

Studies in Computational Intelligence

Volume 599

Series editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
e-mail: kacprzyk@ibspan.waw.pl

About this Series

The series “Studies in Computational Intelligence” (SCI) publishes new developments and advances in the various areas of computational intelligence—quickly and with a high quality. The intent is to cover the theory, applications, and design methods of computational intelligence, as embedded in the fields of engineering, computer science, physics and life sciences, as well as the methodologies behind them. The series contains monographs, lecture notes and edited volumes in computational intelligence spanning the areas of neural networks, connectionist systems, genetic algorithms, evolutionary computation, artificial intelligence, cellular automata, self-organizing systems, soft computing, fuzzy systems, and hybrid intelligent systems. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution, which enable both wide and rapid dissemination of research output.

More information about this series at <http://www.springer.com/series/7092>

K. Sridharan · Vikramkumar Pudi

Design of Arithmetic Circuits in Quantum Dot Cellular Automata Nanotechnology

K. Sridharan
Department of Electrical Engineering
Indian Institute of Technology Madras
Chennai
India

Vikramkumar Pudi
Department of Electrical Engineering
Indian Institute of Technology Madras
Chennai
India

Additional material to this book can be downloaded from <http://extras.springer.com>.

ISSN 1860-949X ISSN 1860-9503 (electronic)
Studies in Computational Intelligence
ISBN 978-3-319-36511-4 ISBN 978-3-319-16688-9 (eBook)
DOI 10.1007/978-3-319-16688-9

Springer Cham Heidelberg New York Dordrecht London

© Springer International Publishing Switzerland 2015

Softcover reprint of the hardcover 1st edition 2015

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media
(www.springer.com)

Dedicated to our families

Preface

Overview

This book grew out of our research enquiry into arithmetic in emerging nanotechnologies. It describes our research starting in early 2008 on design of circuits in Quantum Dot Cellular Automata (QCA) with the objectives of obtaining low-complexity and robust designs for various arithmetic operations. The book investigates systematic reduction of majority logic for realization of multi-bit adders and a multiplier. An extension to computation of a transform is also pursued with a view to examine potential for embedded system design in emerging nanotechnologies. Careful layout design keeping in view high performance is also a goal of the research.

Organization and Features

Chapter 1 presents the motivation for the work described in this book. This chapter also gives an overview of the literature on the subject. Chapter 2 presents terminology pertaining to quantum dot cellular automata. Chapter 3 familiarises the reader with QCA designs for basic logic elements such as gates and flip-flops. Chapter 4 presents material on majority logic optimization for obtaining efficient QCA designs of single and multi-bit adders. In particular, design of an efficient ripple carry adder as well as various prefix adders are presented in this chapter. Chapter 5 studies the design of a custom adder called the *hybrid adder* for QCA technology. Chapter 6 extends the investigations on adders to the design of a high-performance multiplier in QCA. Chapter 7 is devoted to efficient computation of a discrete orthogonal transform, namely the Discrete Hadamard Transform (DHT), in QCA. Chapter 8 presents a discussion on thermal robustness for QCA designs. Chapter 9 presents a summary of the work described in this book and outlines extensions. An appendix on the steps to generate a layout in the CAD tool QCADesigner is included.

Audience

This book presents material that is appropriate for courses at the senior undergraduate and graduate levels in the areas of nanoelectronics, computer arithmetic and embedded systems. It can also be used as a supplement to courses on digital circuits and laboratories on digital systems. The book is also suitable for researchers in the areas of computer arithmetic, nanotechnologies and VLSI design. In addition, the book provides examples and tutorials on a CAD tool that would help beginners to get a head start on QCA layout design. Basic familiarity with logic design is adequate to follow the material presented in this book.

Acknowledgments

The authors owe a word of thanks to many people who helped in various ways. The authors thank their families and friends for their support. The authors thank the researchers who have developed QCADesigner (<http://www.mina.ubc.ca/qcadesigner>). Special thanks go to Dr. Thomas Ditzinger, Springer editor, for obtaining reviews for chapters in this book. The authors thank Dr. Dieter Merkle and Dr. Guido Zosimo-Landolfo of Springer International Publishing for their support. The authors also thank Mr. Holger Schaepe, Mr. Abbas Manthiri and Ms. Vinodhini Sundararajan of Springer for editorial assistance. The authors also acknowledge the support of the Indian Institute of Technology Madras.

K. Sridharan
Vikramkumar Pudi

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Alternatives to MOSFET and Challenges	1
1.2.1	Emerging Transistor-Based Devices	2
1.2.2	Emerging Nanotechnologies Based on Other Paradigms	2
1.3	Quantum Dot Cellular Automata-Origins and Promise	3
1.4	Metrics and Challenges in QCA-Based Digital Design	3
1.5	Contributions of the Book	4
1.5.1	Design of Basic Circuits in QCA	4
1.5.2	Efficient Design of Ripple Carry and Prefix Adders in QCA	4
1.5.3	Efficient Design of a Custom Adder in QCA	5
1.5.4	Multiplier Design in QCA	5
1.5.5	Computation of Discrete Hadamard Transform in QCA	5
1.5.6	Layout Development, Simulation and Robustness Studies	5
1.6	Literature Survey	6
1.6.1	QCA-Theory and Fabrication	6
1.6.2	Majority Logic Manipulation and Synthesis	6
1.6.3	QCA-Based Digital Design	7
1.7	Organization of the Book	8
	References	8
2	QCA Terminology	11
2.1	QCA Basics	11
2.1.1	Cell, Quantum Dot and Tunnel Junctions	11
2.1.2	How Are Different Logic States Obtained?	11

2.2	Logic Primitives in QCA	13
2.2.1	Clocking in QCA-Role and Types	14
2.2.2	Crossovers in QCA	16
2.3	Tool for QCA Simulation	16
2.4	Summary	17
	References	17
3	Design of Basic Digital Circuits in QCA	19
3.1	Design of Logic Gates in QCA	19
3.2	Design of a Multiplexer in QCA	23
3.3	Design of a One-Bit Full-Adder in QCA.	24
3.4	Design of a Flip-Flop in QCA	25
3.5	Summary	26
	References	26
4	Design of Ripple Carry and Prefix Adders in QCA	27
4.1	Design of the Ripple Carry Adder (RCA) in QCA.	27
4.1.1	New Results on Majority Logic.	27
4.1.2	Application of the Majority Logic Results to a Ripple Carry Adder.	29
4.2	Design of Prefix Adders in QCA	31
4.2.1	The Kogge-Stone Adder.	35
4.2.2	The Ladner-Fischer Adder	39
4.2.3	The Brent-Kung Adder.	42
4.3	QCA Implementation	47
4.3.1	Design Rules.	48
4.3.2	Simulation Engine	48
4.3.3	Layout Level Implementation of RCA and Prefix Adders	49
4.4	Comparison Studies	51
4.4.1	Cell Count, Area and Delay for Various Adders	51
4.5	Summary	54
	References	54
5	Design of a Hybrid Adder in QCA	57
5.1	Introduction.	57
5.2	Preliminaries for the Hybrid Adder in QCA	57
5.3	Design of 16-Bit Hybrid Adder in QCA	57
5.3.1	Generalization to an n -bit Hybrid Adder.	62
5.3.2	Delay Analysis for a Hybrid Adder	66
5.4	QCADesigner Layout Diagrams and Simulation Results	67
5.5	Comparisons	68
5.5.1	Comparison of Different Adder Styles	68
5.5.2	Discussion of Results	68

5.6 Summary	71
References	71
6 Design of a Multiplier in QCA	73
6.1 Introduction to Multipliers.	73
6.2 The Baugh-Wooley Multiplier for 2's Complement Numbers.	73
6.3 Design of Baugh-Wooley Multiplier in QCA.	74
6.4 Summary	77
References	78
7 Discrete Hadamard Transform Computation in QCA	79
7.1 Basics of Discrete Hadamard Transform	80
7.2 Mathematical Formulation for DHT Computation.	80
7.3 Architectural Aspects	82
7.4 QCA Realization	83
7.5 Simulation Results	86
7.6 Performance of a Full-Parallel Addition Strategy and Comparisons	88
7.7 Summary	89
References	89
8 Study of Thermal Robustness of QCA Designs	91
8.1 Simulation Engine and Settings for the Study	91
8.2 Approach Used and the Results	91
8.3 Summary	95
References	95
9 The Road Ahead	97
9.1 Contributions of This Research	97
9.2 Extensions	98
9.2.1 Applications of the Work to Other Nanotechnologies.	98
9.2.2 Magnetic Quantum Dot Cellular Automata	98
9.2.3 Nanomagnet Fabrication and Implementation of Arithmetic Circuits.	98
9.2.4 Application to Other Discrete Orthogonal Transforms.	98
9.3 Concluding Remarks	99
References	99
Appendix: A Tutorial on QCADesigner	101
Index	115

About the Authors

K. Sridharan received his Ph.D. from Rensselaer Polytechnic Institute, Troy, New York in 1995. He was an Assistant Professor at Indian Institute of Technology (IIT) Guwahati from 1996 to 2001. Since June 2001, he is with IIT Madras where he is presently a Professor. He was a visiting staff member at Nanyang Technological University (NTU), Singapore in 2000–2001 and 2006–2008. He has supervised four PhDs and holds two patents. He is an author of a book published by Springer in 2008 on hardware-efficient algorithms for robotics. He has also authored/co-authored approximately 80 papers in various journals and conferences. He is a recipient of the 2009 Dr. Vikram Sarabhai Research Award for his contributions to electronics, telematics, informatics and automation. He also received the Tan Chin Tuan fellowship in Engineering for research in nanoelectronics and VLSI at Nanyang Technological University in 2011. He is an Associate Editor of the IEEE Transactions on Industrial Electronics. His research interests include nanoelectronics, embedded systems and VLSI Design.

Vikramkumar Pudi received his Ph.D. from Indian Institute of Technology Madras in 2014. He is currently a postdoctoral fellow with the Microsystems and Nanotechnology group at the University of British Columbia, Canada. His research interests include digital system design, nanotechnologies and FPGA-based design.

Abstract

Contemporary microprocessors and application-specific integrated circuits are largely based on the Complementary Metal Oxide Semiconductor (CMOS) technology. It is believed that performance of various circuits in current CMOS-based architectures is close to reaching the limit. When feature size of transistors is reduced to a nanometer, quantum effects such as tunneling take place. Further, when device scaling takes place, the interconnections do not scale automatically due to the effects of wire resistance and capacitance.

This book introduces the reader to Quantum Dot Cellular Automata (QCA), an emerging nanotechnology and an alternative to CMOS. It then examines the problem of designing efficient arithmetic circuits in QCA. Using the fact that the 3-input majority gate is a primitive in QCA, the book sets out to discover hitherto unknown properties of majority logic in the context of arithmetic circuit designs. The new results on majority logic are used to obtain adder and multiplier designs which have low complexity (in terms of area and delay).

Two directions for efficient adder designs in QCA are taken in the book. One of them involves a study of existing adders and application of new results on majority logic. The full-adder, ripple carry adder and prefix adders are considered for this purpose. The second direction involves development of a custom adder for QCA technology. A QCA adder named *hybrid adder* is proposed and it is shown that it outperforms existing multi-bit adders with respect to area and delay. The work is extended to the design of a low-complexity multiplier for signed numbers in QCA. In particular, an efficient Baugh-Wooley multiplier design in QCA is presented.

Driven by the potential of QCA for development of embedded systems, the book also examines the design of a high-performance architecture for a discrete orthogonal transform, namely the Discrete Hadamard Transform. Another aspect touched upon is the issue of thermal robustness of QCA designs. The book shows that the proposed hybrid adder has low complexity and good robustness.

Chapter 1

Introduction

1.1 Motivation

Personal computers, smartphones, video game consoles and several other gadgets have a processor based on the metal-oxide semiconductor field effect transistor (MOSFET). The MOSFET has shrunk manyfold during the past few decades from several micrometers in the 1960s [1] to tens of nanometers at present. While the MOSFET has become smaller, chips based on it have become faster and consume lower power. This has led to its use in a diverse set of applications.

However, it is believed that performance of various circuits in current MOSFET-based architectures is close to reaching the limit. The 2013 edition of the International Technology Roadmap for Semiconductors [2] presents an assessment of various technologies. MOSFET-based devices act as voltage controlled current switches and as device dimensions shrink (to accommodate more units in the same area), control of the gate would not be possible as before because of what are called *short-channel* effects [3].

1.2 Alternatives to MOSFET and Challenges

Alternatives to MOSFET have therefore been actively sought. A comprehensive discussion on alternate state variables for emerging nanoelectronic devices has been provided in [4]. Two approaches have been adopted in general by various research groups: One of them attempts to tackle the problem through new transistor-based devices. Devices in this category include the tunnel FET, single electron transistor and carbon nanotube FET. The second approach is based on examining alternatives to transistor-based devices. In particular, spin-wave devices and quantum dot cellular automata are in this category. We briefly review various choices.

1.2.1 Emerging Transistor-Based Devices

A field effect transistor-based technology that has been extensively studied in the last decade is the tunnel FET [5]. The tunnel FET (also referred to as TFET) is based on electron tunneling through a barrier and is a promising solution for low-voltage and low-power applications. In particular, the TFET is aimed at supply voltages below 0.5 V. Early proposals of the TFET concept include [6, 7]. TFETs have been investigated with different materials for the channel, namely carbon nanotubes, silicon and germanium.

Another promising FET-based alternative is the Carbon Nanotube FET [8, 9]. A carbon nanotube (CNT) can act either as a metal or as a semi-conductor depending on a parameter called the *chirality*. When the chirality is set to allow the CNT to function as a semi-conductor, it can be used as the channel material for a transistor. The diameter of the tube (rather than the length) plays an important role in the case of a CNTFET making it appropriate for high-performance and low-power nanoelectronic applications. Field-effect transistors have been fabricated with one semiconducting single-wall carbon nanotube (SWNT) connected to two metal electrodes [8]. Digital circuit design in this model has been studied in [10].

A third transistor-based alternative to the MOSFET is the Single Electron Transistor (SET). The device consists of an isolated conductive island separated from source and drain electrodes by what are called *tunneling junctions* [11]. Transfer of individual electrons between source and drain (and the metal island) is controlled by voltage applied to a gate electrode (located in the vicinity of the island). One of the challenges here is accurate positioning of conducting nanoparticles with respect to external electrodes.

In general, a single unit has been fabricated successfully but integration of several units remains a challenge in most of these technologies. Also, design tools are few and further, not much is known on digital design beyond basic logic gates.

1.2.2 Emerging Nanotechnologies Based on Other Paradigms

While the transistor approach, in general, is based on charge transport, other paradigms have been explored. These include spin-based devices and quantum dot cellular automata. Spin-based devices [12, 13] include magnetic cellular automata [14] and devices with spin wave bus [15]. In the former, an electron's spin can be used to realize the functionality of different logic gates [14]. In the case of devices with spin wave bus [15], magnetic films act as spin conduit of wave propagation. Information can be coded into a phase of the propagating spin wave.

Quantum Dot Cellular Automata (QCA) is another nanotechnology that is not transistor-based. In QCA, the basic element is a cell (containing multiple quantum dots). Interaction between cells is purely Coulombic and there is no physical transport of charge [16]. QCA is very promising for a number of reasons. Low power

consumption and high density are two important features [16, 17]. It therefore seems appropriate for design of circuits for general-purpose computation as well as for embedded applications. Digital design in QCA is the focus of this book. In the next section, we discuss the origins of QCA and also indicate the potential of the technology.

1.3 Quantum Dot Cellular Automata-Origins and Promise

This technology was introduced in 1993 [16]. It consists of an array of cells. Each cell contains several quantum dots, each of which is a nanoparticle or a crystal (made of semiconducting materials such as silicon, cadmium selenide etc.). A cell typically contains four quantum dots at the corner of a square and one central dot. Due to Coulomb interaction, electrons may occupy two ‘diametrically’ opposite quantum dots (or two antipodal sites). Labeling the quantum dots at the corners from 1 to 4 (in order), we can observe that electrons may occupy dots 1 and 3 (or 2 and 4). Each cell has bistable behaviour facilitating its use in large scale cellular arrays. The physical interaction between neighboring cells is used to implement various logic functions. The key aspect of QCA is that interaction between cells is purely Coulombic and there is no transport of charge between cells [18]. The cellular automata (CA) notion is due to the fact that state of a given cell at a particular time depends on the state of its neighbors during the previous clock cycle.

Arrays can be regular or irregular in general and can lead to realization of various logic functions. Early efforts were making on a single QCA device using different approaches. Semiconductor, molecular and magnetic implementations have been pursued. One of the early logic functions realized is the three-input majority gate. This has been used for computation of other basic logic functions such as AND and OR. Combining a majority gate with an inverter has facilitated realization of NAND, NOR and other logic elements. All circuits in QCA technology are clocked. Clocking serves as a means of synchronization. Further, circuits for realization of arithmetic functions may involve some type of wire crossing in the plane.

1.4 Metrics and Challenges in QCA-Based Digital Design

There is no consensus (among researchers) on the ‘best’ approach for evaluating digital designs in emerging nanotechnologies. Metrics used for comparing CMOS designs, viz. area, delay and power are, in general, applicable to the new technologies as well. In addition, the new technologies often have some characteristics resulting in special design goals. For example, crossover minimization is a possible goal for digital design in QCA.

To obtain QCA designs that offer low delay, area and power, it is not adequate to simply borrow approaches used for CMOS. There are challenges unique to the

technology. In particular, *interconnects* play a very important role in QCA and influence not only the overall area of a design but also the delay. Further, it is important to investigate development of approaches that apply to a large class of situations.

1.5 Contributions of the Book

Design of digital circuits in QCA is in a nascent stage. This is the focus of our book. We examine design of units that will eventually become constituents of QCA-based microprocessors. (It is to be noted that fabrication of quantum dots-based logic elements is also at a very early stage.) A large part of the material presented in this book is also applicable to other technologies (such as resonant tunneling diodes, single electron transistors etc.) that employ majority logic in some form. The contributions of the book are discussed in detail in the subsections to follow.

1.5.1 Design of Basic Circuits in QCA

An introduction to digital design in QCA is provided by considering basic logic gates such as AND, OR and XOR. This introduction serves to familiarize those who are new to QCA to various aspects such as layout design, clocking and crossover. Since arithmetic circuits for addition and other operations are composed of logic gates, QCA design (in terms of cells) of the gates provides the appropriate background. The design for gates is, in particular, used to realize a one-bit adder.

1.5.2 Efficient Design of Ripple Carry and Prefix Adders in QCA

One of the goals of the book is to highlight properties enjoyed by majority logic (which form a primitive in QCA). It turns out majority logic has a number of interesting properties that one may not find in AND, OR gates etc. The properties relevant to addition operation are explored in detail in this book. These properties are applied to obtain efficient QCA designs for adders existing in the literature, namely the Ripple Carry Adder (RCA) and various prefix adders. Bounds on the number of majority gates for n -bit RCA and n -bit Kogge-Stone, n -bit Ladner-Fischer and n -bit Brent-Kung adder are presented. We further show that the Brent-Kung adder has lower delay than the other prefix adders.

1.5.3 Efficient Design of a Custom Adder in QCA

Besides considering adders available in the literature for adaptation to QCA, one of the goals of this book is development of a custom adder for QCA technology. In this direction, the following observations have been made: the ripple carry adder is simple and its area and cell count requirements are small. However, the growth in delay as the adder size increases is worrisome. On the other hand, prefix adders offer an advantage with respect to delay but, in general, require somewhat large area (owing to increased requirement of majority gates) and have high cell count. An adder that judiciously combines ripple carry style with the prefix style seems appropriate for QCA. This is the motivation for development of a custom QCA adder which we call as a *hybrid adder*.

1.5.4 Multiplier Design in QCA

The book also investigates the QCA design of an arithmetic circuit that can perform multiplication. In particular, we develop a QCA design for the well-known Baugh-Wooley multiplier applicable to 2's complement numbers. It is shown that a direct design of the Baugh-Wooley multiplier does not turn out to be efficient (in terms of cell count, area etc.). Hence, a solution based on one-bit full-adders is proposed.

1.5.5 Computation of Discrete Hadamard Transform in QCA

While the focus of the book is on QCA design of components for a processor, it is also of interest to consider a high-level task as such. This initiates a line of enquiry on choice of algorithms for various problems. Our focus is on a problem arising in video coding. It involves the computation of a discrete orthogonal transform, namely the Discrete Hadamard Transform (DHT). The DHT has a kernel consisting only of 1's and -1's and hence admits a solution based on merely additions and subtractions. However, existing full-parallel addition methodologies (for multi-bit addition) turn out to be inefficient in QCA. We propose a bit-serial alternative and develop layouts and report simulation results.

1.5.6 Layout Development, Simulation and Robustness Studies

Besides development of theory for systematic optimization of logic gates for different arithmetic circuits, the book also presents QCA layouts developed using QCADesigner [19]. The layouts designed have been simulated. Data on cell count, area and

delay for the designs have also been obtained from the layouts. In addition, studies on thermal robustness of designs (in electrostatic QCA) have been carried out. A tutorial on layout design using QCADesigner (freeware) is also provided in an Appendix.

In the next section, we discuss the state of the art on arithmetic circuits in Quantum Dot Cellular Automata. We begin with early research on majority logic algebraic manipulation and synthesis.

1.6 Literature Survey

Our discussion of the state of the art is divided into three subsections: (i) Basics Concepts of QCA and Fabrication (ii) Majority Logic Synthesis and (iii) Digital design in QCA.

1.6.1 QCA-Theory and Fabrication

The basics of QCA technology were introduced in [16]. The authors in [16] and several colleagues presented additional information about the technology in articles that followed. In particular, [3, 20] give an account of the clocking mechanism while [18] describes aspects pertaining to power dissipation in QCA. A more recent work [21] indicates that QCA can operate at densities of 10^{14} devices/cm² and at this density, if each device dissipated $k_B T$ (where k_B is the Boltzmann constant given by 1.38×10^{-23} JK⁻¹ and T is the temperature in Kelvin), the power dissipated per unit area would be 4kW/cm². The authors in [22] explore the relation between computation and energy dissipation. Molecular QCA is explored in [23].

Besides analytical studies, there have been significant efforts on fabrication as well. The fabrication of a QCA cell using aluminium metallic dots has been reported in [24]. Extensions to a binary wire are discussed in [25]. Magnetic quantum dot fabrication is reported in [26]. Experimental demonstration of a QCA unit cell in an ion-implanted phosphorus-doped silicon system is provided in [27]. Very recently, an experimental demonstration of a 1-bit full adder using nanomagnetic logic has been reported [28]. In general, one may observe that there is considerable scope for enhancement in design as well as fabrication of QCA-based circuits.

1.6.2 Majority Logic Manipulation and Synthesis

Research on binary majority decision elements motivated by the development of devices such as parametrons and Esaki diodes has been reported as early as 1960 [29]. The author in [29] describes how ordinary Boolean algebra can be augmented to include a majority decision operator. Reference [29] also shows how to realize a

one-bit adder using three majority gates and two inverters. An extension to this work is presented in [30] where a new algebra for logical design with majority decision elements is derived and applied to one-bit adder design. An approach, based on decomposition and rearrangement, to majority element-based synthesis of networks of components having limited fan-in is presented in [31]. The strategies of [30, 31] are equally elegant with respect to synthesis of a broad class of functions. Extensions to multi-bit adders are not, however, discussed in these works. A distributive law pertaining to majority logic is described in [32].

While [30] presents an *algebraic* method for logical design, the authors in [33] present a *geometric* method that uses Veitch diagrams (and extensions) for synthesis using i -input majority gates of a variety of n -argument switching functions. An approach based on the notion of *Logically Passive Self-Dual (LPSD)* has been presented in [34]. An extension to this work is presented in [35]. All of these are interesting from a theoretical point of view and offer valuable insight for majority logic-based design. However, their focus is on simplification of Boolean expressions rather than on special adder designs.

1.6.3 QCA-Based Digital Design

A simulation tool for QCA has been reported in [19]. The authors in [36] present an evaluation of several 3-variable Boolean functions and conclude with study of a 1-bit full adder. A synthesis tool called Majority Logic Synthesizer (MALS) is described in [37]. An approach to QCA design based on development of code in hardware description languages is suggested in [38]. The authors in [38] suggest that this approach helps to get power and area estimates of QCA designs for fairly complex high-level tasks keeping also in view the actual implementation (magnetic/molecular). The authors in [39] present a methodology for obtaining a minimal majority expression for arbitrary three-variable Boolean functions. A performance comparison of some quantum-dot cellular automata adders is presented in [40]. Modular design of conditional sum adders has been studied in [41]. The authors in [42] have presented a QCA design methodology based on traditional CMOS circuits design flow and a SPICE model. Probabilistic modeling of adder circuits using Bayesian networks is presented in [43]. Cell count, area and *latency* have been studied for multi-bit adders (especially Ripple Carry Adder (RCA) and Carry Lookahead Adder (CLA)) in [44]. Robust QCA adder designs that exploit proper clocking schemes are proposed in [45] but they do not study special adders such as prefix adders. Probabilistic analysis of molecular quantum-dot cellular adders is presented in [46]. Reliability of magnetic QCA adders has been studied via probabilistic transfer matrices in [47]. Robust adders based on QCA are described in [48]. Energy dissipation per clock cycle in QCA adder circuits is studied in [49]. A modular design of 2^n -to-1 multiplexers in QCA is presented in [50]. An interesting extension of [44] to a new type of adder called Carry Flow Adder (CFA) is presented in [51]. The CFA is an enhancement of the basic ripple carry adder to obtain low delay as well as area. The authors in [52]

present a novel design methodology to obtain low delay or low area adders in QCA. They apply their methodology to obtain efficient carry lookahead and Brent-Kung adder designs in QCA. It can, in general, be observed from prior research that the ripple carry adder and its variations do well in QCA technology (unlike in CMOS). This may be attributed to simple (and low) interconnect requirement in the ripple carry approach. Work on QCA designs for higher-level tasks is limited. An approach to matrix multiplication in QCA is presented in [17].

1.7 Organization of the Book

Next chapter presents QCA terminology appropriate for the remainder of the book.

In Chap. 3, we start the QCA design of digital circuits from logic gates.

In Chap. 4, we explore properties of majority logic and use them to first design the simplest multi-bit adder, namely the ripple carry adder in QCA. We then consider advanced adders (that yield low delay), namely prefix adders for QCA realization.

In Chap. 5, we develop a custom adder for QCA called as the *hybrid adder*. The objectives in seeking a dedicated adder for QCA technology are discussed. QCA design of the hybrid adder for different sizes is then examined. Layouts of the designs are also presented along with simulation results.

In Chap. 6, we extend the work on adders to design of a QCA-based multiplier.

In Chap. 7, we consider QCA-based computation of the discrete Hadamard transform.

Chapter 8 is devoted to study of thermal robustness of QCA-based adders.

Chapter 9 is devoted to a summary of the work and a discussion of extensions.

A tutorial on layout design using QCADesigner is presented as an appendix. Videos on creating layouts in QCADesigner are available at the URL http://www.ee.iitm.ac.in/~sridhara/QCADesigner_Videos.

References

1. A. Seabaugh, The tunneling transistor. *IEEE Spectr.* 31–34 (2013)
2. International Technology Roadmap for Semiconductors (ITRS). Technical report, Semiconductor Industry Association (SIA), (2013)
3. W. Porod, Quantum-dot devices and quantum-dot cellular automata. *J. Frankl. Inst.* **334B**(5/6), 1147–1175 (1997)
4. K. Galatsis, A. Khitun, R. Ostroumov, K.L. Wang, W.R. Dichtel, E. Plummer, J.F. Stoddart, J.I. Zink, J.Y. Lee, Y.H. Xie, K.W. Kim, Alternate state variables for emerging nanoelectronic devices. *IEEE Trans. Nanotechnol.* **8**(1), 66–75 (2009)
5. A. Seabaugh, Q. Zhang, Low-voltage tunnel transistors for beyond CMOS logic. *Proc. IEEE* **98**(12), 2095–2110 (2010)
6. J.P. Leburton, J. Kolodzey, S. Biggs, Bipolar tunneling field-effect transistor: a three-terminal negative differential resistance device for high-speed applications. *Appl. Phys. Lett.* **52**(9), 1608–1620 (1988)

7. T. Baba, Proposal for surface tunnel transistors. *Jpn. J. Appl. Phys.* **31**(4B), L455–L457 (1992)
8. S.J. Tans, A.R.M. Verschueren, C. Dekker, Room-temperature transistor based on a single carbon nanotube. *Nature* **393**, 49–52 (1998)
9. T. Yamada, Analysis of submicron carbon nanotube field-effect transistors. *Appl. Phys. Lett.* **76**(5), 628–630 (2000)
10. A. Batchtold, P. Hadley, T. Nakanishi, C. Dekker, Logic circuits with carbon nanotube transistors. *Science* **294**, 1317–1320 (2001)
11. K.K. Likharev, Single-electron devices and their applications. *Proc. IEEE* **87**(4), 606–632 (1999)
12. K.L. Wang, A. Khitun, K. Galatsis, More than Moore’s law: Nanofabrics and architectures. *Proceedings of IEEE Bipolar/BiCMOS Circuits and Technology Meeting (BCTM’07)*, pp. 139–143 (2007)
13. A. Khitun, M. Bao, K.L. Wang, Spin wave magnetic nanofabric: a new approach to spin-based logic circuitry. *IEEE Trans. Magn.* **44**, 2141–2152 (2008)
14. S. Bandyopadhyay, V.P. Roychowdhury, Granular nanoelectronics. *IEEE Potentials*, 8–11 (1996)
15. A. Khitun, K. Wang, Nano scale computational architectures with spin wave bus. *Superlattices Microstruct.* **38**, 184–200 (2005)
16. C.S. Lent, P.D. Tougaw, W. Porod, G. Bernstein, Quantum cellular automata. *Nanotechnology* 49–57, (1993)
17. J.D. Wood, D. Tougaw, Matrix multiplication using quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **10**(5), 1036–1042 (2011)
18. J. Timler, C.S. Lent, Power gain and dissipation in quantum-dot cellular automata. *J. Appl. Phys.* **91**(2), 823–831 (2002)
19. K. Walus, T. Dysart, G. Jullien, R. Budiman, QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **3**(1), 26–29 (2004)
20. C.S. Lent, P.D. Tougaw, A device architecture for computing with quantum dots. *Proc. IEEE* **85**(4), 541–557 (1997)
21. E.P. Blair, E. Yost, C.S. Lent, Power dissipation in clocking wires for clocked molecular quantum-dot cellular automata. *J. Comput. Electron.* (2009). doi:10.1007/s10825-009-0304-0
22. J. Timler, C.S. Lent, Maxwell’s demon and quantum-dot cellular automata. *J. Appl. Phys.* **94**(2), 1050–1060 (2003)
23. M. Liu, Y. Lu, C. Lent, Molecular quantum-dot cellular automata: from molecular structure to circuit dynamics. *J. Appl. Phys.* **102**, 034311–034317 (2007)
24. A. Orlov, I. Amlani, G.H. Bernstein, C.S. Lent, G.L. Snider, Realization of a functional cell for quantum dot cellular automata. *Science* 928–930 (1997)
25. A. Orlov, I. Amlani, G. Toth, C.S. Lent, G.H. Bernstein, G.L. Snider, Realization of a functional cell for quantum dot cellular automata. *Science* 928–930 (1997)
26. R. Cowburn, M. Welland, Room temperature magnetic quantum cellular automata. *Science*, pp. 1466–1468 (2000)
27. M. Mitic, M.C. Cassidy, K.D. Peterson, R.P. Starrett, E. Gauja, R. Brenner, R.G. Clark, A.S. Dzurak, Demonstration of a silicon-based quantum cellular automata cell. *Appl. Phys. Lett.* **89**, 013503–013511 (2006)
28. S. Breitkreutz, J. Kiermaier, I. Eichwald, C. Hildbrand, G. Csaba, D. Schmitt-Landsiedel, M. Becherer, Experimental demonstration of a 1-bit full adder in perpendicular nanomagnetic logic. *IEEE Trans. Magn.* **49**(7), 4464–4467 (2013)
29. R. Lindaman, A theorem for deriving majority-logic networks within an augmented Boolean algebra. *IEEE Trans. Electron. Comput.* **EC-9**(3), 338–342 (1960)
30. M. Cohn, R. Lindaman, Axiomatic majority-decision logic. *IEEE Trans. Electron. Comput.* **EC-10**(1), 17–21 (1961)
31. F. Miyata, Realization of arbitrary logical functions using majority elements. *IEEE Trans. Electron. Comput.* **EC-12**(3), 183–191 (1963)

32. S.B. Akers, On the algebraic manipulation of majority logic. *IEEE Trans. Electronic Comput.* **EC-10**(4), 779–779 (1961)
33. H.S. Miller, R.O. Winder, Majority logic synthesis by geometric methods. *IEEE Trans. Electron. Comput.* **EC-11**(1), 89–90 (1962)
34. S.B. Akers, Synthesis of combinational logic using three-input majority gates, in *Proceedings of the Third Annual Symposium on Switching Circuit Theory and Logical Design*, pp. 149–158, 7–12 (1962)
35. E.M. Riseman, A realization algorithm using three-input majority elements. *IEEE Trans. Electron. Comput.* **EC-16**(4), 456–462 (1967)
36. R. Zhang, K. Walus, W. Wang, G.A. Jullien, A method of majority logic reduction for quantum cellular automata. *IEEE Trans. Nanotechnol.* **3**(4), 443–450 (2004)
37. R. Zhang, P. Gupta, N.K. Jha, Synthesis of majority and minority networks and its applications to QCA, TPL, and SET based nanotechnologies, in *Proceedings of International Conference on VLSI Design*, pp. 229–234 (2005)
38. M. Awais, M. Vacca, M. Graziano, M.R. Roch, G. Masera, Quantum dot cellular automata check node implementation for LDPC decoders. *IEEE Trans. Nanotechnol.* **12**(3), 368–377 (2013)
39. K. Kong, Y. Shang, R. Lu, An optimized majority logic synthesis methodology for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **9**(2), 170–183 (2010)
40. R. Zhang, K. Walus, W. Wang, G.A. Jullien, Performance comparison of quantum-dot cellular automata adders, in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2005)*, pp. 2522–2526 (2005)
41. H. Cho, E. Swartzlander, Modular design of conditional sum adders using quantum-dot cellular automata, In *Proceedings of Sixth IEEE Conference on Nanotechnology (IEEE-NANO 2006)*, pp. 363–366 (2006)
42. R. Tang, F. Zheng, Y-B. Kim, QCA-based nano circuits design [adder design example], in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 2527–2530 (2005)
43. S. Bhanja, S. Sarkar, Probabilistic modeling of QCA circuits using Bayesian networks. *IEEE Trans. Nanotechnol.* **5**(6), 657–670 (2006)
44. H. Cho, E.E. Swartzlander, Adder designs and analyses for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **6**(3), 374–383 (2007)
45. K. Kim, K. Wu, R. Karri, The robust QCA adder designs using composable QCA building blocks. *IEEE Trans. Comput. -Aided Des. Integr. Circuits Syst.* **26**(1), 176–183 (2007)
46. T.J. Dysart, P.M. Kogge, Probabilistic analysis of a molecular quantum-dot cellular automata adder, in *Proceedings of IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pp. 478–486 (2007)
47. T. Dysart, P.M. Kogge, Analyzing the inherent reliability of moderately sized magnetic and electrostatic QCA circuits via probabilistic transfer matrices. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **17**(4), 507–516 (2009)
48. I. Hanninen, J. Takala, Robust adders based on quantum-dot cellular automata, in *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 391–396 (2007)
49. S. Srivastava, S. Sarkar, S. Bhanja, Estimation of upper bound of power dissipation in QCA circuits. *IEEE Trans. Nanotechnol.* **8**(1), 116–127 (2009)
50. V.A. Mardiris, I.G. Karafyllidis, Design and simulation of modular 2^n to 1 quantum-dot cellular automata (QCA) multiplexers. *Int. J. Circuit Theor. Appl.* **38**, 771–785 (2010)
51. H. Cho, E.E. Swartzlander, Adder and multiplier designs in quantum-dot cellular automata. *IEEE Trans. Comput.* **58**(6), 721–727 (2009)
52. S. Perri, P. Corsonello, New methodology for the design of efficient binary addition circuits in QCA. *IEEE Trans. Nanotechnol.* **11**(6), 1192–1200 (2012)

Chapter 2

QCA Terminology

In this chapter, we present key aspects of the QCA technology. In particular, we discuss the components of QCA. We also list the logic primitives in the QCA technology and indicate the importance of *clocking* in the QCA paradigm.

2.1 QCA Basics

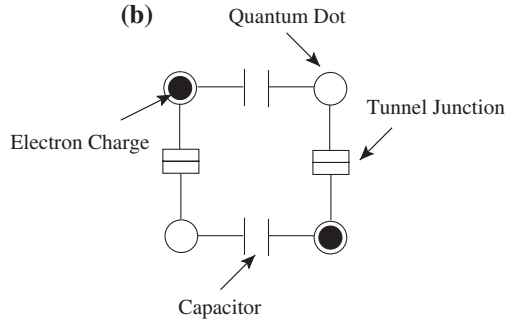
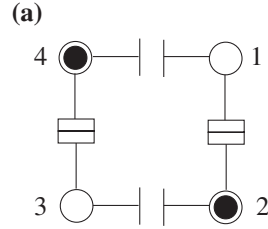
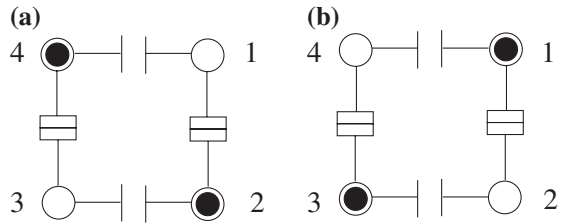
2.1.1 Cell, Quantum Dot and Tunnel Junctions

The basic element in QCA is a cell. Each cell represents a bit through an appropriate configuration of charge. A cell is shown in Fig. 2.1a. It consists of four metal islands (conductors) that constitute what are known as *quantum dots*. Each quantum dot corresponds to a semiconductor nanostructure that exhibits quantum mechanical properties in view of its size. Four dots form a QCA cell with one electron each in two of the four dots occupying “diametrically opposite” locations. Electrons occupy opposite corner (diagonal) quantum-dots because Coulomb repulsion is less compared to the scenario when they are in adjacent quantum dots. Dots are coupled by tunnel junctions. The cellular automata (CA) notion is due to the fact that the state of a given cell at a particular time depends on the state of its neighbors during the previous clock cycle.

Electrons can tunnel between the dots (on application of an appropriate potential). However, electrons are unable to leave the cell. The tunneling happens through *tunnel junctions* shown in Fig. 2.1b.

2.1.2 How Are Different Logic States Obtained?

In a QCA cell, two ground state polarizations (labelled -1 and 1) are possible depending upon the position of (electron) charge. The polarization is calculated via Eq. (2.1).

Fig. 2.1 Basic QCA cell**Fig. 2.2** Representation of logic states in QCA, **a** logic-0, **b** logic-1

$$P = \frac{(p_1 + p_3) - (p_1 + p_3)}{p_1 + p_3 + p_1 + p_3} \quad (2.1)$$

Equation (2.1) shows the calculation of polarization of a QCA cell. Here p_i is the charge of the i th quantum-dot ($p_i = 1$ if an electron is present, otherwise it is 0). These polarizations are represented by logic-0 and logic-1 (in other words, binary 0 and binary 1) as shown in Fig. 2.2.

One can change the logic state of a cell by merely applying a negative potential to a lead near the quantum dot occupied by an electron. This causes the next cell to change the ground states to reduce Coulomb repulsion. Figure 2.3a, b show the propagation of logic-1 and logic-0 respectively from the left end to the right due to Coulomb repulsion. Therefore, in QCA, there is (at most) one position change of electrons in quantum dots (within a cell) to transfer information from one end to the other, whereas in traditional CMOS, there is explicit transfer of charge (from one end to the other) for propagation of information. The arrangement of QCA cells in

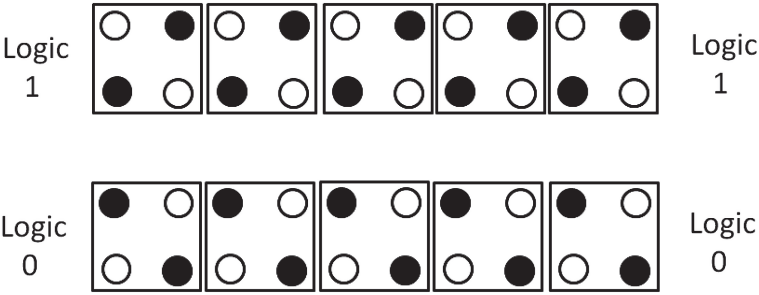


Fig. 2.3 Operation of a QCA wire. **a** QCA wire propagating logic 1. **b** QCA wire propagating logic 0

Fig. 2.3 shows a QCA wire. By application of logic 0 or logic 1 to a QCA cell (at the input), we propagate information to the output ‘side’ taking advantage of the force of repulsion.

2.2 Logic Primitives in QCA

In the previous section, we discussed about operation of a QCA cell and QCA wire. Here we discuss the elements of interest in digital design, namely logic gates. By arranging QCA cells as shown in Fig. 2.4a, b, we can implement an inverter (or a NOT gate) and a majority gate respectively. Figure 2.4a shows that upon applying input “0”, we get output of “1” which amounts to flipping of the input. This corresponds to realization of an inverter. The output of the gate shown in Fig. 2.4b is the majority of its inputs: for example in Fig. 2.4b, “001” is applied to obtain an output of “0” which

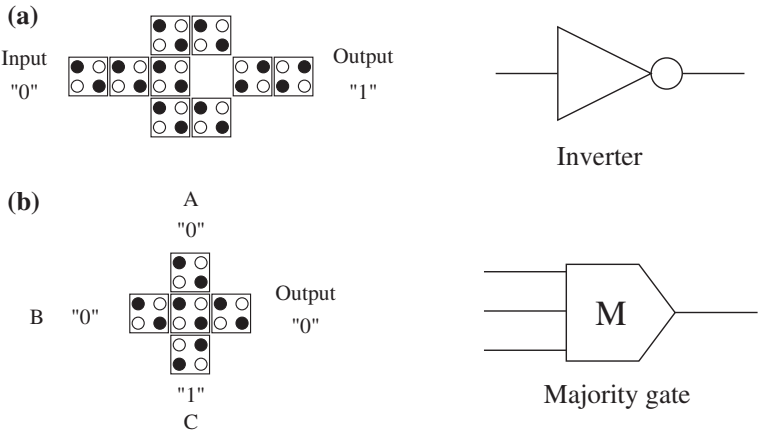


Fig. 2.4 Basic gates in QCA. **a** Inverter. **b** Majority gate $M(A, B, C) = AB + BC + AC$

represents the majority of the inputs. Suppose A , B and C are inputs to the majority gate, then the output is given by $M(A, B, C) = AB + BC + AC$. By setting one input (say C) to 0, the majority gate acts as an AND gate. In other words, $M(A, B, 0) = AB$. Similarly, by setting one of the three inputs to 1, the majority gate functions as an OR gate ($M(A, B, 1) = A + B$). A majority gate has advantages compared to an AND or OR gate, since three inputs enable realization of certain arithmetic functions (such as a carry in a one-bit full-adder) with just one majority gate. Further, a majority gate requires only the same number of cells in QCA as an AND gate and OR gate. This can be contrasted with MOSFET technology (where the number of transistors required to realize a 3-input majority gate is much more than that required for some other gates for example). Note that a majority gate along with an inverter is a universal set and can be used to realize any Boolean function just like NAND or NOR. Further, we will observe that a rich set of properties for majority gates (described in the subsequent chapters) enables very low-complexity QCA design of various arithmetic circuits.

Remark 2.1 In principle, the majority gate permits an odd number of inputs. However, for a 2-D QCA cell, we can have only three inputs (and 1 output) given that there are only four ‘directions’. Extensions to 3-D would permit additional inputs. Further, for arithmetic, three inputs seem to be appropriate. For example, the carry function in a one-bit full-adder takes three inputs.

2.2.1 Clocking in QCA-Role and Types

Several approaches have been suggested for computation with an array of QCA cells. One approach is based on transferring the array to an excited state from a ground state by merely applying input data (without explicit clocking). The array is expected to settle to a new ground state. However, sometimes the transition may result in a metastable intermediate state. To facilitate transfer to a new ground state, another approach based on clocking has been suggested [1]. Clocking (by application of an appropriate voltage to a cell) leads to adjustment of tunneling barriers between quantum dots for transfer of electrons between the dots. Figure 2.5 depicts clocking of a QCA cell. Here V_c corresponds to the clock voltage. Clocking is performed in one of two ways: zone clocking and continuous clocking. In zone clocking [2], each QCA cell is clocked using a four-phase clocking scheme as shown in Fig. 2.6. The four phases correspond to *switch*, *hold*, *release* and *relax*. In the switch phase, cells begin unpolarized and with low potential barriers but the barriers are raised during this phase. In the hold phase, the barriers are held high while in the release phase, the barriers are lowered. In the last phase, namely relax, the barriers remain lowered and keep the cells in an unpolarized state. An alternative to zone clocking, called continuous clocking, involves generation of a potential field by a system of submerged electrodes. The former clocking scheme is adopted in this book since the CAD tool used for simulation supports zone clocking and further, prior works on adders are only based on this clocking scheme. Figure 2.7 shows the interdot barriers

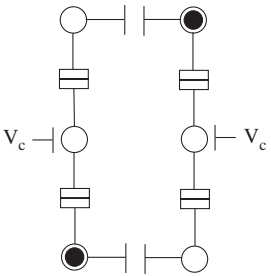


Fig. 2.5 QCA cell with clocking

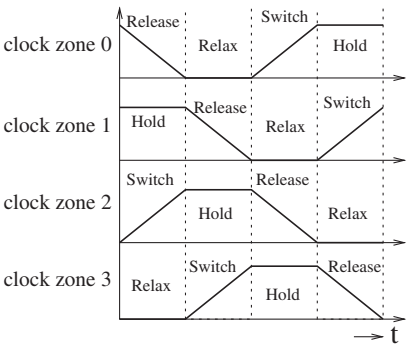


Fig. 2.6 QCA clock zones

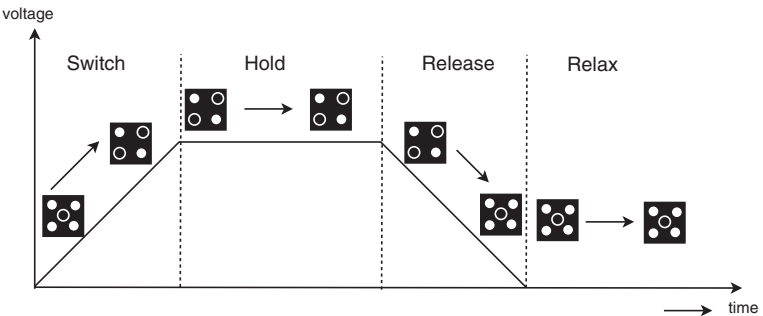


Fig. 2.7 Interdot barriers in a clocking zone

in a clocking zone while Fig. 2.8 shows the operation of a wire in different clock zones.

A key difference between circuit design in QCA and that in conventional CMOS technology is that a circuit in QCA has no control over the clocks [3] unlike in CMOS. Hence, information is merely *transmitted* through each cell and not retained. Each cell “erases” its own state through every clock cycle. Further, every logic element in a QCA circuit is clocked.

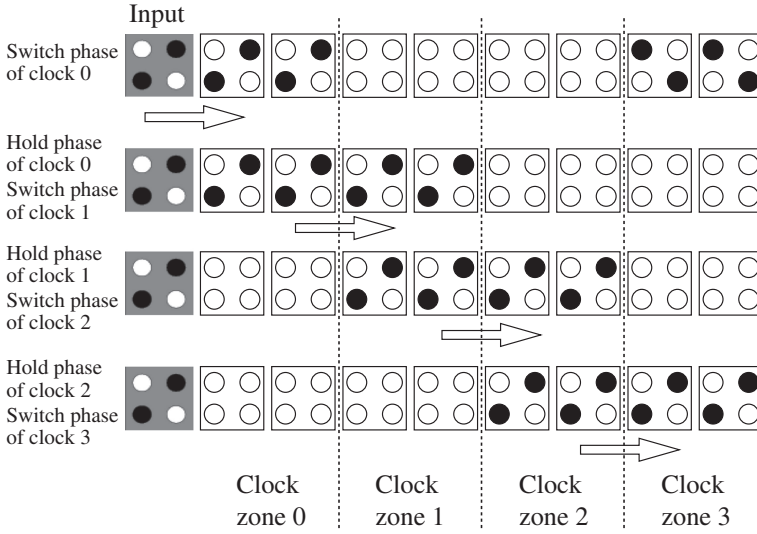


Fig. 2.8 Operation of a wire in different clock zones

2.2.2 Crossovers in QCA

A QCA design permits two options for crossover, termed coplanar crossover and multilayer crossover. While the coplanar crossover uses only one layer but involves usage of two cell types (termed regular and rotated), the multilayer crossover uses more than one layer of cells (analogous to multiple metal layers in a conventional IC). Multilayer crossover is used in this book for wire crossings since we can effectively cross signals over on another layer and the extra layers of QCA can be used as active components of the circuit [2]. Further, multilayer QCA circuits can potentially consume much less area as compared to planar circuits [2].

2.3 Tool for QCA Simulation

In this book, a tool for designing layouts and performing simulations called QCADesigner [3] is used. The tool allows us to describe different components of a digital design. The tool also permits specification of clock zones for a design. Four clocking zones in QCA are represented by four different colours in QCADesigner. In particular, clock zone 0 is represented in green, clock zone 1 in magenta, clock zone 2 in blue and clock zone 3 in white. When cells are first input to a design, they are in clock zone 0. To apply a different clock (zone) to a cell or an array of cells, we need to select the cells and specify the clock. A snapshot of QCADesigner (with an inverter on the left and a majority gate on the right) is shown in Fig. 2.9.

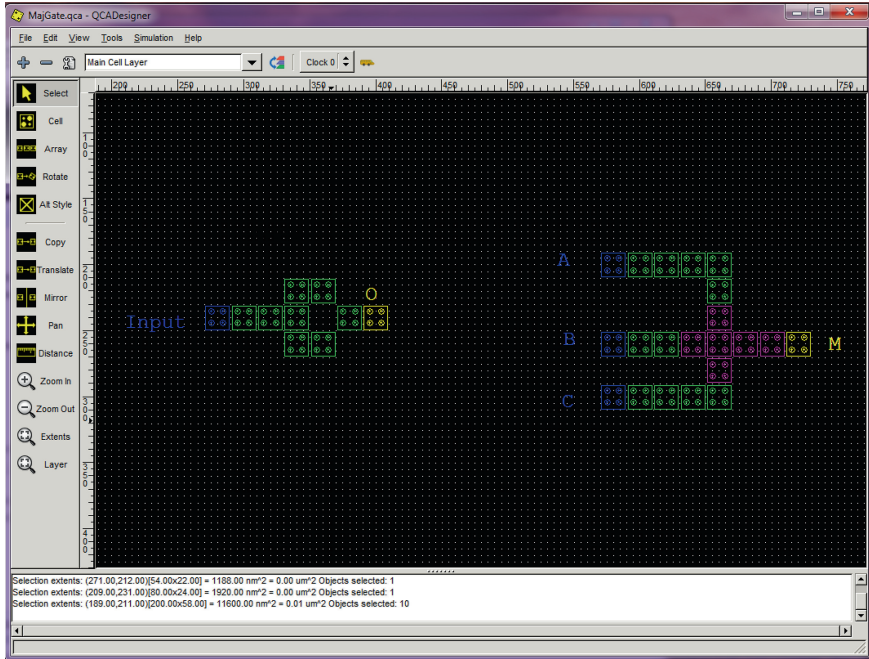


Fig. 2.9 Illustration for QCADesigner

2.4 Summary

In this chapter, we have presented QCA terminology that will be used in the remainder of this book. In the next chapter, we present design of several basic logic elements in QCA.

References

1. C.S. Lent, P.D. Tougaw, A device architecture for computing with quantum dots. *Proc. IEEE* **85**(4), 541–557 (1997)
2. K. Walus, G.A. Jullien, Design tools for an emerging SoC technology: quantum-dot cellular automata. *Proc. IEEE* **94**(6), 1225–1244 (2006)
3. K. Walus, T. Dysart, G. Jullien, R. Budiman, QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **3**(1), 26–29 (2004)

Chapter 3

Design of Basic Digital Circuits in QCA

Designs in QCA are based on three primitives: *3-input majority gate*, *inverter* and *wire*. In this chapter, we show how elementary circuits can be designed using these primitives. We also depict layouts designed using QCADesigner [1] and results of simulation of the circuit elements. The procedures to design the layout are discussed separately in an Appendix.

3.1 Design of Logic Gates in QCA

The three-input majority gate, with inputs A , B and C is given by the logic function $M(A, B, C) = AB + BC + CA$. Using this, we can realize a two-input OR gate as $M(A, B, 1)$. The QCA cell-based realization involves use of only nine cells for the three-input majority gate as well as the (two-input) OR gate. The QCADesigner layout for the 2-input OR is depicted in Fig. 3.1. In QCADesigner terminology, input C for a 2-input OR gate is assigned a *fixed polarization* of 1.0. In general, layout design also involves specifying appropriate *clock zones*. In this case, all the cells are set to *Clock 0*. The complexity of the QCA design depicted in Fig. 3.1 can be expressed by the cell count, area as well as delay. An outline of the procedure for calculating cell count and other parameters is provided in the Appendix.

The results of simulation of a 2-input OR gate in QCADesigner are shown in Fig. 3.2. It can be observed that the output is the OR of the two input values. (We use the label *Output* in the simulation waveforms in place of $M(A, B, 1)$ to facilitate easy interpretation.) Similarly, the AND of two inputs, A and B , is realized as $M(A, B, 0)$. In essence, setting the third input C to 0 leads to AB . The layout for a 2-input AND gate is shown in Fig. 3.3 while the simulation results are shown in Fig. 3.4.

Two-input NAND and two-input NOR in QCA are (each) realized using one majority gate followed by one inverter. The layouts for 2-input NAND and 2-input

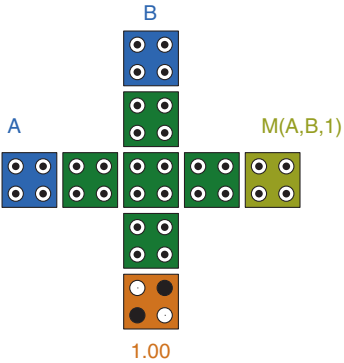


Fig. 3.1 QCADesigner layout for 2-input OR gate

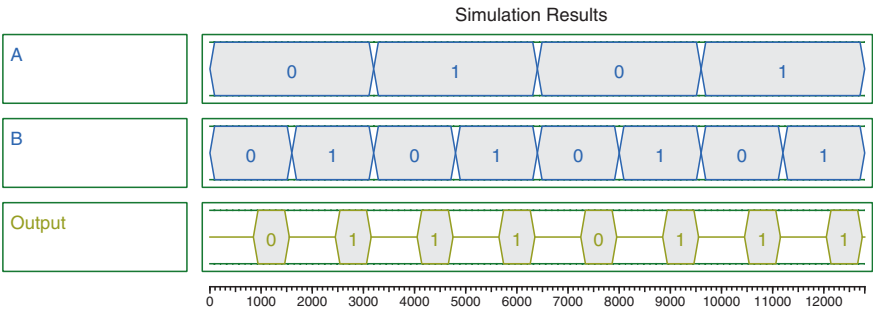


Fig. 3.2 Simulation of QCA 2-input OR gate

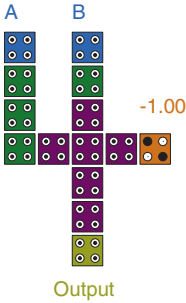


Fig. 3.3 QCADesigner layout for 2-input AND gate

NOR are shown in Figs.3.5 and 3.7. The corresponding waveforms are shown in Figs.3.6 and 3.8 respectively. A two-input XOR can be realized in more than one way. A direct solution is one that involves “crossover”. The layout and waveforms for 2-input XOR with multilayer crossover are depicted in Figs.3.9 and 3.10 respectively. The design requires three majority gates and two inverters.

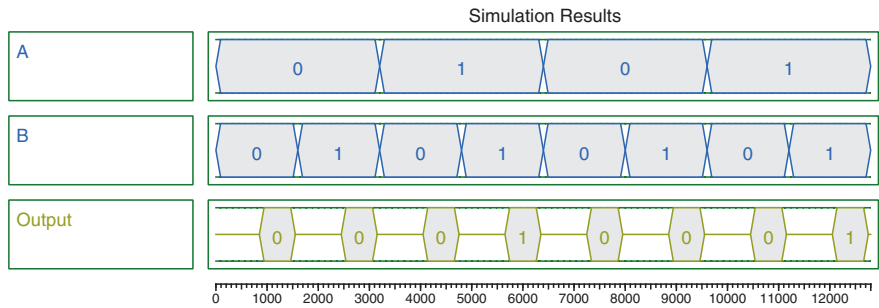


Fig. 3.4 Simulation of QCA 2-input AND gate

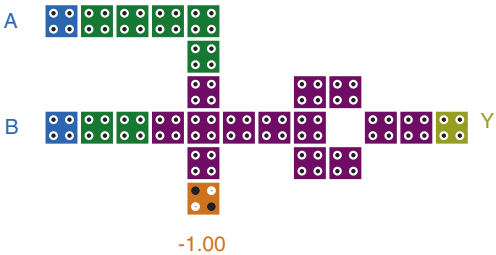


Fig. 3.5 QCADesigner layout for a 2-input NAND gate

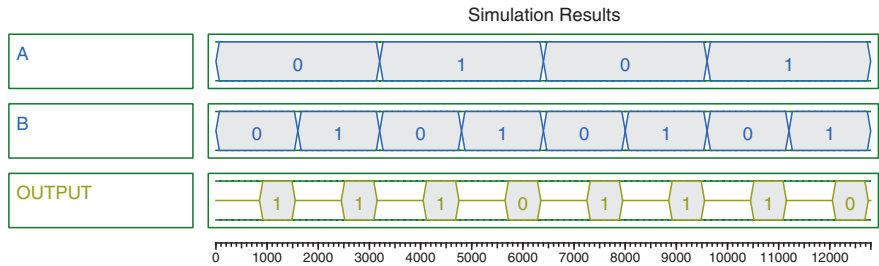


Fig. 3.6 Simulation of a 2-input NAND gate

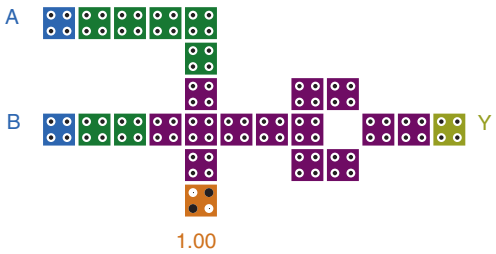


Fig. 3.7 QCADesigner layout for a 2-input NOR gate

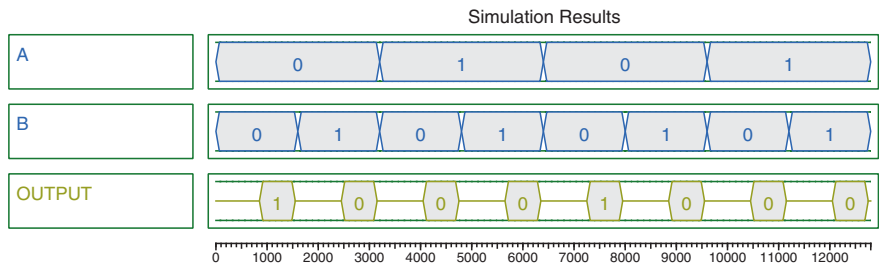


Fig. 3.8 Simulation of a 2-input NOR gate

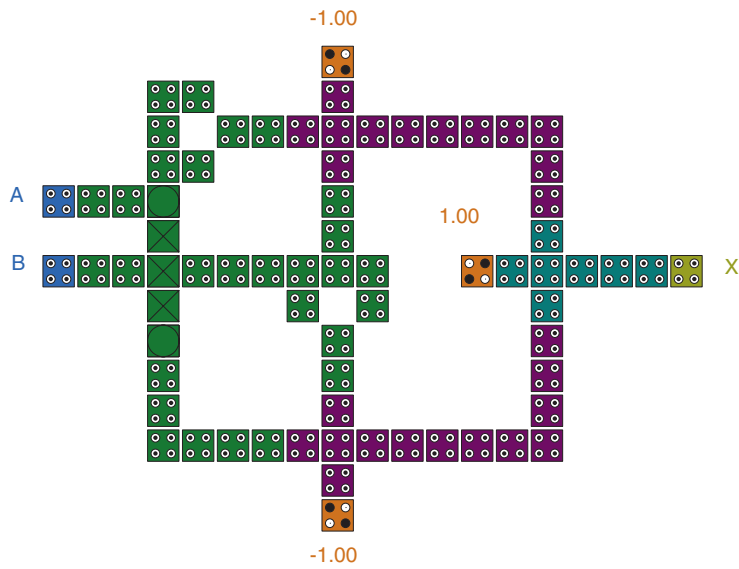


Fig. 3.9 QCADesigner layout for a 2-input XOR gate

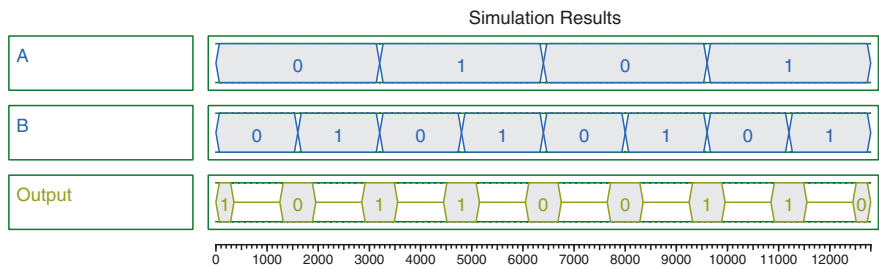


Fig. 3.10 Simulation of a 2-input XOR gate

3.2 Design of a Multiplexer in QCA

Another fundamental unit in digital circuits is a multiplexer. A 2-to-1 multiplexer that implements $C = A\bar{S} + BS$ is depicted in Fig. 3.11. It can be observed that the design needs three majority gates and one inverter. The layout is somewhat similar to the one for an XOR gate. (The XOR design uses one additional inverter.) Simulation results for the multiplexer are shown in Fig. 3.12.

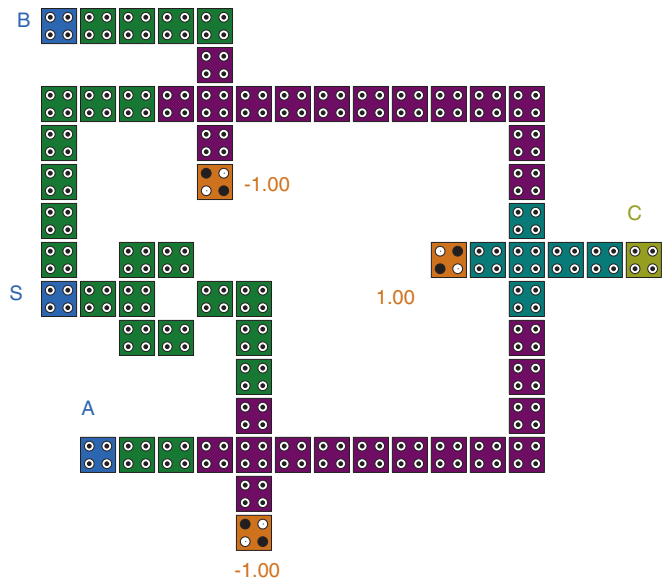


Fig. 3.11 QCADesigner layout for a 2-to-1 multiplexer

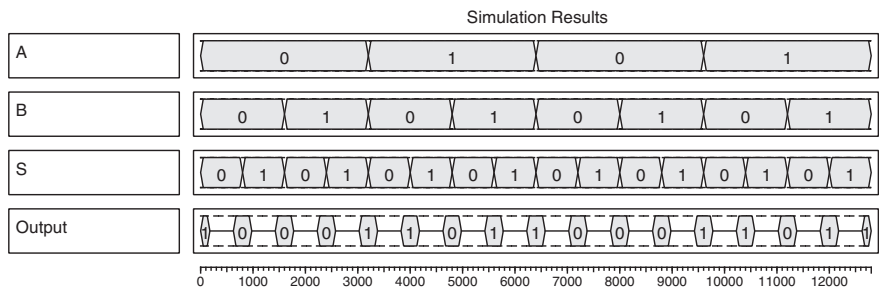


Fig. 3.12 Simulation of a 2-to-1 multiplexer

3.3 Design of a One-Bit Full-Adder in QCA

A full-adder is a circuit that adds three bits. Suppose the three inputs to a full-adder are denoted by X , Y and C_{in} . Then the sum S and output carry C_{out} for the full-adder are given by Eqs. (3.1) and (3.2).

$$S = X \oplus Y \oplus C_{in} \quad (3.1)$$

$$C_{out} = XY + YC_{in} + XC_{in} \quad (3.2)$$

A direct QCA realization of a full-adder from these equations for sum and carry is as follows. Each XOR in the sum equation requires three majority gates and two inverters. The expression for C_{out} is the same as that for a majority gate. Hence, seven majority gates and four inverters are required for a direct realization of a full-adder. The QCA layout and simulation waveforms are given in Figs. 3.13 and 3.14

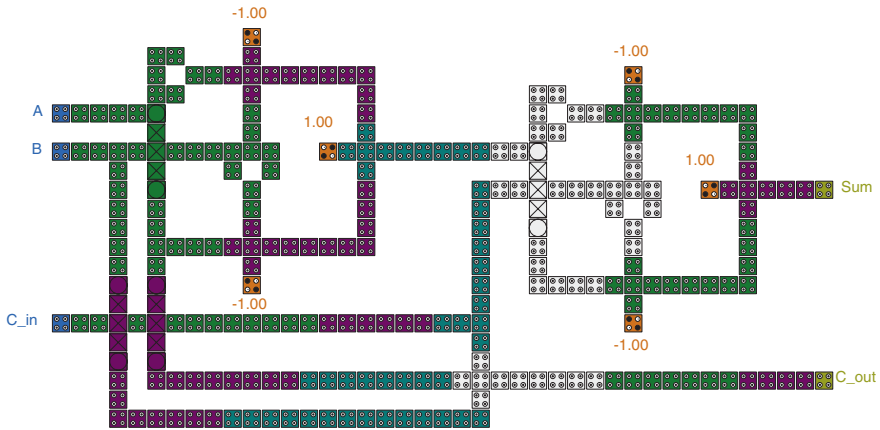


Fig. 3.13 QCADesigner layout for a full-adder (direct realization)

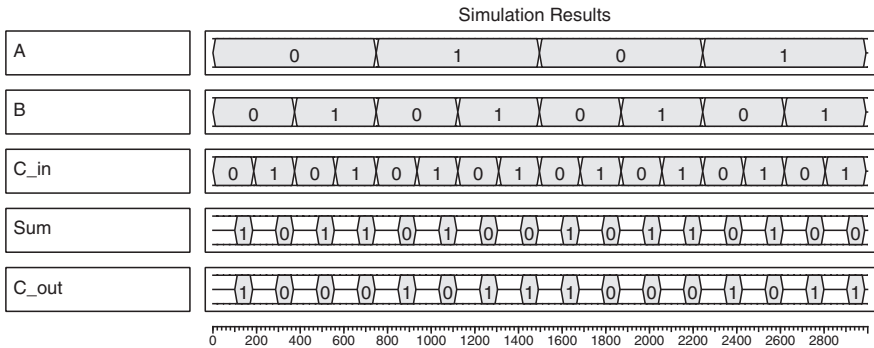


Fig. 3.14 Simulation of a QCA full-adder

respectively. The complexity of the layout can be measured in terms of the cell count, area and delay. Details of measurement are presented in the Appendix.

Remark 3.1 The QCA-based design ideas presented for a one-bit full-adder can be extended to a one-bit full-subtractor. The design of a one-bit full-subtractor involves obtaining the difference and output borrow, denoted by d and b_{out} respectively, given inputs x , y and b_{in} . In particular, b_{out} and d are given by Eqs. (3.3) and (3.4).

$$b_{out} = \bar{x}y + \bar{x}b_{in} + yb_{in} \quad (3.3)$$

$$\begin{aligned} d &= xyb_{in} + x\bar{y}\bar{b}_{in} + \bar{x}y\bar{b}_{in} + \bar{x}\bar{y}b_{in} \\ &= x \otimes y \otimes b_{in} \end{aligned} \quad (3.4)$$

Equation (3.3) reveals that one majority gate and one inverter are required for QCA realization of b_{out} . Equation (3.4) indicates that six majority gates and four inverters are required for d (We note that d is expressed in terms of XOR and each XOR requires three majority gates and two inverters). Altogether, seven majority gates and five inverters are required for realization of a one-bit subtractor.

3.4 Design of a Flip-Flop in QCA

Design of a memory element in QCA is somewhat different from that of basic logic gates or multiplexers. In particular, a flip-flop is realized by creation of loops. The design of a flip-flop using coplanar crossover is presented in [2]. We present a design with multilayer crossover. The design comprises of only three majority gates and one inverter. The layout for the D flip-flop with data input denoted by D and clock input

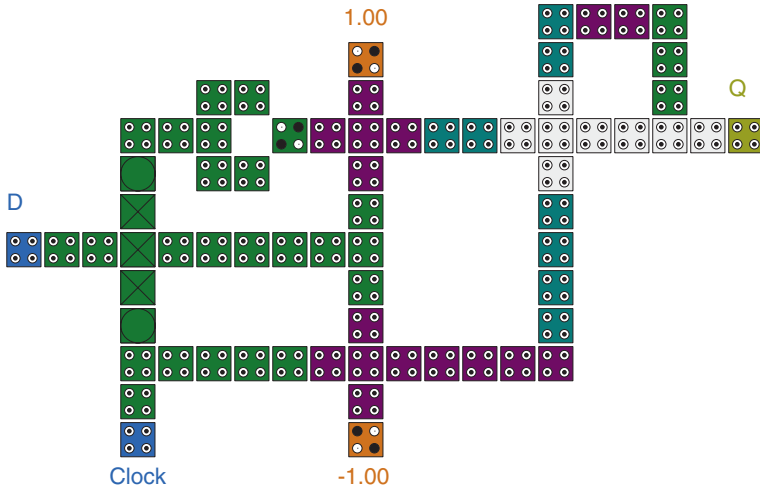


Fig. 3.15 QCADesigner layout for a D flip-flop

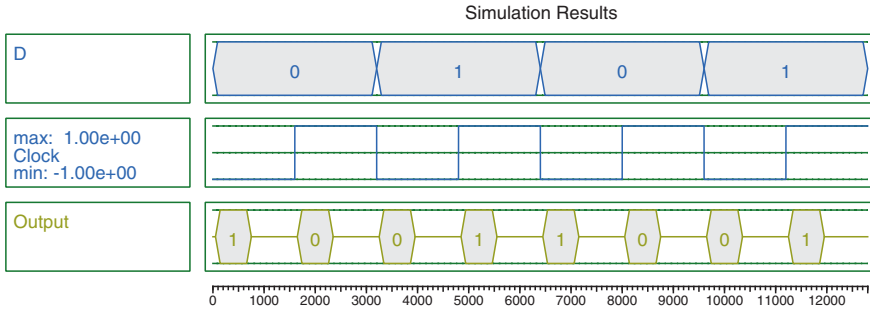


Fig. 3.16 Simulation waveforms for a D flip-flop

denoted by *Clock* is depicted in Fig. 3.15. The simulation results for a QCA-based D flip-flop are shown in Fig. 3.16.

3.5 Summary

In this chapter, we have examined QCA designs for basic digital circuits. In particular, designs for OR, AND, NAND, NOR, XOR, 2-to-1 multiplexer, a one-bit full-adder and a flip-flop have been presented. In the next chapter, we consider the design of multi-bit adders in QCA.

References

1. K. Walus, T. Dysart, G. Jullien, R. Budiman, QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **3**(1), 26–29 (2004)
2. A. Vetteth, K. Walus, V.S. Dimitrov, G.A. Jullien, Quantum-dot cellular automata of flip-flops, in *Proceedings of the 2003 National Conference on Communications (NCC 2003)*, (2003)

Chapter 4

Design of Ripple Carry and Prefix Adders in QCA

In this chapter, we consider multi-bit adder design in QCA. The simplest multi-bit adder (with low area) that has been explored for CMOS implementation in the past is the ripple carry adder. We describe how one can get an efficient QCA design. We also consider advanced adders (with low delay) based on the prefix concept [1]. Some of the results in this chapter have been reported in [2].

4.1 Design of the Ripple Carry Adder (RCA) in QCA

A ripple carry adder is formed by cascading several 1-bit full-adders. A 4-bit ripple carry adder to add numbers $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ is depicted in Fig. 4.1. The input carry to the rightmost one-bit adder is denoted by C_0 . The output carries and sums are denoted by $C_i, i = 1, \dots, 4$ and $S_j, j = 0, \dots, 3$ respectively. This can be extended to an n -bit adder in the same fashion. It is clear that efficient QCA design of the RCA depends on the design of the one-bit full-adder. One approach to full-adder design in QCA is based on directly replacing XOR, AND and OR gates in the realization by majority gates and inverters as mentioned in Chap. 3.

Instead, we can devise efficient (low-complexity) solutions in QCA considering the sum and carry equations for a one-bit full-adder. In particular, we can derive expressions for sum and carry in terms of the 3-input majority function, $M(a, b, c)$.

4.1.1 New Results on Majority Logic

We present three new lemmas on reduction of the number of majority gates which are directly of interest in obtaining efficient designs of ripple carry as well as prefix adders. Given three binary inputs, a, b and c , the majority voting logic function is given by $M(a, b, c) = ab + bc + ac$ [3, 4].

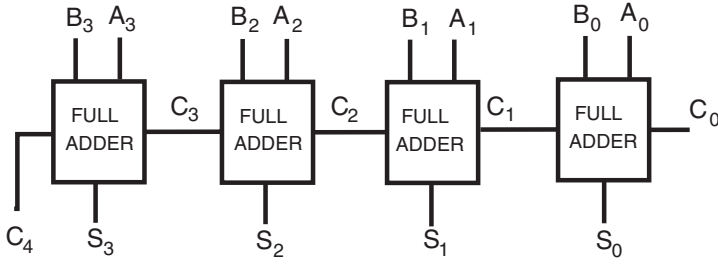


Fig. 4.1 A 4-bit ripple carry adder

Lemma 4.1 *If a , b and c are three binary inputs, then $M(a, b, M(a, b, c)) = M(a, b, c)$*

$$\begin{aligned}
 \text{Proof: } M(a, b, M(a, b, c)) &= ab + b(ab + bc + ac) \\
 &\quad + a(ab + bc + ac) \\
 &= ab + bc + ac + abc \\
 &= M(a, b, c) \quad \text{Q.E.D}
 \end{aligned}$$

As a consequence of Lemma 4.1, we have $M(a, M(a, b, c), c) = M(a, b, c)$ and $M(M(a, b, c), b, c) = M(a, b, c)$.

Lemma 4.2 *Let a, b and c be three binary inputs. Then $M(a, b, \overline{M(a, b, c)}) = M(a, b, \bar{c})$*

$$\begin{aligned}
 \text{Proof: } M(a, b, \overline{M(a, b, c)}) &= ab + a(\bar{a}\bar{b} + \bar{b}\bar{c} + \bar{a}\bar{c}) \\
 &\quad + b(\bar{a}\bar{b} + \bar{b}\bar{c} + \bar{a}\bar{c}) \\
 &= ab + a\bar{b}\bar{c} + \bar{a}b\bar{c} \\
 &= a(b + \bar{b}\bar{c}) + b(a + \bar{a}\bar{c}) \\
 &= a(b + \bar{c}) + b(a + \bar{c}) \\
 &= M(a, b, \bar{c}) \quad \text{Q.E.D}
 \end{aligned}$$

As a consequence of Lemma 4.2, we have $M(a, \overline{M(a, b, c)}, c) = M(a, \bar{b}, c)$ and $M(\overline{M(a, b, c)}, b, c) = M(\bar{a}, b, c)$

Lemma 4.3 *Let f_1 , f_2 and f_3 be three Boolean functions such that f_1 and f_2 satisfy $f_1 f_2 = f_1$ and $f_1 + f_2 = f_2$. Then $M(f_1, f_2, f_3) = f_1 + f_2 f_3$*

$$\begin{aligned}
 \text{Proof: } M(f_1, f_2, f_3) &= f_1 f_2 + f_1 f_3 + f_2 f_3 \\
 &= f_1 + (f_1 + f_2) f_3 \\
 &= f_1 + f_2 f_3 \quad \text{Q.E.D}
 \end{aligned}$$

4.1.2 Application of the Majority Logic Results to a Ripple Carry Adder

An interesting consequence of Lemma 4.2 is Lemma 4.4 described next. This lemma establishes that carry generation requires one majority gate and sum generation requires just two majority gates plus one inverter in a one-bit full adder. Let a_i , b_i and c_i be inputs to a full adder and let s_i and c_{i+1} be its outputs.

Lemma 4.4 *A 1-bit full adder can be realized using 3 majority gates and 1 inverter.*

Proof From the results in [4], we have

$$c_{i+1} = M(a_i, b_i, c_i) \quad (4.1)$$

The sum s_i can be expressed in three equivalent formats as given by Eqs. (4.2)–(4.4).

$$s_i = M \left(\overline{M(a_i, b_i, c_i)}, M(a_i, b_i, \overline{c_i}), c_i \right) \quad (4.2)$$

If b_i in Eq. (4.2) is complemented instead of c_i , we have

$$s_i = M \left(\overline{M(a_i, b_i, c_i)}, M(a_i, \overline{b_i}, c_i), b_i \right) \quad (4.3)$$

Similarly,

$$s_i = M \left(\overline{M(a_i, b_i, c_i)}, M(\overline{a_i}, b_i, c_i), a_i \right) \quad (4.4)$$

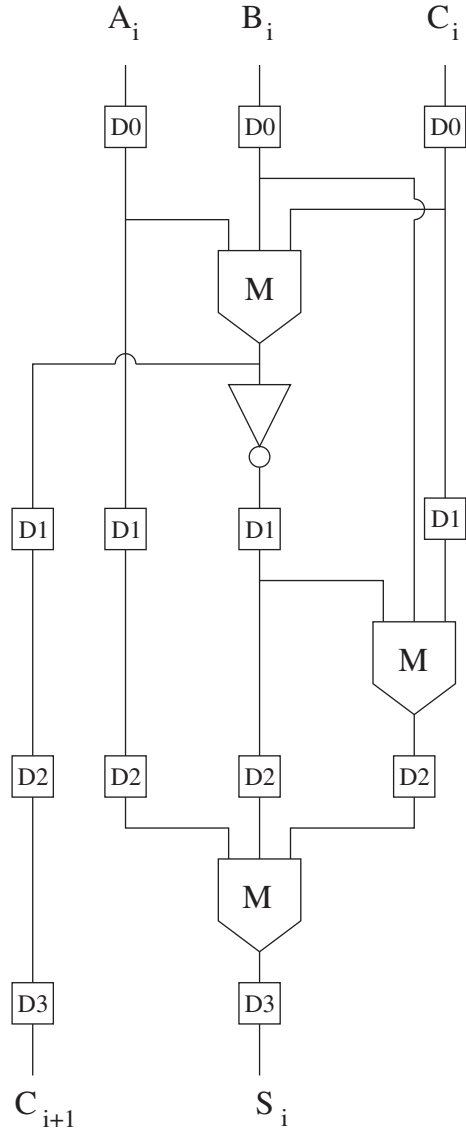
Using Lemma 4.2 in Eq. (4.4), we have

$$s_i = M \left(\overline{M(a_i, b_i, c_i)}, M \left(\overline{M(a_i, b_i, c_i)}, b_i, c_i \right), a_i \right) \quad (4.5)$$

Q.E.D

A one-bit full adder that incorporates appropriate clocking is shown in Fig. 4.2. The D-latch convention presented in [5] enables us to obtain the total circuit delay. One D-latch (namely, D0) is used to indicate that one-quarter of a clock is required to apply the inputs to the majority logic. One-fourth clock zone delay is assumed when a majority gate is immediately followed by an inverter or vice-versa (D1 is introduced at the output of inverter that follows the majority gate [5]). Proceeding this way, we have a total circuit delay of 1 clock (4 clock zones) for generating S_i as well as C_{i+1} for a 1-bit adder.

Fig. 4.2 Full adder realization using three majority gates and one inverter; numbered D-latches enable delay determination



The result presented in Lemma 4.4 improves upon a result in [4] that requires 2 inverters. We can use the result on a 1-bit full adder to derive the following for an n -bit RCA.

Corollary 4.5 *An n -bit RCA requires at most $3n$ majority gates and n inverters.*

From Corollary 4.5, we see that a 4-bit RCA requires 12 majority gates and 4 inverters. Note that the path from input to the “last” output contains seven clock

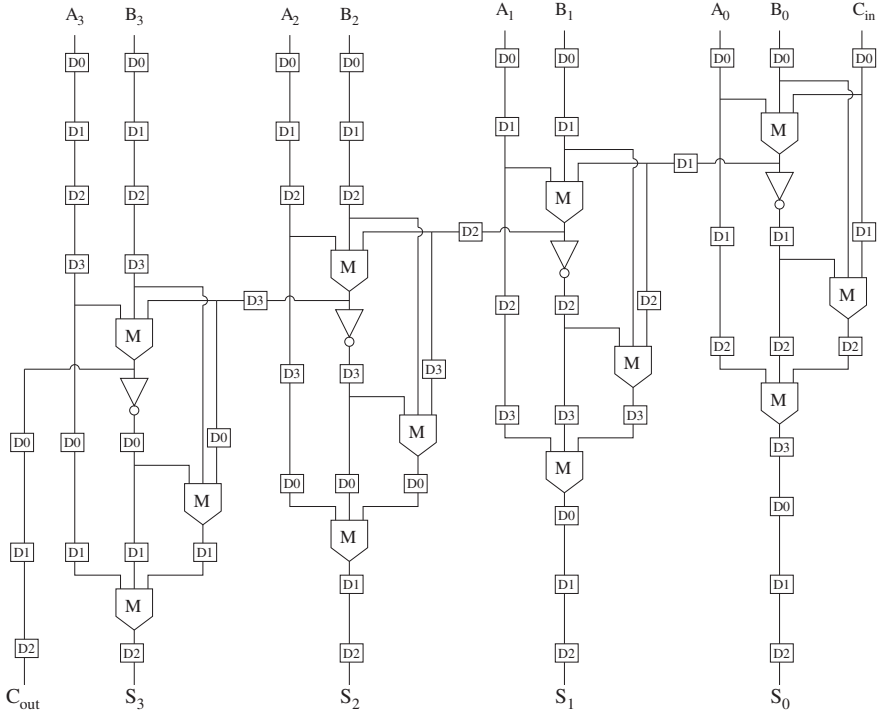


Fig. 4.3 4-bit RCA critical path composed of 7 D-latches (including 1 for input and 1 for each majority gate)

zones as shown in Fig. 4.3. So we have a total delay of $1\frac{3}{4}$ clocks. While the RCA is simple, the delay increases (linearly) as the size of the adder increases. This suggests study of other types of adders.

4.2 Design of Prefix Adders in QCA

Prefix adders constitute an interesting class of parallel adders [1]. They are based on reducing carry computation to a “prefix” computation. Various prefix adders have been reported in the literature [6–8].

We begin by presenting a general formulation of prefix adders in terms of the associative operator $\circ$ defined as follows [8] (Note that $\circ$ is also the fundamental carry operation [1]):

$$(g_i, p_i) \circ (g_j, p_j) = (g_i + (p_i g_j), p_i p_j) \quad (4.6)$$

where g_i , p_i , g_j and p_j are Boolean variables.

In particular, Eqs. (4.7)–(4.10) apply to all forms of prefix adders. Let $g_i = a_i b_i$ (generated carry) and $p_i = a_i + b_i$ (propagated carry).

$$(c_1, 0) = (g_0, p_0) \circ (c_0, 0) \quad (4.7)$$

$$(c_2, 0) = (g_1, p_1) \circ (g_0, p_0) \circ (c_0, 0) \quad (4.8)$$

$$(c_3, 0) = (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ (c_0, 0) \quad (4.9)$$

$$(c_4, 0) = (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ (c_0, 0) \quad (4.10)$$

To develop the equations for the prefix adders, we define the following notation:

$$(g_{i:j}, p_{i:j}) = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \dots \circ (g_{j-1}, p_{j-1}) \circ (g_j, p_j) \quad (4.11)$$

Suppose m is an integer defined as $j < m < i$, then we can rewrite Eq. (4.11) as shown in Eq. (4.12).

$$(g_{i:j}, p_{i:j}) = (g_{i:m}, p_{i:m}) \circ (g_{m-1:j}, p_{m-1:j}) \quad (4.12)$$

Using Eq. (4.12), we can rewrite Eq. (4.10) as

$$\begin{aligned} c_4 &= (g_{3:2}, p_{3:2}) \circ (g_{1:0}, p_{1:0}) \circ (c_0, 0) \\ &= (g_{3:0}, p_{3:0}) \circ (c_0, 0) \end{aligned}$$

In general, c_{i+1} can be expressed as

$$c_{i+1} = (g_{i:0}, p_{i:0}) \circ (c_0, 0)$$

If initial carry $c_0 = 0$, then $c_{i+1} = g_{i:0}$.

Depending upon how we calculate the carries in an n -bit adder, many prefix adders are available. In this section we focus on three types of prefix adders namely Kogge-Stone, Ladner-Fischer and Brent-Kung prefix adders. The prefix graphs of 16-bit Kogge-Stone, Ladner-Fischer and Brent-Kung adder are shown in Figs. 4.4, 4.5 and 4.6 respectively. Each filled circle in Figs. 4.4, 4.5 and 4.6 represents an associative operation and its majority gate diagram is shown in Fig. 4.7a. Each empty circle also represents an associative operation but it only computes the generate term since at the final stage of carry computation, there is no need to calculate the propagate term. Its majority gate diagram is shown in Fig. 4.7b.

The final stage of all prefix adders is sum calculation. Before we derive bounds on majority logic for all the prefix adders, we present a lemma (Lemma 4.6) on majority logic that will be used to obtain an efficient sum design for all prefix adders. Let $g_i = a_i b_i$ and $p_i = a_i + b_i$. Lemma 4.6 shows that the calculation of sum requires just two majority gates and one inverter using g_i , p_i , c_i and c_{i+1} .

Lemma 4.6 *At most 2 majority gates and 1 inverter are required for obtaining the sum at each stage (when g_i and p_i are incoming) for a prefix adder.*

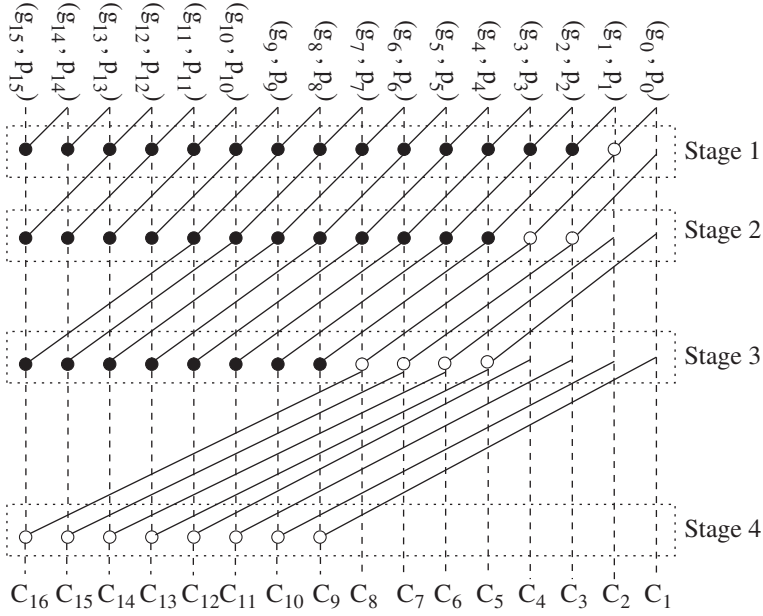


Fig. 4.4 16-bit Kogge-Stone adder prefix graph

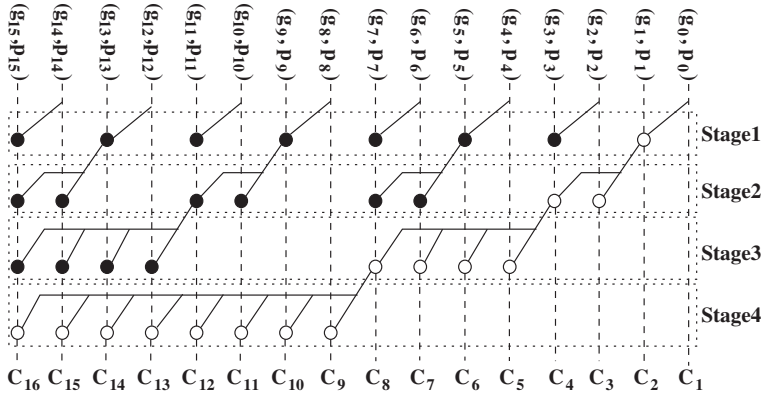


Fig. 4.5 16-bit Ladner-Fischer adder prefix graph

Proof From Eqs. (4.1) and (4.2), we have $s_i = M(\overline{M(a_i, b_i, c_i)}, M(a_i, b_i, \overline{c_i}), c_i)$. However,

$$\begin{aligned}
 M(a_i, b_i, c_i) &= a_i b_i + (a_i + b_i) c_i \\
 &= g_i + p_i c_i \\
 &= M(g_i, p_i, c_i) \text{ (from Lemma 4.3)}
 \end{aligned}$$

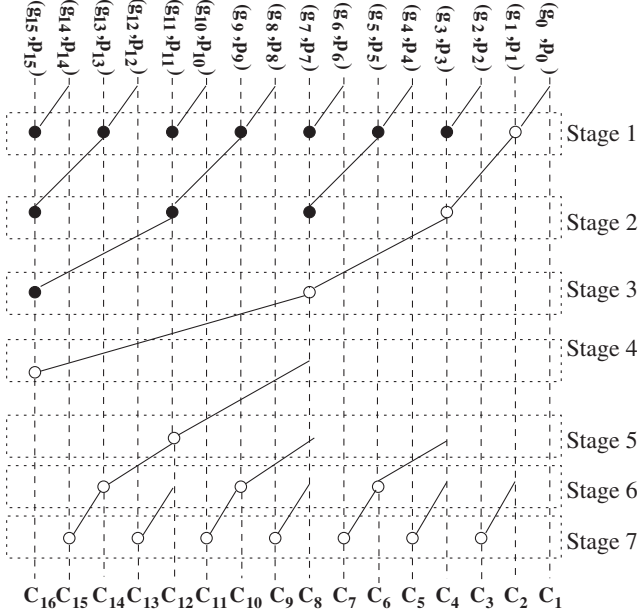


Fig. 4.6 16-bit Brent-Kung adder prefix graph

So $c_{i+1} = M(a_i, b_i, c_i) = M(g_i, p_i, c_i)$

Similarly,

$$M(a_i, b_i, \bar{c}_i) = M(g_i, p_i, \bar{c}_i)$$

From Lemma 4.2

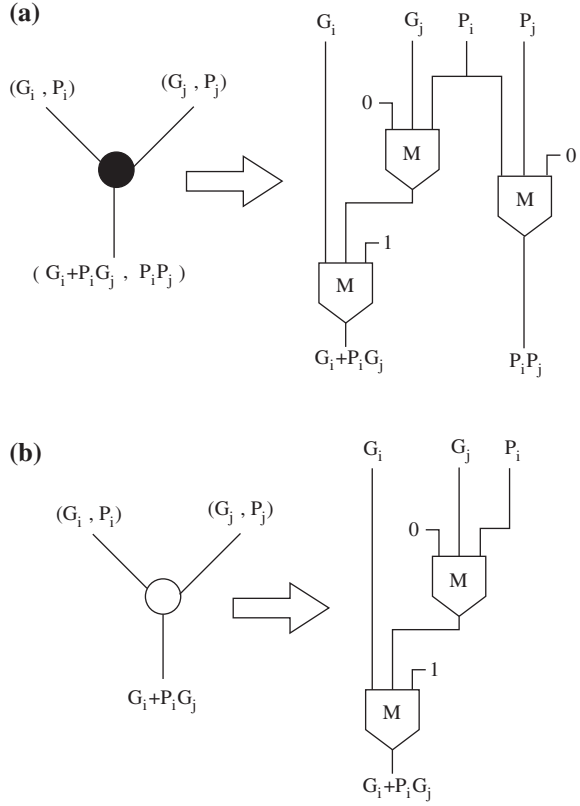
$$M(g_i, p_i, \bar{c}_i) = M\left(g_i, p_i, \overline{M(g_i, p_i, c_i)}\right) = M(g_i, p_i, \overline{c_{i+1}})$$

Therefore,

$$s_i = M(\overline{c_{i+1}}, M(g_i, p_i, \overline{c_{i+1}}), c_i) \quad Q.E.D \quad (4.13)$$

In this section, we derive bounds on majority logic for all the prefix adders. It turns out that the bounds on majority logic are directly related to the number of associative operations for each prefix adder. For each prefix adder, the total number of associative operations can be obtained by examining what is known as the prefix graph of the adder. The prefix graph depicts the calculations that take place stage by stage. We begin with the analysis for the Kogge-Stone adder.

Fig. 4.7 Majority gate diagram of associative operator



4.2.1 The Kogge-Stone Adder

The Kogge-Stone Prefix Adder [6] is the earliest of the prefix adders reported in the literature. Its prefix graph is shown in Fig. 4.4. Proposition 4.7 gives the number associative operations required for an n -bit Kogge-Stone Adder.

Proposition 4.7 *The number of associative operations required for an n -bit Kogge-Stone adder is given by*

$$A(n) = n(\log_2 n - 1) + 1$$

Proof An n -bit Kogge-Stone adder requires $\log_2 n$ levels to compute all carries. The number of associative operations required for each level in 4-bit, 8-bit and 16-bit Kogge-Stone adder are given by (3, 2), (7, 6, 4) and (15, 14, 12, 8) respectively. From these numbers in each level we can observe a regular pattern, that is $n - 1, n - 2, n - 2^2, n - 2^3 \dots, n - 2^{\log_2 n - 1}$. So we can find the total number of associative operations as the sum of these quantities.

$$\begin{aligned}
A(n) &= n - 1 + n - 2 + n - 2^2 + n - 2^3 + \dots + n - 2^{\log_2 n - 1} \\
&= \log_2 n \times n - \left[1 + 2 + 2^2 + \dots + 2^{\log_2 n - 1} \right] \\
&= \log_2 n \times n - \left[2^{\log_2 n} - 1 \right] \\
&= \log_2 n \times n - n + 1 \\
&= n(\log_2 n - 1) + 1
\end{aligned}$$

With regard to implementation of the Kogge-Stone adder in QCA, if we use the associative operator shown in Fig. 4.7, it increases the complexity in design because it requires two majority gates to calculate the generate term (total of three majority gates for calculating generate and propagate). We now present a lemma which shows only one majority gate is required for calculating the generate term in the associative operation.

Lemma 4.8 *If $g_i = a_i b_i$, $p_i = a_i + b_i$ and f_1 are three binary inputs then*

$$g_i + p_i f_1 = M(g_i, p_i, f_1)$$

Proof $g_i + p_i = a_i b_i + a_i + b_i = a_i + b_i = p_i$

$g_i p_i = (a_i b_i)(a_i + b_i) = g_i$

Since g_i and p_i satisfy two conditions of Lemma 4.3, we can write $g_i + p_i f_1$ as

$$g_i + p_i f_1 = M(g_i, p_i, f_1) \quad \text{Q.E.D}$$

Using Lemma 4.8, we have

$$(c_1, 0) = (M(g_0, p_0, c_0), 0)$$

Similarly, $(g_{i+1}, p_{i+1}) \circ (g_i, p_i)$ is calculated as

$$(g_{i+1}, p_{i+1}) \circ (g_i, p_i) = (M(g_{i+1}, p_{i+1}, g_i), M(p_{i+1}, p_i, 0))$$

Using Lemma 4.8, we can perform the associative operation with only two majority gates. However, Lemma 4.8 does not apply to all associative operations since all of them will not satisfy the conditions of Lemma 4.3.

Figure 4.8 shows the 16-bit Kogge-Stone prefix graph with application of Lemma 4.8, where circle (filled or empty) surrounded by a rectangle represents the associative operation with application of Lemma 4.8 and its majority gate diagram is shown in Fig. 4.9.

We now turn our attention to the calculation of the number of majority gates for a Kogge-Stone adder. To see the benefits of our new results on majority logic, we will first obtain the numbers for a direct solution without logic optimization. The total number of majority gates of any prefix adder is sum of majority gates required for g_i s and p_i s, majority gates required for calculating all carries and majority gates

Fig. 4.8 16-bit Kogge-Stone adder prefix graph with applying Lemma 4.8

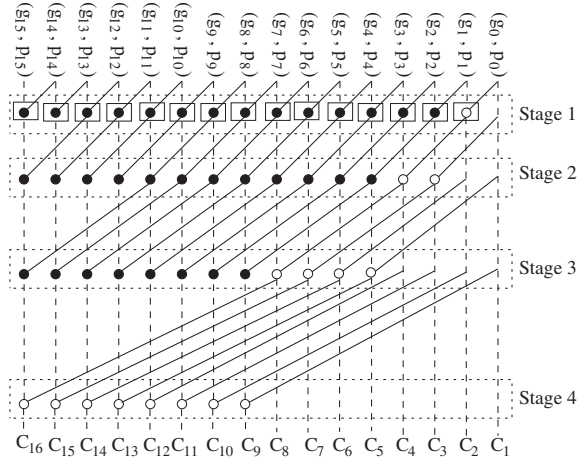
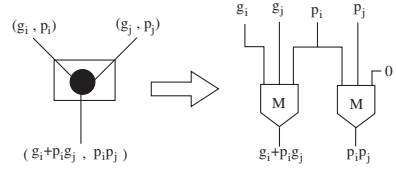


Fig. 4.9 Majority Gate diagram of associative operator using Lemma 4.8



required for all sum outputs. In the case of a 16-bit Kogge-Stone adder, 32 majority gates are required for calculating g_i s and p_i s.

Generalization to n -bit Kogge-Stone Adder

We present a bound on the number of majority gates required, denoted by $I(n)$, for QCA realization of an n -bit Kogge-Stone adder (we focus on majority gates since we know that n inverters are required for an n -bit adder). $I(n)$ corresponds to the sum of $I_c(n)$ (for generating n -carries), $I_s(n)$ (for generating n -sums) and $I_{gp}(n)$ (for obtaining the g_i s and p_i s). To obtain $I_c(n)$, we present a recursive formulation. In particular, we show that $I_c(n) = I_c(n : \frac{n}{2} + 1) + I_c(\frac{n}{2})$ where $I_c(n : \frac{n}{2} + 1)$ is the number of majority gates required for generating the carries from n to $\frac{n}{2}$ while $I_c(\frac{n}{2})$ is the number of majority gates required for generating carries of an $\frac{n}{2}$ -bit adder.

To derive the general formula for $I_c(n)$, we make some observations on the prefix graph shown in Fig. 4.4. In this case, $n = 16$. Let the number of majority gates required for a direct method (without applying Lemma 4.8) be denoted by $I_c^d(16)$. $I_c^d(16)$ is given by the steps (i)–(iv) and in step (v) we calculate $I_c(16)$ for Kogge-Stone Adder shown in Fig. 4.8.

- (i) Considering the left half corresponding to generation of C_{16} to C_9 in Fig. 4.8, we note that Stages 1, 2 and 3 involve eight associative operations each. Each associative operation requires three majority gates. Hence, a total of 72 majority gates is required for these three stages.

- (ii) Stage 4 involves eight associative operations (that is, $\frac{n}{2}$ where n is 16). Only generation of carry takes place in this stage so there is no need for majority gate logic for propagate here. As a consequence, each associative operation in Stage 4 requires 2 majority gates. The total majority gate requirement for this stage is therefore 16. Hence, $I_c^d(16 : 9) = 72 + 16 = 88$.
- (iii) For generation of the remaining carries, namely C_1 to C_8 , we can once again consider the left half corresponding to $C_8 - C_5$. In this case, Stages 1 and 2 involve 4 associative operations each. Each associative operation requires 3 majority gates hence a total of 24 majority gates are required. Stage 3 here involves 4 associative operations and each associative operation requires 2 majority gates (as before, there is no need for majority logic for propagate here). Hence, stage 3 requires 8 majority gates and $I_c^d(8 : 5) = 24 + 8 = 32$.
- (iv) Proceeding as before, we can compute the majority gates required for carries C_4 to C_1 . We have $I_c^d(4 : 3) = 10$ and $I_c^d(2) = 2$ (2 for C_2 and assuming $C_1 = g_1$). Summing up the majority gates in Steps (i)–(iv), we have $I_c^d(16) = I_c^d(16 : 9) + I_c^d(8 : 5) + I_c^d(4 : 3) + I_c^d(2) = 88 + 32 + 10 + 2 = 132$.
- (v) From Fig. 4.8 we can note that Lemma 4.8 is applied to 15 associative operations so $I_c(16) = I_c^d(16) - 15 = 132 - 15 = 117$.

We can generalize this result as follows. The number of majority gates required for an n -bit Kogge-Stone adder is given by Theorem 4.9.

Theorem 4.9 *The number of majority gates required for an n -bit Kogge-Stone adder is given by*

$$I(n) = n(3 \log_2 n - 1) + 5$$

Proof The computation of carries of an n -bit Kogge-Stone adder requires $\log_2 n$ stages. The general formula for calculating the number of majority gates for carries from $\frac{n}{2} + 1$ to n is obtained as follows. Each stage in the Kogge-Stone adder requires $\frac{n}{2}$ associative operations (namely carries from $\frac{n}{2} + 1$ to n). Except the last stage, each associative operation in all other stages requires 3 majority gates. In the last stage, each associative operation requires only 2 majority gates (since there is no calculation of the propagate term). Therefore,

$$\begin{aligned} I_c^d\left(n : \frac{n}{2} + 1\right) &= 3\left(\frac{n}{2}\right)(\log_2 n - 1) + 2\left(\frac{n}{2}\right) \\ &= \frac{n}{2} [3 \log_2 n - 1] \end{aligned}$$

$$\begin{aligned} I_c^d(n) &= \frac{n}{2} [3 \log_2 n - 1] + I_c^d\left(\frac{n}{2}\right) \\ &= 3\left[\frac{n}{2} \log_2 n + \frac{n}{4} \log_2 \frac{n}{2} + \cdots + 2 \cdot 2\right] \\ &\quad - \left[\frac{n}{2} + \frac{n}{4} + \cdots + 2\right] + I_c^d(2) \\ &= 3[2n \log_2 n - (\log_2 n + 1)n] - [n - 2] + 2 \\ &= 3n \log_2 n - 4n + 4 \quad [\text{since } I_c^d(2) = 2] \end{aligned}$$

Lemma 4.8 can be applied to stage 1 of each associative operation shown in Fig. 4.8. We will have a reduction of one majority gate each and a total of $\frac{n}{2}$ majority gates.

$$\begin{aligned} I_r(n) &= \frac{n}{2} + I_r\left(\frac{n}{2}\right) \\ &= n - 1 \end{aligned}$$

Hence, $I_c(n)$ is given as

$$\begin{aligned} I_c(n) &= I_c^d(n) - I_r(n) \\ &= 3n \log_2 n - 5n + 5 \end{aligned}$$

For an n -bit adder, g_i and p_i require one majority gate each, hence $I_{gp}(n) = 2n$ majority gates. Also, each s_i requires two majority gates, so $I_s(n) = 2n$ majority gates. So overall majority gate requirement is given by

$$\begin{aligned} I(n) &= I_c(n) + I_{gp}(n) + I_s(n) \\ &= n(3 \log_2 n - 1) + 5 \quad \text{Q.E.D} \end{aligned}$$

4.2.2 The Ladner-Fischer Adder

Another prefix adder reported in the literature is the Ladner-Fischer adder [7]. This also has $O(\log_2 n)$ stages. Its prefix graph is shown in Fig. 4.5. From Fig. 4.5, we can notice that the 16-bit Ladner-Fischer adder contains four stages ($\log_2 16$) and all stages contain equal number of associative operations i.e. 8 associative operations ($16/2$). Proposition 4.10 gives the number associative operations required for an n -bit Ladner-Fischer Adder.

Proposition 4.10 *The number of associative operations required for an n -bit Ladner-Fischer Adder is given by*

$$A(n) = \frac{n}{2} \log_2 n$$

Proof An n -bit Ladner-Fischer adder contains $\log_2 n$ stages and each stage contains $\frac{n}{2}$ associative operations, so total number of associative operations is given by

$$\begin{aligned} A(n) &= \text{number of stages} \times \text{Associative operations in each stage} \\ &= \log_2 n \times \frac{n}{2} \\ &= \frac{n}{2} \log_2 n \end{aligned}$$

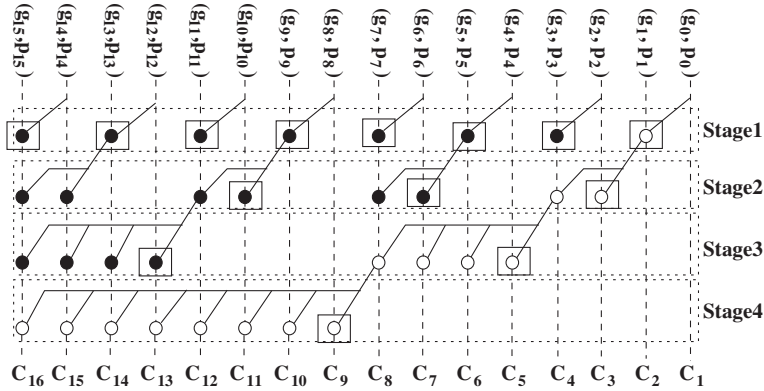


Fig. 4.10 Ladner-Fischer 16-bit adder prefix graph

Figure 4.10 shows a 16-bit Ladner-Fischer prefix graph with application of Lemma 4.8. The circle (filled or empty) surrounded by rectangle represents the associative operation with application of Lemma 4.8 and its majority gate diagram is shown in Fig. 4.9.

Generalization to an n -bit Ladner-Fischer Adder

In this section, we present a bound on the number of majority gates required, denoted by $I(n)$, for QCA realization of an n -bit Ladner-Fischer adder.

To derive the general formula for $I_c(n)$, we make some observations on the prefix graph shown in Fig. 4.5. In this case, $n = 16$. The following steps give the number of majority gates.

- (i) Considering the left half corresponding to generation of C_{16} to C_9 in Fig. 4.5, we note that Stages 1, 2 and 3 involve four associative operations each. Each associative operation requires 3 majority gates. Hence, a total of 36 majority gates is required for these three stages.
- (ii) Stage 4 involves eight associative operations (that is, $\frac{n}{2}$ where n is 16). Only generation of carry takes place in this stage so there is no need for majority gate logic for propagate here. As a consequence, each associative operation in Stage 4 requires 2 majority gates. The total majority gate requirement for this stage is therefore 16. Hence, $I_c^d(16 : 9) = 36 + 16 = 52$.
- (iii) For generation of the remaining carries, namely C_1 to C_8 , we can once again consider the left half corresponding to $C_8 - C_5$. In this case, Stages 1 and 2 involve 2 associative operations each. Each associative operation requires 3 majority gates hence a total of 12 majority gates are required. Stage 3 here involves 4 associative operations and each associative operation requires 2 majority gates (as before, there is no need for majority logic for propagate here). Hence stage 3 requires 8 majority gates and $I_c^d(8 : 5) = 12 + 8 = 20$.

- (iv) Proceeding as before, we can compute the majority gates required for carries C_4 to C_1 . We have $I_c^d(4 : 3) = 7$ and $I_c^d(2) = 2$. Summing up the majority gates in Steps (i)–(iv), we have $I_c^d(16) = I_c^d(16 : 9) + I_c^d(8 : 5) + I_c^d(4 : 3) + I_c^d(2) = 81$.
- (v) From Fig. 4.10 we can note that Lemma 4.8 is applied to 15 associative operations so $I_c(16) = I_c^d(16) - 15 = 81 - 15 = 66$.

Therefore, $I(16) = 66 + 32 + 32 = 130$. We now present a bound on the number of majority gates required for the n -bit case via Theorem 4.11.

Theorem 4.11 *The number of majority gates required for an n -bit Ladner-Fischer adder is given by*

$$I(n) = \frac{n}{2} (3 \log_2 n + 4) + 2$$

Proof Computation of carries of an n -bit Ladner-Fischer adder requires $\log_2(n)$ stages. The count for majority gates for an n -bit adder can be obtained using a recursive formulation noting that the lower order carries (namely, carries from 1 to $\frac{n}{2}$) are those of an $\frac{n}{2}$ -bit adder (this can be extended all the way up to a 2-bit adder).

We therefore obtain a general formula for calculating the majority gates for carries from n to $\frac{n}{2} + 1$. The stages from 1 to $(\log_2 n) - 1$ contain $\frac{n}{4}$ associative operations in each stage. Each associative operation requires 3 majority gates (in a direct method). So total number of majority gates is $3(\log_2 n - 1)\frac{n}{4}$. At the lowermost stage (i.e. $\log_2(n)$), there are $\frac{n}{2}$ associative operations and each requires 2 majority gates. Altogether, we have $2(\frac{n}{2}) = n$ -majority gates. Hence, for a direct solution, we have

$$I_c^d(n) = 3 (\log_2 n - 1) \frac{n}{4} + n + I_c^d\left(\frac{n}{2}\right)$$

$$\begin{aligned}
 I_c^d(n) &= 3 (\log_2 n - 1) \frac{n}{4} + n + I_c^d\left(\frac{n}{2}\right) \\
 &= \frac{n}{4} [3 \log_2 n + 1] + I_c^d\left(\frac{n}{2}\right) \\
 &= \frac{3}{2} \left[\frac{n}{2} \log_2 n + \frac{n}{4} \log_2 \frac{n}{2} + \cdots + 2 \cdot 2 \right] \\
 &\quad - \left[\frac{n}{4} + \frac{n}{8} + \cdots + 2 + 1 \right] + I_c^d(2) \\
 &= \frac{3}{2} [2n \log_2 n - (\log_2 n + 1)n] - [n - 2] + 2 \\
 &= \frac{3}{2} n \log_2 n - n + 1 \quad [\text{since } I_c^d(2) = 2]
 \end{aligned}$$

To obtain a reduction in majority gates in each stage, we can apply Lemma 4.3. For stage 1, stage 2, ..., stage $\log_2(n) - 1$, we will have a reduction of $\frac{n}{4}, \frac{n}{8}, \dots, \frac{n}{2^{\log_2(n)}}$ respectively. At $\log_2(n)$ th stage, there is a reduction of only one majority gate. So total reduction t is

$$\begin{aligned}
t &= \frac{n}{4} + \frac{n}{8} + \cdots + \frac{n}{2^k} + 1 \quad [\text{where } k = \log_2(n)] \\
&= 2^{k-2} + 2^{k-3} + \cdots + 2^0 + 1 \quad [\text{where } n = 2^k] \\
&= 2^{k-1} \\
&= \frac{2^{\log_2(n)}}{2} \\
&= \frac{n}{2}
\end{aligned}$$

This leads to a reduction of majority gates, denoted $I_r(n)$, given by

$$I_r(n) = \frac{n}{2} + I_r\left(\frac{n}{2}\right) = n - 1$$

$$\begin{aligned}
I_c(n) &= I_c^d(n) - I_r(n) \\
&= \frac{3}{2}n \log_2 n - 2n + 2
\end{aligned}$$

For an n -bit adder, g_i and p_i require one majority gate each, hence $I_{gp}(n) = 2n$ majority gates. Also, each s_i requires two majority gates, so $I_s(n) = 2n$ majority gates. So the overall majority gate requirement is

$$\begin{aligned}
I(n) &= I_c(n) + I_{gp}(n) + I_s(n) \\
&= \frac{3}{2}n \log_2 n - 2n + 2 + 2n + 2n \\
&= \frac{n}{2} (3 \log_2 n + 4) + 2 \quad \text{Q.E.D}
\end{aligned}$$

4.2.3 The Brent-Kung Adder

Another prefix adder reported in the literature is the Brent-Kung adder [8]. This has $2 \log_2 n - 1$ stages. Its prefix graph is shown in Fig. 4.6. Proposition 4.12 gives the number of associative operations required for an n -bit Brent-Kung Adder.

Proposition 4.12 *The number of associative operations required for an n -bit Brent-Kung adder is given by*

$$A(n) = 2n - \log_2 n - 2$$

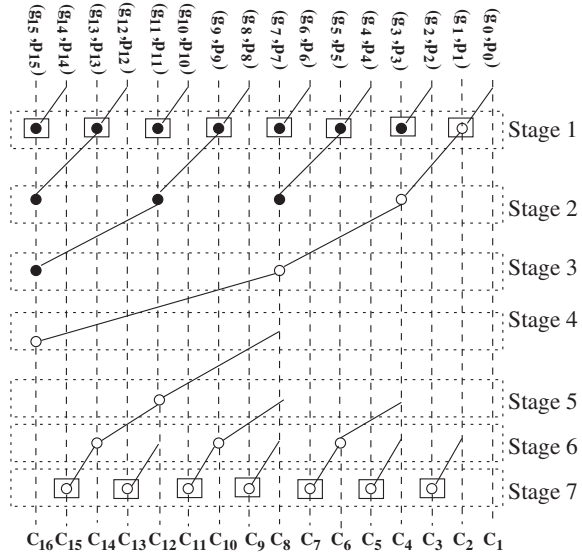
Proof An n -bit Brent-Kung Adder requires $2 \log_2 n - 1$ levels to compute all the carries. So a 4-bit Brent-Kung adder contains three levels and they require 2, 1 and 1 associative operations respectively for a total of four associative operations. An 8-bit Brent-Kung adder contains five levels and they require 4, 2, 1, 1 and 3 operations respectively. Similarly, a 16-bit adder requires seven levels and they require 8, 4, 2, 1, 1, 3 and 7 operations respectively.

From the above calculations, if we observe the number of associative operations in each level, they are decreasing from level 1 to level $\log_2 n$ and then increasing from level $(\log_2 n + 1)$ to level $(2\log_2 n - 1)$. The number of associative operations in the Brent-Kung adder follows a regular pattern for each level, that is $\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \dots, \frac{n}{2}, \frac{n}{n} \frac{n}{2} - 1, \frac{n}{4} - 1 \dots \frac{n}{4} - 1, \frac{n}{2} - 1$. Therefore, the number of associative operations $A(n)$ is given by

$$\begin{aligned}
 A(n) &= \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots + \frac{n}{2} + \frac{n}{n} \\
 &\quad \frac{n}{2} - 1 + \frac{n}{4} - 1 + \dots + \frac{n}{4} - 1 + \frac{n}{2} - 1 \\
 &= 2 \left[\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots + \frac{n}{2} \right] + 1 - (\log_2 n - 1) \\
 &= 2 \frac{n}{2} \frac{1 - 2n^{-1}}{2^{-1}} - \log_2 n + 2 \\
 &= 2(n - 2) - \log_2 n + 2 \\
 &= 2n - \log_2 n - 2
 \end{aligned}$$

Figure 4.11 shows the 16-bit Brent-Kung prefix graph with application of Lemma 4.8 and circle (filled or empty) surrounded by rectangle represents the associative operation with application of Lemma 4.8 and its majority gate diagram is shown in Fig. 4.9.

Fig. 4.11 Brent-Kung 16-bit adder prefix graph with applications of Lemma 4.8



Generalization to an n -bit Brent-Kung Adder

In this section, we present a bound on the number of majority gates required, denoted by $I(n)$, for QCA realization of an n -bit Brent-Kung adder.

To derive the general formula for $I_c(n)$, we make some observations on the prefix graph shown in Fig. 4.6. In this case, $n = 16$. Let the number of majority gates required for a direct method (that does not use Lemma 4.3) be denoted by $I_c^d(16)$. Calculation of $I_c^d(16)$ involves the following steps.

- (i) Consider the left half corresponding to generation of C_9 to C_{16} in Fig. 4.6. Stages 1, 2 and 3 involve 4, 2 and 1 associative operations respectively. Each associative operation requires three majority gates. Hence total number of majority gates is 21 (7×3).
- (ii) Stage 4 involves *one* associative operation while Stages 5, 6 and 7 involve 1, 2 and 4 associative operations respectively. From Stage 4 to Stage 7, only generation of carry takes place so there is no need for majority logic for propagate here. Therefore, each associative operation in Stages 4 to 7 requires 2 majority gates. Hence, a total of 16 (8×2) is required. We therefore have $I_c^d(16 : 9) = 21 + 16 = 37$.
- (iii) For generating the remaining carries, namely C_1 to C_8 , we consider the left half corresponding to $C_8 - C_5$. In this case, 2 and 1 associative operations are present in Stages 1 and 2 respectively. We have 1, 1 and 2 associative operations in Stages 3, 4 and 5 respectively. $I_c^d(8 : 5) = 3 \times 3 + 4 \times 2 = 17$.
- (iv) This way, we can proceed recursively. $I_c^d(16)$ can be expressed as $I_c^d(16 : 9) + I_c^d(8)$, $I_c^d(8)$ as $I_c^d(8 : 5) + I_c^d(4)$, and $I_c^d(4)$ as $I_c^d(4 : 3) + I_c^d(2)$. $I_c^d(2)$ is 2 and $I_c^d(4 : 3)$ requires 7 majority gates. Hence, $I_c^d(16) = 37 + 17 + 7 + 2 = 63$.
- (v) $I_c(16)$ is considerably less than $I_c^d(16)$ since Lemma 4.3 can be applied at Stage 1 and Stage 7 leading to a saving of 15 majority gates. That is, $n - 1$ majority gates are saved here. Hence, $I_c(16) = 48$. We now present a bound on the number of majority gates required for the n -bit case via Theorem 4.13.

Theorem 4.13 *The number of majority gates required for an n -bit Brent-Kung adder is given by*

$$I(n) = 8n - 3 \log_2(n) - 4$$

Proof Computation of carries of an n -bit Brent-Kung adder requires $2 \log_2(n) - 1$ stages (assuming that n is a power of 2). The count for majority gates for an n -bit adder can be obtained using a recursive formulation noting that the lower order carries (namely, carries from 1 to $\frac{n}{2}$) are those of an $\frac{n}{2}$ -bit adder. We therefore obtain a general formula for calculating the majority gates for carries from $\frac{n}{2} + 1$ to n . The stages from 1 to $(\log_2 n - 1)$ contain $\frac{n}{4}, \frac{n}{8}, \frac{n}{16}, \dots, \frac{n}{2^{\log_2(n)}}$ number of associative operations respectively for carries from n to $\frac{n}{2} + 1$. Each associative operation requires 3 majority gates (in a direct method). So total number of majority gates (for the left half), denoted by t_1 , is given as

$$t_1 = 3 \left(\frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \dots + \frac{n}{2^{\log_2(n)}} \right)$$

$$\begin{aligned}
&= 3 \left(2^{\log_2(n)-2} + 2^{\log_2(n)-3} + 2^{\log_2(n)-4} + \dots + 2 + 1 \right) \\
&= 3 \left(2^{\log_2(n)-1} - 1 \right) \\
&= 3 \left(\frac{n}{2} - 1 \right)
\end{aligned}$$

Stages $\log_2(n)$ to $2 \log_2(n) - 1$ contain $1, 1, 2, \dots, \frac{n}{8}, \frac{n}{4}$ number of associative operations respectively and each requires 2 majority gates. The total number of majority gates required for this, denoted by t_2 , is given by

$$\begin{aligned}
t_2 &= 2 \left(1 + 1 + 2 + \dots + \frac{n}{8} + \frac{n}{4} \right) \\
&= 2 \left(1 + 1 + 2 + \dots + 2^{\log_2(n)-3} + 2^{\log_2(n)-2} \right) \\
&= 2 \left(1 + 2^{\log_2(n)-1} - 1 \right) \\
&= 2 \left(2^{\log_2(n)-1} \right) \\
&= 2^{\log_2(n)} \\
&= n
\end{aligned}$$

$I_c^d(n : \frac{n}{2})$ is then expressed as the sum of t_1 and t_2 . So

$$I_c^d \left(n : \frac{n}{2} + 1 \right) = 3 \left(\frac{n}{2} - 1 \right) + n = 5 \left(\frac{n}{2} \right) - 3$$

Hence, for a direct solution, we have

$$\begin{aligned}
I_c^d(n) &= 5 \left(\frac{n}{2} \right) - 3 + I_c^d \left(\frac{n}{2} \right) \\
&= 5 \left(\frac{n}{2} + \frac{n}{4} + \dots + 2 \right) - 3 \left(\log_2(n) - 1 \right) + I_c^d(2) \\
&= 5(n - 2) - 3 \log_2(n) + 3 + 2 \quad [I_c^d(2) = 2] \\
&= 5n - 3 \log_2(n) - 5
\end{aligned}$$

To obtain a reduction in majority gates in each stage, we can apply Lemma 4.3 to two stages, namely stage 1 and stage $(2 \log_2(n) - 1)$ (Lemma 4.3 cannot be applied to other stages). This will lead to a reduction of one majority gate at each associative operation from carry $\frac{n}{2} + 1$ to n (i.e. $\frac{n}{4} + \frac{n}{4} = \frac{n}{2}$). So total reduction of majority gates, denoted by $I_r(n)$, is sum of reduction of majority gates from carry $\frac{n}{2} + 1$ to n (denoted by $I_r(n : \frac{n}{2} + 1)$) and reduction of majority gates from carry $\frac{n}{2}$ to 1 (denoted by $I_r(\frac{n}{2})$).

$$\begin{aligned}
I_r(n) &= I_r \left(n : \frac{n}{2} + 1 \right) + I_r \left(\frac{n}{2} \right) \\
&= \frac{n}{2} + I_r \left(\frac{n}{2} \right) \\
&= \frac{n}{2} + \frac{n}{4} + \dots + 4 + 2 + I_r(2) \\
&= n - 2 + 1 \quad [I_r(2) = 1] \\
&= n - 1
\end{aligned}$$

Therefore,

$$\begin{aligned} I_c(n) &= I_c^d(n) - I_r(n) \\ &= 4n - 3 \log_2(n) - 4 \end{aligned}$$

For an n -bit adder, g_i and p_i require one majority gate each, hence $I_{gp}(n) = 2n$ majority gates. Also, each s_i requires two majority gates, so $I_s(n) = 2n$ majority gates. So the overall majority gate requirement is

$$\begin{aligned} I(n) &= I_c(n) + I_{gp}(n) + I_s(n) \\ &= 8n - 3 \log_2(n) - 4 \quad Q.E.D \end{aligned}$$

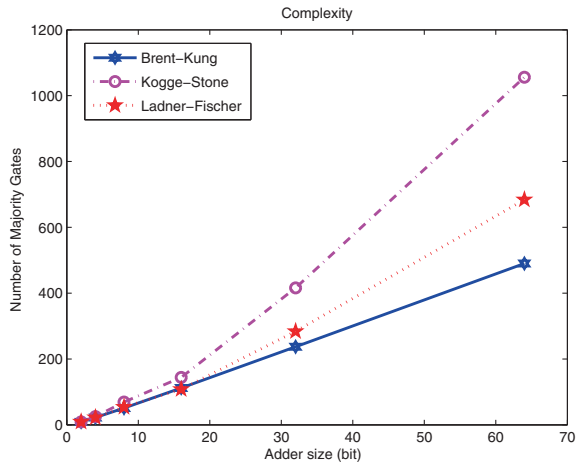
Table 4.1 summarizes the number of levels, associative operations and majority gates for each of the three prefix adders (in terms of the adder size, denoted by n). Graphs showing how majority gates accumulate for different prefix adders, as the adder size grows are shown in Fig. 4.12.

From the graph (Fig. 4.12), we observe that the Brent-Kung prefix adder supports a very efficient solution (in terms of smaller growth in number of majority gates). This is not unexpected, however, given the fact that the Brent-Kung graph has smaller number of operations (as expressed via Table 4.1) and thus gates.

Table 4.1 Number of levels, associative operations and majority gates for various n -bit prefix adders

Adder name	No. of levels	No. of assoc. oper.	No. of maj. gates
Brent-Kung	$2 \log_2 n - 1$	$2n - \log_2 n - 2$	$8n - 3 \log_2(n) - 4$
Kogge-Stone	$\log_2 n$	$n(\log_2 n - 1) + 1$	$n(3 \log_2 n - 1) + 5$
Ladner-Fischer	$\log_2 n$	$\frac{n}{2} \log_2 n$	$\frac{n}{2} (3 \log_2 n + 4) + 2$

Fig. 4.12 Plot of majority gates versus adder size for various prefix adders



4.3 QCA Implementation

In this section, we present results of simulation in QCADesigner [9]. We also present area and time complexity results for various adders.

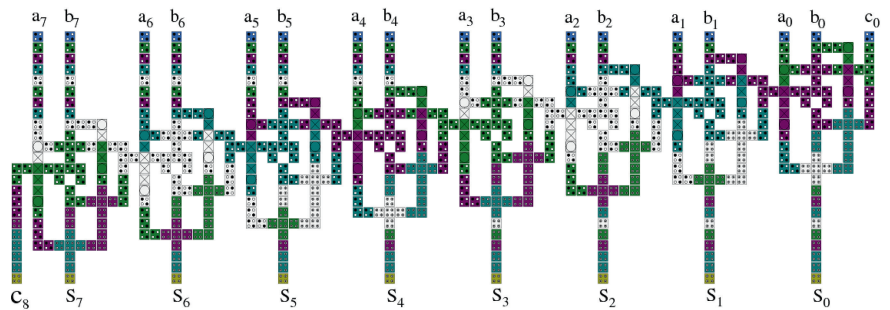


Fig. 4.13 QCADesigner layout for 8-bit ripple carry adder

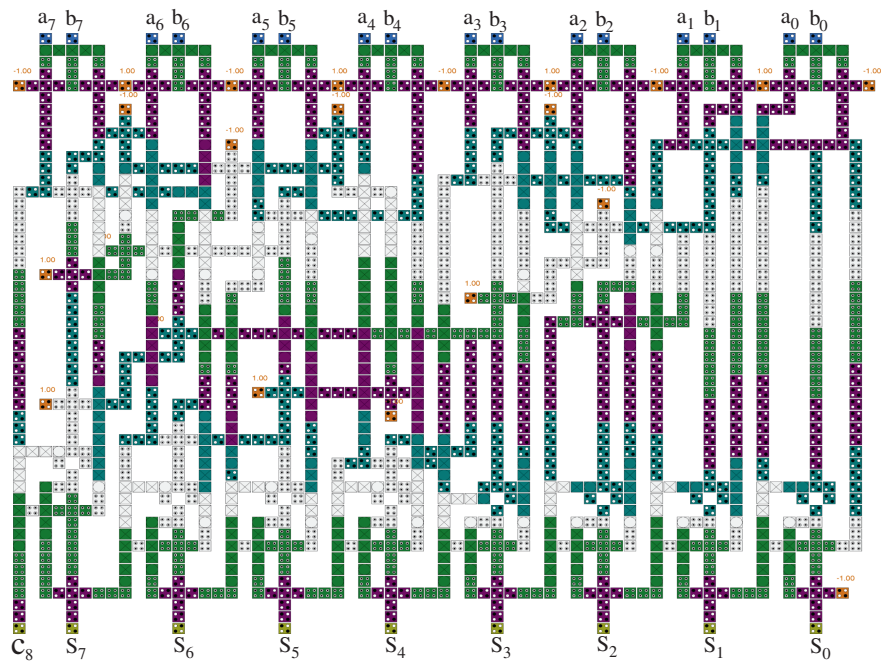


Fig. 4.14 QCADesigner layout for 8-bit Brent-Kung adder

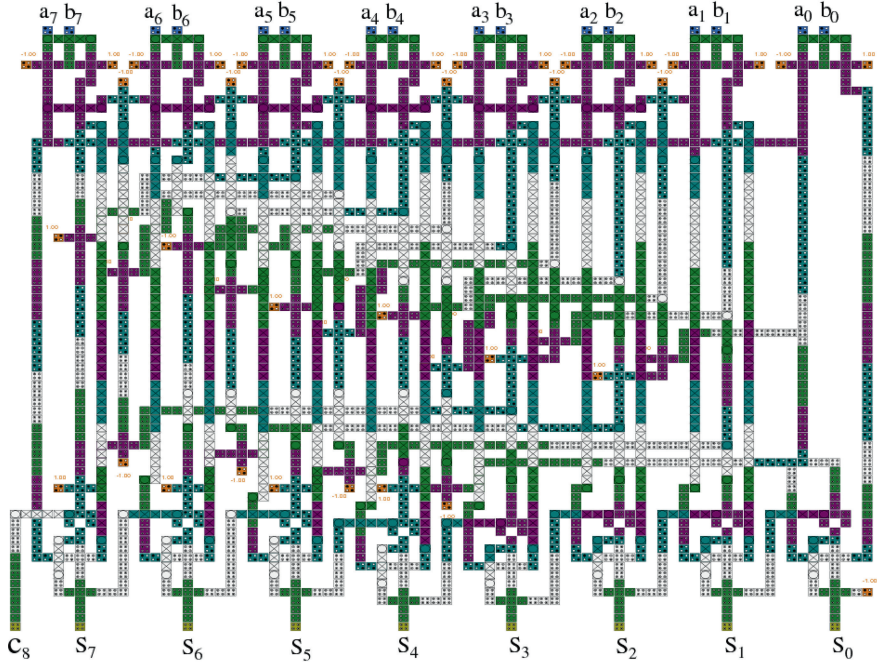


Fig. 4.15 QCADesigner layout for 8-bit Kogge-Stone adder

4.3.1 Design Rules

Cells for our design are assumed to have a height of 18 nm, and width of 18 nm while the quantum dots have a diameter of 5 nm (same as the assumptions of [10]). Further, the cells are placed on a grid with a cell center-to-center distance of 20 nm. A maximum of 16 cells per clock zone is used (as in [10]; it is to be noted that the number of cells per clock zone affects the overall delay).

4.3.2 Simulation Engine

The coherence vector engine has been used for simulations. The options for the simulation were as follows (and are in agreement with the suggestions in [11]): *Temperature*: 1 K; *Relaxation Time*: 1×10^{-15} s; *Time Step*: 1×10^{-16} s; *Total Simulation Time*: 7×10^{-11} s; *Radius of Effect*: 80 nm; *Relative Permittivity*: 12.9 *Layer Separation*: 11.5 nm; *Euler Method*; *Randomize Simulation Order*.

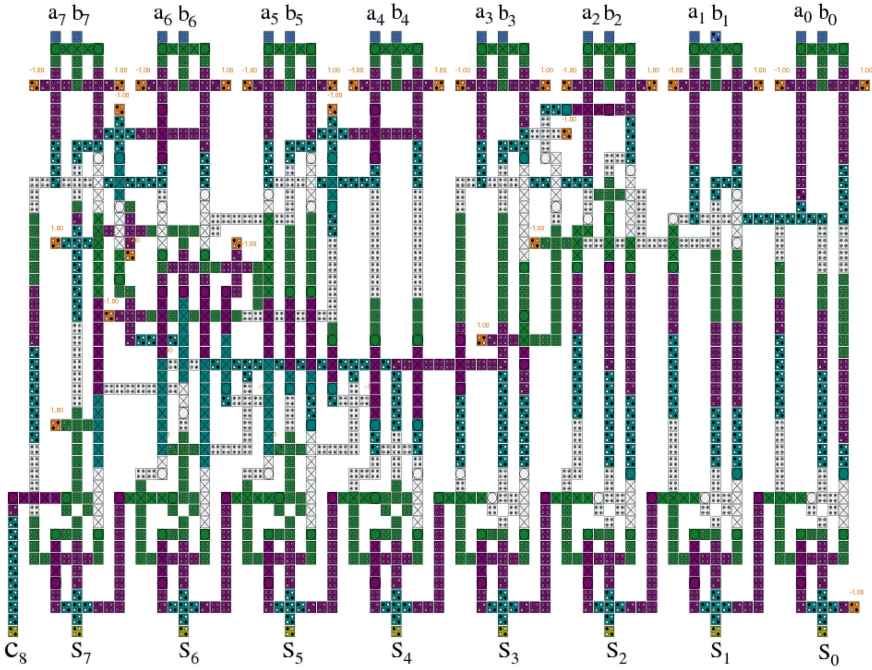


Fig. 4.16 QCADesigner layout for 8-bit Ladner-Fischer adder

4.3.3 Layout Level Implementation of RCA and Prefix Adders

Figure 4.13 shows the QCADesigner layout for an 8-bit ripple carry adder. The layout is labelled to indicate majority gates as well as the outputs, namely $S_0, S_1, \dots, S_7$ and C_8 . Figures 4.14, 4.15 and 4.16, show the layout of an 8-bit Brent-Kung, 8-bit Kogge-Stone and 8-bit Ladner-Fischer adder respectively.

Figures 4.17, 4.18 and 4.19 give the layout for 16-bit Kogge-Stone, Ladner-Fischer and Brent-Kung adders respectively. Figure 4.20 shows the simulation results of a 16-bit Brent-Kung adder (the complete sum output includes also the top carry out). The first set of inputs for simulation shown in Fig. 4.20 corresponds to $A[15 : 0] = 0$; $B[15 : 0] = 0$ (since the initial carry is set to 0 as in [8], this is retained for simulations). The output, $SUM[15 : 0] = 0$ appears after 4 clock delays. (This is also reflected in the “Delay” column in Table 7.2.) The second set of inputs corresponds to $A[15 : 0] = 1024$; $B[15 : 0] = 512$. The output, $SUM[15 : 0] = 1536$.

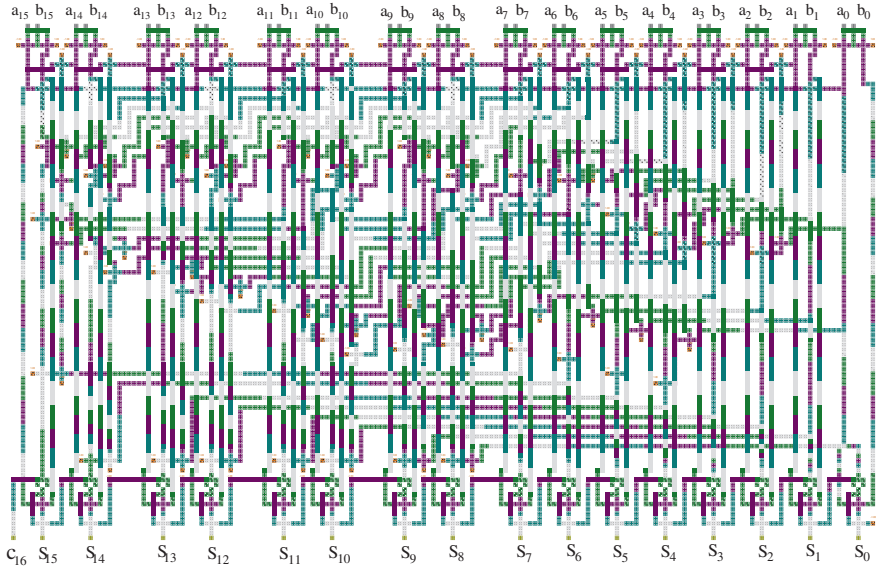


Fig. 4.17 QCADesigner layout for a 16-bit Kogge-Stone adder

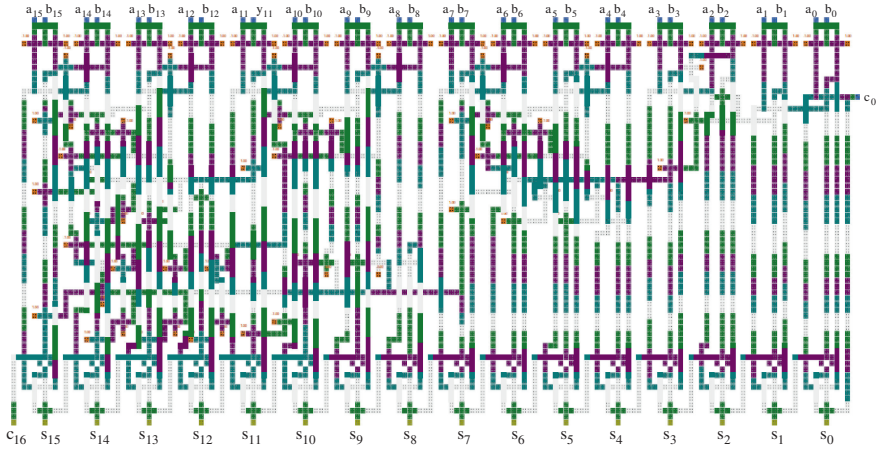


Fig. 4.18 QCADesigner layout for a 16-bit Ladner-Fischer adder

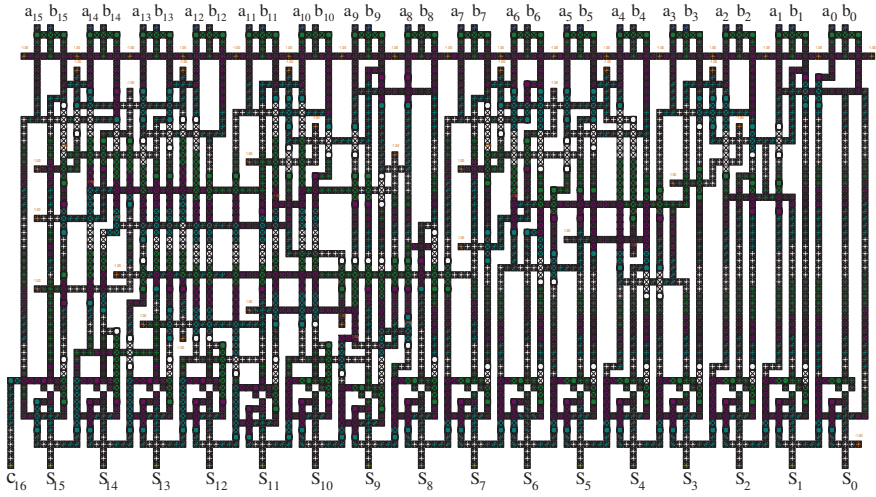


Fig. 4.19 QCADesigner layout for a 16-bit Brent-Kung adder

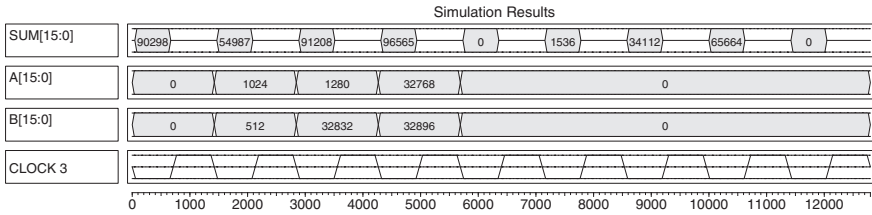


Fig. 4.20 Simulation results for 16-bit Brent-Kung adder

4.4 Comparison Studies

4.4.1 Cell Count, Area and Delay for Various Adders

Table 4.2 gives the details of cell count, area and delay for the proposed RCA, Kogge-Stone, Ladner-Fischer, and Brent-Kung adders. Comparisons with prior work are also presented in Table 4.2. The comparisons are primarily with the results reported in [10, 12, 13]. It is worth noting that the proposed Brent-Kung adder has the lowest delay among prefix adders and this can be attributed due to optimization of majority logic (as well as wires in the design).

Plots of cell count, delay and area as a function of adder size are given in Figs. 4.21, 4.22 and 4.23. These plots are based on the analysis of the QCA layouts.

We present time and space estimates for various adders using the order notation followed by the authors of [12] (this is based on examining the growth of cell count, delay etc. as the adder size doubles). From the statistics, cell counts for a QCA adder

Table 4.2 Cell count, area, overall size, delay and number of total clock phases for different methods

Approach	Cell count	Area ($\mu\text{m} \times \mu\text{m}$)	Overall Size (μm) ²	Delay (clk)	Number of clock phases
RCA4 (Prop)	339	0.82×0.31	0.2542	$1\frac{3}{4}$	7
RCA8 (Prop)	712	1.62×0.46	0.7452	$2\frac{3}{4}$	11
RCA16 (Prop)	1602	3.22×0.62	1.996	$4\frac{3}{4}$	19
RCA32 (Prop)	3901	6.46×1.00	6.46	$8\frac{3}{4}$	35
RCA64 (Prop)	10,926	12.9×1.66	20.916	$16\frac{3}{4}$	67
BK4	680	0.75×0.86	0.645	$1\frac{3}{4}$	7
BK8	1782	1.40×1.07	1.49	$2\frac{2}{4}$	10
BK16	4350	2.69×1.32	3.55	4	16
BK32	11,825	5.61×1.92	10.77	$6\frac{3}{4}$	27
BK64	30,145	11.6×2.69	31.20	$11\frac{2}{4}$	46
KS4	815	0.82×0.9	0.738	2	8
KS8	2938	1.65×1.46	2.409	$3\frac{1}{4}$	13
KS16	8589	3.42×1.93	6.006	6	24
KS32	23,024	7.01×2.62	18.366	$10\frac{1}{4}$	41
KS64	58,945	15.6×3.28	51.17	$17\frac{2}{4}$	70
LF4	698	0.87×0.71	0.618	2	8
LF8	1994	1.67×1.06	1.7702	$2\frac{3}{4}$	11
LF16	5376	3.29×1.46	4.8	$4\frac{1}{4}$	17
LF32	13,552	6.56×2.02	13.25	$7\frac{2}{4}$	30
LF64	35,850	13.4×2.81	37.654	$13\frac{2}{4}$	54
RCA4 ^a	651	1.67×0.72	1.202	$4\frac{1}{4}$	17
RCA8 ^a	1499	3.43×1.04	3.56	$8\frac{1}{4}$	33
RCA16 ^a	3771	6.97×1.69	11.779	$16\frac{1}{4}$	65
RCA32 ^a	10,619	14.03×3.01	42.23	$32\frac{1}{4}$	129
RCA64 ^a	33,531	28.18×5.65	159.2	$64\frac{1}{4}$	257
CFA4 ^b	371	0.90×0.45	0.405	$1\frac{1}{2}$	6
CFA8 ^b	789	1.79×0.53	0.9487	$2\frac{2}{4}$	10
CFA16 ^b	1769	3.55×0.69	2.45	$4\frac{2}{4}$	18
CFA32 ^b	4305	7.09×1.03	7.3	$8\frac{2}{4}$	34
CFA64 ^b	11,681	14.15×1.71	24.196	$16\frac{2}{4}$	66
BK8 ^c	1462	1.26×0.84	1.06	$2\frac{2}{4}$	10

Prop Proposed, *BK* Brent-Kung, *KS* Kogge-Stone, *LF* Ladner-Fischer

^aRefers to [12]

^bRefers to [10]

^cRefers to [13]

with n -bit operands are roughly $O(n^{1.33})$ for Brent-Kung, $O(n^{1.56})$ for Kogge-Stone, $O(n^{1.42})$ for Ladner-Fischer, $O(n^{1.24})$ for proposed RCA, $O(n^{1.21})$ for CFA [10] and

Fig. 4.21 Complexity in terms of cell count for various adders (the plots for CFA and RCA correspond to designs in [10, 12] respectively)

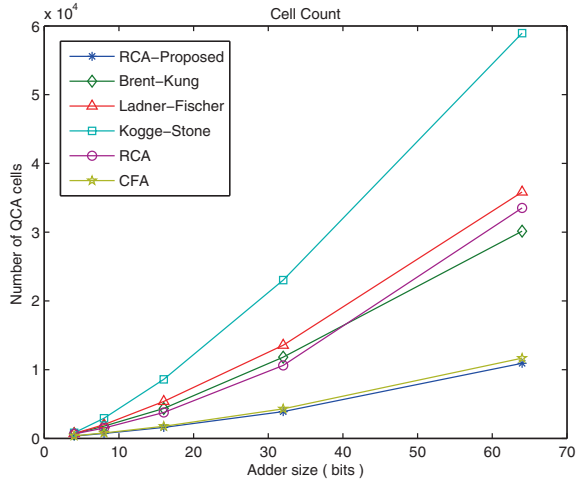
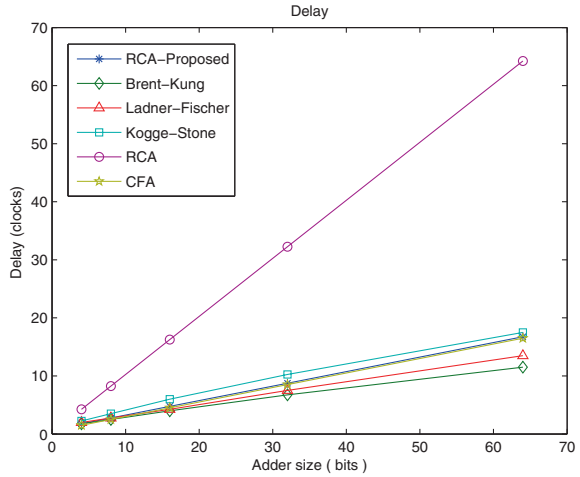
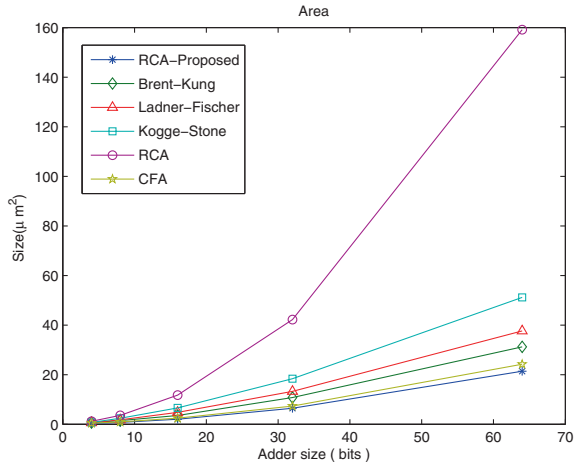


Fig. 4.22 Time complexity in terms of delay for various adders (the plots for CFA and RCA correspond to designs in [10, 12] respectively)



$O(n^{1.35})$ for RCA [12]. Area estimates are $O(n^{1.39})$ for Brent-Kung, $O(n^{1.53})$ for Kogge-Stone, $O(n^{1.48})$ for Ladner-Fischer, $O(n^{1.56})$ for proposed RCA, $O(n^{1.42})$ for CFA [10] and $O(n^{1.72})$ for RCA [12]. Delay estimates are given by $O(n^{0.68})$ for Brent-Kung, $O(n^{0.78})$ for Kogge-Stone, $O(n^{0.7})$ for Ladner-Fischer, $O(n^{0.83})$ for proposed RCA, $O(n^{0.87})$ for CFA [10] and $O(n^{0.97})$ for RCA [12]. From the order results (using the Big Oh notation), we note that Brent-Kung has, in general, lower complexity than other adders in the QCA model. We further note that a feature of the Kogge-Stone adder in the QCA domain is that the growth (ratio) in delay as well as area (when the adder width is doubled) remains nearly constant.

Fig. 4.23 Area complexity for various adders (the plots for CFA and RCA correspond to designs in [10, 12] respectively)



4.5 Summary

We have examined QCA design of various multi-bit adders in this chapter. We began with the simplest multi-bit adder, namely the ripple carry adder and investigated majority logic optimization. The analysis for the ripple carry adder has been based on an efficient QCA-based 1-bit full adder design. Low-delay adders based on the notion of prefix computation have been then considered. It has been shown that the Brent-Kung prefix adder performs best (among prefix adders) in the QCA setting.

References

1. I. Koren, *Computer Arithmetic Algorithms* (A.K. Peters Ltd., Natick, 2002)
2. V. Pudi, K. Sridharan, Low complexity design of ripple carry and Brent-Kung adders in QCA. *IEEE Trans. Nanotechnol.* **11**(1), 105–119 (2012)
3. P.D. Tougaw, C.S. Lent, Logical devices implemented using quantum cellular automata. *J. Appl. Phys.* **75**(3), 1818–1825 (1994)
4. R. Zhang, K. Walus, W. Wang, G.A. Jullien, A method of majority logic reduction for quantum cellular automata. *IEEE Trans. Nanotechnol.* **3**(4), 443–450 (2004)
5. K. Walus, G.A. Jullien, Design tools for an emerging SoC technology: quantum-dot cellular automata. *Proc. IEEE* **94**(6), 1225–1244 (2006)
6. P.M. Kogge, H.S. Stone, A parallel algorithm for the efficient solution of a general class of recurrent equations. *IEEE Trans. Comput.* **C-22**, 786–793 (1973)
7. R.E. Ladner, M.J. Fischer, Parallel prefix computation. *J. Assoc. Comput. Mach.* **27**(4), 831–838 (1980)
8. R.P. Brent, H.T. Kung, A regular layout for parallel adders. *IEEE Trans. Comput.* **C-31**, 260–264 (1982)
9. K. Walus, T. Dysart, G. Jullien, R. Budiman, QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **3**(1), 26–29 (2004)

10. H. Cho, E.E. Swartzlander, Adder and multiplier designs in quantum-dot cellular automata. *IEEE Trans. Comput.* **58**(6), 721–727 (2009)
11. K. Walus, G. Schulhof, G. Jullien, Implementation of a simulation engine for clocked molecular QCA, in *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'06)*, pp. 2128–2131 (2006)
12. H. Cho, E.E. Swartzlander, Adder designs and analyses for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **6**(3), 374–383 (2007)
13. S. Perri, P. Corsonello, New methodology for the design of efficient binary addition circuits in QCA. *IEEE Trans. Nanotechnol.* **11**(6), 1192–1200 (2012)

Chapter 5

Design of a Hybrid Adder in QCA

5.1 Introduction

In the previous chapter, we presented efficient QCA realizations of existing adders based on new results on majority logic. In this chapter, we will pursue another direction. In particular, we take up the task of designing a custom adder for the QCA domain. We call this adder as *hybrid adder*. Some of the results presented in this chapter have been reported in [1].

5.2 Preliminaries for the Hybrid Adder in QCA

We have observed in the previous chapter that the proposed ripple carry adder is simple and its area requirement is small. Also, the cell count requirement is fairly small. However, the growth in delay as the adder size increases is worrisome. The prefix adders offer an advantage with respect to delay but, in general, require somewhat large area (owing to increased requirement of majority gates) and have high cell count. It is therefore of interest to explore the development of an adder which can combine the benefits of the ripple carry adder and a prefix adder.

The proposed hybrid adder, as the name suggests, is a hybrid of two types of adders. In particular, the hybrid adder presented in this chapter is a combination of ripple carry and the prefix adder. Any of the prefix adders can be used in principle (and our hybrid adder resembles to some extent the Ladner-Fischer adder).

5.3 Design of 16-Bit Hybrid Adder in QCA

It is well-known that (the speed of) carry computation determines the performance of adders. The design of the hybrid adder is motivated by the following questions.

1. Does calculation of all carries require prefix style to obtain low delay?
2. Given that the amount of interconnect determines not only the cell count but also the delay in the QCA setting and the fact that a ripple carry adder has low requirement in terms of majority logic and interconnect, can at least some of the carries be calculated in ripple carry fashion?
3. What is the extent to which we can use ripple carry approach without adversely affecting the overall delay?

Questions (1) and (2) are related. The answers to these questions depend on properties of majority logic in the QCA setting. The majority logic properties determine which of the carries benefit from prefix style computation and which do not. Several new results on majority logic are presented. While the new results are applicable to all carries in general, not all the carries can be computed with low delay by the application of the results. Six of the sixteen carries in a 16-bit hybrid adder can take advantage of the new results while the remaining ten carries will be computed in ripple carry style.

Figure 5.1 shows a general tree structure [2] for computation of c_{16} , c_{12} and c_8 . Note that Fig. 5.1 is applicable to all prefix adders in general. The proposed hybrid adder is based on the idea that a number of carries not explicitly labelled in Fig. 5.1 (in particular, $c_1, c_2, c_3, c_4, c_5, c_6, c_9, c_{10}, c_{13}, c_{14}$) can be computed using a ripple carry style leading to a highly efficient adder in terms of majority gates and delay.

The QCA design of the hybrid adder relies on a result presented in the previous chapter so we will restate it without proof.

Lemma 5.1 *Let f_1, f_2 and f_3 be three Boolean functions such that f_1 and f_2 satisfy $f_1 f_2 = f_1$ and $f_1 + f_2 = f_2$.*

Then

$$M(f_1, f_2, f_3) = f_1 + f_2 f_3$$

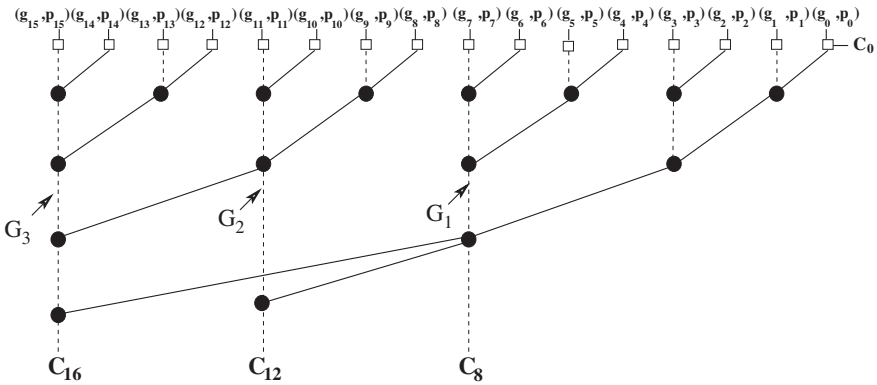


Fig. 5.1 Tree structure for c_8, c_{12} and c_{16} carries

Since the amount of majority logic also influences interconnect (and therefore delay, area etc.), we will investigate first the majority gate requirement for carries c_{16} , c_{12} and c_8 . With reference to Fig. 5.1, we note that c_{16} , c_{12} , and c_8 depend on G_3 , G_2 , and G_1 respectively. Let G_j , $j = 1, 2, 3$ represent the generate of $(g_{i+3}, p_{i+3}) \circ (g_{i+2}, p_{i+2}) \circ (g_{i+1}, p_{i+1}) \circ (g_i, p_i)$ where $i = 4, 8$ and 12 respectively. G_j can be expanded as

$$G_j = g_{i+3} + p_{i+3}(g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1}g_i)) \quad (5.1)$$

G_j in Eq. (5.1) can be expressed using majority gates. Using Lemma 5.1, we note that $g_{i+1} + p_{i+1}g_i$ can be expressed as $M(g_{i+1}, p_{i+1}, g_i)$. Extending this idea to other terms in Eq. (5.1), we have the form given by Eq. (5.2).

$$G_j = M(g_{i+3}, p_{i+3}, M(g_{i+2}, p_{i+2}, M(g_{i+1}, p_{i+1}, g_i))) \quad (5.2)$$

Direct evaluation of G_j using Eq. (5.2) requires up to 10 majority gates since g_{i+3} , p_{i+3} , g_{i+2} , p_{i+2} , g_{i+1} , p_{i+1} and g_i require one majority gate each besides the 3 majority gates shown explicitly in the expression for G_j . Consequently, a total of 30 majority gates is required for G_1 , G_2 and G_3 . We now present a new result that reduces the majority gate requirement substantially.

Lemma 5.2 *Let f_1 , f_2 and f_3 be three Boolean functions such that $f_1 = x_1y_1$ and $f_2 = x_1 + y_1$ where x_1 and y_1 are two binary inputs. Then*

$$M(f_1, f_2, f_3) = M(x_1, y_1, f_3)$$

Proof $M(f_1, f_2, f_3)$ is given by

$$\begin{aligned} M(f_1, f_2, f_3) &= f_1f_2 + f_1f_3 + f_2f_3 \\ &= f_1f_2 + (f_1 + f_2)f_3 \\ &= x_1y_1(x_1 + y_1) + (x_1y_1 + x_1 + y_1)f_3 \\ &= x_1y_1 + (x_1 + y_1)f_3 \\ &= M(x_1, y_1, f_3) \quad Q.E.D \end{aligned}$$

Using Lemma 5.2, we can compute $g_{i+1} + p_{i+1}g_i$ as $M(x_{i+1}, y_{i+1}, g_i)$. In other words, we can replace g_i s and p_i s in Eq. (5.2) by x_i and y_i except for the last term (g_i). Consequently, up to 6 majority gates are eliminated and we have Eq. (5.3). In particular, G_1 requires 4 majority gates. G_2 and G_3 also require 4 majority gates each.

$$G_j = M(x_{i+3}, y_{i+3}, M(x_{i+2}, y_{i+2}, M(x_{i+1}, y_{i+1}, g_i))) \quad (5.3)$$

Using G_1 , G_2 and G_3 , we can compute c_8 , c_{12} and c_{16} as shown in Eqs. (5.4), (5.5) and (5.6) respectively. It is to be emphasized that obtaining c_8 (as also c_{12} and

c_{16}) this way using g_i s and p_i s provides for higher parallelism (when compared to a ripple carry adder).

$$c_8 = G_1 + p_7 p_6 p_5 p_4 c_4 \quad (5.4)$$

$$c_{12} = G_2 + p_{11} p_{10} p_9 p_8 c_8 \quad (5.5)$$

$$c_{16} = G_3 + p_{15} p_{14} p_{13} p_{12} G_2 + p_{15} p_{14} \cdots p_8 c_8 \quad (5.6)$$

Besides c_8 , c_{12} and c_{16} , three other carries are computed in prefix style. These are c_7 , c_{11} and c_{15} . The calculations are based on Eqs. (5.7), (5.8) and (5.9).

$$c_7 = M(x_6, y_6, M(x_5, y_5, g_4)) + p_6 p_5 p_4 c_4 \quad (5.7)$$

$$c_{11} = M(x_{10}, y_{10}, M(x_9, y_9, g_8)) + p_{10} p_9 p_8 c_8 \quad (5.8)$$

$$c_{15} = M(x_{14}, y_{14}, M(x_{13}, y_{13}, g_{12})) + p_{14} p_{13} p_{12} G_2 + p_{14} p_{13} \cdots p_8 c_8 \quad (5.9)$$

We will now examine majority gate requirements for realization of Eqs. (5.4)–(5.6). Suppose in Eq. (5.4), G_1 as well as $p_7 p_6 p_5 p_4$ and c_4 are already available. Then c_8 requires two majority gates by a direct implementation, one for the AND operation and another for the OR operation. Similar arguments can be given for c_{12} and c_{16} . The requirement can be reduced to one majority gate by applying Lemmas 5.1, 5.3 and 5.4.

Lemma 5.3

$$G_j + p_{i+3} = p_{i+3}$$

Proof By expanding G_j , we have

$$\begin{aligned} G_j + p_{i+3} &= g_{i+3} + p_{i+3}(g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1}g_i)) \\ &\quad + p_{i+3} \\ &= g_{i+3} + p_{i+3}[g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1}g_i) + 1] \\ &= g_{i+3} + p_{i+3} \\ &= x_{i+3}y_{i+3} + x_{i+3} + y_{i+3} \\ &= p_{i+3} \quad Q.E.D \end{aligned}$$

Lemma 5.4

$$G_j p_{i+3} = G_j$$

Proof By expanding G_j , we have

$$\begin{aligned} G_j p_{i+3} &= [g_{i+3} + p_{i+3}(g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1}g_i))]p_{i+3} \\ &= g_{i+3}p_{i+3} + p_{i+3}p_{i+3}(g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1}g_i)) \\ &= g_{i+3} + p_{i+3}(g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1}g_i)) \\ &= G_j \quad Q.E.D \end{aligned}$$

Majority logic expressions for c_8 , c_{12} and c_{16} can now be given as shown in Eqs. (5.10), (5.11) and (5.12).

$$c_8 = M(G_1, p_7, p_6 p_5 p_4 c_4) \quad (5.10)$$

$$c_{12} = M(G_2, p_{11}, p_{10} p_9 p_8 c_8) \quad (5.11)$$

$$c_{16} = M(G_3 + p_{15} p_{14} p_{13} p_{12} G_2, p_{15}, p_{14} \cdots p_8 c_8) \quad (5.12)$$

The term $G_3 + p_{15} p_{14} p_{13} p_{12} G_2$ in Eq. (5.12) is realized as shown in Eq. (5.13).

$$G_3 + p_{15} p_{14} p_{13} p_{12} G_2 = M(G_3, p_{15}, p_{14} p_{13} p_{12} G_2) \quad (5.13)$$

The requirements for majority gates for $c_7, c_8, c_{11}, c_{12}, c_{15}$ and c_{16} can now be summarized as follows. For c_7 in Eq. (5.7), 10 majority gates are required (including the requirements for p_i s and g_4 , majority gate for AND of p_6 and p_5 , AND of p_4 and c_4 , AND of $p_6 p_5$ and $p_4 c_4$ and the OR operation). For c_8 in Eq. (5.10), 3 majority gates are required (by reutilization of majority gates for p_6, p_5 etc.) since p_7, G_1 and $M()$ require 1 majority gate each (G_1 is given by $M(x_7, y_7, M(x_6, y_6, M(x_5, y_5, g_4)))$) and the majority gates involved in computation of c_7 , namely $M(x_6, y_6, M(x_5, y_5, g_4)), p_6 p_5 p_4 c_4$ are reutilized for G_1 and c_8). For c_{11} , the calculations are similar to c_7 and so 10 majority gates are required. For c_{12} , the calculations are similar to c_8 , therefore 3 majority gates are required. For c_{15} , the calculations for the part given by $M(x_{14}, y_{14}, M(x_{13}, y_{13}, g_{12})) + p_{14} p_{13} p_{12} G_2$ are similar to c_{11} hence 10 majority gates are required. For $.. + p_{14} p_{13} p_{12} p_{11} p_{10} p_9 p_8 c_8$, 4 majority gates are required (1 majority gate for AND of p_{11} with $p_{10} p_9 p_8$, 1 majority gate for AND of $p_{14} p_{13} p_{12}$ with $p_{11} p_{10} p_9 p_8$, 1 majority gate for AND of $p_{14} p_{13} p_{12} p_{11} p_{10} p_9 p_8$ with c_8 and 1 for the overall OR operation). Hence, altogether 14 majority gates are required. For c_{16} , 4 majority gates are required (one for $M(.)$ and three for Eq. (5.13) and this is in view of the following facts: (i) c_{16} computation reutilizes majority gates involved in c_{15} and this is similar to the manner in which c_8 computation reutilizes majority logic used for c_7 (ii) The first term within $M()$ in Eq. (5.12) is realized as shown in Eq. (5.13) and (iii) In Eq. (5.13), G_3, p_{15} and $M(G_3, \dots)$ require 1 majority gate each. Figure 5.2 illustrates the generation of carries of a 16-bit hybrid adder.

The calculation of the remaining carries, namely $c_i, i = 1, 2, 3, 4, 5, 6, 9, 10, 13, 14$ can be done in ripple-carry style (each c_i can be expressed in terms of $M(x_{i-1}, y_{i-1}, c_{i-1})$). Truly, the calculation of c_{13} depends only on c_{12} while the calculation of c_9 depends just on c_8 . In other words, c_{13} does not have to wait for c_9 . Also, the majority gate requirement is quite small. For example, carry c_5 can be calculated as $M(x_4, y_4, c_4)$. So 10 majority gates are required for these 10 carries. The sum calculation requires 32 majority gates and 16 inverters for a 16-bit hybrid adder.

Remark 5.5 In Fig. 5.2, some of the carries computed in ripple carry style are shown enclosed in ellipses (and labelled with \$). For example, C_5 and C_6 in Stage 3 are computed in ripple carry style with one majority gate (delay) difference between C_5 and C_6 . However, C_7 and C_8 are computed in prefix style and in parallel with C_6 .

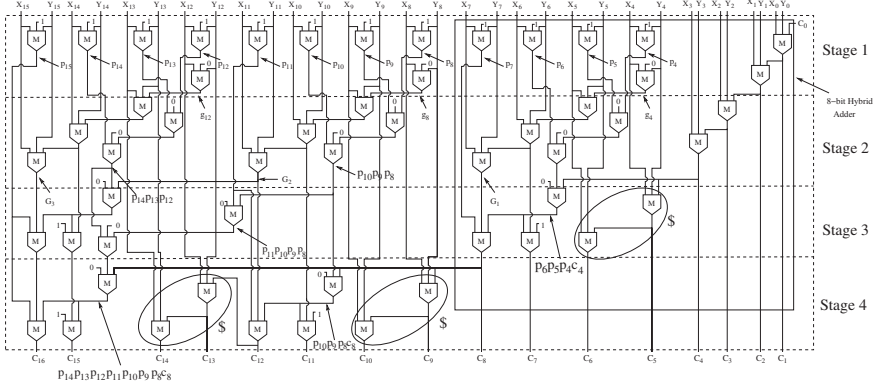


Fig. 5.2 Carry generation for a 16-bit hybrid adder

Similarly, in Stage 4, C_9 and C_{10} are computed in ripple carry style but C_{10} is computed in parallel with C_{11} and C_{12} (the latter are obtained in prefix style).

5.3.1 Generalization to an n -bit Hybrid Adder

Let $I(n)$ denote the number of majority gates required (at most) for an n -bit hybrid adder. For ease of understanding, we define a 4-bit hybrid block as shown in Fig. 5.4. For an n -bit hybrid adder, $(\frac{n}{4} - 1)$ 4-bit hybrid blocks are required (the case of $n = 32$ is shown in Fig. 5.3 and requires 7 hybrid blocks).

We present calculations for majority gates for a 32-bit hybrid adder using Fig. 5.3. This forms the basis of bounds we obtain on majority gates for an n -bit hybrid adder. The carries not explicitly shown in Fig. 5.3 are calculated in ripple carry style. A small shaded circle in Fig. 5.3 represents the associative operation “ $\circ$ ” while the pentagon symbol indicates the majority gate operation shown in Eqs. (5.10)–(5.12).

Consider the four 4-bit hybrid blocks starting from the leftmost one in Fig. 5.3. These correspond to 16 carries from C_{17} to C_{32} (including the ones found in ripple carry style). There are five components that contribute to the overall majority logic requirements for these 16 carries. These can be described as follows.

- (i) Stages 1 and 2 of the 32-bit adder correspond to the 4-bit hybrid block (marked within Fig. 5.4). Each 4-bit hybrid block outputs only a three-term propagate (i.e. $p_i + 2p_{i+1}p_i$ in Fig. 5.4) and each block requires 10 majority gates. The four blocks together require 40 majority gates.
- (ii) Stages 3 and 4 (in the left half in Fig. 5.3) contain two associative operators and two majority operators each. Each majority operation requires one majority gate while one associative operation requires three majority gates. Total number of majority gates for these associative operations (stages 3 to 4) in a 32-bit adder is $16(2(2 * 1 + 2 * 3))$.

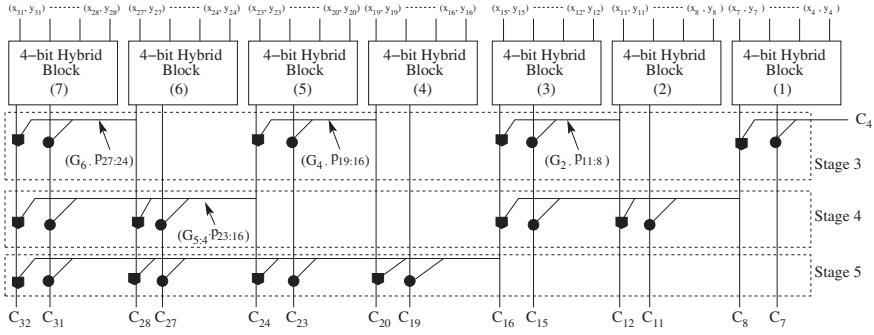
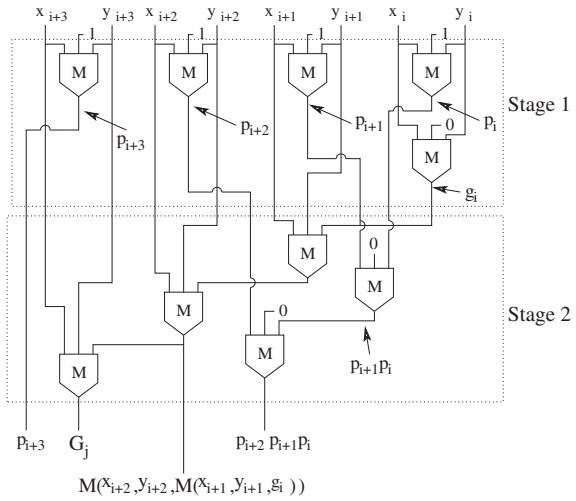


Fig. 5.3 32-bit hybrid adder block diagram; $p_{23:16}$ for example represents $p_{23}p_{22} \dots p_{16}$; $G_{5:4}$ represents $G_5 + p_{23}p_{22}p_{21}p_{20}G_4$

Fig. 5.4 A 4-bit hybrid adder block



- (iii) The lowermost stage (Stage 5) of the 32-bit hybrid adder contains 8 associative operations and 4 majority operations leading to 12 (obtained as $4(1 + 2)$) majority gates.
- (iv) One additional majority gate is required for each of the two terms, namely $p_{27:24}$, $p_{19:16}$ in Stage 3 and one additional majority gate is required for $p_{23:16}$ in Stage 4 (we note that $P_{11:8}$ in Fig. 5.3 also requires one additional majority gate but is part of lower half and hence it will get included in $I_c(16)$ calculations).
- (v) The remaining carries (eight of them) are computed in ripple carry style and therefore require 8 majority gates.

Summing up the requirements for (i)–(v), we have $I_c(32) = 4 * 10 + 16 + 12 + 3 + 8 + I_c(16)$. Adding the 64 majority gates for sum, we have $I(32) = I_c(32) + 64 = 197$.

Remark 5.6 The four outputs of the 4-bit hybrid adder block in Fig. 5.4 can be related to two carries (shown as outputs of each 4-bit hybrid block) in Fig. 5.3 as follows. Consider the rightmost 4-bit hybrid block in Fig. 5.3 whose inputs are $(x_4, y_4), \dots, (x_7, y_7)$. Here $i = 4$ and $j = \frac{i}{4}$. Then, as indicated in Eqs. (5.7) and (5.10), we have

$$\begin{aligned} c_7 &= M(x_6, y_6, M(x_5, y_5, g_4)) + p_6 p_5 p_4 c_4 \\ c_8 &= M(G_1, p_7, p_6 p_5 p_4 c_4) \end{aligned}$$

Lemma 5.7 *The total number of majority gates required for an n -bit hybrid adder, $I(n)$ is given by*

$$I(n) = \frac{n}{2}(\log_2 n + 8) - \log_2 n - 6$$

Proof The computation of carries of an n -bit hybrid adder requires $\log_2(n)$ stages. The total number of majority gates is determined using a recursive formulation noting that the lower order carries (namely, carries from 1 to $\frac{n}{2}$) are those of an $\frac{n}{2}$ -bit adder (we can proceed this way up to a 4-bit adder).

For carries from $\frac{n}{2} + 1$ to n , the majority gates required can be obtained as follows.

- (i) The carries from $\frac{n}{2} + 1$ to n contain $\frac{n}{8}$ 4-bit hybrid blocks. Each block requires 10 majority gates. So total number of majority gates, denoted by H_1 , is given by

$$H_1 = \frac{n}{8} \times 10 = \frac{5n}{4}$$

- (ii) Stages 3 to $(\log_2 n) - 1$ contain $\frac{n}{16}$ number of majority operations and $\frac{n}{16}$ number of associative operations in each stage. Each majority operation requires one majority gate and each associative operation requires 3 majority gates. So total number of majority gates, denoted by H_2 , is

$$H_2 = (\log_2 n - 3)(1 + 3) \frac{n}{16} = (\log_2 n - 3) \frac{n}{4}$$

- (iii) At the lowermost stage (i.e. $\log_2(n)$), there are $\frac{n}{8}$ majority operations and $\frac{n}{8}$ associative operations (note that an associative operation requires 2 majority gates at the lowermost level since there is no need for propagate). So the number of majority gates for the last stage, denoted by H_3 , is

$$H_3 = 3 \left(\frac{n}{8} \right)$$

- (iv) Besides H_1 , H_2 and H_3 , additional gates are required especially from stages 3 to $(\log_2 n - 1)$ and this corresponds to $2^{k-4}, 2^{k-5}, \dots, 2^0$. Denoting the additional gates by H_4 , we have

$$\begin{aligned}
H_4 &= 2^{k-4} + 2^{k-5} + \dots + 2^0 \\
&= 2^{k-3} - 1 \\
&= \frac{n}{8} - 1 \quad [\because k = \log_2 n]
\end{aligned}$$

- (v) Finally, $\frac{n}{4}$ carries are calculated in ripple carry style. and the majority gates requirement for this is given by

$$H_5 = \frac{n}{4}$$

Then

$$\begin{aligned}
I_c \left(n : \frac{n}{2} + 1 \right) &= H_1 + H_2 + H_3 + H_4 + H_5 \\
&= \frac{5n}{4} + (\log_2 n - 3) \frac{n}{4} + 3 \left(\frac{n}{8} \right) + \frac{n}{8} - 1 + \frac{n}{4} \\
&= \frac{n}{4} (\log_2(n) + 5) - 1
\end{aligned}$$

The total number of majority gates required to generate all carries is given by

$$\begin{aligned}
I_c(n) &= I_c \left(n : \frac{n}{2} + 1 \right) + I_c \left(\frac{n}{2} \right) \\
&= \frac{n}{4} (\log_2 n + 5) - 1 + I_c \left(\frac{n}{2} \right) \\
&= \frac{n}{4} (\log_2 n + 5) - 1 + \frac{n}{8} \left(\log_2 \frac{n}{2} + 5 \right) - 1 + I_c \left(\frac{n}{4} \right) \\
&= \frac{n}{4} (\log_2 n + 5) - 1 + \frac{n}{8} \left(\log_2 \frac{n}{2} + 5 \right) - 1 + \dots + \frac{8}{4} (\log_2 8 + 5) \\
&\quad - 1 + I_c(4) \\
&= \frac{n}{4} \left[\log_2 n + \frac{\log_2 \frac{n}{2}}{2} + \frac{\log_2 \frac{n}{4}}{4} + \dots + \frac{\log_2 \left(\frac{n}{8} \right)}{\left(\frac{n}{8} \right)} \right] + \\
&\quad \frac{5n}{4} \left[1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{\left(\frac{n}{8} \right)} \right] - (\log_2 n - 2) + I_c(4) \\
&= \frac{n}{4} \left[2 \log_2 n - \frac{8}{n} - 2 \right] + \frac{5n}{4} \left[2 - \frac{8}{n} \right] - \log_2 n + 2 + I_c(4) \\
&= \frac{n \log_2 n}{2} - 2 - \frac{n}{2} + \frac{5n}{2} - 10 - \log_2 n + 2 + I_c(4) \\
&= \frac{n}{2} (\log_2 n + 4) - \log_2 n - 10 + I_c(4) \\
&= \frac{n}{2} (\log_2 n + 4) - \log_2 n - 6 \quad [\text{Since } I_c(4) = 4]
\end{aligned}$$

For an n -bit hybrid adder, each sum requires 2 majority gates. So total number of majority gates required to compute all sums is given by

$$I_s(n) = 2n$$

Hence, the total number of majority gates required for an n -bit hybrid adder is given by

$$\begin{aligned} I(n) &= I_c(n) + I_s(n) \\ &= \frac{n}{2}(\log_2 n + 4) - \log_2 n - 6 + 2n \\ &= \frac{n}{2}(\log_2 n + 8) - \log_2 n - 6 \quad Q.E.D \end{aligned}$$

Remark 5.8 The formula given for $I(n)$ is applicable for $n > 4$ since ripple carry style is used for a four-bit hybrid adder. Further, the recursive formulation assumes that n is a power of two. From Lemma 5.7, we can infer that for a 64-bit hybrid adder, at most 436 majority gates and 64 inverters are required.

Remark 5.9 Instead of a 4-bit hybrid adder block, one can have an 8-bit hybrid adder block as shown in Fig. 5.5. However, there is no gain with respect to delay since calculation of c_8 using an 8-bit hybrid adder block requires 8 gate delays. On the other hand, with a 4-bit hybrid adder block, it requires only 6 gate delays to compute c_8 . Similar comments apply to larger size hybrid adder blocks.

5.3.2 Delay Analysis for a Hybrid Adder

We study delay for a hybrid adder in terms of the adder size. We enumerate the components in the overall delay.

- Calculation of 4-bit hybrid block (Fig. 5.3) happens in two stages and requires 4 majority gates leading to a delay of 1 clock.
- Each stage from Stage 3 up to Stage $\log_2 n$ requires 0.5 clocks (two majority gate delay)
- Sum computation requires 0.5 clocks (delay for 2 majority gates) while input requires 0.25 clocks.
- Therefore, the total delay is given $(1 + \frac{2}{4}(\log_2 n - 2) + \frac{1}{2} + \frac{1}{4} + \text{wire delay})$. This simplifies to $\frac{1}{2} \log_2(2n) + \frac{1}{4} + \text{wire delay}$.

Wire delay, as indicated earlier, depends on the number of cells per clock zone. For a 16-bit hybrid adder, the wire delay (in QCADesigner) is $\frac{1}{2}$ while for 32-bit and 64-bit hybrid adders, the wire delay is $2\frac{1}{4}$ and $5\frac{3}{4}$ clocks respectively.

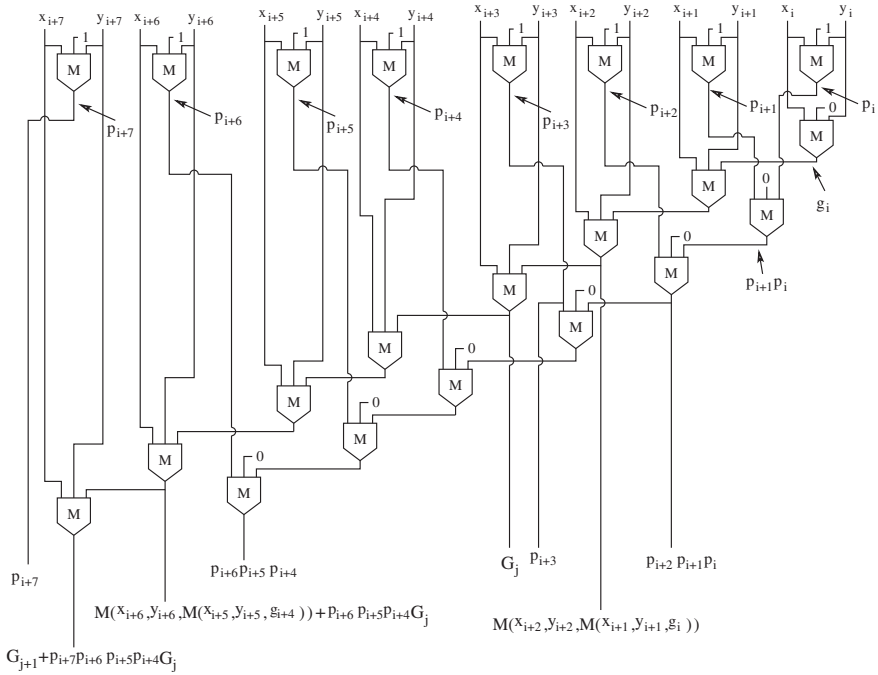


Fig. 5.5 An 8-bit hybrid adder block

5.4 QCA Designer Layout Diagrams and Simulation Results

In this section, we present layouts obtained from QCA Designer [3] and the simulation results for hybrid adder. Figure 5.6 shows the QCA Designer layout of a 16-bit hybrid adder while Fig. 5.7 gives the simulation results for 16-bit hybrid adder. The first set of inputs for simulation corresponds to $X[15 : 0] = 0$; $Y[15 : 0] = 0$ and $C[0] = 0$. The output, $S[15 : 0] = 0$ appears after $3\frac{1}{4}$ clock delays and is labelled in the figure (this is also reflected in the “Delay” column in Table 7.2). Dummy values inserted by the QCA Designer tool are shown before the first sum output of 0. The second set of inputs corresponds to $X[15 : 0] = 65535_{10}$; $Y[15 : 0] = 65535_{10}$; $C[0] = 0$. The output, $S[15 : 0] = 131070_{10}$. The third set of inputs corresponds to $X[15 : 0] = 24576_{10}$; $Y[15 : 0] = 4130_{10}$; $C[0] = 0$. The output, $S[15 : 0] = 28706$.

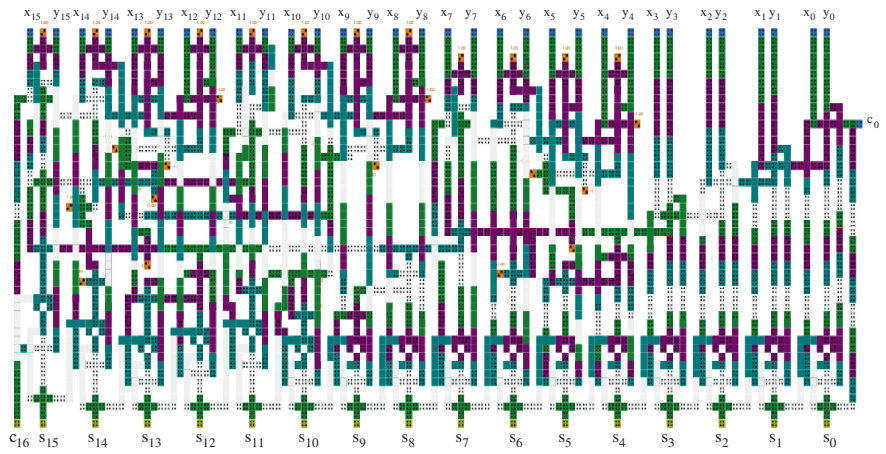


Fig. 5.6 QCADesigner layout for 16-bit hybrid adder

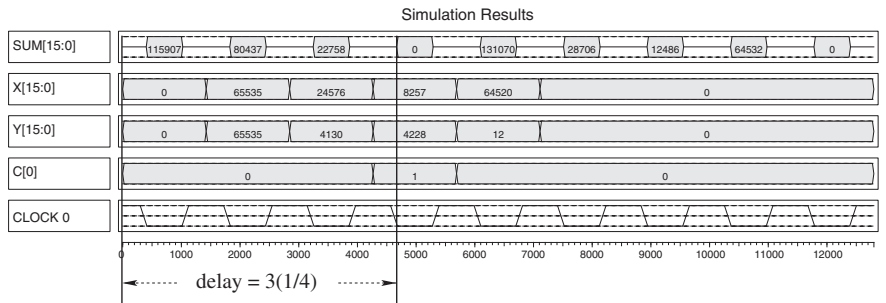


Fig. 5.7 Simulation diagram for 16-bit hybrid adder

5.5 Comparisons

5.5.1 Comparison of Different Adder Styles

Table 5.1 gives the details of cell count, area, delay and area-delay product for various adder styles.

5.5.2 Discussion of Results

The results in the Table 5.1 are for multilayer crossover option for all adders. The proposed hybrid adder has better time complexity (in terms of delay). Further, the area-delay product for a hybrid adder is lower than that for the CFA and other adders

Table 5.1 Complexity of different adder styles

Approach	Cell-count	Area ($\mu\text{m} \times \mu\text{m}$)	Delay (clocks)	Area-delay product
Hybrid4	475	0.80×0.53	1 (3/4)	0.742
Hybrid8	1422	1.34×0.83	2 (1/4)	2.5
Hybrid16	3555	2.61×1.02	3 (1/4)	8.65
Hybrid32	8946	5.55×1.48	5 (2/4)	45.18
Hybrid64	22,427	11.6×2.18	9 (2/4)	240.65
Brent-Kung4	680	0.75×0.86	$1\frac{3}{4}$	1.13
Brent-Kung8	1782	1.40×1.07	$2\frac{2}{4}$	3.74
Brent-Kung16	4350	2.69×1.32	4	14.2
Brent-Kung32	11,825	5.61×1.92	$6\frac{3}{4}$	72.7
Brent-Kung64	30,145	11.6×2.69	$11\frac{2}{4}$	358.84
Kogge-Stone4	815	0.82×0.9	2	1.47
Kogge-Stone8	2938	1.65×1.46	$3\frac{1}{4}$	7.83
Kogge-Stone16	8589	3.42×1.93	6	36
Kogge-Stone32	23,024	7.01×2.62	$10\frac{1}{4}$	188.25
Kogge-Stone64	58,945	15.6×3.28	$17\frac{2}{4}$	895.46
Ladner-Fischer4	698	0.87×0.71	2	1.24
Ladner-Fischer8	1994	1.67×1.06	2 (3/4)	4.86
Ladner-Fischer16	5376	3.29×1.46	4 (1/4)	20.41
Ladner-Fischer32	13,552	6.56×2.02	7 (2/4)	99.38
Ladner-Fischer64	35,850	13.4×2.81	13 (2/4)	506.43
CLA4 ^a	1575	1.74×1.09	3 (2/4)	6.64
CLA8 ^a	3988	3.5×1.58	6 (2/4)	35.9
CLA16 ^a	10,217	7.02×2.21	10 (1/4)	159
CLA32 ^a	25,308	14.06×3.05	19	814.8
CLA64 ^a	59,030	28.2×3.73	31 (2/4)	3313.4
CFA4 ^b	371	0.90×0.45	1 (2/4)	0.61
CFA8 ^b	789	1.79×0.53	2 (2/4)	2.37
CFA16 ^b	1769	3.55×0.69	4 (2/4)	11.23
CFA32 ^b	4305	7.09×1.03	8 (2/4)	62.07
CFA64 ^b	11,681	14.2×1.71	16 (2/4)	399.2

CFA Carry Flow Adder

^a Refers to [4]^b Refers to [5]

especially for large size adders. These can be attributed to the reduction in majority logic as well as the manner in which the most significant carry is produced. In particular, combining prefix and ripple carry styles leads to reduction in majority logic along the critical path (the case of 8-bit hybrid adder is illustrated in Fig. 5.2). Reduction in delay is also depicted for a 16-bit hybrid adder in Fig. 5.7 and the delay

Fig. 5.8 Delay versus Wordlength for various adders (LF denotes Ladner-Fischer; * corresponds to CLA in [4]; \$ corresponds to CFA in [5])

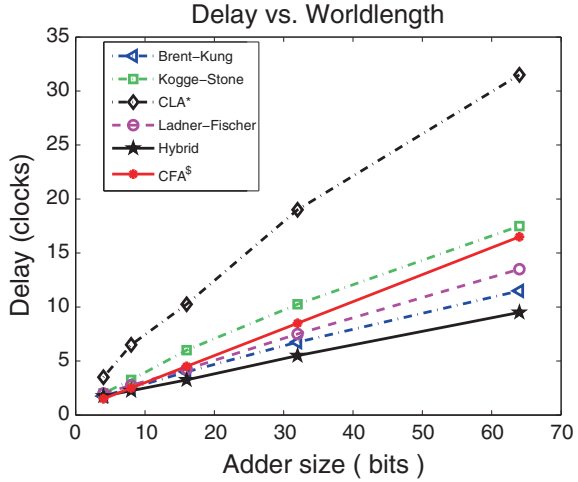
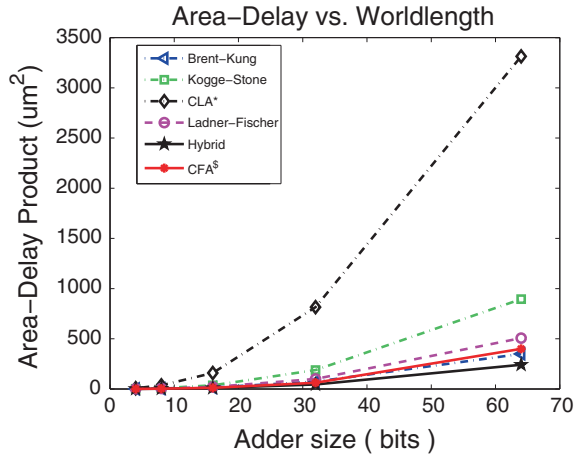


Fig. 5.9 Area-Delay versus Wordlength for various adders (LF corresponds to Ladner-Fischer; * corresponds to CLA in [4]; \$ corresponds to CFA in [5])



calculation is based on clock zones (the delay labelled in Fig. 5.7 indicates the manner in which it is measured).

Graphs comparing various adders are presented in Figs. 5.8 and 5.9. From the graphs, we can infer that the growth rate of delay for a hybrid adder is slower than that of the carry flow adder or other adders reported in the literature. Also, the growth rate of area-delay product is quite small for a hybrid adder.

Besides the advantages in terms of low delay for the hybrid adder, the approach for the development of hybrid adder is expected to have other applications. In particular, the small delay for hybrid adder is advantageous in the design of multiplier units in the QCA paradigm.

5.6 Summary

In this chapter, we have developed a custom adder called hybrid adder for QCA technology. The adder is shown to outperform existing multi-bit adders. In the next chapter, we consider the design of a multiplier in QCA.

References

1. V. Pudi, K. Sridharan, Efficient design of a hybrid adder in quantum-dot cellular automata. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **19**(9), 1535–1548 (2011)
2. P.M. Kogge, H.S. Stone, A parallel algorithm for the efficient solution of a general class of recurrent equations. *IEEE Trans. Comput.* **C-22**, 786–793 (1973)
3. K. Walus, T. Dysart, G. Jullien, R. Budiman, QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **3**(1), 26–29 (2004)
4. H. Cho, E.E. Swartzlander, Adder designs and analyses for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **6**(3), 374–383 (2007)
5. H. Cho, E.E. Swartzlander, Adder and multiplier designs in quantum-dot cellular automata. *IEEE Trans. Comput.* **58**(6), 721–727 (2009)

Chapter 6

Design of a Multiplier in QCA

In the earlier chapters, the focus was on adder designs. In this chapter, we consider QCA design of another common arithmetic unit, namely a multiplier. We also explore the usefulness of majority logic optimization.

6.1 Introduction to Multipliers

Various multiplier designs have appeared in the literature. The multipliers can be classified as either *signed* or *unsigned*. Further, multipliers generally fall in one of three categories: array (parallel), serial and serial-parallel.

Work on QCA multipliers has focussed on serial-parallel multipliers [1, 2]. While there has been some work on parallel unsigned multipliers [3], QCA designs of signed multipliers targeting low latency as well as low area are limited.

6.2 The Baugh-Wooley Multiplier for 2's Complement Numbers

Our interest in this chapter is on a multiplier for signed numbers. In particular, we consider the Baugh-Wooley multiplier that applies to a pair of numbers in 2's complement form. Baugh-Wooley is chosen since it uses only a small number of logical operations in each step of the multiplication operation. Further, the method offers the advantage that it can be implemented merely using a set of full-adders.

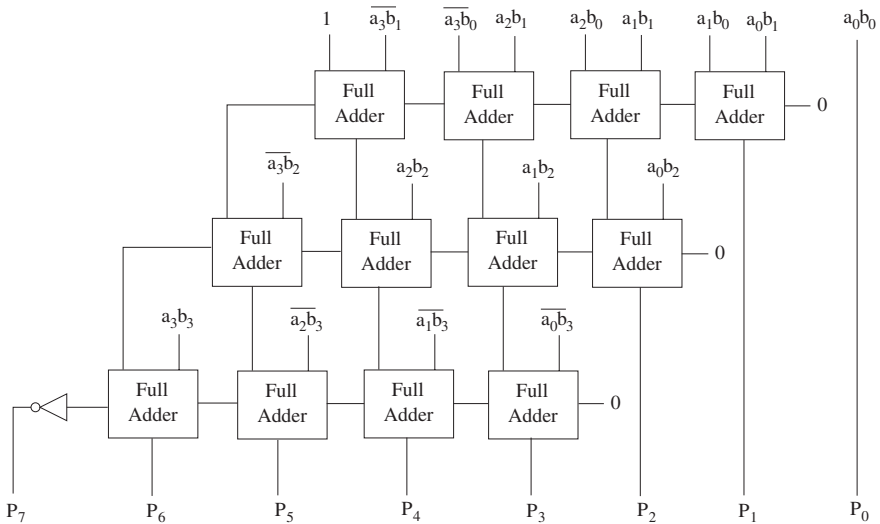
The Baugh-Wooley approach for multiplication of a pair of two's complement numbers A and B (given by $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ respectively) is illustrated in Table 6.1. The product is given by $P_7P_6, \dots, P_0$.

Table 6.1 Baugh-Wooley multiplier illustration

		a_3	a_2	a_1	a_0		
	x	b_3	b_2	b_1	b_0		
		1	$\overline{a_3 b_0}$	$a_2 b_0$	$a_1 b_0$	$a_0 b_0$	
		$\overline{a_3 b_1}$	$a_2 b_1$	$a_1 b_1$	$a_0 b_1$		
		$\overline{a_3 b_2}$	$a_2 b_2$	$a_1 b_2$	$a_0 b_2$		
		$\overline{a_3 b_3}$	$a_2 b_3$	$a_1 b_3$	$a_0 b_3$		
		$\overline{a_3 b_4}$	$a_2 b_4$	$a_1 b_4$	$a_0 b_4$		
		$\overline{a_3 b_5}$	$a_2 b_5$	$a_1 b_5$	$a_0 b_5$		
		$\overline{a_3 b_6}$	$a_2 b_6$	$a_1 b_6$	$a_0 b_6$		
		$\overline{a_3 b_7}$	$a_2 b_7$	$a_1 b_7$	$a_0 b_7$		
		$\overline{a_3 b_8}$	$a_2 b_8$	$a_1 b_8$	$a_0 b_8$		
		$\overline{a_3 b_9}$	$a_2 b_9$	$a_1 b_9$	$a_0 b_9$		
		$\overline{a_3 b_{10}}$	$a_2 b_{10}$	$a_1 b_{10}$	$a_0 b_{10}$		
		$\overline{a_3 b_{11}}$	$a_2 b_{11}$	$a_1 b_{11}$	$a_0 b_{11}$		
		$\overline{a_3 b_{12}}$	$a_2 b_{12}$	$a_1 b_{12}$	$a_0 b_{12}$		
		$\overline{a_3 b_{13}}$	$a_2 b_{13}$	$a_1 b_{13}$	$a_0 b_{13}$		
		$\overline{a_3 b_{14}}$	$a_2 b_{14}$	$a_1 b_{14}$	$a_0 b_{14}$		
		$\overline{a_3 b_{15}}$	$a_2 b_{15}$	$a_1 b_{15}$	$a_0 b_{15}$		
		$\overline{a_3 b_{16}}$	$a_2 b_{16}$	$a_1 b_{16}$	$a_0 b_{16}$		
		$\overline{a_3 b_{17}}$	$a_2 b_{17}$	$a_1 b_{17}$	$a_0 b_{17}$		
		$\overline{a_3 b_{18}}$	$a_2 b_{18}$	$a_1 b_{18}$	$a_0 b_{18}$		
		$\overline{a_3 b_{19}}$	$a_2 b_{19}$	$a_1 b_{19}$	$a_0 b_{19}$		
		$\overline{a_3 b_{20}}$	$a_2 b_{20}$	$a_1 b_{20}$	$a_0 b_{20}$		
		$\overline{a_3 b_{21}}$	$a_2 b_{21}$	$a_1 b_{21}$	$a_0 b_{21}$		
		$\overline{a_3 b_{22}}$	$a_2 b_{22}$	$a_1 b_{22}$	$a_0 b_{22}$		
		$\overline{a_3 b_{23}}$	$a_2 b_{23}$	$a_1 b_{23}$	$a_0 b_{23}$		
		$\overline{a_3 b_{24}}$	$a_2 b_{24}$	$a_1 b_{24}$	$a_0 b_{24}$		
		$\overline{a_3 b_{25}}$	$a_2 b_{25}$	$a_1 b_{25}$	$a_0 b_{25}$		
		$\overline{a_3 b_{26}}$	$a_2 b_{26}$	$a_1 b_{26}$	$a_0 b_{26}$		
		$\overline{a_3 b_{27}}$	$a_2 b_{27}$	$a_1 b_{27}$	$a_0 b_{27}$		
		$\overline{a_3 b_{28}}$	$a_2 b_{28}$	$a_1 b_{28}$	$a_0 b_{28}$		
		$\overline{a_3 b_{29}}$	$a_2 b_{29}$	$a_1 b_{29}$	$a_0 b_{29}$		
		$\overline{a_3 b_{30}}$	$a_2 b_{30}$	$a_1 b_{30}$	$a_0 b_{30}$		
		$\overline{a_3 b_{31}}$	$a_2 b_{31}$	$a_1 b_{31}$	$a_0 b_{31}$		
		$\overline{a_3 b_{32}}$	$a_2 b_{32}$	$a_1 b_{32}$	$a_0 b_{32}$		
		$\overline{a_3 b_{33}}$	$a_2 b_{33}$	$a_1 b_{33}$	$a_0 b_{33}$		
		$\overline{a_3 b_{34}}$	$a_2 b_{34}$	$a_1 b_{34}$	$a_0 b_{34}$		
		$\overline{a_3 b_{35}}$	$a_2 b_{35}$	$a_1 b_{35}$	$a_0 b_{35}$		
		$\overline{a_3 b_{36}}$	$a_2 b_{36}$	$a_1 b_{36}$	$a_0 b_{36}$		
		$\overline{a_3 b_{37}}$	$a_2 b_{37}$	$a_1 b_{37}$	$a_0 b_{37}$		
		$\overline{a_3 b_{38}}$	$a_2 b_{38}$	$a_1 b_{38}$	$a_0 b_{38}$		
		$\overline{a_3 b_{39}}$	$a_2 b_{39}$	$a_1 b_{39}$	$a_0 b_{39}$		
		$\overline{a_3 b_{40}}$	$a_2 b_{40}$	$a_1 b_{40}$	$a_0 b_{40}$		
		$\overline{a_3 b_{41}}$	$a_2 b_{41}$	$a_1 b_{41}$	$a_0 b_{41}$		
		$\overline{a_3 b_{42}}$	$a_2 b_{42}$	$a_1 b_{42}$	$a_0 b_{42}$		
		$\overline{a_3 b_{43}}$	$a_2 b_{43}$	$a_1 b_{43}$	$a_0 b_{43}$		
		$\overline{a_3 b_{44}}$	$a_2 b_{44}$	$a_1 b_{44}$	$a_0 b_{44}$		
		$\overline{a_3 b_{45}}$	$a_2 b_{45}$	$a_1 b_{45}$	$a_0 b_{45}$		
		$\overline{a_3 b_{46}}$	$a_2 b_{46}$	$a_1 b_{46}$	$a_0 b_{46}$		
		$\overline{a_3 b_{47}}$	$a_2 b_{47}$	$a_1 b_{47}$	$a_0 b_{47}$		
		$\overline{a_3 b_{48}}$	$a_2 b_{48}$	$a_1 b_{48}$	$a_0 b_{48}$		
		$\overline{a_3 b_{49}}$	$a_2 b_{49}$	$a_1 b_{49}$	$a_0 b_{49}$		
		$\overline{a_3 b_{50}}$	$a_2 b_{50}$	$a_1 b_{50}$	$a_0 b_{50}$		
		$\overline{a_3 b_{51}}$	$a_2 b_{51}$	$a_1 b_{51}$	$a_0 b_{51}$		
		$\overline{a_3 b_{52}}$	$a_2 b_{52}$	$a_1 b_{52}$	$a_0 b_{52}$		
		$\overline{a_3 b_{53}}$	$a_2 b_{53}$	$a_1 b_{53}$	$a_0 b_{53}$		
		$\overline{a_3 b_{54}}$	$a_2 b_{54}$	$a_1 b_{54}$	$a_0 b_{54}$		
		$\overline{a_3 b_{55}}$	$a_2 b_{55}$	$a_1 b_{55}$	$a_0 b_{55}$		
		$\overline{a_3 b_{56}}$	$a_2 b_{56}$	$a_1 b_{56}$	$a_0 b_{56}$		
		$\overline{a_3 b_{57}}$	$a_2 b_{57}$	$a_1 b_{57}$	$a_0 b_{57}$		
		$\overline{a_3 b_{58}}$	$a_2 b_{58}$	$a_1 b_{58}$	$a_0 b_{58}$		
		$\overline{a_3 b_{59}}$	$a_2 b_{59}$	$a_1 b_{59}$	$a_0 b_{59}$		
		$\overline{a_3 b_{60}}$	$a_2 b_{60}$	$a_1 b_{60}$	$a_0 b_{60}$		
		$\overline{a_3 b_{61}}$	$a_2 b_{61}$	$a_1 b_{61}$	$a_0 b_{61}$		
		$\overline{a_3 b_{62}}$	$a_2 b_{62}$	$a_1 b_{62}$	$a_0 b_{62}$		
		$\overline{a_3 b_{63}}$	$a_2 b_{63}$	$a_1 b_{63}$	$a_0 b_{63}$		
		$\overline{a_3 b_{64}}$	$a_2 b_{64}$	$a_1 b_{64}$	$a_0 b_{64}$		
		$\overline{a_3 b_{65}}$	$a_2 b_{65}$	$a_1 b_{65}$	$a_0 b_{65}$		
		$\overline{a_3 b_{66}}$	$a_2 b_{66}$	$a_1 b_{66}$	$a_0 b_{66}$		
		$\overline{a_3 b_{67}}$	$a_2 b_{67}$	$a_1 b_{67}$	$a_0 b_{67}$		
		$\overline{a_3 b_{68}}$	$a_2 b_{68}$	$a_1 b_{68}$	$a_0 b_{68}$		
		$\overline{a_3 b_{69}}$	$a_2 b_{69}$	$a_1 b_{69}$	$a_0 b_{69}$		
		$\overline{a_3 b_{70}}$	$a_2 b_{70}$	$a_1 b_{70}$	$a_0 b_{70}$		
		$\overline{a_3 b_{71}}$	$a_2 b_{71}$	$a_1 b_{71}$	$a_0 b_{71}$		
		$\overline{a_3 b_{72}}$	$a_2 b_{72}$	$a_1 b_{72}$	$a_0 b_{72}$		
		$\overline{a_3 b_{73}}$	$a_2 b_{73}$	$a_1 b_{73}$	$a_0 b_{73}$		
		$\overline{a_3 b_{74}}$	$a_2 b_{74}$	$a_1 b_{74}$	$a_0 b_{74}$		
		$\overline{a_3 b_{75}}$	$a_2 b_{75}$	$a_1 b_{75}$	$a_0 b_{75}$		
		$\overline{a_3 b_{76}}$	$a_2 b_{76}$	$a_1 b_{76}$	$a_0 b_{76}$		
		$\overline{a_3 b_{77}}$	$a_2 b_{77}$	$a_1 b_{77}$	$a_0 b_{77}$		
		$\overline{a_3 b_{78}}$	$a_2 b_{78}$	$a_1 b_{78}$	$a_0 b_{78}$		
		$\overline{a_3 b_{79}}$	$a_2 b_{79}$	$a_1 b_{79}$	$a_0 b_{79}$		
		$\overline{a_3 b_{80}}$	$a_2 b_{80}$	$a_1 b_{80}$	$a_0 b_{80}$		
		$\overline{a_3 b_{81}}$	$a_2 b_{81}$	$a_1 b_{81}$	$a_0 b_{81}$		
		$\overline{a_3 b_{82}}$	$a_2 b_{82}$	$a_1 b_{82}$	$a_0 b_{82}$		
		$\overline{a_3 b_{83}}$	$a_2 b_{83}$	$a_1 b_{83}$	$a_0 b_{83}$		
		$\overline{a_3 b_{84}}$	$a_2 b_{84}$	$a_1 b_{84}$	$a_0 b_{84}$		
		$\overline{a_3 b_{85}}$	$a_2 b_{85}$	$a_1 b_{85}$	$a_0 b_{85}$		
		$\overline{a_3 b_{86}}$	$a_2 b_{86}$	$a_1 b_{86}$	$a_0 b_{86}$		
		$\overline{a_3 b_{87}}$	$a_2 b_{87}$	$a_1 b_{87}$	$a_0 b_{87}$		
		$\overline{a_3 b_{88}}$	$a_2 b_{88}$	$a_1 b_{88}$	$a_0 b_{88}$		
		$\overline{a_3 b_{89}}$	$a_2 b_{89}$	$a_1 b_{89}$	$a_0 b_{89}$		
		$\overline{a_3 b_{90}}$	$a_2 b_{90}$	$a_1 b_{90}$	$a_0 b_{90}$		
		$\overline{a_3 b_{91}}$	$a_2 b_{91}$	$a_1 b_{91}$	$a_0 b_{91}$		
		$\overline{a_3 b_{92}}$	$a_2 b_{92}$	$a_1 b_{92}$	$a_0 b_{92}$		
		$\overline{a_3 b_{93}}$	$a_2 b_{93}$	$a_1 b_{93}$	$a_0 b_{93}$		
		$\overline{a_3 b_{94}}$	$a_2 b_{94}$	$a_1 b_{94}$	$a_0 b_{94}$		
		$\overline{a_3 b_{95}}$	$a_2 b_{95}$	$a_1 b_{95}$	$a_0 b_{95}$		
		$\overline{a_3 b_{96}}$	$a_2 b_{96}$	$a_1 b_{96}$	$a_0 b_{96}$		
		$\overline{a_3 b_{97}}$	$a_2 b_{97}$	$a_1 b_{97}$	$a_0 b_{97}$		
		$\overline{a_3 b_{98}}$	$a_2 b_{98}$	$a_1 b_{98}$	$a_0 b_{98}$		
		$\overline{a_3 b_{99}}$	$a_2 b_{99}$	$a_1 b_{99}$	$a_0 b_{99}$		
		$\overline{a_3 b_{100}}$	$a_2 b_{100}$	$a_1 b_{100}$	$a_0 b_{100}$		
		$\overline{a_3 b_{101}}$	$a_2 b_{101}$	$a_1 b_{101}$	$a_0 b_{101}$		
		$\overline{a_3 b_{102}}$	$a_2 b_{102}$	$a_1 b_{102}$	$a_0 b_{102}$		
		$\overline{a_3 b_{103}}$	$a_2 b_{103}$	$a_1 b_{103}$	$a_0 b_{103}$		
		$\overline{a_3 b_{104}}$	$a_2 b_{104}$	$a_1 b_{104}$	$a_0 b_{104}$		
		$\overline{a_3 b_{105}}$	$a_2 b_{105}$	$a_1 b_{105}$	$a_0 b_{105}$		
		$\overline{a_3 b_{106}}$	$a_2 b_{106}$	$a_1 b_{106}$	$a_0 b_{106}$		
		$\overline{a_3 b_{107}}$	$a_2 b_{107}$	$a_1 b_{107}$	$a_0 b_{107}$		
		$\overline{a_3 b_{108}}$	$a_2 b_{108}$	$a_1 b_{108}$	$a_0 b_{108}$		
		$\overline{a_3 b_{109}}$	$a_2 b_{109}$	$a_1 b_{109}$	$a_0 b_{109}$		
		$\overline{a_3 b_{110}}$	$a_2 b_{110}$	$a_1 b_{110}$	$a_0 b_{110}$		
		$\overline{a_3 b_{111}}$	$a_2 b_{111}$	$a_1 b_{111}$	$a_0 b_{111}$		
		$\overline{a_3 b_{112}}$	$a_2 b_{112}$	$a_1 b_{112}$	$a_0 b_{112}$		
		$\overline{a_3 b_{113}}$	$a_2 b_{113}$	$a_1 b_{113}$	$a_0 b_{113}$		
		$\overline{a_3 b_{114}}$	$a_2 b_{114}$	$a_1 b_{114}$	$a_0 b_{114}$		
		$\overline{a_3 b_{115}}$	$a_2 b_{115}$	$a_1 b_{115}$	$a_0 b_{115}$		
		$\overline{a_3 b_{116}}$	$a_2 b_{116}$	$a_1 b_{116}$	$a_0 b_{116}$		
		$\overline{a_3 b_{117}}$	$a_2 b_{117}$	$a_1 b_{117}$	$a_0 b_{117}$		
		$\overline{a_3 b_{118}}$	$a_2 b_{118}$	$a_1 b_{118}$	$a_0 b_{118}$		
		$\overline{a_3 b_{119}}$	$a_2 b_{119}$	$a_1 b_{119}$	$a_0 b_{119}$		
		$\overline{a_3 b_{120}}$	$a_2 b_{120}$	$a_1 b_{120}$	$a_0 b_{120}$		
		$\overline{a_3 b_{121}}$	$a_2 b_{121}$	$a_1 b_{121}$	$a_0 b_{121}$		
		$\overline{a_3 b_{122}}$	$a_2 b_{122}$	$a_1 b_{122}$	$a_0 b_{122}$		
		$\overline{a_3 b_{123}}$	$a_2 b_{123}$	$a_1 b_{123}$	$a_0 b_{123}$		
		$\overline{a_3 b_{124}}$	$a_2 b_{124}$	$a_1 b_{124}$	$a_0 b_{124}$		
		$\overline{a_3 b_{125}}$	$a_2 b_{125}$	$a_1 b_{125}$	$a_0 b_{125}$		
		$\overline{a_3 b_{126}}$	$a_2 b_{126}$	$a_1 b_{126}$	$a_0 b_{126}$		
		$\overline{a_3 b_{127}}$	$a_2 b_{127}$	$a_1 b_{127}$	$a_0 b_{127}$		

6.3 Design of Baugh-Wooley Multiplier in QCA

A direct realization of Baugh-Wooley multiplier illustrated in Table 6.1 via parallel QCA adders is not area or delay-efficient. We present a delay and area-efficient realization in Fig. 6.1. It can be observed that the circuit consists primarily of several full-adders so a good full-adder solution in QCA directly contributes to the efficiency of the Baugh-Wooley multiplier. A full-adder can be realized in QCA with three majority gates and one inverter as discussed in Chap. 4. In particular, given inputs a_i , b_i and c_i , the sum can be expressed as given by Eq. (6.1). We know that the carry

**Fig. 6.1** 4-bit Baugh-Wooley multiplier circuit

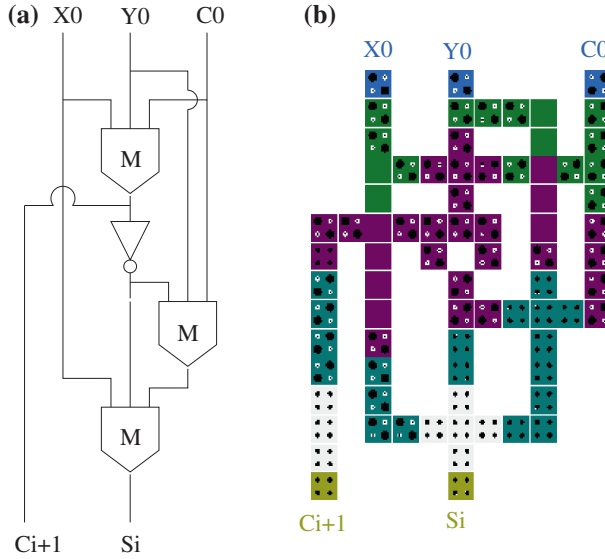


Fig. 6.2 Full Adder. **a** Circuit and **b** QCADesigner layout

realization requires one majority gate and no inverters. Some of the results presented here have been reported in [4].

$$s_i = M(\overline{M(a_i, b_i, c_i)}, M(\overline{M(a_i, b_i, c_i)}, b_i, c_i), a_i) \quad (6.1)$$

As for other circuits, QCADesigner [5] has been used for layout design and simulations. A one-bit full adder circuit (using three majority gates and one inverter) as well as the QCADesigner layout are shown in Fig. 6.2. The QCADesigner layout for a 4-bit Baugh-Wooley multiplier is shown in Fig. 6.3. Figure 6.4 shows the simulation results of a 4×4 Baugh-Wooley multiplier. The first set of inputs for simulation shown in Fig. 6.4 corresponds to $A[3 : 0] = 0$; $B[3 : 0] = 0$. The output, product $P[7 : 0] = 0$ appears after delay of 4.75 clocks or 19 clock zones. The second set of inputs corresponds to $A[3 : 0] = 7$; $B[3 : 0] = 6$. The output $P[7 : 0] = 42$. The third set of inputs corresponds to $A[3 : 0] = -7$; $B[3 : 0] = -7$ and the output $P[7 : 0] = 49$.

Comparisons of multiplier designs with respect to cell count, area, delay and throughput are presented in Table 6.2. Our results show low cell count and low area in general and this can be attributed to a ripple-carry style design with optimization of majority gates and inverters for the basic element (full-adder). Further, throughput of our design is also good.

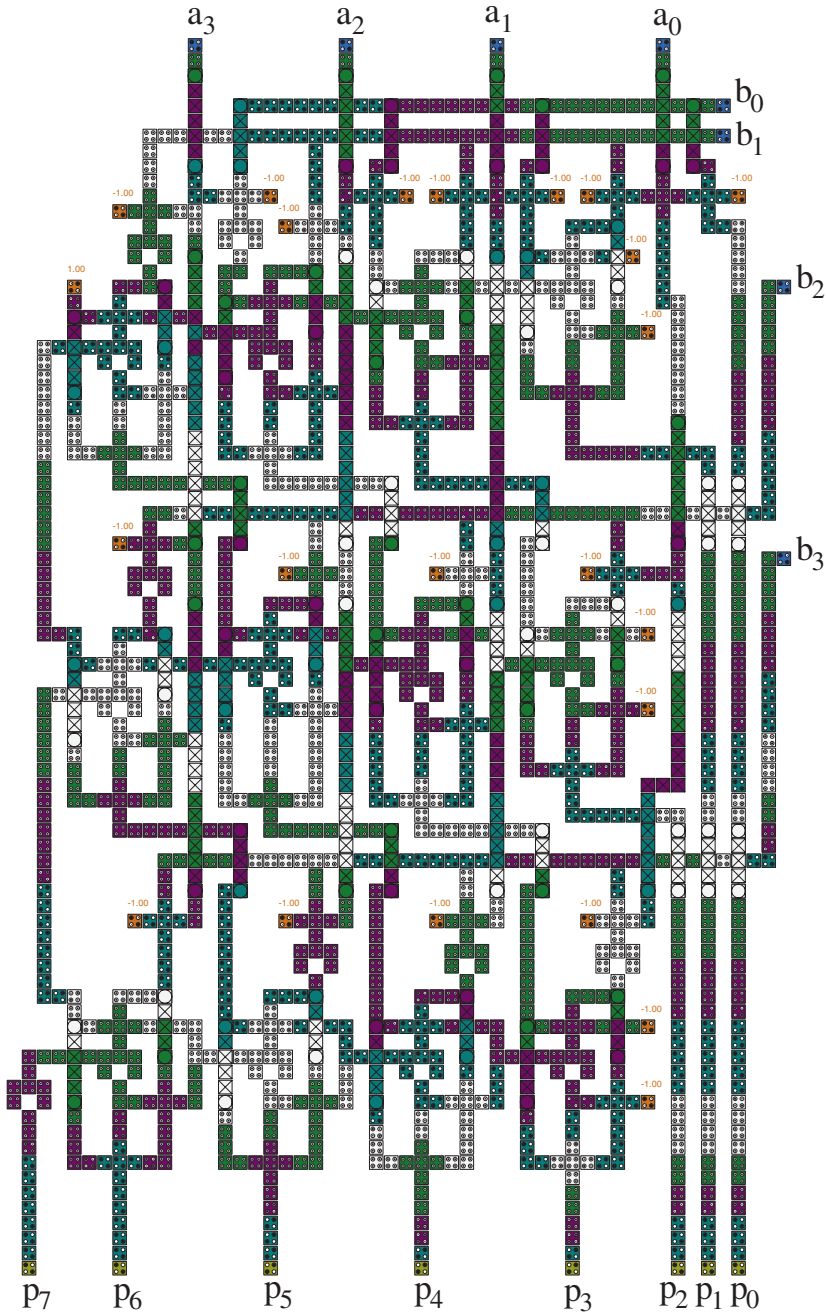


Fig. 6.3 QCADesigner layout for 4-bit Baugh-Wooley multiplier

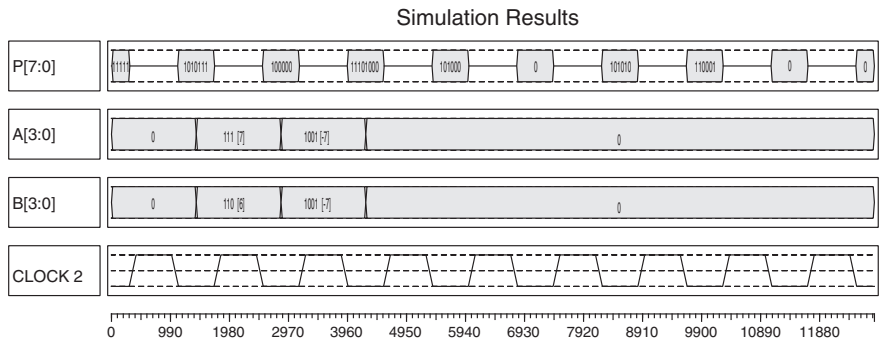


Fig. 6.4 Simulation waveforms for 4×4 Baugh-Wooley multiplier

Table 6.2 Comparison of different QCA multiplier designs

Approach	Cell count	Area (μm^2)	Latency (clocks)	Throughput
4×4 Wallace ^a	3295	7.39	10	1
8×8 Wallace ^a	26,499	82.18	36	1
4×4 Dadda ^a	3384	7.51	12	1
8×8 Dadda ^a	26,973	82.19	38	1
Serial-Parallel (4) ^b	406	0.4935	1	1/8
Serial-Parallel (8) ^b	903	1	1	1/16
Serial-Parallel (4) ^c	–	0.1664	14	1/8
Serial-Parallel (8) ^c	–	0.6656	26	1/16
Prop. Baugh-Wooley 4×4	1982	1.8	4.75	1
Prop. Baugh-Wooley 8×8	10,475	10.2	10.25	1

^aRefers to [3]

^bRefers to [2]

^cRefers to [1]

6.4 Summary

In this chapter, we have discussed briefly computer algorithms for multiplication. We have then considered for QCA design a well-known multiplier for 2’s complement numbers, namely the Baugh-Wooley multiplier. An efficient design based on one-bit full-adders is presented. Comparisons with prior multiplier designs are also given. In the next chapter, we consider the computation of a discrete orthogonal transform in QCA.

References

1. K. Walus, G. Jullien, V. Dimitrov, Computer arithmetic structures for quantum cellular automata, in *Record of Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, pp. 1435–1439 (2003)
2. H. Cho, E.E. Swartzlander, Adder and multiplier designs in quantum-dot cellular automata. *IEEE Trans. Comput.* **58**(6), 721–727 (2009)
3. S.W. Kim, E. Swartzlander, Parallel multipliers for quantum-dot cellular automata, in *Proceedings of IEEE Nanotechnology Materials and Devices Conference*, pp. 68–72 (2009)
4. V. Pudi, K. Sridharan, Efficient design of Baugh-Wooley multiplier in quantum-dot cellular automata, in *Proceedings of Thirteenth IEEE Conference on Nanotechnology (IEEE-NANO 2013)*, pp. 702–706 (2013)
5. K. Walus, T. Dysart, G. Jullien, R. Budiman, QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **3**(1), 26–29 (2004)

Chapter 7

Discrete Hadamard Transform

Computation in QCA

In the previous chapters, we have examined QCA-based designs of multi-bit adders and a multiplier. Adders and multipliers constitute basic arithmetic units and hence designs for these (along with ones for memory elements) take us closer towards QCA-based microprocessors. We now ask if we can consider higher-level computational tasks as such for QCA-based design. Investigations in this direction would lead us towards special-purpose processor design. Further, it would also initiate a line of enquiry on choice of algorithms for various tasks/problems. In other words, we attempt to answer the following question: *Given a task, what is the most appropriate strategy for high-performance QCA realization?*

A frequent computational task is matrix multiplication [1, 2]. A related task is the computation of various discrete orthogonal transforms. In this chapter, we consider the computation of one such transform, namely the Discrete Hadamard Transform (DHT) [3]. The DHT is widely used in audio and image processing [3, 4]. The 4-point DHT finds applications in video coding [5] while higher order DHTs are used in orthogonal frequency-division multiplexing systems [6]. The DHT has a kernel matrix consisting of only 1's and -1's hence computation of DHT can be accomplished with merely addition and subtraction modules.

Since the DHT matrix has a special structure, it lends itself to the development of efficient VLSI architectures. However, prior approaches for the DHT targeted for CMOS implementation are not necessarily appropriate for QCA. In QCA, the length of the *interconnect* influences the overall delay. Further, solutions that have a simple structure with a small number of crossovers are desirable in QCA.

We will observe in this chapter that design of QCA-based solutions for addition-related high level tasks (of direct interest in signal processing) cannot always borrow from efficient multi-bit adder designs. However, QCA designs for one-bit addition and subtraction come in handy. In this context, our earlier discussions in Chaps. 3 and 4 are relevant.

An efficient DHT solution in QCA based on a bit-serial pipelined architecture that uses one-bit adder-subtractor blocks is presented in this chapter. We also derive bounds on the number of majority gates for realization of 4-point and 16-point

DHT. This gives an idea of the complexity of QCA-based design. We also discuss simulations of the designs in QCADesigner and gather information on cell count, area and delay. Since there do not appear to be any prior QCA-based designs for computation of the DHT, we present comparisons with a direct (parallel) solution based on multi-bit addition. Some of the results presented here have been reported in [7].

7.1 Basics of Discrete Hadamard Transform

The DHT is defined using Hadamard matrices. A Hadamard matrix H_N , of size $N \times N$, is defined recursively as $\mathbf{H}_N = \mathbf{H}_{N/2} \otimes \mathbf{H}_2 = \mathbf{H}_2 \otimes \mathbf{H}_{N/2}$ where the symbol $\otimes$ denotes the Kronecker product, and

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (7.1)$$

The Hadamard matrix $\mathbf{H}_N$ is given by

$$\mathbf{H}_N = \begin{bmatrix} \mathbf{H}_{N/2} & \mathbf{H}_{N/2} \\ \mathbf{H}_{N/2} & -\mathbf{H}_{N/2} \end{bmatrix}. \quad (7.2)$$

Given an input vector X of size N , the discrete Hadamard transformed vector is given by $Y = H_N X$.

7.2 Mathematical Formulation for DHT Computation

We present a QCA design that is based on a bit-serial pipelined architecture for DHT. The bit-serial approach leads to low interconnect requirements and facilitates an efficient solution in QCA. To arrive at the bit-serial architecture, we consider first the 4-point DHT (i.e., $N = 4$). From the terminology in Sect. 7.1, we can express the 4-point DHT as

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} \quad (7.3)$$

Let Y_{out} represent a generic (Hadamard-transformed) output corresponding to four inputs X_0, X_1, X_2, X_3 . Further, Y_0 is a special case with all X_i s added. Then, Y_{out} and Y_0 can be expressed as given in Eqs. (7.4) and (7.5).

$$Y_{out} = X_0 \pm X_1 \pm X_2 \pm X_3 \quad (7.4)$$

$$Y_0 = X_0 + X_2 + X_1 + X_3 \quad (7.5)$$

Considering the least significant bit of output Y_0 , namely $Y_{0,0}$, we have the expression given by Eq. (7.6). It is worth noting that the summation on the right hand side in Eq. (7.6) leads to a *Carry* bit (C_f) in addition to $Y_{0,0}$. The output C_f in Eq. (7.6) can be used as the corresponding carry input for computation of $Y_{0,1}$. The $C_{0,1}$ s and $A_{0,0}$ s in Eq. (7.6) correspond to the two adjacent bits resulting from the summation of $x_{0,0}$ and $x_{2,0}$.

$$\begin{aligned} Y_{0,0} + C_f &= (x_{0,0} + x_{2,0}) + (x_{1,0} + x_{3,0}) \\ &= (2C_{0,1} + A_{0,0}) + (2C_{1,1} + A_{1,0}) \\ &= (A_{0,0} + A_{1,0}) + (2C_{0,1} + 2C_{1,1}) \\ &= Y_{0,0} + (2C_{0,1} + 2C_{1,1} + 2C_{2,1}) \end{aligned} \quad (7.6)$$

The description in bit-level format for the computation of the output is shown in Eqs. (7.7)–(7.10). Let $x_{0,i}$ and $x_{2,i}$ denote, respectively, the i th bits of X_0 and X_2 and let $A_{0,i}$ and $S_{0,i}$ denote respectively, the sum and difference of these bits. $A_{1,i}$ and $S_{1,i}$ are defined analogously. Then, we have

$$\begin{aligned} A_{0,i} &= x_{0,i} + x_{2,i} \\ A_{1,i} &= x_{1,i} + x_{3,i} \\ S_{0,i} &= x_{0,i} - x_{2,i} \\ S_{1,i} &= x_{1,i} - x_{3,i} \end{aligned}$$

$$\begin{aligned} Y_0 &= -A_{0,W-1}2^{W-1} + \sum_{i=0}^{W-2} A_{0,i}2^i \\ &\quad + \left[-A_{1,W-1}2^{W-1} + \sum_{i=0}^{W-2} A_{1,i}2^i \right] \end{aligned} \quad (7.7)$$

$$\begin{aligned} Y_1 &= -A_{0,W-1}2^{W-1} + \sum_{i=0}^{W-2} A_{0,i}2^i \\ &\quad - \left[-A_{1,W-1}2^{W-1} + \sum_{i=0}^{W-2} A_{1,i}2^i \right] \end{aligned} \quad (7.8)$$

$$\begin{aligned} Y_2 &= -S_{0,W-1}2^{W-1} + \sum_{i=0}^{W-2} S_{0,i}2^i \\ &\quad + \left[-S_{1,W-1}2^{W-1} + \sum_{i=0}^{W-2} S_{1,i}2^i \right] \end{aligned} \quad (7.9)$$

$$\begin{aligned}
Y_3 = & -S_{0,W-1}2^{W-1} + \sum_{i=0}^{W-2} S_{0,i}2^i \\
& - \left[-S_{1,W-1}2^{W-1} + \sum_{i=0}^{W-2} S_{1,i}2^i \right]
\end{aligned} \quad (7.10)$$

7.3 Architectural Aspects

It is of interest to note that Eqs. (7.7)–(7.10) involve common algebraic expressions. These common algebraic expressions lead to a low-complexity architecture that is made up of only 1-bit adder/subtractor units. The architecture for 4-point DHT is shown in Fig. 7.1. Since the design is bit-serial (to facilitate low area QCA realization), in the first clock cycle, one bit of each of the four outputs, namely $Y_{0,0}$, $Y_{1,0}$, $Y_{2,0}$ and $Y_{3,0}$ will be produced. Assuming 8-bit wordlength, 10 clock cycles will be required to produce the entire output (since output of 8-bit addition is 10 bits at most). The block labelled D in Fig. 7.1 is a D-latch for achieving the required delay between successive bit outputs.

The extension of the architecture presented for 4-point DHT to 8-point DHT is shown in Fig. 7.2. The derivations for the output are similar to the one for 4-point DHT. The 8-point DHT architecture uses two 4-point DHT blocks as the first stage and just four one-bit add-sub blocks as the second stage to obtain the transformed output $Y_{j,i}$ where $j = 0, 1, \dots, 7$. Each 4-point HT block in Fig. 7.2 is realized as shown in Fig. 7.1.

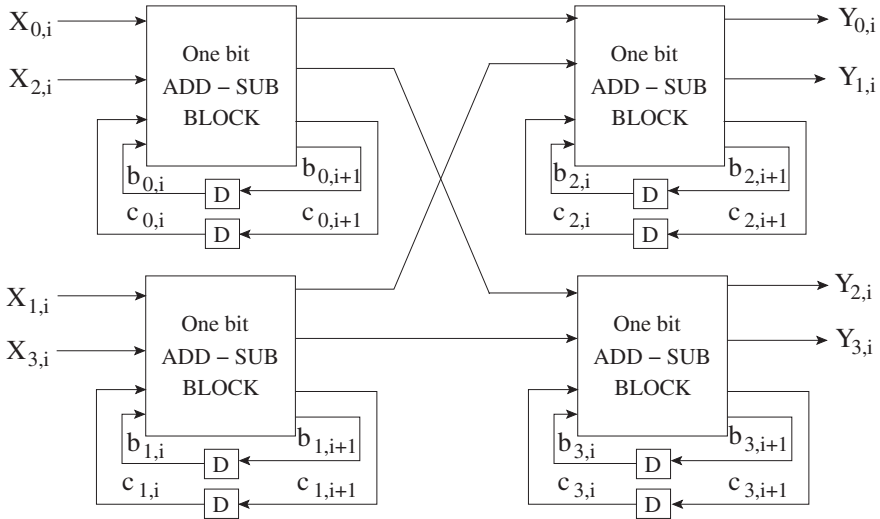


Fig. 7.1 Architecture for bit-serial 4-point DHT

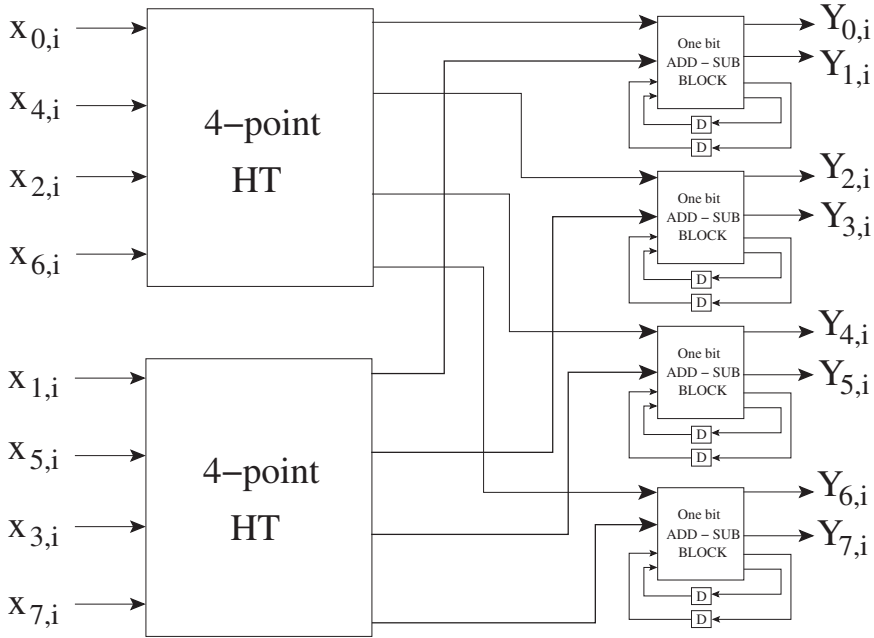


Fig. 7.2 Architecture for bit-serial 8-point Hadamard transform computation

Remark 7.1 The architecture presented for 8-point DHT readily extends to the case of 16-point DHT. In this case, two 8-point DHTs and eight one-bit add-sub blocks are required.

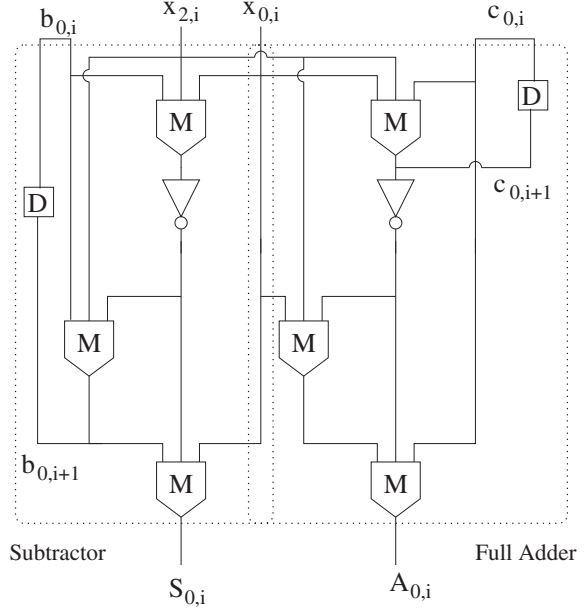
7.4 QCA Realization

The one-bit adder/subtractor units in Fig. 7.1 can be realized efficiently in QCA: the design requires just three majority gates and one inverter for a one-bit full-adder as shown in Fig. 7.3 (and an equal number for a one-bit subtractor). The realization is based on Propositions 7.2 and 7.3. Let $A_{0,i}$ and $c_{0,i+1}$ denote the outputs of a full-adder (as indicated also in Fig. 7.5) while $S_{0,i}$ and $b_{0,i+1}$ denote the outputs of a full-subtractor. We present Proposition 7.2 without proof since the details are available in Chap. 4. We then show that a full-subtractor can also be realized with the same number of majority gates and inverters.

Proposition 7.2 *A full-adder can be realized using only three majority gates and one inverter.*

Proposition 7.3 *A full-subtractor can be realized using only three majority gates and one inverter.*

Fig. 7.3 One-bit adder and subtractor with QCA primitives: majority gate and inverter



Proof

$$b_{0,i+1} = \overline{x_{0,i}}x_{2,i} + \overline{x_{0,i}}b_{0,i} + x_{2,i}b_{0,i} \quad (7.11)$$

$$\begin{aligned} &= x_{0,i}x_{2,i}b_{0,i} + \overline{x_{0,i}}x_{2,i}b_{0,i} + \overline{x_{0,i}}x_{2,i}\overline{b_{0,i}} \\ &\quad + \overline{x_{0,i}}b_{0,i}\overline{x_{2,i}} \\ &= M(\overline{x_{0,i}} \overline{x_{2,i}} + \overline{x_{0,i}} \overline{b_{0,i}} + \overline{x_{2,i}} \overline{b_{0,i}}, x_{2,i}, b_{0,i}) \\ &= M(M(x_{0,i}, x_{2,i}, \overline{b_{0,i}}), x_{2,i}, b_{0,i}) \end{aligned} \quad (7.12)$$

$$S_{0,i} = x_{0,i}x_{2,i}b_{0,i} + \overline{x_{2,i}}x_{0,i}\overline{b_{0,i}} + \overline{x_{0,i}}x_{2,i}\overline{b_{0,i}} + \overline{x_{0,i}}b_{0,i}\overline{x_{2,i}} \quad (7.13)$$

$$\begin{aligned} &= (\overline{x_{0,i}}x_{2,i}\overline{b_{0,i}} + x_{2,i}b_{0,i} + \overline{x_{0,i}}b_{0,i}\overline{x_{2,i}}) \\ &\quad (\overline{x_{0,i}} \overline{x_{2,i}} + \overline{x_{0,i}} \overline{b_{0,i}} + \overline{x_{2,i}} \overline{b_{0,i}}) \\ &\quad + (\overline{x_{0,i}}x_{2,i}\overline{b_{0,i}} + x_{2,i}b_{0,i} + \overline{x_{0,i}}b_{0,i}\overline{x_{2,i}})x_{0,i} \\ &\quad + (\overline{x_{0,i}} \overline{x_{2,i}} + \overline{x_{0,i}} \overline{b_{0,i}} + \overline{x_{2,i}} \overline{b_{0,i}})x_{0,i} \\ &= M(\overline{x_{0,i}}x_{2,i}\overline{b_{0,i}} + x_{2,i}b_{0,i} + \overline{x_{0,i}}b_{0,i}\overline{x_{2,i}}, x_{0,i}, \\ &\quad \overline{x_{0,i}} \overline{x_{2,i}} + \overline{x_{0,i}} \overline{b_{0,i}} + \overline{x_{2,i}} \overline{b_{0,i}}) \\ &= M(\overline{x_{0,i}}x_{2,i}\overline{b_{0,i}} + x_{2,i}b_{0,i} + \overline{x_{0,i}}b_{0,i}\overline{x_{2,i}}, x_{0,i}, \\ &\quad \overline{M(x_{0,i}, x_{2,i}, \overline{b_{0,i}})}) \\ &= \overline{M(M(\overline{x_{0,i}} \overline{x_{2,i}}, \overline{b_{0,i}} \overline{x_{2,i}}, \overline{b_{0,i}} \overline{x_{0,i}}), x_{0,i}, \\ &\quad \overline{M(x_{0,i}, x_{2,i}, \overline{b_{0,i}})})} \end{aligned}$$

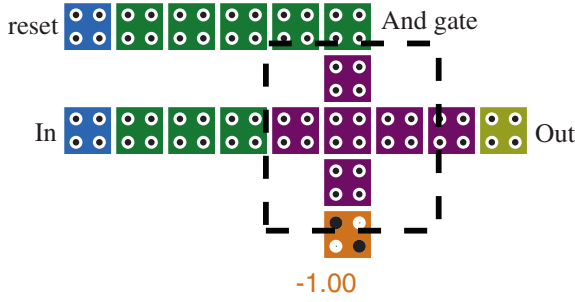


Fig. 7.4 A D-latch with reset in QCA

$$= \frac{M(M(\overline{M(x_{0,i}, x_{2,i}, b_{0,i})}), x_{2,i}, b_{0,i}), x_{0,i}, \overline{M(x_{0,i}, x_{2,i}, b_{0,i})})}{M(x_{0,i}, x_{2,i}, b_{0,i})} \quad (7.14)$$

From Eqs. (7.12) and (7.14), we have the result. Q.E.D

Remark 7.4 Another element in the 4-point DHT architecture (shown in Fig. 7.1) is the delay block. This is realized in QCA as shown in Fig. 7.4.

The complexity of the DHT computation in QCA is expressed by Propositions 7.5 and 7.6. Proposition 7.5 presents bounds on logic required for realization of 4-point DHT. Extension to 16-point DHT is presented in Proposition 7.6.

Proposition 7.5 *At most 32 majority gates are required to realize the 4-point DHT in QCA.*

Proof Each 1-bit adder-subtractor block requires 8 majority gates including the two majority gates for the D-latches. Since 4 adder-subtractor blocks are used for a 4-point DHT, the overall requirement of majority gates is 32. Q.E.D

Proposition 7.6 *At most 256 majority gates are required to realize the 16-point DHT in QCA.*

Proof This is a consequence of the result for 4-point DHT given in Proposition 7.5. Since eight 4-point DHT blocks are used for realizing the 16-point DHT, the overall requirement of majority gates for 16-point DHT is 256. Q.E.D

The majority gate requirement for DHT of various transform lengths is summarized in Table 7.1. The results of simulation as well as cell count, area and delay for the designs are presented in Sect. 7.5.

Table 7.1 Majority gate requirement for N -point DHT

Transform size (N)	Majority gates
4	32
8	96
16	256

7.5 Simulation Results

The proposed designs for DHT have been simulated in QCADesigner [8]. Figure 7.5 gives the layout for the one-bit adder and subtractor while Fig. 7.6 gives the layout for the 4-point DHT. The simulation results of the 4-point DHT layout are shown in Fig. 7.7. The first set of inputs for simulation correspond to $X(0) = 1$, $X(1) = 2$, $X(2) = 3$, $X(3) = 4$. The outputs $Y(0)$, $Y(1)$, $Y(2)$ and $Y(3)$ are shown at the top (the rectangular blocks indicate the binary output with the MSB on the right).

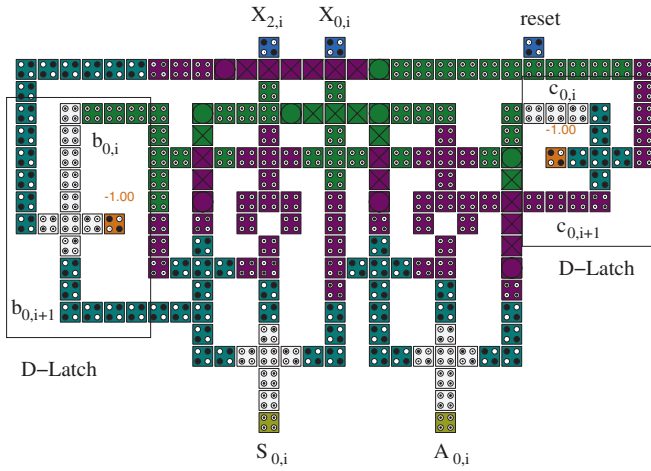


Fig. 7.5 QCA layout for one-bit adder and subtractor

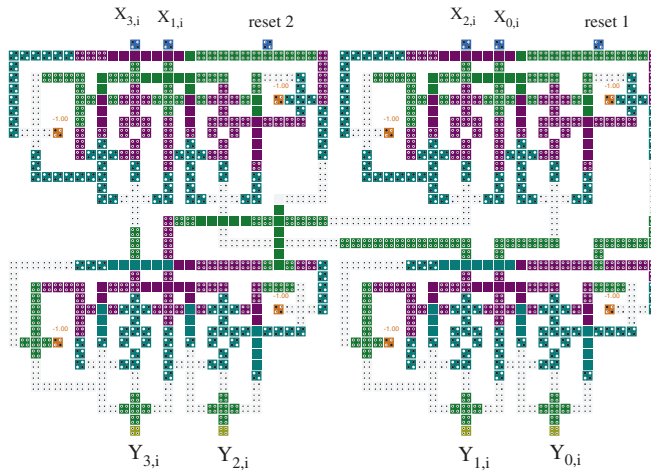


Fig. 7.6 QCA layout of 4-point DHT realized using the proposed bit-serial approach

7.6 Performance of a Full-Parallel Addition Strategy and Comparisons

Since there does not appear to be any other QCA designs for the DHT, we have also implemented a direct method based on multi-bit addition. The direct method can be described as follows. Consider the computation of $Y(k)$ as $H_4 \times X(k)$ where $k = 0, \dots, 3$. Let each $X(k)$ be 4-bit. $Y(0)$ is, for example, obtained as the sum of $X(0)$, $X(1)$, $X(2)$ and $X(3)$. To add $X(0)$ to $X(1)$, we use a multi-bit addition method [9]. This approach has been implemented in QCADesigner [8] for 4-point DHT calculation where each input element ($X(i)$) is 4-bit wide. The layout for 4-point DHT using the direct method is given in Fig. 7.9. We can observe that the structure is quite complex with a large number of crossovers.

To allow comparisons between a full-parallel (addition-based) strategy and the bit-serial approach, we need an appropriate metric. Merely using the area (or even the product of area and delay) for bit-serial as well as the full-parallel approach does not do justice to the latter since area requirements are bound to be more for a full-parallel approach. To overcome this, we include the number of clock cycles taken and define a measure called ADC which is the product of area, delay and number of clock cycles. The last column in Table 7.3 shows the ADC values for the two approaches. From Table 7.3, we can infer that the bit-serial pipelined strategy leads to low ADC and small number of crossovers. In particular, the bit-serial method leads to only about 15 % crossovers in comparison to the direct method. Further, the bit-serial pipelined method leads to approximately 73 % reduction in ADC (in comparison to the direct method).

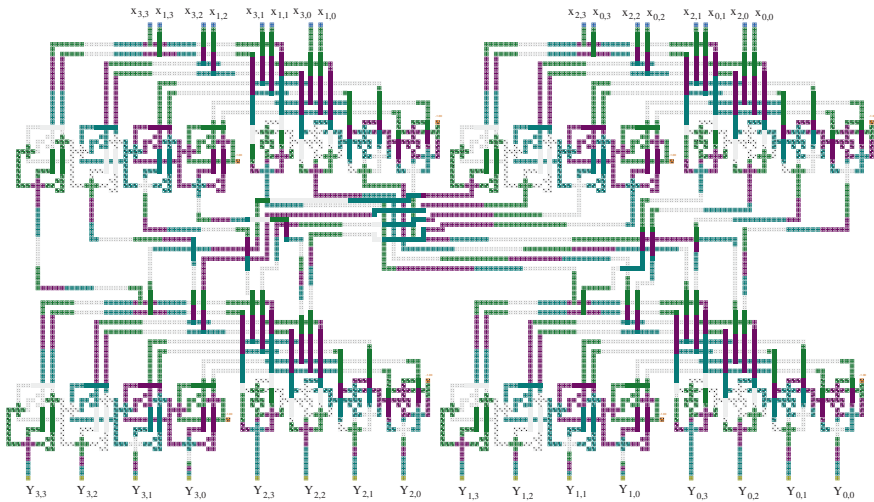


Fig. 7.9 QCA layout of 4-point DHT realized using the direct multi-bit addition approach

Table 7.3 Comparison of bit-serial approach with a direct strategy (multi-bit addition) for 4-point DHT

Method	Cell count	Area (μm^2)	Delay (clk zones)	Crossovers	ADC
Bit-serial	1009	0.91	$2\frac{1}{4}$	22	12.28
Direct (multi-bit addition)	6208	6.68	$6\frac{3}{4}$	142	45.09

7.7 Summary

In this chapter, we have examined the problem of computing DHT. An approach appropriate for QCA has been identified. Complexity of the QCA design for computing DHT for different transform sizes has been discussed. Simulation results show that the proposed approach has low delay as well as small number of crossovers.

References

1. J.D. Wood, D. Tougaw, Matrix multiplication using quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **10**(5), 1036–1042 (2011)
2. L. Lu, W. Liu, M. O'Neill, E.E. Swartzlander, QCA systolic array design. *IEEE Trans. Comput.* **62**(3), 548–560 (2013)
3. K.R. Castleman, *Digital Image Processing* (Prentice-Hall, New Jersey, 1996)
4. K.G. Beauchamp, *Applications of Walsh and Related Functions* (Academic Press, New York, 1984)
5. S.T. Wei, C.W. Tien, B.D. Liu, J.F. Yang, Adaptive truncation algorithm for Hadamard transformed H.264/AVC lossless video coding. *IEEE Trans. Circuits Syst. Video Technol.* **21**(5), 538–549 (2011)
6. J. Xiao, J. Yu, X. Li, Q. Tang, H. Chen, F. Li, Z. Cao, L. Chen, Hadamard transform combined with companding transform technique for PAPR reduction in an optical direct-detection OFDM system. *IEEE/OSA J. Opt. Commun. Netw.* **4**(10), 709–714 (2012)
7. V. Pudi, K. Sridharan, A bit-serial pipelined architecture for high-performance DHT computation in quantum dot cellular automata. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* doi:10.1109/TVLSI.2014.2363519 (to appear)
8. K. Walus, T. Dysart, G. Jullien, R. Budiman, QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **3**(1), 26–29 (2004)
9. V. Pudi, K. Sridharan, Low complexity design of ripple carry and Brent-Kung adders in QCA. *IEEE Trans. Nanotechnol.* **11**(1), 105–119 (2012)

Chapter 8

Study of Thermal Robustness of QCA Designs

One aspect that is important for QCA circuits in specific technologies is thermal robustness. For example, the performance of a metal-dot QCA circuit is subject to thermal fluctuation [1]. In particular, one has to take care of polarization, since in large designs this could be as important as delay or area. In this chapter, we present brief studies and comparisons of adder designs with respect to output polarization. In particular, we consider the hybrid adder which has the best performance among the adders for thermal robustness studies. We also compare with the carry flow adder (which has low cell count and delay [2]). The results presented are for the 4-bit version of the adders.

8.1 Simulation Engine and Settings for the Study

The coherence vector simulation in QCADesigner has been performed with temperature set to various values starting from 1 K. Default settings (*Euler Method* and *Randomize Simulation Order* option) have been chosen for simulation. The clock frequency for the studies is set to 100 GHz.

8.2 Approach Used and the Results

We interpret the waveforms resulting from simulation in QCADesigner using a simple threshold system as suggested in [3]: polarization $< 0.5 \implies$ logic 0; polarization $> 0.5 \implies$ logic 1, otherwise the state is indeterminate.

The change in polarization has been studied for sum bits, $S_i, i = 0, \dots, 3$ and carry bit C_4 for two different inputs. The first set of input values (referred to as input 1 in figure captions) is: $X = 15; Y = 15, C_{in} = 1$. Plots of polarization versus

Fig. 8.1 Polarization versus Temp. for S0 for input 1

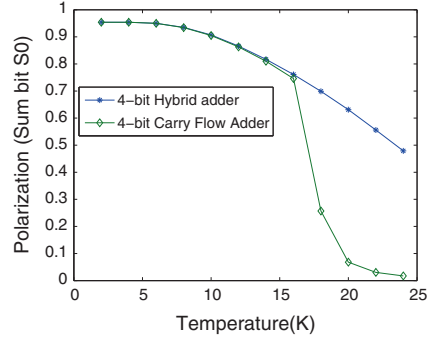


Fig. 8.2 Polarization versus Temp. for S1 for input 1

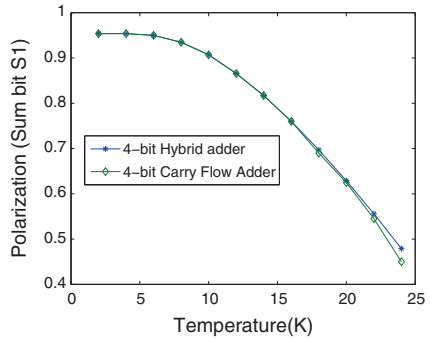
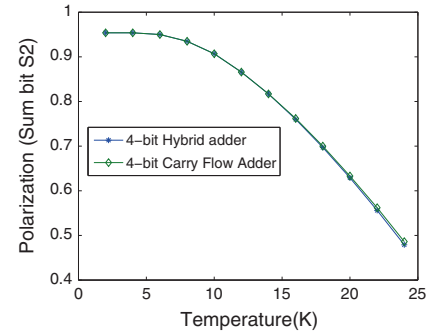


Fig. 8.3 Polarization versus Temp. for S2 for input 1



temperature for S0, S1, S2, S3 and C4 for this input set are shown respectively in Figs. 8.1, 8.2, 8.3, 8.4, 8.5.

The second set of input values (referred to as input 2 in figure captions) is $X = 7$; $Y = 7$; $C_{in} = 0$. Plots of polarization versus temperature for S0, S1, S2, S3 and C4 for this input set are shown respectively in Figs. 8.6, 8.7, 8.8, 8.9, 8.10. Since negative polarization (-1) corresponds to a binary value of 0, the “min” value is taken in the results of simulation in QCADesigner for both C4 and S0 (for the remaining output bits, the “max” value is taken since these bits are 1). From the

Fig. 8.4 Polarization versus Temp. for S3 for input 1

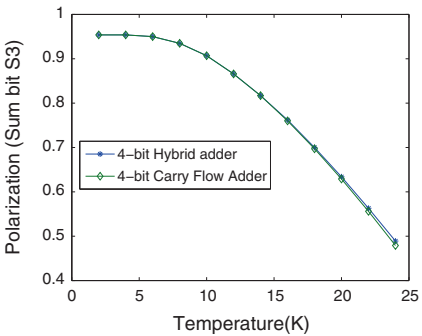


Fig. 8.5 Polarization versus Temp. for C4 for input 1

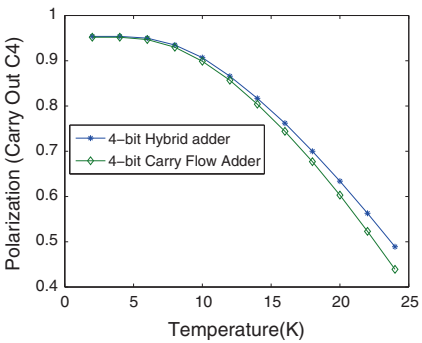


Fig. 8.6 Polarization versus Temp. for S0 for input 2

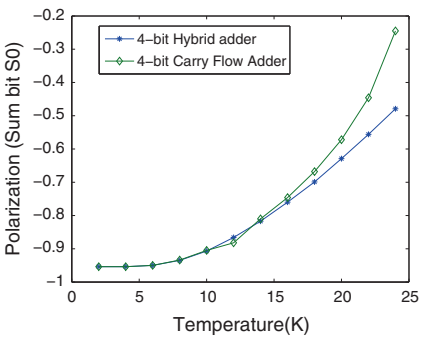


Fig. 8.7 Polarization versus Temp. for S1 for input 2

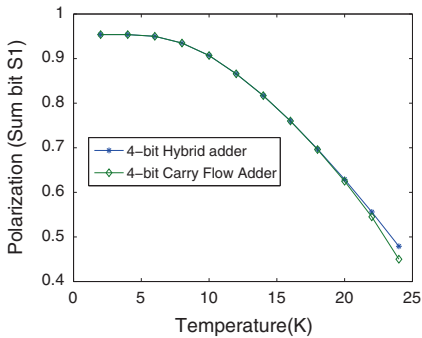


Fig. 8.8 Polarization versus Temp. for S2 for input 2

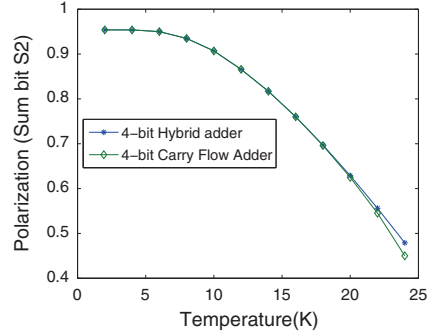


Fig. 8.9 Polarization versus Temp. for S3 for input 2

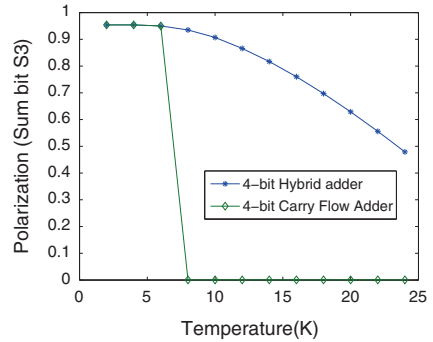
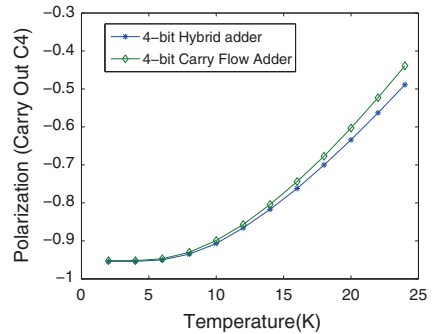


Fig. 8.10 Polarization versus Temp. for C4 for input 2



plots, we can infer that the proposed hybrid adder has better performance than the CFA with respect to output polarization. S0 is in error (it is 0) at higher temperatures (beyond 16 K) in the case of CFA leading to an error in overall sum for inputs given by: $X = 15$, $Y = 15$, $C_{in} = 1$. All the outputs have the correct value up to 23 K for the proposed hybrid adder. For the input given by $X = 7$, $Y = 7$, $C_{in} = 0$, S3 is in error for temperatures exceeding 6 K in the case of CFA while accuracy of output is maintained up to 22 K in the case of the hybrid adder. In general, the hybrid adder is robust up to a larger value of temperature in comparison to the carry flow adder.

8.3 Summary

In this chapter, we have discussed thermal robustness of a high-performance QCA adder namely the hybrid adder. The next chapter summarizes the work presented in this book and explores directions for further research.

References

1. M. Liu, Robustness and power dissipation in quantum-dot cellular automata. PhD thesis, University of Notre Dame, 2006
2. H. Cho, E.E. Swartzlander, Adder and multiplier designs in quantum-dot cellular automata. *IEEE Trans. Comput.* **58**(6), 721–727 (2009)
3. G. Schulhof, K. Walus, G.A. Jullien, Simulation of random cell displacements in QCA. *ACM J. Emerg. Technol. Comput. Syst.* **3**(1), 1–14 (2007)

Chapter 9

The Road Ahead

This research has studied digital design in the context of emerging nanotechnologies. In particular, we have studied the problem of designing arithmetic circuits in Quantum Dot Cellular Automata. We have developed a number of theoretical results on majority logic. The new results on majority logic facilitate logic minimization as well as wire reduction and thereby enable us to get efficient QCA designs.

9.1 Contributions of This Research

The contributions of this book are as follows.

- An introduction to QCA-based design of various logic elements.
- A new one-bit full-adder requiring only three majority gates and one inverter.
- Efficient QCA design of the ripple carry adder and prefix adders.
- A new multi-bit adder for QCA called *hybrid adder*.
- A low-complexity Baugh-Wooley multiplier in QCA.
- A high-performance bit-serial architecture for computation of the Discrete Hadamard Transform (DHT) in QCA.
- Study of thermal robustness of the hybrid adder.

We conclude with a list of possibilities that enhance the value of the majority logic optimization-based approach for digital design.

9.2 Extensions

9.2.1 Applications of the Work to Other Nanotechnologies

Systematic majority logic reduction, in general, has interesting consequences for other nanotechnologies as well. For example, the Single Electron Transistor (SET) consists of tunnel junctions and capacitors as basic units and a (3-input) majority gate can be realized using four tunnel junctions and nine capacitors. Hence, efficient adder designs in SET can be realized by majority logic reduction techniques. The approach is also valuable for designs using resonant tunneling diodes [1].

9.2.2 Magnetic Quantum Dot Cellular Automata

With respect to QCA, we have examined the thermal robustness of the designs for electrostatic realizations. Another implementation of QCA is based on interaction between magnetic nanoparticles. Magnetic QCA has been realized at room temperature [2]. The majority logic gate for magnetic quantum dot cellular automata has been explored in [3]. Our results on majority logic in this book will be valuable for magnetic QCA. It is also of interest to explore realization of adders in magnetic QCA.

9.2.3 Nanomagnet Fabrication and Implementation of Arithmetic Circuits

Another extension of the work presented here is fabrication of a low-cell count QCA-based one-bit adder. The state of the art full-adder [4] that has been fabricated using nano-magnets uses three majority gates and four inverters while we can exploit the results here to reduce the number of inverters (and therefore cells). One approach for fabrication of our proposed full-adder (that uses just three majority gates and one inverter) is Focussed Ion Beam (FIB) irradiation. The environment described in [4] that consists of Platinum/Cobalt multilayer for making nanomagnets can be adopted. The fabricated structure should then be analysed (and magnetic properties measured). An alternate approach for fabrication is based on nanoimprint lithography and electrodeposition.

9.2.4 Application to Other Discrete Orthogonal Transforms

The approach presented for the discrete Hadamard transform can be extended to other discrete orthogonal transforms such as the discrete Fourier transform and the

discrete cosine transform. It would be of interest to see if a unified strategy can be developed.

9.3 Concluding Remarks

Research on logic optimization, particularly focussing on alternatives to NAND and NOR, that was initiated in 1960s has been revived with the emergence of new nanotechnologies. We believe that, in addition to theoretical development, advanced tools for automated optimization as well as design in upcoming nanotechnologies will emerge in the years to come.

References

1. P. Mazumder, S. Kulkarni, M. Bhattacharya, J.P. Sun, G.I. Haddad, Digital circuit applications of resonant tunneling devices. *Proc. IEEE* **86**(4), 664–686 (1998)
2. R. Cowburn, M. Welland, Room temperature magnetic quantum cellular automata. *Science* 1466–1468 (2000)
3. A. Imre, G. Csaba, L. Ji, A. Orlov, G.H. Bernstein, W. Porod, Majority logic gate for magnetic quantum-dot cellular automata. *Science* **311**, 205–208 (2006)
4. S. Breitkreutz, J. Kiermaier, I. Eichwald, C. Hildbrand, G. Csaba, D. Schmitt-Landsiedel, M. Becherer, Experimental demonstration of a 1-bit full adder in perpendicular nanomagnetic logic. *IEEE Trans. Magn.* **49**(7), 4464–4467 (2013)

Appendix

A Tutorial on QCADesigner

In this appendix, we describe how to (i) create QCA layouts using QCADesigner freeware (ii) perform a simulation and (iii) measure complexity of the design at the layout level. Readers are also encouraged to go through the webpage of the authors of QCADesigner (<http://www.mina.ubc.ca/qcadesigner>) for additional documentation.

A.1 Design of Layouts for QCA Primitives

The three primitives in QCA are the 3-input majority gate, inverter and wire. The majority function, $M(A, B, C)$ with three inputs A, B and C is given by $M(A, B, C) = AB + BC + AC$. However, one need not design layouts for a 2-input AND and a 3-input OR to realize a majority gate in QCA. A realization with nine cells is shown in Fig. A.1. (Note that a 5-cell majority gate is depicted in Fig. 2.4 in Chap. 2, however additional cells for input and output ensure correct simulation). It is worth noting that creation of the nine cells involves left clicking on “Cell” on the list in the left in QCADesigner and again left clicking in the design area. By default, cells (for a new design) are in the “Main Cell Layer” (this can be observed from the entry in the box at the top). Since three inputs are to be given, three cells at the ends are assigned a “cell function”. This is done by clicking on “Select” and left clicking on the appropriate cells. This brings up a window where a cell can be given a label as well as identified as “input” or “output”. The specification for one cell on the right (as output along with a label) is also done in a similar manner. All circuits in QCA are clocked. All inputs should be in the same clock zone.

The layout design for an inverter is similar to the one for the majority gate. The layout is shown in Fig. A.2. The layout for a wire (that connects for example a majority gate to an inverter) can be obtained by clicking on “Cell” followed by left clicking as many times as required (in the design area) to get the desired number of cells. Alternatively, a long wire can be more conveniently created by a left click on the “Array” button and dragging the mouse in the design area up to the desired

Fig. A.1 QCADesigner
layout for 3-input
majority gate

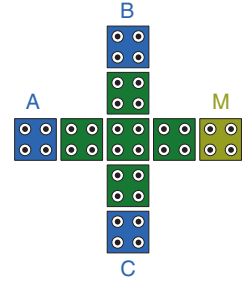
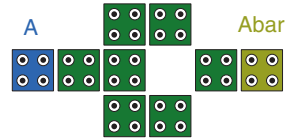


Fig. A.2 QCADesigner
layout for an inverter



length. Deletion of an (unwanted) cell is also via left click on “Select” followed by pressing the Delete key. A group of cells can be deleted by left clicking on “Select” followed by creating a box around the set of cells to be deleted and pressing the Delete key.

A.2 Performing a Simulation

Once the layout for a design is completed, the next step is simulation. Simulation requires specification of inputs (including format) as well as the engine. The default engine available under “Simulation Engine Setup” is “Bistable Approximation”. This is adequate for simulations of basic circuits. (The other choice, namely “Coherence vector” is appropriate if one is interested in studying thermal robustness of designs.) Specification of inputs for a design is accomplished via “Simulation —> Simulation Type Setup”. In the window that appears, one can specify the manner in which the input is to be given. In general, it is preferable to choose the “Vector Table” option and specify various choices for the inputs by clicking on the “+” symbol. Once the set of desired inputs is specified, the simulation is run via “Start Simulation”.

A.3 Measuring the Complexity of a Design at Layout Level

Given the layout, one can measure the cell count, area and delay for a design. The measurements are performed by ‘making a box’ around the cells in the design. The cell count and area appear at the bottom of the window. In the case of larger designs, one needs to click on ‘Extents’. The delay is obtained by counting the number of clock zones of a specific colour.

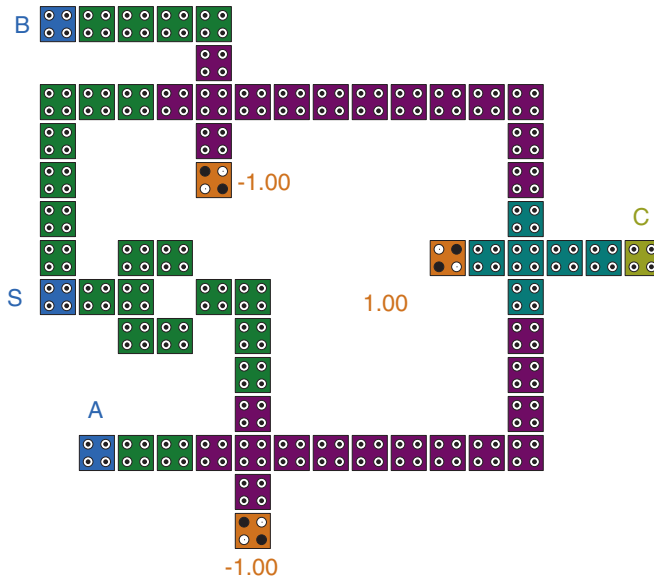


Fig. A.3 QCADesigner layout for a 2-to-1 multiplexer

A.4 Clocking a Circuit

As indicated earlier, all circuits in QCA are clocked. Appropriate clock zones need to be assigned to different parts of the circuit for correct functionality. A simple combinational circuit that illustrates this is a 2-to-1 multiplexer that takes two (data) inputs A and B and a select input S to output $C = A\bar{S} + BS$. The QCA layout for a 2-to-1 multiplexer is depicted in Fig. A.3. The inputs are in the same clock zone (clock zone 0). Note a change of clock zone for the majority gate that implements $A\bar{S}$ as well as BS . Thereafter, there is a change of clock zone for the final OR (of $A\bar{S}$ with BS). We also note that the third input for an AND gate is logic-0 (indicated by a polarization of -1) while the third input for an OR gate is logic-1 (indicated by a polarization of 1).

A.5 Crossover

The QCADesigner layouts for other elements such as AND and OR are obtained similarly. In particular, given the design for a 3-input majority gate, one can immediately obtain layouts for 2-input AND and OR gates. In particular, setting the third input to a '1' or '0' leads to these realizations. In particular, $A + B = M(A, B, 1)$ while $AB = M(A, B, 0)$. The layouts for these are shown in Chap. 3. While layouts

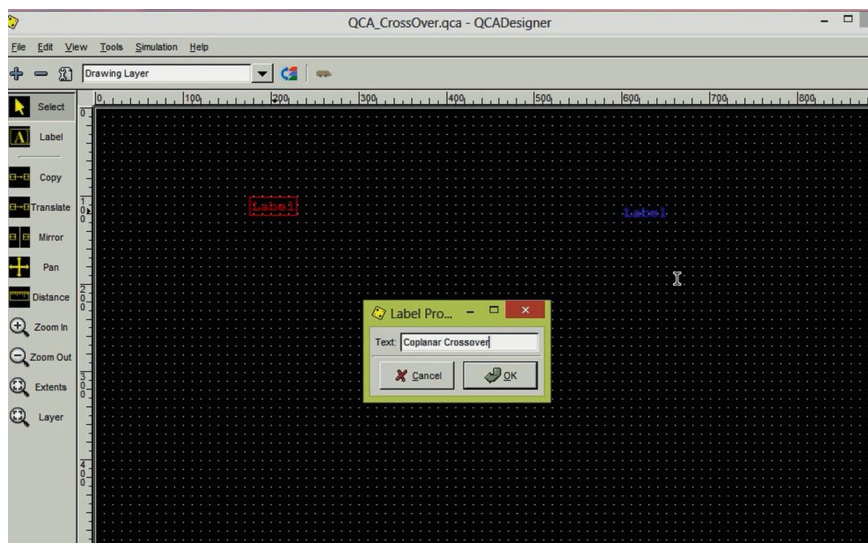


Fig. A.4 Coplanar crossover—snap 1

for AND and OR are straightforward, designs for many arithmetic circuit elements involve *crossover*. Two types of crossover are possible: *multilayer* and *coplanar*. The coplanar crossover is illustrated in Figs. A.4, A.5, A.6 and A.7. The multilayer crossover is illustrated in Figs. A.8, A.9, A.10, A.11, A.12, A.13, A.14, A.15, A.16, A.17, A.18 and A.19. The multilayer crossover is used for various arithmetic circuits in this book since it requires only one type of cell (regular). Further, there are no misalignment problems while designing a circuit with multilayer crossover.

A.6 Illustration of a Circuit with Crossover

In this section, we illustrate the layout for a circuit that involves crossover. In particular, the layout in Fig. A.20 implements a 2-input XOR given by $X = A\bar{B} + B\bar{A}$. The design can be thought of as a modification of the 2-to-1 multiplexer layout presented earlier. The circuit in Fig. A.20 involves one multilayer crossover.

A.7 Printing Layouts to a File

With regard to printing of the layouts, QCADesigner provides options for margins and scale under 'Print —> Printer Setup'. Further, the Print menu also provides an item called 'Printed Objects'. Setting the 'Margins' to 0, 'Scale' to 'Fit' and deselecting 'Substrate' and 'Drawing layer' under 'Printed Objects' generates output of the desired size.

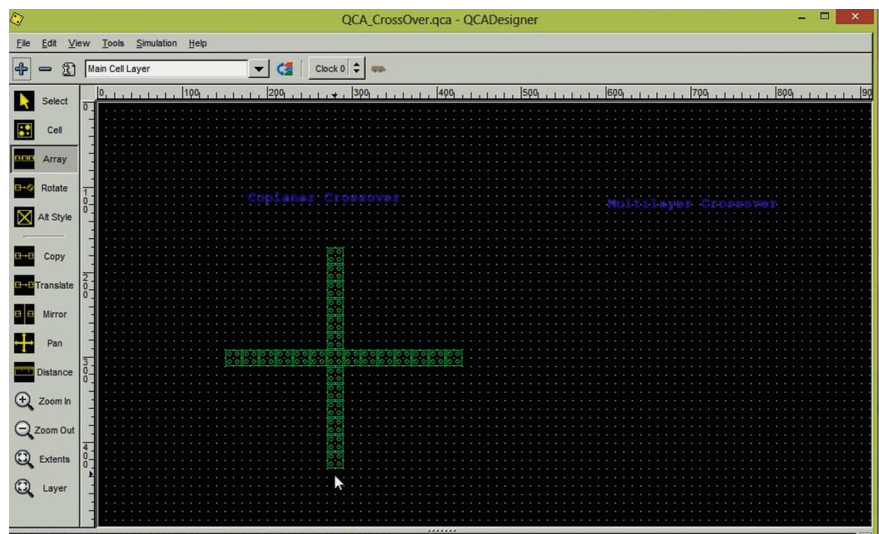


Fig. A.5 Coplanar crossover—snap 2

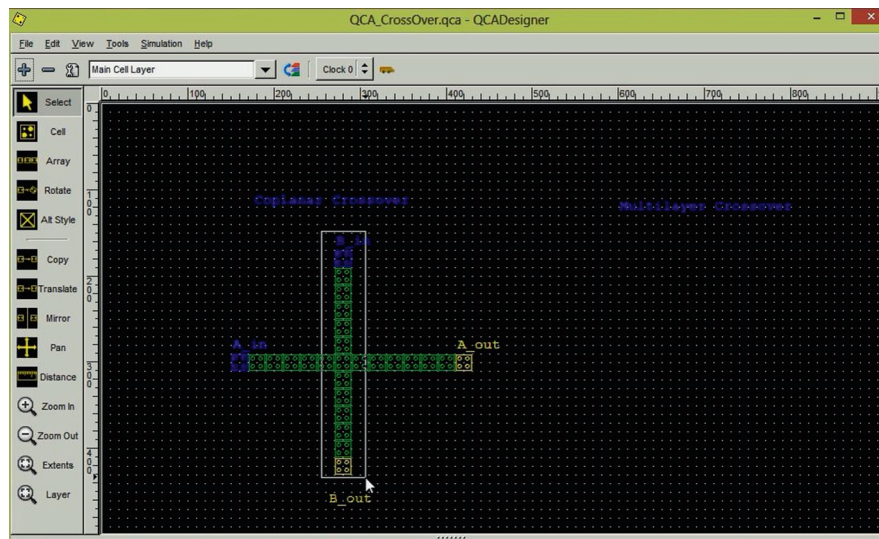


Fig. A.6 Coplanar crossover—snap 3

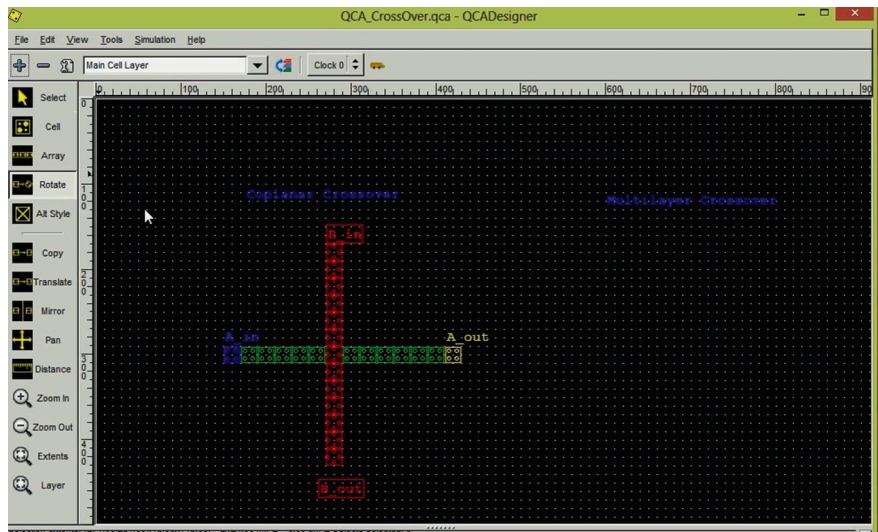


Fig. A.7 Coplanar crossover—snap 4

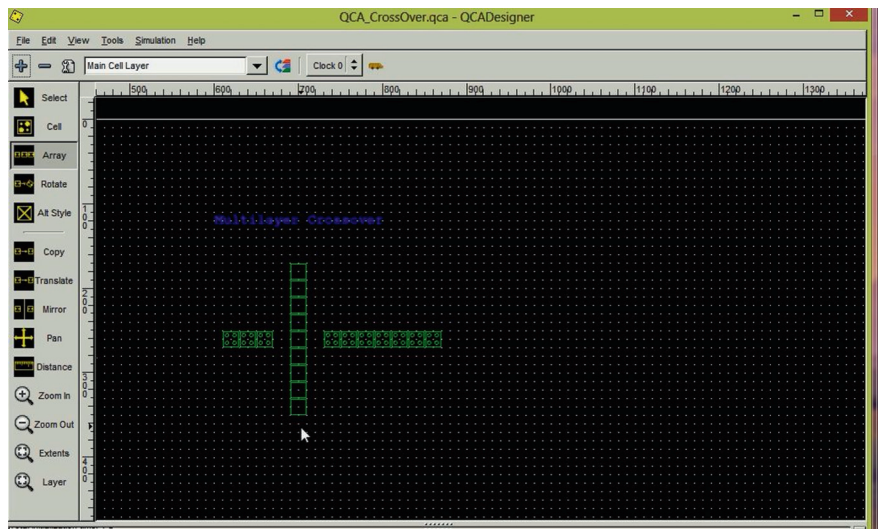


Fig. A.8 Multilayer crossover—snap 1

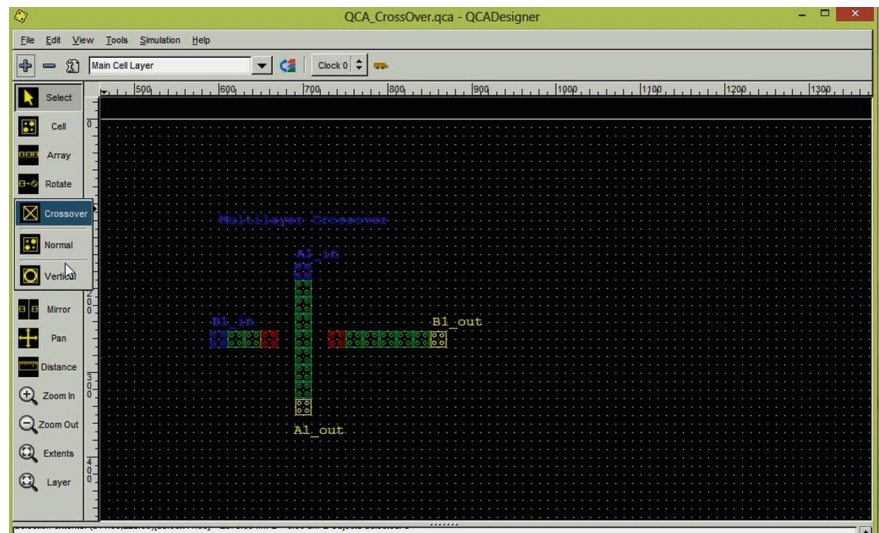


Fig. A.9 Multilayer crossover—snap 2

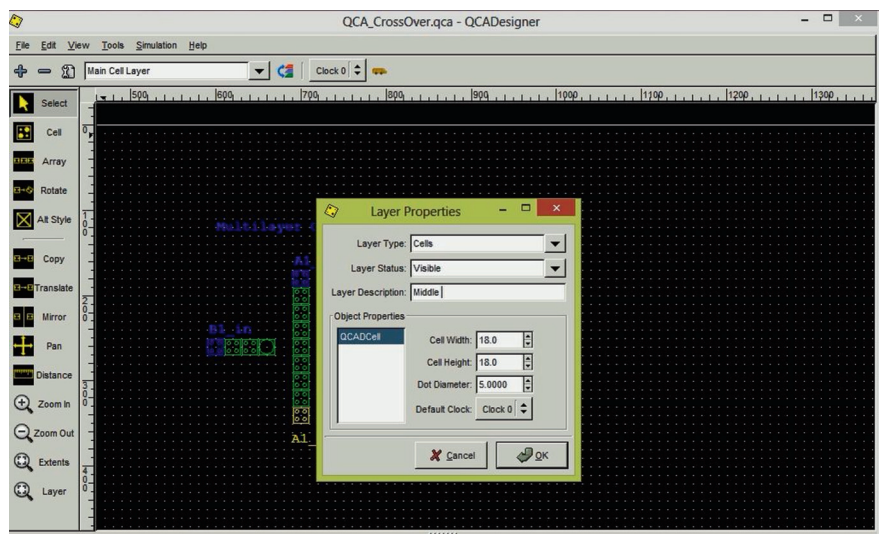


Fig. A.10 Multilayer crossover—snap 3

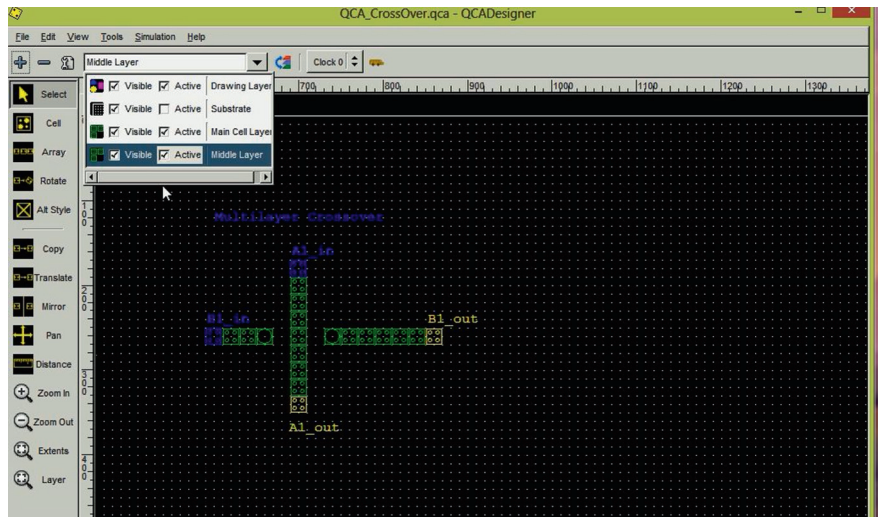


Fig. A.11 Multilayer crossover—snap 4

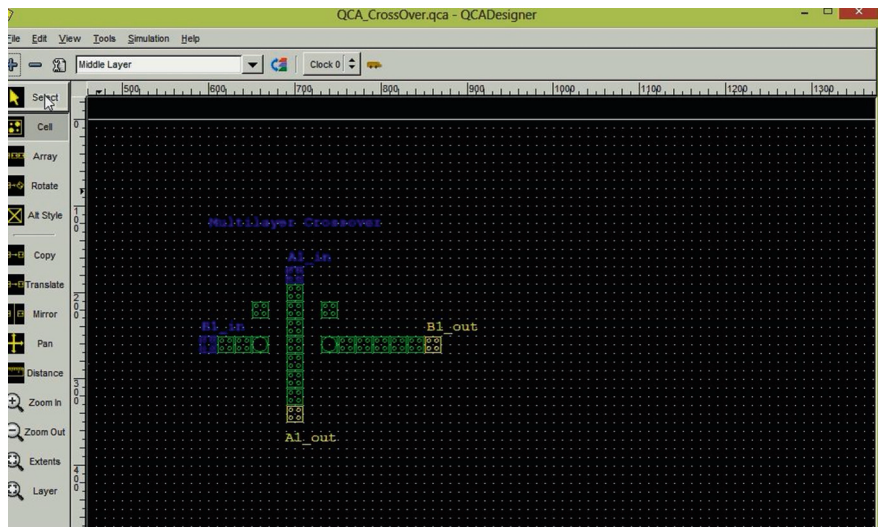


Fig. A.12 Multilayer crossover—snap 5

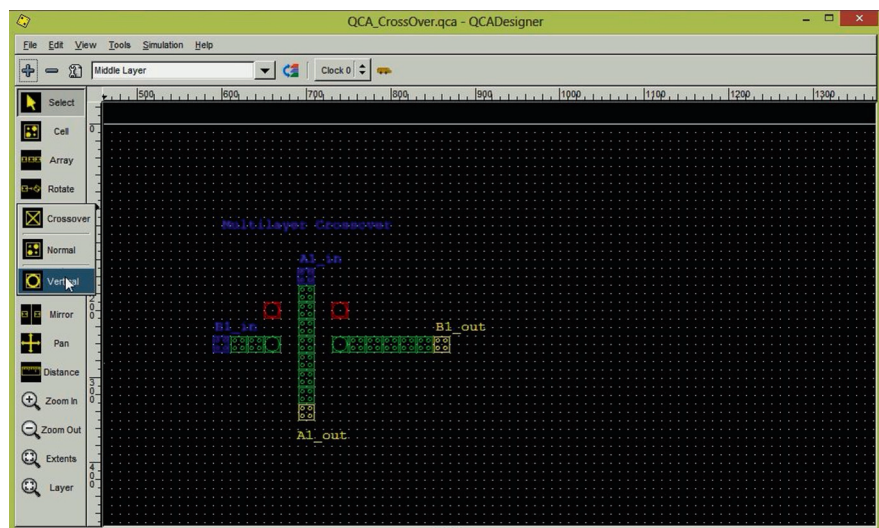


Fig. A.13 Multilayer crossover—snap 6

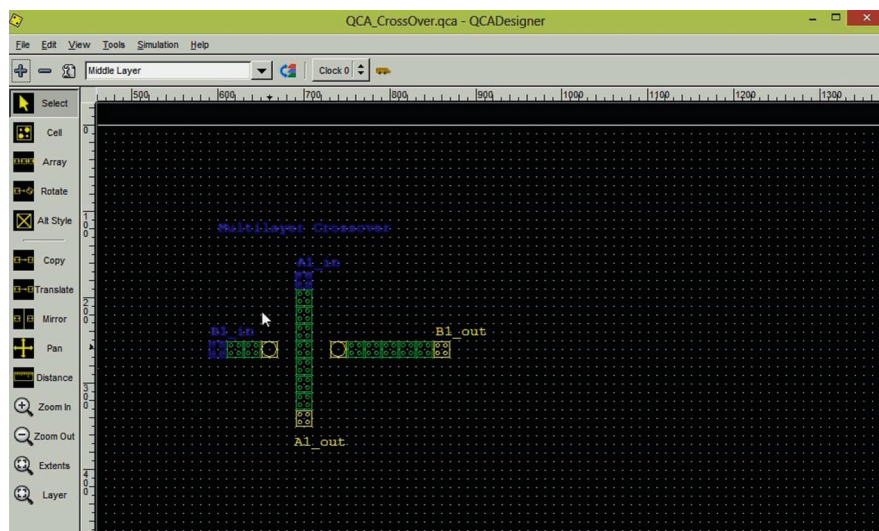


Fig. A.14 Multilayer crossover—snap 7

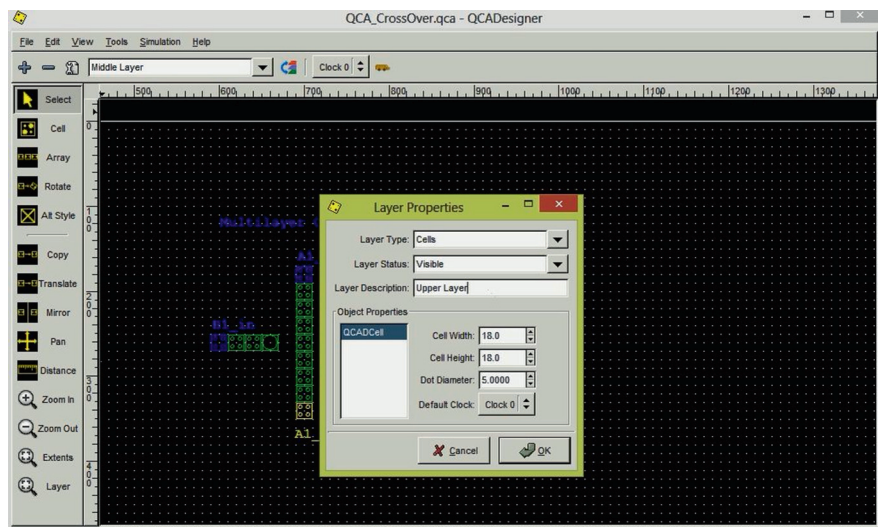


Fig. A.15 Multilayer crossover—snap 8

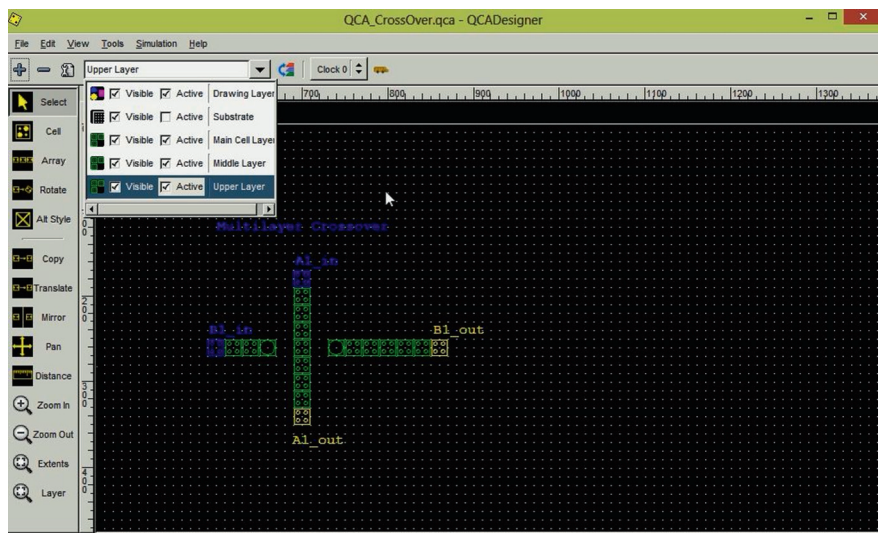


Fig. A.16 Multilayer crossover—snap 9

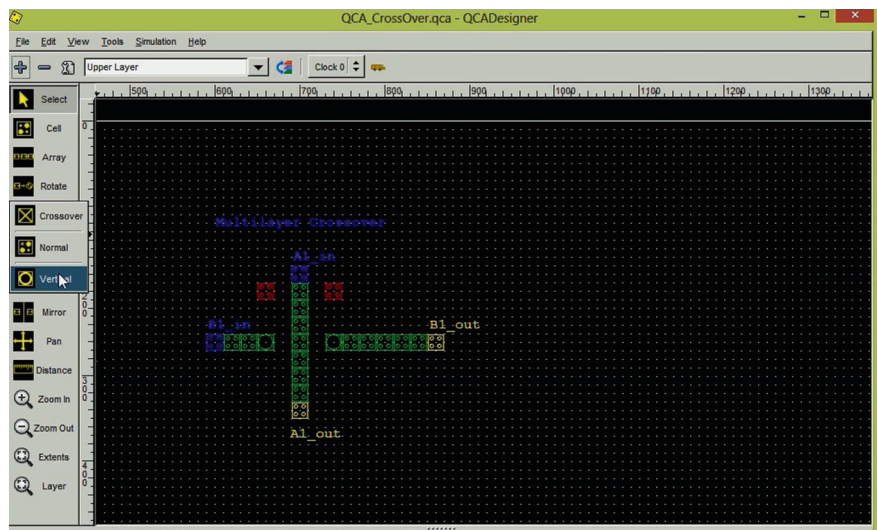


Fig. A.17 Multilayer crossover—snap 10

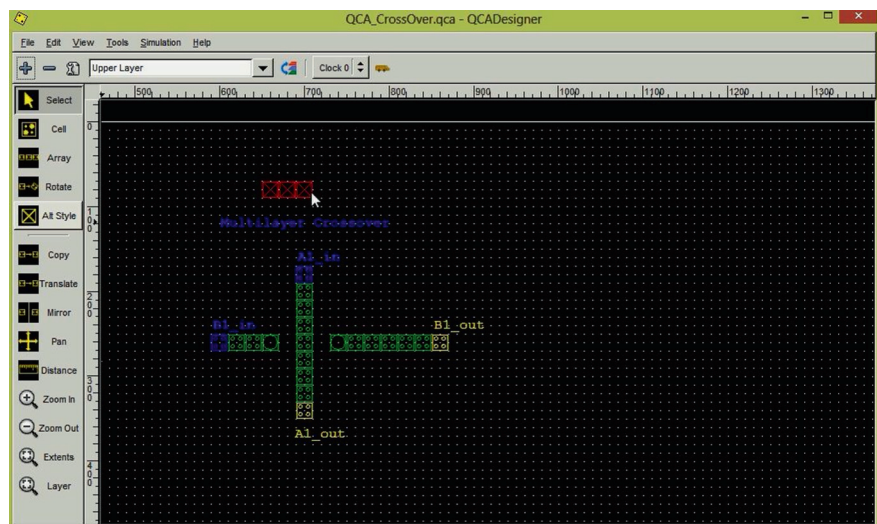


Fig. A.18 Multilayer crossover—snap 11

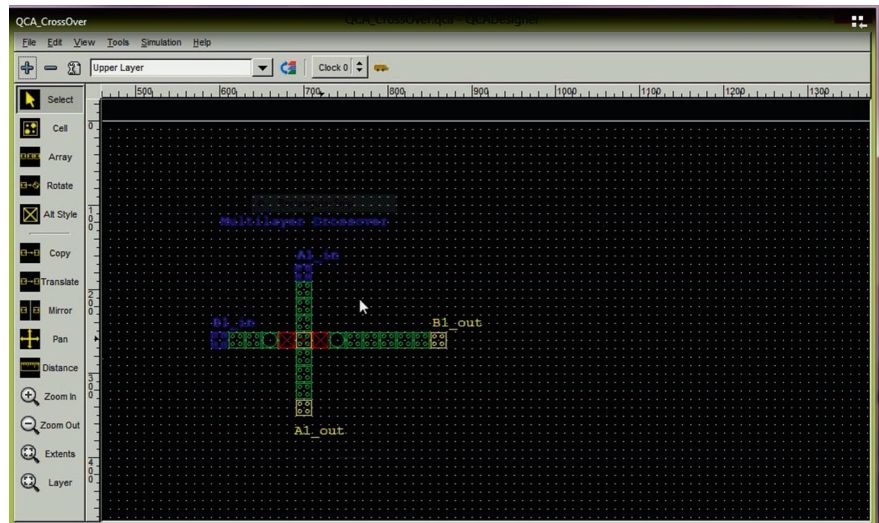


Fig. A.19 Multilayer crossover—snap 12

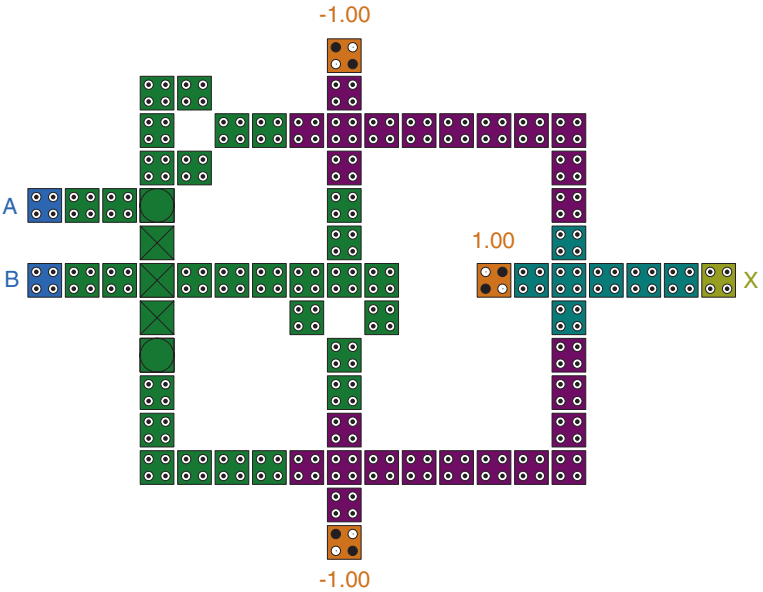


Fig. A.20 QCADesigner Layout of an XOR gate depicting crossover

A.8 Summary

In this appendix, we have briefly described design of layouts in QCADesigner. Videos of clocking and crossover are available at the URL http://www.ee.iitm.ac.in/~sridhara/QCADesigner_Videos.

Index

A

AND gate, 19
Area-Delay-Cycle (ADC) product, 88

B

Baugh-Wooley multiplier, 73
Bit-serial pipelined architecture, 80
Brent-Kung adder, 42

C

Carbon nanotube FET, 1
Clocking, 14
Coulomb repulsion, 12
Crossover, 3

D

Discrete Fourier Transform, 98
Discrete Hadamard Transform, 79

F

Flip-flop, 25
Full-adder, 24

G

Ground state polarization, 11

H

Hybrid adder, 5, 57

K

Kogge-Stone adder, 34

L

Ladner-Fischer adder, 39

M

Magnetic QCA, 98
Majority gate, 3
MOSFET, 1
Multilayer crossover, 16
Multiplexer, 23

O

OR gate, 19

P

Prefix adder, 31

Q

QCADesigner, 5
Quantum dot cellular automata, 2
Quantum dots, 3

R

Ripple carry adder, 27

S

Single electron transistor, 1
Spin-based devices, 2

T

Thermal robustness, 91
Tunnel FET, 1