64-BIT HIGH EFFICIENCY BINARY COMPARATOR IN QUANTUM-DOT CELLULAR AUTOMATA

A PROJECT REPORT

Presented to the Department of Electrical Engineering

California State University, Long Beach

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

By Dinkar Patalay

B.Tech., 2013, Jawaharlal Nehru Technological University, Andhra Pradesh, India

May 2016

ABSTRACT

64-BIT HIGH EFFICIENCY BINARY COMPARATOR IN QUANTUM-DOT CELLULAR AUTOMATA

By

Dinkar Patalay

May 2016

Quantum-dot Cellular Automata (QCA) are proposed models of quantum computation, which are articulated in analogy to Von Neumann's conventional models of cellular automata. These models are worthy for the architecture of ultra-dense low-power and high-performance digital circuits. Efficient solutions have recently been proposed for several arithmetic circuits, such as adders, multipliers, and comparators. Since the design of digital circuits in QCA still poses several challenges, novel implementation strategies and methodologies are highly desirable. This project demonstrates a new design approach oriented to the implementation of binary comparators using QCA. This strategy is implemented for designing various architectures of binary comparator. With respect to existing counterparts, the comparators proposed here exhibit significantly higher speed and reduced overall area.

ACKNOWLEDGMENT

I take immense pleasure in thanking the EE Chair, Dr. Anastasios Chassiakos and other committee members, for having permitted me to carry out this project work.

I wish to express my deep sense of gratitude to my project advisor, Prof. Boi Tran (Ph. D.), for his able guidance and useful suggestions, which helped me in completing the project work in time. He has been a source of inspiration throughout the project and his timely guidance in providing the artifacts is worth mentioning.

Words are inadequate in offering my thanks to the Project Trainees and Project Assistants for their encouragement and cooperation in carrying out the project work.

Finally, yet importantly, I would like to express my heartfelt thanks to my beloved parents for their blessings, my friends/classmates and those who directly or indirectly were involved, helped and wished for the successful completion of this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CB	Cascade-Based
CMOS	Complimentary Metal-Oxide Semiconductor
LSB	Least Significant Bit
MG	Majority Gate
MSB	Most Significant Bit
QCA	Quantum-Dot Cellular Automata
RTL	Register Transfer Level
TB	Tree-Based
VLSI	Very Large Scale Integration

CHAPTER 1

INTRODUCTION

Until a few years ago, engineers across the globe were able to shrink the size of Complimentary Metal-Oxide Semiconductor **(**CMOS) transistors and hence could accommodate higher number of transistors on the same chip. CMOS technology uses transistor as switches, representing binary information as currents and voltages. The devices, however, would have various limitations when they reduce in size: the on/off levels become inadequate, the resistance high, the charge quantized, and the wires very large in comparison.

The most severe limitation is caused by heat generation due to the circuit capacitances. These capacitances will charge to a desired potential and again discharged to ground, mostly dissipating all the energy held in the logic signal. At molecular device densities and operational frequencies of high rates, the power consumption can reach megawatt ranges per square centimeter for a transistor to move one electron across one volt potential. The thought of reaching the physical limits of photo-lithography and also the saturating behavior of physical science has made scientists look for contemporary methods for building VLSI chip.

Quantum-dot cellular automata (QCA) is one such solution which provides circuit densities and clock frequencies many decades beyond the technological peak of CMOS. The idea was introduced in early nineties and has been indisputable in laboratory surroundings. The potential advantages are that the technology exhibits higher speed, reduced overall area, and no repeated dissipation of signal energy. Since then, QCA based logical and arithmetic circuits have attracted lots of attention. In this project, we deal with one such arithmetic circuit – The binary comparator.

An ideal *comparator* or a *magnitude comparator* is a basic logical device that compares two values (or their binary equivalents) at a given instance to determine if the first value is greater than, equal to or less than the second one. Figure 1 below depicts a general binary comparator with inputs *A* and *B*. *A* and *B* are the n-bit input data to the *n-bit binary comparator*. The job of the binary comparator is to compare each bit from Most Significant bit (MSB) to Least Significant bit (LSB) and then decide whether A > B, A = B or A < B.

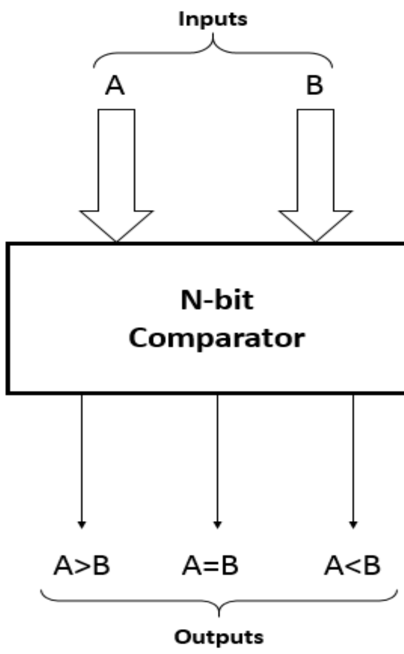


FIGURE 1. A basic n-bit comparator.

Only few examples of comparators have the ability to process n-bit operands with n > 2. The XNOR gate is a digital logic gate whose function is the logical complement of the exclusive OR (XOR) gate. XNOR gate can be used as a basic comparator, which results “true” only when both the inputs are equal. Comparators are key elements with a wide range of applications from Zero-crossing Detector to Central Processing Units.

CHAPTER 2

LITERATIRE REVIEW

Basics of Quantum-Dot Cellular Automata (QCA):

As conventional digital circuit architectures like CMOS are reaching their practical and theoretical limits, the continuous down-scaling of designing electronic circuits is becoming more complicated.

Unlike the CMOS architecture, the information storage and transport on quantum-dot cellular automata is not based on the flow of electrical particle movement, but on the local position of the charged particles inside a small section of the circuit, called a cellular automation. The fundamental unit of QCA is a square shaped QCA cell consisting four quantum dots at each corner, as shown in figure 2(a) [1].

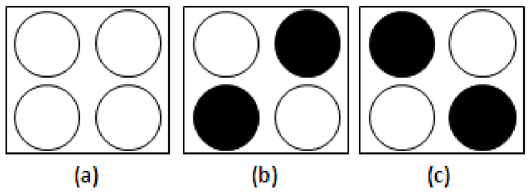


FIGURE 2. (a) Cell; (b) P= +1; (c) P= -1.

A QCA cell has area of 20x20 nm^2. The cells are placed on a grid with a cell center-to-center distance of 25nm; there is at least one cell spacing between adjacent wires. The quantum-dot diameter is 5nm (see figure 3(a) and (b) on page 4). This cell is charged with two extra electrons, which tends to occupy diagonal corners or dots as a result of their coulomb repulsion property. The electrons are permitted to align between the dots in a cell by the theory of quantum mechanical tunneling.

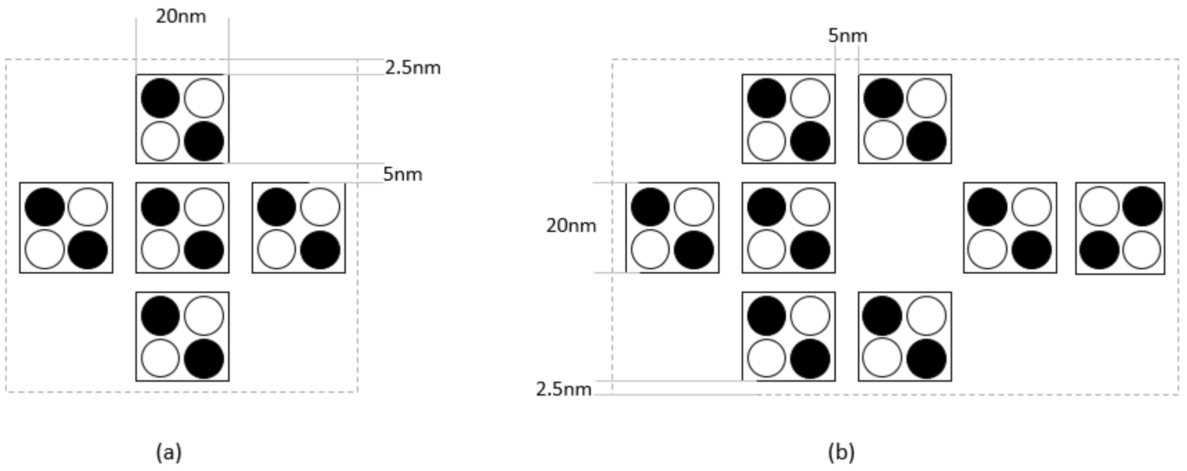


FIGURE 3. (a) MG effective area 90x90; (b) Inverter: effective area 150x90.

However, they are not allowed to tunnel between the two individual cells. Thus, there are only two possible arrangements denoted as cell polarization; P=+1 and P=-1[2]. By using cell polarization, P=+1 that represents logic 1 and P=-1 representing logic 0, binary information can thus be encoded (see figure 2(b) and (c) on page 3). Arrays of such QCA cells can be arranged to perform all wire and logic functions. A QCA wire is used instead of the traditional metal wire to construct a digital logic circuit. In a QCA wire, the binary signal propagates from input to output because of the electrostatic or coulombic interactions between the cells. There are two kinds of QCA wires as shown in figure 4(a) and (b) – one being a binary wire implemented with cells of 90° orientation, and the other, an inversion chain implemented with the cells of 45° orientation.

An input QCA cell is forced to a polarization, say P=+1 drives the next cell into the same polarization, i.e., P=+1; this way, the circuit can maintain minimum energy in the electric field between the charged particles among the neighboring cells. Information is thus transferred to next cells and propagated in the wire consisting of the cell automata.

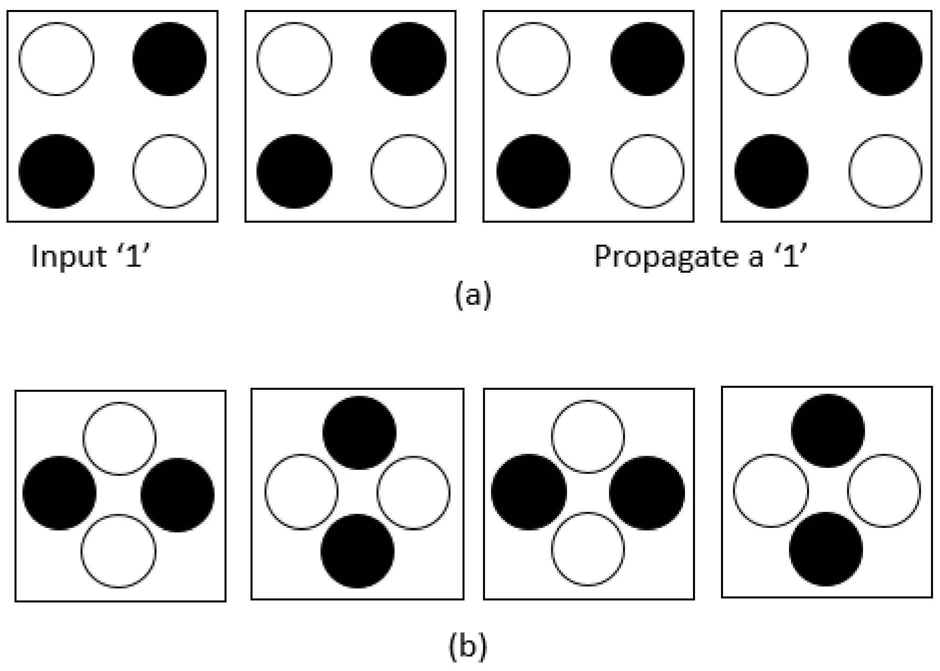
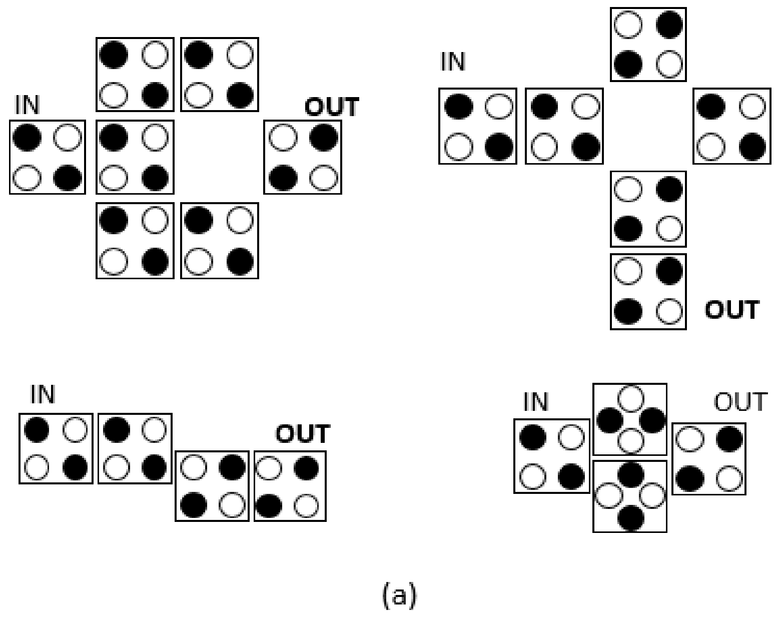


FIGURE 4. (a) 90-degree wire; (b) 45-degree wire.

Primitives Logic Gates in QCA

The QCA cells arranged in a particular manner can form the primitive logic gates shown in figure 5(a) and (b) [3][4]. The simplest structure is that of an inverter usually formed by placing the cells with their corners touching. The electrostatic interaction with minimum electrical energy will show that the output cell polarization is inverted. So you can imagine an inverter to comprise just two cells; one to force the input and the other to obtain the output. However, for a definite inverting outcome, several number of cells are used as shown in figure 3(a) on page 6.



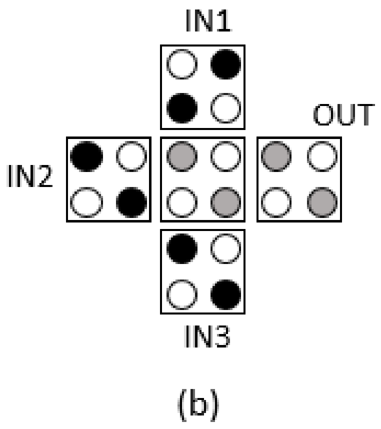


FIGURE 5. QCA primitive logic gates: a) several inverter types, and b) 3-input majority gate.

Another primitive logic gate is the Majority Gate as shown in figure 5(b). The remaining logic gates are usually derived using this majority gate logic[5]. A majority gate returns true only if more than half of its inputs are true, and hence the name. In this project, we deal with a three-input majority gate whose Boolean expression can be given as:

$$M\ (a, b, c) = a.b + b.c + c.a \tag{1}$$

We can see that the gate performs a two-input AND-operation when the third input is fixed at logical **zero**, and when this input is fixed at logical **one**, the same gate functions as a two-input OR-operation. A basic three-input majority gate would be obtained in the QCA structure by placing 9 quantum cells arranged as shown in figure 3(b). Along with the inverter, it is possible for a Majority Gate (MG) to implement any combinatorial computation.

In the equation (1), when $c = 1$; i.e.

M(a, b, 1) = a.b + b.1 + a.1 = a.b + b + a = b.(a + 1) + a = **b + a** => OR gate

In the equation (1), when $c = 0$; i.e.

M(a, b, 1) = a.b + b.0 + a.0 = a.b + 0 + 0 = a.b => AND gate

Likewise, other gates can be formed using the primitive logic gates (*Inverter and Majority Gate*) as shown in table 1 below.

TABLE 1. Logic Gates Implementation Using Primitive Gates

Logic Gate	**Implementation using Primitive Gates**
NAND	NOT AND
NOR	NOT OR
XOR	NOT [AND + NOR]
XNOR	AND + NOR

CHAPTER 3

METHODOLOGY

Implementation of a Comparator Using QCA Gates:

Let us first discuss a 1-bit binary comparator. The device receives two bits, say, A and B as inputs and determines whether they are less than, equal or greater than each other. We represented these states by three output signals, here named A *eq* B, A *big* B, B *big* A. When a comparator can judge all these possible outcomes, then it's a Full Comparator. Non-full comparators would recognize just one or two of them. An exclusive-NOR gate can be an example of a non-full comparator. This can be depicted by the figure 4(a) on page 9. This circuit can verify only whether A equals B. However, the circuits shown in figure 4(b) and (c) on page 9 are full comparators [7].

Let us assume a 4-bit full comparator as shown in the figure 4(d) on page 9. The architecture is referred to as tree-based architecture to achieve high speed. As shown in figure, where 4-bit operands are considered, one instance of the 1-bit comparator presented for each bit position. The intermediate results obtained are then further processed through a series of cascaded 2-input OR and AND gates implemented using majority gates.

As seen from the above figures, the comparators are designed directly under the influence of basic Boolean functions for the CMOS architecture to MGs and inverters; thereby reducing the utilization of computational capability of each MG implemented in the design. As a result, this will impact in more complexity and greater overall delay of the comparators designed in QCA architecture.

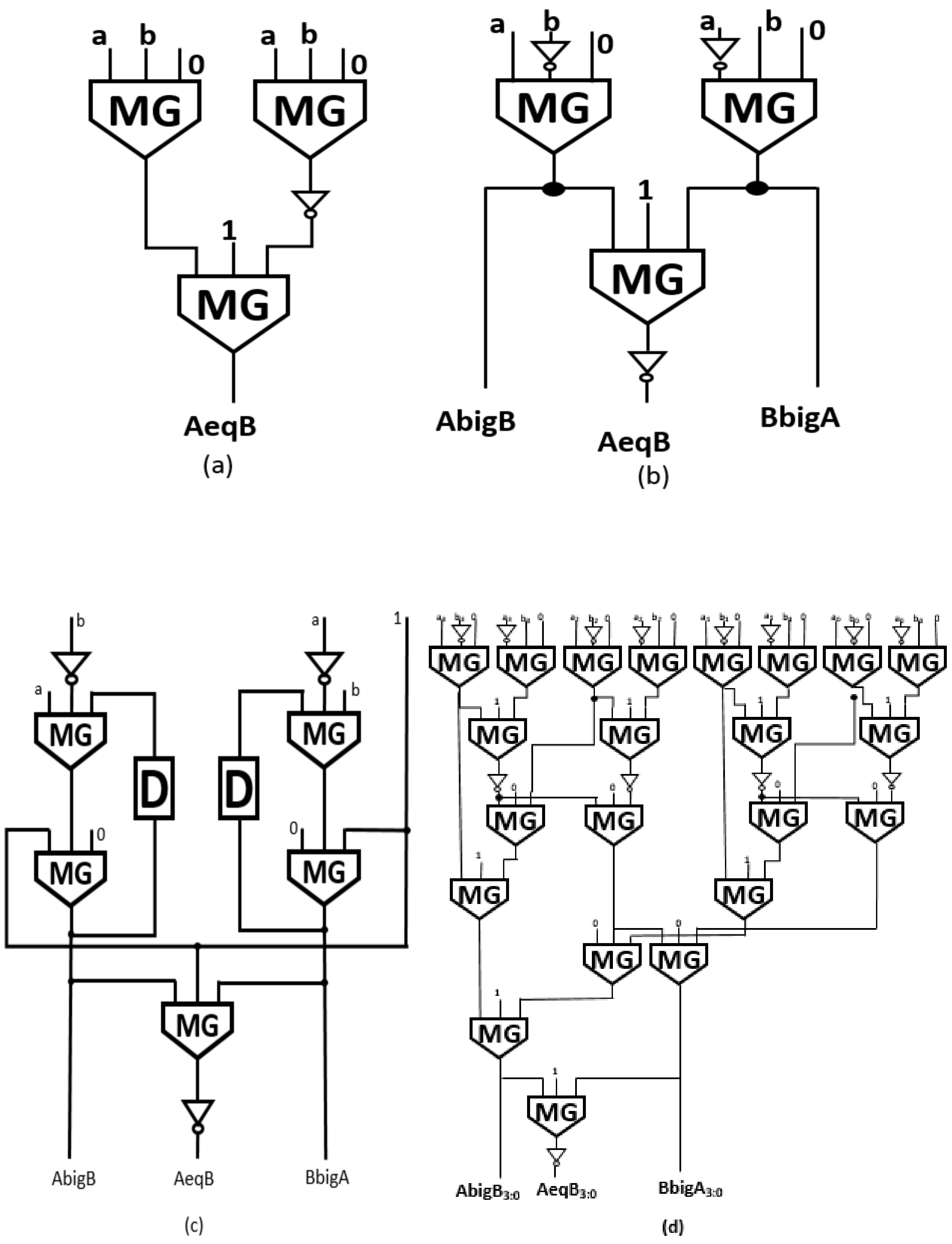


FIGURE 6. QCA based comparators. (a) 1-bit half comparator; (b) 1-bit full comparator; (c) 1-bit full comparator with clock delay; (d) 4-bit full comparator.

New Formulations for QCA Implementations of N-Bit Full Comparators

In order to reduce the above mentioned computational complexity, four original theorems and two corollaries are enunciated that can significantly increase the speed performances of QCA-based designs of full comparators and can significantly reduce the number of used MGs and inverters with respect to existing comparators, thus reducing also the number of used cells and the overall active area.

The novel formulations can be exploited in the design of n-bit full comparators splitting the operands $A_{(n-1:0)}=a_{n-1}...a_0$ and $B_{(n-1:0)}=b_{n-1}...b_0$ into a proper number of 2-bit and 3-bit sub words that can be compared applying Theorems 1 and 2. The intermediate results obtained in this way can be then further processed by applying Theorems 3 and 4 together with Corollaries 1 and 2.

Theorem 1: If $A_{(z-1:z-2)}=a_{z-1}a_{z-2}$ and $B_{(z-1:z-2)}= b_{z-1}b_{z-2}$, with z=2, 4,..., n–2,n, are two 2-bit sub words of the n-bit numbers $A_{(n-1:0)}$and $B_{(n-1:0)}$, respectively, then A big $B_{(z-1:z-2)}$as defined in (2) is equal to 1 if and only if $A_{(z-1:z-2)}>B_{(z-1:z-2)}$; $\overline{B_{big}A_{(z-1:z-2)}}$ as defined in (3) is equal to 0 if and only if $A_{(z-1:z-2)} < B_{(z-1:z-2)}$. Theorem 1 (T1) is depicted in figure 7 below.

$$A_{big}B_{(z-1:z-2)} = M(a_{z-1}, \overline{b_{z-1}}, a_{z-2}) \,.\, M(a_{z-1}, \overline{b_{z-1}}, \overline{b_{z-2}}) \qquad (2)$$

$$\overline{B_{big}A_{(z-1:z-2)}} = M(a_{z-1}, \overline{b_{z-1}}, a_{z-2}) + M(a_{z-1}, \overline{b_{z-1}}, \overline{b_{z-2}}) \qquad (3)$$

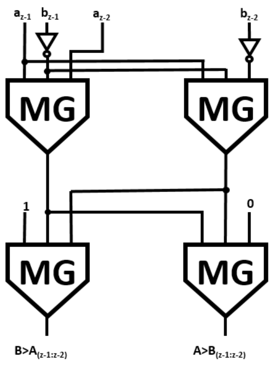


FIGURE 7. Theorem 1(T1).

Theorem 2: If $A_{(z-1:z-3)}=a_{z-1}a_{z-2}a_{z-3}$ and $B_{(z-1:z-3)} = b_{z-1}b_{z-2}b_{z-3}$, with z=3,6,...,n−3,n, are 3-bit sub words of the n-bit numbers $A_{(n-1:0)}$ and $B_{(n-1:0)}$, respectively, then $A_{big}\,B_{(z-1:z-3)}$ as defined in (4) is equal to 1 if and only if $A_{(z-1:z-3)}>B_{(z-1:z-3)}$; $\overline{B_{big}A_{(z-1:z-3)}}$ as given in (5) is equal to 0 if and only if $A_{(z-1:z-3)}<B_{(z-1:z-3)}$. Theorem 2 (T2) is depicted in figure 8 below.

$$A_{big}B_{(z-1:z-3)} = M(M(a_{z-1}, \overline{b_{z-1}}, a_{z-2}), M(a_{z-1}, \overline{b_{z-1}}, \overline{b_{z-2}}), a_{z-3}.\overline{b_{z-3}}) \tag{4}$$

$$\overline{B_{big}A_{(z-1:z-3)}} = M(M(a_{z-1}, \overline{b_{z-1}}, a_{z-2}), M(a_{z-1}, \overline{b_{z-1}}, \overline{b_{z-2}}), a_{z-3} + \overline{b_{z-3}}) \tag{5}$$

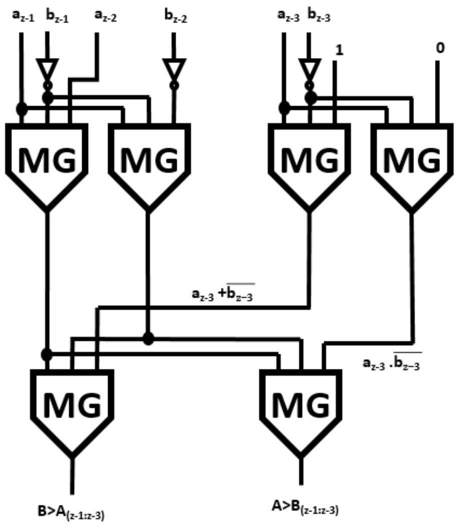


FIGURE 8. Theorem 2(T2).

Theorem 3: Given two n-bit numbers $A_{(n-1:0)}$ and $B_{(n-1:0)}$, $A_{big}B_{(n-1:0)}$ is 1 if and only if $A_{(n-1:0)} > B_{(n-1:0)}$, and $\overline{B_{big}A_{(n-1:0)}}$ results in 0 if and only if $A_{(n-1:0)} < B_{(n-1:0)}$. Theorem 3 (T3) is depicted in figure 9 on page 12.

$$A_{big}B_{(n-1:0)} = M(M(a_{n-1}, \overline{b_{n-1}}, a_{n-2}), M(a_{n-1}, \overline{b_{n-1}}, \overline{b_{n-2}}), A_{big}B_{(n-3:0)}) \tag{6}$$

$$\overline{B_{big}A_{(n-1:0)}} = M(M(a_{n-1}, \overline{b_{n-1}}, a_{n-2}), M(a_{n-1}, \overline{b_{n-1}}, \overline{b_{n-2}}), \overline{B_{big}A_{(n-3:0)}}) \tag{7}$$

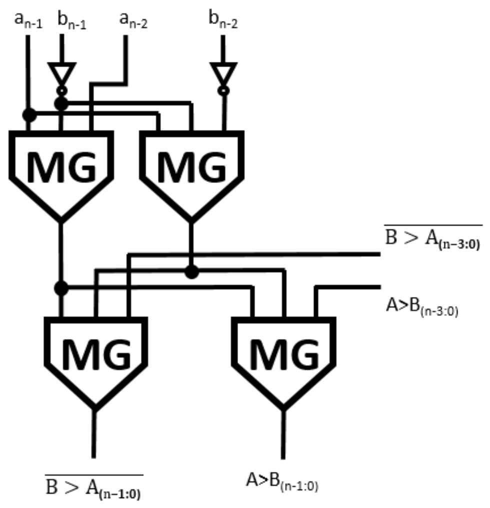
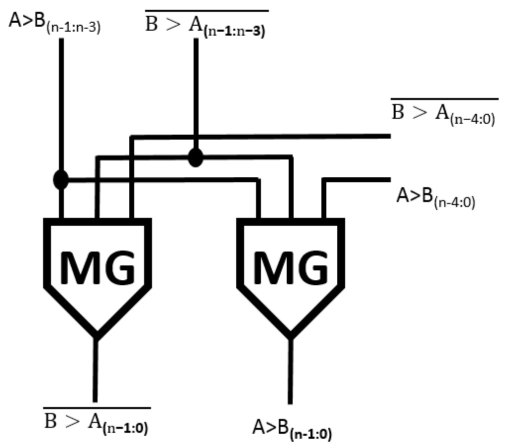


FIGURE 9. Theorem 3(T3).

Theorem 4: If $A_{(n-1:0)}$ and $B_{(n-1:0)}$ are two n-bit numbers, $A_{big}B_{(n-1:n-3)}$ and $\overline{B_{big}A_{(n-1:n-3)}}$ being computed by (4) and (5), respectively, then $A_{big}B_{(n-1:0)}$ as defined in (8) is equal to 1 if and only if $A_{(n-1:0)} > B_{(n-1:0)}$, whereas $\overline{B_{big}A_{(n-1:0)}}$ as defined in (9) is equal to 0 if and only if $A_{(n-1:0)} < B_{(n-1:0)}$. Theorem 4 (T4) is depicted in 10 figure below.

$$A_{big}B_{(n-1:0)} = M(A_{big}B_{(n-1:n-3)}, \overline{B_{big}A_{(n-1:n-3)}}, A_{big}B_{(n-4:0)}) \quad (8)$$

$$\overline{B_{big}A_{(n-1:0)}} = M(A_{big}B_{(n-1:n-3)}, \overline{B_{big}A_{(n-1:n-3)}}, \overline{B_{big}A_{(n-4:0)}}) \quad (9)$$



$A>B_{(n-1:n-3)}$
$\overline{B > A_{(n-1:n-3)}}$
$\overline{B > A_{(n-4:0)}}$
$A>B_{(n-4:0)}$
MG
MG
$\overline{B > A_{(n-1:0)}}$
$A>B_{(n-1:0)}$

FIGURE 10. Theorem 4(T4).

Corollary 1: Let's consider two n-bit numbers $A_{(n-1:0)}$and $B_{(n-1:0)}$, and let's suppose that they are split into the subwords $A_{(n-1:h)}$, $A_{(h-1:0)}$, $B_{(n-1:h)}$ and $B_{(h-1:0)}$. If 0 are computed by applying Theorems 3 and 4, then $A_{big}B_{(n-1:0)}$ as defined in (10) is equal to 1 if and only if $A_{(n-1:0)} > B_{(n-1:0)}$, whereas $B_{big}A_{(n-1:0)}$ as defined in (11) is equal to 0 if and only if $A_{(n-1:0)} < B_{(n-1:0)}$. Corollary 1 (C1) is depicted in figure 11 below.

$$A_{big}B_{(n-1:0)} = M(A_{big}B_{(n-1:h)}, \overline{B_{big}A_{(n-1:h)}}, A_{big}B_{(h-1:0)}) \qquad (10)$$

$$\overline{B_{big}A_{(n-1:0)}} = M(A_{big}B_{(n-1:h)}, \overline{B_{big}A_{(n-1:h)}}, \overline{B_{big}A_{(h-1:0)}}) \qquad (11)$$

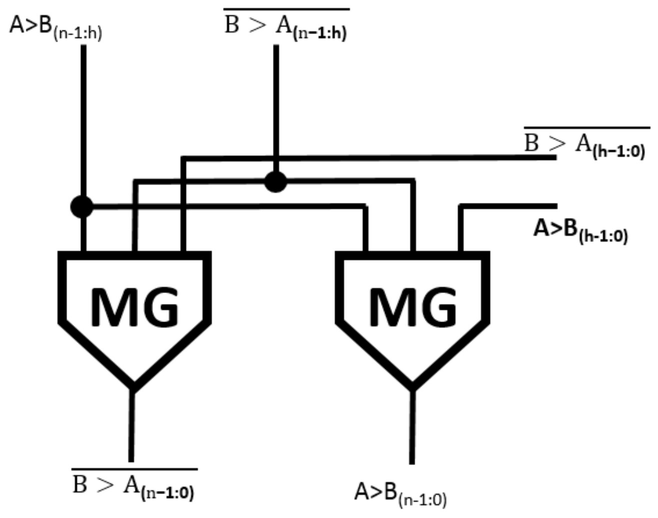

A>B(n-1:h)
$\overline{B > A_{(n-1:h)}}$
$\overline{B > A_{(h-1:0)}}$
A>B(h-1:0)
MG
MG
$\overline{B > A_{(n-1:0)}}$
A>B(n-1:0)

FIGURE 11. Corollary 1(C1).

Corollary 2: Given two n-bit numbers $A_{(n-1:0)}$ and $B_{(n-1:0)}$, if $A_{big}B_{(n-1:0)}$ and $B_{big}A_{(n-1:0)}$ are computed by applying Theorems 1, 2, 3, and 4 and/or Corollary 1, then $A_{eq}B_{(n-1:0)}$ defined in (12) is equal to 1 if and only if $A_{(n-1:0)} = B_{(n-1:0)}$. Corollary 2 (C2) is depicted in figure 12 on page 14.

$$A_{eq}B_{(n-1:0)} = M(\overline{A_{big}B_{(n-1:0)}}, \overline{B_{big}A_{(n-1:0)}}, 0) \qquad (12)$$

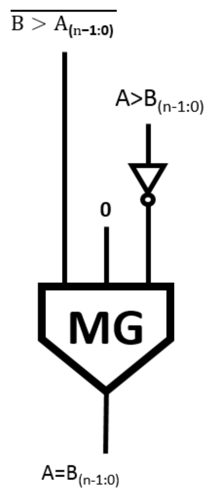


FIGURE 12. Corollary 2(C2).

Designing Binary Comparators Exploiting the New Theorems

The logical inventory shown in Fig. 5(a), (b), (c) and (d) depict the circuits based on above Theorems 1, 2, 3 and 4 respectively. The same way, Fig. 5(e) and (f) describe the implementation of Corollaries 1 and 2 respectively. We would thus refer these circuits as generic modules naming T1, T2, T3 and T4 that represent the four theorems; and C1 and C2 which interpret the two corollaries.

Many structures can be designed by combining the basic modules in different manners. In this project, we will discuss the application of the above QCA modules to design a 16-bit and a 64-bit cascade-based architecture of full comparator and its advantages over a conventional 64-bit high-speed CMOS comparator.

16-Bit Cascade-Based QCA Comparators

The simplest form of comparator that comes to one's mind exploits the arrangement of a cascade-based (CB) achitecture. The CB full comparator has been designed for operands' word lengths ranging from 2 to 64 and using the split criterion for n > 2 summarized in Table 2 for only one operand. The same input bits are applied to the other operand. Obviously, alternative splits could be used.

TABLE 2. Splitting Criterion Adopted in the CB Comparators

n	Splitting of an operand
4	$A_{(3:2)}A_{(1:0)}$
8	$A_{(7:5)}A_{(4:2)}A_{(1:0)}$
16	$A_{(15:14)}A_{(13:11)}A_{(10:8)}A_{(7:5)}A_{(4:2)}A_{(1:0)}$
32	$A_{(31:29)}A_{(28:26)}A_{(25:23)}A_{(22:20)}A_{(19:17)}A_{(16:14)}A_{(13:11)}A_{(10:8)}A_{(7:5)}A_{(4:2)}A_{(1:0)}$
64	$A_{(63:62)}A_{(61:59)}A_{(58:56)}A_{(55:53)}A_{(52:50)}A_{(49:47)}A_{(46:44)}A_{(43:41)}A_{(40:38)}A_{(37:35)}$ $A_{(34:32)}A_{(31:29)}A_{(28:26)}A_{(25:23)}A_{(22:20)}A_{(19:17)}A_{(16:14)}A_{(13:11)}A_{(10:8)}A_{(7:5)}A_{(4:2)}A_{(1:0)}$

Figure 13 on page 16 illustrates a schematic diagram that explains the overall computation of a 16-bit comparator. In this architecture, for an n-bit full comparator uses:

i. n/3 instances of T1 and/or T2,

ii. n/3 cascaded instances of T4 needed to compute $A_{big}B_{(n-1:0)}$ and $\overline{B_{big}A_{(n-1:0)}}$, and

iii. one instance of C2 to determine $A_{eq}B_{(n-1:0)}$.

The circles depicted in figure 13, represent the additional clock phases to ensure proper synchronisation of the whole design.

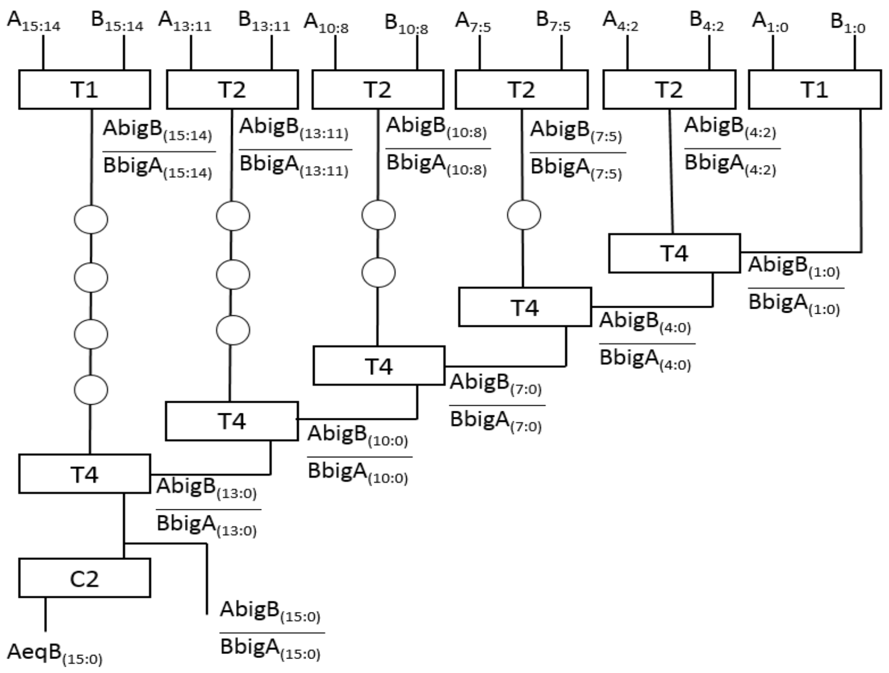


FIGURE 13. Novel 16-bit CB full comparator.

As a matter of fact, the number of cascaded MGs within the worst computational path directly affects the time delay in the QCA design. Each MG brings in one clock phase in overall delay. Going into the roots, it is noticed that modules T1 and T2 contribute with one inverter and two MGs. Similarly, T4 introduces one more MG; and C2 add up to one MG and an inverter. As a result, the critical computational path of the novel n-bit full comparator comprises 3 + (n/3) MGs and 2 inverters.   As an example, the 16-bit implementation depicted in figure 13 has the worst-case path made up of 8 MGs and 2 inverters.

To analyze the performance, the design has been simulated in the softwares called ModelSim III 6.4b and Xilinx ISE Design Suite 14.7 in the next section.

64-Bit Cascade Based QCA Comparator

A 64 bit cascade design can process larger amount of data within short time as it can take more input data than the 16 or 32 bit ones. Figure 14 depicts the schematic design of a 64- bit comparator.

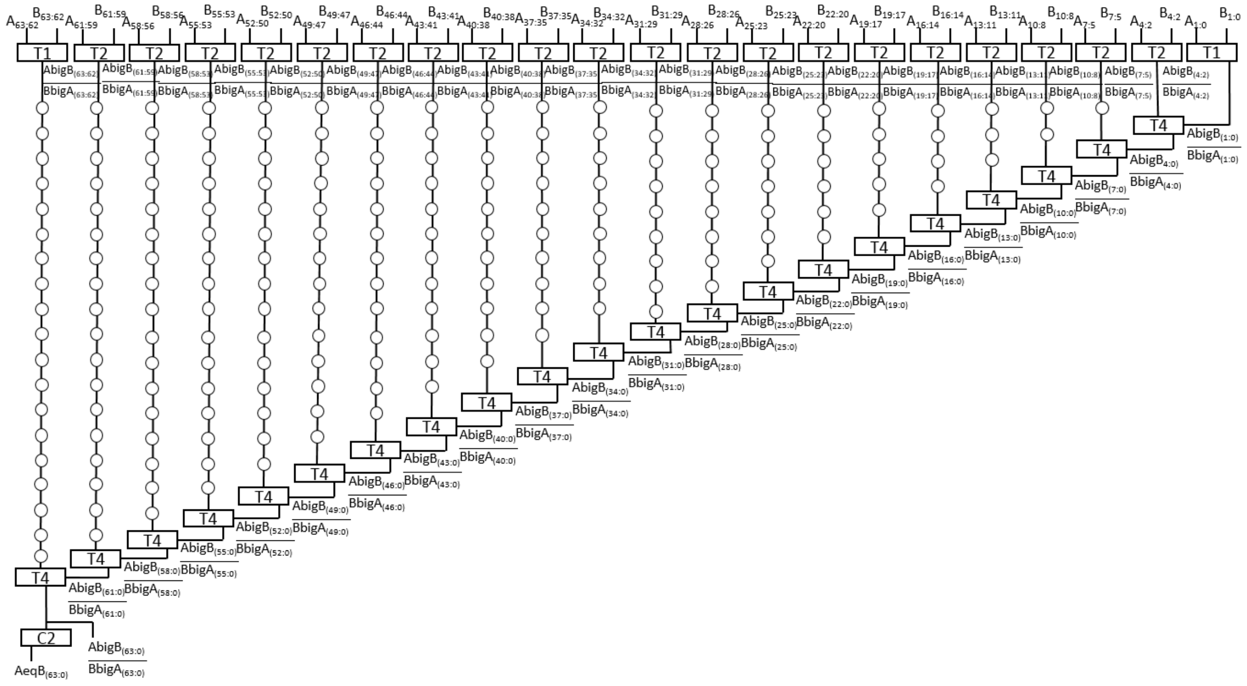


FIGURE 14. Novel 64-bit CB full comparator.

CHAPTER 4

SIMULATION RESULTS

The block diagram(figure 15), RTL Schematic(figure 16) and the Technology Schematic(figure 17) have been depicted for a 16-bit binary comparator. As seen from the block diagram, A and B are two 16-bit operands and AbigB, AeqB and BbigA are three possible outcomes whose logic will be true when A > B, A = B and A < B respectively.

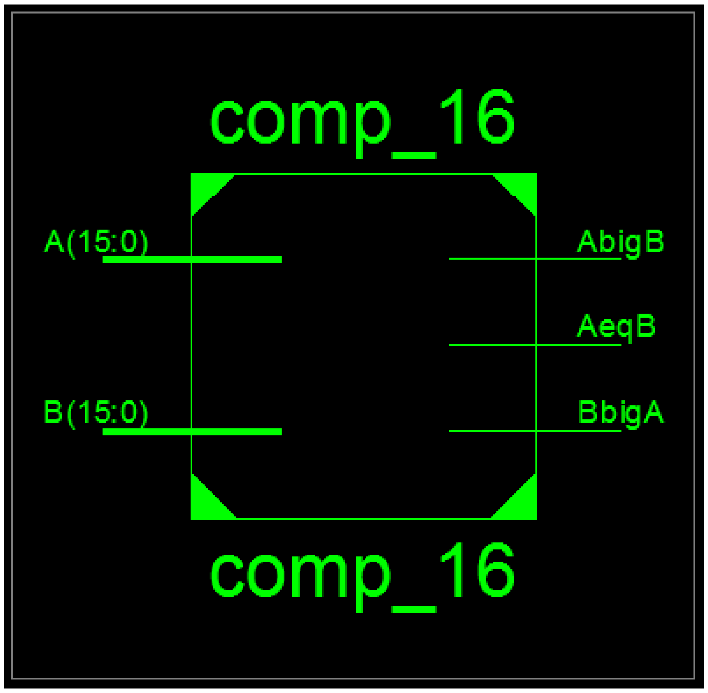


FIGURE 15. Block diagram for 16-bit binary comparator.

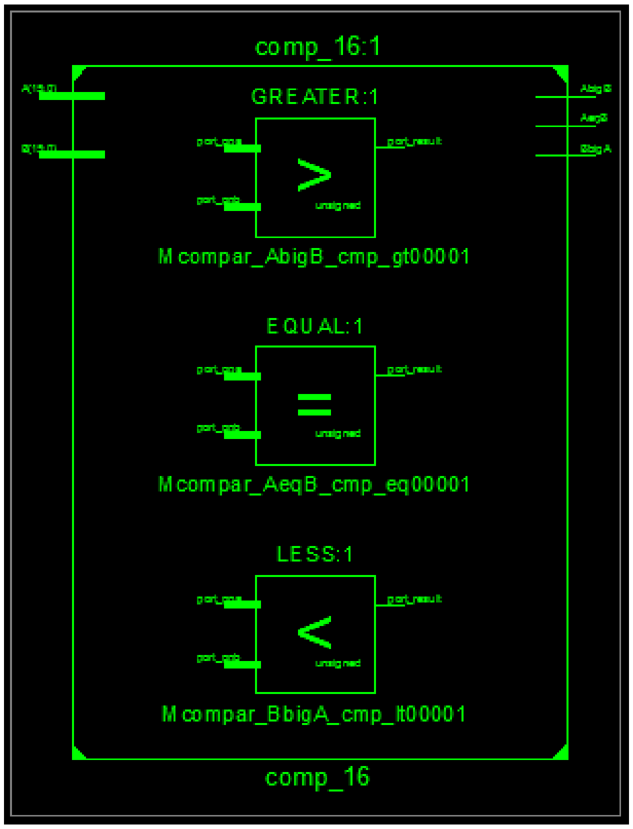


FIGURE 16. RTL schematic for 16-bit binary comparator.

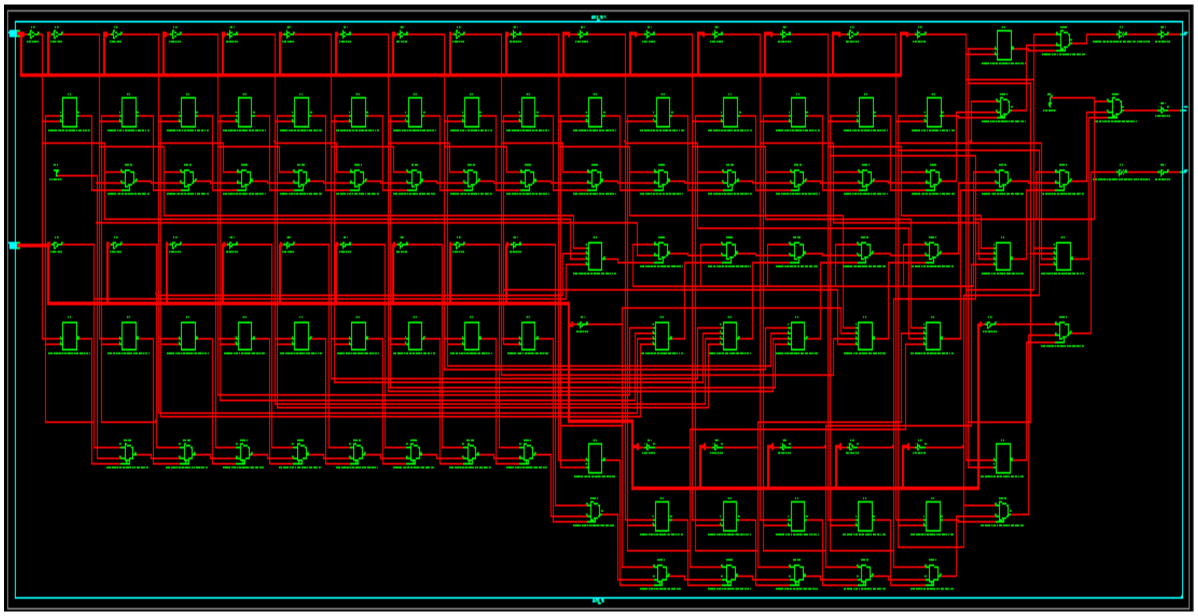
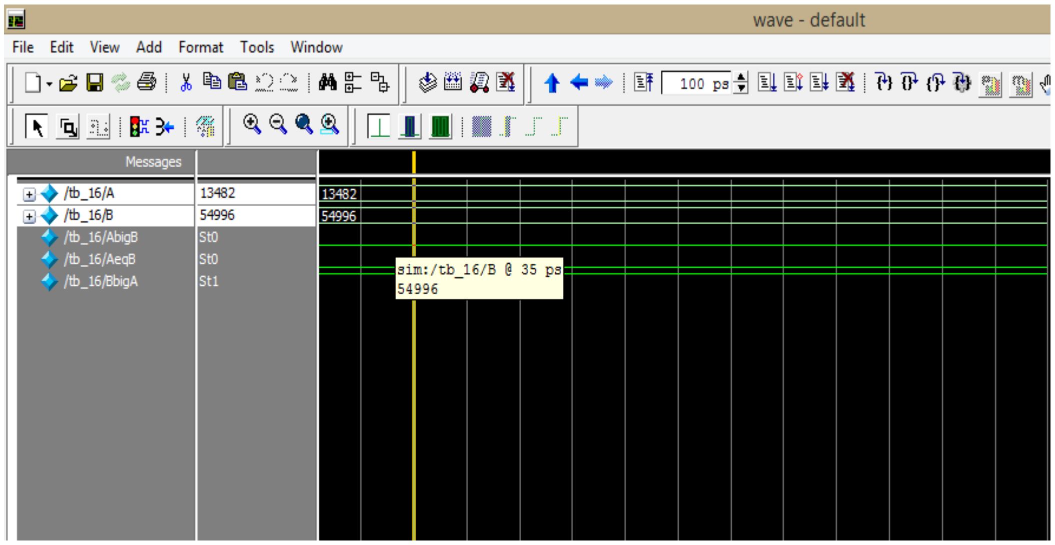


FIGURE 17. Technology schematic for 16-bit binary comparator.

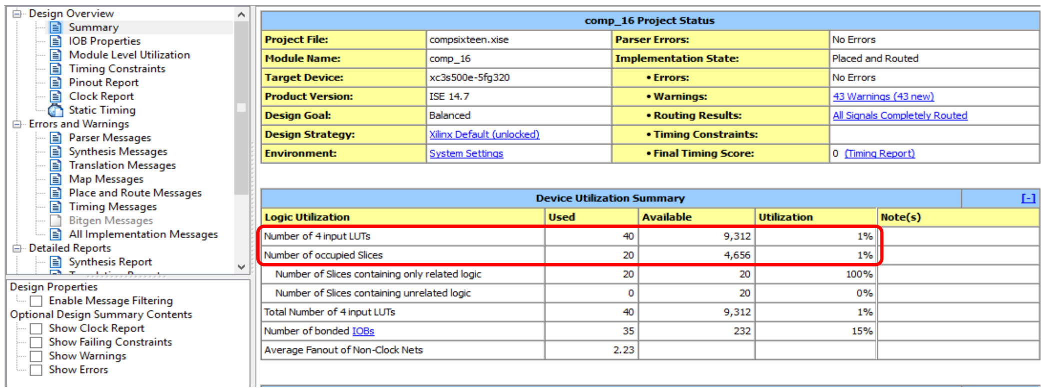
Figure 17 on page 19 depicts the technology schematic for 16-bit binary comparator. Moving on to simulating the working of the comparators, A was applied with hexadecimal 34AA (decimal equivalent: 13482) and B with D6D4 hexadecimal (decimal equivalent: 54996). As we see, 54996 is greater 13482 (B > A); the output result is AbigB = 0, AeqB = 0 and BbigA = 1 (see figure 18).



```
/tb_16/A      13482
/tb_16/B      54996
/tb_16/AbigB  St0
/tb_16/AeqB   St0
/tb_16/BbigA  St1
```

```
sim:/tb_16/B @ 35 ps
54996
```

FIGURE 18. Simulation results for 16-bit binary comparator.



comp_16 Project Status			
Project File:	compsixteen.xise	**Parser Errors:**	No Errors
Module Name:	comp_16	**Implementation State:**	Placed and Routed
Target Device:	xc3s500e-5fg320	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	43 Warnings (43 new)
Design Goal:	Balanced	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	40	9,312	1%	
Number of occupied Slices	20	4,656	1%	
Number of Slices containing only related logic	20	20	100%	
Number of Slices containing unrelated logic	0	20	0%	
Total Number of 4 input LUTs	40	9,312	1%	
Number of bonded IOBs	35	232	15%	
Average Fanout of Non-Clock Nets	2.23			

FIGURE 19. Synthesis results for 16-bit binary comparator.

Figure 19 shows the synthesis results for 16-bit binary comparator. We can see that we have used only 40 out of 9312 4-input LUTs. The number of occupied slices are only 20 out of 4656 slices.

The block diagram(figure 20), RTL Schematic(figure 21) and the Technology Schematic(figure 12) have been depicted for a 64-bit binary comparator. As discussed in the case of 16-bit comparator, here too, there are two inputs operands A and B, and three possible outcomes. The difference is that the input operands are 64-bit long.

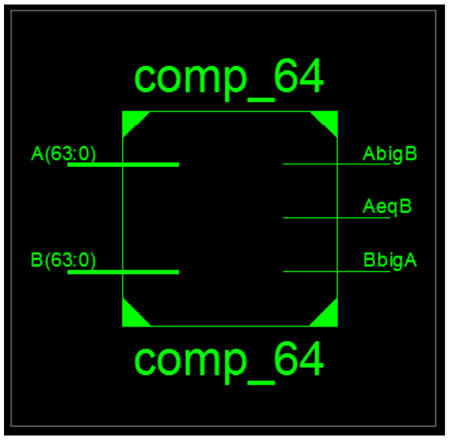


FIGURE 20. Block diagram for 64-bit binary comparator.

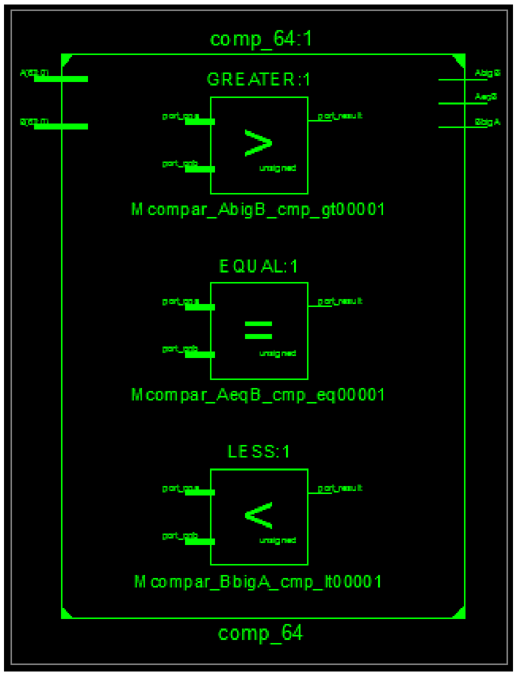


FIGURE 21. RTL schematic for 64-bit binary comparator.

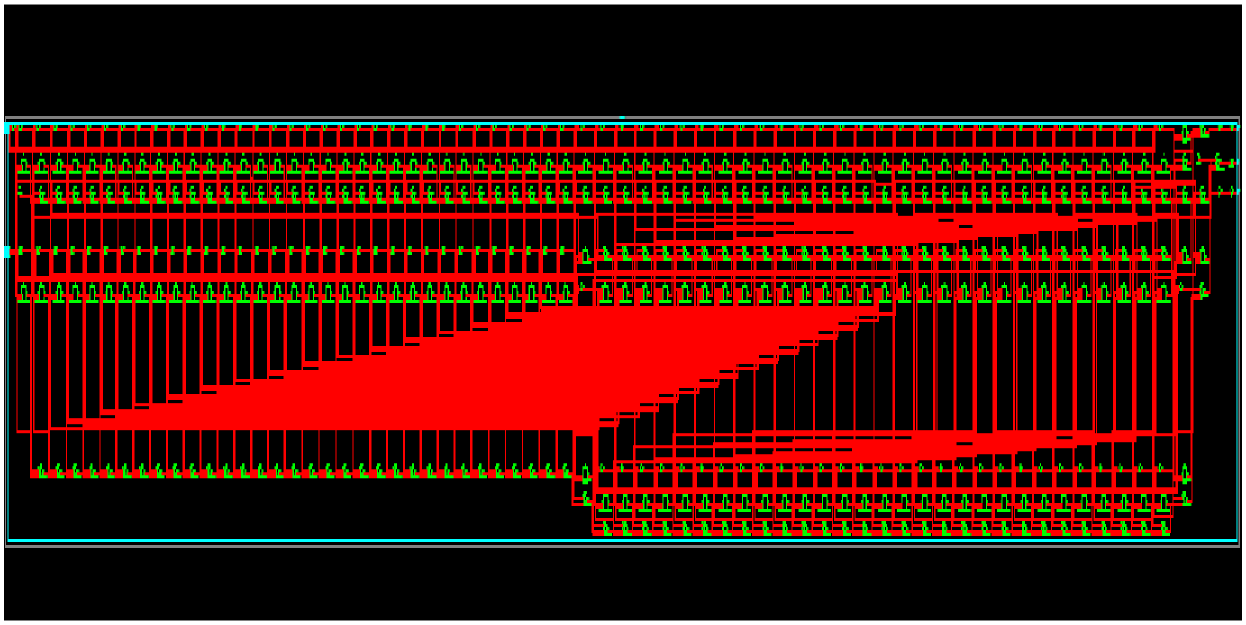


FIGURE 22. Technology schematic for 64-bit binary comparator.

Figure 22 depicts the technology schematic for 64-bit binary comparator. Simulation tests were performed on a 64-bit comparator as well. As an example, A was applied with hexadecimal ‘eeee eeee eeee eeee’ (decimal equivalent: 17216961135462248174) and B with hexadecimal ‘dddd dddd dddd dddd’ (decimal equiavalent: 15987178197214944733). It can be seen here that 17216961135462248174 > 15987178197214944733 (A > B); thus the result output shows up to be AbigB = 1, AeqB = 0 and BbigA = 0. After 200 ps, another set of inputs were applied to A and B. The inputs this time provided were ‘aaaa aaaa aaaa aaaa’ (decimal equivalent: 12297829382473034410) to both the operands. Since the same value was applied to both the terminals (A = B), the result was AbigB = 0, AeqB = 1 and BbigA = 0 (See figure 23).

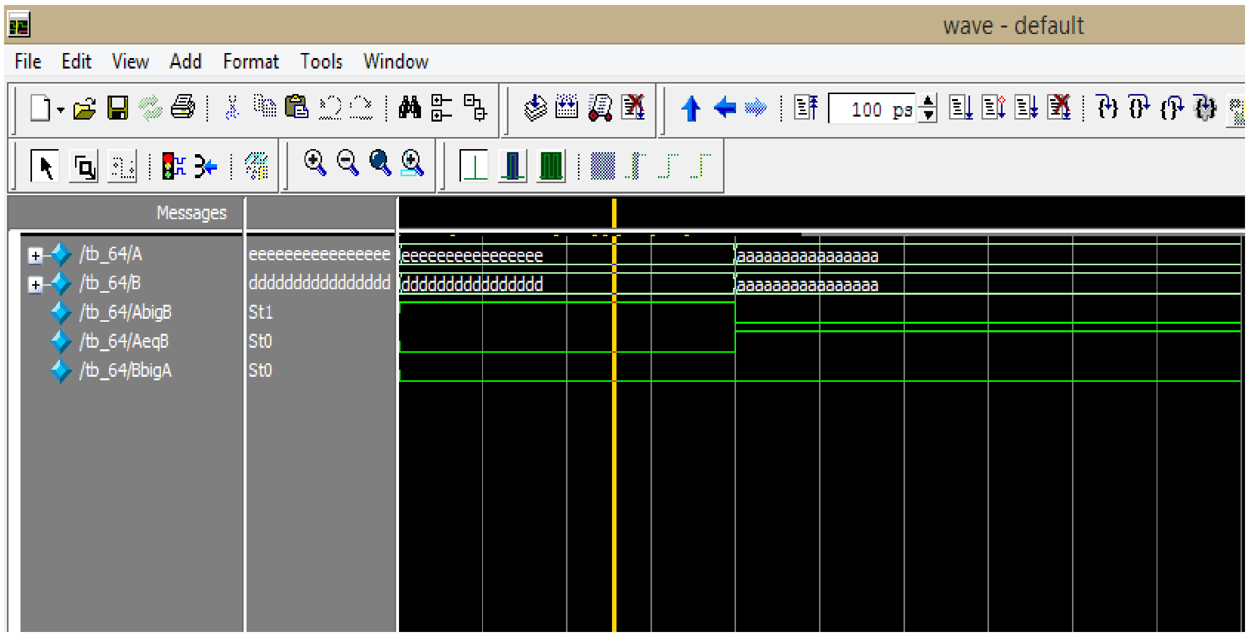
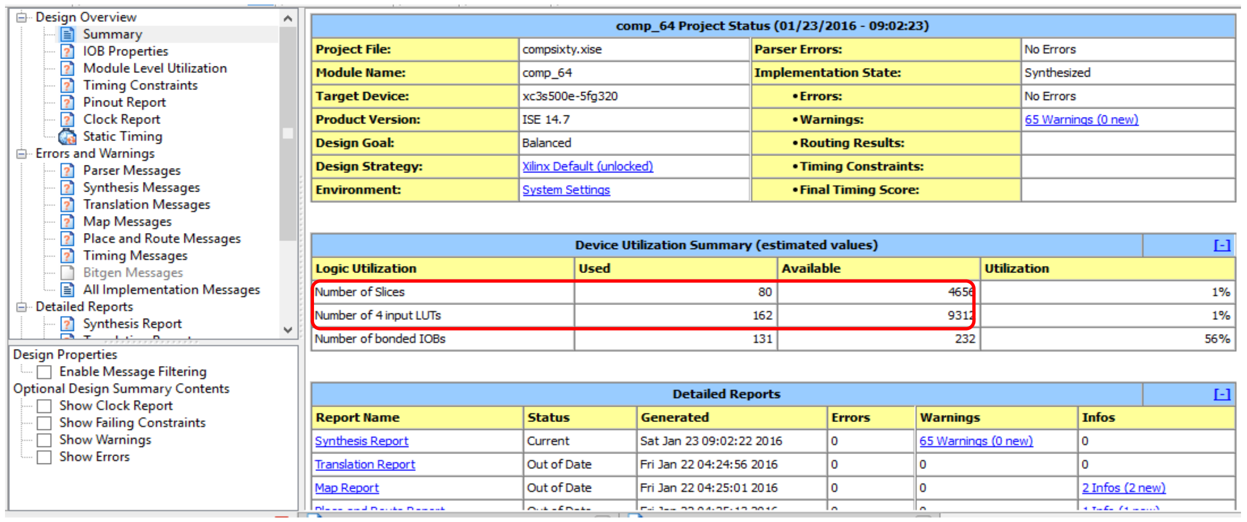


FIGURE 23. Simulation results for 64-bit binary comparator.



comp_64 Project Status (01/23/2016 - 09:02:23)			
Project File:	compsixty.xise	**Parser Errors:**	No Errors
Module Name:	comp_64	**Implementation State:**	Synthesized
Target Device:	xc3s500e-5fg320	**•Errors:**	No Errors
Product Version:	ISE 14.7	**•Warnings:**	65 Warnings (0 new)
Design Goal:	Balanced	**•Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	
Environment:	System Settings	**•Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	80	4656	1%
Number of 4 input LUTs	162	9312	1%
Number of bonded IOBs	131	232	56%

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sat Jan 23 09:02:22 2016	0	65 Warnings (0 new)	0
Translation Report	Out of Date	Fri Jan 22 04:24:56 2016	0	0	0
Map Report	Out of Date	Fri Jan 22 04:25:01 2016	0	0	2 Infos (2 new)

FIGURE 24. Synthesis results for 64-bit binary comparator.

Figure 24 shows the synthesis results for 64-bit binary comparator. We can see that we have used only 162 out of 9312 4-input LUTs. The number of occupied slices are only 80 out of 4656 slices.

Table 3 reflects various aspects of n-bit comparators.

TABLE 3. Implementation Results for QCA Comparator

N	Total Simulation Delay(ns)	Area (μm^2)	Number of cells
4*	6.572	0.41	297
8*	7.485	0.79	699
16	8.341	1.5	1392
32*	9.649	2.66	2783
64	10.813	3.78	3574

*[11]

We will now compare these different aspects on a conventional CMOS Comparator (table 4). Various designs have been developed over the years in this field. We will consider a High Speed design of a 100nm monolithic device.

TABLE 4. Conventional CMOS Comparator Results [14]

n-bit	Simulation Delay (ns)	Area (μm^2)
4	116	2.43
8	175	5.23
16	220	11.43
32	300	25.07
64	462	54.47

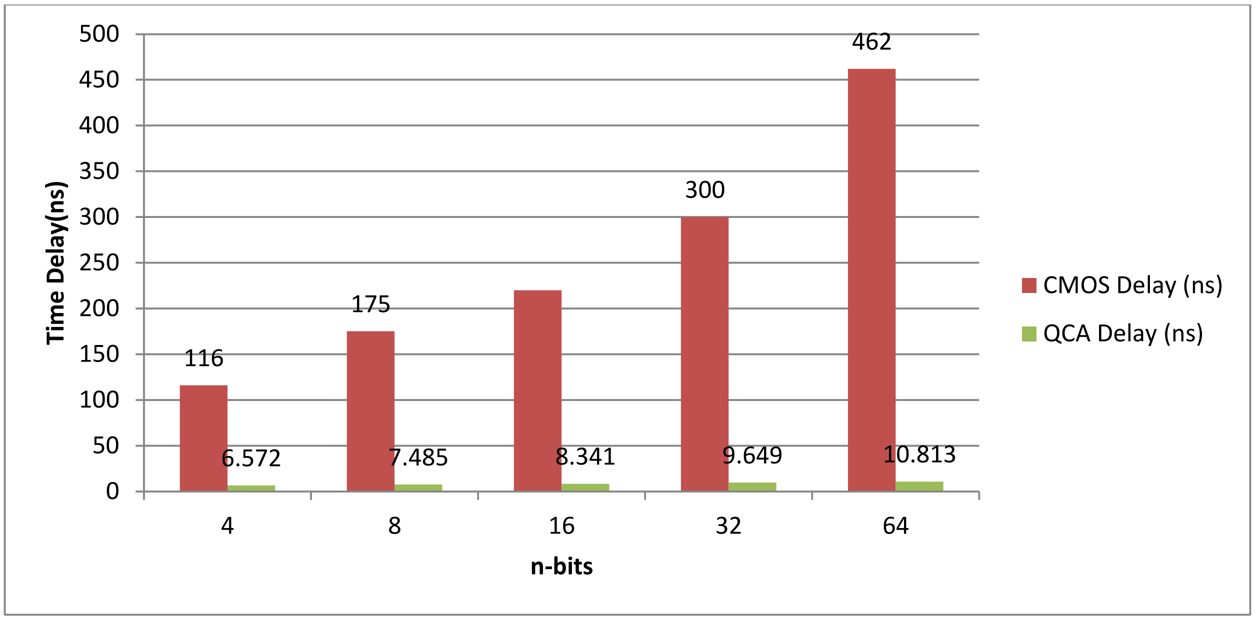


FIGURE 25. Time delay response for various architectures.

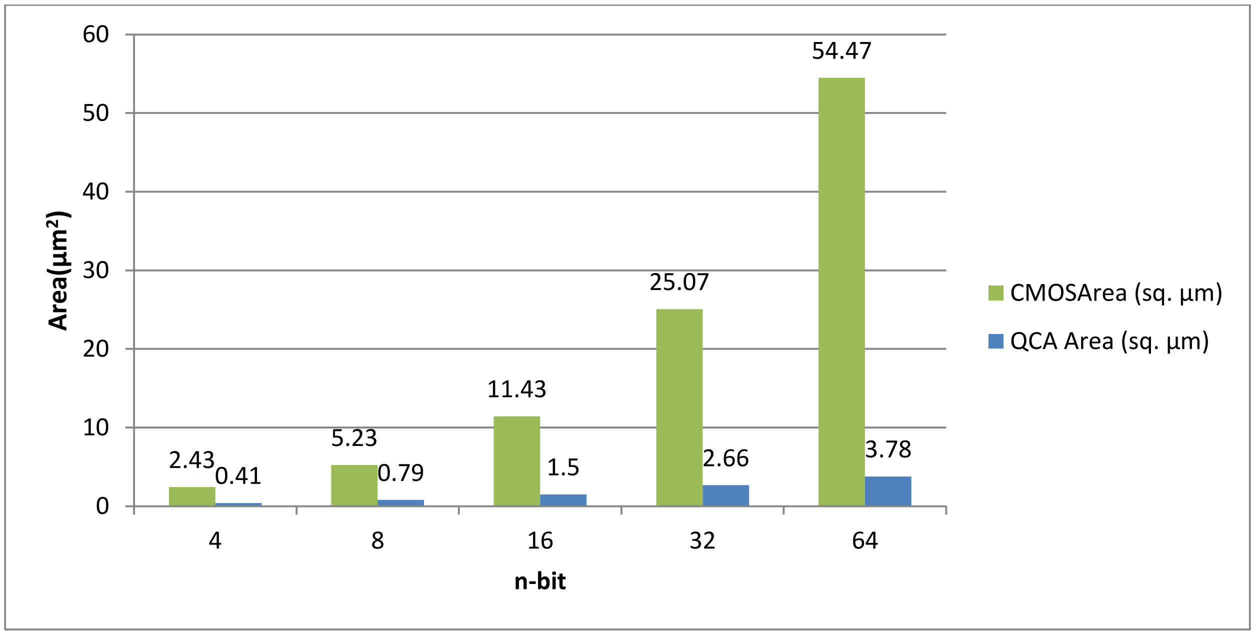


FIGURE 26. Area consumed for various bit architectures.

We can notice from the graphs represented in figures 25 and 26 on page 27, there is a considerable amount of difference in the delay time and also the area required to fulfill the design. For example, considering the 64-bit comparator in QCA architecture, the total delay in obtaining the output response is about 11ns, which is 40 times faster than the CMOS design of a 64-bit comparator. The same applies to the area consumed by the QCA comparator that is around 20 times smaller than a CMOS comparator.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

We have presented a novel QCA cascade-based 64-bit binary comparator in this project. It can be noted that by exploiting various concepts and theorems introduced in this document, a novel comparator could be designed with lower complexity and still accomplish high speed and large data processing than its rivals. Results obtained demonstrate that this architecture has 40 times faster response and can accommodate more than 15 logic circuits in the same area than those of existing competitors. The innovative formulations allow increased speed performances and reduced overall sizes achieved with respect to existing competitors. In future prospect, 128–bit or higher bit design of comparator can be designed using this technology. Also 3-input Majority Gates can be upgraded and implemented as 5-input Majority Gates in the circuit.

APPENDIX

CODE

XOR

```
module and_xor(a, b, p, g);

 //very first inputs - and/xor

 input a, b;

 output p, g;


 xor(p, a, b);

 and(g, a, b);


endmodule
```

2-input OR

```
module M_cell(Gkj, Pik, Gik, G);

 //M cell

 input Gkj, Pik, Gik;

 output G;

 wire Y;


 and(Y, Gkj, Pik);

 or(G, Y, Gik);


endmodule
```

2-input AND

```
module M11_cell(Gkj, Pik, Gik, Pkj, G, P);
 //M11 cell
 input Gkj, Pik, Gik, Pkj;
 output G, P;
 wire Y;

 and(Y, Gkj, Pik);
 or(G, Gik, Y);
 and(P, Pkj, Pik);

endmodule
```

16-bit comparator:

```
module comp_16(A, B,AbigB,AeqB,BbigA);
 input [15:0] A, B;
 output AbigB,AeqB,BbigA;

wire [7:0]  G_Z, P_Z,G_A, P_A,G_B, P_B,G_C, P_C;


 M_cell level_0A(1'b0, P_Z[0], G_Z[0], G_A[0]);
 M11_cell level_1A(G_Z[0], P_Z[1], G_Z[1], P_Z[0], G_A[1],
P_A[1]);
 M11_cell level_2A(G_Z[1], P_Z[2], G_Z[2], P_Z[1], G_A[2],
P_A[2]);
 M11_cell level_3A(G_Z[2], P_Z[3], G_Z[3], P_Z[2], G_A[3],
P_A[3]);
 M11_cell level_4A(G_Z[3], P_Z[4], G_Z[4], P_Z[3], G_A[4],
P_A[4]);
 M11_cell level_5A(G_Z[4], P_Z[5], G_Z[5], P_Z[4], G_A[5],
P_A[5]);
 M11_cell level_6A(G_Z[5], P_Z[6], G_Z[6], P_Z[5], G_A[6],
P_A[6]);
 M11_cell level_7A(G_Z[6], P_Z[7], G_Z[7], P_Z[6], G_A[7],
P_A[7]);


 M_cell level_1B(cin, P_A[1], G_A[1], G_B[1]);
 M_cell level_2B(G_A[0], P_A[2], G_A[2], G_B[2]);
 M11_cell level_3B(G_A[1], P_A[3], G_A[3], P_A[1], G_B[3],
P_B[3]);
 M11_cell level_4B(G_A[2], P_A[4], G_A[4], P_A[2], G_B[4],
P_B[4]);
```

```
 M11_cell level_5B(G_A[3], P_A[5], G_A[5], P_A[3], G_B[5],
P_B[5]);
 M11_cell level_6B(G_A[4], P_A[6], G_A[6], P_A[4], G_B[6],
P_B[6]);
 M11_cell level_7B(G_A[5], P_A[7], G_A[7], P_A[5], G_B[7],
abigb);


 M_cell level_3C(1'b0, P_B[3], G_B[3], G_C[3]);
 M_cell level_4C(G_A[0], P_B[4], G_B[4], G_C[4]);
 M_cell level_5C(G_B[1], P_B[5], G_B[5], G_C[5]);
 M_cell level_6C(G_B[2], P_B[6], G_B[6], G_C[6]);
 M11_cell level_7C(G_B[3], P_B[7], G_B[7], P_B[3], G_C[7],
bbiga);


 M_cell level_7D(1'b0, P_C[7], G_C[7],aeqb);


assign AbigB=(A>B)?1'b1:1'b0;

assign AeqB=(A==B)?1'b1:1'b0;assign BbigA=(A<B)?1'b1:1'b0;
```

```
 and_xor level_Z0(A[0], B[0], P_Z[0], G_Z[0]);
 and_xor level_Z1(A[1], B[1], P_Z[1], G_Z[1]);
 and_xor level_Z2(A[2], B[2], P_Z[2], G_Z[2]);
 and_xor level_Z3(A[3], B[3], P_Z[3], G_Z[3]);
 and_xor level_Z4(A[4], B[4], P_Z[4], G_Z[4]);
 and_xor level_Z5(A[5], B[5], P_Z[5], G_Z[5]);
 and_xor level_Z6(A[6], B[6], P_Z[6], G_Z[6]);
 and_xor level_Z7(A[7], B[7], P_Z[7], G_Z[7]);
 and_xor level_Z8(A[8], B[8], P_C[7], P_B[7]);
endmodule
```

16-bit comparator test-bench:

```
module tb_16();
  reg [15:0] A, B;
 wire AbigB,AeqB,BbigA;

 comp_16 m1(A,B,AbigB,AeqB,BbigA);

 initial
 begin
   A=16'h0017;
   B=16'h0012;
   #200
   A=16'h1234;
   B=16'h1237;
 end


 endmodule
```

64-bit comparator:

```
module comp_64(A, B,AbigB,AeqB,BbigA);
 input [63:0] A, B;
 output AbigB,AeqB,BbigA;

wire [7:0]  G_B, P_B,G_C, P_C;
wire [15:0]  G_Z,P_Z,G_A, P_A;



 M_cell level_0A(1'b0, P_Z[0], G_Z[0], G_A[0]);
 M11_cell level_1A(G_Z[0], P_Z[1], G_Z[1], P_Z[0], G_A[1],
P_A[1]);
 M11_cell level_2A(G_Z[1], P_Z[2], G_Z[2], P_Z[1], G_A[2],
P_A[2]);
 M11_cell level_3A(G_Z[2], P_Z[3], G_Z[3], P_Z[2], G_A[3],
P_A[3]);
 M11_cell level_4A(G_Z[3], P_Z[4], G_Z[4], P_Z[3], G_A[4],
P_A[4]);
 M11_cell level_5A(G_Z[4], P_Z[5], G_Z[5], P_Z[4], G_A[5],
P_A[5]);
 M11_cell level_6A(G_Z[5], P_Z[6], G_Z[6], P_Z[5], G_A[6],
P_A[6]);
 M11_cell level_7A(G_Z[6], P_Z[7], G_Z[7], P_Z[6], G_A[7],
P_A[7]);
  M11_cell level_8A(G_Z[7], P_Z[8], G_Z[8], P_Z[7], G_A[8],
P_A[8]);
 M11_cell level_9A(G_Z[8], P_Z[9], G_Z[9], P_Z[8], G_A[9],
P_A[9]);
 M11_cell level_10A(G_Z[9], P_Z[10], G_Z[10], P_Z[9], G_A[10],
P_A[10]);
 M11_cell level_11A(G_Z[10], P_Z[11], G_Z[11], P_Z[10], G_A[11],
P_A[11]);
```

```
 M11_cell level_12A(G_Z[11], P_Z[12], G_Z[12], P_Z[11], G_A[12],
P_A[12]);
 M11_cell level_13A(G_Z[12], P_Z[13], G_Z[13], P_Z[12], G_A[13],
P_A[13]);
 M11_cell level_14A(G_Z[13], P_Z[14], G_Z[14], P_Z[13], G_A[14],
P_A[14]);


 M_cell level_1B(cin, P_A[1], G_A[1], G_B[1]);
 M_cell level_2B(G_A[0], P_A[2], G_A[2], G_B[2]);
 M11_cell level_3B(G_A[1], P_A[3], G_A[3], P_A[1], G_B[3],
P_B[3]);
 M11_cell level_4B(G_A[2], P_A[4], G_A[4], P_A[2], G_B[4],
P_B[4]);
 M11_cell level_5B(G_A[3], P_A[5], G_A[5], P_A[3], G_B[5],
P_B[5]);
 M11_cell level_6B(G_A[4], P_A[6], G_A[6], P_A[4], G_B[6],
P_B[6]);
 M11_cell level_7B(G_A[5], P_A[7], G_A[7], P_A[5], G_B[7],
abigb);

 M_cell level_3C(1'b0, P_B[3], G_B[3], G_C[3]);
 M_cell level_4C(G_A[0], P_B[4], G_B[4], G_C[4]);
 M_cell level_5C(G_B[1], P_B[5], G_B[5], G_C[5]);

 M_cell level_6C(G_B[2], P_B[6], G_B[6], G_C[6]);

 M11_cell level_7C(G_B[3], P_B[7], G_B[7], P_B[3], G_C[7],
bbiga);


 M_cell level_7D(1'b0, P_C[7], G_C[7],aeqb);


assign AbigB=(A>B)?1'b1:1'b0;


assign AeqB=(A==B)?1'b1:1'b0;assign BbigA=(A<B)?1'b1:1'b0;

 and_xor level_Z0(A[0], B[0], P_Z[0], G_Z[0]);

 and_xor level_Z1(A[1], B[1], P_Z[1], G_Z[1]);

 and_xor level_Z2(A[2], B[2], P_Z[2], G_Z[2]);

 and_xor level_Z3(A[3], B[3], P_Z[3], G_Z[3]);

 and_xor level_Z4(A[4], B[4], P_Z[4], G_Z[4]);

 and_xor level_Z5(A[5], B[5], P_Z[5], G_Z[5]);

 and_xor level_Z6(A[6], B[6], P_Z[6], G_Z[6]);

 and_xor level_Z7(A[7], B[7], P_Z[7], G_Z[7]);

 and_xor level_Z8(A[7], B[7], P_C[7], P_B[7]);

 and_xor level_Z9(A[8], B[8], P_Z[8], G_Z[8]);

 and_xor level_Z10(A[9], B[9], P_Z[9], G_Z[9]);

 and_xor level_Z11(A[10], B[10], P_Z[10], G_Z[10]);

 and_xor level_Z12(A[11], B[11], P_Z[11], G_Z[11]);
 and_xor level_Z13(A[12], B[12], P_Z[12], G_Z[12]);
 and_xor level_Z14(A[13], B[13], P_Z[13], G_Z[13]);
 and_xor level_Z15(A[14], B[14], P_Z[14], G_Z[14]);
 and_xor level_Z16(A[15], B[15], P_Z[15], G_Z[15]);
endmodule
```

64-bit comparator test-bench:

```
module tb_64();
  reg [63:0] A, B;
 wire AbigB,AeqB,BbigA;
 comp_64 m1(A,B,AbigB,AeqB,BbigA);
 initial
 begin
   A=64'heeeeeeeeeeeeeeee;
   B=64'hdddddddddddddddd;
   #200
   A=64'haaaaaaaaaaaaaaaa;
   B=64'haaaaaaaaaaaaaaaa;
 end
 endmodule
```

REFERENCES

REFERENCE

[1] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernestein, “Quantum cellular automata,”*Nanotechnology*, vol. 4, no. 1, pp. 49–57, 1993.

[2] G. H. Bernstein, A. Imre, V. Metlushko, A. Orlov, L. Zhou, L. Ji, G. Csaba,and W. Porod, “Magnetic QCA systems,” *Microelectron*. J., vol. 36,pp. 619–624, 2005.

[3] J. Huang and F. Lombardi, *Design and Test of Digital Circuits by Quantum Dot Cellular Automata*. Norwood, MA: Artech House, 2007.

[4] W. Liu, L. Lu, M. O’Neill, and E. E. Swartzlander Jr., “Design rules forquantum-dot cellular automata,” in *Proc. IEEE Int. Symp. Circuits Syst.(ISCAS)*, Rio De Janeiro, Brazil, May 2011, pp. 2361–2364.

[5] K. Kim, K. Wu, and R. Karri, “Towards designing robust QCA architectures in the presence of sneak noise paths,” in *Proc. IEEE Design, Automation Test Eur. Conf. Exhib. (DATE)*, Munich, Germany, Mar. 2005,pp. 1214–1219.

[6] K. Navi, M. H. Moaiyeri, R. F. Mirzaee, O. Hashemipour, and B. M. Nezhad, “Two new low-power full adders based on majority-not gates,”*Microelectron*. J., vol. 40, pp. 126-130, 2009.

[7] H. Cho and E. E. Swartzlander Jr., “Adder design and analyses for quantum-dot cellular automata,”*IEEE Trans. Nanotechnol.*, vol. 6, no. 3, pp. 374–383, May 2007

[8] H. Cho and E. E. Swartzlander Jr., “Adder and multiplier design in quantum-dot cellular automata,”*IEEE Trans. Comput.*, vol. 58, no. 6, pp. 721–727, Apr. 2009.

[9] V. Pudi and K. Sridharan, “Efficient design of a hybrid adder in quantumdot cellular automata,”*IEEE Trans. VLSI Syst.*, vol. 19, no. 9, pp. 1535–1548, Jul. 2011.

[10] M. Gladshtein, “*Quantum-dot cellular automata serial decimal adder,”IEEE Trans. Nanotechnol.*, vol. 10, no. 6, pp. 1377–1382, Nov. 2011.

[11] V. Pudi and K. Sridharan, “Low complexity design of ripple carry and Brent-Kung adders in QCA,”*IEEE Trans. Nanotechnol.*, vol. 11, no. 1,pp. 105–119, Jan. 2012.

[12] S. Perri and P. Corsonello, “New methodology for the design of efficient binary circuits addition in QCA,”*IEEE Trans. Nanotechnol.*, vol. 11, no. 6, pp. 1192–1200, Nov. 2012.

[13] V. Pudi and K. Sridharan, “New decomposition theorems on majority logic for low-delay adder designs in quantum dot cellular automata,” *IEEE Trans. Circuits Syst. II: Exp. Brief*, vol. 59, no. 10, pp. 678–682, Oct. 2012.

[14] H. Cho and E. E. Swartzlander Jr., “Serial parallel multiplier design in quantum-dot cellular automata,” in *Proc. IEEE Symp. Comput. Arithmetic*, 2007, pp. 7–15.

[15] S. W. Kim and E. E. Swartzlander Jr., “Parallel multipliers for quantumdot cellular automata,” in *Proc. IEEE Nanotechnol. Mater. Devices Conf.*,2009, pp. 68–72

[16] C.-H. Huang and J.-S. Wang, .High-performance and power efficient CMOS comparators,. *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 254.262, Feb. 2003.