

1-1-2013

Emerging Design Methodology And Its Implementation Through Rns And Qca

Omar Dajani
Wayne State University,

Follow this and additional works at: http://digitalcommons.wayne.edu/oa_dissertations

Recommended Citation

Dajani, Omar, "Emerging Design Methodology And Its Implementation Through Rns And Qca" (2013). *Wayne State University Dissertations*. Paper 646.

This Open Access Dissertation is brought to you for free and open access by DigitalCommons@WayneState. It has been accepted for inclusion in Wayne State University Dissertations by an authorized administrator of DigitalCommons@WayneState.

**EMERGING DESIGN METHODOLOGY AND ITS IMPLEMENTATION
THROUGH RNS AND QCA**

by

OMAR DAJANI

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2013

MAJOR: ELECTRICAL ENGINEERING

Approved by:

Advisor

Date

© COPYRIGHT BY

OMAR DAJANI

2013

All Rights Reserved

DEDICATION

To my family

ACKNOWLEDGMENTS

I sincerely thank my research advisor Prof. Harpreet Singh for guiding me through this research project and for giving me the encouragement and advice throughout the course of this work. Dr. Singh's dedication and flexibility is what made it possible for me to succeed. I would like to thank Dr. Pepe Siy for his help during the initial phase of my PhD research.

Also, I would like to thank my dissertation committee: Dr. Mohamad Berri, Dr. Feng Lin and Dr. Le Yi Wang for their helpful comments and encouragement.

Many thanks to my colleagues for their assistance in FPGA, Multisim and Cadence that have helped me in my research development.

Finally, I would like to thank my family for their support and encouragement.

TABLE OF CONTENTS

Dedication.....	ii
Acknowledgments.....	iii
List of Tables.....	vii
List of Figures	viii
Chapter 1 Introduction.....	1
1.1 Introduction to RNS.....	1
1.2 Residue Number System Representation.....	2
1.3 Residue Dynamic Range.....	2
1.4 Quantum-Dot Cellular Automata (QCA).....	5
1.5 Problem Statement.....	6
1.6 Thesis Organization.....	8
Chapter 2 Literature Review	9
2.1 Introduction.....	9
2.2 RNS Arithmetic Operations.....	9
2.3 Conversion from RNS to Binary.....	11
2.3.1 Chinese Remainder Theorem Conversion.....	11
2.3.2 Mixed Radix Conversion.....	13
2.4 Residue Number System Sign Detection.....	15
2.5 Scaling in Residue Number System.....	16
2.6 RNS Fast Processing Applications.....	17
2.7 Quantum-Dot Cellular Automata (QCA).....	17
2.8 QCA Clocking.....	20
2.9 Conclusion.....	22

Chapter 3 Novel Parallel - Prefix Structure Binary to Residue Number System Conversion

Method.....	23
3.1 Introduction.....	23
3.2 Residue Number System.....	24
3.3 New Novel Conversion Method from Binary to Residue Representation.....	24
3.4 Illustrative Example.....	34
3.5 Implementation Selection.....	38
3.6 Comparison Selection to Pervious Work.....	41
3.7 Conclusion.....	42
Chapter 4 Simplified RNS Scaling Algorithm.....	43
4.1 Introduction.....	43
4.2 Division remainder zero.....	43
4.3 Scaling.....	43
4.4 General Division.....	44
4.5 New Base Extension Algorithm.....	45
4.6 New Scaling Algorithm.....	47
4.7 Conclusion.....	52
Chapter 5 VLSI Implementation of Residue Adder and Subtract.....	53
5.1 Introduction.....	53
5.2 Residue Adder and Subtractor.....	54
5.3 V LSI Implementation.....	56
5.4 Conclusion.....	59
Chapter 6 Novel Quantum Boolean Circuits Construction by Using XOR-AND Reduction	
Method.....	60
6.1 Introduction.....	60

6.2 QCA Background Material.....	61
6.3 Novel QCA Extraction.....	66
6.4 Conclusion.....	71
Chapter 7 Implementation of Generalized Pipeline Cellular Array Using Quantum-Dot	
Cellular Automata.....	72
7.1 Introduction.....	72
7.2 QCA Pipeline Array.....	77
7.3 QCA Pipeline Implementation.....	85
7.4 Conclusion.....	99
Chapter 8 Conclusion and Future Work.....	
8.1 Introduction.....	100
8.2 Summary of Work.....	100
8.3 Recommended Future Work.....	102
Appendix.....	106
Preferences.....	143
Abstract.....	153
Autobiographical Statement.....	155

LIST OF TABLES

Table (1.1): Residue Digits for Unsigned Numbers.....	4
Table (1.2): Residue Digits for Signed Numbers	5
Table (1.3): Majority Function $M(A,B,C)$	6
Table (2.1): Equivalent QCA Expressions for Major Boolean Gates	20
Table (3.1): Shows Comparison Among the Three Designs Implementation	40
Table (3.2): Comparison Among Behrooa, Alia and the new design.....	42
Table (5.1): Total RNS Adder and Subtractor Delay Time (PS).....	58
Table (5.2): RNS Adder and Subtractor Design Parameter	58
Table (6.1): The Chart for Deriving XOR Equivalent Function “f1”.....	66
Table (6.2): The Chart for Deriving XOR Equivalent Function “f2”	70
Table (7.1): Boolean Functions and Their Equivalent QCA.....	75
Table (7.2): QCA Pipeline Array Arithmetic Summary Operations.....	81
Table (7.3): Subtrahend at Different Levels for Square Rooting.....	82
Table (7.4): QCA Performance Comparison Between the Two Designs.....	91
Table (7.5): COMS Performance Comparison Between the Two Designs.....	96
Table (7.7): Design Delay Time (PS).....	96
Table (7.8): Complexity Fuzzy Results for Pipeline Array Cells.....	98

LIST OF FIGURES

Figure (1.1): Block Diagram that Shows the Research Logical Flow.....	7
Figure (2.1a): QCA Cells and Binary Encoding	18
Figure (2.1b): QCA Majority Gate	18
Figure (2.1c): QCA Inverter Gate.....	19
Figure (2.1d): QCA Wire Types.....	19
Figure (2.2a): QCA Clock Phases.....	20
Figure (2.2b): Four QCA Interdot Barrier State.....	21
Figure (3.1): Two Bits (b1 & b0) Binary to RNS Conversion	25
Figure (3.2): Prefix Logic Operation and Their Implementation	26
Figure (3.3): Three Bits (b2, b1, b0) Binary to RNS Conversion.....	27
Figure (3.4): Four Bits Binary to RNS System.....	28
Figure (3.5): Six Bits Binary to Residue Number System Conversion.....	32
Figure (3.6): Prefix Structure of 8 Bits Binary to RNS	34
Figure (3.7a): Eight Bits Binary to Residue Number System Conversion.....	36
Figure (3.7b): Example for signal propagation.....	37
Figure (3.8): Prefix Structure of 10 Bits binary to RNS.....	39
Figure (4.1): Shows the Fermat's Implementation for Finding Coefficient "c".....	48
Figure (4.2): Shows Extended Euclid Algorithm Implementation.....	48
Figure (4.3): Shows Implementation of Finding Coefficient "aij".....	48
Figure (5.1): RNS Adder and Subtractor	55
Figure (5.2): Waveform of RNS Adder and Subtractor.....	56
Figure (5.3): RNS Adder and Subtractor Layout	57
Figure (5.4): Encounter Part of RNS Adder and Subtractor	57
Figure (5.5): Virtuoso Part of RNS Adder and Subtractor	59

Figure (6.1a): QCA Cells and Binary Encoding	62
Figure (6.1b): QCA Majority Gate	62
Figure (6.1c): QCA Inverter Gate.....	63
Figure (6.1d): QCA Wire Types	63
Figure (6.2a): QCA Clock Phases.....	64
Figure (6.2b): Four QCA Interdot Barrier State.....	64
Figure (6.3): XOR-AND Function Extraction Methodology.....	65
Figure (6.4): Majority Tree for Function “f1”.....	67
Figure (6.5): Majority Gates Schematic for Function “f1”.....	68
Figure (7.1): QCA Cells and Binary Encoding	72
Figure (7.2): QCA Wire Types	73
Figure (7.3): QCA Majority Gate	73
Figure (7.4): QCA Inverter Gate.....	74
Figure (7.5): QCA Clock Phases.....	74
Figure (7.6): Block Diagram for Pipeline Array.....	78
Figure (7.7a): Arithmetic Cell	79
Figure (7.7b): Control Logic Cell.....	79
Figure (7.8): QCA Arithmetic Cell and Control Cell Arithmetic Cell.....	84
Figure (7.9): QCA High Speed Arithmetic Cell.....	85
Figure (7.10): QCADesigner Layout for Arithmetic Cell Unit	86
Figure (7.11): QCADesigner Layout for Control Cell Unit	87
Figure (7.12): Simulation for Control Cell Unit.....	87
Figure (7.12): Simulation for Arithmetic Cell Unit	88
Figure (7.13): QCADesigner Layout for Arithmetic Cell Unit.....	89
Figure (7.14): Simulation for High Speed Arithmetic Cell Unit.....	90

Figure (7.15): Waveform of Pipeline Squaring Output Result	91
Figure (7.16): Waveform of Pipeline Square Rooting Output Result.....	92
Figure (7.17): Multisim Implementation of Arithmetic Cell.....	92
Figure (7.18): Multisim Implementation of Control Cell	93
Figure (7.19): Multisim Implementation of High Speed Arithmetic Cell.....	93
Figure (7.20): Arithmetic cell FPGA packaging.....	94
Figure (7.21): Control cell FPGA packaging.....	94
Figure (7.22): High speed arithmetic cell FPGA packaging.....	94
Figure (7.23): Arithmetic cell FPGA schematic layout.....	94
Figure (7.24): High Speed arithmetic cell FPGA schematic layout	95
Figure (7.25): Control cell FPGA schematic layout.....	95
Figure (7.26): Encounter Part of QCA Pipeline Array.....	97
Figure (7.27): Virtuoso Part of Padding.....	98
Figure (7.28): Virtuoso Part of Pipeline Array.....	99
Figure (7.29): Hardware Complexity Fuzzy Concept.....	99

CHAPTER 1

INTRODUCTION

1.1 Introduction to RNS

In the last decade, Residue Number System (RNS) has received increased attention due to its ability to support high speed concurrent arithmetic applications such as Fast Fourier Transform (FFT), image processing and digital filters utilizing the efficiencies of RNS arithmetic in addition and multiplication. In spite of its effectiveness, RNS has remained more an academic challenge and very little impact in practical applications due to the complexity involved in the conversion process, magnitude comparison, overflow detection, sign detection, parity detection, scaling and division.

The advancements in Very Large Scale Integration (VLSI) technology and the demand for parallelism computation have enabled researchers to consider RNS as an alternative approach to high speed concurrent arithmetic [10], [11].

RNS is an unweighted representation system of numbers. The difference between RNS and fixed radix systems is that no fixed base is used in the representation of RNS numbers. RNS is based on modular arithmetic operations and it is a carry-free system that performs addition, subtraction and multiplication as parallel operations.

1.2 Residue Number System Representation

For any given set of relatively prime modulo set $(m_1, m_2, m_3, \dots, m_n)$, the residue representation of an binary number X is $(x_1, x_2, x_3, \dots, x_n)$; Where X can be defined by N equations.

$$X = m_i q_i + x_i \quad (1.1)$$

Where x_i is the least positive remainder of division X by m_i and $0 \leq x_i < m_i$; q_i is the

smallest positive integer of $\left\lfloor \frac{X}{m_i} \right\rfloor$.

1.3 Residue Dynamic Range

The residue representation of number is unique for any integer $X \in [0, M - 1]$, where M is called dynamic range.

$$M = \prod_{i=1}^n m_i \quad (1.2)$$

For signed numbers, one has to distinguish two cases

Case 1:

The product M is an even number. This occurs if one modulo is an even number and the range is defined as

$$X \in \left[-\frac{M}{2}, \frac{M}{2} - 1 \right] \quad (1.3)$$

and all number

$$X \geq \frac{M}{2} \quad (1.4)$$

are negative numbers

Case 2:

The product M is an odd number. This occurs if all modulo are odd numbers and the range is defined as

$$X \in \left[-\frac{M-1}{2}, \frac{M-1}{2} \right] \quad (1.5)$$

and all numbers according to

$$X \geq \frac{M-1}{2} + 1 \quad (1.6)$$

are negative numbers.

The RNS representation of negative number $-X$ is

$$(-X)_{10} \xrightarrow{RNS} (|m_1 - x_1|_{m_1}, |m_2 - x_2|_{m_2}, \dots, |m_h - x_n|_{m_h})_{m_1, m_2, \dots, m_h} \quad (1.7)$$

To illustrate the residue representation, consider the three modulo set (2, 3, 5) example. The list of the positive numbers from 0 to $M-1$ and their RNS representation is shown in table 1.1. The list of positive and negative numbers from (-15, 14) and their RNS representation is shown in table 1.2.

TABLE 1.1

RESIDUE DIGITS FOR UNSIGNED NUMBERS FOR $M < 30$

Integer	Residue digits			Integer	Residue digits		
	Modulo				Modulo		
	2	3	5		2	3	5
	x_1	x_2	x_3		x_1	x_2	x_3
0	0	0	0	15	1	0	0
1	1	1	1	16	0	1	1
2	0	2	2	17	1	2	2
3	1	0	3	18	0	0	3
4	0	1	4	19	1	1	4
5	1	2	0	20	0	2	0
6	0	0	1	21	1	0	1
7	1	1	2	22	0	1	2
8	0	2	3	23	1	2	3
9	1	0	4	24	0	0	4
10	0	1	0	25	1	1	0
11	1	2	1	26	0	2	1
12	0	0	2	27	1	0	2
13	1	1	3	28	0	1	3
14	0	2	4	29	1	2	4

TABLE 1.2

RESIDUE DIGITS FOR SIGNED NUMBERS (-15, 14)

Integer	Residue digits			Integer	Residue digits		
	Modulo				Modulo		
	2	3	5		2	3	5
	x ₁	x ₂	x ₃		x ₁	x ₂	x ₃
0	0	0	0	-1	1	2	4
1	1	1	1	-2	0	1	3
2	0	2	2	-3	1	0	2
3	1	0	3	-4	0	2	1
4	0	1	4	-5	1	1	0
5	1	2	0	-6	0	0	4
6	0	0	1	-7	1	2	3
7	1	1	2	-8	0	1	2
8	0	2	3	-9	1	0	1
9	1	0	4	-10	0	2	0
10	0	1	0	-11	1	1	4
11	1	2	1	-12	0	0	3
12	0	0	2	-13	1	2	2
13	1	1	3	-14	0	1	1
14	0	2	4	-15	1	0	0

1.4 Quantum-Dot Cellular automata (QCA)

During past decade, Quantum-Dot Cellular Automata (QCA) has demonstrated the ability to implement both combinational and sequential logic devices [76]-[82]. Unlike conventional Boolean AND-OR-NOT based circuits, the fundamental logical device in QCA Boolean networks is majority gate which implements the Boolean function

$$M(A,B,C) = AB+AC+BC \quad (1.8)$$

TABLE 1.3

MAJORITY FUNCTION $M(A,B,C)$

A	B	C	$M(A,B,C)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

With combining these QCA gates with NOT gates any combinational or sequential logical device can be constructed from QCA cells [76]-[82]. The process of QCA Boolean logic is more sophisticated than Boolean logic. The traditional Boolean logic reduction methods such as Karnaugh maps produce simplified Boolean expressions, However, converting these forms to QCA Boolean is not simple process due to complexity of multilevel majority gates. In chapter two, we will present literature review and background material for RNS and QCA.

1.5 Problem Statement

Residue number system is a robust parallel system and it received attention due to its ability to support high speed concurrent arithmetic applications such as addition, subtraction and multiplication in modular levels. This system suffers from some weakness such as conversion process, scaling, division, overflow detection and

magnitude comparison. In this dissertation, we have proposed new techniques to solve the conversion and scaling issues. These techniques have been proved mathematically and verified through VLSI simulation.

One of the new fields of nanotechnology is Quantum-dot cellular automata. Due to its ultra-power and small size, QCA has the potential to become the future of CMOS technology. Quantum-dot cellular automata uses different logic devices to design circuits other than Boolean logic devices. Converting Boolean circuits to QCA Boolean is not simple process due to complexity of QCA and existing Boolean reduction methods do not work with QCA logic. In this thesis, we have proposed a new QCA construction reduction method that utilizes the VLSI techniques that were used in RNS system. Fig 1.1 shows block diagram that explains the flow of our research.

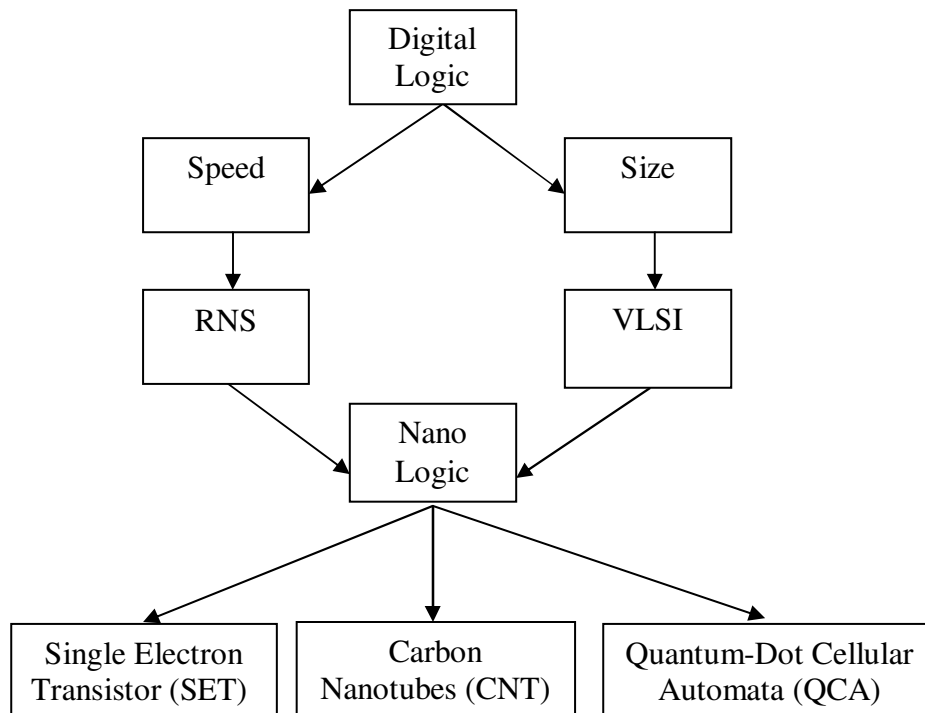


Fig. 1.1. Block diagram that shows the research logical flow

1.6 Thesis Organization

This thesis contains eight chapters. Chapter one is introduction, Chapter two starts with lecture reviews of residue number system and quantum-dot cellular automata system. Chapter three presents the new binary to residue number system conversion method. Chapter four presents the new scaling methods. VLSI RNS adder and subtractor implementation is presented in chapter five. Chapter six presents the new quantum boolean circuit construction reduction methodology. Chapter seven presents the QCA implementation of pipeline array and chapter eight discusses the summary of the thesis work and future research work. The dissertation also includes an appendix that shows the updated FPGA, Cadence Encounter and Virtuoso procedures.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The Residue number system has attracted researchers due to its advantage in the modular arithmetic operations such as addition, subtraction and multiplication since RNS provides the ability to add, subtract or multiply without the need to wait for the carry propagation as required by the weighted number systems. Also, RNS has shown significant efficiency in implementing Discrete Fourier Transformation and digital filters. In addition, quantum-dot cellular automata has showed the ability to implement both combinational and sequential logic devices. It is one of the new emerging nanotechnology and it has the potential to become the future of CMOS technology due to its ultra-power and small size. This chapter provides a review on the RNS and QCA topics that are relevant to the dissertation works. Section 2.2 introduces addition, subtraction and multiplication RNS operations. Section 2.3, 2.4, 2.5 and 2.6 give an overview on conversion from RNS to binary, sign detection, RNS scaling and RNS fast processing applications. Section 2.7 presents review for QCA topics.

2.2 RNS Arithmetic Operations

Residue Number System is an unweighted system with carry-free and borrow-free arithmetic operations. Addition, subtraction and multiplication are carried out on each residue digit concurrently and independently. This simplifies supporting parallel high-speed concurrent computation.

Addition can be accomplished simply by adding the small integer values together.

The following equation explains the RNS addition operation.

$$(a + b)_{10} \Leftrightarrow \left| \left| a \right|_m + \left| b \right|_m \right|_m \quad (2.1)$$

Where a and b are integers. For example, the addition of the decimal number $(10 + 8)_{10}$ using the modulo set (2, 3, 5) is illustrated below.

$$\begin{array}{r} (10)_{10} \xrightarrow{RNS} (0,1,0)_{2,3,5} \\ + (8)_{10} \xrightarrow{RNS} + (0,2,3)_{2,3,5} \\ \hline (0,3,3)_{2,3,5} \\ (0,3,3)_{2,3,5} \xrightarrow{RNS^{-1}} (18)_{10} \end{array}$$

The subtraction operation can be performed similar to the addition operation using the additive inverse of subtrahend (equation 1.7)

For example, the subtraction of decimal number $(18 - 10)$ using the modulo set (2, 3, 5) is illustrated below

$$\begin{array}{r} (-10)_{10} \xrightarrow{RNS} (0,2,0)_{2,3,5} \\ \text{and} \\ (18)_{10} \xrightarrow{RNS} (0,3,3)_{2,3,5} \\ - (10)_{10} \xrightarrow{RNS} + (0,2,0)_{2,3,5} \\ \hline (0,2,3)_{2,3,5} \\ (0,2,3)_{2,3,5} \xrightarrow{RNS^{-1}} (8)_{10} \end{array}$$

Multiplication can be accomplished in a manner similar to addition and subtraction as follows.

$$(a * b)_{10} \Leftrightarrow \left| \left| a \right|_m * \left| b \right|_m \right|_m \quad (2.2)$$

For Example, the multiplication of $(3 * 9)_{10}$ using modulo set $(2, 3, 5)$ is illustrated below.

$$\begin{array}{r}
 (3)_{10} \xrightarrow{RNS} (1,0,3)_{2,3,5} \\
 * (9)_{10} \xrightarrow{RNS} * (1,0,4)_{2,3,5} \\
 \hline
 (1,0,2)_{2,3,5} \\
 (1,0,2)_{2,3,5} \xrightarrow{RNS^{-1}} (27)_{10}
 \end{array}$$

2.3 Conversion from RNS to Binary

Several methods are available for converting residue to binary system, most of these methods are based on two techniques the first one is Chinese Remainder Theorem (CRT) and the second one is Mixed Radix Converters (MRC) [9].

2.3.1 Chinese Remainder Theorem Conversion

The Chinese Remainder Theorem is a basic conversion method. The problem associated with CRT approach is the requirements of M modulo adders.

Definition

For any given set of relatively prime modulo set $(m_1, m_2, m_3, \dots, m_n)$, the residue representation of an binary number X is $(x_1, x_2, x_3, \dots, x_n)$. Number X can be represented as

$$X = \left| \sum_{i=1}^n m_i \hat{m}_i \left| \begin{array}{c} x_i \\ \hat{m}_i \end{array} \right| \right|_M$$

Where $0 \leq x_i < m_i$, $M = \prod_{i=1}^n m_i$

$$\hat{m}_i = \frac{M}{m_i}, \quad \text{and} \quad \left| \begin{array}{c} 1 \\ \hat{=} \\ \hat{m}_i \end{array} \right|_{m_i} \text{ is the multiplicative inverse of } \left| \begin{array}{c} \hat{=} \\ \hat{m}_i^{-1} \end{array} \right|_{m_i}$$

$$\left| \begin{array}{c} 1 \\ \hat{=} \\ \hat{m}_i \end{array} \right|_{m_i} = \left| \begin{array}{c} \hat{=} \\ \hat{m}_i^{m_i-2} \end{array} \right|_{m_i}$$

To illustrate how the method works, consider the following example

Example:

For modulo set (5, 7, 11), find the decimal number whose residue representation is

(4, 5, 7) and $M = 385$

Solution:

$$\hat{m}_1 = \frac{385}{5} = 77, \quad \hat{m}_2 = \frac{385}{7} = 55, \quad \hat{m}_3 = \frac{385}{11} = 35$$

$$\hat{m}_1^{-1} = \left| \begin{array}{c} 1 \\ \hat{=} \\ 77 \end{array} \right|_5 = 3, \quad \hat{m}_2^{-1} = \left| \begin{array}{c} 1 \\ \hat{=} \\ 55 \end{array} \right|_7 = 6, \quad \hat{m}_3^{-1} = \left| \begin{array}{c} 1 \\ \hat{=} \\ 35 \end{array} \right|_{11} = 6$$

$$\left| \begin{array}{c} x_1 \\ \hat{=} \\ \hat{m}_1 \end{array} \right|_{m_1} = |4 * 3|_5 = 2, \quad \left| \begin{array}{c} x_2 \\ \hat{=} \\ \hat{m}_2 \end{array} \right|_{m_2} = |5 * 6|_7 = 2, \quad \left| \begin{array}{c} x_3 \\ \hat{=} \\ \hat{m}_3 \end{array} \right|_{m_3} = |7 * 6|_{11} = 9$$

$$X = \left| \begin{array}{c} \sum_{i=1}^3 \hat{m}_i \left| \begin{array}{c} x_i \\ \hat{=} \\ \hat{m}_i \end{array} \right|_{m_i} \\ \hat{=} \\ \hat{m}_i \end{array} \right|_M = |77 * 2 + 55 * 2 + 35 * 9|_{385} = 194$$

$$(4,5,7)_{5,7,11} \xrightarrow{RNS^{-1}} (194)_{10}$$

2.3.2 Mixed Radix Conversion

Converting RNS back to binary numbering system is challenging process and one of the methods that can be used to do this conversion is done by using mixed radix number system [9]. The problem associated with MRC lies in the complexity of finding mixed radix digits.

Definition

An integer number X may be expressed in mixed radix form as

$$X = a_1 + a_2 \times m_1 + a_3 \times m_1 \times m_2 + \dots + a_n \prod_{i=1}^{n-1} m_i \quad (2.3)$$

Where the mixed radix digits are determined sequentially by following manner starting with a_1 .

All terms in equation (2.4) except the first are multiples of m_1 , consequently

$$a_1 = |X|_{m_1} = x_1$$

To obtain a_2

- Subtract $X - a_1$
- Divide by m_1 $\frac{(X - a_1)}{m_1}$
- Then take mod m_2 $\left| \frac{(X - a_1)}{m_1} \right|_{m_2}$

and by successively subtracting and dividing in residue notation, all mixed radix digits can be obtained. In general form mixed radix digits can be defined as:

$$X = a_1 + a_2 \times m_1 + a_3 \times m_1 \times m_2 + \dots + a_n \prod_{i=1}^{n-1} m_i$$

The following example illustrates how this method works.

Example:

Find the decimal number that is represented by (1, 1, 3) for modulo set (2, 5, 7).

Solution:

Modulo set	2	5	7		
RNS	1	1	3	\Leftrightarrow	$a_1 = 1$
Subtract a_1	1 1 0				
	0	4	3	\Leftrightarrow	$ x_j - x_1 _{m_j} = x_j + m_j - x_1 _{m_j} _{m_j}$

Multiply by	$\begin{vmatrix} 1 \\ = \\ 2 \end{vmatrix}_{m_1}$		2	5	\Leftrightarrow	$a_2 = 2$
-------------	---	--	---	---	-------------------	-----------

Subtract a_2	2 0				
	0	5			

Multiply by	$\begin{vmatrix} 1 \\ = \\ 5 \end{vmatrix}_7 = 3$		1	\Leftrightarrow	$a_3 = 1$
-------------	---	--	---	-------------------	-----------

$$X = a_1 + a_2 * m_1 + a_3 * m_1 * m_2 = 1 + 2 * 2 + 1 * 2 * 5 = 15$$

$$(2,0,1)_{3,5,7} \xrightarrow{RNS^{-1}} (15)_{10}$$

2.4 Residue Number System Sign Detection

In residue number system sign detection is relatively a difficult operation compared to weighted systems.. The sign in RNS is a function of each digit and it is closely related to magnitude determination.

In section 1.3 case 1, showed that all numbers in range $[0, M/2 - 1]$ are positive and all numbers in range $[M/2, M-1]$ are negative; and in case 2, all numbers in range $[0, (M - 1)/2]$ are positive and all numbers in range $[(M+1)/2, M-1]$ are negative. Magnitude sign detection algorithm based on mixed radix process and case 1 is found in [9].

Definition

Select the last modulo in the mixed radix conversion to be even (m_n). Then, it is easily

seen that $0 \leq a_n \leq \frac{m_n}{2} - 1$ implies that $|X|_M$ falls into the interval $[0, \frac{M}{2} - 1]$ and therefore,

can be considered positive. Conversely, $\frac{m_n}{2} \leq a_n \leq m_n - 1$ implies that $|X|_M$ falls into the

interval $[\frac{M}{2}, M - 1]$ and therefore, can be considered negative [3].

Example:

Find the sign of $[1, 2, 3]$ for modulo set $(2, 3, 5)$

Solution:

The dynamic range is $M = 30$ and number in range of $(0, 14)$ are positive and numbers in range of $(15, 29)$ are negative.

First arrange the ordering of the modulo to place the even modulo 2 on the end

Modulo set	5	3	2		
RNS	3	2	1	\Leftrightarrow	$a_1 = 3$

$$\begin{array}{r} \text{Subtract } a_1 \\ \hline 3 \quad 3 \quad 0 \\ 0 \quad 2 \quad 1 \end{array}$$

$$\text{Multiply by } \left. \begin{array}{c} 1 \\ \hline 5 \end{array} \right|_{m_1} \quad \begin{array}{cc} & 1 \quad 1 \\ & \hline & 1 \quad 1 \end{array} \Leftrightarrow a_2 = 1$$

$$\begin{array}{r} \text{Subtract } a_2 \\ \hline 1 \quad 0 \\ 0 \quad 1 \end{array}$$

$$\text{Multiply by } \left. \begin{array}{c} 1 \\ \hline 3 \end{array} \right|_2 \quad \begin{array}{cc} & 1 \\ & \hline & 1 \end{array} \Leftrightarrow a_3 = 1$$

$a_3 = 1$ implies that number (1, 2, 3) falls into interval (15, 29). It follows that (1, 2, 3) is negative

$$X = a_1 + a_2 * m_1 + a_3 * m_1 * m_2 = 3 + 1 * 5 + 1 * 5 * 3 = 23$$

2.5 Scaling in Residue Number System

Scaling is comparatively a difficult RNS operation. Scaling is an essential operation in several signal processing algorithms. In binary system, the scaling constant is usually a power of 2. Many scaling techniques reported in RNS literatures [12]. The RNS scaling by constant Q is defined as

$$Y = \left[\frac{X}{Q} \right] \quad (2.4)$$

$$Y = \frac{X - |X|_Q}{Q} \quad (2.5)$$

and in RNS representation

$$|Y|_{m_i} = \left| \frac{X - |X|_Q}{Q} \right|_{m_i} \quad (2.6)$$

If Q , the divisor is a modulo or product of the first powers of modulo, multiplicative inverse property can be used to simplify the division by Q

$$|Y|_{m_i} = \left| \frac{X - |X|_Q}{\overline{Q}} \right|_{m_i} \quad (2.7)$$

2.6 RNS Fast Processing Applications

The advantages of residue number system are discussed in several publications and books [9]. Carry- free computation, simplified and fast addition and multiplication, which helps to obtain parallel architectures, are among the important advantages. Potential applications for RNS processors include fast DSP applications, adaptive array processing, Kalman filtering, Fast Fourier Transforms, and image processing for communications, surveillance, and intelligence systems.

2.7 Quantum-Dot Cellular Automata (QCA)

In this section, we present background QCA material that will be helpful to understand the QCA topics.

QCA Cell: A quantum cell can be viewed as a set of four charge containers or dots positioned at the corners of a square as shown in fig. 2.1a. Each QCA cell contains two mobile electrons that can move to any quantum dot through electron tunneling. Thus there are two equivalent electrons arrangement polarization $P = +1$ (Logic 1) and $P = -1$ (logic 0).

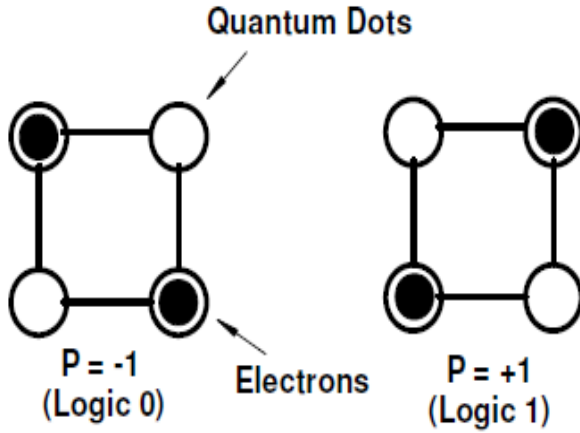


Fig. 2.1a. QCA cells and binary encoding

QCA Majority Gate: The basic QCA logic element is a majority gate as shown in fig. 2.1b. It produces an output of one if the majority of inputs one. The classical AND and OR gates can be realized with majority gate by fixing one of three inputs as 0 or 1 respectively, as follows:

$$M(A,B,0) = AB \quad (2.8a)$$

$$M(A,B,1) = A+B \quad (2.8b)$$

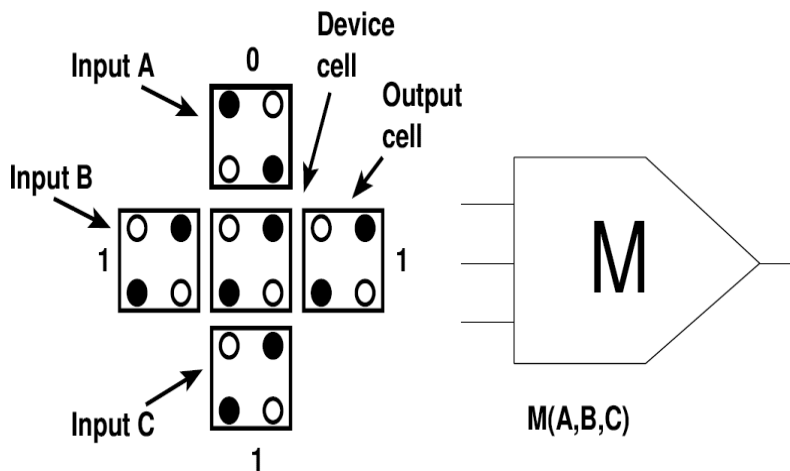


Fig. 2.1b. QCA majority gate

QCA Inverter: QCA cells layout of an inverter is shown in fig. 2.1c. The polarization of the output QCA cell “out” is opposite of input QCA cell “in”.

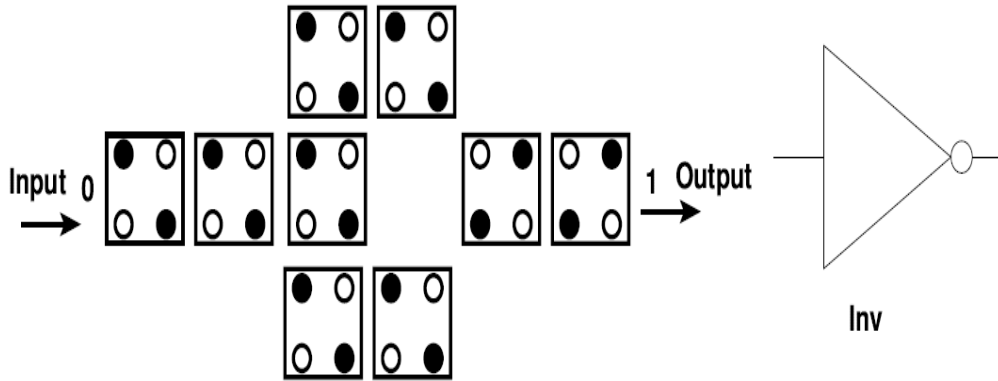


Fig. 2.1c. QCA inverter gate

QCA Wire: There are two types of QCA wires normal (also called 90°) and diagonal (also called 45°). Fig. 2.1d shows the two QCA wire types with logic one polarized.

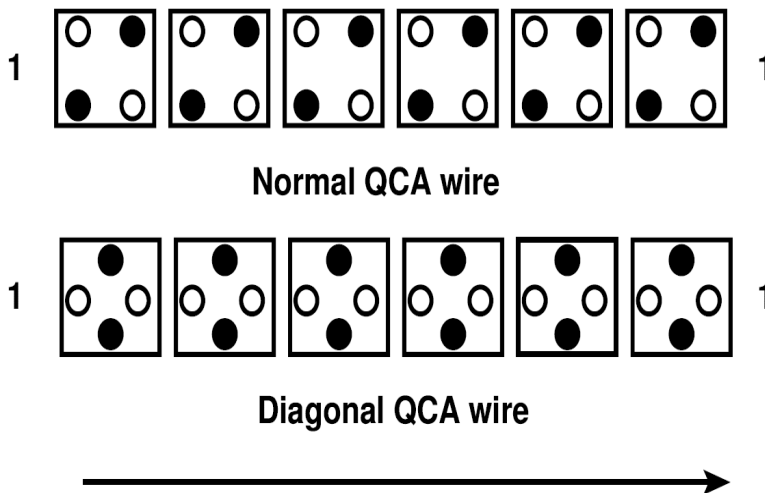


Fig. 2.1d. QCA wire types

2.8 QCA Clocking

QCA cells use four phase scheme namely clock 1, clock 2, clock 3 and clock 4 as shown in fig. 2.2a. Every clock is 90° out of phase form its pervious clock and each clock has four states namely switch, hold, release, and relax [77]. In switch state, QCA cells start polarized. In hold state, the cells retain it polarization and during release and relax states, QCA cells are unpolarized as shown fig. 2.2b.

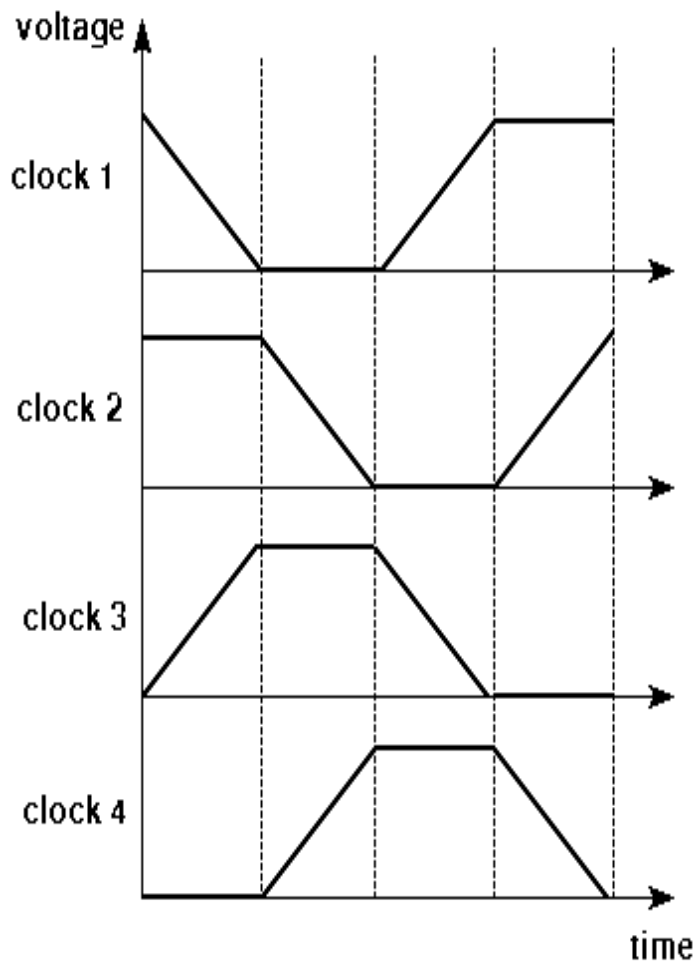


Fig. 2.2a. QCA clock phases; each clock lagging its prior by 90°

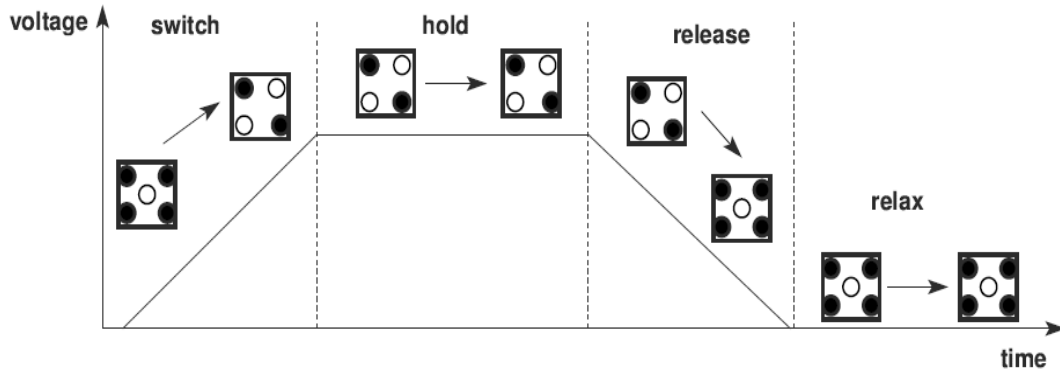


Fig. 2.2b. Four QCA interdot barrier states

Table 2.1 shows the equivalent QCA expressions for major Boolean gates.

TABLE 2.1

MAJORITY EXPRESSIONS AND DIAGRAMS FOR MAJOR BOOLEAN GATES

Boolean Gate	Boolean Function	Majority Expression	Majority Diagram
AND Gate	$F = AB$	$M(A, B, 0)$	
OR Gate	$F = A + B$	$M(A, B, 1)$	
NAND Gate	$F = \overline{AB}$ $= \overline{A} + \overline{B}$	$M(\overline{A}, \overline{B}, 1)$	
NOR Gate	$F = \overline{A + B}$ $= \overline{A}\overline{B}$	$M(\overline{A}, \overline{B}, 0)$	
Exclusive -OR (XOR) Gate	$F = A \oplus B$ $= A\overline{B} + \overline{A}B$	$M(M(A, \overline{B}, 0), M(\overline{A}, B, 0), 1)$	
Exclusive -NOR (XNOR) Gate	$F = A \odot B$ $= AB + \overline{A}\overline{B}$	$M(M(A, B, 0), M(\overline{A}, \overline{B}, 0), 1)$	

2.9 Conclusion

Residue number system is a robust parallel system that supports high speed concurrent arithmetic applications such as addition, subtraction and multiplication in modular levels. However it suffers from some drawbacks. RNS has weakness such as conversion process, scaling, division, overflow detection and magnitude comparison. Quantum-dot cellular automata also is one of the new emerging nanotechnology and it is the future of CMOS technology due to its ultra-power and small size. Quantum-dot cellular automata uses different logic devices to design circuits other than Boolean logic devices. Converting Boolean circuits to QCA Boolean is not simple process due to complexity of QCA and existing Boolean reduction methods do not work with QCA logic. In thesis, we present a new binary to residue number system, new RNS scaling methodology, RNS adder and subtractor implementation, a new QCA construction reduction method and QCA pipeline array implementation.

CHAPTER 3

NOVEL PARALLEL - PREFIX STRUCTURE BINARY TO RESIDUE NUMBER SYSTEM CONVERSION METHOD

3.1. Introduction

In the last decade, Residue Number System (RNS) has received increased attention due to its ability to support high speed concurrent arithmetic applications [1-3] such as Fast Fourier Transform (FFT), image processing and digital filters utilizing the efficiencies of RNS arithmetic in addition and multiplication. The advancements in Very Large Scale Integration (VLSI) technology and demand for parallelism computation have enabled researchers to consider RNS as an alternative approach to high speed concurrent arithmetic.

Several methods are found in literature for binary to RNS conversion. Alia and Martinelli [4] have proposed a method for binary to residue conversion based on powers of 2. A modification to the above method was proposed by Cappocelli and Giancarlo [5]. Mohan [6] has proposed a similar method but with difference that his method is based on the cyclic property of power of 2 modulo set. Behrooa[7] proposed a table lookup schemes for binary to Residue conversions.

In this chapter, we present a novel binary to residue number system conversion method. The organization of this chapter is as follows. Section two explains RNS system. In section three, we present new conversion from binary to RNS algorithm. Section four and five show illustrative example and implementation selection techniques. Section six is comparison between the new method and pervious work. Conclusion is in section seven.

3.1. Residue Number System

Any n-bit nonnegative integer number X, in the range $0 \leq X \leq 2^n - 1$ is represented in binary

$$\text{number system as } X = 2^{n-1} b_{n-1} + \dots + 2^2 b_2 + 2 b_1 + b_0 = \sum_{j=0}^{n-1} 2^j b_j$$

where $b_j \in \{0, 1\}$.

Meanwhile in RNS, X is represented by k residue digits x_i as $X = \{x_1, x_2, x_3, \dots, x_k\}$ where $x_i = X \bmod m_i$ and m_i belong to set of relatively prime modulo; $m_i \in \{m_1, m_2, m_3, \dots, m_k\}$ [9]. If the modulo are relatively prime numbers, there is a unique RNS representation for each integer in range $0 \leq X \leq \prod_{i=1}^s m_i$

3.2. New Novel Conversion Method from Binary to Residue Representation

As shown above an integer number X can be represented in binary system as

$$X = 2^{n-1} b_{n-1} + \dots + 2^2 b_2 + 2 b_1 + b_0 = \sum_{j=0}^{n-1} 2^j b_j$$

And RNS representation of number X is

$$\begin{aligned} |X|_m &= \left| \sum_{j=0}^{n-1} 2^j b_j \right|_m && \text{for } m > 2 && \text{for } m > 2 \\ &= \left| \sum_{j=0}^{n-1} 2^j \right|_m |b_j|_m && && \end{aligned} \quad (3.1)$$

Let $M_{A_1 A_0} = (A_1, A_0)[Y_0, Y_1, Y_2, Y_3]$ denotes a 2-bit multiplexer where the 2 control bits (A_1, A_0) select the inputs (Y_0, Y_1, Y_2, Y_3) to be outputted

Lemma 3.1: For any pair of bits b_j & b_i for j & $i \geq 0$,

$$\left| 2^j \right|_m b_j + \left| 2^i \right|_m b_i \Big|_m = |X_{ji}|_m$$

can be implemented using 2-bit multiplexer :

$$M_{ji} = (b_j, b_i)[0, \left| 2^i \right|_m, \left| 2^j \right|_m, \left| \left| 2^j \right|_m + \left| 2^i \right|_m \right|_m] \quad (3.2)$$

Where the control bits (A_1, A_0) equal (b_j, b_i)

Proof:

Rewrite equation $\left\| 2^j \Big|_m b_j + 2^i \Big|_m b_i \Big|_m$ as

$$(0\bar{b}_j\bar{b}_i) + (2^i \Big|_m \bar{b}_j b_i) + (2^j \Big|_m b_j \bar{b}_i) + (2^j \Big|_m + 2^i \Big|_m \Big|_m b_j b_i)$$

This is equivalent to 2-bit multiplexer M_{ji} with control bits (A_1, A_0) equal (b_j, b_i) . Fig. 3.1 shows the implementation for equation (3.2) with $b_j = b_1$ and $b_i = b_0$

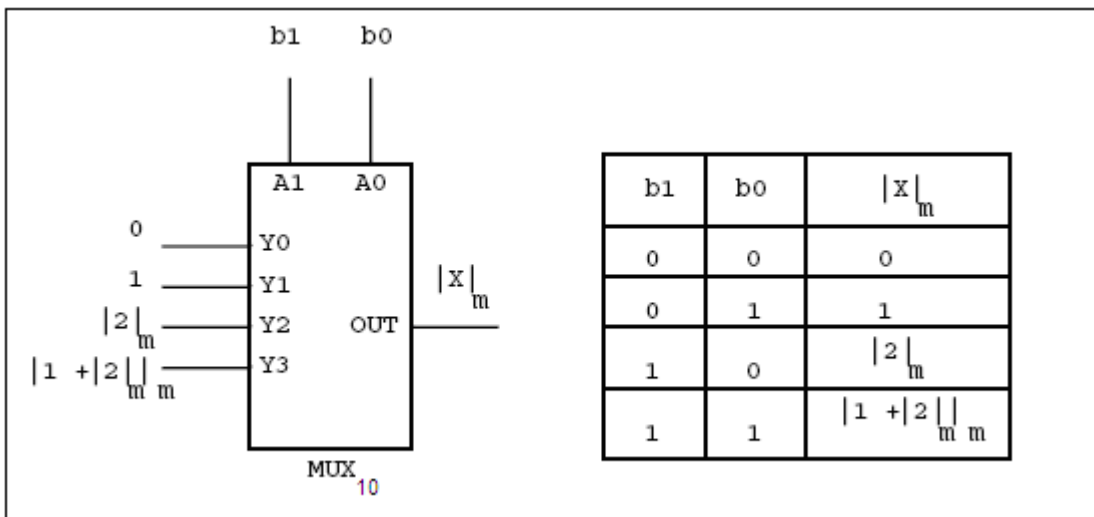


Fig. 3.1. Two bits (b1 & b0) binary to residue number system conversion

This pre-processing operator M_{ji} is represented in acyclic graph as node " \circ " in fig. 3.2a, where all the inputs are constants and pre-calculated .

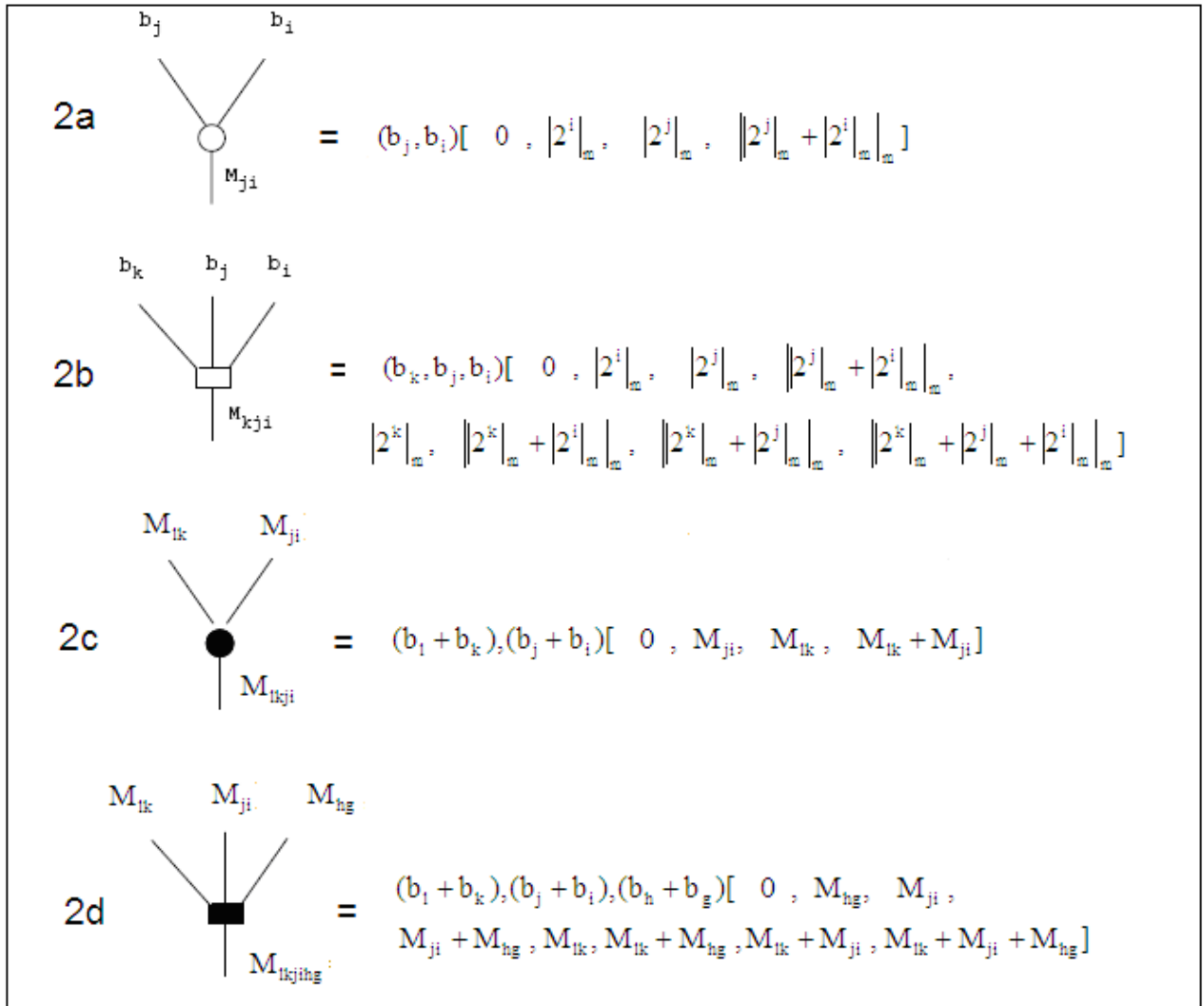


Fig. 3.2. Prefix logic operation and their implementation

In three bit system, let $M_{A_2A_1A_0} = (A_2, A_1, A_0)[Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7]$ denotes a 3-bit multiplexer where the 3 control bits (A_2, A_1, A_0) select the inputs ($Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7$) to be outputted.

Lemma 3.2: For any three bits b_k, b_j & b_i for k, j & $i \geq 0$,

$$\left| 2^k \mid_m b_k + 2^j \mid_m b_j + 2^i \mid_m b_i \right|_m = \left| X_{kji} \right|_m \text{ can be}$$

implemented using 3-bit multiplexer :

$$M_{kji} = (b_k, b_j, b_i)[0, |2^i|_m, |2^j|_m, ||2^j|_m + |2^i|_m|_m, |2^k|_m, ||2^k|_m + |2^i|_m|_m,$$

$$\left[2^k \Big|_m + 2^j \Big|_m, \left[2^k \Big|_m + 2^j \Big|_m + 2^i \Big|_m \right] \right] \tag{3.3}$$

Where control bits (A₂, A₁, A₀) equal (b_k, b_j, b_i)

Proof:

Rewrite $\left[2^k \Big|_m b_k + 2^j \Big|_m b_j + 2^i \Big|_m b_i \right]_m$ as

$$\begin{aligned} & (0 \cdot \bar{b}_k \cdot \bar{b}_j \cdot \bar{b}_i) + (2^i \Big|_m \cdot \bar{b}_k \bar{b}_j \cdot b_i) + (2^j \Big|_m \cdot \bar{b}_k \cdot b_j \cdot \bar{b}_i) + \left(\left[2^j \Big|_m + 2^i \Big|_m \right]_m \cdot \bar{b}_k \cdot b_j \cdot b_i \right) + \left(2^k \Big|_m \cdot b_k \cdot \bar{b}_j \cdot \bar{b}_i \right) + \\ & \left(\left[2^k \Big|_m + 2^i \Big|_m \right]_m \cdot b_k \bar{b}_j \cdot b_i \right) + \left(\left[2^k \Big|_m + 2^j \Big|_m \right]_m \cdot b_k \cdot b_j \cdot \bar{b}_i \right) + \\ & \left(2^k \Big|_m + 2^j \Big|_m + 2^i \Big|_m \cdot b_k \cdot b_j \cdot b_i \right) \end{aligned}$$

Above equation is equivalent to 3-bit multiplexer with b_k, b_j & b_i as selection control inputs. Fig. 3.3 shows the implementation for equation (3.3) with b_k = b₂ b_j = b₁ and b_i = b₀

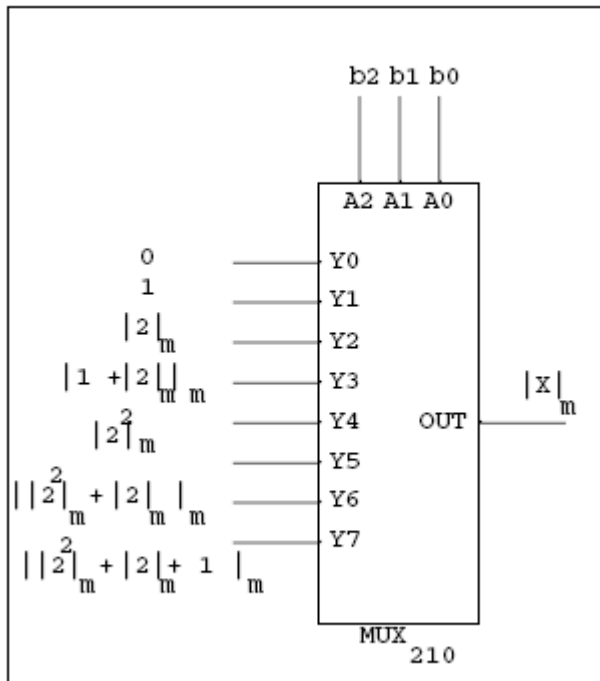


Fig. 3.3. Three bits (b₂, b₁, b₀) binary to residue number system conversion

This pre-processing operator M_{kji} is represented in acyclic graph as node "□" in fig. 3.2b, where all the inputs are constants and pre-calculated.

Theorem 3.1: For any two pairs of bits (b_1 & b_k) (b_j & b_i for j, i, l & $k \geq 0$ with the given expression

$$\left| 2^l \Big|_m b_1 + 2^k \Big|_m b_k + 2^j \Big|_m b_j + 2^i \Big|_m b_i \right|_m = \left| X_{lkji} \right|_m$$

can be implemented using 2-bit multiplexer

$$M_{lkji} = (b_1 + b_k), (b_j + b_i) [0, M_{ji}, M_{lk}, M_{lk} + M_{ji}] \tag{3.4}$$

Where control bits (A_1, A_0) equal ($b_1, +b_k, b_j, +b_i$)

Proof:

$$\left| X_{lkji} \right|_m = \left| 2^l \Big|_m b_1 + 2^k \Big|_m b_k + 2^j \Big|_m b_j + 2^i \Big|_m b_i \right|_m$$

$$\left| X_{lkji} \right|_m = \left| M_{lk} + M_{ji} \right|_m \tag{3.5}$$

Where $M_{lk} = \left| 2^l \Big|_m b_1 + 2^k \Big|_m b_k \right|_m$ and $M_{ji} = \left| 2^j \Big|_m b_j + 2^i \Big|_m b_i \right|_m$ by Lemma 1

Let $b_{lk} = (b_1 + b_k)$ and $b_{ji} = (b_j + b_i)$

Rewrite equation (3.5) as $(0 \cdot \bar{b}_{lk} \cdot \bar{b}_{ji}) + (M_{ji} \cdot \bar{b}_{lk} \cdot b_{ji}) + (M_{lk} \cdot b_{lk} \cdot \bar{b}_{ji}) + (M_{lk} + M_{ji} \cdot b_{lk} \cdot b_{ji})$.

And this is equivalent to 2-bit multiplexer M_{lkji} with control bits (A_1, A_0) equal ($b_1 + b_k, b_j + b_i$). Fig.

3.4 shows implementation for two pair bits (b_3, b_2) & (b_1, b_0)

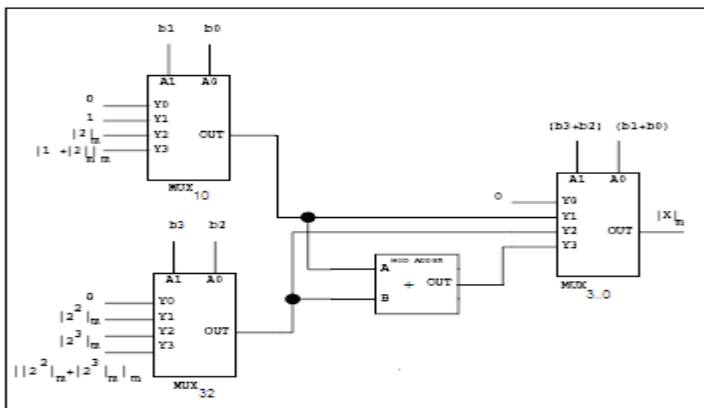


Fig. 3.4. Four bits binary to RNS $\left| X_{3,0} \right|_m = \left| 2^3 \Big|_m b_3 + 2^2 \Big|_m b_2 + 2^1 \Big|_m b_1 + b_0 \right|_m$

Lemma 3.3:

Combining two pairs of bits $(b_1 \& b_k)$ $(b_j \& b_i)$ requires one 2-bit multiplexer and one 2 input mod adder. The delay time $\tau_{\text{Total}} = \tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

Proof:

Equation (3.4) and fig. 3.4 show that $(b_1 + b_k), (b_j + b_i)[0, M_{ji}, M_{lk}, M_{lk} + M_{ji}]$

is equivalent to one 2-bit multiplexer and one 2-input mod adder; and delay time is equal to

$$\tau_{\text{mux}_2} + \tau_{\text{modadder}_2} .$$

Fig. 3.2c represents acyclic graph "●" for node M_{lkji} where M_{lk} & M_{ji} are inputs.

Lemma 3.4:

The parallel prefix operator ● has the following properties

1) Commutative

$$M_{lk} \bullet M_{ji} = M_{ji} \bullet M_{lk}$$

2) Associative

$$M_{lk} \bullet (M_{hg} \bullet M_{ji}) = (M_{lk} \bullet M_{hg}) \bullet M_{ji}$$

Proof:

$$M_{lk} \bullet M_{ji} = M_{lkji}$$

$$= (b_1 + b_k), (b_j + b_i)[0, M_{ji}, M_{lk}, M_{lk} + M_{ji}]$$

$$= \left\| 2^1 \Big|_m b_1 + 2^k \Big|_m b_k + 2^j \Big|_m b_j + 2^i \Big|_m b_i \right\|_m \quad (3.6)$$

$$M_{ji} \bullet M_{lk} = M_{jilk}$$

$$= (b_j + b_i), (b_1 + b_k)[0, M_{kl}, M_{ji}, M_{ji} + M_{lk}]$$

$$= \left\| 2^j \Big|_m b_j + 2^i \Big|_m b_i + 2^1 \Big|_m b_1 + 2^k \Big|_m b_k \right\|_m \quad (3.7)$$

Both expressions (3.6) and (3.7) are the same by commutative property of "+" hence \bullet operator is commutative

$$\begin{aligned}
 M_{lk} \bullet (M_{hg} \bullet M_{ji}) &= M_{lk} \bullet M_{hgji} \\
 &= M_{lkhgji} \\
 &= \left| 2^l \Big|_m b_k + 2^k \Big|_m b_k + 2^h \Big|_m b_h + 2^g \Big|_m b_g + 2^j \Big|_m b_j + 2^i \Big|_m b_i \right|_m
 \end{aligned} \tag{3.8}$$

$$\begin{aligned}
 (M_{lk} \bullet M_{hg}) \bullet M_{ji} &= M_{lkhg} \bullet M_{ji} \\
 &= M_{lkhgji} \\
 &= \left| 2^l \Big|_m b_k + 2^k \Big|_m b_k + 2^h \Big|_m b_h + 2^g \Big|_m b_g + 2^j \Big|_m b_j + 2^i \Big|_m b_i \right|_m
 \end{aligned} \tag{3.9}$$

Both expressions (3.8) and (3.9) are the same by associative property of "+" hence \bullet operator is associative

Theorem 3.2: For any three pairs of bits (b_l & b_k),

(b_j & b_i) and (b_h & b_g) for l, k, j, i, h & $g \geq 0$ with given expression

$$\left| 2^l \Big|_m b_l + 2^k \Big|_m b_k + 2^j \Big|_m b_j + 2^i \Big|_m b_i + 2^h \Big|_m b_h + 2^g \Big|_m b_g \right|_m = \left| X_{lkjihg} \right|_m$$

can be implemented using 3-bit multiplexer

$$\begin{aligned}
 M_{lkjihg} = (b_l + b_k), (b_j + b_i), (b_h + b_g) [0, M_{hg}, & M_{ji}, M_{ji} + M_{hg}, M_{lk}, M_{lk} + M_{hg}, \\
 M_{lk} + M_{ji}, M_{lk} + M_{ji} + M_{hg}] & \tag{3.10}
 \end{aligned}$$

Where control bits (A_2, A_1, A_0) equal ($b_l+b_k, b_j+b_i, b_h+b_g$)

Proof:

$$\begin{aligned}
 \left| X_{lkjihg} \right|_m &= \left| 2^l \Big|_m b_l + 2^k \Big|_m b_k + 2^j \Big|_m b_j + 2^i \Big|_m b_i + 2^h \Big|_m b_h + 2^g \Big|_m b_g \right|_m \\
 \left| X_{lkjihg} \right|_m &= \left| M_{lk} + M_{ji} + M_{hg} \right|_m
 \end{aligned} \tag{3.11}$$

Where $M_{lk} = \left| 2^l \Big|_m b_l + 2^k \Big|_m b_k \right|_m$,

$$M_{ji} = \left\lfloor 2^j \right\rfloor_m b_j + \left\lfloor 2^i \right\rfloor_m b_i \quad \text{and}$$

$$M_{hg} = \left\lfloor 2^h \right\rfloor_m b_h + \left\lfloor 2^g \right\rfloor_m b_g \quad \text{by Lemma 1}$$

Let $b_{lk} = (b_l + b_k)$, $b_{ji} = (b_j + b_i)$ and $b_{hg} = (b_h + b_g)$

Rewrite equation (3.11) as

$$\begin{aligned} & (0 \cdot \bar{b}_{lk} \cdot \bar{b}_{ji} \cdot \bar{b}_{hg}) + (M_{hg} \cdot \bar{b}_{lk} \cdot \bar{b}_{ji} \cdot b_{hg}) + (M_{ji} \cdot \bar{b}_{lk} \cdot b_{ji} \cdot \bar{b}_{hg}) + ((M_{ji} + M_{hg}) \cdot \bar{b}_{lk} \cdot b_{ji} \cdot b_{hg}) + \\ & (M_{lk} \cdot b_{lk} \cdot \bar{b}_{ji} \cdot \bar{b}_{hg}) + ((M_{lk} + M_{hg}) \cdot b_{lk} \cdot \bar{b}_{ji} \cdot b_{hg}) + \\ & ((M_{lk} + M_{ji}) \cdot b_{lk} \cdot b_{ji} \cdot \bar{b}_{hg}) + \\ & ((M_{lk} + M_{ji} + M_{hg}) \cdot b_{lk} \cdot b_{ji} \cdot b_{hg}) \end{aligned}$$

This is equivalent to 3-bit multiplexer M_{lkjihg} with control bits (A_2, A_1, A_0) equal $(b_l + b_k, b_j + b_i, b_h + b_g)$

Fig. 3.5 shows implementation for three pairs of bits (b_5, b_4) , (b_3, b_2) & (b_1, b_0)

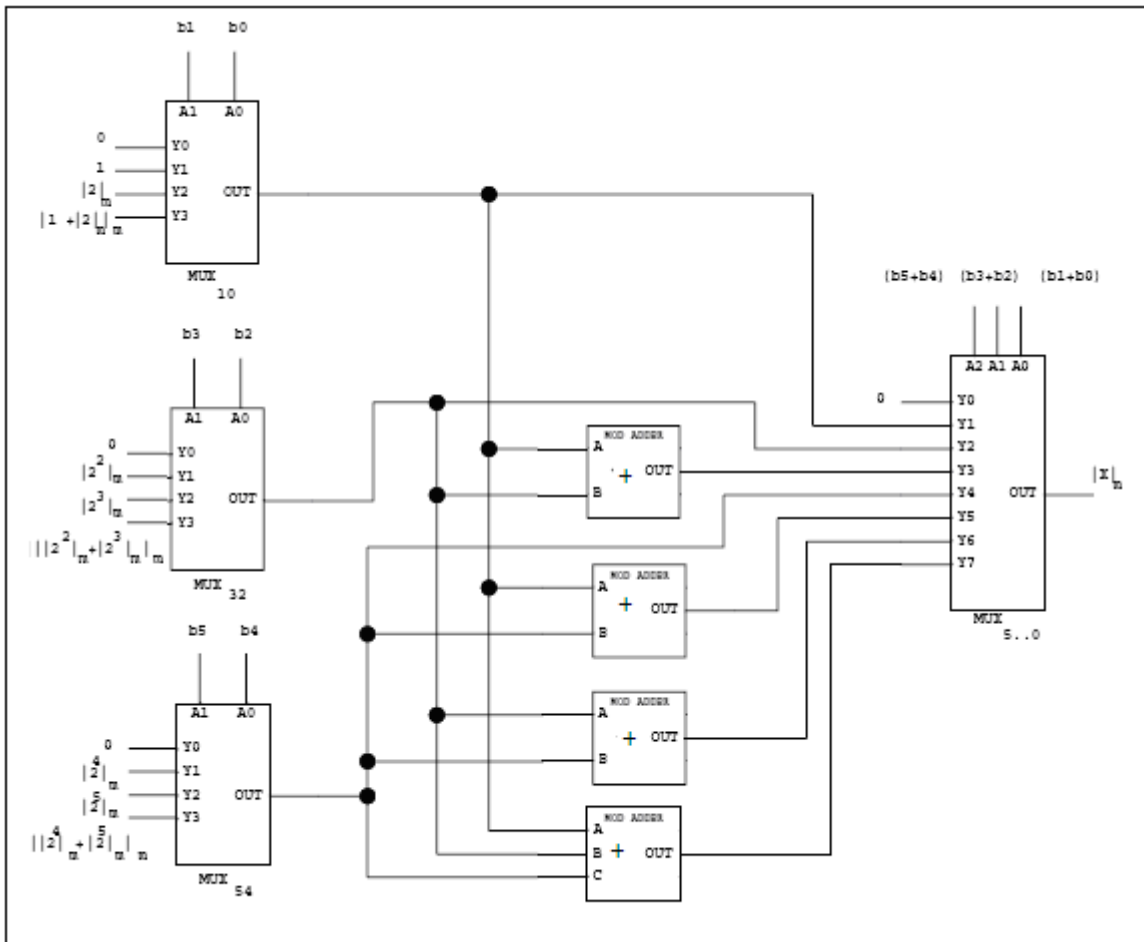


Fig. 3.5. Six bits binary to residue number system conversion

$$|X_{5..0}|_m = \left\| 2^5 \Big|_m b_5 + 2^4 \Big|_m b_4 + 2^3 \Big|_m b_3 + 2^2 \Big|_m b_2 + 2^1 \Big|_m b_1 + b_0 \right\|_m$$

Lemma 3.5:

Combining three pairs of bits (b_l & b_k), (b_j & b_i) & (b_h & b_g) requires one 3-bit multiplexer and three 2-input mod adder and one 3-input mod adder. The delay time equals

$$\tau_{\text{Total}} = \tau_{\text{mux}_3} + 2\tau_{\text{modadder}_2} = \tau_{\text{mux}_3} + \tau_{\text{modadder}_3}$$

Proof:

Equation (3.10) and fig. 3.5 show that $(b_l + b_k), (b_j + b_i), (b_h + b_g)[0, M_{hg},$

$M_{ji}, M_{ji} + M_{hg}, M_{lk}, M_{lk} + M_{hg}, M_{lk} + M_{ji}, M_{lk} + M_{ji} + M_{hg}]$ is equivalent to one 3-bit multiplexer and three 2-input mod adder and one 3-input mod adder; and delay time is equal to

$$\tau_{\text{mux}_3} + 2\tau_{\text{modadder}_2}$$

Fig. 3.2d represents acyclic graph for "■" for node M_{lkhgji} where M_{lk}, M_{hg} & M_{ji} are inputs

Lemma 3.6:

The parallel prefix operator ■ has the following properties

1) Commutative

$$M_{lkjihg} \blacksquare M_{tsrqpo} = M_{tsrqpo} \blacksquare M_{lkjihg}$$

2) Associative

$$M_{lkjihg} \blacksquare (M_{tsrqpo} \blacksquare M_{zyxwru}) = (M_{lkjihg} \blacksquare M_{tsrqpo}) \blacksquare M_{zyxwru}$$

Proof:

The proof is similar to Lemma 3.4

3.3. Illustrative Example

In this section, we will use to illustrate how theorem 3.1, theorem 3.2, lemma 3.1 and lemma 3.2 can be combined to design a binary to residue convertor. Fig. 3.6 shows how $|X|_m$ for $n = 8$ is computed.

In the first layer, using pre-processing operator \circ each consecutive pair of bits are group together (b_7, b_6) (b_5, b_4) (b_3, b_2) (b_1, b_0) creating nodes M_{76} , M_{54} , M_{32} , M_{10} . In the second layer, using parallel prefix operator \bullet each consecutive M node are combined (M_{76}, M_{54}) (M_{32}, M_{10}) forming nodes $M_{7..4}$ & $M_{3..0}$. In the last layer, using parallel prefix operator \bullet the last 2 M nodes are combined $(M_{7..4}, M_{3..0})$ forming node $M_{7..0} = |X_{7..0}|_m$. Fig. 3.7a shows the actual hardware implementation.

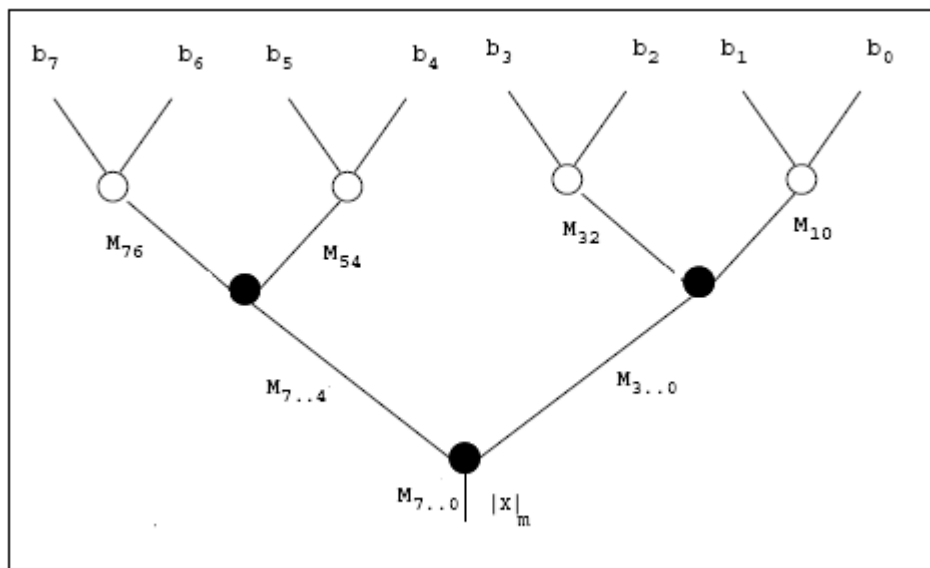


Fig. 3.6. Prefix structure of 8 bits binary to RNS

Total delay time for this example is calculated by counting the delay introduced by the operator in each layer

by using lemma 3.3 as follows

Layer 1: delay time is τ_{mux_2}

pre-processing operator \odot doesn't require an adder

Layer 2: delay time is $\tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

Layer 3: delay time is $\tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

Total delay is the sum of all layers delay time $\tau_{\text{Total}} = 3\tau_{\text{mux}_2} + 2\tau_{\text{modadder}_2}$

To show that hardware works, the signal propagation for binary number

$|X|_7 = |(1110 \ 0110)_2|_7 = |244_{10}|_7 = 6$ is illustrated in fig. 3.7b. Similarly, the reader can try any bit

pattern in fig. 3.7b to check the validity of the design. For example

$|X|_7 = |(1111 \ 1111)_2|_7 = |255_{10}|_7 = 3$ where each multiplexer select line is 3 and the selected output

are shown in parenthesis.

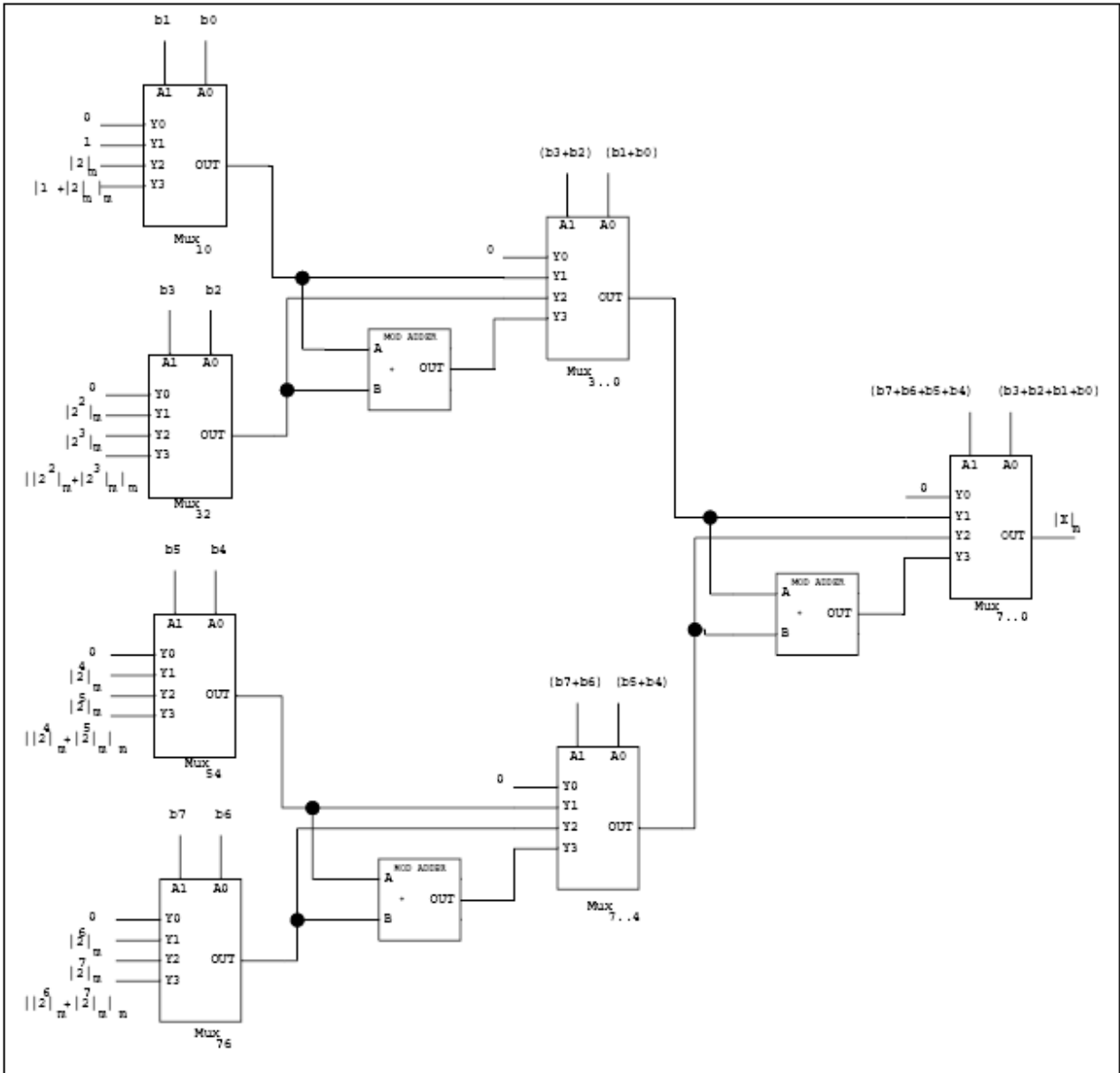


Fig. 3.7a. Eight bits binary to residue number system conversion

$$|X_{7..0}|_m = \left| 2^7 \Big|_m b_7 + 2^6 \Big|_m b_6 + 2^5 \Big|_m b_5 + 2^4 \Big|_m b_4 + 2^3 \Big|_m b_3 + 2^2 \Big|_m b_2 + 2 \Big|_m b_1 + b_0 \right|_m$$

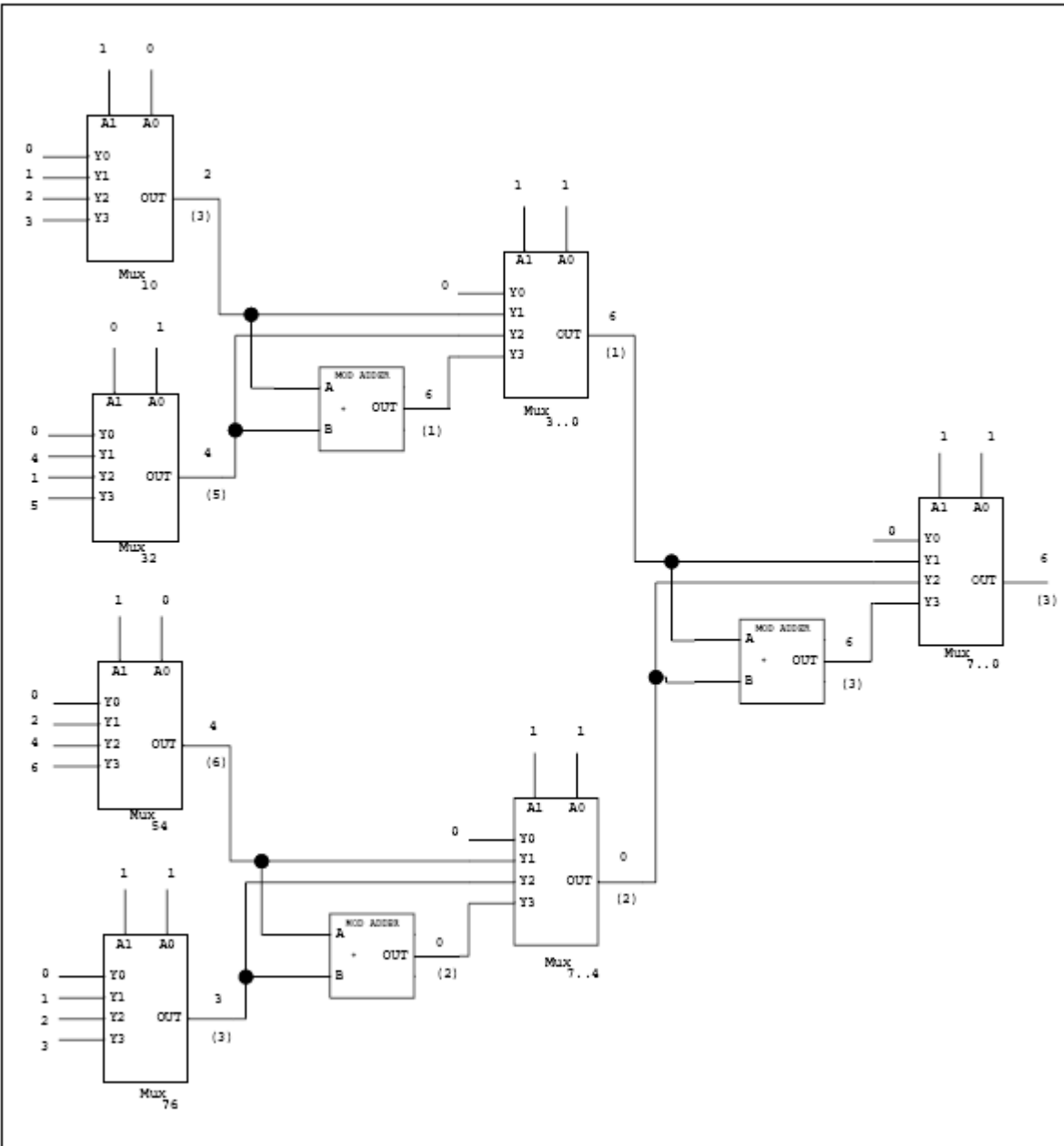


Fig. 3.7b. Example for signal propagation of $|(1110 \ 0110)_2|_7$ and $|(1111 \ 1111)_2|_7$

3.4. Implementation Selection

There are several possible binary to RNS implementations using a combination of 2-bit and 3-bit multiplexers. Fig. 3.8 shows three different implementations (design 1, design 2 and design 3) for 10 bits binary to residue conversion system.

To simplify comparison, the following reasonable assumptions are made

$$\tau_{\text{mux}_2} = \tau_{\text{mux}_3}; \quad \tau_{\text{modadder}_3} = 2\tau_{\text{modadder}_2}$$

Design 1 uses nine 2-bit multiplexers and four 2-input mod adders with

Layer 1: delay time is τ_{mux_2}

Layer 2: delay time is $\tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

Layer 3: delay time is $\tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

Layer 4: delay time is $\tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

Total delay is sum of all layers delay time $\tau_{\text{Total}} = 4\tau_{\text{mux}_2} + 3\tau_{\text{modadder}_2}$

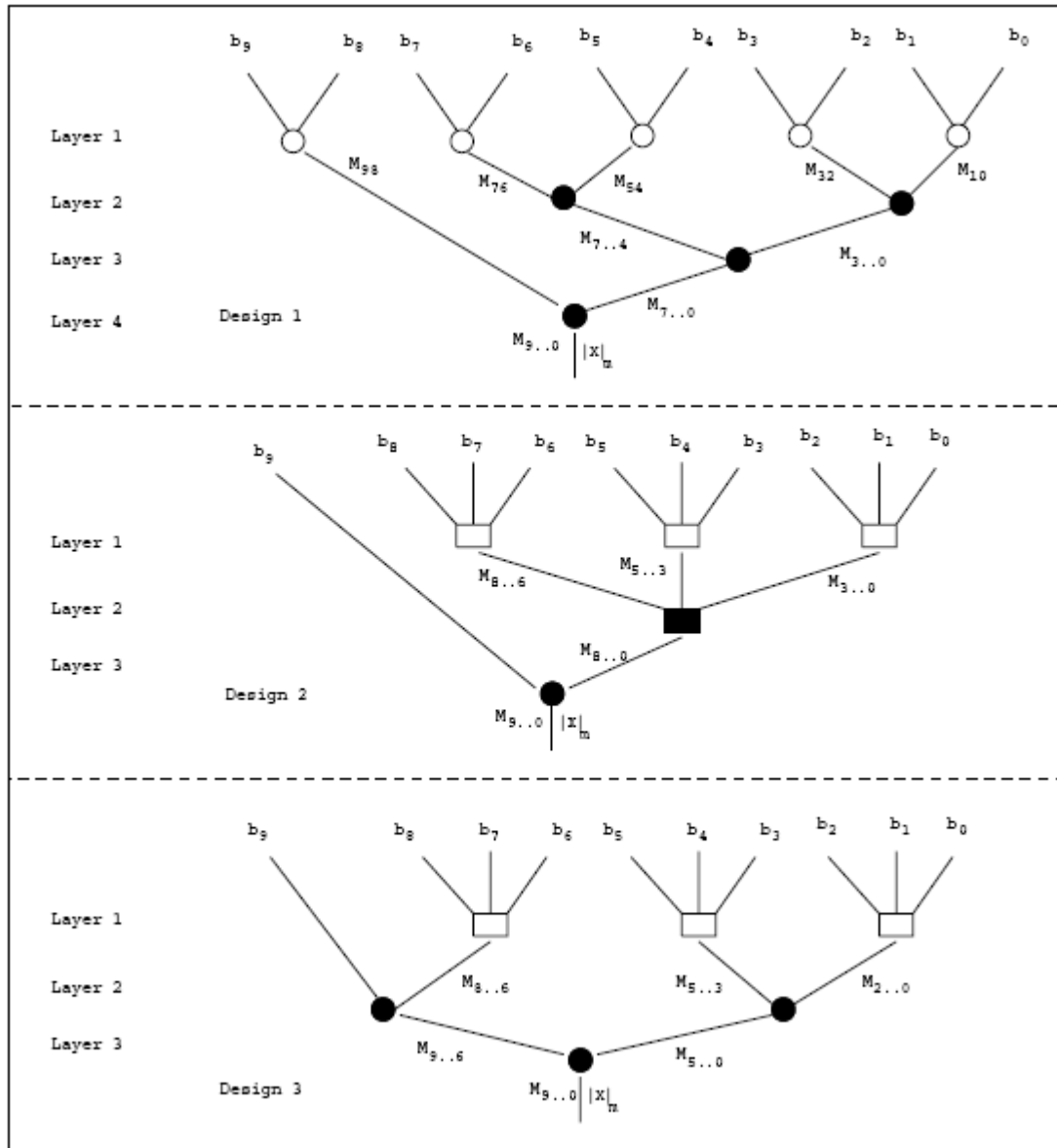


Fig. 3.8. Prefix structure of 10 bits binary to RNS

$$|X_{9..0}|_m = \left| 2^9|_m b_9 + 2^8|_m b_8 + 2^7|_m b_7 + 2^6|_m b_6 + 2^5|_m b_5 + 2^4|_m b_4 + 2^3|_m b_3 + 2^2|_m b_2 + 2|_m b_1 + b_0 \right|_m$$

Design 2 uses four 3-bit multiplexers, one 2-bit multiplexers, four 2-input mod adders and one 3-input adder with

Layer 1: delay time is τ_{mux_3}

Layer 2: delay time is $\tau_{\text{mux}_3} + 2\tau_{\text{modadder}_2}$

Layer 3: delay time is $\tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

$$\begin{aligned}\tau_{\text{Total}} &= \tau_{\text{mux}_2} + 2\tau_{\text{mux}_3} + 3\tau_{\text{modadder}_2} \\ &= 3\tau_{\text{mux}_2} + 3\tau_{\text{modadder}_2}\end{aligned}$$

Design 3 uses three 3-bit multiplexers, three 2-bit multiplexer and three 2-input mod adders with

Layer 1: delay time is τ_{mux_3}

Layer 2 : delay time is $\tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

Layer 3: delay time is $\tau_{\text{mux}_2} + \tau_{\text{modadder}_2}$

$$\begin{aligned}\tau_{\text{Total}} &= 2\tau_{\text{mux}_2} + \tau_{\text{mux}_3} + 2\tau_{\text{modadder}_2} \\ &= 3\tau_{\text{mux}_2} + 2\tau_{\text{modadder}_2}\end{aligned}$$

Table 3.1 shows that design 3 uses less hardware and is faster than other designs.

TABLE 3.1

SHOWS COMPARISON AMONG THE THREE DESIGNS IMPLEMENTATION

Design		Hardware count			
#	Time Delay	Mux ₂	Mux ₃	Mod add ₂	Mod add ₃
1	$4\tau_{\text{mux}_2} + 3\tau_{\text{modadder}_2}$	9	0	4	0
2	$3\tau_{\text{mux}_2} + 3\tau_{\text{modadder}_2}$	1	4	4	1
3	$3\tau_{\text{mux}_2} + 2\tau_{\text{modadder}_2}$	3	3	3	0

3.5. Comparison Selection to Pervious Work

This Novel method has hardware advantages greater than any competitive converters. In 1984, Alia and Martinelli [3] published a binary to RNS conversion design based on power 2 mod m_i . The design uses processing elements (PE) and each PE is associated with two registers. Each of these registers is serially loaded with $|2^i|_m$ and $|2^{i+1}|_m$ respectively. The two outputs are added in a modular adder. Thus, at the first level, $n/2$ PEs are required. The number of stages in this method is $\lceil \log_2 n \rceil$. After successive transformation and addition, the residue result is available. Cappocelli and Giancarlo [4] suggested the use of t PEs where $t = n / \log_2 n$, each PE computing the residue corresponding to k - bit binary word where $k = \log_2 n$, the residue $2^{k\hat{k}} \bmod m_i$ is serially fed to \hat{k} th PE ($\hat{k} = 0, 1, 2, \dots, t-1$). Based on these initial residues, the residues corresponding to the next $(k-1)$ powers are computed by first doubling and then weighting according to the input bits in each PE. The partial residues of k -bit words computed over parallel t PEs are then added to yield the final residue. Mohan [5] has proposed a similar method but with a difference that X is divided into t sections based on the cyclic property of $2^j \bmod m_i$. Using the fact that, 2^j , 2^{j+lo} and 2^{j+2lo} have the same residues due to the periodicity of period lo , lo bits are first added. The width of the result is confined to lo bits by adding the carry bit resulting from previous addition to LSB of the result. The residue results are then determined by using methods given in [3].

Complexity calculation is very important for any design development. Reduction in complexity of design can be done using adjustment in flow of design, which is made before implementation. Table 3.2 shows hardware comparison among our design, Behrooa [7] and Alia [4]. There are various criteria that can be used to measure the hardware complexity number of gates, number of I/O, delay time, fan in / fan out, area / size, power dissipation, and rank of design matrix. McCabe metric and Halstead's software science are two common codes for software complexity

measures. McCabe metric determines code complexity based on number of control paths created by the code as follows

$$v = e - n + 2p$$

Where e is the number of edges in a program flow graph, n the number of nodes, and p the number of connected components. Halstead introduced software science in order to measure properties of the programs. Halstead's program volume is defined to be

$$v = (N_1 + N_2) \log_2(\mu_1 + \mu_2)$$

Where μ_1 is number of distinct operators, μ_2 is number of distinct operands, N_1 is total number of operators and N_2 is total number of operands.

TABLE 3.2

COMPARISON AMONG BEHROOA[7], ALIA[4] AND THE NEW DESIGN

Parameter	Our Methods	Behrooa[7]	Alia and Martinelli [4]
Need for Table Lookup	No	Yes Table size ($2^b \times b$)	No
Hardware components	Multiplexer Modulo adder	ROM Modulo adder	Processing elements (PE) Each PE has two modulo adders, two registers
Delay Time	$n\tau_{\text{mux}} + (n - 1)\tau_{\text{modadder}}$	$n\tau_{\text{ROM}} + (n - 1)\tau_{\text{modadder}}$	$n/2\tau_{\text{PE}}$

3.6. Conclusion

In this chapter, we presented a new novel binary to residue conversion method that eliminates the need for processing elements (PE) as the above competitive converter designs. This new novel design doesn't use table lookup as in Behrooa Parhami [6]. The new method that we present here is based on multiplexers concept which makes it practical and suitable for VLSI implementation.

CHAPTER 4

SIMPLIFIED RNS SCALING ALGORITHM

4.1 Introduction

Recent advances in computer architecture and VLSI technology have brought about a resurgence of interest in RNS based digital systems. RNS system has a very big advantage in the modular arithmetic operations like addition, subtraction and multiplication since this system provides the ability to add, subtract or multiply without the need to wait for the carry propagation as required by the weighted number systems. The non modular operation such as sign detection, division and conversion presents a challenge to researchers. A lot of research has been done to address these issues [9]. In this chapter we present new algorithms for residue number system scaling that utilizes a simplified base extension process [9].

4.2 Division remainder zero

Division remainder zero method is a special simple case of division where the divisor is relatively prime to the modulo set and the dividend is a multiple of the divisor. This operation is accomplished by multiplying the dividend by the multiplicative inverse of the divisor.

$$\left\lfloor \frac{b}{a} \right\rfloor_{m_i} = \left\lfloor b \left\lfloor a^{-1} \right\rfloor_{m_i} \right\rfloor_{m_i} \text{ for all } m_i \text{ in the moduli set } (m_1, m_2, \dots, m_n)$$

If and only if a divides by b without remainder and a and m_i are relatively prime for all i where $\left\lfloor a^{-1} \right\rfloor_{m_i}$ is the multiplicative inverse of a over m_i

4.2 Scaling

Scaling is a restricted division operation where the divisor is one of the modulo or a product of modulo [9]. Several algorithms have been presented for scaling. The common idea of these algorithms is breaking the scaling process into two processes. A division remainder zero operation and a base extension operation [9]. If we assume that a number X is the dividend and

the number Y is the divisor over the modulo set (m_1, m_2, \dots, m_n) . The result of dividing X by Y can be expressed as follows

$$X = \left[\frac{X}{Y} \right] Y + |X|_Y$$

$$X = QY + R \quad (4.1)$$

Where $Q = \left[\frac{X}{Y} \right]$ is the integer quotient is value of X over Y and $R = |X|_Y$ is the least positive integer remainder. The objective of scaling is to find the quotient Q for restricted Y values.

$$Q = \left[\frac{X}{Y} \right] = \frac{X - |X|_Y}{Y} \Rightarrow \left| \frac{X}{Y} \right|_{m_i} = \left| \frac{X - |X|_Y}{Y} \right|_{m_i} \text{ for all } i, \text{ where } (m_i, Y) = 1$$

$$\Rightarrow q_i = \left| \frac{X}{Y} \right|_{m_i} = \left| (X - |X|_Y) \times |Y^{-1}|_{m_i} \right|_{m_i} \text{ where } Q = (q_1, q_2, \dots, q_n) \quad (4.2)$$

Equation (4.2) is a division remainder zero operation and can be used to get the residue digits q_i for all i where $(m_i, Y) = 1$. For the remaining digits where $(m_i, Y) \neq 1$, base extension algorithm is needed to find all residues of the quotient Q .

4.3 General Division

General division operation is the operation where the divisor does not fit the restrictions mentioned in the division remainder zero or scaling operations [9]. General division can be divided into two categories, multiplicative and subtractive. Most of the multiplicative algorithms first compute the reciprocal of the divisor and then multiply the reciprocal by the dividend. The subtractive algorithms employ subtraction of multiples of the divisor until the difference is less than the divisor [9]. The algorithm presented in [9] seems attractive because of its simplicity. It converts the general division operation into iterative scaling operation and it uses a lookup table to identify the candidates for the scaling operations. The disadvantages of this algorithm along with many similar division algorithms are the scaling operation which is slow due to the complicated calculations which are either based on Galois Field or based on basic MRC.

4.4 New Base Extension Algorithm

RNS base extension problem, is the problem of finding the residue digits of one set over another set that is an extension of the original set.

Let $(m_1, m_2, \dots, m_n, m_{n+1}, m_{n+2}, \dots, m_{n+k})$ be relatively prime modulo, the base extension problem is finding the residues $|X|_{m_{n+1}}, |X|_{m_{n+2}}, \dots, |X|_{m_{n+k}}$ given the residue $|X|_{m_1}, |X|_{m_2}, \dots, |X|_{m_n}$; and $0 \leq X < M = \prod_{i=1}^n m_i$.

The algorithm described in [9] is based on MRC conversion, where the algorithm assigns a variable to the residue representation of $|X|_{m_{n+1}}$ and then performs the MRC conversion on the new modulo set which ends up with a linear equation of the MRC coefficient as a function of $|X|_{m_{n+1}}$. This is a lengthy operation. Also, alternative method was presented [9] which are based on CRT. The advantage of CRT is that it is faster than MRC; however it requires large modulo adders because of the need to perform mod M operations.

Lemma 4.1: The general solution of linear Diophantine equation

$$\alpha u + \beta v = \gamma \quad (4.3)$$

Where α , β , and γ are given integers and integer solutions u , v are desired

$$\text{are } u = u^* + \frac{\beta k}{\gcd(\alpha, \beta)} \quad \text{and} \quad v = v^* - \frac{\alpha k}{\gcd(\alpha, \beta)}$$

Where k is any integer and u^* and v^* are any particular solution. Also, this solution will exist if and only if γ is a multiple of $\gcd(\alpha, \beta)$. Equation (4.3) is equivalent to the following RNS equation:

$$m_i q_i + x_i = m_j q_j + x_j \quad i \neq j \quad (4.4)$$

$$\gcd(\alpha, \beta) = \gcd(m_i, m_j) = 1$$

Comparing equations (4.3) and (4.4), equation (4.3) can be written as matrix notation

$$m_1 a_{ij} - m_j a_{ji} = a_{1j} - a_{1i} \quad (4.5)$$

Where $a_{ij} = q_i$, $a_{ji} = q_j$ are the unknowns

and $a_{1j} = x_j$ and $a_{1i} = x_i$ are given integers

The general solution for equation (4.5) is

$$a_{ij} = c \times (a_{1j} - a_{1i}) + m_j \times k \quad (4.6)$$

Where k and c is integers and can be obtained by using Extended Euclidean Algorithm or Fermat's Theorems. Each coefficient a_{ij} represents one element of Matrix A which is needed to solve (4.6).

$$A = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ & a_{22} & a_{23} & \dots & a_{2n} \\ & & a_{33} & \dots & a_{3n} \\ & & & \dots & \dots \\ & & & & a_{nn} \end{bmatrix}$$

$$\text{Where } a_{ij} = \begin{cases} x_j & i = 1, 1 \leq j \leq n \\ c_{ij} \times (a_{(i-1)j} - a_{(i-1)(i-1)}) + m_j \times k & 2 \leq i, j \leq n, j \geq i \\ 0 \rightarrow \text{No need to find them} & \text{Other} \end{cases} \quad (4.7)$$

The smallest positive integer solution of each of the a_{ij} can be obtained by iterations of equation (4.7) and the solution of Diophantine diagonal coefficients “ a_{ii} ” is equivalent to mixed radix digits in MRC conversion

$$X = a_{11} + a_{22} \times m_1 + a_{33} \times m_1 \times m_2 + \dots + a_{nn} \prod_{i=1}^{n-1} m_i \quad (4.8)$$

Example:

Given the residue representation $X = (0, 1, 1)$ with modulo set $(m_1, m_2, m_3) = (2, 3, 5)$, extend the base to $m_4=7$, i.e. find $|X|_7$

Solution:

Convert X to its decimal value using an RNS to decimal conversion algorithm

$$X = (0, 1, 1) = 16$$

$$|X|_7 = |16|_7 = 2$$

The new representation of X over the new modulo set $(2, 3, 5, 7)$ is $(0, 1, 1, 2)$

Example:

Given the residue representation $x = (0, 1, 1)$ for the base with modulo $(m_1, m_2, m_3) = (2, 3, 5)$.

Extend the base to $m_4=7, m_5=11$, i.e. find $|X|_7$ and find $|X|_{11}$

Solution:

Convert X to its decimal value using an RNS to decimal conversion algorithm

$$X = (0, 1, 1) = 16$$

$$|X|_7 = |16|_7 = 2$$

$$|X|_{11} = |16|_{11} = 5$$

The new representation of x over the new modulo set $(2, 3, 5, 7, 11)$ is $(0, 1, 1, 2, 5)$

4.5 New Scaling Algorithm

The following algorithms can be used to find coefficient “c” in equation (4.7)

Fermat’s implementation for finding coefficient “c” as follows:

```

%-----%
% finding coefficient c using Fermat method
%-----%
v = mod(mod(power(mj,mk-2),mk),mk);
v1 = mk-v;
if v <= v1
    c = v
else
    c = -1*(mk-v)
end

```

Fig. 4.1. Fermat's implementation for finding coefficient "c"

Extended Euclid Algorithm Implementation

```

%-----%
% Finding coefficient "c" using Extended Euclid Algorithm
%-----%
u1 = 1; u2 = mj; v1 = 0; v2 = mk;
while v2
    q = int(u2/v2);
    t1 = u1 - v1*q; t2 = u2 - v2*q;
    u1 = v1;    u2 = v2;
    v1 = t1;    v2 = t2;
end
c = u1

```

Fig. 4.2. Extended Euclid algorithm implementation

```

%-----%
% Finding the smallest positive integer "ajj" solutions for
% Diophantine equations
%-----%
a(j,k) = c*(a(j-1,k)-a(j-1,j-1));
n = 1;
if a(j,k) < 0          % case 1  ex -6 + 11n
    while (a(j,k) + n*mk) < 0
        n = n + 1;
    end
    a(j,k) = a(j,k) + n*mk;
elseif a(j,k) > mk    % case 2  ex 30 + 11n
    while (a(j,k) - n*mk) >= 0
        n = n + 1;
    end
    a(j,k) = a(j,k) - (n-1)*mk;
else
    a(j,k) = a(j,k);    % case 3  a(j,k) < mk
end

```

Fig. 4.3. Implementation of finding coefficient (a_{ij})

Example:

Find RNS representation of number 300 using modulo set (5, 7, 11) and also find mixed radix digits using Diophantine method

Solution:

$$(300)_{10} \xrightarrow{RNS} (0,6,3)_{5,7,11}$$

From equation (4.5)

$$a_{11} = x_1 = 0$$

$$a_{12} = x_2 = 6$$

$$a_{13} = x_3 = 3$$

$$a_{22} = c_{22} * [a_{12} - a_{11}] + m_2 * k$$

and from Extended Euclidean Algorithm or Fermat's Theorems $c_{22} = 3$

$$a_{22} = 3 * [6 - 0] + 7k = 18 + 7k$$

at $k = -2$, $a_{22} = 4$ is the smallest positive integer solution

$$a_{23} = c_{23} * [a_{13} - a_{11}] + m_3 * k$$

$$c_{23} = -2$$

$$a_{23} = -2 * [3 - 0] + 11k = -6 + 11k$$

at $k = 1$, $a_{23} = -6 + 11 = 5$ is the smallest positive integer solution

$$a_{33} = c_{33} * [a_{23} - a_{22}] + m_3 * k$$

$$c_{33} = -3$$

$$a_{33} = -3 * [5 - 4] + 11k = -3 + 11k$$

at $k = 1$, $a_{33} = 8$ is the smallest positive integer solution

$$A = \begin{bmatrix} 0 & 6 & 3 \\ & 4 & 5 \\ & & 8 \end{bmatrix}$$

From equation (4.8)

$$X = a_{11} + a_{22} \times m_1 + a_{33} \times m_1 \times m_2$$

$$X = 0 + 4 \times 5 + 8 \times 5 \times 7 = 300$$

Diophantine equation can also help to solve RNS scaling by power of two. The new method is based on division remainder zero theorem [9] and be used to any set of prime modulo set ($m_1, m_2, m_3, \dots, m_n$).

Theorem 4.1 (Division Remainder Zero)

$$\left| \frac{X}{S} \right|_M = |XS^{-1}|_M = \left(|X_1 S^{-1}|_{m_1}, |X_2 S^{-1}|_{m_2}, \dots, |X_n S^{-1}|_{m_n} \right)$$

Only if s divide by X without a remainder and $\gcd(s, m_i) = 1$

Theorem 4.2:

For any set of prime modulo set ($m_1, m_2, m_3, \dots, m_n$).

Case A: X is an even number “scaling without a reminder”

$$\left| \frac{X}{2} \right|_M = (y_1, y_2, \dots, y_n, R)$$

Where $y_i = \left| x_i \left| 2^{m_i - 2} \right|_{m_i} \right|_{m_i}$ for all odd modulo and

R is the scaling result for $m = 2$ and it is found by following manner

- Set $R = 0$
- Find mixed radix digits ($a_{11}, a_{22}, \dots, a_{nn}, D$) for $(y_1, y_2, \dots, y_n, 0)$ by using Diophantine RNS to binary conversion method

➤ If $D = 0$ then $\left\lfloor \frac{X}{2} \right\rfloor_M \Leftrightarrow (y_1, y_2, \dots, y_n, 0)$ and

➤ If $D = 1$ then $\left\lfloor \frac{X}{2} \right\rfloor_M \Leftrightarrow (y_1, y_2, \dots, y_n, 1)$

Case B: X is odd number “rounding to nearest integer”

Replace X with $(X+1)$

$$\left\lfloor \frac{X+1}{2} \right\rfloor_M \Leftrightarrow (y_1, y_2, \dots, y_n, R)$$

Where $y_i = \left\lfloor (x_i + 1) \left\lfloor 2^{m_i-2} \right\rfloor_{m_i} \right\rfloor_{m_i}$ for all odd modulo and

R is the scaling result for $m = 2$ and it always equals to one.

Example:

For modulo set $(2, 3, 5)$ determine the residue representation for $8/2$, $6/2$ and $5/2$

Solution:

Let $m_1 = 3$, $m_2 = 5$, $m_3 = 2$

$$\text{a) } (8)_{10} \xrightarrow{RNS} (2, 3, 0)_{3, 5, 2}$$

$$y_1 = \left\lfloor 2 \left\lfloor 2^{3-2} \right\rfloor_3 \right\rfloor_3 = 1$$

$$y_2 = \left\lfloor 3 \left\lfloor 2^{5-2} \right\rfloor_5 \right\rfloor_5 = 4$$

Set $R = 0$ for $m_3 = 2$

$$(y_1, y_2, R) = (1, 4, 0)$$

Mixed radix digits “ a_{33} ” = 0 $\rightarrow R = 0$

$$\left(\frac{8}{2} \right)_{10} \xrightarrow{RNS} (1, 4, 0)_{3, 5, 2} \xrightarrow{RNS^{-1}} 4$$

$$\text{b) } (6)_{10} \xrightarrow{RNS} (0,1,0)_{3,5,2}$$

$$y_1 = \left| 0 \left| 2^{3-2} \right|_{3,3} \right| = 0$$

$$y_2 = \left| 1 \left| 2^{5-2} \right|_{5,5} \right| = 3$$

Set $R = 0$ for $m_3 = 2$

$$(y_1, y_2, R) = (0, 3, 0)$$

Mixed radix digits “ a_{33} ” = 1 $\rightarrow R = 1$

$$\left(\frac{6}{2} \right)_{10} \xrightarrow{RNS} (0,3,1)_{3,5,2} \xrightarrow{RNS^{-1}} 3$$

$$\text{c) } (5)_{10} \xrightarrow{RNS} (2,0,3)_{3,5,2}$$

$$y_1 = \left| (2+1) \left| 2^{3-2} \right|_{3,3} \right| = \left| (2+1) * 2 \right|_{3,3} = 0$$

$$y_2 = \left| (0+1) \left| 2^{5-2} \right|_{5,5} \right| = \left| (0+1) * 3 \right|_{5,5} = 3$$

And $R = 1$ for $m_3 = 2$ as shown in case B

$$(y_1, y_2, R) = (0, 3, 1)$$

$$\left(\frac{5}{2} \right)_{10} \xrightarrow{RNS} (0,3,1)_{3,5,2} \xrightarrow{RNS^{-1}} 3 \quad \text{“Rounding to nearest integer”}$$

4.6 Conclusion

Diophantine RNS parallel algorithm provides an alternative method of finding mixed radix digits with a high degree of parallelism. The algorithm has advantages over MRC method and CRT methods since it avoids the use of modulo computations and use of multiplicative inverse.

CHAPTER 5

VLSI IMPLEMENTATION OF RESIDUE ADDER AND SUBTRACTOR

5.1 Introduction

In the last decade RNS arithmetic has become an attractive design option [1-3] for real time application fields such as signal processing, image processing and computer graphics. The merits of using RNS arithmetic lies in its capability of performing addition, subtraction and multiplication without the generation of carry propagation. Also, RNS arithmetic has the capability of being designed and fabricated using VLSI techniques. These characteristics of RNS makes it most suitable for digital signal processor hardware.

In RNS the large numbers are represented by an n -tuple of smaller numbers that are independent of each other, where n is the number of modulo in the modulo set. Hence, the number operation can be done on these smaller numbers rather than the original number. Furthermore, because the numbers in the n -tuple are independent of each other, the operation can be done in parallel. The RNS is defined by a set of modulo (m_1, m_2, \dots, m_n) that are pairwise relatively prime positive integers. It can be shown that there is a unique representation for each number in the range of $0 \leq X < M$ Where $M = \prod_{i=1}^n m_i$ and is called dynamic range [9]. Each integer X can be represented by an n -tuple of residues $X = (x_1, x_2, \dots, x_n)$,

where $x_i = X \bmod m_i$ or it can be written as $x_i = |X|_{m_i}$. The RNS represents any number within the range of $[0, M)$ for unsigned numbers or $\left[-\frac{M}{2}, \frac{M}{2}-1\right]$ for signed numbers. In RNS, the

binary operations $\{ +, -, * \}$ are defined as follow If $Z = A \square B$ then $(z_1, z_2, \dots, z_n) = (a_1, a_2, \dots,$

$a_n) \square (b_1, b_2, \dots, b_n)$ where $z_i = (a_i \square b_i) \bmod m_i$. Each residue digit can be computed independently of the others allowing fast data processing in n parallel independent channels.

VLSI implementations for adder and subtractor have been realized by many researchers. Bayoumi [65] used three approaches (look up table approach, binary adder approach and hybrid implementation approach). Banerji [67] and Piestrack [68] have their contributions as well for implementation of binary adders using VLSI techniques. In this chapter, we have implemented an RNS adder and subtractor. We have used Cadence (Virtuoso and Encounter) for the layout and XILINX for the simulation results of the design.

5.2 Residue Adder and Subtractor

If we have two integers A and B of modulo m , then their addition and subtraction is expressed as sum and subtraction of $A \pm B \bmod m$. The operation addition and subtraction can be described as below in equation 1 and 2 respectively.

$$|A + B|_m = \begin{cases} A + B & \text{if } A + B < m \\ A + B - m & \text{if } A + B \geq m \end{cases} \quad (5.1)$$

$$|A - B|_m = \begin{cases} A - B & \text{if } A - B \geq 0 \\ A - B + m & \text{if } A - B < 0 \end{cases} \quad (5.2)$$

Fig. 5.1 shows the implementation of RNS adder /subtractor based on above equations.

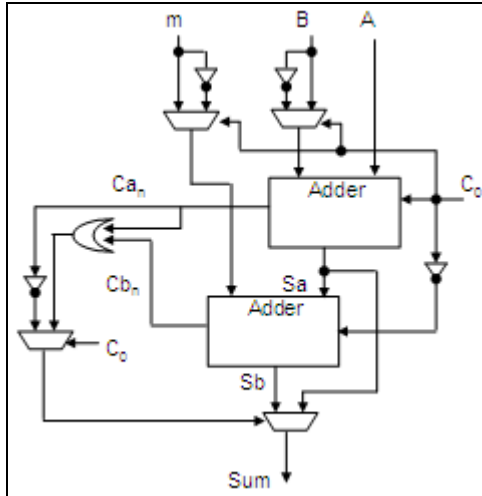


Fig. 5.1. RNS adder and subtractor

The operation addition or subtraction is decided by the select line C_0 . When C_0 is 0, the RNS operation is addition. When C_0 is 1, the RNS operation is subtraction. For n -bit RNS adder the select line C_0 is 0 and it will simply perform as an adder. Inputs to adder are A , B and m . Fig. 5.1 shows all the components that are used in RNS adder and subtractor. There are two full adders, four 2×1 multiplexers and one OR gate are used. As already mentioned for Adder C_0 is zero, therefore inputs A , B and C_0 are fed to the first full adder, which in turn will yield carryout Ca_n and sum Sa . In next step $m, \overline{C_0} = 1$ and sum of previous full adder Sa goes to the next full adder in the hierarchy which in turn will give us sum Sb, Cb_n . Then Ca_n and Cb_n are added and sent to the third 2×1 multiplexer along with signal Ca_n . The output of the third 2×1 multiplexer is used as a control signal to the final 2×1 multiplexer which generates the final output Sum based on equation (5.1). For subtractor the scenario is same and equation (5.2) is implemented, but the control signal C_0 in this case is 1.

5.3 VLSI Implementation

In this section, we implemented the RNS adder and subtractor using VLSI techniques such as XILINX and Cadence tools (virtuoso, Encounter). For simulation we have used XILINX and for layout we used encounter and virtuoso.

a) XILINX is a platform where we can generate the schematic using the verilog code of the design. At the same time, we can test if the desired schematic or design is correct by looking at the behavioral simulation. Once we have created a verilog code for the adder and subtractor, we used the Xilinx tools to synthesize the schematic. Eventually for the verification of design, we built a testbench and run the simulation to see the correct desired output. Fig. 5.2 shows the result of RNS adder and subtractor.

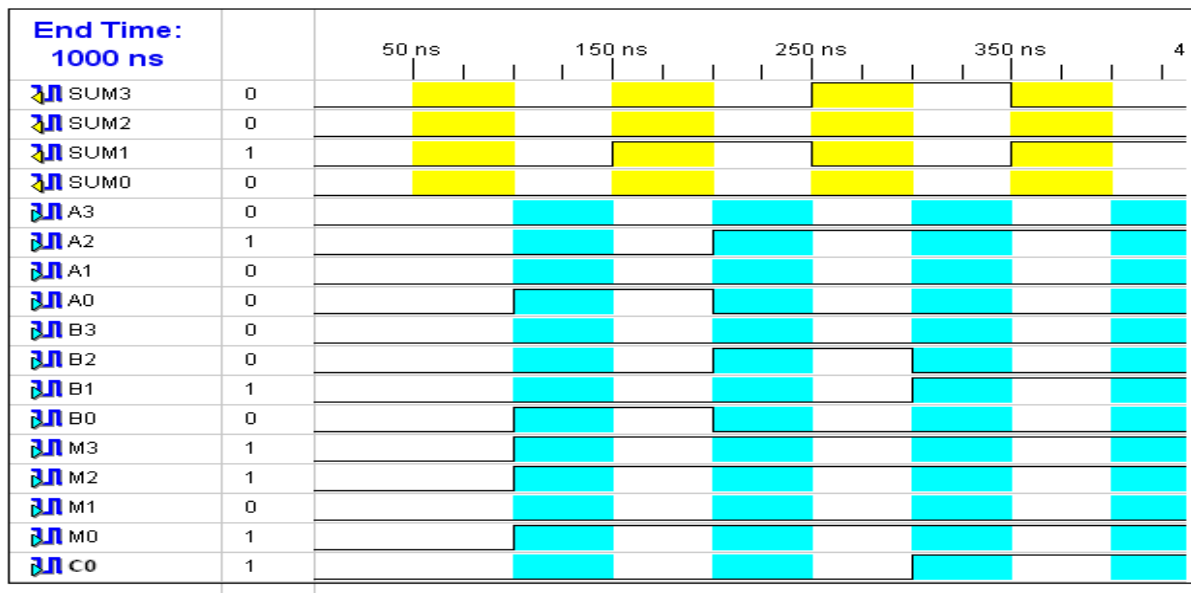


Fig. 5.2. Waveform of RNS adder and subtractor

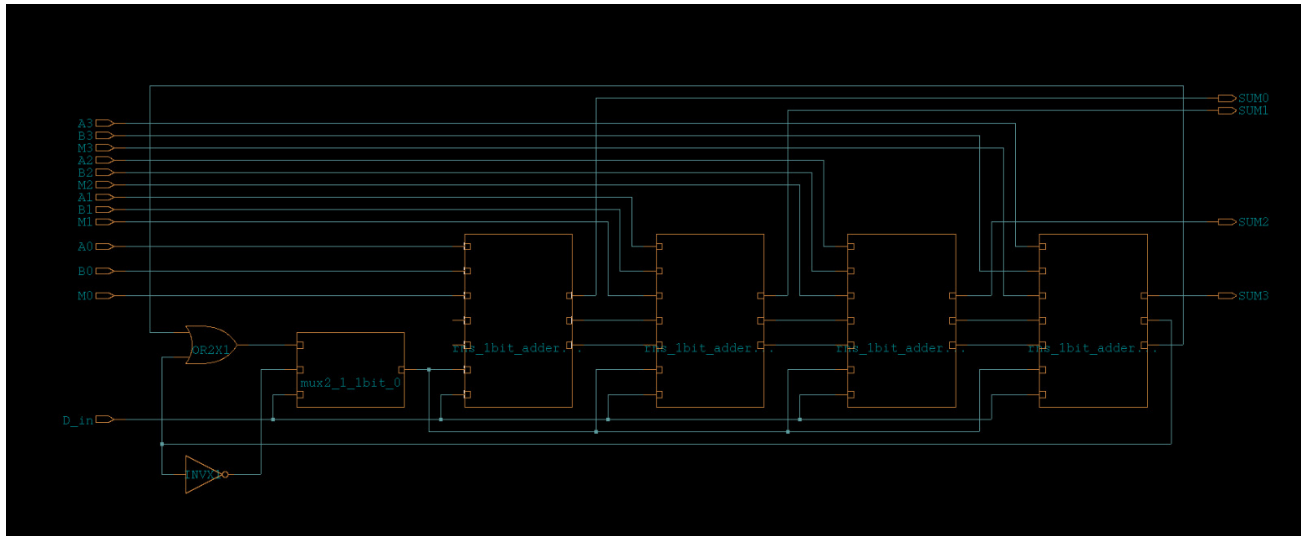


Fig. 5.3. RNS adder and subtractor layout

b) Encounter is a technique where we can generate a layout of our design. Ambient Buildgates is used to generate the netlist then the netlist is uploaded to the encounter. Further, by assigning floor plans, appropriate layers and nano routing we can get the design in .gds and .def format. We can also check for the design if it is flawless or not in terms of connectivity, density etc. Fig. 5.4 shows the encounter part of our design.

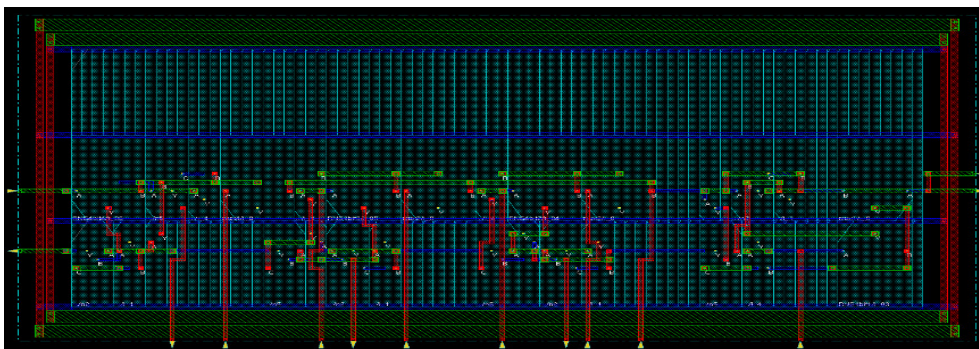


Fig. 5.4. Encounter part of RNS adder and subtractor

TABLE 5.1
TOTAL RNS ADDER AND SUBTRACTOR DELAY TIME (PS)

Delay time from input pin to output pin (ps)					
A0	->	SUM0	4.86	B0	-> SUM0 5.25
A1	->	SUM1	4.19	B1	-> SUM1 4.59
A2	->	SUM2	3.53	B2	-> SUM2 3.92
A3	->	SUM3	2.54	B3	-> SUM3 2.92
m0	->	SUM0	4.23	D_in	-> SUM0 4.32
m1	->	SUM1	3.52	D_in	-> SUM1 2.59
m2	->	SUM2	2.85	D_in	-> SUM2 1.83
m3	->	SUM3	1.85	D_in	-> SUM3 1.54

TABLE 5.2
RNS ADDER AND SUBTRACTOR DESIGN PARAMETER

Parameter	Value
Number of Gates	488
Number of I/O	17
Number of Nets	759
Area μm^2	28008
Max Delay Time(PS)	4.86

c) Virtuoso is a tool through which we can send our layout for the chip fabrication. We can pursue Virtuoso in two ways, one is to build the design from basic building blocks, such as from transistor level (pmos, nmos etc.) and the second way is to import the design from encounter with the help of certain technology libraries. In our work, we have used the later method where we have imported the design from encounter and the same was padframed in order to send it to MOSIS for chip fabrication. The complete Layout along with the connections is shown in fig. 5.4 below.

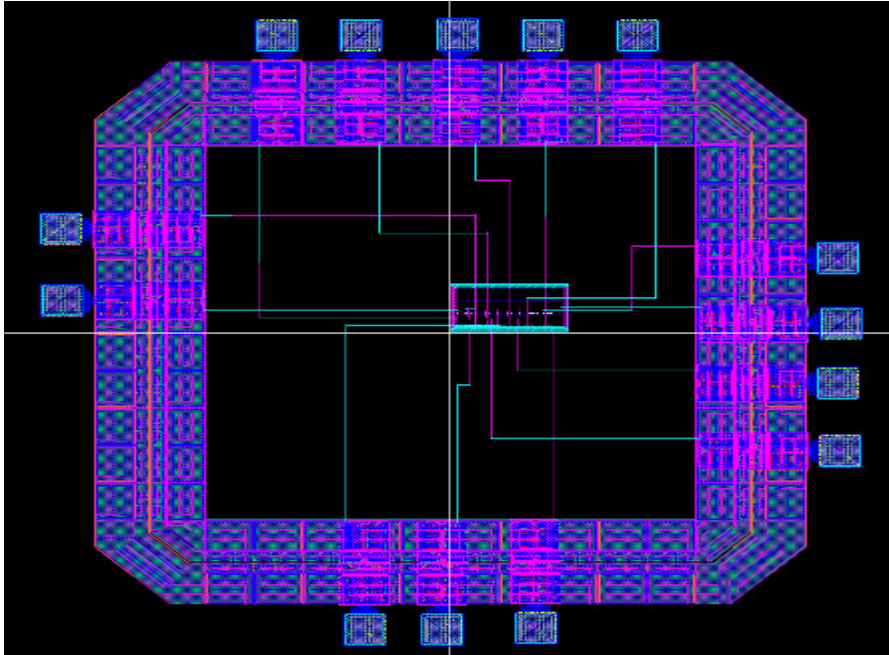


Fig. 5.5. Virtuoso part of RNS adder and subtractor

5.4 Conclusion

We have discussed the VLSI approach for the realization of RNS adder and subtractor. XILINX is used to get the behavioral simulation of the design with the help of which we were able to verify that our design is performing as desired. Cadence encounter is used to build the layout of the design. Design layout along with the technology libraries and layers is exported to Virtuoso. Furthermore, padframing was performed in order to send the design to MOSIS for fabrication.

CHAPTER 6

NOVEL QUANTUM BOOLEAN CIRCUITS CONSTRUCTION BY USING XOR-AND REDUCTION METHOD

6.1 Introduction

One of the new fields of nanotechnology is quantum-dot cellular automata that provides an alternative design to CMOS architectures. Recent research studies [76-82] show that the advantages of using QCA technology are smaller circuit size, faster switching speed, and less power consumption.

During the past decade, quantum-dot cellular automata has demonstrated the ability to implement both combinational and sequential logic devices. Unlike conventional Boolean AND-OR-NOT based circuits. The fundamental logical device in QCA Boolean networks is majority gate which implements the Boolean function:

$$M(A,B,C) = AB+AC+BC \quad (6.1)$$

With combining these QCA gates with NOT gates any combinational or sequential logical device can be constructed from QCA cells. The process of QCA Boolean logic is more sophisticated than Boolean logic. The traditional Boolean logic reductions methods such as Karnaugh maps produce simplified Boolean expressions. However, converting these forms to QCA Boolean is not simple process due to complexity of multilevel majority gates. R. Zhang [19] proposed thirteen standard functions to present all three variables Boolean functions that can be used to produce simplified majority logic. Chin-Yung [13] used Tabulation method to simplify Boolean logic functions and to produce a simplified QCA logic. In this chapter, we present a novel methodology for multilevel majority logic synthesis, our methodology takes as

its input a Boolean circuit, generates simplified XOR-AND equivalent circuit and output an equivalent majority gate circuits.

6.2 QCA Background Material

QCA Cell: A quantum cell can be viewed as a set of four charge containers or dots positioned at the corners of a square as shown in fig. 6.1a. Each QCA cell contains two mobile electrons that can move to any quantum dot through electron tunneling. Thus, there are two equivalent logic polarization $P = +1$ (Logic 1) and $P = -1$ (logic 0).

QCA Majority Gate: The basic QCA logic element is a majority gate as shown in fig. 6.1b. It produces an output of one if the majority of inputs one. The classical AND and OR gates can be realized with majority gate by fixing one of three inputs as 0 or 1 respectively, as follows:

$$M(A,B,0) = AB \quad (6.2a)$$

$$M(A,B,1) = A+B \quad (6.2b)$$

QCA Inverter: QCA cells layout of an inverter is shown in fig. 6.1c. The polarization of the output QCA cell “out” is opposite of input QCA cell “in”.

QCA Wire: There are two types of QCA wires normal (also called 90°) and diagonal (also called 45°). Fig. 6.1d shows the two QCA wire types with logic one polarized.

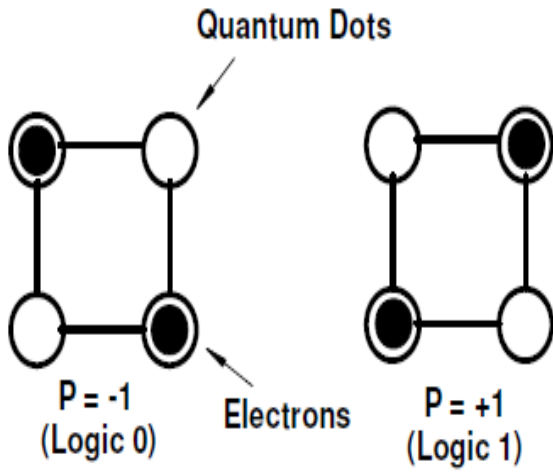


Fig. 6.1a. QCA cells and binary encoding

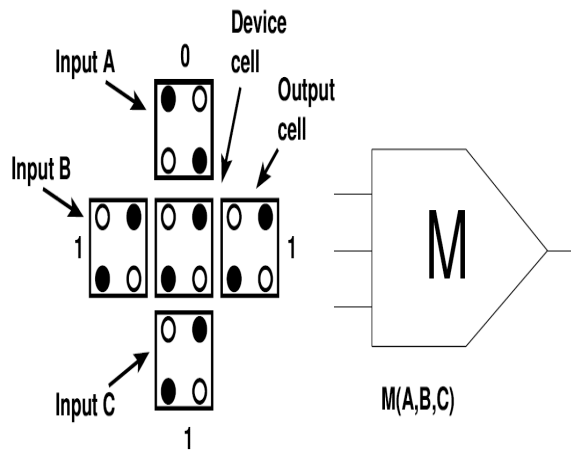


Fig. 6.1b. QCA majority gate

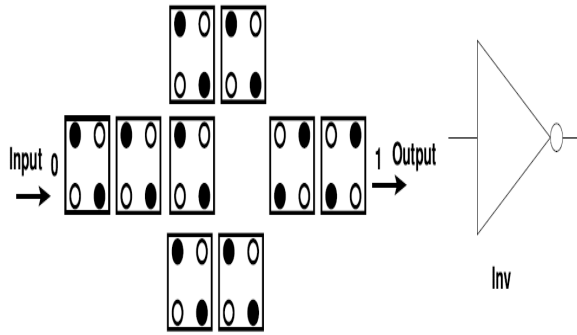


Fig. 6.1c. QCA inverter gate

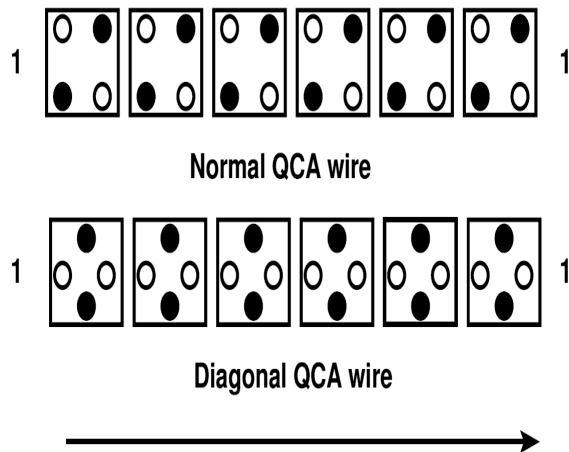


Fig. 6.1d. QCA wire types

QCA Clocking: QCA cells use four phase scheme namely clock 1, clock 2, clock 3 and clock 4 as shown in fig. 6.2a. Every clock is 90° out of phase from its pervious clock and each clock has four states namely switch, hold, release, and relax [40]. In switch state, QCA cells start polarized. In hold state, cells retain thier polarization. Additionally, during release and relax states, QCA cells are unpolarized as shown in fig. 6.2b.

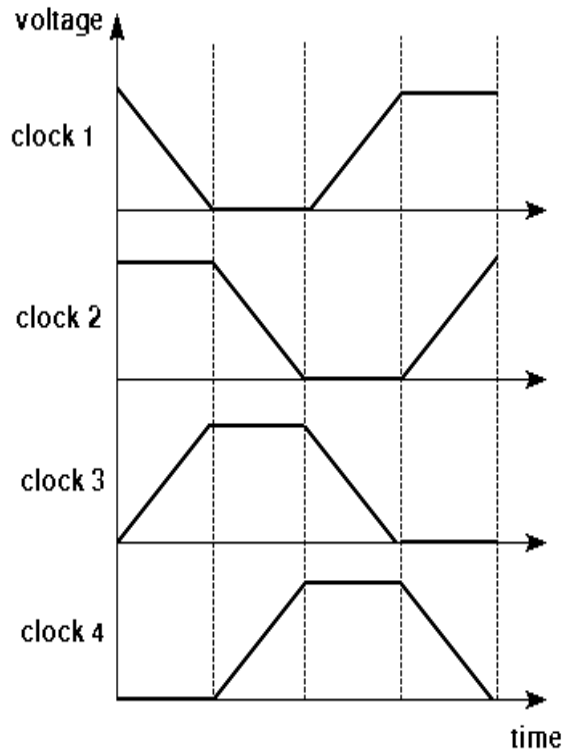


Fig 6.2a. QCA clock phases; each clock lagging its prior by 90°

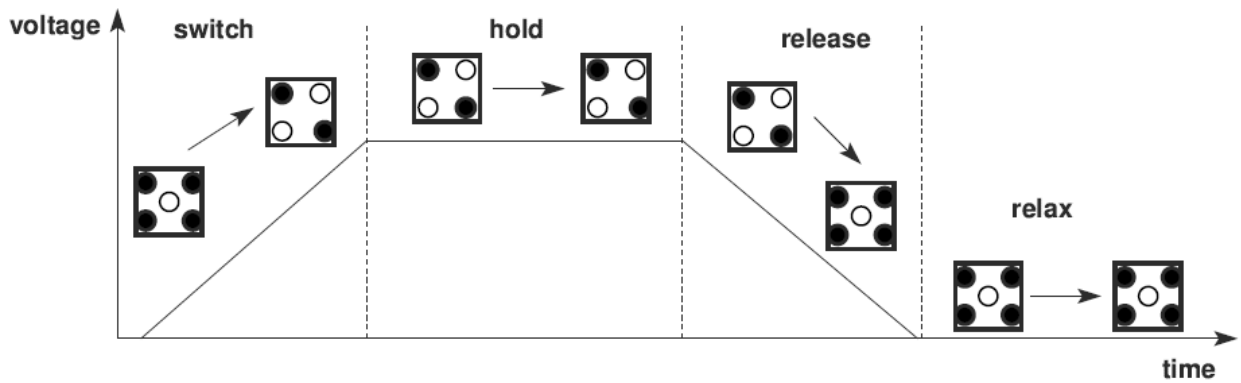


Fig. 6.2b. Four QCA interdot barrier states

6.3 Novel QCA Extraction

XOR algebra can be used very effectively to yield gate-minimum results not possible by conventional mapping methods. Our novel QCA extraction procedure takes as its input a Boolean network, generates simplified XOR-AND equivalent network and output an equivalent majority gate network as shown in fig 6.3.

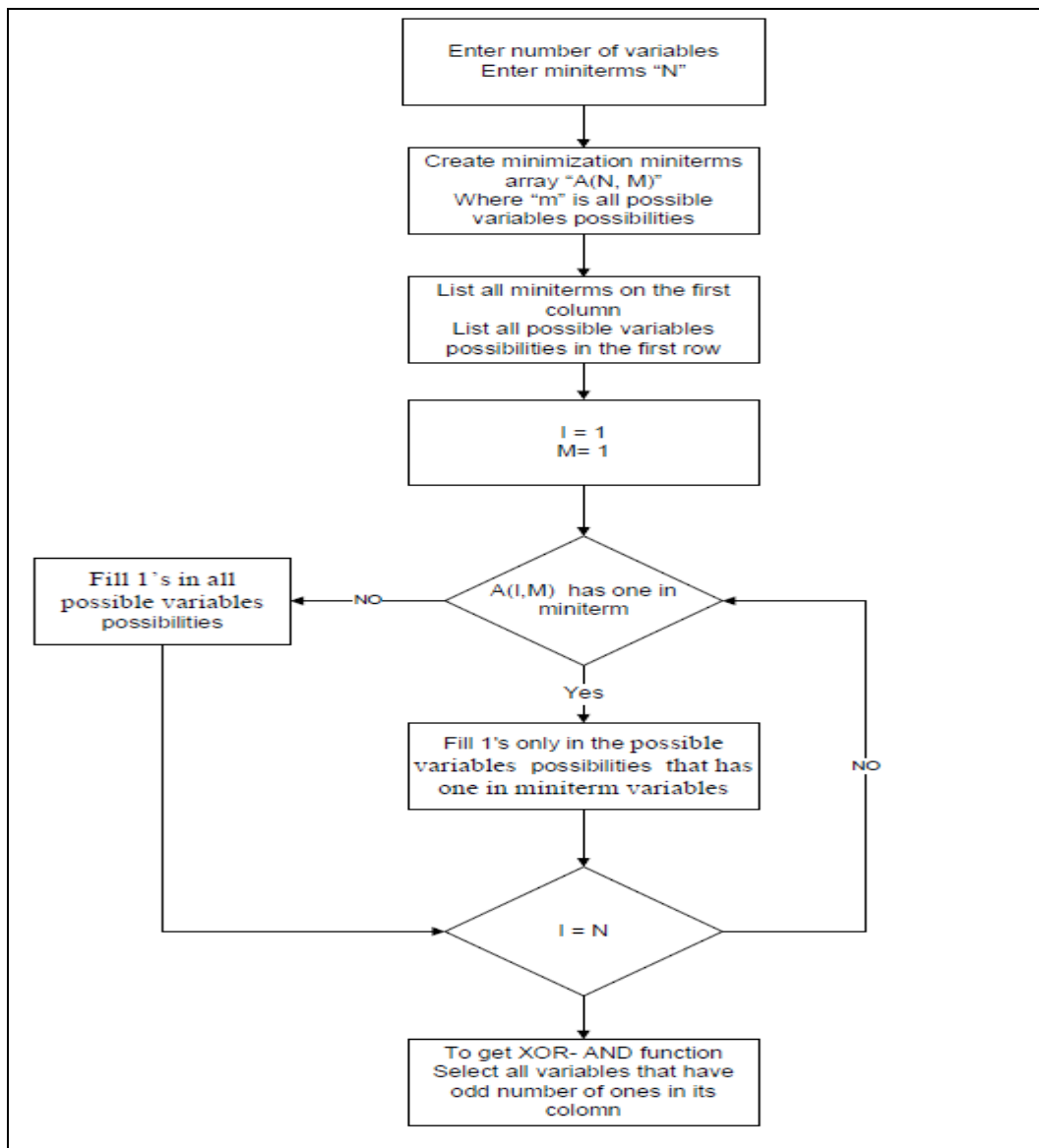


Fig 6.3. XOR-AND function extraction methodology

We will use the following two Boolean examples to illustrate the QCA extraction methodology.

Example: Generate the equivalent QCA circuit for $f_1(a,b,c,d) = \sum(2,3,5,7,8,12,13,14)$

Solution:

Step1: Draw up the minimization chart and list all miniterms on the first column [10]. Refer to table 6.1 for detail of construction.

TABLE 6.1

THE CHART FOR DERIVING XOR EQUIVALENT FUNCTION “F₁”

f_1	1	a b c d	ab ac ad bc bd cd	abc abd acd bcd	abcd
0010		1	1 1 1	1 1 1	1
0011				1 1	1
0101				1 1	1
0111				1	1
1000		1	1 1 1	1 1 1	1
1100			1	1 1	1
1101				1	1
1110				1	1
	x	√ x √ x	x x √ √ √ x	x x √ x	x

Step2: List all possible variables possibilities in the first row. Start with one, all possible pairs of variables, then all triples of variables and so on up to columns for all the variables possibility.

Step3: Filling 1's in all possible variables columns that have unprimed variables in miniterms as shown in table 6.1.

Step4: To get the function in final XOR-AND, cross out all columns that have even number of 1s in them. The XOR-AND function for this example is

$$f_1 = a \oplus c \oplus ad \oplus bc \oplus bd \oplus acd$$

Step5: Utilizing ($\bar{x} = 1 \oplus x$) XOR property, the above function can be simplified to

$$f_1 = a \oplus \bar{b}c \oplus bd \oplus a\bar{c}d$$

Step6: Construct a majority gate tree as shown in fig. 6.4 and then replace each node with an equivalent majority XOR and AND gates as shown in fig. 6.5.

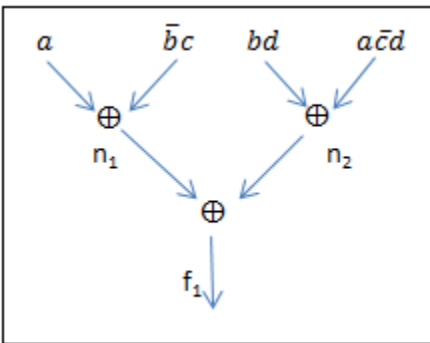


Fig. 6.4. Majority tree for function f_1

Lemma 6.1: If x and y are two binary inputs, then $xy = M(x, y, 0)$

Proof:

By equation (6.1)

$$M(x, y, 0) = xy + 0 + 0 = xy$$

Lemma 6.2: If x and y are two binary inputs, then $x \oplus y = M(M(\bar{x}, y, 0), M(x, \bar{y}, 0), 1)$

Proof:

$$M(M(\bar{x}, y, 0), M(x, \bar{y}, 0), 1) = M((\bar{x}y + 0 + 0), (x\bar{y} + 0 + 0), 1) =$$

$$(\bar{x}y)(x\bar{y}) + \bar{x}y + x\bar{y} =$$

$$\bar{x}y + x\bar{y} = x \oplus y$$

Lemma 6.3: If x and y are two binary inputs, then $\overline{M(x, y, 0)} = M(\bar{x}, \bar{y}, 1)$

Proof:

$$M(x, y, 0) = xy$$

$$\overline{M(x, y, 0)} = \overline{xy} = \bar{x} + \bar{y}$$

By equation (6.2b)

$$\bar{x} + \bar{y} = M(\bar{x}, \bar{y}, 1)$$

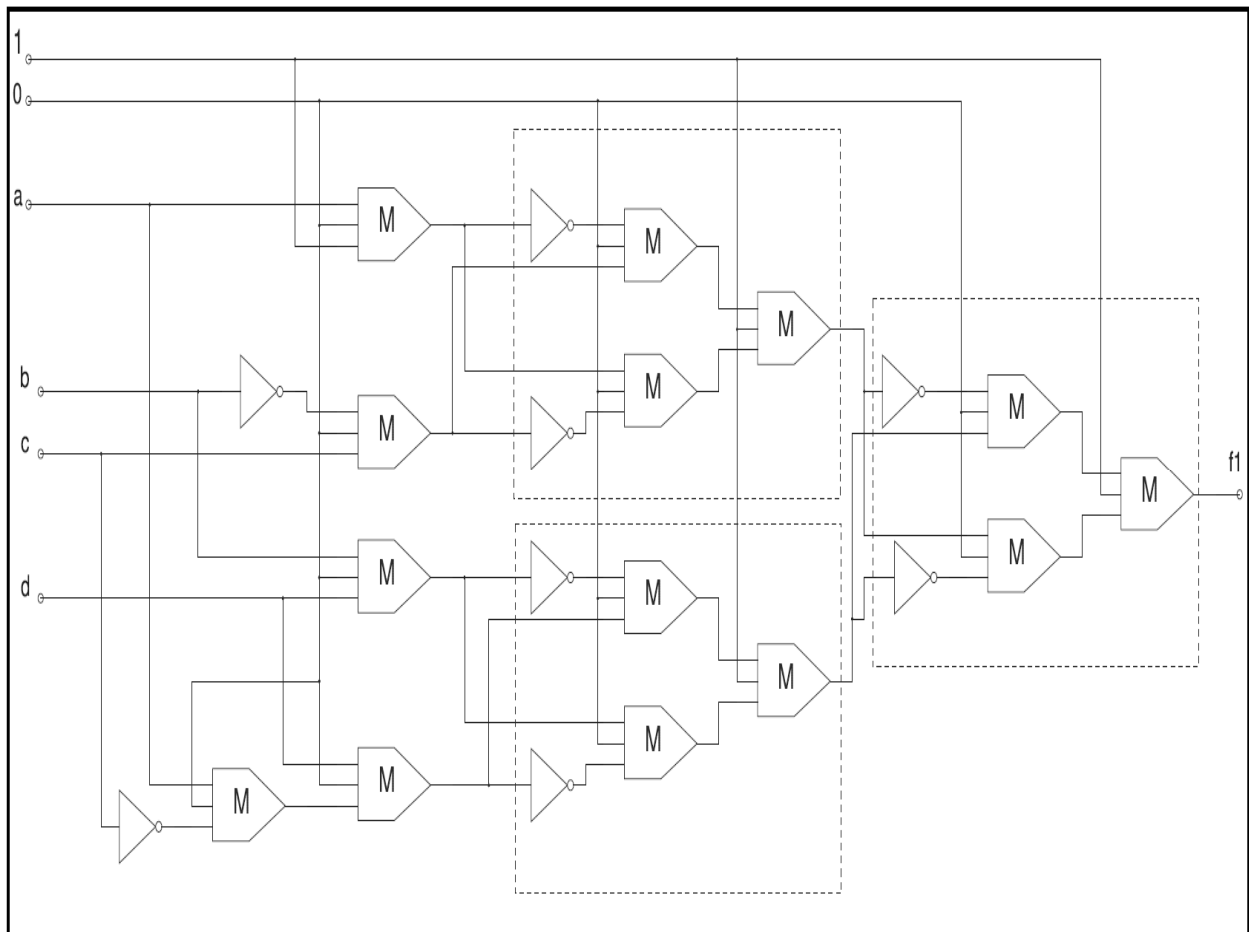


Fig. 6.5. Majority gates schematic for function f1

Step7: Majority tree can also be used to construct QCA expressions for any node. QCA expressions for n_1 and n_2 are as follows

By Lemma 6.1

$$a = M(a,1,0)$$

$$\bar{bc} = M(\bar{b},c,0)$$

$$bd = M(b,d,0)$$

$$acd = M(M(a,c,0),d,0)$$

By Lemma 6.2 and Lemma 6.3

$$n_1 = a \oplus \bar{bc}$$

$$n_1 = M(M(\bar{a},\bar{bc},0),M(a,\bar{\bar{bc}},0),1)$$

$$n_1 = M(M(\overline{M(a,1,0)},M(\bar{b},c,0),0), \\ M(M(a,1,0),M(\bar{b},c,0),0),1)$$

$$n_1 = M(M(M(\bar{a},0,1),M(\bar{b},c,0),0), \\ M(M(a,1,0),M(\bar{b},c,1),0),1)$$

And

$$n_2 = bd \oplus acd$$

$$n_2 = M(M(\bar{bd},acd,0),M(bd,\bar{acd},0),1)$$

$$n_2 = M(M(\overline{M(b,d,0)},M(M(a,c,0),d,0),0), \\ M(M(b,d,0),\overline{M(M(a,c,0),d,0)},0),1)$$

$$n_2 = M(M(M(\bar{b}, \bar{d}, 1), M(M(a, c, 0), d, 0), 0), \\ M(M(b, d, 0), M(\overline{M(a, c, 0)}, \bar{d}, 1), 1), 1)$$

$$n_2 = M(M(M(\bar{b}, \bar{d}, 1), M(M(a, c, 0), d, 0), 0), \\ M(M(b, d, 0), M(\overline{M(a, c, 0)}, \bar{d}, 1), 1), 1)$$

Example: Generate the equivalent QCA logic expression for $f_2(a, b, c) = \sum(0, 2, 3, 4, 7)$.

Table 6.2 shows the minimization chart for function f_2 . For unprimed miniterms in minimization chart, 1s are filled for every column [10].

TABLE 6.2

MINIMIZATION CHART FOR FUNCTION F_2

f_2	1	a	b	c	ab	ac	bc	abc
000	1	1	1	1	1	1	1	1
010			1		1		1	1
011							1	1
100		1			1	1		1
111								1
	✓	x	x	✓	✓	x	✓	✓

The XOR-AND function for this example is

$$f_2 = 1 \oplus c \oplus ab \oplus bc \oplus abc$$

And utilizing ($\bar{x} = 1 \oplus x$) XOR property, the above function can be simplified to

$$f_2 = \bar{c} \oplus ab \oplus \bar{abc}$$

Fig. 6.6 shows the majority gate tree for function f_2 which helps to construct majority gates layout by replacing each node with equivalent majority XOR and AND gates.

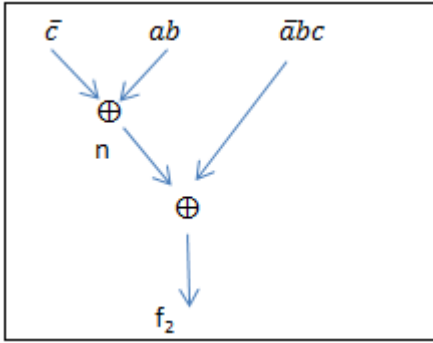


Fig. 6.6. Majority gate tree for function f_2

QCA logic expression for function f_2 is

$$f_2 = M(M(c, ab, 0), M(c, \overline{ab}, 0), 1)$$

$$f_2 = M(M(c, M(a, b, 0), 0), \\ M(c, \overline{M(a, b, 0)}, 0), 1)$$

$$f_2 = M(M(c, M(a, b, 0), 0), \\ M(c, M(\overline{a}, \overline{b}, 1), 0), 1)$$

6.4 Conclusion

We presented a systematic QCA logic construction method. Our novel method takes Boolean function as its input, generates simplified XOR-AND equivalent circuit and outputs an equivalent QCA logic circuits. In our novel method, we were able to simplify the Boolean functions and reduce number of majority gates with the help of XOR-AND reduction techniques then mapping QCA logic to Boolean functions.

CHAPTER 7

IMPLEMENTATION OF GENERALIZED PIPELINE CELLULAR ARRAY USING QUANTUM-DOT CELLULAR AUTOMATA

7.1 Introduction

During the last decade, Quantum-Dot Cellular Automata (QCA) has attracted a lot of attention due to its extremely small size and its ultralow power consumption as compared to COMS technology. It has been demonstrated that QCA has the ability to implement both combinational and sequential logic devices [76]–[83].

The fundamental unit of a QCA circuit is quantum cell which typically contains four quantum dots, placed near the corners of the cell where free electrons can reside. Quantum cells have two distinct stable polarizations, as shown in fig. 7.1. These states allow the cell to represent binary data.

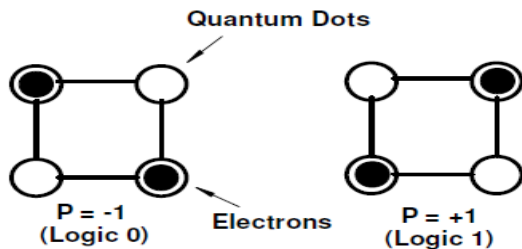


Fig. 7.1. Quantum cells with polarity -1 & polarity +1

QCA Binary wires are the simplest QCA structures and consist of a series of quantum cells in close proximity to each other. The cells interact through Coulombic interactions with each other as shown in fig. 7.2. Binary wires can also be constructed by orienting the dots in each cell at a 45 degree angle from the standard cell. This allows binary wires to cross in the same plane or layer without interacting with each other.

There are two logic gates that make up the fundamental set of logic in QCA: majority gate and inverter. By carefully arranging the location of QCA cells, one can create a majority logic gate, which is capable of functioning as either an AND or an OR gate.

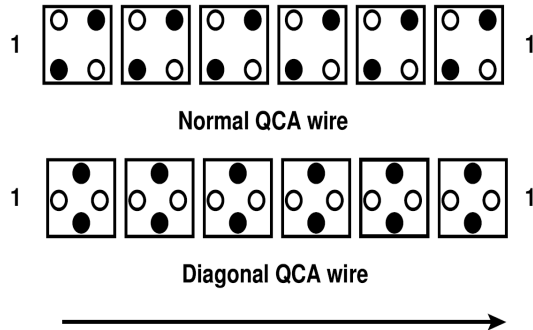


Fig. 7.2. QCA binary wire arrangements

QCA majority gate takes three inputs and outputs a value that occurs most frequently as

$$M(A, B, C) = AB + AC + BC \quad (7.1)$$

The majority gate can also be used to create AND and OR gates. If one input is held at 1, the majority gate functions as a standard 2-input OR gate. If one input is held at 0, the majority gate functions as a 2-input AND gate. Fig. 7.3 shows standard QCA majority gate construction.

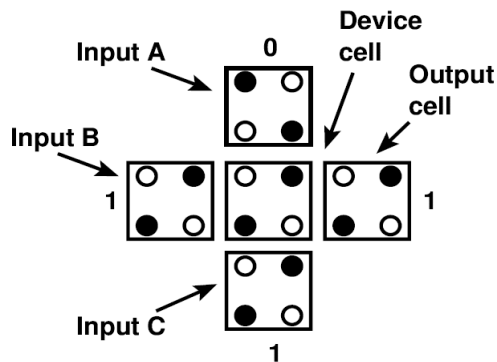


Fig. 7.3. QCA majority voter gate

QCA inverter gate has a single input and output. It simply returns the opposite of the value that was put in as shown in fig. 7.4.

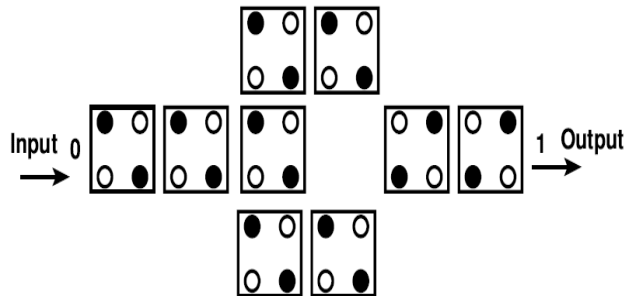


Fig. 7.4. QCA inverter gate

QCA has a four-phase clocking mechanism. The sequence of the states in this scheme is the switch state, hold state, release and relax states [76]. In the switch state, QCA cells start getting polarized. In hold state, the cells retain their polarization. During release and relax states, QCA cells are unpolarized as shown in fig. 7.5.

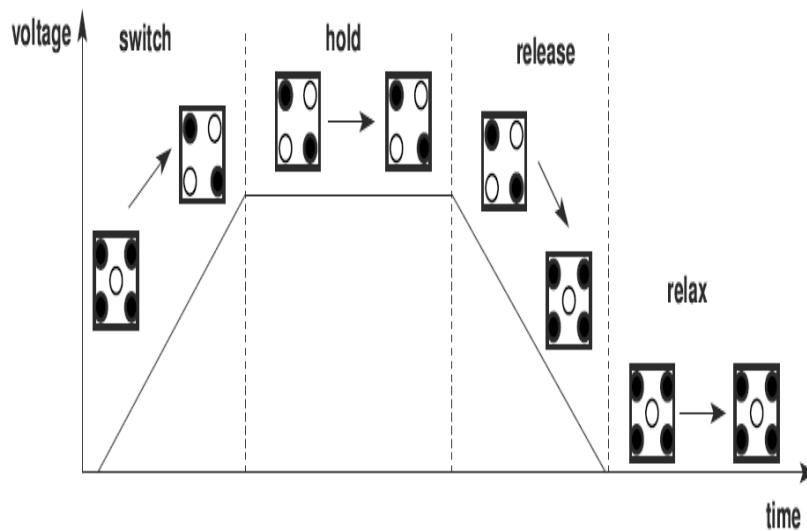
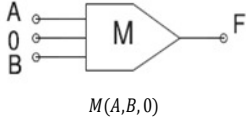
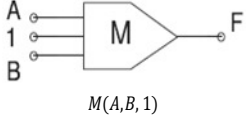
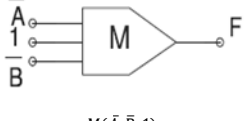
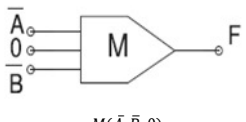
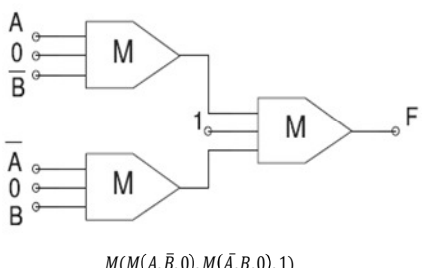
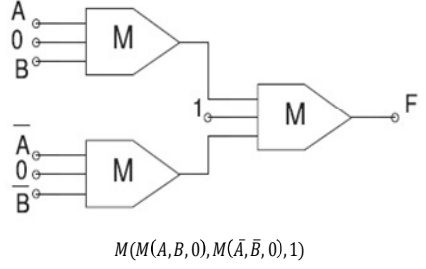


Fig. 7.5. Four QCA four phase clocking mechanism

TABLE 7.1

BOOLEAN FUNCTIONS AND THEIR EQUIVALENT QCA EXP.

Boolean Function	Majority Diagram / Expression
$F = AB$	 <p style="text-align: center;">$M(A, B, 0)$</p>
$F = A + B$	 <p style="text-align: center;">$M(A, B, 1)$</p>
$F = \overline{AB}$ $= \overline{A} + \overline{B}$	 <p style="text-align: center;">$M(\overline{A}, \overline{B}, 1)$</p>
$F = \overline{A + B}$ $= \overline{A} \overline{B}$	 <p style="text-align: center;">$M(\overline{A}, \overline{B}, 0)$</p>
$F = A \oplus B$ $= A\overline{B} + \overline{A}B$	 <p style="text-align: center;">$M(M(A, \overline{B}, 0), M(\overline{A}, B, 0), 1)$</p>
$F = A \odot B$ $= AB + \overline{A}\overline{B}$	 <p style="text-align: center;">$M(M(A, B, 0), M(\overline{A}, \overline{B}, 0), 1)$</p>

Lemma 7.1: If x and y are two binary inputs, then $xy = M(x, y, 0)$

Proof:

By equation (7.1)

$$M(x, y, 0) = xy + 0 + 0 = xy$$

Lemma 7.2: If x and y are two binary inputs, then $x \oplus y = M(M(\bar{x}, y, 0), M(x, \bar{y}, 0), 1)$

Proof:

$$M(M(\bar{x}, y, 0), M(x, \bar{y}, 0), 1) = M((\bar{x}y + 0 + 0), (x\bar{y} + 0 + 0), 1) =$$

$$(\bar{x}y)(x\bar{y}) + \bar{x}y + x\bar{y} =$$

$$\bar{x}y + x\bar{y} = x \oplus y$$

Lemma 7.3: If x and y are two binary inputs, then $\overline{M(x, y, 0)} = M(\bar{x}, \bar{y}, 1)$

Proof:

$$M(x, y, 0) = xy$$

$$\overline{M(x, y, 0)} = \overline{xy} = \bar{x} + \bar{y}$$

By equation (2b)

$$\bar{x} + \bar{y} = M(\bar{x}, \bar{y}, 1)$$

The use of generalized cellular pipeline arrays for various arithmetic operations has shown considerable promise in optical computer architecture because of the obvious advantages of improvement in speed and reduction in the cost and size. Cellular pipeline array consist of regular interconnections of selected logic sub-circuits called cells or processing elements (PE). The basic approach is to keep the number of I/O terminals to cellular array module to a minimum and supply control parameters as inputs to the arithmetic cells.

Different pipeline array designs have appeared in literature [88]-[92]. Singh [91] presented a generalized cellular array which can perform all of the basic arithmetic operations such as multiplication, division, squaring, and square rooting; and exploits the concept of pipelining. The basics of pipeline array is that the arithmetic operations are grouped together in single array with some additional control logic that can be used to realize the required arithmetic operation. Grouping these arrays (processing element units) can provide a single array network that can perform fast processing arithmetic operation. In this chapter we implemented pipeline array using QCA method design and comparing the design with [92].

7.2 QCA Pipeline Array

The generalized QCA pipeline array can perform all the basic arithmetic operations such as multiplication, division, squaring, and square rooting. The electronic implementation of a generalized pipeline array is adopted from an existing architecture [91]. Fig. 7.6 shows a block diagram for cellular pipeline array. The array consists of processing elements (PEs) with each PE communicating with its neighbours in the array either directly or through latches. The arithmetic cells marked as A are controlled 1-bit adders. The cells marked as C are control cells that specify the type of arithmetic operations to be performed by the arithmetic cells. The cells marked as M are used for multiplication. The Cells marked S are used for squaring and square rooting.

Fig. 7.7a shows a block diagram of an arithmetic cell, where lines A, B, and C are operand inputs, and lines X and F are control signals. The control unit specifies the type of operations to be performed in each PE. Fig. 7.7b shows a block diagram for control unit where P is an input, F_i is output, and X and C_0 are inputs and pass-through a PE to adjacent cells.

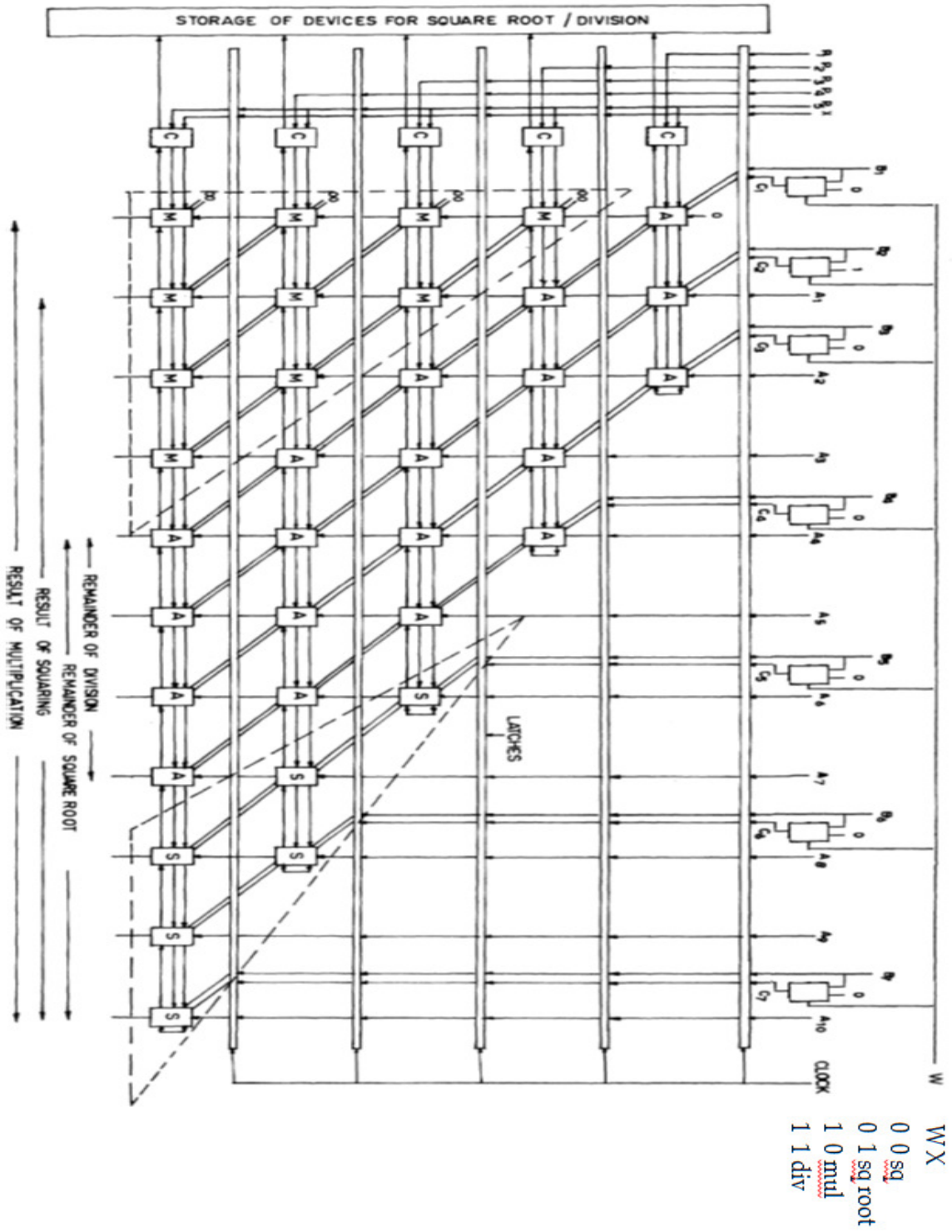


Fig. 7.6. Block diagram for pipeline array [91]

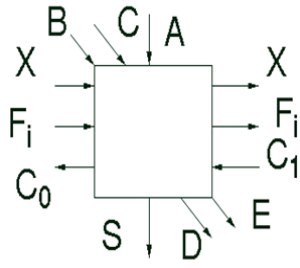


Fig. 7.7a. Arithmetic cell

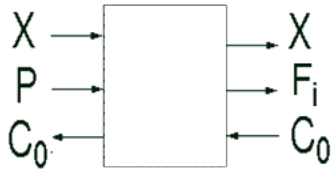


Fig. 7.7b. Control logic cell

The arithmetic cell is capable of performing the following Boolean operations:

$$S = [A \oplus (B \oplus X) \oplus C_1]F_i + A\bar{F}_i$$

$$C_0 = (B \oplus X)(A + C_1) + AC_1$$

$$D = C(B + F_i)$$

$$E = (B + C)(B + F_i)$$

And the Boolean expression for control cell is:

$$F_i = C_0X + P_i\bar{X}$$

Using lemma 7.1, 7.2 and 7.3, the above Boolean equations can be written in following QCA format

$$S = M(M(n_3, F_i, 0), M(A, \bar{F}_i, 0), 1)$$

$$C_0 = M(M(A, C_1, 1), n_2, 0), M(A, C_1, 0), 1)$$

$$D = M(C, M(B, F_i, 1), 0)$$

$$E = M(M(B, C, 1), M(B, F_1, 1), 0)$$

$$F = M(M(C_0, X, 0), M(P_1, \bar{X}, 0), 1)$$

Where

$$n_3 = M(M(\bar{n}_1, n_2, 0), M(n_1, \bar{n}_2, 0), 1)$$

$$n_2 = M(M(\bar{A}, C_1, 0), M(A, \bar{C}_1, 0), 1)$$

$$n_1 = M(M(\bar{B}, X, 0), M(B, \bar{X}, 0), 1)$$

Table 7.2 explains the required control signals for each arithmetic operation. When $X=0$, the arithmetic cell acts as an adder, and as a subtracter when $X = 1$. Sum and carry output are S and C_0 respectively. The operands are applied at inputs A and B . The most significant bits of the inputs are A_1 and B_1 . The most significant bit of the sum is S_1 . The array is capable of finding the square root of a ten-bit binary number A_{1-10} with control inputs P_{1-5} are made zero and X is made 1. The B and C inputs to the first level are given as 00, 01, 10, 10, 10, and 10, as shown in fig. 7.6. To find the square root of a number, it is applied across A , and then 01 is subtracted from the two most significant bits of A . If the remainder is positive, then the value of F_1 is 1; otherwise, it is 0. If F_1 is 0, the original value is kept for the next subtraction.

Table 7.3 shows the value of the subtrahend for each succeeding stage.

TABLE 7.2

QCA PIPELINE ARRAY ARITHMETIC SUMMARY OPERATIONS

Function	Description of pipeline array operations [7]
+	$X=0, F_1=1$ $S = A \oplus B \oplus C_1$
-	$X=1, F_1=1$ $S = A \oplus \bar{B} \oplus C_1$
x	$X=0, A=0, B=C$ Right shift add method is used Multiplicand in B Multiplier in P
÷	$X=1, B=C; P=0$ Right shift and subtract method is used Dividend in A Divisor in B
() ²	$X=0, A=0$ $B=2$'s comp of "10", $C="10"$ Operand in P
$\sqrt{\quad}$	$X=1, P=0$ $B=2$'s comp of "10", $C="10"$ Operand in A

TABLE 7.3

SUBTRAHEND AT DIFFERENT LEVELS FOR SQUARE ROOTING

0	1								
0	F_1	0	1						
0	0	F_1	F_2	0	1				
0	0	0	F_1	F_2	F_3	0	1		
0	0	0	0	F_1	F_2	F_3	F_4	0	1

The array is also capable of taking the square of a 5-bit number. To find the square of a number, it is applied across P_i with $X=0$, the arithmetic cells act as an adder, and the control cell transform P_i to F_i . Resulting in square of the number. The array can also be used to multiply a three-bit number B_{1-3} by a five-bit number P_{1-5} with control bit X and A inputs are made zero. The array can divide a seven bit number A_{1-7} by four-bit number B_{1-4} , giving a four-bit quotient and a four-bit remainder. For this case, the control input X is made 1, and P inputs are made zero. Similar to the multiplication operation, the C inputs are kept the same as the B inputs. The array requires $n(n+2)$ arithmetic cells and n control cells. The delay of an arithmetic operation depends on the delay in processing the last level which uses $2n+1$ arithmetic units and it is given by [91].

$$\tau_{delay} = n\tau_a + \tau_c + \tau_l$$

Where τ_a , τ_c , and τ_l are the delays in arithmetic cell, control cell, and latch circuit, respectively. Fig. 7.8 shows QCA design arithmetic and control cell unit layout.

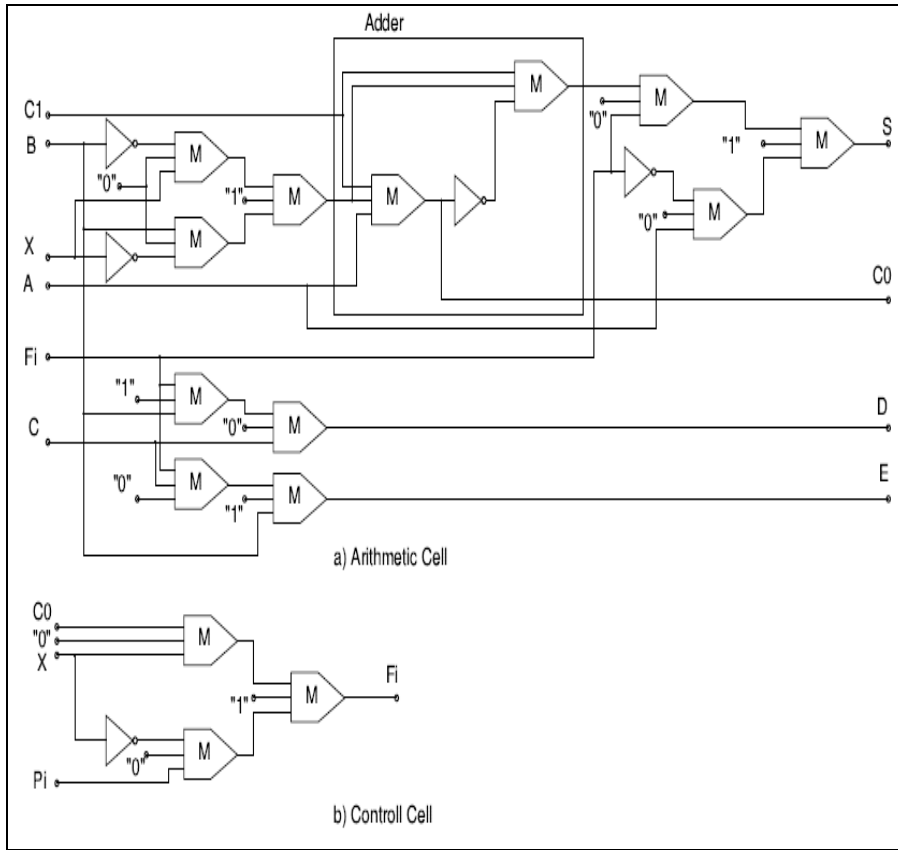


Fig. 7.8. (a) QCA arithmetic cell and (b) control cell

Agrawal [92] proposed high-speed multifunction array for multiplication, division, square-root operations. QCA equations for Agrawal's arithmetic cell can be giving by:

$$S_i = M(M(a_i, \bar{n}_3, 0), M(\bar{a}_i, n_3, 0), 1)$$

$$c_{i+1} = M(n_6, n_5, 1)$$

$$e_{i+1} = M(n_6, n_7, 1)$$

$$G_{i+1} = M(n_9, e_i, 0)$$

$$P_{i+1} = M(n_9, e_i, 1)$$

$$g_{i-1} = M(n_{10}, d_i, 0)$$

$$h_{i-1} = M(n_{11}, b_i, 1)$$

Where

$$n_1 = M(M(b_i, \bar{x}, 0), M(b_i, x, 0), 1)$$

$$n_2 = M(r_j, n_1, 1)$$

$$n_3 = M(M(\bar{c}_i, n_2, 0), M(c_i, \bar{n}_2, 0), 1)$$

$$n_4 = M(a_i, c_i, 1)$$

$$n_5 = M(n_4, n_2, 0)$$

$$n_6 = M(a_i, c_i, 0)$$

$$n_7 = M(n_6, n_1, 0)$$

$$n_8 = M(M(\bar{c}_i, n_1, 0), M(c_i, \bar{n}_1, 0), 1)$$

$$n_9 = M(M(\bar{a}_i, n_8, 0), M(a_i, \bar{n}_8, 0), 1)$$

$$n_{10} = M(r_j, b_i, 1)$$

$$n_{11} = M(r_j, d_i, 0)$$

Fig. 7.9 shows QCA design high speed arithmetic cell unit layout.

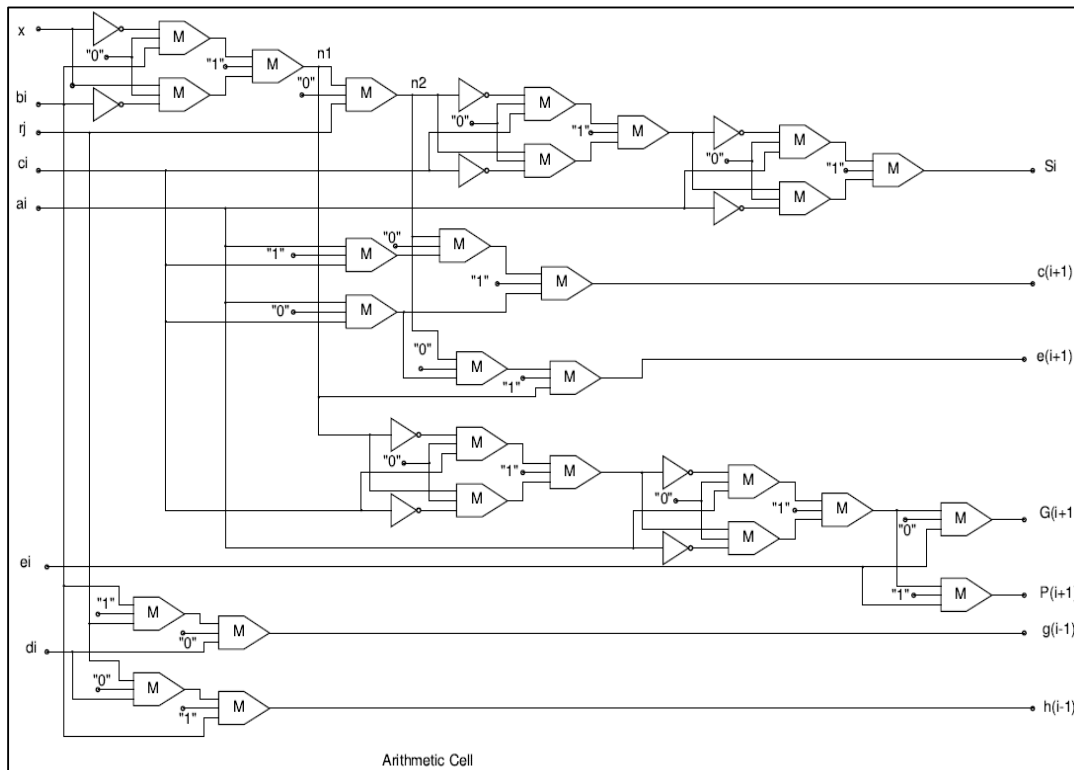


Fig. 7.9 QCA high speed arithmetic cell

7.3 QCA Pipeline Implementation

For creating QCA pipeline array and verifying the design functionality, QCADesigner [86] is used. The tool provides two simulation engines: bistable engine and coherency vector engine. QCA cells are assumed to have a height of 18nm and width of 18nm while the quantum dots have a diameter of 5nm. This follows the same assumptions as given in [87] and coherency vector engine has been used for simulations. Fig. 7.10 and fig. 7.11 show the QCADesigner layout for the arithmetic cell and control cell respectively. The layout is labeled to indicate majority gates inputs as well as the outputs. Fig. 7.12 and fig. 7.13 show the QCADesigner simulations results.

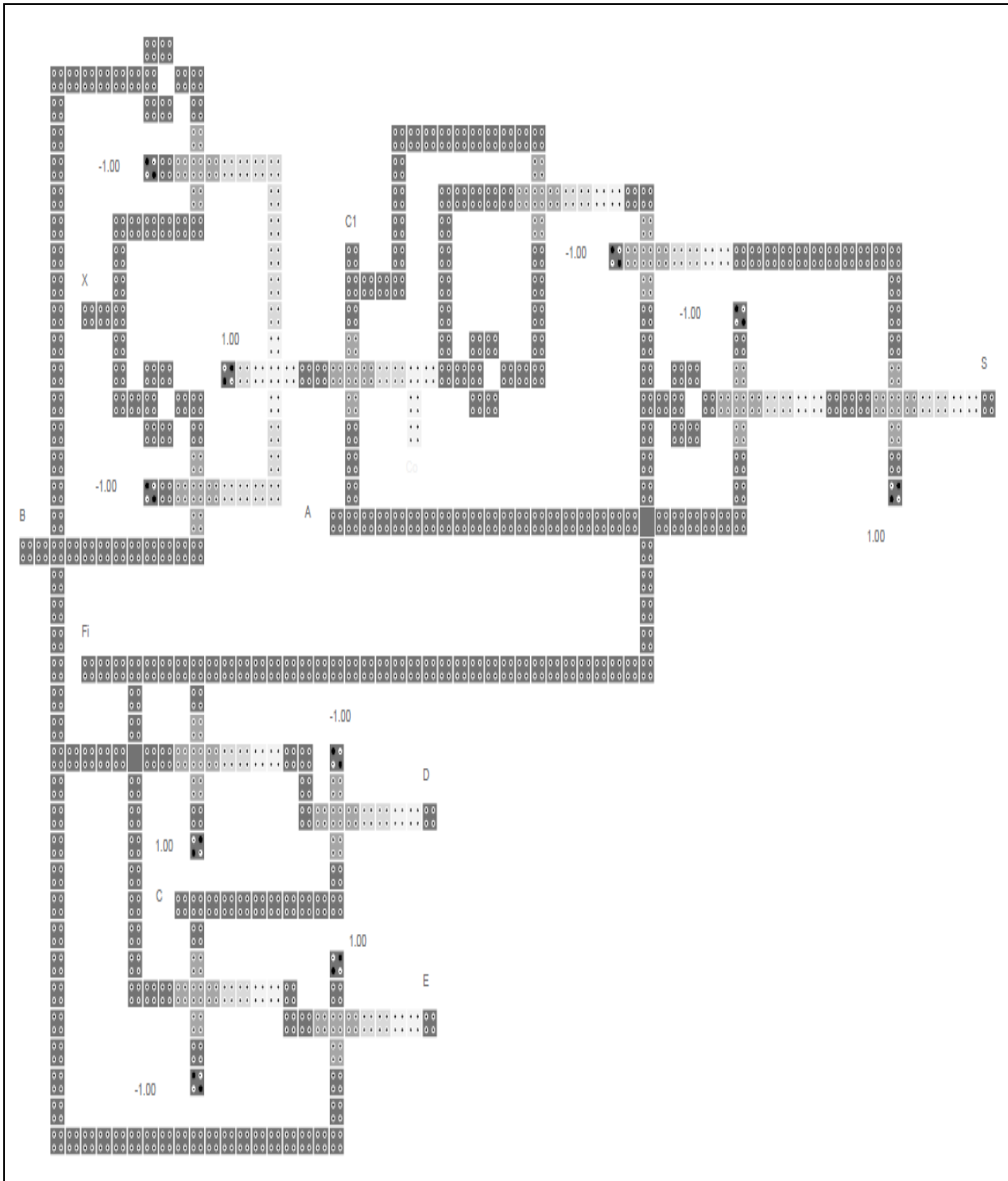


Fig. 7.10. QCADesigner layout for arithmetic cell unit

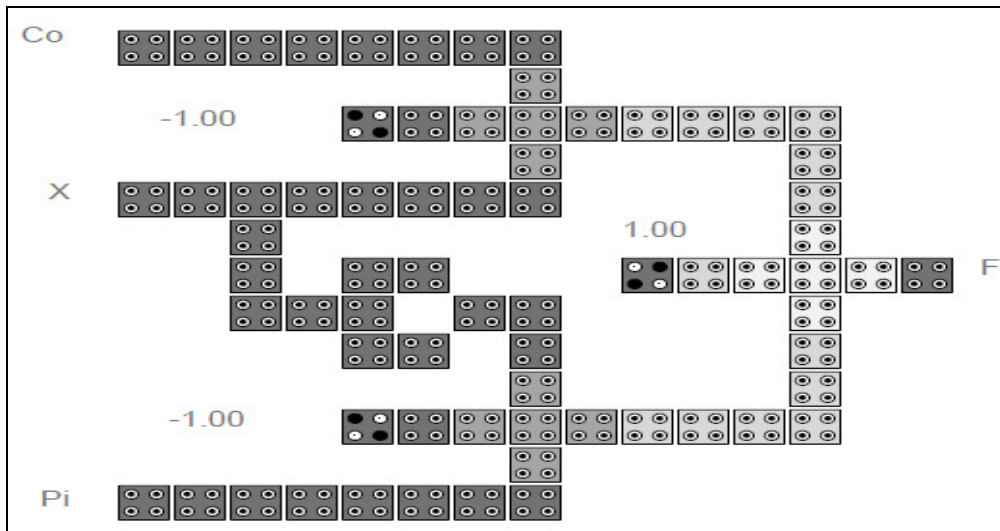


Fig. 7.11. QCADesigner layout for control cell unit.

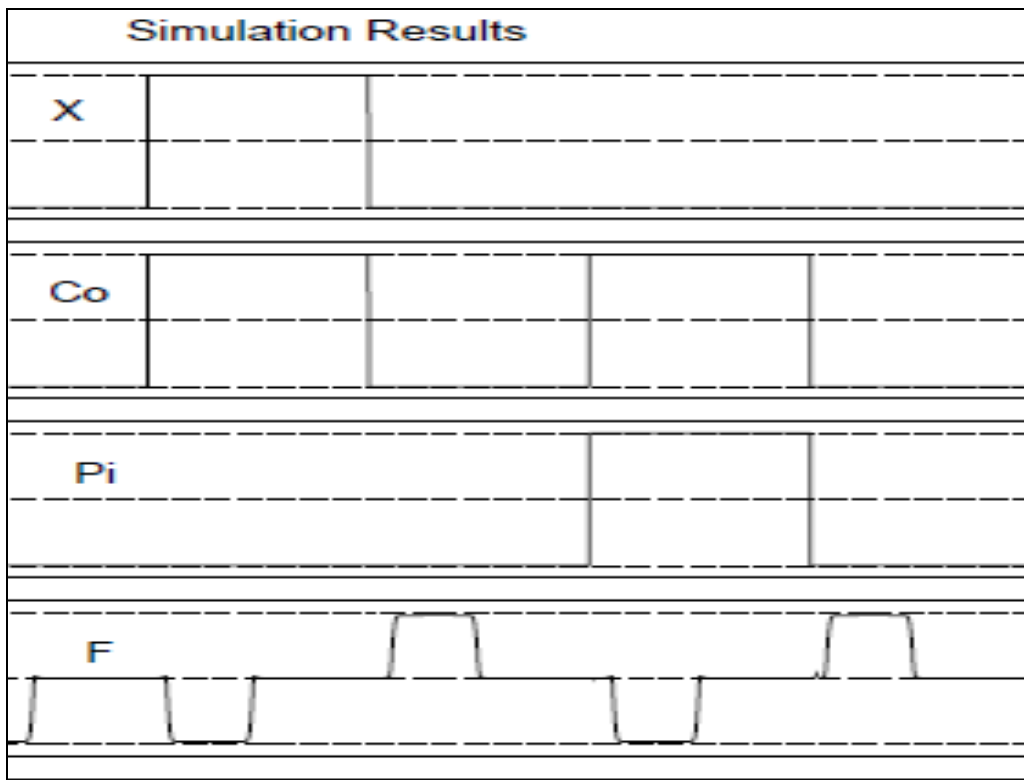


Fig. 7.12. Simulation for control cell unit

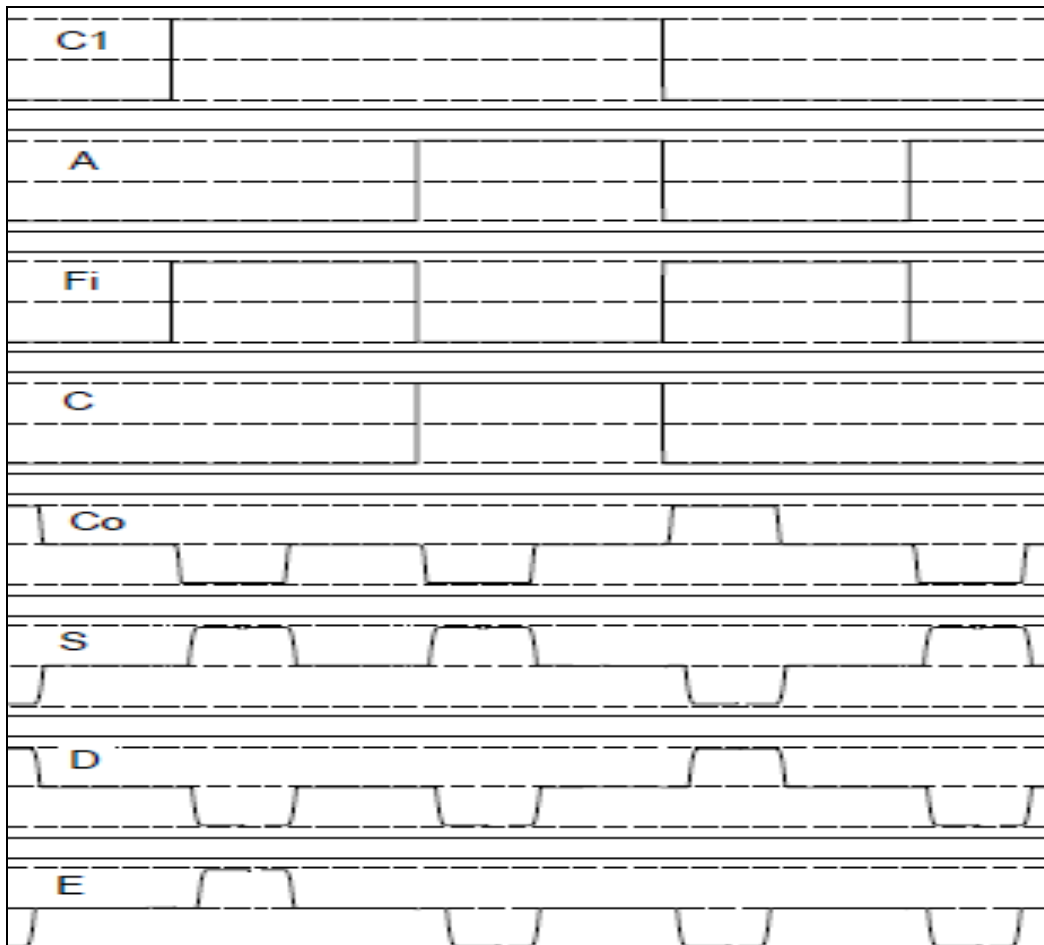


Fig. 7.12. Simulation for arithmetic cell unit

Fig. 7.13 and fig. 7.14 show the QCADesigner layout for the Agrawal's arithmetic cell and simulation results respectively.

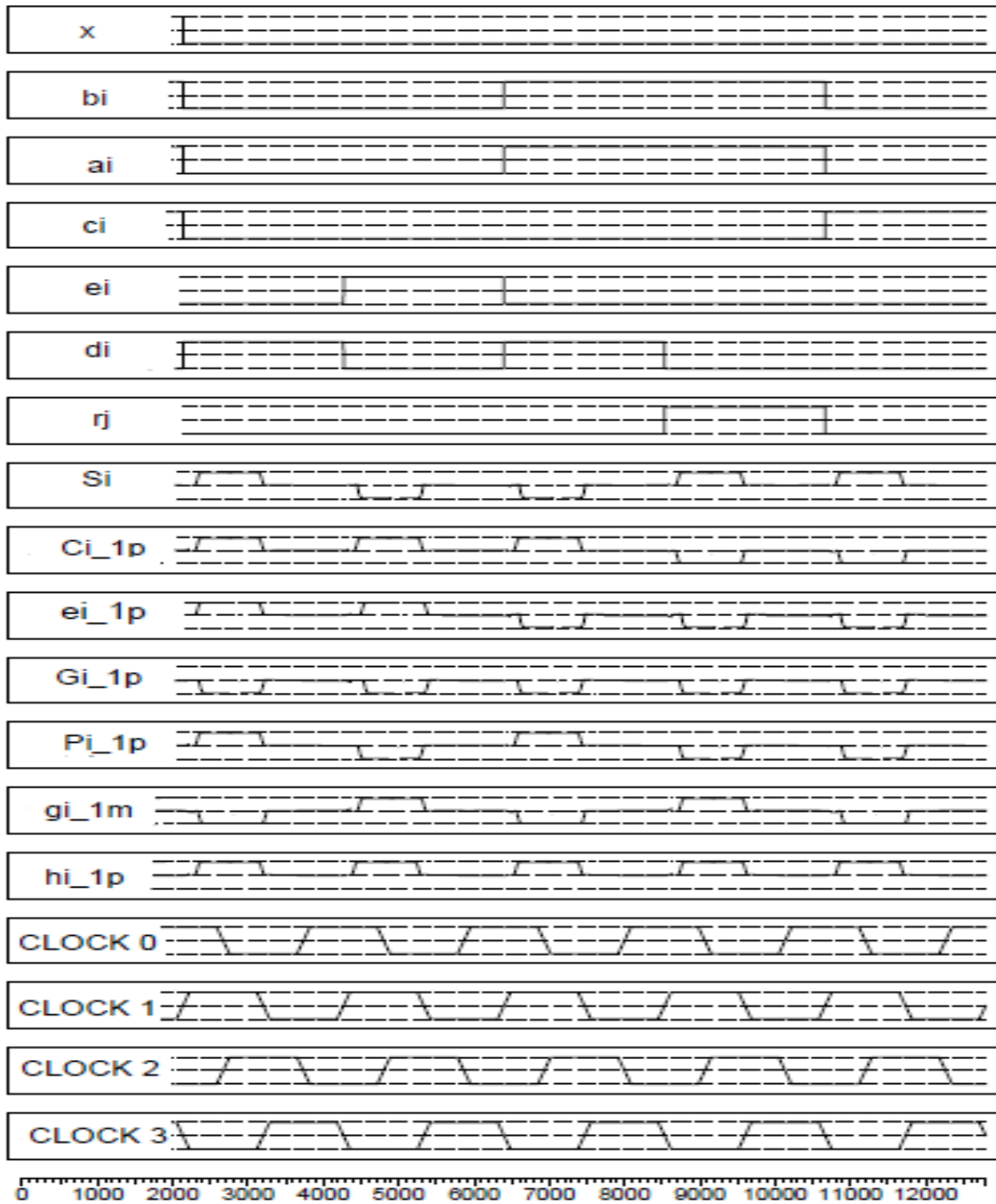


Fig. 7.14. Simulation for high speed arithmetic cell unit

Table 7.4 shows performance comparison between the two QCA designs showing the arithmetic and control cells have a simpler structure than Agrawal's arithmetic cell.

TABLE 7.4

QCA PERFORMANCE COMPARISON BETWEEN THE TWO DESIGNS

Parameter	Value	
	Arithmetic & Control Cells	Agrawal's Arithmetic Cell
Number of Cells	398	1004
Number of Majority Gates	15	28
Number of NOT Gates	5	10
Covered Area μm^2	0.58	1.46

A different modeling approach has been used to simulate 10-bit QCA pipeline array. We created a behavior Verilog model for majority gate and used it as building block for creating majority AND, OR, NOT and XOR QCA gates. Then we used Cadence NCLaunch simulation tool to test our design. Fig. 7.15 and fig. 7.16 show the result of squaring and square rooting outputs respectively.

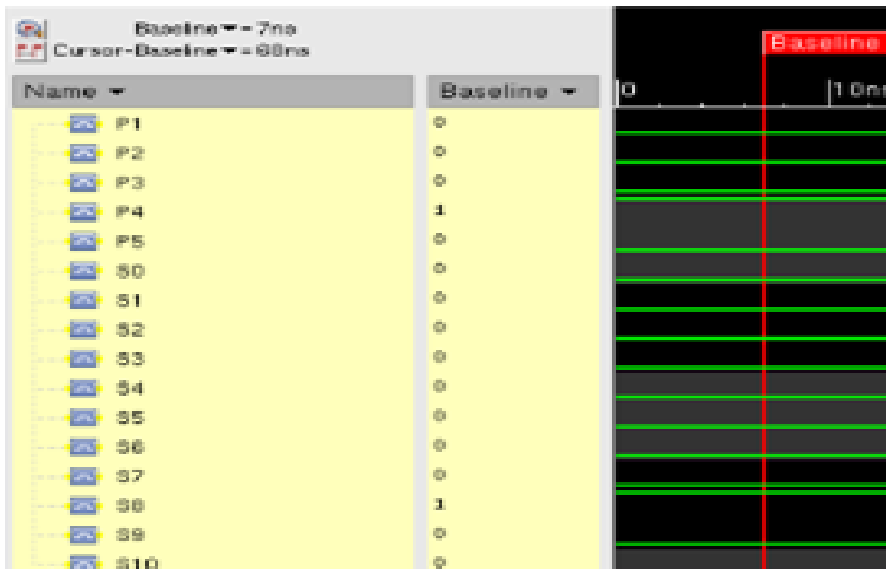


Fig. 7.15. Waveform of pipeline squaring output result

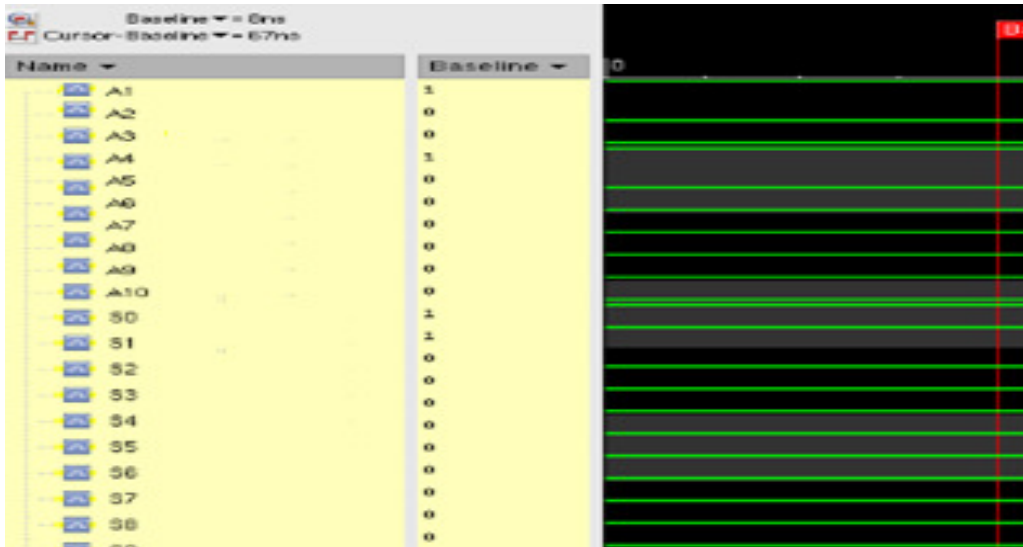


Fig. 7.16. Waveform of pipeline square rooting output result

Fig. 7.17, fig 7.18 and fig. 7.19 show Multisim implementation for arithmetic cell, control cell and high speed arithmetic cell.

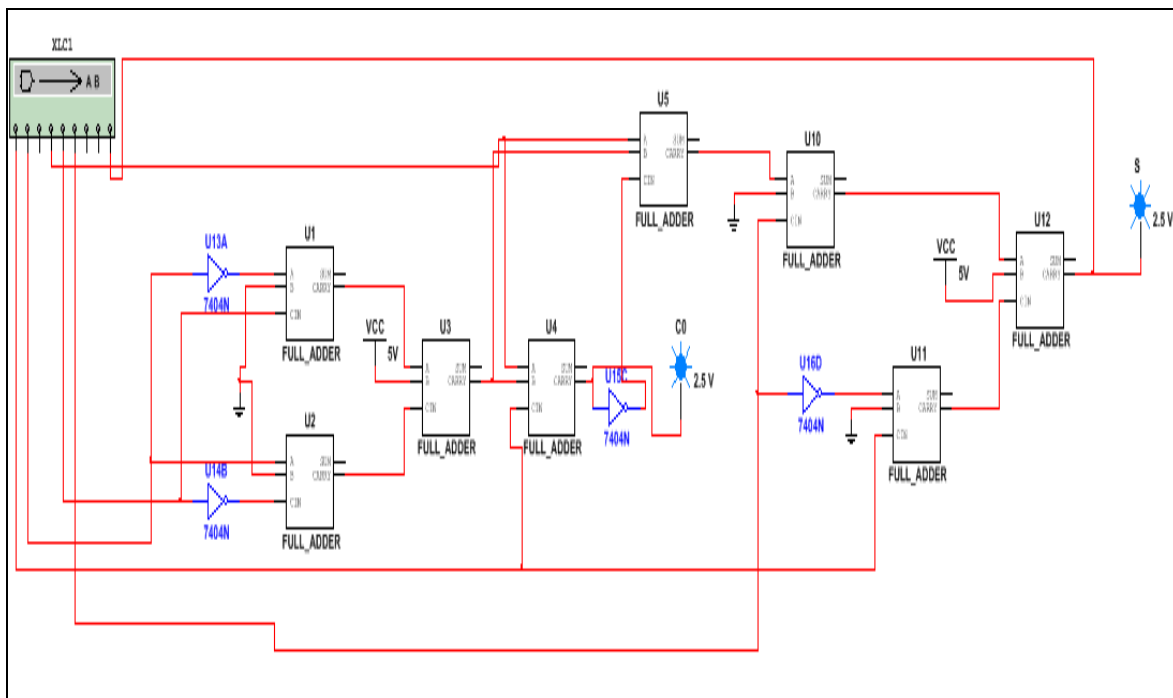


Fig. 7.17. Multisim implementation of arithmetic cell

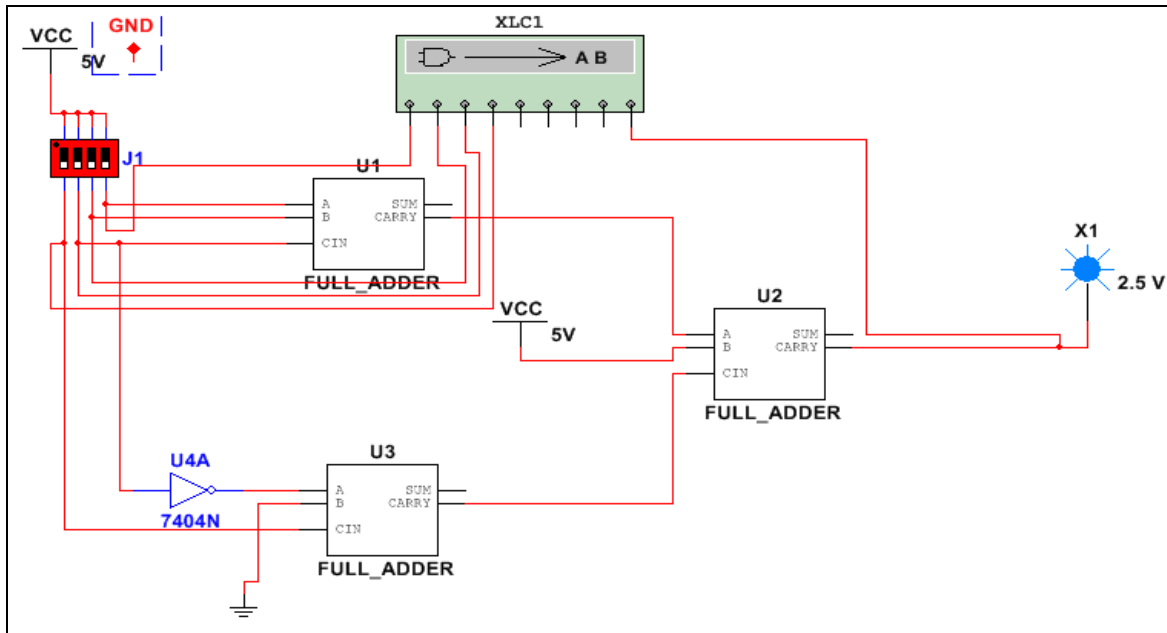


Fig. 7.18. Multisim implementation of control cell

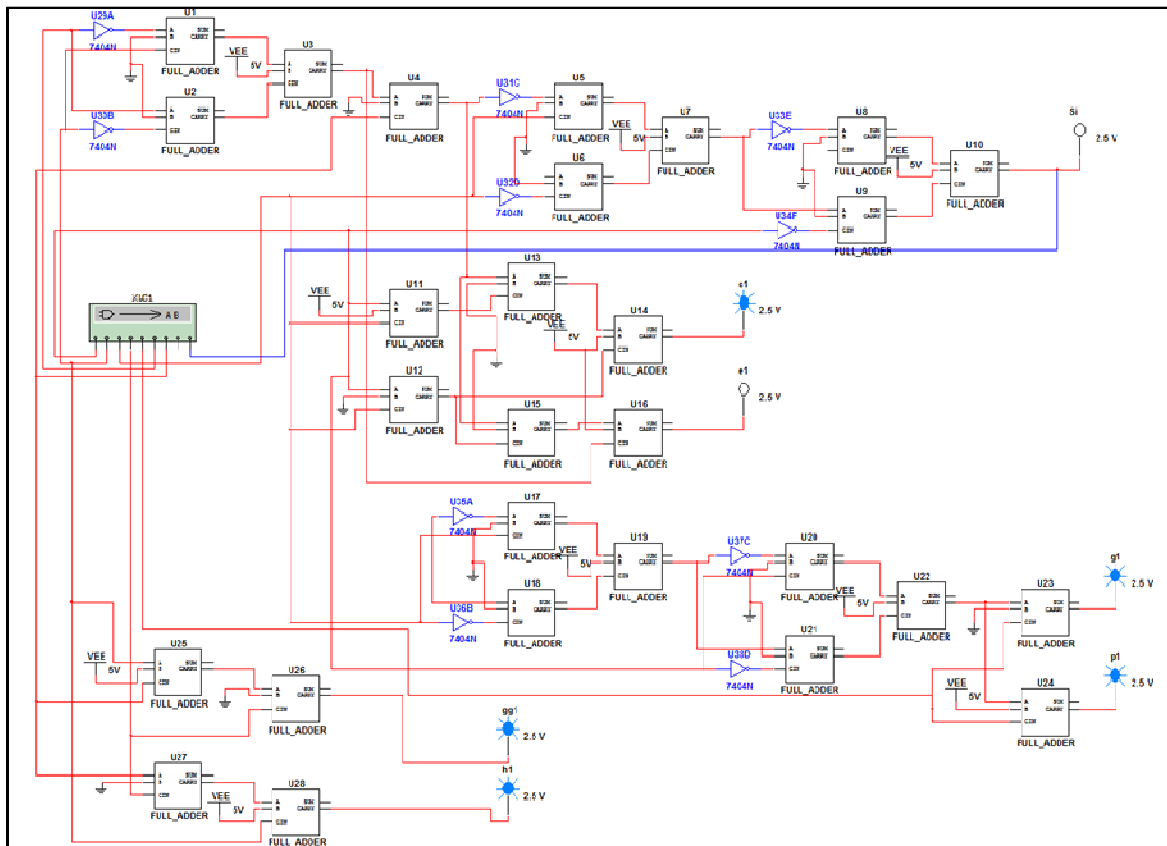


Fig.7.19 Multisim implementation of high speed arithmetic cell

Fig. 7.20, fig 7.21, fig. 7.22, fig. 7.23, fig 7.24 and fig. 7.25 show FPGA implementation for arithmetic cell, control cell and high speed arithmetic cell.

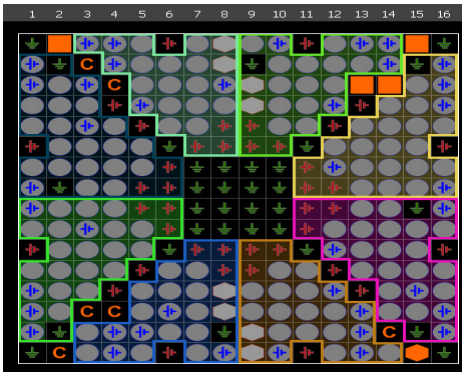


Fig 7.20. Arithmetic cell FPGA packaging

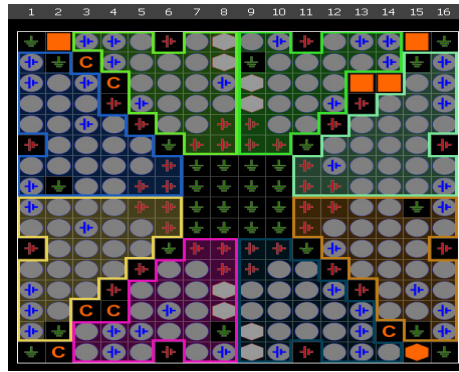


Fig 7.21. Control cell FPGA packaging

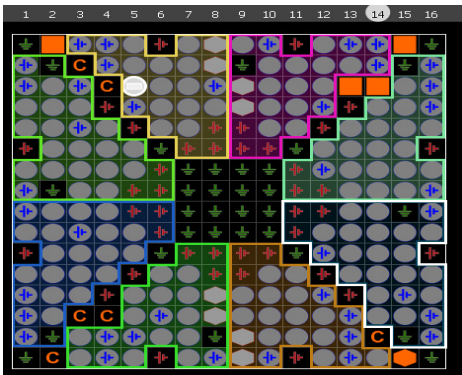


Fig 7.22. High speed arithmetic cell FPGA packaging

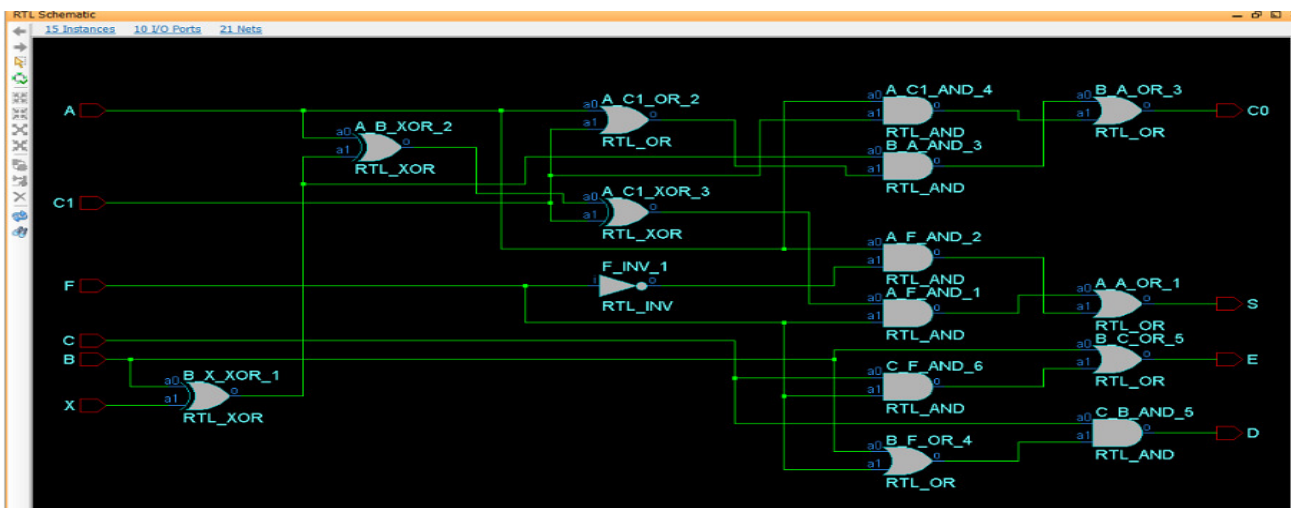


Fig 7.23. Arithmetic cell FPGA schematic layout

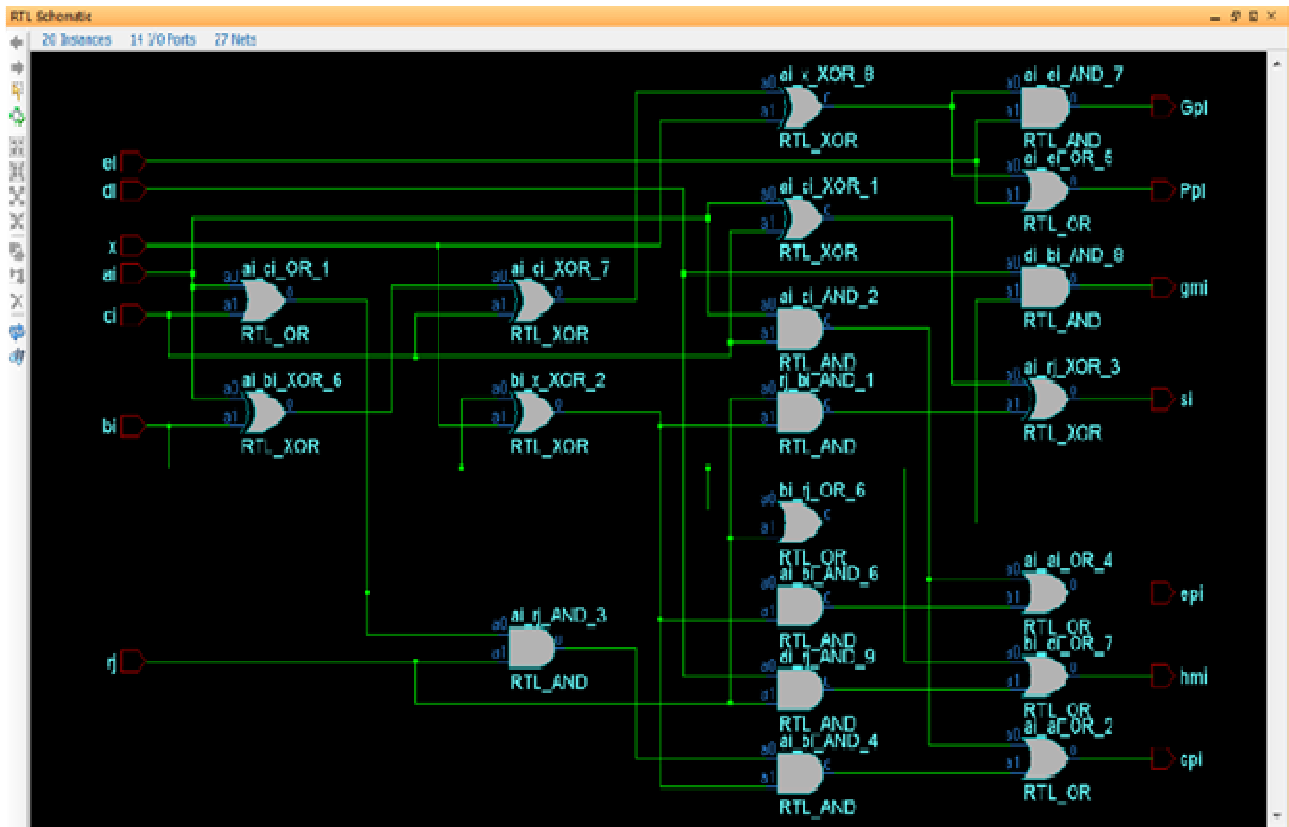


Fig 7.24. High Speed arithmetic cell FPGA schematic layout

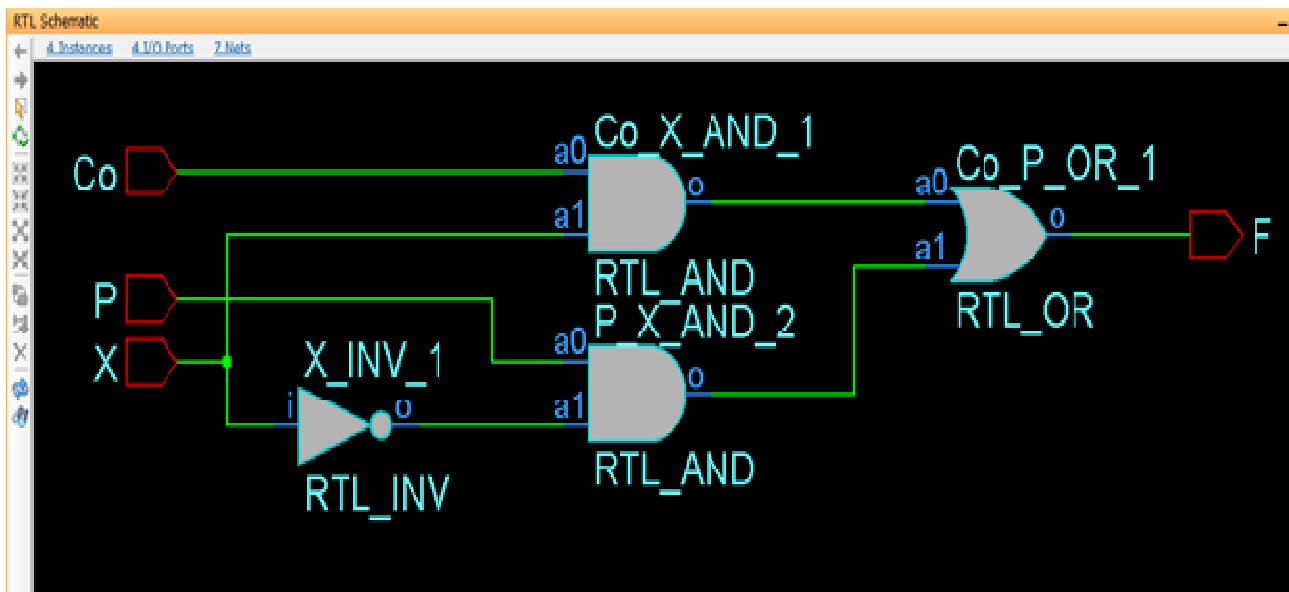


Fig 7.25. Control cell FPGA schematic layout

TABLE 7.6

QCA PERFORMANCE COMPARISON BETWEEN THE TWO DESIGNS

Parameter	Value		
	Control Cell	Arithmetic Cell	High Speed Arithmetic Cell
Number of Gates	4	15	20
Number of I/O	4	10	14
Number of Nets	7	21	27
Covered Area μm^2	648	2592	5400
Max. Delay Time (PS)	0.197	0.785	0.911
Chip Floor Plan Aspect Ratio	0.27	0.54	0.8
INVX1, OAI21X1 & NAND2X1 operation voltage	5	5	5

TABLE 7.7

DELAY TIME (PS)

Delay time from input pin to output pin (ps)																	
A	->	S	0.145	b_i	->	s_i	0.698	b_i	->	P_{i-1}	0.827	sqr	A_5	->	F_5	10.08	
B	->	S	0.528	x	->	s_i	0.911	x	->	P_{i+1}	0.304	mult	P_1	->	S_1	3.113	
X	->	S	0.785	c_i	->	s_i	0.454	c_i	->	P_{i+1}	0.581	mult	P_2	->	S_2	2.047	
C_{i-1}	->	S	0.762	r_j	->	s_i	0.706	e_i	->	P_{i+1}	0.328	mult	P_3	->	S_3	1.521	
F_i	->	S	0.308	a_i	->	c_{i+1}	0.297	b_i	->	g_{i-1}	0.163	mult	P_4	->	S_4	0.167	
A	->	C_0	0.356	b_i	->	c_{i+1}	0.505	r_j	->	g_{i-1}	0.14	mult	B_1	->	S_1	2.622	
B	->	C_0	0.308	x	->	c_{i+1}	0.526	d_i	->	g_{i-1}	0.129	mult	B_2	->	S_2	1.879	
X	->	C_0	0.327	c_i	->	c_{i+1}	0.306	b_i	->	h_{i-1}	0.139	mult	B_3	->	S_3	1.85	
C_{i-1}	->	C_0	0.325	r_j	->	c_{i+1}	0.326	r_j	->	h_{i+1}	0.15	mult	B_4	->	S_4	1.703	
B	->	D	0.169	a_i	->	e_{i+1}	0.3	d_i	->	h_{i-1}	0.136	div	A_1	->	F_1	0.407	
C	->	D	0.138	b_i	->	e_{i+1}	0.31	sq	P_1	->	S_1	1.918	div	A_2	->	F_2	2.594
F_i	->	D	0.146	x	->	e_{i+1}	0.37	sq	P_2	->	S_2	1.968	div	A_3	->	F_3	4.751
B	->	E	0.187	c_i	->	e_{i+1}	0.311	sq	P_3	->	S_3	2.084	div	A_4	->	F_4	8.506
C	->	E	0.17	a_i	->	G_{i+1}	0.796	sq	P_4	->	S_4	1.674	div	A_5	->	F_5	8.577
F_i	->	E	0.181	b_i	->	G_{i+1}	0.824	sq	P_5	->	S_5	0.339	div	B_1	->	F_1	0.27
X	->	F_i	0.138	x	->	G_{i+1}	0.331	sqr	A_1	->	F_1	0.271	div	B_2	->	F_2	2.509
P	->	F_i	0.197	c_i	->	G_{i+1}	0.578	sqr	A_2	->	F_2	2.326	div	B_3	->	F_3	5.537
C_0	->	F_i	0.129	e_i	->	G_{i+1}	0.319	sq	A_3	->	F_3	4.97	div	B_4	->	F_4	8.433
a_i	->	s_i	0.13	a_i	->	P_{i+1}	0.799	sqr	A_4	->	F_4	8.81					

Table 7.6 and 7.7 show performance comparison between the two designs. The results show that number of gates that were used in control cell are four gates, four I/O pins and total nets are seven. For arithmetic cell, the total gates are fifteen, ten input I/O pins and total nets are twenty one. Meanwhile, in high speed arithmetic cell, total gates are twenty, I/O pins are fourteen and total nets are twenty seven. The allocated covered area and chip floor plan aspect ratio for the high speed arithmetic cell were the highest

due to the total gates for it is more than arithmetic cell. Also, the tables show that the maximum delay time for arithmetic cell is smaller than in high speed arithmetic cell. The results show that arithmetic cell has less complex in hardware and processes smaller delay time than high speed arithmetic cell as high speed arithmetic requires two half adder in serial and needs more processing time.

We used Cadence Encounter to generate a COMS equivalent layout of our QCA pipeline array design. Ambit Buildgates is used to generate the netlist. Encounter used to assign floor plans, appropriate layers nano routing, and obtain the design in .gds and .def. In Encounter, we also checked for any design flawless in terms of connectivity, density etc. Fig. 7.26 shows the encounter part of our design.

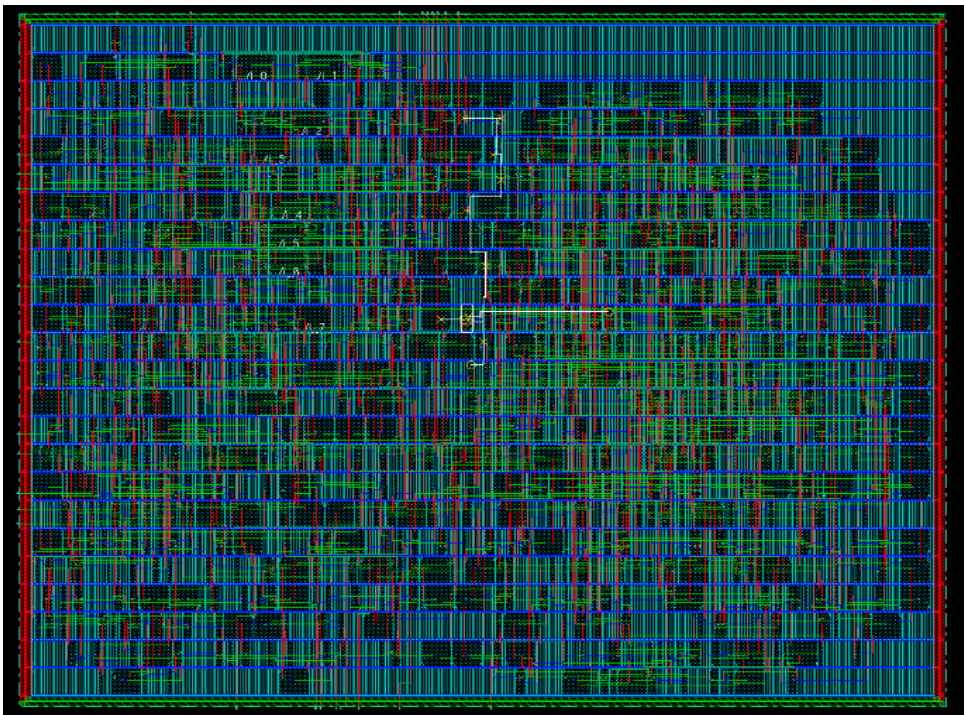


Fig. 7.26. Encounter part of QCA pipeline array

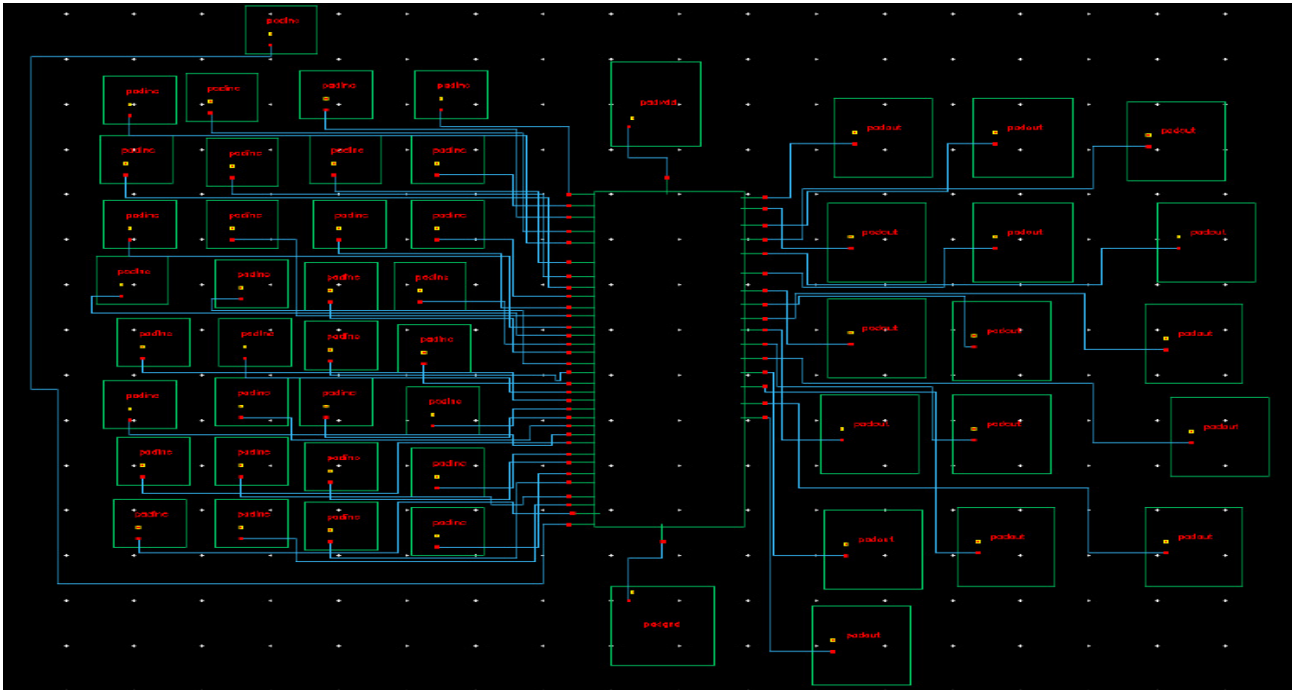


Fig.7.27. Virtuo part of padding pipeline array

For chip fabrication, we used Virtuoso where we have imported the design from encounter and the same padframed is used to send it to MOSIS for chip fabrication. The complete Layout along with the connections is shown in fig. 7.28 below. In digital circuits complexity is not appears to be defined because of a lot of parameters that can be used to measure digital complexity such as number of gates, number of I/O, delay, area/size. In this present work, we suggest a fuzzy complexity system. This concept needs to be validated and more works to be done. Fig 7.29 shows the hardware complexity fuzzy concept.

TABLE 7.7

COMPLEXITY FUZZY RESULTS FOR PIPELINE ARRAY CELLS

Parameter	Value		
	Control Cell	Arithmetic Cell	High Speed Arithmetic Cell
Complexity	0.41	0.63	0.86
Number of Gates	4	15	20
Number of I/O	4	10	14
Number of Nets	7	21	27

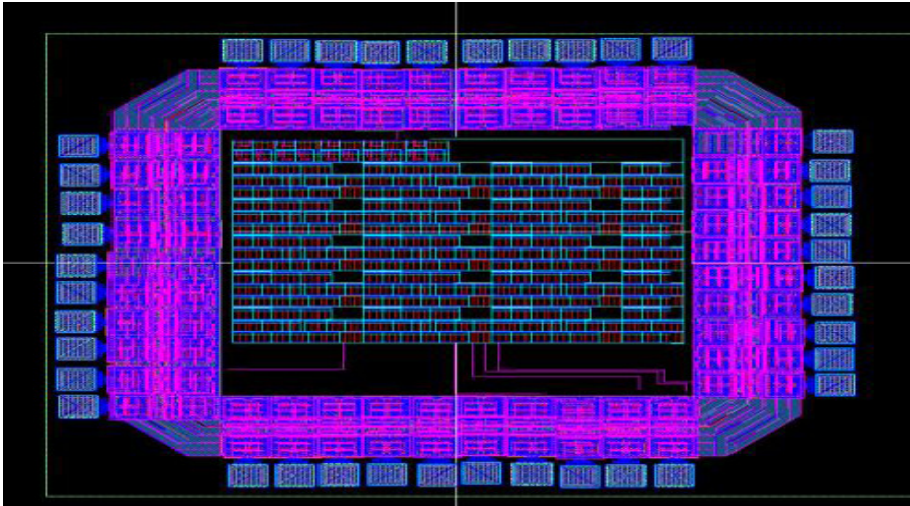


Fig.7.28. Virtuosio part of pipeline array

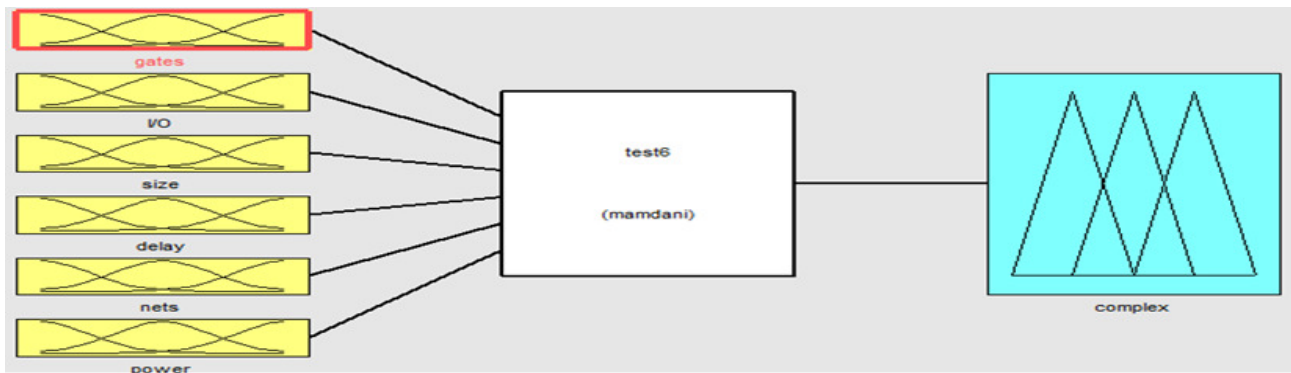


Fig.7.29. Hardware complexity fuzzy concept

7.4 Conclusion

We demonstrate that arithmetic cells can be successfully implemented using QCA cells. These arrays can perform multiplication, division, squaring and square rooting. All different modes of operation are controlled by a single control line. QCADesigner tool set is used to simulate both designs. We also used different VLSI approach to simulate 10-bit QCA pipeline array, we created behavior Verilog models for the design. Cadence NCLaunch simulation tool is used to simulate the pipeline array and verify that our design is performing as desired. Cadence encounter is used to generate a COMS equivalent layout of our QCA pipeline array design. Our Design layout is exported to virtuosio. Furthermore, padframing is performed in order to send the design to MOSIS for fabrication.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Introduction

Residue Number System (RNS) has received increased attention due to its ability to support high speed concurrent arithmetic applications such as Fast Fourier Transform (FFT), image processing and digital filters. In spite of its effectiveness, conversion and sign detection, base extension, scaling and division are complex operations with current methods. In this dissertation, we have addressed conversion, scaling and implementation of RNS adder and subtractor.

Quantum-Dot Cellular Automata (QCA) is attracting a lot of attention due to its extremely small feature size and ultralow power consumption compared to CMOS technology. This new type of nanotechnology uses different logic devices to design circuits. The traditional Boolean logic reduction methods such as Karnaugh maps produce a simplified Boolean expression. However, converting these forms to QCA Boolean is not simple process due to the complexity of multilevel majority gates. In this dissertation, we presented a novel methodology for generating QCA Boolean circuits from multi-output Boolean circuits and implementation of QCA pipeline array.

8.2 Summary of Work

A detailed research has been conducted on the residue number system and quantum-dot cellular automata. Specific solutions have been developed to provide solutions for stressing issues. The following gives an executive summary of the contributions and results of this research.

- A simplified algorithm method for conversion from binary to residue number system was introduced. The algorithm requires less hardware size compared to those required by existing algorithms. It utilizes parallel-prefix techniques with multiplexers and modulo adders as the main building blocks without the use of lookup tables which makes it practical and suitable for VLSI implementation. While other existing methods such as Behrooa [7] uses a table lookup schemes for binary to residue conversion and Alia [4] uses processing elements (PE) with complex hardware.
- A new scaling algorithm based on mixed radix conversion was presented. The algorithm utilizes a simplified base extension process that works on a smaller modulo. It provides an alternative method of finding the mixed radix digits with high degree of parallelism. The algorithm has advantages over CRT methods since it avoids the use of modulo computations and the use multiplicative inverse operation.
- An efficient VLSI approach for the implementation of RNS adder and subtractor was introduced. XILINX is used to get the behavioral simulation of the design with the help of which we were able to verify that our design is performing as desired. Cadence encounter is used to build the layout of the design. Design layout along with the technology libraries are exported to virtuoso. Furthermore, padframing was performed in order to send the design to for fabrication.

- A novel methodology for generating QCA Boolean circuits from multi-output Boolean circuits was developed. Our methodology takes as its input a Boolean circuit, generates simplified XOR-AND equivalent circuit and output an equivalent majority gate circuits.
- An efficient approach for implementation of a generalized pipeline cellular array using quantum-dot cellular automata cells was presented. The QCA pipeline array can perform all basic operations such as multiplication, division, squaring and square rooting. The different modes of operation are controlled by a single control line. We created behavior Verilog models for the design. Cadence NCLaunch simulation tool is used to simulate the pipeline array and verify that our design is performing as desired. Cadence encounter is used to generate a COMS equivalent layout of our QCA pipeline array design. Our Design layout is exported to virtuoso. Furthermore, padframing is performed in order to send the design to MOSIS for fabrication.

8.3 Recommended Future Work

The results of this research are promising and the following are recommended research topics that can be done as a continuation of this work.

- Further research can be done on investigating new techniques for generating Carbon Nanotubes (CNT) Boolean circuits. Carbon Nanotubes have many different structures, differing in length, thickness, and number of layers. Although, they are formed from essentially the same material sheet, their electrical characteristics differ depending on these variations. With these

properties, they are acting either as metals or as semiconductors. Nanotube-based transistors, also known as Carbon Nanotube Field-Effect Transistors (CNTFETs), are capable of digital switching using a single electron. However, one major obstacle for this new emerging technology has been the lack of new methods for mapping existing multi-output boolean logic circuits to carbon nanotubes logic circuits also creating new methods for generating carbon nanotubes logic circuits is other area of research.

- Modifying QCADesigner program to include QCA reduction algorithms that were presented in this dissertation. QCADesigner is program that had been developed by University of British Columbia to create a simulation tool and design for quantum-dot cellular automata. QCADesigner is open code source and can be downloaded from University of British Columbia. This tool is still under development and provided free of cost to the research community.
- Also, further research can be done on creating new methods for generating boolean circuits for Single Electron Transistor (SET) circuits. SET operates by injecting or ejecting a single electron into or from a dot of silicon, so producing a change in electronic potential. That change must overcome thermal agitation, making optimized smallness of the dot essential for SET operation. With this property, the single electron transistor is type of switching device that uses controlled electron tunneling to amplify current. Usually, SET is made from two tunnel junctions that share a common electrode. A tunnel junction consists of two pieces of metal separated by a very thin insulator. The only way for electrons in one of the metal electrodes is to travel through the insulator. One major obstacle

for this new emerging technology has been the lack of new techniques for creating SET circuits.

- Implementing five pins majority QCA gate. The basic elements in QCA are majority and inverter gates. As result, using a majority gate with more inputs in QCA circuit will cause reduction in cell count, latency and complexity. Furthermore, implementing seven input majority gate also could simplify and optimize QCA designs. Creating new VLSI techniques that can support five and seven majority gates and implementation different exciting circuits such as full adder with these QCA gates can be other research area.
- Investigating new fabrication techniques of QCA logic devices is other research areas that can be conducted. Fabrication of QCA is still ongoing challenging research area. These challenges include development of new manufacturing methodology for QCA circuit fabrication. Also implementation of detect free fault tolerant circuit. These new techniques require development new CAD methods and tools to help simulation.
- Implementing residue arithmetic logic unit that can perform all modulo operations. RNS is an unweighted representation system of numbers. RNS is based on modular arithmetic operations and it is a carry-free system. Creating modular logic unit that be used in applications such as Fast Fourier Transform (FFT), image processing and digital filters can simplify the RNS design and application implementation.
- Investigating new residue number system techniques to simplify RNS magnitude comparison overflow detection, sign detection, parity detection and division. In

spite of its effectiveness, RNS has remained more an academic challenge due to the complexity involved in the magnitude comparison, overflow detection, sign detection, parity detection and division. These RNS areas are other research area to investigate.

- Validating and improving the fuzzy complexity concept is other research area that can be investigated.

APPENDIX

Setting up your Working Environment:

1. Login to your Unix machine.
 - Use your WSU access ID and password.
2. Double click on the “?????’s Home” folder on your desktop.
 - (“?????’” should be your AccessID).

If you already did steps 3 to 10 go to step 11

3. Click “View” and check “Show Hidden Files”.
4. Scroll down to find the .cshrc file.
 - The file is currently Read Only.
 - Right click on the file and choose “Properties”.
 - Go to the “Permissions” tag and check “Owner ->Write”.
 - Click “Close”.
 - Now the file can be edited.
5. Right click on the file and choose “Open with Text Editor”.
 - This will open the .cshrc file in the text editor.
6. Add these two lines to the file:

```
source /opt/cds/class/cds_setup
source /opt/cds/class/setup_files/vhdl/.vhdl_setup
```

 - Save and close the editor.
7. Open a new terminal (by right click on the desktop and choose “Open Terminal”) and type the commands:
 - cd \$HOME
 - source .cshrc
8. Create new directory, name it cadence, under you home directory.
 - mkdir cadence
9. Create vhdl directory under cadence directory.
 - mkdir vhdl
10. Execute the following commands:
 - cd vhdl

- cp \$NCVHDL/cds.lib \$CDSVHDL
- cp \$NCVHDL/hdl.var \$CDSVHDL

11. Open cadence/vhdl/cds.lib file and add the following line

```
DEFINE NCSU_TechLib_ami06 /opt/cds/class/local/lib/oa/NCSU_TechLib_ami06
```

The cadence/VHDL/cds.lib file will look like

```
include $CDS_INST_DIR/tools/inca/files/cds.lib
#DEFINE ieee /opt/cds/ldv/tools/inca/files/IEEE
#DEFINE std /opt/cds/ldv/tools/inca/files/STD
DEFINE vhdl ~/cadence/vhdl
DEFINE NCSU_TechLib_ami06 /opt/cds/class/local/lib/oa/NCSU_TechLib_ami06
```

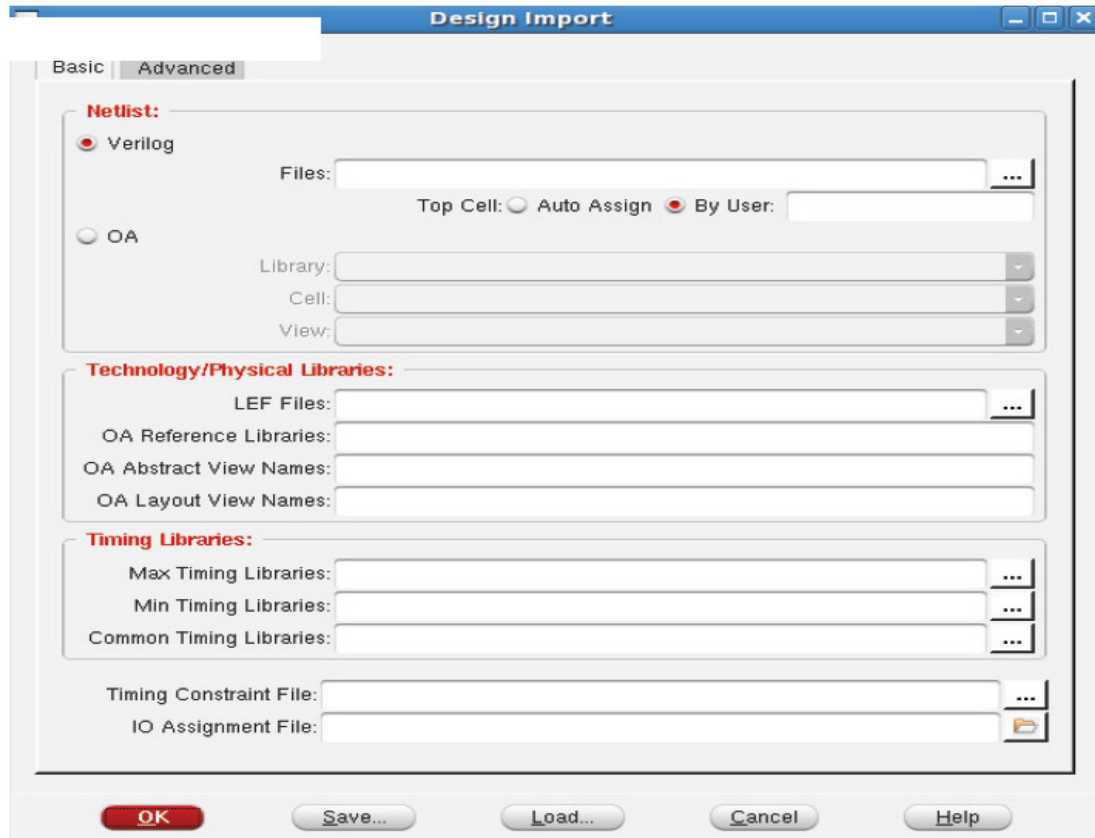
12. Encounter setup: Type these commands in you terminal

- cd \$CDSVHDL
- mkdir fe
- cd fe
- cp \$DSMSE/ece753.conf ece753.conf

PART 1: ENCOUNTER

The original procedure version was contributed from Dr. Singh VLSI Lab and this is update to current procedure:

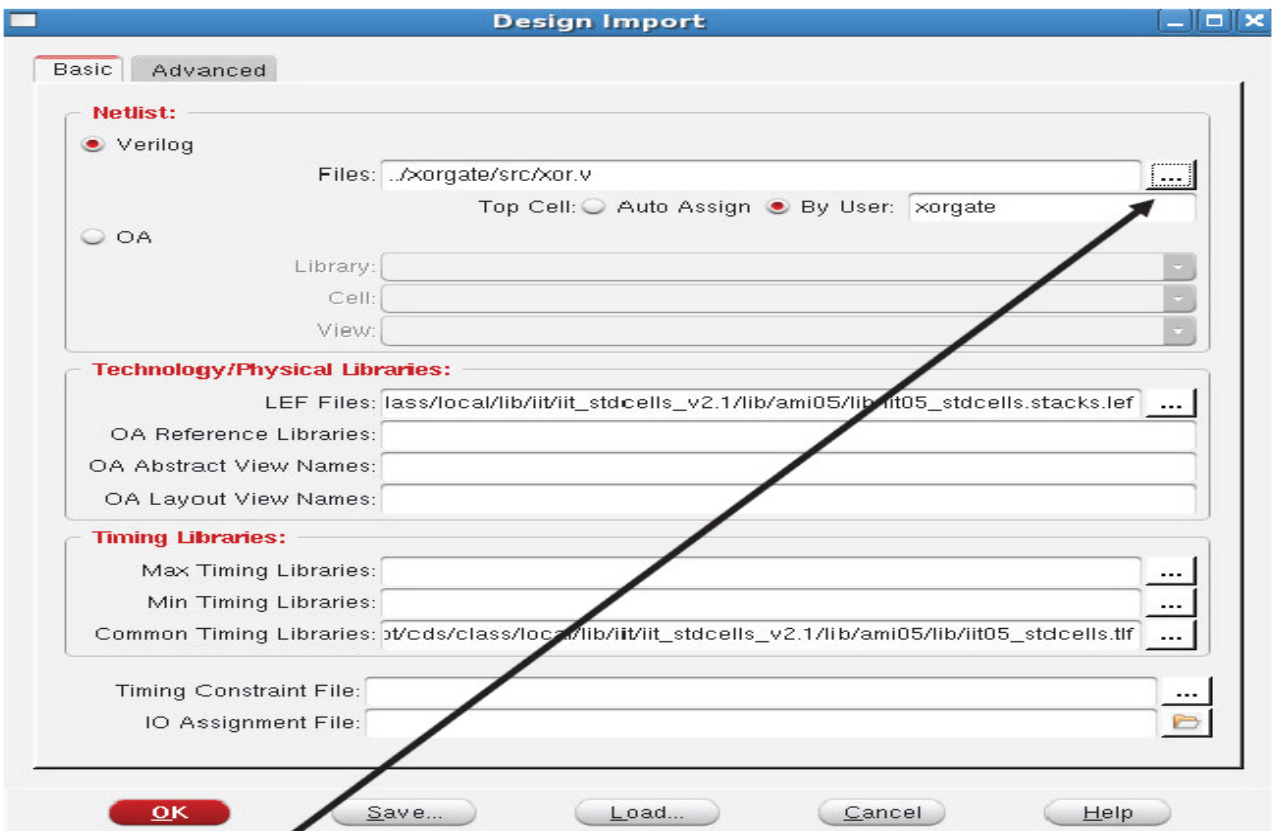
- 1) Open Unix terminal and type **encounter**
- 2) Click on **File** (Top Left) -> **Import Design** The following screen should open.



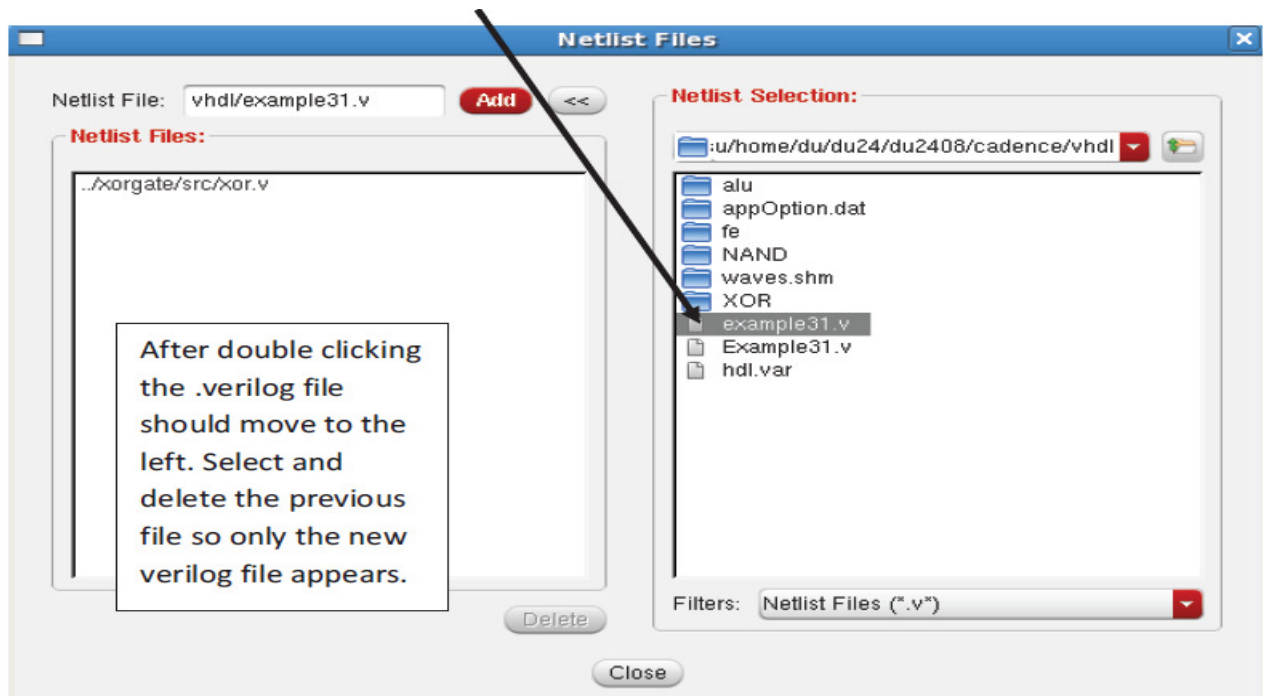
- 3) Click on **Load...**
Navigate to cadence/vhdl/fe and select ece753.conf file



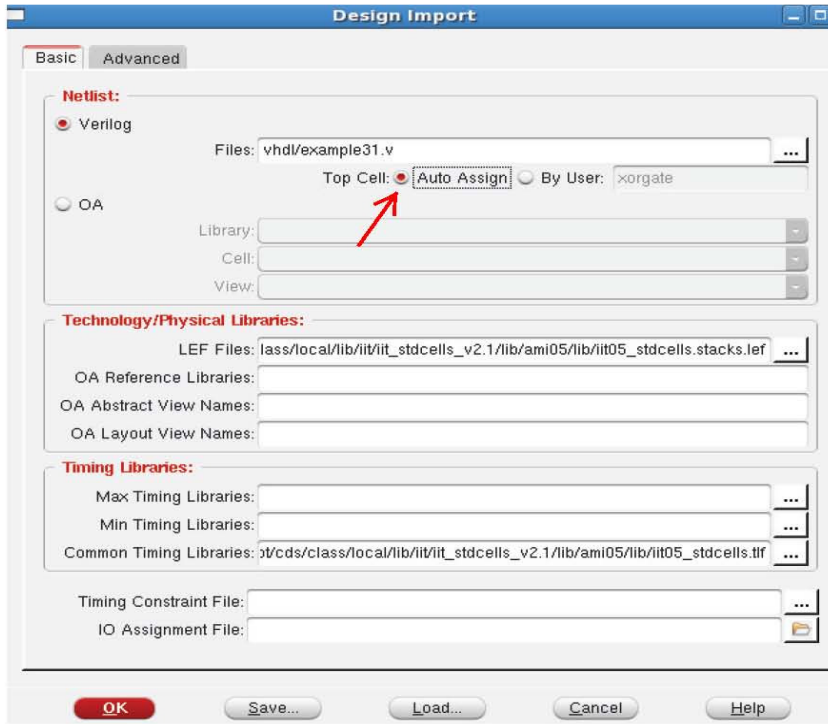
- 4)



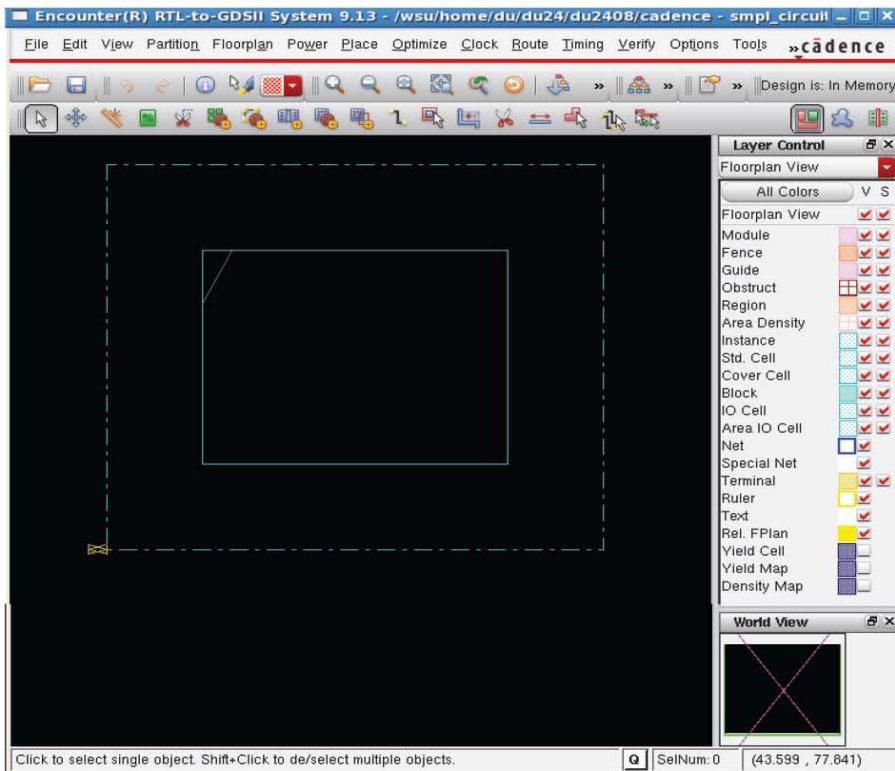
- 5) Click on ... (top right of the screen above)
- 6) **Double Click** the designated **.verilog** file which was created using BGX commands



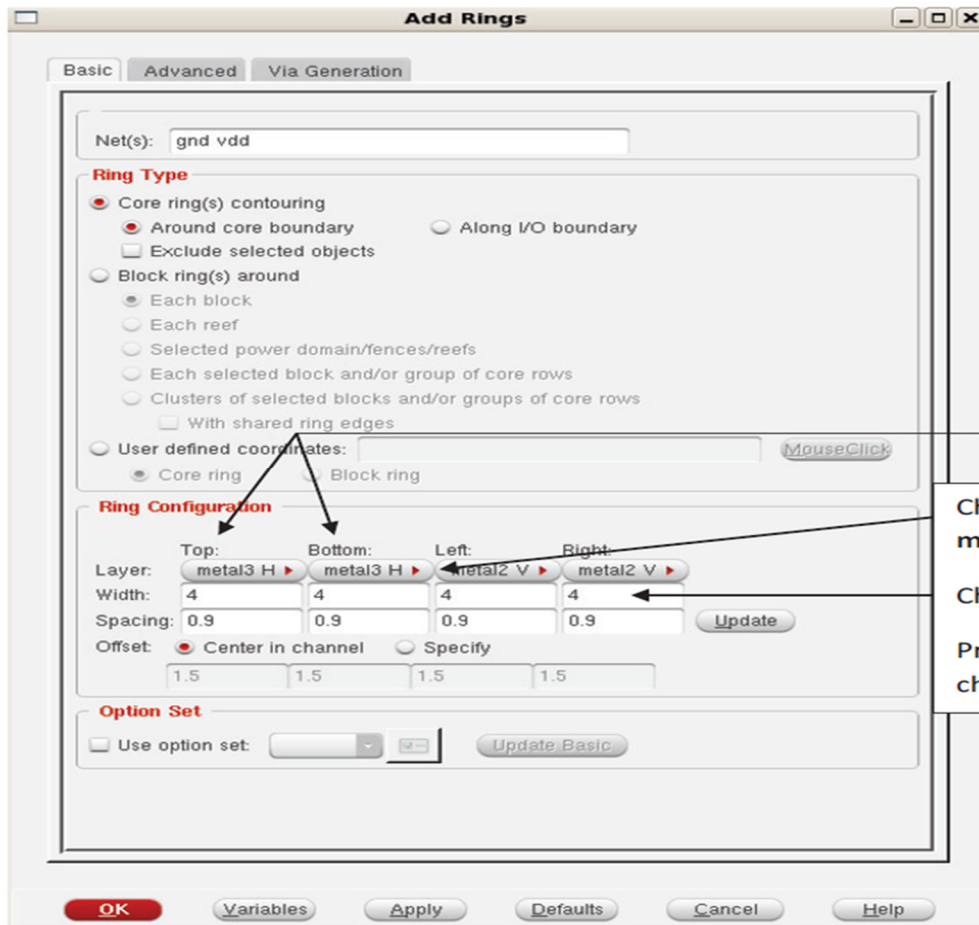
- 7) Press OK after selecting the desired .verilog file



8)

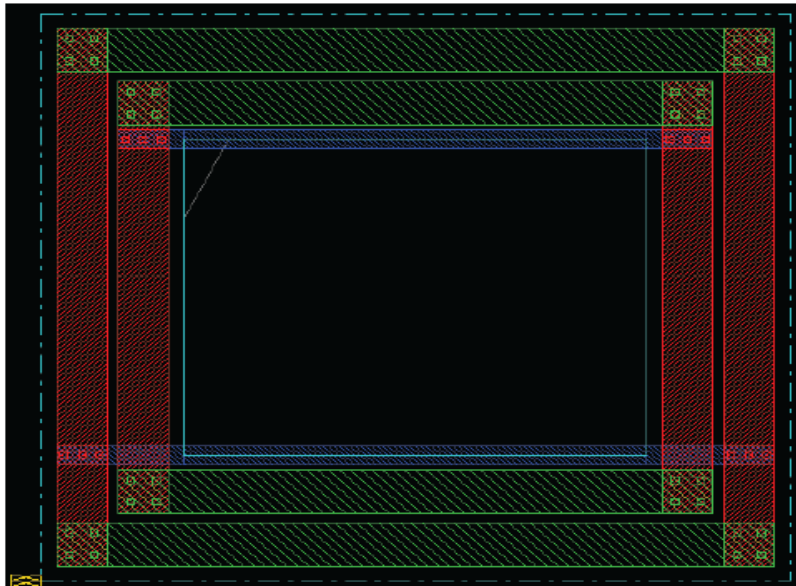


- 9) Click on **Floorplan -> Specify Floorplan -> Ok** (Just Press Ok, no need to change anything)
- 10) Click on **Power -> Power Planning -> Add Ring**



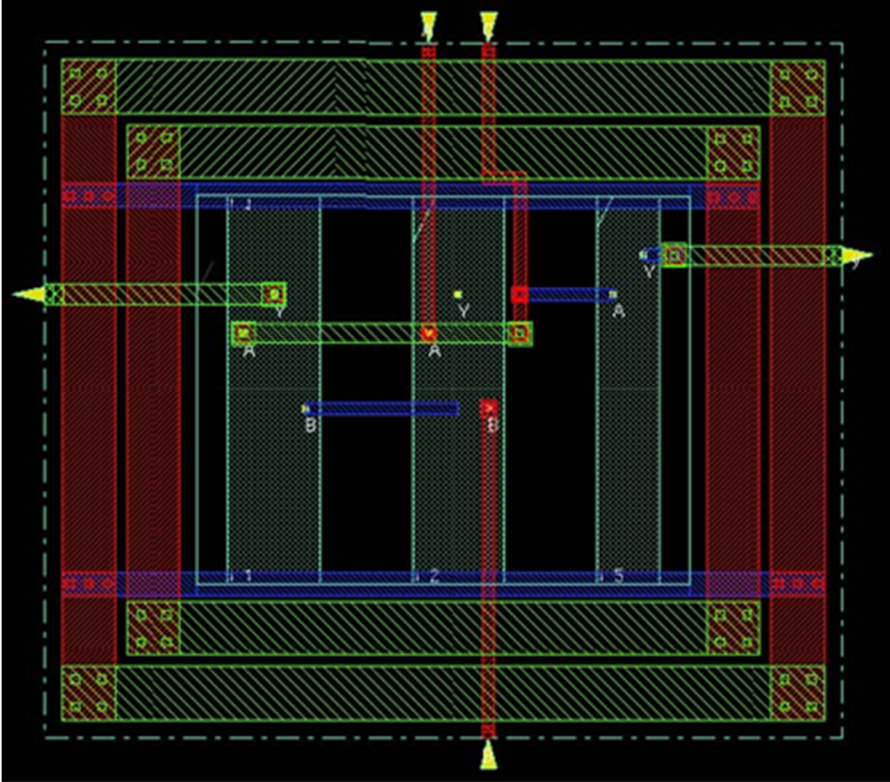
Change Top: and Bottom: to metal3H
 Change all Width: to 4
 Press Ok after making the changes

11)

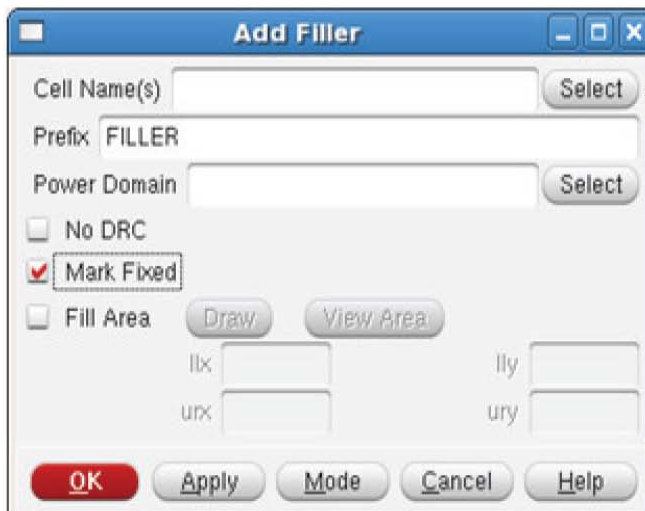


A figure similar to this should appear without the blue line

- 12) Click **Route -> Special Route -> Ok** (The blue line should appear now)
- 13) Click **Place -> Place Jtag -> OK**
- 14) Click **Place -> Place Standard Cells -> OK**
- 15) Click **Route -> Nanoroute -> Route -> Ok**
- 16) After doing step 15, a figure similar to the one below should



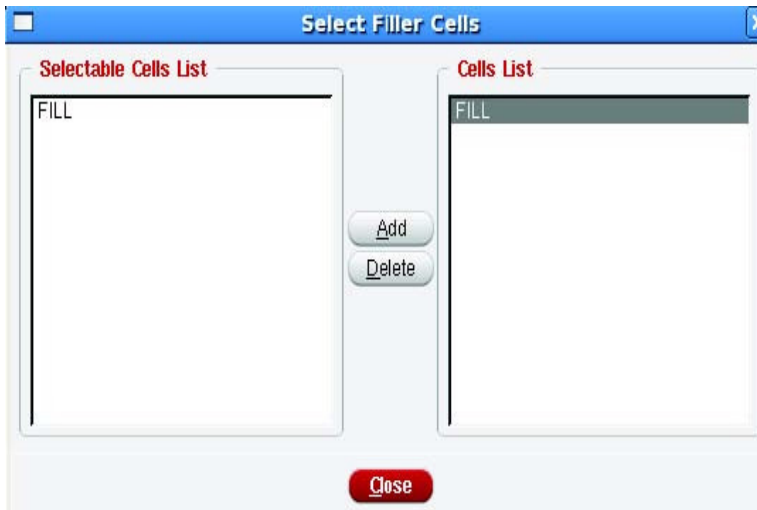
- 17) Click **Place -> Physical Cell -> Add Filler**



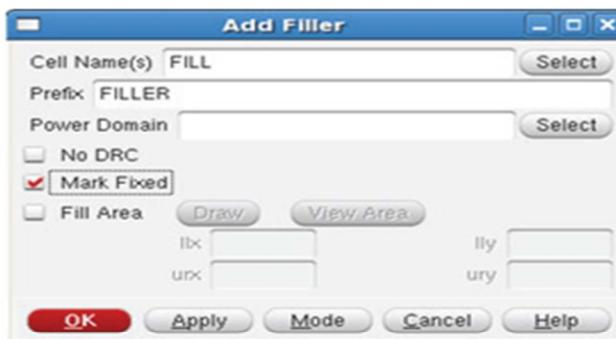
Add Filler screen should pop up

Click Select

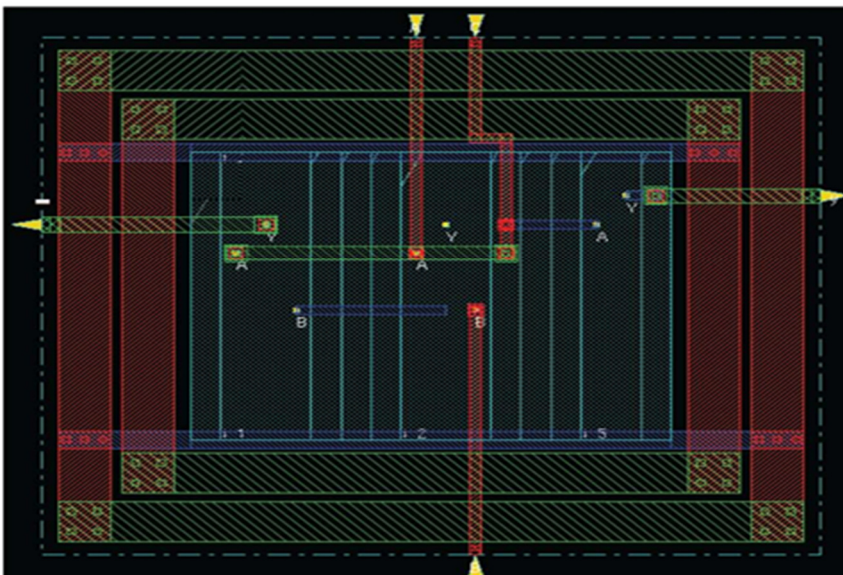
18) Select and Add 'FILL' (Right Side) and Click Close



19) FILL should appear in the Cell Name(s) as shown below. Then Click OK



20)



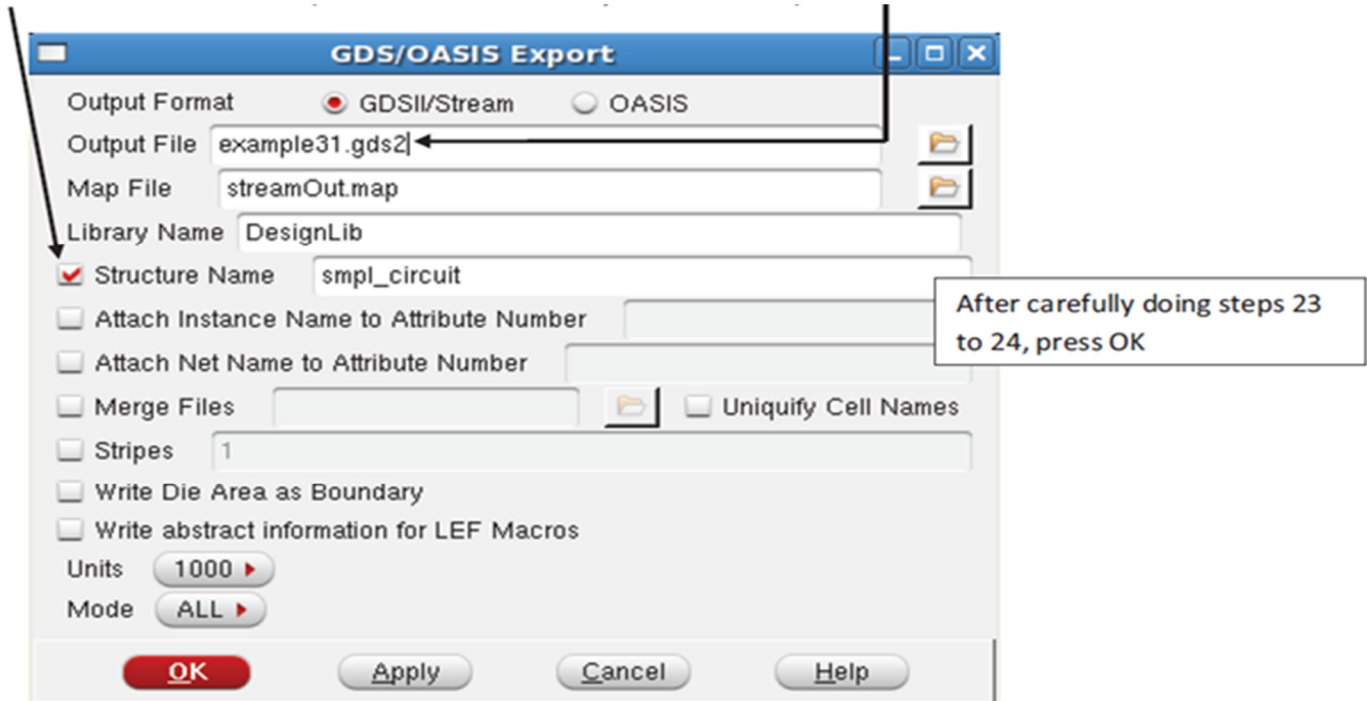
21) Click **Verify-> Verify Connectivity -> Ok** (There is no need to change anything, just press OK)

22) Click **File -> Save ->GDS\OASIS**

(There are two Saves, one on top and other on the bottom. **CLICK THE ONE ON THE BOTTOM!!!**)

23) Enter desired verilog name and **GIVE THE EXTENSION .gds2**

24) Check the Structure Name (Do Not Edit the Already filled in Name)



25) Click **File -> Save -> DEF**



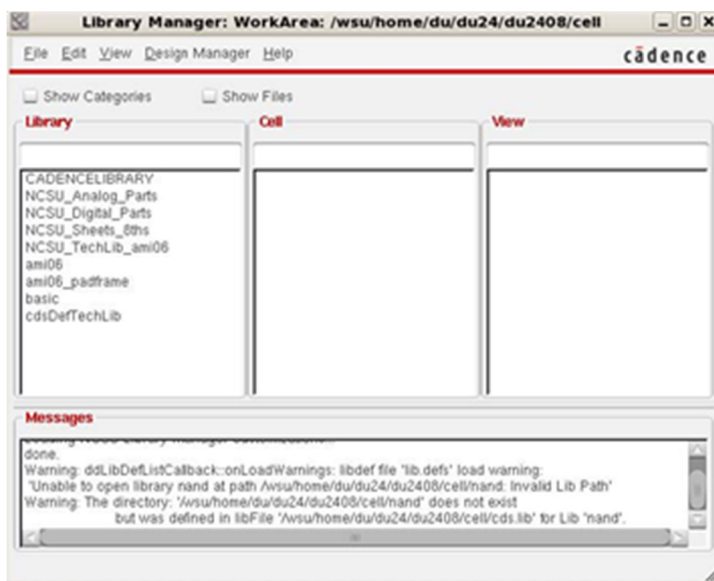
26) In the encounter terminal type the following commands
 a. On your query to determine the area occupied, use command

==>queryPlaceDensity

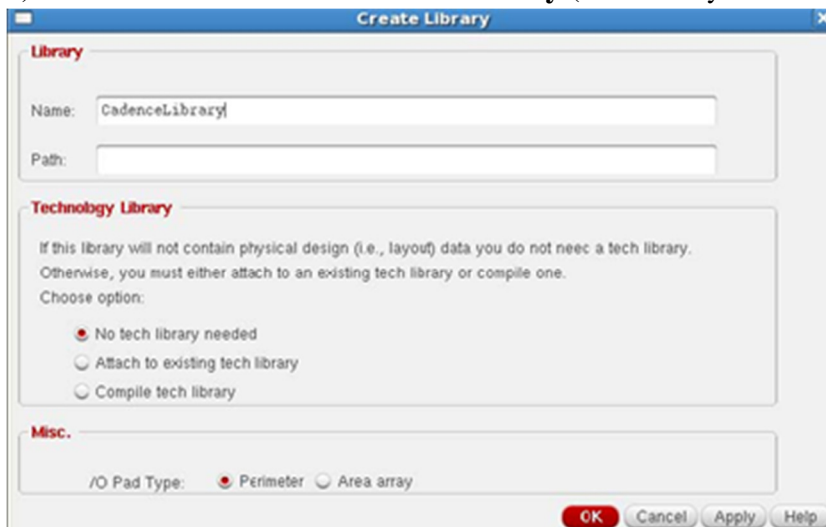
- b. For determining number of instances do,
 ==>selectInst *
 ==>length [dbGet selected]
- c. For Calculating delay use
 ==> report_timing -from input_pin_name -to output_pin_name -unconstrained
- d. To view the schematic from GUI click on ==>tools -> schematic viewer
- e. To get the Aspect Ratio use command:
 ==>dbHeadAspectRatio
- f. To get the coordinate of selected box use :
 selectInst instance_name
 dbGet selected.box
- g. To Get the x and y dimensions of a particular cell,use the command after importing the design:
 set a [dbGetInstByName instance_name]
 set b [dbInstCell \$a]
 dbCellDim \$b
- h. To Get the voltage for the specified cell
 Set a [dbGetCellByName cell_name]
 dbCellVoltage \$a

PART 2: VIRTUOSO

- 1) Open Virtuoso
- 2) Close all the screens except LIBRARY MANAGER and the LOG window

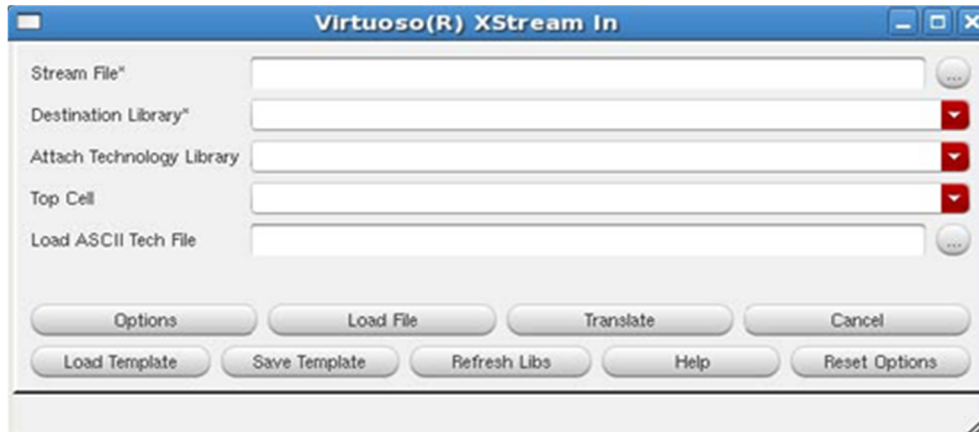


- 3) Click **File->New->Create Library** (The Library Window, not the Log window)

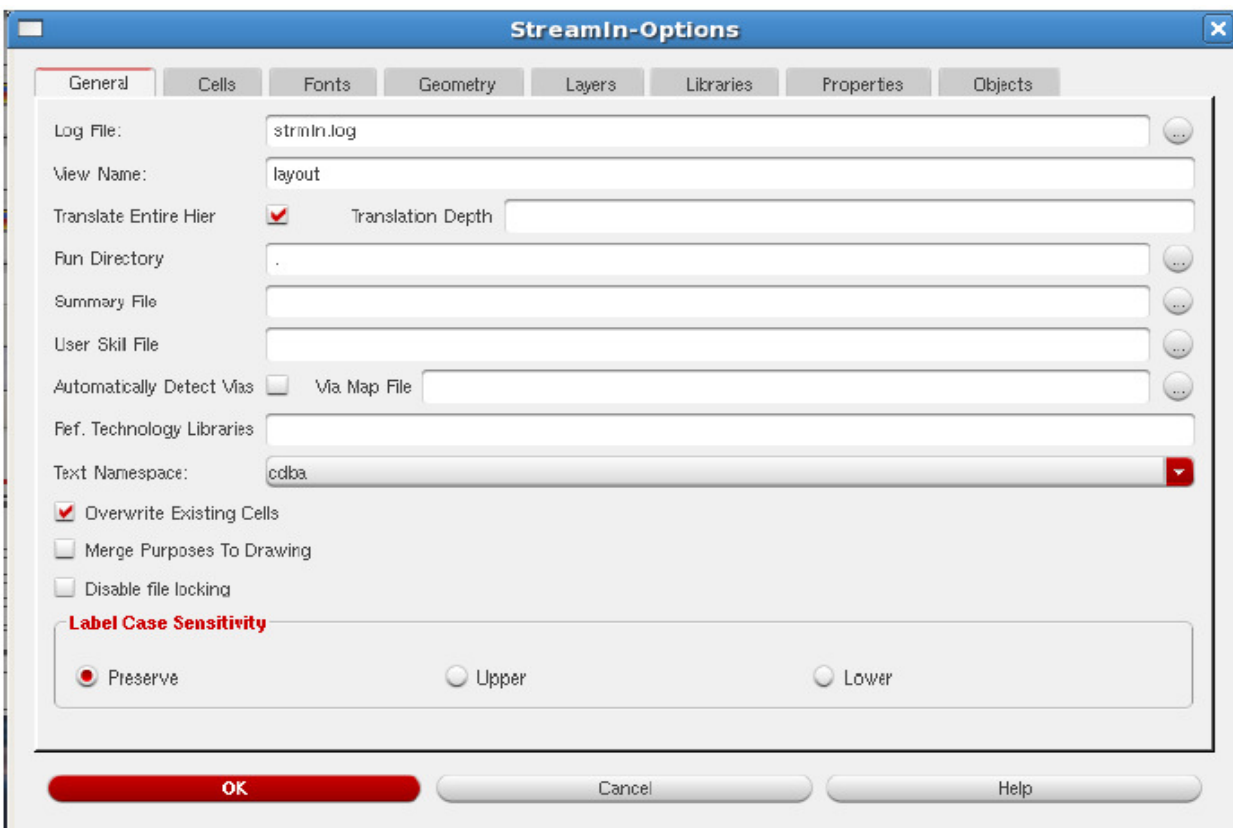


- 4) Click OK.

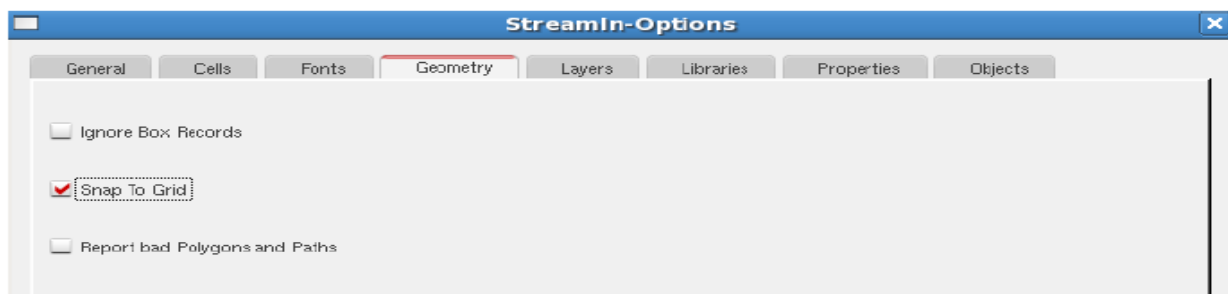
- 5) **Click File -> Import -> Stream** (The LOG window, not the Library manager)



- 6) **Click on Options then click on Geometry**



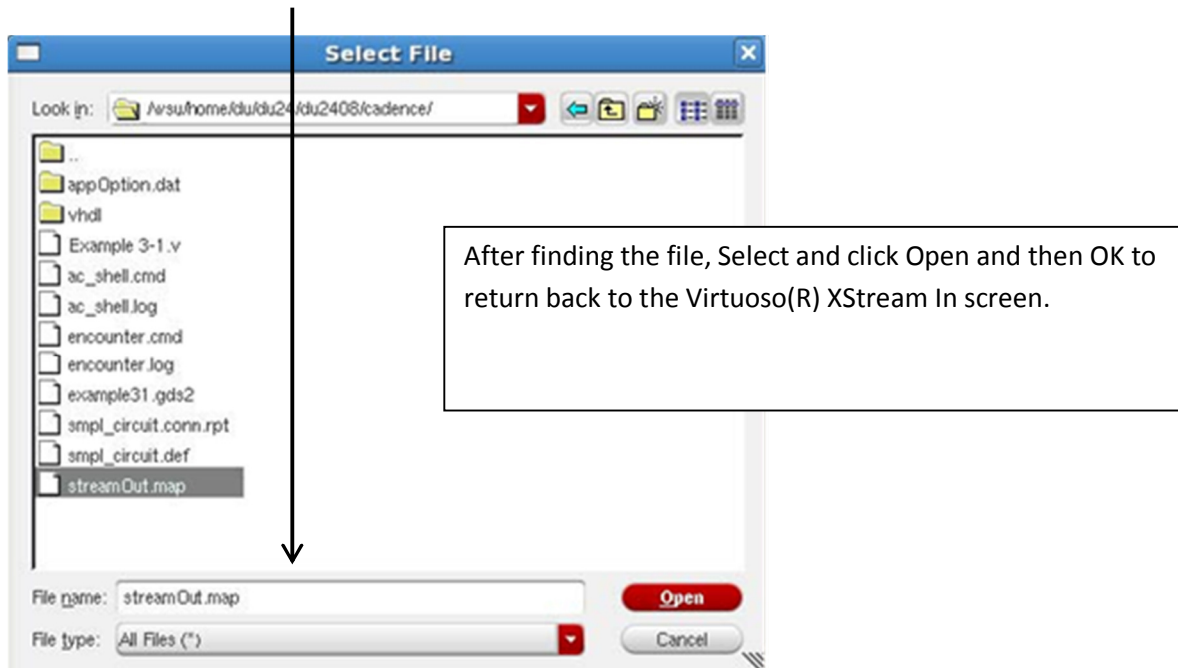
- 7) **Select Snap To Grid**



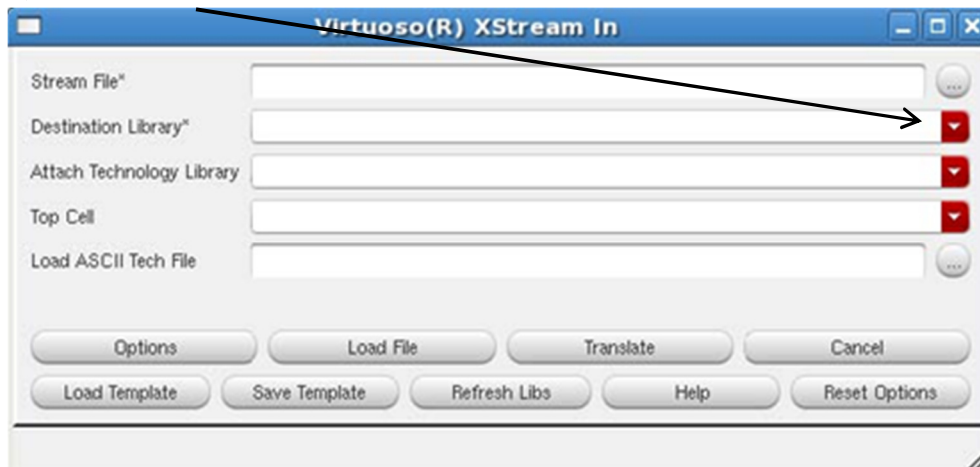
8) Click Layers -> Load File



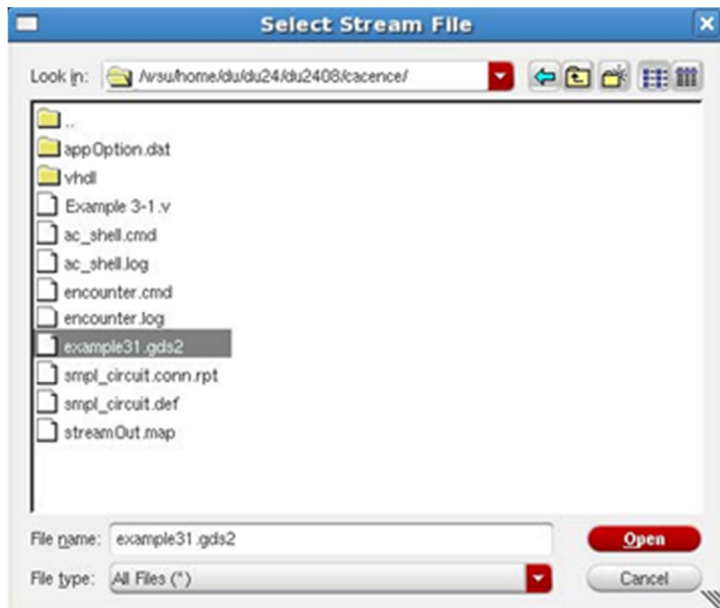
9) Find the streamOut.map file in your cadence folder. (You might have to look around more)



10) Click on ...

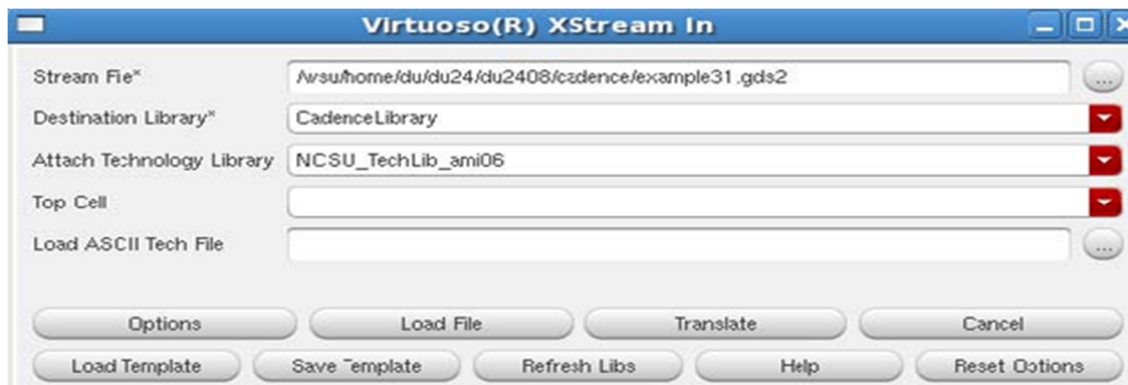


11) Find, Select and Open the .gds2 file that was saved in PART 1, STEP 23.



12) Select the Destination Library (The new library created in step 3)

13) Select the NCSU_TechLib_ami06 for your Technology Library Attachment



14) **Click Translate.** (A warning Log file will open, Just press No)

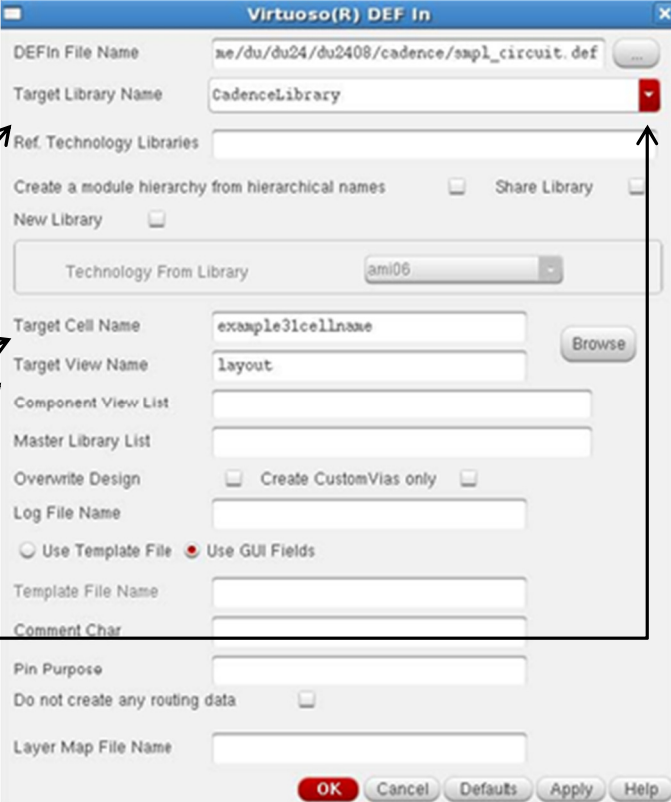
15) **Click File->Import->DEF**

Select Target Library Name (Same one you created before)

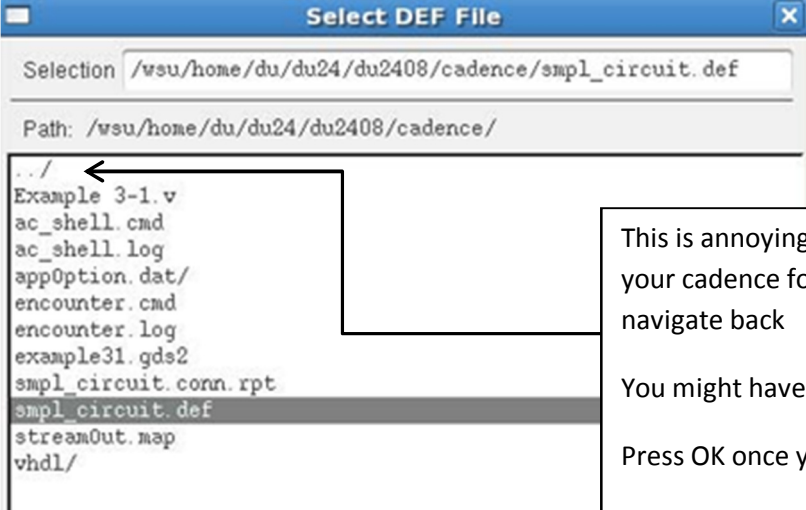
Fill Target Cell Name as shown (add 'cellname' in the end)

Fill Target View Name as shown (just add 'layout')

Finally Click on ... (Top Right Corner)



16) Find the .def file in the cadence folder

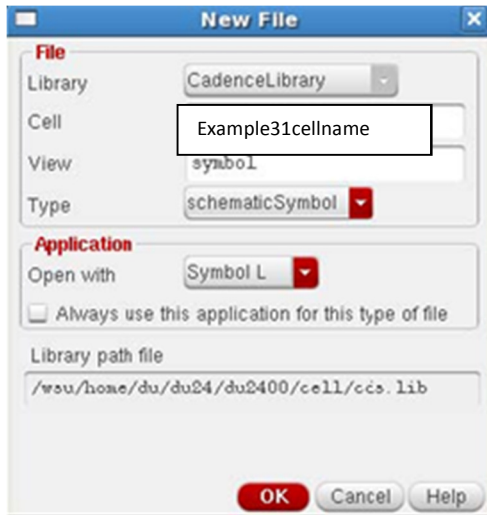


This is annoying to use, but you need to browse back to your cadence folder and find the .def file. Click ../ to navigate back

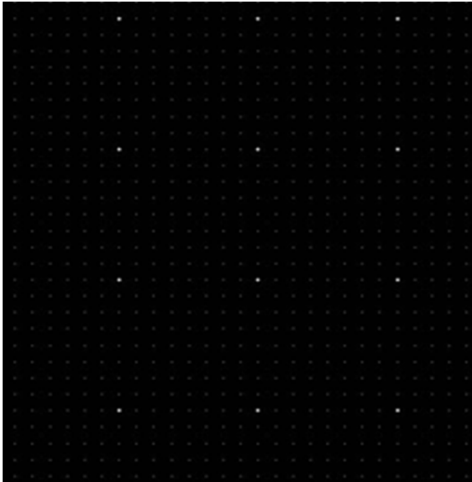
You might have to look around.

Press OK once you find the .def file

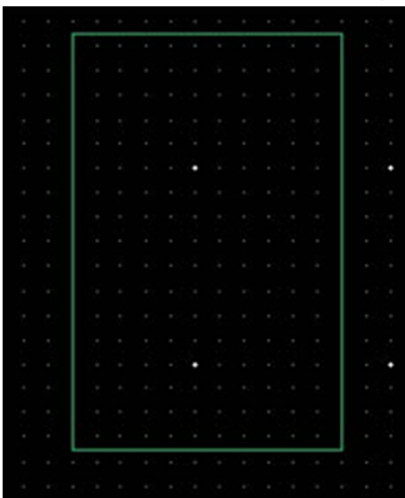
17) Click File->New->Cell View



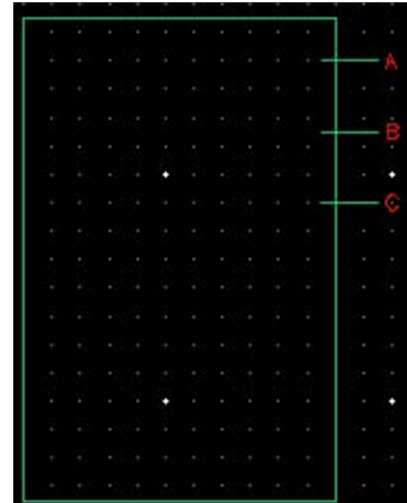
18)



19) Click Create and draw rectangle



20) Click Create->Add Pins



Add the Pin Names.

Insert Space in between.

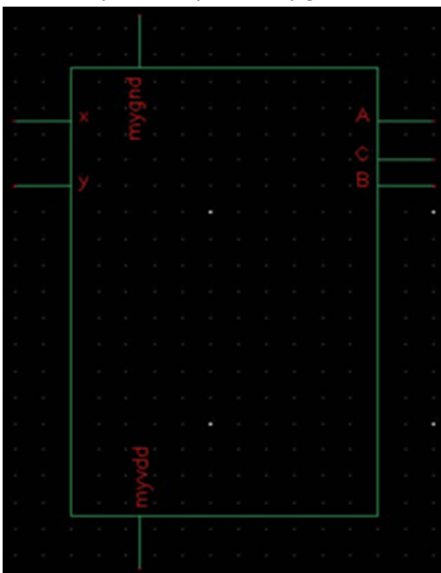
Make sure the Direction is correct (input/output)

Then go to the Black screen and just click away the pins. As you place each pin, it should automatically go to the next entered pin.

The red ending is pointing out.

Do the same for your outputs.

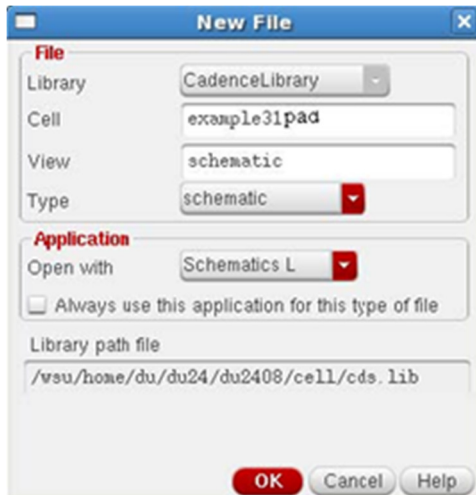
21) Manually insert your 'mygnd' and 'myvdd' as inputs in the symbol as shown below



Click this to save symbol and then close



22) Click File->New->Cell View



Fill Cell (Just add 'Pad' in the end)

Fill View (Just add schematic)

Select Type (schematic)

Press OK

23)

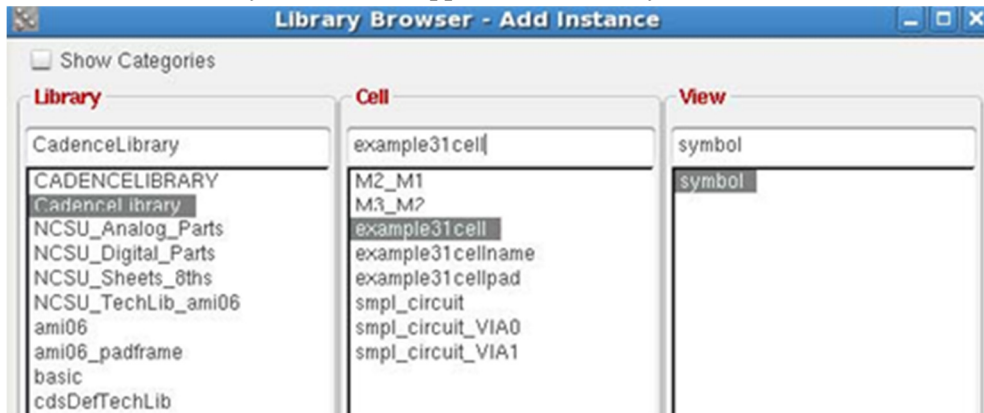


A blank black screen should open again.

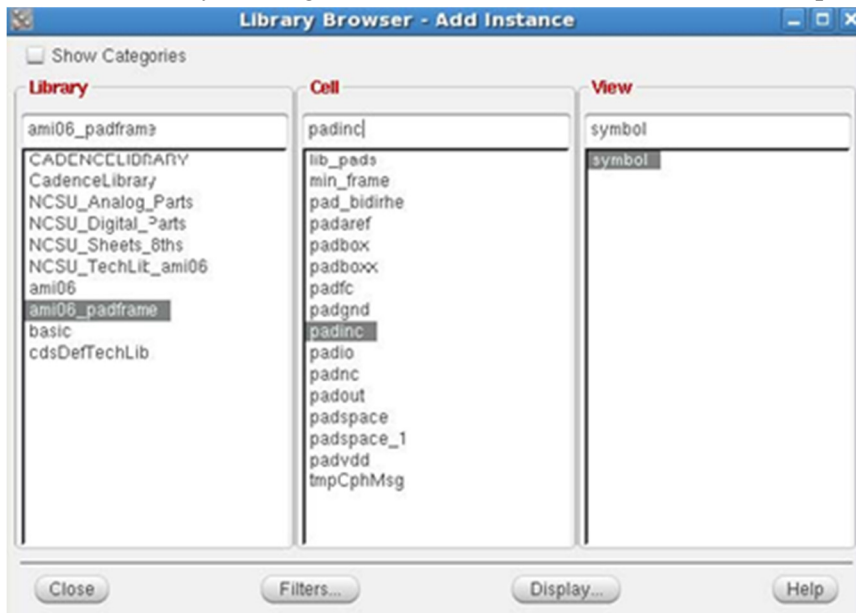
Press I on the keyboard. (Figure on the left should pop up)

Click Browse and browse for the symbol you created.

- 23) Find, Select and Place the symbol to the black screen. (after selecting the symbol, simply move cursor to black screen, the symbol should appear automatically)



- 24) Press I on the keyboard again and Click Browse and look for ami06_padframe (on the left)



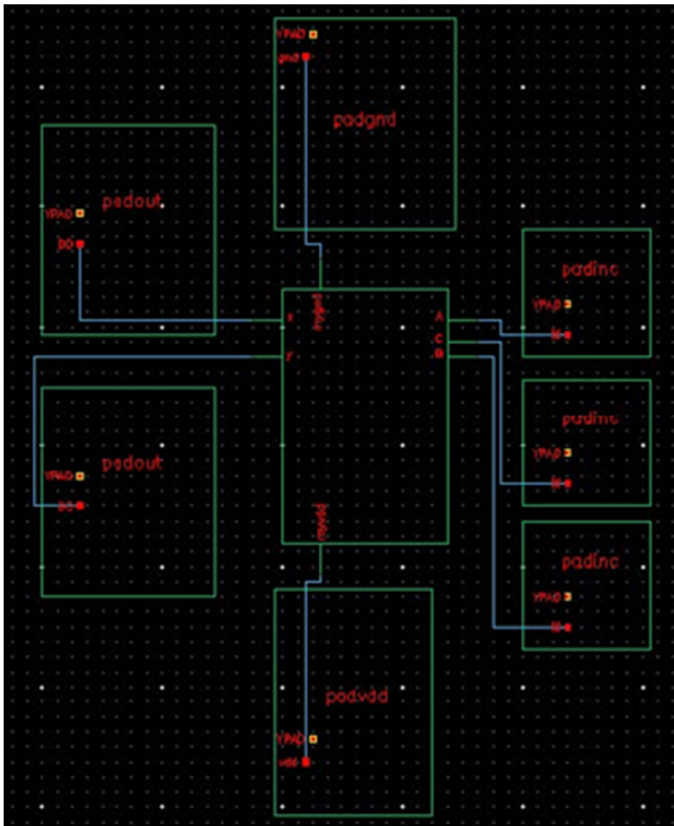
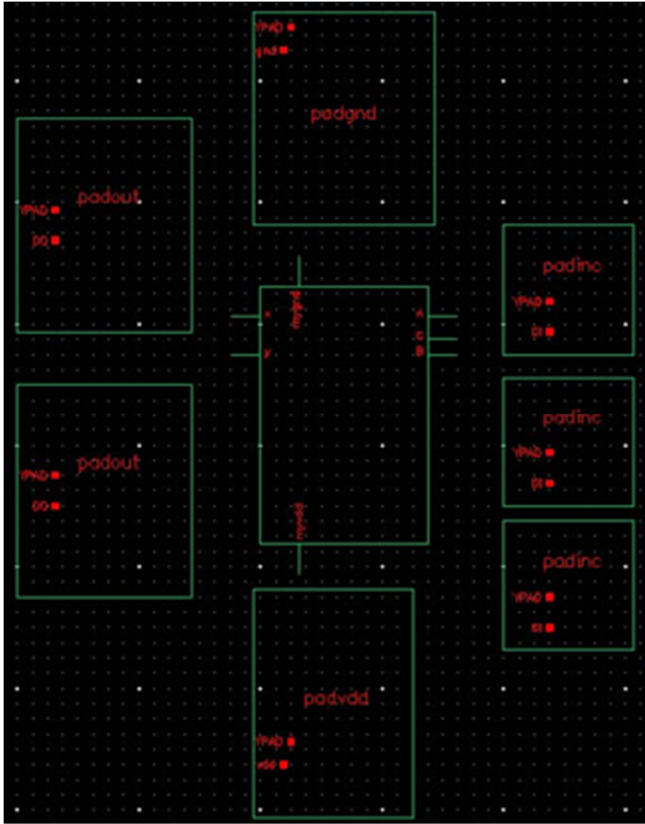
In this part, you will add padding to your circuit.

In the middle column 'Cell', select and place padinc to the black screen. Padinc is the input padding corresponding to your input pins. Place a pad for each input.

Do the same for output by using padout.

Do the same for myvdd and mygnd by using padvdd and padgnd respectively.

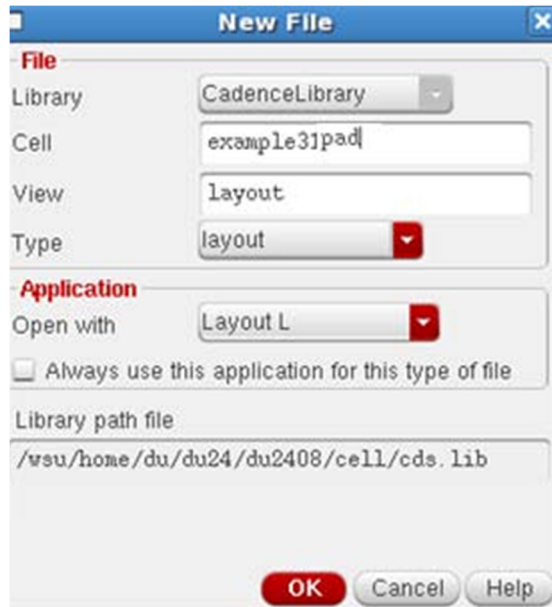
Press W on the keyboard to wire like so in the picture below.



- 25) Click this to save schematic with padding and close.



- 26) Click File->New->Cell View



Fill the same name in the Cell as the one before

Fill View as 'layout'

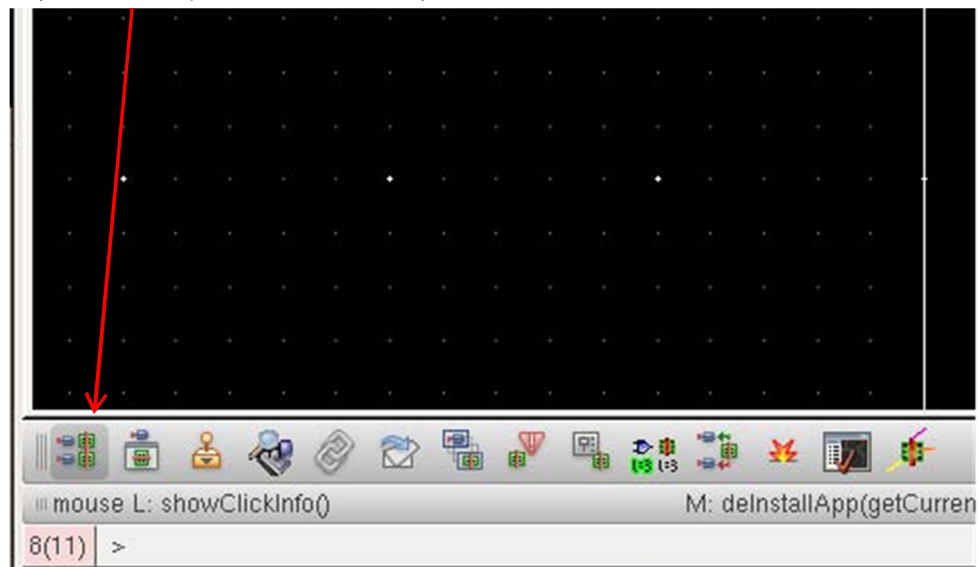
Select Type as 'layout'

Press OK

A black screen should appear again.

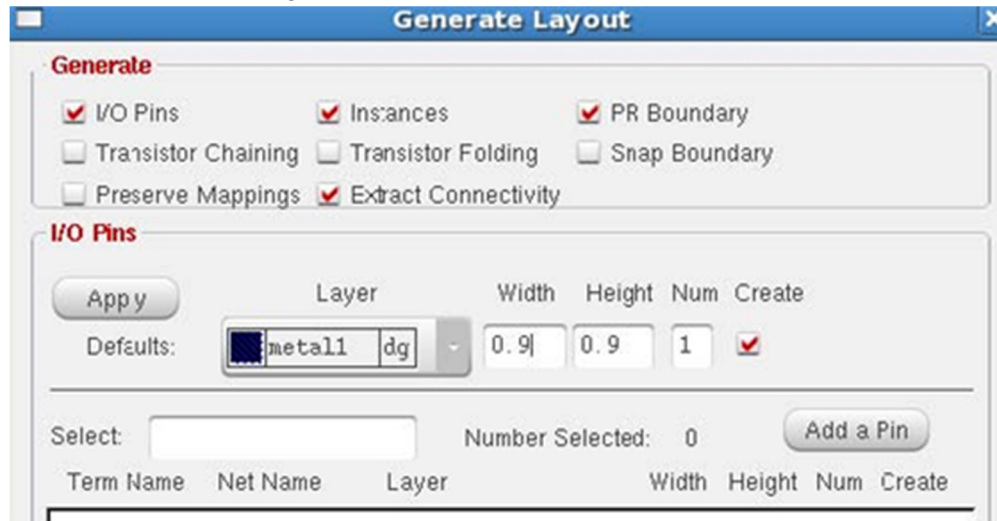
- 28) Click File->Launch XL (A screen with the placed pads should open on the left)

- 29) Click this (bottom left corner)

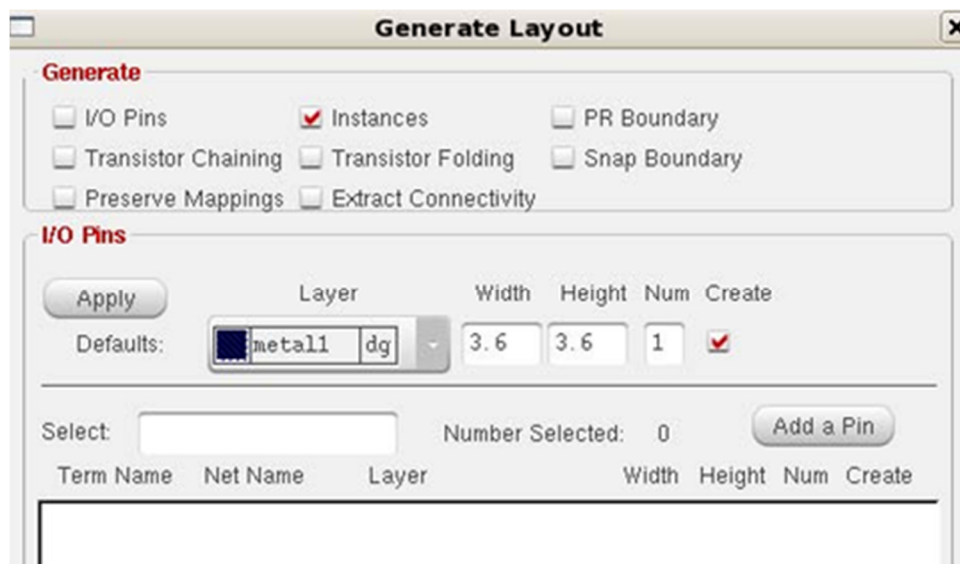


30) This screen should pop up. Deselect everything under Generate except Instances

31) Edit Width and Height to 3.6

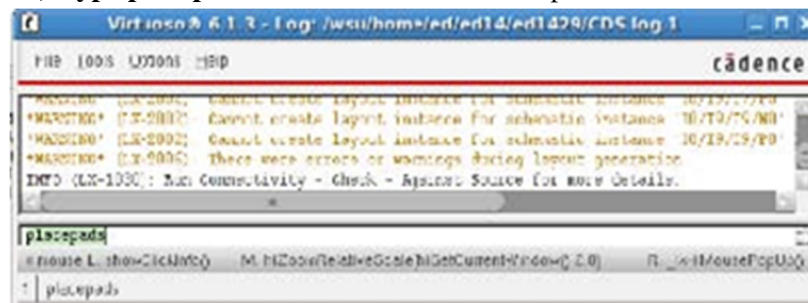


Like this

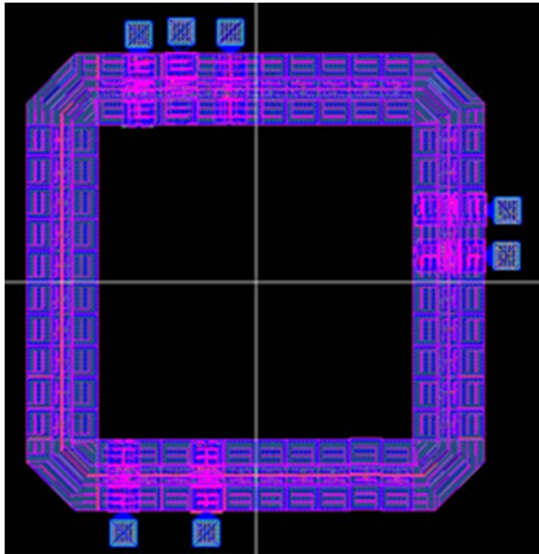


Press OK

32) Type **placepads** in the LOG window and press enter



- 33) Rectangular blocks in red should appear on the screen at this point.
34) Press shift+f together to turn the red figure into



NOTE: each block ONLY MOVES IN ONE DIRECTION. So move it left/right first then move it top/bottom or the other way around.

To rotate block, select block and then click VIEW->Rotate and then click anywhere on the screen to initiate the rotate.

SAVE THIS FILE. YOU ARE DONE

FPGA Implementation Procedure Update:

Note: The following procedure was done on Xilinx Version 13. You may see variations between different versions. However, the general procedure for FPGA implementation remains the same, which is:

- 1) Create a project, add existing source.
- 2) Synthesize code by double clicking on View RTL Schematic.
- 3) Assign Package Pins, under 'User Constraints'.
- 4) Generate programming file to generate .bit file
- 5) Configure DeviceBoundary Scan FPGA Implementation ... selecting .bit file for your device.... Download program to the board.
- 6) Testing code on the board.

The below is a very quick and general overview of the entire process

Procedure for FPGA Implementation:

- 1) Create a new project
File -> New Project
- 2) The following screen will appear. Make sure you specify the correct family name, device name , package name and Preferred language.

New Project Wizard

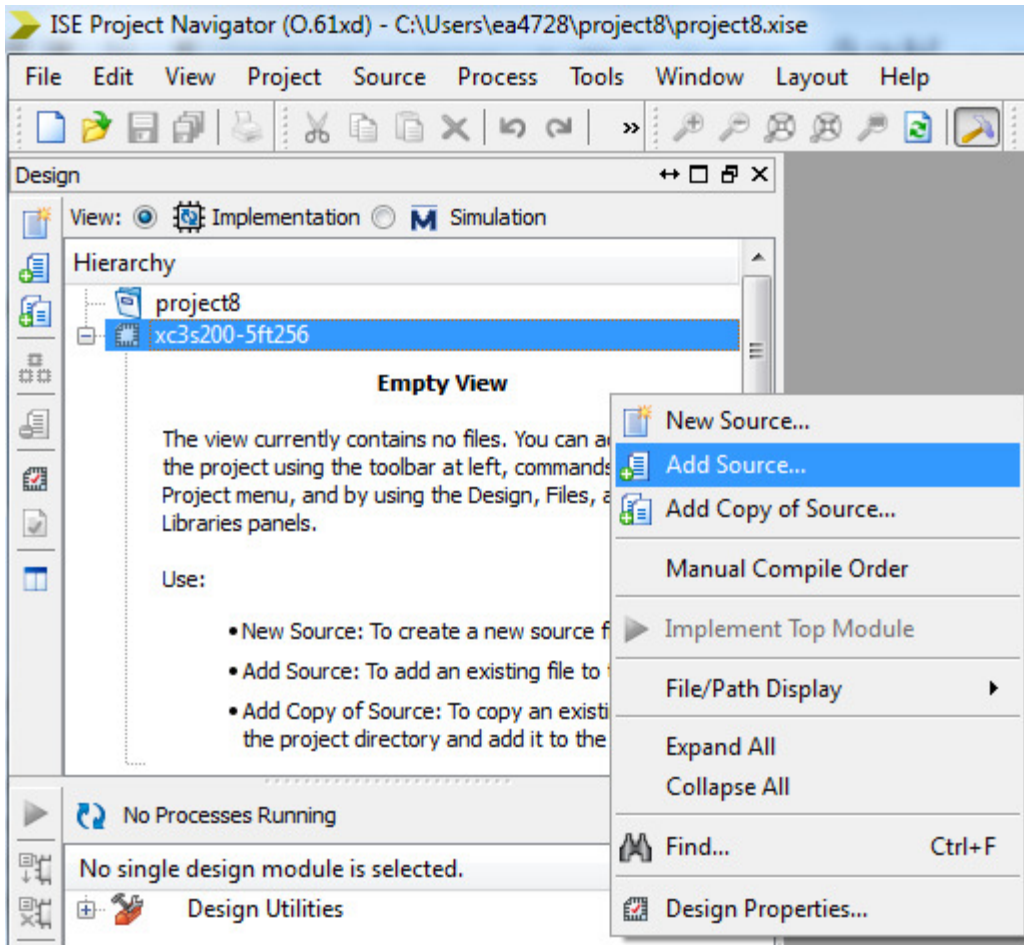
Project Settings

Specify device and project properties.
Select the device and design flow for the project

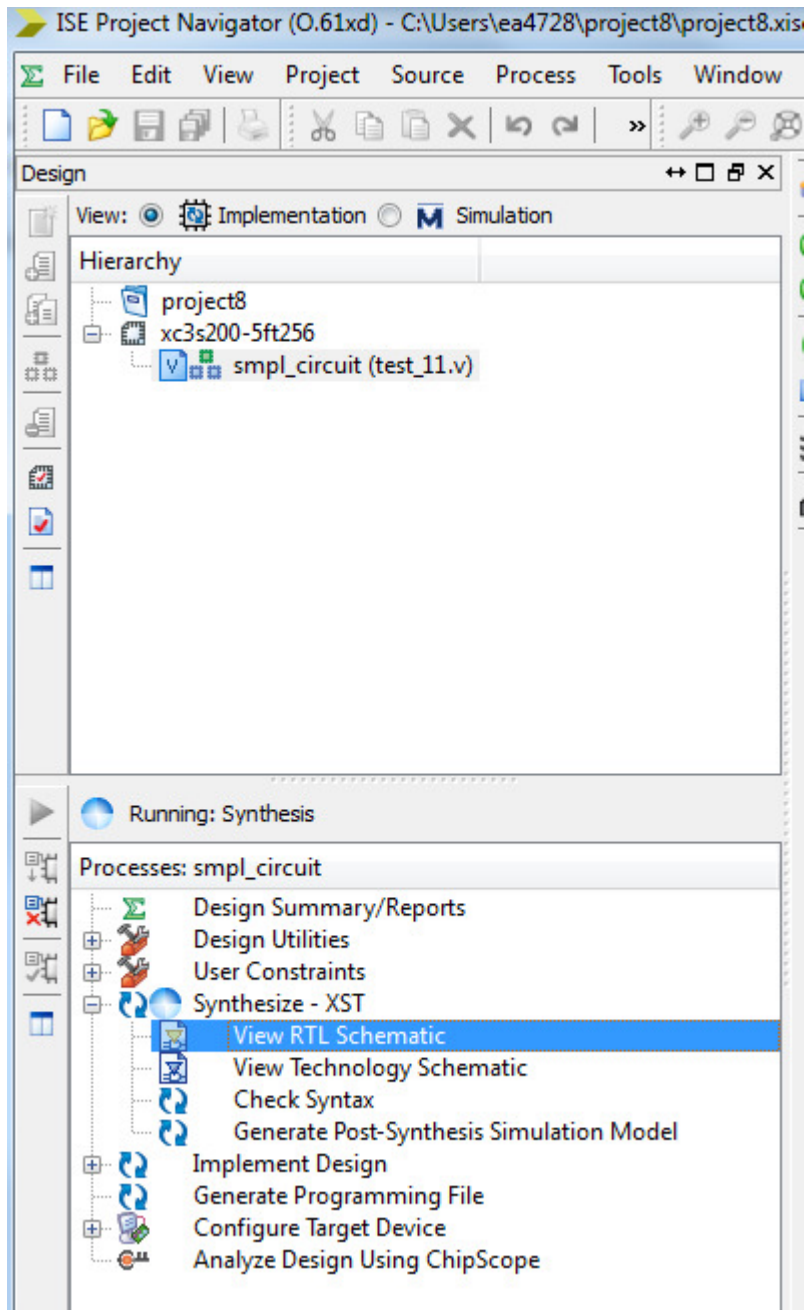
Property Name	Value
Evaluation Development Board	None Specified
Product Category	General Purpose
Family	Spartan3
Device	XC3S200
Package	FT256
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE Verilog
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

More Info **Next** Cancel

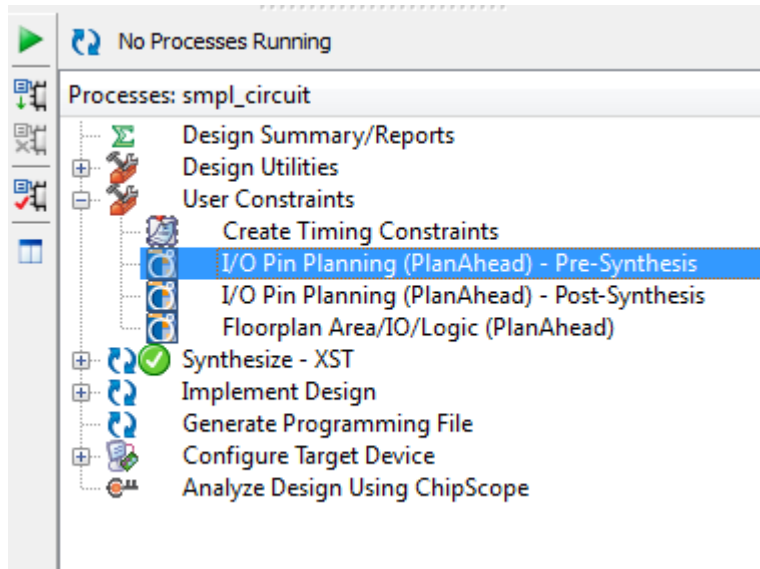
- 3) Click next. Xilinx will show you the project summary. Verify that and click Finish.
- 4) Add an existing Verilog file.
Right click Design window and select 'Add Source'.



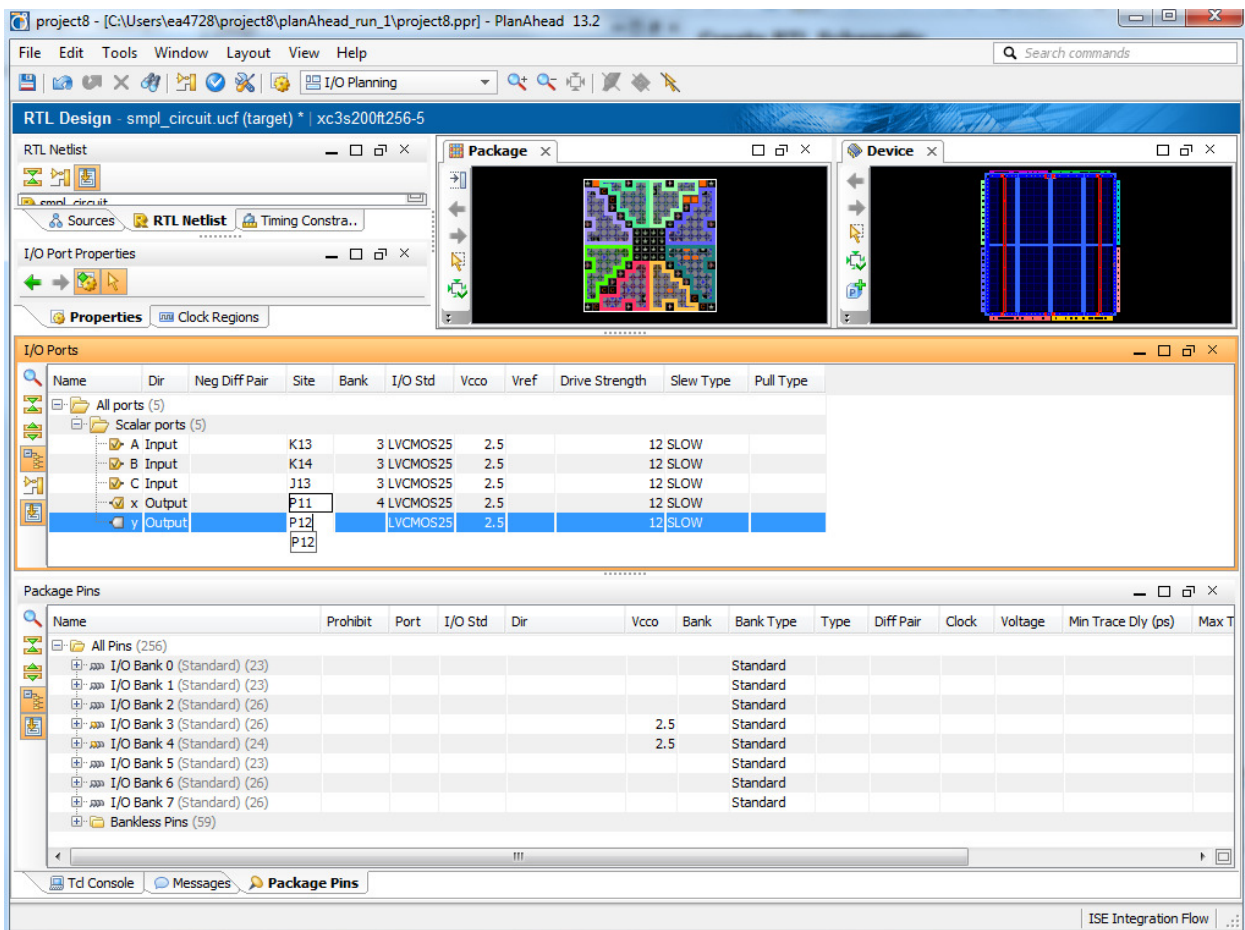
- 5) Xilinx will then prompt you to select Verilog file. Add it.
- 6) Double click on View 'RTL Schematic' under Synthesize – XST



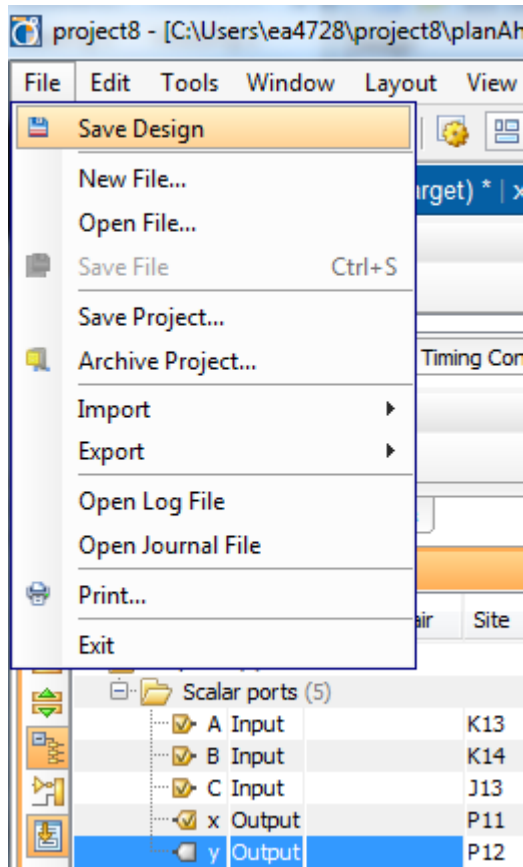
7) Under 'User Constraints', double click 'I/O Planning- Pre-Synthesis'.



- 8) You will then see the following screen. Please note: give Xilinx sometime to open 'Plan Ahead Screen'.
- 9) Then assign pins as you like in 'I/O Ports' wizard.

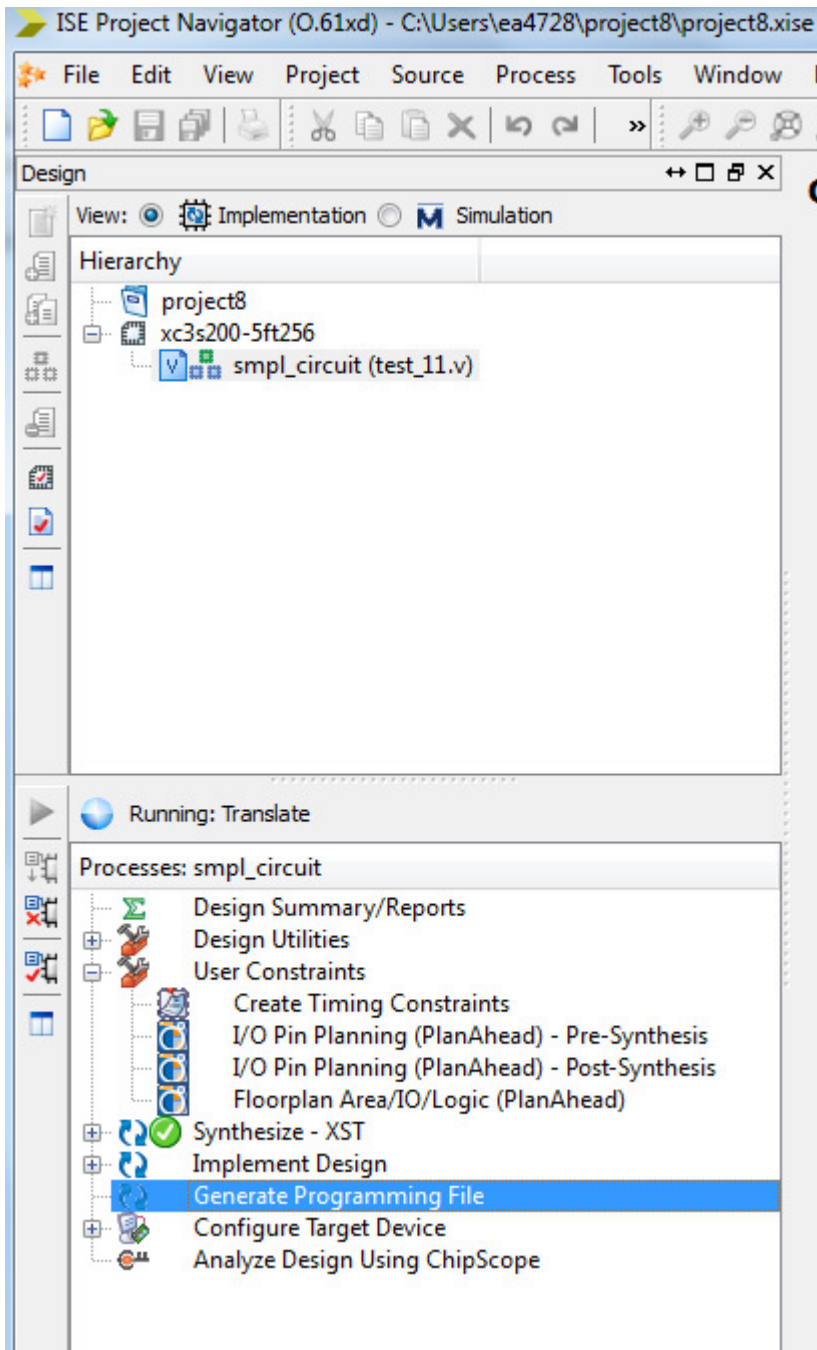


10) Save the design.

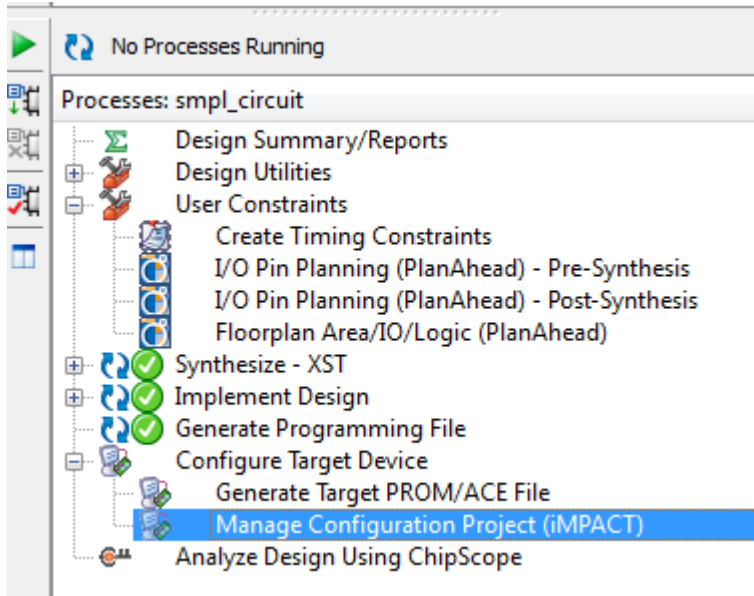


11) Minimize Plan Ahead window and go back to Xilinx.

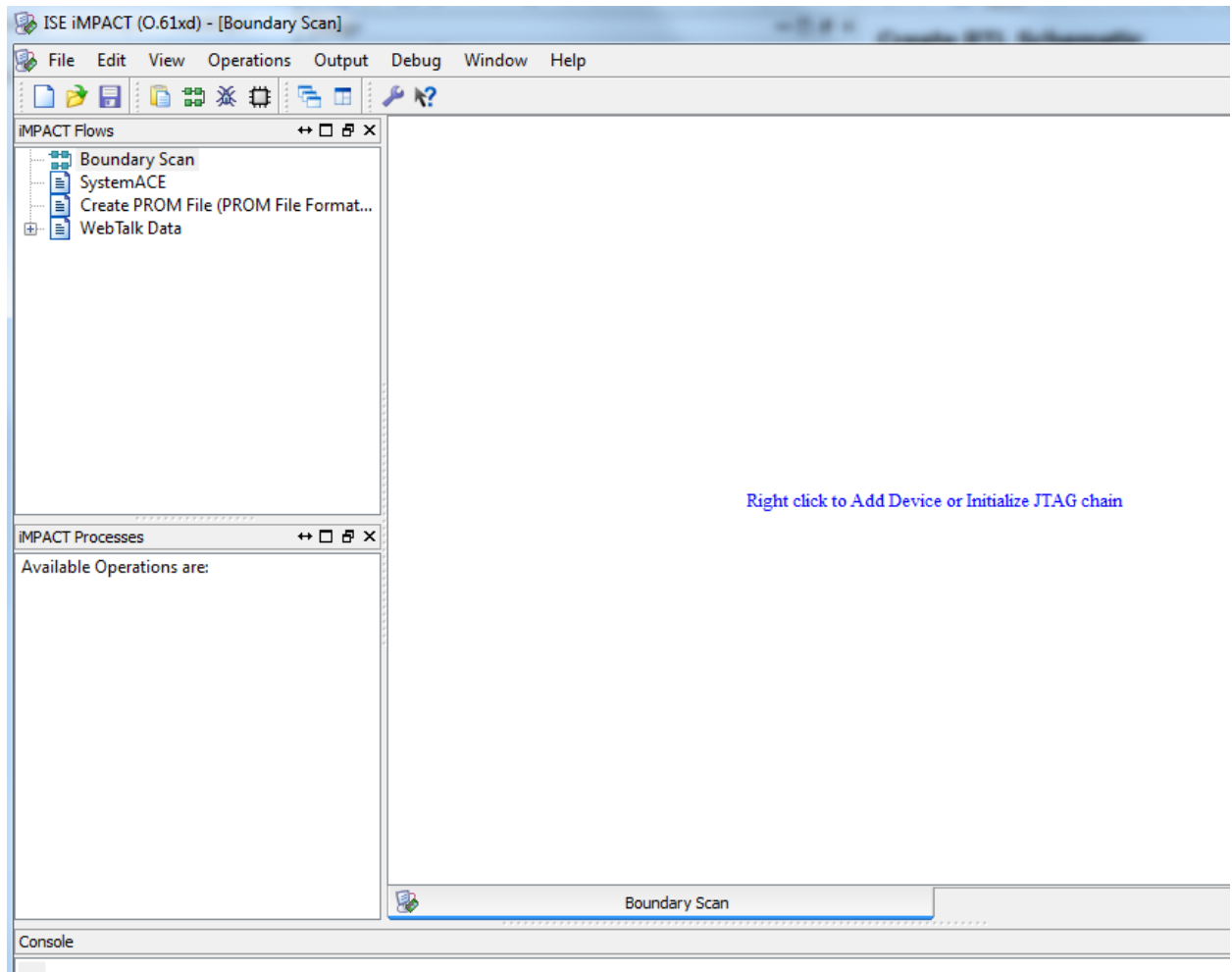
12) Double click on 'Generate Programming File' as shown below. In this step, Xilinx will create a file with extension .bit



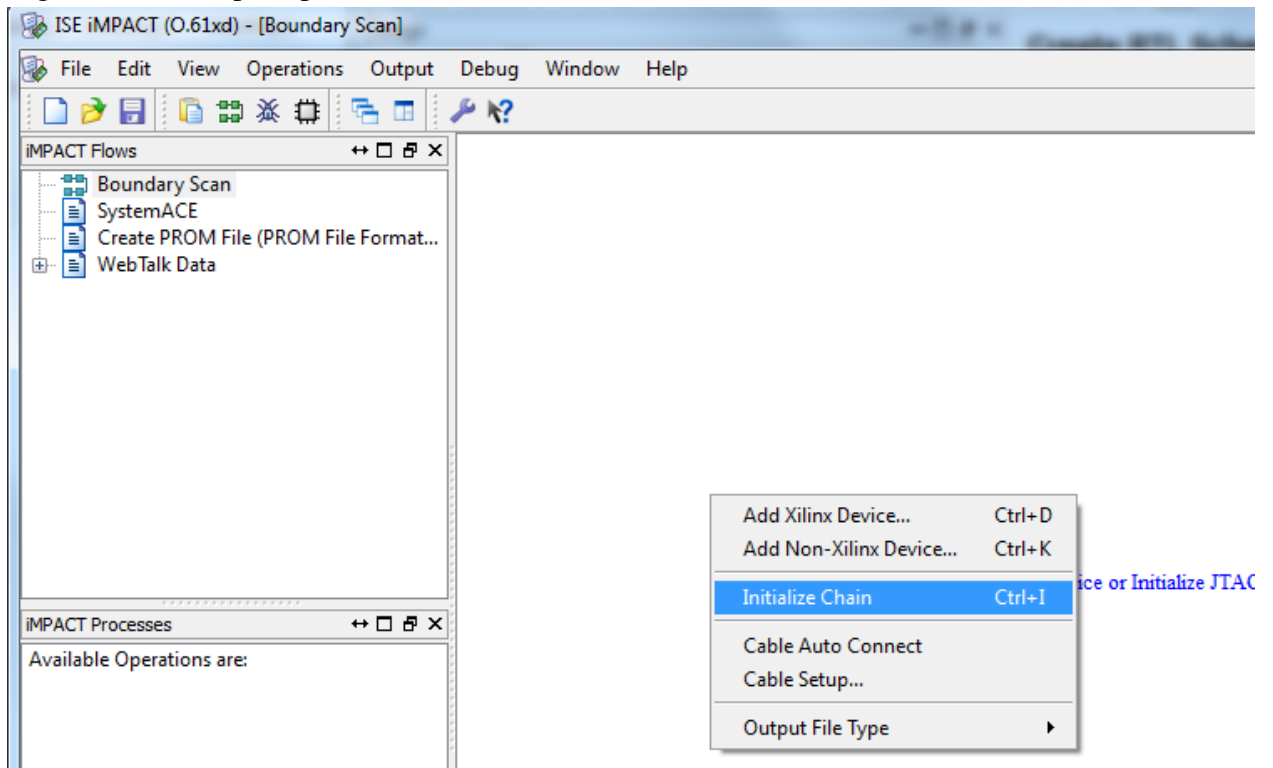
13) Next, expand 'Configure Target Device' and double click on iMPACT.



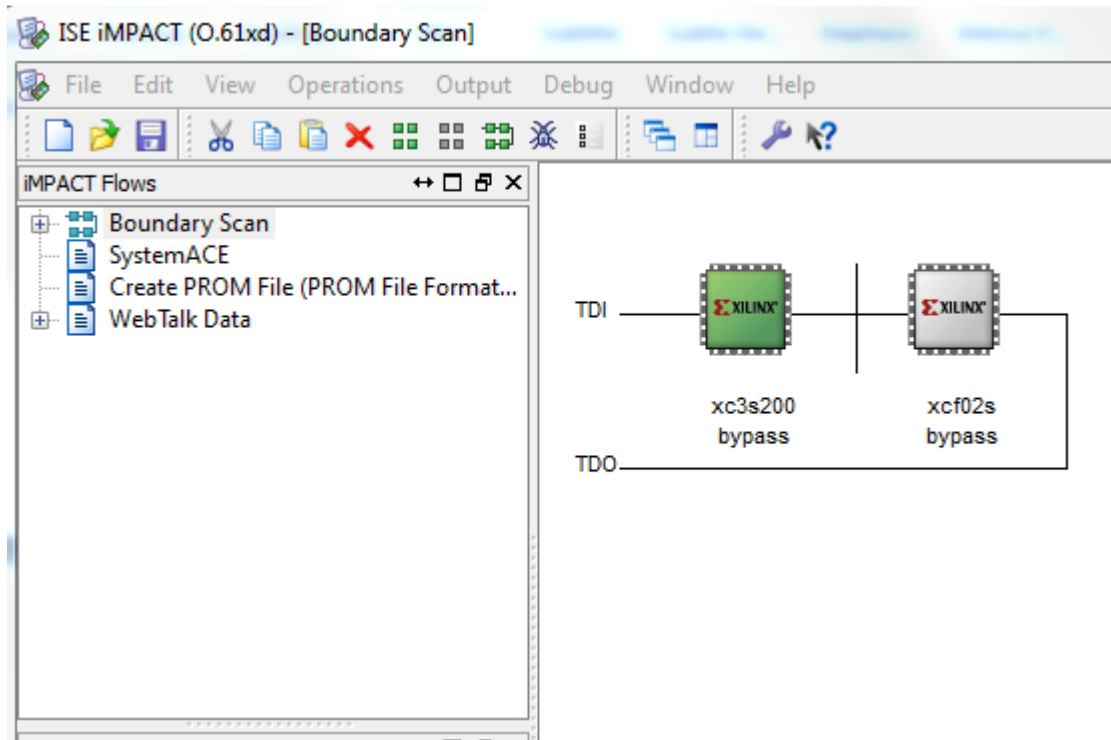
14) You will see the following screen. Select Boundary Scan.



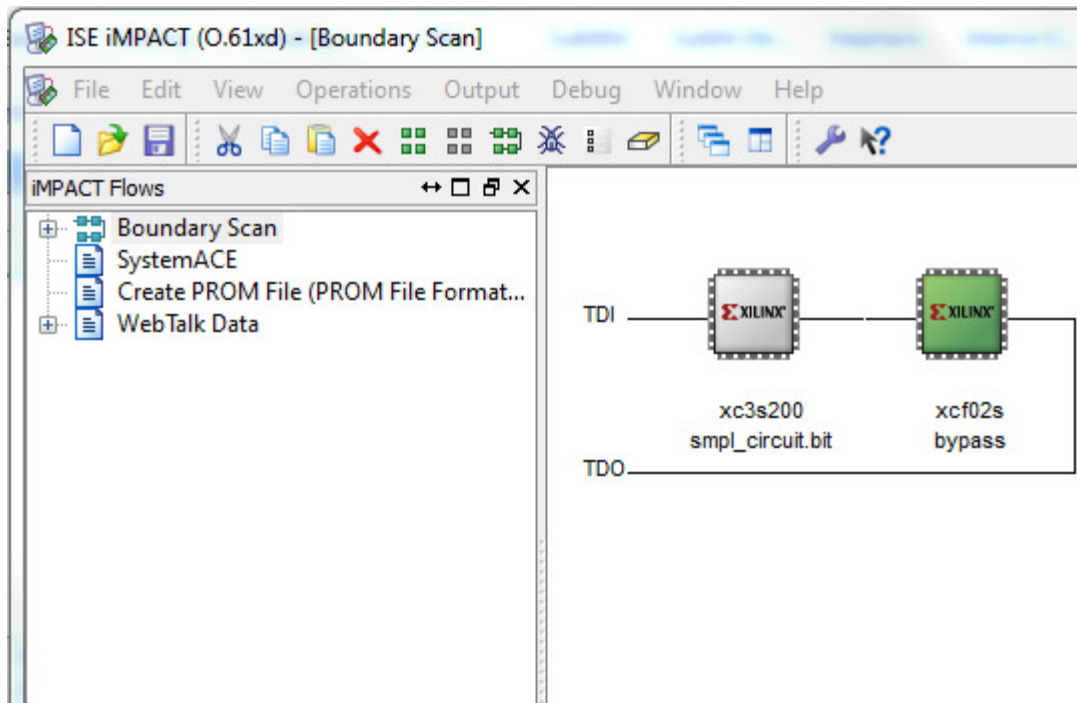
15) Right click when prompted and select 'Initialize Chain'.



16) You should then then see the following two devices.

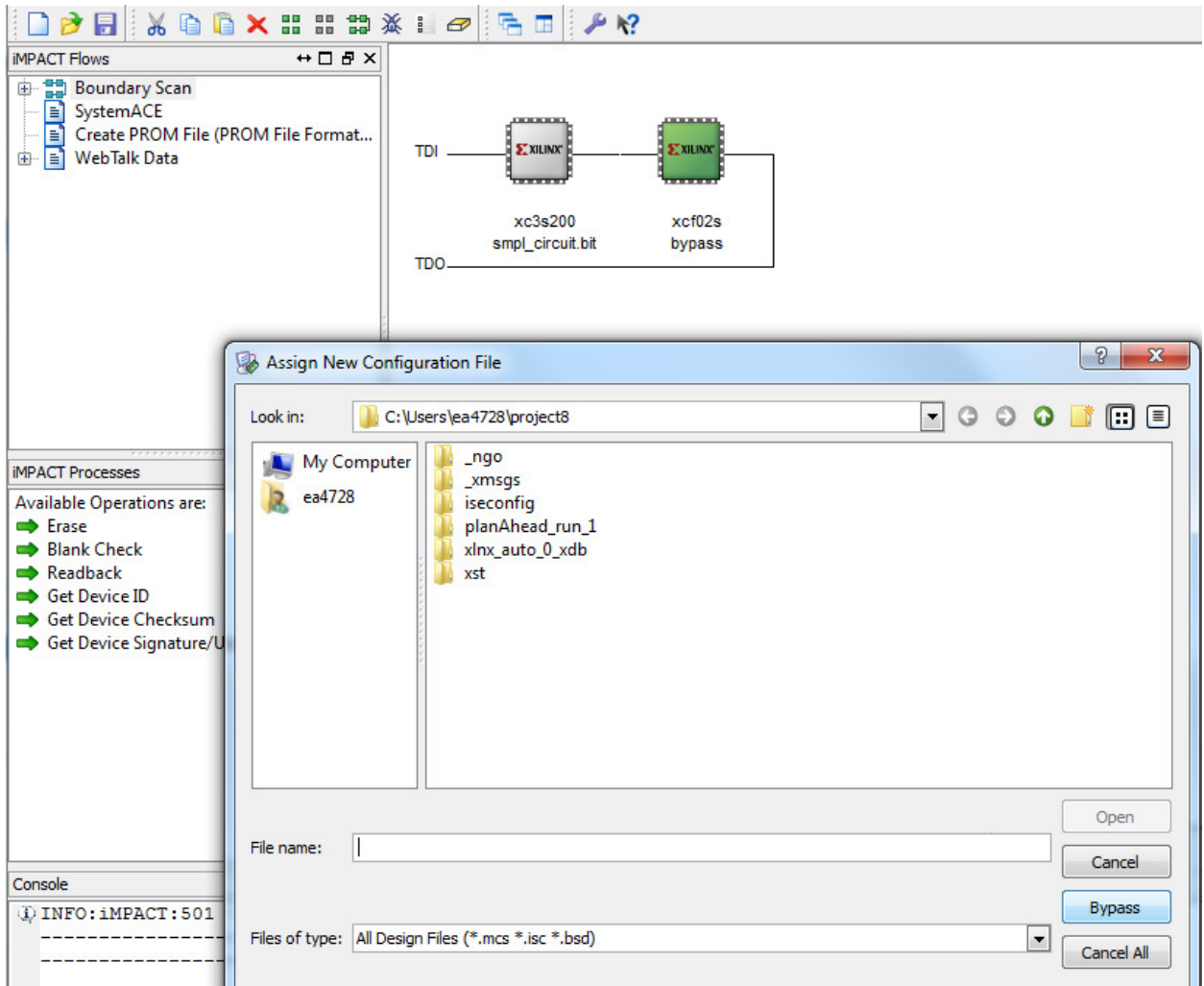


The device that we are interested in is XC3S200. You will be prompted to select .bit file that was generated earlier. Select that .bit file .

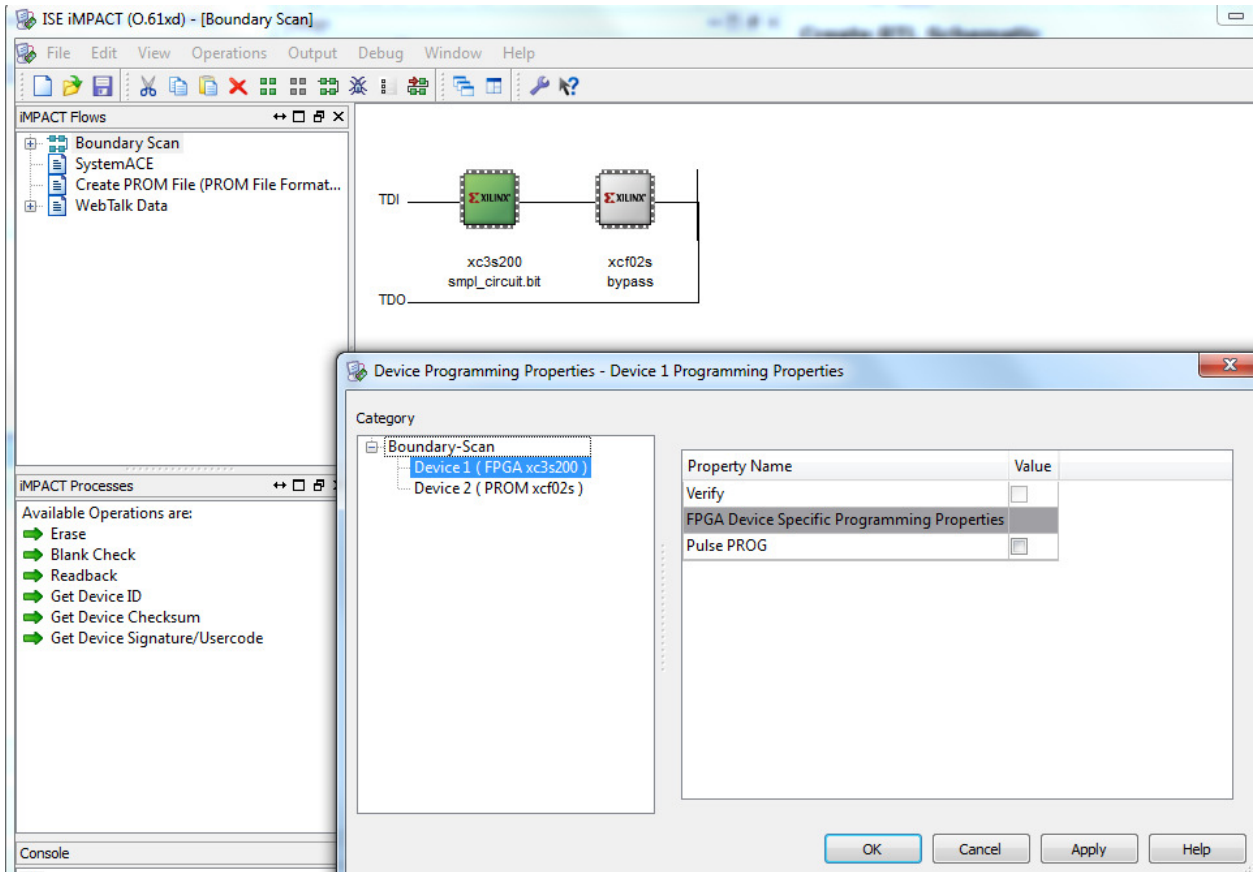


Note: Underneath the device name, you should now see the name of the bit file you selected earlier.

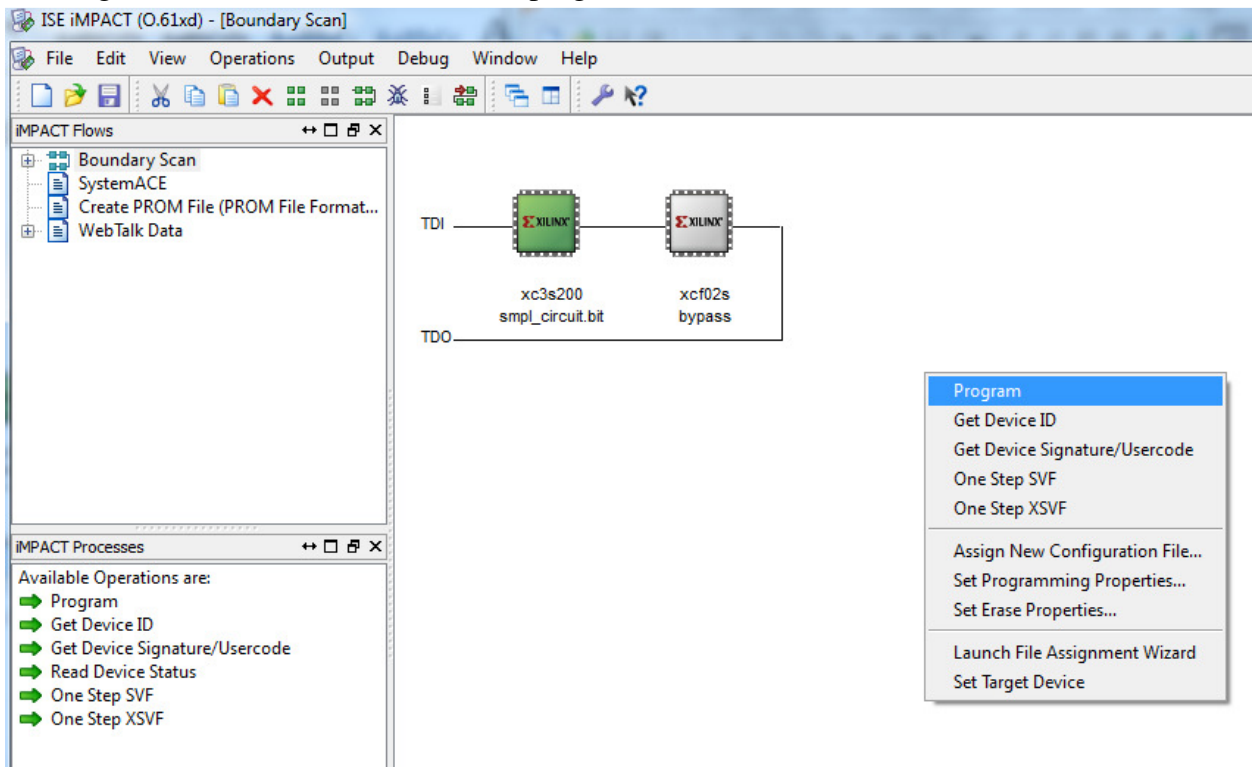
Xilinx will then ask you to add a .bit file to the second device as well. We are not interested in this, so click BYPASS for this one.



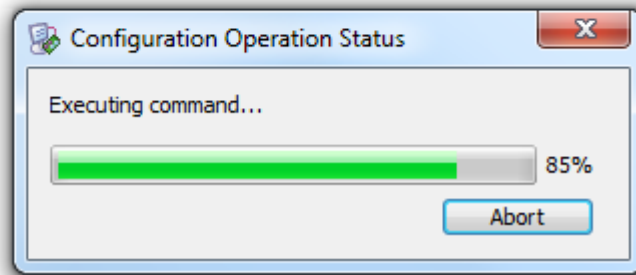
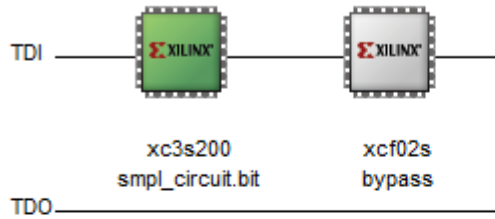
- 17) Then select FPGA Device when prompted as shown below.
Click Apply → OK



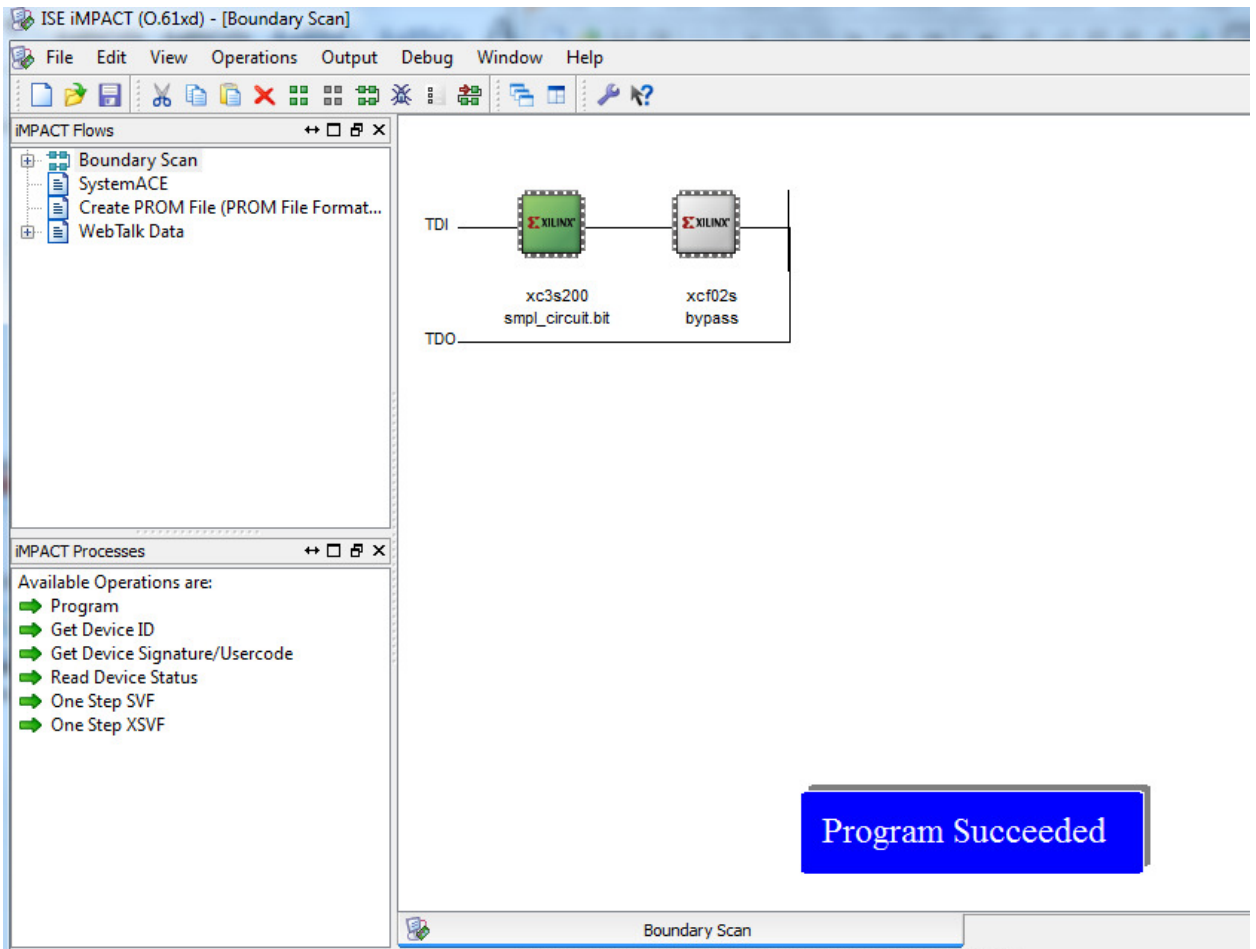
18) Then right click on the screen and select program



Now you are ready to download the program to the board.



19) After successful completion, you should see the message shown below:



20) Now, go ahead and test your code on the board using the pins you assigned earlier.

REFERENCES

- [1] M.A. Bayoumi, G.A Jullien, W.C Miller, " A look-up table VLSI design methodology for RNS structures used in DSP applications," *IEEE Trans. Circuits and Syst.*, vol.34, no. 6, pp 604–616, Jun. 1987.
- [2] J. Bajard, and L. Imbert "A full RNS implementation of RSA" , *IEEE Trans. on Comp.*, vol. 53, no. 6, pp. 769-774 , June 2004.
- [3] K. Konstantinides and V. Bhaskaran, "Monolithic architectures for image processing and compression," *IEEE Computer Graphics and Applications*, vol. 12, no. 6, pp. 75–86, Nov. 1992.
- [4] G. Alia and E. Martinelli, "A VLSI algorithm for direct and reverse conversion from weighted binary number to residue number system," *IEEE Trans. Circuits and Syst.*, vol. 31, no. 12, pp. 1033–1039, Dec. 1984.
- [5] R. M. Capocelli and R. Giancarlo, "Efficient VLSI networks for converting an integer from binary system to residue number system and vice versa," *IEEE Trans. Circuits and Syst.*, vol. 35, no.11, pp. 1425–1431, Nov. 1988.
- [6] A. Mohan, "Novel design for binary to RNS converters," *IEEE Int. Symp conf. Circuits and Systems*, vol. 2, no. 2, pp. 357–360, June 1994.
- [7] P. Behrooa, "Optimal table-lookup schemes for binary-to-residue and residue-to-binary conversions," *IEEE Signals, Systems and Compluter Conf.*, vol. 1, pp 812-816, Nov. 1993.
- [8] Mohamed. Akkal and Pepe Siy, "A new mixed radix conversion algorithm," *Journal of Systems Architecture*, vol. 5, no. 9, pp. 577-586, Sept. 2007.
- [9] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*, New York: McGraw Hill, 1967.
- [10] D. N. Warren-Smith, *Introduction to Digital Circuit Theory: A Monograph on Digital Circuit Theory from the Beginning*, Digital Logic Systems, 2nd ed., 2006, ISBN: 978-095818941-5

- [11] J. Timler and C. S. Lent, "Power gain and dissipation in quantum-dot cellular automata," *Journal Appl. Phys.*, vol. 91, no. 2, pp. 823-831, Jan. 2002.
- [12] F. Barsi and M. Cristina Pinotti, "Fast base extension and precise scaling in RNS for look-up table implementations", *IEEE Trans. on Signal Processing*, vol. 43, no. 10, pp. 2427-2430, Oct. 1995.
- [13] Chin-Yung, Shiou-An Wang and Sy-Yen, "Quantum boolean circuits construction using tabulation method," *4th IEEE Nanotechnol. Conf.*, vol. 1, pp. 596-598, Aug. 2004.
- [14] Yuke Wang, "Residue-to-binary converters based on new chinese remainder theorems, " *IEEE Trans. Circuits and Syst II*, vol. 47, no. 3, pp. 197-205, March 2000.
- [15] Yuke Wang, Xiaoyu Song and Mostapha Aboulhamid, "A new algorithm for RNS magnitude comparison based on new chinese remainder theorems," *IEEE Candian Conf. on Elec. and Comp.*, vol. 1, pp. 571-576, May 1999.
- [16] M. Anatha Shenoy and Ramdas Kumaresan, " A fast and accurate RNS scaling technique for high speed signal processing," *IEEE Trans. on Acoustics. Speech and Signal Processing Processing*, vol 37, no. 6, pp. 929-937, June 1989.
- [17] D. Banerji, "On the use of residue arithmetic for computation," *IEEE Trans. Comp.*, vol.C-23, no. 12, Dec. 1974.
- [18] G. Dimauro, S. Impedovo, and G. Pirlo, "A new technique for fast number comparison in residue number system", *IEEE Trans. on Comp.*, vol. 42, no. 5, pp. 608-612, May 1993.
- [19] R. Zhang, K. Walus, W. Wang and G. A. Jullien, "A method of majority logic reduction for quantum cellular automata," *IEEE Trans. Nanotechnol.*, vol. 3, no. 4, pp. 443-450, Dec. 2004.
- [20] C. Efstathiou, D. Nikolos and J. Kalamatianos. "Area-time efficient modulo $2n-1$ adder design," *IEEE Trans. On Circuits and Systems-II*, vol. 41, no. 7, pp. 463-467, July 1994.

- [21] M. D. Ercegovic and T. Lang, "Simple radix-4 division with operands scaling," *IEEE Trans.on Comp.*, vol. 39, no. 9, pp.1204-1208, Sep. 1997.
- [22] A. Hiasat, "New designs for a sign detector and a residue to binary converter," *IEEE Proc. on Circuits, Devices and Systems*, vol. 140, no. 4, pp. 247-252, August 1993.
- [23] H. S. Miller and R. O. Winder, "Majority-logic synthesis by geometric methods," *IEEE Trans. Elec. Comp.*, vol. EC-11, no. 1, pp. 89-90, Feb. 1962.
- [24] K. M. Ibrahim and S. N. Saloum, "An efficient residue to binary converter design," *IEEE Trans. on Circuits and Sys.*, vol.35, no. 9. pp.1156-1158, Sept. 1988.
- [25] G. A. Jullien, "Residue number scaling and other operations using ROM arrays," *IEEE Trans. on Comp.*, vol. 27, no. 4, pp. 325-337, April 1978.
- [26] F. Miyata, "Realization of arbitrary logical functions using majority elements," *IEEE Trans. Electronic. Comput.*, vol. EC-12, no. 3, pp. 183-191, Jun. 1963.
- [27] M. Lu and J. S. Chinag, "A novel division algorithm for the residue number system," *IEEE Trans. on Comp.*, vol.41, No. 8, pp. 1026-1032, August 1992.
- [28] Roy D. Merrill, "Improving digital computer performance using residue number theory", *IEEE Trans. Electronic Comp.*, vol. EC-13, no. 2, pp. 93-101, April 1964.
- [29] S. B. Akers, "On the algebraic manipulation of majority logic," *IEEE Trans. Electronic. Comp.*, vol. EC-10, no. 4, pp. 779-779, April 1961.
- [30] L. TAI, and F. Chen, "Overflow detection in a redundant residue number system," *IEEE Proceedings, Part E, Computers and Digital Techniques*, vol. 131, no. 3, pp. 97-98, May 1984.
- [31] N. Takagi, H. Yasura, and S. Yajima, "High-speed VLSI multiplication algorithm with a redundant binary addition tree, " *IEEE Transaction on comp*, vol. C-34, no. 9, pp. 789-796, Spet. 1985.
- [32] S. Talahmeh, and P. Siy, "Arithmetic division in RNS using galois field GF(p)," *Computer Math. with Appl.*, vol. 39, no. 5, pp. 227-238, March 2000.

- [33] C. S. Lent and P. D. Tougaw, "A device architecture for computing with quantum dots," *Proc. IEEE*, vol. 85, no. 4, pp. 541–557, Apr. 1997.
- [34] J. C. Majithia, "A pipeline array for square-root extraction," *IEEE Electron. Lett.*, vol. 9, no.1, pp. 4-5, Jan. 1973.
- [35] A.B. Premkumar, "An RNS to binary converter in $(2n-1), 2n, (2n+1)$ moduli set", *IEEE Trans. Circuits and Syst.*, vol.39, no. 11, pp 480- 482, July 1992.
- [36] A.P. Shenoy and R. Kumaresan, "Residue to binary conversion for RNS arithmetic using only modular look-up tables", *IEEE Trans. Circuits and Syst.*, vol.35, no.9. pp 1158-1162, Sept 1988.
- [37] P. Bernardson, "Fast memoryless, over 64 bits, residue-to binary converter," *IEEE Trans. Circuits and Syst.*, vol. 32, no.3, pp 298-300, Mar. 1985.
- [38] M.Hitz and E. Kaltofen," Integer division in residue number systems", *IEEE Trans. Comp.*, vol. 44, no. 8, pp.983-989, Aug. 1995
- [39] H. Cho and E. Swartzlander, "Adder design and analyses for quantum-dot cellular automata," *IEEE Trans on Nanotechnology*, vol. 6, no. 3, pp. 374-383, May 2007.
- [40] P. D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," *Journal Appl. Phys.*, vol. 75, no. 3, pp. 1818–1825, Feb. 1994.
- [41] T. Oya, T. Asai, T. Fukui and Y. Amemiya, "A majority-logic device using an irreversible single-electron box," *IEEE Trans. Nanotechnol.*, vol. 2, no. 1, pp. 15–22, Mar. 2003.
- [42] Y. Fu and M. Wdlander, "Modelling and design of quantum dot cellular automata," *Journal Appl. Phys.*, vol. 83, no. 6, pp. 3186-3191, March 1997.
- [43] A. Gin, S. Williams, H. Meng, and P. D. Tougaw, "Hierarchical design of quantum cellular automata," *Journal. Appl. Phys.*, vol. 85, no. 7, pp.3713–3720, Apr. 1999.
- [44] S. Perri and P. Corsonello, "New methodology for the design of efficient binary addition circuits in QCA," *IEEE Trans. Nanotechnol.*, vol. 11, no. 6, pp. 1192–1200, Nov. 2012.

- [45] S . Andraos and H. Ahmed, "A new efficient memoryless residue to binary converter," *IEEE Trans. Circuits and Syst.*, vol.35, no. 11, pp. 1441-1444, Nov. 1988.
- [46] J. C. Lusth and D. J. Jackson, "Graph theoretic approach to quantum cellular design and analysis" *Journal. Appl. Phys.*, vol. 79, no. 4, pp. 2097 - 2102, Feb. 1996.
- [47] Rui Zhang, P. Gupta, and N. K. Jha, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Trans. on Comp.-Aided Design of Intergrated Circuits and Systems*, vol. 24, no. 1, pp. 107-118, Jan. 2005.
- [48] Rui Zhang, P. Gupta and N. K. Jha, "Majority and minority networks synthesis with and applications to QCA-, SET- and SET- based nanotechnologies," *IEEE Trans. on Comp.-Aided Design of Intergrated Circuits and Systems*, vol. 26, no. 7, pp. 1233 - 1245, July 2007.
- [49] T. V. Vu, "Efficient implementations of the chinese remainder theorem for sign detection and residue decoding" *IEEE Trans. Comp.*, vol. 34, no. 7, pp. 646-651, July 1985.
- [50] M. B. Tahoori, J. Huang, M. Momenzadeh and F. Lombardi, "Testing of quantum cellular automata," *IEEE Trans. Nanotechnol.*, vol. 3, no.4, pp. 432–442, Dec. 2004.
- [51] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comp.*, vol. 30, no. 3, pp. 215–222, Mar. 1981.
- [52] A. Chaudhary, D. Z. Chen, X. S. Hu, M. T. Niemier, R. Ravichandran and K. Whitton, "Fabricatable interconnect and molecular QCA circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 11, pp. 1978–1991, Nov. 2007.
- [53] S. Roy and V. Beiu, "Majority multiplexing-economical redundant fault-tolerant designs for nanoarchitectures," *IEEE Trans. Nanotechnol.*, vol. 4, no. 4, pp. 441–451, Jul. 2005.
- [54] W. Ibrahim, V. Beiu, and M. Sulieman, "On the reliability of majority gates full adders," *IEEE Trans. Nanotechnol.*, vol. 7, no. 1, pp. 56–67, Jan. 2008.

- [55] S. Srivastava and S. Bhanja, "Hierarchical probabilistic macromodeling for QCA circuits," *IEEE Trans. Comput.*, vol. 56, no. 2, pp. 174–190, Feb. 2007.
- [56] S. Bhanja and S. Sarkar, "Probabilistic modeling of QCA circuits using bayesian networks," *IEEE Trans. Nanotechnol.*, vol. 5, no. 6, pp. 657–670, Nov. 2006.
- [57] K. Kim, K. Wu and R. Karri, "The robust QCA adder designs using composable QCA building blocks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 1, pp. 176–183, Jan. 2007.
- [58] J. Janulis, P. Tougaw, S. Henderson, and E. Johnson, "Serial bit-stream analysis using quantum-dot cellular automata," *IEEE Trans on Nanotechnology*, vol. 3, no.1, pp. 158-164, March 2004.
- [59] V. Vankamamidi, M. Orravi, and F. Lombardi, "A line-based parallel memory for QCA implementation," *IEEE Trans. Nanotechnol.*, vol. 4, no. 6, pp. 690-698, Nov. 2005.
- [60] Richard F. Tinder, "Multilevel logic minimization using K-map XOR patterns," *IEEE Trans. On Eud.*, vol. 38, no. 5, pp. 370 - 375, Nov. 1995.
- [61] P. D. Tougaw and C. S. Lent, "Dynamic behavior of quantum cellular automata," *Journal Appl. Phys.*, vol. 80, no. 8, pp. 4722–4736, Oct. 1996.
- [62] C. S. Lent and B. Isaksen, "Clocked molecular quantum-dot cellular automata," *IEEE Trans. Electron Devices*, vol. 50, no. 9, pp. 1890-1896, Sep. 2003.
- [63] E. S. Mandell and M. Khatun, "Quasi-adiabatic clocking of quantum-dot cellular automata," *Journal Appl. Phys.*, vol. 94, no. 6, pp. 4116–4121, Sep. 2003
- [64] Seminario JM, Derosa PA, Cordova LE and Bozard BH, "A molecular device operating at terahertz frequencies: theoretical simulations," *IEEE Trans. on Nanotechnology* , vol. 3, no. 1, pp.215–218, March 2004.
- [65] M. A.Bayoumi, G. A. Jullien and W. C. Miller, "A VLSI implementation of residue adders," *IEEE Trans. Circuits and Syst.*, vol. 34, no.3, pp. 284-288, Mar. 1987.

- [66] F. J. Taylor, "A single modulus complex ALU for signal processing," *IEEE Trans. Acoust., Speech Signal Processing*, vol. 33, no.5, pp. 1302 – 1315, Oct. 1985.
- [67] D. K. Banerji, "A novel implementation method for addition and subtraction in residue number systems," *IEEE Trans. Comput.*, vol. C-23, no.1, pp. 106-109, Jan. 1974
- [68] S. J. Piestrack, "Design of residue generators and multioperand adders modulo 3 built of multioutput threshold circuits," *Proc. IEEE Computers and Digital Techniques*, vol. 141, no. 2, pp. 129 - 134, March 1994.
- [69] C. H. Huang, "A fully parallel mixed-radix conversion algorithm for residue number applications", *IEEE Transactions on Computers*, vol. 32, no. 4, pp. 398 – 402, April 1983.
- [70] M. Akkal and P. Siy, "A new mixed radix conversion algorithm MRC-II", *Journal of System Architecture*, vol. 53, no. 9, pp. 577-586, May. 2007.
- [71] M. Becherer, G. Csaba, W. Porod, R. Emling, P. Lugli, D. Schmitt-Landsiedel, "Magnetic ordering of focused-ion-beam structured cobalt-platinum dots for field-coupled computing", *IEEE Trans. on Nanotechnology*, vol.7, no.3, pp. 316-320, May 2008.
- [72] B. Qiaa and H. E. Ruda, "Evolution of a two-dimensional quantum cellular neural network driven by an external field," *Journal Appl. Phys.*, vol. 85, no. 5, pp. 2952-2961, March 1999.
- [73] M. Akkal and P. Siy "Optimum RNS sign detection algorithm using MRC-II with special moduli set," *Journal of System Architecture*. vol 54, no. 10, pp. 911-918. Oct. 2008.
- [75] G. C. Cardarilli, "RNS-to-binary conversion for efficient VLSI implementation," *IEEE Circuits and Systems I*, vol. 46, no. 6, pp. 2427 – 2430, Oct. 1995.
- [76] V. Pudi and K. Sridharan, "Low complexity design of ripple carry and Brent–Kung adders in QCA," *IEEE Trans. Nanotechnol.*, vol. 11, no. 1, pp. 105–119, Jan. 2012.
- [77] M. Covemale, M. Macucci, G. Iannaccone, C. Ungarelli and J. Martorell, "Modeling and manufacturability assessment of bistable quantumdot cells," *Journal Appl. Phys.*, vol. 85, no. 5, pp. 2962-2971, Mar. 1999.

- [78] K. Walus, R. A. Budiiman and G. A. Jullien, "Split current quantum-dot cellular automata modeling and simulation," *IEEE Trans. Nano.*, vol. 3, no. 2, pp. 249-255, June 2004.
- [79] G. Toth and C. S. Lent "Role of correlation in the operation of quantum-dot cellular automata," *Journal Appl. Phys.*, vol. 89, no. 12, pp. 7943-7953, June 2001.
- [80] J. Timler and C. S. Lent, "Power gain and dissipation in quantum-dot cellular automata," *Journal Appl. Phys.*, vol. 91, no. 2, pp. 823-831, Jan. 2002.
- [81] I. Amlani, "Experimental demonstration of a leadless quantum-dot cellular automata cell," *Appl. Phys. Lett.*, vol. 77, no. 5, pp. 738-740, July 2000.
- [82] R. Brayton, G. Hachtel and A. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, no. 2, pp. 264-300, Feb. 1990.
- [83] K. Walus, "High level exploration of quantum dot automata (QCA)," *Conf. Signal, System. and Comp.*, vol. 1, pp. 30-33, Nov. 2004.
- [84] T. Sasao and P. Besslich, "On the complexity of Mod-2-sum PLA's," *IEEE Trans. Comput.*, vol. 39, no. 2, pp. 262-265, Feb. 1990.
- [85] H. Cho and E. E. Swartzlander, "Adder and multiplier designs in quantum-dot cellular automata," *IEEE Trans. Comput.*, vol. 58, no. 6, pp. 721-727, Jun. 2009.
- [86] K. Walus, T. Dysart, G. Jullien and R. Budiman, "QCADesigner: A rapid design and simulation tool for quantum-dot cellular automata," *IEEE Trans. Nanotechnol.*, vol. 3, no. 1, pp. 26-29, Mar. 2004.
- [87] K. Walus, G. Schulhof and G. Jullien, "Implementation of a simulation engine for clocked molecular QCA," *Proc. IEEE Can. Conf. Electr. Comput. Eng.*, vol. 1, pp. 2128-2131, May 2006.
- [88] T. Yatagai, "Cellular logic arithmetic for optical computers", *Applied Optics*, vol. 25, no. 10, pp. 1571-1577, May 1986.
- [89] G. Strucke, "Parallel architecture for digital optical computer" *Applied Optics*, vol. 28, no. 2, pp. 363-370, Jan 1989.

- [90] Dharma P. Agrawal and T. R. N. Rao, "On multiple operand addition of signed binary numbers," *IEEE Trans. Comp.*, vol. C-27, no. 11, pp. 1068-1070, Nov. 1978.
- [91] A.K Kamal, H. Singh and D. P. Agrawal, "A generalized pipeline array," *IEEE Trans. Comp.*, vol. 23, no. 5, pp. 533-536, May 1974.
- [92] Dharma P. Agrawal, "High-speed Arithmetic arrays," *IEEE Trans. Comp.*, vol. C-28, no. 3, pp. 215-224, March 1979.
- [93] R. Golshan and J. S. Bedi, "Reversible nonlinear interface optical computing," *Opt. Eng.*, vol. 28, no. 6, pp. 683-686, June 1989.
- [94] M. Dagenais and W.F Sharfin "Optical switching of semiconductor laser amplifiers," *Appl. Phys. B.*, vol. B46, no. 1, pp. 35-41, May 1988.
- [95] J. Tanda and Y Ichioka "Modular Components for an array logic system", *Applied Optics*, vol. 26, no. 18, pp.3954 -3960, Oct. 1987
- [96] J. C. Majithia and R. Kitai, "A cellular array for the nonrestoring extraction of square roots," *IEEE Trans. Comput. (Corresp.)*, vol. C-20, pp. 1617-1618, Dec 1971.
- [97] H. H. Majithaia and R. Kitai, "Cellular array for nonrestoring square root extraction," *Electron. Lett.*, vol. 6, pp. 66-87, 1970.
- [98] J. C. Hoffman, B. Lacaze and P. Csillag, "Iterative logical network for parallel multiplication," *Electron. Lett.*, vol. 4, pp. 178-179, May 1968.
- [99] K. J. Dean, "Binary division using a data-dependent iterative array," *Electron. Lett.*, vol. 4, pp. 283-284, July 1968.
- [100] R. C. Devries and M. H. Chao, "Fully iterative array for extracting square roots," *Electron. Lett.*, vol. 6, pp. 255-256, Apr. 1970.
- [101] A. Gex, "Multiplier-divider cellular array," *Electron. Lett.*, vol. 7, pp. 442-444, July 1971.
- [102] J. C. Majithia, "Cellular array for extraction of squares and square roots of binary numbers," *IEEE Trans. Comput. (Short Notes)*, vol. C-21, pp. 1023-1024, Sept. 1972.

- [103] D. P. Agrawal and H. Singh, "Iterative array for square-root, division, and multiplication," *8th Ann. Conv. Comput. Soc. of India*, Mar. 26, 1973.
- [104] J. C. Majithia, "A pipeline array for square-root extraction," *Electron. Lett.*, vol. 9, pp. 4-5, Jan. 1973.
- [105] R. Ranjan, H. Singh, A. Awasthi, W. Smuda and G. R. Gerhart, "Finite state modeling of Mobile robots for complexity determination," *Proc. SPIE, Unmanned Systems Technology VIII*, vol. 6230, May 2006.
- [106] Wei Wang, M. Swamy, M Ahmed and Yuke Wang., "A high speed residue-to-binary converter and a scheme for its VLSI implementation, ". *IEEE Trans. Comput.*, vol. 6, no. 1, pp. 330 – 333, July 1999.
- [107] B. Taskin, "Improving line-based QCA memory cell design through dual phase clocking" *IEEE Trans. Very Large Scale Integration (VLSI)*, vol. 16, no. 12, pp. 1648 - 1656, Dec. 2008.
- [108] Yuke Yang and Mustafa Abd-El-Bar, "A new algorithm for RNS Decoding," *IEEE Transactions in Circuits and Systems-I: Fundamental Theory and Applications*, vol 43, no 12, Dec. 1996.
- [109] W. K. Jenkins and B. J. Leon, "The use of residue number systems in the design of finite impulse response digital filters," *IEEE Trans. Circuits Syst.*, vol. 24, no. 4, pp. 191-201, Apr. 1977.

ABSTRACT**EMERGING DESIGN METHODOLOGY AND ITS IMPLEMENTATION
THROUGH RNS AND QCA**

by

OMAR DAJANI**August 2013****Advisor:** Dr. Harpreet Singh**Major:** Electrical Engineering**Degree:** Doctor of Philosophy

Digital logic technology has been changing dramatically from integrated circuits, to a Very Large Scale Integrated circuits (VLSI) and to a nanotechnology logic circuits. Research focused on increasing the speed and reducing the size of the circuit design. Residue Number System (RNS) architecture has ability to support high speed concurrent arithmetic applications. To reduce the size, Quantum-Dot Cellular Automata (QCA) has become one of the new nanotechnology research field and has received a lot of attention within the engineering community due to its small size and ultralow power.

In the last decade, residue number system has received increased attention due to its ability to support high speed concurrent arithmetic applications such as Fast Fourier Transform (FFT), image processing and digital filters utilizing the efficiencies of RNS arithmetic in addition and multiplication. In spite of its effectiveness, RNS has remained more an academic challenge and has very little impact in practical applications due to the complexity involved in the conversion process, magnitude comparison, overflow detection, sign detection, parity detection, scaling and division. The advancements in

very large scale integration technology and demand for parallelism computation have enabled researchers to consider RNS as an alternative approach to high speed concurrent arithmetic. Novel parallel - prefix structure binary to residue number system conversion method and RNS novel scaling method are presented in this thesis.

Quantum-dot cellular automata has become one of the new nanotechnology research field and has received a lot of attention within engineering community due to its extremely small feature size and ultralow power consumption compared to COMS technology. Novel methodology for generating QCA Boolean circuits from multi-output Boolean circuits is presented. Our methodology takes as its input a Boolean circuit, generates simplified XOR-AND equivalent circuit and output an equivalent majority gate circuits.

During the past decade, quantum-dot cellular automata showed the ability to implement both combinational and sequential logic devices. Unlike conventional Boolean AND-OR-NOT based circuits, the fundamental logical device in QCA Boolean networks is majority gate. With combining these QCA gates with NOT gates any combinational or sequential logical device can be constructed from QCA cells. We present an implementation of generalized pipeline cellular array using quantum-dot cellular automata cells. The proposed QCA pipeline array can perform all basic operations such as multiplication, division, squaring and square rooting. The different mode of operations are controlled by a single control line.

AUTOBIOGRAPHICAL STATEMENT

Omar Dajani

He received his BS degree in Computer Engineering from Jordan University of Science and Technology, Jordan, in 1995 and MS degree in Electrical and Computer Engineering from University of Detroit Mercy, Detroit, MI in 1996. He is currently working as electrical system engineer for Ford Motor Company and pursuing a Ph.D. degree in Electrical Engineering at Wayne State University, Detroit, MI. His research interests are in residue number system, nanotechnology, parallel processing and VLSI.

PUBLICATIONS:

- [1] O. Dajani and P. Siy, "Novel parallel - prefix structure binary to residue number system conversion method," *CSC, Journal of Int. Logic and Comp.*, vol. 3, no. 1, pp. 1-13, Oct. 2012.
- [2] O. Dajani and P. Siy, "Simplified RNS scaling algorithm and division algorithm," *CSC, Journal of Int. Logic and Comp.*, submitted April 2011 and accepted Aug. 2012.
- [3] O. Dajani, G. Bawa and H. Singh, "VLSI implementation of residue adder and subtractor," *Int'l Conf. Frontiers on Comp. Sci. and Comp. Eng. (FECS'12)*, vol. 1, pp. 604 -607, July 2012.
- [4] O. Dajani and H. Singh, "Quantum boolean circuit construction methodology using XOR-AND reduction technique," *IEEE SEM Fall Conf.*, Nov. 2012.
- [5] O. Dajani, H. Singh and D. P. Agrawal, "Implementation of generalized pipeline cellular array using quantum-dot cellular automata," *IEEE Trans. Nanotechnol.*, submitted March 2013.