

Design and Synthesis of Carbon Nanotube Field Effect Transistor (CNFET)-based Ternary Logic Circuits

THESIS

Submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

Chetan Kumar V

ID No. 2009PH230006H

Under the Supervision of
Prof. M. B. Srinivas



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,
PILANI**

2018

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,
PILANI**

CERTIFICATE

This is to certify that the thesis titled Design and Synthesis of Carbon Nanotube Field Effect Transistor (CNFET)-based Ternary Logic Circuits and submitted by Chetan Kumar V ID No 2009PH230006H for award of PhD of the Institute embodies original work done by him under my supervision.

Signature of the Supervisor

Name: Dr. M. B. SRINIVAS

Designation: Professor

Date:

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. M. B. Srinivas for his constant support, patience and valuable guidance throughout my work.

I would also like to thank my Doctoral Advisory Committee members, Prof. BVVSN Prabhakar Rao, Dr. Surya Shankar Dan and Prof. Y Yoganandam for their time and their constructive comments and suggestions.

Next I would like to express my deep sense of gratitude to BITS Pilani, Hyderabad Campus for providing all the necessary facilities and support to complete the research work.

I am thankful to Sai Phaneendra P. for the help I received in developing the circuit simulation framework. I also appreciate the help from Goutham M, Avinash S Vaidya and Ganesh Kumar, during different stages of my work.

Furthermost, I would like to thank all members of Department of Electrical Engineering at BITS-Pilani, Hyderabad Campus, who have supported me by their suggestions and discussions.

Finally, I dedicate this work to my parents Pampana Gowda and Vishalakshi. Words can not express my gratitude to them for their constant support, inspiration and love. I deeply value the encouragement and support from my wife, Nydhili and my sister, Swetha.

Hyderabad

Chetan Kumar V

Abstract

Multi-Valued Logic (MVL) circuits have attracted the attention in recent times because of the advantages they offer in reducing the interconnect complexity and increasing the information content per unit area. Ternary Logic is a special case of MVL that has three logic levels. Implementation of voltage mode ternary logic circuits requires transistors with different threshold voltages. Since traditional Metal-Oxide-Semiconductor (MOS) transistors use body biasing to change the threshold voltage, design of ternary logic circuits using MOS transistors becomes complex. On the otherhand, Carbon-Nanotube Field Effect Transistor (CNFET) are becoming popular in the implementation of ternary logic circuits. This is because Carbon-Nanotube (CNT) is used as a conduction channel in CNFET and variations of the diameter of CNT results in variation in threshold voltage of CNFET. This property of CNFET makes it suitable for implementation of MVL circuits in general and ternary logic circuits in particular.

In this thesis, we initially present three general design approaches to implement basic ternary logic circuits. The first design approach avoids the use of decoder and uses a novel low-power encoder resulting in ternary circuits that have low transistor count when compared to existing approaches. The second design approach uses a delay optimized decoder and low-power encoder leading to energy efficient ternary circuits. The third design approach uses 2:1 multiplexers to realize basic ternary logic circuits. This approach leads to ternary circuits which have low power consumption but large delay. Basic 2-input circuits are implemented and their design parameters are compared with that of existing approaches.

The second contribution of this thesis is two design techniques to implement multi-

digit adders. The first technique is a half-adder based ripple carry adder, in which outputs (instead of main inputs) are used for carry-out computation resulting in delay-optimized carry propagation path. The second design technique uses the concept of carry propagate-generate. In this thesis, a technique is presented, which enables the use of propagate-generate concept and aids in realizing ternary prefix adders. The proposed adder designs are compared with existing CNFET-based multi-digit ternary adders with respect to different design parameters.

The analysis of ternary adders shows that, the ternary encoder is a critical element and contributes significantly to the overall power consumption of the ternary circuit. In this thesis, new designs for CNFET-based ternary encoders, which are optimized for delay and power consumption, are presented. These designs are used to develop encoder based optimization algorithms which choose appropriate encoder for different outputs of a ternary logic circuit. The proposed algorithms are applied on an example ternary circuit to show their advantages in optimizing the design parameters of the circuit.

Apart from novel designs, this thesis also presents a synthesis technique to implement ternary logic circuits. Traditionally Binary Decision Diagram (BDD) and Ternary Decision Diagram (TDD) based algorithms have been used to synthesize binary and ternary logic functions respectively. This thesis presents a synthesis technique based on proposed Ternary-Transformed Binary Decision Diagram (TBDD), to implement ternary logic circuits using 2:1 multiplexers. This technique is used to synthesize a set of benchmark ternary functions and the resulting circuits are compared with circuits synthesized using existing techniques.

Contents

Acknowledgements	iii
Abstract	iv
Contents	vi
List of Figures	ix
List of Tables	xiv
List of Abbreviations	xvi
1 Introduction	1
1.1 Overview and Motivation	1
1.2 Contributions	3
1.3 Thesis Outline	4
2 Background and Related Work	6
2.1 Ternary Logic	6
2.2 Carbon-Nanotube Field Effect Transistor (CNFET)	7
2.3 Ternary Logic Circuits using CNFETs	11
2.4 Research Gaps, Objectives and Scope of Current Work	15
3 Design Approaches for Basic Ternary Circuits	17
3.1 Introduction	17
3.2 Existing Design Approaches	18

3.2.1	Decoder-Encoder based approach	18
3.2.2	Multiplexer based approach	20
3.3	Novel Approaches to Design Ternary Logic Circuits	21
3.3.1	Approach I: Using Low-power Encoder and without Decoder . .	21
3.3.1.1	Overview	21
3.3.1.2	Steps Involved	23
3.3.1.3	Function Simplification	23
3.3.1.4	Implementation of Unary Functions	25
3.3.1.5	Low-power Encoder	27
3.3.2	Approach II: Using low-delay Decoder and low-power Encoder .	30
3.3.2.1	Low-Delay Decoder	32
3.3.2.2	Alternate Representation of Logic Expressions	32
3.3.3	Approach III: Using 2:1 Multiplexers	35
3.3.3.1	Basic Idea	35
3.3.3.2	Ternary circuits using CNFET-based 2:1 Multiplexers	37
3.4	Implementation and Simulation	39
3.4.1	Simulation Environment	39
3.4.2	Results and Discussion	42
3.4.2.1	Impact of Process, Voltage and Temperature (PVT) Variations	47
3.4.2.2	Noise Immunity Analysis	49
3.5	Conclusions	50
4	Design of Multi-digit Ternary Adders	52
4.1	Introduction	52
4.2	Previous Work on CNFET-based Ternary Adders	53
4.2.1	Single-Digit Adders	53
4.2.2	Multi-digit Adders	55
4.3	Proposed Half-Adder Based Ripple-carry Ternary Adders	59
4.3.1	Basic Idea	59

4.3.2	Implementation using CNFET	60
4.3.2.1	Designs for Decoder and Half-Adder	60
4.3.2.2	Design of Carry Generator Block	63
4.3.2.3	Design of Final Half-Sum Generator	65
4.3.2.4	Design of Low-Power Encoder	65
4.4	Proposed Ternary Prefix Adder designs	67
4.4.1	Concept of Carry Propagate-Generate in Binary Addition	67
4.4.2	Basic Idea for Ternary Prefix Adders	69
4.4.3	Proposed Implementation of Ternary Prefix Adders using CNFET	71
4.4.3.1	Propagate and Generate for Ternary Adders	72
4.4.3.2	Carry Generation using Binary Prefix networks	74
4.4.3.3	Final Sum and Carry Computation	75
4.5	Simulation Results	78
4.5.1	Results and Discussion	78
4.6	Conclusions	86
5	Encoder-based Optimization of Ternary Circuits	88
5.1	Introduction	88
5.2	Review of Ternary Encoders	89
5.3	Proposed CNFET-based Ternary Encoders	91
5.4	Effects of varying chiralities of CNFETs that are used in Encoders	95
5.5	Algorithms for choosing appropriate Encoders in Ternary Circuits	99
5.5.1	Problem Formulation	99
5.5.2	Power optimization	102
5.5.3	Delay Optimization	103
5.5.4	PDP Optimization	104
5.6	Example: Encoder-based Optimization of Multi-Digit Ternary Adder	106
5.7	Conclusions	112

6	Synthesis of Ternary Logic Circuits using 2:1 Multiplexers	113
6.1	Introduction	113
6.2	Preliminaries	114
6.2.1	Binary Decision Diagrams	114
6.2.2	Ternary Decision Diagrams	115
6.3	Proposed Synthesis Methodology	116
6.3.1	General Ternary-Transformed Binary Decision Diagrams (TBDD)	116
6.3.2	TBDD-based synthesis for 1-input ternary functions (Unary Op- erators)	120
6.3.3	TBDD-based synthesis for 2-input ternary functions	125
6.3.4	TBDD-based synthesis for n -input ternary functions	129
6.4	Algorithm for 2:1 Multiplexer based Synthesis	130
6.5	Synthesis using CNFETs	133
6.6	Results	136
6.6.1	Synthesis	136
6.6.2	SPICE Simulation	138
6.7	Conclusions	141
7	Summary and Future Work	143
7.1	Summary	143
7.2	Future Work	145

List of Figures

2.1	Unrolled sheet of graphite and the rolled lattice structure of CNT [42]	8
2.2	3D view of Carbon-Nanotube Field Effect Transistor (CNFET)	8
2.3	I-V Characteristics of N-CNFET	12
2.4	I-V Characteristics of N-CNFET	12
3.1	Realization of a ternary function using existing decoder-encoder based approach	19
3.2	Implementation of function shown in Table 3.1 using 3:1 Mux based approach [60]	20
3.3	Transistor level Implementation of 3:1 Multiplexer [60]	21
3.4	Implementation of F^2 and F^1 in proposed approach	22
3.5	K -Map simplification for function F	24
3.6	Transistor level realization of X^0 , X^1 and X^2 unary terms	26
3.7	Transistor level realization of $X^0 + X^1$, $X^1 + X^2$ and $X^0 + X^2$ unary terms	27
3.8	Implementation of F^2 , F^1 and F^0 using proposed approach	28
3.9	Encoder with F^2 and F^1 as inputs ((F^2, F^1) -Encoder)	29
3.10	Encoder design for (F^2, F^0) and (F^1, F^0) combination	30
3.11	Implementation of Half-Sum using Decoderless approach	31
3.12	Implementation of Half-Sum using Existing Decoder-encoder based approach (all CNFETs have chirality as (19, 0))	31
3.13	Decoder Implementations	33

3.14	Implementation of Half-Sum with reduced Transistor Stacking (all CN-FETs have chirality as $(19, 0)$)	35
3.15	A 3:1 Multiplexer operation using 2:1 Multiplexers	37
3.16	CNFET-based Implementation of NTI-Mux and PTI-Mux	37
3.17	2:1 Multiplexer based Implementation for Ternary Function in Table 3.4	38
3.18	Half-Adder using Existing Decoder-Encoder based Approach (all CN-FETs have chirality as $(19, 0)$)	40
3.19	Half-Adder using Existing 3:1 Multiplexer based Approach	40
3.20	Half-Adder using Proposed Decoderless Approach (I)	41
3.21	Half-Adder using Proposed Decoder-Encoder based Approach (II) (all CNFETs have chirality as $(19, 0)$)	41
3.22	Half-Adder using Proposed 2:1 Multiplexer based Approach (III)	42
3.23	Simulation Waveforms for Half-Adder	43
3.24	Propagation Delay Vs Output Load for Half-adder	46
3.25	Monte-Carlo Simulations for (a) Delay (b) Power	48
3.26	(a) Delay (b) Power for Supply Voltage Variation	48
3.27	(a) Delay (b) Power for Temperature Variation	49
3.28	Noise Immunity Curve for Half-Adders	50
4.1	Design techniques used to implement Multi-digit adders	54
4.2	Proposed design technique to implement Multi-digit adders	58
4.3	Decoder Implementation	61
4.4	Gate-level Implementation to generate Half-Adder outputs [36]	62
4.5	Transistor-level Implementation for HS_i^2	62
4.6	Proposed Transistor-Level Implementation of Half-carry Generator and half-sum Generator I (all transistors with chirality of $(19, 0)$)	63
4.7	Delay-Optimized Carry Propagation Path with Proposed Carry Generator Blocks (all transistors with chirality of $(19, 0)$)	65

4.8	Transistor-level implementation of Half-sum generator II (all transistors with chirality of $(19, 0)$)	66
4.9	Designs for Encoder	67
4.10	Table showing Carry Propagate and Generate conditions for Binary addition	68
4.11	Example of ternary addition using proposed transformation	69
4.12	Table showing Carry Propagate and Generate conditions for ternary addition	70
4.13	Table showing Carry Propagate and Generate conditions after proposed transformation	71
4.14	Block-level Implementation of Proposed Ternary Adders	72
4.15	CNFET-based Implementation of Simplified Half-Adder (all CNFETs have chirality of $(19, 0)$)	74
4.16	Prefix Networks used for Carry Computation in 6-digit Ternary Adder	76
4.17	CNFET-based Implementation of Cells used in Prefix Networks (all CNFETs have chirality of $(19, 0)$)	77
4.18	Simulation Waveforms for Kogge-Stone Prefix Adder	80
4.19	A Comparison of Power consumption	81
4.20	A Comparison of Propagation Delay for FO4 load	82
4.21	Propagation delay vs load for 12-digit adder	84
4.22	A Comparison of PDP	85
5.1	Ternary Encoders	90
5.2	Proposed Encoder (with inputs $\overline{X^2}$ and X^0)	92
5.3	Proposed Low-Power Encoder (with inputs X^2 and X^0)	93
5.4	Proposed Low-Delay Encoder (with inputs X^2 and X^0)	93
5.5	I-V Characteristics of N-CNFET	96
5.6	Power Consumption for Encoders with Transistors of Different Chirality	97
5.7	Propagation Delay for Encoders with Transistors of Different Chirality	98
5.8	Ternary Circuit as a Graph	100

5.9	Multi-digit adder presented in [58].	107
5.10	Power Consumption Vs Propagation Delay for 9-digit Ternary Adders	112
6.1	Binary and Ternary Decision Diagrams	114
6.2	BDD and its 2 : 1 Mux based implementation for a given Truth-table	115
6.3	TDD and its 3 : 1 Mux based implementation for a given Truth-table	115
6.4	Ternary-Transformed Binary Decision Diagram	118
6.5	2 : 1 Multiplexer based implementation of TBDD	118
6.6	Illustration for Rule 1	119
6.7	K-map for 1-input Ternary Function	121
6.8	TBDDs for 1-input Ternary Function	121
6.9	TBDD for $F_{A^0} = 2$, $F_{A^1} = 0$ and $F_{A^2} = 0$	122
6.10	Multiplexer-based implementation of Reduced TBDD in Figure 6.9 and its equivalent Ternary Gate	122
6.11	TBDD templates, their 2:1 multiplexer based implementations and equivalent gates.	122
6.12	TBDD Example for Proposition 6.4	123
6.13	K-map Representation of 2-input Function	125
6.14	K-map Representation of of 2-input Function using 1-input Functions	126
6.15	An Example for 2-input Function	127
6.16	TBDD representation when decomposed w.r.t A	127
6.17	TBDD representation when decomposed w.r.t B	128
6.18	TBDD representation for 2 input function in Figure 6.15	128
6.19	An Example for 2-input Function	129
6.20	TBDD representation for function in Figure 6.19, when decomposed w.r.t A	129
6.21	TBDD representation for function in Figure 6.19, when decomposed w.r.t B	130
6.22	TBDD for Ternary Full Adder	134
6.23	Implementation of 2:1 Multiplexers	135

6.24	Implementation of TBDD in Figure 6.18	136
6.25	Implementation of Ternary Full Adder	137

List of Tables

2.1	Ternary Inverters [36]	7
2.2	Logic Symbols	7
2.3	Relation Between Chirality, CNT Diameter and Threshold Voltage [44]	10
2.4	Technology Parameters for CNFET model in [44, 45, 53]	11
3.1	Truth Table (Example 1)	19
3.2	Truth Table (Example 2)	24
3.3	Half-Sum (HS)	30
3.4	Ternary Function (Example 1)	38
3.5	Truth-Table for basic Ternary Circuits	39
3.6	Simulation Results for Basic Circuits	44
3.7	Propagation Delay vs Load	46
3.8	ANTE for Ternary Half-Adders	50
4.1	Half-Adder Karnaugh Maps	61
4.2	Truth-Table for Carry-Out	64
4.3	Average Power Consumption for N -digit Adders	79
4.4	Propagation (FO4) Delay for N -digit Adders	81
4.5	Delay for N -digit Adders for different output loads	83
4.6	Power-Delay Product for N -digit Adders	84
4.7	Number of Transistors required for N -digit Adders	85
5.1	A Comparison of Encoders	94
5.2	Encoder Mapping for Ternary Adders	108

5.3	Simulation Results for N -digit Ternary Adders	111
6.1	Unary Operators	124
6.2	Comparison of Transistor Count of Proposed 2:1 Multiplexer based Algorithm with that of Existing 3:1 Multiplexer based Algorithm [41]	139
6.3	Simulation Results for Ternary Benchmark Circuits	140
6.4	Comparison of Ternary Adders	141
6.5	Comparison of Existing Manual Designs with those Generated using the Proposed Algorithm	142

List of Abbreviations

ANTE	Average Noise Threshold Energy
BDD	Binary Decision Diagram
CMOS	Complementary Metal-Oxide-Semiconductor
CNFET	Carbon-Nanotube Field Effect Transistor
CNT	Carbon-Nanotube
CSA	Conditional Sum Adder
HC	Half-Carry
HS	Half-Sum
IC	Integrated circuit
MHTA	Multi-Digit Half-Adder based Ternary Adder
MOS	Metal-Oxide-Semiconductor
MOSFET	Metal-Oxide Semiconductor Field Effect Transistor
MSD	Most Significant Digit
MTA	Multi-Digit Ternary Adders
MTPA	Multi-Digit Ternary Prefix Adder
MVL	Multi-Valued Logic

NIC	Noise Immunity Curve
NWFET	Nano-Wire Field Effect Transistor
PDP	Power-Delay Product
qFET	Quantum-dot gate field effect transistor
QROBDD	Quasi Reduced BDD
QROTDD	Quasi Reduced TDD
RBDD	Reduced BDD
RTDD	Reduced TDD
TBDD	Ternary-Transformed Binary Decision Diagram
TDD	Ternary Decision Diagram
VLSI	Very Large Scale Integrated Circuit

Chapter 1

Introduction

1.1 Overview and Motivation

Integrated circuit (IC) technology has enabled rapid advances in design and implementation of innovative devices, and systems that have changed the way we live and communicate. Integration of more transistors increases the computing power and helps in building efficient systems [1]. The number of transistors that can be integrated on a chip has been doubling every 1-2 years as predicted by the Gordon Moore, an industry pioneer, in 1960s [2]. This prediction, famously known as Moore's Law, has been proven correct, time and again. This has been made possible mainly due the continuous scaling or miniaturization of components that are integrated onto a chip [3]. For example, in CMOS technology, the gate length of a Metal-Oxide Semiconductor Field Effect Transistor (MOSFET) has been scaling by a factor of 0.7 every two years. Over the last few years, FinFET [4] (a variation of MOSFET) based devices have been fabricated at $22nm$ and the $14nm$ technology is foreseen to be reached in near future [5]. CMOS technology scaling beyond deep sub-micron/nano range, while enabling higher integration of VLSI designs, has caused various reliability issues. Some of the issues of CMOS scaling beyond nanometer range are increased leakage current, processes variations etc [6]. These non-idealities have caused the I-V characteristics of MOSFETs to be different from what is expected. It has become more difficult to improve performance by technology scaling.

This has led to the emergence of alternate computing paradigms (reversible computing [7], multi-valued logic (MVL) computing [8]) coupled with emerging devices (carbon-nanotube field effect transistor (CNFET) [9], Quantum-dot gate field effect transistor (qFET) [10]etc). Researchers have also been investigating new materials and devices in sub-10nm which could possibly replace MOS based transistors. Based on ITRS road map [5], some of the emerging devices that have the characteristics to replace traditional MOS based devices are Carbon-Nanotube Field Effect Transistor (CNFET) [11, 12], Nanowire Field Effect Transistor (NWFET) [13], Graphene Transistor [14, 15] and III-V compound semiconductor [16, 17].

One of the computing paradigms that has received considerable attention over the last few decades is MVL [18]. Three-valued or Ternary Logic, which is a special case of MVL has attracted considerable interest over the last couple of decades. A recent survey presents various contemporary aspects related to MVL [8]. Some of the advantages of MVL include reduced interconnect complexity, less device count etc. This is due to the fact that more information is embedded per digit. For example, it is possible to represent a 14-digit (N-digit) binary number using only 9 ($\log_3(2^N - 1)$) ternary digits. Ternary logic is a special case of MVL with three significant states. There have been many CMOS-based implementations for ternary logic [19, 20]. It has been shown that the performance of CMOS-based designs is enhanced by adding Multi-Valued Logic (MVL) blocks to binary designs [21, 22]. A design for ternary memory units and sequential circuits has been presented in [23]. A CMOS based ternary Wallace-tree multiplier has been implemented in [24]. Apart from the works which focus on novel designs [23–26], there have been many works which focus on synthesis of MVL logic circuits [27–29].

The CMOS implementations of MVL are mainly classified as current-mode circuits [30] which require transistor biasing and voltage-mode circuits [22] which require additional voltage sources to create multi-threshold transistors. Due to the problems in MOS-based devices and non-availability of appropriate devices, design of efficient MVL circuits has for long remained a concern [18]. But, the emergence of several new

device technologies [9, 10, 31] has led to renewed interest in ternary and quaternary logic in particular.

CNFETs have been used widely in the implementation of ternary logic circuits. CNFET is one of the promising alternatives to MOSFET due to their unique one-dimensional band-structure that suppresses backscattering and makes near-ballistic operation a realistic possibility [32–35]. CNFETs use single walled CNT as a conducting channel, which is conducting or semiconducting depending on the angle of atom arrangement along the tube also called as chirality vector. Unlike in MOS technology, where body biasing is used to control threshold voltages, in CNFET technology the threshold voltage is controlled by changing the diameter of CNT which in turn depends on the chirality vector. [36]. This dependence makes CNTFET suitable for implementation of MVL circuits.

There have been many CNFET-based design [36–40] and synthesis techniques [41] that are used to realize ternary logic circuits. The existing work on CNFET-based ternary logic circuits is relatively recent and there is scope to explore new design techniques to realize efficient ternary circuits. Therefore in this thesis, new design and synthesis techniques, which aid in realizing efficient CNFET-based ternary logic circuits, have been investigated.

1.2 Contributions

In this thesis, design and synthesis techniques for ternary logic circuits, which exploit the threshold voltage variability of CNFETS, are investigated. Initially, design approaches for basic ternary circuits are presented. Based on the analysis of these basic circuits, ternary adders have been implemented using one of the proposed approaches resulting in energy efficient circuits. Ternary encoder is a critical element which affects the power consumption of the ternary circuit. Hence new low-power encoders are designed and their performance studied. In addition to design techniques, a new synthesis technique is also presented which is used to implement ternary logic circuits. The main contributions of the thesis are listed below:

1. New designs for low-power ternary encoders, which help in designing low-power and energy efficient ternary circuits.
2. A new decoderless design approach to implement ternary circuits.
3. A novel ternary prefix-adder design which uses for carry generation/propagation.
4. Algorithms to choose appropriate encoders for different output stages of a ternary circuit, such that the circuit is optimized with respect to delay or power consumption.
5. A novel Ternary-Transformed Binary Decision Diagram (TBDD) is presented which can be used to implement ternary logic circuits using 2 : 1 Multiplexers.

1.3 Thesis Outline

This thesis is organized as follows:

Chapter 2 presents the background related to ternary logic and CNFETs. This chapter also presents a brief review of earlier work related to design and implementation of CNFET-based ternary logic circuits available in literature.

Chapter 3 presents three general design approaches to implement basic ternary logic circuits, resulting in reduced area, power consumption and delay. The first approach avoids the use of decoder and uses a novel low-power encoder. The second approach uses a delay optimized decoder and low-power encoder leading to energy efficient ternary circuits while the third approach uses 2:1 multiplexers for realizing basic ternary logic circuits. Basic ternary circuits namely half-adder and 1-digit multiplier are implemented using different approaches and important metrics such as propagation delay, power consumption and power-delay product are quantified and compared.

In **Chapter 4**, two designs of multi-digit ternary adders, which use one of the proposed design approaches, are presented. In the first design, the half-adder outputs (instead of main inputs) are used to compute carry-out at each digit-adder stage resulting in delay optimized carry propagation path. The second design is based on

the concept of carry Propagate-Generate, which is used in the design of binary prefix adders. Since this concept cannot be applied directly to implement ternary prefix adders. A novel technique that enables the use of Propagate-Generate resulting in design of ternary prefix adders, is developed. Proposed and existing CNFET-based adder designs are implemented and their design parameters are compared to quantify the performance.

The analysis of ternary adders showed that, the ternary encoder is an integral part of the designs and contributes significantly to the overall power consumption of the ternary circuit. **Chapter 5** presents improved encoder designs used in implementation of ternary logic circuits. This chapter also presents optimization algorithms, which map appropriate encoders for different output stages of a multi-output ternary logic circuit resulting in a circuit which is optimized for power consumption or propagation delay or power-delay product.

In **Chapter 6**, novel algorithms which are used to synthesize ternary logic circuits using 2 : 1 Multiplexers, are presented. Traditionally Binary Decision Diagram (BDD) based algorithms have been used to synthesize binary logic functions. A BDD can be transformed into circuit implementation by replacing each node in the BDD with a 2:1 multiplexer. Similarly a Ternary Decision Diagram (TDD) can be transformed in to circuit implementation using 3:1 Multiplexers. In Chapter 6 a methodology, which transforms a 2-input ternary logic function in to a Ternary-Transformed Binary Decision Diagram (TBDD), is presented. This TBDD aids in realizing any ternary logic function using 2:1 multiplexers. This TBDD-based approach is used to develop a synthesis algorithm which is applied on a set of ternary functions and the resulting circuits are compared with 3 : 1 multiplexers based circuits.

Finally, **Chapter 7** summarizes the key findings and contributions of this thesis, and proposes some recommendations for future work.

Chapter 2

Background and Related Work

This chapter provides an overview of ternary logic and CNFETs. A brief review of existing CNFET-based implementations of ternary logic circuits is also presented in this chapter.

2.1 Ternary Logic

Binary logic, when given a significant third value is called ternary logic or three valued logic and functions realized with three values are called ternary logic functions. The values 0, 1 and 2 form the nomenclature to denote the ternary values in this work. A function $f(X)$ is defined as a ternary logic function mapping $\{0, 1, 2\}^n$ to $\{0, 1, 2\}$ where X is given by X_1, \dots, X_n . When $X_i, X_j \in \{0, 1, 2\}$, the basic operations of ternary logic can be defined as:

$$X_i + X_j = \max\{X_i, X_j\} \quad (2.1)$$

$$X_i \cdot X_j = \min\{X_i, X_j\} \quad (2.2)$$

where equations (2.1) and (2.2) indicate OR and AND operations respectively for ternary logic [36]. Another important logic function in ternary logic is a ternary inverter. Table 2.1 shows the outputs of different ternary inverters that are used

in ternary logic. Corresponding to each of the outputs three inverters are defined namely, Negative Ternary Inverter (NTI), Standard Ternary Inverter (STI) and Positive Ternary Inverter (PTI) respectively. The logic values assumed for different voltage levels are shown in Table 2.2 where, voltages 0, $V_{dd}/2$ and V_{dd} correspond to logic values 0, 1 and 2 respectively.

Table 2.1: Ternary Inverters [36]

Input x	NTI (x)	STI (x)	PTI (x)
0	2	2	2
1	0	1	2
2	0	0	0

Table 2.2: Logic Symbols

Voltage Level	Logic Value
0	0
$V_{dd}/2$	1
V_{dd}	2

Implementation of ternary logic circuits requires transistors with different threshold voltages. Hence CNFET technology, where the threshold voltage of transistor can be modified by changing its physical dimensions, is suitable to implement ternary logic circuits [36]. The following section presents a brief overview of CNFET.

2.2 Carbon-Nanotube Field Effect Transistor (CNFET)

A single-walled carbon nanotube (SWCNT) is obtained by rolling up a sheet of graphite along a roll-up vector $C = na + mb$, as shown in Figure 2.1, where m and n are positive integers which specify the chirality of the tube and ' a ' and ' b ' are lattice unit vectors [42]. The angle of atom arrangement along the tube, also called as chiral angle or roll-up vector or chirality vector in a single wall CNT (SWCNT), is represented by an integer pair (n, m) . The value of (n, m) determines if CNT is metallic or semiconducting.

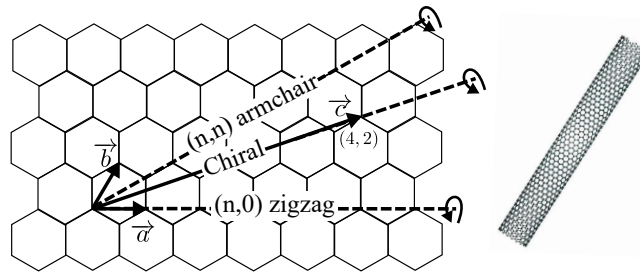


Figure 2.1: Unrolled sheet of graphite and the rolled lattice structure of CNT [42]

SWCNT is further classified into three groups, depending on the angle of atom arrangement, i.e. chirality vector, along which the CNT is rolled. The three groups of CNT are named as armchair CNT if CNT has $n = m$, zigzag CNT if $n = 0$ or $m = 0$ and chiral CNT if m and n are different and nonzero. All armchair CNTs behave as conductors. On the other hand, zigzag and chiral CNTs show metallic (conducting) behavior when $n = m$ or $n - m = 3i$, where i is an integer, otherwise they show semiconducting behavior. Hence zigzag and chiral CNTs are used in realizing a CNTFET [43]. The chirality vector (n, m) also sets the diameter of the CNT.

Carbon-Nanotube Field Effect Transistor (CNFET) is a transistor which makes use of semiconducting carbon nanotubes as channel material between two metal electrodes that act as source and drain contacts. The operating principle of CNFET is similar to that of MOS transistors. As shown in Figure 2.2, this three (or four) terminal device consists of a semiconducting nanotube, acting as conducting channel, bridging the source and drain contacts. The device is turned on or off electrostatically via the gate. The drain current is directly proportional to the number of CNTs connected between the source and the drain and their respective diameters [44, 45].

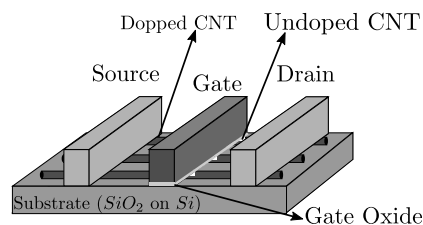


Figure 2.2: 3D view of Carbon-Nanotube Field Effect Transistor (CNFET)

Three types of CNTFET devices have been reported in the literature. They are known as schottky barrier CNTFET (SB-CNTFET), MOSFET-like CNTFET (M-

CNTFET) and band-to-band tunneling CNTFET (T-CNTFET). Due to the similarities of M-CNTFET with MOSFET in terms of operation and intrinsic attributes, it has been used in implementation of logic circuits [45]. The gate width of CNTFET can be approximated using equation below [44]:

$$W \approx \min(W_{min}, N \times S) \quad (2.3)$$

In equation (2.3), W_{min} is the minimum gate width, N is the number of tubes and S the distance between the centers of two adjoining CNTs under the same gate, also called as Pitch. The diameter of CNT, D_{CNT} , which depends on the chirality vector (n, m) can be calculated using equation below:

$$D_{CNT} = \frac{\sqrt{3}a_0}{\pi}(\sqrt{n^2 + m^2 + mn}) \quad (2.4)$$

where $a_0 = 0.142nm$ is the inter atomic distance between each carbon atom and its neighbour. The threshold voltage, which is the voltage needed to turn ON the device electrostatically via the gate, can be approximated to the first order as the half band gap and can be calculated by equation (2.5) [44].

$$V_{th} \approx \frac{E_g}{2e} = \frac{1}{\sqrt{3}} \frac{aV_\pi}{eD_{CNT}} = \frac{0.43}{D_{CNT}(nm)} \quad (2.5)$$

In the above equation, $V_\pi (= 3.033eV)$ is the carbon $\pi - \pi$ bond energy in the tight bonding model, $a (= 0.249nm)$ is the carbon-carbon atom distance and e is the unit electron charge. If the chirality vector of CNT changes then the threshold voltage of the CNTFET will also change. Assuming the m value in the chirality vector is always zero, the ratio of the threshold voltages of two CNTFETs with different chirality vectors can be represented by equation below:

$$\frac{V_{th1}}{V_{th2}} = \frac{D_{CNT2}}{D_{CNT1}} = \frac{n_2}{n_1} \quad (2.6)$$

Equation (2.6) shows that threshold voltage of CNFET is inversely proportional

to the diameter of CNT which, as mentioned above, depends on its chirality vector. It is the threshold voltage controlability of CNFET that makes it well suited for the implementation of multi-valued logic circuits.

The relationship between chirality, CNT diameter and threshold voltage can be derived from relations presented in [44] and is shown in Table 2.3. There have been advances in the manufacturing processes of well controlled CNTs [46, 47]. While techniques exist to synthesize CNFETs of desired chirality [48, 49], those with three chiralities i.e. $(19, 0)$, $(13, 0)$ and $(10, 0)$ are normally used in the implementation of CNFET-based ternary logic circuits [36].

Table 2.3: Relation Between Chirality, CNT Diameter and Threshold Voltage [44]

Chirality	Diameter of CNT	Threshold Voltage of N-CNTFET	Threshold Voltage of P-CNTFET
$(19, 0)$	$1.487nm$	$0.289V$	$-0.289V$
$(17, 0)$	$1.330nm$	$0.328V$	$-0.328V$
$(16, 0)$	$1.253nm$	$0.348V$	$-0.348V$
$(14, 0)$	$1.100nm$	$0.398V$	$-0.398V$
$(13, 0)$	$1.018nm$	$0.428V$	$-0.428V$
$(11, 0)$	$0.861nm$	$0.506V$	$-0.506V$
$(10, 0)$	$0.783nm$	$0.559V$	$-0.559V$

While there are many CNTFET device models in the literature [44, 45, 50–52], Stanford CNFET device models reported in [53] which are based on work presented in [44, 45] have been widely used for the implementation of CNFET-based circuits. Thus this CNFET model is used for simulations in this work. The technology parameters of CNTFET along with their brief description and numeric value are given in Table 2.4.

The characteristics of CNFET and the effect of chirality variations is studied with the help of input and output characteristics of a N-CNFET which are simulated in HSPICE using the CNFET model in [53]. The CNFET is configured to have three CNTs (all with same chirality) and all parameters set to their default value. Figures 2.3 and 2.4 show the output characteristics for a fixed V_{GS} of 0.45 and input characteristics for a fixed V_{DS} of 0.45 for CNFET with different chirality. As seen from Figure 2.3, for a fixed V_{GS} (V_{GS} of 0.45 chosen as an example), the drain current (I_{DS}) is proportional to the diameter of CNTs, which in turn is proportional to value of n in chirality vector.

Table 2.4: Technology Parameters for CNFET model in [44, 45, 53]

Parameter	Description	Value
L_{ch}	Physical channel length	32nm
L_{geff}	Mean free path in the intrinsic CNT channel region	100nm
L_{ss}	Length of doped CNT source-side extension region	32nm
L_{dd}	Length of doped CNT drain-side extension region	32nm
E_{fi}	Fermi level of the doped S/D tube	0.6eV
K_{gate}	Dielectric constant of high-k top gate dielectric material	16
T_{ox}	Thickness of high-k top gate dielectric material	4.0nm
C_{sub}	Coupling capacitance between the channel region and the substrate	40pF/m
V_{fbn} & V_{fbp}	Flat-band voltage for n-CNTFET and p-CNTFET, respectively	0eV, 0eV
$L_channel$	Physical gate length	32nm
$Pitch$	Distance between the centers of two adjacent CNTs	20nm
L_{eff}	Mean free path in p+/n+ doped CNT	15nm
ϕ_{i_M}	Work function of Source/Drain metal contact	4.6eV
ϕ_{i_S}	CNT work function	4.5eV

Also, it is clear from Figure 2.4 that the threshold voltage of CNFET varies with diameter of CNTs which in turn is proportional to value of n in chirality vector. This feature of CNFET makes it well suited for the implementation of multi-valued logic circuits. A review of CNFET-based ternary logic circuits is presented in the next section.

2.3 Ternary Logic Circuits using CNFETs

In ternary logic circuits, transistors with different threshold voltages are required for implementation of basic ternary gates like NTI, PTI, encoder, decoder etc. Unlike in MOS technology, where body biasing is used to control threshold voltages, in CNFET technology the threshold voltage is controlled by changing the diameter (i.e. V_{th} dependent on physical dimension) of CNT which in turn depends on the chirality vector. This dependence makes CNTFET suitable for implementation of MVL circuits. While interest in design of CNFET-based logic circuits waned over recent years due to complex fabrication technology and reliability issues, recent demonstration of a CNFET-based

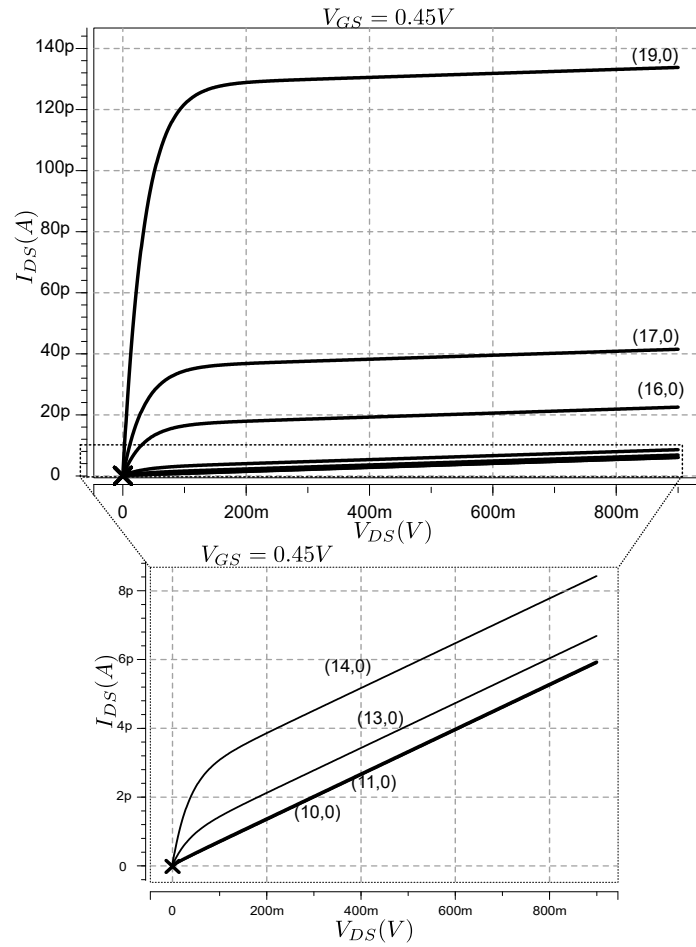


Figure 2.3: I-V Characteristics of N-CNFET

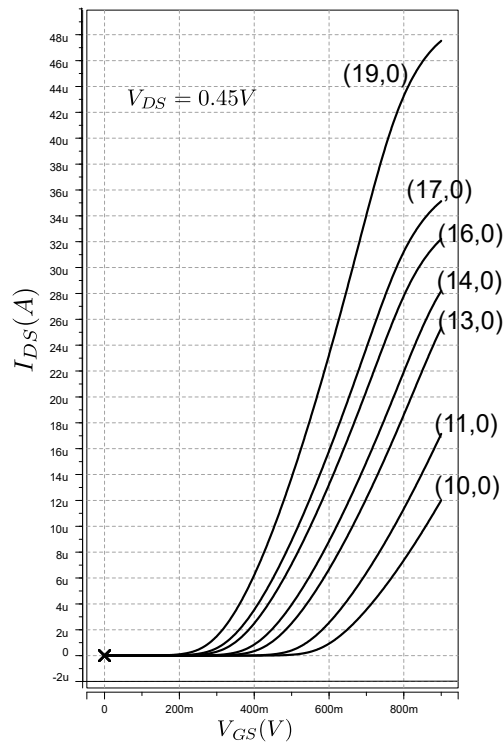


Figure 2.4: I-V Characteristics of N-CNFET

processor/computer by Stanford researchers [54] has reignited the interest.

CNFET-based ternary logic circuits using resistive loads have been presented in [55]. The disadvantage of this approach however is that it needs large off-chip resistances. A more efficient design methodology, which eliminates the need for large resistances by employing an active load with p-type CNFETs, has been presented in [36,37]. This work presented designs for ternary NAND and NOR gates simulated using HSPICE with Stanford CNTFET model of [53]. A design technique which uses both ternary logic gates and binary logic gates can also be found in this work. This technique can be divided in to three main stages: In the first stage, a ternary decoder is used to convert a ternary signal into mutually exclusive unary functions. These decoder outputs can take only two logic values i.e., logic 2 and logic 0, corresponding to logic 1 and logic 0 in binary logic. These outputs are combined using binary logic gates in the second stage. In the third and final stage, the outputs of the second stage are combined using an encoder to generate the ternary outputs. The ternary encoder consists of a level shifter and a ternary OR gate. It was shown that this technique leads to a reduction in power-delay product, for example, a ternary half adder and a 1-bit multiplier with respect to their counterparts designed using CNTFET based ternary gates only.

Another CNFET-based design methodology for ternary and quaternary circuits, which uses pseudo N-type CNFETs, has been presented in [56]. Recently there have been many implementations of CNFET-based ternary arithmetic circuits (Adders [38, 39,57–60] and ALU [40]) that focus on optimizing the design parameters. An improved version of the ternary adder has been presented in [58]. This adder uses an encoder with reduced complexity and a fast carry generation unit resulting in less propagation delay for multi-digit adders. Although the encoder used in [58] has reduced delay and complexity, it consumes large power resulting in designs such as multi-digit adders that consume very high power.

Energy efficient single-digit and multi-digit adders have been presented in [39]. In single-digit adder designs, positive and negative ternary complements of inputs are generated in the first stage. Intermediate outputs are then generated from first stage

outputs and original inputs using a network of transistors with different chiralities. These outputs, which are binary in nature, are converted in to ternary outputs using a transistor-based voltage divider. The multi-digit adder presented in [39] uses two half-adders to generate sum whereas it uses a standalone circuit to generate carry. This carry is ternary in nature and has to propagate to the next adder stage. The major disadvantage of this design however is that at each adder stage, ternary carry is generated using a voltage divider circuit resulting in a multi-digit adder that has large delay and power consumption.

Low-delay and low-power single-digit and multi-digit adders have been presented in [60]. In this work, unary operators are implemented using efficient circuits, which are further used in the design of 3 : 1 multiplexer-based single-digit adders. These adders have low-propagation delay and least PDP when compared to other existing designs. This work also presented efficient conditional sum and carry look-ahead-based ternary multi-digit adders. At each digit-adder stage of Conditional Sum Adder (CSA), three different sums and carry-outs, corresponding to carry-in of 0, 1 and 2 are computed using 3:1 multiplexer-based single-digit adders. Further, depending on the actual carry-in, an array of 3:1 multiplexers is used to compute the final sum digits (Sum_i) and carry-out ($Cout$). In Carry-Lookahead Adder (CSA) presented in [60], four propagate functions (p_i^1, p_i^2, p_i^3 and p_i^4), which correspond to different carry-in and carry-out conditions are generated at each digit-adder stage. These single-digit propagate functions are given as inputs to carry-lookahead generator, which implements a set of logic expressions [60] to compute group propagate functions and carry-out signals. Finally the carry-out signals and inputs are given to a 3 : 1 multiplexer-based single-digit adders, which generate the final sum digits. Although multi-digit CSA and CLA designs have low-propagation delays and least PDP, they have complex carry propagation path and consume large power when compared to other designs.

Ternary multiplier, implemented using unary operators and 3 : 1 multiplexers, has been presented in [61]. This design is based on the classical Wallace multiplier and includes a novel 3 : 1 ternary multiplexer design that requires only a small number of

CNFETs. Two ternary full-adder configurations have also been proposed based on an examination of the multiplier structure. Additionally, the design includes a new single-digit and multi-digit multipliers. Apart from the effort to build novel designs, there have also been attempts to develop synthesis algorithms for CNFET-based ternary logic circuits. Recently, a synthesis technique for ternary logic circuits, which exploits the advantages of CNFET, has been presented in [41]. This technique combines the cube representation [62] and the unary operators [63] to arrive at a 3 : 1 multiplexer based synthesis procedure. This work also presented a procedure for obtaining expressions for two and three variable functions using the unary operators. Ternary functions with three (or more) variables are handled by a decomposition procedure based on work presented in [64].

2.4 Research Gaps, Objectives and Scope of Current Work

Based on the literature review presented above, a few of the gaps and issues in CNFET-based ternary logic design are addressed in this thesis. A summary of these is presented below:

1. Generalized design approaches to implement ternary logic circuits have been limited in the literature so far. Thus, there is a need to develop techniques to design CNFET-based ternary circuits optimized for different design parameters. one of the objectives of this thesis is to develop designs for the implementation of ternary circuits
2. Most of the CNFET-based adder designs in the literature have complex carry propagation paths. In this thesis, two new techniques to implement ternary adders that have efficient carry propagation paths, are proposed. One of these approaches uses binary prefix networks for carry propagation leading to adder designs that have logarithmic delay as apposed to linear delay of the existing designs.

3. Ternary encoder is an important block which is used in many CNFET-based ternary circuits. This block contributes significantly to the overall propagation delay and power consumption of the logic circuits. Hence there is need for low-power low-delay encoder designs, as well as a methodology which uses appropriate encoder designs depending on the design constraints such as power, delay or power-delay product.
4. Automatic synthesis leads to accelerated development of logic circuits. There are synthesis techniques available in literature that can be used to implement ternary circuits, but very few of them take the advantages of special properties of CNFET. In this thesis, we propose novel techniques to synthesize ternary circuits using 2:1 multiplexers.

Chapter 3

Design Approaches for Basic Ternary Circuits

3.1 Introduction

The CNFET based ternary circuit design presented in [36] is a generalized approach which could be used for implementing any ternary function. This approach however uses a complex encoder and requires a ternary decoder for each input, resulting in large area and power consumption for higher operand sizes. Another approach presented in [41] uses multiplexers to implement ternary logic circuits which have low power consumption but large propagation delay. Thus this thesis focuses on developing techniques for designing CNFET-based ternary circuits optimized for multiple design parameters.

In this chapter three general design approaches, which can be used to implement basic ternary logic circuits, are presented. The first approach avoids the use of decoder and uses a novel low-power encoder resulting in ternary circuits with low transistor count compared to existing approaches. This approach results in circuits which are optimized for two design parameters namely area and power consumption. The next approach uses a delay optimized decoder and low-power encoder leading to energy efficient ternary circuits. Finally, the third design approach uses 2:1 multiplexers for

realizing basic ternary logic circuits leading to ternary circuits, which have low power consumption. Basic 2-input circuits are implemented using proposed design approaches and their design parameters are compared to that of existing approaches.

3.2 Existing Design Approaches

3.2.1 Decoder-Encoder based approach

The decoder-encoder based approach presented in [36] can be divided in to three main stages. In first stage, a ternary decoder is used to convert a ternary signal into mutually exclusive unary functions which will have two logic levels, logic 0 and logic 2. The relation between ternary input X and decoder outputs (indicated by X^0, X^1, X^2) is given by

$$X^k = \begin{cases} 2, & \text{if } X = k \\ 0, & \text{if } X \neq k \end{cases} \quad (3.1)$$

These decoder outputs can take only two logic values i.e., logic 2 and logic 0, corresponding to logic 1 and logic 0 in binary logic. The outputs of these ternary decoders are combined using binary logic gates in the second stage. In the third and final stage, the outputs of the second stage are combined using an encoder to generate the ternary outputs. This ternary encoder consists of a level shifter and a ternary OR gate.

As an example, consider the ternary function shown in Table 3.1, where A and B are ternary inputs and F is ternary output. The output function F is equal to logic 2 for the input signals $A = 1$ and $B = 0$ or $A = 2$ and $B = 0$ while it is equal to logic 1 for signal values $A = 0$ and $B = 1$ or $A = 0$ and $B = 2$. F is equal to logic 0 in all the remaining cases. Using Table 3.1 output function F can be written as

Table 3.1: Truth Table (Example 1)

Decimal Equivalent	A	B	F
0	0	0	0
1	0	1	1
2	0	2	1
3	1	0	2
4	1	1	0
5	1	2	0
6	2	0	2
7	2	1	0
8	2	2	0

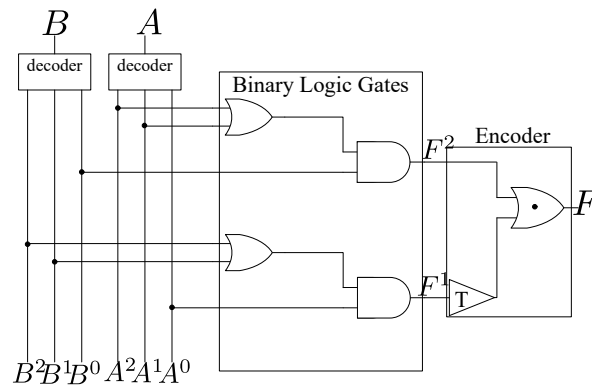


Figure 3.1: Realization of a ternary function using existing decoder-encoder based approach

$$F = 2 \cdot \sum(3, 6) + 1 \cdot \sum(1, 2) + 0 \cdot \sum(0, 4, 5, 7, 8) \quad (3.2)$$

$$F = 2 \cdot (A^1 B^0 + A^2 B^0) + 1 \cdot (A^0 B^1 + A^0 B^2) \quad (3.3)$$

where A^k and B^k ($k = 0, 1, 2$) represent unary outputs of the ternary signals A and B . F^2 and F^1 represent unary signals of output F which are combined in the final stage using an encoder, which consists of a level shifter and a ternary OR gate, to generate ternary output F . Ternary logic gates are usually represented with a \cdot (dot) on the gate. The implementation of the function F following the methodology in [36] is shown in Figure 3.1.

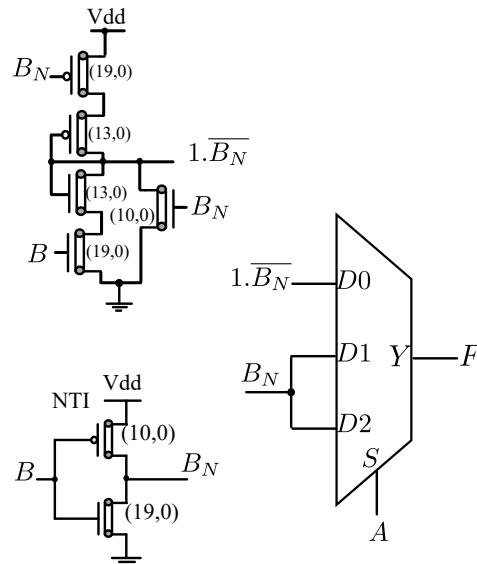


Figure 3.2: Implementation of function shown in Table 3.1 using 3:1 Mux based approach [60]

3.2.2 Multiplexer based approach

Another approach, that uses unary operators and 3 : 1 multiplexers for implementing ternary logic circuits has been presented in [60,61]. To illustrate this approach, consider again the ternary function shown in Table 3.1. Since it has two inputs, one of the inputs is chosen as select line for a 3:1 multiplexer and the other input is used in generation of unary operators. If input A is chosen as the select signal and $A = 0$, then $(0, 1, 2)$ is transformed into $(0, 1, 1)$ with respect to input B . Similarly when $A = 1$ or $A = 2$, $(0, 1, 2)$ is transformed into $(2, 0, 0)$ with respect to input B . These transformations, also called unary operators, have been implemented in such a way that they have low power consumption [60]. Figure 3.2 illustrates the implementation using the 3:1 multiplexers and unary operators. This design requires 18 CNFETs as shown in Figure 3.3.

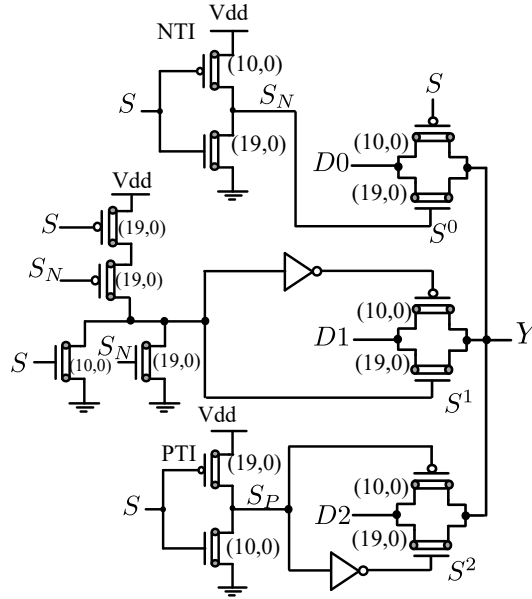


Figure 3.3: Transistor level Implementation of 3:1 Multiplexer [60]

3.3 Novel Approaches to Design Ternary Logic Circuits

3.3.1 Approach I: Using Low-power Encoder and without Decoder

3.3.1.1 Overview

As seen from the existing decoder-encoder based approach, each of the inputs is converted into mutually exclusive binary signals using ternary decoders. As the number of inputs increases, there will be an increase in the number of decoders needed resulting in more area and power consumption. In the example discussed earlier two decoders were needed for two inputs A and B in the implementation of the function F . The approach being proposed obviates the need for a decoder by optimally grouping the terms in equation (3.3). Considering the earlier example, the function F can be represented as

$$F = 2 \cdot ((A^1 + A^2)B^0) + 1 \cdot (A^0(B^1 + B^2)) = 2.F^2 + 1.F^1 \quad (3.4)$$

where F^2 and F^1 are the unary signals of output function F , which can take the values of logic 2 (logic high) or logic 0 (logic low). In the above equations, $F^2 = 2$, if

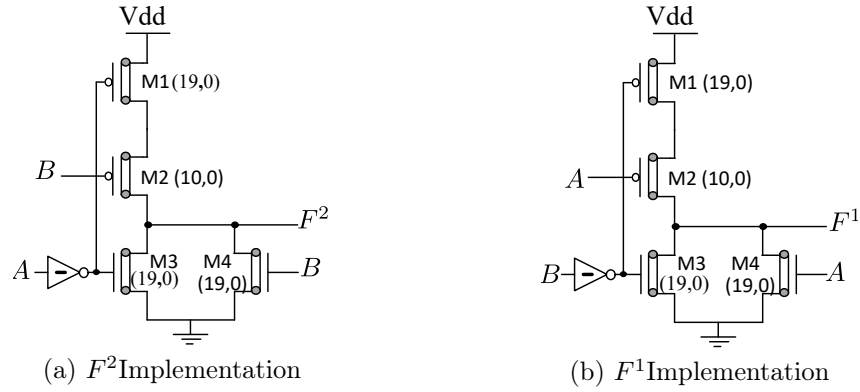


Figure 3.4: Implementation of F^2 and F^1 in proposed approach

$B^0 = 2$ and $(A^1 + A^2) = 2$. The term $B^0 = 2$, if $B = 0$. The other term of F^2 i.e. $(A^1 + A^2) = 2$, if $A = 1$ or 2 . Thus the complete term $(A^1 + A^2)B^0$ of the function F^2 can be realized using NTI gates and CNFET transistors, as shown in Figure 3.4(a).

It consists of one NTI gate and four additional transistors M1, M2, M3 and M4. M1 and M2 are p-type CNFETs, whereas M3 and M4 are of n-type. The chiralities of the respective transistors are also shown in the figure. Accordingly, M1, M2, M3 and M4 transistors have threshold voltages equivalent to $-0.428V$, $-0.559V$, $0.428V$ and $0.289V$ respectively. Input A is given through an NTI gate and connected to base of M1 and M3. Since M1 has a threshold voltage of $-0.428V$, M1 is *ON* only when the output of NTI is logic 0 (i.e. when A is logic 1 or logic 2). M3 has a threshold voltage of $0.428V$ and thus M3 is *ON* only when output of NTI gate is logic 2 (i.e. when A is logic 0). Similarly, the base of transistors M2 and M4 are connected to input B . Since M2 and M4 have their threshold voltages as $-0.559V$ and $0.289V$ respectively, M2 is *ON* when B is logic 0 and M4 is *ON* when B is logic 1 or logic 2. The overall output F^2 is logic 2 when both M1 and M2 are *ON* and F^2 is logic 0 when either M3 or M4 are *ON*. Similar analysis can be carried out for the circuit implementation of F^1 shown in Figure 3.4(b).

As seen from the above example, the use of decoder circuit for each ternary input can be avoided using the proposed approach. This results in less number of transistors as well as less delay and power consumption when compared to the existing decoder-encoder approach. The above analysis can be described more formally as below:

Proposition 3.1. *Without the loss of generality, let $F(A, B)$ be any ternary function with A and B as inputs. If it is possible to represent the unary terms of the inputs A and B using any combination of transistors, PTI and NTI gates, then the use a decoder for any input (at the initial stage) is redundant.*

3.3.1.2 Steps Involved

Based on the above Proposition 3.1 an approach for implementation of CNFET based ternary circuits without decoder is presented below.

1. Initially, a given function F is represented using a K -map appropriate equations are determined.
2. Equations can then be used to determine the components (such as NTI and PTI, etc.) required for CNFET based implementation. Also, the number of transistors required for implementation of unary signals of F , that is, F^2 , F^1 and F^0 are determined. Since unary signals are mutually exclusive, implementation of any two out of three functions is enough. The selection of functions is made in such a way that least number of transistors is required for the implementation.
3. Based on the unary functions chosen, an optimized encoder circuit is used to generate the final ternary output.

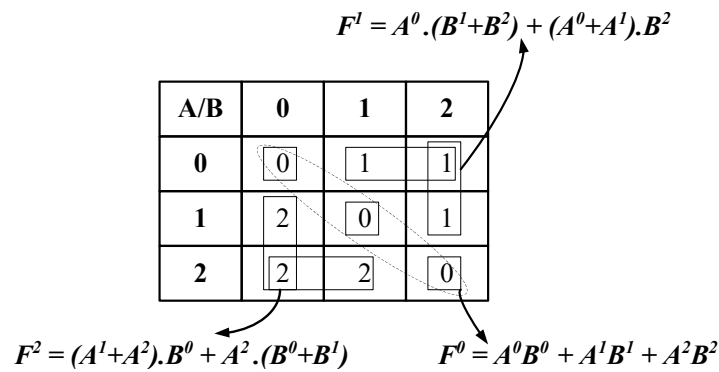
In what follows, the above steps are described in detail.

3.3.1.3 Function Simplification

The given function F is first represented using K -map and the equations in terms of unary functions. The terms in the K -map can then be grouped or the equations can be simplified such that the terms in final simplified equation can be realized using the ternary inputs directly. This is achieved by using a combination of NTI gates, binary inverters and transistors. As an example, consider a function $F(A, B)$ represented by the truth table shown in Table 3.2. It can be observed from the table that, the output is logic 0 when $A = B$, is logic 1 when $A < B$ and logic 2 when $A > B$. This function

Table 3.2: Truth Table (Example 2)

Decimal Equivalent	A	B	F
0	0	0	0
1	0	1	1
2	0	2	1
3	1	0	2
4	1	1	0
5	1	2	1
6	2	0	2
7	2	1	2
8	2	2	0

Figure 3.5: K -Map simplification for function F

can be represented using the K -map as shown in Fig 3.5. To get the unary functions corresponding to three levels, the 1s, 2s and 0s are grouped separately. This results in minimized functions for F^2 , F^1 and F^0 . The grouping of terms can be carried out using the K -map shown in Fig 3.5.

The functions can also be derived using the simplification of equation as illustrated below:

$$F = 2 \cdot \sum(3, 6, 7) + 1 \cdot \sum(1, 2, 5) + 0 \cdot \sum(0, 4, 8) \quad (3.5)$$

which can be expressed as

$$F = 2 \cdot (A^1 B^0 + A^2 B^0 + A^2 B^1) + 1 \cdot (A^0 B^1 + A^0 B^2 + A^1 B^2) + 0 \cdot (A^0 B^0 + A^1 B^1 + A^2 B^2) \quad (3.6)$$

or

$$F = 2 \cdot (A^1 + A^2)B^0 + A^2(B^0 + B^1) + 1 \cdot (A^0(B^1 + B^2) + B^2(A^0 + A^1)) + 0 \cdot (A^0B^0 + A^1B^1 + A^2B^2) \quad (3.7)$$

resulting in

$$F = 2 \cdot F^2 + 1 \cdot F^1 + 0 \cdot F^0 \quad (3.8)$$

3.3.1.4 Implementation of Unary Functions

Functions F^2 , F^1 and F^0 in equation (3.8) represent the unary functions of output F and are mutually exclusive. Thus any two of the three unary functions are enough to generate the final output F . As mentioned earlier, the two functions that are to be implemented are chosen in such a way that realization of the circuit results in least number of transistors. After this simplification is carried out, the resulting expressions would contain either individual unary terms of the inputs, i.e. A^0 , B^0 , A^1 etc. or group terms such as $(A^1 + A^2)$, $(B^0 + B^1)$, etc. These terms can be realized using PTI gates, NTI gates and transistors.

For example, A^0 indicates that the output function is logic 2 when input A is logic 0 i.e. a connection to VDD is made possible when A is logic 0. This pull-up functionality is realized by designing a p-type CNFET with a threshold voltage of $-0.559V$ i.e. a chirality equivalent to $(10, 0)$. Such a transistor will however be *OFF* when gate voltage is greater than $0.341V$ and hence for logic 1 ($0.45V$) and logic 2 ($0.9V$), it will be in *OFF* state. The values of input for which the output remains logic 0 are considered for designing the n-type CNFET structure. In case of A^0 , the output remains logic 0 when the input is at logic 1 or logic 2. This pull-down functionality is realized with an n-type CNFET having a threshold voltage of $0.289V$ which is equivalent to a chirality of $(19, 0)$. Such a transistor will be *ON* when gate voltage is greater than $0.289V$ and hence for logic 1 ($0.45V$) and logic 2 ($0.9V$), it remains in *ON* state. A similar analysis can be carried out to design p-type and n-type structures for different terms of the

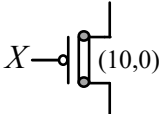
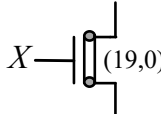
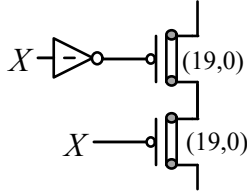
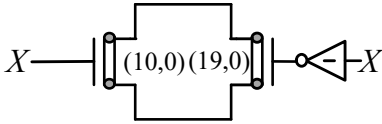
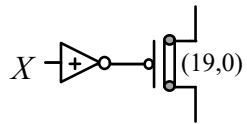
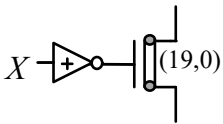
Function (Transistors Required)	P-Type CNFET	N-Type CNFET
X^0 (2)		
X^1 (6)		
X^2 (4)		

Figure 3.6: Transistor level realization of X^0 , X^1 and X^2 unary terms

simplified equation. Figures 3.6 and 3.7 show the realization of different unary terms (individual and group) that might appear in any simplified equation corresponding to input X . To realize the output unary functions, the p-type CNFET structures corresponding to the input unary terms are placed in series to realize AND (\cdot) function and in parallel to realize OR ($+$) function. On the otherhand n-type CNFET structures are placed in series to realize OR ($+$) function and in parallel to realize the AND (\cdot) function.

Figures 3.6 and 3.7 also show the number of transistors required to implement the individual and group unary terms corresponding to an input X . This information is used to calculate the number of transistors required to implement the unary functions F^2 , F^1 and F^0 as shown in Figure 3.8.

The simplified function of F^2 contains $(A^1 + A^2)B^0$ as the first term and thus the p-type structure corresponding to B^0 is placed in series with that of $(A^1 + A^2)$. The resulting circuit is then placed in series with the structure corresponding to $A^2(B^0 + B^1)$ to get the final p-type CNFET circuit. A process similar to complementary logic style is followed to design the n-type circuit resulting in the design shown in Figure 3.8. The realization of all unary functions F^2 , F^1 and F^0 results in 12, 12 and 24 transistors respectively.

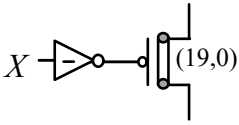
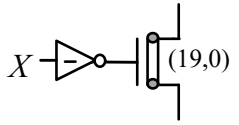
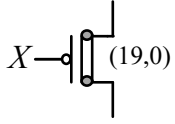
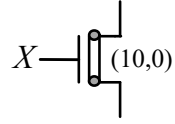
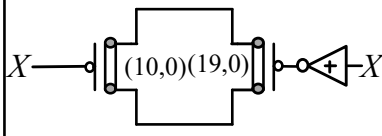
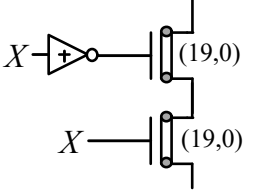
Function (Transistors Required)	P-Type CNFET	N-Type CNFET
$X^1 + X^2$ (4)		
$X^0 + X^1$ (2)		
$X^0 + X^2$ (6)		

Figure 3.7: Transistor level realization of $X^0 + X^1$, $X^1 + X^2$ and $X^0 + X^2$ unary terms

3.3.1.5 Low-power Encoder

In the final stage, an encoder is used to combine the unary signals resulting in a final ternary output getting generated. An encoder generates logic 2 and logic 0 respectively by a direct connection to VDD and GND . But for generation of *logic 1* at output, a direct path from VDD to GND is created. One of the major disadvantages of the existing ternary adder designs [39, 58] is that they use encoders which have low resistance path between VDD and GND to generate *logic 1*. This results in a large static current and hence large static power consumption. The encoder design in existing approach [36] (shown in Figure 3.1) uses a level shifter and a ternary OR gate. Hence it requires large number of transistors and also has large power consumption. The power consumption is mainly because of multiple direct paths, that exist from VDD to GND , while generating logic 1. An improved encoder, which has lower power consumption when compared to encoders in [36, 39, 58], is presented in this section. Figure 3.9 shows the proposed encoder, which has unary functions F^2 and F^1 as inputs. Since the functions F^2 and F^1 are unary function and can be either logic 0 or logic 2, simple binary inverters can be used to get their complements. Implementation of a simple binary inverter is also shown in Figure 3.9.

Existing encoder and the encoder shown in Figure 3.9 use F^1 and F^2 as inputs to

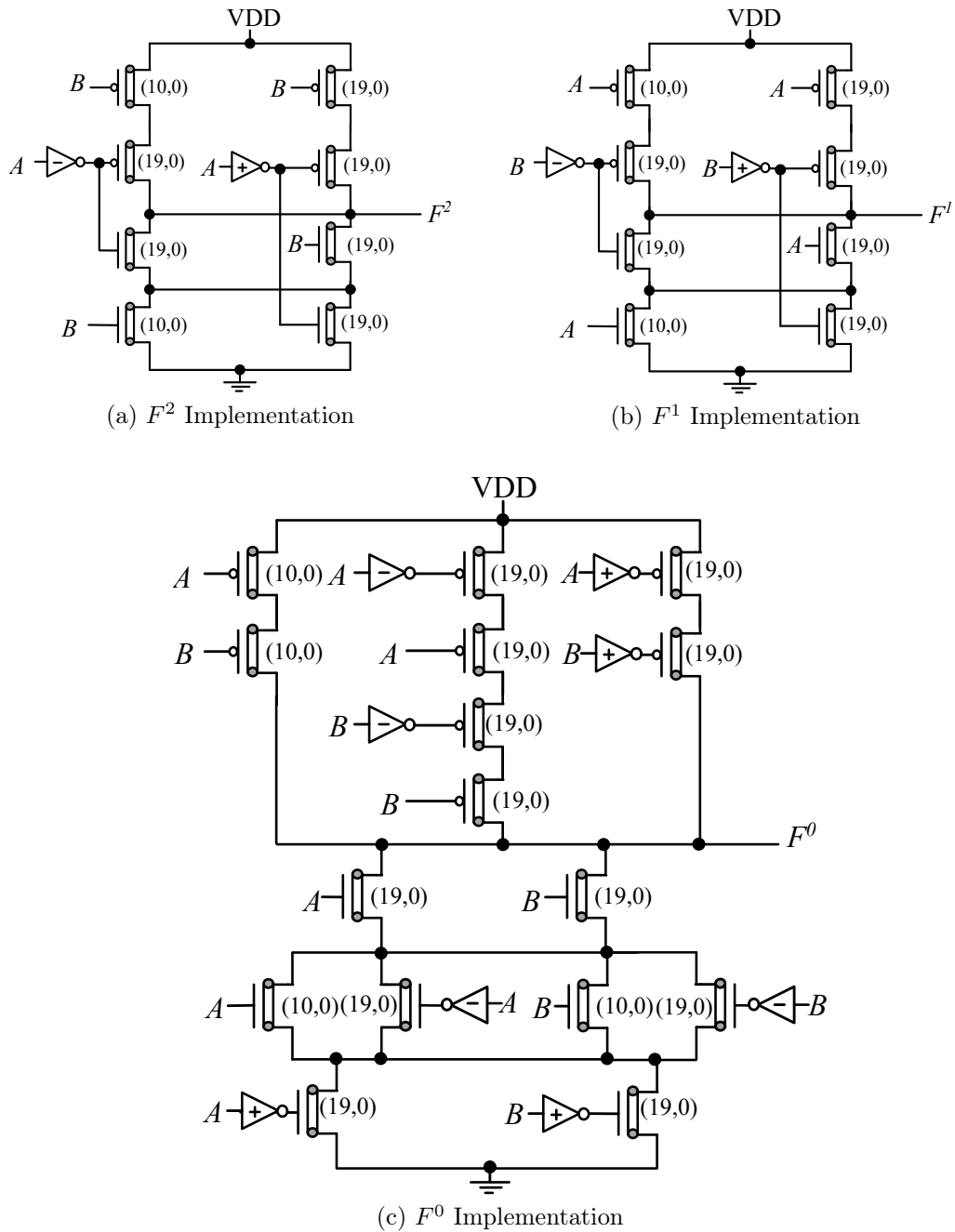


Figure 3.8: Implementation of F^2 , F^1 and F^0 using proposed approach

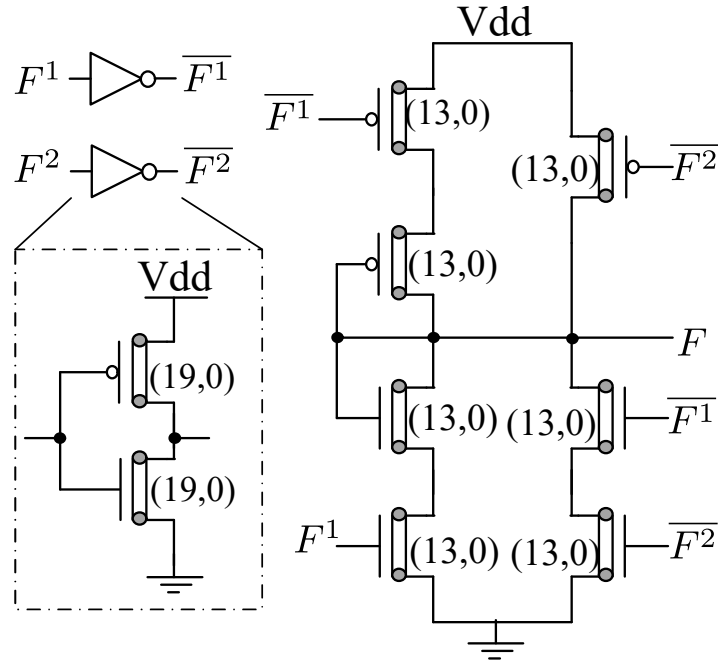
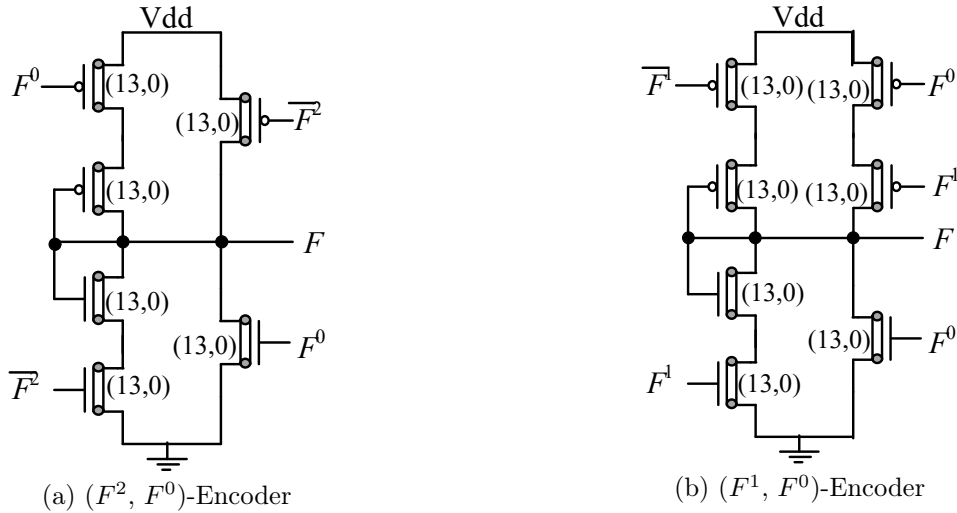


Figure 3.9: Encoder with F^2 and F^1 as inputs ((F^2, F^1) -Encoder)

generate ternary output F . Hence, if any other pair of functions, i.e. F^2 and F^0 or F^1 and F^0 , are implemented to reduce the number of transistors as explained in Section 3.3.1.4, F^1 or F^2 have to be generated using binary NOR gate. To avoid this, two more encoders are proposed that take combinations of (F^2, F^0) or (F^1, F^0) as inputs. Figure 3.10 shows encoder designs for the function pairs (F^1, F^0) and (F^2, F^0) . As in the case of (F^2, F^1) -encoder, simple binary inverters can be used to get the complement of unary functions for encoders presented. For the example under consideration (equation 3.7), implementation of F^2 and F^1 results in least number of transistors and hence the encoder design presented in Figure 3.9 is used in the final stage. As seen from the proposed approach any ternary function can be realized without decoders resulting in reduced area. Although the proposed approach is explained using an example of a 2-input function, the same methodology can be extended to any number of inputs. However, care should be taken to avoid stacking of multiple transistors.

Figure 3.10: Encoder design for (F^2, F^0) and (F^1, F^0) combination

3.3.2 Approach II: Using low-delay Decoder and low-power Encoder

The previous section presented a decoderless approach to design ternary logic circuits. Although this approach results in reduced area due to the elimination of decoders, it results in large transistor stacking leading to higher propagation delay for some circuits. To illustrate this, consider an example ternary function shown in Table 3.3 which computes the Half-sum (HS) of two ternary digits A and B .

Table 3.3: Half-Sum (HS)

A/B	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Based on Table 3.3 the logical expressions for half-adder outputs can be determined as below:

$$HS^2 = A^2B^0 + A^1B^1 + A^0B^2 \quad (3.9)$$

$$HS^1 = A^2B^2 + A^1B^0 + A^0B^1 \quad (3.10)$$

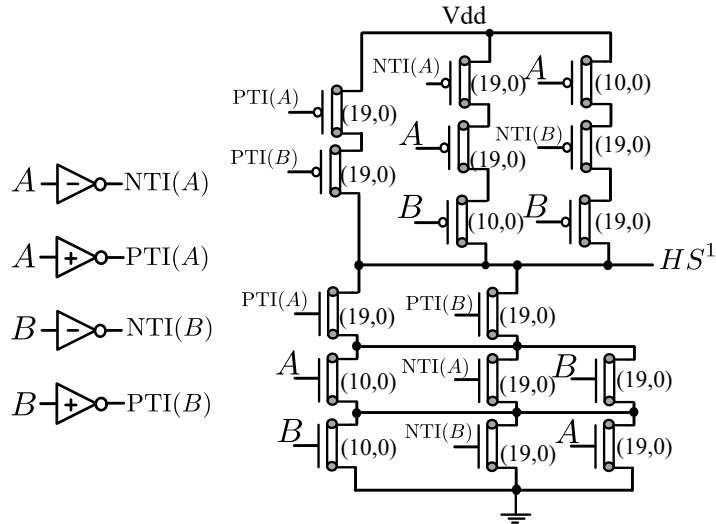


Figure 3.11: Implementation of Half-Sum using Decoderless approach

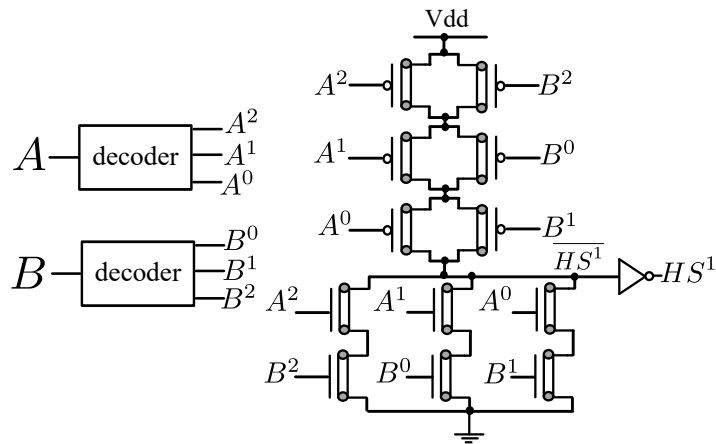


Figure 3.12: Implementation of Half-Sum using Existing Decoder-encoder based approach (all CNFETs have chirality as (19, 0))

$$HS^0 = A^2B^1 + A^1B^2 + A^0B^0 \quad (3.11)$$

Since HS^2 , HS^1 and HS^0 are mutually exclusive, circuit implementations are required only for two of these signals. The third signal can be then generated using *NOR* operation. For example, if HS^2 , HS^1 are available then HS^0 can be generated as $\overline{HS^2 + HS^1}$. The transistor level implementation of HS^1 using the decoderless approach is shown in the Figure 3.11. Alternatively, assuming a decoder is used, HS_i^1 can be directly implemented at transistor-level using complementary logic style as shown in Figure 3.12.

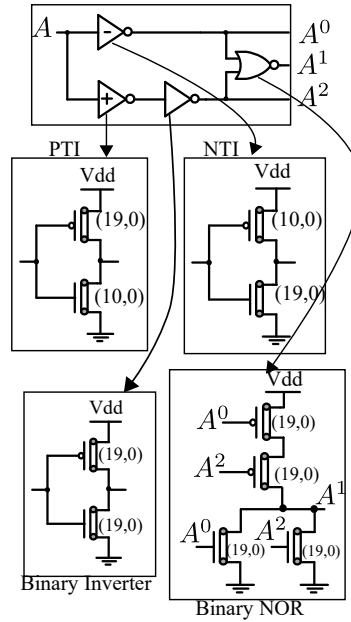
In both these cases, the circuits have increased transistor stacking (three transistors) for pull-up/pull-down network as evident from Figure 3.12 and 3.11. Similarly implementations of HS^2 and HS^0 also result in circuits with large transistor stacking. Thus in this section, we propose an implementation with reduced transistor stacking, leading to lower delay, by exploiting the mutually exclusive property of the decoder outputs. This section also presents a low-delay decoder which aids further in reducing the delay of the ternary circuit.

3.3.2.1 Low-Delay Decoder

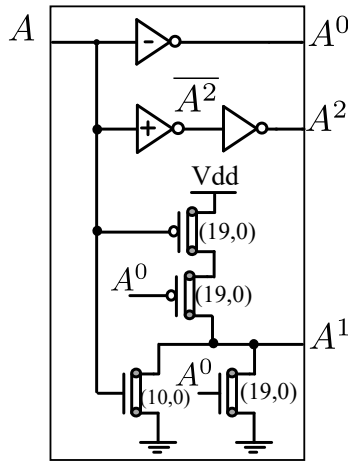
The approach presented in this section uses a decoder to generate mutually exclusive binary signals (as in equation 3.1). The existing design of the decoder [36], [58], which uses NTI, PTI, binary inverter and binary NOR gate, is shown in Figure 3.13(a). This design, which has two inverters and one binary *NOR* gate in its critical path, can be improved further to achieve better delay. Figure 3.13(b) shows the delay-optimized design for the decoder. In this design, a NOR like structure is used to generate A^1 using A^0 and ternary input A instead of A^0 and A^2 . Ternary input A is connected to N-CNFET which switches ON when A is logic 2. A is also connected to P-CNFET which switches ON when A is either logic 0 or logic 1. As evident from the circuit, A^1 will be logic 2 when both the transistors in the pull up network of *NOR* like structure are ON. This happens only when A^0 is logic 0 and A is logic 1. The critical path of this decoder, which includes one inverter and a *NOR* like structure, is better than the existing decoder implementation [36, 58].

3.3.2.2 Alternate Representation of Logic Expressions

The basic idea behind this approach is to modify the logic equations for a given ternary function so that it results in reduced transistor stacking. This is illustrated with the ternary function considered earlier, which computes Half-Sum (HS) of two ternary digits, A and B . The logic expressions for half-sum are represented by equations (3.9) - (3.11). A direct implementation of these equations leads to increased transistor



(a) Existing Decoder [36, 58]



(b) Proposed Decoder

Figure 3.13: Decoder Implementations

stacking as seen in Figures 3.11 and 3.12. Hence instead of using equations (3.9) - (3.11) directly, they are modified to product-of-sum form using properties of unary operators to optimize the pull-up/pull-down network. The product-of-sum expression for HS^2 can be derived either using the K-map or by representing HS^2 as $\overline{HS^1 + HS^0}$, where HS^1 and HS^0 are given by equations (3.10) and (3.11) respectively. The modification of logical expression for HS^2 is shown below:

$$HS^2 = \overline{HS^1 + HS^0} \tag{3.12}$$

$$HS^2 = \overline{A^2B^2 + A^1B^0 + A^0B^1 + A^2B^1 + A^1B^2 + A^0B^0} \quad (3.13)$$

$$HS^2 = \overline{A^2(B^2 + B^1) + A^1(B^0 + B^2) + A^0(B^1 + B^0)} \quad (3.14)$$

$$HS^2 = \overline{A^2\overline{B^0} + A^1\overline{B^1} + A^0\overline{B^2}} \quad (3.15)$$

$$HS^2 = (A^2 + \overline{B^0})(A^1 + \overline{B^1})(A^0 + \overline{B^2}) \quad (3.16)$$

A similar process is used to modify the expressions for HS^1 and HS^0 , resulting in the expressions below:

$$HS^1 = (A^2 + \overline{B^2})(A^1 + \overline{B^0})(A^0 + \overline{B^1}) \quad (3.17)$$

$$HS^0 = (A^2 + \overline{B^1})(A^1 + \overline{B^2})(A^0 + \overline{B^0}) \quad (3.18)$$

Equations (3.9), (3.10) and (3.11) which are in product-of-sum form, represent the modified logical expressions of equations (3.16), (3.17) and (3.18) which are in sum-of-product form, respectively. For optimal implementation of pull-up/pull-down network both product-of-sum and sum-of-product expressions are used appropriately. For example, equation (3.10) can be used to implement the pull-down network and an equivalent equation (3.17) can be used to implement the pull-up network of HS_i^1 , leading to lower transistor stacking. Figure 3.14 shows the transistor-level implementations to generate unary signals for the half-sum output.

The proposed implementation of half-adder needs $\overline{B_i^2}$, $\overline{B_i^1}$ and $\overline{B_i^0}$, which can be generated using binary inverters along with the proposed decoder. Hence the decoder circuit for B_i has two inverters and one binary *NOR* like structure in the critical path which is similar to the existing decoder circuit presented in [36, 58]. However, our approach results in a balanced pull-up and pull-down network with reduced transistor

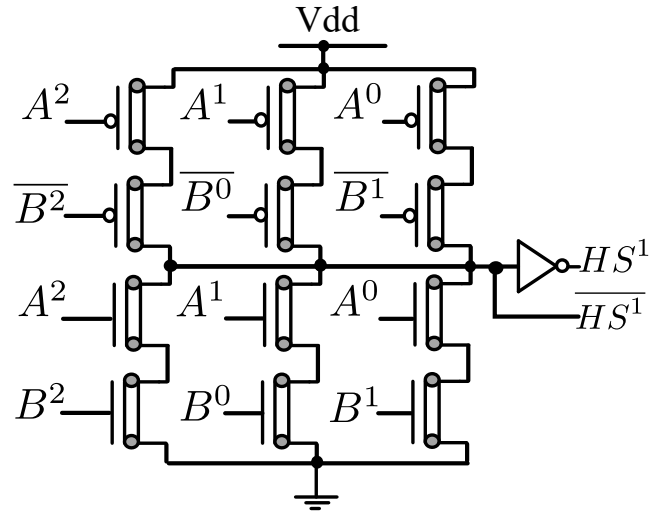


Figure 3.14: Implementation of Half-Sum with reduced Transistor Stacking (all CNFETs have chirality as $(19, 0)$)

stacking when compared to existing decoder-encoder based approach and the decoder-less approach which is presented in Section 3.3.1. After generating the unary functions of outputs, the encoder design presented in Section 3.3.1.5 is used to generate final ternary output.

The approach presented in this section uses a delay optimized decoder and alternate expressions for logic expression to implement basic ternary logic circuits which have low delay when compared to existing design approaches. However, this approach requires more number of CNFETs when compared to decoderless approach presented in Section 3.3.1.

3.3.3 Approach III: Using 2:1 Multiplexers

In this section, an approach which uses 2:1 multiplexers for implementation of ternary logic circuits, is presented.

3.3.3.1 Basic Idea

The basic idea involved in this approach is presented using Proposition given below:

Proposition 3.2. *A 3:1 multiplexer can be implemented using two 2:1 multiplexers*

Proof. Consider the expression for a 3:1 multiplexer (shown in Figure 3.2), which is

represented as $Y = S^0 \cdot D_0 + S^1 \cdot D_1 + S^2 \cdot D_2$, where D_0 , D_1 and D_2 represent inputs while S represents select signal. S^0 , S^1 and S^2 are related to S according to equation (3.1) and can be generated as shown in the Figure 3.3.

$$Y = S^0 \cdot D_0 + S^1 \cdot D_1 + S^2 \cdot D_2 \quad (3.19)$$

$$Y = S^0 \cdot D_0 + (S^1 + S^0) \cdot (S^1 + S^2) \cdot D_1 + S^2 \cdot (S^1 + S^2) \cdot D_2 \quad (3.20)$$

$$\because S^1 = (S^1 + S^0) \cdot (S^1 + S^2), S^2 \cdot S^2 = S^2, \text{ and } S^2 \cdot S^1 = 0$$

$$Y = S^0 \cdot D_0 + (S^1 + S^2) \cdot ((S^1 + S^0) \cdot D_1 + S^2 \cdot D_2) \quad (3.21)$$

$$Y = S^0 \cdot D_0 + \overline{S^0} \cdot (\overline{S^2} \cdot D_1 + S^2 \cdot D_2) \quad (3.22)$$

$\because (S^1 + S^2) = \overline{S^0}$, $(S^1 + S^0) = \overline{S^2}$, where $\overline{S^0}$, $\overline{S^1}$ and $\overline{S^2}$ represent the binary NOT of signals S^0 , S^1 and S^2 respectively as given by equation (3.23).

$$\overline{S^k} = \begin{cases} 2 & \text{if } S^k = 0 \\ 0 & \text{if } S^k = 2 \end{cases} \quad (3.23)$$

Alternatively equation (3.19) can also be represented as equation (3.24) and (3.25).

$$Y = \overline{S^2} \cdot (S^0 \cdot D_0 + \overline{S^0} \cdot D_1) + S^2 \cdot D_2 \quad (3.24)$$

$$Y = \overline{S^1} \cdot (S^0 \cdot D_0 + \overline{S^0} \cdot D_2) + S^1 \cdot D_1 \quad (3.25)$$

The relation in equations (3.22), (3.24) and (3.25) are similar to relation for a 2:1 multiplexer, $Y = \overline{S} \cdot D_0 + S \cdot D_1$, where D_0 and D_1 are inputs and S is the select line. Hence a 3:1 multiplexer can be implemented using two 2:1 multiplexers as shown in the Fig 3.15. \square

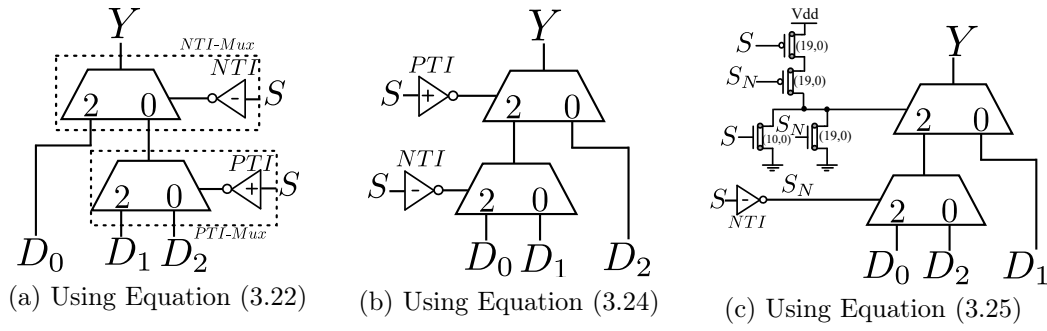


Figure 3.15: A 3:1 Multiplexer operation using 2:1 Multiplexers

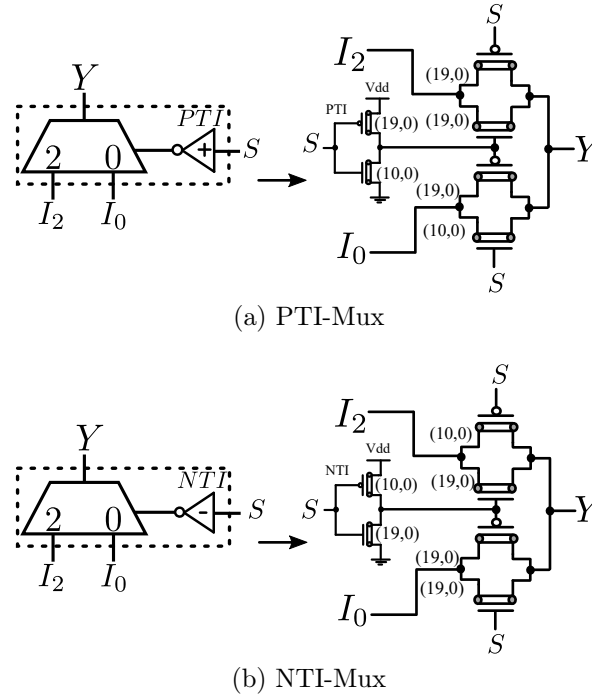


Figure 3.16: CNFET-based Implementation of NTI-Mux and PTI-Mux

3.3.3.2 Ternary circuits using CNFET-based 2:1 Multiplexers

As evident from Figure 3.15, equations (3.22) and (3.24) are less complex to implement, when compared to equation (3.25) because generation of S^0 , $\overline{S^0}$, S^2 and $\overline{S^2}$ requires PTI and NTI whereas implementation of S^1 and $\overline{S^1}$ requires complex *NOR* like structure in addition to a NTI. Hence circuits shown in Figures 3.15(a) and 3.15(b) are used for implementing ternary functions. These circuits use two types of 2:1 multiplexers namely PTI-Mux and NTI-Mux, which are implemented using CNFETs as shown in Figures 3.16(a) and 3.16(b) respectively.

A 3:1 multiplexer presented in [41] requires 18 transistors. However, proposed 2:1 multiplexers with inverters (NTI-Mux and PTI-Mux), which are equivalent to one 3:1

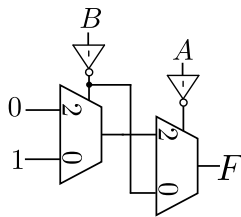


Figure 3.17: 2:1 Multiplexer based Implementation for Ternary Function in Table 3.4

multiplexer, requires only 12 transistors. Further PTI-Mux and NTI-Mux are used in the implementation of ternary logic circuits. To illustrate 2:1 multiplexer based approach consider the same example ternary function represented in Table 3.1, which was used to show 3:1 multiplexer based approach. The K-map for this ternary function is shown in Table 3.4.

Table 3.4: Ternary Function (Example 1)

A/B	0	1	2
0	0	1	1
1	2	0	0
2	2	0	0

For this ternary function A and B are ternary inputs and F is ternary output. In 2:1 multiplexer based approach, similar to 3:1 multiplexer based approach, one of the inputs is chosen as the select line and second input is used to realize the unary operators. But unlike in 3:1 multiplexer based approach, the proposed approach implements the unary operators using 2:1 multiplexers, PTI and NTI. Figure 3.17 shows the implementation of ternary function shown in Table 3.4, where A is chosen as the select line and unary operator is realized using 2:1 multiplexer. This implementation requires 12 CNFETs when compared to 3:1 multiplexer based implementation, which requires 25 transistors. Since both decoders and encoders are avoided in this approach, it results in low area and power consumption when compared to design approaches presented earlier. However, Since this approach uses transmission gates for realizing multiplexers, it results in ternary circuits with large propagation delay.

Table 3.5: Truth-Table for basic Ternary Circuits

Inputs		Half-Adder		1-digit Multiplier	
<i>A</i>	<i>B</i>	<i>Sum</i>	<i>Carry</i>	<i>Product</i>	<i>Carry</i>
0	0	0	0	0	0
0	1	1	0	0	0
0	2	2	0	0	0
1	0	1	0	0	0
1	1	2	0	1	0
1	2	0	1	2	0
2	0	2	0	0	0
2	1	0	1	2	0
2	2	1	1	1	1

3.4 Implementation and Simulation

For a comparison of the proposed approaches with the existing ones [36, 60], basic ternary logic functions namely half-adder and 1-digit multiplier have been implemented using different approaches and circuit parameters like delay, power, and number of transistors have been compared to understand relative performance. The functionality of the basic circuits is represented by Table 3.5.

As seen from this table, half-adder has two outputs (*Sum* and *Carry*) and 1-digit multiplier has two outputs (*Product* and *Carry*). As an example, the circuit implementations for Half-adder using different approaches is shown in Figures 3.18, 3.19, 3.20, 3.21 and 3.22. These circuits along with the circuits for multiplier have been implemented using CNTFET model available at [53] and simulated using HSPICE. The following sub-sections explain the simulation environment and the results obtained.

3.4.1 Simulation Environment

All the circuits are simulated in HSPICE using the CNTFET model of [44, 45, 53] at 0.9V power supply and room temperature. The CNFETs used in the implementation are configured to have three tubes and a default pitch value equal to 20nm. All the other parameters are set to their default values as presented in Table 2.4. In this work, ternary logic values 0, 1 and 2 correspond to voltages 0, $V_{dd}/2$ and V_{dd} respectively. For binary logic gates the logic values 0 and 1 correspond to voltages 0 and V_{dd}

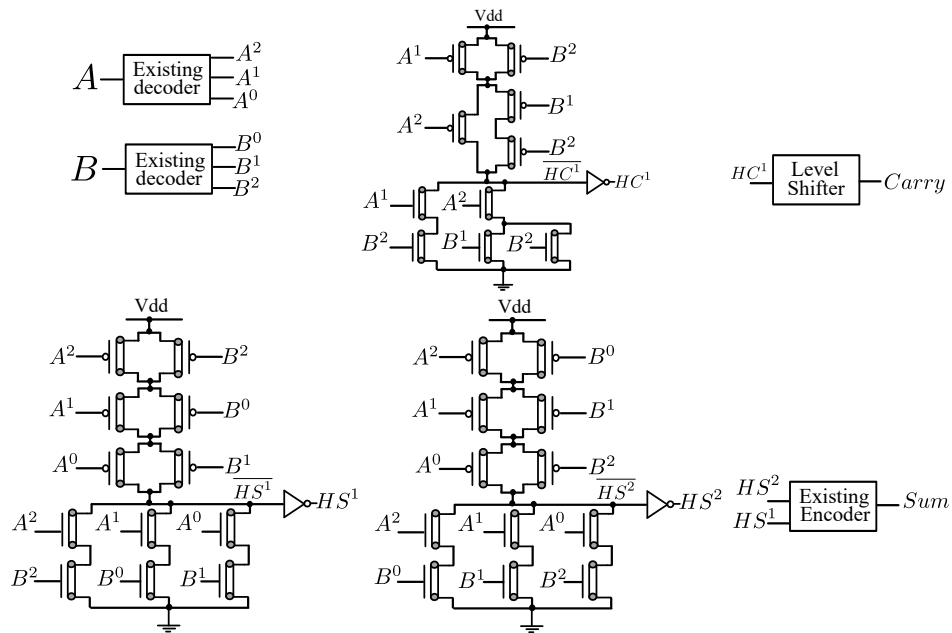


Figure 3.18: Half-Adder using Existing Decoder-Encoder based Approach (all CNFETs have chirality as (19, 0))

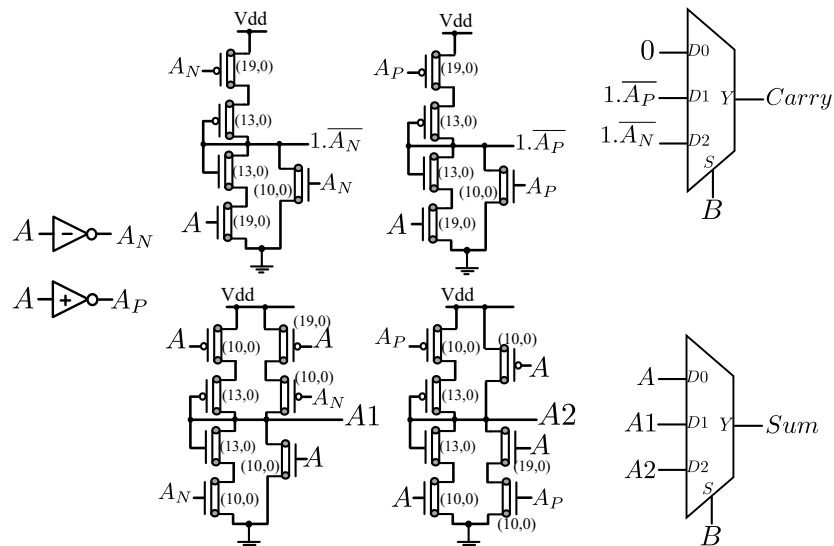


Figure 3.19: Half-Adder using Existing 3:1 Multiplexer based Approach

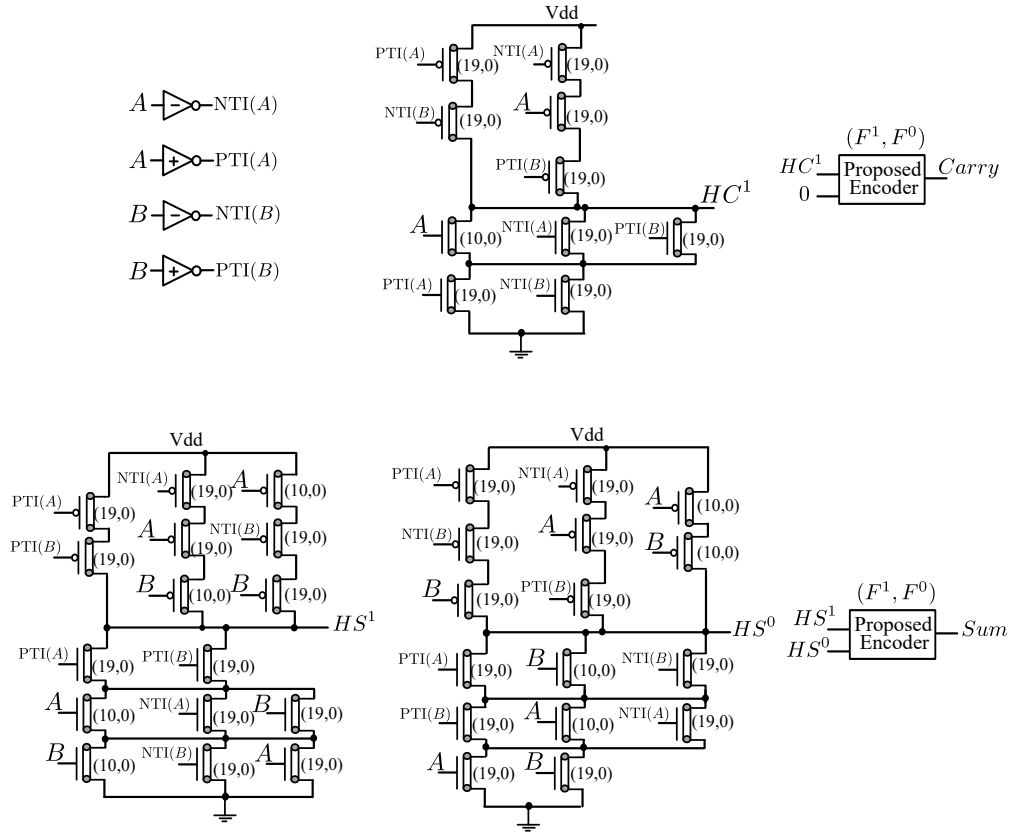


Figure 3.20: Half-Adder using Proposed Decoderless Approach (I)

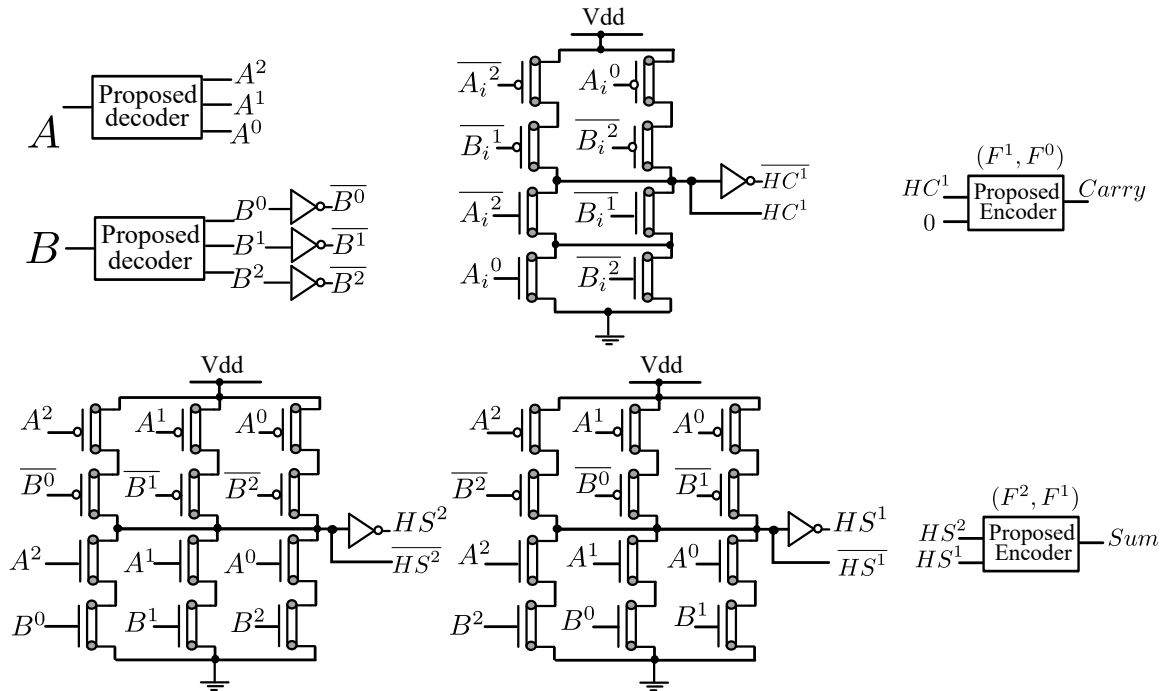


Figure 3.21: Half-Adder using Proposed Decoder-Encoder based Approach (II) (all CNFETs have chirality as (19, 0))

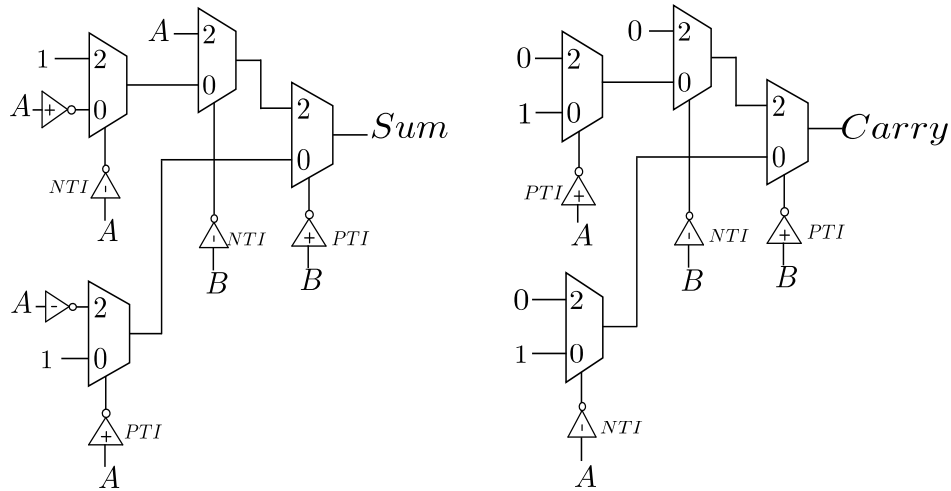


Figure 3.22: Half-Adder using Proposed 2:1 Multiplexer based Approach (III)

respectively. Binary gates are implemented using transistors, which are connected in complementary logic style and have chirality of $(19, 0)$. Different ternary circuits are implemented using proposed and existing approaches and the design parameters are compared. For fair comparison, all the circuits have been simulated with same test pattern. Power consumption results are obtained by simulating the circuits with random input patterns at switching frequency of $500MHz$. Propagation delay results for different circuits are obtained by finding the worst case Fan-Out of 4 (FO4) delay of the critical path. FO4 delay is calculated by loading the output node with four STI gates (implementation of STI gate is presented in [36]). In this section Proposed Approach I, II and III refer to decoderless, decoder-encoder based and 2:1 multiplexer based approaches respectively.

3.4.2 Results and Discussion

Table 3.6 summarizes the simulation results of different parameters for various ternary functions namely ternary half-adder and multiplier, which are implemented using existing and proposed approaches. In this table, Approach I, II and III refer to proposed decoderless, decoder-encoder based and 2:1 multiplexer based approaches respectively. These approaches are compared with existing decoder-encoder based approach [36] and 3:1 multiplexer based approach [41, 60]. Fig. 3.23 presents the logical simulation waveforms of half-adder implemented using 2:1 multiplexer based approach to show

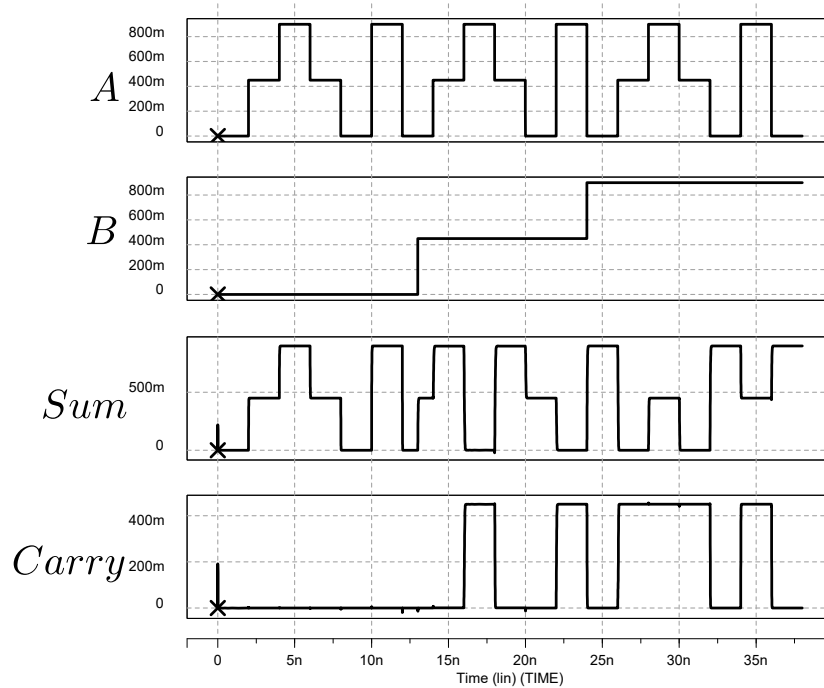


Figure 3.23: Simulation Waveforms for Half-Adder

the correctness of the design. The same is also true with other proposed designs.

The existing decoder-encoder based approach uses a decoder for each input and a complex encoder for each ternary output resulting in large propagation delay, power consumption and transistor count when compared to other approaches. The problem of increased transistor count is addressed in existing 3:1 multiplexer based approach [41,60]. But the disadvantage of this approach is that it uses a complex implementation for realizing unary operators resulting in large power consumption, when compared to decoder-encoder based approach in [36].

Approach I uses a decoderless implementation and a low-power encoder to realize ternary logic circuits. This implementation shows a reduction of up to 18% in transistor count and up to 40% in power consumption when compared to existing approach in [36]. Approach I also shows reduction in propagation delay (up to 7%) and power consumption (up to 54%), but requires more transistors (up to 38% more) for implementation when compared to 3:1 multiplexer based approach [41,60]. However, for some circuits like ternary half-adder, the decoderless approach results in circuits with large transistor stacking leading to higher propagation delay. This problem is solved in Approach II by using both sum-of-product and product-of-sum expressions of ternary

Table 3.6: Simulation Results for Basic Circuits

Design Approach	Half-Adder	1-digit Multiplier
Power Consumption (in μW)		
Decoder-Encoder Based [36]	1.24 (100%)	0.780 (100%)
3:1 Multiplexer based [41, 60]	1.62(131%)	0.779(100%)
Approach I	0.745(60%)	0.372(48%)
Approach II	0.828(67%)	0.417(53%)
Approach III	0.121(10%)	0.069(9%)
Propagation (FO4) Delay (in ps)		
Decoder-Encoder Based [36]	42.4(100%)	35.7(100%)
3:1 Multiplexer based [41, 60]	47.0(111%)	26.4(74%)
Approach I	43.0(101%)	25.3(71%)
Approach II	31.61(74%)	19.5(55%)
Approach III	44.2(104%)	17.5(49%)
Power-Delay Product (PDP) (in $\times 10^{-17} J$)		
Decoder-Encoder Based [36]	5.25(100%)	2.78(100%)
3:1 Multiplexer based [41, 60]	7.62(145%)	2.06(74%)
Approach I	3.20(61%)	0.94(34%)
Approach II	2.62(50%)	0.81(29%)
Approach III	0.53(10%)	0.12(4%)
Number of CNFETs		
Decoder-Encoder Based [36]	88(100%)	66(100%)
3:1 Multiplexer based [41, 60]	52(59%)	40(61%)
Approach I	72(82%)	50(76%)
Approach II	82(93%)	62(94%)
Approach III	36(41%)	30(45%)

circuits appropriately to reduce transistor stacking.

Approach II also uses a optimized decoder and low power encoder resulting in reduction upto 7% in transistor count, 45% in propagation delay and 47% in power consumption when compared to existing decoder-encoder based approach in [36]. This approach also shows improvement with respect to propagation delay and power consumption when compared to existing 3:1 multiplexer based approach. However, this approach uses decoder and hence requires more number of transistors to implement the circuit when compared to decoderless approach.

Approach III uses 2:1 multiplexer in the implementation of ternary logic circuits and resulting circuits show a reduction of up to 30% transistor, 34% in propagation delay and 91% power consumption when compared to existing 3:1 multiplexer based approach [41, 60]. This is because, unlike in the existing 3:1 multiplexer based approach, where unary operators are implemented using complex circuits that have multiple direct paths between VDD and GND , in the proposed approach, unary operators are implemented using 2:1 multiplexers resulting in low power consumption. The 2:1 multiplexer based approach has least power consumption and transistor count when compared to existing and other proposed approaches. However, since 2:1 multiplexer based approach uses transmission gates, ternary circuits designed using this approach cannot drive large load capacitance.

To check the dependence propagation delay on load capacitance, ternary logic circuits designed using different approaches have been simulated with varying loads. Table 3.7 shows the propagation delay results for ternary circuits with load capacitance of $1fF$, $2fF$ and $3fF$. As an example Figure 3.24 shows a graph of variation in propagation delay of a half-adder with respect to changes in the output load.

As seen from this figure, the propagation delay is heavily dependent on load for the half-adder circuit designed using existing and proposed multiplexer based approaches. These approaches result in circuits with large propagation delay for large load. For ternary half-adders designed using other existing and proposed approaches, the propagation delay is less dependent on load variations.

Table 3.7: Propagation Delay vs Load

Propagation Delay (in ps) for Half-Adder			
Design Approach	Load Capacitance		
	$1fF$	$2fF$	$3fF$
Decoder-Encoder Based [36]	65.15	92.36	119.5
3:1 Multiplexer based [41, 60]	120.1	226.8	333.4
Approach I	66.68	101.6	137.3
Approach II	54.26	89.8	123.9
Approach III	110.7	201.1	291.2
Propagation Delay (in ps) for 1-bit Multiplier			
Decoder-Encoder Based [36]	58.38	85.69	113.1
3:1 Multiplexer based [41, 60]	74.70	138.5	202.8
Approach I	34.44	59.40	85.90
Approach II	37.34	62.37	88.94
Approach III	47.79	88.53	129.5

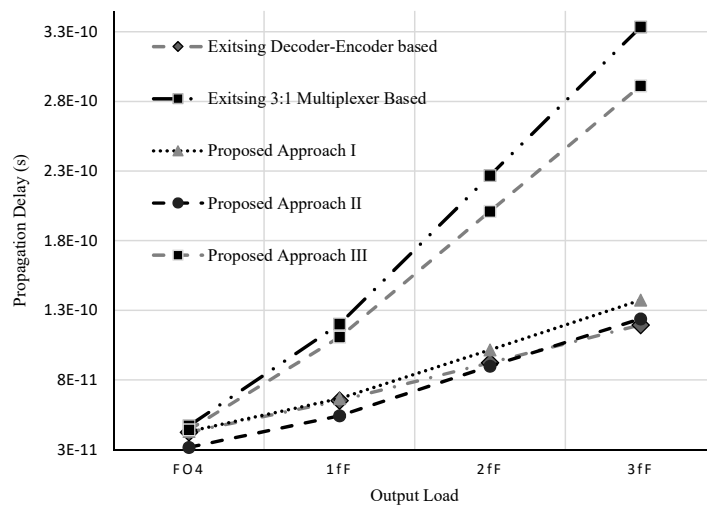


Figure 3.24: Propagation Delay Vs Output Load for Half-adder

As seen from Table 3.6 and Table 3.7, a ternary half-adder implemented using proposed Approach II has least propagation delay and lower power consumption when compared to existing approaches. Although, this half-adder consumes more power, it is less dependent on load variations when compared to half-adder implemented using 2:1 multiplexer based approach.

3.4.2.1 Impact of Process, Voltage and Temperature (PVT) Variations

The performance metrics such as delay and power consumption are impacted by variations in process parameters, voltage and temperature (PVT). Thus in this section, we present the results of variation in delay and power due to PVT variations for half-adders which are implemented using proposed and existing design approaches. Diameter variation is a significant issue in CNFET-based ternary logic circuits since they consist of CNFETs with different CNT diameter. To examine the effect of diameter variation on the performance of the adder, Monte Carlo simulations have been performed with upto $\pm 15\%$ Gaussian distribution, variations at $\pm 3\sigma$ level and 30 iterations for each simulation. The significance of 30 iterations has been discussed in [65]. Figure 3.25 illustrates the variations in delay and power in the existing and proposed full adders as a function of diameter variation. It is clear from the figure that the half-adder implemented using the Approach I (i.e. decoderless) is least sensitive to diameter variations when compared to those implemented using other existing and proposed approaches. The proposed and existing decoder-encoder based approaches show similar variation in delay and power with respect to variation in diameter. Figure 3.25 also shows that multiplexer based approaches are more sensitive to diameter variations when compared to decoder-encoder based approaches.

Another aspect of interest in CNFET-based circuits is the impact of voltage variations. Figure 3.26 illustrates the propagation delay and power consumption in full adders for different supply voltages. The multiplexer based approaches are excluded from this analysis because The increase in the supply voltage increases the current drawn by the circuit. As the current through CNFET increases the propagation delay

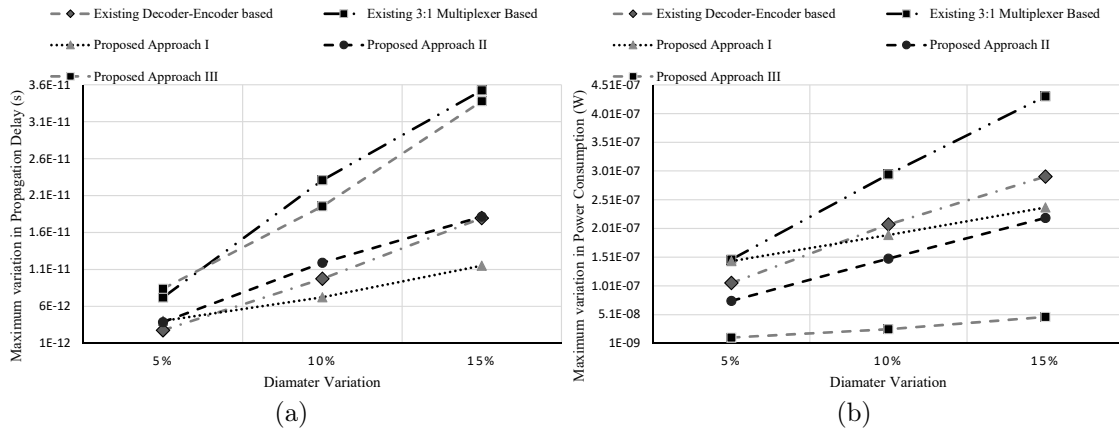


Figure 3.25: Monte-Carlo Simulations for (a) Delay (b) Power

decreases while the power consumption increases. It is clear from the Figure 3.26 that the half-adders implemented using the proposed and existing approaches show similar variation in design metrics for variation in supply voltage.

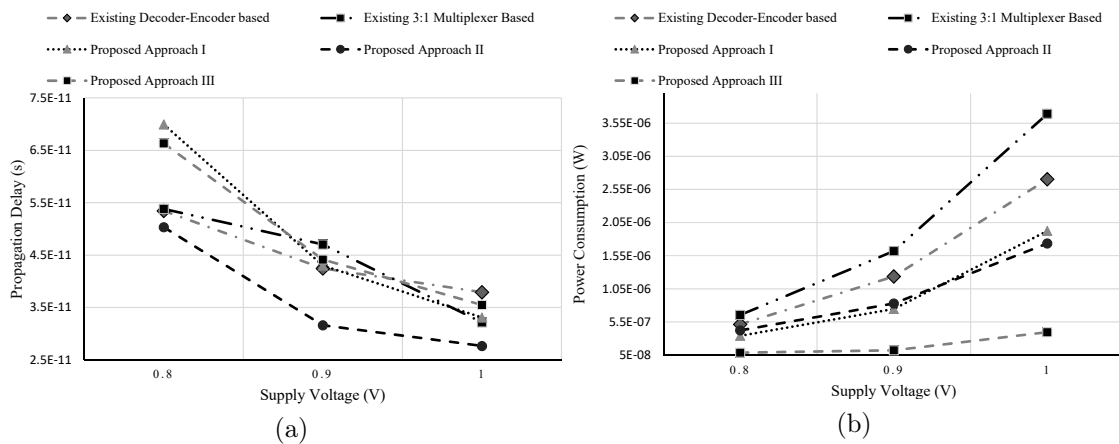


Figure 3.26: (a) Delay (b) Power for Supply Voltage Variation

Apart from process and voltage, temperature variations also impact the performance of CNFET-based circuits. Figure 3.27 illustrates the propagation delay and power consumption in full adders for different temperatures. The increase in temperature increases the current and thus propagation delay is directly proportional while power consumption is inversely proportional to temperature. It is clear from the Figure 3.27 that the half-adders implemented using the proposed and existing approaches show similar variation in design metrics for variation in temperature.

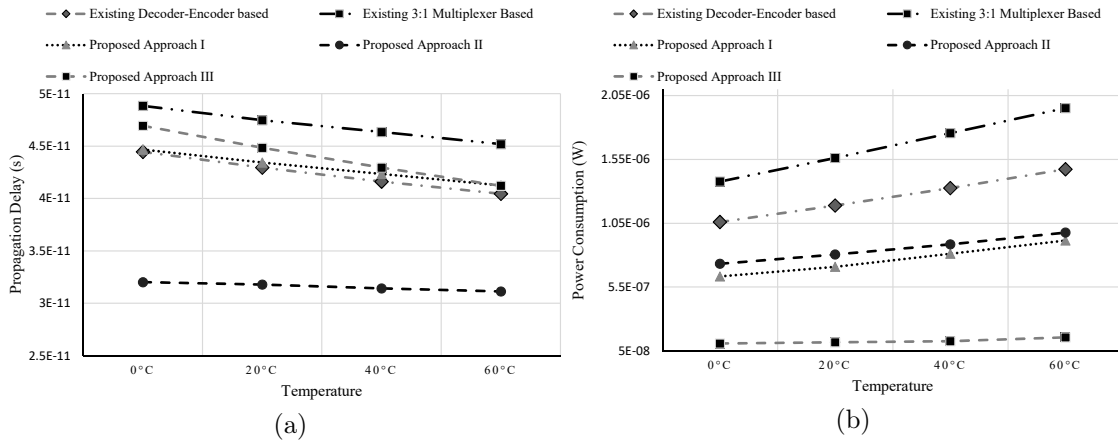


Figure 3.27: (a) Delay (b) Power for Temperature Variation

3.4.2.2 Noise Immunity Analysis

Noise tolerance is a term commonly used to describe the ability of logic circuits to function properly in the presence of noise pulses. Noise pulses are characterized by their width and amplitude and a pulse with adequate width and amplitude may cause a glitch (spurious switching). Noise immunity curve (NIC) is typically used to measure the noise tolerance of a logic gate in presence of such pulses. The horizontal and vertical axes in the NIC curve correspond to noise width (T_{noise}) and noise amplitude (V_{noise}), respectively and any point lying above the curve is indicative of a glitch at the output. Noise immunity curves have been obtained through simulations using techniques described in [65, 66]. Figure 3.28 presents the noise immunity curves for half adders implemented using proposed and existing approaches. It is clear from this figure that the half-adder implemented using proposed decoderless approach (Approach II) shows better noise immunity when compared to those implemented using decoder-encoder based and multiplexers based approaches. The ternary half-adder implemented using proposed decoder-encoder and multiplexer based approaches have noise immunity characteristics similar to half-adder implemented using existing decoder-encoder and multiplexer based approaches respectively.

Another quantitative measure, known as, average noise threshold energy (ANTE) derived from NIC curve is also used to quantify the noise immunity. It is equal to $E(V_{noise}^2 \times T_{noise})$, where $E()$ denotes the expectation operator. A higher ANTE measure

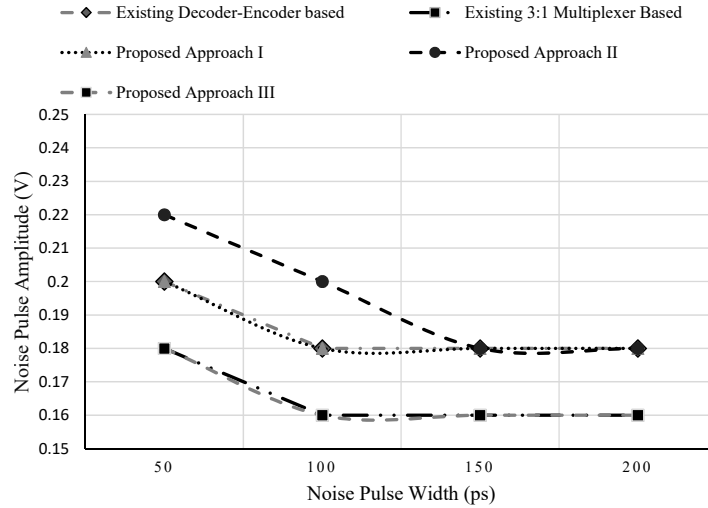


Figure 3.28: Noise Immunity Curve for Half-Adders

Table 3.8: ANTE for Ternary Half-Adders

Design Approach	ANTE (in V^2-ps)
Decoder-Encoder Based [36]	4.15
3:1 Multiplexer based [41, 60]	3.29
Approach I	4.44
Approach II	4.15
Approach III	3.29

implies better immunity to input noise. The ANTE for the different half-adders are shown in Table 3.8. It is clear from the table that the proposed decoderless approach has better noise immunity when compared to decoder-encoder and multiplexer based approaches.

3.5 Conclusions

This chapter presented three general design approaches, which can be used to implement basic ternary logic circuits. These approaches lead to ternary circuits which are optimized for one more design parameters. Simulation results indicate that basic ternary circuits, namely half-adder and 1-digit multiplier, which have been implemented using proposed approaches result in up to 91% reduction in power consumption, up to 45% reduction in delay and 30% reduction in transistor count when compared to the existing design approaches. Based on the analysis of simulation results the following conclusions are drawn:

-
- Approach I, which uses a low-power encoder and does not use a decoder, leads to efficient ternary logic circuits with respect to transistor count, power consumption and drive capability. This approach also leads to circuits which are less sensitive to diameter variations and have better noise immunity.
 - Approach II, which uses delay optimized decoder, and uses both sum-of-product and product-of-sum expressions, leads to implementation of ternary logic circuits which have least propagation delay, and are less dependent on load variations.
 - Approach III, which uses 2:1 multiplexers for implementing ternary circuits, leads to circuits which are optimized for power consumption and transistor count, but are heavily dependent on output load.

Chapter 4

Design of Multi-digit Ternary Adders

4.1 Introduction

In Chapter 3, three new design approaches to design ternary logic circuits have been presented. Simulation results show that a ternary half-adder, which is implemented using Approach II, has least propagation delay, and is less dependent on load variations. Hence in this chapter, half-adder implemented using Approach II is used in the implementation of ternary adders.

Adder is the basic building block of Arithmetic and Logical Unit (ALU), which in turn is an integral part of any general purpose processor. Adders are also used in many hardware implementations like logarithmic converters, address generators etc. There have been many implementations of CNFET-based adder circuits [38, 39, 57–60] that focus on optimizing the design parameters. Most of the multi-digit ternary adders in literature are ripple-carry based and/or have complex carry propagation paths leading to large delay. The adder designs presented in [38, 39, 58, 59] also consume large power due to the complexity involved in generating logic 1.

This chapter presents two new techniques that can be used to implement multi-digit ternary adders. In the first technique, the ternary adders are implemented using half-adders (implemented using Approach II), which generate Half-Sum (*HS*) and Half-Carry (*HC*). These half-adder outputs (instead of main inputs) are used to compute carry-out at each digit-adder stage using a delay optimized carry generator. The half-

sum and carry-out are then used to compute final sum at each digit-adder stage with the help of a sum generator and low-power encoders. Employing delay optimized carry generator along with low-power encoder results in energy efficient multi-digit ternary adder design. The second technique is based on the concept of carry Propagate-Generate, which used in implementation of ternary prefix adders. The carry propagate-generate concept cannot be directly applied in the implementation of ternary prefix adders. In this work a technique, which enables the use of carry Propagate-Generate concept in multi-digit ternary adders, is presented.

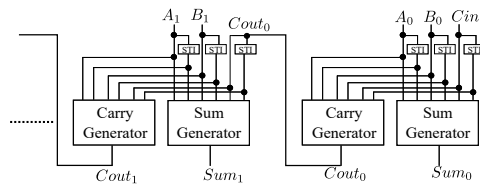
4.2 Previous Work on CNFET-based Ternary Adders

There have been many CNFET-based ternary adder designs proposed in the literature. A brief overview of these adders is presented in the following subsections.

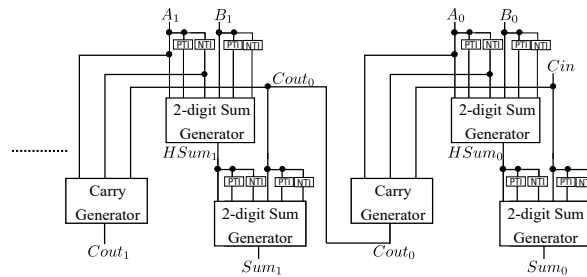
4.2.1 Single-Digit Adders

The ternary adder presented in [36] uses a ternary decoder in the first stage to generate binary versions of inputs. The ternary decoder is a one-input, three-output circuit and generates unary functions for an input X . The relation between ternary input X and decoder outputs (indicated by X^0, X^1, X^2) is given by equation (3.1). These decoder outputs can take only two logic values i.e., *logic 2* and *logic 0*, corresponding to *logic 1* and *logic 0* in binary logic. The decoder outputs are used to compute intermediate binary outputs with the help of binary logic gates. Finally ternary sum and carry are generated from binary outputs using a combination of ternary buffers and/or encoder. A modified and improved version of the above adder has been presented in [58]. This adder uses an encoder with reduced complexity and a fast carry generation unit resulting in less propagation delay for multi-digit adders. Although the encoder used in [58] has reduced delay and complexity, it consumes large power resulting in multi-digit adders with very high power consumption.

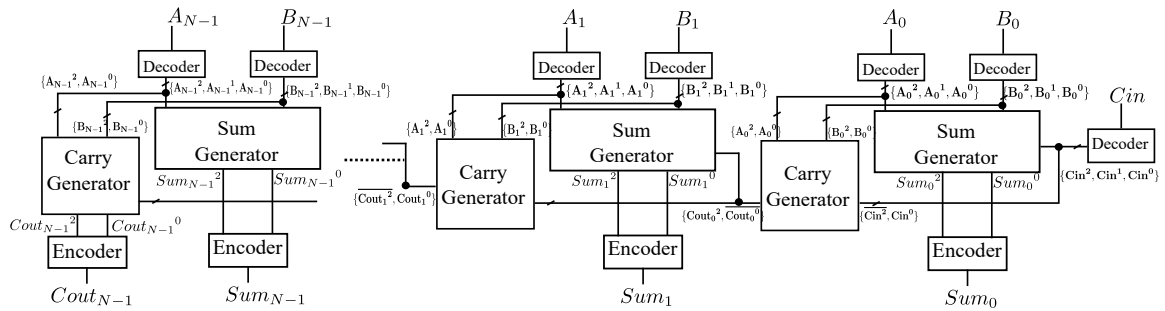
A ternary single-digit adder (full adder), which does not use decoders, has been



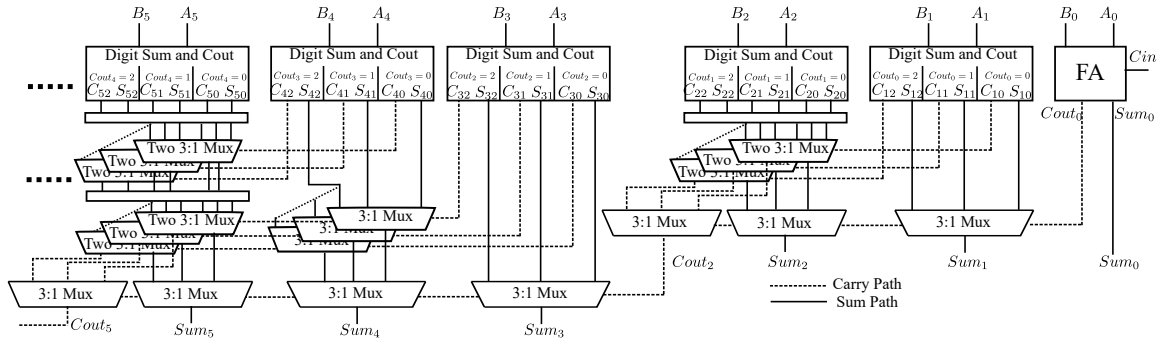
(a) Multi-digit adder design based on Single-digit adder of [38].



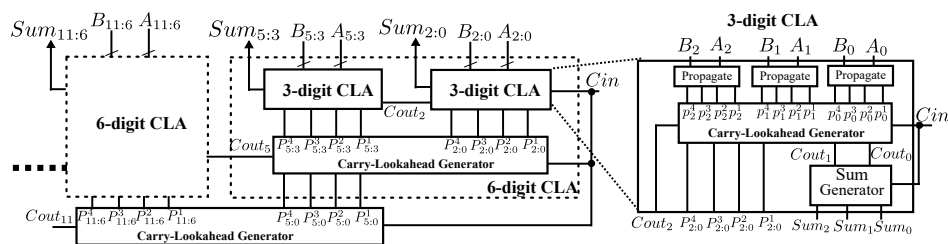
(b) Multi-digit adder presented in [39].



(c) Multi-digit adder presented in [58].



(d) Multi-digit Conditional Sum Adder (CSA) presented in [60]



(e) Multi-digit carry look-ahead Adder (CLA) presented in [60]

Figure 4.1: Design techniques used to implement Multi-digit adders

presented in [38]. In this design, inputs and their standard ternary complements are given as inputs to the sum generator and carry generator blocks. These blocks, which compute ternary sum and carry, are implemented using a network of transistors with different chiralities. This design avoids using a separate encoder and hence has least power consumption when compared to other designs. The disadvantage of this design is that it has large propagation delay as a result of complex sum generator and carry generator blocks. A modified version of this adder has been presented in [59] with lesser number of transistors. This design has ternary decoder in first stage and uses a symmetrical pull up and pull down network with voltage divider to generate ternary sum and carry.

Energy efficient single-digit and multi-digit adders have been presented in [39]. In single-digit adder design, positive and negative ternary complements of inputs are generated in the first stage. Intermediate outputs are then generated from first stage outputs and original inputs using a network of transistors with different chiralities. The intermediate outputs, which are binary in nature, are converted in to ternary outputs using a transistor-based voltage divider. Single-digit adder presented in [39] have moderate power consumption and PDP when compared to other existing designs.

A low-delay and low-power single-digit and multi-digit adders have been presented in [60]. In this work, unary operators are implemented using efficient circuits, which are further used in the design of 3 : 1 multiplexer-based single-digit adders. These adders have low-propagation delay and least PDP when compared to other existing designs.

4.2.2 Multi-digit Adders

Figure 4.1 illustrates design techniques available in literature that are used to implement multi-digit (N -digit) adders. Here $A(A_{N-1}...A_1A_0)$, $B(B_{N-1}...B_1B_0)$, Cin are ternary inputs and $Sum(Sum_{N-1}...Sum_1Sum_0)$, $Cout$ are ternary outputs. $Cout_{N-2}...Cout_0$ represent intermediate ternary carries. Only single-digit adder design is presented in [38], which can be connected in series to implement multi-digit adder (shown

in Figure 4.1(a)). A similar technique can be used to implement multi-digit adder based on one-digit adder presented in [59].

The multi-digit adder presented in [39] is shown in Figure 4.1(b). This design uses two half-adders to generate sum, whereas it uses a standalone circuit to generate carry. This carry is ternary in nature and has to propagate to the next adder stage. The major disadvantage of designs in [38, 39, 59] is that at each adder stage, ternary carry is generated using a voltage divider circuit resulting in a multi-digit adder that has large delay and power consumption.

Design presented in [58] is shown in Figure 4.1(c), where intermediate binary signals are represented using X_i^j notation. Here X_i^j corresponds to i^{th} digit-adder stage whose value is either *logic 2* (if $X = j$) or *logic 0* (if $X \neq j$), where $j \in \{0, 1, 2\}$. For example A_0^1 corresponds to input of 0^{th} digit-adder stage whose value is *logic 2* only if ternary signal A is equal to *logic 1*. Also, $\{\}$ is used to represent a group of binary signals and its complements. In multi-digit adder of the same work, the sum generation is similar to other existing designs, whereas carry generation/propagation is optimized for delay by avoiding redundant encoder-decoder pairs. At each digit-adder stage, binary carry signals (i.e. $Cout_i^2, Cout_i^0$) are generated using a low-delay carry generator block. Unlike other multi-digit adder designs where ternary carries are generated for every digit-adder stage, design in [58] computes ternary carry ($Cout_i$) only for the final (i.e. $N - 1^{\text{th}}$) digit-adder stage. Although this design has least propagation delay, it has large power consumption and power-delay product when compared to similar designs in literature. This is because the encoder used in [58] consumes large static power.

Efficient conditional sum and carry look-ahead-based ternary multi-digit adders have been presented in [60]. Figure 4.1(d) shows ternary multi-digit Conditional Sum Adder (CSA) presented in [60]. At each digit-adder stage of CSA three different sums (S_{i0}, S_{i1}, S_{i2}) and carry-outs (C_{i0}, C_{i1}, C_{i2}), corresponding to carry-in of 0, 1 and 2 are computed using 3 : 1 multiplexer-based single-digit adders. Further depending on the actual carry-in, an array of 3 : 1 multiplexers are used to compute the final sum digits (Sum_i) and carry-out ($Cout$).

Figure 4.1(e) shows the block level implementation of multi-digit Carry-Lookahead Adder (CLA) presented in [60], where 3-digit CLA is used as basic unit to design multi-digit adders. In this design four propagate functions (p_i^1, p_i^2, p_i^3 and p_i^4), which correspond to different carry-in and carry-out conditions are generated at each digit-adder stage. p_i^1 corresponds to the input carry of 0 producing an output carry of 1. This happens when the sum of A_i and B_i is greater than 2. p_i^2 corresponds to carry out of 1 for an input carry of 1 which happens when the sum of A_i and B_i is greater than 1. Similarly, p_i^3 corresponds to an input carry of 2 and an output carry of 1. Further, p_i^4 corresponds to input carry of 2 and output carry of 2. The definitions of propagate signals are given in equations (4.1)-(4.4), where A_i^0, A_i^1 and A_i^2 correspond to $A_i = 0, 1$ and 2 respectively, and B_i^0, B_i^1 and B_i^2 correspond to $B_i = 0, 1$ and 2 respectively.

$$p_i^1 = A_i^2 \cdot (B_i^1 + B_i^2) + A_i^1 \cdot B_i^2 \quad (4.1)$$

$$p_i^2 = A_i^0 \cdot B_i^2 + A_i^1 \cdot (B_i^1 + B_i^2) + A_i^2 \quad (4.2)$$

$$p_i^3 = A_i^0 \cdot (B_i^1 + B_i^2) + A_i^1 + A_i^2 \cdot (B_i^0 + B_i^1) \quad (4.3)$$

$$p_i^4 = A_i^2 \cdot B_i^2 \quad (4.4)$$

These single-digit propagate functions are given as inputs to carry-lookahead generator, which implements a set of logic expressions [60] to compute group propagate functions ($P_{i:j}^1, P_{i:j}^2, P_{i:j}^3$ and $P_{i:j}^4$) and carry-out signals. The group propagate functions and carry-out for implementation of 3-digit CLA are obtained from single-digit propagate functions as given by equations (4.5)-(4.9), where C_3 represents carry-out, $C_{in0}, C_{in1}, C_{in2}$ represent C_{in} (carry-in) being 0, 1 and 2.

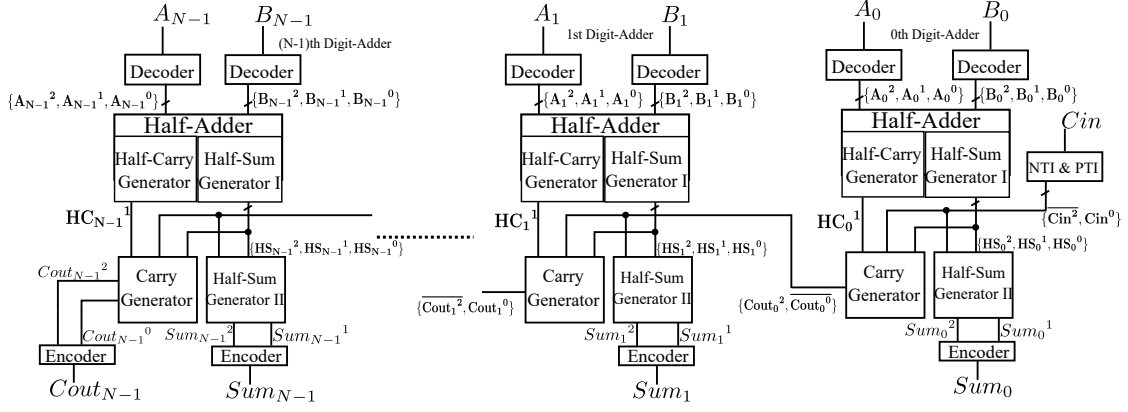


Figure 4.2: Proposed design technique to implement Multi-digit adders

$$P_{2:0}^1 = p_2^2(p_2^1 + p_1^1 + p_1^2 p_0^1) \quad (4.5)$$

$$P_{2:0}^2 = p_2^2(p_2^1 + p_1^1 + p_1^2 p_0^2) \quad (4.6)$$

$$P_{2:0}^3 = (p_1^1 + p_1^2 p_0^3 + p_1^3 p_0^4)(p_2^1 + p_2^2 p_1^3 + p_2^3 p_1^4) \quad (4.7)$$

$$P_{2:0}^4 = p_0^4 p_1^4 p_2^4 \quad (4.8)$$

$$C_3 = 1.(P_0^1 C_{in0} + P_0^2 C_{in1} + P_0^3 C_{in2}) + P_0^4 C_{in2} \quad (4.9)$$

Finally the carry-out signals ($Cout_{i-1}$) and inputs (A_i , B_i) are given to a 3 : 1 multiplexer-based single-digit adders, which generate the final sum digits (Sum_i). Although multi-digit CSA and CLA designs have low-propagation delays and least PDP, they have complex carry propagation path and consume large power when compared to other existing designs.

4.3 Proposed Half-Adder Based Ripple-carry Ternary Adders

The overview of existing adder designs presented in previous section shows that they are optimized either to reduce propagation delay [58, 60] or power consumption [38] or PDP [39, 60]. In this work, we present new designs for CNFET-based multi-digit adders which are optimized to reduce all the design parameters i.e. delay, power and power-delay product. Section 4.3.1 presents the proposed design technique to implement multi-digit adders while Sections 4.3.2.1 to 4.3.2.4 provides the implementation details of different blocks used in the design.

4.3.1 Basic Idea

The proposed technique used to implement N -digit adder is shown in Figure 4.2. Here $A(A_{N-1}...A_1A_0)$, $B(B_{N-1}...B_1B_0)$, Cin are ternary inputs and Sum ($Sum_{N-1}.. Sum_0$), $Cout$ are ternary outputs. Intermediate binary signals are represented using the same notation as explained in section 4.2.2.

This technique differs from existing techniques mainly in the way carry generation/propagation are handled. Existing designs [38, 39, 58] implement the carry generation block, which computes carry-out signals for i^{th} digit-adder stage using inputs A_i , B_i and carry signals from $(i - 1)^{th}$ stage. This results in a complex carry generation circuit leading to a large delay in carry propagation chain.

The proposed carry generation block computes carry-out signals of i^{th} stage from half-adder outputs of i^{th} stage (instead of inputs A_i , B_i) and carry signals from $(i - 1)^{th}$ stage. It uses a delay optimized half-adder that consists of a half-sum generator and a half-carry generator. This half-adder generates binary signals corresponding to ternary Half-Sum (HS_i) and Half-Carry (HC_i) for inputs A_i and B_i .

While generation of carry-out becomes more complex due to the usage of two blocks (i.e. half-adder and carry generator) instead of one, there is a reduction in the delay of the carry propagation path. This is due to two main factors: First is the fact that the

half-adder outputs for all digit-adder stages are computed in parallel, resulting in only one half-adder contributing to critical path delay. Second, since a part of the carry-out generation logic is already implemented within the half-adder, it results in the reduced complexity of the carry generator block leading to reduced delay in carry propagation.

The final sum computation at each digit-adder stage is carried out using another half-sum generator and an encoder. Further, the multi-digit adder design avoids redundant encoder-decoder pairs in the carry propagation path and generates ternary carry-out only for the final (i.e. $N - 1^{th}$) digit-adder stage similar to the design presented in [58].

4.3.2 Implementation using CNFET

4.3.2.1 Designs for Decoder and Half-Adder

The decoder and half-adder circuits are implemented using Approach II, which is presented in Section 3.3.2. This approach uses a low-delay decoder which is shown in Figure 4.3. The mutually exclusive binary signals generated by the decoder are used as inputs to a half-adder which consists of a half-sum generator I to compute Half-Sum (HS) and a half-carry generator to compute Half-Carry (HC). The outputs of half-sum generator I are further used to generate final sum digit (Sum_i) with the help of another half-sum generator denoted as half-sum generator II. Instead of generating ternary signal HS , mutually exclusive binary signals HS^2, HS^1, HS^0 are generated and used as inputs to the half-sum generator II. This avoids the use of encoder-decoder pair after the first half-sum generator. The half-carry generator generates binary signal HC^1 which is used in computing the carry-out.

Half-adder in i^{th} digit-adder stage computes the mutually exclusive binary signals HS_i^2, HS_i^1, HS_i^0 and HC_i^1 , that correspond to half-sum and half-carry of ternary digits A_i, B_i . Since addition of two ternary digits will result in a carry of either 0 or 1, only one signal HC_i^1 is used to represent the output of half-carry generator block.

The Karnaugh maps (K-map) of ternary Half-Sum (HS) and Half-Carry (HC) are shown in Table 4.1(a) and 4.1(b) respectively. Based on Table 4.1 the logical

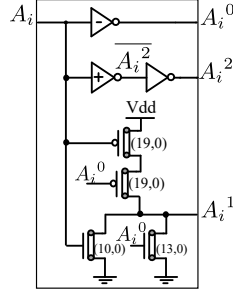


Figure 4.3: Decoder Implementation

Table 4.1: Half-Adder Karnaugh Maps

(a) Half-Sum (HS)

A/B	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

(b) Half-Carry (HC)

A/B	0	1	2
0	0	0	0
1	0	0	1
2	0	1	1

expressions for half-adder outputs can be determined as below:

$$HS_i^2 = A_i^2 B_i^0 + A_i^1 B_i^1 + A_i^0 B_i^2 \quad (4.10)$$

$$HS_i^1 = A_i^2 B_i^2 + A_i^1 B_i^0 + A_i^0 B_i^1 \quad (4.11)$$

$$HS_i^0 = A_i^2 B_i^1 + A_i^1 B_i^2 + A_i^0 B_i^0 \quad (4.12)$$

$$HC_i^1 = A_i^2 B_i^1 + A_i^2 B_i^2 + A_i^1 B_i^2 \quad (4.13)$$

Figure 4.4 shows the existing gate-level implementation (in [36]) to generate half-adder outputs HS_i^2 , HS_i^1 , HS_i^0 and HC_i^1 . The logic gates are implemented at transistor level using complementary logic style. Alternatively, equations (4.10) - (4.13) can be directly implemented at transistor-level using complementary logic style. Figure 4.5 shows the implementation of HS_i^2 using the equation (4.10). Similar transistor-level implementations can be used to implement HS_i^1 and HC_i^1 with the help of equations (4.11) and (4.13) respectively. Direct transistor-level implementations would result in

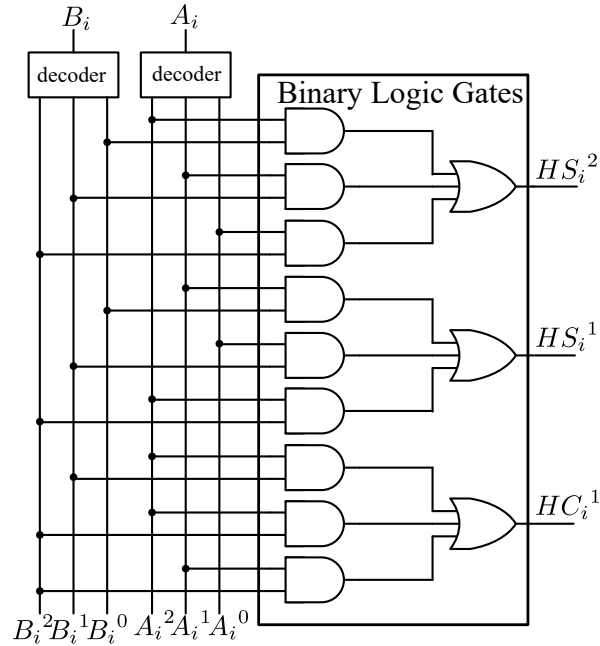
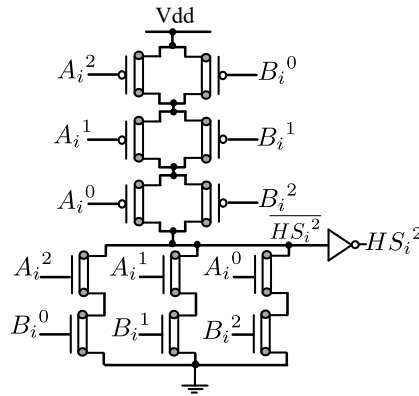


Figure 4.4: Gate-level Implementation to generate Half-Adder outputs [36]

Figure 4.5: Transistor-level Implementation for HS_i^2

a circuit with increased transistor stacking (three transistors) for pull-up network as evident from Figure 4.5. Hence Approach II presented in Section 3.3.2 is used to derive alternate equivalent logical expressions, which help in reducing transistor stacking. Equations (4.14), (4.15) and (4.16) are alternate logic expressions which are equivalent to equations (4.10), (4.11) and (4.13) respectively. Figure 4.6 shows the transistor-level implementations to generate half-adder outputs.

$$HS_i^2 = (A_i^2 + \overline{B_i^0})(A_i^1 + \overline{B_i^1})(A_i^0 + \overline{B_i^2}) \quad (4.14)$$

$$HS_i^1 = (A_i^2 + \overline{B_i^2})(A_i^1 + \overline{B_i^0})(A_i^0 + \overline{B_i^1}) \quad (4.15)$$

$$\overline{HC_i^1} = (\overline{A_i^2} + \overline{B_i^1})(A_i^0 + \overline{B_i^2}) \quad (4.16)$$

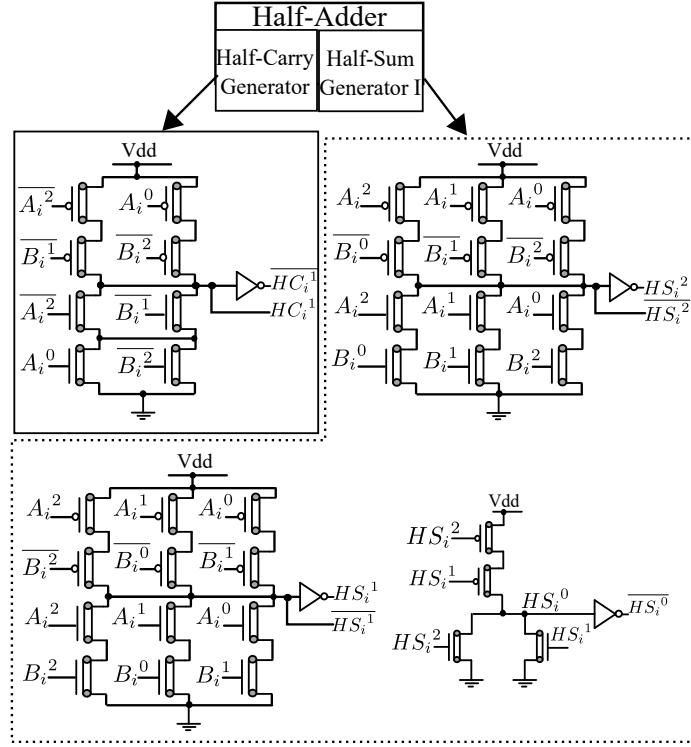


Figure 4.6: Proposed Transistor-Level Implementation of Half-carry Generator and half-sum Generator I (all transistors with chirality of (19, 0))

4.3.2.2 Design of Carry Generator Block

The proposed carry generator block uses outputs of the half-adder in the i^{th} stage and carry signals from $i - 1^{th}$ stage to compute carry-out signals for the i^{th} stage. The relation between carry-out of i^{th} stage and half-adder outputs for different cases of carry-in is shown below:

Case 1 (carry-in = 0 i.e. $Cout_{i-1} = 0$): For this case, $Cout_i$ is equal to 1 if sum of A_i and B_i is greater than or equal to 3, else $Cout_i$ is equal to 0.

Case 2 (carry-in = 1 i.e. $Cout_{i-1} = 1$): For this case, $Cout_i$ is equal to 1 if sum of A_i and B_i is greater than or equal to 2, else $Cout_i$ is equal to 0.

Case 3 (carry-in = 2 i.e. $Cout_{i-1} = 2$): For this case, $Cout_i$ is equal to 1 if sum of A_i and B_i is greater than or equal to 1 but less than 4, $Cout_i$ is equal to 2 if sum of A_i and B_i is greater than or equal to 4, else $Cout_i$ is equal to 0.

Based on the above relations, truth-table for carry-out signal can be obtained and is shown in Table 4.2. Here half-adder outputs HS_i and HC_i represent the sum of A_i and B_i .

Table 4.2: Truth-Table for Carry-Out

Sum of A_i, B_i	HC_i	HS_i	$Cout_i$		
			$Cout_{i-1} = 0$	$Cout_{i-1} = 1$	$Cout_{i-1} = 2$
0	0	0	0	0	0
1	0	1	0	0	1
2	0	2	0	1	1
3	1	0	1	1	1
4	1	1	1	1	2

Table 4.2 is used to derive the expressions for i^{th} stage carry-out signals as shown by logic equations (4.17) and (4.18). These equations are used to implement the carry generator blocks in the proposed design.

$$Cout_i^0 = \overline{HC_i^1}(\overline{HS_i^2} + Cout_{i-1}^0)(HS_i^0 + \overline{Cout_{i-1}^2}) \quad (4.17)$$

$$Cout_i^2 = HC_i^1 HS_i^1 Cout_{i-1}^2 \quad (4.18)$$

Figure 4.7 shows the proposed delay optimized carry propagation path with transistor-level implementation of carry generator blocks. As seen in the Figure 4.7, carry generator block in i^{th} digit-adder stage generates $\overline{Cout_i^0}$ and $Cout_i^2$ when i is *EVEN*. Similarly it generates $Cout_i^0$ and $\overline{Cout_i^2}$ when i is *ODD*. This avoids the usage of extra inverters in the carry propagation path.

The transistor-level implementation of the carry generator block has lower transistor stacking with a maximum of 3 stacked transistors when compared to the carry generator design presented in [58], which has a maximum of 7 stacked transistors. Due to the lower transistor stacking in the proposed carry generator block, the carry propagation

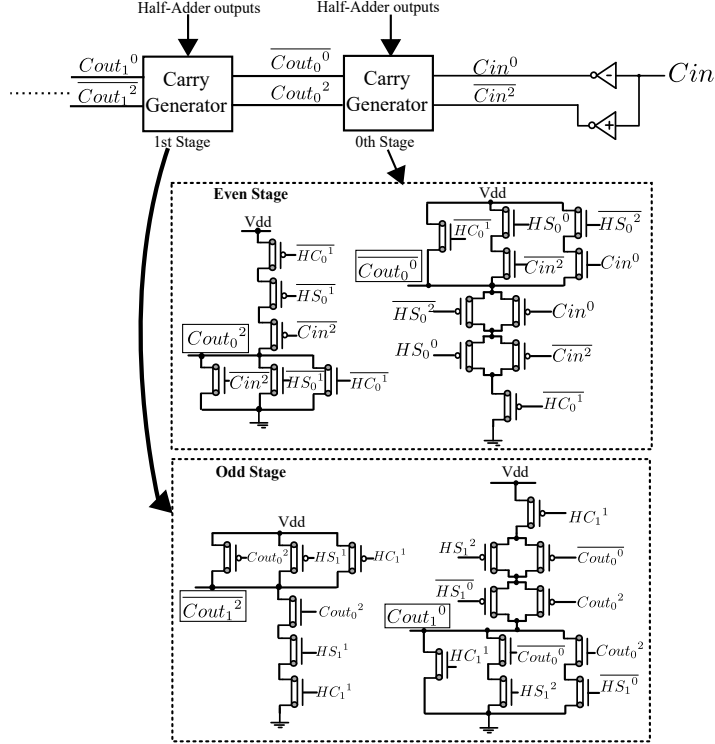


Figure 4.7: Delay-Optimized Carry Propagation Path with Proposed Carry Generator Blocks (all transistors with chirality of (19, 0))

path has lower delay.

4.3.2.3 Design of Final Half-Sum Generator

Outputs of half-sum generator I and carry generator blocks are further used to obtain final sum digit with the help of half-sum generator II. Figure 4.8 shows the transistor-level implementation of half-sum Generator II.

In this implementation, $Cout_{i-1}^2$ (if i is EVEN) or $Cout_{i-1}^0$ (if i is ODD) is generated using a binary inverter and $Cout_{i-1}^1$ is generated using a *NOR* gate. Binary signals corresponding to final ternary sum i.e. Sum_i^2 and Sum_i^1 are generated using transistor-level circuits similar to that of half-sum generator I.

4.3.2.4 Design of Low-Power Encoder

An improved encoder, which uses transistors of same chiralities has been presented in Section 3.3. This encoder is used to compute final ternary Sum and Carry. Figure 4.9 shows the implementation of the encoder, where presence of additional back to

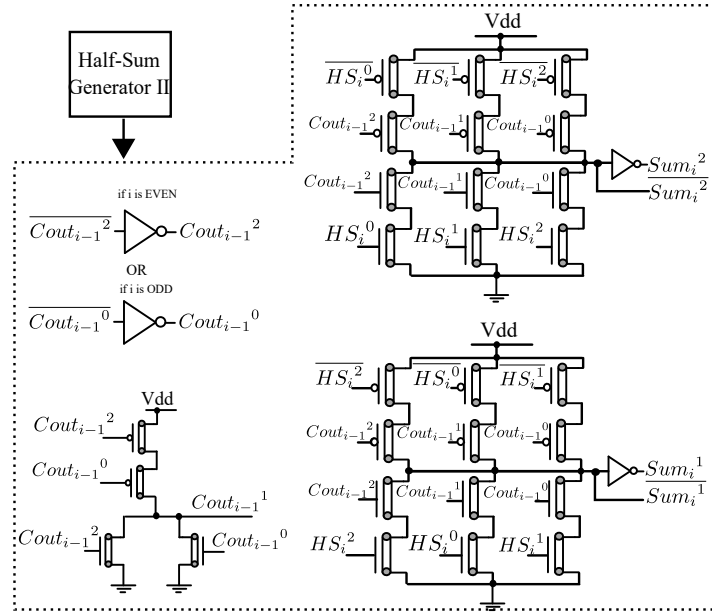


Figure 4.8: Transistor-level implementation of Half-sum generator II (all transistors with chirality of $(19, 0)$)

back connected transistor creates a high resistance path between VDD and GND to generate *logic 1* at the output. This high resistance path while limiting the static current thereby reducing the static power consumption, however, causes the increase in encoder delay. This encoder delay contributes to the overall propagation delay of the proposed multi-digit adders, albeit, at lower operand sizes. However at higher operand sizes, the same delay is dominated by the carry propagation delay and thus is not relevant. Figure 4.9 shows the implementation of two encoders used in the proposed design of multi-digit adders. The encoder shown in Figure 4.9(a) is used to generate the ternary sum digits (Sum_i) at each digit-adder stage. In this circuit, if Sum_i^2 is equal to *logic 2* then transistor M1 switches *ON* and *logic 2* is obtained at the output. Instead, if Sum_i^1 is equal to *logic 2* then a direct path from VDD to GND is created (via transistors M2-M5) causing *logic 1* to appear at Sum_i . If neither of Sum_i^2 and Sum_i^1 are *logic 2* then transistors M6 and M7 switch *ON* pulling output node to *logic 0*. An encoder, shown in Figure 4.9(b) is used to generate the final carry-out ($Cout_{N-1}$) of the adder.

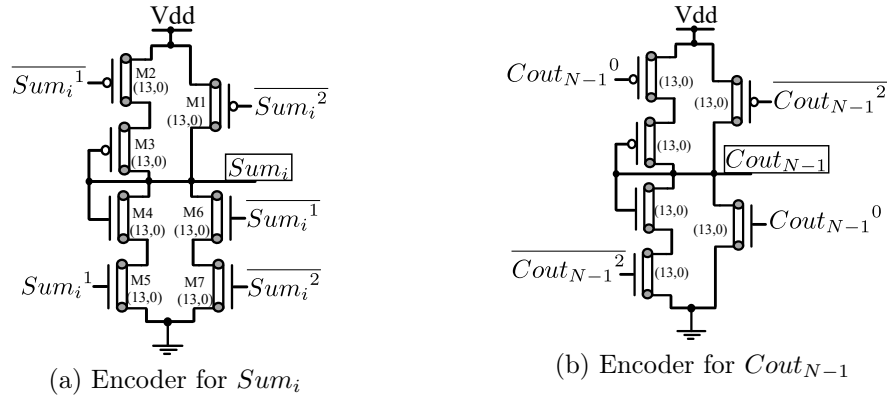


Figure 4.9: Designs for Encoder

4.4 Proposed Ternary Prefix Adder designs

The main disadvantage of the CLA in [60] is the generation of four propagate functions p_i^1, p_i^2, p_i^3 and p_i^4 (refer equations (4.1)-(4.4)), which correspond to different carry-in and carry-out conditions. Four different propagate functions are required because of the ternary nature of carry-in and carry-out. This results in a complex carry-lookahead generator (refer equations (4.5)-(4.9)) which is used to compute group propagate functions and carry-out signals. In this section a new technique to implement ternary adders, which use binary carry propagation networks, is presented.

4.4.1 Concept of Carry Propagate-Generate in Binary Addition

The carry Propagate-Generate technique is widely used in the implementation of carry-lookahead [67] and binary prefix adders [68]. To illustrate this, consider a binary addition where addition of inputs $A(A_{N-1}...A_1A_0)$, $B(B_{N-1}...B_1B_0)$ and carry-in Cin results in $Sum(Sum_{N-1}...Sum_1Sum_0)$ and carry-out $Cout_{N-1}$. Assume intermediate carry-out generated at i^{th} digit-adder stage is represented as $Cout_i$. At each digit-adder stage, the inputs and outputs are related according to the table shown in Figure 4.10.

This table clearly shows that at i^{th} stage, carry-out $Cout_i$ is either equal to carry-in $Cout_{i-1}$ or to half-carry (HC_i) which is generated by adding A_i and B_i . Based on this dependency, two conditions are defined namely Carry Propagate (when carry-in $Cout_{i-1}$ **propagates** as carry-out $Cout_i$) and Carry Generate (carry-out $Cout_i$ **is**

A_i	B_i	$Cout_{i-1}$ ($Cout_{-1} = Cin$)	$A_i + B_i$ Half-Adder		$A_i + B_i + Cout_{i-1}$	
			Carry $HC_i = g_i$	Sum $HS_i = p_i$	$Cout_i$	Sum_i
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	1	0	1
0	1	1	0	1	0	0
1	0	0	0	1	0	1
1	0	1	0	1	0	0
1	1	0	1	0	1	0
1	1	1	1	0	1	1

Figure 4.10: Table showing Carry Propagate and Generate conditions for Binary addition

generated from inputs A_i , B_i).

In binary addition, the carry propagate and carry generate conditions are used to compute propagate (p_i) and generate (g_i) signals using equations (4.19) and (4.20) respectively.

$$p_i = HS_i = A_i \oplus B_i \quad (4.19)$$

$$g_i = HC_i = A_i \cdot B_i \quad (4.20)$$

Carry computation at each digit-adder stage can now be transformed to a prefix problem [69] using the associative operator \circ , which associates pairs of generate and propagate signals as shown by equation (4.21).

$$(g_i, p_i) \circ (g_j, p_j) = (g_i + p_i \cdot g_j, p_i \cdot p_j) = (G_{[i:j]}, P_{[i:j]}) \quad (4.21)$$

where $i > j$, $G_{[i:j]}$ and $P_{[i:j]}$ represent group propagate and generate functions respectively. Here $+$ and \cdot indicate logical *OR* and *AND* operations respectively.

Using the operator \circ consecutive propagate and generate pairs can be grouped to generate carry-out at i^{th} digit-adder stage by using equation (4.22).

$$Cout_i = G_{[i:0]} = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \dots \circ (g_0, p_0) \quad (4.22)$$

After the carries are generated, the sum at each digit-adder stage is computed by

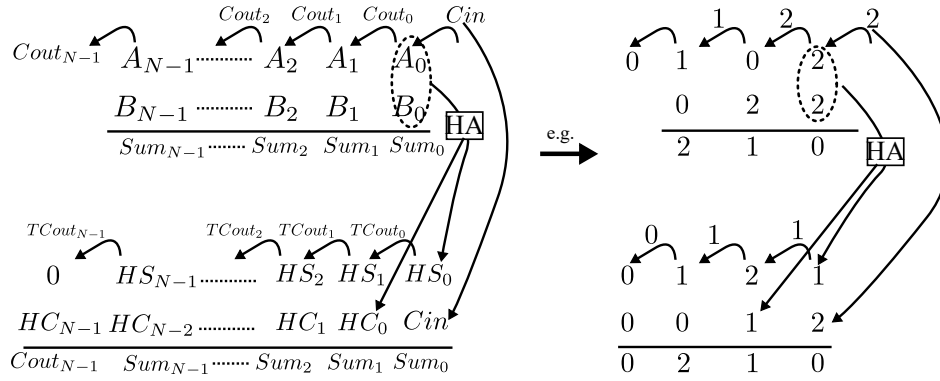


Figure 4.11: Example of ternary addition using proposed transformation

using equation (4.23).

$$Sum_i = p_i \oplus Cout_{i-1} \quad (4.23)$$

Several variants of binary prefix adders can be found in the literature [68–72] which while using different prefix networks, use the same principle of carry “Propagate-Generate” given in equations (4.21) and (4.22).

4.4.2 Basic Idea for Ternary Prefix Adders

Consider an example of ternary addition illustrated in Figure 4.11. Here N -digit ternary inputs $A(A_{N-1} \dots A_1 A_0)$, $B(B_{N-1} \dots B_1 B_0)$ are added with carry-in Cin resulting in a ternary $Sum(Sum_{N-1} \dots Sum_1 Sum_0)$ and carry-out $Cout_{N-1}$. The intermediate carries which propagate to next stage are represented by $Cout_{N-2} \dots Cout_0$.

Since A_i , B_i , Cin and $Cout_i \in \{0, 1, 2\}$, using the concept of carry Propagate-Generate in ternary addition is not as straightforward as it is in conventional binary addition [68]. To demonstrate this, consider the table shown in Fig 4.12, which lists all possible inputs and outputs at each digit-adder stage. Here the output carry $Cout_i$ is equal to input carry $Cout_{i-1}$ (carry propagate condition) or carry (HC_i) which is generated by adding A_i and B_i (carry generate condition), only for some cases. Hence the concept of carry Propagate-Generate cannot be used directly in ternary addition. To solve this problem, the CLA presented in [60] uses four different propagate functions corresponding to different carry-in and carry-out conditions, thus increasing the

complexity. A careful observation of table in Figure 4.12 however reveals that the carry propagate or carry generate conditions fail when $Cout_i = 2$. Hence, if the carries that propagate to next stage are restricted to 0 or 1, then the concept of carry Propagate-Generate can be applied to ternary addition. To accomplish this, the original operands $A(A_{N-1}...A_1A_0)$, $B(B_{N-1}...B_1B_0)$ and Cin are transformed to intermediate operands which are represented as $(HS_{N-1}...HS_1HS_0)$ and $(HC_{N-1}...HC_0Cin)$. This transformation, shown in Figure 4.11, is achieved by adding A_i and B_i resulting in Half-Sum ($HS_i \in \{0, 1, 2\}$) and Half-Carry ($HC_i \in \{0, 1\}$). After the transformation the addition at each digit-stage will result in a transformed carry-out $TCout_i \in \{0, 1\}$, which propagates to the next stage. Since these variables can assume values 0 or 1, the binary technique of carry Propagate-Generate can now be applied. The final carry-out $Cout_{N-1}$ is generated by adding $HC_i \in \{0, 1\}$ and $TCout_i \in \{0, 1\}$.

A_i	B_i	$Cout_{i-1}$ ($Cout_{-1} = Cin$)	$A_i + B_i$ Half-Adder Carry Sum		$Cout_i$	Sum_i
			HC_i	HS_i		
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	0	2	0	0	0	2
0	1	0	0	1	0	1
0	1	1	0	1	0	2
0	1	2	0	1	1	0
0	2	0	0	2	0	2
0	2	1	0	2	1	0
0	2	2	0	2	1	1
1	0	0	0	1	0	1
1	0	1	0	1	0	2
1	0	2	0	1	1	0
1	1	0	0	2	0	2
1	1	1	0	2	1	0
1	1	2	0	2	1	1
1	2	0	1	0	1	0
1	2	1	1	0	1	1
1	2	2	1	0	1	2
2	0	0	0	2	0	2
2	0	1	0	2	1	0
2	0	2	0	2	1	1
2	1	0	1	0	1	0
2	1	1	1	0	1	1
2	1	2	1	0	1	2
2	2	0	1	1	1	1
2	2	1	1	1	1	2
2	2	2	1	1	2	2

Figure 4.12: Table showing Carry Propagate and Generate conditions for ternary addition

The conditions, under which the transformed input carry ($TCout_{i-1}$) propagates

HS_i	HC_{i-1} ($HC_{-1} = Cin$)	$TCout_{i-1}$ ($TCout_{-1} = 0$)	$HS_i + HC_{i-1}$ Half-Adder Carry Sum		$HS_i + HC_{i-1} + TCout_{i-1}$	
			THC_i	THS_i	$TCout_i$	Sum_i
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	1	0	2
0	1	1	0	1	0	1
0	2	0	0	2	0	2
0	2	1	0	2	0	0
1	0	0	0	1	0	1
1	0	1	0	1	0	2
1	1	0	0	2	0	2
1	1	1	0	2	0	0
1	2	0	1	0	1	0
1	2	1	1	0	1	1
2	0	0	0	2	0	2
2	0	1	0	2	0	0
2	1	0	1	0	1	0
2	1	1	1	0	1	1
2	2	0	1	1	1	1
2	2	1	1	1	1	2

Figure 4.13: Table showing Carry Propagate and Generate conditions after proposed transformation

as output carry ($TCout_i$), are defined with the help of the table shown in Figure 4.13. Here THC_i and THS_i represent half-carry and half-sum of transformed inputs HS_i and HC_{i-1} . As seen from the table, the carry-in propagates as carry-out, if THS_i is equal to 2. In other cases, the output carry ($TCout_i$) is equal to the half-carry THC_i that is generated by adding transformed inputs. This is similar to the concept of carry Propagate-Generate which is used in implementation of binary prefix adders (refer section 4.4.1). This similarity enables the use of prefix-based carry propagation networks in the implementation of ternary adders.

4.4.3 Proposed Implementation of Ternary Prefix Adders using CNFET

In this subsection CNFET-based designs of multi-digit ternary adders, based on the concept of Propagate-Generate, are presented. Figure 4.14 illustrates the block level implementation of the proposed ternary adders. Here $A(A_{N-1}...A_1A_0)$, $B(B_{N-1}...B_1B_0)$, Cin are ternary inputs and $Sum(Sum_{N-1}...Sum_1Sum_0)$, $Cout$ are ternary outputs. The intermediate binary signals, which are represented by notation X_i^j , correspond to a ternary signal X_i . Binary Signal, X_i^j corresponds to i^{th} digit-adder stage whose

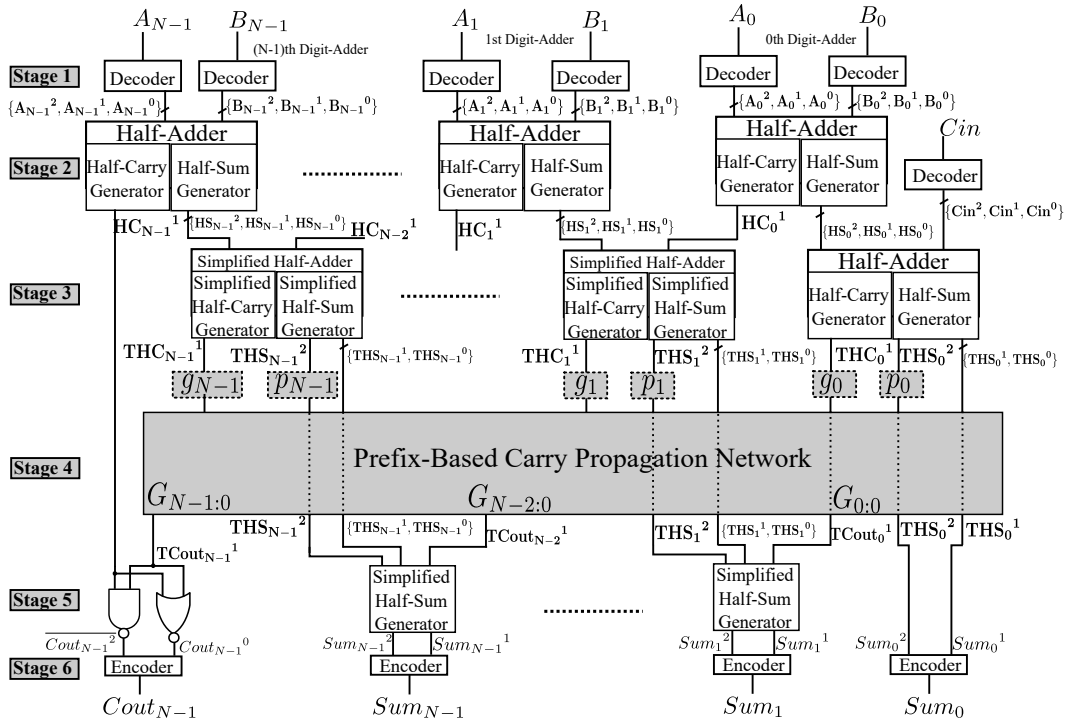


Figure 4.14: Block-level Implementation of Proposed Ternary Adders

value is either *logic 2* (if $X_i = j$) or *logic 0* (if $X_i \neq j$), where $j \in \{0, 1, 2\}$. For example, A_0^1 corresponds to the input of 0^{th} digit-adder stage whose value is *logic 2*, only if ternary signal A_0 is equal to *logic 1*. Binary logic values, 0 and 1 correspond to ternary logic values 0 and 2. Also, $\{\}$ is used to represent a group of binary signals and their complements.

4.4.3.1 Propagate and Generate for Ternary Adders

In Stage 1 of the proposed adders, the ternary inputs are converted to binary using ternary decoders. The CNFET-based decoder circuit is shown in Figure 3.13b is used at this stage. In Stage 2, the mutually exclusive binary signals corresponding to original ternary inputs, $A(A_{N-1} \dots A_1 A_0)$, $B(B_{N-1} \dots B_1 B_0)$ and Cin are transformed to binary signals corresponding to intermediate operands, $(HS_{N-1} \dots HS_1 HS_0)$ and $(HC_{N-1} \dots HC_0 Cin)$. This transformation is achieved by using Half-Adder (HA) which in turn consists of half-sum generator and a half-carry generator. Since $HC_i \in \{0, 1\}$ for $i \geq 0$, only one signal HC_i^1 is used to represent the output of half-carry generator block. The implementation of Half-Adder used in Stage 2 is shown in Figure 4.5, which

uses equations (4.14), (4.15) and (4.16).

In Stage 3 the binary signals, which represent the half-sum (THS_i) and half-carry (THC_i), are generated by adding the transformed inputs HS_i and HC_{i-1} . Since HC_i can either be logic 0 or 1 (for $i \geq 0$), simplified half-adders which have less complexity than those in Stage 2 are used for computing the binary signals corresponding to THS_i and THC_i . Only exception is for the 0^{th} digit-adder stage, where $Cin(HC_{-1}) \in \{0, 1, 2\}$ and thus, normal half-adder shown in Figure 3.21 is used to generate THS_0 and THC_0 . Similar to HC_i , the half-carry (THC_i) generated at this stage is either 0 or 1. Thus, one binary signal i.e. THC_i^1 is used to represent the half-carry generated in stage 3. Using table shown in Figure 4.13, the logical expressions for simplified half-adder outputs at i^{th} stage (for $i > 0$) can be determined as below:

$$THS_i^2 = HS_i^1 \cdot HC_{i-1}^1 + HS_i^2 \cdot \overline{HC_{i-1}^1} \quad (4.24)$$

$$THS_i^1 = HS_i^1 \cdot \overline{HC_{i-1}^1} + HS_i^0 \cdot HC_{i-1}^1 \quad (4.25)$$

$$THC_i^1 = HS_i^2 \cdot HC_{i-1}^1 \quad (4.26)$$

Based on the carry propagate and generate conditions shown in Figure 4.13, if the half-sum THS_i is equal to 2 then the input carry propagates to output. Otherwise carry is generated and is equal to half-carry THC_i . Thus for i^{th} stage, Propagate (p_i) and Generate (g_i) are defined by equations (4.27) and (4.28) respectively. Figure 4.15 shows the CNFET-based implementation of simplified half-adder along with p_i and g_i .

$$p_i = THS_i^2 \quad (4.27)$$

$$g_i = THC_i^1 \quad (4.28)$$

Further carry-out, group propagate and generate signals can be defined by equations

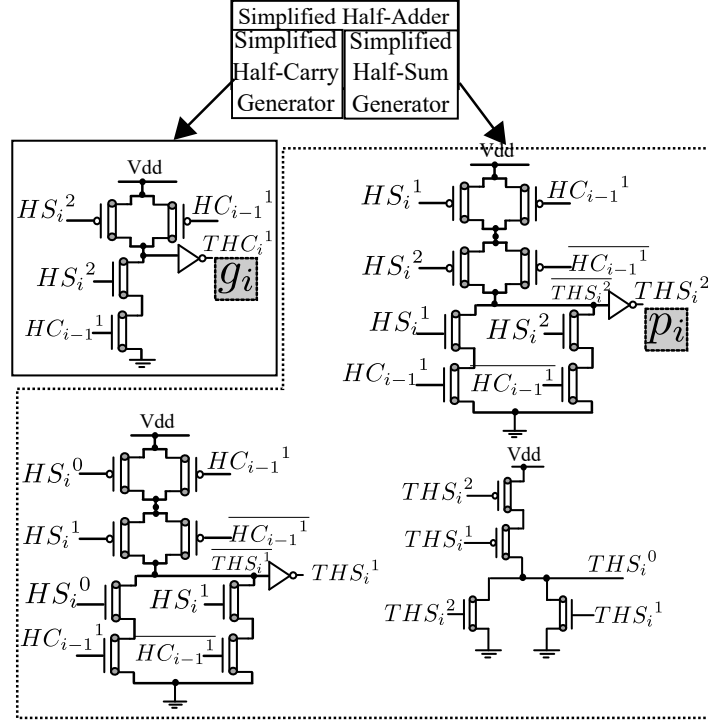


Figure 4.15: CNFET-based Implementation of Simplified Half-Adder (all CNFETs have chirality of (19, 0))

(4.29), (4.30) and (4.31) respectively.

$$G_{[i:0]} = TCout_i^1 = g_i + p_i \cdot TCout_{i-1}^1 = THC_i^1 + THS_i^2 \cdot TCout_{i-1}^1 \quad (4.29)$$

$$P_{[i:j]} = p_i \cdot p_j = THS_i^2 \cdot THS_j^2 \quad (4.30)$$

$$G_{[i:j]} = g_i + p_i \cdot g_j = THC_i^1 + THS_i^2 \cdot THC_j^1 \quad (4.31)$$

4.4.3.2 Carry Generation using Binary Prefix networks

Carry computation for ternary addition has now transformed in to prefix problem similar to that of binary addition. Hence the equations (4.21) and (4.22) can be used for carry generation. This enables the use of prefix-based carry propagation networks, like Kogge-Stone [69], Ladner-Fischer [70] etc., as Stage 4 in the proposed multi-digit

ternary adders. In this work, we have used three different types of carry propagation networks namely Ripple-based, Kogge-Stone and Carry lookahead-based. It should be noted that any parallel prefix-network [68], which follows equation (4.21) and (4.22), can be used in Stage 4 of the proposed ternary adders. Figure 4.16 shows different types of networks that are used for carry computation in 6-digit ternary adder. The CNFET-based implementation of cells, which are used in carry propagation networks, are shown in Figure 4.17.

4.4.3.3 Final Sum and Carry Computation

After all the carries ($TCout_i^1$) are computed, the binary signals corresponding to Sum_i are computed in Stage 5 by using a simplified half-sum generator (shown in Figure 4.15) which implements the logical expressions given below:

$$Sum_i^2 = THS_i^2 \cdot \overline{TCout_i^1} + THS_i^1 \cdot TCout_i^1$$

$$Sum_i^1 = THS_i^1 \cdot \overline{TCout_i^1} + THS_i^0 \cdot TCout_i^1$$

Since $HC_i \in \{0, 1\}$ and $TCout_i \in \{0, 1\}$, the binary signals corresponding to final carry-out $Cout_{N-1}$, are generated by using binary *NAND* and *NOR* gates. The logical expressions for binary signals $\overline{Cout_{N-1}^2}$ and $Cout_{N-1}^0$ which are needed as inputs in final stage are given below:

$$\overline{Cout_{N-1}^2} = \overline{HC_{N-1}^1 \cdot TCout_{N-1}^1}$$

$$Cout_{N-1}^0 = \overline{HC_{N-1}^1 + TCout_{N-1}^1}$$

The last stage, Stage 6, of proposed adder consists of ternary encoders. Figure 4.9 shows the encoders, which are used to compute final ternary ($Sum_{N-1} \dots Sum_1 Sum_0$) and carry-out ($Cout_{N-1}$). The encoders shown in Figures 4.9(a) and 4.9(b) are used to generate the ternary sum digits (Sum_i) and final carry-out ($Cout_{N-1}$) respectively.

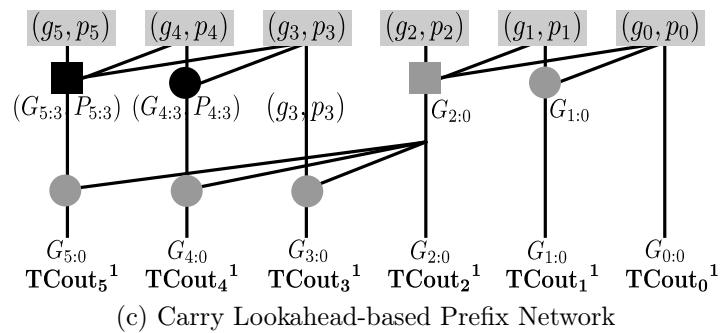
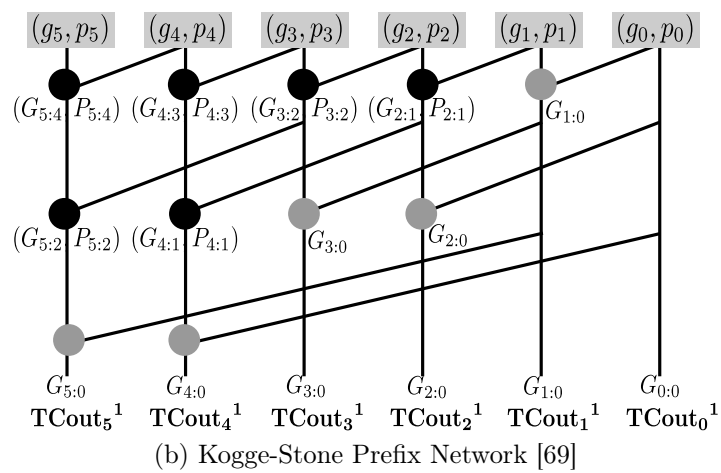
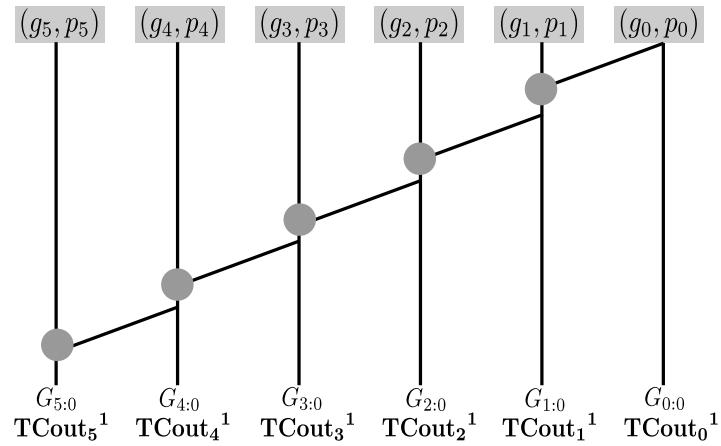


Figure 4.16: Prefix Networks used for Carry Computation in 6-digit Ternary Adder

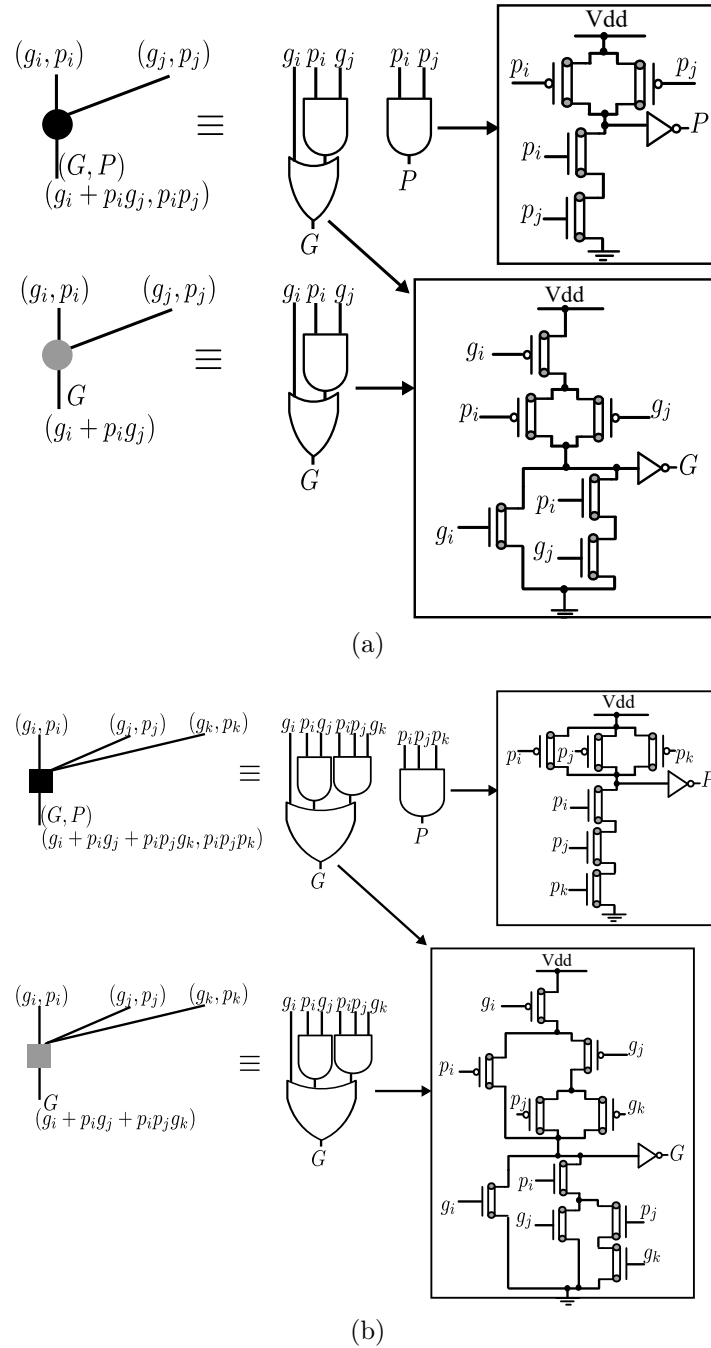


Figure 4.17: CNFET-based Implementation of Cells used in Prefix Networks (all CNFETs have chirality of (19, 0))

4.5 Simulation Results

In this section, simulation results for the proposed and existing designs of CNFET-based multi-digit adders are presented. All the circuits are simulated in HSPICE using the CNTFET model of [44, 45, 53] at $0.9V$ power supply and room temperature. The CNFETs used in the implementation are configured to have three tubes and a default pitch value equal to $20nm$. All the other parameters are set to their default values as presented in Table 2.4. In this work, ternary logic values 0, 1 and 2 correspond to voltages 0, $V_{dd}/2$ and V_{dd} respectively. For binary logic gates the logic values 0 and 1 correspond to voltages 0 and V_{dd} respectively. Binary gates are implemented using transistors, which are connected in complementary logic style and have chirality of (19, 0). Different ternary circuits are implemented using proposed and existing approaches and the design parameters are compared. For fair comparison, all the adders have been simulated with same test pattern.

Power consumption results are obtained by simulating the circuits with random test input patterns at switching frequency of $500MHz$. To measure the delay of the designs, test patterns have been chosen in such a way that the signal change propagates through the critical path and the maximum delay is measured. FO4 delay is calculated by loading the output node with four STI gates (implementation of STI gate is presented in [36]). The power-delay product is the product of worst-case propagation delay and average power consumption. Multi-digit adders of different operand sizes are realized and circuit parameters are compared to understand the relative performance. The following sub-section explains the results obtained.

4.5.1 Results and Discussion

Multi-Digit Ternary Adders (MTA), which are implemented using half-adders and the concept of Carry Propagate-Generate, are compared with other designs to evaluate the relative performance. In this section, MTA-1 and MTA-2 refer to the ripple-carry multi-digit adders presented in [58] and [39], whereas MTA-3 and MTA-4 refer to ripple-carry adders designed using single-digit adders presented in [38] and [59] respectively.

Two different multi-digit adders, Conditional Sum Adder (CSA) and Carry-Lookahead Adder (CLA), are presented in [60] and are indicated as MTA-5 and MTA-6 respectively. The Multi-digit half-adder based ternary adder design is represented as MHTA. The designs of Multi-digit Ternary Prefix Adder (MTPA), which use ripple-based, Kogge-Stone and carry lookahead-based prefix networks, are represented as MTPA-1, MTPA-2 and MTPA-3 respectively. Fig. 4.18 presents the logical simulation waveforms of the Kogge-Stone based prefix adder to show the correctness of the proposed structure. The same is also true with other proposed adders.

Table 4.3: Average Power Consumption for N -digit Adders

Power Consumption (in μW)				
N	3	6	9	12
MTA-1 [58]	51.10	103.7	134.3	212.4
MTA-2 [39]	8.20	16.69	23.41	31.85
MTA-3 [38]	5.69	12.18	17.61	22.64
MTA-4 [59]	28.88	60.89	83.62	117.0
MTA-5 [60]	6.68	13.36	20.17	27.07
MTA-6 [60]	15.10	30.06	43.92	67.15
Proposed MHTA	3.21	5.83	8.41	11.37
Proposed MTPA-1	2.97	5.30	7.48	10.03
Proposed MTPA-2	3.03	5.52	7.98	10.88
Proposed MTPA-3	2.97	5.38	7.62	10.30
Improv. in MHTA w.r.t. MTA-3	44%	52%	52%	50%
Improv. in MTPA-1 w.r.t. MTA-3	48%	57%	58%	56%
Improv. in MTPA-2 w.r.t. MTA-3	47%	55%	55%	52%
Improv. in MTPA-3 w.r.t. MTA-3	48%	56%	57%	54%

Table 4.3 shows the average power consumption for existing and proposed multi-digit ternary adders with different operand sizes. To measure power, all multi-digit adder designs are simulated with same random test patterns and the average power consumption is determined. The large power consumption in MTA-1, MTA-2 and MTA-4 is due to low resistance path created between VDD and GND while generating *logic 1*. The complexity involved in conditional sum and carry-lookahead logic results in large power consumption of MTA-5 and MTA-6.

The proposed designs result in up to 58% reduction in power consumption when compared to MTA-3, which has least power consumption among all the designs existing

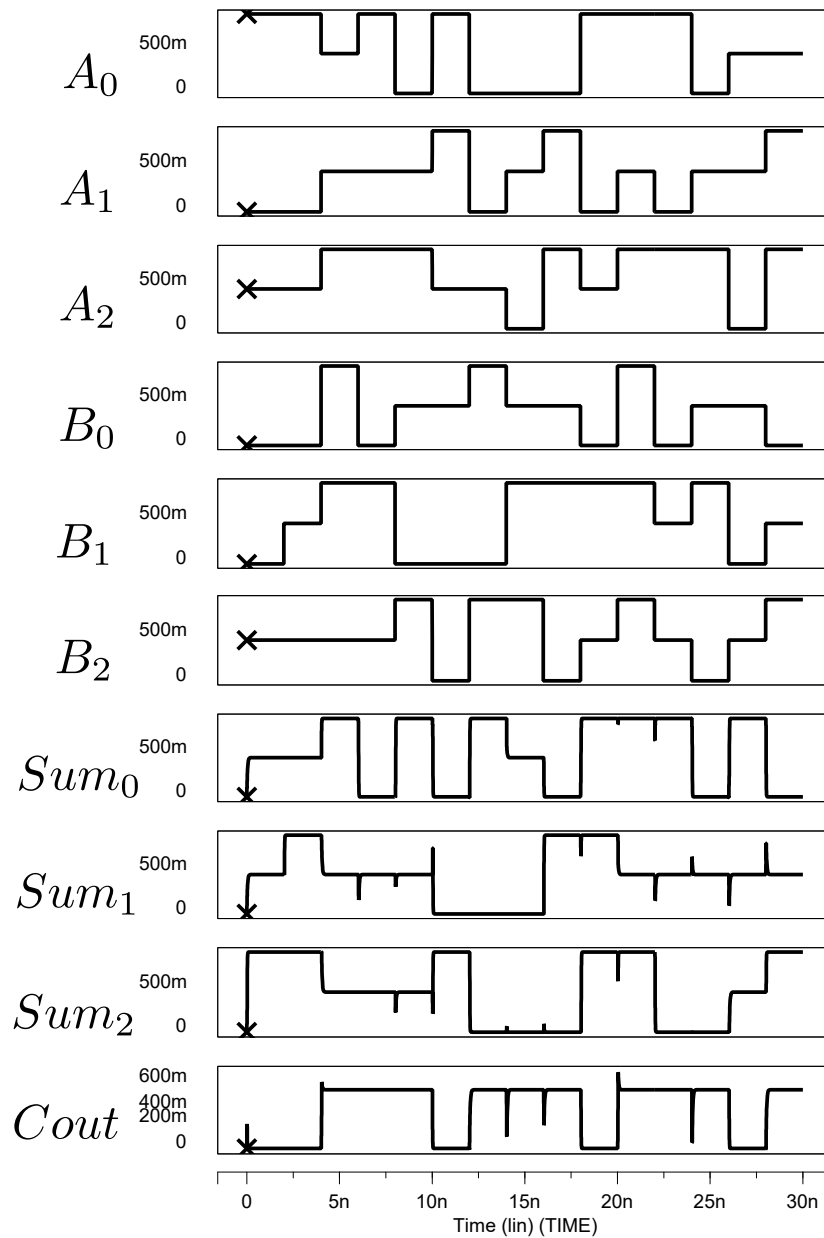


Figure 4.18: Simulation Waveforms for Kogge-Stone Prefix Adder

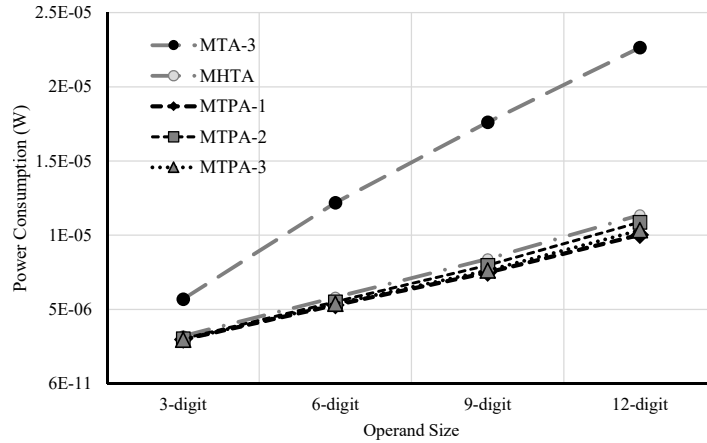


Figure 4.19: A Comparison of Power consumption

in literature. Figure 4.19 illustrates the reduction in power consumption of proposed adder designs when compared to MTA-3 for different operand sizes. This reduction in power consumption is mainly due to the use of low-power encoders.

Table 4.4 compares the propagation delay of the proposed and existing ternary adders. To measure the delay of the multi-digit adders, test patterns have been chosen in such a way that the signal change propagates through the critical path and the maximum delay is measured. FO4 delay is calculated by loading each of the output nodes in the critical path with four STI gates.

Table 4.4: Propagation (FO4) Delay for N -digit Adders

Propagation (FO4) Delay (in ps)				
N	3	6	9	12
MTA-1 [58]	63.76	117.2	170.7	223.7
MTA-2 [39]	97.52	205.4	313.4	422.6
MTA-3 [38]	290.5	526.9	762.5	997.9
MTA-4 [59]	108.7	206.1	303.3	381.4
MTA-5 [60]	64.50	93.07	125.9	132.5
MTA-6 [60]	81.84	126.3	143.3	163.7
Proposed MHTA	62.84	93.77	124.4	155.6
Proposed MTPA-1	51.06	75.48	101.2	131.4
Proposed MTPA-2	45.10	55.22	64.03	64.63
Proposed MTPA-3	49.08	65.02	76.34	88.25
Improv. in MHTA w.r.t. MTA-5	3%	1%	1%	-17%
Improv. in MTPA-1 w.r.t. MTA-5	20%	18%	19%	0%
Improv. in MTPA-2 w.r.t. MTA-5	30%	40%	49%	51%
Improv. in MTPA-3 w.r.t. MTA-5	23%	30%	39%	33%

Among the existing designs MTA-1, MTA-2, MTA-3 and MTA-4 are ripple-carry

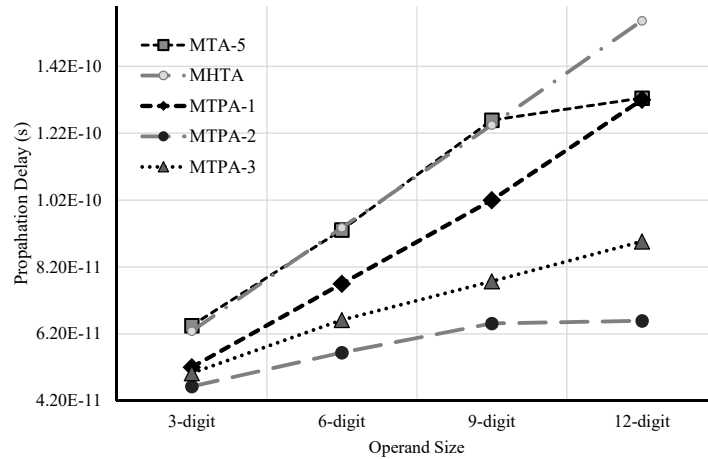


Figure 4.20: A Comparison of Propagation Delay for FO4 load

based and thus have large delay. Multi-digit ternary adder designs MTA-5, MTA-6 and proposed designs use Conditional sum, carry lookahead and prefix-based approaches respectively thus resulting in low delay.

Among the proposed designs, ternary adder MTPA-2, which uses Kogge-Stone [69] prefix network for carry computation, has least propagation delay. As expected, the design MHTA, which is ripple-carry based, has large delay when compared to other proposed designs. The proposed ternary adder MTPA-2 has up to 51% less delay when compared to MTA-5, which has least delay among existing designs. In Kogge-Stone prefix network, which is used in MTPA-2, the number of cells in the critical path is equal to $\log_2 N$, where N is operand length. Hence for linear increase in operand length, there is logarithmic increase in propagation delay of MTPA-2. Figure 4.20 illustrates the reduction in propagation delay for prefix-based designs when compared to MTA-5 for different operand sizes. This reduction is mainly due to the use of prefix-based networks in carry computation.

Different multi-digit adders have different complexity in the the final stage, due to which the propagation delay depends on the output load. To test this dependency, propagation delays of delay optimized designs are compared with that of proposed half-adder based ripple carry adder (MHTA) and ripple-based prefix adder (MTPA-1), which have large delay among proposed designs, under different load conditions. Table 4.5 shows the propagation delays for multi-digit adders with output load capacitance

Table 4.5: Delay for N -digit Adders for different output loads

N	3	6	9	12
Propagation Delay with a load of $1fF$ (in ps)				
MTA-1 [58]	68.47	121.5	175.2	228.0
MTA-5 [60]	120.3	158.0	190.7	205.4
MTA-6 [60]	117.5	163.2	179.3	200.1
MHTA	79.58	110.7	141.2	172.7
MTPA-1	68.86	93.91	118.8	149.4
Propagation Delay with a load of $2fF$ (in ps)				
MTA-1 [58]	74.95	128.2	181.3	234.6
MTA-5 [60]	198.3	251.7	287.2	313.5
MTA-6 [60]	159.5	206.6	221.8	242.2
MHTA	101.9	132.6	162.7	193.8
MTPA-1	91.12	115.8	140.3	170.2
Propagation Delay with a load of $3fF$ (in ps)				
MTA-1 [58]	81.45	134.3	188.2	242.0
MTA-5 [60]	277.0	347.0	378.9	421.9
MTA-6 [60]	210.9	255.9	264.6	284.1
MHTA	122.7	154.1	184.3	214.9
MTPA-1	111.9	137.3	161.9	191.3

of $1fF$, $2fF$ and $3fF$. Comparison is shown only with three designs (i.e MTA-1, MTA-5 and MTA-6), which have lower propagation delay when compared to other existing designs. As the load increases, MTA-1 and proposed designs have less delay when compared to MTA-5 and MTA-6. This is mainly due to the fact that MTA-5 and MTA-6 use a transmission gate-based 3 : 1 multiplexer, which does not supply enough current to the load limiting its driving capability. Ternary adder designs MHTA and MTPA-1 have lower delay (up to 40%) when compared to MTA-1 for FO4 load. Even as the load increases, the proposed designs have lower delay when compared to MTA-1, but only for larger operand size (> 3). This is because, for lower operand sizes the delay of low-power encoder (Shown in Figure 4.9) becomes a major contributor to the overall delay. As operand size increases, the delay-optimized carry propagation path dominates the overall propagation delay, thus resulting in reduced delay for proposed designs.

Figure 4.21 shows the analysis of variation in propagation delay with respect to change in load for different 12-digit adders. The propagation delay of MDA-5 is heavily dependent on the load variations because it uses array of 3 : 1 multiplexers with limited

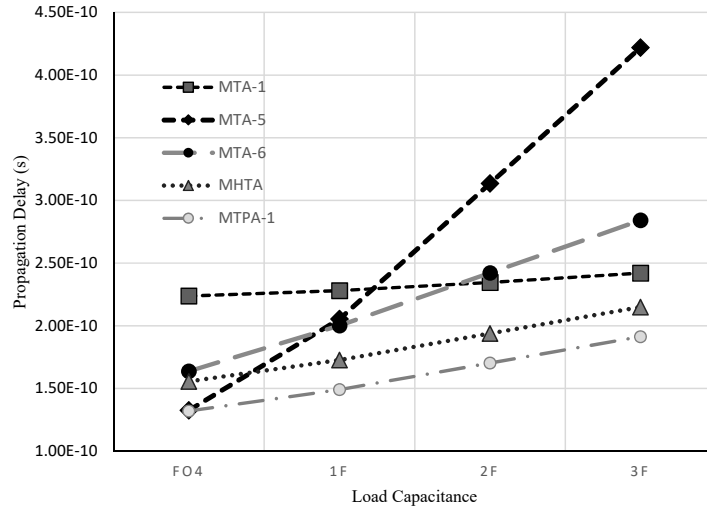


Figure 4.21: Propagation delay vs load for 12-digit adder

driving capability. Since MDA-6 uses 3 : 1 multiplexer only in final stage, its delay is less dependent on load variations when compared to that of MDA-5. Among all the designs, MDA-1, which uses a low-complexity encoder, has the best drive capability and hence its delay is least dependent on load variations. The proposed designs (MHTA, MTPA-1) use a low-power encoder, which is more complex than the encoder in MDA-1 but has a better driving capability when compared to 3 : 1 multiplexer. Hence delay of proposed designs is relatively more dependent on load variations when compared to that of MDA-1 but less dependent when compared to that of MDA-5 and MDA-6.

Table 4.6: Power-Delay Product for N -digit Adders

PDP (in fJ)				
N	3	6	9	12
MTA-1 [58]	3.26	12.2	22.9	47.5
MTA-2 [39]	0.80	3.43	7.34	13.4
MTA-3 [38]	1.66	6.42	13.4	22.6
MTA-4 [59]	3.14	12.6	25.4	44.6
MTA-5 [60]	0.43	1.24	2.54	3.59
MTA-6 [60]	1.24	3.80	6.29	11.0
Proposed MHTA	0.20	0.55	1.05	1.77
Proposed MTPA-1	0.15	0.41	0.76	1.32
Proposed MTPA-2	0.14	0.31	0.52	0.72
Proposed MTPA-3	0.15	0.36	0.60	0.93
Improv. in MHTA w.r.t. MTA-5	53%	56%	59%	51%
Improv. in MTPA-1 w.r.t. MTA-5	64%	67%	70%	63%
Improv. in MTPA-2 w.r.t. MTA-5	67%	75%	80%	80%
Improv. in MTPA-3 w.r.t. MTA-5	65%	71%	77%	74%

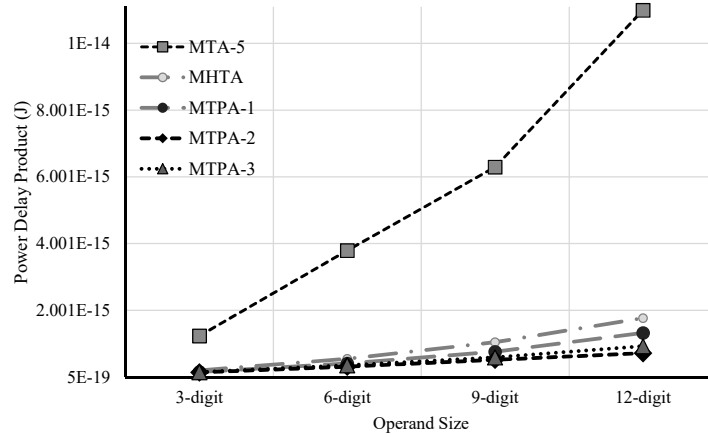


Figure 4.22: A Comparison of PDP

Table 4.7: Number of Transistors required for N -digit Adders

Transistor Count				
N	3	6	9	12
MTA-1 [58]	364	724	1084	1444
MTA-2 [39]	288	576	864	1152
MTA-3 [38]	396	792	1188	1584
MTA-4 [59]	234	468	702	936
MTA-5 [60]	357	798	1239	1726
MTA-6 [60]	541	1128	1877	2266
Proposed MHTA	385	760	1135	1510
Proposed MTPA-1	425	854	1283	1712
Proposed MTPA-2	439	938	1465	2020
Proposed MTPA-3	429	892	1373	1878

Table 4.6 presents the PDP for the existing and proposed multi-digit ternary adders with different operand sizes. The power-delay product is the product of worst-case propagation delay and average power consumption. The prefix-based designs result in up to 80% reduction in PDP when compared to MTA-5, which has least PDP among existing designs. Figure 4.22 shows the reduction achieved in PDP for different operand sizes. This reduction is due to: 1) reduced complexity of carry propagation technique when compared to other existing ones 2) use of a low-power encoder.

Table 4.7 compares the transistor count, which gives an indication of area, for the proposed and existing adders. Among the existing designs MTA-1, MTA-2, MTA-3 and MTA-4 are ripple-carry based and thus require lesser number of transistors when compared to MTA-5 (CSA) and MTA-6 (CLA). The proposed prefix based designs re-

quire more transistors for implementation when compared to the existing ripple carry adders. This is mainly due to the extra half-adder stage which is required to transform the inputs and enable the use of Propagate-generate technique. However, when compared to the existing Carry-lookahead adder, MTA-6, the proposed CLA based prefix adder requires up to 20% less transistors for its implementation. Though the prefix based adders perform better with respect to power consumption, propagation delay and PDP, they require up to 25% more CNFETs for implementation when compared to half-adder based ternary adder.

4.6 Conclusions

In this chapter we presented two techniques to implement the CNFET-based multi-digit ternary adders. The first design technique results in a ternary adder which is ripple-carry based and uses half-adder, delay optimized carry generator, a sum generator and low-power encoders. In this ternary adder, reduction in delay has been achieved in multi-digit adders by optimizing the carry propagation path and reduction in power has been achieved by using a low-power encoder. The second technique enabled the use of carry Propagate-Generate concept in ternary addition resulting in multi-digit ternary prefix adders. Three different ternary prefix adders, which use ripple-based, Kogge-Stone and carry lookahead-based prefix networks, have been implemented using CNFETs.

Existing and the proposed multi-digit adders with varied operand sizes have been implemented in HSPICE. Simulation results show that there is a significant reduction in power consumption (up to 52%) and PDP (up to 58%) in the half-adder based ternary adders when compared to the existing adders. Results also showed a reduction in FO4 delay (up to 30%) for proposed adder when compared to other ripple-carry multi-digit adders. For ternary prefix adders, simulation results show that there is reduction in power consumption (up to 58%), propagation delay (up to 50%) and PDP (up to 80%) in the proposed prefix-based ternary adders when compared to the existing adders.

Analysis of design metrics of proposed ternary adders showed that there is signifi-

cant reduction in power consumption mainly due to the use of low power encoder. A ternary encoder is a critical element and contributes significantly to the overall power consumption of the ternary circuit. Hence in the next chapter we develop new designs for ternary encoders. These designs are used to develop encoder based optimization algorithms which choose appropriate encoder for different outputs of a multi-output ternary logic circuit to optimize the circuit with respect to different design metrics.

Chapter 5

Encoder-based Optimization of Ternary Circuits

5.1 Introduction

As explained in the earlier chapters, an encoder generates logic 2 and logic 0 by a direct connection to VDD and GND respectively. But for generation of *logic 1* at output, a direct path from VDD to GND is created resulting in large static current. If this direct path has low resistance it leads to lower delay and large power consumption. else it leads to large delay and low power consumption. Hence the choice of encoder effects the overall propagation delay and power consumption of ternary circuit. There is need for encoders which are optimized for different design constraints. There is also a need to develop a methodology to choose appropriate encoders such that the design constraints are met..

This chapter presents improved encoder designs which are used in implementation of ternary logic circuits. A detailed analysis is carried out on encoders to understand the effect of using CNFETs with CNTs of different diameter on the overall propagation delay and power consumption of encoder. Based on this analysis, algorithms are presented, which map appropriate encoders for different output stages of a multi-output ternary logic circuit in such a way that design constraints are met. A ripple-carry based

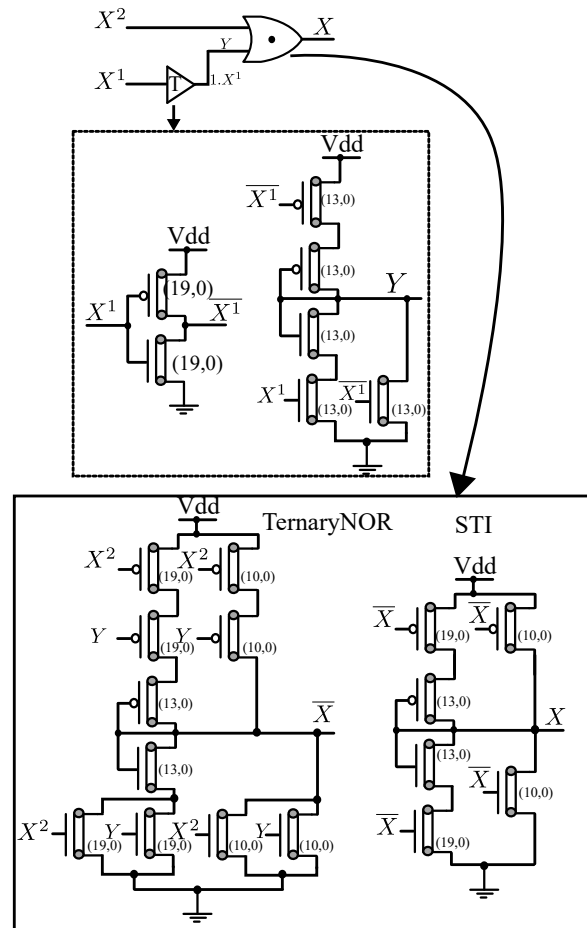
ternary adder is taken as an example and the proposed algorithms are used to obtain different encoder mapping resulting in ternary adder designs which are optimized either for power consumption, propagation delay or Power-Delay Product (PDP). These designs are compared with other ternary ripple-carry based adders in literature with respect to different design parameters.

5.2 Review of Ternary Encoders

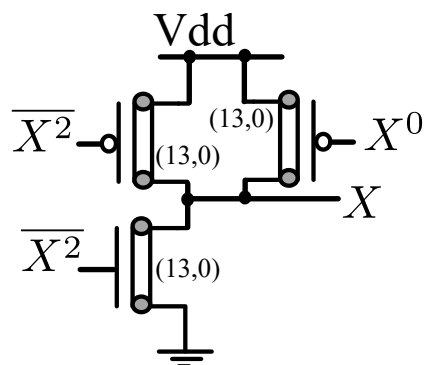
The design technique presented in [36] uses a ternary decoder in the first stage to generate binary versions of inputs. The ternary decoder is a one-input, three-output circuit and generates unary functions for an input X . The relation between ternary input X and decoder outputs (indicated by X^0, X^1, X^2) is given by

$$X^k = \begin{cases} 2, & \text{if } X = k \\ 0, & \text{if } X \neq k \end{cases} \quad (5.1)$$

These decoder outputs can take only two logic values i.e., *logic 2* and *logic 0*, corresponding to *logic 1* and *logic 0* in binary logic. The decoder outputs are used to compute intermediate binary outputs with the help of binary logic gates. Finally ternary outputs are generated from binary signals using a combination of ternary buffers and/or encoder. This technique has been used in implementation of multi-digit ternary adder presented in [58]. This adder used a different encoder which has reduced complexity when compared to the design presented in [36]. One of the major disadvantages of the encoder design that is presented in [58] is the existence of low resistance path between VDD and GND while generating *logic 1*. This results in a large static current and hence large static power consumption. Although the encoder used in [36] has high resistance path between VDD and GND while generating *logic 1*, there exist multiple such paths leading to large power consumption as well as large propagation delay.



(a) Encoder presented in [36]



(b) Encoder presented in [58]

Figure 5.1: Ternary Encoders

Although the encoder used in [58] has reduced delay and complexity, it consumes large power resulting in multi-digit adders with very high power consumption. Figures 5.1(a) and 5.1(b) show the encoders presented in [36] and [58] respectively.

5.3 Proposed CNFET-based Ternary Encoders

Encoder is a critical element in the design of ternary logic circuits and is used to convert intermediate binary signals to final ternary outputs. An encoder generates *logic 2* or *logic 0* by a direct connection to VDD or GND respectively. But for generation of *logic 1* at output, a direct path from VDD to GND is created. The existing encoder designs either consume large power or require large number of transistors for their implementation. Improved encoders, which use transistors of same chiralities have been presented in Section 3.3. Figure 5.2 shows the implementation of the proposed encoder, where presence of additional diode connected P-CNFET and N-CNFET transistors create a high resistance path between VDD and GND to generate *logic 1* at the output. This high resistance path, while limiting the static current, thereby reducing the static power consumption, however, causes the increase in encoder delay when compared to encoder in [58]. Also unlike the encoder used in [36], the improved encoder has only one direct path between VDD and GND while generating *logic 1*.

The encoder shown in Figure 5.2 is used to generate the ternary output (X) from binary inputs $\overline{X^2}$ and X^0 . In this circuit, if X^0 is equal to *logic 2* then transistor M6 switches *ON* and *logic 0* is obtained at the output. Instead, if $\overline{X^2}$ is equal to *logic 0* (i.e. X^2 is *logic 2*) then M3 will be *ON* pulling output node to *logic 2*. If neither of X^0 and X^2 are *logic 2* then a direct path from VDD to GND is created (via transistors M1-M2-M4-M5) causing *logic 1* to appear at X .

The encoder presented in Figure 5.2 is further improved by varying the resistance of path which is created between VDD and GND while generating *logic 1* at the output. If the resistance of $VDD - GND$ path is increased then it will result in an encoder which is optimized for power consumption. The resistance of $VDD - GND$ path is increased by adding additional diode connected transistors in series. Figure 5.3 shows

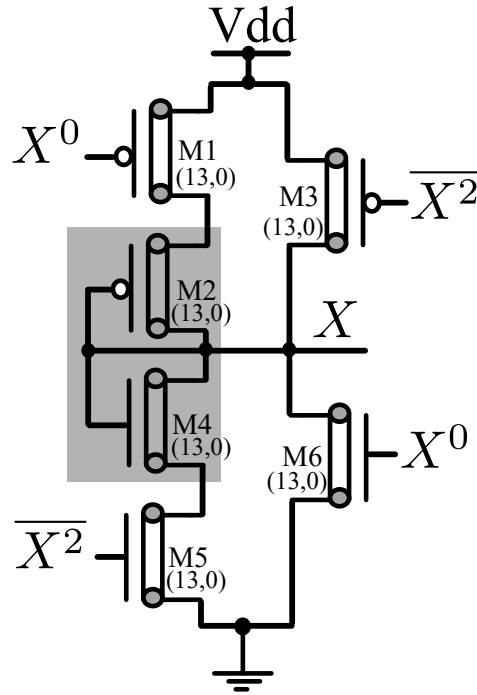
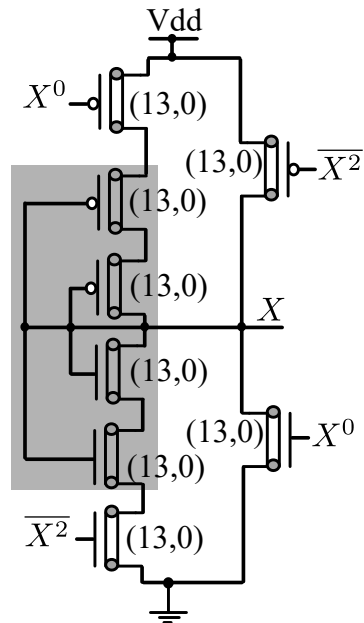
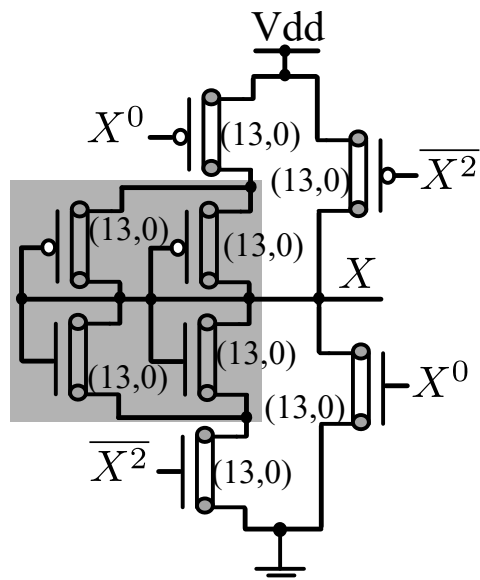


Figure 5.2: Proposed Encoder (with inputs $\overline{X^2}$ and X^0)

the implementation of the encoder (with inputs $\overline{X^2}$ and X^0), which is optimized for power consumption. Although, the high resistance path between $VDD - GND$ results in reduced power consumption, it causes increase in propagation delay of encoder. Another encoder which is optimized for propagation delay is presented in Figure 5.4. In this design, diode connected transistor pair are connected in parallel to achieve low resistance in $VDD - GND$ path of the encoder.

The proposed encoders are compared with the other existing encoder designs [36, 58]. Since the outputs of encoder are ternary in nature, FO4 delay is calculated by connecting four STI gates (as load) at the outputs. Power consumption is measured by simulating all the encoders with a random pattern such that they generate same output. In all encoders designs, a direct path from VDD to GND is created while generating *logic 1* resulting in large static power. This is the reason due to which encoder power is the major contributor to the overall power consumption of ternary logic circuits.

Different encoder designs are implemented in HSPICE using the simulation environment presented in Section 3.4.1. Table 5.1 shows the comparison of different encoders with respect to propagation delay, power consumption and PDP. The encoder design

Figure 5.3: Proposed Low-Power Encoder (with inputs X^2 and X^0)Figure 5.4: Proposed Low-Delay Encoder (with inputs X^2 and X^0)

presented in [36] (shown in Figure 5.1(a)), which uses a level-shifter and a ternary *NOR* gate, is indicated as Encoder-1. This design has moderate power consumption and large delay when compared to other designs. A low-complexity encoder design, which is presented in [58] (shown in Figure 5.1(b)), is indicated as Encoder-2. This design has least propagation delay and large power consumption due to the low resistance $VDD - GND$ path. The proposed encoders use additional diode connected transistors to create a high resistance path between VDD and GND to generate *logic 1* at the output. This high resistance path results in encoder with lower power consumption but higher propagation delay when compared to the encoder presented in [58]. As seen in Table 5.1, proposed encoders achieve 44 – 68% reduction in propagation delay, 45 – 80% reduction in power and 82 – 89% reduction in power-delay product when compared to the encoder presented in [36]. When compared to the encoder presented in [58], the proposed encoders result in 94 – 97% reduction in power and 90 – 94% reduction in power-delay product. However, proposed encoders have 57 – 175% more delay when compared to Encoder-2 [58] which has least delay among existing designs.

Table 5.1: A Comparison of Encoders

Encoder	(FO4) Delay (in ps)	Power (in μW)	PDP (in $10^{-18} J$)
Encoder-1 [36] (Fig. 5.1(a))	25.2 (100%)	0.99 (100%)	25.2 (100%)
Encoder-2 [58] (Fig. 5.1(b))	5.09 (20.20%)	9.31 (940%)	47.3 (187%)
Proposed Encoder-1 (Fig. 5.2)	10.33 (41.0%)	0.33 (33.33%)	3.40 (13.49%)
Proposed Encoder- 2 (Low-Power Encoder) (Fig. 5.3)	14.03 (55.67%)	0.19 (19.19%)	2.68 (10.63%)
Proposed Encoder- 3 (Low-Delay Encoder) (Fig. 5.4)	8.03 (31.87%)	0.54 (54.55%)	4.34 (17.22%)

5.4 Effects of varying chiralities of CNFETs that are used in Encoders

In this section, a detailed analysis is carried out on encoders to understand the effect of using CNFETs with CNTs of different diameter on the overall propagation delay and power consumption of the encoder. The effect of chirality variations on an N-CNFET is studied with the help of the $I - V$ characteristics of transistor, which are simulated in HSPICE using the CNTFET model in [53]. The CNFET is configured to have three CNTs (all with same chirality) and a default pitch value equal to $20nm$. Figure 5.5 shows the $I - V$ characteristics for a V_{GS} of $0.45V$, where x -axis indicates the drain-to-source voltage (V_{DS}) and y -axis indicates the drain current (I_{DS}). As seen from this figure, for a fixed V_{GS} , the drain current (I_{DS}) is proportional to the diameter of CNTs, which in turn is proportional to value of n in chirality vector (see Table 2.3). The drain current (I_{DS}) of transistors, which are used in implementation of ternary logic circuits, directly effects the power consumption and propagation delay of the circuits.

In the proposed encoders (shown in Figures 5.2, 5.3 and 5.4) and existing encoder [58] (shown in Figure 5.1(b)) transistors of chirality $(13, 0)$ (Diameter of CNTs $1.018nm$) have been used. However, transistors with any chirality (shown in Table 2.3) can be used in the implementation of these encoders. For a detailed analysis, different encoder designs with different chiralities are implemented in HSPICE using the CNFET model of [53]. All encoders operate at $0.9V$ power supply and room temperature. The CNFETs used in the implementation are configured to have three CNTs (all with same chirality) and a default pitch value equal to $20nm$. Power consumption results are obtained by simulating the circuits with random input patterns at switching frequency of $500MHz$. Propagation delay results for different circuits are obtained by finding worst case Fan-Out of 4 (FO4) delay. The encoder design which is shown in Figure 5.1(a) is excluded from this analysis because it uses ternary *OR* gate and *STI*, which requires transistors of specific chirality.

Figures 5.6 and 5.7 show the comparison of different encoders, which use transistors

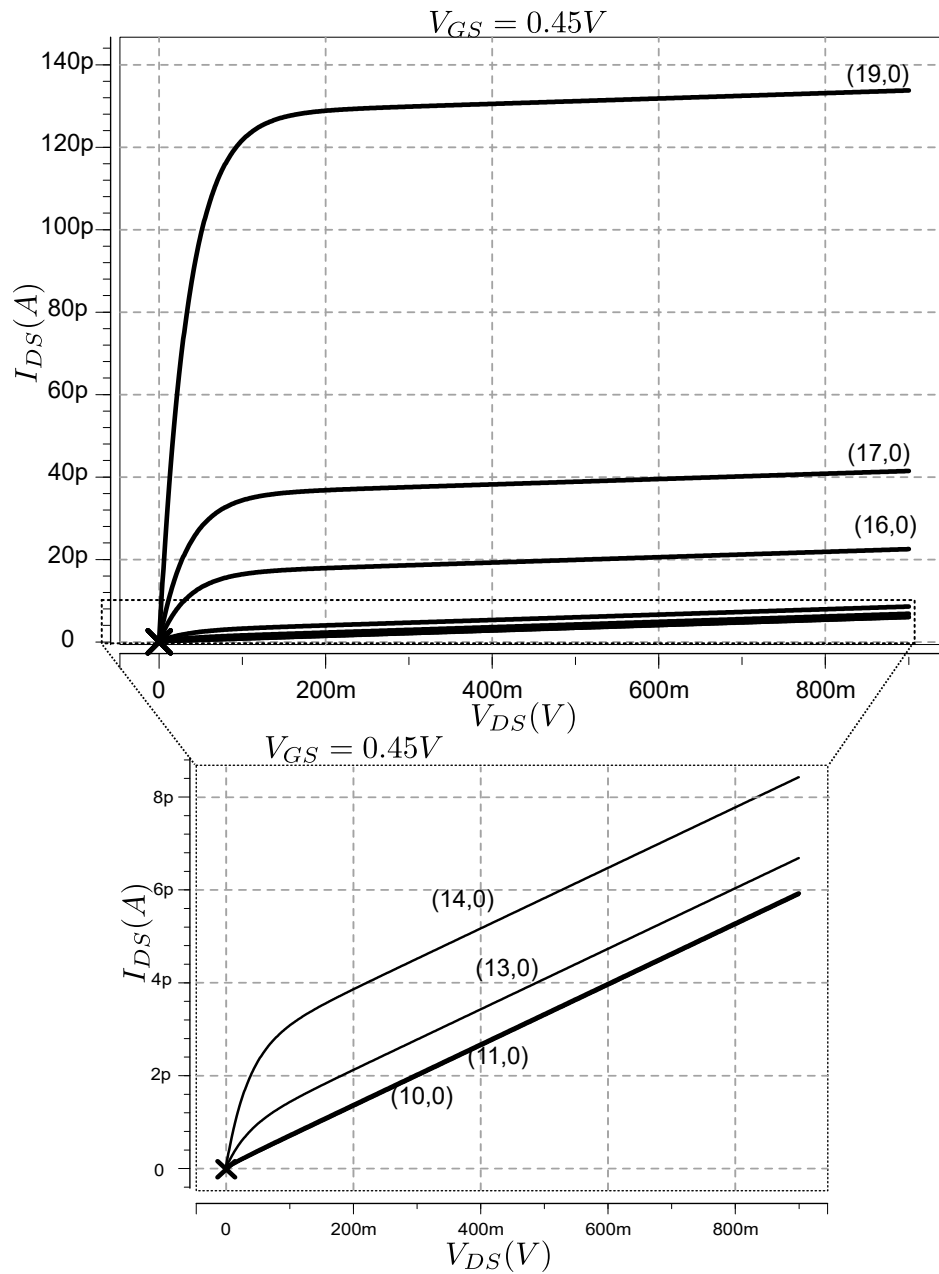


Figure 5.5: I-V Characteristics of N-CNFET

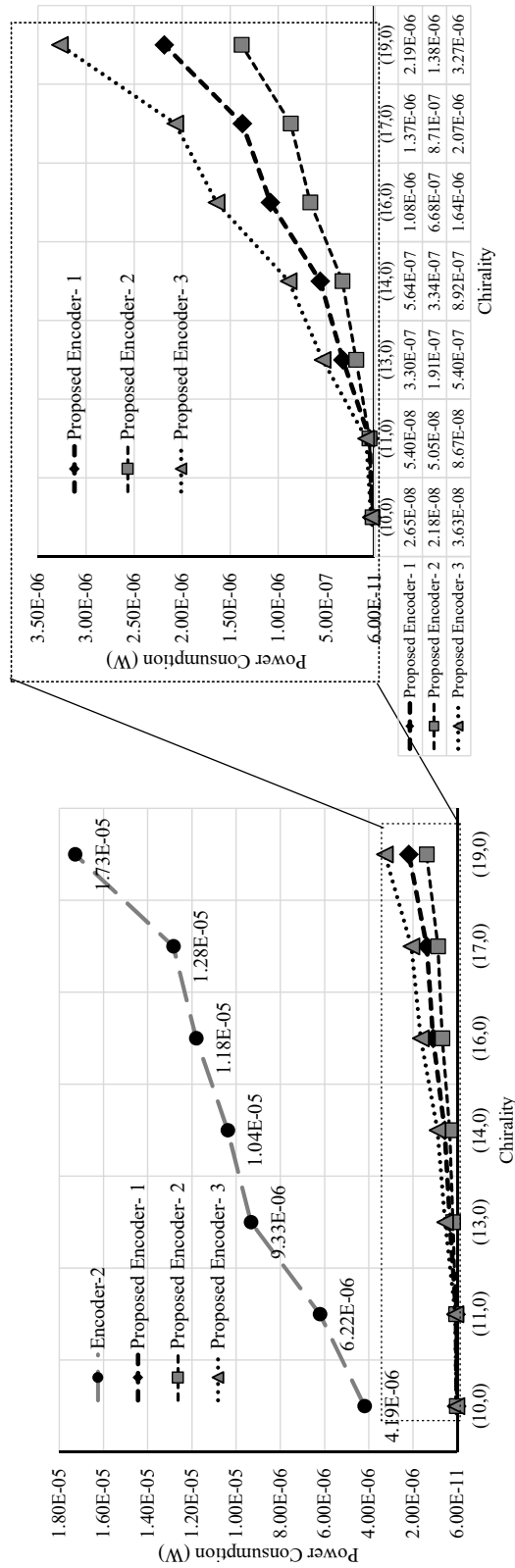


Figure 5.6: Power Consumption for Encoders with Transistors of Different Chirality

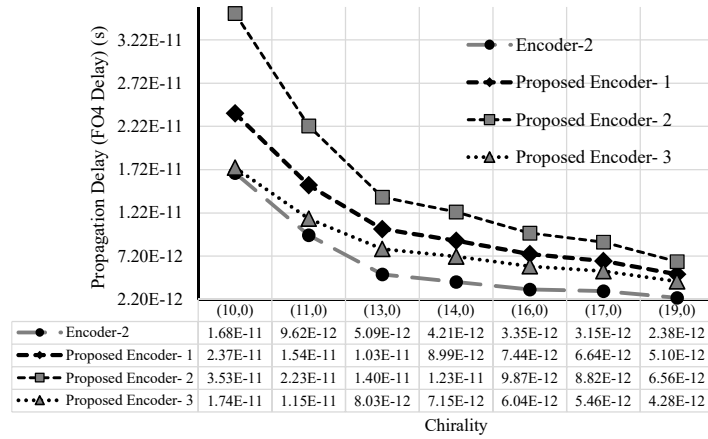


Figure 5.7: Propagation Delay for Encoders with Transistors of Different Chirality

of different chirality (i.e. transistors with CNTs of different diameter), with respect to power consumption and propagation delay respectively. The chiralities (12, 0), (15, 0) and (18, 0) are not considered because they result in metallic CNTs. Figure 5.6 shows the variation in power consumption of the encoders with respect to variation in chirality vector of transistors. As seen from this figure, the power consumption is directly proportional to the value of n in chirality vector of transistors used in encoder. Figure 5.7 shows the variation in propagation delay of the encoders with respect to variation in chirality vector of transistors used in encoder. The propagation delay decreases as the n value of chirality vector of transistors (diameter of CNTs) increases. The increase in power consumption and decrease in propagation delay due to increase in n value of chirality vector of transistors is mainly because of increase in drain current (I_{DS}) of transistors.

The analysis of design parameters of encoders with respect to variations in transistor chirality show that there are 28 different encoders corresponding to 7 chirality variations for one existing [58] and three proposed designs. These encoders have a trade-off between propagation delay and power consumption i.e. encoders with low propagation delay have large power consumption and vice-versa. Hence there is a need for a methodology or an algorithm, which chooses appropriate encoders for generating different ternary outputs in multi-output ternary logic circuit, such that the overall circuit is optimized with respect to power, delay and/or PDP.

5.5 Algorithms for choosing appropriate Encoders in Ternary Circuits

5.5.1 Problem Formulation

The existing methodology, which is used to implement ternary circuit, has three main stages [36]. The first stage uses a ternary decoder to convert ternary signal into mutually exclusive binary signals which are given as inputs to binary computation stage. The outputs of binary stage are converted into ternary output using encoders. This ternary circuit can be modeled as a directed acyclic graph (*ternary circuit graph*), which is represented as $G = (V, E)$ where V is vertex (or node) set and E is edge set. Edge e_{ij} is called an outgoing edge with respect to node v_i and an incoming edge with respect to node v_j . In the graph (V, E) , Primary Inputs (*PIs*) are nodes with no incoming edges and Primary Outputs (*POs*) are the nodes with no outgoing edges. In addition to these nodes there are additional special nodes defined for *ternary circuit graph* namely Decoder nodes (*DNs*) and Encoder nodes (*ENs*). Decoder nodes are the nodes which have one incoming edge that is connected to primary input and any number of outgoing edges. Encoder nodes are the nodes which have one outgoing edge that is connected to primary output and any number of incoming edges. A directed path p in graph (V, E) is a sequence (or set) of nodes (from primary input to primary output) which are connected by directed edges. A set of all the paths in the graph is represented by P .

In the graph (V, E) , associated with each node $v_i \in V$, there is a delay variable $dv_i \geq 0$, whose value represents the propagation delay. The delay for all primary inputs and outputs is equal to zero, i.e. $dv_i = 0 \forall v_i \in PI \cup PO$. The delay of path $p \in P$ is summation of delays of all the nodes in the path i.e. $\sum_{v_i \in p} dv_i$. The path which has largest delay among all the paths in the graph is called *critical path* (p_c).

In the graph (V, E) , the incoming edge of decoder node (which is outgoing edge for primary input) and the outgoing edge of a encoder node (which is incoming edge for primary output, represents a ternary signal. The nodes $v_i \in V - (PI \cup PO \cup DN \cup EN)$

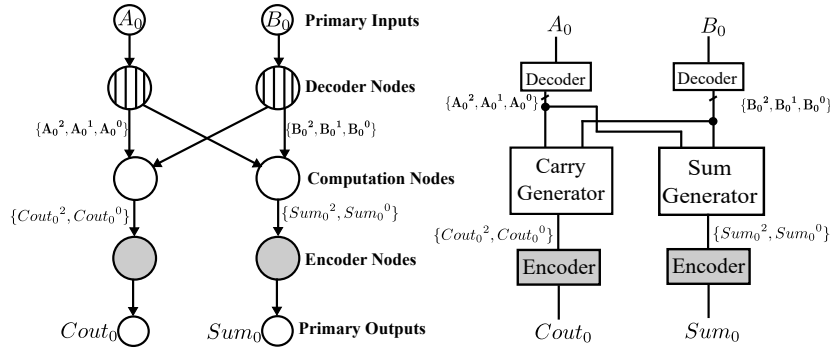


Figure 5.8: Ternary Circuit as a Graph

represent binary computation elements (a complex gate or combination of gates) whose incoming edges and outgoing edges represent a group of mutually exclusive binary signals (X^0, X^1 , and X^2) corresponding to a ternary signal (X). Figure 5.8 shows an example of a ternary half-adder and its graph representation.

In ternary logic circuit, encoder is a critical element and is used to convert intermediate binary signals to final ternary outputs. One of the major disadvantages of the ternary encoder is the existence of low resistance path between VDD and GND while generating *logic* 1. This results in a large static current and hence large static power consumption. Thus the average power consumption of ternary circuit can be approximated as the sum of average power consumption of encoders used in the circuit. Hence in addition to variable dv_i , associated with encoder nodes $v_i \in EN$, there is a variable $pv_i > 0$, whose value represents the power consumption. Since any one of the encoders with chirality variations (presented in Section 5.4) can be used in implementation of ternary circuit, let there be a set ED which consists of different encoder designs. In this work, the elements of set ED are represented using notation $ed_{(i,n)}$, where $i = 1, 2, 3, \text{ or } 4$ represents existing encoder-2 [58], proposed encoder-1, proposed encoder-2 (low-power encoder) or proposed encoder-3 (low-delay encoder) respectively and n is equal to the n value of chirality vector (refer Section 2.2) of transistors used in encoder. For example $ed_{(1,13)}$ represents encoder-2 [58] that is implemented using transistors of chirality (13, 0). Each element $ed_{(i,n)}$ of this set is associated with two values namely $ded_{(i,n)}$ and $ped_{(i,n)}$ representing the delay and power consumption of the encoder designs. When an element $ed_{(i,n)} \in ED$ is selected as (mapped to) encoder

node $v_j \in EN$ then the values $ded_{(i,n)}$ and $ped_{(i,n)}$ are assigned to variables dv_j and pv_j respectively.

The problem of selecting suitable encoder to optimize design parameters of a ternary circuit is formulated as below:

Problem Statement: Given a ternary circuit graph $G = (V, E)$ for which path delays, i.e. $\sum_{v_i \in p-EN} dv_i \forall p \in P$, are known and set of encoder designs ED are available, map $ed_{(i,n)} \in ED$ to $v_j \in EN$ such that to optimize power,

$$\min\left(\sum_{v_i \in EN} pv_i\right) \quad (5.2)$$

to optimize delay,

$$\min\left(\sum_{v_i \in p_c} dv_i\right) \quad (5.3)$$

$$\sum_{v_i \in p} dv_i \leq \sum_{v_i \in p_c} dv_i \quad \forall p \in P - \{p_c\} \quad (5.4)$$

$$\min\left(\sum_{v_i \in EN \cap (P - \{p_c\})} pv_i\right) \quad (5.5)$$

or to optimize PDP

$$\min\left(\sum_{v_i \in p_c} dv_i \times \sum_{v_i \in EN} pv_i\right) \quad (5.6)$$

Here equations (5.2-5.6) specify the constraints on the encoder mapping such that the resulting ternary circuit, which is represented by the ternary graph, is optimized for power consumption, propagation delay or PDP. The following sections present algorithms for selecting suitable encoder for a ternary circuit while optimizing different design parameters.

5.5.2 Power optimization

Since the average power consumption of ternary circuit can be approximated as the sum of average power consumption of encoders used in the circuit, for optimization with respect to power, encoders should be selected such that the sum of average power consumption of encoders, which are used in the circuit, is minimum. This optimization, which is specified by equation (5.2), is implemented using Algorithm 5.1.

Algorithm 5.1 Algorithm for Power Optimization

```

1: Inputs:  $(V, E)$  with  $\sum_{v_i \in p-EN} dv_i \forall p \in P, ED$ 
2: Output:  $(V, E)$  with values for  $dv_i, pv_i \forall v_i \in EN$ 
3: begin
4:  $P' = \text{maxpaths\_oneperencoder}(P)$ 
5:  $ed' = \text{get\_Encoder\_MinPower}(ED)$ 
6: for each  $p \in P'$  do
7:    $\text{map\_ED\_EN}(p, ed')$ 
8: end for
9: return  $(V, E)$ 
10: end

```

The inputs to Algorithm 5.1 are ternary circuit graph (V, E) (where all the delays of nodes are known) and the list of encoder designs with power and delay values for each. Initially the power (pv_i) and delays (dv_i) variables of all encoder nodes, i.e. $v_i \in EN$, are set to default values. Since an encoder node might exist in multiple paths (e.g. in Fig 5.8, each encoder node is part of two different paths), a function $P' = \text{maxpaths_oneperencoder}(P)$ is used to identify a path which has maximum delay (excluding encoder delay) among all paths which have same encoder node. This function returns a set P' which consists of maximum delay paths (one per encoder node) corresponding to the respective encoder nodes. A function $\text{get_Encoder_MinPower}(ED)$ gets the encoder ed' which has least power consumption among the list of encoder designs (ED). This least power encoder is mapped to encoder nodes for all the paths such that each path has one encoder mapped. The output of Algorithm 5.1 is a graph (V, E) with mapping information for all encoder nodes. As seen from the analysis in Section 5.4, the proposed encoder-2 (Figure 5.3) which uses transistors of chirality $(10, 0)$ (i.e. $ed_{(3,10)}$) has least power consumption. Hence this encoder is used to generate ternary

output for all the paths of ternary circuit.

5.5.3 Delay Optimization

The delay refers to the propagation delay of the critical path in the ternary circuit. As seen from Figures 5.6 and 5.7, the existing and proposed encoders have a trade off between propagation delay and power consumption. Hence to achieve optimization with respect to delay, encoders with least delay (and large power consumption) can be used in critical paths whereas encoders with low power consumption (and large delay) can be used in non-critical paths of ternary circuit. This optimization, which is specified by equations (5.3-5.5), is implemented using Algorithm 5.2.

Algorithm 5.2 Algorithm for Delay Optimization

```

1: Inputs:  $(V, E)$  with  $\sum_{v_i \in p-EN} dv_i \forall p \in P, ED$ 
2: Output:  $(V, E)$  with values for  $dv_i, pv_i \forall v_i \in EN$ 
3: begin
4:  $P' = \text{maxpaths\_oneperencoder}(P)$ 
5:  $Max\_pathDelay = 0$ 
6: for each  $p \in P'$  do
7:   if  $\sum_{v_i \in p-EN} dv_i > Max\_pathDelay$  then
8:      $Max\_pathDelay = \sum_{v_i \in p-EN} dv_i$ 
9:      $p_c = p$ 
10:  end if
11: end for
12:  $ed' = \text{get\_Encoder\_MinDelay}(ED)$ 
13:  $map\_ED\_EN(p_c, ed')$ 
14:  $Max\_Delay = Max\_pathDelay + \text{delay}(ed')$ ,
15:  $ED' = \text{sort\_ascending\_power}(ED)$ 
16: for each  $p \in P' - \{p_c\}$  do
17:   for each  $ed_{(i,n)} \in ED'$  do
18:      $path\_delay = \sum_{v_i \in p-EN} dv_i + \text{delay}(ed_{(i,n)})$ 
19:     if  $path\_delay \leq Max\_Delay$  then
20:        $map\_ED\_EN(p, ed_{(i,n)})$ 
21:       break
22:     end if
23:   end for
24: end for
25: return  $(V, E)$ 
26: end

```

The inputs to Algorithm 5.2 are ternary circuit graph (V, E) (where all the delays of nodes are known) and the list of encoder designs with power and delay values for each.

As explained earlier (in Section 5.5.2), the function $P' = \text{maxpaths_oneperencoder}(P)$ returns a set P' which consists of maximum delay paths corresponding to the each of the encoder nodes. All the paths in P' are traversed and the path with largest delay in the graph (excluding the encoder delay), i.e. p_c , is found along with its delay Max_pathDelay . The encoder node in path p_c is mapped with an encoder design ed' , which has least delay, using function $\text{get_Encoder_MinDelay}(ED)$. The function $\text{delay}(ed')$ returns the delay of encoder design ed' . The critical path delay (including the encoder delay) is now referred to as Max_Delay . The set ED is now sorted using function $\text{sort_ascending_power}(ED)$ such that the resulting set, ED' , has encoder designs which are arranged in increasing order of their power consumption. For all the remaining paths, $p \in P' - \{p_c\}$, encoders are selected from ED' and mapped to encoder nodes such that the encoder design has least power while restricting the delay of these paths to a value that is less than or equal to Max_Delay . As seen from Figures 5.3 and 5.4, encoder-2 [58] that is implemented using transistors of chirality $(19, 0)$ ($ed_{(1,19)}$), has least propagation delay, but has large power consumption when compared to other proposed encoders. Hence this encoder is used in the critical path of the circuit. For non-critical paths, encoders are mapped such that the encoder design has least power while restricting the delay of these paths to value which is less than critical path delay.

5.5.4 PDP Optimization

Power-Delay Product of a circuit is defined as the product of worst-case propagation delay (i.e. delay of critical path) and the average power consumption of the circuit. Since majority of the power consumption happens in the encoder, PDP can be approximated as the product of critical path delay and sum of average power consumption of encoders used in the circuit. The optimization with respect to PDP is given as equation (5.6) and is implemented using Algorithm 5.3. The inputs to Algorithm 5.3 are ternary circuit graph (V, E) (where all the delays of nodes are known) and the list of encoder designs with power and delay values for each.

Algorithm 5.3 Algorithm for PDP Optimization

```

1: Inputs:  $(V, E)$  with  $\sum_{v_i \in p-EN} dv_i \forall p \in P, ED$ 
2: Output:  $(V, E)$  with values for  $dv_i, pv_i \forall v_i \in EN$ 
3: begin
4:  $optimize\_power((V, E), ED)$ 
5:  $ED' = sort\_ascending\_power(ED)$ 
6:  $ed' = get\_Encoder\_MinPower(ED')$ 
7:  $P' = maxpaths\_oneperencoder(P)$ 
8:  $p_c = get\_critical\_path(P')$ 
9:  $pdp_{new} = (\sum_{v_i \in p_c} dv_i \times \sum_{v_i \in EN} pv_i)$ 
10: repeat
11:    $pdp_{old} = pdp_{new}$ 
12:    $EN' = Save\_Mapping(EN, P')$ 
13:    $ed' = get\_one\_lower(ED', ed')$ 
14:    $map\_ED\_EN(p_c, ed')$ 
15:    $Max\_Delay = \sum_{v_i \in p_c} dv_i$ 
16:   for each  $p \in P' - \{p_c\}$  do
17:     for each  $ed_{(i,n)} \in ED'$  do
18:        $path\_delay = \sum_{v_i \in p-EN} dv_i + delay(ed_{(i,n)})$ 
19:       if  $path\_delay \leq Max\_Delay$  then
20:          $map\_ED\_EN(p \cap EN, ed_{(i,n)})$ 
21:         break
22:       end if
23:     end for
24:   end for
25:    $p_c = get\_critical\_path(P')$ 
26:    $pdp_{new} = (\sum_{v_i \in p_c} dv_i \times \sum_{v_i \in EN} pv_i)$ 
27: until  $pdp_{old} < pdp_{new}$ 
28:  $map\_saved(EN', P')$ 
29: return  $(V, E)$ 
30: end

```

Initially the function $optimize_power((V, E), ED)$, which implements Algorithm 5.1, is used to map all the encoder nodes to encoder design with least power consumption. As explained earlier (in Section 5.5.2), the function $P' = maxpaths_oneperencoder(P)$ returns a set P' which consists of maximum delay paths corresponding to the each of the encoder nodes. Functions $sort_ascending_power(ED)$ and $get_Encoder_MinPower(ED')$ return list of encoder designs ED' sorted in order of increasing power consumption and design with least power consumption ed' . The set of maximum delay paths P' corresponding to the each of the encoder nodes is computed by using $maxpaths_oneperencoder(P)$. The critical path among paths in P' is found using function $get_critical_path(P')$ and the approximate power-delay product (product of critical path delay and summation

of power consumption of encoders) is calculated.

The mapping information of the encoder nodes is saved by using function $EN' = Save_Mapping(EN, P')$. Now the least power encoder design in the critical path p_c is replaced with a design which has slightly higher power consumption (i.e. second encoder design in ED') and lower delay. For all the remaining paths, $p \in P' - \{p_c\}$, encoder designs from ED' are mapped to encoder nodes such that the encoder design has least power while restricting the delay of these paths to a value that is less than or equal to delay of path p_c . The new value of approximate power-delay product is compared with earlier value to see if there is increase in PDP. This process is repeated as long as the approximate power-delay product keeps reducing. Once the approximate PDP starts increasing the mapping information of encoder nodes with least PDP is mapped back to EN by using function $map_saved(EN', P')$.

5.6 Example: Encoder-based Optimization of Multi-Digit Ternary Adder

The optimization algorithms presented in Section 5.5 are applied on ternary adder, which is implemented using the methodology presented in [58]. Figure 5.9 shows the multi-digit (N -digit) adder design where $A(A_{N-1}...A_1A_0)$, $B(B_{N-1}...B_1B_0)$, Cin are ternary inputs and $Sum(Sum_{N-1}...Sum_1Sum_0)$, $Cout$ are ternary outputs. $Cout_{N-2}...Cout_0$ represent intermediate ternary carries. Here X_i^j corresponds to i^{th} digit-adder stage whose value is either *logic 2* (if $X = j$) or *logic 0* (if $X \neq j$), where $j \in \{0, 1, 2\}$. For example A_0^1 corresponds to input of 0^{th} digit-adder stage whose value is *logic 2* only if ternary signal A is equal to *logic 1*. Also, $\{\}$ is used to represent a group of binary signals and its complements. In multi-digit adder of the same work, the sum generation is similar to other existing designs, whereas carry generation/propagation is optimized for delay by avoiding redundant encoder-decoder pairs. At each digit-adder stage, binary carry signals (i.e. $Cout_i^2, Cout_i^0$) are generated using a low-delay carry generator block. Unlike other multi-digit adder designs where ternary carries are

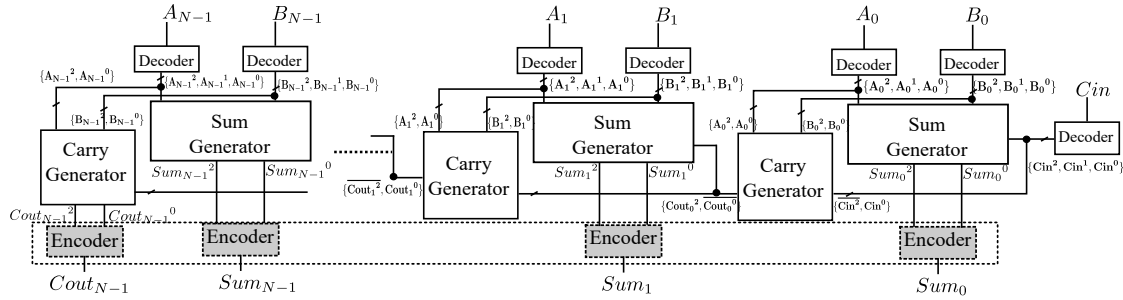


Figure 5.9: Multi-digit adder presented in [58].

generated for every digit-adder stage, design in [58] computes ternary carry ($Cout_i$) only for the final (i.e. $N - 1^{th}$) digit-adder stage.

The decoder, sum generator and the carry generator blocks are implemented as presented in [58]. HSPICE simulations are carried out without encoders and the worst-case delays for different blocks and different paths are tabulated. The delays of different blocks along with the delays and power consumption of different encoder designs (in Figures 5.6 and 5.7) are given as inputs to the algorithms which are presented in Section 5.5. Algorithms 5.1 – 5.3 have been implemented using Python. The output of these algorithms is the encoder mapping information, which specifies the encoder designs that are to be used for generating different outputs such that the ternary adders are optimized for delay, power consumption and PDP.

Table 5.2 specifies the encoder mapping information, which is result of Algorithms 5.2 and 5.3, for 3-digit, 6-digit and 9-digit ternary adders. Here different encoder designs are represented using notation $ed_{(i,n)}$, where $i = 1, 2, 3, \text{ or } 4$ represents existing encoder-2 [58], proposed encoder-1, proposed encoder-2 (low-power encoder) or proposed encoder-3 (low-delay encoder) respectively and n is equal to the n value of chirality vector (refer Section 2.2) of transistors used in encoder. For power optimization the encoder design $ed_{(3,10)}$ is used to generate all the outputs of ternary adders.

The mapping information in Table 5.2 is used to implement the delay optimized, power optimized and PDP optimized ternary adders. These optimized adders are simulated in HSPICE and results are compared with other ternary ripple-carry based adders in literature. Table 5.3 shows the comparison of different Multi-Digit Ternary Adders

Table 5.2: Encoder Mapping for Ternary Adders

(a) Encoders used for 3-digit Ternary Adder

Encoders used to generate	Delay Optimization (Algorithm 5.2)	PDP Optimization (Algorithm 5.3)
Sum_0	$ed_{(2,10)}$	$ed_{(3,10)}$
Sum_1	$ed_{(4,10)}$	$ed_{(3,10)}$
Sum_2	$ed_{(1,16)}$	$ed_{(2,10)}$
$Cout_2$	$ed_{(1,19)}$	$ed_{(2,10)}$

(b) Encoders used for 6-digit Ternary Adder

Encoders used to generate	Delay Optimization (Algorithm 5.2)	PDP Optimization (Algorithm 5.3)
Sum_0	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_1	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_2	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_3	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_4	$ed_{(4,10)}$	$ed_{(3,10)}$
Sum_5	$ed_{(1,16)}$	$ed_{(2,10)}$
$Cout_5$	$ed_{(1,19)}$	$ed_{(2,10)}$

(c) Encoders used for 9-digit Ternary Adder

Encoders used to generate	Delay Optimization (Algorithm 5.2)	PDP Optimization (Algorithm 5.3)
Sum_0	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_1	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_2	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_3	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_4	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_5	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_6	$ed_{(3,10)}$	$ed_{(3,10)}$
Sum_7	$ed_{(4,10)}$	$ed_{(3,10)}$
Sum_8	$ed_{(1,16)}$	$ed_{(2,10)}$
$Cout_8$	$ed_{(1,19)}$	$ed_{(2,10)}$

(MTA) with respect to propagation delay, power consumption and PDP. In this section, MTA-1 refers to the ripple-carry multi-digit adder presented in [39], whereas MTA-2 and MTA-3 refer to ripple-carry adders designed using single-digit adders presented in [38] and [59] respectively. The adder architecture presented in [58], which is used as an example in our work is referred as MTA-4. The ternary adder presented in [58], does not specify chirality of transistors used in its encoder design. Hence results for two adder designs one which uses encoder $ed_{(1,19)}$ and the other with encoder $ed_{(1,10)}$ are presented. The existing adders are compared with the ternary adder MTA-4, which uses encoder mapping obtained from proposed Algorithms 5.1, 5.2 and 5.3. Results for ternary adder MTA-4, which uses proposed encoders ($ed_{(2,10)}, ed_{(2,10)}, ed_{(3,19)}, ed_{(4,10)}, ed_{(4,19)}$) to generate all ternary outputs, are also presented.

To measure power, all multi-digit adder designs are simulated with same random test patterns at switching frequency of $500MHz$ and the average power consumption is determined. To measure the delay of the multi-digit adders, test patterns have been chosen in such a way that the signal change propagates through the critical path and the maximum delay is measured. FO4 delay is calculated by loading each of the output nodes in the critical path with four STI gates. The power-delay product is the product of worst-case propagation delay and average power consumption.

The power optimized implementation of MTA-4 uses Algorithm 5.1, which maps proposed encoder-2 (low-power encoder), i.e. $ed_{(3,10)}$ to all paths of the circuit. Hence this ternary adder has least power consumption when compared to other ripple-carry based designs. The proposed encoder-based power optimization results in a ternary adder which has 73 – 79% reduction in power consumption when compared to MTA-2, which has least power consumption among all the designs existing in literature. The reduction in power consumption of power optimized design is mainly due to the use of a low-power encoder ($ed_{(3,10)}$), where *logic 1* is generated at the output by creating a high-resistance path between VDD and GND . The power optimized design also shows 70 – 75% reduction in propagation delay and 92 – 94% reduction in PDP when compared to MTA-2.

The delay optimized implementation of MTA-4 is obtained by mapping encoder designs as shown in Table 5.2, which is a result of Algorithm 5.2. This algorithm maps least delay encoders to critical path and least power encoders for non-critical paths. The resulting delay optimized design has similar delay when compared to MTA-4 of [58] which uses encoder $ed_{(1,19)}$ for all paths. Unlike the implementation of MTA-4 with only $ed_{(1,19)}$ encoders, the power consumption of the MTA-4, which is implemented using Algorithm 5.2, increases marginally with increase in operand size. This is because the encoder design $ed_{(1,19)}$, which has least delay and large power consumption, is used only for critical paths and for non-critical paths proposed encoders ($ed_{(3,10)}$ and $ed_{(4,10)}$), which have lower power consumption when compared to $ed_{(1,19)}$, are used. Hence the delay optimized design has up to 53 – 78% lower power consumption and PDP when compared to design presented in [58].

The PDP optimized implementation of MTA-4 is obtained by using using encoder mapping, which is a result of Algorithm 5.3 and is shown in Table 5.2. The PDP optimization of ternary adder results in at least 86 – 90% reduction in PDP when compared to MTA-1, which has least PDP among existing ripple-carry based ternary adder designs. The PDP optimized design also shows 81 – 84% reduction in power consumption and 21 – 41% reduction in propagation delay when compared to MTA-1.

Figure 5.10 shows a plot of power consumption v/s propagation delay, for different 9-digit ternary adders. In this figure, x -axis represents the power consumption and y -axis represents propagation delay. This figure clearly shows that MTA-4 designs which are implemented using proposed encoders and/or algorithms are optimized with respect to power consumption and propagation delay. Although the Algorithms 5.1, 5.2 and 5.3 are used in implementation of ternary adders, they can be used for implementation of any ternary logic circuit which needs encoders.

Table 5.3: Simulation Results for N -digit Ternary Adders

Ternary Adder Designs	N			3-digit			6-digit			9-digit		
	Power (μW)	Delay (ps)	PDP (fJ)	Power (μW)	Delay (ps)	PDP (fJ)	Power (μW)	Delay (ps)	PDP (fJ)	Power (μW)	Delay (ps)	PDP (fJ)
MTA-1 [39]	8.20	97.52	0.80	16.69	205.4	3.43	23.41	313.4	7.34			
MTA-2 [38]	5.69	290.5	1.66	12.18	526.9	6.42	17.61	762.5	13.4			
MTA-3 [59]	28.88	108.7	3.14	60.89	206.1	12.6	83.62	303.3	25.4			
MTA-4 with encoder $ed_{(1,19)}$ [58]	51.10	63.76	3.26	103.7	117.2	12.2	134.3	170.7	22.9			
MTA-4 with encoder $ed_{(1,10)}$ [58]	19.12	78.91	1.51	29.70	130.8	3.88	32.69	183.8	6.01			
MTA-4 with proposed encoder $ed_{(2,19)}$	10.90	66.00	0.72	17.03	119.4	2.03	19.48	172.7	3.36			
MTA-4 with proposed encoder $ed_{(2,10)}$	1.51	76.73	0.12	2.66	128.7	0.34	4.30	181.6	0.78			
MTA-4 with proposed encoder $ed_{(3,19)}$	7.47	69.06	0.52	11.70	123.0	1.44	13.90	176.5	2.44			
MTA-4 with proposed encoder $ed_{(4,19)}$	15.60	65.87	1.03	24.20	118.9	2.87	27.03	172.2	4.65			
MTA-4 with proposed encoder $ed_{(4,10)}$	1.54	74.46	0.12	2.68	127.9	0.33	4.24	181.4	0.77			
Power optimized MTA-4 using Algorithm 5.1	1.50	84.31	0.13	2.53	136.9	0.35	4.17	189.1	0.79			
Delay optimized MTA-4 using Algorithm 5.2	23.55	63.72	1.50	25.82	117.5	3.03	28.60	171.0	4.89			
PDP optimized MTA-4 using Algorithm 5.3	1.51	76.71	0.11	2.52	128.8	0.32	4.18	181.8	0.76			
Impr. of MTA-4 (using Algo. 5.1) w.r.t. MTA-2	74%	71%	92%	79%	74%	94%	76%	75%	94%			
Impr. of MTA-4 (using Algo. 5.2) w.r.t. MTA-4 (with $ed_{(1,19)}$)	54%	0%	54%	75%	0%	75%	79%	0%	79%			
Impr. of MTA-4 (using Algo. 5.3) w.r.t. MTA-1	82%	21%	86%	85%	37%	90%	82%	42%	90%			

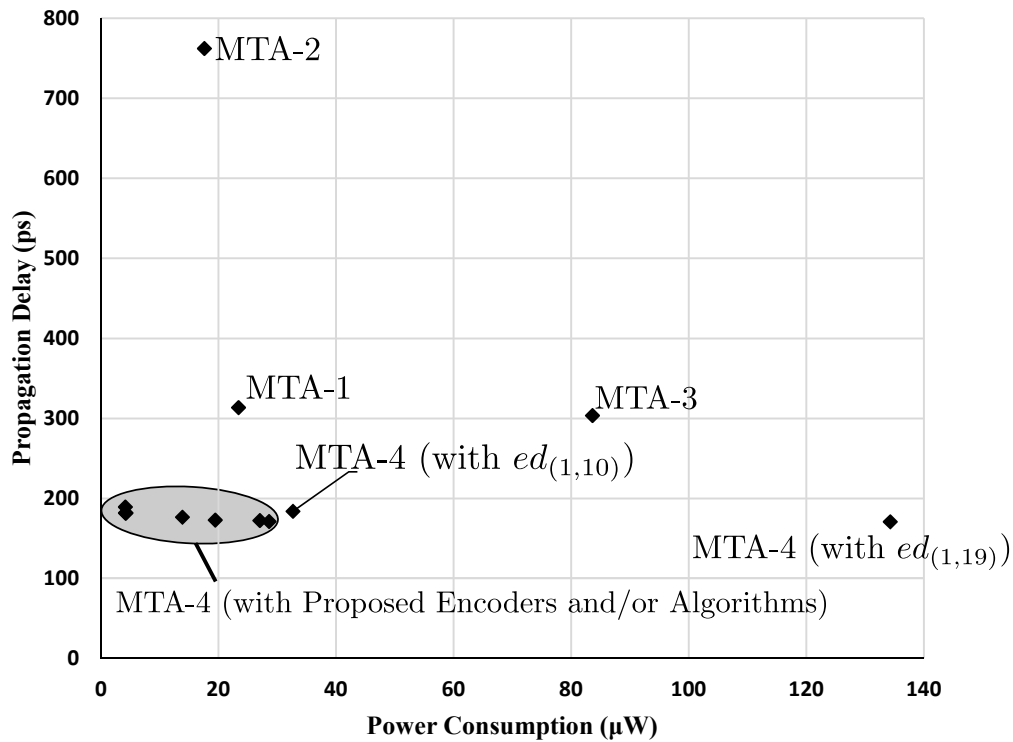


Figure 5.10: Power Consumption Vs Propagation Delay for 9-digit Ternary Adders

5.7 Conclusions

Encoder is a critical element in the design of ternary logic circuits and is used to convert intermediate binary signals to final ternary outputs. This chapter presents improved encoder designs which are used in implementation of ternary logic circuits. A detailed analysis is carried out on encoders to understand the effect of using CNFETs with CNTs of different diameter on the overall propagation delay and power consumption. Based on this analysis, optimization algorithms are presented which choose suitable encoders for different output stages of a ternary circuit while optimizing delay, power or power-delay product. Ternary ripple-carry based adder is taken as an example and the proposed encoder mapping algorithms are applied on it. The resulting ternary adder designs are implemented in HSPICE and compared with other ternary adders in literature with respect to different design parameters. Simulation results indicate that the ternary adder designs, which use encoder mapping obtained from proposed algorithms, result in 54 – 82% reduction in power consumption, 0 – 75% in propagation delay and 54 – 94% in power-delay product when compared to different existing ripple carry-based ternary adders.

Chapter 6

Synthesis of Ternary Logic Circuits using 2:1 Multiplexers

6.1 Introduction

Design approaches for ternary circuits can be classified into two main categories namely decoder-encoder based and multiplexer based. Chapter 3 presented an overview of existing decoder-encoder based and 3:1 multiplexer based approaches. Additionally this chapter presented three design approaches, two of which was encoder based, and the third approach was multiplexer based. Scaling existing and proposed design approaches to implement more complex ternary circuits, requires development of synthesis algorithms.

The encoder based approaches rely on using binary circuits to implement the expressions for unary functions of ternary outputs. These binary circuits can be synthesized using existing binary synthesis algorithms. A synthesis approach which uses 3:1 multiplexers has been presented in [41]. In this chapter a new synthesis technique, which aids in implementation of complex ternary circuits, is presented. This novel technique uses “2:1 multiplexers” for implementing ternary logic circuits, and is based on transformation of a Ternary Decision Diagram (TDD) into a Binary Decision Diagram (BDD). This transformed TDD is then used to implement the ternary logic function using

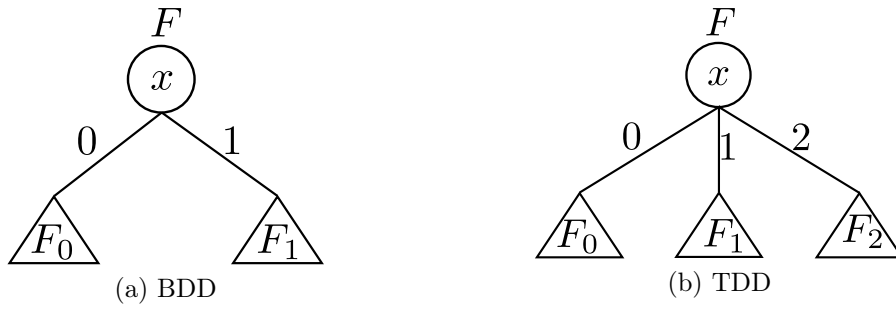


Figure 6.1: Binary and Ternary Decision Diagrams

2:1 multiplexers. This chapter also presents a procedure which decomposes ternary functions with three (and more) inputs into multiple 1-input ternary functions. This methodology is used to design a synthesis algorithm, which is used to synthesize various ternary benchmark functions.

6.2 Preliminaries

6.2.1 Binary Decision Diagrams

A binary decision diagram (BDD) is used to represent a two-valued logic function F . Let $F = \bar{x} \cdot F_0 + x \cdot F_1$ be the Shannon expansion of F with respect to variable x . The BDD for F is represented as shown in Figure 6.1(a), where F_0 and F_1 represent the sub-graphs.

A BDD for a function F , whose truth table is known, is constructed by a procedure as shown in Figure 6.2 which has one-to-one correspondence between 2^n rows of the table and the 2^n paths to the outputs of the diagram. These outputs may then be labeled with the corresponding binary values of f resulting in the required diagram. Figure 6.2 also shows the implementation of BDD using 2:1 multiplexers. With n variables, there will initially be $2^n - 1$ nodes in a BDD. There are several ways in which the number of nodes can be reduced [73]. The decision diagrams with reduced nodes are called as Quasi Reduced BDD (QRBDD) or Reduced BDD (RBDD).

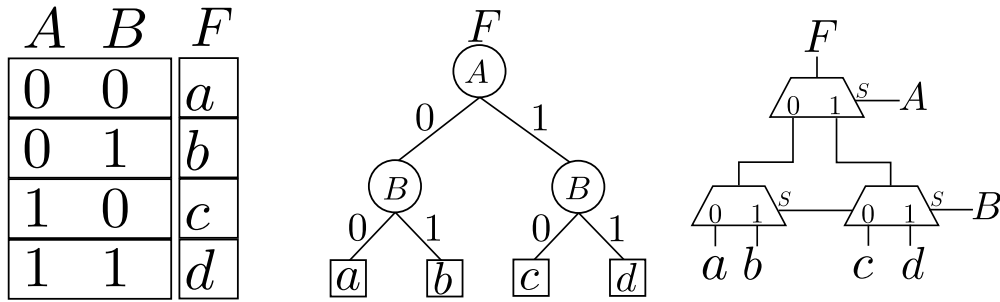


Figure 6.2: BDD and its 2 : 1 Mux based implementation for a given Truth-table

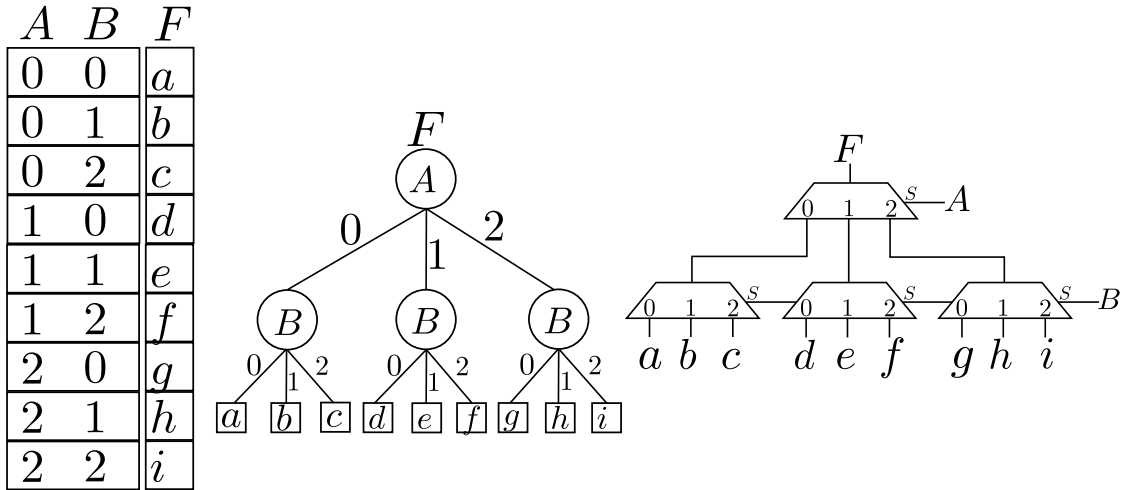


Figure 6.3: TDD and its 3 : 1 Mux based implementation for a given Truth-table

6.2.2 Ternary Decision Diagrams

A general-TDD is a natural extension of the BDD to the three-valued case. Let $F = x^0 \cdot F_0 + x^1 \cdot F_1 + x^2 \cdot F_2$ be the three-valued version of the Shannon expansion of an arbitrary three-valued function $F : T^n \rightarrow T$, $T = \{0, 1, 2\}$ with respect to variable x . The TDD for F is represented as shown in Figure 6.1(b), where F_0 , F_1 and F_2 represent the sub-graphs and the relation between values x^0, x^1, x^2 and x is given by equation (6.1). As seen from this equation, the values x^0, x^1, x^2 are binary in nature and are equal to 2 or 0.

$$x^k = \begin{cases} 2 & \text{if } x = k \\ 0 & \text{if } x \neq k \end{cases} \tag{6.1}$$

Similar to a BDD, the truth table for a function F can be translated into a TDD, which can be implemented using 3 : 1 multiplexers. Figure 6.3 shows an example truth table along with TDD and its implementation using multiplexers. With n variables, there will initially be $\frac{3^n-1}{2}$ nodes in a TDD, which can be reduced. The decision diagrams with reduced nodes are called as Quasi Reduced TDD (QRTDD) or Reduced TDD (RTDD). Recently, a 3:1 multiplexer based synthesis procedure has been presented in [41]. This procedure is similar to TDD based implementation (in Figure 6.3) except for the last stage, where the multiplexers were replaced with equivalent realization of unary operators.

6.3 Proposed Synthesis Methodology

The proposed synthesis technique for ternary logic circuits is based on transforming a TDD into a BDD. First, a general procedure to transform TDD to BDD is presented. This enables the implementation of ternary logic circuits using 2:1 multiplexers. The transformed TDD is called as Ternary-Transformed Binary Decision Diagram (TBDD). This TBDD representation is then used in the synthesis of ternary functions. Initially, TBDD-based synthesis techniques for handling one and two variable (2-input) functions are presented. These techniques are further used in synthesis of circuits with more than two inputs.

6.3.1 General Ternary-Transformed Binary Decision Diagrams (TBDD)

The basic idea behind TDD to BDD transformation is based on Proposition 3.2, and is presented using Proposition 6.1.

Proposition 6.1. *A TDD can be transformed to a BDD.*

Proof. Consider the general TDD relation $F = x^0 \cdot F_0 + x^1 \cdot F_1 + x^2 \cdot F_2$, which is represented in the Figure 6.1(b), where F_0 , F_1 and F_2 represent the sub-graphs, signals x^0 , x^1 and x^2 are mutually exclusive and are related to x according to equation (6.1).

$$F = x^0 \cdot F_0 + x^1 \cdot F_1 + x^2 \cdot F_2 \quad (6.2)$$

$$F = x^0 \cdot F_0 + (x^1 + x^0) \cdot (x^1 + x^2) \cdot F_1 + x^2 \cdot (x^1 + x^2) \cdot F_2$$

$$\because x^1 = (x^1 + x^0) \cdot (x^1 + x^2), \quad x^2 \cdot x^2 = x^2, \quad \text{and } x^2 \cdot x^1 = 0$$

$$F = x^0 \cdot F_0 + (x^1 + x^2) \cdot ((x^1 + x^0) \cdot F_1 + x^2 \cdot F_2) \quad (6.3)$$

$$F = x^0 \cdot F_0 + \overline{x^0} \cdot (\overline{x^2} \cdot F_1 + x^2 \cdot F_2) \quad (6.4)$$

$\because (x^1 + x^2) = \overline{x^0}$, $(x^1 + x^0) = \overline{x^2}$, where $\overline{x^0}$, $\overline{x^1}$ and $\overline{x^2}$ represent the binary NOT of signals x^0 , x^1 and x^2 respectively as given by equation (6.5).

$$\overline{x^k} = \begin{cases} 2 & \text{if } x^k = 0 \\ 0 & \text{if } x^k = 2 \end{cases} \quad (6.5)$$

Alternatively equation (6.2) can also be represented as equation (6.6) and (6.7).

$$F = \overline{x^2} \cdot (x^0 \cdot F_0 + \overline{x^0} \cdot F_1) + x^2 \cdot F_2 \quad (6.6)$$

$$F = \overline{x^1} \cdot (x^0 \cdot F_0 + \overline{x^0} \cdot F_2) + x^1 \cdot F_1 \quad (6.7)$$

The relation in equations (6.4), (6.6) and (6.7) are similar to BDD relation, $F = \bar{x} \cdot F_0 + x \cdot F_1$ and hence can be represented as the graph similar to that of a BDD. Figure 6.4 shows the TBDDs for equations (6.4) and (6.6). A similar procedure can be followed to develop TBDD for (6.7). \square

The graphs shown in Figure 6.4 can be implemented using 2:1 multiplexers. This implementation differs from multiplexer based implementation of BDD, with respect to the selection signal. Here, the selection signal is three-valued and hence there should

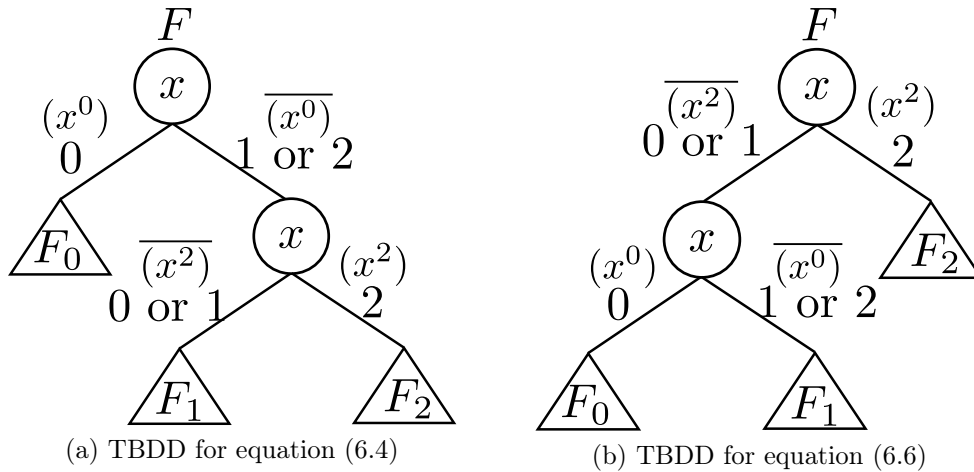


Figure 6.4: Ternary-Transformed Binary Decision Diagram

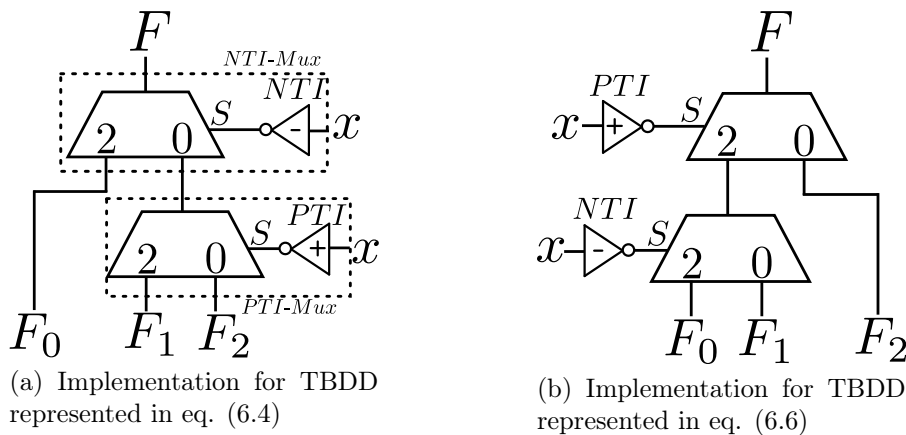
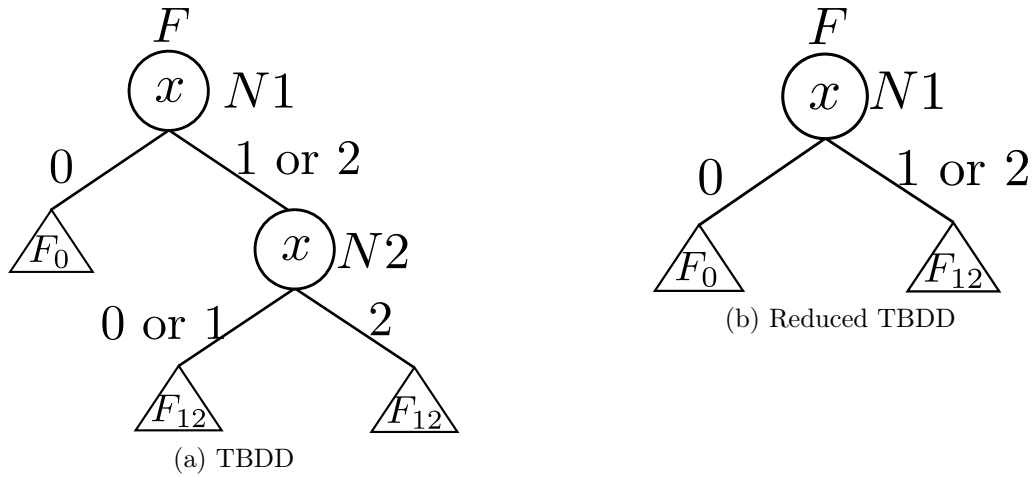


Figure 6.5: 2 : 1 Multiplexer based implementation of TBDD

be a way to differentiate between three possible values i.e. 0, 1 and 2. This can be achieved for TBDD representation of equations (6.4) and (6.6), by passing the selection signal through the NTI and PTI gates. But for implementation of TBDD in equation (6.7), a NOR like structure is needed to generate x^1 and \bar{x}^1 (See Figure 3.15(c)), which increases the complexity. Hence, in this work, only TBDDs represented in Figures 6.4(a) and 6.4(b) are used. Figures 6.5(a) and 6.5(b) show the implementation of TBDDs in Figures 6.4(a) and 6.4(b) respectively, using 2:1 multiplexers and ternary inverter gates. To differentiate between the two different multiplexers, 2:1 multiplexer with NTI gate is referred as NTI-Mux and the one with PTI gate is referred as PTI-Mux.

The TBDDs shown in Figure 6.4 are reduced, depending on F_0 , F_1 and F_2 , with the help of rule and propositions presented below:

Figure 6.6: Illustration for **Rule 1**

Rule 1. *In a TBDD, if the children of a node are identical then the node is removed from the graph and its incoming edges are directed to the child.*

Rule 1 is derived from BDD reduction rules presented in [73]. This rule is illustrated with an example below:

Example 1. In the TBDD shown in Figure 6.4(a), let $F_1 = F_2 = F_{12}$. The corresponding TBDD is shown in Figure 6.6(a), where the node $N2$ has identical children. By applying Rule 1, the node $N2$ is replaced with function F_{12} . The reduced TBDD is shown in Figure 6.6(b). It is possible to achieve TBDD reduction with the help of equation (6.4). This equation is simplified as below:

$$F = x^0 \cdot F_0 + \overline{x^0} \cdot (\overline{x^2} \cdot F_1 + x^2 \cdot F_2) \quad (6.8)$$

$$F = x^0 \cdot F_0 + (x^1 + x^2) \cdot ((x^1 + x^0) \cdot F_1 + x^2 \cdot F_2)$$

$$F = x^0 \cdot F_0 + (x^1 + x^2) \cdot ((x^1 + x^0) \cdot F_{12} + x^2 \cdot F_{12})$$

$$F = x^0 \cdot F_0 + (x^1 + x^2) \cdot F_{12} \quad (6.9)$$

Equation (6.9) corresponds to a TBDD shown in Figure 6.6(b). Rule 1 helps in choosing one among the two possible TBDDs shown in Figures 6.4(a) and 6.4(b), such

that the TBDD leads to an optimal implementation. To fully exploit Rule 1, a set of propositions, which lead to optimized implementation, are presented below:

Proposition 6.2. *For a general TDD (represented in Figure 6.1(b)), if $F_1 = F_2$ then TBDD represented by equation $x^0 \cdot F_0 + \overline{x^0} \cdot (\overline{x^2} \cdot F_1 + x^2 \cdot F_2)$ (shown in Figure 6.4(a)) leads to optimal implementation.*

Proof. Consider Example 1, where $F_1 = F_2 = F_{12}$. Here, equation $x^0 \cdot F_0 + \overline{x^0} \cdot (\overline{x^2} \cdot F_{12} + x^2 \cdot F_{12})$ can be simplified by applying Rule 1 resulting in equation $x^0 \cdot F_0 + (x^1 + x^2) \cdot F_{12}$, leading to a reduced TBDD as shown in Figure 6.6(b). Representing the same function using equation $\overline{x^2} \cdot (x^0 \cdot F_0 + \overline{x^0} \cdot F_{12}) + x^2 \cdot F_{12}$ (shown in Figure 6.4(b)) does not lead to reduction since Rule 1 is not applicable. \square

Proposition 6.3. *For a general TDD (represented in Figure 6.1(b)), if $F_0 = F_1$ then TBDD represented by equation $\overline{x^2} \cdot (x^0 \cdot F_0 + \overline{x^0} \cdot F_1) + x^2 \cdot F_2$ (shown in Figure 6.4(b)) leads to optimal implementation.*

If the conditions required for applying Proposition 6.2 ($F_1 = F_2$) or Proposition 6.3 ($F_0 = F_1$) are not met, then TBDD shown in Figure 6.4(b) or 6.4(b) is used for representing the ternary function.

6.3.2 TBDD-based synthesis for 1-input ternary functions (Unary Operators)

The TBDD synthesis technique for 1-input ternary functions, also called as unary operators, uses Karnaugh-map (K-map) representation. Consider a 1-input function represented by a K-map shown in the Figure 6.7, where A is ternary inputs and F is ternary output. The notation, F_{Ax} , is used to represent the entries of the K-map, where F_{Ax} is ternary output when $A = x$ and $x \in \{0, 1, 2\}$. The K-map can be transformed into TBDD in two ways, according to equations (6.4) and (6.6), as shown in Figure 6.8.

Propositions 6.2 and 6.3 are used to choose one of the two TBDDs shown in Figures 6.8(a) and 6.8(b) and Rule 1 is used to reduce them. The resulting TBDD is imple-

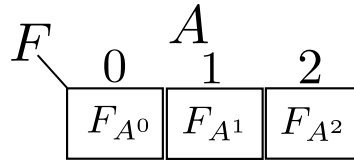


Figure 6.7: K-map for 1-input Ternary Function

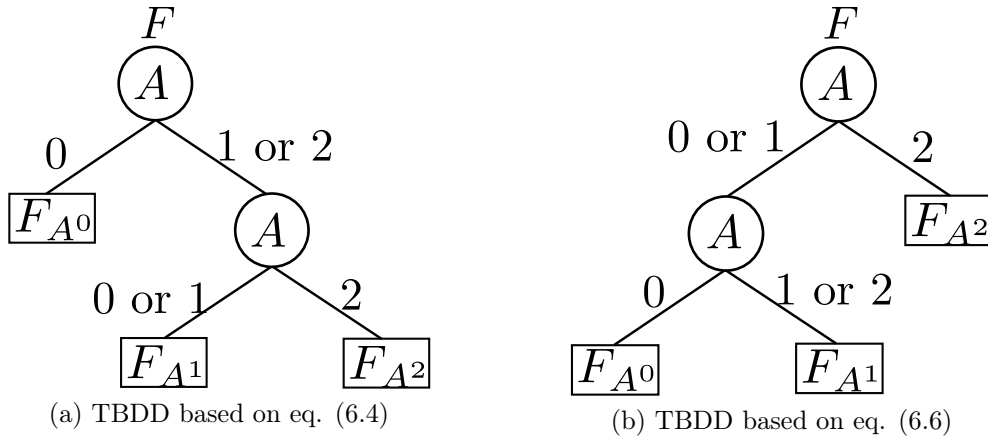


Figure 6.8: TBDDs for 1-input Ternary Function

mented using 2:1 multiplexers. Some of the 2:1 multiplexer based implementations are optimized further by replacing multiplexers with equivalent ternary gates. This is illustrated with the help of an example, where $F_{A^0} = 2$, $F_{A^1} = 0$ and $F_{A^2} = 0$. Figures 6.9(a) and 6.9(b) show the initial TBDD and reduced TBDD after applying Rule 1. Here the output, $F = 2$, if $A = 0$ and $F = 0$, if $A = 1$ or 2. This is equivalent to the output of an NTI gate with A as input. Hence the 2:1 multiplexer based implementation of reduced TBDD can be replaced with an NTI gate as shown in Figure 6.10. Apart from this, there are three more TBDD templates, which, along with their 2:1 multiplexer based implementations and equivalent gates are shown in Figure 6.11. The binary NOT gate shown here corresponds to the equation (6.5).

To fully exploit Rule 1 and the templates, a set of propositions, which lead to optimized implementation of 1-input ternary function, are presented below:

Proposition 6.4. *For a 1-input ternary function (represented in Figure 6.7), if $F_{A^0} \neq F_{A^1}$ and $F_{A^1} = 2, F_{A^2} = 0$ or $F_{A^1} = 0, F_{A^2} = 2$ then TBDD represented by equation $A^0 \cdot F_{A^0} + \overline{A^0} \cdot (\overline{A^2} \cdot F_{A^1} + A^2 \cdot F_{A^2})$ (shown in Figure 6.8(a)) leads to optimal implementation.*

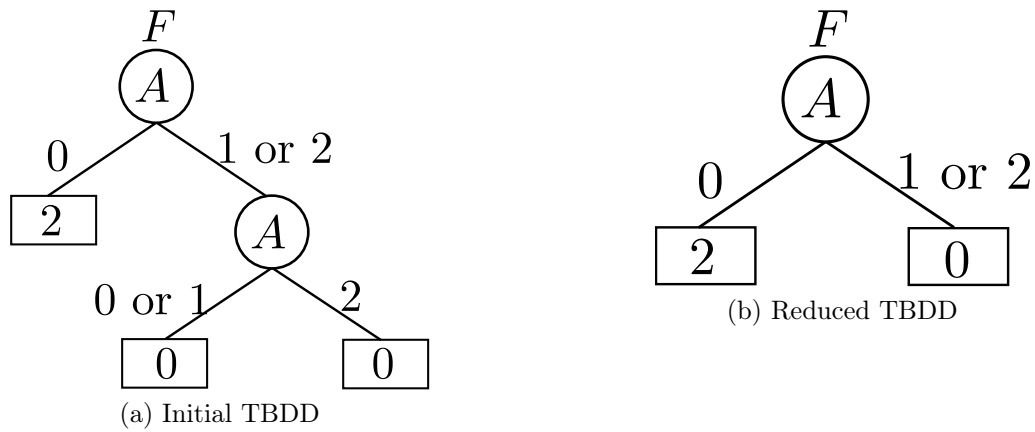


Figure 6.9: TBDD for $F_{A^0} = 2$, $F_{A^1} = 0$ and $F_{A^2} = 0$

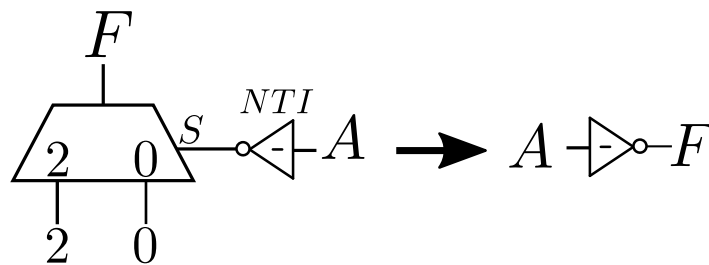


Figure 6.10: Multiplexer-based implementation of Reduced TBDD in Figure 6.9 and its equivalent Ternary Gate

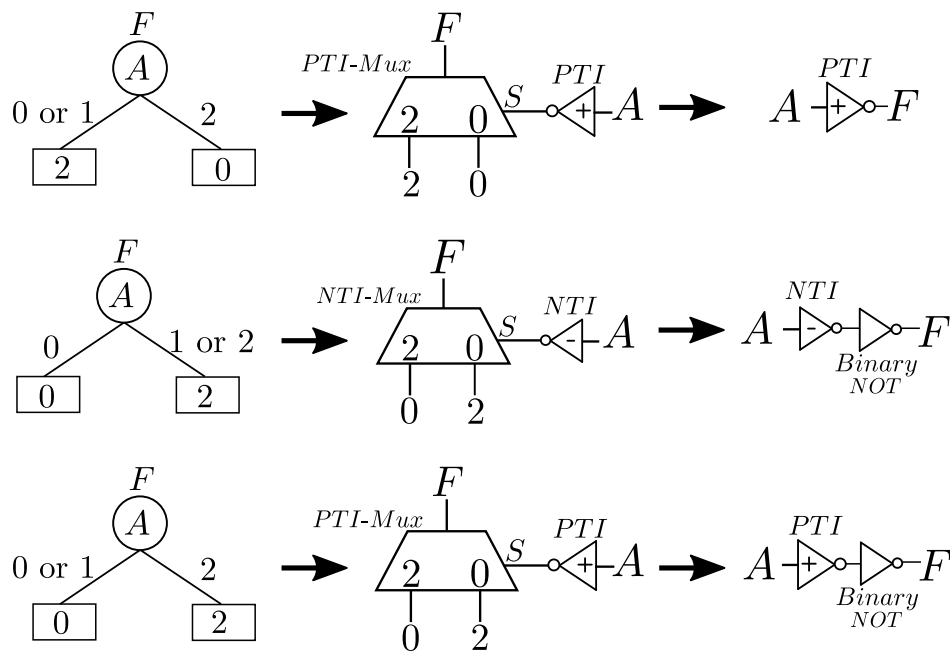


Figure 6.11: TBDD templates, their 2:1 multiplexer based implementations and equivalent gates.

Proof. Consider a 1-input ternary function with $F_{A^0} = 1$, $F_{A^1} = 2$ and $F_{A^2} = 0$. Let this function be represented by equation $A^0 \cdot F_{A^0} + \overline{A^0} \cdot (\overline{A^2} \cdot F_{A^1} + A^2 \cdot F_{A^2})$ as shown in Figure 6.12(a), where sub-graph N is equivalent to one of templates shown in Fig 6.11 and hence a PTI gate is required for its implementation instead of a multiplexer. But implementation of TBDD represented by equation $\overline{A^2} \cdot (A^0 \cdot F_{A^0} + \overline{A^0} \cdot F_{A^1}) + A^2 \cdot F_{A^2}$ (shown in Figure 6.12(b)) requires two multiplexers. \square

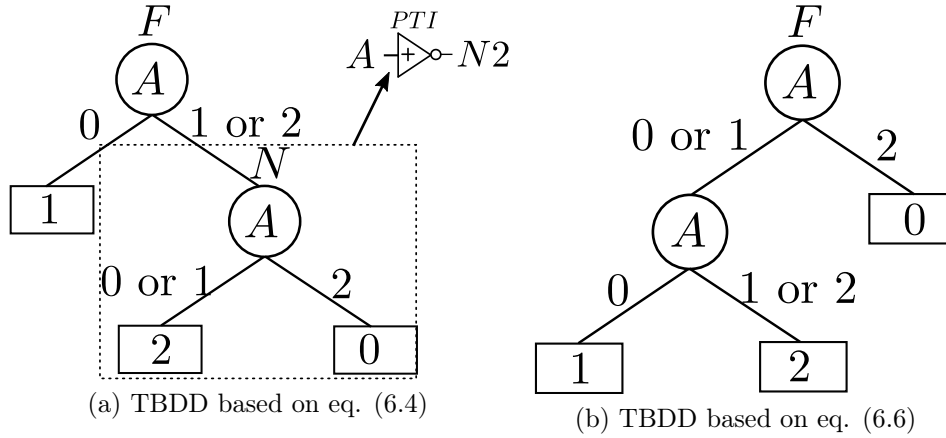


Figure 6.12: TBDD Example for Proposition 6.4

Proposition 6.5. For a 1-input ternary function (represented in Figure 6.7), if $F_{A^1} \neq F_{A^2}$ and $F_{A^0} = 2, F_{A^1} = 0$ or $F_{A^0} = 0, F_{A^1} = 2$ then TBDD represented by equation $\overline{A^2} \cdot (A^0 \cdot F_{A^0} + \overline{A^0} \cdot F_{A^1}) + A^2 \cdot F_{A^2}$ (shown in Figure 6.8(b)) leads to optimal implementation.

Any 1-input ternary function can be implemented using propositions 6.2, 6.3, 6.4 and 6.5. There are 27 1-input ternary functions (unary operators) for ternary case [41], which are shown in Table 6.1 for an input A . The operators are arranged in increasing order of complexity of implementation and can be divided into seven groups namely *Group* – 0, 1, 2, 3, 4, 5, 6. The complexity of different groups shown in Table 6.1 are summarized below:

- *Group* – 0: Implementation of unary operators T_0, T_1, T_2 and T_3 does not require any circuit elements. This is because T_0, T_1, T_2 are constant functions and T_3 is identity function.

Table 6.1: Unary Operators

(a) Group 0 to 3

<i>Grp.</i>	0				1			2		3						
<i>A</i>	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
0	0	1	2	0	2	2	0	0	0	0	0	1	1	1	1	2
1	0	1	2	1	2	0	0	2	0	0	1	0	1	1	2	2
2	0	1	2	2	0	0	2	2	1	2	1	0	0	2	2	1

(b) Group 4 to 6

<i>Grp.</i>	4				5		6				
<i>A</i>	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}	T_{21}	T_{22}	T_{23}	T_{24}	T_{25}	T_{26}
0	0	2	1	2	0	1	0	1	1	2	2
1	2	0	2	0	2	0	1	0	2	1	1
2	0	2	0	1	1	2	0	1	1	0	2

- *Group – 1*: Implementation of T_4 and T_5 requires NTI or PTI gates. (Use Proposition 6.2 or 6.3, apply Rule 1 and use templates)
- *Group – 2*: Implementation of T_6 and T_7 requires a binary NOT gate in addition to NTI or PTI gates. (Use Proposition 6.2 or 6.3, apply Rule 1 and use templates)
- *Group – 3*: Implementation of each of the operators $T_8 – T_{15}$ requires either an NTI-Mux or a PTI-Mux. (Use Proposition 6.2 or 6.3 and apply Rule 1)
- *Group – 4*: Implementation of each of the operators $T_{16} – T_{19}$ requires either an NTI-Mux or a PTI-Mux in addition to NTI or PTI gate. (Use Proposition 6.4 or 6.5, apply Rule 1 and use templates)
- *Group – 5*: Implementation of T_{20} and T_{21} requires either an NTI-Mux or a PTI-Mux, an NTI or a PTI gate and a binary NOT gate. (Use Proposition 6.4 or 6.5, apply Rule 1 and use templates)
- *Group – 6*: Implementation of each of the operators $T_{22} – T_{26}$ requires two multiplexers, an NTI-Mux and a PTI-Mux.

6.3.3 TBDD-based synthesis for 2-input ternary functions

The TBDD synthesis technique for 2-input ternary circuit uses Karnaugh-map (K-map) representation, which is similar to cube representation in [41, 62]. Consider a 2-input function represented by a K-map shown in the Figure 6.13, where A , B are ternary inputs and F is ternary output. The notation, $F_{A^x B^y}$, is used to represent the entries of the K-map, where $F_{A^x B^y}$ is ternary output when $A = x$, $B = y$ and $x, y \in \{0, 1, 2\}$. For example, when $A = 0$, $B = 1$, the corresponding K-map entry is represented as $F_{A^0 B^1}$.

The K-map can be transformed into TBDD in many ways depending on the selection variable (A or B) and the relation (equation (6.4) or (6.6)) used on it. The Propositions 6.2 and 6.3 can be applied by decomposing a 2-input function into three 1-input functions, which are then implemented using processes as explained in Section 6.3.2. The K-map shown in Figure 6.13 can also be represented in terms of 1-input row functions or 1-input column functions depending on inputs. Figure 6.14 shows alternate representations of 2-input function. Here, $F_{A^0}(B)$, $F_{A^1}(B)$ and $F_{A^2}(B)$ represent 1-input row functions, which are dependent on value of B . Similarly let $F_{B^0}(A)$, $F_{B^1}(A)$ and $F_{B^2}(A)$ represent 1-input column functions, which are dependent on value of A . These functions are related to K-map entries in Figure 6.13 according to equations (6.10) and (6.11).

$$F_{A^x}(y) = F_{A^x B^y} \text{ where } x, y \in \{0, 1, 2\} \quad (6.10)$$

F	B	0	1	2
A	0	$F_{A^0 B^0}$	$F_{A^0 B^1}$	$F_{A^0 B^2}$
1	1	$F_{A^1 B^0}$	$F_{A^1 B^1}$	$F_{A^1 B^2}$
2	2	$F_{A^2 B^0}$	$F_{A^2 B^1}$	$F_{A^2 B^2}$

Figure 6.13: K-map Representation of 2-input Function

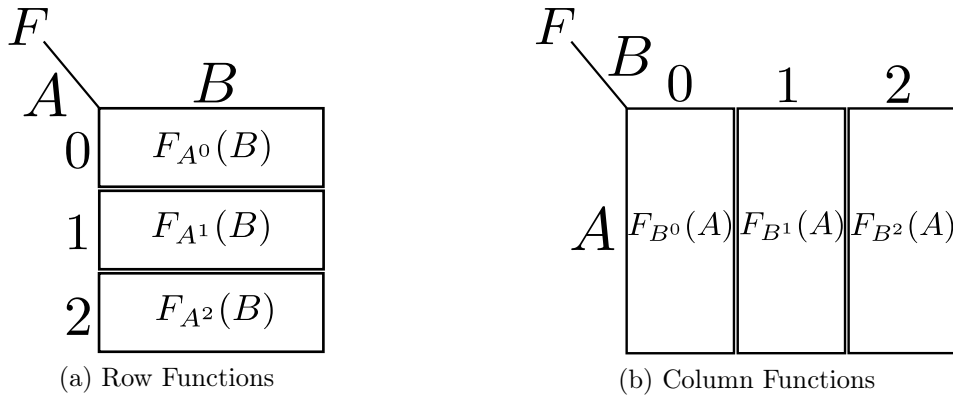


Figure 6.14: K-map Representation of of 2-input Function using 1-input Functions

$$F_{B^y}(x) = F_{A^x B^y} \text{ where } x, y \in \{0, 1, 2\} \quad (6.11)$$

The choice of decomposition (into 1-input row functions or 1-input column functions), depends on whether Proposition 6.2 or 6.3 is applicable for TBDD representation of 2-input function. The following propositions are used to choose one of the input variables A or B such that the TBDD representation of 2-input function leads to optimal implementation.

Proposition 6.6. *For a 2-input ternary function $F(A, B)$, if $F_{A^0}(B) = F_{A^1}(B)$ or $F_{A^1}(B) = F_{A^2}(B)$, then decomposition along input A (row functions) leads to a reduced TBDD.*

Proof. If $F_{A^1}(B) = F_{A^2}(B)$ or $F_{A^0}(B) = F_{A^1}(B)$ then Proposition 6.2 or 6.3 can be applied leading to reduced TBDD. \square

Corollary 6.1. *For a 2-input ternary function $F(A, B)$, if $F_{B^0}(A) = F_{B^1}(A)$ or $F_{B^1}(A) = F_{B^2}(A)$, then decomposition along input B (column functions) leads to a reduced TBDD.*

Proposition 6.6 is illustrated with an example, whose K-map representation is shown in Figure 6.15. Since $F_{A^1}(B) = F_{A^2}(B)$, decomposing along input A and using Proposition 6.2 leads to reduced TBDD, shown in Figure 6.16, which has only one node. Decomposing this 2-input function along in input B leads to TBDD with two nodes as shown in Figure 6.17. After decomposing along A , the resulting 1-input row/column

F	A	B 0	1	2	
		0	2	0	1
	1	1	1	2	$\rightarrow F_{A^1}(B)$
	2	1	1	2	$\rightarrow F_{A^2}(B)$
		\downarrow	\downarrow	\downarrow	
		$F_{B^0}(A)$	$F_{B^1}(A)$	$F_{B^2}(A)$	

Figure 6.15: An Example for 2-input Function

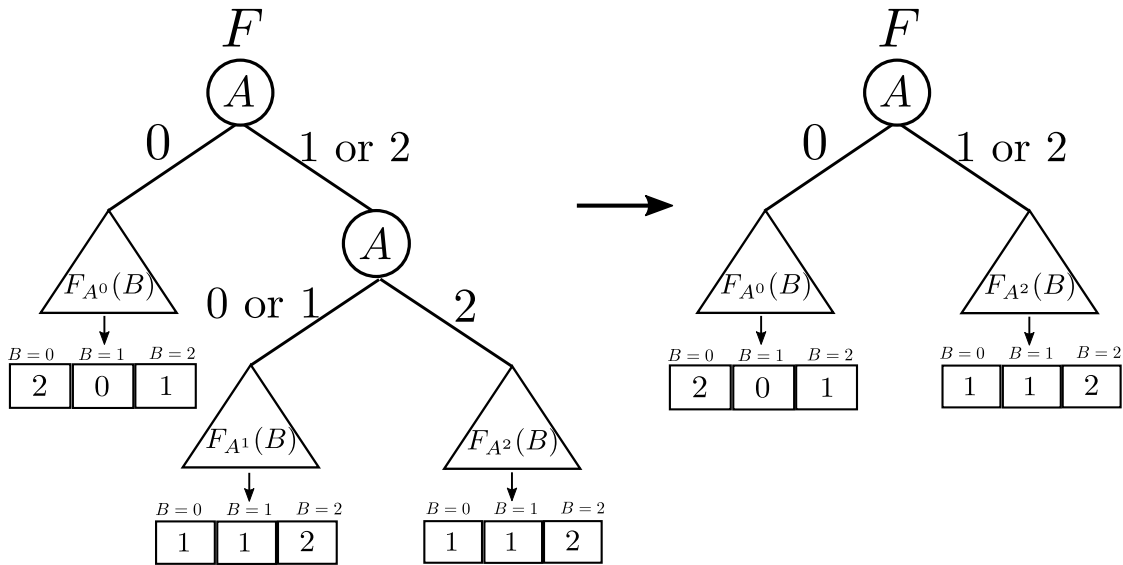


Figure 6.16: TBDD representation when decomposed w.r.t A

functions are further represented by TBDD using Rule 1 and Propositions 6.2 - 6.5.

Figure 6.18 shows the final TBDD for 2-input function (in Figure 6.15) which is used for implementation using 2:1 multiplexers.

If the conditions required for Proposition 6.6 or Corollary 6.1 are not satisfied, then the decomposition is done by using the propositions described below:

Proposition 6.7. *For a 2-input ternary function $F(A, B)$, if $F_{A^0}(B)$ is constant function or $F_{A^2}(B)$ is constant function (i.e. function value is same irrespective of value of B) decomposition along input A leads to a reduced TBDD.*

Corollary 6.2. *For a 2-input ternary function $F(A, B)$, if $F_{B^0}(A)$ is constant function or $F_{B^2}(A)$ is constant function (i.e. value of function is same irrespective of value of A) decomposition along input B leads to a reduced TBDD.*

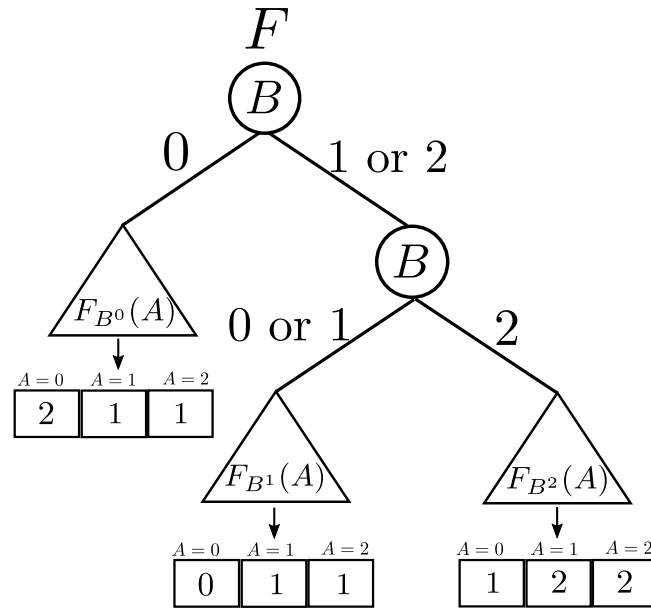


Figure 6.17: TBDD representation when decomposed w.r.t B

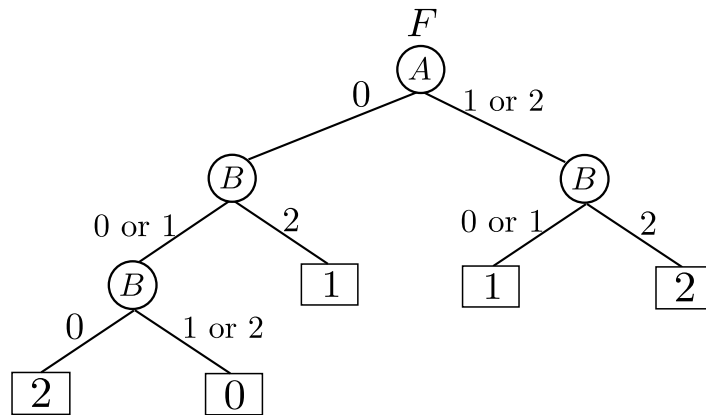


Figure 6.18: TBDD representation for 2 input function in Figure 6.15

Proposition 6.7 is illustrated with an example, whose K-map representation is shown in Figure 6.19. Since $F_{A^2}(B) = 2$, irrespective of value of B , decomposing along input A and using Proposition 6.2 leads to TBDD, which has four nodes as shown in Figure 6.20. Decomposing this function along input B results in TBDD, shown in Figure 6.21, which has six nodes. Hence decomposing with respect to input B leads to reduced TBDD.

The propositions presented in this subsection along with propositions presented in Section 6.3.2 are used to construct a TBDD for any 2-input function. The TBDD is further used circuit implementation using 2:1 multiplexers.

		F			
		B			
		0	1	2	
	A				
0		2	1	1	$\rightarrow F_{A^0}(B)$
1		1	1	0	$\rightarrow F_{A^1}(B)$
2		2	2	2	$\rightarrow F_{A^2}(B)$
		↓	↓	↓	
		$F_{B^0}(A)$	$F_{B^1}(A)$	$F_{B^2}(A)$	

Figure 6.19: An Example for 2-input Function

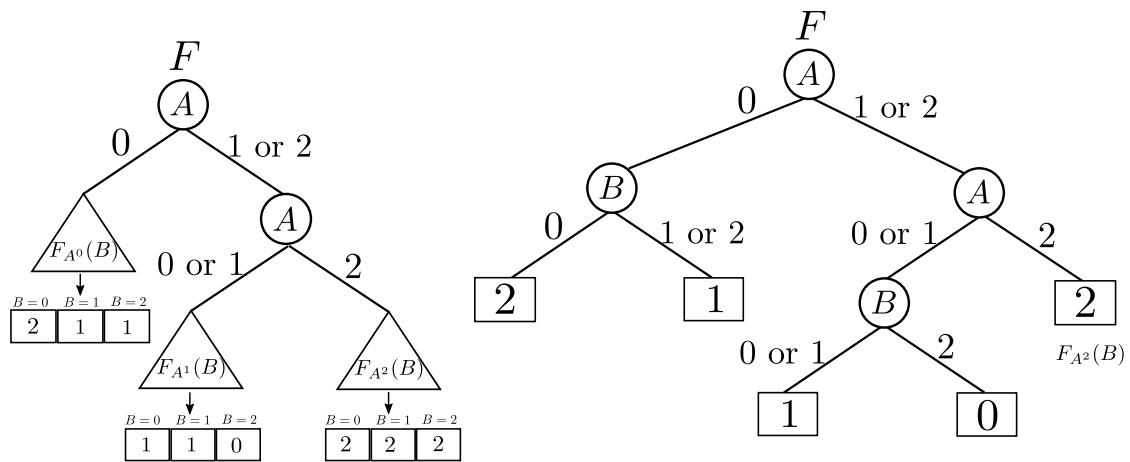


Figure 6.20: TBDD representation for function in Figure 6.19, when decomposed w.r.t A

6.3.4 TBDD-based synthesis for n-input ternary functions

The approach presented to implement a 2-input function can be extended to handle n-input ternary functions. This is achieved by decomposing an n-input function into multiple 1-input functions using Propositions 6.6, 6.7 and related Corollaries 6.1, 6.2 successively. For example, a 3-input function will be decomposed into multiple 2-input functions, which are further decomposed into multiple 1-input functions. As seen earlier, a 2-input function can be decomposed in two ways with respect to inputs (A, B). Similarly, n-input function can be decomposed into (n-1)-input functions in n-ways. One of these n ways is chosen for decomposition with respect to corresponding input using propositions presented below:

Proposition 6.8. For an n-input ternary function $F(x_1, x_2, x_3 \dots x_n)$, if $F_{x_i^0}(x_1 \dots x_{i-1} x_{i+1} \dots x_n) = F_{x_i^1}(x_1 \dots x_{i-1} x_{i+1} \dots x_n) = F_{x_i^2}(x_1 \dots x_{i-1} x_{i+1} \dots x_n)$, then decomposition along input x_i leads

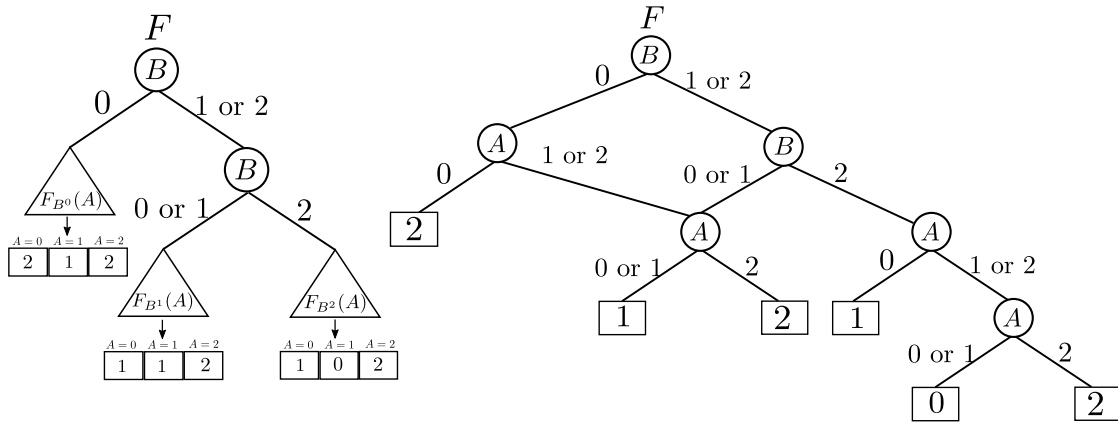


Figure 6.21: TBDD representation for function in Figure 6.19, when decomposed w.r.t B

to reduced TBDD.

Proposition 6.9. For an n -input ternary function $F(x_1, x_2, x_3 \dots x_n)$, if $F_{x_i^0}(x_1 \dots x_{i-1} x_{i+1} \dots x_n) = F_{x_i^1}(x_1 \dots x_{i-1} x_{i+1} \dots x_n)$ or $F_{x_i^1}(x_1 \dots x_{i-1} x_{i+1} \dots x_n) = F_{x_i^2}(x_1 \dots x_{i-1} x_{i+1} \dots x_n)$, then decomposition along input x_i leads to reduced TBDD expressed in terms of $(n-1)$ -input functions.

Proposition 6.10. For an n -input ternary function $F(x_1, x_2, x_3 \dots x_n)$, if $F_{x_i^0}(x_1 \dots x_{i-1} x_{i+1} \dots x_n)$ is constant function or $F_{x_i^2}(x_1 \dots x_{i-1} x_{i+1} \dots x_n)$ is constant function decomposition along input x_i leads to reduced TBDD expressed in terms of $(n-1)$ -input functions.

For an n -input function, if conditions for Proposition 6.8 are satisfied, then Rule 1 can be applied on its TBDD, which is represented in terms of $(n-1)$ -input functions, leading to a reduction. Propositions 6.9 and 6.10 are derived from Propositions 6.6 and 6.7 respectively. The decomposition process is repeated until an n -input function is decomposed into multiple 1-input functions. A TBDD is constructed by appending nodes at each level of decomposition similar to the process used in the 2-input case which is shown in Figure 6.20. At each step of decomposition, the input corresponding to the decomposition is chosen as the decision variable for appended nodes.

6.4 Algorithm for 2:1 Multiplexer based Synthesis

In this section, we present an algorithm which can be used to generate TBDD graph for an n -input ternary function. The inputs to this algorithm, i.e. Algorithm 6.1, are

the list of inputs and the truth table for the ternary function.

Initially a list is created which has a mapping between inputs and the truth table. This is indicated by a function *create_List*(X, O) in Algorithm 6.1. An empty TBDD-graph (number of nodes is equal to zero) namely, *TBDD*, is created initially, which will be updated with nodes and edges. The ternary truth table of n -input ternary function is now decomposed into three truth tables corresponding to $(n - 1)$ -input ternary functions. As discussed earlier, for a n -input ternary function, there are n -ways in which it can be decomposed. One of these ways is selected based on the Propositions 6.8, 6.9 and 6.10. If the decomposition is possible with at least one of these propositions then, *decompose* is set to 1 and the corresponding input is removed from the input list ($X' = \text{remove}(i, X')$). A new list, *listnew* is created (using functions *emptylist*(), *append*()), which now consists of truth tables and input list corresponding to $(n - 1)$ inputs. Also, *TBDD* is updated (indicated by function *updateGraph*()), with the help of Propositions 6.2 and 6.3, by choosing an input, which is used for decomposition, as selection signal.

If the conditions for Propositions 6.8, 6.9 are not met 6.10 (*decompose* = 0), then the default decomposition is done along the input corresponding to Most Significant Digit (MSD). The *listnew* and TBDD-graph are updated accordingly. There is a possibility that the *listnew* contains duplicate functions. Hence duplicate functions are removed and the corresponding nodes in the TBDD-graph (*TBDD*) are merged. This process is represented in Algorithm 6.1 by a function *remove_fun*(*TBDD*, *listnew*), which returns the updated list and TBDD graph.

The new list containing decomposed functions is copied to *out_list*. The process of decomposition is repeated until the original function is decomposed into 1-input functions. This condition is checked by extracting the size of truth tables from *out_list*., and checking if it is equal to 3. Once an n -input function is decomposed into 1-input functions, TBDD graphs for these functions are created using Propositions 6.2, 6.3, 6.4 and 6.5, and are appended to the original graph represented as *TBDD*. Finally the Algorithm 6.1 returns the TBDD graph which can be used for circuit implementation

Algorithm 6.1 Algorithm for Synthesis

```

1: Input1: list of Inputs  $X = (x_1, x_2, x_3 \dots x_n)$ 
2: Input2: Truth table as a list  $O = (O_0, O_1 \dots O_{3^n-1})$ 
3: Output: TBDD graph corresponding to Ternary function
4: begin
5:  $out\_list = create\_List(X, O)$  //creates a list  $[[X, O]]$ 
6:  $out\_length = get\_Size(O)$  //get the truth table size
7:  $TBDD = Create\_emptygraph()$ 
8: while ( $out\_length > 3$ ) do //check for 1-input function
9:   for each  $list$  in  $out\_list$  do
10:     $X' = get\_inputs(list)$ 
11:     $O' = get\_output\_truthtable(list)$ 
12:     $listnew = emptylist()$  //creates an empty list
13:     $decompose = 0$  //Start Decomposition
14:    for each  $i$  in  $X'$  do
15:       $(O'_{i=0}, O'_{i=1}, O'_{i=2}) = Decompose(O', i)$ 
16:      if ( $O'_{i=0} = O'_{i=1} = O'_{i=2}$ ) then //Proposition 6.8
17:         $X' = remove(i, X')$ 
18:         $append(listnew, [[X', O'_{i=0}]])$ 
19:         $decompose = 1$ 
20:         $updateGraph(TBDD, i, O'_{i=0})$ 
21:      end if
22:      break
23:      else if ( $O'_{i=0} = O'_{i=1}$ ) then //Proposition 6.9 & 6.3
24:         $X' = remove(i, X')$ 
25:         $append(listnew, [[X', O'_{i=0}], [X', O'_{i=2}]])$ 
26:         $decompose = 1$ 
27:         $updateGraph(TBDD, i, [O'_{i=0}, O'_{i=2}])$  Use Proposition 6.2 or 6.3
28:      end if
29:      break
30:      //Similarly check for Proposition 6.10
31:    end for
32:    if ( $decompose = 0$ ) then
33:      // i.e. if conditions for Propositions 6.8, 6.9 and 6.10
34:      // are not met then decompose along MSD
35:      // update  $listnew$  and  $TBDD$  graph
36:    end if
37:    end for
38:     $(TBDD, listnew) = remove\_fun(TBDD, listnew)$ 
39:     $out\_list = listnew$ 
40:     $O' = get\_output\_truthtable(listnew)$ 
41:     $out\_length = get\_Size(O')$ 
42:  end while
43:  for each  $list$  in  $out\_list$  do
44:     $X' = get\_inputs(list)$ 
45:     $O' = get\_output\_truthtable(list)$ 
46:     $TBDD = fun\_one\_input(X', O', TBDD)$ 
47:  end for
48: return  $TBDD$ 
49: end

```

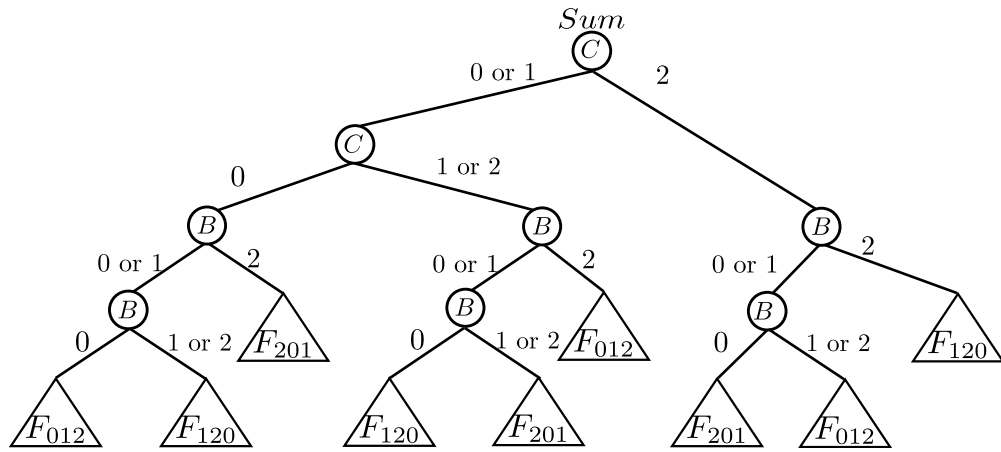
using 2:1 multiplexers. As an example, TBDD graphs for ternary full adder obtained from the proposed synthesis algorithm is shown in Figure 6.22. Ternary adder has three inputs A , B and C and two outputs Sum and $Carry$. Figures 6.22(a) and 6.22(b) show the decomposition of 3-input Sum and $Carry$ function resulting in 1-input functions, whose TBDDs are shown in Figures 6.22(c) and 6.22(d) .

Algorithm 6.1 decomposes the n -input ternary function into 3 $(n - 1)$ -input functions using one of the possible n -ways. In the worst case the algorithm iterates through all the possible n -ways corresponding to n inputs, available for decomposition. Further the algorithm decomposes each of the $(n - 1)$ -input ternary functions into $(n - 2)$ -input functions using one of the possible $(n - 1)$ -ways. This process is repeated until the original function is decomposed into 1-input functions. For a single output ternary function, the algorithm complexity for decomposing a n -input ternary function into 1-input functions is $O(n3^n)$. In the worst case, for any value of n , there can be 27 unique 1-input functions. Hence the time complexity for implementing 1-input functions is independent of n value and does not contribute to the overall complexity of the algorithm. For an n -input, m -output ternary function the complexity of algorithm is $O(mn3^n)$. This complexity can be reduced by selecting the input variable for decomposition randomly, instead of iterating through all the inputs. Although this technique reduces the complexity of the algorithm to $O(m3^n)$, it might lead to sub-optimal circuits.

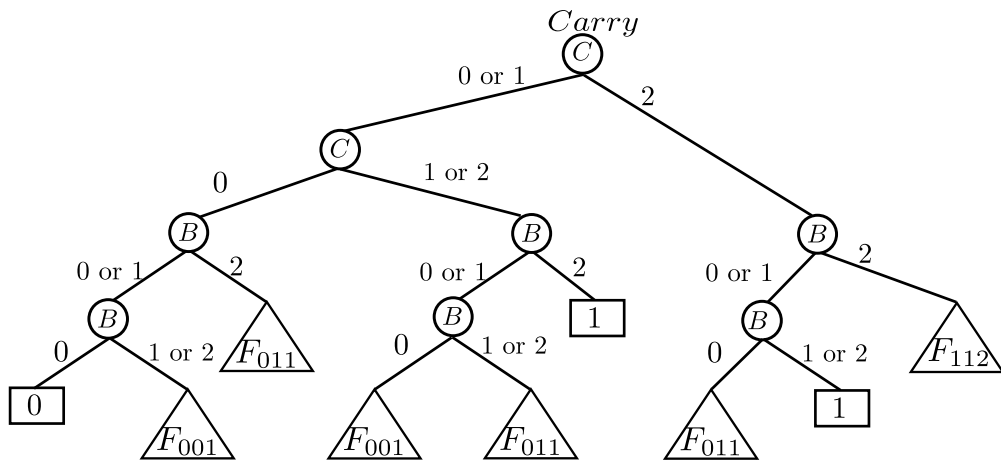
Algorithm 6.1 can be used to synthesize ternary logic circuits using 2:1 multiplexer, Binary NOT, PTI and NTI gates. In this work, the proposed algorithm is used to synthesize CNFET-based ternary logic circuits and is compared with the existing CNFET-based ternary synthesis algorithm. However, the proposed algorithm is not limited to CNFET technology and can be applied to any device technology which supports the implementation of 2:1 multiplexer, Binary NOT, PTI and NTI gates.

6.5 Synthesis using CNFETs

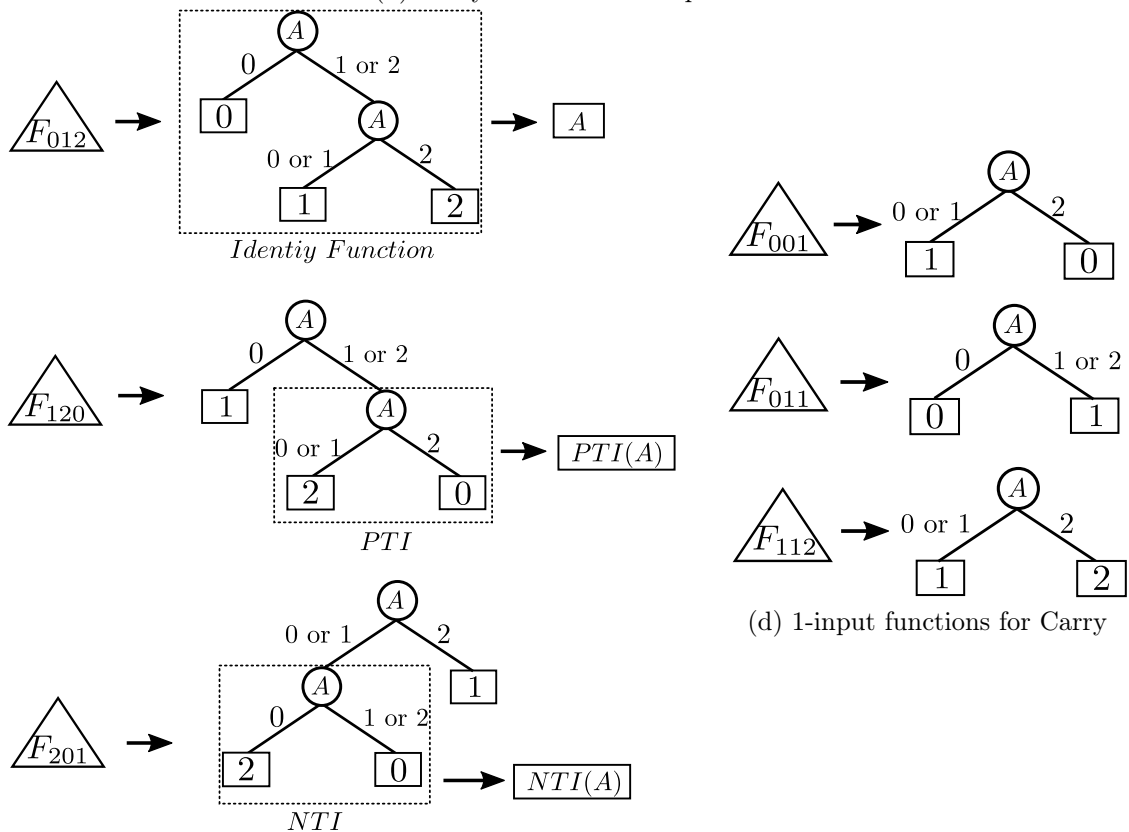
In the proposed TBDD-based approach, TBDD is converted into circuit implementation by replacing the nodes with 2:1 multiplexers. Hence this approach needs circuit



(a) Sum Function Decomposition



(b) Carry Function Decomposition



(c) 1-input functions for Sum

(d) 1-input functions for Carry

Figure 6.22: TBDD for Ternary Full Adder

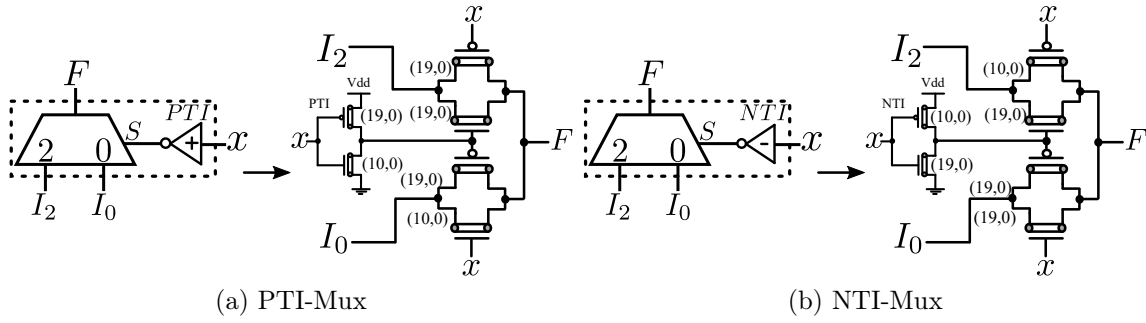


Figure 6.23: Implementation of 2:1 Multiplexers

implementations for 2:1 multiplexers (NTI-Mux and PTI-Mux), NTI and PTI. As explained in Section 6.3, two types of 2:1 multiplexers are used in synthesis of ternary circuits. Figures 6.23a and 6.23b show the implementation of these multiplexers, PTI and NTI using CNFETs. Each of the multiplexers require two transmission gates (4 transistors) and a PTI or an NTI (2 transistors) for implementation. In addition to these circuits, a binary NOT gate is also used to implement templates shown in Figure 6.11. The binary NOT gate can be implemented with transistors, which use CNTs of any of the diameters shown in Table 2.3. In this work, binary NOT gate is implemented using transistors which use CNTs with diameter $1.487nm$.

The procedure to implement the TBDD using CNFET-based 2:1 multiplexers is illustrated with an example. Consider a 2-input ternary function shown in Figure 6.15 and its TBDD shown in the Figure 6.18. Here each node corresponds to a 2:1 multiplexer. However, some of the nodes which are equivalent to one of the templates shown in 6.10 and 6.11 are replaced with their equivalent circuits. Figure 6.24 shows block-level and transistor-level implementation of TBDD shown in Figure 6.18. It should be noted that if two multiplexers have same select signal then the total number of fansistors required is equal to 18 (four transmission gates plus one inverter) and not 24.

Figure 6.25 shows the block-level implementation of ternary adder TBDDs (shown in Figure 6.22) with three inputs A , B and C and two outputs Sum and $Carry$. The implementation of ternary adder using the proposed approach requires 21 2:1 multiplexers (10 for Sum and 11 for $Carry$), where each multiplexer requires two transmission gates (4 transistors). The select signals generated depend on the number

of inputs. In a ternary adder, 3 NTI and 3 PTI gates are required for select signal generation. Hence a ternary full adder, implemented using the proposed approach requires $(21 \times 4) + (6 \times 2)$, i.e. a total of 96 CNFETs which is less when compared to 3:1 multiplexer based implementation [41] which requires 105 CNFETs. Different benchmark ternary circuits have been synthesized using the proposed approach and algorithm. The synthesis and simulation results of the benchmark circuits have been presented in next section.

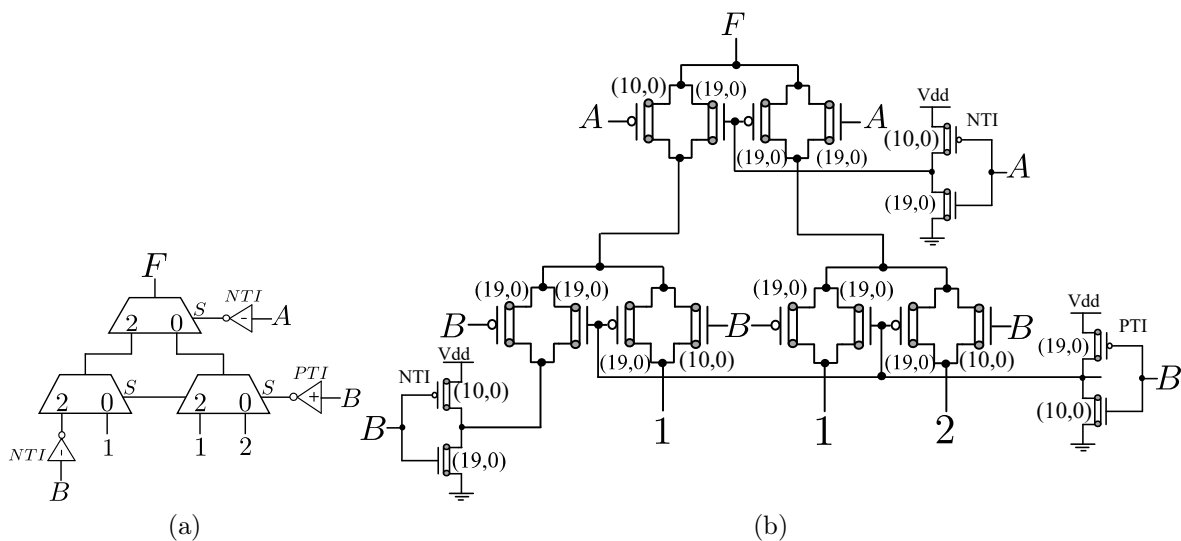


Figure 6.24: Implementation of TBDD in Figure 6.18

6.6 Results

6.6.1 Synthesis

The proposed algorithm (Algorithm 6.1) has been implemented using Python 3.5 in Scientific PYthon Development EnviRonment (Spyder) on Windows 10 running on a Intel(R) Core(TM) *i5* – 5200U CPU 64-bit@2.2GHz and 4GB of RAM. The algorithm takes the truth table along with inputs and generates the TBDD graph which is used for circuit implementation. In the proposed approach, each node in the TBDD is equivalent to one 2:1 multiplexer (two transmission gates) and the selection signals for these are generated from inputs. The selection signal is generated by using either an NTI or a PTI

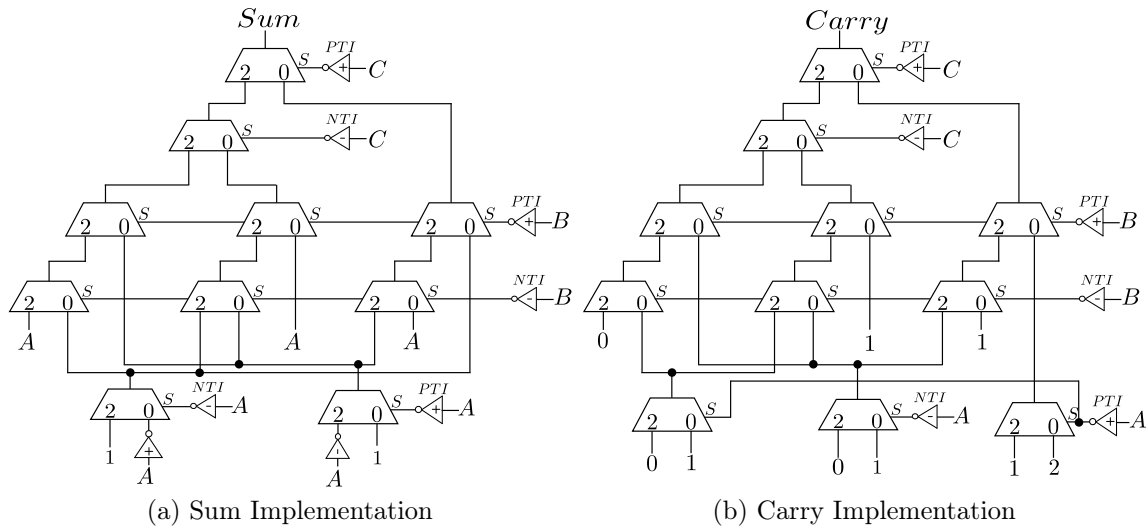


Figure 6.25: Implementation of Ternary Full Adder

gate. Hence in the worst case, the number of inverters required to generate all required selection signals is equal to two multiplied by number of inputs. In addition, there might be a need of binary NOT gates to replace some of the templates with equivalent signals. Each binary NOT gate requires two CNFETs for implementation. For testing the efficiency of the proposed 2:1 multiplexer based synthesis, ternary benchmark functions [29, 41] have been considered. The functions include (i) sum_i , which outputs the *sum* of i ternary inputs (ii) $prod_i$, which finds the *product* of i ternary inputs (iii) avg_i to calculate the *average* of i ternary inputs (iv) ncy_r which corresponds to a ternary sum-of-product expression of n input variables taken r at a time and (v) $counter_i$ which counts number of 1's and 2's in a given i digit ternary number. As in the case of work presented in [41], we have chosen $n \geq 8$ and $r \geq 5$ for ncy_r and $i \geq 8$ for other functions to evaluate the performance of the proposed approach.

Table 6.2 shows the comparison of the proposed 2:1 multiplexer based synthesis approach with 3:1 multiplexer (and unary operator) based synthesis approach presented in [41], for different ternary benchmark functions. This table also shows the execution time of the proposed algorithm for different benchmarks. It can be observed that the execution times are consistent with the complexity analysis carried out on the algorithm presented in Section 6.4. Synthesis results of ternary benchmark functions using the proposed approach show a significant reduction (up to 99%) in the number

of CNFETs used when compared to [41]. This improvement is mainly due to two factors 1) use of 2:1 multiplexers based approach 2) proposed algorithm (Algorithm 6.1) eliminates duplicate functions, which is equivalent to merging equivalent sub-graphs (nodes), which is not the case in [41].

6.6.2 SPICE Simulation

In addition to synthesis, the benchmark circuits have been simulated in HSPICE. A program, which is used to convert the TBDD into a spice netlist which uses sub-circuits for 2:1 multiplexers and inverter gates, is implemented in Python 3.5. The resulting netlist have been simulated using Synopsys HSPICE. All the circuits are simulated using the CNTFET model of [44, 45, 53] at $0.9V$ power supply and room temperature. The CNFETs used in the implementation are configured to have three tubes and a default pitch value equal to $20nm$. All the other parameters are set to their default values as presented in Table 2.4. In this work, ternary logic values 0, 1 and 2 correspond to voltages 0, $V_{dd}/2$ and V_{dd} respectively. For binary logic gates the logic values 0 and 1 correspond to voltages 0 and V_{dd} respectively. Binary gates are implemented using transistors, which are connected in complementary logic style and have chirality of $(19, 0)$. Different ternary circuits are implemented using proposed and existing approaches and the design parameters are compared. For fair comparison, all the adders have been simulated with same test pattern.

Power consumption results are obtained by simulating the circuits with random test input patterns at switching frequency of $500MHz$. To measure the delay of the designs, test patterns have been chosen in such a way that the signal change propagates through the critical path and the maximum delay is measured. FO4 delay is calculated by loading the output node with four STI gates (implementation of STI gate is presented in [36]). The power-delay product is the product of worst-case propagation delay and average power consumption. Table 6.3 shows the propagation delay, power consumption and power-delay product of the simulated ternary benchmark circuits.

Although, the work presented in [41] shows the HSPICE simulation results, it does

Table 6.2: Comparison of Transistor Count of Proposed 2:1 Multiplexer based Algorithm with that of Existing 3:1 Multiplexer based Algorithm [41]

Ternary Function	I/O	No. of 2:1 MUXes	No. of CNFETs		Improv.	Exec. time (s)
			Proposed Approach	3:1 MUX based approach [41]		
sum_8	8/3	194	812	960	15.4%	0.8
sum_9	9/3	240	1000	2346	57.4%	2.4
sum_{10}	10/3	299	1240	6822	81.8%	8.2
sum_{11}	11/3	358	1480	19968	92.6%	28.3
sum_{12}	12/3	417	1720	59346	97.1%	93.0
sum_{13}	13/3	476	1960	177×10^3	98.9%	307.1
$prod_8$	8/6	210	876	934	6.2%	1.0
$prod_9$	9/6	290	1200	2404	50.1%	3.6
$prod_{10}$	10/7	385	1584	6796	76.7%	13.5
$prod_{11}$	11/7	496	2032	19936	89.8%	45.9
$prod_{12}$	12/8	629	2568	59320	95.7%	170.1
$prod_{13}$	13/9	803	3268	177×10^3	98.2%	621.2
avg_8	8/1	68	308	871	64.6%	0.2
avg_9	9/1	86	384	2341	83.6%	0.7
avg_{10}	10/1	105	464	6727	93.1%	2.4
avg_{11}	11/1	127	556	19861	97.2%	8.1
avg_{12}	12/1	150	652	59239	98.9%	26.1
avg_{13}	13/1	176	760	177×10^3	99.6%	105.6
$8cy_5$	8/1	143	608	908	33.0%	0.2
$9cy_5$	9/1	238	992	2378	58.3%	0.8
$10cy_6$	10/1	295	1224	6764	81.9%	2.9
$11cy_9$	11/1	161	692	19898	96.5%	9.2
$12cy_8$	12/1	533	2184	61258	96.4%	33.7
$13cy_7$	13/1	1118	4528	177×10^3	97.4%	114.9
$14cy_8$	14/1	1256	5084	531×10^3	99.0%	387.6
$counter_8$	8/4	142	604	958	36.9%	0.5
$counter_9$	9/6	203	852	2508	66.0%	2.5
$counter_{10}$	10/6	266	1108	6924	84.0%	9.5
$counter_{11}$	11/6	329	1364	20098	93.2%	31.7
$counter_{12}$	12/6	392	1620	59521	97.3%	103.7
$counter_{13}$	13/6	455	1876	177×10^3	98.9%	381.2

Table 6.3: Simulation Results for Ternary Benchmark Circuits

Ternary Function	Propagation Delay/FO4-Delay (ps)	Power Consumption (μW)	Power-Delay Product (fJ)
sum_8	173.3	1.46	0.252
sum_{10}	235.0	2.07	0.485
sum_{12}	298.1	2.55	0.761
$prod_8$	235.9	2.14	0.503
$prod_{10}$	349.3	3.03	1.06
$prod_{12}$	490.4	4.37	2.14
avg_8	163.7	0.334	0.055
avg_{10}	245.4	0.519	0.127
avg_{12}	343.0	0.692	0.237
$8cy_5$	168.0	0.659	0.111
$10cy_6$	197.0	1.29	0.254
$12cy_8$	302.3	1.66	0.502
$counter_8$	104.7	1.35	0.141
$counter_{10}$	144.8	2.42	0.350
$counter_{12}$	166.2	3.26	0.541

not mention the simulation environment like number of tubes used in CNFET, input switching frequency etc. Moreover the circuits synthesized using algorithm in [41] are available only for ternary adder and not for other circuits. Hence, a direct comparison of simulation results is presented for ternary full adder. Table 6.4 shows the comparison of simulation results for ternary adder synthesized using the proposed and existing approaches with different loads. As seen from the table, ternary adder implemented using the proposed synthesis approach has up to 87% less power consumption and 86% less PDP when compared to adder implemented using 3:1 multiplexer based synthesis approach [41]. This is mainly because, the 3:1 multiplexer based approach uses complex circuits, which have large power consumption, for implementing unary operators. The proposed approach uses transmission gate based 2:1 multiplexers for implementing the unary operators. For FO4 load, ternary adder implemented using the proposed approach has 7% more propagation delay when compared to 3:1 multiplexer based approach. However as the load increases, 2:1 multiplexer based ternary adder achieves a reduction of up to 17% in delay when compared to 3:1 multiplexer based adder.

In addition to the above comparison, we also present a comparison of some of the

Table 6.4: Comparison of Ternary Adders

	Load	Propagation Delay/FO4-Delay (ps)	Power Consumption (μW)	Power-Delay Product (fJ)
[41]	$FO4$	62.31	2.14	0.133
	$1fF$	135.8	2.24	0.305
	$2fF$	242.5	2.46	0.596
	$3fF$	349.2	2.67	0.933
Proposed	$FO4$	66.53	0.28	0.019
	$1fF$	124.3	0.40	0.050
	$2fF$	205.5	0.58	0.119
	$3fF$	289.7	0.76	0.220

existing manual designs of ternary circuits (presented in [36, 60, 61, 74]) with those generated from the proposed synthesis algorithm. Table 6.5 shows the comparison for half-adder, 1-digit multiplier and 1-digit comparator. As seen from this table, the proposed algorithm results in half-adder, 1-digit multiplier and 1-digit comparator circuits that require least number of CNFETs for implementation when compared to existing designs. A half-adder implemented using the proposed synthesis algorithm has least power consumption and power-delay product when compared to the existing designs presented in [36, 60]. It is also evident from Table 6.5 that 1-digit multiplier synthesized using the proposed algorithm is better than existing designs presented in [36, 61], with respect to all design parameters. However for 1-digit comparator, the existing design presented in [74] has less propagation delay, power consumption and power-delay product when compared to the one synthesized using the proposed algorithm.

6.7 Conclusions

This chapter presented a methodology to transform a TDD to BDD. The transformed decision diagram, also called as Ternary-Transformed Binary Decision Diagram (TBDD), is then used to implement the ternary logic functions using 2:1 multiplexers. This methodology is used to develop a synthesis algorithm, which is used to synthesize

Table 6.5: Comparison of Existing Manual Designs with those Generated using the Proposed Algorithm

	Propagation Delay (ps)	Power Consumption (μW)	Power-Delay Product (in $\times 10^{-17} J$)	Number of CNFETs
Half- Adder				
[36]	42.4	1.24	5.25	88
[60]	47.0	1.62	7.62	52
Proposed	44.2	0.121	0.53	36
1-digit Multiplier				
[36]	35.7	0.780	2.78	66
[61]	26.4	0.779	2.06	40
Proposed	17.5	0.069	0.12	30
1-digit Comparator				
[36]	34.3	0.294	1.01	52
[74]	19.7	0.201	0.40	32
Proposed	35.5	0.249	0.88	28

various ternary benchmark functions. Synthesis results for ternary benchmark functions indicate that the proposed algorithm results in circuits that have, on an average 79% and up to 99% less transistors when compared to recent 3:1 multiplexer based algorithm. The synthesized circuits have been implemented using Carbon-Nanotube Field Effect Transistors and simulated in HSPICE. Simulation results for ternary adder show that the proposed synthesis approach results in upto 17% reduction (for load of $3fF$) in propagation delay, 87% reduction in power consumption and 86% reduction in power-delay product when compared to 3:1 multiplexer based synthesis approach.

Chapter 7

Summary and Future Work

7.1 Summary

This thesis presented new design and synthesis techniques to implement ternary logic circuits using CNFET. Initially, three new design approaches (Approach I, II and III) were presented which are used in the implementation of ternary logic circuits. In addition, new designs for ternary decoder and low-power encoder are presented. Simulation results of the resulting circuits show reduction in different design metrics such as power consumption, propagation delay and power-delay product, when compared to existing design approaches. A ternary half-adder implemented using Approach II, which uses delay optimized decoder, and both sum-of-product and product-of-sum expressions, has the least delay. This adder has less dependency on load variations and has lower power consumption when compared to the existing designs. Thus, this half-adder has been used in the implementation of multi-digit ternary adders.

The existing multi-digit adders have complex carry propagation paths. Hence, two new techniques have been proposed, which lead to implementation of multi-digit ternary adders with optimized carry propagation/generation path. The first technique uses half-adder outputs instead of original inputs for carry computation thus resulting in a carry generation/propagation circuit with less complexity. The second technique uses the concept of propagate-generate to implement ternary prefix adders. HSPICE simulation results show that the proposed ternary prefix adders have up to 58% less

power consumption and 50% less propagation delay when compared to existing multi-digit ternary adders.

Analysis of different adder designs shows that ternary encoder contributes significantly to the overall propagation delay and power consumption. Hence different encoder designs have been presented which are optimized for different design parameters. Also algorithms have been developed to choose appropriate encoders for output stages of ternary circuits such that the overall circuit is optimized for delay or power or power-delay product. These algorithms have been validated by applying them on an example ternary circuit. Simulation results show that the proposed encoder design and algorithms aid in optimizing the ternary logic circuits.

Techniques available for designing ternary logic circuits are mainly classified as encoder based and multiplexer based. For scaling these approaches to implement more complex ternary logic circuits, efficient synthesis algorithms are required. Encoder based approach relies on binary circuits which are then transformed to final ternary outputs with the help of encoders. Hence existing binary synthesis techniques can be used implement ternary circuits. But for scaling the multiplexer based approaches there is a need for new synthesis algorithms. In this work we presented a novel synthesis technique to implement ternary logic circuits using “2:1 multiplexers”. This technique uses a transformation which converts a TDD in to BDD. This transformed TDD is then used to implement the ternary logic function using 2:1 multiplexers.

A procedure to decompose ternary functions with three (or more) inputs to multiple 1-input ternary functions has also been presented in this work. This procedure is developed into a synthesis algorithm, which is further used in the synthesis of benchmark ternary functions. Synthesis results of these functions indicate that the proposed algorithm results in circuits that have, on an average, 79% and up to 99% less transistors when compared to the existing 3:1 multiplexer based designs. Simulation results for ternary adder, implemented using proposed algorithm, show that there is upto 17% reduction (for load of $3fF$) in propagation delay, 87% reduction in power consumption and 86% reduction in power-delay product when compared with adder implemented

using 3:1 multiplexers.

7.2 Future Work

Any technology, where it is possible to have transistors with atleast two different threshold voltages ($0 < V_{th1} < VDD/2$, $VDD/2 < V_{th2} < VDD$), can be used to implement the ternary logic circuits. It would be interesting to explore the proposed techniques in context of these new device technologies. The existing literature on ternary logic mainly focuses on design of combinational ternary circuits. Hence new design and synthesis techniques to implement sequential ternary circuits can also be explored.

In Chapter 5, we proposed some designs of ternary encoder that are optimized for different design parameters. Encoder is a critical element that significantly affects the overall power consumption and propagation delay of a ternary circuit. It would be interesting to explore the possibility of designing more efficient encoders which would aid in designing more efficient ternary circuits.

In Chapter 6 a novel technique to implement ternary circuits using “2:1 multiplexers” has been presented. This approach can be further extended to implement other MVL circuits using 2:1 multiplexers. In this context, it would be interesting to explore more generalized synthesis techniques which use 2:1 multiplexers for implementation of any MVL circuit.

Bibliography

- [1] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. USA: Addison-Wesley Publishing Company, 2010.
- [2] E. Mollick, “Establishing moore’s law,” *IEEE Ann. Hist. Comput.*, vol. 28, no. 3, pp. 62–75, Jul. 2006. [Online]. Available: <http://dx.doi.org/10.1109/MAHC.2006.45>
- [3] M. Horowitz, “Scaling, power and the future of cmos,” in *Proceedings of the 20th International Conference on VLSI Design Held Jointly with 6th International Conference: Embedded Systems*, ser. VLSID ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 23–. [Online]. Available: <http://dx.doi.org/10.1109/VLSID.2007.140>
- [4] X. Huang, W.-C. Lee, C. Kuo, D. Hisamoto, L. Chang, J. Kedzierski, E. Anderson, H. Takeuchi, Y.-K. Choi, K. Asano, V. Subramanian, T.-J. King, J. Bokor, and C. Hu, “Sub-50 nm p-channel finfet,” *IEEE Transactions on Electron Devices*, vol. 48, no. 5, pp. 880–886, May 2001.
- [5] “International technology roadmap for semiconductors 2.0 2015 edition executive report,” Available: <http://www.itrs.net>, Tech. Rep., 2015.
- [6] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, “Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits,” *Proceedings of the IEEE*, vol. 91, no. 2, pp. 305–327, Feb 2003.

- [7] E. P. DeBenedictis, J. K. Mee, and M. P. Frank, “The opportunities and controversies of reversible computing,” *Computer*, vol. 50, no. 6, pp. 76–80, 2017.
- [8] V. Gaudet, “A survey and tutorial on contemporary aspects of multiple-valued logic and its application to microelectronic circuits,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 1, pp. 5–12, March 2016.
- [9] S. J. Tans, A. R. M. Verschueren, and C. Dekker, “Room-temperature transistor based on a single carbon nanotube,” *Nature*, vol. 393, no. 6680, pp. 49–52, May 1998.
- [10] S. Karmakar, J. A. Chandy, and F. C. Jain, “Design of ternary logic combinational circuits based on quantum dot gate fets,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 5, pp. 793–806, May 2013. [Online]. Available: <http://dx.doi.org/10.1109/TVLSI.2012.2198248>
- [11] A. Javey, J. Guo, Q. Wang, M. Lundstrom, and H. Dai, “Ballistic carbon nanotube field-effect transistors,” *Nature*, vol. vol. 424, pp. 654–657, 2003.
- [12] J. Deng, N. Patil, K. Ryu, A. Badmaev, C. Zhou, S. Mitra, and H. S. P. Wong, “Carbon nanotube transistor circuits: Circuit-level performance benchmarking and design options for living with imperfections,” in *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, Feb 2007, pp. 70–588.
- [13] Y. Cui, Z. Zhong, D. Wang, W. U. Wang, and C. M. Lieber, “High performance silicon nanowire field effect transistors,” *Nano Letters*, vol. 3, no. 2, pp. 149–152, 2003. [Online]. Available: <http://dx.doi.org/10.1021/nl025875l>
- [14] X. Wang, Y. Ouyang, X. Li, H. Wang, J. Guo, and H. Dai, “Room-temperature all-semiconducting sub-10-nm graphene nanoribbon field-effect transistors,” *Phys. Rev. Lett.*, vol. 100, p. 206803, May 2008. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevLett.100.206803>

- [15] M. C. Lemme, T. J. Echtermeyer, M. Baus, and H. Kurz, "A graphene field-effect device," *IEEE Electron Device Letters*, vol. 28, no. 4, pp. 282–284, April 2007.
- [16] T. Ashley, A. R. Barnes, L. Buckle, S. Datta, A. B. Dean, M. T. Emery, M. Fearn, D. G. Hayes, K. P. Hilton, R. Jefferies, T. Martin, K. J. Nash, T. J. Phillips, W. A. Tang, P. J. Wilding, and R. Chau, "Novel insb-based quantum well transistors for ultra-high speed, low power logic applications," in *Proceedings. 7th International Conference on Solid-State and Integrated Circuits Technology, 2004.*, vol. 3, Oct 2004, pp. 2253–2256 vol.3.
- [17] C. I. Kuo, H. T. Hsu, C. Y. Wu, E. Y. Chang, Y. Miyamoto, Y. L. Chen, and D. Biswas, "A 40-nm-gate inas/in_{0.7}ga_{0.3}as composite-channel hemt with 2200 ms/mm and 500-ghz ft," in *2009 IEEE International Conference on Indium Phosphide Related Materials*, May 2009, pp. 128–131.
- [18] S. L. Hurst, "Multiple-valued logic: its status and its future," *IEEE Transactions on Computers*, vol. C-33, no. 12, pp. 1160–1179, Dec 1984.
- [19] P. C. Balla and A. Antoniou, "Low Power Dissipation MOS Ternary Logic Family," *IEEE Journal of Solid-State Circuits*, vol. 19, no. 5, pp. 739–749, Oct 1984.
- [20] A. Heung and H. T. Mouftah, "Depletion/enhancement CMOS for a Lower Power Family of Three-valued Logic Circuits," *IEEE Journal of Solid-State Circuits*, vol. 20, no. 2, pp. 609–616, Apr 1985.
- [21] D. A. Rich, "A Survey of Multivalued Memories," *IEEE Trans. Comput.*, vol. 35, no. 2, pp. 99–106, Feb. 1986. [Online]. Available: <http://dx.doi.org/10.1109/TC.1986.1676727>
- [22] Y. Yasuda, Y. Tokuda, S. Zaima, K. Pak, T. Nakamura, and A. Yoshida, "Realization of Quaternary Logic Circuits by n-channel MOS Devices," *IEEE Journal of Solid-State Circuits*, vol. 21, no. 1, pp. 162–168, Feb 1986.
- [23] I. Jordan and H. Mouftah, "Design of ternary cos/mos memory and sequential circuits," *IEEE Transactions on Computers*, vol. 26, pp. 281–288, 1977.

- [24] D. Mateo and A. Rubio, "Design and implementation of a 5 times;5 trits multiplier in a quasi-adiabatic ternary cmos logic," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 7, pp. 1111–1116, Jul 1998.
- [25] I. Halpern and M. Yoeli, "Ternary arithmetic unit," *Electrical Engineers, Proceedings of the Institution of*, vol. 115, no. 10, pp. 1385–1388, October 1968.
- [26] S. Kawahito, M. Kameyama, T. Higuchi, and H. Yamada, "A 32*32-bit multiplier using multiple-valued mos current-mode circuits," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 1, pp. 124–132, Feb 1988.
- [27] S. Y. H. Su and P. T. Cheung, "Computer minimization of multivalued switching functions," *IEEE Transactions on Computers*, vol. C-21, no. 9, pp. 995–1003, Sept 1972.
- [28] H. M. Wang, C. L. Lee, and J. E. Chen, "Algebraic division for multilevel logic synthesis of multi-valued logic circuits," in *Multiple-Valued Logic, 1994. Proceedings., Twenty-Fourth International Symposium on*, May 1994, pp. 44–51.
- [29] M. Hawash, M. Lukac, M. Kameyama, and M. Perkowski, "Multiple-valued reversible benchmarks and extensible quantum specification (xqs) format," in *2013 IEEE 43rd International Symposium on Multiple-Valued Logic*, May 2013, pp. 41–46.
- [30] K. W. Current, "Current-mode cmos multiple-valued logic circuits," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 2, pp. 95–107, Feb 1994.
- [31] M. Klein, J. A. Mol, J. Verduijn, G. P. Lansbergen, S. Rogge, R. D. Levine, and F. Remacle, "Ternary logic implemented on a single dopant atom field effect silicon transistor," *Applied Physics Letters*, vol. 96, no. 4, p. 043107, 2010. [Online]. Available: <http://dx.doi.org/10.1063/1.3297906>
- [32] A. Rahman, J. Guo, S. Datta, and M. S. Lundstrom, "Theory of ballistic nanotransistors," *IEEE Transactions on Electron Devices*, vol. 50, no. 9, pp. 1853–1864, Sept 2003.

- [33] Y.-M. Lin, J. Appenzeller, J. Knoch, and P. Avouris, "High-performance carbon nanotube field-effect transistor with tunable polarities," *IEEE Trans. Nanotechnol.*, vol. 4, no. 5, pp. 481–489, Sep. 2005. [Online]. Available: <http://dx.doi.org/10.1109/TNANO.2005.851427>
- [34] A. Akturk, G. Pennington, N. Goldsman, and A. Wickenden, "Electron transport and velocity oscillations in a carbon nanotube," *IEEE Transactions on Nanotechnology*, vol. 6, no. 4, pp. 469–474, July 2007.
- [35] H. Hashempour and F. Lombardi, "Device Model for Ballistic CNFETs Using the First Conducting Band," *IEEE Des. Test*, vol. 25, no. 2, pp. 178–186, Mar. 2008. [Online]. Available: <http://dx.doi.org/10.1109/MDT.2008.34>
- [36] S. Lin, Y. B. Kim, and F. Lombardi, "CNTFET-Based Design of Ternary Logic Gates and Arithmetic Circuits," *IEEE Transactions on Nanotechnology*, vol. 10, no. 2, pp. 217–225, March 2011.
- [37] —, "A Novel CNTFET-based Ternary Logic Gate Design," in *2009 52nd IEEE International Midwest Symposium on Circuits and Systems*, Aug 2009, pp. 435–438.
- [38] P. Keshavarzian and R. Sarikhani, "A Novel CNTFET-based Ternary Full Adder," *Circuits, Systems, and Signal Processing*, vol. 33, no. 3, pp. 665–679, 2014.
- [39] R. F. Mirzaee, K. Navi, and N. Bagherzadeh, "High-Efficient Circuits for Ternary Addition," *VLSI Design*, vol. 2014, Article ID 534587, 15 pages, 2014. doi:10.1155/2014/534587, vol. 2014, 2014.
- [40] S. L. Murotiya and A. Gupta, "Hardware-Efficient Low-power 2-bit Ternary ALU Design in CNTFET Technology," *International Journal of Electronics*, vol. 103, no. 5, pp. 913–927, 2016. [Online]. Available: <http://dx.doi.org/10.1080/00207217.2015.1082199>

- [41] B. Srinivasu and K. Sridharan, "A synthesis methodology for ternary logic circuits in emerging device technologies," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 8, pp. 2146–2159, Aug 2017.
- [42] R. Saito, G. Dresselhaus, and M. S. Dresselhaus, *Physical Properties of Carbon Nanotubes*. Imperial College Press, London, 1998.
- [43] J. Appenzeller, "Carbon Nanotubes for High-Performance Electronics - Progress and Prospect," *Proceedings of the IEEE*, vol. 96, no. 2, pp. 201–211, Feb 2008.
- [44] J. Deng and H. S. P. Wong, "A Compact SPICE Model for Carbon-Nanotube Field-Effect Transistors Including Nonidealities and Its Application-Part I: Model of the Intrinsic Channel Region," *IEEE Transactions on Electron Devices*, vol. 54, no. 12, pp. 3186–3194, Dec 2007.
- [45] —, "A Compact SPICE Model for Carbon-Nanotube Field-Effect Transistors Including Nonidealities and Its Application-Part II: Full Device Model and Circuit Performance Benchmarking," *IEEE Transactions on Electron Devices*, vol. 54, no. 12, pp. 3195–3205, Dec 2007.
- [46] Y. Li, W. Kim, Y. Zhang, M. Rolandi, D. Wang, and H. Dai, *Journal of Physical Chemistry B Materials*, vol. 105, no. 46, pp. 11 424–11 431, 11 2001.
- [47] A. Lin, N. Patil, K. Ryu, A. Badmaev, L. G. D. Arco, C. Zhou, S. Mitra, and H. S. P. Wong, "Threshold voltage and on-off ratio tuning for multiple-tube carbon nanotube fets," *IEEE Transactions on Nanotechnology*, vol. 8, no. 1, pp. 4–9, Jan 2009.
- [48] Y. Ohno, S. Kishimoto, T. Mizutani, T. Okazaki, and H. Shinohara, "Chirality Assignment of Individual Single-walled Carbon Nanotubes in Carbon Nanotube Field-effect Transistors by Micro-photocurrent Spectroscopy," *Applied Physics Letters*, vol. 84, no. 8, pp. 1368–1370, 2004. [Online]. Available: <http://scitation.aip.org/content/aip/journal/apl/84/8/10.1063/1.1650554>

- [49] B. Wang, C. H. P. Poa, L. Wei, L.-J. Li, Y. Yang, and Y. Chen, "(n,m) Selectivity of Single-Walled Carbon Nanotubes by Different Carbon Precursors on Co-Mo Catalysts," *Journal of the American Chemical Society*, vol. 129, no. 29, pp. 9014–9019, 2007.
- [50] G. Gelao, R. Marani, R. Diana, and A. G. Perri, "A semiempirical spice model for n-type conventional cntfets," *IEEE Transactions on Nanotechnology*, vol. 10, no. 3, pp. 506–512, May 2011.
- [51] S. Fregonese, H. C. d'Honinchtun, J. Goguuet, C. Maneux, T. Zimmer, J. P. Bourgoïn, P. Dollfus, and S. Galdin-Retailleau, "Computationally efficient physics-based compact cntfet model for circuit design," *IEEE Transactions on Electron Devices*, vol. 55, no. 6, pp. 1317–1327, June 2008.
- [52] R. Marani, G. Gelao, and A. G. Perri, "Modelling of carbon nanotube field effect transistors oriented to spice software for a/d circuit design," *Microelectronics Journal*, vol. 44, no. 1, pp. 33 – 38, 2013. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0026269211001613>
- [53] S. University, *Stanford University CNFET model Website. Stanford University, Stanford, CA [Online]*, 2008, <http://nano.stanford.edu/model.php?id=23>.
- [54] M. M. Shulaker, G. Hills, N. Patil, H. Wei, H.-Y. Chen, H. S. P. Wong, and S. Mitra, "Carbon Nanotube Computer," *Nature*, vol. 501, no. 7468, pp. 526–530, Sep 2013, letter. [Online]. Available: <http://dx.doi.org/10.1038/nature12502>
- [55] A. Raychowdhury and K. Roy, "Carbon-Nanotube-based Voltage-mode Multiple-valued logic design," *IEEE Transactions on Nanotechnology*, vol. 4, no. 2, pp. 168–179, March 2005.
- [56] J. Liang, L. Chen, J. Han, and F. Lombardi, "Design and Evaluation of Multiple Valued Logic Gates using pseudo N-Type Carbon Nanotube FETs," *IEEE Transactions on Nanotechnology*, vol. 13, no. 4, pp. 695–708, 2014.

- [57] S. A. Ebrahimi, P. Keshavarzian, S. Sorouri, and M. Shahsavari, "Low Power CNTFET- Based Ternary Full Adder Cell for Nanoelectronics," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. 2, pp. 291–295, 2012.
- [58] K. Sridharan, S. Gurindagunta, and V. Pudi, "Efficient multiterinary digit adder design in CNTFET technology," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 283–287, 2013.
- [59] S. L. Murotiya and A. Gupta, "Design of High Speed Ternary Full Adder and Three-Input XOR Circuits Using CNTFETs," *2015 28th International Conference on VLSI Design*, pp. 292–297, 2015. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7031749>
- [60] B. Srinivasu and K. Sridharan, "Carbon nanotube FET-based low-delay and low-power multi-digit adder designs," *IET Circuits, Devices & Systems*, pp. 1–13, 2016. [Online]. Available: <http://digital-library.theiet.org/content/journals/10.1049/iet-cds.2016.0013>
- [61] —, "Low-complexity multiterinary digit multiplier design in cntfet technology," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 8, pp. 753–757, Aug 2016.
- [62] S. L. Hurst, "An extension of binary minimization techniques to ternary equations," *Computers Journal*, vol. 11, no. 3, pp. 277–286, 1968.
- [63] M. H. Moaiyeri, A. Doostaregan, and K. Navi, "Design of Energy-Efficient and Robust Ternary Circuits for Nanotechnology," *IET Circuits, Devices Systems*, vol. 5, no. 4, pp. 285–296, July 2011.
- [64] T. Sasao, "Multiple-valued decomposition of generalized boolean functions and the complexity of programmable logic arrays," *IEEE Transactions on Computers*, vol. C-30, no. 9, pp. 635–643, Sept 1981.

- [65] R. F. Mirzaee, T. Nikoubin, K. Navi, and O. Hashemipour, "Differential cascode voltage switch (dcvs) strategies by cntfet technology for standard ternary logic," *Microelectronics Journal*, vol. 44, no. 12, pp. 1238 – 1250, 2013. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0026269213001808>
- [66] F. Sharifi, M. H. Moaiyeri, K. Navi, and N. Bagherzadeh, "Robust and energy-efficient carbon nanotube fet-based mvl gates: A novel design approach," *Microelectronics Journal*, vol. 46, no. 12, Part A, pp. 1333 – 1342, 2015. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0026269215002323>
- [67] A. Weinberger and J. L. Smith, "A logic for high-speed addition," *National Bureau of Standards Circulation*, vol. Vol. 591, pp. 3–12, 1958.
- [68] D. Harris, "A taxonomy of parallel prefix networks," in *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, vol. 2, Nov 2003, pp. 2213–2217 Vol.2.
- [69] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, Aug 1973.
- [70] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980. [Online]. Available: <http://doi.acm.org/10.1145/322217.322232>
- [71] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, March 1982.
- [72] T. Han and D. A. Carlson, "Fast area-efficient vlsi adders," in *1987 IEEE 8th Symposium on Computer Arithmetic (ARITH)*, May 1987, pp. 49–56.
- [73] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509–516, June 1978.

-
- [74] C. Vudadha, P. P. Sai, V. Sreehari, and M. B. Srinivas, “CNFET Based Ternary Magnitude Comparator,” in *Communications and Information Technologies (ISCIT), 2012 International Symposium on*, Oct 2012, pp. 942–946.

Related Journal Publications

1. **Chetan Vudadha**, Srinivasan Rajagopalan, Aditya Dusi, Sai Phaneendra P and M B Srinivas, “Encoder-Based Optimization of CNFET-Based Ternary Logic Circuits,” in *IEEE Transactions on Nanotechnology*, vol. 17, no. 2, pp. 299-310, March 2018. DOI: 10.1109/TNANO.2018.2800015
2. **Chetan Vudadha**, Sai Phaneendra Parlapalli, M.B. Srinivas, “Energy efficient design of CNFET-based multi- digit ternary adders”, *Microelectronics Journal (Elsevier)*, Volume 75, pp. 75-86, May 2018. DOI:10.1016/j.mejo.2018.02.004
3. **Chetan Vudadha** and M.B. Srinivas “Design of High Speed and Power Efficient Ternary Prefix Adders using CNFETs,” in *IEEE Transactions on Nanotechnology*, vol. 17, no. 4, pp. 772-782, July 2018. DOI: 10.1109/TNANO.2018.2832649
4. **Chetan Vudadha**, Ajay Surya K, Saurabh Agrawal and M.B. Srinivas “Synthesis of Ternary Logic Circuits using 2:1 Multiplexers,” (accepted for publication in *IEEE Transactions on Circuits and Systems I: Regular Papers*) DOI: 10.1109/TCSI.2018.2838258

Related Conference Publications

1. C. K. Vudadha and M. Srinivas, "Design Methodologies for Ternary Logic Circuits," *2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL)*, Linz, 2018, pp. 192-197.
2. C. Vudadha, P. S. Phaneendra and M. B. Srinivas, "An Efficient Design Methodology for CNFET Based Ternary Logic Circuits," *2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*, Gwalior, 2016, vol., no., pp. 278-283, 19 -21 Dec 2016.
3. Vudadha, C.; Katragadda, S.; Phaneendra, P.S., "2:1 Multiplexer based design for ternary logic circuits," *IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, 2013, vol., no., pp.46-51, 19-21 Dec. 2013. (won "Bronze Leaf Certificate")
4. Vudadha, Chetan; Sai, Phaneendra P; Sreehari, V; Srinivas, M B, "CNFET based ternary magnitude comparator," *International Symposium on Communications and Information Technologies (ISCIT)*, 2012, vol., no., pp.942-946, 2-5 Oct. 2012.
5. Vudadha, Chetan; Sreehari, V.; Srinivas, M. B.; , "Multiplexer Based Design for Ternary Logic Circuits," , *2012 8th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, vol., no., pp.1-4, 12-15 June 2012.
6. Vudadha, C.; Phaneendra P, S.; Makkena, G.; Sreehari, V.; Muthukrishnan, N.M.; Srinivas, M.B.; , "Design of CNFET based ternary comparator using grouping logic," *2012 IEEE Faible Tension Faible Consommation (FTFC)*, vol., no., pp.1-4, 6-8 June 2012.

Other Publications

1. Parlapalli, Sai Phaneendra and Vudadha, Chetan and Srinivas, M. B., “An ESOP Based Cube Decomposition Technique for Reversible Circuits”, *Springer International Publishing (2017)*, 127--140.
2. Parlapalli, Sai Phaneendra and Vudadha, Chetan and Srinivas, M. B., “Optimizing the Reversible Circuits Using Complementary Control Line Transformation”, *Springer International Publishing (2017)*, 111--126.
3. Pal, Subhankar; Vudadha, Chetan; Phaneendra, P.Sai; Veeramachaneni, Sreehari; M.B. Srinivas, “A New Design of an N-Bit Reversible Arithmetic Logic Unit,” in *Fifth International Symposium on Electronic System Design (ISED), 2014* , vol., no., pp.224-225, 15-17 Dec. 2014.
4. Phaneendra, P.S.; Vudadha, C.; Sreehari, V.; Srinivas, M.B., “An Optimized Design of Reversible Quantum Comparator,” *27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, 2014* , vol., no., pp.557,562, 5-9 Jan. 2014.
5. Vudadha, Chetan; Phaneendra, P. Sai; Sreehari, V.; Ahmed, Syed Ershad; Muthukrishnan, N. Moorthy; Srinivas, M.B.; , “Design of Prefix-Based Optimal Reversible Comparator,” *IEEE Computer Society Annual Symposium on VLSI (ISVLSI),2012*, vol., no., pp.201-206, 19-21 Aug. 2012.
6. Vudadha, Chetan; Phaneendra, P. Sai; Sreehari, V.; Ahmed, Syed Ershad; Muthukrishnan, N. Moorthy; Srinivas, M.B.; , “Design and Analysis of Reversible Ripple,

- Prefix and Prefix-Ripple Hybrid Adders,” *IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2012*, vol., no., pp.225-230, 19-21 Aug. 2012.
7. Vudadha, C.; Makkena, G.; Nayudu, M.V.S.; Phaneendra, P.S.; Ahmed, S.E.; Veeramachaneni, S.; Muthukrishnan, N.M.; Srinivas, M.B.; , “Low-Power Self Reconfigurable Multiplexer Based Decoder for Adaptive Resolution Flash ADCs,” *2012 25th International Conference on VLSI Design (VLSID)*, vol., no., pp.280-285, 7-11 Jan. 2012.
 8. Kumar, V. Chetan; Phaneendra, P. Sai; Ahmed, Syed Ershad; Sreehari, V.; Muthukrishnan, N. Moorthy; Srinivas, M.B.; , “A Reconfigurable INC/DEC/2’s Complement/Priority Encoder Circuit with Improved Decision Block,” *International Symposium on Electronic System Design (ISED), 2011*, vol., no., pp.100-105, 19-21 Dec. 2011.
 9. Chetan Kumar, V; Sai Phaneendra, P; Ershad Ahmed, S; Sreehari, V; Moorthy Muthukrishnan, N; Srinivas, M.B.; , “Higher radix sparse-2 adders with improved grouping technique,” *TENCON 2011 - 2011 IEEE Region 10 Conference* , vol., no., pp.676-679, 21-24 Nov. 2011.
 10. Phaneendra, P.S.; Vudadha, C.; Ahmed, S.E.; Sreehari, V.; Muthukrishnan, N.M.; Srinivas, M.B.; “Increment/decrement/2’s complement/priority encoder circuit for varying operand lengths,” *11th International Symposium on Communications and Information Technologies (ISCIT), 2011* , vol., no., pp.472-477, 12-14 Oct. 2011.
 11. Vudadha, C.; Veeramachaneni, S.; Srinivas, M.B.; “Non-linear partitioning for decimal logarithm approximation,” *Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia), 2011*, vol., no., pp.102-105, 6-7 Oct. 2011.
 12. Chetan Kumar, V.; Sai Phaneendra, P.; Ershad Ahmed, S.; Veeramachaneni, S.; Moorthy Muthukrishnan, N.; Srinivas, M.B.; , “A Prefix Based Reconfigurable

-
- Adder,” *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2011, vol., no., pp.349-350, 4-6 July 2011.
13. Chetan Kumar, V.; Sai Phaneendra, P.; Ahmed, S.E.; Veeramachaneni, S.; Moorthy Muthukrishnan, N.; Srinivas, M.B.; , “A Unified Architecture for BCD and Binary Adder/Subtractor,” *14th Euromicro Conference on Digital System Design (DSD)*, 2011, vol., no., pp.426-429, Aug. 31 2011-Sept. 2 2011.

Biographies

Candidate Biography

Chetan Kumar V received his M. E. (Microelectronics) degree in 2011 from Birla Institute of Technology and Science (BITS)-Pilani, Hyderabad Campus, India, where he is currently working as a faculty in the Department of Electrical and Electronics Engineering. His current research interests include CNFET based Multi-Valued Logic Design, Computer Arithmetic and reversible logic circuits.

Supervisor Biography

Prof. M. B. Srinivas received his PhD from the Indian Institute of Science, Bangalore, India. He is currently a Professor and Dean, School of Engineering and Technology at BML Munjal University, Gurgaon, India. His research interests include nano-scale circuit design, VLSI arithmetic, data converters and ICT for healthcare. He has served as Chairman of IEEE Hyderabad Section during 2007 and 2008 and founding Chairman CAS/EDS Joint Chapter, Hyderabad Section, during 2012 and 2013. He is a recipient of Microsoft Research 'Digital Inclusion' award in 2006 and Stanford Medicine 'MedTech Innovation' award in 2017.