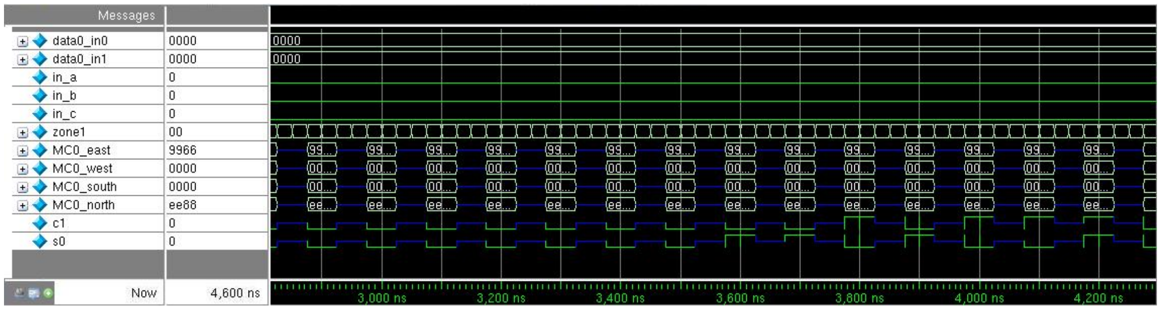


Figure A.9: Full adder waveform in HDLQ 3 of 3.

Appendix B

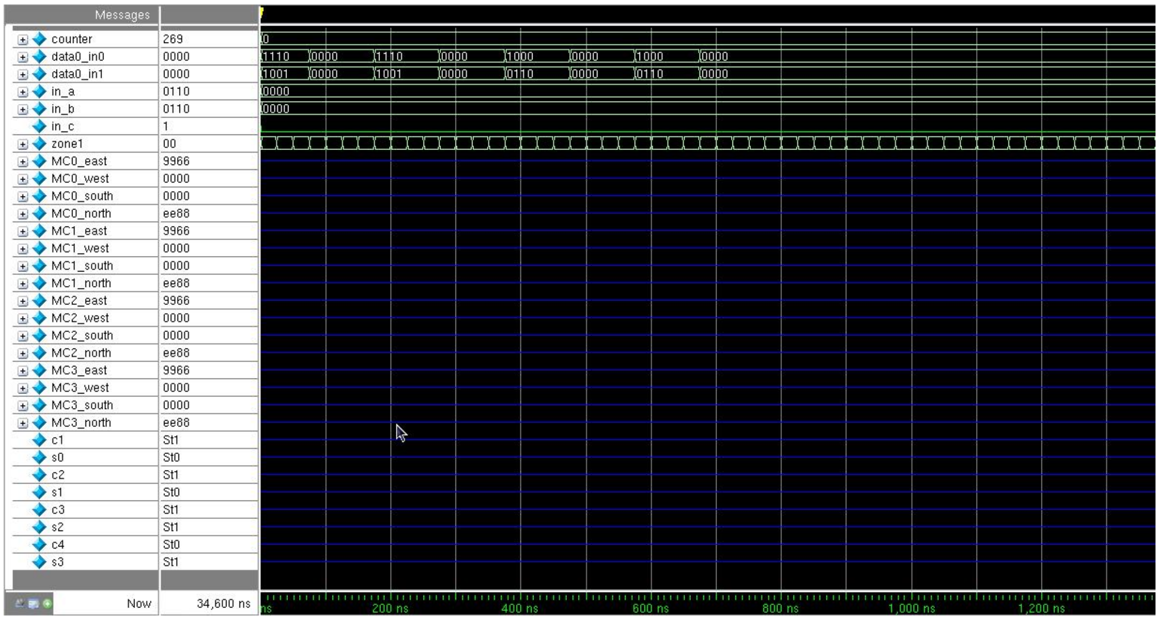
RCA Waveform Simulation

This appendix shows part of the exhaustive test waveforms of the RCA. The following addition processes are being simulated. Beginning with cycle 190, each cycle a new test vector is given to the RCA. Table B.1 shows the test vectors applied to this RCA simulating waveform. The complete 4-bit $A = \langle a_3, a_2, a_1, a_0 \rangle$ and $B = \langle b_3, b_2, b_1, b_0 \rangle$ are given 48 cycles after the first bits are shifted in. $counter$ is a variable that keeps track of the clock cycles and starts to count when the test vectors are applied. $data0\_in0$ and $data0\_in1$ are used to configure all the memory units in the CLB, which are all the MC signals. in_a, in_b, and in_c are the test vector inputs, A, B, and c_0. $c1$ to $c4$ and $s0$ to $s3$ are the outputs of the RCA.

Figure B.1 to Figure B.6 show how the memory contents are being configured to be the RCA. Figure B.7 to Figure B.12 demonstrate the test vectors applied from Table B.1 and the associated outputs.

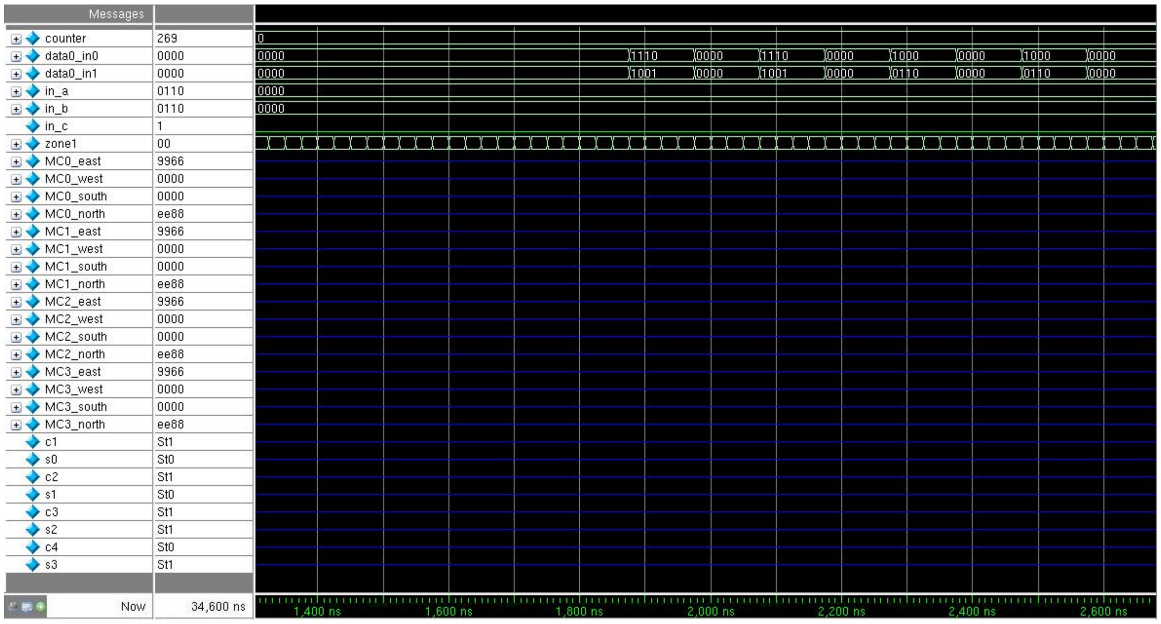
Table B.1: RCA test vectors applied.

Set	Test vectors
1	0111 + 1110
2	1111 + 0110
3	1111 + 1110
4	0001 + 0001
5	0001 + 1001
6	1001 + 0001
7	1001 + 1001
8	0001 + 0101
9	0001 + 1101
10	1001 + 0101
11	1001 + 1101
12	0101 + 0001
13	0101 + 1001
14	1101 + 0001
15	1101 + 1001



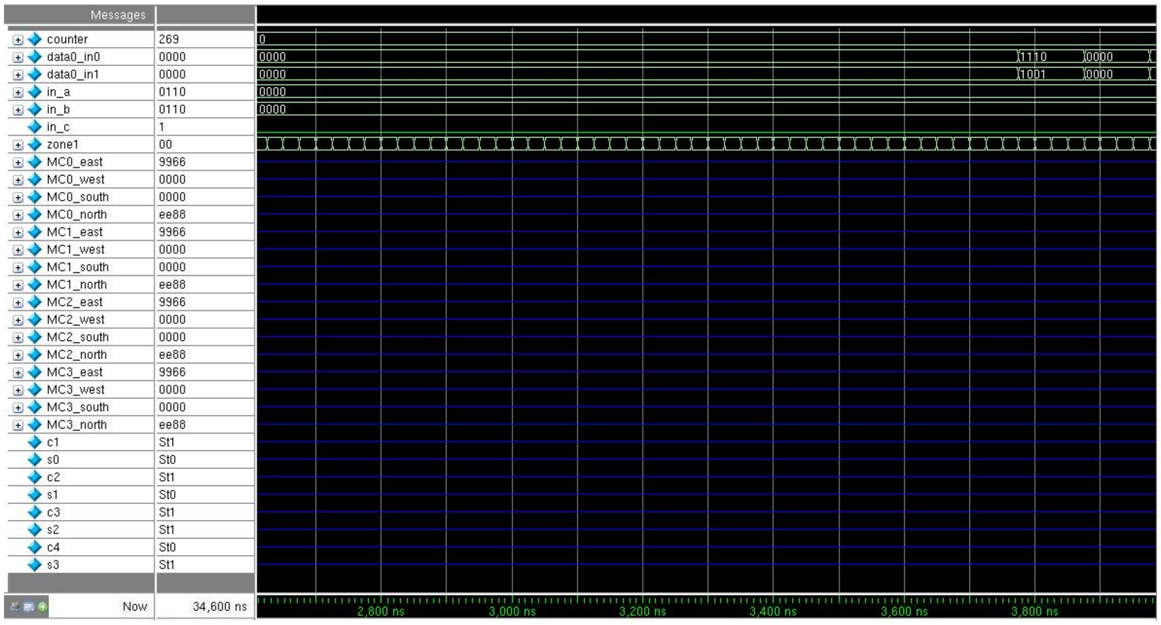
Messages	
counter	269
data0_in0	0000
data0_in1	0000
in_a	0110
in_b	0110
in_c	1
zone1	00
MC0_east	9966
MC0_west	0000
MC0_south	0000
MC0_north	ee88
MC1_east	9966
MC1_west	0000
MC1_south	0000
MC1_north	ee88
MC2_east	9966
MC2_west	0000
MC2_south	0000
MC2_north	ee88
MC3_east	9966
MC3_west	0000
MC3_south	0000
MC3_north	ee88
c1	St1
s0	St0
c2	St1
s1	St0
c3	St1
s2	St1
c4	St0
s3	St1
Now	34,600 ns

Figure B.1: RCA waveform in HDLQ 1 of 12.



Messages	
counter	269
data0_in0	0000
data0_in1	0000
in_a	0110
in_b	0110
in_c	1
zone1	00
MC0_east	9966
MC0_west	0000
MC0_south	0000
MC0_north	ee88
MC1_east	9966
MC1_west	0000
MC1_south	0000
MC1_north	ee88
MC2_east	9966
MC2_west	0000
MC2_south	0000
MC2_north	ee88
MC3_east	9966
MC3_west	0000
MC3_south	0000
MC3_north	ee88
c1	St1
s0	St0
c2	St1
s1	St0
c3	St1
s2	St1
c4	St0
s3	St1
Now	34,600 ns

Figure B.2: RCA waveform in HDLQ 2 of 12.



Messages	
counter	269
data0_in0	0000
data0_in1	0000
in_a	0110
in_b	0110
in_c	1
zone1	00
MC0_east	9966
MC0_west	0000
MC0_south	0000
MC0_north	ee88
MC1_east	9966
MC1_west	0000
MC1_south	0000
MC1_north	ee88
MC2_east	9966
MC2_west	0000
MC2_south	0000
MC2_north	ee88
MC3_east	9966
MC3_west	0000
MC3_south	0000
MC3_north	ee88
c1	St1
s0	St0
c2	St1
s1	St0
c3	St1
s2	St1
c4	St0
s3	St1
Now	34,600 ns

Figure B.3: RCA waveform in HDLQ 3 of 12.

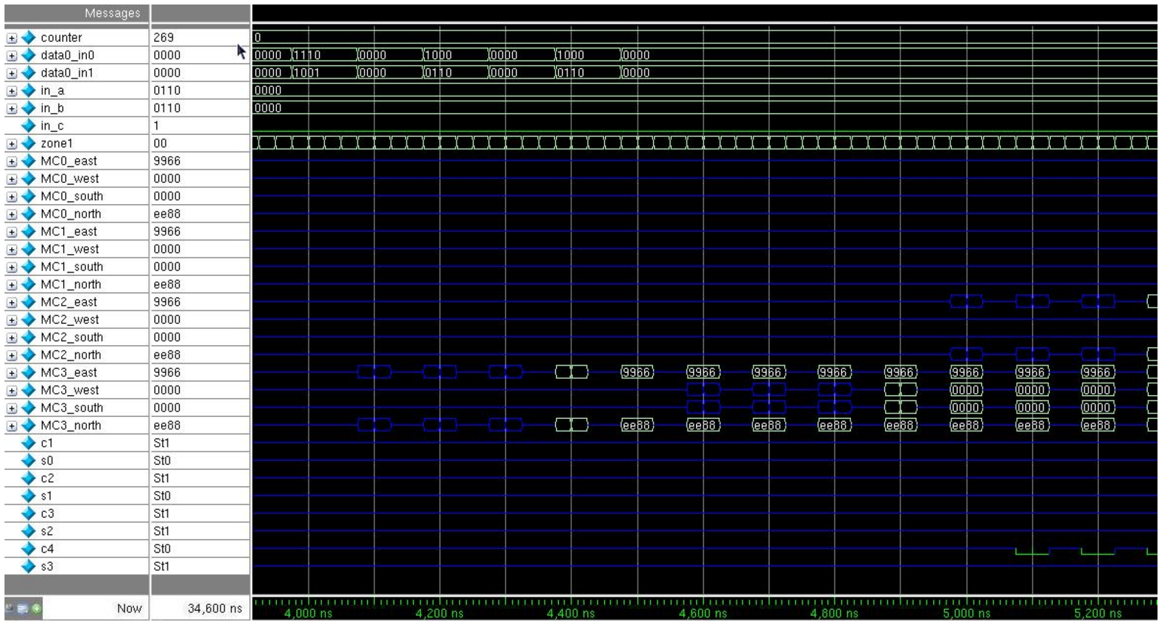


Figure B.4: RCA waveform in HDLQ 4 of 12.

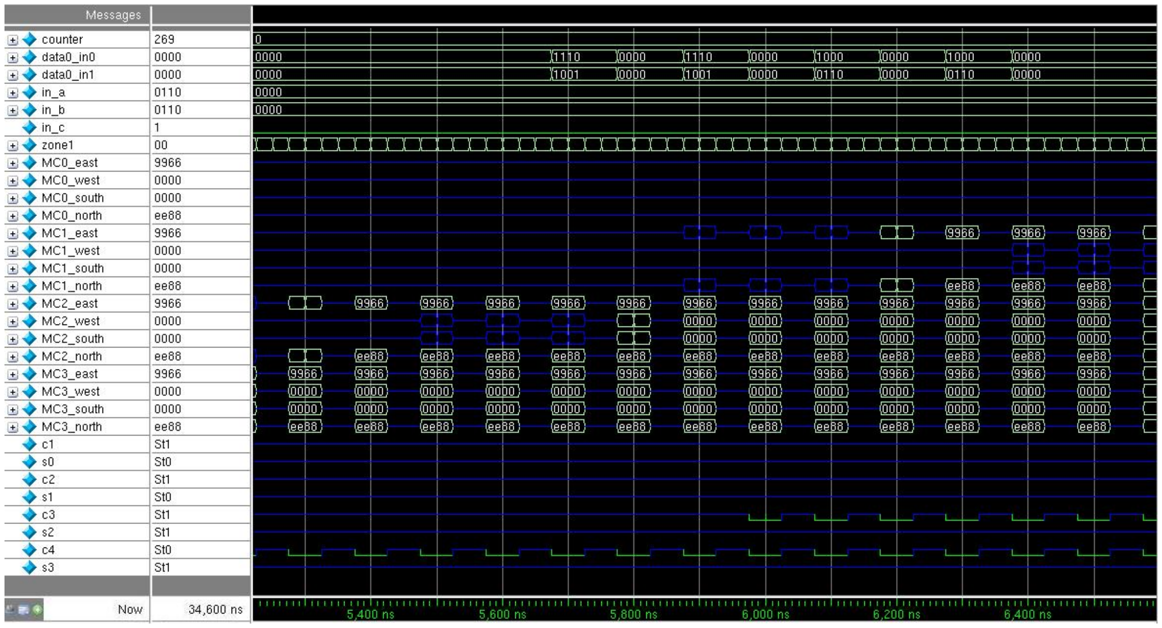
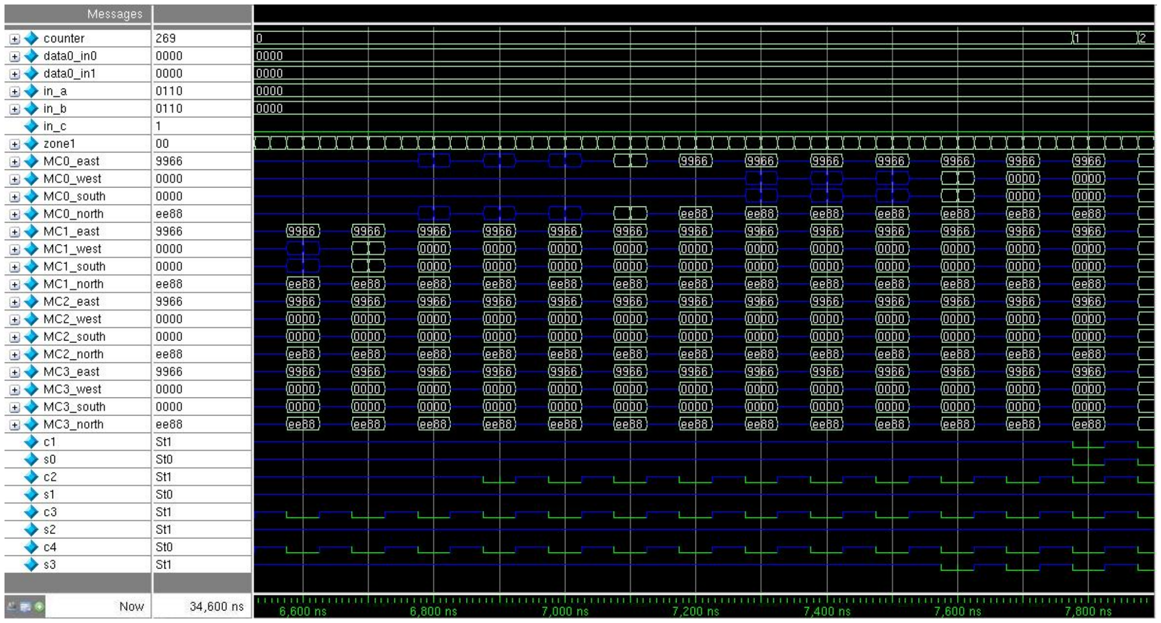
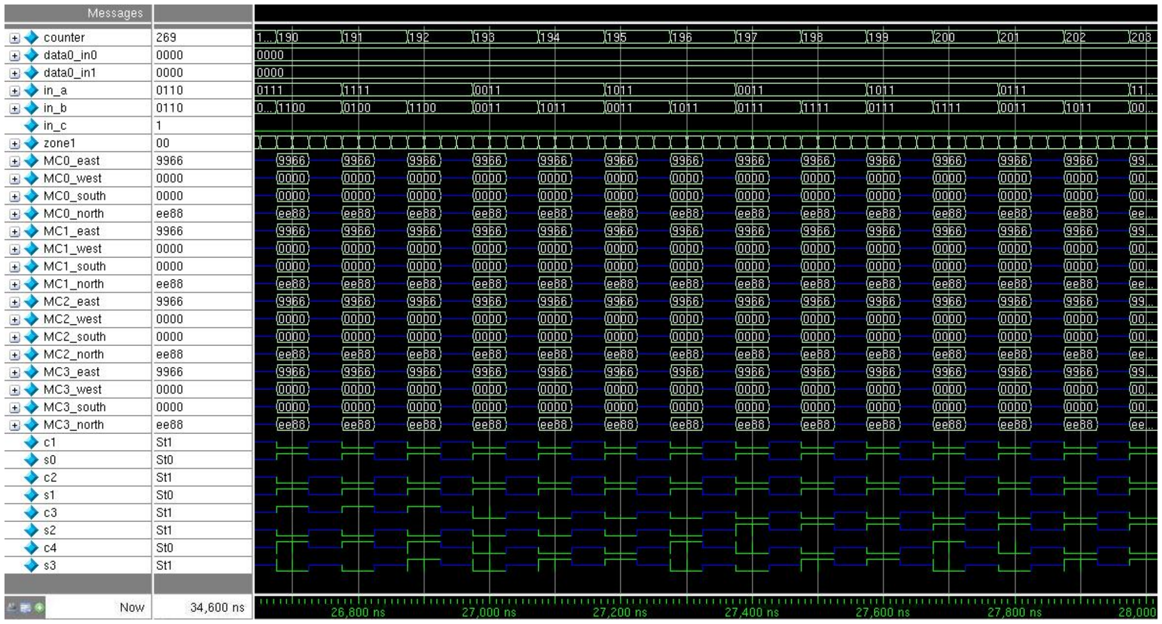


Figure B.5: RCA waveform in HDLQ 5 of 12.



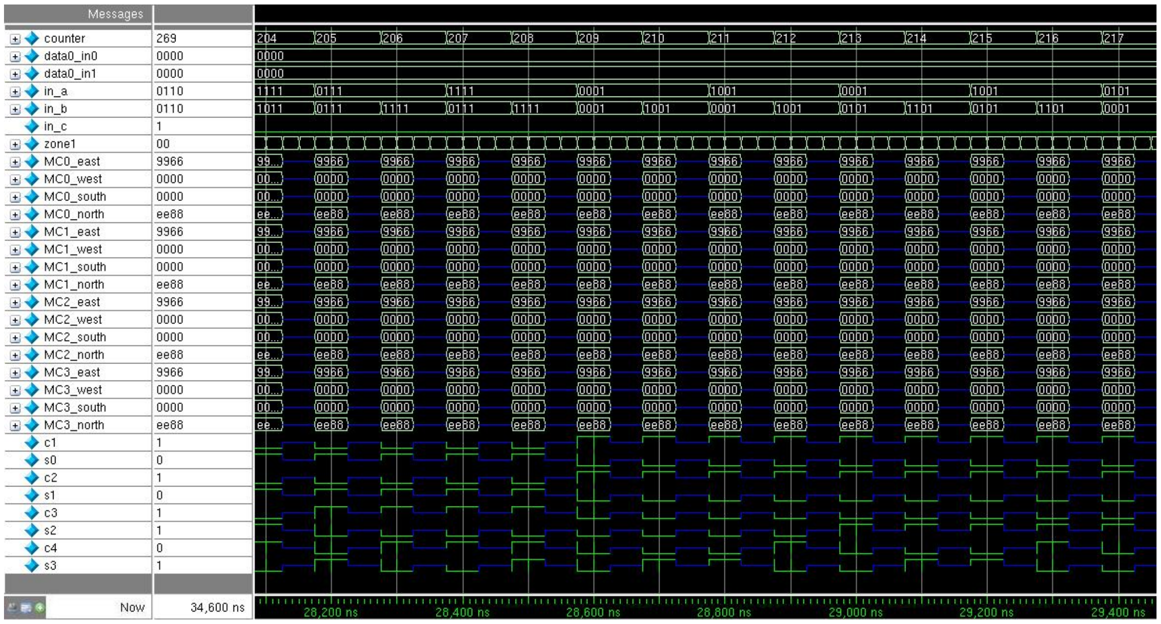
```
Messages
counter      269
data0_in0    0000
data0_in1    0000
in_a         0110
in_b         0110
in_c         1
zone1        00
MC0_east     9966
MC0_west     0000
MC0_south    0000
MC0_north    ee88
MC1_east     9966
MC1_west     0000
MC1_south    0000
MC1_north    ee88
MC2_east     9966
MC2_west     0000
MC2_south    0000
MC2_north    ee88
MC3_east     9966
MC3_west     0000
MC3_south    0000
MC3_north    ee88
c1           St1
s0           St0
c2           St1
s1           St0
c3           St1
s2           St1
c4           St0
s3           St1
Now          34,600 ns
```

Figure B.6: RCA waveform in HDLQ 6 of 12.



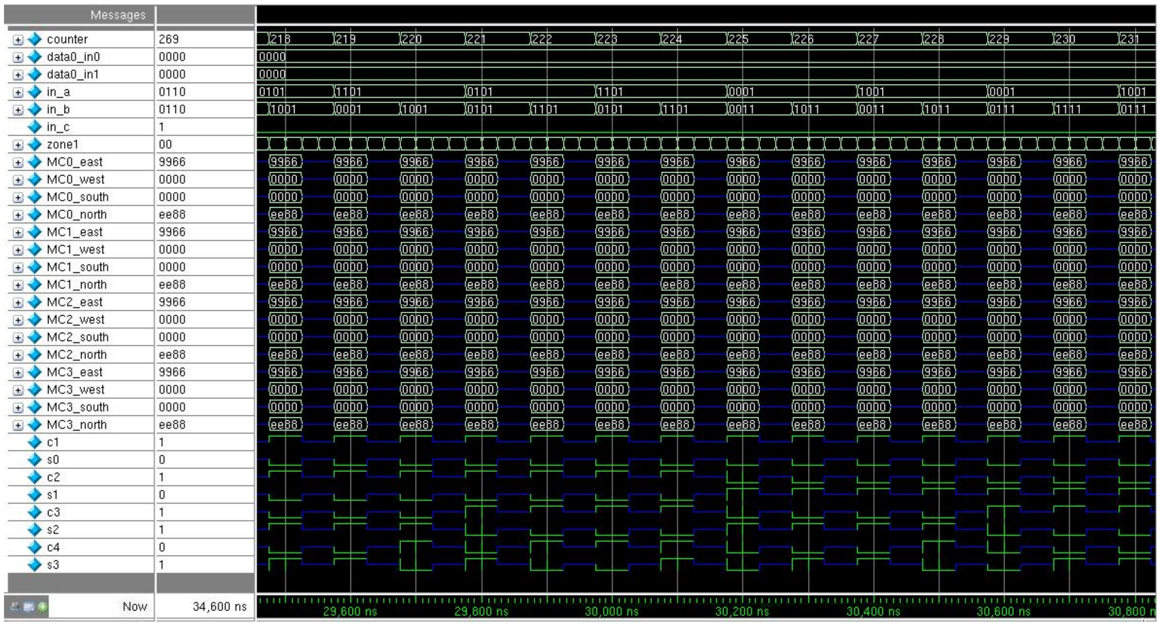
```
Messages
counter      269
data0_in0    0000
data0_in1    0000
in_a         0110
in_b         0110
in_c         1
zone1        00
MC0_east     9966
MC0_west     0000
MC0_south    0000
MC0_north    ee88
MC1_east     9966
MC1_west     0000
MC1_south    0000
MC1_north    ee88
MC2_east     9966
MC2_west     0000
MC2_south    0000
MC2_north    ee88
MC3_east     9966
MC3_west     0000
MC3_south    0000
MC3_north    ee88
c1           St1
s0           St0
c2           St1
s1           St0
c3           St1
s2           St1
c4           St0
s3           St1
Now          34,600 ns
```

Figure B.7: RCA waveform in HDLQ 7 of 12.



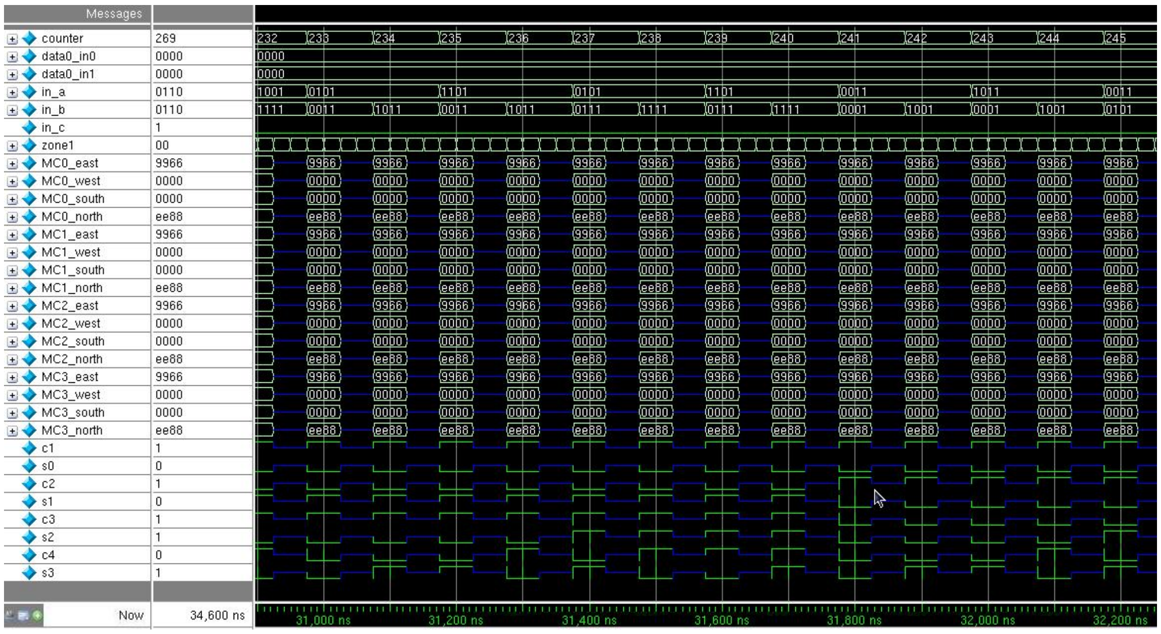
Messages	
counter	269
data0_in0	0000
data0_in1	0000
in_a	0110
in_b	0110
in_c	1
zone1	00
MC0_east	9966
MC0_west	0000
MC0_south	0000
MC0_north	ee88
MC1_east	9966
MC1_west	0000
MC1_south	0000
MC1_north	ee88
MC2_east	9966
MC2_west	0000
MC2_south	0000
MC2_north	ee88
MC3_east	9966
MC3_west	0000
MC3_south	0000
MC3_north	ee88
c1	1
s0	0
c2	1
s1	0
c3	1
s2	1
c4	0
s3	1
Now	34,600 ns

Figure B.8: RCA waveform in HDLQ 8 of 12.



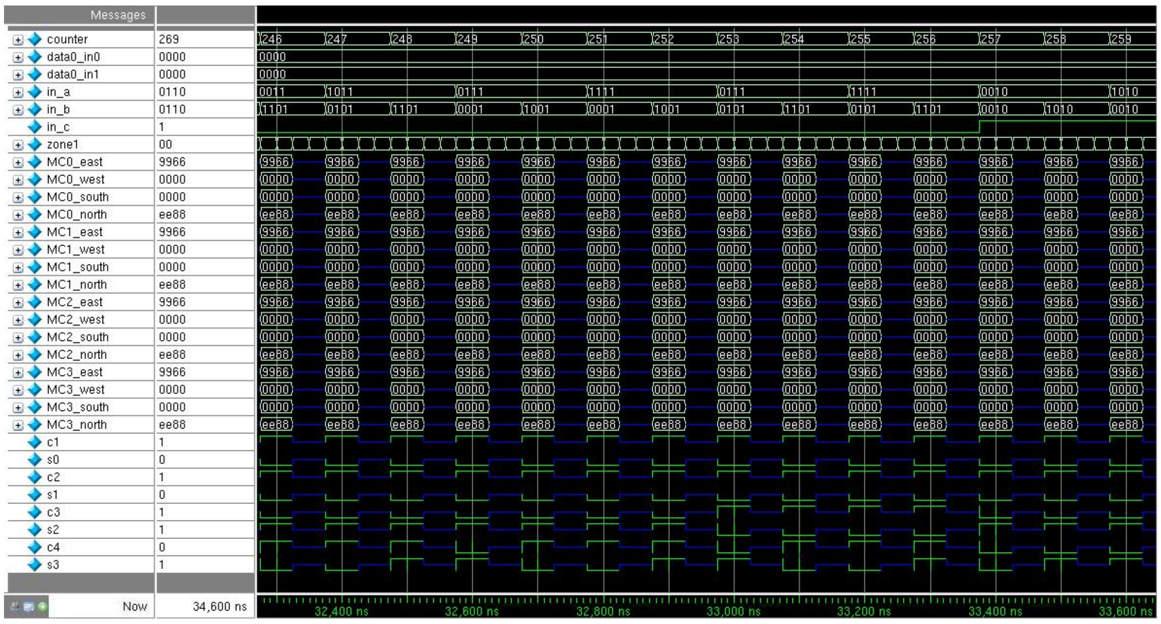
Messages	
counter	269
data0_in0	0000
data0_in1	0000
in_a	0110
in_b	0110
in_c	1
zone1	00
MC0_east	9966
MC0_west	0000
MC0_south	0000
MC0_north	ee88
MC1_east	9966
MC1_west	0000
MC1_south	0000
MC1_north	ee88
MC2_east	9966
MC2_west	0000
MC2_south	0000
MC2_north	ee88
MC3_east	9966
MC3_west	0000
MC3_south	0000
MC3_north	ee88
c1	1
s0	0
c2	1
s1	0
c3	1
s2	1
c4	0
s3	1
Now	34,600 ns

Figure B.9: RCA waveform in HDLQ 9 of 12.



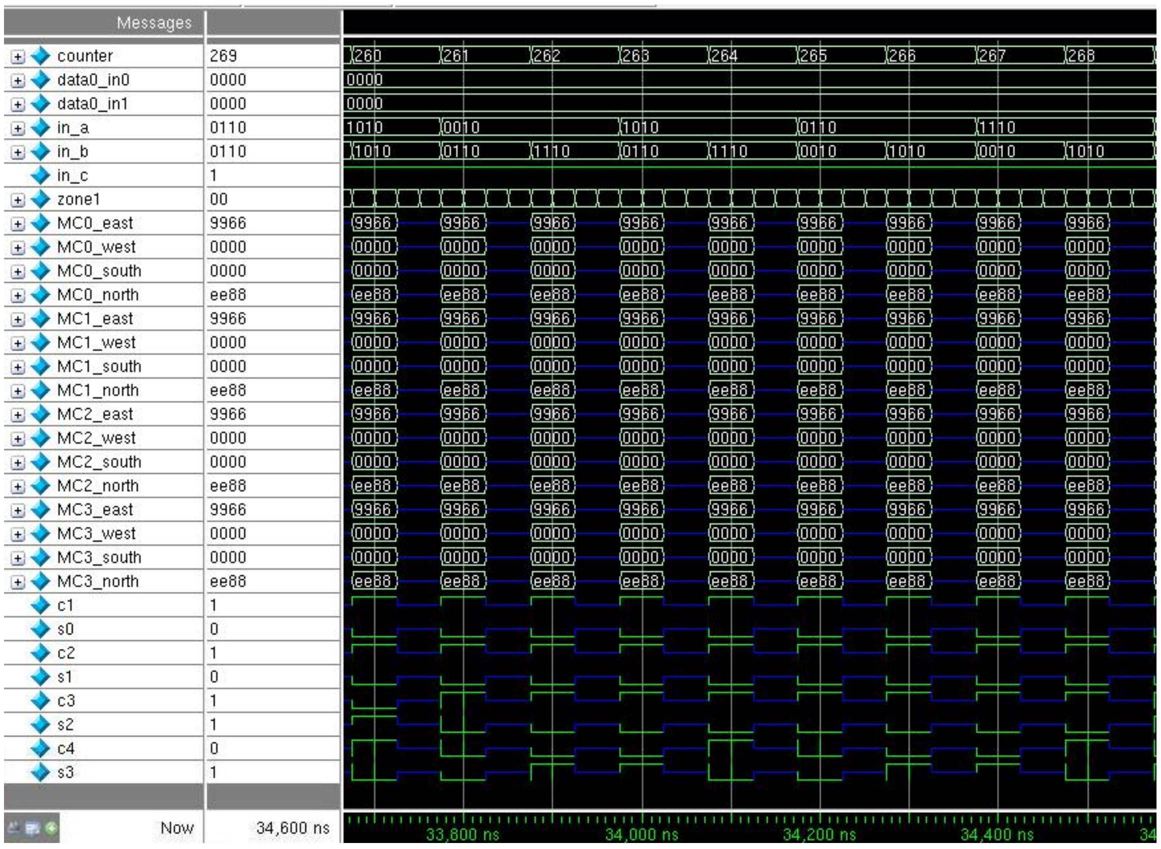
Messages	
counter	269
data0_in0	0000
data0_in1	0000
in_a	0110
in_b	0110
in_c	1
zone1	00
MC0_east	9966
MC0_west	0000
MC0_south	0000
MC0_north	ee88
MC1_east	9966
MC1_west	0000
MC1_south	0000
MC1_north	ee88
MC2_east	9966
MC2_west	0000
MC2_south	0000
MC2_north	ee88
MC3_east	9966
MC3_west	0000
MC3_south	0000
MC3_north	ee88
c1	1
s0	0
c2	1
s1	0
c3	1
s2	1
c4	0
s3	1
Now	34,600 ns

Figure B.10: RCA waveform in HDLQ 10 of 12.



Messages	
counter	269
data0_in0	0000
data0_in1	0000
in_a	0110
in_b	0110
in_c	1
zone1	00
MC0_east	9966
MC0_west	0000
MC0_south	0000
MC0_north	ee88
MC1_east	9966
MC1_west	0000
MC1_south	0000
MC1_north	ee88
MC2_east	9966
MC2_west	0000
MC2_south	0000
MC2_north	ee88
MC3_east	9966
MC3_west	0000
MC3_south	0000
MC3_north	ee88
c1	1
s0	0
c2	1
s1	0
c3	1
s2	1
c4	0
s3	1
Now	34,600 ns

Figure B.11: RCA waveform in HDLQ 11 of 12.



Messages	
counter	269
data0_in0	0000
data0_in1	0000
in_a	0110
in_b	0110
in_c	1
zone1	00
MC0_east	9966
MC0_west	0000
MC0_south	0000
MC0_north	ee88
MC1_east	9966
MC1_west	0000
MC1_south	0000
MC1_north	ee88
MC2_east	9966
MC2_west	0000
MC2_south	0000
MC2_north	ee88
MC3_east	9966
MC3_west	0000
MC3_south	0000
MC3_north	ee88
c1	1
s0	0
c2	1
s1	0
c3	1
s2	1
c4	0
s3	1
Now	34,600 ns

Figure B.12: RCA waveform in HDLQ 12 of 12.

Appendix C

BSM Waveform Simulation

This appendix shows the partial waveforms of the BSM. Table C.1 shows the test vectors applied in the simulation. $counter$ is a virtual clock used in the test bench. a_3 to a_0 join together to form the 4-bit multiplicand A. x_0 is one bit of the multiplier, which is input serially. $east\_L8$ is the actual output from the BSM, and $csf0$ is the expected output. The outputs are produced in the reverse order every other cycle in serial. For an example, Figure C.1 shows the process of 1111×1111. The answer is 11100001. The actual outputs received are 10000111. According to the $counter$, 16 cycles are needed to produce one product. However, the $counter$ is a virtual clock for this simulation. One $counter$ cycle is actually 16 clock cycles, because the proposed CLB has 16 clock cycles of latency.

Table C.1: BSM test vectors applied.

Set	Test vectors
1	1111×1111
2	1110×1111
3	1101×1111
4	1100×1111
5	1011×1111
6	1010×1111
7	1001×1111
8	1000×1111
9	0111×1111

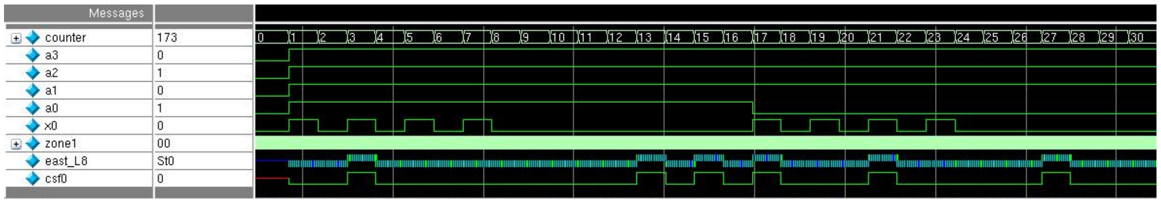


Figure C.1: BSM waveform in HDLQ 1 of 5.

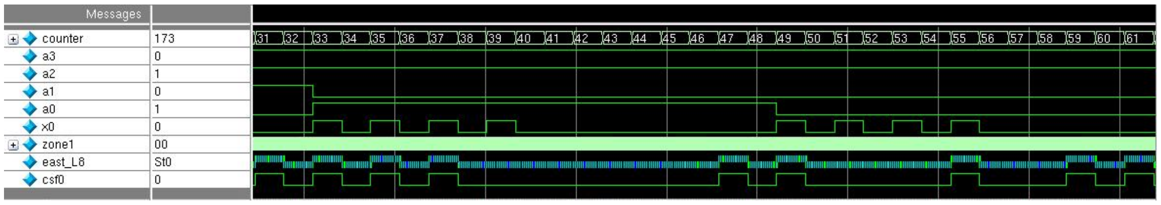


Figure C.2: BSM waveform in HDLQ 2 of 5.

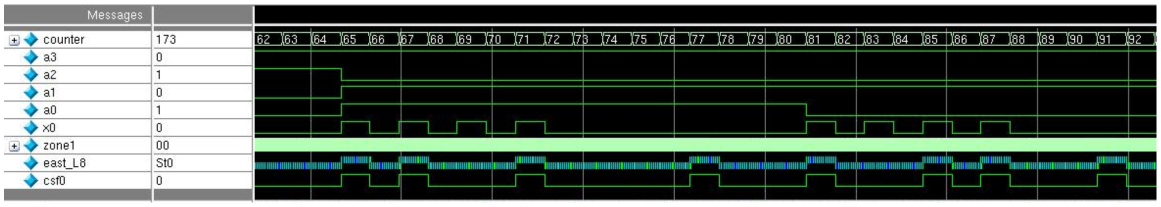


Figure C.3: BSM waveform in HDLQ 3 of 5.

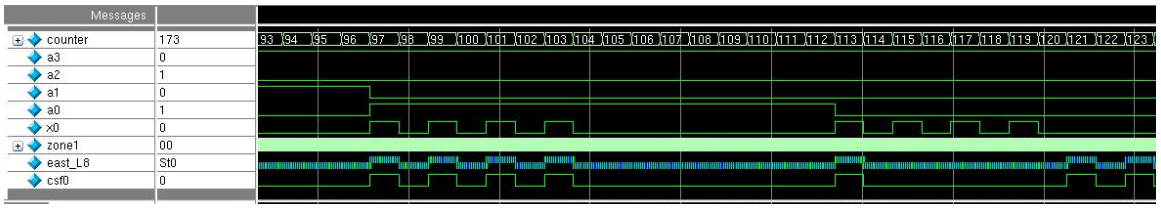


Figure C.4: BSM waveform in HDLQ 4 of 5.

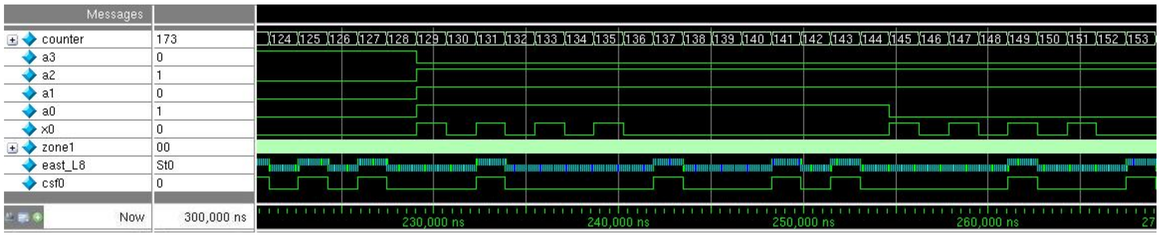


Figure C.5: BSM waveform in HDLQ 5 of 5.

Appendix D

Glitchless Reconfiguration Simulation

This appendix shows the completed simulation waveforms of glitchless reconfiguration in QCADesigner. The function used is still the full adder but the CLB is configured twice while test vectors are applied. Figure D.1 to Figure D.7 are the waveforms in QCADesigner. OUT_SOUTH and OUT_WEST are the outputs, s_i and c_{i+1}. $Data\_Bus\_1$ and $Data\_Bus\_2$ are used to configure the CLB into the desired function, which is stored in N_MC, E_MC, W_MC, and S_MC. IN_NORTH, IN_SOUTH, and IN_EAST are the test vector inputs, a_i, b_i, and c_i.

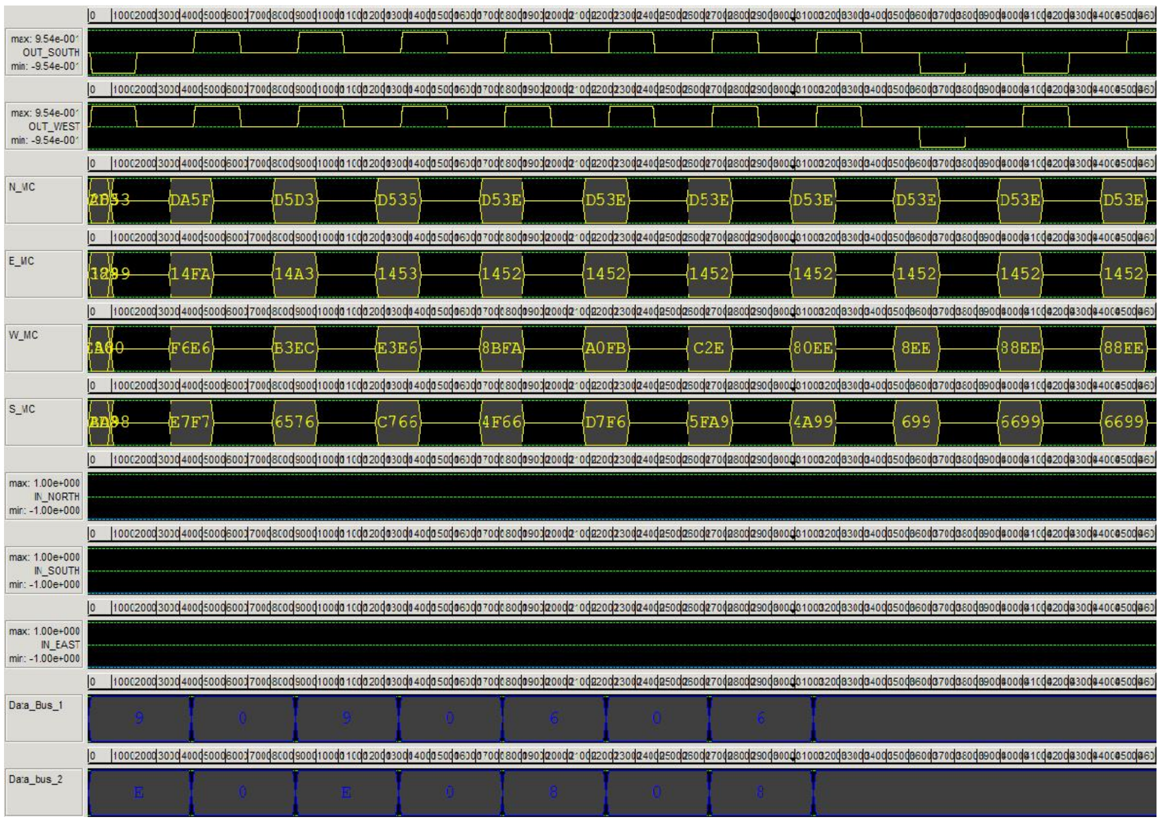


Figure D.1: Glitchless reconfiguration waveform 1 of 7.

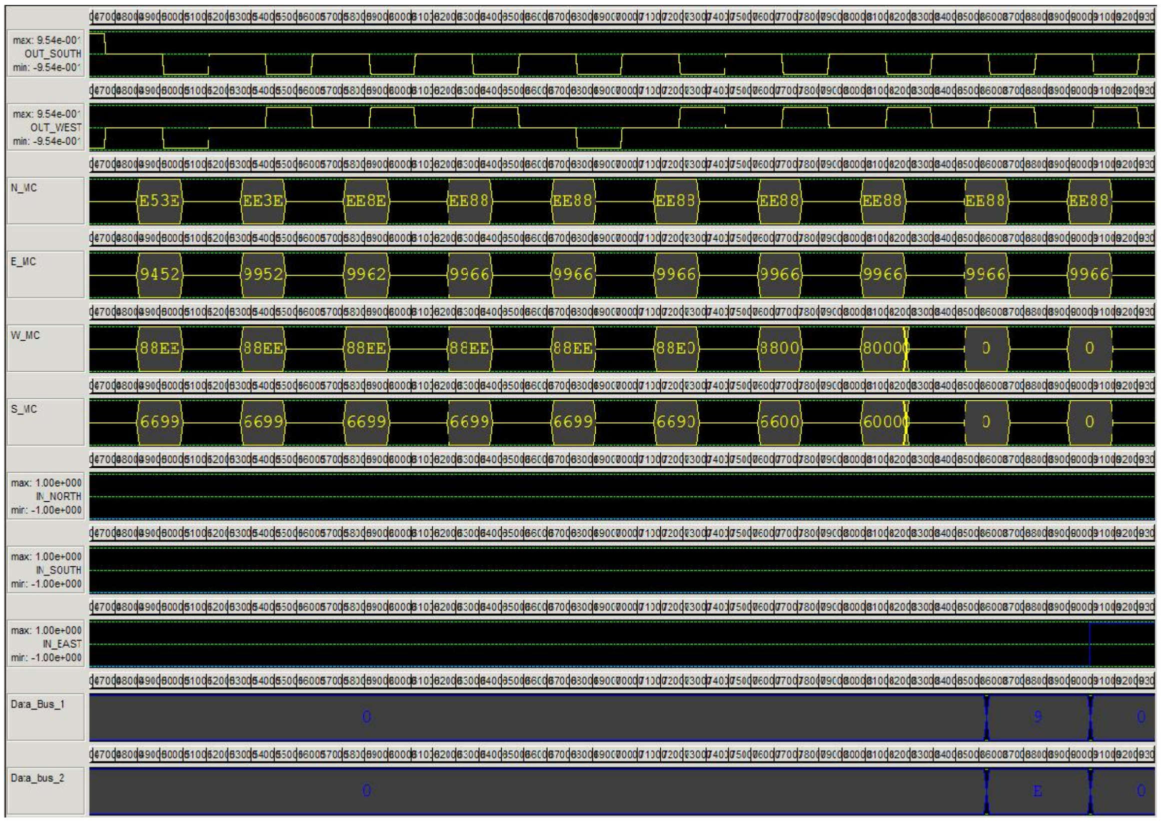


Figure D.2: Glitchless reconfiguration waveform 2 of 7.

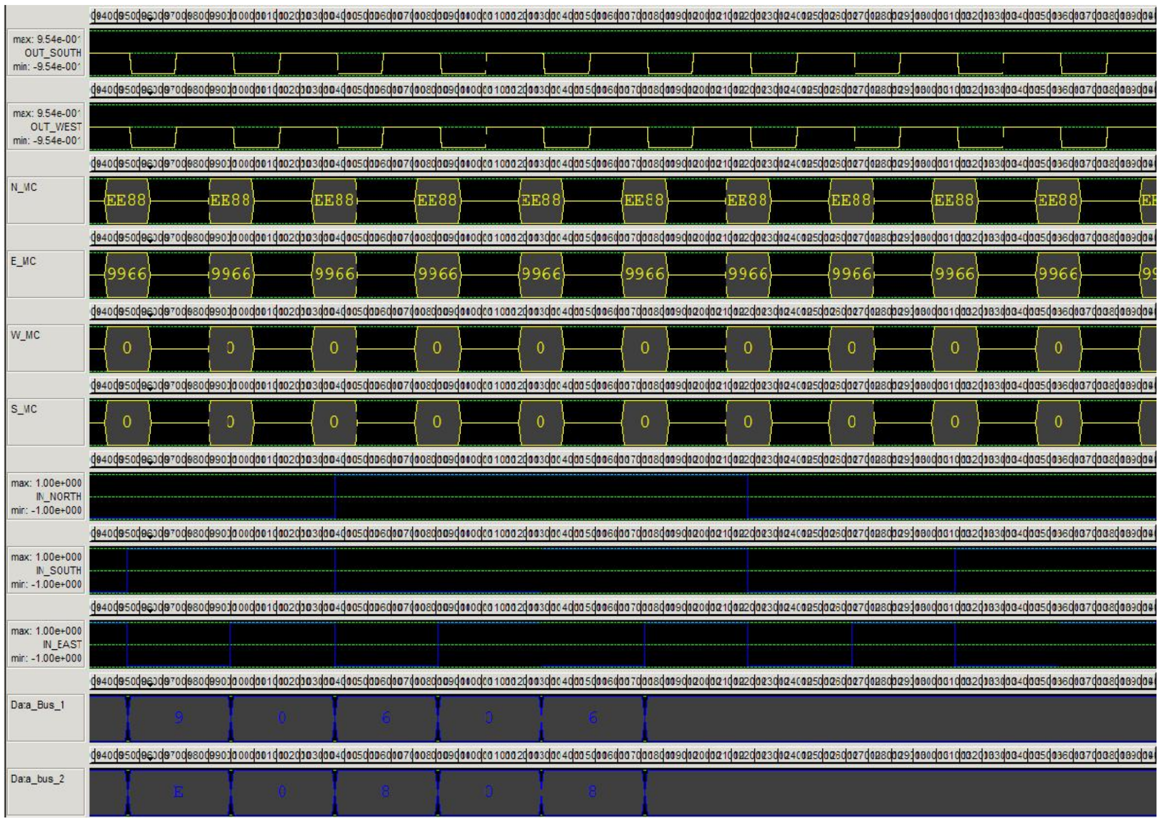


Figure D.3: Glitchless reconfiguration waveform 3 of 7.

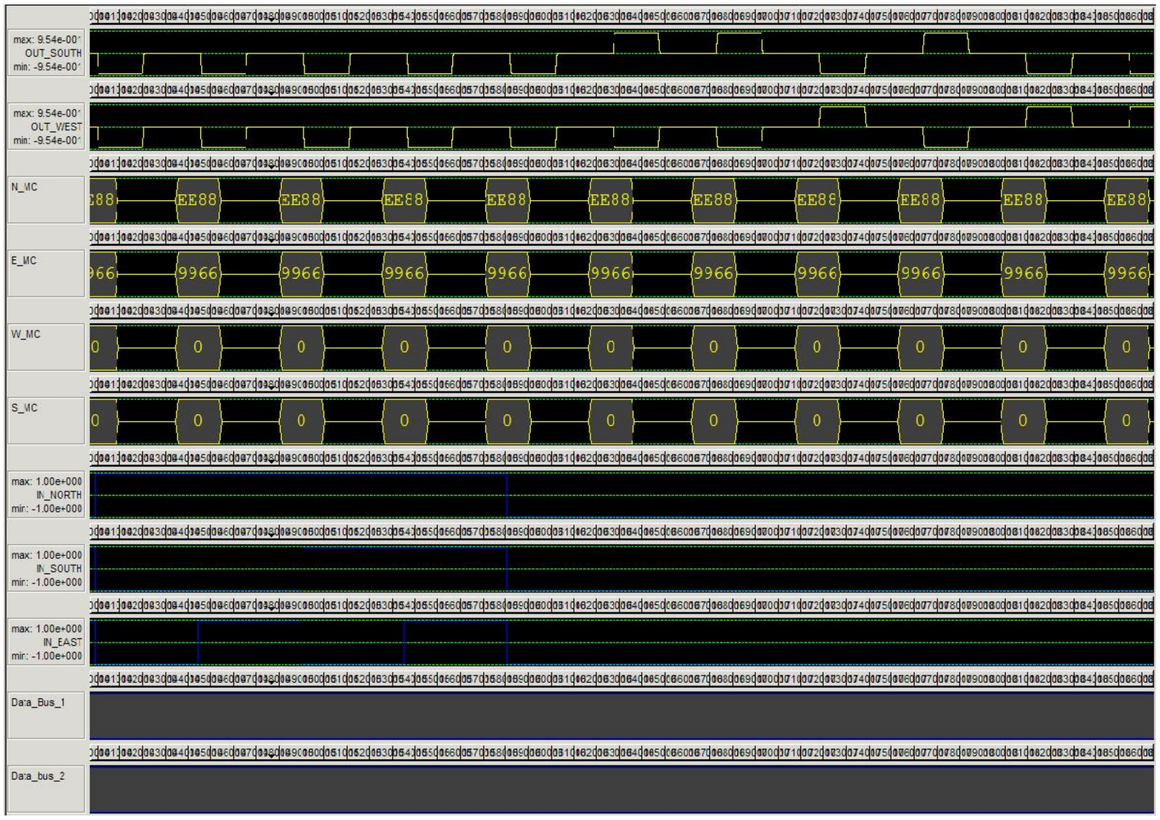


Figure D.4: Glitchless reconfiguration waveform 4 of 7.

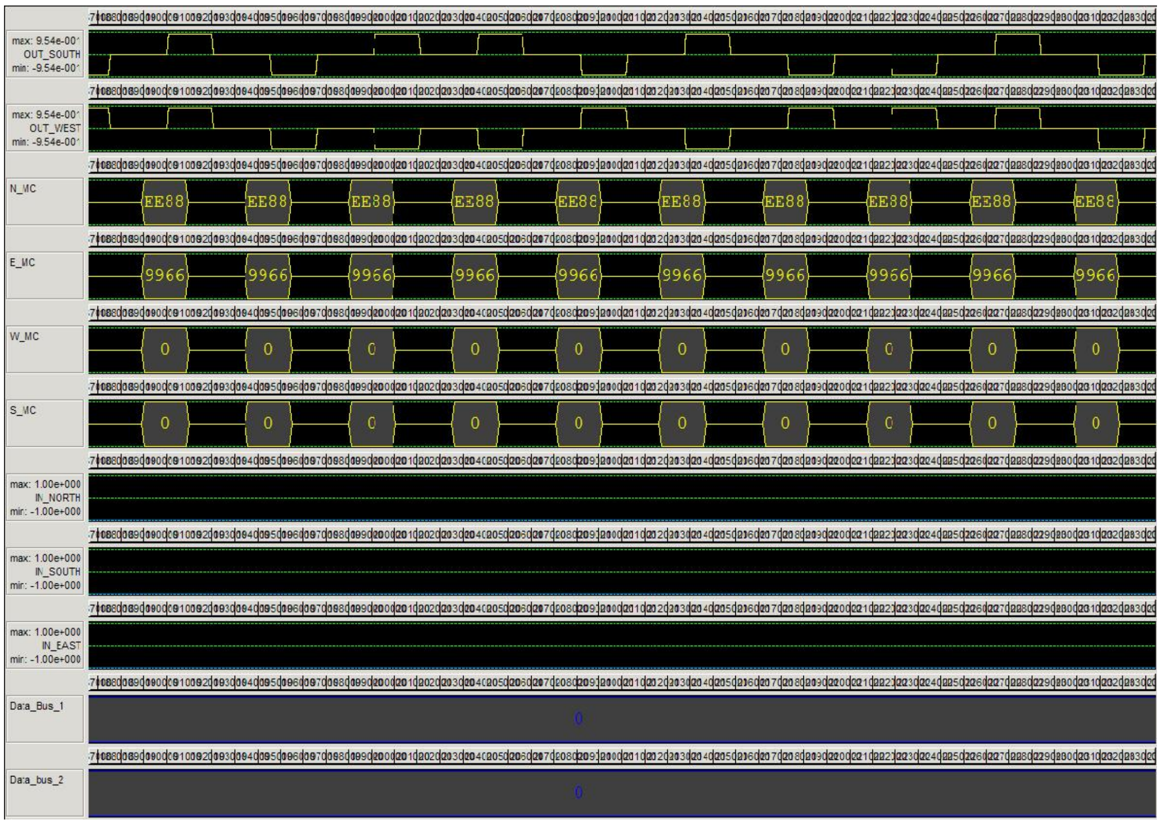


Figure D.5: Glitchless reconfiguration waveform 5 of 7.

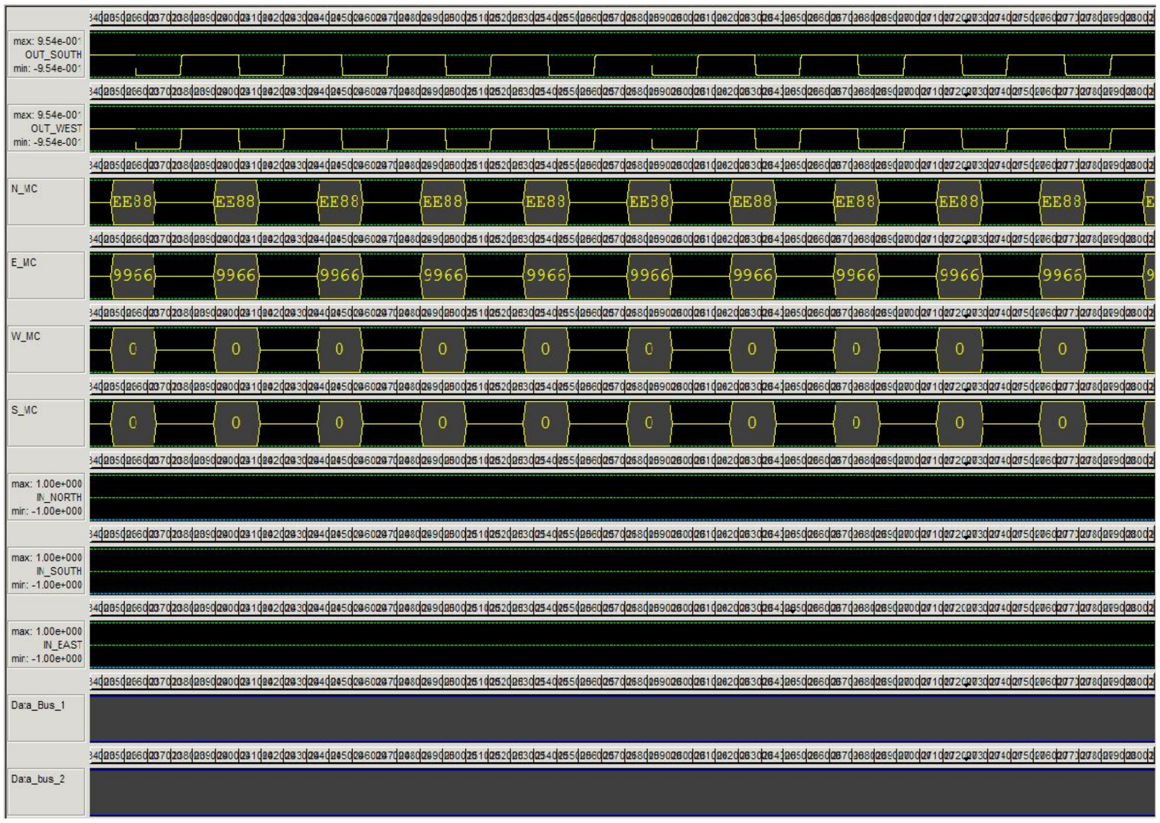


Figure D.6: Glitchless reconfiguration waveform 6 of 7.

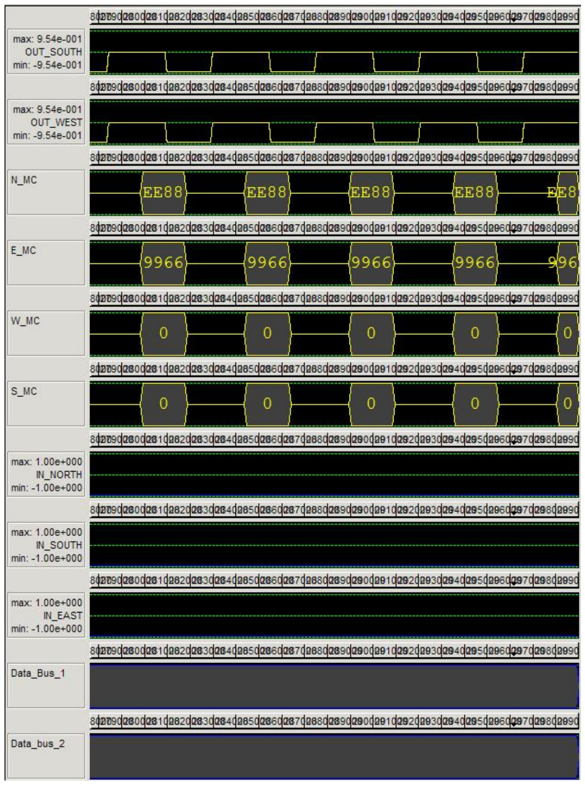


Figure D.7: Glitchless reconfiguration waveform 7 of 7.