

Portland State University

**PDXScholar**

---

Dissertations and Theses

Dissertations and Theses

---

1-1-2011

# Logic Realization Using Regular Structures in Quantum-Dot Cellular Automata (QCA)

Rahul Singhal

*Portland State University*

Follow this and additional works at: [https://pdxscholar.library.pdx.edu/open\\_access\\_etds](https://pdxscholar.library.pdx.edu/open_access_etds)

**Let us know how access to this document benefits you.**

---

## Recommended Citation

Singhal, Rahul, "Logic Realization Using Regular Structures in Quantum-Dot Cellular Automata (QCA)" (2011). *Dissertations and Theses*. Paper 196.

<https://doi.org/10.15760/etd.196>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: [pdxscholar@pdx.edu](mailto:pdxscholar@pdx.edu).

Logic Realization Using Regular Structures in Quantum-Dot Cellular Automata (QCA)

by

Rahul Singhal

A thesis submitted in partial fulfillment of the  
requirement for the degree of

Master of Science  
in  
Electrical and Computer Engineering

Thesis Committee:  
Marek A. Perkowski, Chair  
Douglas V. Hall  
Christof Teuscher

Portland State University  
©2011

## ABSTRACT

Semiconductor industry seems to approach a wall where physical geometry and power density issues could possibly render the device fabrication infeasible. Quantum-dot Cellular Automata (QCA) is a new nanotechnology that claims to offer the potential of manufacturing even denser integrated circuits, which can operate at high frequencies and low power consumption. In QCA technology, the signal propagation occurs as a result of electrostatic interaction among the electrons as opposed to flow to the electrons in a wire. The basic building block of QCA technology is a QCA cell which encodes binary information with the relative position of electrons in it. A QCA cell can be used either as a wire or as logic. In QCA, the directionality of the signal flow is controlled by phase-shifted electric field generated on a separate layer than QCA cell layer. This process is called clocking of QCA circuits.

The logic realization using regular structures such as PLAs have played a significant role in the semiconductor field due to their manufacturability, behavioral predictability and the ease of logic mapping. Along with these benefits, regular structures in QCA's would allow for uniform QCA clocking structure. The clocking structure is important because the pioneers of QCA technology propose it to be fabricated in CMOS technology. This thesis presents a detailed design implementation and a comparative analysis of logic realization using regular structures, namely Shannon-Lattices and PLAs for QCAs. A software tool was developed as a part of this research, which automatically generates complete QCA-Shannon-Lattice and QCA-PLA layouts for single-output Boolean

functions based on an input macro-cell library. The equations for latency and throughput for the new QCA-PLA and QCA-Shannon-Lattice design implementations were also formulated. The correctness of the equations was verified by performing simulations of the tool-generated layouts with QCADesigner. A brief design trade-off analysis between the tool-generated regular structure implementation and the unstructured custom layout in QCA is presented for the full-adder circuit.

## TABLE OF CONTENTS

<b>ABSTRACT.....</b>	<b>i</b>
<b>LIST OF TABLES .....</b>	<b>v</b>
<b>LIST OF FIGURES .....</b>	<b>vi</b>
<b>CHAPTER 1: INTRODUCTION.....</b>	<b>1</b>
<b>CHAPTER 2: QCA BACKGROUND .....</b>	<b>7</b>
2.1 QCA Cell .....	7
2.2 Cell-to-Cell Response .....	8
2.3 QCA Wire .....	9
2.4 QCA Wire Fan-out.....	11
2.5 QCA Gates .....	12
2.5.1 Inverter.....	12
2.5.2 Majority, AND and OR Gates.....	12
2.6 QCA Wire Crossing.....	13
2.7 QCA Clocking .....	14
2.8 QCA Clocking Architecture .....	17
<b>CHAPTER 3: REGULAR STRUCTURES.....</b>	<b>20</b>
3.1 Motivation for Regular Structures .....	20
3.2 Programmable-Logic-Array (PLA).....	21
3.3 Shannon-Lattice .....	21
3.4 Generating a Shannon-Lattice.....	23
3.5 Symmetric and Non-symmetric Functions .....	26
3.5.1 Symmetric functions .....	26
3.5.2 Non-symmetric functions.....	28
<b>CHAPTER 4: QCA CIRCUIT IMPLEMENTATION .....</b>	<b>32</b>
4.1 Basic QCA Circuit Design Considerations.....	32
4.1.1 Signal synchronization.....	32
4.1.2 Wire latency .....	34
4.1.3 Basic gates and logic synthesis .....	36
4.1.4 Wire length.....	37
4.2 Wire Length and Clocking Strategies .....	37
4.2.1 2-Dimensional (2D) clocking scheme.....	38
4.2.2 2-Dimensional-Diagonal-Wave (2DD) clocking scheme .....	41
4.3 QCA Circuits .....	43
4.3.1 QCA Multiplexer .....	44
4.3.2 QCA AND-OR-INVERT (AOI) Logic.....	46

4.3.3 QCA Full Adder.....	47
4.3.4 QCA D-Latch.....	48
4.3.5 QCA PLA.....	50
4.4 Macro Cells for Regular Layout .....	55
4.4.1 Multiplexer-Cell: QCA-Shannon-Lattice .....	55
4.4.2 AND & OR-Plane Cells: QCA-PLA .....	58
<b>CHAPTER 5: LAYOUT GENERATOR TOOL USING REGULAR CELLS.....</b>	<b>68</b>
5.1 Motivation for Layout Software .....	68
5.2 Layout File Generation Flow .....	69
5.3 File Formats .....	71
5.3.1 Non-minimized-Boolean-Function file.....	72
5.3.2 Function-boolean-expression file.....	73
5.3.3 Configuration file.....	74
5.3.4 Cell-layout files.....	75
5.4 Layout Generator Algorithm.....	76
5.4.1 Algorithm for QCA-Shannon-Lattice layout .....	76
5.4.2 Algorithm for QCA-PLA layout.....	88
<b>CHAPTER 6: ANALYSIS AND RESULTS.....</b>	<b>93</b>
6.1 Macro-Cell Characterization.....	93
6.1.1 QCA-Multiplexer macro-cell.....	93
6.1.2 QCA-PLA AND-plane macro-cell.....	95
6.1.3 QCA-PLA OR-plane macro-cell.....	97
6.2 QCA-PLA Latency and Throughput Calculations.....	100
6.2.1 QCA-PLA latency calculation .....	100
6.2.2 QCA-PLA throughput calculation .....	104
6.3 QCA-Shannon-Lattice Latency and Throughput Calculations .....	110
6.3.1 QCA-Shannon-Lattice latency calculation .....	110
6.3.2 QCA-Shannon-Lattice Throughput Calculation .....	112
6.4 Comparison Table: QCA-Shannon-Lattice and QCA-PLA.....	114
6.5 Full-Adder (FA) implementations: regular structures and custom layout .....	125
6.5.1 FA implementation: custom layout.....	125
6.5.2 FA implementation: regular structures .....	128
<b>CHAPTER 7: CONCLUSION AND FUTURE WORK .....</b>	<b>136</b>
<b>APPENDIX: TOOL GENERATED LAYOUTS .....</b>	<b>142</b>

## LIST OF TABLES

Table 3-1 Truth Table for $X = (A \& B)   C$ .....	23
Table 6-1 QCA-Multiplexer macro-cell characteristics.....	95
Table 6-2 QCA-PLA AND-plane macro-cell characteristics. ....	97
Table 6-3 QCA-PLA AND-plane macro-cell characteristics. ....	99
Table 6-4 Combined table of macro-cell characteristics. ....	99
Table 6-5 QCA-Shannon-Lattice benchmark data.....	116
Table 6-6 QCA-PLA benchmark data. ....	117
Table 6-7 Modified FA custom layout characteristics .....	127
Table 6-8 FA QCA-Shannon-Lattice layout with combined sum and carry-out characteristics .	131
Table 6-9 Full-Adder QCA-PLA layout with combined sum and carry-out characteristics .....	133
Table 6-10 Combined table for characteristics of different FA layout types.....	134

## LIST OF FIGURES

Figure 2-1 QCA cell polarization. ....	8
Figure 2-2 Rotated (45°) QCA cells.....	8
Figure 2-3 Nonlinear cell-to-cell response function. ....	9
Figure 2-4 QCA Wire. ....	10
Figure 2-5 QCA inverter chain. ....	10
Figure 2-6 Wire Fan-outs.....	11
Figure 2-7 Types of QCA inverters. ....	12
Figure 2-8 QCA gates. ....	13
Figure 2-9 QCA wire crossing.....	14
Figure 2-10 QCA clocking phases.....	15
Figure 2-11 Single cell-switching behavior with clock. ....	16
Figure 2-12 Cell phases in a wire transmitting logic 1 .....	17
Figure 2-13 QCA clocking architectures .....	18
Figure 3-1 Schematic for Programmable-Logic-Array.....	21
Figure 3-2 Aker Arrays and Shannon Lattice .....	22
Figure 3-3 Table-division level-1. ....	23
Figure 3-4 Multiplexer implementation level .....	24
Figure 3-5 Table-division level-2. ....	24
Figure 3-6 Shannon Lattice for function $X = (A \& B)   C$ .....	25
Figure 3-7 Shannon Lattice Diagram and corresponding multiplexer implementation of Symmetric Function in Section 3.5.1.....	28
Figure 3-8 Shannon Lattice Diagram of Asymmetric Function in Section 3.5.2.....	31
Figure 4-1 Signal Synchronization in QCA Circuits. ....	34
Figure 4-2 Clocking zone assignment for a QCA-wire. ....	36
Figure 4-3 1D and 2D Clocking Strategies for QCA Circuits. ....	39
Figure 4-4 2D-Diagonal clocking for QCA circuits. ....	42
Figure 4-5 QCA 2-to-1 Multiplexer Design and Simulation Waveforms.....	45
Figure 4-6 AND-OR-INVERT (AOI) Logic and simulation waveforms. ....	47
Figure 4-7 QCA Full-Adder.....	48
Figure 4-8 QCA D-Latch and simulation waveforms.....	50
Figure 4-9 PLA: AND-plane. ....	52
Figure 4-10 PLA: OR-plane.....	54
Figure 4-11 Logic diagram: QCA PLA implementing XOR and OR logic. ....	55
Figure 4-12 QCA Multiplexer macro-cell and QCA-Shannon-Lattice: simulated in QCADesigner to verify functionality. ....	57
Figure 4-13 QCA-PLA AND-plane macro-cell and AND-Plane: simulated in QCADesigner to verify functionality.....	62
Figure 4-14 QCA-PLA OR-plane macro-cell and OR-plane: simulated in QCADesigner to verify functionality. ....	65
Figure 4-15 QCA-PLA for an arbitrary function: simulated in QCADesigner.....	67



Figure 5-1 Layout-generation flow. ....	70
Figure 5-2 Detailed layout-generation and simulation flow. ....	71
Figure 5-3 exam3_d benchmark in <i>blif</i> format and its ESPRESSO minimized equivalent. ....	72
Figure 5-4 Minimized function of Figure 5-3(b) as Boolean SOP expression with four terms. ....	73
Figure 5-5 Configuration file. ....	74
Figure 5-6 An example QCA-cell description for QCADesigner simulator. ....	83
Figure 5-7 Pictorial representation of algorithm for creating QCA-cell data for Shannon-Lattice. .....	85
Figure 5-8 Pictorial representation of algorithm for creating QCA-cell data for QCA-PLA. ....	91
Figure 6-1 Latency computation for QCA-Multiplexer. ....	94
Figure 6-2 Latency computation for QCA-PLA AND-plane macro-cell. ....	96
Figure 6-3 Latency computation for QCA-PLA OR-plane macro-cell. ....	98
Figure 6-4 QCA-PLA latency calculation. ....	101
Figure 6-5 QCA PLA throughput computation. ....	107
Figure 6-6 Simulation results for latency and throughput of QCA-PLA. ....	110
Figure 6-7 QCA-Shannon-Lattice latency calculation. ....	111
Figure 6-8 Simulation results for latency and throughput of QCA-Shannon-Lattice. ....	114
Figure 6-9 Number of product-terms vs. latency. ....	119
Figure 6-10 Number of variables vs. latency. ....	119
Figure 6-11 Latency comparison of QCA-PLA and QCA-Shannon-Lattice. ....	120
Figure 6-12 Number of variables vs. throughput. ....	121
Figure 6-13 Number of product-terms vs. throughput. ....	122
Figure 6-14 Throughput comparison of QCA-PLA and QCA-Shannon-Lattice. ....	122
Figure 6-15 Area comparison of QCA-PLA and QCA-Shannon-Lattice. ....	124
Figure 6-16 QCA FA .....	125
Figure 6-17 Modified FA custom layout .....	127
Figure 6-18 Simulation waveforms of modified FA generated using QCADesigner. ....	128
Figure 6-19 FA sum function realized using QCA-Shannon-Lattice .....	130
Figure 6-20 FA carry-out function realized using QCA-Shannon-Lattice .....	130
Figure 6-21 FA sum function realized using QCA-PLA .....	132
Figure 6-22 FA carry-out function realized using QCA-PLA .....	133
Figure A-1 QCA-Shannon-Lattice layout of exam1_d. ....	142
Figure A-2 QCA-PLA layout of exam1_d. ....	143
Figure A-3 QCA-Shannon-Lattice layout of <b>S0, 1, 3</b> (a, b, c, d, e, f).....	144
Figure A-4 QCA-PLA layout of <b>S0, 1, 3</b> (a, b, c, d, e, f).....	145
Figure A-5 QCA-Shannon-Lattice layout of xor5_d .....	146
Figure A-6 QCA-PLA layout of xor5_d .....	147
Figure A-7 QCA-Shannon-Lattice layout of <b>S0, 1, 7, 8</b> (a, b, c, d, e, f, g, h).....	148
Figure A-8 QCA-PLA layout of <b>S0, 1, 7, 8</b> (a, b, c, d, e, f, g, h).....	149

## Chapter 1

# INTRODUCTION

For last 40 years the advancements in the semiconductor industry have been able to keep up with the Moore's law. According to the International Technology Roadmap for Semiconductors (ITRS) [1] projections, this trend is expected to continue until 2020 but beyond that the physical and power density limitations would prohibit the further scaling of the integrated circuits in contemporary manufacturing technology of complementary-metal-oxide-semiconductor (CMOS). For this very reason, researchers have been searching the avenues of nanotechnology for the rescue. They claim that nanotechnology using new fabrication materials could potentially replace CMOS for the most part because of the high device densities and low power consumption in nanotechnologies [2].

One of the most promising nanotechnologies is Quantum-Dot Cellular Automata (QCA) which was first proposed by P.D. Tougaw and C.S. Lent in early 90's [3]. In this technology, the basic device for circuit implementation is a QCA-Cell which is very small thus enabling high densities. Both theoretical and experimental research in QCA claims that QCA circuits can operate at THz frequencies with low power consumption [2]. A QCA cell contains electrons and the relative positions of the electrons in a cell are used to encode the binary information as opposed to voltage levels in CMOS. The counterpart of switching in CMOS is the Coulombic interaction between the electrons of QCA cells. This Coulombic interaction determines the position of electrons and thus the logic state of a cell. Chapter 2 provides background details on QCA.

In CMOS or earlier technologies, the logic realization using regular structures, like Shannon lattice [4, 5-8] and Programmable-Logic-Arrays (PLA) have played an important role due to their ease of logic mapping and minimal routing requirements. In addition, it is easier to develop Electronic-Design-Automation (EDA) tools for logic synthesis using these regular structures compared to custom logic of unstructured logic networks. These properties of regular structures are highly desirable and beneficial for logic synthesis in QCA in particular. The advantages are threefold; First, QCA circuits inherit the advantages of easier logic mapping and routing. Secondly, these properties are highly desirable since aforementioned processes are more complicated in QCA compared to previous circuit design technologies as the QCA is still developing [9]. Thirdly, the use of regular structures results in simplified and uniform clocking structures (described in chapter 2.), which is one of the biggest challenges in the implementation of QCA circuits [10]. A simple and uniform clocking structure would increase manufacturability assuming that QCA technology would advance to that level in future.

EDA tools play a vital role in the design and manufacturing of integrated circuits. EDA tools have enabled a faster and reduced effort design cycle as the complexity of chips increased considerably. It is evident that any new IC design technology would require a similar set of tools and QCA is no different. The QCA designs are not yet ready to scale to large sizes, however, the fundamental differences in circuit implementation between QCA and previous techniques require modeling and simulation EDA tools for correct understanding of QCA technology characteristics [11]. So far, the work involving the development of tools for QCA has been purely academic. The first tools developed for

QCA simulations were MAQUINAS and QBert [12, 13]. The initial version, AQUINAS, which was able to perform quantum mechanical simulations of small circuits, was expanded to MAQUINAS which focused on molecular QCA. QBert tool was able to simulate larger designs but was limited to the digital logic simulations. One of the most popular recent tools is QCADesigner [11, 14] which allows users to create circuit layouts using graphical user interface and perform both quantum mechanical and digital simulations. Another layout tool called QCA-LG [15] generates an automatic layout of simple combinational circuits by reading the net-list produced by synthesis tools. QCA-LG generates an equivalent QCA circuit representation of the input net-list and then computes the QCA-cell properties and their coordinates based on that representation. It then writes all the data to a file following the QCADesigner cell definition. This file can then be simulated using QCADesigner simulator. Although the focus of the research presented in this thesis was essentially not to create a tool, the analysis needed for this work led to development of a simple but efficient logic synthesis and layout tool. It follows the same methodology, of generating QCA-cell data for whole QCA design and writing them to a file, as QCA-LG but uses QCA-cell data of input macro-cells as building blocks.

The main focus of this research is to study the trade-offs in logic implementation using regular structures in QCA domain. In particular, the goal is to compare the single output logic implementations using Shannon lattices with PLAs for area, latency and throughput using QCAs. The term regular structures or regularity in context to this thesis implies that no routing or specific placement process is needed at the layout level to connect

modules used for logic implementation. In other words, regular structures should enable the generation of complete QCA layout for a logic function simply by replicating and abutting a QCA structure in a tile-based fashion. This condition applies to both Shannon-Lattice and PLA layouts. The choice of regular structures, Shannon-Lattice and PLA, for this thesis was based on the fact that these two structures are the most common and simple regular structures for logic realization. The simplicity of these structures is also favored by the number of QCA structures needed to be replicated to create each type of layout. The QCA-Shannon-Lattice requires only a QCA-multiplexer as a basic structure and QCA-PLA requires two basic structures: AND-plane and OR-plane structure. These structures and their use are discussed in detail in section 4.4 of this thesis. The reasoning behind the consideration of only single output functions in this thesis owes to their simpler logic mapping to both Shannon-Lattice and PLA. The extension of PLA to multi-output is easier than Shannon-Lattice because generating multi-output Shannon-Lattice would require several optimizations. The implementation of such optimization algorithms for Shannon-Lattice in software tool, developed in this thesis, would be complicated and is out of the scope of this research work. A study comparing single output functions when realized using CMOS Shannon-Lattice and PLA has already been presented in [7].

This goal involves generating QCA layout for different logic functions using both Shannon lattices and PLAs which can then be simulated with QCADesigner. Due to the fundamental differences in QCA and previous semiconductor-based technologies, we may need to re-evaluate the designs and established studies for silicon based technologies

such as one presented in [7] differently. There are no similar previous studies related to this work that would compare any of: area, latency and throughput. The closest published research is the reliability comparison of PLA and custom logic implementation of a QCA adder [14]. The authors of [16] claim that smaller component requirements of custom logic make them more reliable than regular designs assuming that the components used to build each circuit are faulty.

The main contributions of this thesis are:

- Realized and characterized the macro-cells for both QCA-PLA and QCA-Shannon-Lattice to be used for a complete regular layout [Section 6.1].
- Presented detailed clocking and layout methodology for QCA-PLA and QCA-Shannon-Lattice that can be verified for functionality in QCADesigner simulator [Sections 4.4.1 and 4.4.2].
- Developed a software tool to perform simple logic synthesis for Shannon-Lattice and generate QCA layout for both QCA-PLA and QCA-Shannon-Lattice for symmetric Boolean functions [Chapter 5].
- Presented a comparative-analysis of logic implementation using QCA-PLA and QCA-Shannon-Lattice in terms of area and latency [Section 6.4].

The presented thesis is organized as follows. Chapter 2 provides the QCA background and circuit theory by discussing the basic building blocks like device cell, gates, interconnects etc. Chapter 3 provides details on two categories of logic functions;

namely, symmetric and non-symmetric functions. It also describes generating Shannon-Lattice using Shannon's expansion and differences in the process for two aforementioned types of functions. The process of mapping the lattice diagram to a tree of multiplexers is also discussed in detail. Building on the knowledge of chapter 2, chapter 4 focuses on circuit implementation in QCA. It also briefly describes some basic circuit design differences between QCA circuit and silicon-based circuits. In addition, chapter 4 presents the idea of generating QCA layout using a library of regular QCA macro-cells by describing the layout and placement of QCA-multiplexer-cell and QCA-PLA AND-plane and OR-plane-cells. Chapter 5 focuses on algorithms and file formats used by the layout tool developed in this work for generating QCA layout for arbitrary logic functions. Chapter 6 shows the characterization of the input macro-cells used for both QCA-PLA and QCA-Shannon-Lattice and presents the comparative analysis on logic realization using both QCA-PLA and QCA-Shannon-Lattice. Chapter 6 concludes with a brief design trade-off analysis between the tool-generated regular structure implementation and the unstructured custom layout in QCA using the full-adder circuit as the example.

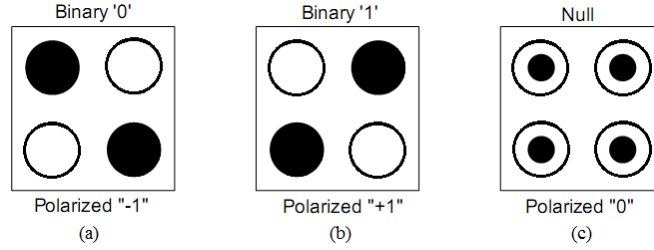
## CHAPTER 2

### QCA BACKGROUND

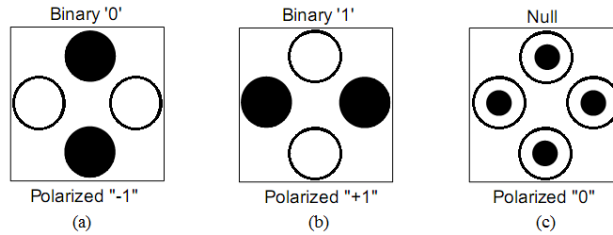
#### 2.1 QCA Cell

The basic device in QCA is a ‘QCA-cell’ which enables both the computation and transmission of the information. A QCA cell consists of a hypothetical square space with four electronic sites and two electrons. The electronics sites, called ‘Dots’, represent the locations which the electrons can occupy. The dots are coupled through quantum mechanical tunneling barriers and electrons can tunnel through them depending on the state of the system. Due to their Coulombic repulsion, the electrons tend to occupy the farthest dots in a cell which corresponds to the lowest energy state of the system. The relative positions of the electrons in a cell are used to represent the ‘cell-polarization’ which can then be used to encode binary states 0 or 1. There are two types of QCA cells namely  $90^\circ$  and  $45^\circ$  cells. Figure 2-1(a) shows  $90^\circ$  cells with polarization of  $P = +1$  which represents binary 1 and Figure 2-1(b) shows a polarization of  $P = -1$  representing binary 0. These two states are called the cell’s ACTIVE states. A cell is driven into a particular polarization due to an external field or adjacent polarized cells. Thus a completely isolated cell would have a polarization of  $P = 0$  which is called NULL state shown in Figure 2-1(c). Other types of cells are shown in Figure 2-2(a) and (b). These cells have same properties as  $90^\circ$  cells except that the dots have been rotated by  $45^\circ$  thus called as  $45^\circ$  cells. The cell in Figure 2-2(a) represents a binary 1 and Figure 2-2(b) represents a binary 0.





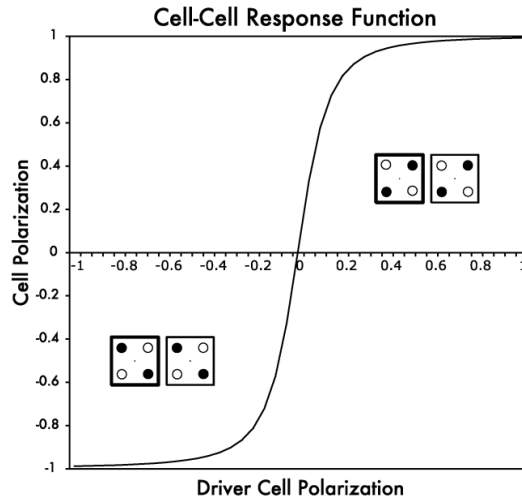
**Figure 2-1 QCA cell polarization.**



**Figure 2-2 Rotated (45°) QCA cells.**

## 2.2 Cell-to-Cell Response

The cell-to-cell response function is shown in Figure 2-3. It illustrates that the nature of the response is highly non-linear i.e. a weak polarization of one cell causes a strong polarization of the neighboring cell. This implies that, in a wire, if the driver or any of the intermediate cells had a weak polarization, the subsequent cell would still get strongly polarized. This behavior corresponds to a buffer restoring a signal value to rails of supply voltage  $V_{cc}$  or ground in conventional digital circuits [3]. This behavior is desirable for defect-tolerance in QCA circuits where one cell may not get strongly polarized due to placement-issues but would still result in strong polarization in neighboring cells.



**Figure 2-3 Nonlinear cell-to-cell response function<sup>1</sup>.**

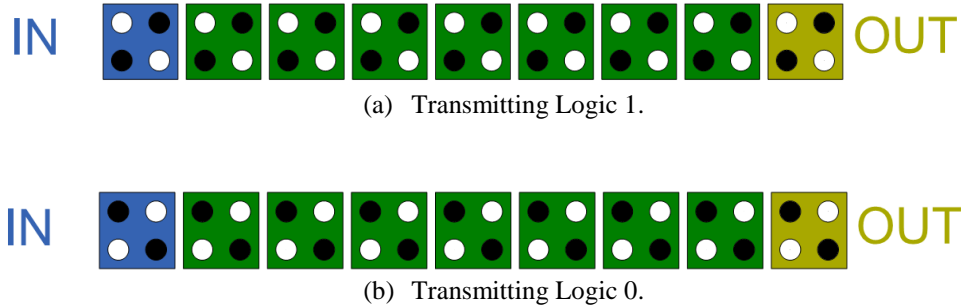
## 2.3 QCA Wire

A QCA wire can be formed by placing the QCA cells in a linear fashion. The Columbic interaction between the adjacent cells causes the polarization of a cell to align to its neighboring cells thus allowing the transmission of information along the array of cells. Figure 2-4 shows QCA wires transmitting 0 and 1 using  $90^\circ$  cells. In both Figures, the input cell is driven by an external source and is strongly polarized in one direction. The input cell then acts as a driver to the other cells in the NULL state which align themselves to the input cell polarization to reach the system's ground state. In Figure 2-4(a), the input cell IN is strongly polarized to +1 and to bring the system of the first two cells to the ground state, the electrons in first adjacent green cell would tend to align in an orientation with the least electrostatic repulsive forces of the electrons in the cells. This

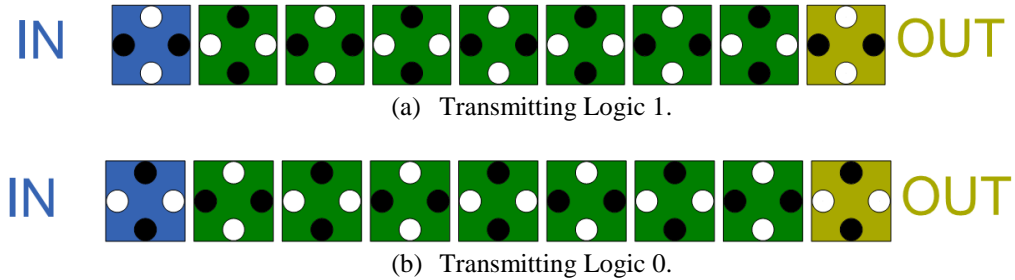
---

<sup>1</sup> Figure 2-3 Source [11]

is only possible if the top electron in green cell moves to the top-right dot of the cell. The same theory applies to the rest of the green cells and finally the output cell in yellow polarizes to binary 1 as well. Similarly, a binary '0' is transmitted from input to output cell OUT in Figure 2-4(b) [3].



**Figure 2-4 QCA Wire.**

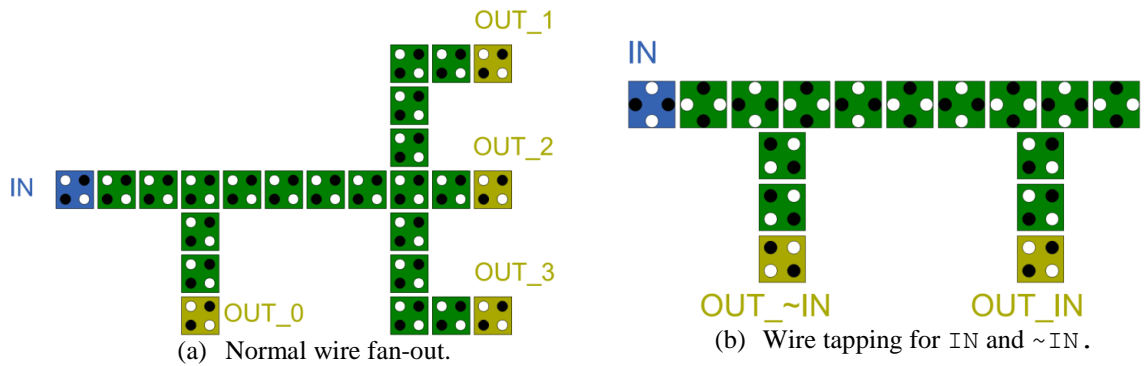


**Figure 2-5 QCA inverter chain.**

However, if a wire is formed using  $45^\circ$  cells then it results in an inverter chain as shown in Figure 2-5 (a) and (b) above. When two cells  $45^\circ$  are placed adjacent to each other, they always have opposite polarities, which results in the ground state of the system of those cells. The wire with  $45^\circ$  cells is called an inverter-chain because the logic level of any cell in the wire is the negation of its adjacent previous cell.

## 2.4 QCA Wire Fan-out

The fan-out in QCA is achieved in a similar fashion as in the conventional semiconductor technology, by branching out the wire into multiple end-points or wire-tapping from a wire. These end-points or taps could then serve as inputs to other gates or final output. This is illustrated in Figure 2-6(a) with input  $IN$  as logic 1.



**Figure 2-6 Wire Fan-outs.**

Some circuits often require a signal and its complement at the same time. In semiconductor domain, it is generally obtained by globally running a signal wire along with its negated wire or locally negating the signal using inverter. In QCA, both signal and its complement can be generated by routing signal as an inverter chain and tapping off at an appropriate cell-pair. This arrangement is shown in Figure 2-6(b). Solving for the ground state of cell, it can be found that output  $OUT\_IN$  of Figure 2-6(b) is equal to the input  $IN$  and the output  $OUT\_ \sim IN$  is the input's  $IN$  complement.

## 2.5 QCA Gates

### 2.5.1 Inverter

The QCA cells can be arranged in a particular fashion to easily create traditional logic gates. The basic gates in QCA technology are the majority gate and the inverter. Figure 2-7 shows two ways of creating inverters. In Figure 2-7(a), the inverter uses only two cells which are displaced with respect to each other. It can be observed that the gate cells reach the ground state when they have opposite polarities as the electrons are farthest apart. This inverter suffers from signal integrity issues because the displaced cell does not get highly polarized in opposite direction than previous cell. Figure 2-7(b) shows a different type of inverter which is bigger in size but is more robust when compared of two-cell inverter. Another advantage of this inverter is that the negated output is aligned to the input [3]. This would be helpful, for instance, in a bigger QCA circuits when a signal needs to be inverted and still connected to an input of a QCA-gate.



Figure 2-7 Types of QCA inverters.

### 2.5.2 Majority, AND and OR Gates

The majority gate is illustrated in Figure 2-8 (a). The output  $F$  is defined as  $F = AB+AC+BC$ . The output cell of the gate polarizes to the computation cell in the center of the gate. The output  $F$  can be propagated using a QCA wire which can then act as an input to other gates. The majority gate can be used to build the AND and OR gates. If one of the inputs is fixed to 0/1, the resulting function  $F$  is the AND/OR of remaining two inputs. AND and OR gates are shown in Figure 2-8 (b) and Figure 2-8 (c) respectively [3].

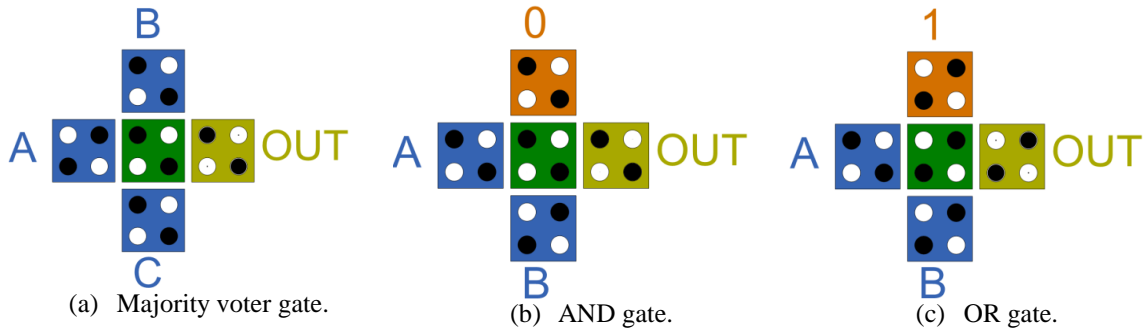
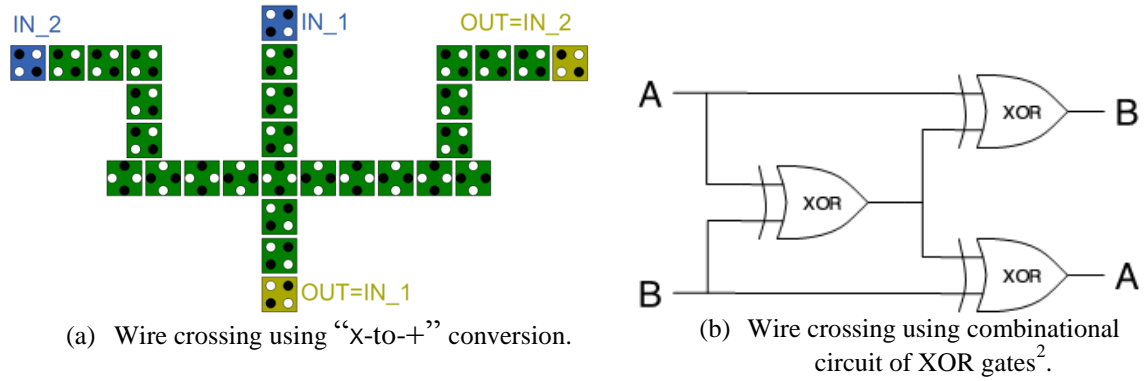


Figure 2-8 QCA gates.

## 2.6 QCA Wire Crossing

In conventional technologies, the wire crossing is enabled by using different wire layers with insulation between them. The wires in one layer connect to the wires in other layers and the devices on bottom layer through metal vias. Many current advanced integrated circuits require up to 6 – 8 metal layers. On one hand, this method of signal routing and connection complicates the manufacturing process but on the other hand, it makes wire crossing simpler since no two wires cross each other on the same layer. In QCA technology, the current proposals restrict the designs to a single layer, which makes wire crossing complex. There are two ways in which this problem can be solved. The first

one is to convert one of the  $90^\circ$  wires into a  $45^\circ$  wire then cross the wires normally in the same plane. This is also called x-to-+ conversion as shown in Figure 2-9(a). This wire crossing is possible because the state of normal ('x' shaped cell) has no switching effect on rotated ('+' shaped cell) directly in line with it. The details can be found in [3]. (Note: Only single layer QCA designs have been considered throughout this work).



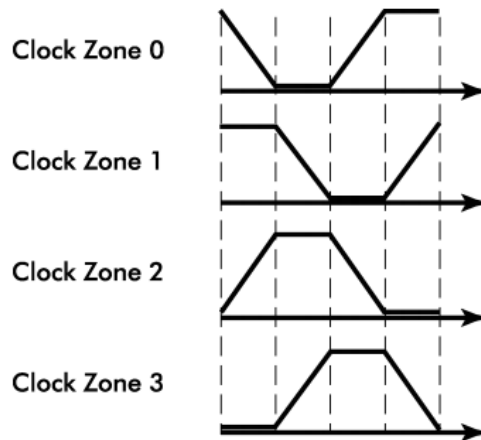
**Figure 2-9 QCA wire crossing.**

Second one is to use a combinational circuit which is equivalent to wire-crossing as shown in Figure 2-9(b) [2, 10]. The XOR gates are not the primitives gates in the QCA technology and thus have to be implemented using other basic gates. This makes the use of XOR gates expensive in QCA and the circuit of Figure 2-9(b) would consume a much larger area compared to the first solution of wire crossing of Figure 2-9(a).

## 2.7 QCA Clocking

<sup>2</sup> Figure 2-9(b) Sources [2], [10]

The clocking in the QCA circuits controls the information flow and the synchronization in the circuit. It also provides the power gain and avoids the meta-stable states [11, 17]. In case of a QCA-cell, the meta-stable state corresponds to polarization of a cell that cannot be distinctively identified as logic 1 or logic 0. In other words, the cell polarization is neither strongly -1 (logic 0) nor +1 (logic 1). The clock in QCA technology, which is different than the conventional definition of clock in CMOS, consists of four phases: *switch* (unpolarized cells are driven by some input and get polarized depending on their neighbors' polarization), *hold* (cells are held in some definite polarization representing a binary state), *release* (cells lose their polarization) and *relax* (when cells lose their polarization in *release* phase, they remain unpolarized or null in this state) [18-20]. Each of these phases is a quarter of cycle apart from the previous phase which can be implemented by generating four clocks each with  $\pi/2$  phase difference from previous one. The four phases of a QCA clock are shown in Figure 2-10.



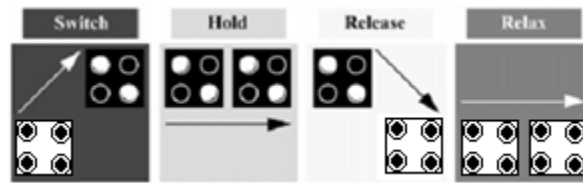
**Figure 2-10 QCA clocking phases<sup>3</sup>.**

---

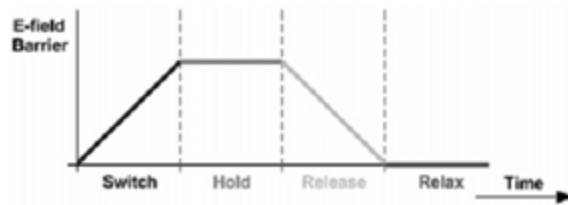
<sup>3</sup> Figure 2-10 Source [11]



Figure 2-11(a) and (b) below illustrate the behavior of a cell for different phases of the clock.



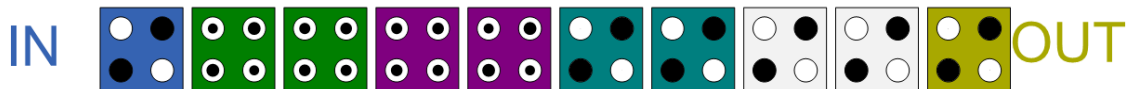
(a) QCA-cell behavior with clock phase.



(b) Different clock phases. Colors correspond to figure (a).

**Figure 2-11 Single cell-switching behavior with clock.**

Figures 2-12(a) to (e) show the cell phases with time while transmitting logic 1 from the input to the output. The different colors represent the different clocking zones i.e. the cells with same color are assigned the same clocking zone except that the blue and yellow colors represent input and output cell colors respectively although they are also in one of the clocking zones.



(a) Relax, Release, Hold, Latch.

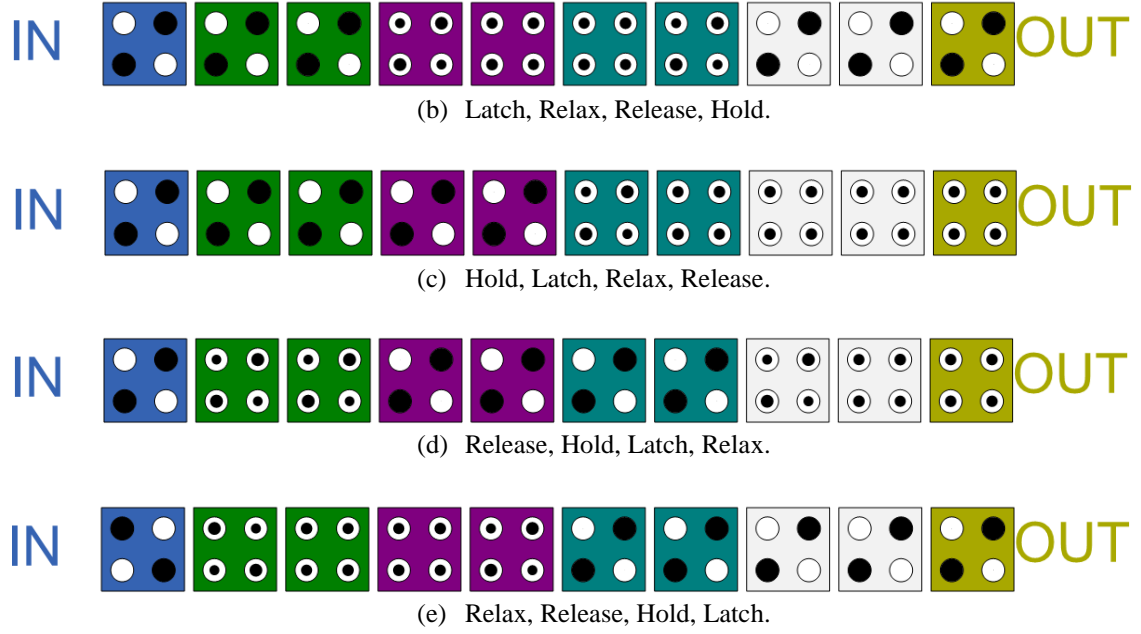


Figure 2-12 Cell phases in a wire transmitting logic 1<sup>4</sup>

In this manner, the information flows through a QCA wire with successive latching and unlatching at each clocking zone every quarter-cycle. Thus, each group of cells in a particular clocking zone can be considered as one D latch. It can be inferred that the more D-latches (clocking zones) a QCA wire has, the higher would be its latency. It implies that the number of the clocking zones should be minimized or alternatively as many cells should be grouped in a single clocking zone as possible. However, the number of cells in a zone should be chosen carefully because as the wire length increases, the probability of all cells in the wire switching successfully decreases [20].

## 2.8 QCA Clocking Architecture

<sup>4</sup>Figure 2-12 based on [21]

The authors of [11] describe two types of clocking mechanisms: zone clocking and continuous clocking.

**Zone clocking:** In this type of clocking, all the cells in a particular clocking zone are grouped and connected to one of clocking zones maintaining the clock phases in order [22]. This arrangement is illustrated in Figure 2-13(a) using an example of a larger design built with basic building blocks of QCA. The different shades of grey color represent different clocking zones. The QCA-cells in one clocking zone are switched together which results in propagation of data from left to right and finally to the output cell.

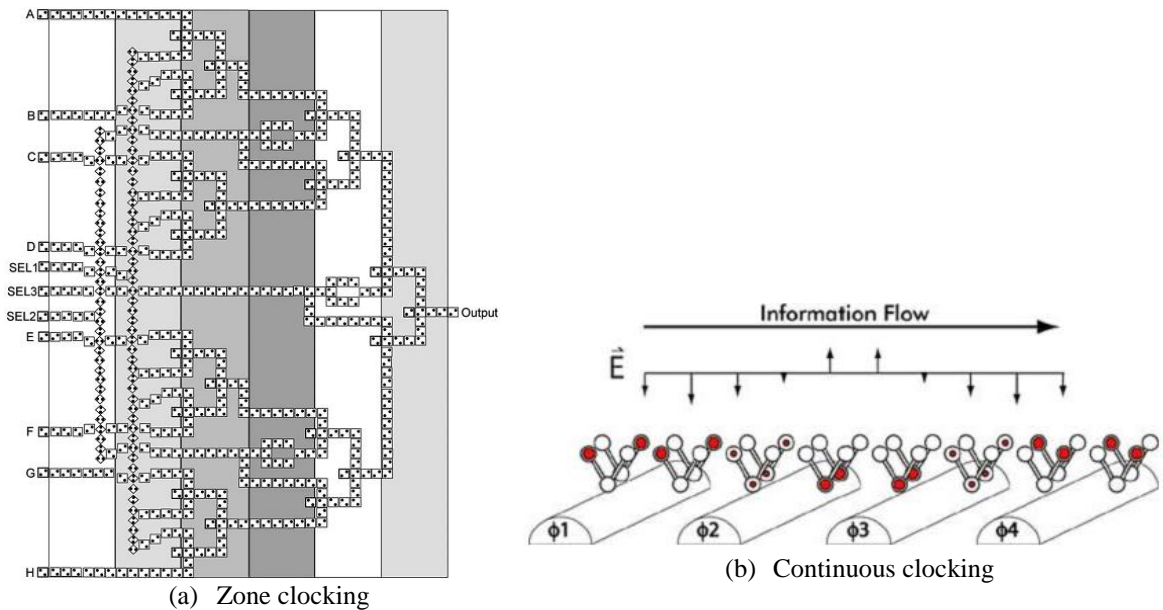


Figure 2-13 QCA clocking architectures<sup>5</sup>

<sup>5</sup> Figure 2-13(a) Source [23], Figure 2-13(b) Source [11]

**Continuous clocking:** In Continuous clocking, the potential field to enable clocking is generated by a system of electrodes submerged under the cell layer. The electrodes are driven by phase shifted sinusoids which results in a forward moving wave [17, 24]. This is shown in Figure 2-13(b) above.

The different implementations of Zone-clocking in QCA circuits have also been proposed in [23]. It is important to understand these extensions because they have been adapted for clocking QCA-PLA in this thesis. The details of these implementations are discussed in chapter 4 of this thesis.

## CHAPTER 3

### REGULAR STRUCTURES

#### 3.1 Motivation for Regular Structures

The CMOS technology has progressed to such smaller device geometries that the probabilities of variations in the manufacturing process are high. These factors result in the actual layout pattern that is different from the ones produced by the CAD tool. Regularity is a feature that can be employed to mitigate some effect of the variability. It can provide higher confidence in fabrication done by replicating layout pattern designed using CAD tool. Regularity improves the predictability of the physical implementation and allows for more accurate circuit analysis. Regularity is one of the design-for-manufacturability techniques exploited to some extent by the use of standard cells in CMOS [25-27]. With small device size, exploiting regularity with regular designs is also significantly important in the QCA technology.

As mentioned in the previous sections, the regular structures chosen for this study were the single output Shannon lattice and PLA. Any function minimized into a sum-of-products (SOP) form can be directly mapped to a PLA. PLAs do not require the placement and routing process as is the case with standard cell designs. Similarly, the Shannon lattice for a Boolean function is created by repeatedly applying the Shannon's expansion to the function. Each level can then be mapped on to an array of multiplexers or pass-transistors [8]. The inputs of the multiplexer at a higher level are directly driven

by the output of multiplexer at the immediate lower level. Thus, the whole Boolean function can be synthesized to a lattice of multiplexers abutted together without the need for placement and route processes.

### 3.2 Programmable-Logic-Array (PLA)

A simple PLA implementing two logic functions is shown in Figure 3-1. A PLA consists of a regular AND-plane which produces the product of literals and an OR-plane which produces the sum of the product terms that come from the AND-plane. Hence, implementing a Boolean function defined in the SOP form. The area of a PLA is directly related to the logic function. Figure 3-1 shows a PLA with two output signals since it is implementing two Boolean functions where the term  $(A \& B)$  is shared. In this thesis, only single output PLA implementations were considered.

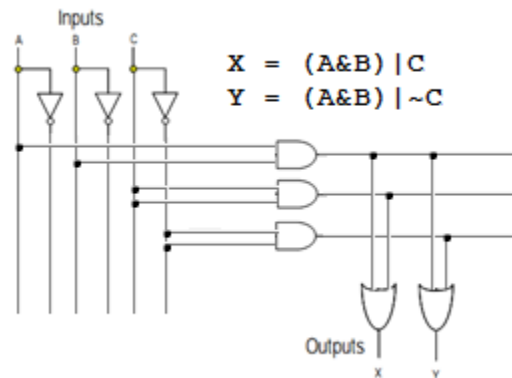
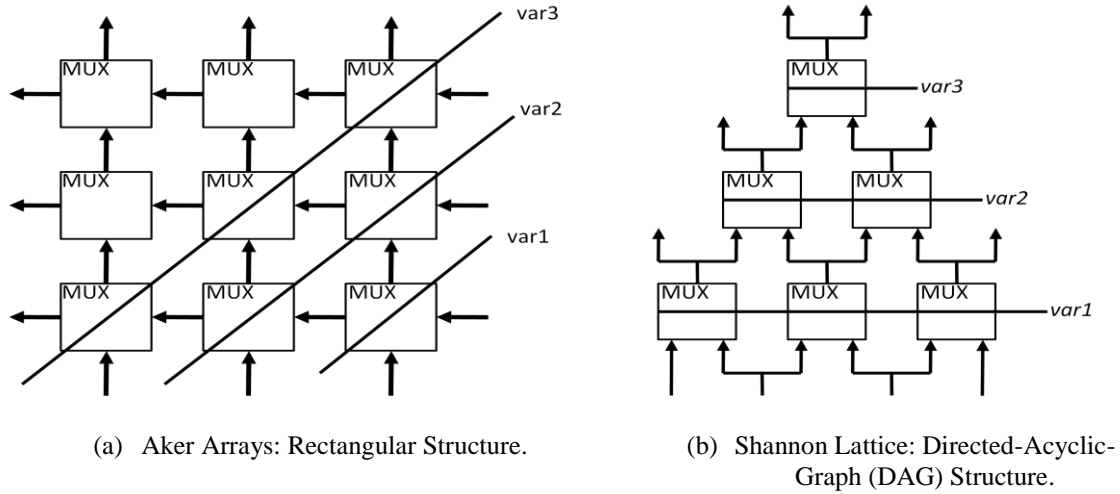


Figure 3-1 Schematic for Programmable-Logic-Array.

### 3.3 Shannon-Lattice

A Shannon lattice is a regular array based on *Universal-Aker-Arrays* (UAA) that attempts to combine the properties of *PLA-like* and *tree-like* cellular structures. UAAs were proposed by S.B. Akers in 1972 [28]. UAA can realize any Boolean function in a regular and planar layout. UAA is a rectangular array of multiplexers where each cell obtains inputs from South and East and produces outputs towards North and West. All diagonal multiplexers are connected to the same control variable that decides which input of the multiplexer flows to its output [5, 28]. This architecture is shown in Figure 3-2(a).



**Figure 3-2 Aker Arrays and Shannon Lattice**

Unlike UAAs, the Shannon lattice produces the layout triangular in shape as shown in Figure 3-2(b). Repeated Shannon expansion is applied at each level until the terms are reduced to constant or variables themselves. Any switching function of  $n$  variables  $f(x_1, x_2, \dots, x_n)$  can be expressed as:

$$f = x_i \cdot f_{xi} + x'_i \cdot f_{x'_i}$$

where:

$f_{x_i} = f(x_1, x_2, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$  is called positive co-factor of function  $f$  with respect to  $x_i$ .

$f_{x'_i} = f(x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$  is called negative co-factor function  $f$  with respect to  $x_i$ .

### 3.4 Generating a Shannon-Lattice

The analysis of the function:  $X = (A \& B \mid C)$  from previous section when implemented using Shannon lattice is presented below. Table 3-1 shows the truth table for function  $X$ .

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

**Table 3-1 Truth Table for  $X = (A \& B) \mid C$ .**

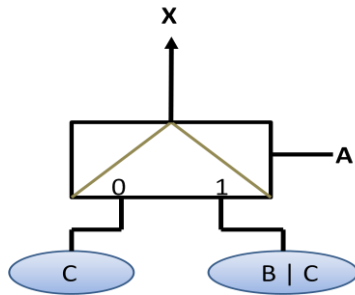
A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

**Figure 3-3 Table-division level-1.**

Referring to Figure 3-3, Table 1 can be thought of divided into two sections as shown by dotted line. One section with  $A = 0$  and other with  $A = 1$ . It can be observed that when  $A = 0$ , then output  $X$  of the function is equal to  $C$  shown in red box and when  $A = 1$ ,  $X$  is equal to OR of  $B$  and  $C$  shown in blue box. This behavior can be implemented



using a 2-to-1 multiplexer, as shown in Figure 3-4. The expressions in blue ovals are Boolean functions themselves, thus table in Figure 3-3 can be further divided. The top section of table in Figure 3-3 does not need to be divided since it will produce the same result, however, the bottom section of the dotted line can be divided into two sections when  $B = 0$  and  $B = 1$  as shown in Figure 3-5.



**Figure 3-4 Multiplexer implementation**  
level

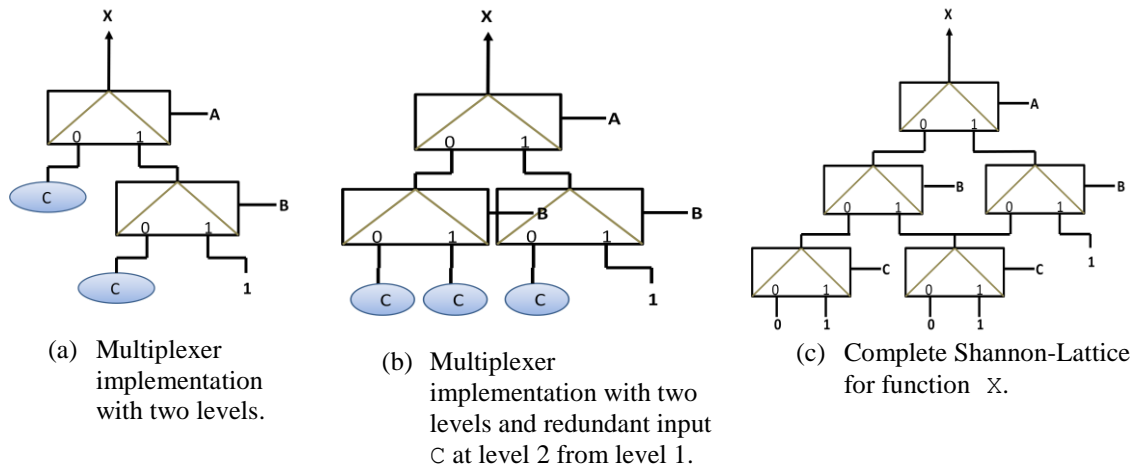
A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Annotations in the table:  
- Red box around rows A=0:  $[X = C]_{A=0, B=dc}$   
- Blue box around rows A=1:  $[X = C]_{A=1, B=0}$   
- Green box around rows A=1, B=1:  $[X = 1]_{A=1, B=1}$

**Figure 3-5 Table-division level-2.**

Thus, the Boolean function  $X = (A \ \& \ B) \mid C$  was decomposed for two levels using repeated Shannon expansions and relationship between the output X and input variables A, B, and C can be made from Figure 3-5. Levels 1 and 2 have reducing variables as A and B respectively. Similar to Figure 3-4, Figure 3-5 can be implemented using multiplexers as shown in Figure 3-6(a). The blue ovals could be driven directly by input C or be replaced with multiplexers. If decided to be implemented with multiplexers then some redundancy needs to be introduced. For example, for the blue oval containing the Boolean expression C at level 2 in Figure 3-6(a), both the positive and negative co-factors with respect to variable B are equal to C. The implementation of expression C using a multiplexer with variable B as the select line results in a multiplexer with both

inputs as C. A multiplexer with the same signal driving both inputs is redundant since its behavior would be equivalent to the input signal itself. This can be observed from Figure 3-6(b), however, this redundancy allows having terminal nodes as only constants.



**Figure 3-6 Shannon Lattice for function  $X = (A \& B) | C$ .**

From the table in Figure 3-5 and in Figure 3-6(b), it can be observed that, at level 3, several nodes are equivalent. Two equivalent nodes can be fused together into one node which can then be represented using a multiplexer. In this particular case of function  $X$ , it is coincident that all three nodes at the bottom are equivalent but not all three can be combined to one node as this method is not applicable to any general category of functions. However, the two equivalent nodes in the center can be combined into one node. This method is applicable to a general category of functions called *Symmetric Functions* (described in section 3.5.1 later). The complete mapping of  $X$  to Shannon lattice after combining equivalent/isomorphic nodes at level 3 and representing them with multiplexers is shown in Figure 3-6(c)

### 3.5 Symmetric and Non-symmetric Functions

#### 3.5.1 Symmetric functions

A switching function of  $n$  variables  $f(x_1, x_2, \dots, x_n)$  is called a (totally) symmetric function if and only if it is invariant under any permutations of variables. An example of symmetric function is:

$$f(a, b, c, d) = a.b.c.d + a'.b'.c'.d + a'.b'.c.d' + a'.b.c'.d + a.b'.c'.d'$$

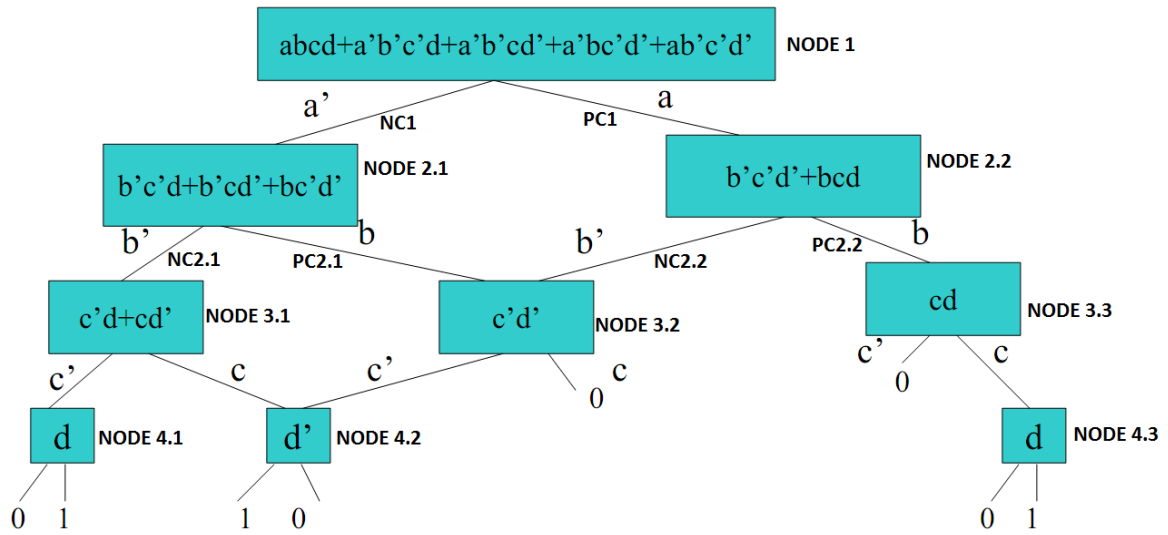
To generate a Shannon lattice for it, the function needs to be decomposed using repeated Shannon's expansion. For every iteration of decomposition, a reduction variable is selected from the variables still present in the function and reduction of function is performed. For instance, for first iteration of this function, 'a' could be selected then both positive and negative co-factors are calculated. For second iteration, 'b' could be selected and reduction is performed on the positive and negative co-factor expressions obtained by reducing 'a' in the first iteration. Similar process can, then, be repeated for variables 'c' and 'd'. Thus the order for the Shannon decomposition of the function  $f$  is  $a \rightarrow b \rightarrow c \rightarrow d$  and function's Shannon Lattice is shown in Figure 3-7(a).

Node  $P.Q$  indicates a node at level  $P$  with serial number  $Q$  left to right.  $NC_{P.Q}$  and  $PC_{P.Q}$  represent negative co-factor and positive co-factor, respectively, of node at level  $P$  with serial number  $Q$ . At level 1 'a' is selected for reduction which produces  $NODE_{2.1}$  and  $NODE_{2.2}$  as negative co-factor ( $NC_1$ ) and positive co-factor

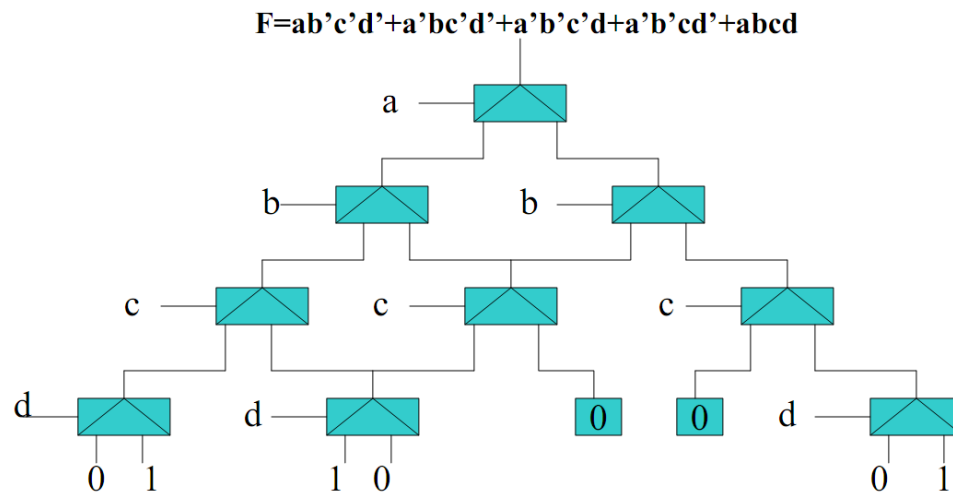
(PC1) respectively, of the second level. For level 2, variable 'b' is selected for reduction and Shannon's expansion is applied for NODE 2.1 and NODE 2.2. The positive co-factor PC 2.1 of NODE 2.1 is equivalent or isomorphic to negative co-factor NC 2.1 of NODE 2.2. Thus, they can be combined and represented by one node NODE 3.2. The lattice diagram in Figure 3-7(a) is implemented with multiplexers in Figure 3-7(b).

Symmetric functions have the following important properties [4, 5]:

1. For two adjacent parent nodes, their adjacent co-factors are always equivalent or isomorphic. This property can be observed in Figure 3-7(a) with nodes: NODE 3.2 and NODE 4.2.
2. No variable needs to be repeated for the complete decomposition of the Boolean function into constants. Figure 3-7(b) shows that each variable was used only at one level in the entire lattice.
3. The number of levels in a Shannon lattice of a symmetric function is independent of the order of variables chosen for decomposition.



(a) Shannon Lattice Diagram of function at NODE 1.



(b) Multiplexer Implementation of Shannon Lattice in figure 3-7(a).

**Figure 3-7 Shannon Lattice Diagram and corresponding multiplexer implementation of Symmetric Function in Section 3.5.1<sup>6</sup>.**

### 3.5.2 Non-symmetric functions

Non-symmetric functions have following properties [7, 8, 30]:

<sup>6</sup> Figure 3-7 Source [29]

1. For two adjacent parent nodes, their adjacent co-factors are **not** always isomorphic.
2. The variables **often need** to be repeated for the complete decomposition of the Boolean function into constants for mapping to Shannon lattice.
3. The number of levels in a Shannon lattice of a symmetric function **generally depends** on the order of variables chosen for decomposition.

The totally symmetric functions presented in section 3.5.1 can be directly mapped to Shannon lattice due to their regularity that arises from merging together isomorphic geometrically-adjacent co-factors. As mentioned above, however, adjacent co-factors might not be isomorphic in arbitrary functions. The idea of merging the non-isomorphic adjacent nodes is extended to non-symmetric functions using a *Join-Vertex operation* presented in [7, 8, 30].

An example of asymmetric function presented in [5] is:

$$f(x_1, x_2, x_3, x_4) = x_1.x_2'.x_4' + x_2.x_3 + x_1.x_2.x_4$$

The corresponding Shannon lattice for this function generated by using Join-Vertex operation is shown in Figure 3-8. For NODE 1.1,  $x_1$  is selected as the reduction variable which produces NODE 2.1 and NODE 2.2. At level 2,  $x_2$  is selected as the reduction variable. The Shannon expansion of NODE 2.1 produces the PC 2.1 as  $x_3$  and Shannon expansion of NODE 2.2 produces the NC 2.2 as  $x_4'$ . Unlike the symmetric function described in section 3.5.1, PC 2.1 and NC 2.2 are not equivalent

thus it is not possible to have either one both as PC 2.1 for NODE 2.1 and NC 2.2 for NODE 2.2 at the same time. In such a case, Join-Vertex operation is applied to PC 2.1 and NC 2.2 which provides a way to represent both with one node NODE 3.1 as shown in Figure 3-8. However, it should be noticed that this operation resulted in the reintroduction of the reduction variable  $x_2$  into node expression. The variable  $x_2$  was selected again for reduction at level 5. Similar observation can be made for NODE 4.2 and variable  $x_3$ . The Shannon lattice below is the result of variable order chosen as  $x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_2 \rightarrow x_3$  but a different order of variables might result in a different lattice diagram. This lattice diagram can be implemented using multiplexers as shown in Figure 3-7(b). In summary, the Join-Vertex operation allows for the merging of non-isomorphic nodes with reintroduction of the reduction variables. These variables can be selected again for decomposition in different orders which could produce different lattice diagrams having different number of levels. Finding the best variable ordering for decomposition in least number of levels is still an area under research.

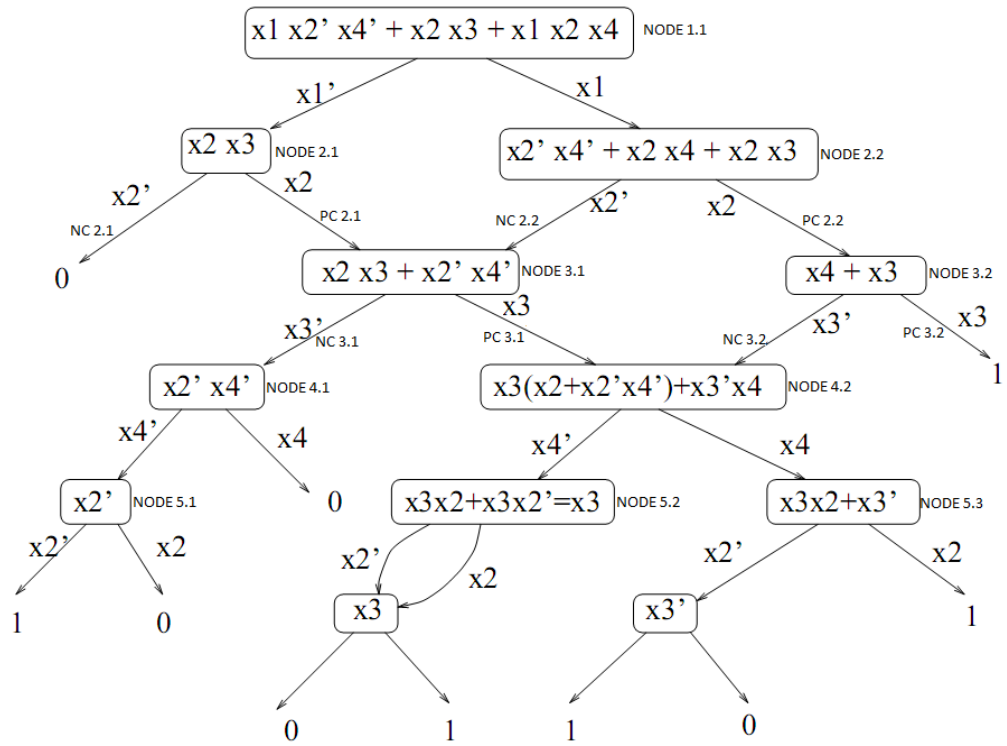


Figure 3-8 Shannon Lattice Diagram of Asymmetric Function in Section 3.5.2<sup>7</sup>.

<sup>7</sup> Figure 3-8 Source [5]



## CHAPTER 4

### QCA CIRCUIT IMPLEMENTATION

#### 4.1 Basic QCA Circuit Design Considerations

Chapter 2 provided detailed background information on QCA technology. However, when it comes to implementing circuits using the QCA building blocks, the following factors play a significant role:

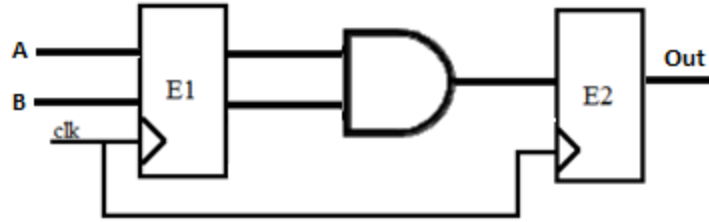
- (i) Signal synchronization
- (ii) Wire latency
- (iii) Basic gates and logic synthesis and
- (iv) Wire length; described in following sections.

##### 4.1.1 Signal synchronization

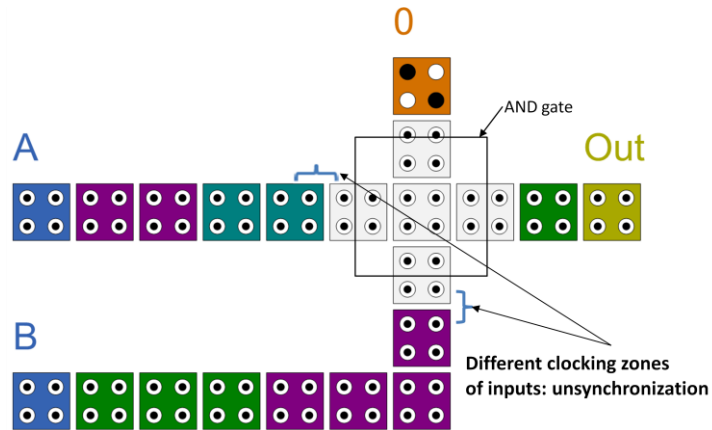
Even though, the signal synchronization is a challenge in circuit implementation in previous technologies as well, it is the granularity of the synchronization in QCA which leads to bigger complications [31]. In traditional technologies, the signals values may not be consistent with the expected values mainly due to the following two reasons (ignoring the second order effects like interconnect capacitance and cross-talk): (i) the timing requirements across the delay elements are not met at circuit implementation level or (ii) the signals go out of synchronization due to misplaced delay elements at the logic or architectural level. If these two conditions are satisfied, then the wire layout should not

affect the information transmission. But in QCA technology, signal synchronization is required at the wire layout level due to the nature of signal flow in QCA. The signal flow in QCA is determined at the layout level by the clocking structure since a group of cells in a particular clocking zone acts as a delay element.

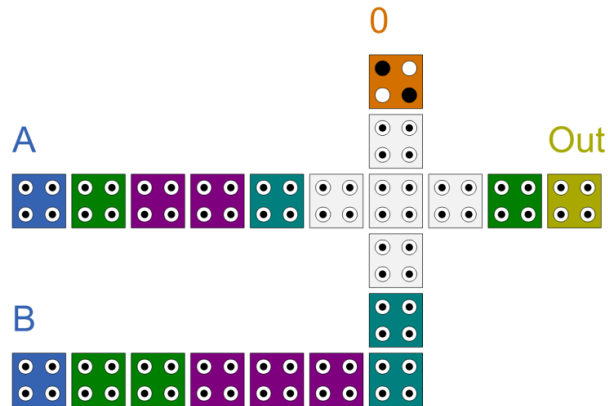
This behavior can be observed by considering AND gates in both QCA and CMOS with signals routed to their inputs. This is shown in Figures 4-1(a) and (b). Figure 4-1(a) shows an AND gate in CMOS with buffered inputs and output. As long as the timing requirements of the delay elements E1 and E2 are met, the output Out would produce the correct value of A & B irrespective of the layout of input and output wires. Figure 4-1(b) shows an incorrect clocking zone assignment for an equivalent circuit in QCA. The signals on which AND operation is to be performed should be in the same clocking zone when they reach the input of the gate. This also implies that a wire transmitting a signal should not violate the clocking zone phase-cycle-order as shown in chapter 2, i.e., *Hold -> Latch -> Relax -> Release*. In other words, all the paths in the circuit should follow this order from input to output. In this particular case of the AND gate, the signal reaches input B quarter a cycle later than the signal reaching input A. Thus the output of the AND gate is not reliable. The correct clock zone assignment is shown in Figure 4-1(c).



(a) AND-gate with Clocked Input and Output.



(b) QCA AND-Gate with unsynchronized inputs due incorrect clocking zone assignment.



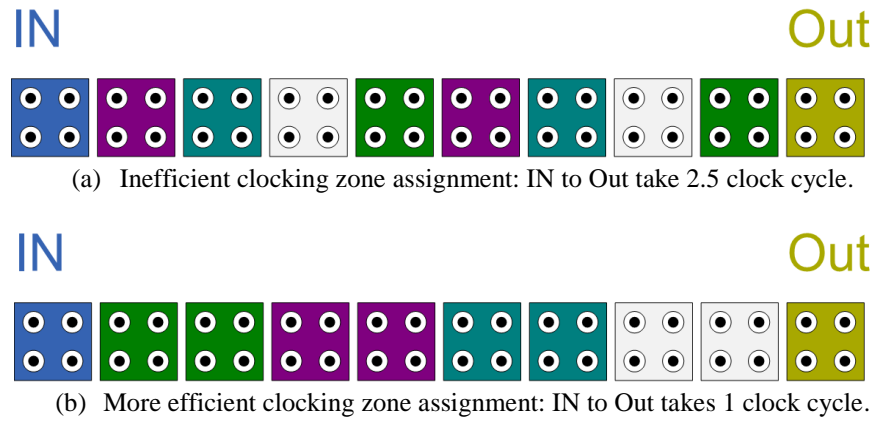
(c) QCA AND-gate with correct clocking zone assignment to input signals.

**Figure 4-1 Signal Synchronization in QCA Circuits.**

#### 4.1.2 Wire latency

In traditional technologies, the latency of short wires is primarily governed by the material properties like resistance, capacitance etc. and likewise the latency of a short QCA wire is defined by the switching time of electrons between quantum dots in a cell. So, for short wires in both technologies, the latency depends on the underlying materials. However, the long QCA wires require them to be divided into multiple clocking zones for signal integrity. Then the latency of the wire becomes the function of the clocking structure i.e. total number of clocking zones that a long wire has been divided into, assuming that the frequency of clocking and electron switching is the same for both short and long wires. A long wire divided into clocking zones is equivalent to back-to-back D flip-flops and thus dividing a wire into more clocking zones than required for maintaining signal integrity, is equivalent to adding more D flip-flops in back-to-back fashion. In such a case, it is not difficult to observe that a wire would require more clock cycles to transfer data from input to output [2]. Figure 4-2 presents examples of assigning the clocking zones to a wire. In Figure 4-2(a), each cell is placed in a different clocking zone which accounts for a total of ten clocking zone for the wire i.e. it would take ten quarter-clock-cycles or 2.5 clock-cycles from a input IN to transfer to Out. Figure 4-2(b) provides a better solution where two cells are grouped in one clocking zone thus giving a total of four clocking zones for the wire which would take five quarter-clock-cycles for information transfer. The clocking zone assignment in Figure 4-2(a) is inefficient compared to assignment in Figure 4-2(b). Figure 4-2(b) is simply used as an example here and does not necessarily show the best clocking assignment, because more number of cells in wire of Figure 4-2(b) can potentially be grouped together in one clocking zone giving a lower number of total clocking zones than shown in Figure 4-2(b). In fact, in

this particular example all the cells could possibly be assigned only one zone since the number of cells is small. In such a case, the information transfer from input to output would only take a quarter-cycle. However, as the number of cells in a wire increases, there is an upper bound on how many cells can be grouped in one clocking zone. This problem is discussed in a later section.



**Figure 4-2 Clocking zone assignment for a QCA-wire.**

#### 4.1.3 Basic gates and logic synthesis

In previous technologies, the logic circuits were generally defined in SOP or POS form and implemented using AND and OR gates. In QCA, the logic primitives are majority gate and inverter. The authors in [32] proposed a Boolean algebra based interpretation to convert the sum-of-product expressions into the reduced majority logic. Their results show that synthesizing logic into majority gates for QCA implementation results in

smaller circuits than implementing QCA circuits based on their SOP or POS representations.

#### **4.1.4 Wire length**

As in CMOS/silicon designs, the long wires present concerns in QCA technology as well. In semiconductor domain, the long wires are provided gain using repeaters/buffers to restore the signal strength and also to maintain rise-fall times [33]. A long QCA wire also requires similar treatment in term of power gain. The power gain in QCA wire is provided by the clock. According to [20] and [23], as the number of cells in a clocking zone increases, their successful switching becomes less reliable. It is, therefore, desirable that number of cells in a wire in a clocking zone is kept to minimum. Readers are encouraged to refer to [18, 23] for a detailed reasoning of the subject matter. Some clocking strategies have been proposed to deal with this issue in [23]. These are described in section 4.2.

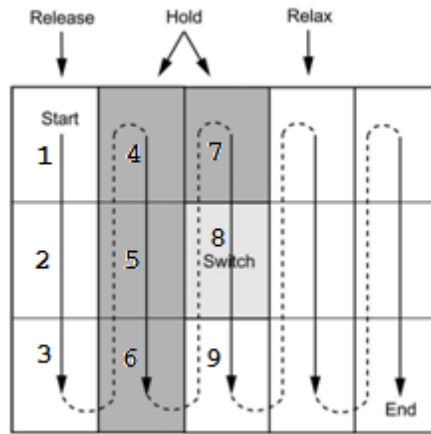
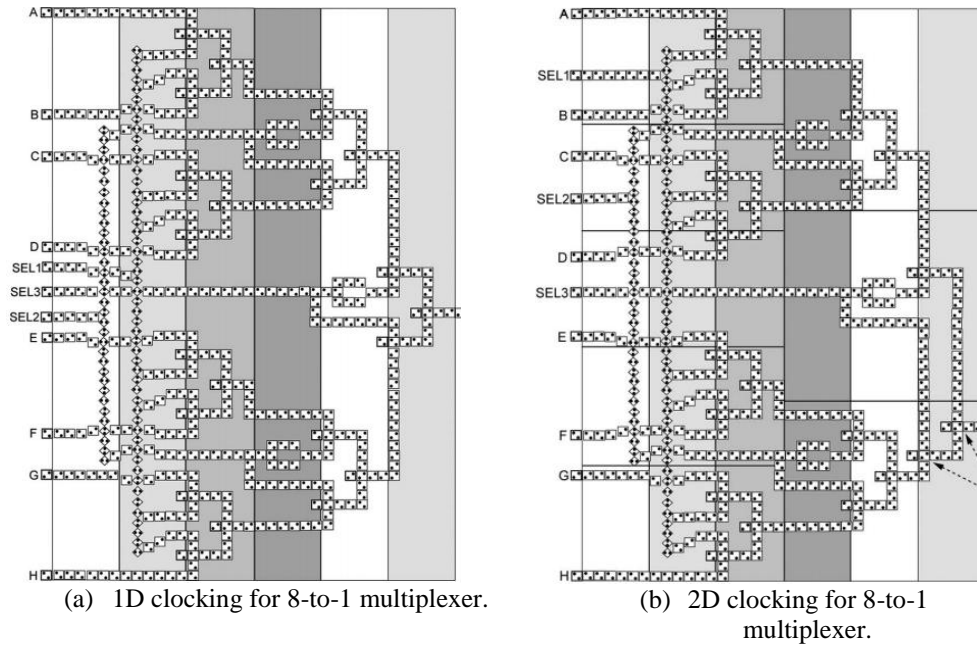
### **4.2 Wire Length and Clocking Strategies**

To alleviate the problem of wire length as described in section 4.1.4, other proposed clocking mechanisms are presented in this section. The PLA design used in this thesis also has long wires and thus it is important to gain the understanding of these schemes since similar scheme has been used for PLA clocking.

#### 4.2.1 2-Dimensional (2D) clocking scheme

Figure 4-3(a) shows a 8-to-1 multiplexer built using three stages of 2-to-1 multiplexers. The whole design is divided vertically into several clocking zones. The different shades of grey-color represent different clocking zones with a phase difference of  $\Pi/2$ . It can be observed that the first two clocking zones have long vertical arrays of cells which might produce unreliable signal values. The authors in [23] propose to solve this issue by partitioning the design along the  $y$ -axis in addition to the  $x$ -axis (which is already present). The authors call this scheme *2D or 2-dimensional* clocking and the arrangement is shown in Figure 4-3(b). The length of a QCA wire in a zone is restricted by the dimensions of a zone. In this scheme, the switching pattern of the zones would be first top-to-bottom, row-wise within a column and then column-wise, left-to-right. In other words, instead of having the whole column switch at once, portions of it are allowed to switch and then held in *hold-phase* until the whole column has switched. This corresponds to switching of the whole column at once. Thus, the cells in the next clocking zone should would see the cells in the previous clocking zone in the same state as they (previous-clocking-zone-cells) would have been after switching with 1D-clocking scheme i.e. all switching at once. This is equivalent to sequential processing in linear fashion. This makes 2D clocking more complicated than 1D-clocking since a finer control is needed on the clocking phases of the zones. Implementing 2D-clocking scheme would, however, require some modifications to the QCA design for synchronization purposes because signals reach the final computation cells at different

times. This can be observed from alterations in the placements of the final multiplexer's output and input select lines in Figures 4-3(a) and (b) [23].



(c) Signal propagation in 2D-clocking.

**Figure 4-3 1D and 2D Clocking Strategies for QCA Circuits<sup>8</sup>.**

<sup>8</sup> Figure 4-3 Source [23]



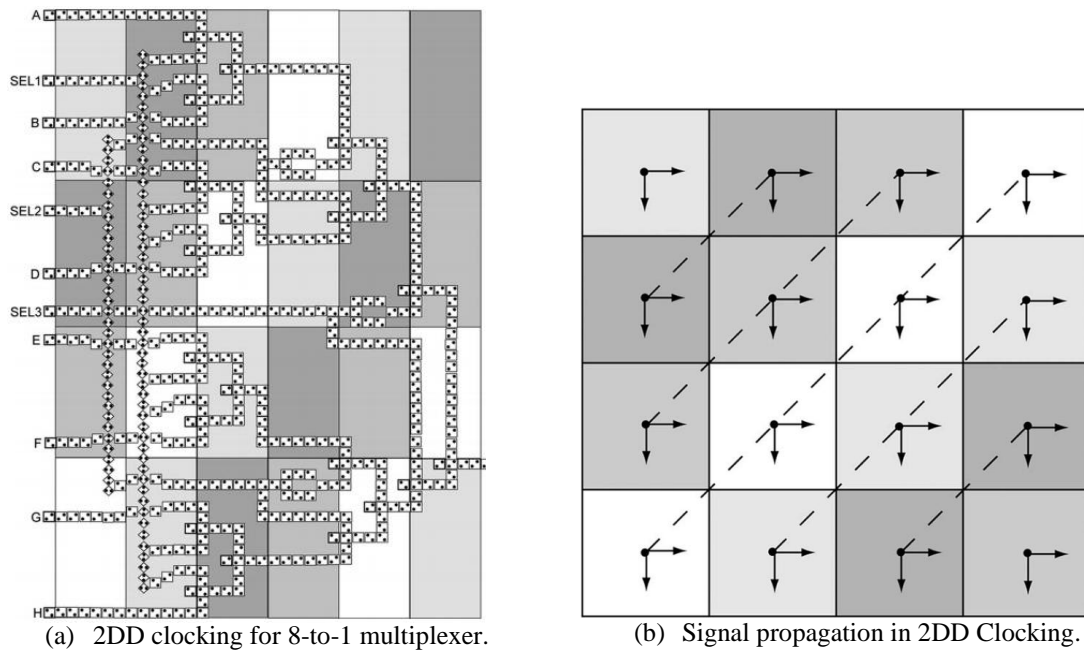
The signal propagation for 2D-clocking is shown in Figure 4-3(c) which corresponds to the Figure 4-3(b). Without the loss of generality, for 2D clocking scheme, a vertical clocking zone can be divided into multiple sections depending on the wire length even though figure 4-3(c) illustrates the flow for division of clocking zones into three sections. The signal propagation (i.e. the process of applying clock to cells that puts them in *switch* phase) starts at section 1 in Figure 4-3(c). Since the number of cells in section 1 is less than the total number of cells in entire left-most vertical clocking zone of Figure 4-3(a), all of the cells in section 1 should switch successfully. After these cells switch, the clock is changed to put them in the *hold* phase and cells in section 2 are put in the *switch* phase. With the same assumptions as earlier, the cells in section 2 should switch successfully. Now the cells in section 2 are put into the *hold* phase along with the cells in section 1 and the cells in section 3 are put in the *switch* phase. Once the cells in section 3 have switched, the same process is repeated sections 4 to 6 while keeping section 1 to 3 in the *hold* phase. It can be observed that this process of switching cells in Figure 4-3(c) results in switching of cells in left-most column based on the inputs and switching of cells in next column based on the signals from left-most column. This whole process is equivalent to switching the entire left-most vertical column and then the next column of Figure 4-3(a) while avoiding the problem of long wires. When starting to switch the cells in sections 7 to 9, keeping cells in section 4 to 6 in hold phase, the cells in sections 1 to 3 can be put in *release* phase (next in order after *hold* phase) since only sections 4 to 6, which are *hold* phase, are required for sections 7 to 9 to switch. This type of clocking strategy would result in a different signal flow as compared to signal flow when whole column is switched at one as in Figure 4-3(a). For instance, for right-most column in

Figure 4-3(a), the bottom wire propagates the signal up while the top wire propagates the signal top-down. With the 2D clocking scheme, the cells in the right-most-bottom-most section of Figure 4-3(b) are the last ones to switch. If same circuit structure as in Figure 4-3(a) is kept with 2D clocking scheme, the bottom wire would be the last one to switch and would switch after the right-most final output cell. This is an incorrect behavior because the computation would be performed on the signals from different time instances. This problem can be resolved by re-synchronizing the signals for final computation cell. This requires modifying the circuit of Figure 4-3(a) to move the final output into right-most-bottom-most section of Figure 4-3(b)

#### **4.2.2 2-Dimensional-Diagonal-Wave (2DD) clocking scheme**

Unlike 2D clocking, which requires the sequential clocking in one column and then next column, the 2DD-clocking mechanism allows for parallel switching of the clocking zones located along the diagonals of the design. This provides gains in terms of throughput as compared to the 2D-clocking while still having a bound on the wire-length in a clocking zone. Thus the computation time with 2DD-clocking is shorter than both 1D and 2D schemes (which are essentially equivalent). The clocking-zone assignment for 2DD clocking is shown in Figure 4-4(a) and the corresponding signal propagation is shown in Figure 4-4(b). Like 2D-clocking, some circuit modifications are required in order to synchronize the computation with the signal flow in 2DD clocking scheme as well. It can be observed from different placements of the final multiplexer's output and the input select lines in Figures 4-3(a) and 4-4(a). The signals from the top and bottom in Figure

4-3(a) reach the final output at the same time because the entire vertical column is switched at once, however, due to diagonal-clocking approach in 2DD-clocking scheme, the signals at the top are clocked before the bottom signals and thus are ahead in time than bottom signals. For this reason, in a circuit without modification, the signals reach the final computation cell in unsynchronized fashion. To resolve this issue, the final multiplexer output is stretched to pass through additional clocking zones [23].



**Figure 4-4 2D-Diagonal clocking for QCA circuits<sup>9</sup>.**

Although, the 2D-clocking and rearrangement of logic solved the problem of unreliable long wires and signals becoming unsynchronized (due to different clocking approach) respectively, it resulted in the deviation from the regular and uniform design of Figure 4-3 (which uses same kind of 2-to-1 multiplexers throughout) to irregular design of Figure

<sup>9</sup> Figure 4-4 Source [23]

4-4 using different multiplexers to account for clock offset. In later sections, a variant of 2DD-clocking is adapted for the clocking assignment of QCA-PLA, which is able to maintain the regular structure of PLA while dealing with the issues of the long-wires.

### **4.3 QCA Circuits**

Having built a foundation about the basics of QCA in chapter 2 and the QCA circuit design considerations in previous sections of chapter 4, this section presents example QCA circuits. This section also shows the circuits which are used as standard library macro-cells to generate complete regular layout by the layout tool developed in this thesis. QCADesigner simulator [14] was extensively used in this research work for designing and simulating the circuits. It is very important to note that the QCA circuits or their clocking assignments illustrated in this section may not necessarily represent the best implementation for a particular circuit. However, the designs presented below are chosen, from a pool of other designs considered and simulated using QCADesigner tool, based on following five design-constraints:

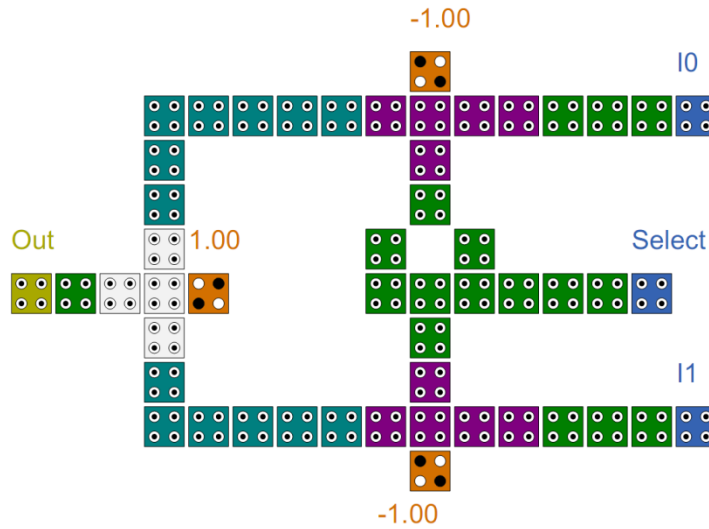
1. An acceptable switching behavior and signal level of important QCA-cell at key positions in the circuit.
2. A minimum size/area and number of required clocking zones when simulating with QCADesigner while satisfying constraint 1.
3. A minimum array of three QCA-cells in one clocking zone and maximum defined by constraint 1.

4. A minimum cell spacing of two cells between two wires satisfying constraint 1.
5. A minimum number of QCA-cell in the design while satisfying the above constraints.

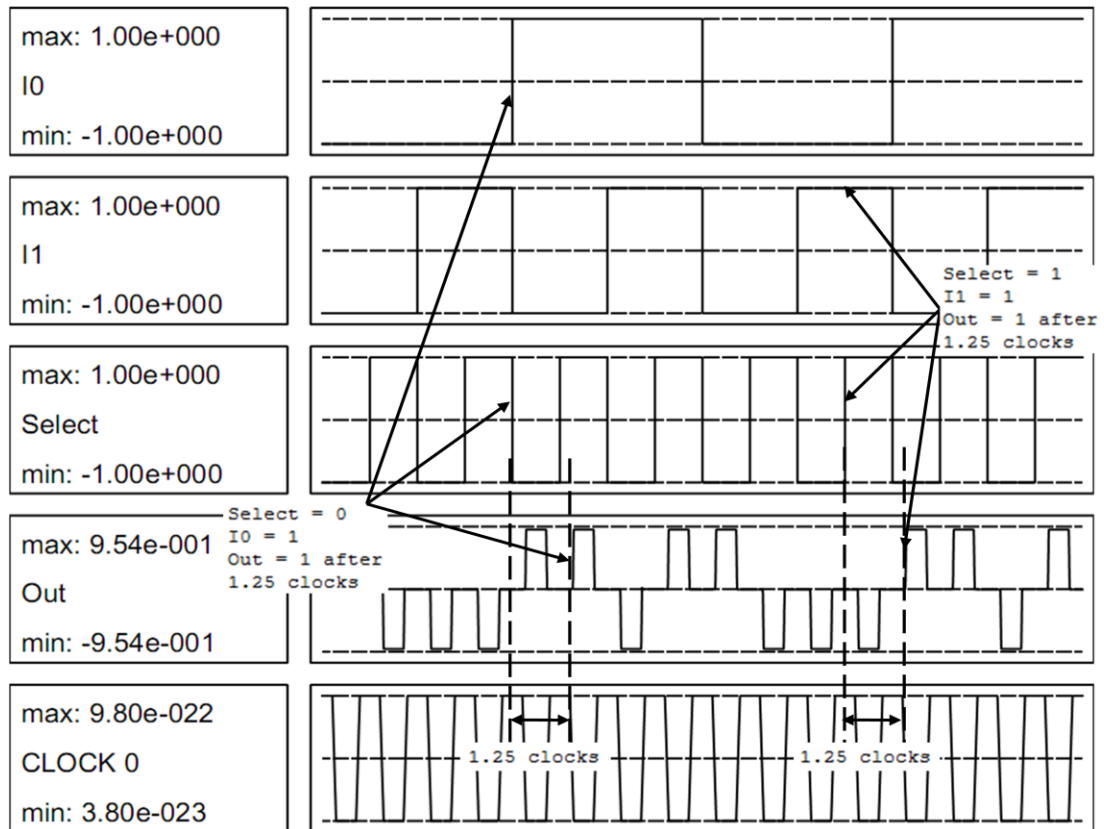
### 4.3.1 QCA Multiplexer

Figure 4-5 (a) shows a multiplexer built using majority gates configured as AND-gate and OR-gate. It can be seen that for both AND gates the input signals `I0/~Select` and `I1/Select` are in the same clocking zone throughout and hence, synchronized. The signals reach the inputs of respective gates at the same time and the computation is performed on the signals from the same time instant. Likewise, the inputs to the OR-gates are in the same clocking zone when they reach the gate and hence synchronized. After the OR-gate the output cell, `Out`, and a previous cell are in a different clocking zone. It is advised to keep the majority gates in separate clocking zones than their inputs, which is evident from the layout below. The whole design required to be divided into 5 clocking zone horizontally, so the design has the latency of one and a quarter cycle from input to output. The simulation results for the QCA-multiplexer in Figure 4-5(a) is shown in Figure 4-5(b) which are obtained using QCADesigner simulator. A variant of this design is used as a cell by the layout tool for generating Shannon lattices for input Boolean functions. The logic equation for multiplexer is given as:

$$\text{Out} = \text{I0}.\overline{\text{Select}} + \text{I1}.\text{Select}$$



(a) QCA 2-to-1 multiplexer.  
Simulation Results

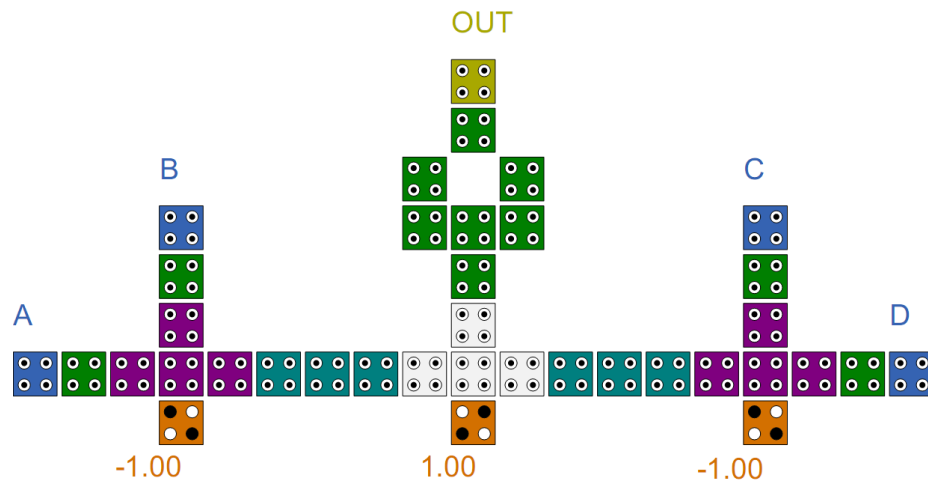


(b) Simulation waveforms of 2-to-1 QCA multiplexer generated using QCADesigner.

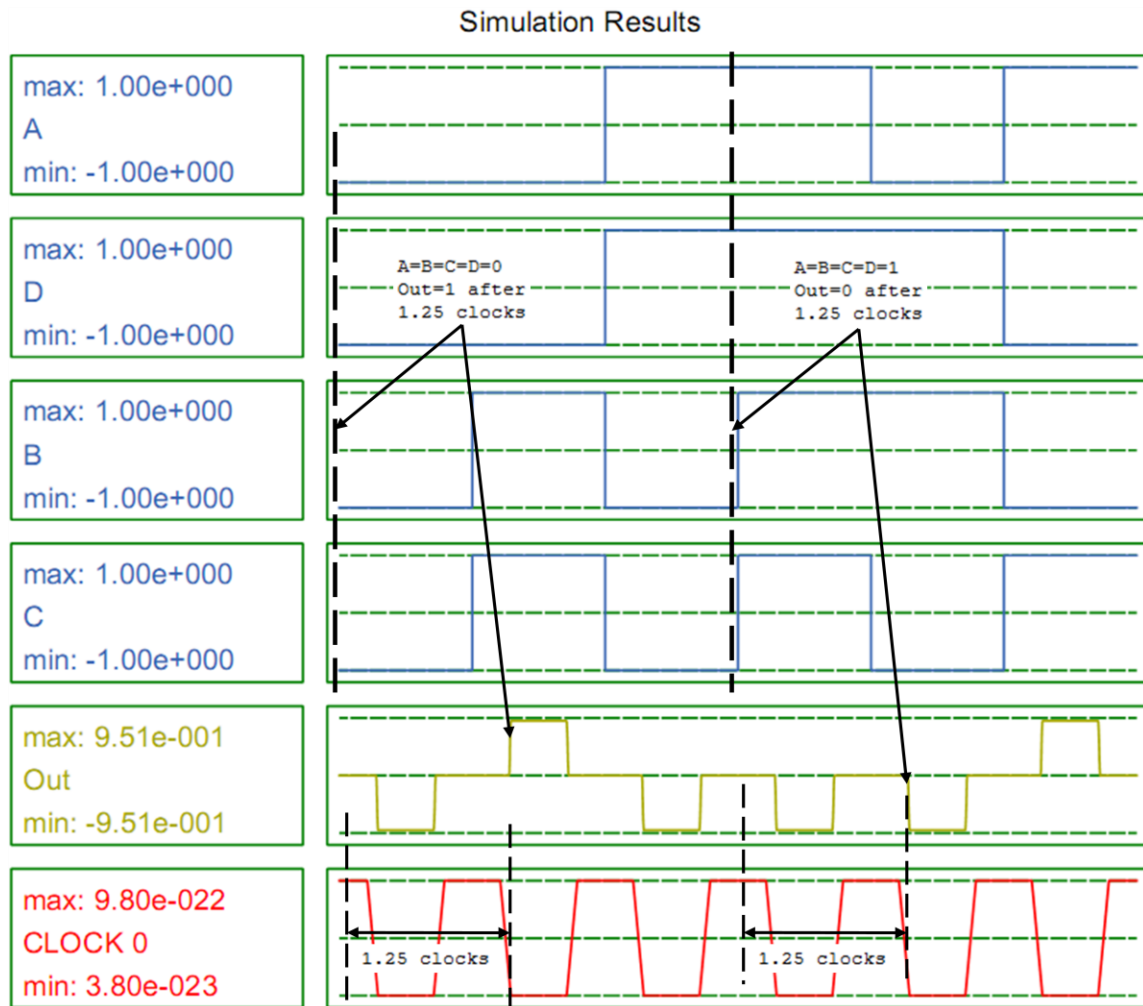
**Figure 4-5 QCA 2-to-1 Multiplexer Design and Simulation Waveforms.**

### 4.3.2 QCA AND-OR-INVERT (AOI) Logic

Figure 4-6(a) shows an AOI gate that implements a function  $\overline{(A.B + C.D)}$ . Due to the way clocking zones are assigned, A, B, C and D signals (in zone 0) will propagate towards their respective AND-gates which is assigned zone 1. The outputs of both AND-gates are directed towards the OR-gate in the center-bottom of the circuit using the wires assigned zone 2. The OR-gate in zone 3 performs the operation when it is in *switch* phase which is immediately a quarter-cycle after zone 2 is in *switch* phase. A quarter-cycle after its *switch* phase, the OR-gate goes into *hold* phase and the inverter is in *switch* phase and inverts the output of OR-gate. This gate also takes one and a quarter-cycle to generate the output corresponding to the inputs. . The simulation results for the QCA AOI logic of Figure 4-6(a) is shown in Figure 4-6(b) which are obtained using QCADesigner simulator. The logic function implemented by the circuit is:  $Out = \overline{(A.B + C.D)}$



(a) QCA AOI logic.



(b) Simulation waveforms of AOI logic generated using QCADesigner

**Figure 4-6 AND-OR-INVERT (AOI) Logic and simulation waveforms.**

### 4.3.3 QCA Full Adder

The QCA layout of full-adder shown below is taken from [34]. This design contains long vertical wires which could potentially result in unreliable signal values. The simulation of this circuit in QCADesigner does not always produce accurate results most likely due



to long wires. The logic equations for output functions for the QCA full-adder shown below are:

$$\text{Cout} = \text{Majority}(A, B, C)$$

$$\text{Sum} = \text{Majority}(\overline{\text{Cout}}, C, \text{Majority}(A, B, \overline{C}))$$

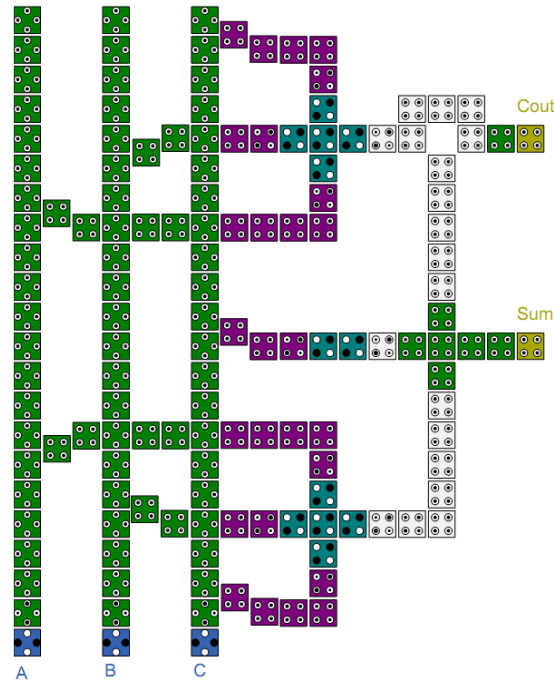


Figure 4-7 QCA Full-Adder<sup>10</sup>.

#### 4.3.4 QCA D-Latch

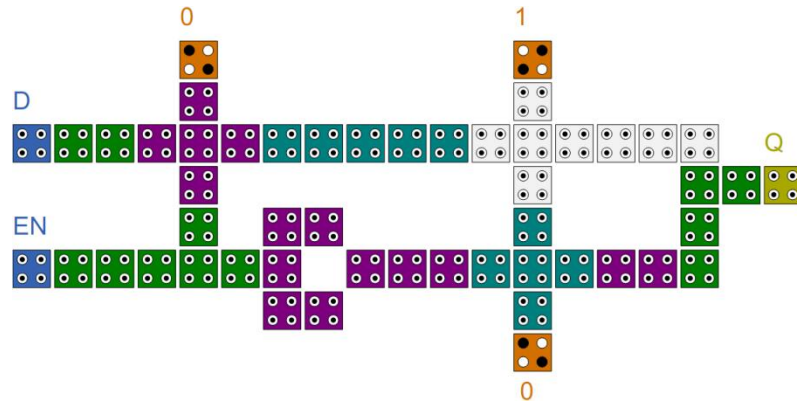
Delay elements can also be created in QCA as shown in Figure 4-8 taken from [35]. When EN is 1, the input to lower AND-gate is 0 which causes its output to be 0. This output of AND-gate at the bottom is the input to the OR-gate. With EN as 1, the output

<sup>10</sup> Figure 4-7 created using QCADesigner based on [34]

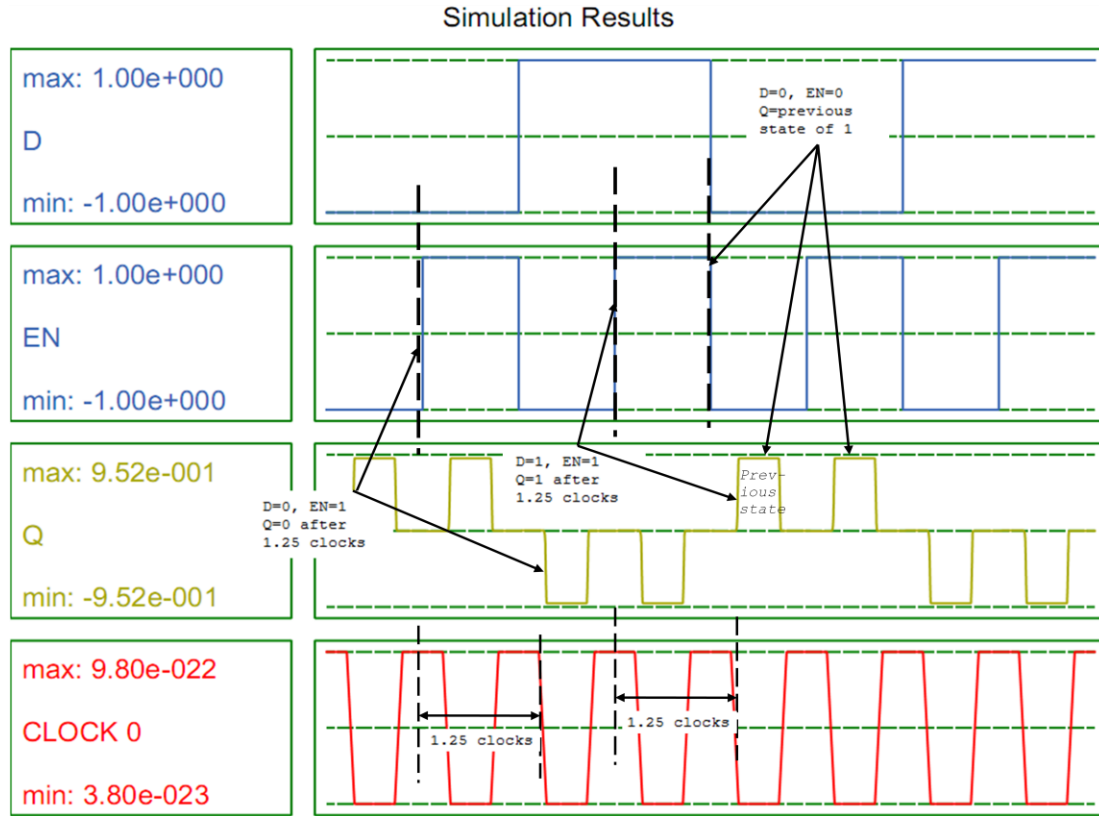
of top-left AND-gate is equal to  $D$  which acts as another input to the OR-gate. Thus the OR-gate, with one input as  $D$  (from left) and other as 0 (from bottom), produces output equal to  $D$  which goes to  $Q$ . When  $EN$  becomes 0, the two inputs to the OR-gate are 0 (from left) and  $Q$  (from bottom) which produces an output equal to  $Q$  which again goes to output cell  $Q$  and hence the feedback to the previous stored state. The operation of D-latch is defines as follows:

$$Q = D \text{ when } EN = 1$$

$$Q = Q \text{ when } EN = 0$$



(a) QCA D-Latch.



(b) Simulation waveforms of QCA D-Latch generated using QCADesigner.

**Figure 4-8 QCA D-Latch and simulation waveforms.**

### 4.3.5 QCA PLA

Realizing a Boolean function with Shannon Lattice requires only one type of cell, multiplexer, but in case of a PLA two types of regular cells is required to create the entire AND-OR plane. The replication of AND-cell creates the AND-plane and OR-cell creates the OR-plane. The systematic placement of both the planes together creates the PLA layout. The QCA-PLA layout in this work is directly based on the design presented in [36] for functions with multiple outputs where logic can be shared. For this research,

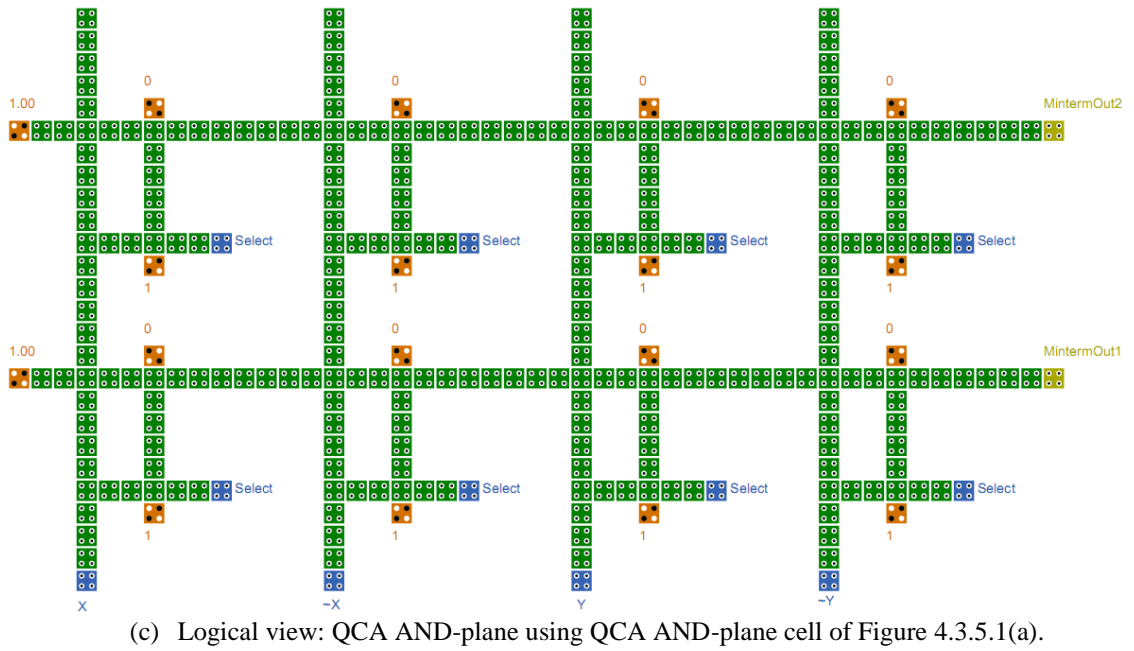
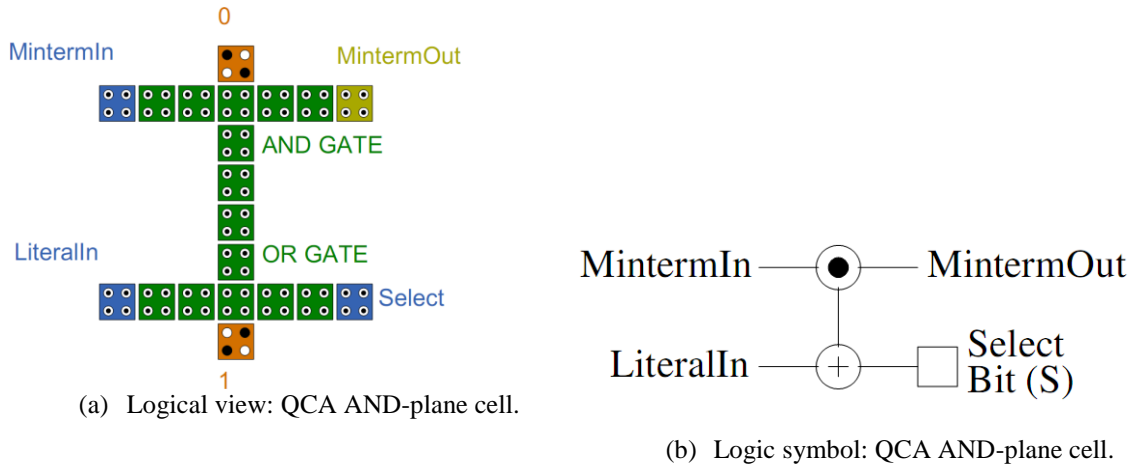
only single output functions have been considered and thus a modified OR-plane is used. The AND-plane cell and OR-plane cell from [36] are shown and their operation is described below.

#### 4.3.5.1 QCA PLA: AND-Plane Cell

For PLA cells are designed to contain a programmable select bit which is used to program a cell to work either as a logic gate or as a wire. Figure 4-9(a) shows the AND-plane cell as QCA layout and its equivalent logic symbol in (b). The AND-plane in a QCA generated the product terms by performing logical AND on the inputs. In Figure (a) shown below, the two inputs to the AND-gate are the `LiteralIn` from the bottom and `MintermIn` from the left. The value of `Select` cell determines if the AND-gate would perform the logical AND of two or simply let the `MintermIn` pass to `MintermOut` making AND-gate act as a wire. The functionality of the AND-cell can be defined as follows:

$$\text{MintermOut} = (\text{LiteralIn}) \cdot (\text{MintermIn}) \text{ when } \text{Select} = 0$$

$$\text{MintermOut} = (1) \cdot (\text{MintermIn}) = (\text{Minterm}) \text{ when } \text{Select} = 1$$



**Figure 4-9 PLA: AND-plane<sup>11</sup>.**

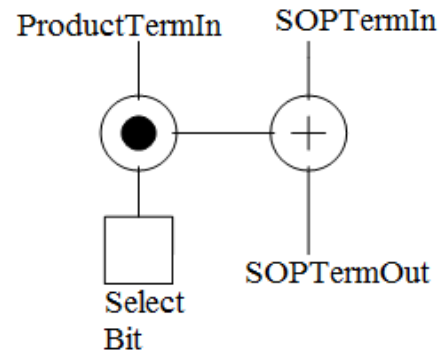
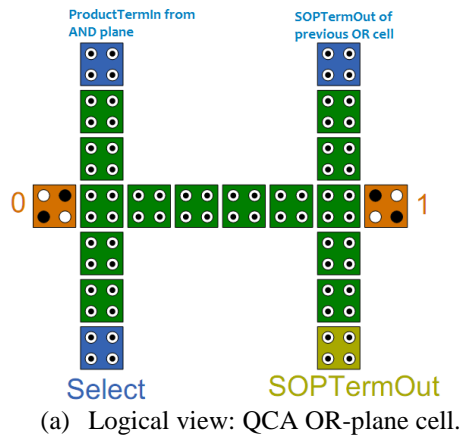
The MintermOut of AND-plane cell, say  $x$ , acts as the MintermIn input of the next cell and LiteralIn, say  $y$ , for the next cell would be driven by the variable wire,  $y$ , of that AND-cell. If the Select of next AND-plane cell is 0 i.e. literal selected, then the

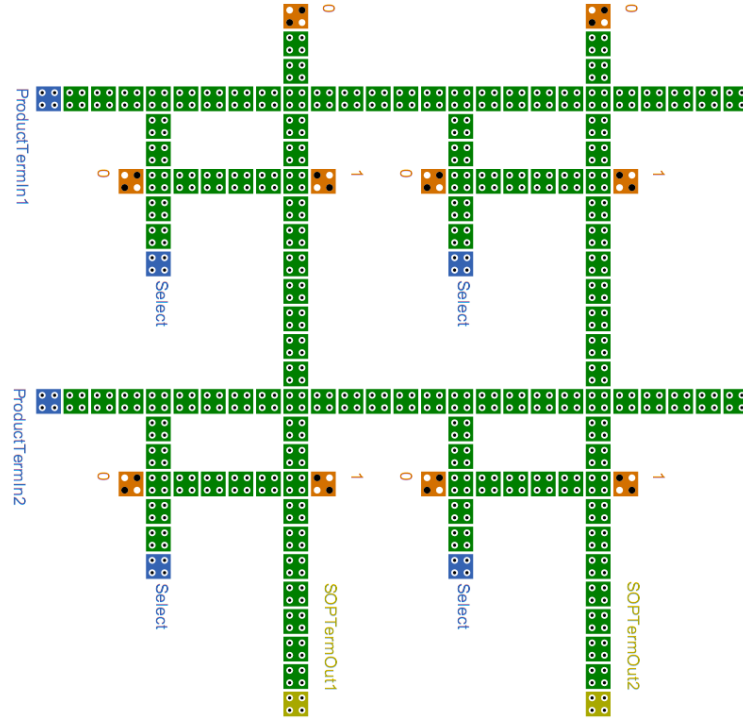
<sup>11</sup> Figure 4-9 created using QCADesigner based on [36]

MintermOut of next cell would be  $x \cdot y$  implementing the product term. Such an arrangement would generate a product term with both positive literals and in order to generate terms that contain negated variables/literals, AND-plane cell can be replicated horizontally as an array with LiteralIn input driven alternately with literal and its complement. This would still allow creating only one min-term/product-term. To generate several product-terms, multiple levels of the same horizontal structure can be created with a long variable wire driving LiteralIn at each level. This is illustrated in Figure 4-9(c) above.

#### 4.3.5.2 QCA PLA: OR-Plane Cell

The structure and operation of OR-plane cell is very similar to the AND-plane cell. Rotating AND-plane cell clockwise by 90-degrees creates an OR-plane cell and the cell operates as a logic wire when select bit is 0 else as an OR-gate with select bit is 1.





(c) Logical view: QCA OR-plane using QCA OR-plane cell of Figure 4-10(a).

**Figure 4-10 PLA: OR-plane<sup>12</sup>.**

### 4.3.5.3 QCA PLA Logic Diagram

Connecting the AND-OR planes from the previous two sections, the entire PLA layout can be obtained. The logic diagram for a QCA-PLA realizing XOR and OR functions is shown in Figure 4-11. The following is taken from [36]. There design issues in implementing QCA-PLA by integrating AND-plane and OR-plane design as presented in earlier section. These design issues, their solution and potentially implementation QCA-PLA design is presented in a later section 4.4.

<sup>12</sup> Figure 4-10 created using QCADesigner based on [36]

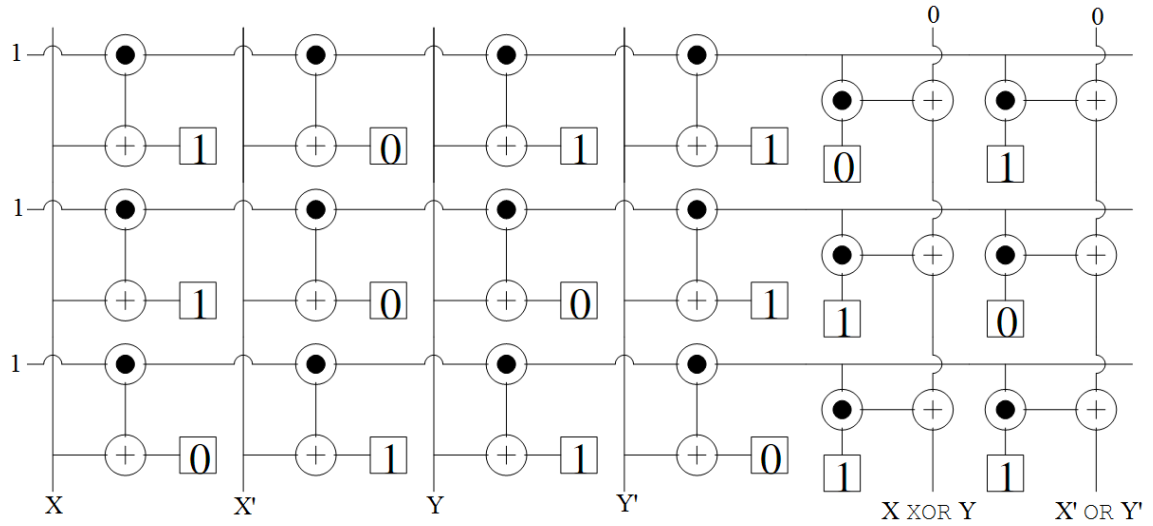


Figure 4-11 Logic diagram: QCA PLA implementing XOR and OR logic<sup>13</sup>.

#### 4.4 Macro Cells for Regular Layout

This section illustrates the QCA design/layout and clocking assignments of the cell which are used as the building blocks by layout generator tool to create the complete layout of input Boolean function.

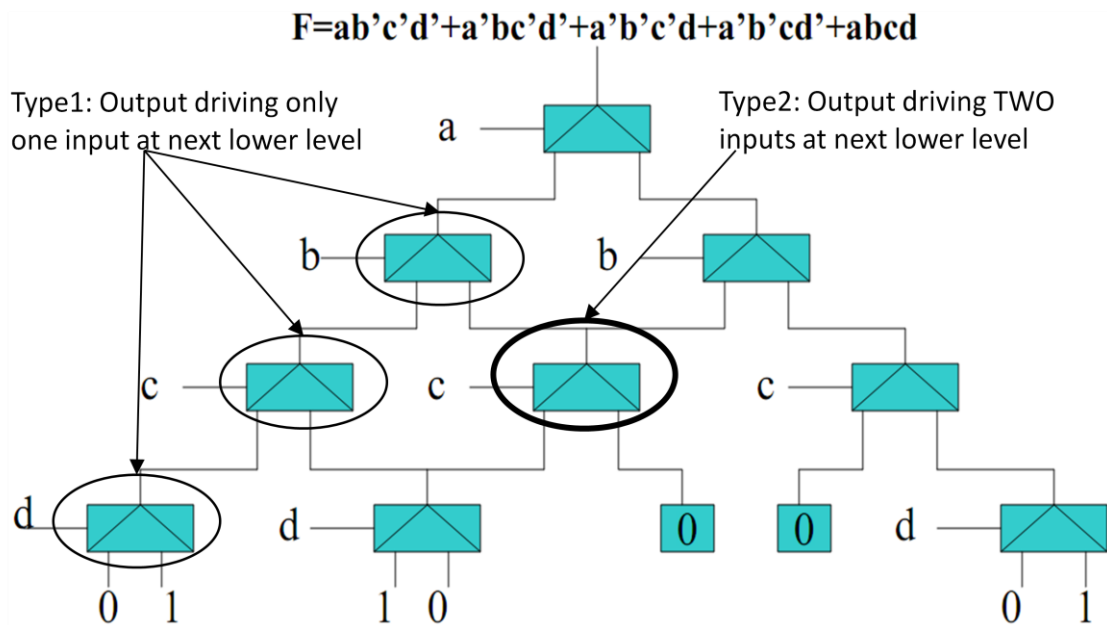
##### 4.4.1 Multiplexer-Cell: QCA-Shannon-Lattice

As described in chapter 3, the Shannon Lattice of a Boolean function can be realized by a lattice structure of multiplexers corresponding to the Shannon Lattice Diagram. The multiplexers can be abutted together without the need of any routing between them to create the lattice. This requires a multiplexer layout such that it can be cloned at every

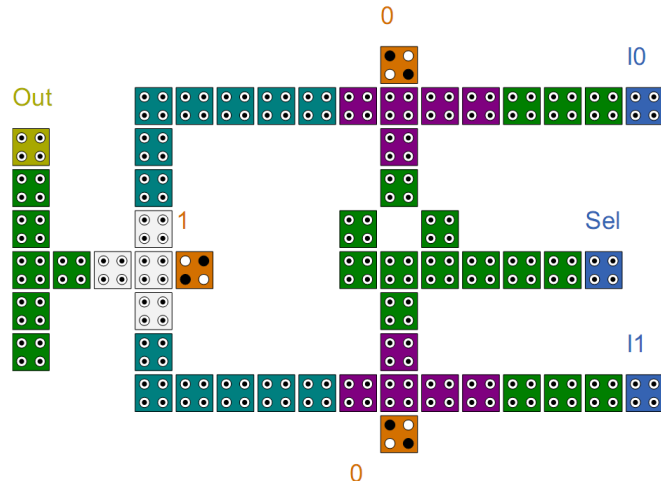
<sup>13</sup> Figure 4-11 Source [36]



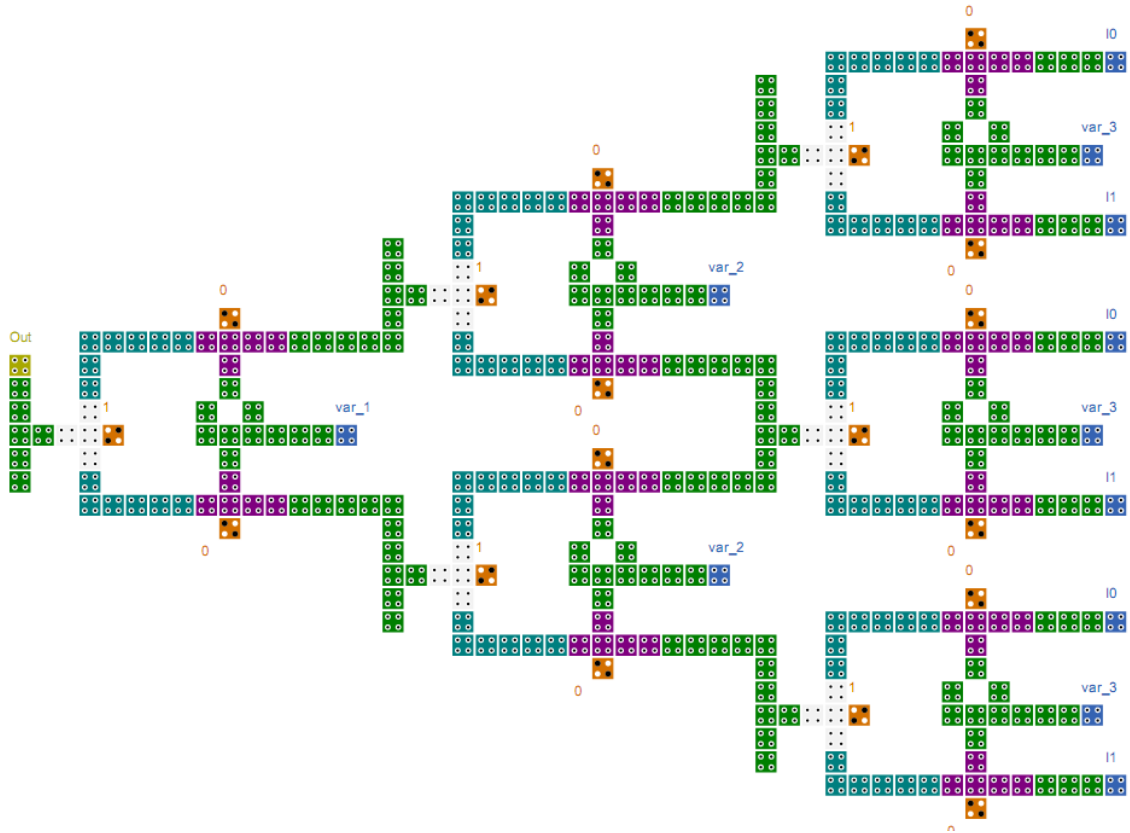
level depending on the lattice diagram. Analyzing the Shannon Lattice shown in Figure 4-12(a), it is easy to see that it uses two types of multiplexers; one with an output driving only one input at next lower level (*top-to-bottom is low-to-high level*) and other with an output driving two inputs at next lower level as shown in the Figure (a). This difference in the multiplexer-types requires a QCA-multiplexer layout which is equivalent to both types of multiplexers and still allows for replication and abutting. Such a multiplexer would be used for all the nodes producing a regular layout. The layout for such a multiplexer cell is shown in Figure 4-12(b) created by a very slight modification of multiplexer of Figure 4-5 above. This is similar to designing a standard macro-cell for the standard cell library in the previous silicon-based technology. The multiplexer cell layout file is used by the layout generator tool as an input.



<sup>14</sup> Figure 4-12(a) Source [29]



(b) Multiplexer-cell layout and clocking for regular layout.



(c) QCA-Shannon-Lattice for arbitrary function.

**Figure 4-12 QCA Multiplexer macro-cell and QCA-Shannon-Lattice: simulated in QCADesigner to verify functionality.**

For the QCA multiplexer in Figure 4-12(b), the *Sel* input corresponds to a variable at a level, for example: *a*, *b*, *c* or *d* from Figure 4-12(a). *I0* and *I1* are the two inputs for a multiplexer at any level. Figure 4-12(c) shows the complete QCA Shannon Lattice for an arbitrary function to verify the fact that multiplexer cell of Figure 4-12(b) can be used for both Type1 and Type2 multiplexers from Figure 4-12(a) and still provide the correct functionality. Generally, a lattice diagram or tree is drawn from top to bottom as in Figure 4-12(a) but for this work the QCA Shannon Lattice are generated from left to right which is simply an orientation factor with same functionality. The tool can generate a Shannon Lattice from top to bottom by changing few input parameters. It should be noticed that clocking zones are assigned such that signals flow from right to left in a synchronized fashion. The layouts similar to one shown above were simulated for this thesis in QCADesigner simulator to verify them for correct operation. The analysis on latency and throughput of the QCA Shannon Lattice circuits is presented in later sections.

#### **4.4.2 AND & OR-Plane Cells: QCA-PLA**

The logic level view of QCA-PLA structure discussed in section 4.3.5 is presented in [36]. However, the paper [36] or any other source does not discuss the specific details of layout and clocking zone assignment. Without a clear understanding of these issues, it is difficult to create a layout that can be simulated in QCADesigner simulator.

1. The paper [36] proposes to use diagonal clocking (discussed in section 4-2 ) but does not state the issue and solution of the problem of product terms reaching the outputs of AND-plane in an unsynchronized fashion due to diagonal clocking.
2. Another proposal in the paper [36] is to use the global clocking method in which the entire AND-plane is clocked together and same for OR-plane. However, it is not clear how such an approach would handle the signal integrity problem for long wires as discussed in section 4.2 of this thesis.

In this thesis, a complete PLA design with clocking assignment and regular clocking structure is presented that can be easily simulated in QCADesigner. This requires slight modifications to the designs of AND-OR plane cells which are described in the following sub-sections.

#### **4.4.2.1 AND-plane cell**

The simplified AND-plane shown in Figure 4-9(c) does not conform to the guidelines for designing a correct functioning QCA circuit. The following conditions need to be accounted for:

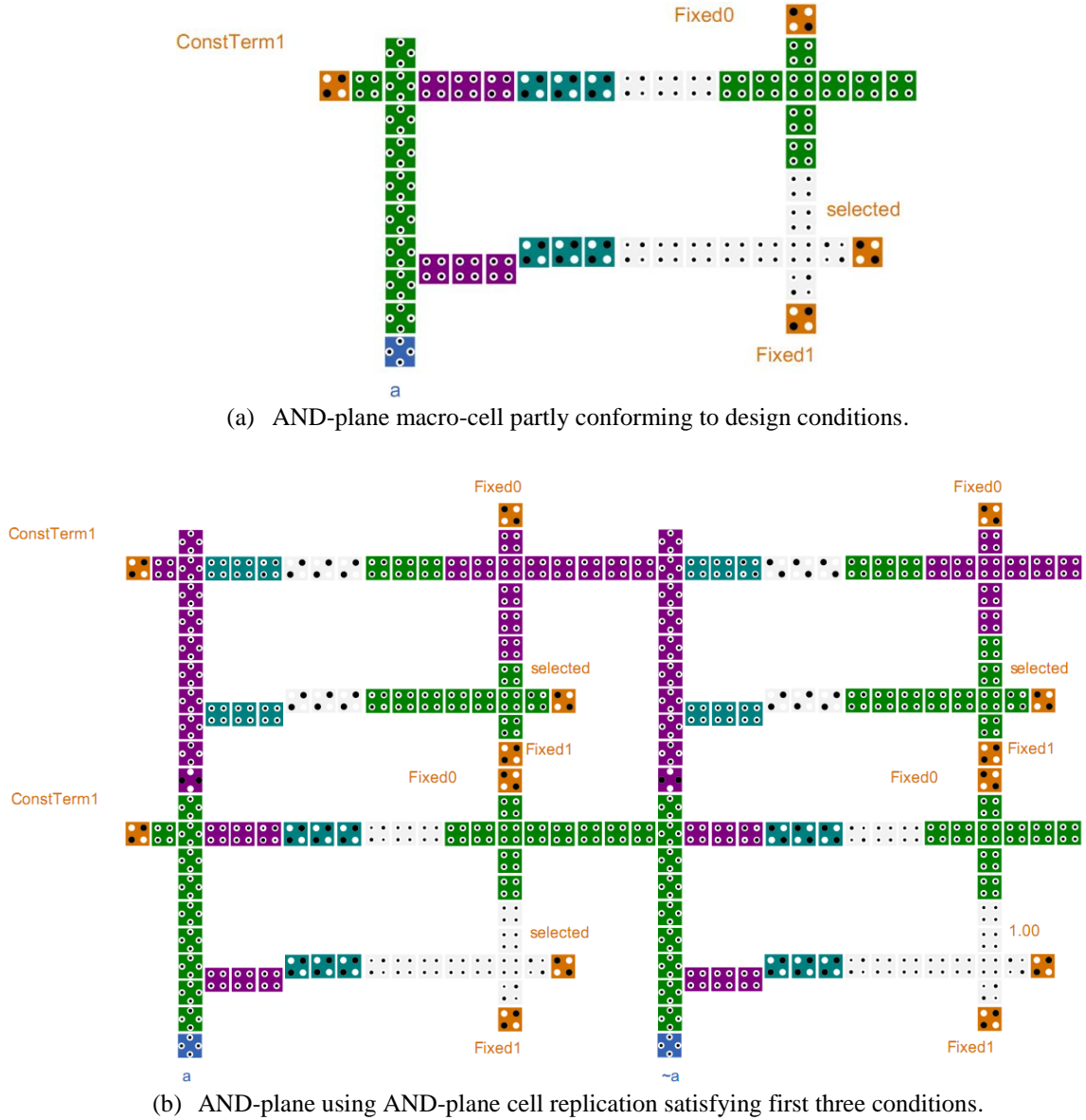
- I. AND-plane contains long wires both vertically and horizontally which should be divided into different clocking zones.
- II. The vertical variable/literal wire crosses the horizontal min-term wire; this is only possible when one of the wires is implemented with rotated cells.

- III. Since the horizontal min-term wires need to be divided into clocking zones, the output of previous AND-plane cell should be synchronized with the input of next AND-plane cell literal. In other words, the output of an AND-plane cell *A* should reach the input of AND-gate of next AND-plane cell *B* in synchronous fashion with literal input of AND-plane cell *B*, to generate correct SOP term.
- IV. The division of long vertical wires into separate clocking zones results in min-terms reaching the OR-plane in an unsynchronized fashion. In order to perform logical-OR on the term, they should be in the same clocking zone. The clocking zone offset needs to be fixed and these signals must be synchronized when they reach the OR-gates in OR-plane.

The layout and the clocking zone assignment of AND-plane cell should be such that it would satisfy the above requirements and still could be replicated to produce the entire plane. Figure 4-13(a) shows an example AND-plane macro-cell layout which obeys conditions from above: condition [I] partially and condition completely [II] from the above. The input *a* is driven in bottom-to-up direction while the logic flow is from left-to-right and macro-cell output is the right-most QCA-cell in normal mode. The `ConstTerm1` is always logic 1 for the very first AND-plane macro-cell because there is no macro-cell before it to send a literal for logical-AND. Thus, the output of the first AND-plane macro-cell would be equal to the logical-AND of 1 and the literal which is equal to the literal itself. The horizontal wire is divided into multiple clocking zones and literal wire is rotated to enable wire crossing.

This cell can be replicated horizontally but not vertically-up because it would again create a long vertical literal wire in one clocking zone, violating condition [I]. Figure 4-13(b) shows an example AND-plane using AND-plane macro-cell that outputs product-terms on the very-right of the plane at both levels. The condition [I] is completely satisfied by replicating the cell vertically-up with an offset of one clocking zone as shown in Figure 4-13(b). It can be observed from Figure 4-13(b) at any horizontal level that the AND-plane cell satisfies condition [III] completely; this is achieved by dividing the output wire of one AND-plane cell into some additional clocking zones to synchronize it with the literal of the next cell.

The first three conditions are completely satisfied as described above, but meeting condition [IV] is difficult since any alteration in the structure of the cell, done to synchronize outputs of all levels at the end, would be reflected in all prior cells as well. The product term outputs of the AND-plane are the inputs to the OR-plane. To resolve this issue, the OR-plane was assigned the clocking zones such that it synchronizes all the AND-plane output for OR operation. The structure of OR-plane is discussed in section 4.4.2.2.



**Figure 4-13 QCA-PLA AND-plane macro-cell and AND-Plane: simulated in QCADesigner to verify functionality.**

#### 4.4.2.2 OR-plane cell

As a reminder to the reader, this thesis focuses only on single output functions thus the design of OR-plane is different than the one shown in section 4.3.5.2 which generates

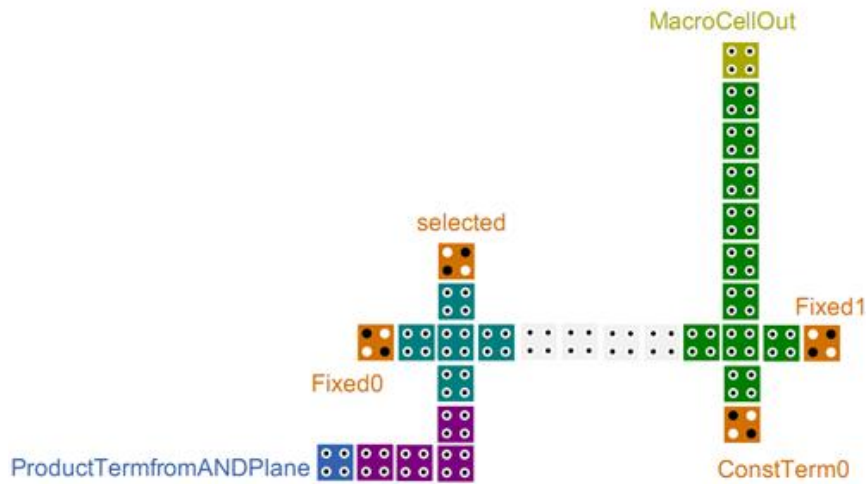
multiple outputs. An OR-plane macro-cell is shown in Figure 4-14(a) and corresponding OR-plane is shown in Figure 4-14(b). The `ProductTermFromANDPlane` is the product-term output of a level in AND-plane. The `selected` represents that this product-term from the AND-plane is selected for OR-operation as a part of forming SOP term. Since only single-output functions are considered in this thesis, it is intuitive that the product-terms from all levels are AND-plane will be `selected` in OR-plane. The `ConstTerm0` is 0 for the first OR-plane macro-cell since there is no other macro-cell below it to send a product-term for logical-OR. Thus, the output of the first OR-plane macro-cell in this study would always be equal to its input.

Similar to AND-plane case, the OR-plane macro-cell cannot be stacked vertically-up for all levels, as this would violate condition [I] of a long vertical wire. Similar approach is used in the OR-plane for stacking OR-plane macro-cell as in multiple levels of the AND-plane macro-cells, i.e. replicating vertically-up by offsetting clocking zones of the corresponding cells by one, as illustrated in Figure 4-14(b). This type of clocking assignment for OR-plane also helps in satisfying condition [IV] of AND-plane. As mentioned in section 4.4.2.1 condition [IV], in AND-plane the min-terms realized at higher levels are slower in propagation towards the OR-plane than the min-terms realized at lower levels. This is due to the fact that, for the min-terms at higher level, the literals have to first propagate vertically up and then right towards the OR-plane. Higher a min-term is; more would the literal need to propagate up for that min-term, while literal starts moving right towards the OR-plane earlier for lower levels. Thus the min-terms on the

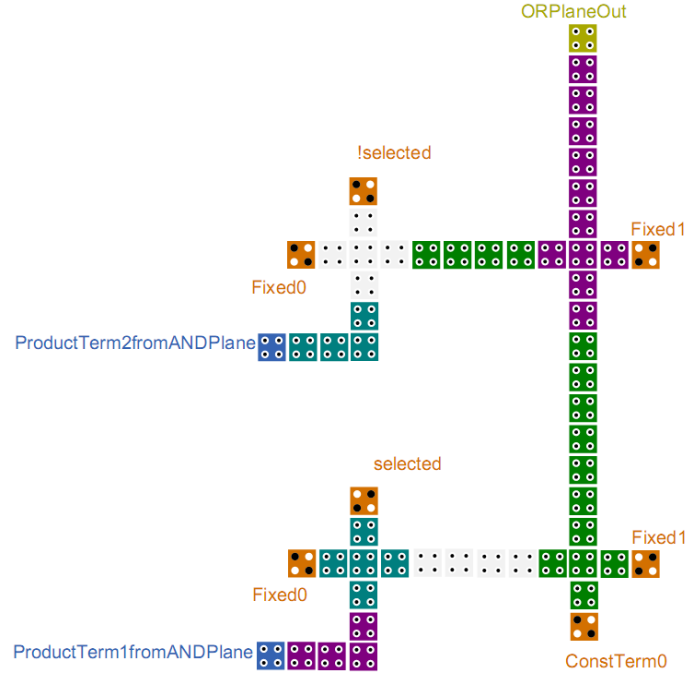


higher level are slower to reach the OR-plane than min-terms at lower levels, which is not desired since two min-terms should be in the same clocking phase to perform logical-OR correctly in the OR-plane.

The clocking assignment for the OR-plane as shown in Figure 4-14(b) slows down the min-terms proceeding faster toward OR-plane by exactly the amount needed to synchronize min-terms from different AND-plane levels. This is achieved by having the faster min-terms at lower levels pass through additional clocking zone, thus slowing them down, on their way towards the OR-gates in OR-plane. For instance, the min-term at level 0 passes through one additional clocking zone as compared to min-term at level 1 on their way to OR-gate in OR-plane. This pattern can be observed from the complete QCA-PLA layout of an arbitrary function shown in Figure 4-15 in a later section.



(a) OR-plane macro-cell partially conforming to design condition [I].



(b) OR-Plane using OR-plane cell replication satisfying applicable conditions and enabling AND-plane satisfy condition [IV].

**Figure 4-14 QCA-PLA OR-plane macro-cell and OR-plane: simulated in QCADesigner to verify functionality.**

#### 4.4.2.3 QCA-PLA: Complete Layout

Figure 4-15 shows the complete QCA Shannon Lattice for an arbitrary function of two variables with two product-terms. Two variables account for four AND-plane macro-cells in horizontal direction since both literal and its complement needs to be generated. There are two AND-plane levels in this layout, each of which implement one product term from the function. The final output of the realized function is generated at QCA-cell F which the right-most-top-most cell of the design. Figure 4-15 verifies the fact that AND-plane cell of Figure 4-13(a) and OR-plane cell of Figure 4-14(a) can both be replicated with clock offset in vertical direction and combined to generate the entire QCA

PLA layout. This layout provides the correct functionality while satisfying all four conditions of section 4.4.2.1. This fact was confirmed by simulating various Boolean functions realized with QCA-PLA following this structure. It is easy to observe that clocking structure required for this design would be uniform. The standard macro-cells in this case are AND-plane and OR-plane cells which are added to the standard cell library to be used by the tool for PLA layout generation.

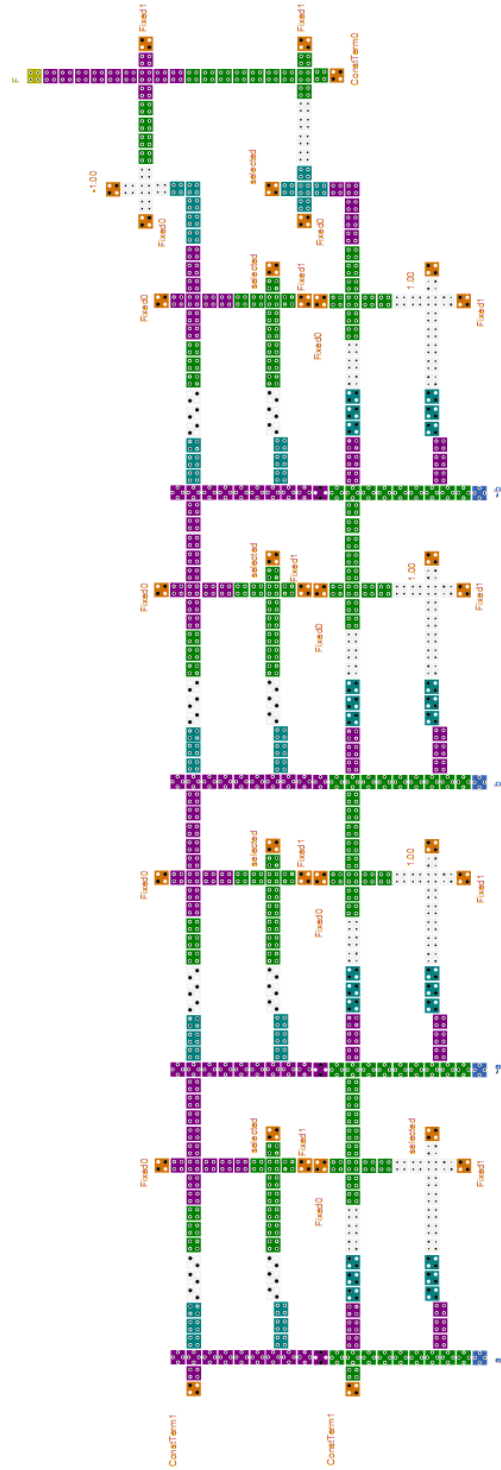


Figure 4-15 QCA-PLA for an arbitrary function: simulated in QCADesigner.

## Chapter 5

### LAYOUT GENERATOR TOOL USING REGULAR CELLS

#### 5.1 Motivation for Layout Software

This research work presents an analysis on the performance/area trade-offs in logic realization using Shannon lattice and Programmable-Logic-Array (PLA) in QCA for single output. As mentioned in the introduction of this thesis, only single output functions were considered due to their simplicity and ease of layout tool development for them. Performing the study needs creating and simulating QCA layout of different Boolean functions in both Shannon lattice and PLA. This process is simple when number of variables in the Boolean functions is small but as this number increases, it becomes a tedious task to perform a manual layout of whole design, define input-output cells and assign clock zones. An additional step of complexity is incurred when mapping the logic function to Shannon lattice which is the decomposition of the input Boolean function into its equivalent Shannon lattice representation. As a part of this thesis, a tool was developed in C programming language which accepts Boolean function in simple logic equation form or *blif* format and generates the desired Shannon lattice or PLA layout. It also performs the decomposition of Boolean function into Shannon lattice using simple algorithms. Even though the basic macro-cell can simply be replicated to generate the whole layout and bigger designs can be studied based on macro-cell properties, it is still worthwhile to develop such tools in order to analyze the consistency in the behavior of bigger circuits with macro-cells. This type of analysis is more important in a technology

where the state of a cell is affected largely with respect to the type and location of its neighbors. In addition, this tool provides the flexibility of using different designs of macro-cells to generate the layout which can be used to compare the behavior of varied layouts. In following section, the tool's structure and algorithm for generating the layout files are described. The tool's output layout file can be simulated in QCADesigner simulator.

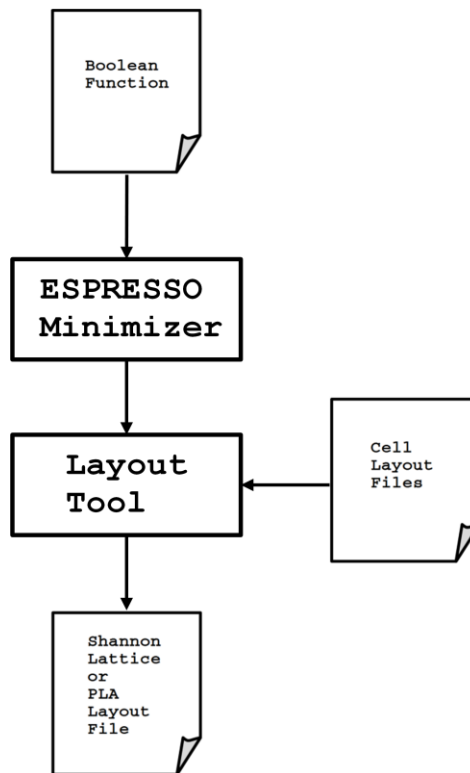
## 5.2 Layout File Generation Flow

The objective of the tool is to produce a complete Shannon Lattice or PLA layout for the input Boolean function using the cells described in chapter 4. The tool reads the configuration file that contains the Boolean function and other parameters (discussed later) to generate the layout. The Boolean function can either be directly specified in the configuration file as Boolean expression or converted from *blif* format into equivalent Boolean expression using the tool. The converted Boolean expression, however, needs to be manually added to the configuration file. If the input Boolean function is not in its minimized form, it is minimized using *ESPRESSO* minimize software. The following are the inputs to the tool:

- **Boolean Function *blif* format:** converts the function into Boolean expression for config-file.
- **Configuration File:** contains the Boolean function and other parameters.

- **QCA Multiplexer Cell Layout File:** used to generate the layout for Shannon Lattice.
- **QCA AND- and OR-Plane Cells Layout File:** used to generate the QCA PLA layout.

The output of the tool is a .qca file containing the layout information which can then be simulated in QCADesigner. The overall high level view of layout generation flow is shown in Figure 5-1:



**Figure 5-1 Layout-generation flow.**

The detailed flow from defining Boolean function to QCA Designer simulation is shown below:

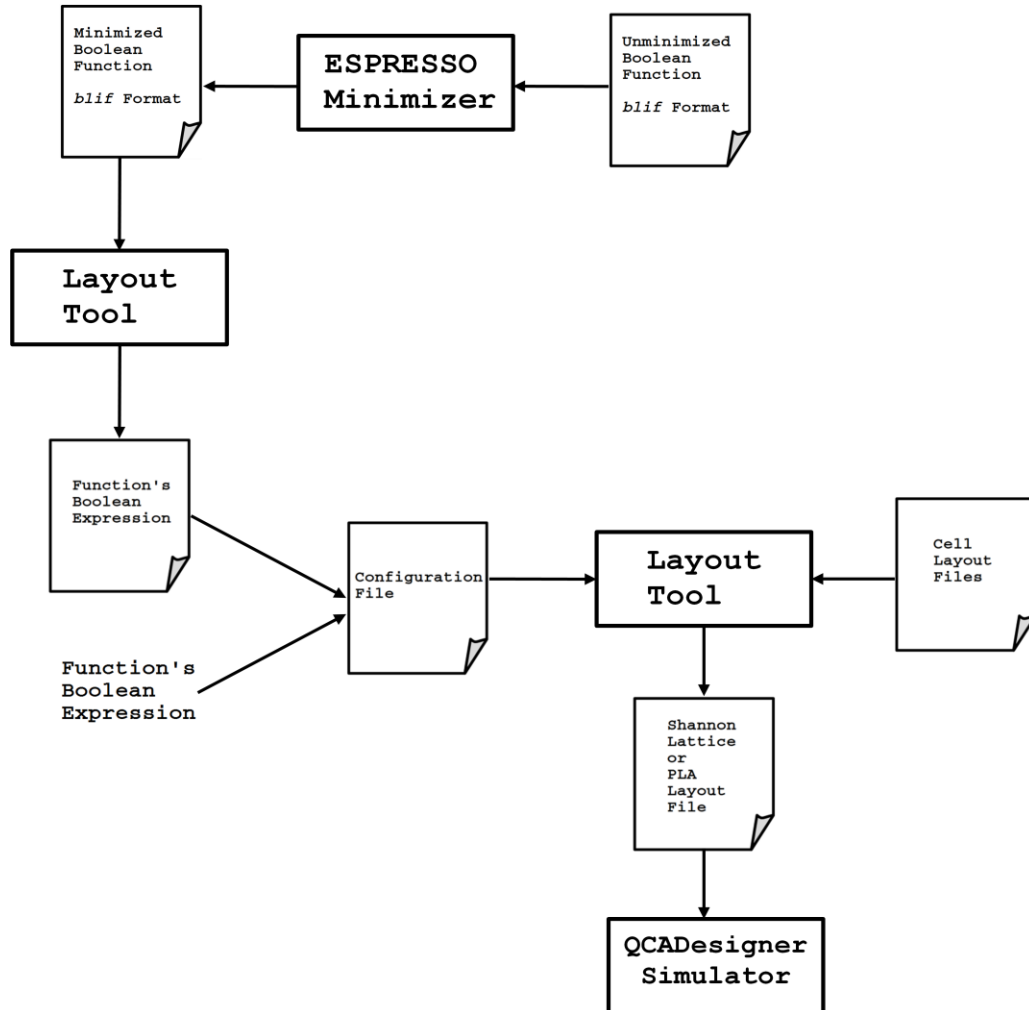


Figure 5-2 Detailed layout-generation and simulation flow.

### 5.3 File Formats



As seen from Figure 5-2, the final layout file generation involves creating and processing several files with intermediate format conversions. This section provides the details of the all the required files.

### 5.3.1 Non-minimized-Boolean-Function file

This file is either a single output function from benchmark suite or a user-defined function in *blif* format.

```
.i 4
.o 1
.p 6
0010 1
0011 1
0100 1
0110 1
1001 1
1010 1
.e
```

(a) exam3\_d PLA benchmark function.

```
.i 4
.o 1
.p 4
1001 1
01-0 1
-010 1
001- 1
.e
```

(b) Minimized exam3\_d by ESPRESSO.

**Figure 5-3 exam3\_d benchmark in *blif* format and its ESPRESSO minimized equivalent.**

The Figure 5-3 shows the exam3\_d benchmark and its ESPRESSO-minimized equivalent. The parameters and data representation are as follows:

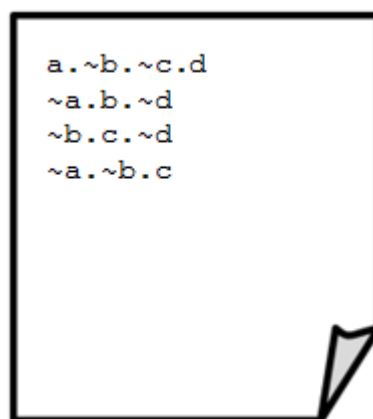
- |   |                                  |
|---|----------------------------------|
| 'i': number of the variables/literals/input | '1': Variable/literal positive   |
| 'o': number of outputs                      | '0': Variable/literal complement |
| 'p': number of terms                        | '-': Variable Don't-Care         |

\.e': end of file

The terms presents in the files are the terms for which the Boolean function evaluate to 1. For instance, term '0010' represents ' $\sim a \cdot \sim b \cdot c \cdot \sim d$ ' in Figure 5-3(a) and function's output is 1 for this term. Each of the term is the product term of the entire logic function defined in SOP form. Figure 5-3(b) is the minimized Boolean function-blif format shown in Figure 5-2.

### 5.3.2 Function-boolean-expression file

The layout tool requires the input function to be in Boolean expression form thus blif format needs to be converted. The tool has the functionality to convert the *blif* format into Boolean expression and generate an output file that contains minimized function in Boolean expression format as shown in Figure 5-4. These expressions will be added to the configuration file.

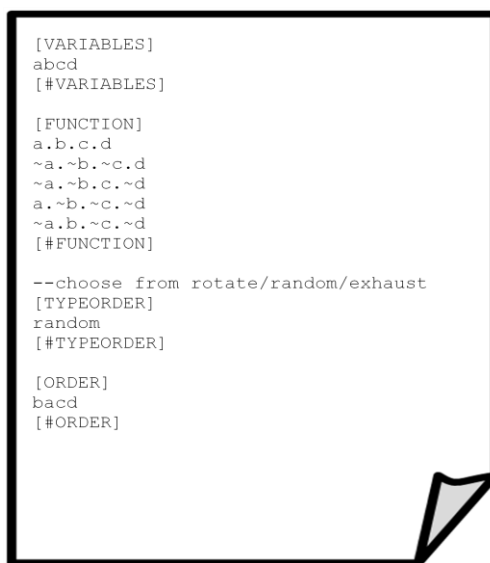


```
a.~b.~c.d
~a.b.~d
~b.c.~d
~a.~b.c
```

**Figure 5-4** Minimized function of Figure 5-3(b) as Boolean SOP expression with four terms.

### 5.3.3 Configuration file

The logic function can either be defined directly in the configuration file or added from the file generated that contains the minimized Boolean expressions as in Figure 5-4. Due to this reason Figure 5-2 shows two sources of the function for the configuration file. A sample configuration file is shown in Figure 5-5.



```
[VARIABLES]
abcd
[#VARIABLES]

[FUNCTION]
a.b.c.d
~a.~b.~c.d
~a.~b.c.~d
a.~b.~c.~d
~a.b.~c.~d
[#FUNCTION]

--choose from rotate/random/exhaust
[TYPEORDER]
random
[#TYPEORDER]

[ORDER]
bacd
[#ORDER]
```

**Figure 5-5 Configuration file.**

The fields in the configuration file are described below. A field starts with [`<field>`] and ends with [#`<field>`] with field-data between the two.

- [`VARIABLES`] : This block contains the list of all the literals that form the Boolean function.

- [FUNCTION] : This block contains the product-terms of the SOP representation of the Boolean function. This is either added from the file obtained after conversion of the minimized Boolean function in blif format or specified directly.
- [TYPEORDER] : This field is only used in generating the Shannon Lattice layout for asymmetric functions. As described in section 3.5.2, the lattice diagram of an asymmetric function could also depend on the order of the variables chosen for decomposition. If [TYPEORDER] field is 'rotate' then the order of the variables defined in [ORDER] field is chosen for creating lattice diagram. If variables need to be repeated, then the variables are chosen starting from the beginning of the string defined in [ORDER]. If [TYPEORDER] field is 'random' then variables are chosen for decomposition from the randomly generated strings of variables.
- [ORDER] : This field is only used for generating the Shannon Lattice layout for asymmetric functions. The variables are chosen in the order defined in this field for the decomposition. The repetition algorithm for variables, if needed, depends on the [TYPEORDER] field.

#### 5.3.4 Cell-layout files

These are the layout files of multiplexer, and-plane and or-plane cells which are used as standard cells to create the complete layout of input Boolean function. These cells have

already been characterized for working as a part of bigger designs by manually creating them.

## **5.4 Layout Generator Algorithm**

The basic algorithm for generating, either Shannon Lattice or PLA, layout is to read QCA-cell data from the input macro-cell files and, since the layout is regular and properties of all the QCA-cells in the final layout can be computed based on the characteristics of macro-cells, generate all QCA-cells for the final layout and write the QCA-cell data to a file to be simulated in QCADesigner. However, the series of steps that the layout-tool goes through is different between Shannon-Lattice and PLA layout. This is because of the fact that creating a Shannon-Lattice also involves the decomposition of the Boolean function and creating the lattice diagram. A Boolean function can be directly mapped to the PLA without any decomposition.

### **5.4.1 Algorithm for QCA-Shannon-Lattice layout**

Creating Shannon Lattice layout involves two major steps. First, generate the data structure representing the Shannon Lattice diagram and second, populate the fields of a QCA-cell data record using the data structure created in first step and input QCA multiplexer standard-cell. This QCA-cell data is written into a file. The process is repeated for all cells required to generate the entire layout.

#### 5.4.1.1 Algorithm for lattice-diagram

It can be observed from the lattice diagram shown in Figure 3-7, that each node is a multiplexer that is connected to its positive and negative co-factors which are also implemented using multiplexers. Therefore, the entire structure is a DAG of multiplexers which can be implemented as linked-list in *C* language. Following this observation, a *C* data record '`struct mux`' is defined that holds the information about itself and *C*-pointer connections to its children multiplexers. It also holds information about its neighbors on the same level, as *C*-pointers, which is needed for lattice minimization and mapping it to QCA layout. A vertical array of *C*-pointers is defined which point to the left-most multiplexer-record at each level thus providing handles to records at any level for processing. Therefore, a pointer in vertical array points to the head of linked-list of multiplexer records at a level equal to the index of pointer in array. The multiplexer-records are dynamically allocated in the memory as the lattice grows. A node is identified by its Boolean-expression-string in a file and its record-properties in the data structure in memory. The co-factors of a node are dynamically computed through string manipulation. The Boolean-expression-string corresponding to a node is stored in a text file along with the serial number of the node assigned to it dynamically as the lattice grows. The same serial number is also stored in the corresponding multiplexer record in data structure. The serial number of a node acts as a link between the Boolean-expressions-strings in the text file and records in the data structure. The serial number of

a node in data structure can be used to read node's Boolean-expression-string from the text file by reading the strings related to the serial number. This is needed either to compute the co-factors of node or to create an equivalent node by performing Join-Vertex operation. For any node, once both its co-factors are computed, they are also written into same text file with their serial number, since they eventually act as nodes themselves. It is easy to observe that the Boolean-expression-string corresponding to a node is kept in a text file and other properties (connection data) of the node are kept in the data structure in the memory. A symmetric function is a special case of non-symmetric function so same algorithms would work to generate the lattice diagram for both cases. The following is the algorithm for generating the Shannon lattice data structure:

### **Algorithm for generating Shannon-Lattice data structure**

```

Set node_serial_number = 1
Set tree_level = 1
Read Boolean function from configuration file = root_node
Write to text file: root_node's Boolean-expression-string and
node_serial_number = 1
Create corresponding multiplexer-record:
    a. mux_record_serial_number = node_serial_number
    b. mux_record_level = tree_level
    c. mux_record_node_index = 0;
Add multiplexer-record to linked-list at level = mux_record_serial_number

Starting from index equal to tree_level of vertical-pointer-array
While (all nodes not constants at tree_level) do following:

    red_var = get_reduction_variable()
    node_index_at_tree_level = 0

    foreach (multiplexer-record at tree_level) do following:
        NCF = get_negative_cofactor using red_var
        if (NCF != constant)
            a. node_serial_number++
            b. Write to text file: new multiplexer-record with
                serial number = node_serial_number

```

```

        c. set its mux_record_serial_number =
            node_serial_number
        d. set its mux_record_node_index =
            node_index_at_tree_level
        e. set its parent connections through pointers
        f. add it to linked-list at level = tree_level+1

node_index_at_tree_level++

PCF = get_positive_cofactor using red_var
if (PCF != constant)
    a. node_serial_number++
    b. Write to text file: new multiplexer-record with
        serial number = node_serial_number
    c. set its mux_record_serial_number =
        node_serial_number
    d. set its mux_record_node_index =
        node_index_at_tree_level
    e. set its parent connections through pointers
    f. add it to linked-list at level = tree_level+1

node_index_at_tree_level++

Connect multiplexer-records of NCF and PCF into data
structure
end foreach

Call graph_reduction function for level = tree_level+1
tree_level++: repeat for the next level

End While

Call translate_node_index to manipulate mux_record_node_index for all
records/nodes in the data structure to correspond to QCA-multiplexer-placement-
offset

```

### **Algorithm for function *graph-reduction()***

```

Traverse the linked-list at level = tree_level+1 from left to right
Find equivalent adjacent nodes using mux_record_node_index at this level.
If found and their parents are different but have the same grand-parent
    a. Remove one of the nodes from the data structure
    b. Adjust the pointers in the data structure
    c. Fix the mux_record_node_index for all records at this level due to
        removal of one record

If not found then for adjacent nodes with different parents but same grand-
parent
    a. Apply Join-Vertex operation to fuse two non-equivalent nodes

```



- b. node\_serial\_number++
- c. Remove joined nodes from the data structure
- d. Create new equivalent multiplexer-record after joining two nodes
- e. Write to text file: new multiplexer-record with serial number = node\_serial\_number
- f. set its mux\_record\_serial\_number = node\_serial\_number
- g. Adjust pointers in data structures
- h. Fix the mux\_record\_node\_index for all records at this level due to removal of records

Go back

### **Algorithm for function *get\_reduction\_variable()***

If **REDUCTION\_TYPE** is **ORDER** in header file

    If **TYPEORDER** = **rotate** in configuration file

- a. Return the variable in the order defined in **ORDER** field string of configuration file
- b. If function doesn't reduce to constants with one pass of whole string of **ORDER** field, start same process again from the beginning of the **ORDER** field string

    Else if **TYPEORDER** = **random** in configuration file

- a. Generate a random string based on the **VARIABLES** field string from configuration file
- b. Return variable from start to end of the generated random string
- c. If function doesn't reduce to constants with one pass of whole random string, generate another random string and repeat the process again from the start of the string

Else if **REDUCTION\_TYPE** is **DEFAULT** in header file

    Return the variable that appears in the first term of node with node\_index = 0 at this level

Else if **REDUCTION\_TYPE** is **GREEDY** in header file

    Return the variable based on the greedy algorithm defined in [8]

### **Algorithm for function *translate\_node\_index()***

Starting from root multiplexer record node 1

Repeat for all level of Shannon-Lattice data structure

    NCF multiplexer record's node index = low parent's node index

    PCF multiplexer record's node index = high parent's node index + 1

#### 5.4.1.2 Algorithm for QCA-cell data

The layout file of circuit for QCADesigner contains the information about all the QCA-cells of that circuit. Figure 5-6 shows an example of QCA-cell definition for QCADesigner simulator. All these fields are needed to be defined for a QCA-cell to be correctly read by the QCADesigner tool. When a user performs the layout in QCADesigner, this information is generated automatically by QCADesigner as a QCA-cell is laid out which is stored in a file. If more cells are added or other parameters are changed, the file is updated with the information. In order to automatically generate a layout, the tool developed in this thesis generates all QCA-cells with their properties for the entire layout and writes them into a file following the format as in Figure 5-6. A QCA-cell description for QCADesigner is enclosed with `[TYPE:QCADCell]` and `[#TYPE:QCADCell]` and contains three major sections:

- Overall Cell position enclosed with `[TYPE:QCADesignObject]` and `[#TYPE:QCADesignObject]`
- `cell_function` field describing type of cell: input, output or fixed.
- Four `cell_dots` each enclosed in `[TYPE:CELL_DOT]` and `[#TYPE:CELL_DOT]`.
- Cell label, if present, and its properties enclosed with `[TYPE:QCADLabel]` and `[#TYPE:QCADLabel]`.

```

[TYPE:QCADCell]
[TYPE:QCADDesignObject]
x=480.000000
y=1080.000000
bSelected=FALSE
clr.red=0
clr.green=0
clr.blue=65535
bounding_box.xWorld=471.000000
bounding_box.yWorld=1071.000000
bounding_box.cxWorld=18.000000
bounding_box.cyWorld=18.000000
[#TYPE:QCADDesignObject]
cell_options.cxCell=18.000000
cell_options.cyCell=18.000000
cell_options.dot_diameter=5.000000
cell_options.clock=0
cell_options.mode=QCAD_CELL_MODE_NORMAL
cell_function=QCAD_CELL_INPUT
number_of_dots=4
[TYPE:CELL_DOT]
x=484.500000
y=1075.500000
diameter=5.000000
charge=8.010882e-020
spin=0.000000
potential=0.000000
[#TYPE:CELL_DOT]
[TYPE:CELL_DOT]
x=484.500000
y=1084.500000
diameter=5.000000
charge=8.010882e-020
spin=0.000000
potential=0.000000
[#TYPE:CELL_DOT]
[TYPE:CELL_DOT]
x=475.500000
y=1084.500000
diameter=5.000000
charge=8.010882e-020
spin=0.000000
potential=0.000000
[#TYPE:CELL_DOT]
[TYPE:CELL_DOT]
x=475.500000
y=1075.500000
diameter=5.000000
charge=8.010882e-020
spin=0.000000
potential=0.000000
[#TYPE:CELL_DOT]
[TYPE:QCADLabel]
[TYPE:QCADStretchyObject]
[TYPE:QCADDesignObject]
x=484.943182
y=1059.951299
bSelected=FALSE
clr.red=0
clr.green=0

```

```

clr.blue=65535
bounding_box.xWorld=471.000000
bounding_box.yWorld=1048.000000
bounding_box.cxWorld=27.886364
bounding_box.cyWorld=23.902598
[#TYPE:QCADDesignObject]
[#TYPE:QCADStretchyObject]
psz=I0
[#TYPE:QCADLabel]
[#TYPE:QCADCell]

```

**Figure 5-6 An example QCA-cell description for QCADesigner simulator.**

To compute QCA-cell properties of entire QCA Shannon-Lattice layout, the following values are needed:

- Standard cell (multiplexer-cell from Figure 4-12(b)) dimensions:** These are needed to compute the coordinates of QCA-cells for the multiplexer-layouts at different levels. For example: Traversing level-wise, QCA-cells in multiplexers at level 2 in lattice would be displaced in x-direction by multiplexer's width from the corresponding QCA-cells in multiplexer at level 1. Likewise, traversing at a particular level, QCA-cells in multiplexer with node-index 2 would be displaced in y-direction by multiplexer's height from corresponding QCA-cells in multiplexer at node index 1. Thus, for a particular level, the coordinates of a reference-cell can be computed using the QCA-multiplexer width displacement with level number. Then, using the reference-cell of a level, the coordinates of QCA-cells for multiplexers at that level can be computed using the QCA-multiplexer width and node indices for multiplexers.
- Node-indices from lattice data structure:** These are used as offsets to compute the coordinates of QCA-cells belonging to multiplexer at a level.

- **QCA-cell properties of standard-multiplexer-cell:** The coordinates of the QCA-cells read from the input QCA multiplexer macro-cell are used as the reference for all computations. For instance, the coordinates of the reference cell at any level is obtained by performing computation on the corresponding QCA-cell's coordinates from input QCA-multiplexer, QCA-multiplexer width and level number for which reference cell needs to be created.

The pictorial representation of the algorithm is shown in Figure 5-7 below.

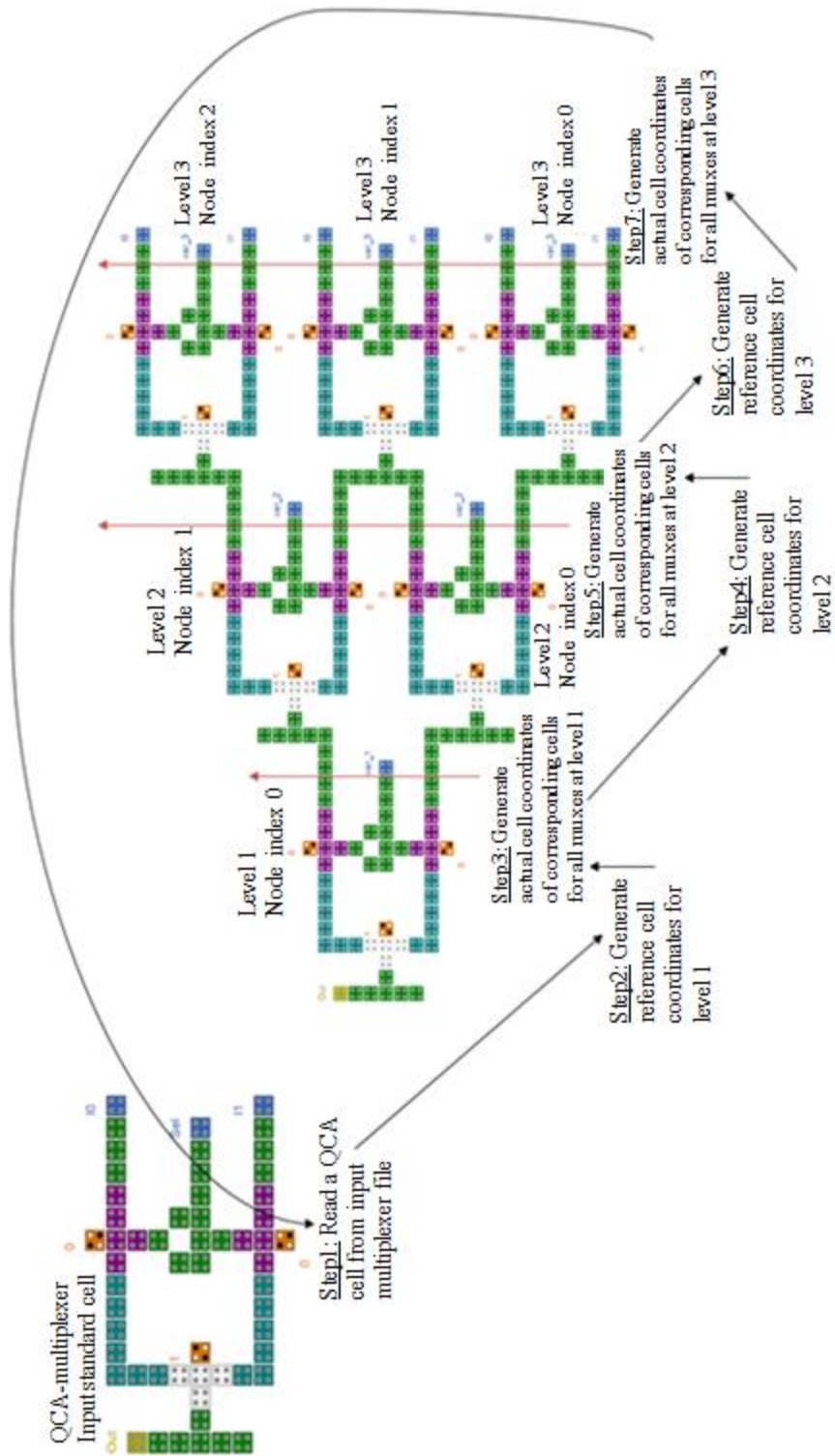


Figure 5-7 Pictorial representation of algorithm for creating QCA-cell data for Shannon-Lattice.

To perform these computations, three C-records are defined as follows:

**lib\_cell**: Library Cell; C-record that is populated with QCA-cell properties of cell read from input multiplexer file.

**lev\_ref\_cell**: Level Reference Cell; C-record that is populated with QCA-cell properties computed using lib\_cell properties.

**lev\_act\_cell**: Level Actual Cell; C-record that is populated with QCA-cell properties computed using lev\_ref\_cell properties.

The equations used for coordinate calculations are:

```
lev_ref_cell->x_f = lib_cell->x_f + (lev * ADD_NXT_LEV_X_DIS)
lev_ref_cell->y_f = lib_cell->y_f + (lev * ADD_NXT_LEV_Y_DIS)
lev_ref_cell->clock_i = get_clock(lib_cell->clock_i, lev)
```

### **Library QCA-cell coordinate to reference QCA-cell coordinate mapping**

<b>lev_ref_cell-&gt;x_f</b>	x-coordinate of reference cell at level = lev
<b>lev_ref_cell-&gt;y_f</b>	y-coordinate of reference cell at level = lev
<b>lev_ref_cell-&gt;clock_i</b>	clocking zone of reference cell for a level = lev which is same as the clocking zone of the library standard-cell based on the QCA-multiplexer design here
<b>lib_cell-&gt;x_f</b>	x-coordinate of a QCA-cell read from input multiplexer file
<b>lib_cell-&gt;y_f</b>	y-coordinate of a QCA-cell read from input multiplexer file
<b>lib_cell-&gt;clock_i</b>	clocking zone of a QCA-cell read from input multiplexer file
<b>lev</b>	Level of lattice for which QCA-cell coordinates are calculated in this iteration
<b>ADD_NXT_LEV_X_DIS</b>	QCA multiplexer width + Space between two QCA-cells
<b>ADD_NXT_LEV_Y_DIS</b>	QCA multiplexer height + Space between two QCA-cells

**Table 5-1** Library cell to reference-cell coordinate mapping.

Other fields of the QCA reference-cell are computed in the similar fashion. Table 5-1 describes the parameters used for computing reference-cell coordinates. It can be inferred that the reference-cell has same values of all the fields as library QCA-cell

except of the properties involving the coordinates. The following equations show the relationship between the coordinates of the reference-cell and actual-cell. The properties of actual-cells are written to the final layout file for simulation.

```
lev_act_cell->x_f = lev_ref_cell->x_f - (mux_at_lev->node_ndx *
SUB_SAME_LEV_X_DIS)

lev_act_cell->y_f = lev_ref_cell->y_f - (mux_at_lev->node_ndx *
SUB_SAME_LEV_Y_DIS)

lev_act_cell->clock_i = lev_ref_cell->clock_i

set_act_cell_prop_mux(lev_ref_cell, lev_act_cell, lev, mux_at_lev)
```

### **Library QCA-cell coordinate to Layout/Actual QCA-cell coordinate mapping**

<b>lev_act_cell-&gt;x_f</b>	x-coordinate of actual cell for node-index = node_ndx at level = lev
<b>lev_act_cell-&gt;y_f</b>	y-coordinate of actual cell for node-index = node_ndx at level = lev
<b>lev_act_cell-&gt;clock_i</b>	clocking zone of actual cell for node-index = node_ndx which is same as the clocking zone of the reference-cell
<b>mux_at_lev</b>	Pointer to multiplexer record at level = lev; compute cell-properties for actual QCA-cells for this QCA-multiplexer
<b>node_ndx</b>	Node index of multiplexer record pointed by mux_at_lev at level = lev
<b>set_act_cell_prop_mux()</b>	Function that sets the properties other than coordinates of actual cell. For example: convert reference-cell from QCADesigner-input-cell to normal cell depending on the lattice data structure, change color from input cell to normal
<b>lev</b>	Level of lattice for which QCA-cell coordinates are calculated in this iteration
<b>lev_ref_cell</b>	Pointer to reference-cell record for level = lev
<b>lev_act_cell</b>	Pointer to actual-cell record whose properties are computed using reference-cell. The properties of this cell are written to layout file.
<b>mux_at_lev</b>	Pointer to mux-record in data structure which is used to manipulate the properties of actual cell in function set_act_cell_prop_mux()

**Table 5-2** Reference cell to layout/actual-cell coordinate mapping.

The high-level overall algorithm for QCA-Shannon-Lattice generation is shown below:



```

Define library-QCA-cell-record
Define reference-QCA-cell-record
Define actual-QCA-cell-record

Repeat until all QCA-cells in input QCA-multiplexer macro-cell are read

    Read next QCA-cell from file; populate library-cell-record with cell
    properties

    for_1 all levels of lattice-diagram data structure traversing top to
    bottom, do following:
        Compute reference QCA-cell properties for this level using library
        QCA-cell
        for_2 all multiplexer-records in linked-list at this level
        traversing left to right, do following:
            a. Compute actual QCA-cell coordinate-properties
            using reference-cell and node index
            b. Modify the other cell properties based on its
            position and configuration needed for the
            implementing the input logic function
            c. Write actual QCA-cell properties to output file
        end for_2
    end for_1

End Repeat

```

### 5.4.2 Algorithm for QCA-PLA layout

Generating the QCA-PLA layout for an input Boolean function is simpler compared to QCA Shannon-Lattice layout. This is because QCA-PLA layout does not involve generating any type of lattice diagram which then needs to be mapped to QCA layout as in case of Shannon-Lattice. The only step that is needed to create a QCA-PLA is to duplicate the AND-plane standard cell by two times the number of variables/literals times the number of product-terms and duplicate the OR-plane standard cell as many times as the number of product-terms in the Boolean function. The OR-plane is duplicated less number of times because in this thesis only single output logic functions are considered which require only one stack of OR-plane cells. As shown in Figure 4-15, the clocking

zones of the cells are offset by one between adjacent levels; this fact is taken into account when generating clock field for any QCA-cell. The coordinates for the QCA-cells are computed in similar way as for Shannon-Lattice in previous section; using the dimensions of the input AND- or OR-plane cells. The important property in QCA-PLA is the polarity of the ‘Select-bit’ in both the planes. Since only single output functions are considered, the select-bit cells for all OR-plane cell are set to logic ‘1’ i.e. all product terms selected for logical OR operation. However, the polarity of select-bits of the AND-plane cells at a level depends on polarity or presence of the literal in the term realized at that level.

To compute QCA-cell properties of entire QCA-PLA layout, the following values are needed:

- **Standard cells AND- and OR-plane dimensions:** These are needed to compute the coordinates of QCA-cells for AND-plane and OR-plane at different levels and for all literals. For example: traversing horizontally at a particular level that implements one product-term, the QCA-cells in an AND-plane cell are displaced in x-direction from the corresponding QCA-cells in previous AND-plane cell by AND-plane cell’s width. Likewise, the QCA-cells in the AND-plane cell at a level are displaced in y-direction from the corresponding QCA-cells in AND-plane cell below by AND-plane cell’s height. As for OR-plane, the displacement is only in y-direction which is equal to the width of OR-plane cell.

- **Input Boolean function:** Each term of the input Boolean function is implemented by one level of AND-plane cells as a product term. The Boolean function is needed in order to find out the polarity of the select-cells for AND-plane cells at this level. This, in turn, depends on whether a literal is present or not in the product-term and also on, if present, what polarity i.e. variable or its complement.
- **QCA-cell properties of standard AND-plane and OR-plane cells:** The properties of the QCA-cells in these macro-cells are used as the reference to compute the properties of the QCA-cell which are eventually written to the layout file.

The pictorial representation of the algorithm is shown in Figure 5-8 below.



The high-level overall algorithm for QCA-PLA generation is shown below:

```
Define library-QCA-cell-record
Define reference-QCA-cell-record
Define actual-QCA-cell-record

Repeat until all QCA-cells in input QCA AND-plane macro-cell are read

    Read next QCA-cell from file; populate library-cell-record with cell
    properties

    for_1 (number of product-terms/levels); do following
        Compute reference QCA-cell properties for this level using library
        QCA-cell
        for_2 (number of variables*2); do following
            a. Compute actual QCA-cell coordinate-properties using
               reference-cell properties and variable serial number
            b. Modify the other cell properties based on its position
               and configuration needed for the implementing the input
               logic function
            c. Write actual QCA-cell properties to output file
            d. Store the coordinates for bottom-most-right-most QCA-
               cell
        end for_2
    end for_1

end Repeat; AND-plane complete

Repeat until all QCA-cells in input QCA OR-plane macro-cell are read

    Read next QCA-cell from file; populate library-cell-record with cell
    properties

    for_1 (number of product-terms/levels); do following
        e. Compute actual QCA-cell coordinate-properties using
           library-cell properties, variable-serial-number and
           coordinates for bottom-most-right-most QCA-cell in AND-
           plane
        f. Modify the other cell properties based on its position
           and configuration needed for the implementing the input
           logic function
        g. Write actual QCA-cell properties to output file
    end for_1

end Repeat; OR-plane complete
```

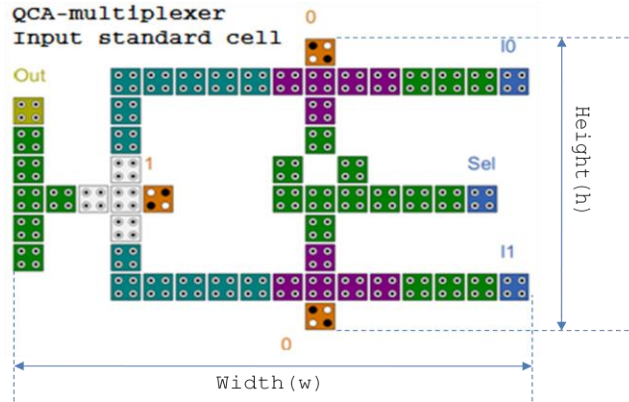
## Chapter 6

### ANALYSIS AND RESULTS

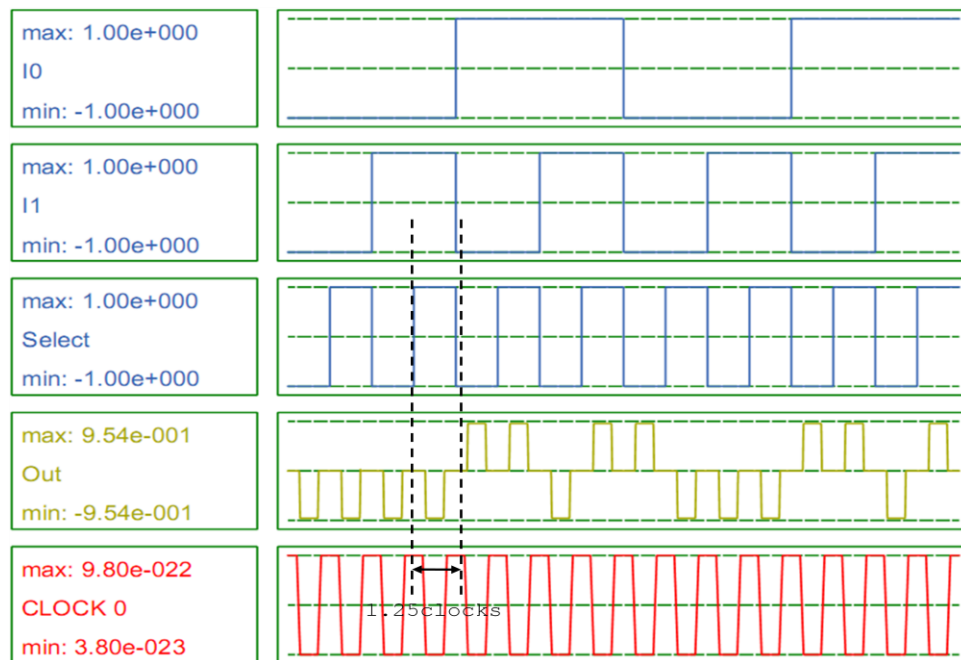
#### 6.1 Macro-Cell Characterization

Both QCA-PLA and QCA-Shannon-Lattice layout, presented in this thesis are generated using macro-cells as a basic element of the whole design. Thus, it is natural to characterize these macro-cells for area, cell-density and latency. As a reminder, these macro-cells were designed with design-constraints laid out in section 4-3 of this thesis. All simulation were performed using QCADesigner with Bistable Approximation Simulation Engine for 12,800 – 600,000 number of samples with 100 – 500 iterations per sample on a computer system with Windows 7 OS running Intel Corei7 CPU and 8GB of RAM. The layout files were generated using a computer system with 64-bit Ubuntu OS, Intel Corei7 CPU and 8GB of RAM. It advised to use the layout tool on a 64-bit OS system with large memory as this one because some issues were observed on systems with smaller memory.

##### 6.1.1 QCA-Multiplexer macro-cell



(a) QCA-multiplexer macro-cell.



(b) Simulation waveforms: Sel to Out latency.

**Figure 6-1 Latency computation for QCA-Multiplexer.**

**Latency:** In the QCA-Multiplexer macro-cell of Figure 6-1(a), the select signal  $Sel$  or its complement are connected to the same majority gates as the inputs  $I0$  or  $I1$ . Thus the latency from  $Sel$  or any of the two inputs to the output is same. The latency for this circuit is 1.25 clock cycles as shown in figure 6-1(b). The latency of this circuit can also

be computed by counting the number of clocking zone, from any input to the output, which is five. Since QCA-clock consists of four phases, it would take a quarter of a cycle to move data from one clocking zone to next. Thus, five clocking zones would take five quarters of a cycle i.e. 1.25 clocks. The latency for the whole design is computed differently when latencies from inputs to output are different. In such a case, the latency of whole design would be the worst-case latency i.e. the latency of the input-output path that takes most clock cycles.

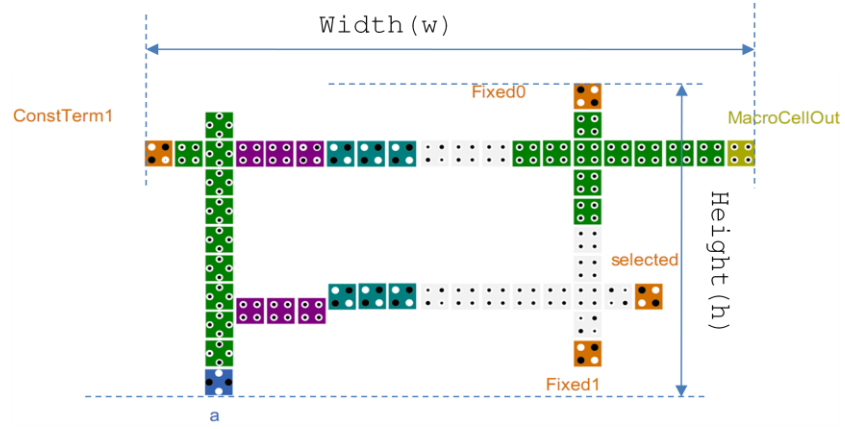
**Area:** As reported by QCA Designer, the area of QCA-Multiplexer ( $h \times w$ ) shown in Figure 6-1(a) is  $63192.98 \text{ nm}^2$  or  $0.06 \mu\text{m}^2$ . Total number of QCA-cells in this design is 56, which gives a cell-density of  $890 \text{ cells}/\mu\text{m}^2$ . The throughput is defined as number of clock cycles between successive output appearances or outputs per clock cycle. These properties are summarized in table 6-1.

<b>QCA-Multiplexer Macro-Cell</b>	
Number of QCA-cells	56
Worst-case-latency (clocks)	1.25
Throughput (output/clock cycle)	1
Area ( $\text{nm}^2$ )	62964.00
Cell-density ( $\text{cells}/\mu\text{m}^2$ )	890

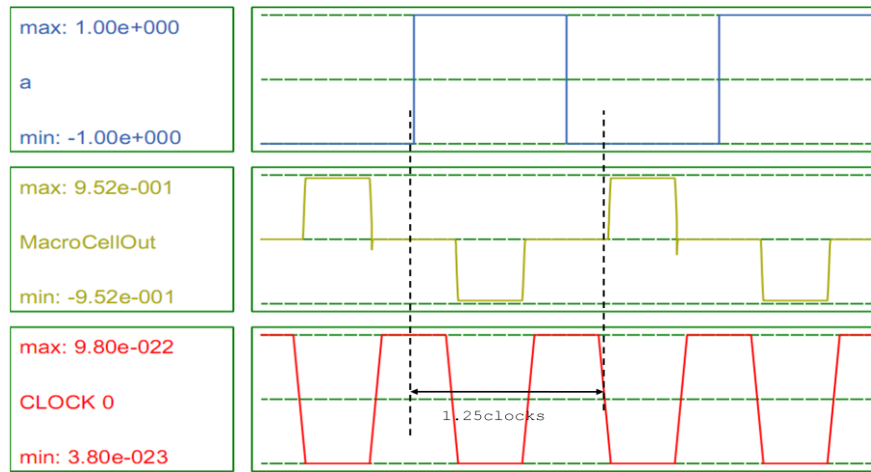
**Table 6-1 QCA-Multiplexer macro-cell characteristics.**

### **6.1.2 QCA-PLA AND-plane macro-cell**





(a) QCA-PLA AND-plane macro-cell.



(b) Simulation waveforms: a to MacroCellOut latency.

**Figure 6-2 Latency computation for QCA-PLA AND-plane macro-cell.**

**Latency:** The AND-plane was assigned clocking zones such that the variable wire of an AND-plane macro-cell is synchronized with product-term coming from pervious AND-plane macro-cell with respect to the clocking zones. This enables creating a horizontal array of AND-plane macro-cells for any number of variables without adjusting the clocking within each macro-cell to synchronize literal and incoming product-term with the AND-gate. In the QCA-PLA AND-plane macro-cell of Figure 6-2(a), the signal

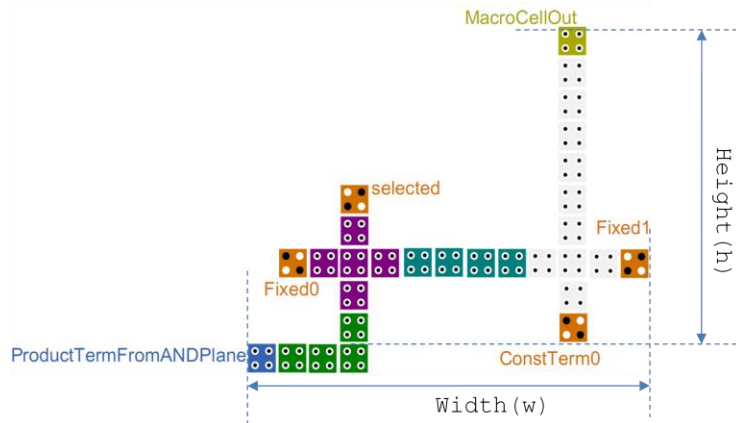
ConstTerm1 or product-term reach the AND-gate as the same time as the literal, if selected. Thus the latency of the macro-cell is same from incoming product-term and literal to the output which is the worst-case latency of this macro-cell. In the waveform of figure 6-2(b), the output MacroCellOut is the output of the right-most normal-mode cell. Since, for this particular cell the incoming product-term is 1 and select-bit is 0 i.e. literal selected, the output MacroCellOut is would be logical AND of two. The latency for this circuit is 1.25 clock cycles as shown in figure 6-2(b) as computed by counting the number of clocking zone, from input product-term or the literal to the output, which is five.

**Area:** As reported by QCADesigner, the area of QCA-PLA AND-Plane macro-cell ( $h \times w$ ) shown in Figure 6-1(a) is  $86990.10 \text{ nm}^2$  or  $0.09 \mu\text{m}^2$ . Total number of QCA-cells in this design is 49, which gives a cell-density of  $627 \text{ cells}/\mu\text{m}^2$ . These properties are summarized in table 6-1.

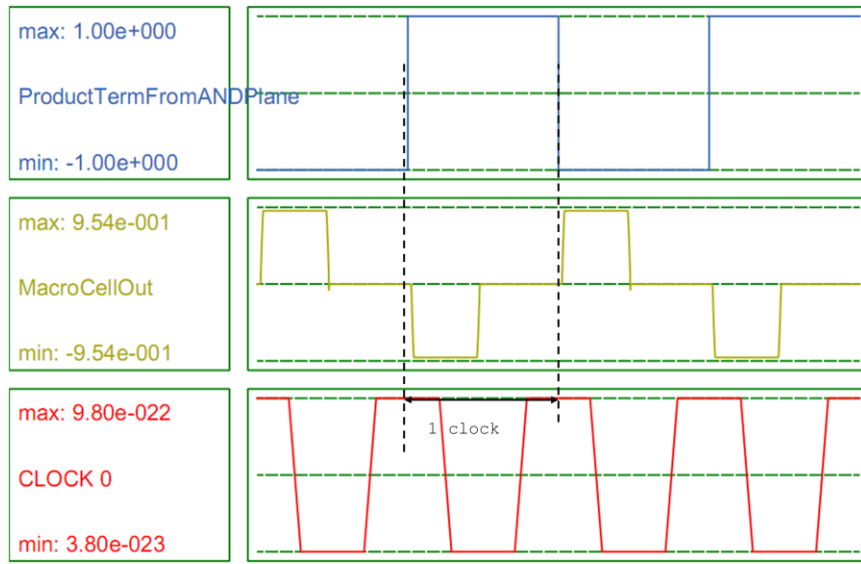
<b>QCA-PLA AND-Plane Macro-Cell</b>	
Number of QCA-cells	49
Worst-case-latency (clocks)	1.25
Throughput (output/clock cycle)	1
Area ( $\text{nm}^2$ )	78218.44
Cell-density ( $\text{cells}/\mu\text{m}^2$ )	627

**Table 6-2 QCA-PLA AND-plane macro-cell characteristics.**

### **6.1.3 QCA-PLA OR-plane macro-cell**



(a) QCA-PLA OR-plane macro-cell.



(b) Simulation waveforms: ProductTermFromANDPlane to MacroCellOut latency.

**Figure 6-3 Latency computation for QCA-PLA OR-plane macro-cell.**

**Latency:** In Figure 6-2(a) ProductTermInFromANDPlane is one of the product-terms from the AND-plane which becomes a part of SOP-term, if selected, generated in OR-plane. The MacroCellOut is the output of this macro-cell. MacroCellOut is the logical-OR of selected product-term input of this particular macro-cell and the SOP-term, generated by OR-plane macro-cells below this cell. The correct SOP term from the

OR-macro-cell at lower level is synchronized with the ProductTermInFromANDPlane when they reach the OR-gate. Thus, the latency is same for both and is one clock cycle shown in Figure 6-2(b). The number of clocking zones is four which equates to one clock cycle delay.

**Area:** As reported by QCA Designer, the area of QCA-PLA OR-Plane macro-cell ( $h \times w$ ) shown in Figure 6-1(a) is  $57032.72 \text{ nm}^2$  or  $0.06 \mu\text{m}^2$ . Total number of QCA-cells in this design is 29, which gives a cell-density of  $508 \text{ cells}/\mu\text{m}^2$ . These properties are summarized in table 6-1.

QCA-PLA OR-Plane Macro-Cell	
Number of QCA-cells	29
Worst-case-latency (clocks)	1
Area ( $\text{nm}^2$ )	57042.24
Cell-density ( $\text{cells}/\mu\text{m}^2$ )	508

**Table 6-3 QCA-PLA AND-plane macro-cell characteristics.**

Property / Cell-type	Multiplexer Macro-Cell	AND-Plane Macro-cell	OR-Plane Macro-Cell
<b>Number of QCA-cells</b>	56	49	29
<b>Worst-case-latency (clocks)</b>	1.25	1.25	1
<b>Area (<math>\text{nm}^2</math>)</b>	62964.00	78218.44	57042.24
<b>Cell-density (<math>\text{cells}/\mu\text{m}^2</math>)</b>	890	627	508

**Table 6-4 Combined table of macro-cell characteristics.**

## **6.2 QCA-PLA Latency and Throughput Calculations**

### **6.2.1 QCA-PLA latency calculation**

The complete layout of QCA-PLA presented in section 4.4.2.3 has a regular structure which scales only based on the number of product-terms in the function and the number of literals in the function. Thus the latency of a QCA-PLA can be computed based on aforementioned factors.

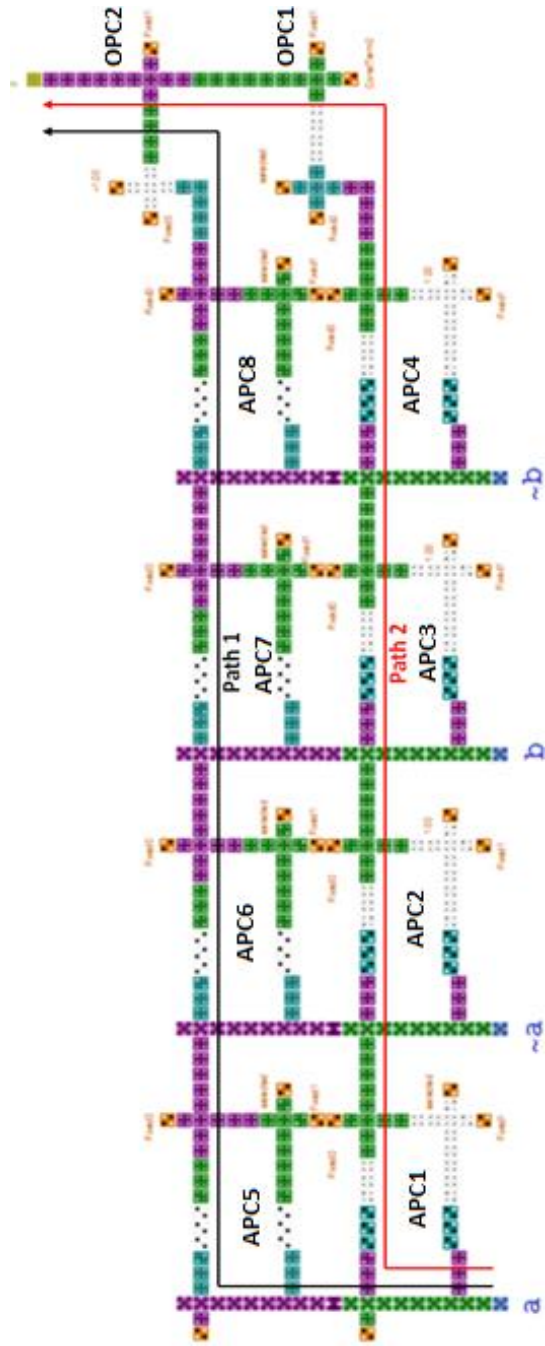


Figure 6-4 QCA-PLA latency calculation.

As discussed in earlier chapters, due to the nature of QCA technology, a CA wire is equivalent to back to back delay elements in silicon domain thus it takes multiple cycles

for data to transfer from one end to another. Thus a change in signal value of any input would take multiple quarter clock-cycles to travel from input to computation cells in different AND-Plane macro-cells. Figure 6-4 shows the QCA-PLA layout of an arbitrary function where both levels of AND-Plane implement product-terms. APC stands for AND-Plane macro-cell and OPC stands for OR-Plane macro-cell. Assuming that variable 'a' is present in both the terms, the selected-bit at both levels for variable 'a' would be set to logic 0 i.e. variable selected. The variable 'a' would, now, travel from the input cell to both the level and may be used at different AND-gates of AND-Plane cells at both levels for computing product-terms. Even though, it may be used as a part of logical AND at some AND-Plane cells, it can be thought of as travelling from input cell all the way to the output F. Path 1 and Path 2 represent the flow direction of variable 'a'. Thus variable 'a' would take as many quarter cycles on each path to reach output F as the number of clocking zones. However, it can be observed that for both paths 1 and 2 the number of clocking zone from input to output is equal. This implies that latency of variable 'a' from input to output on both paths is same. Similar would be the case with other input variables. Without the loss of generality, it can be deduced that for any number of product-terms and literals, the latency from an input variable to the output F through any level would be same. To calculate the latency for variable 'a', if other inputs are kept constant and only 'a' changes, the output, corresponding to the changed 'a' value, would appear at cell F after clock cycles equal to (number of clocking zone on path)/4. On the other hand, if all variables are kept constant but only '~b' changes, then the total number of clocking zones in the path from its input cell to the

output are the least. Hence, it can be understood that the design would have worst-latency when variable 'a' changes (farthest from the output) and best-latency when variable '~b' changes (closest to the output). The number of clocking zones between input and output would depend on the number of product-terms and variables in a function. From figure 6-4, general equations for the worst-case and best-case latencies for the QCA-PLA design considered in this thesis can be computed. Thus, for a Boolean function of 'N' number of variables with 'P' number of product-terms, the latencies are given as:

**Worst-case-latency (farthest input to output) =**

$$\{(N * 8) + P + 4\} \text{ quarter clocks or } \{(N * 2) + \frac{P}{4} + 1\} \text{ clock cycles (eq. 1)}$$

**Best-case-latency (closest input to output) =**

$$(8 + P) \text{ quarter clocks or } (2 + \frac{P}{4}) \text{ clock cycles (eq. 2)}$$

With two variables and two product-terms, it is easy to calculate the worst-case-latency for Boolean function realized in Figure 6-4 which is 22 quarter clocks or  $5\frac{1}{2}$  clock cycles. Only worst-case-latencies would be considered for analysis from this point on. Since it is not possible to predict which inputs would change and which ones would not and at what time instants, the term best-case-latency is not significant. It is important to note that, since the signals don't propagate instantaneously, once a set of input is applied they cannot be changed on the next clock cycle. This is because the inputs which are farther



away from the output would take longer to reach an AND-gate in an AND-Plane macro-cell to form a product term with an input which is closer to the output. If the closer input is changed before the farther input reaches the AND-gate, then an incorrect product-term would be generated as it would be the logical-AND of input signals from different time instances. For example, if a product term ' $a \cdot \sim b$ ' is realized at upper level in figure 6-4 thus this product term would be generated at APC8. Assume that the initial values of ' $a$ ' and ' $\sim b$ ' both 1, then input cell of ' $\sim b$ ' should remain constant at logic 1 for as many clock cycles such that  $a = 1$  takes to reaches the AND-gate in APC8 at the same time as ' $\sim b$ ' = 1 to correctly generate ' $a$ ' (= 1) AND ' $\sim b$ ' (= 1) equal to 1. Say, if value of ' $\sim b$ ' is changed to 0 earlier than minimum clock cycles to be constant, then ' $\sim b$ ' = 0 would start propagating towards the AND-gate in APC8. In such a case, by time ' $a$ ' = 1 reaches the AND-gate, ' $\sim b$ ' = 0 would also reach the AND-gate and the logical-AND would be performed on ' $a$ ' (= 1) AND ' $\sim b$ ' (= 0) and the computation for ' $a$ ' (= 1) AND ' $\sim b$ ' (= 1) will be missed which is an incorrect behavior. Thus, there is a minimum number of clock cycles, only after which ' $\sim b$ ' can change. This constraint directly affects the throughput of the circuit since it puts a lower bound on how fast different sets of inputs can be applied to the input cells. The throughput computations for the considered QCA-PLA are presented in section 6.2.2 below.

### **6.2.2 QCA-PLA throughput calculation**

Since the relative latency of variables reaching a particular AND-gate is the same, the above analysis would hold for 'a . ~b' realized at lower level as well. It would be easier for a reader to understand the throughput analysis if it is done using the lower level as reference and the analysis, again, would hold for all the level due to constant relative latencies between variables. Consider again the Path 2 which, in this case, realizes the product-term 'a . ~b' and numeral represent the clocking zone numbers. Assuming that a function is implemented as cubes of literals, the throughput would be least when number of clock cycles to wait, before applying the next set of input signals, is the highest. This happens when the distance between the literal of a product-term is largest which, in this case, is between variable 'a' and variable '~b'. In other words, throughput would be higher when '~b' can be changed earlier. As described in previous section, variable '~b' can only change when 'a' had travelled sufficiently close enough to '~b' such that both 'a' and '~b' from same time instant would reach the AND-gate at clocking zone 17<sup>th</sup> in figure 6-5. Finding solution to the puzzle of "when is the earliest can '~b' change for highest throughput?" can also be viewed as solving "how many clocking zone should 'a' have travelled towards '~b' on Path 2 before '~b' can change for correct behavior?". Looking closely to the figure 6-5, suppose 'a' has reached clocking zone 12<sup>th</sup> which is in *hold* phase then clocking zone 13<sup>th</sup> would be in *switch* phase. It took 12 quarter cycles for variable 'a' to reach and then, if on quarter cycle 13<sup>th</sup> '~b' is changed, the old 'a' and new '~b' would proceed towards the AND-gate (at clocking zone 17<sup>th</sup>) in 14<sup>th</sup> quarter of clock cycle which is incorrect. This implies that even at 13<sup>th</sup> quarter of clock cycle, '~b' should be kept constant. Thus '~b' should be changed at 14<sup>th</sup> quarter of clock cycle to be

in synchronization with 'a' in its propagation towards the AND-gate i.e. when clocking zone 14<sup>th</sup> is in *switch* phase. However, in 14<sup>th</sup> quarter of clock cycle, when clocking zone 14<sup>th</sup> is in *switch* phase, clocking zone 13<sup>th</sup> cannot be *switched* to new inputs as it would be in *hold* phase. The clocking zone 13<sup>th</sup> cannot be switched with new ' $\sim b$ ' for two more quarters of clock because, after hold phase, clocking zone 13<sup>th</sup> would go through *release* and *relax* phases as well. Hence, the earliest ' $\sim b$ ' can be switched to new value is at 17<sup>th</sup> quarter of clock cycle which is when clocking zone 13<sup>th</sup> enters into *switch* phase again. In such a case, both 'a' and ' $\sim b$ ' have reached the AND-gate synchronously with clocking zone 16<sup>th</sup> in *hold* phase and 17<sup>th</sup> in *switch* phase. This means that ' $\sim b$ ' has to be kept constant for four clock cycles after changing i.e. until 'a' reaches clocking zone 16<sup>th</sup>.

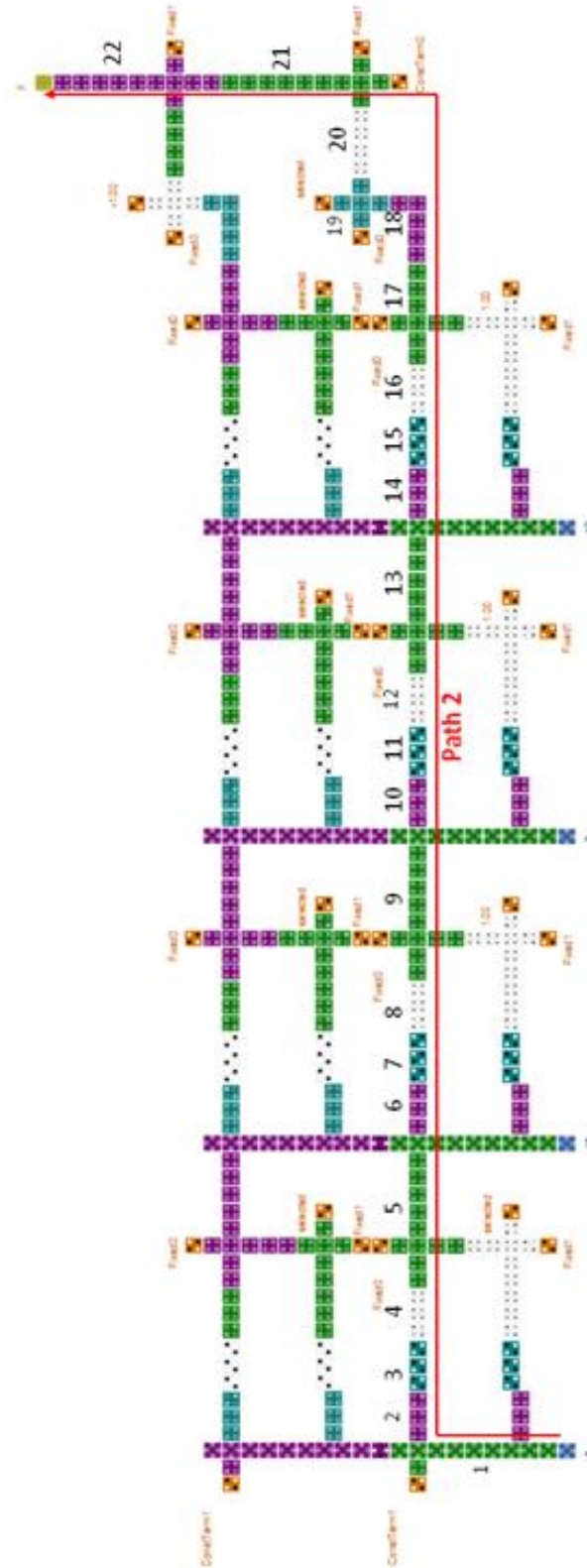


Figure 6-5 QCA PLA throughput computation.

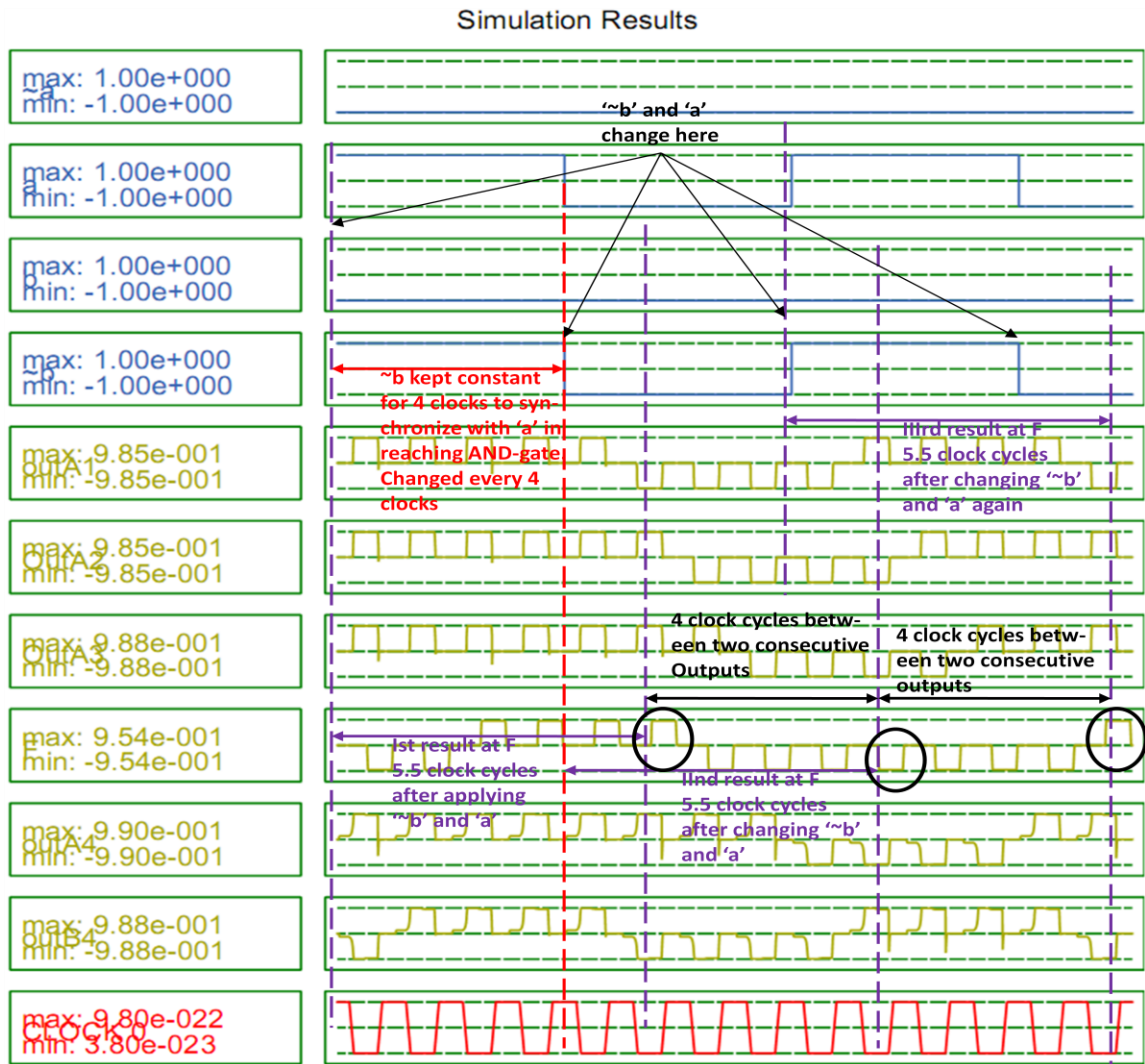
Since the worst-case throughput scenario is being discussed and ' $\sim b$ ' can be changed a few clock cycles earlier than waiting for previous output to appear at output cell F, the propagation of ' $a$ ' towards ' $\sim b$ ' from new set of values is overlapped with the propagation of older ' $a . \sim b$ ' towards the output cell F. This results in value of ' $a . \sim b$ ' to appear at the output cell F earlier than the worst-case-latency. In this particular case, the result for the new set of ' $a$ ' and ' $\sim b$ ' would appear at the output 16 quarter cycles after previous result. Thus the throughput would be one output per 16 clock cycles. A generalized solution of the throughput of QCA-PLA considered in this thesis, for a Boolean function of ' $N$ ' number of variables with ' $P$ ' number of product-terms can be given as:

$$\begin{aligned} \text{Throughput} &= \text{one output per } (8 * N) \text{ quarter cycles or} \\ &\text{equivalently} \\ &\text{one output per } (2 * N) \text{ clock cycles} \end{aligned}$$

**(eq. 3)**

The simulation results from QCADesigner for a QCA-PLA implementing ' $a . \sim b$ ' at Path 2 in Figure 5-6 is shown in Figure 6-6. The signal values provide the confirmations on above claims. The values of ' $\sim a$ ' and ' $b$ ' can be ignored since they were not selected. The outputs OutA1, OutA2, OutA3, OutA4 and OutB4 are intermediate signals outputted to observe the signal flow and can also be ignored for the purposes of understanding the

latency and throughput. The purple-text shows that there is a latency of  $5\frac{1}{2}$  clock cycles from change of inputs to the appearance of result at output cell F. All three outputs appear  $5\frac{1}{2}$  clock cycles after changing the inputs. Also, red-text indicates that ' $\sim b$ ' was kept constant for four clock cycles to generate the correct product-term. The text in black indicates the number of clock cycles between two consecutive correct outputs. The signal values in black ovals correspond to the valid output for applied set of input variables. This supports the correctness of equations 1 and 3 deduced earlier.



**Figure 6-6 Simulation results for latency and throughput of QCA-PLA.**

### **6.3 QCA-Shannon-Lattice Latency and Throughput Calculations**

A calculation procedure, similar to one used for QCA-PLA latency and throughput computation above, can be used for QCA-Shannon-Lattice.

#### **6.3.1 QCA-Shannon-Lattice latency calculation**

Shannon Lattice's regular structure scales depending on the number of levels in the lattice diagram. In turn, the number of levels in a lattice diagram depends on the type of a Boolean function. For the symmetric Boolean functions the number of levels is directly equal to the number of variables but for the non-symmetric functions the number of levels depends on the variable ordering, as described in section 3.5.2. It is easier to calculate the latency for QCA-Shannon-Lattice as compared to QCA-PLA since it involves only one type of macro-cells and terms implemented in multiple levels. The latency can be calculated by counting the number of clocking zones on the longest path from input to the output. Using the example of QCA-Shannon-Lattice for an arbitrary function from 4.4.1, it can be seen from Figure 6-6, that total number of clocking zones on the longest path is 13 thus it would take 13 quarter clock cycles to propagate the data from input to output. Thus the latency is 13 quarter clocks or  $3\frac{1}{4}$  clock cycles. The

generalized equations for latency of QCA-Shannon-Lattice of a Boolean function of  $N$  variables and  $L$  levels can be given as follows:

For Symmetric functions ( $N = L$ )

Latency =  $(4 * N) + 1$  quarter clocks or  $(N + \frac{1}{4})$  clock-cycles  
(eq. 4)

For Non-symmetric functions ( $N$  may not be equal to  $L$ )

Latency =  $(4 * L) + 1$  quarter clocks or  $(L + \frac{1}{4})$  clock-cycles  
(eq. 5)

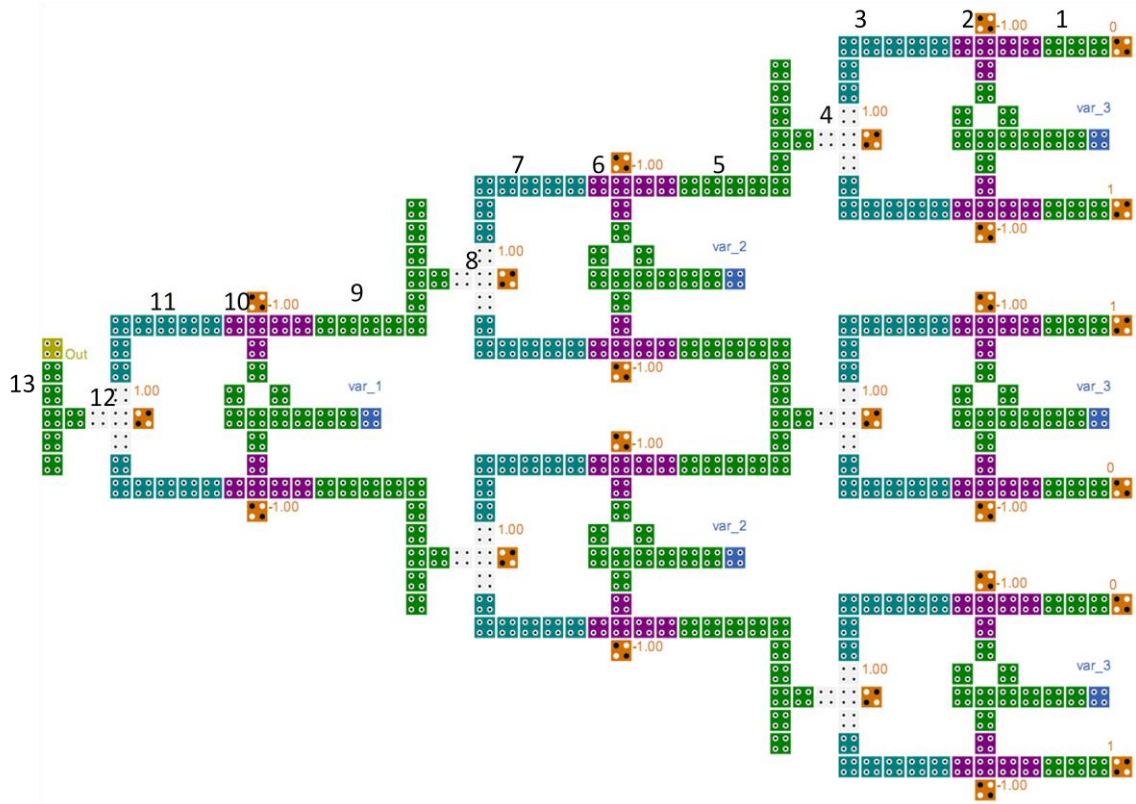


Figure 6-7 QCA-Shannon-Lattice latency calculation.



### 6.3.2 QCA-Shannon-Lattice Throughput Calculation

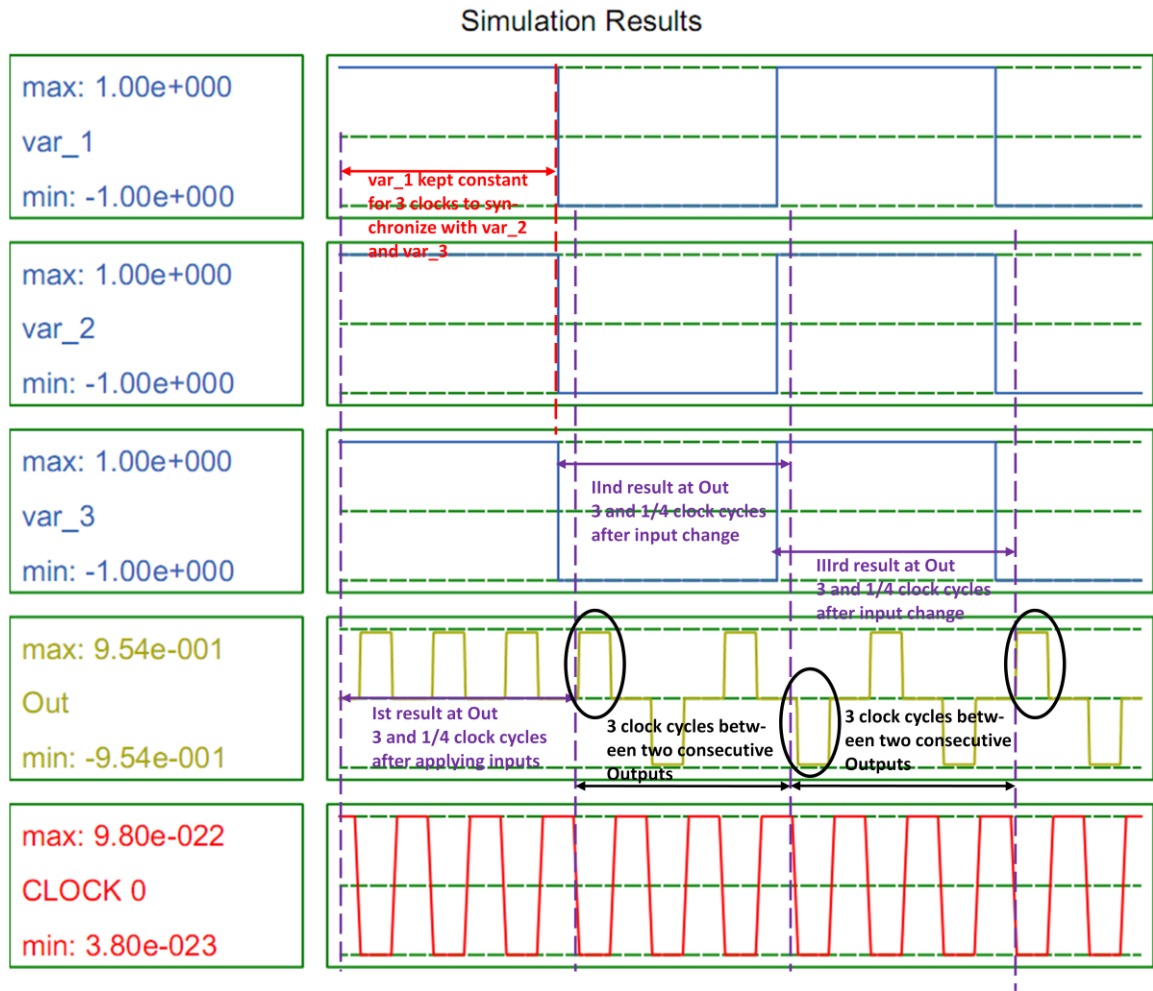
The same observations can be used here as in the case of throughput calculations for QCA-PLA. One can see that for highest throughput of QCA-Shannon-Lattice in Figure 6-6, after applying a set of inputs, `var_1` should be *switched* as soon as possible but only in synchronization with other input from the same time instant. This can only be done when the result from previous result has reached clocking zone 13<sup>th</sup> which is the final output. Thus the new set of inputs can only be overlapped for one quarter of clock cycle giving a throughput of one result per 12 quarter clock cycles or one result per three clock cycles. The generalized equation for QCA-Shannon-Lattice in this thesis for a Boolean function of N variables and that can be decomposed into L levels of lattice diagram can be given as:

Throughput = one result per (L \* 4) quarter clocks or  
equivalently  
one result per L clock cycles; For symmetric functions (N = L) **(eq. 6)**

The simulation results from QCADesigner for a implementing an arbitrary QCA-Shannon-Lattice of Figure 6-7 is shown in Figure 6-8. The Boolean function implemented by QCA-Shannon-Lattice in Figure 6-7 is:

$$Out = var_1 \cdot \overline{var_2} \cdot \overline{var_3} + \overline{var_1} \cdot \overline{var_2} \cdot var_3 + var_1 \cdot var_2 \cdot var_3 + \overline{var_1} \cdot var_2 \cdot \overline{var_3}.$$

The signal values provide the confirmations on above claims about the latency and throughput. The signal values in black ovals correspond to the valid output for applied set of input variables. It can be seen that after the change in inputs, the valid output appears at the output cell 'Out' after three and a quarter clock cycles. The purple text shows the delay between the input change and appearance of valid output signal. The text in black confirms the fact that the consecutive outputs appear with a delay of three clock cycles thus the throughput is one result per three clock cycles. The lattice has three levels and throughput is also three thus the equation 6 holds.



**Figure 6-8 Simulation results for latency and throughput of QCA-Shannon-Lattice.**

#### **6.4 Comparison Table: QCA-Shannon-Lattice and QCA-PLA**

This section presents the comparison between implementing Boolean functions using QCA-PLA and QCA-Shannon-Lattice using the equations derived above. The comparison is performed for latency, throughput, area, number of QCA-cell and cell-density. It was already mentioned in earlier sections that the QCA-Shannon-Lattice

layout is a function of levels in the lattice diagram. For symmetric functions it is easier to generate a lattice diagram because adjacent children nodes of separate parents are always equal. This results in a lattice diagram of symmetric functions such that the number of levels is always equal to number of variables in the Boolean function. This fact is not always true for non-symmetric functions and the number of levels in their lattice diagram depends heavily on variable ordering. The algorithms implemented in the tool (developed for this thesis) for Shannon-Decomposition of non-symmetric functions are currently not efficient to generate a variable ordering that converges the decomposition for non-symmetric functions. Due to this issue only symmetric functions are considered for comparative study. Since there are no or very few benchmarks which are symmetric, some synthetic symmetric functions were generated for this analysis. The benchmark functions in the table below are custom if mentioned otherwise with their suite name. The area for QCA-Shannon-Lattice is the area of the rectangle that encloses the layout and not the space covered only by the layout. It is important to note that Boolean functions were minimized for using ESPRESSO minimization software for QCA-PLA and not for QCA-Shannon-Lattice. This is evident from the difference in the number of product terms for same function in both tables 6-5 and 6-6. The reasoning behind this decision is that, for random functions, the QCA-Shannon-Lattice would not benefit from minimization. This is because, for the symmetric functions, although the number of levels would be equal for any order to variables, the area i.e. the width at each level would depends on the order of variables used for Shannon decomposition. On the other hand, QCA-PLA layout would always benefit from logic minimization because minimization could possible reduce the number of min-terms. As shown earlier, the

number of min-terms translates into number of levels of the QCA-PLA thus affecting area. The QCA layouts for some of these benchmarks, generated automatically by the tool, are included in Appendix of this thesis.

Benchmarks	Function Density	# of inputs	# of terms	Latency (clock cycles)	Throughput (n) One output per n clocks	Area ( $\mu m^2$ )	# of QCA-cells	Cell density (cells/ $\mu m^2$ )
<b>exam1_d</b> (MCNC)		3	4	$3\frac{1}{4}$	3	0.66	336	510
<b>xor5_d</b> (MCNC)		5	16	$5\frac{1}{4}$	5	1.87	840	450
$S^{0,1,2}f(6)$	Low	6	22	$6\frac{1}{4}$	6	1.83	728	398
$S^{0,1,5,6}f(6)$	Medium	6	14	$6\frac{1}{4}$	6	2.29	896	392
$S^{2,3,4}f(6)$	Medium	6	50	$6\frac{1}{4}$	6	2.29	952	416
$S^{4,5,6}f(6)$	High	6	22	$6\frac{1}{4}$	6	1.83	728	398
$S^{0,1}f(4)$	Low	4	5	$4\frac{1}{4}$	4	0.76	336	443
$S^{2,3}f(4)$	Medium	4	10	$4\frac{1}{4}$	4	1.07	448	419
$S^{3,4}f(4)$	High	4	5	$4\frac{1}{4}$	4	0.76	336	443
$S^{0,1,2}f(8)$	Low	8	37	$8\frac{1}{4}$	8	3.05	1064	349
$S^{0,1,7,8}f(8)$	Medium	8	17	$8\frac{1}{4}$	8	4.84	1456	301
$S^{3,4,5}f(8)$	Medium	8	182	$8\frac{1}{4}$	8	3.67	1568	428
$S^{6,7,8}f(8)$	High	8	37	$8\frac{1}{4}$	8	3.05	1064	349
$S^{0,1,2}f(10)$	Low	10	56	$10\frac{1}{4}$	10	4.58	1400	306
$S^{8,9,10}f(10)$	High	10	56	$10\frac{1}{4}$	10	4.58	1400	306

**Table 6-5 QCA-Shannon-Lattice benchmark data.**

Benchmarks	Function Density	# of inputs	# of terms	Latency (clock cycles)	Throughput (n) One output per n clocks	Area ( $\mu m^2$ )	# of QCA-cells	Cell density (cells/ $\mu m^2$ )
<b>exam1_d (MCNC)</b>		3	4	8	6	2.38	1292	543
<b>xor5_d (MCNC)</b>		5	16	15	10	13.11	8304	634
$S^{0,1,2}f(6)$	Low	6	15	$16\frac{3}{4}$	12	14.83	9255	625
$S^{0,1,5,6}f(6)$	Medium	6	12	16	12	12.01	7404	617
$S^{2,3,4}f(6)$	Medium	6	16	17	12	15.77	9872	626
$S^{4,5,6}f(6)$	High	6	15	$16\frac{3}{4}$	12	14.83	9255	625
$S^{0,1}f(4)$	Low	4	4	10	8	3.14	1684	536
$S^{2,3}f(4)$	Medium	4	6	$10\frac{1}{2}$	8	4.40	2526	575
$S^{3,4}f(4)$	High	4	4	10	8	3.14	1684	536
$S^{0,1,2}f(8)$	Low	8	28	24	16	35.57	22764	640
$S^{0,1,7,8}f(8)$	Medium	8	16	21	16	20.71	13008	629
$S^{3,4,5}f(8)$	Medium	8	57	$31\frac{1}{4}$	16	72.94	46341	636
$S^{6,7,8}f(8)$	High	8	28	24	16	35.57	22764	640
$S^{0,1,2}f(10)$	Low	10	45	$32\frac{1}{4}$	20	70.73	45405	642
$S^{8,9,10}f(10)$	High	10	45	$32\frac{1}{4}$	20	70.73	45405	642

**Table 6-6 QCA-PLA benchmark data.**

From Tables 6-5 and 6-6, it can be observed that QCA-Shannon-Lattice is much better than QCA-PLA in every respect for each benchmark. This is primarily because of the symmetric nature of all the benchmark functions which results in decomposition into lattice diagram with low number of levels. A non-symmetric function may result in a large number of levels in its lattice diagram due to re-introduction of the variables to perform Join-Vertex operation. The other issue is that only single output PLA functions are considered although multiple output PLAs can save area by sharing logic terms. The following inferences, however, can be made from the values in Table 6-5.

**Latency:** For QCA Shannon-Lattice, the latency is directly proportional to the number of variables in the Boolean function and insensitive to the number of product-terms. For QCA-PLA, the latency depends on both the number of variables and product-terms in the Boolean function. But the increase in the latency is more sensitive to the increase in the number of variables than the number of product terms (it can be inferred from the slopes of graph lines in Figures 6-9 and 6-10). This is because the increase in the number of variables causes AND-Plane macro-cells to be added horizontally at two times per addition of variable, but addition of product terms results in increase of number of OR-Plane cells at the same rate. Also, the number of clocking zones are much more in an AND-Plane macro-cell than an OR-Plane macro-cell which results in greater increase in the number of clocking zone, a variable has to travel from input to output, when number of variables increase. It can, then, be deduced that the QCA-PLA has higher latency than QCA-Shannon-Lattice because the QCA-Shannon-Lattice does not depend on the number of product-terms which QCA-PLA does and the increase in latency with the increase in number of variables in QCA-Shannon-Lattice is not two times of macro-cell latency per variable as it is for QCA-PLA. This result is summarized in the latency comparison graph of benchmarks functions for both realizations in Figure 6-7. Even with minimization of each benchmark function for PLA, its latency is always worse than Shannon Lattice because Shannon lattice is insensitive to number of product terms for latency. It can be observed that the latency of QCA Shannon lattice is same for the functions with same number of variables and different product-terms and only changes when number of variables changes. For QCA-PLA, the latency changes for functions with same variable count but different product-term count.

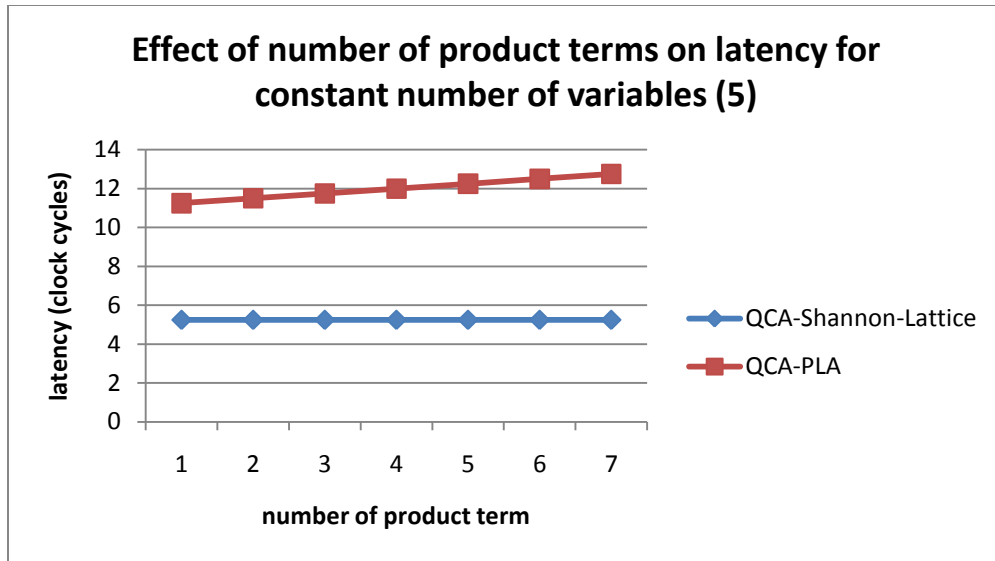


Figure 6-9 Number of product-terms vs. latency.

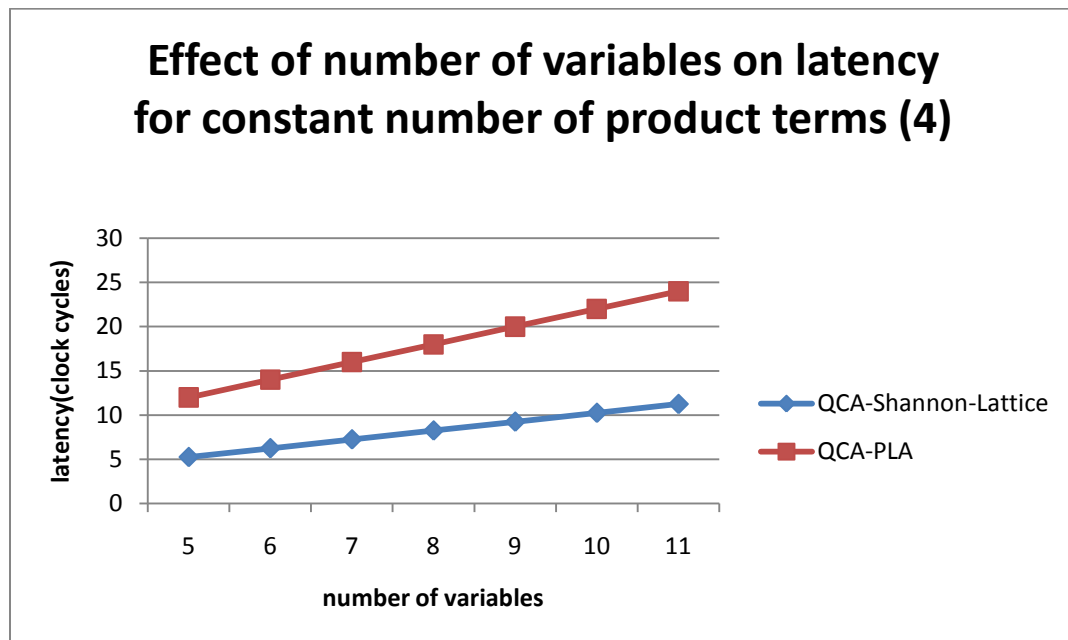


Figure 6-10 Number of variables vs. latency.



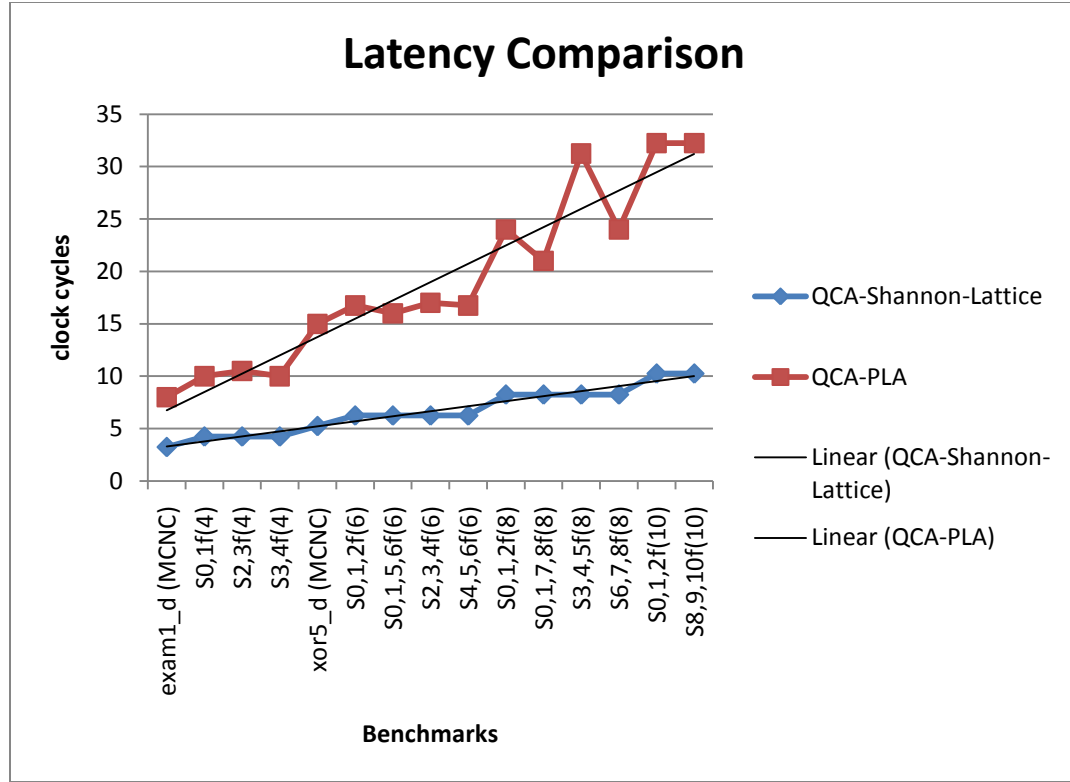


Figure 6-11 Latency comparison of QCA-PLA and QCA-Shannon-Lattice.

**Throughput:** For both QCA-PLA and QCA-Shannon-Lattice, the throughput is inversely proportional to the number of variables in the Boolean function. It is simply because the higher the number of variables in the function, more would the one variable close to output have to wait to synchronize with the signals coming from previous macro-cells. Moreover, the rate of decrease of throughput with increase in number of variables is higher for QCA-PLA since each variable increase results in the addition of two and-plane cells which implies more wait clocks for the last variables, referring Figure 6-12. It is important to note that the throughput plots show number of clock cycles between consecutive outputs, thus, higher value in plot implies a worse throughput. In QCA-Shannon-Lattice, the number of product terms is insignificant for throughput. In QCA-

PLA, the number of product-terms does not affect the throughput because of pipelining effect, where the result propagation to the output cell is overlapped with propagation of variables to generate new product value. Thus irrespective of the number of product-terms, the clock cycle interval between the results remains the same since it is computed for the worst-case, referring Figure 6-13. The throughput comparison for both types of logic implementation of benchmarks is shown in Figure 6-14. The throughput is always lower for the benchmarks when implemented using PLA. However in this case, since the throughput is only affected by the number of variables (and not by the number of product terms), slope of throughput graph-line for QCA-PLA is smaller than the slope of its latency graph-line. The QCA-Shannon-Lattice is only affected by the number of the variables in latency and throughput, the slopes of its latency and throughput graph-lines are equal, referring to Figures 6-11 and 6-14.

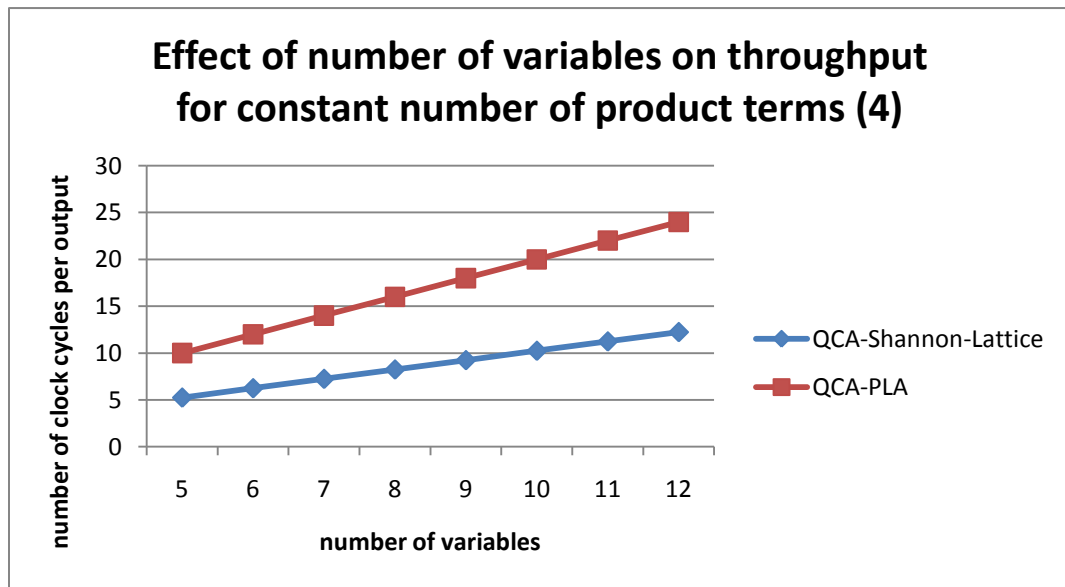


Figure 6-12 Number of variables vs. throughput.

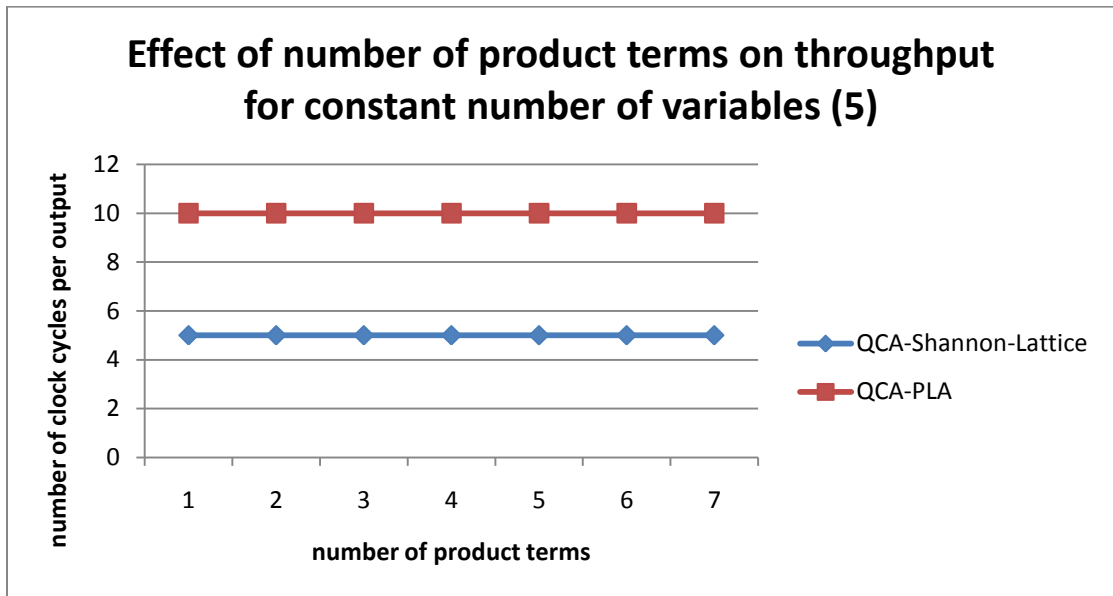


Figure 6-13 Number of product-terms vs. throughput.

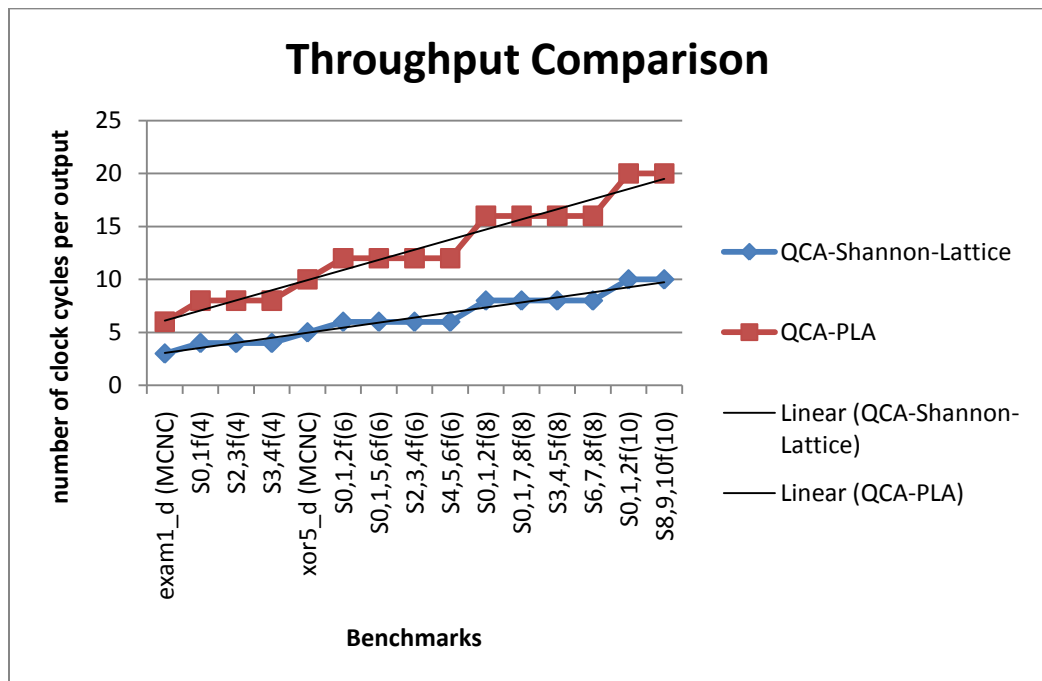
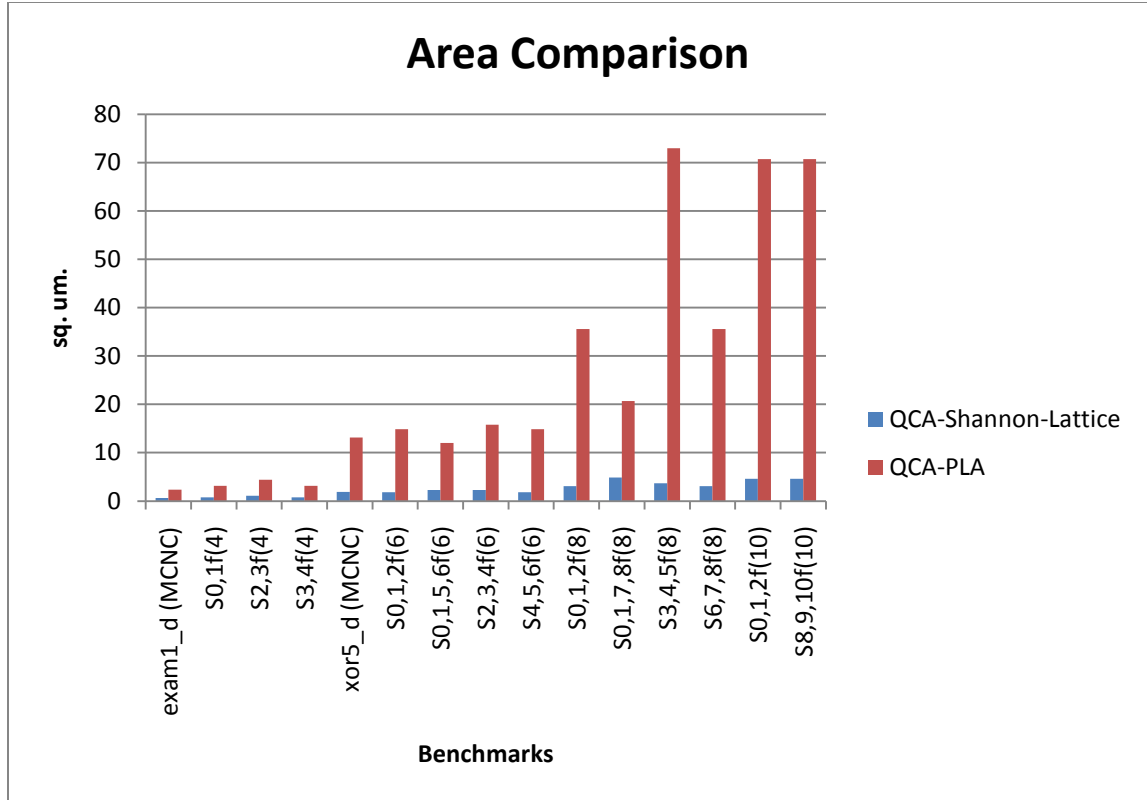


Figure 6-14 Throughput comparison of QCA-PLA and QCA-Shannon-Lattice.

**Area:** From table 6-6, it can be noticed that the proportional increase in the area of QCA-PLA is much more when the number of product-terms increases than due to the increase in number of variables in the function. This is evident because with an additional product-term, an entire new level of AND-Plane cell is added which contains AND-Plane macro-cells of other literals which may not be required for added product term, however, addition of one more variable in the function simply adds two AND-plane macro cells at every level. The number of levels in QCA-PLA depends on the number of product terms in the function which, in turn, depends on the logic minimization. The QCA-PLA area is always larger as the number of variables increase for the same number of product term due to reasons explained earlier. From the graph in figure 6-15, it can be observed that for QCA-PLA, the high-density and low-density functions have smaller area as compared to the medium density functions with same number of the variables. This is because the former two types get minimized to lower number of product terms than later type. This depends on how medium density function is generated. A medium density function generated by combining few terms of both low and high density functions does not necessarily conform to this observation. This is because in such a medium function, both low and high density terms get minimized well. For QCA-Shannon-Lattice as well, the area increases with the increase in the number of variables but with same number of variables, the area depends on the function's decomposition into lattice diagram.



**Figure 6-15 Area comparison of QCA-PLA and QCA-Shannon-Lattice.**

**Number of cells and cell-density:** QCA-PLA needs more QCA-cells to implement the logic because for each variable QCA-PLA requires two AND-Plane macro-cells: one for variable and other for its complement. The number of cells in QCA-PLA also depends on the number of product-terms because it decides the number of levels in QCA-PLA and increases the number of QCA-cells in both AND-Plane and OR-Plane. The number of cells in QCA-Shannon-Lattice depends on the number of variables in the function and its lattice diagram. In the worst case, the number of QCA Multiplexer macro-cell would be equal to the level index at each level of lattice diagram; however, it would still be lower

than QCA-PLA. The cell density, however, suggests that QCA-PLA is a denser design thus making an effective use of the real-estate.

## 6.5 Full-Adder (FA) implementations: regular structures and custom layout

This section presents the comparative analysis when implementing full-adder circuit using the regular structures and custom logic.

### 6.5.1 FA implementation: custom layout

The QCA implementation of the full-adder circuit with custom layout as proposed in [34] is shown in Figure 6-16.

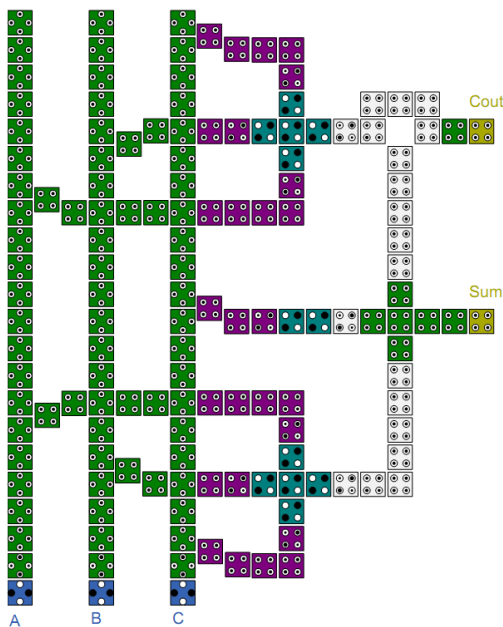


Figure 6-16 QCA FA<sup>15</sup>

<sup>15</sup> Figure 4-7 created using QCADesigner based on [34]

There are two reasons for choosing the full-adder for this analysis:

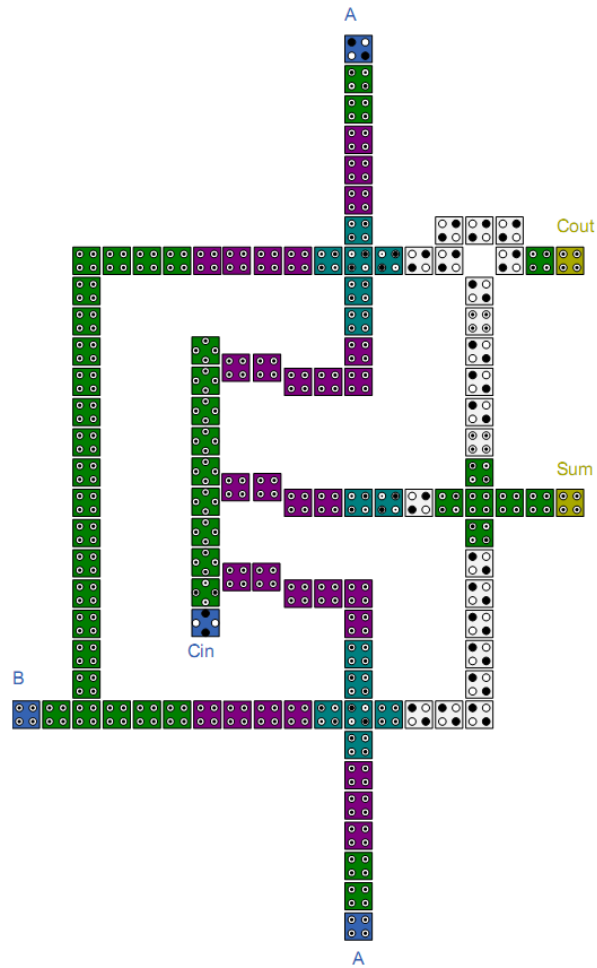
1. It is one of the most useful circuits in logic design.
2. The logic functions for both outputs of full-adder, sum and carry-out, are symmetric in nature. Only symmetric functions are considered in this thesis.

**Limitation:**

As for QCA-Shannon-Lattice and QCA-PLA designs in this thesis, the full-adder layout should produce correct outputs in simulations using QCADesigner. It was, however, observed that the behavior of full-adder of Figure 6-16 was inconsistent due to sneak path noise [37]. This problem required modification to the full-adder layout such that it produces a correct simulation behavior. The modified full-adder layout and its simulation waveforms are shown in Figure 6-17 and Figure 6-18 respectively. Table 6-7 shows the properties of the modified full-adder's custom layout. The modifications to the full-adder layout were made following the circuit design constraint of section 4.3 of this thesis.

The logic equation used is same for both the layouts in Figure 6-16 and 6-17 and are given by:

$$\begin{aligned} \text{Cout} &= \text{Majority}(A, B, \text{Cin}) \\ \text{Sum} &= \text{Majority}(\overline{\text{Cout}}, \text{Cin}, \text{Majority}(A, B, \overline{\text{Cin}})) \end{aligned}$$



**Figure 6-17 Modified FA custom layout**

Modified FA Custom Layout	
Number of QCA-cells	114
Worst-case-latency (clocks)	1.25
Throughput (output/clock cycle)	1
Area ( $\mu m^2$ )	0.23
Cell-density (cells/ $\mu m^2$ )	496

**Table 6-7 Modified FA custom layout characteristics**



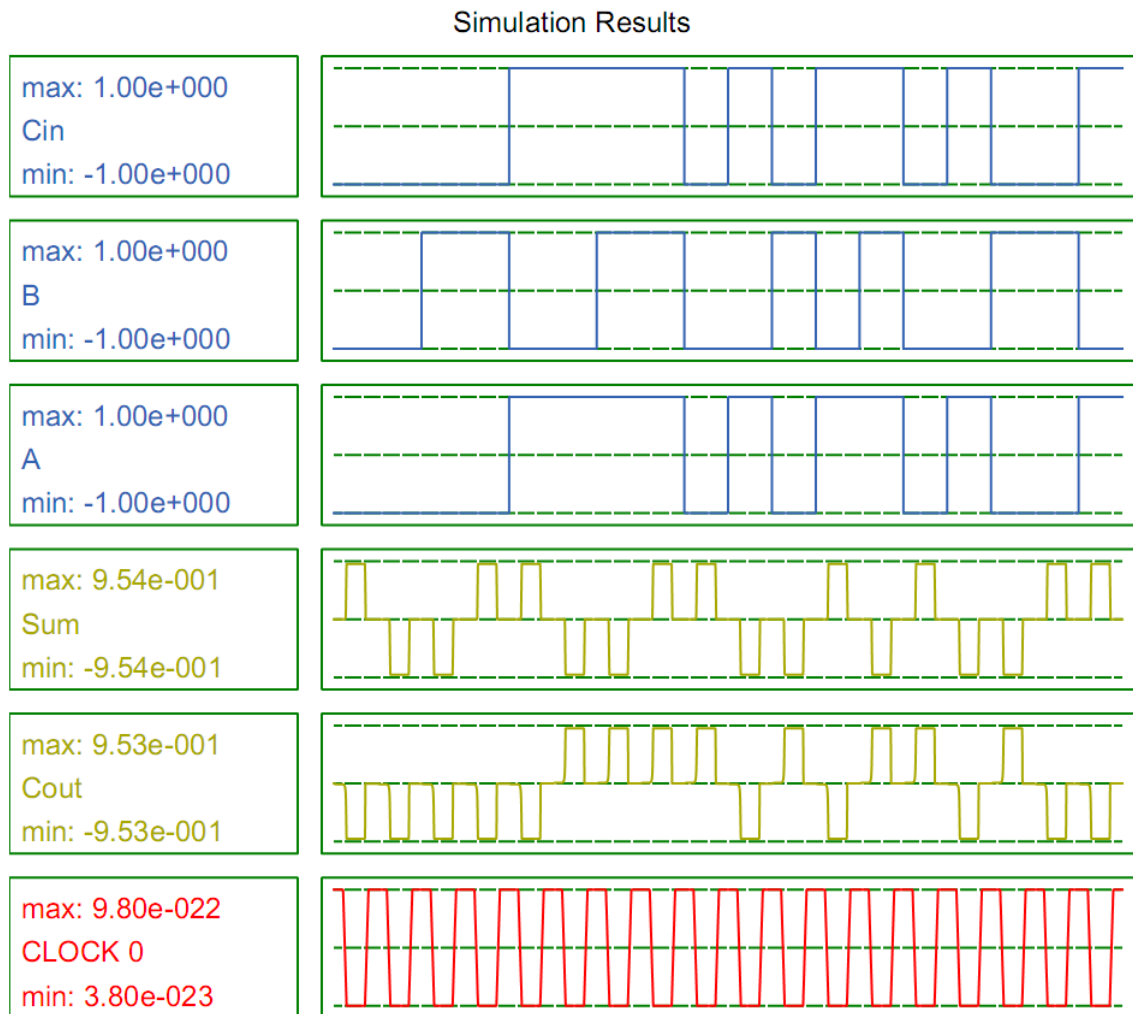


Figure 6-18 Simulation waveforms of modified FA generated using QCADesigner

### 6.5.2 FA implementation: regular structures

This section presents the full-adder realizations using QCA-Shannon-Lattice and QCA-PLA.

#### Limitation:

The full-adder circuit is a multiple output function (sum and carry-out) whereas only single output Shannon-Lattice and PLA are considered in this thesis. Thus, for the area comparison the total area of QCA-Shannon-Lattice or QCA-PLA implementation of full-adder is computed by adding the individual areas of sum and carry-out realizations together. Also, since the latency and throughput properties of sum and carry-out realizations in separate regular structures might be different, the worse specification of the two is considered as the overall specification for the implementation. Thus the analysis may not show an exact comparison but should still be able to provide a general idea.

The logic equation used is same for both QCA-Shannon-Lattice and QCA-PLA layouts are given by:

$$C_{out} = a.b + b.c + a.c$$

$$Sum = a.b.c + \bar{a}.\bar{b}.c + \bar{a}.b.\bar{c} + a.\bar{b}.\bar{c}$$

It should be noted that the logic equations for Sum and Cout in section 6.5.1 are different than the ones above but are equivalent. The equations in section 6.5.1 use majority logic to produce optimal circuit, however, the equations are required to be in the sum-of-product form for both Shannon-Lattice and PLA implementations. Thus, the sum-of-product logic expressions for Sum and Cout are used in following sections.

#### **6.5.2.1 FA implementation: QCA-Shannon-Lattice**

Figures 6-18 and 6-19 show the QCA-Shannon-Lattice realizations of the full-adder's sum and carry-out functions respectively. Table 6-8 shows the combined layout characteristics for sum and carry-out functions.

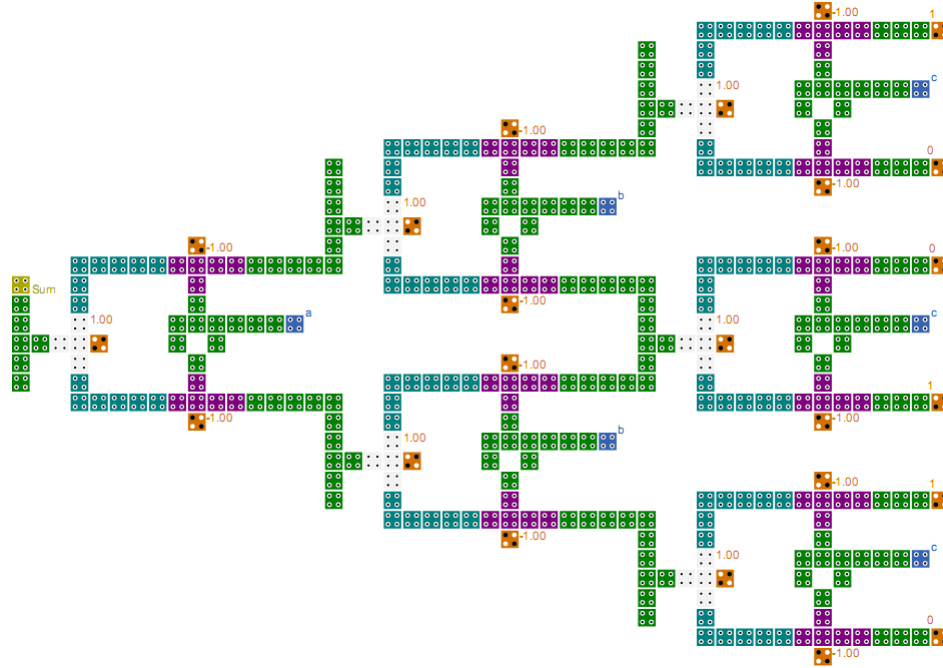


Figure 6-19 FA sum function realized using QCA-Shannon-Lattice

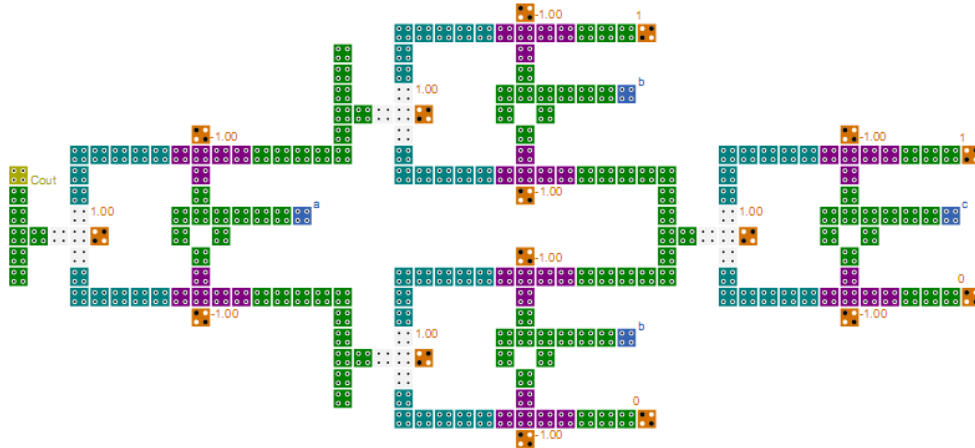


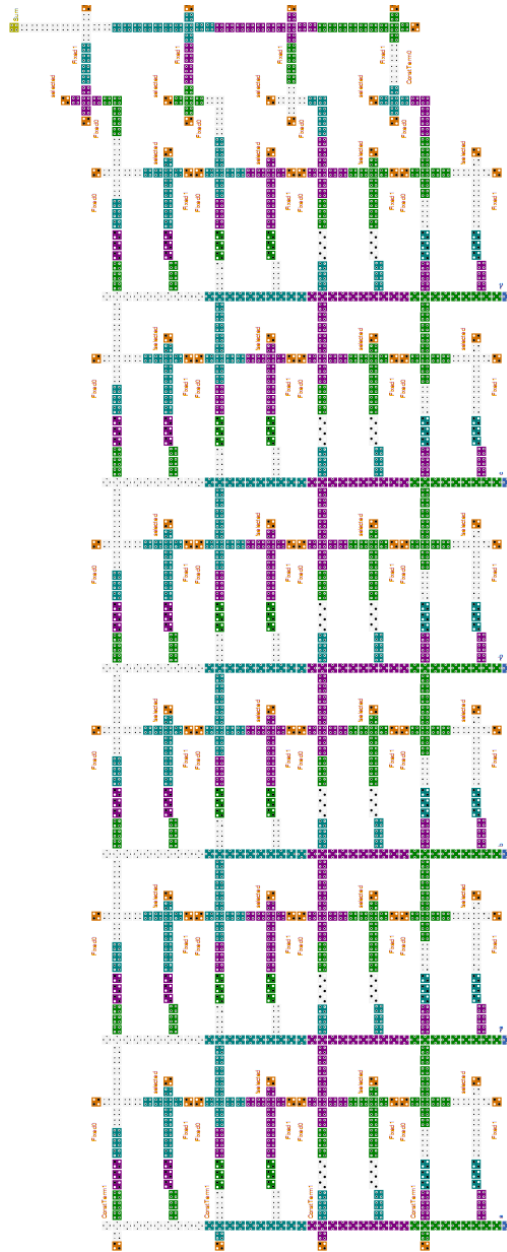
Figure 6-20 FA carry-out function realized using QCA-Shannon-Lattice

<b>FA QCA-Shannon-Lattice Layout</b>	
Number of QCA-cells	560
Worst-case-latency (clocks)	3.25
Throughput (n; one output/n clock cycles)	3
Area: sum layout + carry-out layout ( $\mu m^2$ )	1.10
Cell-density (cells/ $\mu m^2$ )	510

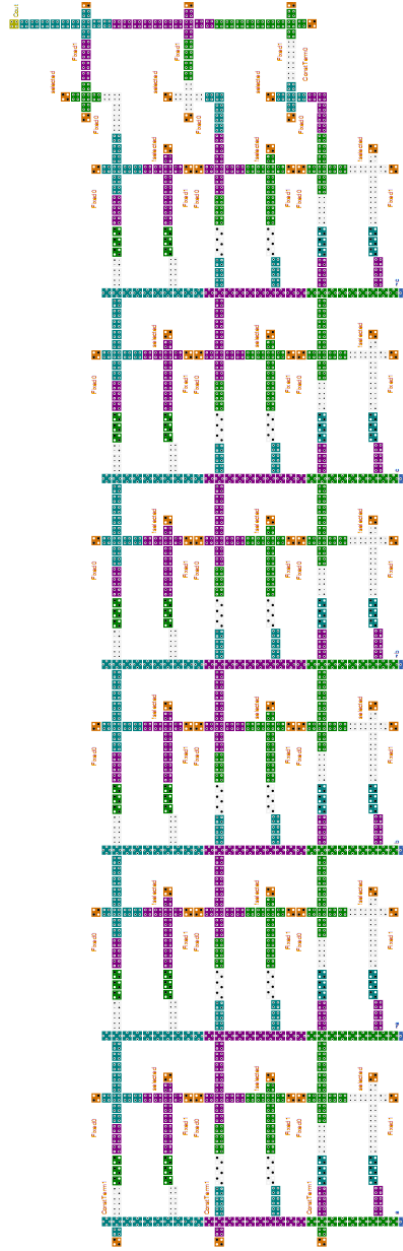
**Table 6-8 FA QCA-Shannon-Lattice layout with combined sum and carry-out characteristics**

### **6.5.2.2 FA implementation: QCA-PLA**

Figures 6-20 and 6-21 show the QCA-PLA realizations of the full-adder's sum and carry-out functions respectively. Table 6-9 shows the combined layout characteristics for sum and carry-out functions.



**Figure 6-21 FA sum function realized using QCA-PLA**



**Figure 6-22 FA carry-out function realized using QCA-PLA**

<b>Full-Adder QCA-PLA Layout</b>	
Number of QCA-cells	2261
Worst-case-latency (clocks)	8
Throughput (n; one output/n clock cycles)	6
Area; sum layout + carry-out layout ( $\mu\text{m}^2$ )	4.28
Cell-density (cells/ $\mu\text{m}^2$ )	529

**Table 6-9 Full-Adder QCA-PLA layout with combined sum and carry-out characteristics**

Property / FA Layout-type	Custom layout	QCA-Shannon-Lattice layout	QCA-PLA layout
Number of QCA-cells	114	560	2261
Worst-case-latency (clocks)	1.25	3.25	8
Throughput (n; one output/n clock cycles)	1	3	6
Area; sum layout + carry-out layout ( $\mu m^2$ )	0.23	1.10	4.28
Cell-density (cells/ $\mu m^2$ )	496	510	529

**Table 6-10 Combined table for characteristics of different FA layout types**

From Table 6-10, it is easy to observe that custom layout is superior to both type of regular structure realizations of full-adder. The throughput of custom layout is three and six times higher than that of QCA-Shannon-Lattice and QCA-PLA respectively. In terms of area, custom layout consumes approximately five times and 20 times less area compared to QCA-Shannon-Lattice and QCA-PLA layout respectively. As a reminder to the reader, the comparative analysis in this section favors custom layout more so than the layout using regular structures, as mentioned earlier, due to separate layouts of sum and carry-out functions in QCA-Shannon-Lattice and QCA-PLA realizations of the full-adder circuit. Even in the silicon domain, it is a well known fact that the custom layout is performed to optimize the circuit for area, delay and power. However, it comes with a price of long design time along with tedious design process. As illustrated in section 6.5.1, the initial layout of full-adder required manual modifications to obtain a layout which shows a correct simulation behavior. This process took approximately 15 hours of work, this included performing the layout and verifying the correctness of simulation waveforms. It is important to mention that generating a modified FA layout of Figure 6-

17 was a repetitive process which involved several different/smaller modifications. For each layout change, simulations were performed to observe the correctness of the layout. Thus, final layout was obtained after the repetitions of such smaller changes which were time consuming. On the other hand, for full-adder realization, using regular structures was a simpler and faster process because the behavior of regular structures' macro-cell was predictable and layout was done automatically using the tool developed in this thesis. This is definitely an advantage of using regular structures layout generated automatically by the tool over creating a custom layout. The regular layout generation using the tool about two hours of work, this included verifying correctness of simulation waveforms and negligible time for generating the layouts. In conclusion, clearly there are trade-offs in creating a custom layout versus tool-generated layout using regular structures.



## Chapter 7

### CONCLUSION AND FUTURE WORK

From the results presented in chapter 6, it can be concluded that QCA-Shannon-Lattice is better than QCA-PLA in all factors considered except for cell density. A detailed analysis is presented on computing the latency and throughput of the QCA-PLA and QCA-Shannon-Lattice designs considered in this thesis. The types of regular structures chosen in this thesis were majorly due to their simpler designs and ease of software-tool development for automatic layout generation. The effect of various factors like number of variables, product terms and types of functions on latency, throughput and area of both types of design was also presented. A complete design implementation and clocking zone assignment is also shown for realizing Boolean functions using both types of regular layouts. Finally, the overhead and trade-offs in logic realization using both types regular structures compared to custom layout were presented for 1-bit full-adder circuit. It was shown that although the custom layout of full-adder was better in terms of area, throughput and area, the design effort required for it was much higher than the implementations using regular structures. The research work in this thesis adapts the idea of generating layout using standard-characterized-macro-cells like conventional CMOS designs and presented layout of larger circuits based on simple replication and clock offset of input macro-cells. The results of consistent behavior between the macro-cells and larger design built using them were also proven by simulation performed using QCADesigner. In addition, for this research a flexible software tool was also developed which generates the layout of the input Boolean function using the macro-cells and their

properties. This tool can play an important role in similar studies where different types of macro-cells can be used to generate layouts for larger circuits with different clocking zone assignments. These larger layouts can then be simulated to verify the behavior and robustness of macro-cells when used as building blocks in larger designs.

Although this work adds to the knowledge base of scientific community, the study has some limitations. One major limitation of this comparison is that non-symmetric function were not included due to which QCA-Shannon-Lattice came out to be a clear winner in each field. This is because the current lattice diagram generation algorithms implemented in the software can only efficiently handle symmetric functions. For future work, the algorithms in the tool for logic synthesis of Boolean function to Shannon-Lattice should be replaced with some advanced algorithms as presented in [7, 8, 30]. The tool currently has limitations on the size of Boolean function that can be handled by it to generate layout. It can be improved by refining the code to use the memory efficiently. In future, the tool can be extended to work with the function net-list and generate layouts for sequential circuits using PLA and memory.

## REFERENCES

- [1] "International Technology Roadmap for Semiconductors." [Online]. Available: [http://www.itrs.net/links/2010itrs/2010Update/ToPost/2010\\_Update\\_Overview.pdf](http://www.itrs.net/links/2010itrs/2010Update/ToPost/2010_Update_Overview.pdf). [Accessed: 08-May-2011].
- [2] M. Niemier, P. Kogge, R. Murphy, A. Rodrigues, T. Dysart, and S. Frost, *Dataflow in molecular QCA: Logic can "sprint" but the memory wall can still be a "hurdle."*, Technical Report, TR-2006-16, University of Notre Dame, 2006.
- [3] P. Tougaw and C. Lent, "Logical devices implemented using quantum cellular automata," *Journal of Applied Physics*, vol. 75, no. 3, pp. 1818-1825, Feb. 1994.
- [4] M. Perkowski and E. Pierzchala, *New Canonical Forms for Four-Valued Logic*. Portland, Oregon: Portland State University, 1993.
- [5] M. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Lattice Diagrams Using Reed-Muller Logic," *Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 85-102, 1997.
- [6] M. Perkowski and A. Mishchenko, "Logic Synthesis for Regular Layout using Satisfiability." 02-Apr-2008.
- [7] M. Chrzanowska-Jeske, "Logic Synthesis for a Regular Layout," *VLSI DESIGN - LANGHORNE*, vol. 10, pp. 35-56, 1999.
- [8] M. Chrzanowska-Jeske and Y. Xu, "Variable ordering for regular layout representation," *Electronics, Circuits and Systems, 1998 IEEE International Conference on*, vol. 1, pp. 93-96 vol.1, 1998.
- [9] Timothy Dysart, "Defect Properties and Design Tools for Quantum Dot Cellular Automata," M.S., Notre Dame, 2005.
- [10] T. Dysart, "It's All About the Signal Routing: Understanding the Reliability of QCA Circuits and Systems," Ph.D., University of Notre Dame, 2009.

- [11] K. Walus and G. Jullien, "Design Tools for an Emerging SoC Technology: Quantum-Dot Cellular Automata," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1225-1244, Jun. 2006.
- [12] C. Lent, "Aquinas: A quantum interconnected network array simulator," *Proceedings of Fifth International Workshop on Computational Electronics*, May. 1997.
- [13] M. Niemier, M. Kontz, and P. Kogge, "A design of and design tools for a novel quantum dot based microprocessor," in *Proceedings of the 37th Annual Design Automation Conference*, New York, NY, USA, p. 227-232, 2000.
- [14] K. Walus, T. Dysart, G. Jullien, and R. Budiman, "QCADesigner: A Rapid Design and Simulation Tool for Quantum-Dot Cellular Automata," *IEEE Transactions On Nanotechnology*, vol. 3, no. 1, pp. 26-31, Mar. 2004.
- [15] T. Teodosio and L. Sousa, "QCA-LG: A tool for the automatic layout generation of QCA combinational circuits," *Proceedings of Norchip*, pp. 1-5, Nov. 2007.
- [16] T. Dysart and P. Kogge, "Comparing the Reliability of PLA and Custom Logic Implementations of a QCA Adder," in *2008 IEEE International Workshop on Design and Test of Nano Devices, Circuits and Systems*, Cambridge, MA, pp. 53-56, 2008
- [17] K. Hennessy and C. Lent, "Clocking of molecular quantum-dot cellular automata," *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures*, vol. 19, no. 5, p. 1752, 2001.
- [18] M. Niemier and P. Kogge, "Problems in designing with QCAs: Layout = Timing," *International Journal of Circuit Theory and Applications*, vol. 29, no. 1, pp. 49-62, Jan. 2001.
- [19] M. Niemier, A. Rodrigues, and P. Kogge, "A Potentially Implementable FPGA for Quantum Dot Cellular Automata," *1st Workshop on Non-Silicon Computation, NSC-1*, vol. 69, pp. 38-45, 2002.
- [20] C. Lent and P. Tougaw, "A device architecture for computing with quantum dots," *Proceedings of the IEEE*, vol. 85, no. 4, pp. 541-557, Apr. 1997.
- [21] T. Lantz, "A QCA Implementation of a Look-up Table for an FPGA," M.S. Thesis, Rochester Institute Of Technology, 2006.

- [22] G. Toth and C. Lent, "Quasiadiabatic switching for metal-island quantum-dot cellular automata," *Journal of Applied Physics*, vol. 85, no. 5, pp. 2977-2984, Mar. 1999.
- [23] V. Vankamamidi, M. Ottavi, and F. Lombardi, "Two-Dimensional Schemes for Clocking/Timing of QCA Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 34-44, Jan. 2008.
- [24] E. Blair and C. Lent, "An architecture for molecular computing using quantum-dot cellular automata," in *the Third IEEE Conference on Nanotechnology, IEEE-NANO 2003.*, San Francisco, CA, USA, pp. 402-405, Sept. 2003.
- [25] F. Mo and R. Brayton, "PLA-based regular structures and their synthesis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 723- 729, Jun. 2003.
- [26] K. Tong, V. Rovner, L. Pileggi, and V. Kheterpal, "Design Methodology of Regular Logic Bricks for Robust Integrated Circuits," in the *International Conference on Computer Design*, San Jose, CA, USA, pp. 162-167, 2006.
- [27] C. Menezes, C. Meinhardt, R. Reis, and R. Tavares, "Design of Regular Layouts to Improve Predictability," in *2006 International Caribbean Conference on Devices, Circuits and Systems*, Playa del Carmen, Mexico, pp. 67-72, 2006.
- [28] S. Akers, "A Rectangular Logic Array," *Computers, IEEE Transactions on*, vol. 21, no. 8, pp. 848-857, Aug. 1972.
- [29] M. Perkowski, "Dr. Marek Perkowski's webpage." [Online]. Available: [http://web.cecs.pdx.edu/~mperkows/CLASS\\_572\\_99/572-2005.html](http://web.cecs.pdx.edu/~mperkows/CLASS_572_99/572-2005.html). [Accessed: 13-Apr-2011].
- [30] M. Chrzanowska-Jeske, "Regular symmetric arrays for non-symmetric functions," in *ISCAS'99. Proceedings of the 1999 IEEE International Symposium on Circuits and Systems VLSI (Cat. No.99CH36349)*, Orlando, FL, USA, pp. 391-394, 1999.
- [31] M. Niemier and P. Kogge, "Exploring and exploiting wire-level pipelining in emerging technologies," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, Goteborg, Sweden, pp. 166-177, May 2001.

- [32] R. Zhang, K. Walus, W. Wang, and G. Jullien, "A Method of Majority Logic Reduction for Quantum Cellular Automata," *IEEE Transactions On Nanotechnology*, vol. 3, no. 4, pp. 443-450, Dec. 2004.
- [33] R. Ho, K. Mai, and M. Horowitz, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490-504, Apr. 2001.
- [34] K. Walus, G. Jullien, and V. Dimitrov, "Computer arithmetic structures for quantum cellular automata," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers*, Pacific Grove, CA, USA, pp. 1435-1439, 2003.
- [35] M. Niamat, S. Panuganti, and T. Raviraj, "QCA design and implementation of SRAM based FPGA Configurable Logic Block," in *the 53rd IEEE International Midwest Symposium on Circuits and Systems*, Seattle, WA, USA, pp. 837-840, 2010.
- [36] M. Crocker, X. Hu, M. Niemier, M. Yan, and G. Bernstein, "PLAs in Quantum-Dot Cellular Automata," *IEEE Transactions on Nanotechnology*, vol. 7, no. 3, pp. 376-386, May. 2008.
- [37] K. Kyosun, W. Kaijie, and R. Karri, "Towards Designing Robust QCA Architectures in the Presence of Sneak Noise Paths," in *Design, Automation and Test in Europe*, Munich, Germany, pp. 1214-1219, March 2005.

## APPENDIX: TOOL GENERATED LAYOUTS

This section presents the layouts for some of the benchmark functions used in section 6.4, generated automatically using the tool. **It should be noted that the QCA-Shannon-Lattice layouts are not to the scale of QCA-PLA layouts.** At the same scale, QCA-Shannon-Lattice layouts occupy much smaller area than QCA-PLA layouts.

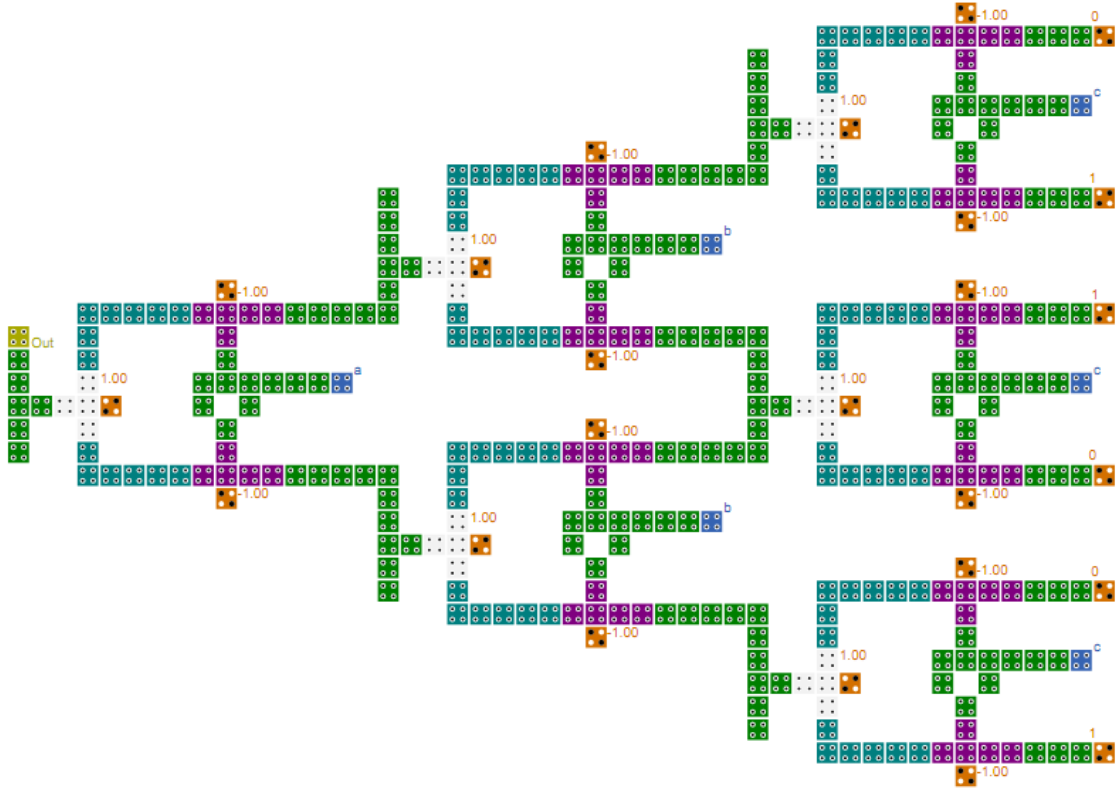


Figure A-1 QCA-Shannon-Lattice layout of exam1\_d

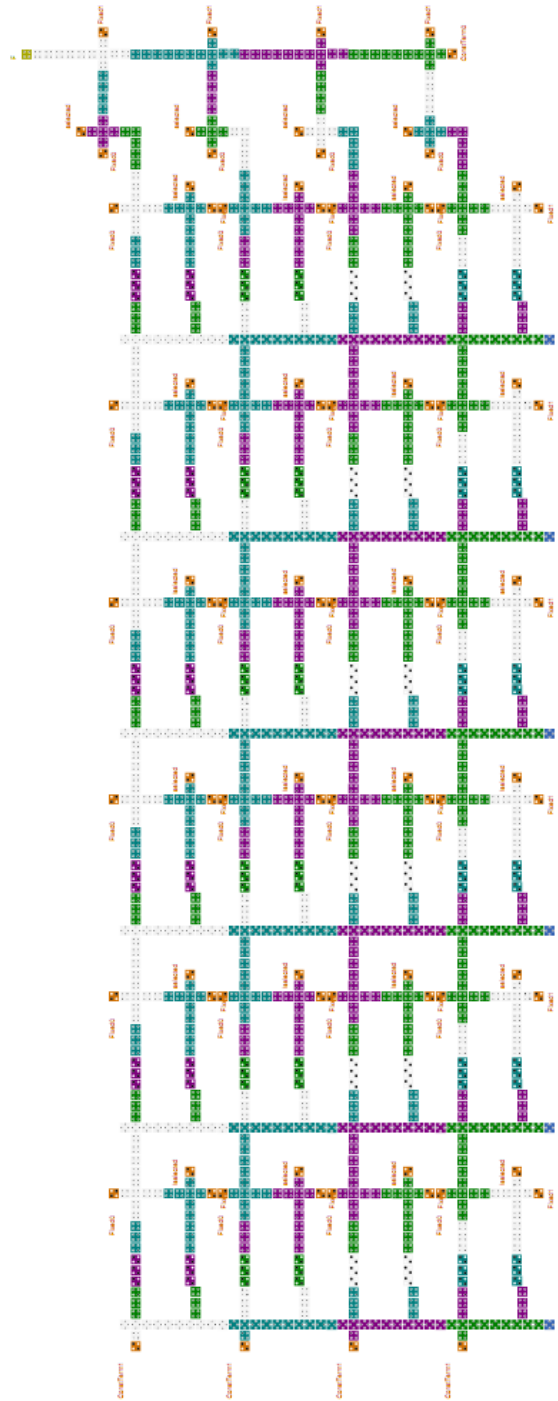


Figure A-2 QCA-PLA layout of exam1\_d



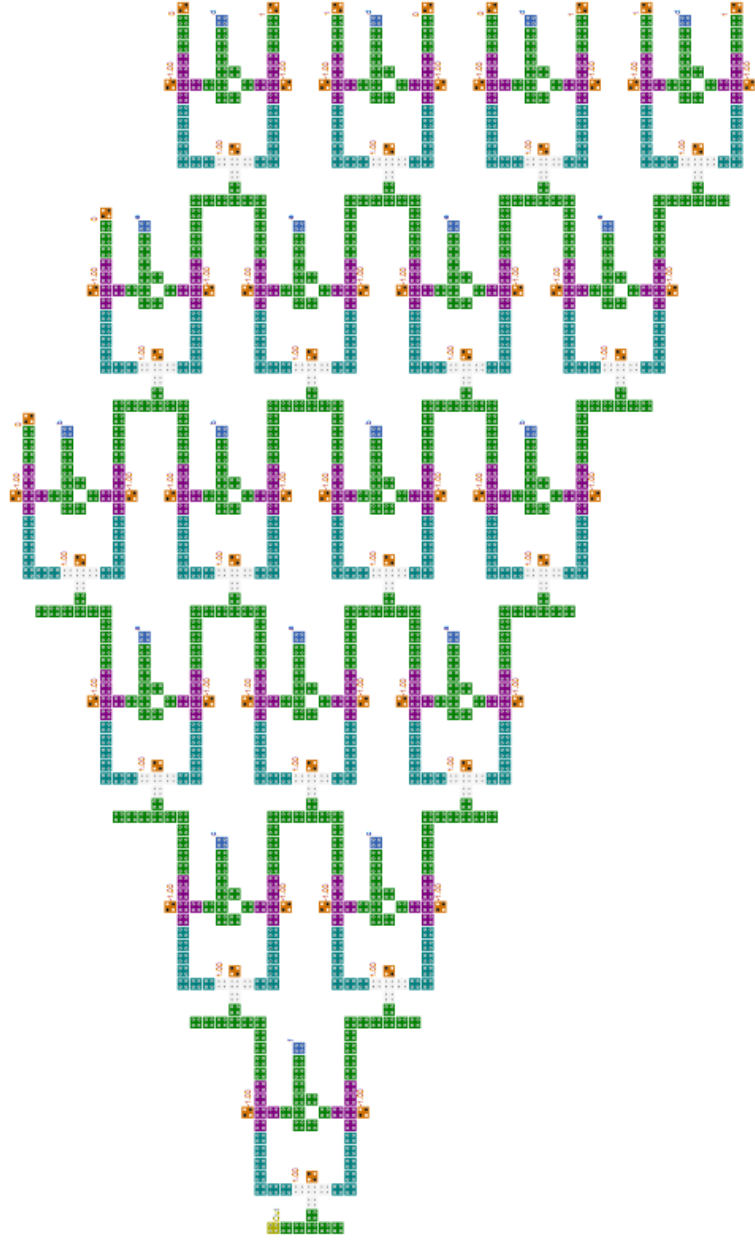


Figure A-3 QCA-Shannon-Lattice layout of  $S^{0.13}$ (a, b, c, d, e, f)

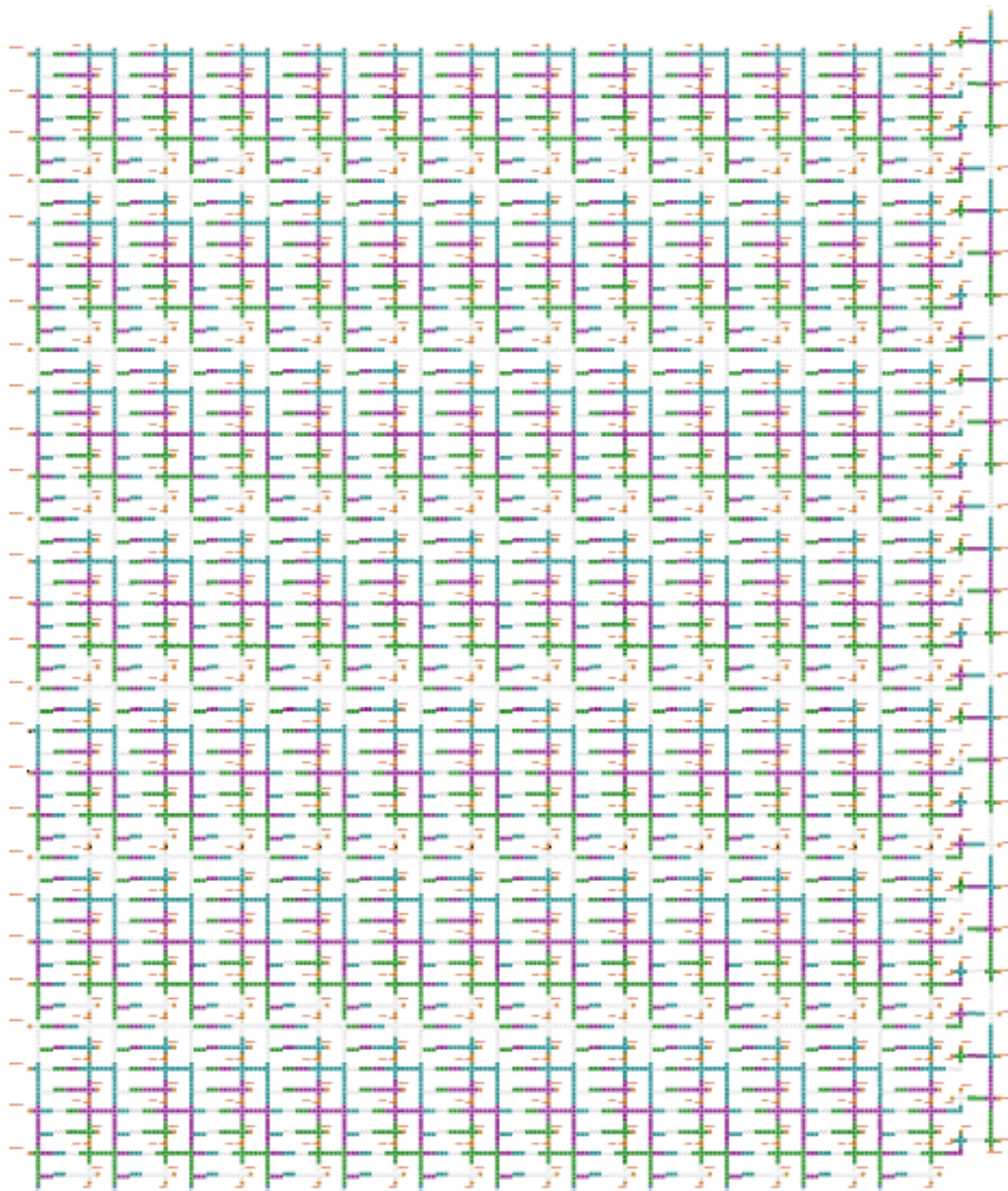
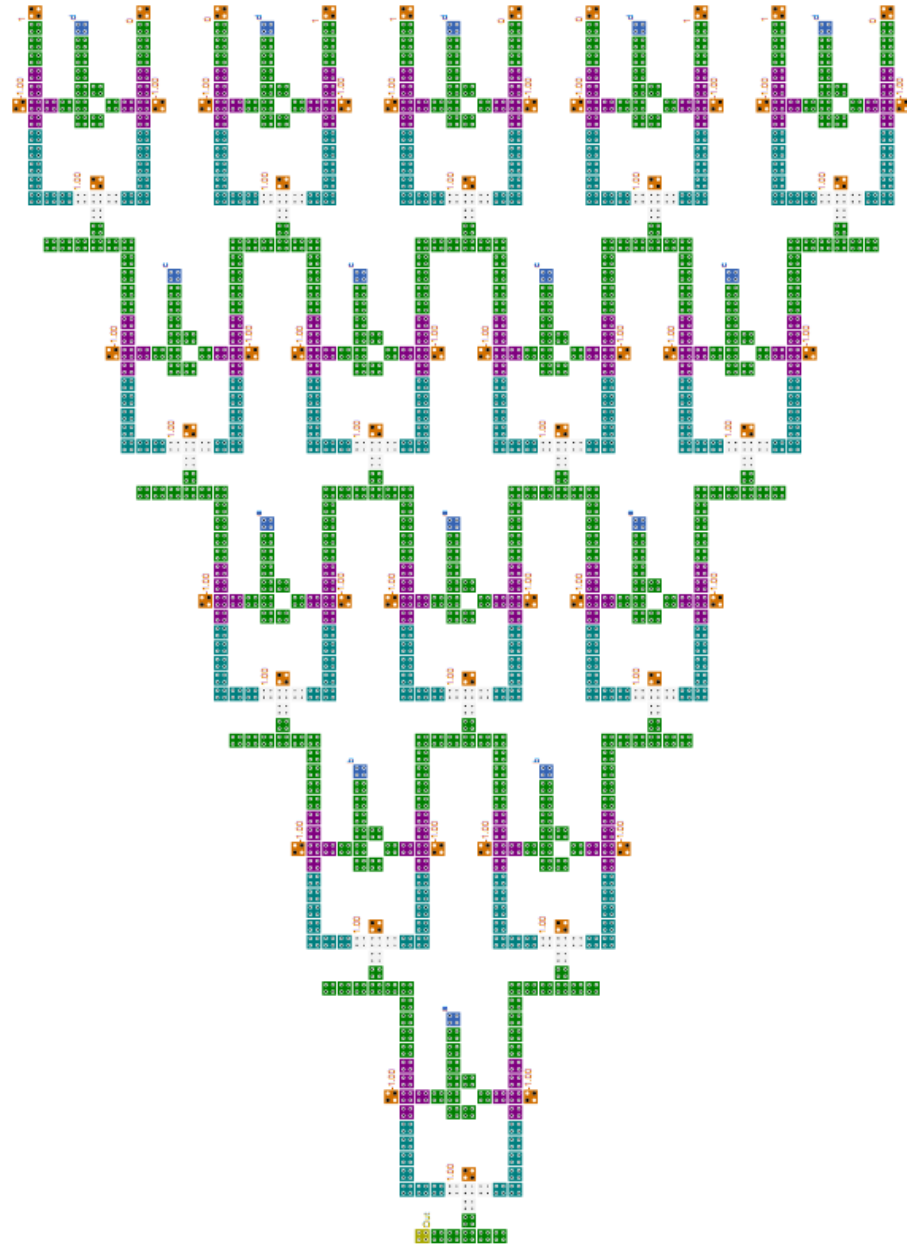


Figure A-4 QCA-PLA layout of  $S^{0,1,3}$  (a, b, c, d, e, f)



**Figure A-5 QCA-Shannon-Lattice layout of xor5\_d**

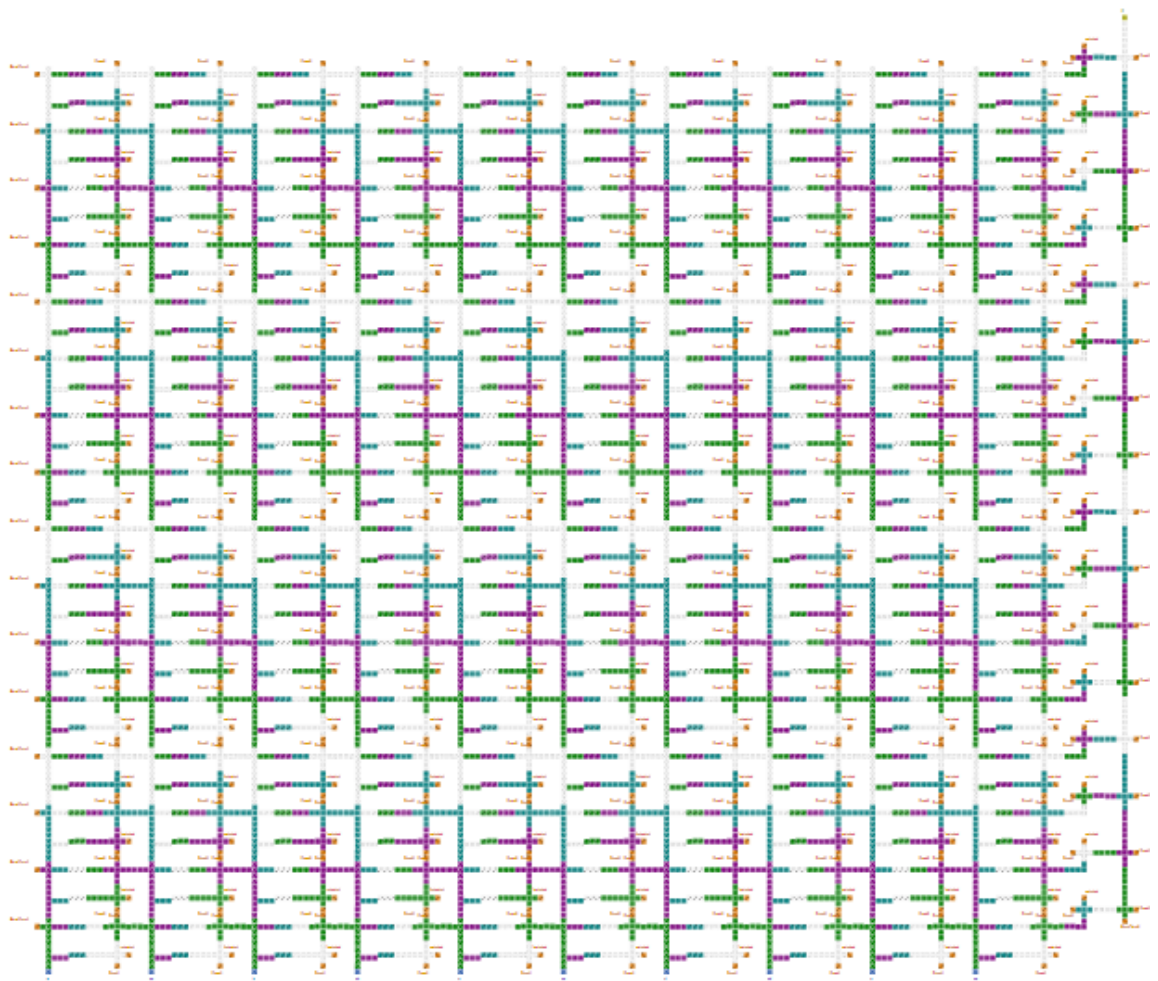


Figure A-6 QCA-PLA layout of xor5\_d

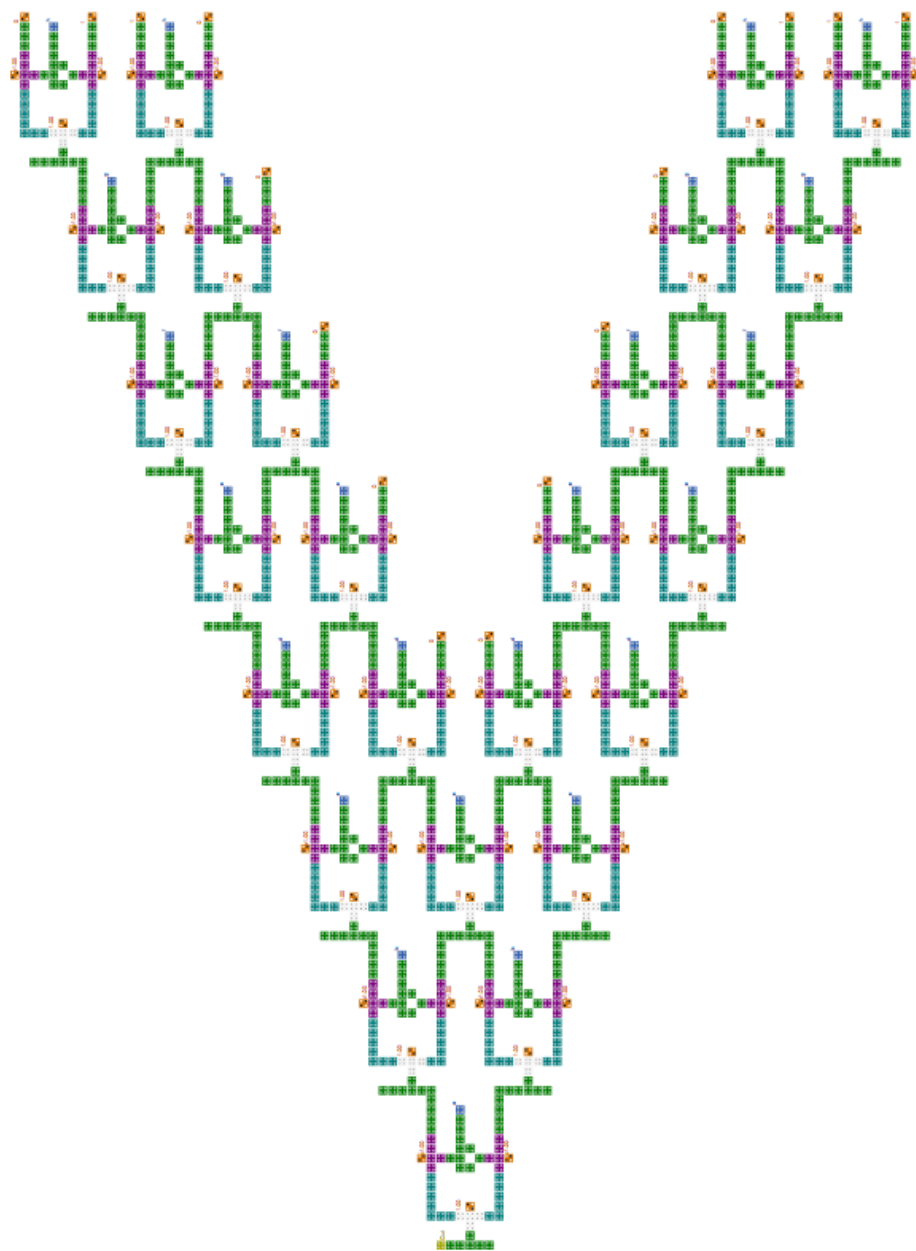


Figure A-7 QCA-Shannon-Lattice layout of  $S^{0,1,7,8}$ (a, b, c, d, e, f, g, h)



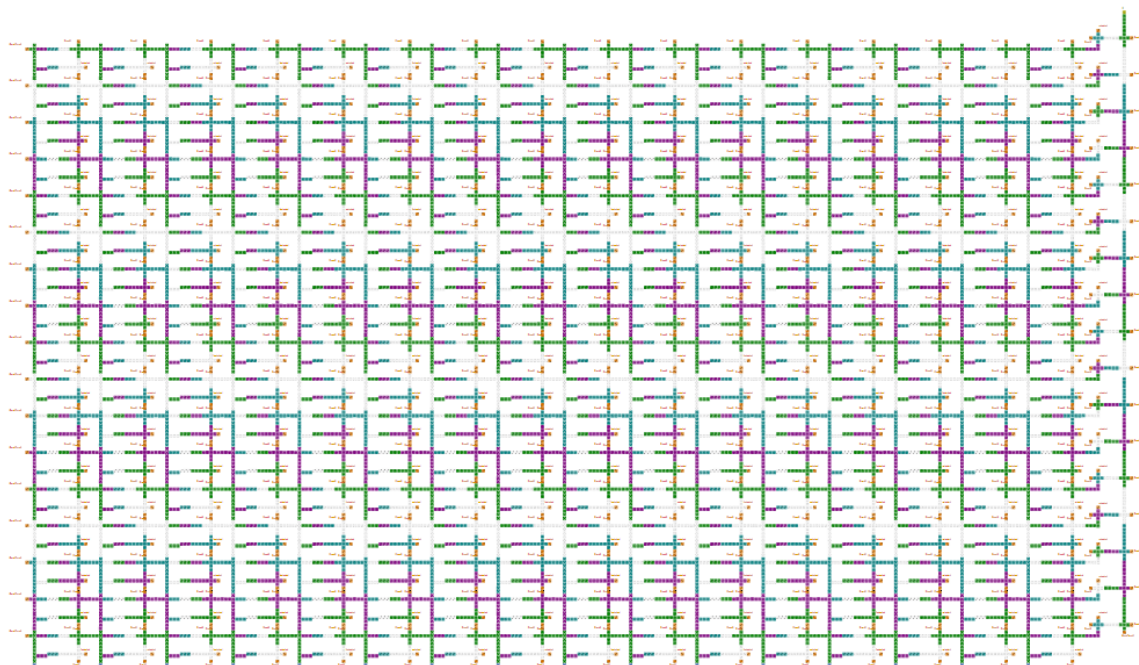


Figure A-8 QCA-PLA layout of  $S^{0,1,7,8}(a, b, c, d, e, f, g, h)$