MINIMIZATION OF LINES IN REVERSIBLE CIRCUITS

JAYATI J LAW
Bachelor of Science, Rajasthan Technical University, 2012

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

© Jayati J Law, 2015

MINIMIZATION OF LINES IN REVERSIBLE CIRCUITS

JAYATI J LAW

Date of Defense: August 6, 2015

Dr. Jackie Rice Supervisor	Associate Professor	Ph.D.
Dr. Stephen Wismath Committee Member	Professor	Ph.D.
Dr. Robert Benkoczi Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

For my mother.

Abstract

Reversible computing has been theoretically shown to be an efficient approach over conventional computing due to the property of virtually zero power dissipation. A major concern in reversible circuits is the number of circuit lines or qubits which are a limited resource. In this thesis we explore the line reduction problem using a decision diagram based synthesis approach and introduce a line reduction algorithm— Minimization of lines using Ordered Kronecker Functional Decision Diagrams (MOKFDD). The algorithm uses a new sub-circuit for a positive Davio node structure in addition to the existing node structures. We also present a shared node ordering for OKFDDs. OKFDDs are a combination of OBDDs and OFDDs. The experimental results shows that the number of circuit lines and quantum cost can be reduced with our proposed approach.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my supervisor Dr. Jackie Rice for believing in me and guiding me throughout the study. I learned a lot from her professionally and personally. Her expertise, understanding and patience helped me completing my research work and writing this thesis. I would like to thank the members of my committee, Dr. Robert Benkoczi and Dr. Stephen Wismath for the discussions and continuous support they provided at all levels of the research.

I would also like to acknowledge Dr. Daya Gaur and Dr. Saurya Das for helping me understand few important concepts related to my study. Thanks goes to my friends and my lab mates, Ram Dahal, Fariha Naz, Zamilur and Asif for all the discussions and valuable suggestions. It would have been difficult to work without them.

Lastly, I thank my mother and my younger brother for encouraging me with their best wishes.

Contents

List of Tables

Chapter 1

Introduction

In today's world power minimization is one of the most concerning issues in electronics. Traditional computing systems use logically irreversible circuits. In irreversible systems once a final state of information is reached, the information cannot be traced back to its initial state. In a logically irreversible system whenever any two inputs have a single output, the two input states transform into one output state as shown in the example given in Figure 1.1. This results in loss of information bits. According to Frank [12] the irreversible information loss is explained in detail by Landauer's principle. According to Landauer's principle [18] every logical manipulation of information on an irreversible system results in the increase of entropy. Entropy defines the state of a system in terms of temperature and heat transfer. As given in [18] every time a bit is erased $KTln2$ amount of energy is released, where K is the Boltzmann constant and T is the room temperature (for T = 300 Kelvin this energy is about 2.9×10^{21} joules). During each operation on an irreversible system this energy is transferred to the environment. This large amount of energy is proportional to the number of transistors being used on a single integrated chip. Moore's Law says that the number of transistors on an integrated chip will double in approximately every two years [37]. With this increase in the number of transistors, irreversible circuits are becoming highly inefficient. In 1973, Charles Bennett of IBM Research showed that any irreversible computation can be carried out in a reversible manner to avoid energy dissipation. In reversible circuits no information bits are erased, thus there is potentially nearly zero energy dissipation depending on the underlying technology.

Recently reversible circuits have emerged to be useful components of quantum computing. Quantum computations work with unitary matrices [2] imposing a constraint of being

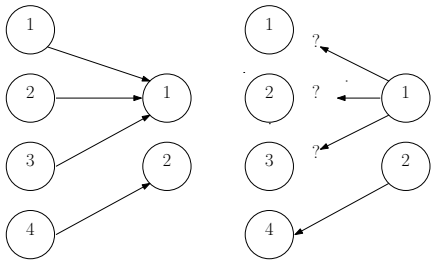


Figure 1.1: Irreversible state change

a reversible computation (discussed in 2.2). More information on quantum computers is given in [28]. Some of the applications of reversible circuits are in the field of digital signal processing, optical computing [38] and nano-computing technologies [23], communication, computer graphics, and cryptography [40].

1.1 Motivation

Every resource such as time, energy and distance traveled from a source to destination to perform a job in a working system always has a price depending on its usage and availability. Similarly, in reversible circuits qubits (or quantum bits) which are the information bits used in a quantum circuit are considered to be a very expensive resource. In classical circuits a bit has to be in one of two states, either 0 or 1. As will be described in Chapter 2 qubits may attain any superposition state i.e. 0, 1 or both at the same time. Each qubit is represented by a wire or line in a reversible circuit. The components of a reversible circuit are explained in detail in Chapter 2.

To illustrate the importance of line reduction consider an example where a full adder with three inputs and two outputs is to be realized by a reversible circuit. To implement the truth table from Figure 1.2 as a reversible circuit, extra qubits are required to make the truth table reversible. These extra qubits allow each of the input values of the truth table to be assigned unique output values. The process of converting an irreversible truth table to a reversible truth table is known as 'Embedding' [46]. An embedding process for the full adder is shown in Figure 1.3. In the worst case such a function would require four additional

lines to make a seven-input seven-output circuit. This means seven qubits are required. It becomes difficult and costly to build a reversible circuit if the process of embedding causes the number of qubits to increase drastically. The latest quantum computer built is a 128-qubit computer [52], therefore the necessity to reduce the qubits required in a single computation is vital.

cin	x	y	cout	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 1.2: Full adder

0	cin	x	y	cout	s	g1	g2
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	1	0	0	0
0	1	0	0	0	1	1	0
0	1	0	1	1	0	0	1
0	1	1	0	1	0	1	0
0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	0
1	0	1	0	0	0	1	1
1	0	1	1	0	1	1	1
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1

Figure 1.3: Embedding

1.2 Structure of the Thesis

In this thesis we detail our investigation into ways to minimize the number of qubits or circuit lines in a reversible circuit. The remainder of the thesis is structured as follows:

Chapter 2 provides the necessary background to give the basic understanding of reversible logic and circuits. It includes the definitions related to reversible logic and the components of a reversible circuit. It also gives an overview of reversible logic synthesis techniques such as ESOP (Exclusive-Or Sum of Products) -based synthesis.

The reversible circuit line reduction problem has been tackled with some heuristics in

the past. To understand the concept behind the heuristics, we explain related works in Chapter 3.

The Decision-Diagram (DD) -based synthesis approach is described in Chapter 4. This chapter illustrates the construction of decision diagrams as well as different DD-based synthesis algorithms such as BDD (Binary Decision Diagram) and KFDD (Kronecker Functional Decision Diagrams) -based synthesis methods. Here we deliver the primary concept of decision diagrams.

In chapter 5 we discuss the existing bounds on the line reduction problem given in [20]. We introduce our algorithm based on decision diagrams for line reduction with suitable examples. Our approach includes modifications in the KFDD synthesis algorithm. Our results appear in the proceedings of the 2015 IEEE Pacific Rim Conference [15].

Chapter 6 consists of the experimental results obtained by our algorithm. We compare our results with the existing approaches. We also compare our results with the lower bound on the line reduction problem.

The thesis concludes with Chapter 7 summarizing the contributions of this thesis in the field of reversible logic synthesis. The chapter also suggests some possibilities for future work.

Chapter 2

Background

The research on reversible computing began considering the thermodynamic limits of non-reversible computing. The inspiration behind the technology shift is already justified in chapter 1. To understand the significance of the reversible technology, we must comprehend the computational model involved. This chapter provides the necessary background to explore the circuit line minimization problem.

2.1 Basic Definitions

Definition 2.1. *A multi-output Boolean Function $f : A^m \rightarrow A^n$ is reversible if it is bijective.*

Let A be a finite set and $f : A^m \rightarrow A^n$ be a Boolean function which maps each input vector to a unique output vector (bijection); then this function is reversible. According to [39], a gate is reversible if the function it computes is bijective, and a circuit is reversible if it consists entirely of reversible gates. A cascade of reversible gates implements a reversible function with no fan-out (an output feeding more than one inputs) or feedback [25]. Figures 2.1 and 2.2 show an irreversible and a reversible function truth table.

xy	f(*xy*)
00	0
01	0
10	0
11	1

Figure 2.1: Irreversible function

xy	*x'y'*
00	00
01	01
10	11
11	10

Figure 2.2: Reversible function

Definition 2.2. *If a gate computes a (Boolean) function which is bijective then the gate*

*is **Reversible**.*

A necessary condition for a gate to be reversible is that it should have same number of input and output wires. A gate is a $k \times k$ gate if it has k wires [39]. Let $X := \{x_1, ..., x_n\}$ be the set of Boolean variables. Then, a reversible gate has the form $g(C,T)$, where $C = \{x_{i1}, ...x_{ik}\} \subset X$ is the set of control lines and, $T = \{x_{j1}, ..., x_{jl}\} \subset X$ with $C \cap T = \phi$ is the set of target lines [47] and $k + l = n$. The following definitions illustrate examples of reversible gates.

Definition 3. *A **k-CNOT Gate** is a $(k+1) \times (k+1)$ gate which leaves the k inputs unchanged and inverts the $k+1^{th}$ input if all the k inputs are 1.*

A $0-CNOT$ gate is a simple NOT gate which inverts the input without any controls $(x) \rightarrow (x \oplus 1)$ while a $1-CNOT$ gate is a **Controlled NOT Gate** which performs $(x,y) \rightarrow (x \oplus y)$ where $\oplus$ is the XOR operation. Figures 2.3 and 2.4 illustrate NOT and CNOT gates.

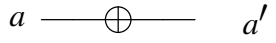


Figure 2.3: NOT Gate

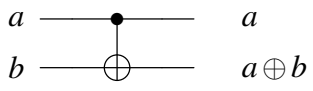


Figure 2.4: CNOT Gate

Definition 2.4. ***A Multiple Control Toffoli Gate*** (MCT) *with target line x_j and control lines $x_{i1}, x_{i2}...x_{ik}$ maps $(x_1x_2...x_j...x_n)$ to $(x_1x_2...(x_{i1}x_{i2}...x_{i1}) \oplus x_j...x_n)$. All control lines must be 1 in order for the target qubit to be inverted. A MCT gate with no control is a **NOT gate**. A MCT gate with one control gate is a **controlled-NOT gate**. A MCT with two control lines is a Toffoli [34] gate.*

A MCT gate is identical to the k-CNOT gate by definition given that the value of $k > 1$. It is important to note that the MCT can use k-CNOT notation and vice-versa. Generally, a k-CNOT gate is expressed in terms of Toffoli gates. A k-CNOT gate with $k = 0$ is said to be a TOF0 gate or a Toffoli gate with 0 controls (NOT gate), a TOF1 for $k = 1$ and so on. An example of a MCT is given in Figure 2.5:

Definition 2.5. *A **Multiple Control Fredkin Gate*** (MCF) *with target lines x_p and x_q and control lines $x_{i1}, x_{i2}, ...x_{ik}$, maps $(x_1x_2...x_p...x_q...x_n)$ to $(x_1x_2...x_q..x_p...x_n)$ if all the*

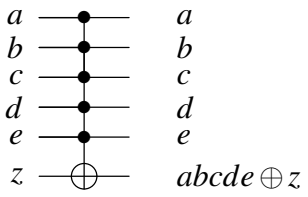


Figure 2.5: MCT Gate

control lines have value 1. Therefore, it is also called a ***Swap gate*** *[34]. A MCF gate is shown in Figure 2.6.*

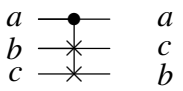


Figure 2.6: MCF Gate

Definition 2.6. *A* ***Dual*** *is a gate which reverses the logic function. A gate is* ***Self-reversible*** *if the dual is identical to the gate itself [29].*

Every gate has a dual which transforms the output vectors to input vectors. For example the dual of the NOT gate is the NOT gate: $x'(NOT) = x$.

The Hamming weight is the number of logical 1s in the set of values.

Definition 2.7. *A gate is a* ***Conservative*** *gate if the Hamming weight of the set of input values is similar to the set of output values. Similarly, a* ***Non-Conservative*** *gate has unequal Hamming weights for its input and output values [29].*

Definition 2.8. *A gate is* ***Universal*** *if it can implement other basic reversible logic gates independently.*

The NAND gate is a universal gate in Boolean logic. Similarly, the Toffoli gate is also an example of a universal gate in reversible logic [29].

Definition 2.9. ***Lines*** *or wires in a reversible circuit represent the variables of a reversible truth table.*

Definition 2.10. ***Garbage outputs*** *are additional outputs which do not produce any desired functionality.*

In [26] it is shown that at least $g = \lceil log_2(\mu) \rceil$ garbage outputs are required for converting an irreversible function to a reversible function, where μ is the maximum number of times a single output pattern is repeated in an irreversible truth table. Converting an irreversible function with n inputs and m outputs into a reversible function will require $m+g$ qubits. Since $m+g > n$, c number of additional lines with a constant input for each line are added to make a function reversible. Thus it becomes $n+c = m+g$. In Figure 2.7 the garbage outputs are labeled as 'g' and the additional line is labeled as '0'.

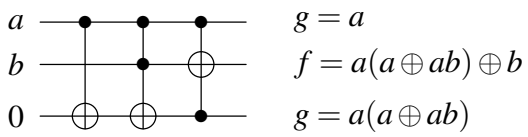


Figure 2.7: Reversible circuit with garbage outputs.

Definition 11. A ***Reversible Circuit*** *comprises a cascade of reversible gates on circuit lines implementing a reversible function. It may contain garbage outputs.*

2.2 Quantum Concepts

2.2.1 Quantum States

Qubits exhibit the property of linear superposition of basis states (0,1) as described in Chapter 1. The state of a qubit $|\psi>$ is defined as [5]

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

Here we use the Dirac notations of basis state vectors $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ whereas the α and β are complex numbers satisfying the condition $|\alpha|^2 + |\beta|^2 = 1$. In the vector form the state of a single qubit is shown by the vector $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$. Moreover,

the principles of quantum mechanics [28] declare that two quantum states are similar if they differ by a phase factor of $e^{i\theta}$, $\theta \in \mathbb{R}$. Thus, the quantum state ψ is also written as $|\psi\rangle = cos(\theta/2)|0\rangle + e^{i\phi}sin(\theta/2)|1\rangle$ where $0 \leq \phi < 2\pi$, $0 \leq \theta < \pi$. θ and ϕ are the angle coordinates indicating a qubit state in the Bloch sphere [4]. The Bloch sphere, as shown in Figure 2.8 provides a geometrical representation of a single-qubit state. The north pole (+Z) represents the state $|0\rangle$ while the south pole (-Z) represents the state $|1\rangle$. The states on the equator are the superpositions of the states $|0\rangle$ and $|1\rangle$ with equal weights $\theta = \pi/2$ and different phases [14]. Further information on these concepts can be found in [28].

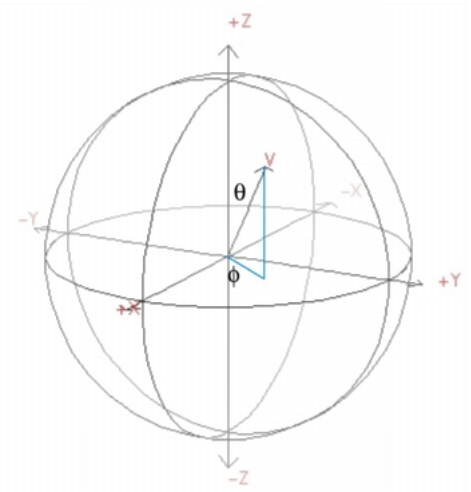


Figure 2.8: Bloch Sphere representation

2.2.2 Quantum Gates

Quantum gates are small circuits operating on qubits. They are reversible by nature and are represented by unitary matrices. A matrix U is a unitary matrix provided that $UU^{\dagger} = I$ where I is the identity matrix. The 2×2 identity matrix is given as:

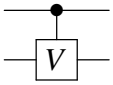
$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The unitary operator $U^{-1} = U^{\dagger}$ ensures the reversible characteristic of a matrix.

In reversible logic there exists quantum gate libraries such as NCT (NOT, CNOT, Toffoli) , GT (Generalized Toffoli) and NCV (NOT, CNOT, V gates) to design quantum circuits. The use of these libraries affect the cost metrics (explained in 2.3) associated with the circuit design. The NCT library is the most common library used to design quantum circuits. In this thesis we will examine synthesis methods utilizing the NCT library for circuit design. However, the decomposition of the reversible gates to the NCV library provides the exact computation for quantum cost (explained in Section 2.3). The NCV library consists of NOT, CNOT, V and $V^{\dagger}$ operators. The NCV operators are defined by the following quantum gates [28]:

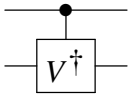
1. **NOT gate:** The NOT gate simply inverts the t qubit. The gate is denoted by $T(\emptyset, t)$, where the control is $\emptyset$ and target is t. The unitary matrix for the NOT gate is $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

2. **CNOT gate:** The Controlled NOT, $T(c,t)$, inverts the qubit at target t only if the control c is 1. The unitary matrix for CNOT is $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

3. **V gate:** The controlled V gate, $V(c,t)$, performs the V operation on the target t qubit when the control c is 1. The V operation is equivalent to square root of NOT. Two consecutive V operations result in a NOT operation. This qubit operator causes the half spin of a qubit, as shown in Figure 2.9(b). Thus, two half spins (two V gates) of a qubit concludes in the inversion of a qubit state. The V gate is depicted as 

.

The matrix for the V gate is $\begin{pmatrix} \frac{1+i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}$.

4. **$V^\dagger$ gate:** The controlled $V^\dagger$ gate, $V^\dagger(c,t)$, performs the $V^\dagger$ operation on the target qubit t if the control qubit c is 1. The $V^\dagger$ operation is similar to the inverse of V operation i.e. $V^\dagger = V^{-1}$. Thus, V gate and $V^\dagger$ gate if applied together form an identity gate. This operator also results in the half spin of a qubit but in an opposite direction to the V operator. Similar to the V operation two $V^\dagger$ operations in series produce a NOT operation. The $V^\dagger$ gate is depicted as 

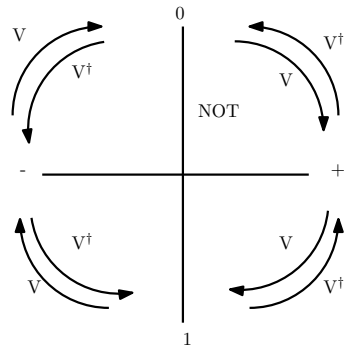
. The unitary matrix for the $V^\dagger$ gate is $\begin{pmatrix} \frac{1-i}{2} & \frac{1+i}{2} \\ \frac{1+i}{2} & \frac{1-i}{2} \end{pmatrix}$.

   The V and $V^\dagger$ operations are summarized in Figure 2.9(a). Figure 2.9(b) explains the spin of a qubit induced by each V or $V^\dagger$ gate. The ends of the vertical pole are labeled as 0 and 1 for $|0\rangle$ state and $|1\rangle$ state of a qubit respectively. Similarly, the ends of the horizontal pole are labeled as $+$ and $-$ for any respective positive and negative imaginary state of a qubit. The imaginary state of a qubit can be defined by any complex number.

   These basic NCV quantum gates can be used to build other quantum gates such as the Toffoli gate. An example to show the quantum cost calculation of a Toffoli is to decompose a Toffoli gate into NCV quantum gates. Since there are 5 quantum gates in the decomposed Toffoli gate we say that the quantum cost of a TOF3 gate is 5.

$$V = \sqrt{NOT}$$
$$V^\dagger = \sqrt{NOT^\dagger}$$
$$V^2 = V^{\dagger 2} = NOT$$
$$V \cdot V^\dagger = I$$

(a)



(b)

Figure 2.9: NOT, CNOT, V and V† operations

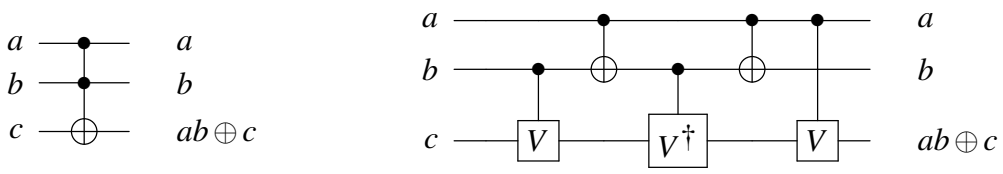


Figure 2.10: Toffoli gate decomposition

2.3 Cost Metrics

Cost metrics allow a circuit designer to compute a measurement of a reversible circuit in terms of some cost. The cost computation of every component of a reversible circuit is calculated according to the standard metrics. The basic parameters for the measurement are as follows:

1. **Quantum Cost:** Quantum cost evaluates the cost of gates in a reversible circuit. For every reversible gate there is an associated cost built on the number of underlying quantum gates. As an illustration, the cost of a Toffoli gate is 5 based on the construction shown in Figure 2.10. Table 2.1 shows the quantum cost of Toffoli gates of size n, where $n \in \mathbb{Z}^+$ [19]. Likewise, the cost of a Fredkin gate of size n is the sum of the cost of a n-bit Toffoli gate and integer value 2 (for 2 CNOT gates). Therefore, for any two identical circuits with similar line count, the circuit with a lesser quantum cost is considered to be more economical compared to the one with a higher quantum

cost. There has been a lot of study [11] [47] [32] [51] [21] on reducing quantum cost.

Table 2.1: Quantum cost table

Size (n)	Garbage	Name	Quantum Cost
1	0	NOT, t1	1
2	0	CNOT, t2	1
3	0	Toffoli, t3	5
4	0	Toffoli4, t4	13
5	0	t5	29
5	2	t5	26
6	0	t6	61
6	1	t6	52
6	3	t6	38
7	0	t7	125
7	1	t7	80
7	4	t7	50
8	0	t8	253
8	1	t8	100
8	5	t8	62
9	0	t9	509
9	1	t9	128
9	6	t9	74
10	0	t10	1021
10	1	t10	152
10	7	t10	86
n >10	0	tn	2n - 3
n >10	1	tn	24n - 88
n >10	n-3	tn	12n - 34

2. **Gate Count:** This parameter refers to the number of gates required for implementing a function. A change in the gate count may increase or decrease the quantum cost subject to the size of gates being used. Thus, the gate count is not directly proportional to the quantum cost. As an illustration in Figure 2.11 the function ($acd \oplus abc$) is implemented in two ways, first with a quantum cost of 23 with gate count of 3 and secondly with a quantum cost of 26 with gate count of 2 respectively.

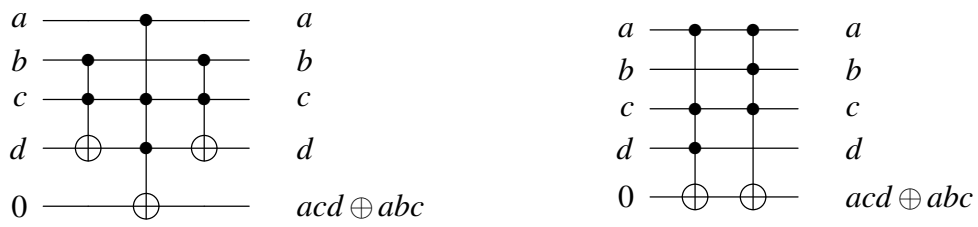


Figure 2.11: Toffoli gates for the Boolean function ($acd \oplus abc$)

3. **Line Count:** The number of lines in a reversible circuit is generally equal to the number of variables in the truth table. In other words lines represent qubits. Qubits are particles that demand a controlled system to keep them in a stable initial state and change states. Thus, they are expensive to sustain. As a result, minimizing circuit lines is often considered preferable compared to reducing quantum cost [49]. There is usually a trade-off between reducing quantum cost and circuit lines [50]. The problem of reducing circuit lines in reversible circuits is discussed in works such as [48] [20] [9] [49].

2.4 Reversible Logic Synthesis Techniques

Conventional logic synthesis approaches use the classical universal gate library of a Boolean function— AND, NOT and OR gates. Reversible logic synthesis, unlike classical logic synthesis techniques, implements a Boolean function using quantum gate libraries (discussed in 2.2) with no fan-out. Since no fan-out is permitted the output of each gate in a reversible circuit is used only once. Reversible logic synthesis produces reversible circuits containing a sequence of gates with no loops [31].

In the literature there are various methods for reversible logic synthesis which are broadly divided into the following categories.

2.4.1 ESOP-based Synthesis

An Exclusive-or Sum Of Products (ESOP) is a variant of the basic Sum Of Products (SOP) representation of a Boolean specification. In an ESOP the product of the literals

i.e. OR of the AND terms are Ex-ORed. For example: $(ab \oplus cd)$. A SOP formulation of a Boolean function is expressed as the sum of the product of the literals. For example: $(a \wedge b \vee c \wedge d)$, also written as $(ab + cd)$. The first step of ESOP based synthesis is to obtain an ESOP expression. A given SOP expression $(a + b)$ is formulated as $(a \oplus b \oplus ab)$ in ESOP. An important reason for moving from SOP to ESOP formulation is the efficiency involved. The worst case complexity of SOP formulation considering the size of the truth table is $O(2^{n-1})$, for a n-variable Boolean function, which is greater than the complexity of ESOP $(O(3.2^{n-3}))$ [36].

The product of the literals in an ESOP or SOP expression are represented by cubes. These cubes combine to form a cube-list. A major difference between the cubes of ESOP and SOP formulations is that the former includes don't cares resulting in a smaller set of input values to scan for building a circuit. For both ESOP and SOP representations a lesser number of cubes result in a smaller circuit. Minimizing the SOP expression is a well known DNF (Disjunctive Normal Form) minimization problem [1]. The most commonly used algorithm for ESOP minimization is EXORCISM-4 [27]. The main idea behind both SOP and ESOP minimization is to work with the Karnaugh Map. A Karnaugh Map or K-Map is a method to simplify Boolean expressions. The truth table is transfered onto a two dimensional grid where each cell represents a combination of input conditions while the value of each cell is the corresponding output value for the input conditions. For an SOP formulation the 1 bits are covered by a minimum number of cubes (covers). These cubes are ORed to get the complete SOP expression. However, for an ESOP formulation all the 0 bits are covered by an odd number of cubes and 1 bits are covered by an even number of cubes [8]. The ESOP cubes are then Ex-ORed to get an ESOP expression. Figure 2.12 (a) and (b) show minimized covers with bold lines. Each rectangle identifies a cube for their respective formulations. As illustrated the SOP cubes are covered by 3 covers while in ESOP cubes each 1 bit is covered by a single cover (odd) and 0 bits are covered by two covers (even) . The initial Boolean expression in Figure 2.12 (a) covered by dashed squares

in the K-map is $f = a\bar{c}d + cd + ac\bar{d}$ the minimized expression shown by bold squares is $f' = ad + ac + cd$. Similarly, in Figure 2.12 (b) the ESOP expression by dashed squares is $f = \bar{a}cd \oplus ab\bar{c} \oplus abc\bar{d} \oplus a\bar{b}cd$ and the minimized expression is $f' = ab \oplus cd$. The work on efficient and exact ESOP minimization is discussed in [30] [35].

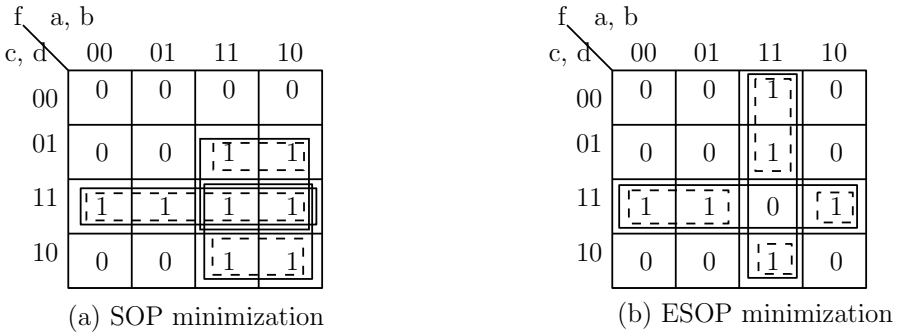


Figure 2.12: (a)SOP minimization and (b) ESOP minimization for the given Boolean functions.

The ESOP synthesis algorithm [10] starts with a cube-list, as shown in Figure 2.13. A reversible circuit is formed by starting with an ESOP cubelist representation of a Boolean function. The cubelist consists of inputs and outputs similar to a truth table. Initially, an empty circuit i.e with $2n+m$ qubits and 0 gates is created. Here n is the number of inputs. $2n$ covers both positive and negative polarity of the input qubits. m denotes the number of outputs. Each cube in the cube-list is then mapped on to the circuit. The circuit formation from the cube-list is performed according to these steps:

1. Create an empty circuit of size $2n+m$.

2. Insert input qubits $(x_0, x_1, ..., x_{2n})$ in the circuit, where $n \in \mathbb{Z}^+$, for every complemented and uncomplemented literal in an ESOP formulation such as a and a'. The outputs of these corresponding inputs are labeled as 'g' or garbage value.

3. Insert constant qubits $(k_0, k_1, k_2, ..., k_m)$ in the circuit, where $m \in \mathbb{Z}^+$. These qubits are initialized to a value of 0 or 1 and remain constant throughout. The outputs for 'm' inputs are labeled as $f1, f2, f3...., fm$ respectively.

4. Scan the cube-list with the inputs $(x_0, x_1, ..., x_i)$ and outputs $(x_0, x_1, ...x_j)$, where i, j are positive integers. Add a Toffoli gate to the circuit for each cube in the cubelist with $x_i = 1$ as a positive control and $x_i = 0$ as a negative control.

An example of a circuit resulting from this process is given in Figure 2.14.

```
.i    3
.o    2
.type    esop
0-0    01
11-    10
-00    00
1-1    11
.e
```

Figure 2.13: Full adder ESOP cube-list

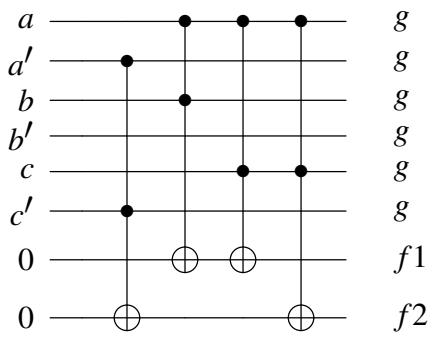


Figure 2.14: Reversible circuit

2.4.2 Transformation-based Synthesis

Transformation-based synthesis takes an n variable reversible function specification as input and produces an $n \times n$ reversible circuit. The approach [24] is more convenient than the other approaches such as proposed in [17] but requires a reversible function as an input. There are two algorithms for transformation based synthesis, the basic algorithm and the bidirectional algorithm. We will only discuss the basic algorithm here. The algorithms derive Toffoli gates by manipulating the input or output bits of a given truth table. The transformation techniques avoid an extensive search of the best collection of gates for the near-optimal reversible circuit.

The basic algorithm is a simple greedy approach which generates Toffoli gates by exploiting the output side of the specification. According to [24] a reversible function can be represented by an ordered set of integers such as $\{4,1,0,7,6,3,5,2\}$. Thus, the function over these integers is defined as $f(0) = 4$, $f(1) = 1$ and so on. Initially, consider a reversible function as the mapping over $\{0,1,...,2^n - 1\}$ bits. The algorithm iterates over the following steps:

1. Check the integer function $f(0)$. If $f(0) \neq 0$, invert the 1-bits of the corresponding to the $f(0)$ output bits. For each inversion a single Toffoli gate of size one ($TOF1$) or a NOT gate is required. Identify the transformed function as $f^{+}(0) = 0$.

2. For every integer $i = \{0, 1, ..., 2^n - 1\}$, $f^{+}(i) \neq i$, a transformation to a new specification $f^{++}(i)$ is required and the Toffoli gates map the transformation $f^{+}(i) \rightarrow i$. Otherwise, if $f^{+}(i) = i$.

Figure 2.15 illustrates the manipulation of the bits during each transformation of the specifications in bold. Firstly, a NOT gate is applied to the bit a^0 of the specification (i) function $f(0)$ and then, the corresponding gate $TOF1(a^0)$ is added to the circuit. In the next step, we map $f^{+}(5) \rightarrow 5$ with the application of $TOF3(\{c^1, b^1\}, a^1)$ on (ii) and $TOF3(\{a^3, c^3\}, b^3)$ on (iii). Lastly, to map $f^{+}(6) \rightarrow 6$ we use $TOF3(\{c^4, b^4\}, a^4)$ on (iv). The mapping of these gates to the circuit shown in the steps above in the process of transformation is in reverse order. The resultant mapping of the transformation to a reversible Toffoli gate cascade is shown in Figure 2.16

The algorithm generates the circuit with at most $(m-1)2^m + 1$ gates for a m variable specification with the complexity of $O(n2^n)$ [24]. Sometimes the final specification does not map the outputs with their correct inputs. In that case an output permutation [24] is applied to the specification. In order to reduce the circuit width template matching [24] is performed on the resultant circuit. The templates are proposed in the paper [24].

2.4.3 Search-based Synthesis

Search-based synthesis methods traverse through a search tree built on the factors of the Boolean expression. The search tree is explored for the best set of factors in a path consisting of smallest expressions to build a circuit of minimal quantum cost. Positive Polarity Reed-Muller (PPRM) expansion is the most commonly used method to generate a search tree for a reversible function. The PPRM expansion is obtained only from uncomplemented variables available in ESOP form. The expansion has a canonical form of

	(i)	(ii)	(iii)	(iv)	(v)
cba	$c^0b^0a^0$	$c^1b^1a^1$	$c^2b^2a^2$	$c^3b^3a^3$	$c^4b^4a^4$
000	00**1**	000	000	000	000
001	00**0**	001	001	001	001
010	01**1**	010	010	010	010
011	01**0**	011	011	011	011
100	10**1**	100	100	100	100
101	11**1**	11**0**	1**1**1	101	101
110	10**0**	101	1**0**1	11**1**	110
111	11**0**	11**1**	110	11**0**	111

Figure 2.15: An example of manipulating the truth table in transformation-based synthesis.

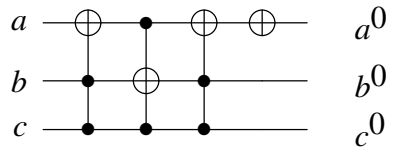


Figure 2.16: Reversible circuit for transformation-based synthesis.

$f(x_1,x_2,x_3,....,x_n) = a_0 \oplus a_1x_1 \oplus a_nx_n \oplus a_{12}x_1x_2 \oplus a_{13}x_1x_3.... \oplus a_{n-1,n}x_{n-1}x_n..... \oplus a_{1,2,3,...n}x_1x_2...x_n$ [13]

Here $a \in \{0,1\}$, constant value and x_i are all uncomplemented variables. Figure 2.18 shows the PPRM expansion of the truth table in Figure 2.17.

c	b	a	c_0	b_0	a_0
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Figure 2.17: Reversible Function

$$a_0 = a \oplus 1$$
$$b_0 = b \oplus c \oplus ac$$
$$c_0 = b \oplus ab \oplus ac$$

Figure 2.18: PPRM Expansion

Here we describe a search-based algorithm from [13] which uses PPRM expansion to derive a reversible circuit from a given Boolean function. The algorithm begins with the PPRM expansion of the function $f(v_1,v_2,v_3,....,v_n)$. Figure 2.19 shows the expansion of

an example function. The algorithm works as follows:

1. Create and initialize a root node (Node 0) of the search tree with the PPRM expansion as shown in Figure 2.19(a).

2. Push Node 0 into a priority queue for further exploration. The root node is set to the current best solution.

3. For every output variable in the PPRM expansion of the node being explored, consider all the factors $v_{out,i}$ that do not contain v_i. For example for $a_{out} = a \oplus 1 \oplus bc \oplus ac$ the factors are 1 and bc, as they do not contain the literal a.

4. Substitute $v_i = v_i \oplus factor$ to create child nodes for the further exploration. These factors label the edges of the search tree. Each substitution should reduce the number of terms of the synthesized child node.

5. The child node is explored further if the terms in the child node are less than the parent node. Insert the child node into the priority queue and update the best solution as the child node.

6. The algorithm iterates over these steps until no more nodes in the priority queue are left to explore and the best solution is found.

7. The algorithm returns the path which consists of best factors for a given function. The best factors are recognized if the terms are decreased after the substitution. The path from the root node to the best solution creates the desired reversible circuit. The path guarantees to construct a circuit with minimal number of gates. Each factor on the edge of the path in the tree relates to a Toffoli gate in the circuit.

2.4.4 BDD based Synthesis

Most of the logic synthesis techniques discussed above in this chapter lack efficiency to deal with the large number of variables in a Boolean function. Binary Decision Diagrams

(BDD) and PPRM based synthesis approaches are more efficient in terms of run-time, since they use more compact data structures (tree structure). On the other hand all the other approaches depend on a truth table for the synthesis process.

In BDD a Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ is represented by a directed acyclic graph $G = (V, E)$ where the Shannon decomposition [45] :

$$f = \bar{x}_i f_{x_i=0} + x_i f_{x_i=1} \qquad (1 \leq i \leq n) \text{ for any integer n,}$$

is carried out on each node $v \in V$ labeled by x_i of a BDD. Here x_i is the variable of a Boolean function and, $(f_{x_i=0})$ is the function f when $x_i = 0$ and $(f_{x_i=1})$ is the function f when $x_i = 1$. The node v has two types of outgoing edges $\{0\text{-}edge, 1\text{-}edge\} \in E$. The 0-*edge* = $low(f)$ where $low(f)$ is $(f_{x_i=0})$ and 1-*edge* = $high(f)$ where $high(f)$ is $(f_{x_i=1})$ in a BDD. The $low(f)$ and $high(f)$ can be any internal node marked as a sub-function or a terminal node. The terminal nodes of a BDD have values either 0 or 1. Figure 2.20(a) illustrates a BDD representing the function $x_1 \oplus x_2$.

A network of Toffoli gates is created by adding reversible gates for each node v in a BDD. The reversible gates for each node depend on the type of the node. For a general case, a node has a cascade of Toffoli gates such as shown in Figure 2.20(b) and (c). Other types of nodes are specified in Table 2.2 [45].

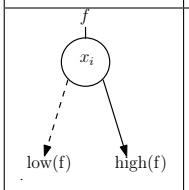
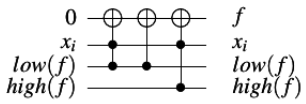
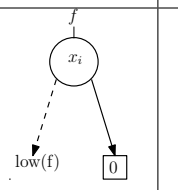
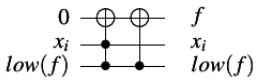
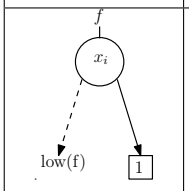
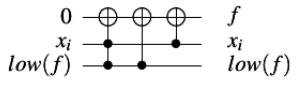
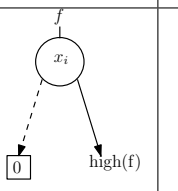
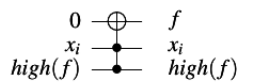
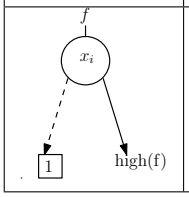
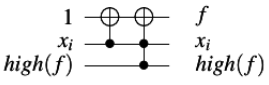
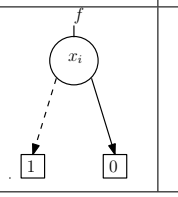
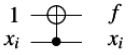
The size of a BDD is defined by the number of nodes a BDD consists of. Shared nodes are an important component to significantly reduce the size of a BDD. Any node v with more than one predecessor is identified as a shared node. Figure 2.21 displays an example of a BDD with a shared node and the corresponding reversible circuit. Complementary edges are also considered as a technique to decrease the count of nodes. With complementary edges the function and its negation can be represented by the same node [7].

The algorithm to generate a reversible circuit using BDDs is as follows:

1. Generate a BDD for the Boolean function f to be synthesized.

2. Scan every node in the BDD. If the node v is the identity of the input variable x_i

Table 2.2: Toffoli gates for BDD node types with additional constant line [45].

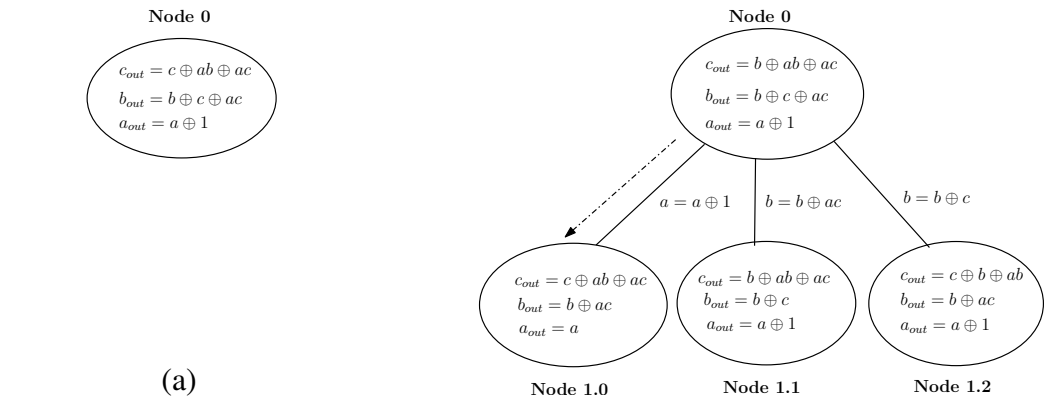
BDD	Toffoli gates	BDD	Toffoli gates
			
			
			

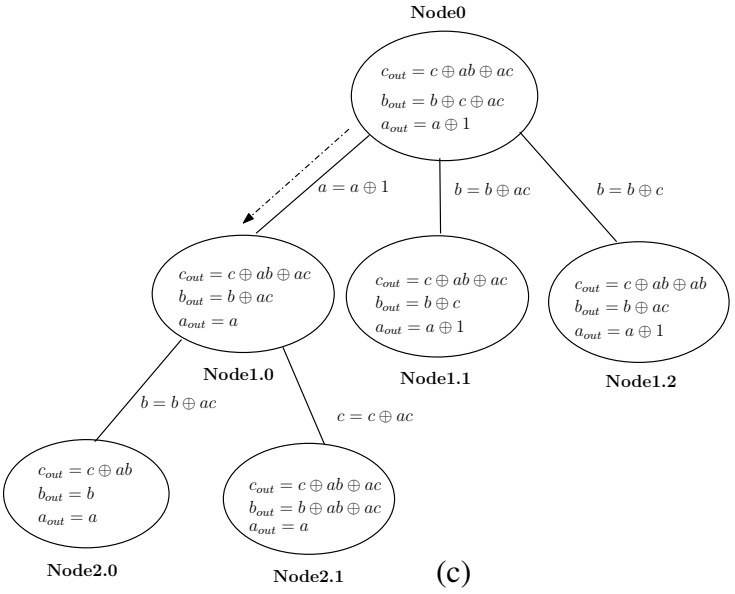
($low(f) = 0$ or 1 and $high(f) = 0$ or 1), then no constant circuit line is added. The node is represented by the input circuit line.

3. Otherwise the Toffoli gates shown in Table 2.2 for each node type are added to the circuit.

4. The successors $low(f)$ and $high(f)$ of the node v are preserved using an additional constant line for each if the successors are shared nodes or identity of the input variable. In this case templates shown in Table 2.2 are used. If none of the above cases apply on the nodes, then the template shown in Figure 2.20(c) is used.

The size of a reversible circuit depends on the size of the corresponding BDD. Considering a BDD of size k (K nodes) for a Boolean function f of n variables, a reversible circuit with at most $k+n$ circuit lines is generated. The resultant circuit consists of the maximum of $3 \cdot k$ gates since at most 3 gates are added to the circuit for each node in a BDD. In the worst case scenario a BDD can have 2^n nodes for a single output function.





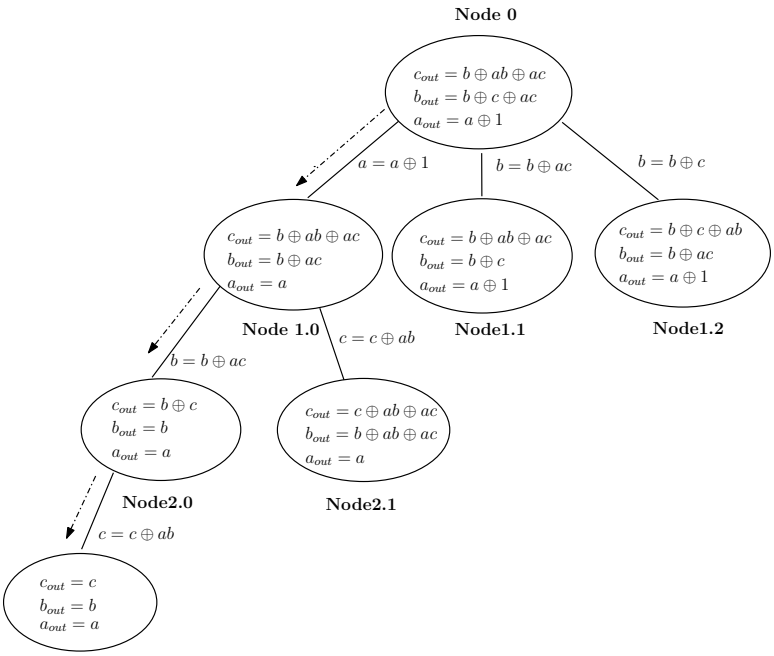
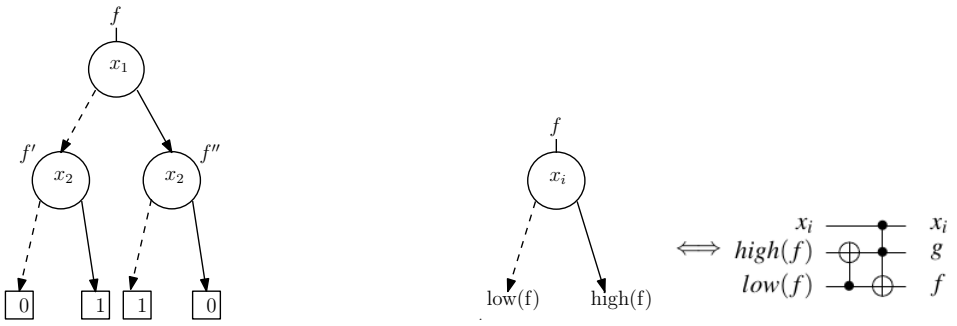


Figure 2.19: Search tree using PPRM expansion

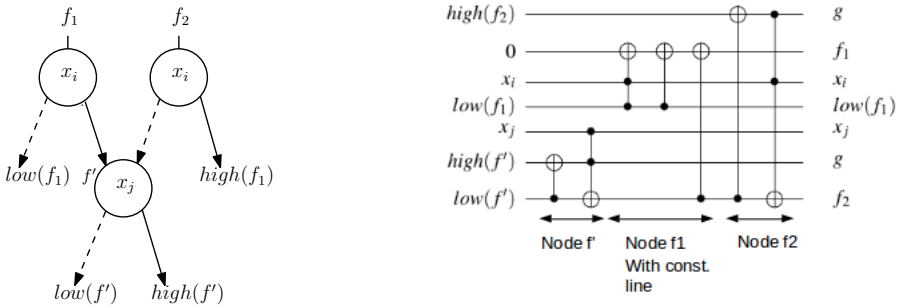


(a) A BDD for a function $x_1 \oplus x_2$

(b) A BDD node

(c) Equivalent circuit

Figure 2.20: A BDD and Toffoli gates for a node [45].



(a) BDD with shared node

(b) Reversible circuit

Figure 2.21: A BDD with shared node and the equivalent reversible circuit [45].

Chapter 3

Decision Diagrams

There are several means of evaluating an expression to obtain a value of true or false (1 or 0). One popular way is using decision diagrams to test for the value of a Boolean expression. As discussed in Chapter 2 Binary Decision Diagrams are one way to represent a function $f \in \mathbb{B}_n$, where $\mathbb{B}_n$ states the set of all n variable Boolean functions. Each variable $x_1, x_2, ..., x_i$ assigned to a node is tested for an input value defining a path from the root to the leaf node of the tree. The path may consist of a 0-edge or a 1-edge. Leaf nodes give the output value for the sequence of input values assigned to each node variable. We now proceed to discuss different categories of binary decision diagrams.

3.1 Reduced Ordered BDD (ROBDD)

Before understanding the functionalities of a ROBDD it is important to define an OBDD. The authors of [22] define an OBDD as follows:

Definition 3.1. Considering the order of the variables $\rho = (x_1, x_2, ..., x_n)$ an Ordered Binary Decision Diagram is a directed acyclic graph with respect to the order ρ. An OBDD satisfies the following properties:

1. There is exactly one root and two nodes labeled by the constants 0 and 1. These two nodes do not have any outgoing edges and are called sinks.

2. Each internal node is marked by a variable x_i and has two outgoing edges namely its s1-edge and 0-edge. These edges are labeled by 1 and 0 respectively.

3. The sequence in which the variables occur in a path from the root to the sink is the same as the order of the variables defined by ρ. This means if there exists a path from

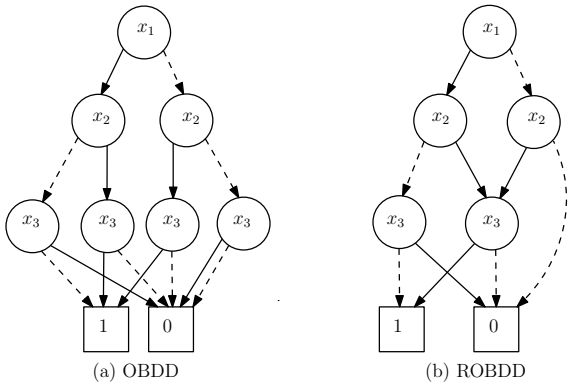


Figure 3.1: OBDD and ROBDD of $f = x_2x_3 + x_1\overline{x_2}\,\overline{x_3}$

a node x_i to x_j then $x_i < x_j$ in ρ.

The variable ordering in a BDD plays an important role to reduce the size of a BDD. Finding the best variable ordering is a NP-Hard problem [6]. However, the package CUDD [43] has a feature for implementing variable ordering using a sifting algorithm [33].

A matter of concern with OBDDs is the occurrence of redundancies of the following types:

1. The 0-edge and 1-edge of a node v lead to the same successor which means no new information is produced at node v.

2. The same information is represented by the OBDD in the form of similar subgraphs.

Therefore, to resolve these issues we define ROBDDs with the reduction rules.

Definition 3.2. An OBDD is a ROBDD if the following cases exist [22]

1. There must not exist any node v with $high(v) = low(v)$.

2. No two nodes u and v should exist such that the similar subgraphs are rooted at nodes u and v.

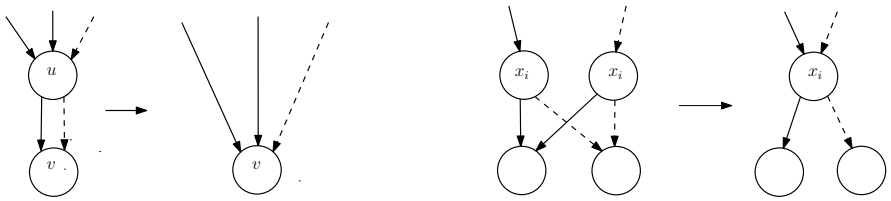
Figure 3.1 displays an OBDD and a ROBDD of the function $f = x_2x_3 + x_1\overline{x_2}\,\overline{x_3}$ with the variable order of $x_1 < x_2 < x_3$. The 0-edge and 1-edge is displayed by a dashed edge and

regular edge respectively. This definition leads to the reduction rules for OBDDs to form ROBDDs. There are two reduction rules as stated below [22]:

1. **Elimination rule:** For a node v if the 0-edge and 1-edge have the same successor node u , then eliminate v and redirect all the incoming edges of node v to node u.

2. **Merging Rule:** If two internal nodes u and v with the same variable name have their 1-edge directed to the same node and 0-edge directed to the same node, then eliminate either u or v and redirect all the incoming edges of the removed node to the remaining node.

Figure 3.2 shows the two reduction rules.



(a) Elimination rule (b) Merge rule

Figure 3.2: Reduction rules for OBDDs [22].

3.2 Ordered Functional Decision Diagram (OFDD)

Earlier in Chapter 3 in BDD-based synthesis we defined the Shannon's decomposition type for constructing BDDs. Here we add two other decomposition types, namely the PPRM or Davio decompositions.

Consider a Boolean function f for n variables. The functions f_0, f_1 and f_2 are defined as:

$$f_0(x) = f(x_1, ..., x_{n-1}, 0)$$
$$f_1(x) = f(x_1, ..., x_{n-1}, 1)$$
$$f_2(x) = f_0(x) \oplus f_1(x)$$

Given the above definitions the two decomposition types can be defined by:

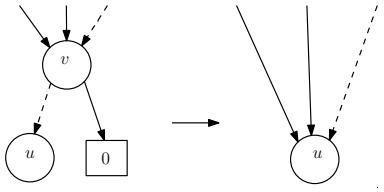


Figure 3.3: Elimination rule for OFDDs

$f = f_0 \oplus x_n f_2$ for **Positive Davio Decomposition**.

$f = f_1 \oplus \overline{x_n} f_2$ for **Negative Davio Decomposition**.

Definition 3.4. Ordered Functional Decision Diagrams (OFDD) are similar to OBDDs except the function f_i on node i is computed by the Reed-Muller decomposition as given below [22]:

1. A node v labeled as 1 or 0 represents the function $f_v = 1$ or $f_v = 0$ respectively.

2. A node v labeled as x_i whose $low(v)$ and $high(v)$ denote the functions h and g respectively, defines $f_v = g \oplus x_i h$.

The nodes of the OBDD in Figure 3.1 for the function $f = x_2x_3 + x_1\overline{x_2}\ \overline{x_3}$ represent an OFDD for the function $f = x_1x_2x_3 \oplus x_1 \oplus x_2x_3$ provided that the node decomposition is Davio. The nodes of the OFDD represent the following functions:

1. Nodes labeled by x_3: $f_{x_3,1}(x) = 1 \oplus x_3 \cdot 0 = 1, \quad f_{x_3,2}(x) = 0 \oplus x_3 \cdot 1 = x_3$

2. Nodes labeled by x_2: $f_{x_2,1}(x) = 1 \oplus x_2x_3, \quad f_{x_2,2}(x) = 0 \oplus x_2x_3 = x_2x_3$

3. Node labeled by x_1: $f(x) = x_2x_3 \oplus x_1(1 \oplus x_2x_3) = x_2x_3 \oplus x_1 \oplus x_1x_2x_3$

The reduction rules for OFDDs are similar to the rules applied to BDDs except:

1. **Elimination rule:** If the successor of the 1-edge of a node v is 0, then eliminate v and direct all the incoming edges to the successor of the 0-edge of node v.

Figure 3.3 shows the elimination rule for OFDDs.

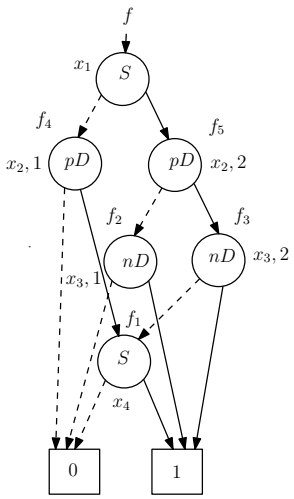
3.3 Ordered Kronecker Decision Diagram (OKFDD)

For some classes of Boolean functions, the OFDD representation is more compact than OBDD. Furthermore, there are some classes of functions that have polynomial size OKFDDs but exponential size OBDDs and OFDDs [3]. OKFDDs are an elegant combination of OFDDs and BDDs which showcase the advantages of each type.

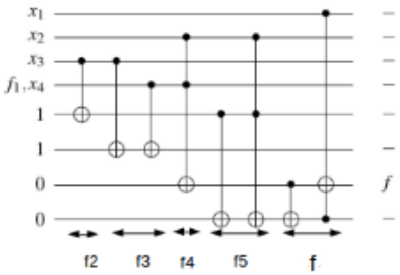
Definition 3.5. The Ordered Kronecker Decision Diagram is a representation type where each node v labeled by x_i is assigned a decomposition type using Decomposition Type List (DTL) $d : \{d_1, d_2, ...d_n\}$ where $d_i \in \{S, pD, nD\}$. Here S is Shannon decomposition, nD is negative Davio decomposition and pD is positive Davio decomposition [22].

In Figure 3.4(a) the OKFDD representing a function $f = x_1x_2x_4 \oplus x_1x_2\overline{x_3} \oplus x_1\overline{x_3} \oplus \overline{x_1}x_2x_4$ is constructed by the variable ordering $x_1 < x_2 < x_3 < x_4$. Each node has a decomposition type given by the DTL $d : \{S, pD, nD, S\}$. The node x_1 decomposes f into $f_{x_2,1} = x_2x_4$ and $f_{x_2,2} = x_2x_4 \oplus x_2\overline{x_3} \oplus \overline{x_3}$. The node $x_2, 2$ is further decomposed into $f_{x_3,1} = \overline{x_3}$ and $f_{x_3,2} = \overline{x_3} \oplus x_4$. Lastly, the node $x_3, 2$ factors into $f_{x_4} = x_4$ and 1 while the node $x_2, 1$ factors into $f_{x_4} = x_4$ and 0.

The algorithm to generate the reversible circuit from an OKFDD is similar to the BDD synthesis algorithm discussed in section 2.4 except that an OKFDD uses all the decomposition types for node structures such as shown in Table 3.1. Figure 3.4b shows the equivalent circuit for the OKFDD in Figure 3.4a using the Toffoli gates given in Table 3.1.



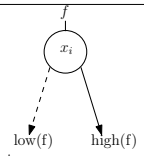
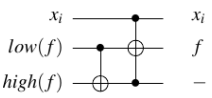
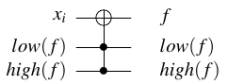
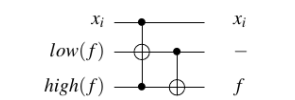
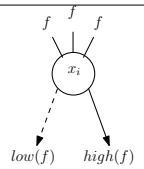
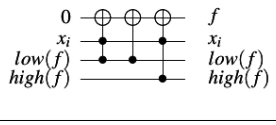
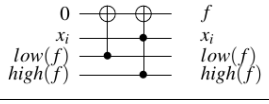
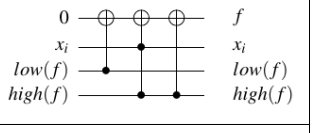
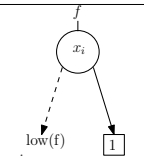
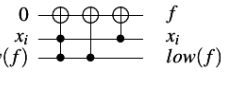
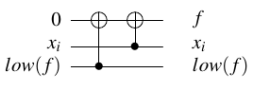
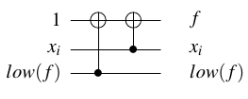
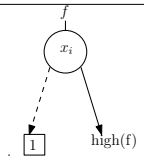
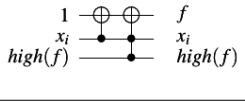
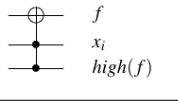
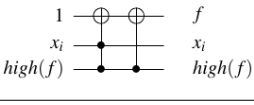
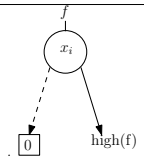
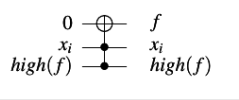
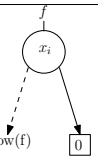
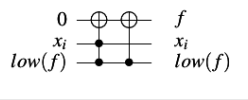
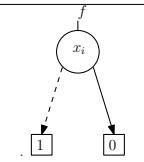
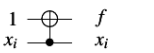
(a) OKFDD for the function $f = x_1x_2x_4 \oplus x_1x_2\overline{x_3} \oplus x_1\overline{x_3} \oplus \overline{x_1}x_2x_4$.



(b) Equivalent circuit.

Figure 3.4: OKFDD with the specified DTL [3] and its equivalent circuit.

Table 3.1: Toffoli gate circuits for node structures of decomposition types.

	Nodes	Shannon	Positive Davio	Negative Davio
1.				
2.				
3.				
4.				
	Nodes	**Shannon**	**Nodes**	**Shannon**
5.			6. 	
7. ‘			—	—

Chapter 4

Reducing lines using OKFDDs

The demand for an efficient and compact reversible circuit has led to the introduction of various optimization techniques in the past years. Optimization of a reversible circuit is carried out based on the parameters discussed in section 2.3. One of these parameters, which counts the number of lines (qubits) in a reversible circuit is line count. As we know now qubits are a limited resource and thus it is desirable to have reversible circuits generated with minimum line count. In this chapter we discuss the upper and the lower bounds on the number of lines required by a reversible circuit to realize a Boolean function as explained in [20]. We also propose an algorithm to reduce the number of lines generated using OKFDDs for reversible logic synthesis.

4.1 Bounds on reversible circuit lines

Theorem 1. *For a given function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ the number of garbage bits required is at most $log_2\mu$ where μ denotes the maximum repetition of an output pattern [20].*

Proof. As discussed in Section 1.1 an irreversible function is embedded into a reversible function to be implemented. Since an irreversible function is not bijective, the outputs of an irreversible truth table repeat for some input values. To make the output of the truth table unique additional bits are added. For example if the output pattern $(o_1, o_2, ..o_m)$ occurs most frequently i.e. μ times, then $\lceil log_2(\mu) \rceil$ new bits are required to make the output unique. Therefore, $2^{\lceil log_2(\mu) \rceil}$ extra output patterns are created.

Corollary 1. (Lower Bound) *A reversible circuit requires at least $m + \lceil log_2(\mu) \rceil$ lines to implement an irreversible Boolean function [20].*

Let μ be the maximum number of times an output pattern repeats itself in an irreversible

truth table of a Boolean function. Then, approximately $\lceil log_2(\mu) \rceil$ additional lines (garbage outputs) are required to convert an irreversible function to a reversible function [20]. Therefore, to implement a Boolean function $\mathbb{B}^n \rightarrow \mathbb{B}^m$ at least $m + \lceil log_2(\mu) \rceil$ lines are required. The final circuit consists of n inputs, m outputs and $\lceil log_2(\mu) \rceil$ additional constant inputs. If in any case the number of inputs n is greater than $\lceil log_2(\mu) \rceil$ then the minimum number of lines required is n as the number of lines in a circuit cannot be less than the number of variables present in the function.

The value of μ is evaluated by scanning the truth table of the function. The computation involves a two-level description [48] of a SOP truth table of a Boolean function. A two-level description represents a SOP truth table with '1' for positive variables, '0' for negative variables and '-' for don't care values. This representation is similar to ESOP cube-list Figure 2.13. Each row denotes the conjunction of the variables. In a two-level description a function is represented by the disjunction of each row while in ESOP a function is represented by the Ex-OR of each row. The complexity of computing the value of μ depends on the size of the truth table. Therefore, the complexity is polynomial in the number of rows of the truth table (or other two-level description); however it may be exponential in the number of variables if there are no don't care conditions in the truth table.

Example: Consider the truth table shown in Figure 4.1 in two-description level format. The first row $(11-01)$ has two don't cares so the output 101 is repeated $2^1 = 2$ times. Similarly, for the second row $(10-01)$ which represents the conjunction of x_1, $\overline{x_2}$, $\overline{x_4}$ and x_5 the output pattern 110 is repeated $2^1 = 2$ times plus $2^3 = 8$ times in fourth row $(00---)$. Figure 4.1 shows how many times an output pattern is repeated. Since the output 110 is repeated the maximum number of times (10 times), the value of $\mu = 10$ and $\lceil log_2(\mu) \rceil = \lceil log_2(10) \rceil = 4$. Thus, the minimum number of lines required to implement the function is $m + \lceil log_2(\mu) \rceil = 3 + 4 = 7$ where $m = 3$.

Corollary 2. (Upper Bound) *A given function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ requires at most $n + m$ circuit lines to implement [48].*

x_1	x_2	x_3	x_4	x_5	f_1	f_2	f_3	Freq
1	1	-	0	1	1	0	1	$2^1 = 2$
1	0	-	0	1	1	1	0	$2^1 = 2$
0	-	-	-	1	1	0	1	$2^3 = 8$
0	0	-	-	-	1	1	0	$2^3 = 8$
0	-	1	1	1	1	0	0	$2^1 = 2$
-	-	0	0	1	1	0	0	$2^2 = 4$

Figure 4.1: Calculating μ for an irreversible truth table

According to the **Theorem 1** the minimal number of lines required to implement a function is $m + \lceil log_2(\mu) \rceil$. Considering the worst case the maximum number of times an output pattern can repeat for 2^n i.e. $\mu = 2^n$ for an n variable Boolean function. Therefore, in this case the number of lines required is $m + \lceil log_2(2^n) \rceil = m + n$.

4.2 The Algorithm

Algorithm MOKFDD is the proposed algorithm for line reduction in OKFDDs. The algorithm explains the process of synthesizing a Boolean function from a given OKFDD. Before discussing the algorithm we introduce the shared node ordering for the algorithm.

Shared nodes: Shared node implementation is previously explained in section 2.4 under BDD based synthesis. In OKFDDs the implementation of a shared node structure depends on the decomposition types of the nodes involved. As discussed earlier if two or more nodes have a same successor node *S* then *S* is a shared node. To obtain an optimized circuit from an OKFDD we introduce the node ordering in case of a shared node. The illustration of the concept is given in Figure 4.2 using Positive Davio (pD) decomposition where the function f_3 is represented by a shared node.

In Figure 4.2 the node labeled x_j is a shared node as it is shared by two x_i nodes. In the case of a shared node, the node (v_{f_1}) with the 1-edge leading to the x_j is realized first and then the node (v_{f_2}) with the 0-edge to x_j. In this case the circuit has no additional constant circuit lines but when synthesizing from a OKFDD the shared node implementation

depends on the decomposition type and [45] gives a shared node implementation, but only for Shannon decomposition types. The cost of the circuit depends on the decomposition type of the nodes.

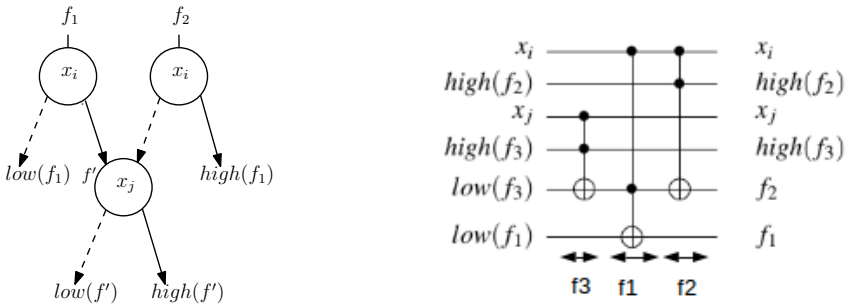


Figure 4.2: Shared node and equivalent circuit

The algorithm MOKFDD is based on synthesizing reversible circuits using the factors of a Boolean function through a decision diagram. Our addition to the work in [42] is a new sub-circuit or template (highlighted in Table 4.1) for positive-Davio decompositions and an ordering in which the variables are to be addressed when a shared node is encountered in the OKFDD. A template in the decision diagram is a representation of a node structure in the form of reversible gates in a reversible circuit according to the given decomposition type. These templates can be computed using the corresponding decomposition type formulas. The algorithm MOKFDD introduces a process of mapping an OKFDD to a reversible circuit in such a way that the circuit lines are minimized. The input for the algorithm MOKFDD is the OKFDD or acyclic directed graph $G(V,E)$ generated by the algorithm in [42]. The output of the algorithm MOKFDD is a reversible circuit termed as *rev_cascade*. Initially, we take an empty circuit *rev_cascade* and add lines or gates when required. In the next three steps we define the depth d of the OKFDD, the number of nodes k at each level and an empty list L respectively. In step 5 we start traversing the graph bottom-up for each level l from the non-terminal nodes (level $d-1$) to the root node. In step 6 each unvisited node v_j^l at level l is scanned for the decomposition type. In Step 7 and 8 we implement each node v_j^l with an update in *rev_cascade* and mark it as visited. In the next step we search for

the parent nodes V_j of each node v_j^l. Steps 10 to 15 define the cases for a shared node. For a shared node there can be only two parent nodes ($|v_j| = 2$). Step 11 checks the case where (considering the Figure 4.2) 0-*edge* of f_2 (v_j^{p1}) and 1-*edge* of f_1 (v_j^{p2}) leads to the same shared node x_j. In this case f_1 (v_j^{p2}) is inserted in the L first and then f_2 (v_j^{p1}). The second case is the alternate case of the first one. Finally in Steps 17 to 19 the list L is traversed and each element or node in the list v is implemented according to the insertion order before going to the next level. Each of these nodes are marked visited and *rev_cascade* is updated.

The procedure $parent(G, v_j^l)$ searches for the parent nodes v_j^{p1} and v_j^{p2} of the input node v_j^l at level l in the graph by tracking the incoming edges to the input node. In the procedure $sub\_circuit(v)$ each node v in an OKFDD is implemented by selecting a sub-circuit from Table 4.1 depending on the node decomposition type and structure. The most frequent sub-circuits for each decomposition type shown in Table 4.1 are given in [42]. We have added a new sub-circuit for Davio decomposition of node structure 3 which requires no additional lines and only one CNOT gate.

Input: A KFDD (directed acyclic graph) $G(V,E)$ where $|V| = n$ and $\{0-edge, 1-edge\} \in E$.

Output: A reversible circuit *rev_cascade* with minimal lines.

$update\_cascade(rev\_cascade, sub\_circuit(v))$ procedure evaluates each sub-circuit selected by the procedure $sub\_circuit(v)$ adds to the main circuit *rev_cascade*. If the *rev_cascade* has the input states required by the sub-circuit then they merge otherwise new constant additional lines are added for the required input states.

Example: *Figure 4.3 shows an OKFDD for the function* $\overline{x_1}(x_2 \oplus x_3) \oplus \overline{x_2}x_3$ *where* $\rho = x_1 < x_2 < x_3$. *Firstly,* f *decomposes into* $f_3 = \overline{x_2}x_3 \oplus x_3$ *and* $f_2 = x_2 \oplus x_3$ *by nD decomposition type. Then,* f_3 *decomposes into* $f_1 = x_3$ *and terminal node* 1 *by pD type. Next,* f_2 *into* $f_1 = x_3$ *and* 0 *by pD type. Finally,* f_1 *decomposes into terminal nodes* 0 *and* 1 *by Shannon type.*

Algorithm 1 $MOKFDD(G)$

1: $rev\_cascade \leftarrow \emptyset$
2: $d \leftarrow$ depth of an OKFDD.
3: $k \leftarrow$ no. of nodes in each level of an OKFDD.
4: $L \leftarrow \emptyset$
5: **for each** level l; $d-1 \leq l \leq 0$ **do**
6: **for each** unvisited node $v_j^l \in V$; $0 \leq k$ **do**
7: $rev\_cascade \leftarrow update\_cascade(rev\_cascade, sub\_circuit(v_j^l))$
8: Mark v_j visited.
9: $V_j \leftarrow parent(G, v_j^l)$
10: **if** $|V_j|$ equals 2 **then**
11: **Case 1:** 0-*edge* of $v_j^{p1} \in V_j$ and 1-*edge* of $v_j^{p2} \in V_j$ share same node **then**
12: Insert v_j^{p2} and then v_j^{p1} in L.
13: **Case 2:** 1-*edge* of $v_j^{p1} \in V_j$ and 0-*edge* of $v_j^{p1} \in V_j$ share same node **then**
14: Insert v_j^{p1} and then v_j^{p2} in L.
15: **end if**
16: **end for**
17: **for each** element $v \in L$ **do**
18: $rev\_cascade \leftarrow update\_cascade(rev\_cascade, sub\_circuit(v))$
19: Mark v visited.
20: **end for**
21: **end for**

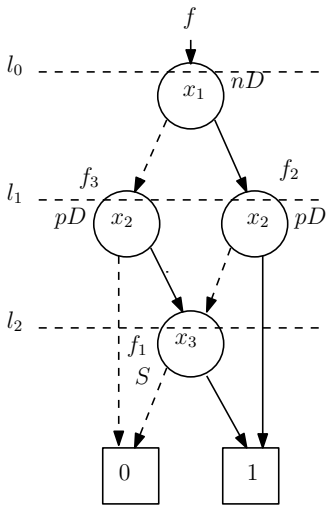
To generate the circuit using MOKFDD, we traverse the OKFDD from node x_3 to the root node x_1 level wise starting from l_2. The suitable sub-circuit is selected from Table 4.1 for each node. Here node f_1 at l_2 is a shared node and thus, f_3 is implemented before f_2. For node f_1 at l_2 since the function is x_3 (a single variable) we require only a single line in the circuit. At l_1 for node f_2 the pD node structure #3 is used from the Table 4.1. Similarly, for the node f_3, the pD node structure 5 is used which is similar to a S decomposition. Lastly, for node f at l_0, the nD node structure #1 is used. Figures 4.3b and 4.3c show the equivalent circuit implementations using the previous algorithm [42] and our algorithm respectively. As illustrated our approach produces a smaller circuit (quantum cost = 12 and line count = 4) as compared to the previous approach (quantum cost = 13 and line count =

Algorithm 2 Procedure $parent(G, v_j^l)$

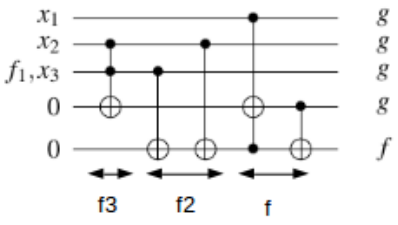
$V_j \leftarrow$ parent nodes of v_j^l
Return V_j

Algorithm 3 Procedure $sub\_circuit(v)$

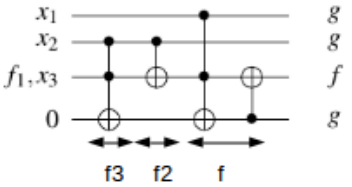
if child nodes of v are non-terminal **then**
  $circuit(v) \leftarrow$ template 1 or 2 from Table 4.1
else
  $circuit(v) \leftarrow$ other matching template from Table 4.1
end if
Return $circuit(v)$



(a) OKFDD for the function $\overline{x_1}(x_2 \oplus x_3) \oplus \overline{x_2}x_3$



(b) Previous approach



(c) Our approach

Figure 4.3: OKFDD and its reversible circuit from different algorithms.

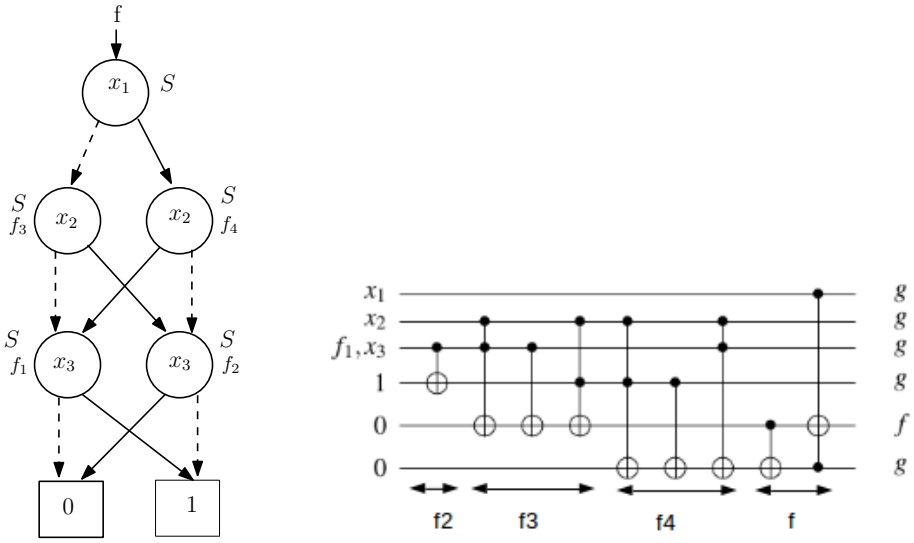
5) [42].

In order to compare our approach with the existing BDD based synthesis, consider the SOP formulation for the function used in the example above i.e. $f = x_1 + x_2 + x_3$. The

Algorithm 4 Procedure *update_cascade*(*rev_cascade*, *sub_circuit*(v))

if *rev_cascade* has all the required *sub_circuit*(v) input states **then**
Merge *sub_circuit*(v) and *rev_cascade*
else
Add extra constant lines for the missing input states to *rev_cascade* and merge *sub_circuit*(v)
end if
Return *rev_cascade*

OBDD for this function using the order of nodes $\rho = x_1 < x_2 < x_3$ is shown in Figure 4.4(a). As illustrated the nodes in an OBDD are more compared to the nodes in an OKFDD for the same function. This shows that the number of nodes in a decision diagram depend on the decomposition type being used. The resultant circuit of the OBDD is shown in Figure 4.4(b). The number of lines in the circuit is 6 and the quantum cost is 29. In this case the cost of the circuit is more than the cost of the circuit synthesized by OKFDD even for a simple function with three variables. The comparison of our approach to OBDD based synthesis for the reversible benchmarks is shown in the results chapter 5. The comparison

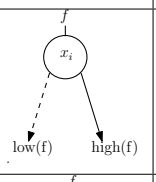
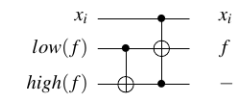
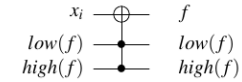
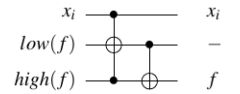
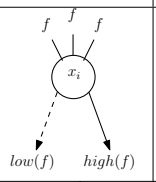
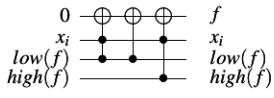
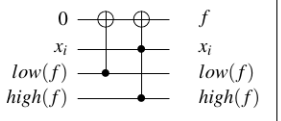
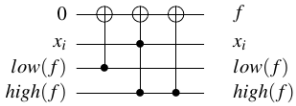
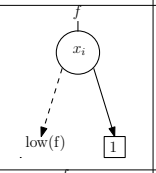
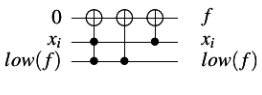
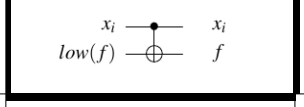
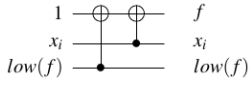
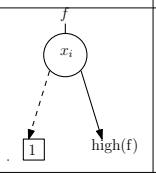
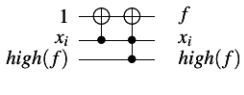
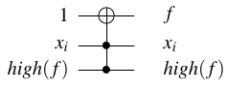
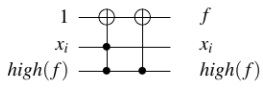
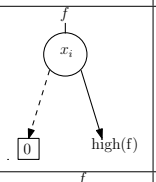
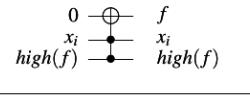
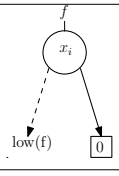
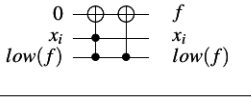
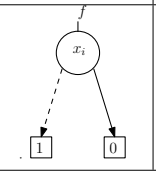


(a) OBDD (b) Reversible circuit.

Figure 4.4: (a) OBDD for the function $f = x_1 + x_2 + x_3$ and (b) its equivalent reversible circuit.

of our approach with other algorithms help to analyze the benefits and limitations of the proposed algorithm. The next section discusses these points along with the contribution.

Table 4.1: Toffoli gate circuits for node structures of decomposition types.

	Nodes	Shannon	Positive Davio	Negative Davio
1.				
2.				
3.				
4.				
	Nodes	**Shannon**	**Nodes**	**Shannon**
5.			6. 	
7.			—	—

4.3 Discussion

The proposed algorithm MOKFDD uses OKFDD for the synthesis of a reversible circuit. In OBDD synthesis approach the function is decomposed into smaller sub-functions using Shannon decomposition until a constant value is reached. These sub-functions are then mapped on to the reversible circuit. The drawback of using an OBDD based synthesis is that the circuit uses more additional lines to preserve the sub-function for future use compared to OKFDD based synthesis approach for large functions. However, OBDD based synthesis is the first hierarchical synthesis approach which can synthesize Boolean

functions upto 70 variables [42]. The illustration for this is given by an example in the previous section. This reason motivates us to use an OKFDD based synthesis for the algorithm MOKFDD. In an OKFDD approach the functions are decomposed using the Davio decomposition types with Shannon type which allows more compact representations of the sub-functions. Therefore, the advantages of MOKFDD include:

1. The algorithm can take upto 70 variables as input.

2. The resulting reversible circuit is cost effective (reduces line count) compared to other algorithms.

3. The algorithm does not traverse the truth table to generate the reversible circuit.

The algorithm MOKFDD uses the advantages of an OKFDD based synthesis along with the reduction in the circuit lines. It overcomes the limitations introduced by other heuristic methods of synthesis [24], [16], [13] such as limited input variables, scanning of large truth tables and costly circuits. However, there are few limitations associated with the algorithm. They are as follows:

1. Although MOKFDD produces cost effective reversible circuits for large inputs, the circuits are still expensive to implement practically.

2. The size of the decision diagram may increase exponentially with the increase in the number of input variables.

With all these advantages and disadvantages discussed above MOKFDD significantly reduces the number of lines in the reversible benchmarks. The experimental evaluation done in the next chapter shows the reduction table for the benchmarks. The three important contributions from this study are:

1. Introduction of a new template in the pD node structure significantly reduces the number of lines in the circuits.

2. The proposed ordering for the shared nodes allows to implement the template for the line reduction efficiently.

3. The level-wise traversal of the decision diagram checks and implements the shared nodes according to the shared node order in a single pass.

The results form the first contribution depend on the frequency of the presence of the node structure highlighted in Table 4.1 in the decision diagram. Since the node structure is one of the most frequently used structure the line reduction in the circuits is notable. The study can be extended to find more of such node structures with different decomposition types to furthur reduce the number of circuit lines.

Chapter 5

Experimental Evaluation

The algorithm MOKFDD is implemented in C++ in Revkit [41]. The reversible functions in Table 5.1 are from Revlib [44]. The algorithm to generate an OKFDD is given in Revkit under KFDD-based synthesis algorithms which includes the PUMA package for decompositions and optimizing algorithms. The sifting algorithm [33] is used by PUMA to find a variable ordering and DTL that results in the fewest nodes in the OKFDD. The circuits obtained by our algorithm (Table 5.1) have been verified using Revkit's equivalence checker. The runtime of the experiment including the verification process for all the benchmarks shown in Table 5.1 is few seconds. The experiment was performed on a 1.9 GB, Intel Core 2 Duo processor Linux machine.

In Table 5.1 the first column shows the functions. The next two columns consist of the number of inputs and outputs for corresponding functions. The results of our algorithm are compared with the results of the previous KFDD approach [42] and BDD approach [45]. The notation 'L' denotes the number of lines in the circuit while 'QC' and 'GC' denote the quantum cost and gate count respectively. The changes in the metrics are shown by 'Δ L', 'Δ QC' and 'Δ GC'. The results show a significant decrease in the number of lines as well as in quantum cost and gate count. In the best case (e.g. *plus*127) the line reduction is 42% compared to the KFDD approach and 29% (e.g. *tial*) compared to the BDD approach. The average line reduction is approximately 10% in comparison to the KFDD approach. Comparing the quantum cost values the average reduction is around 7% and 23% for the KFDD and the BDD approaches respectively.

Since an OKFDD uses all the decomposition types with the variable ordering, this type of decision diagram is more likely to generate a smaller realization compared to other DD

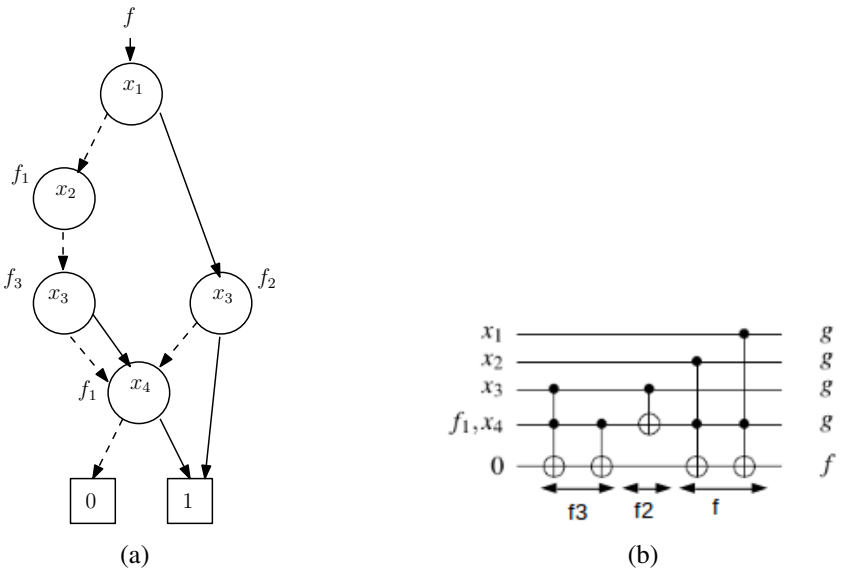
based algorithms for large functions. Some of the functions show great improvement such as *plus*127 and *tial* due to the frequent presence of node structure #3. We can see that if a pD decomposition type #3 is used then only one gate and no additional lines are required. Although there are a few functions that do not show any improvement compared to lines in KFDD approach, they display QC or GC minimization such as *sqrt*8 and *ex*2. We hypothesize that when a node is shared with more than two nodes then an additional line is required to preserve the function for future use. This compensates for the previous removal of lines.

A comparison of our experimental results to the lower bound discussed in section 4.1 on the number of lines in a reversible circuit is shown in Table 5.2. Column 1 and 2 show the number of inputs and outputs of a function respectively. Column 3 shows the lower bound ($\lceil log_2(\mu) \rceil$) or the least additional lines required for a function. Column 4 shows the total number of lines ($m + \lceil log_2(\mu) \rceil$) required. The last column in the table shows the number of lines required by a function using our approach. The difference in the values for some of the functions is small such as *ex*1 and *rd_32*, and large in other functions such as *plus*63 and *sqn*. The difference in the values may depend on the decomposition of a function. Importantly, the comparison shows that there is scope for more improvement.

The variable ordering of a DD plays an important role in the designing of a reversible circuit. The size of a DD depends on the chosen variable order [33]. The number of gates and lines in a reversible circuit depends on the number of the non-terminal nodes in a DD. If a DD has k nodes and each node needs at least a single extra line then at most $k+n$ lines are required to implement an n input Boolean function [45]. Similarly, $3 \times k$ Toffoli gates are required to realize a DD with k nodes considering that each node can have at most 3 Toffoli gates [45]. In order to minimize the size of DD, the sifting algorithm [33] is used to obtain the best possible variable ordering for a DD. The algorithm tests for all the possible variable orderings and selects the best ordering to minimize the size of a DD. It initially takes a variable and swaps it with other variables until the best position is confirmed. Since

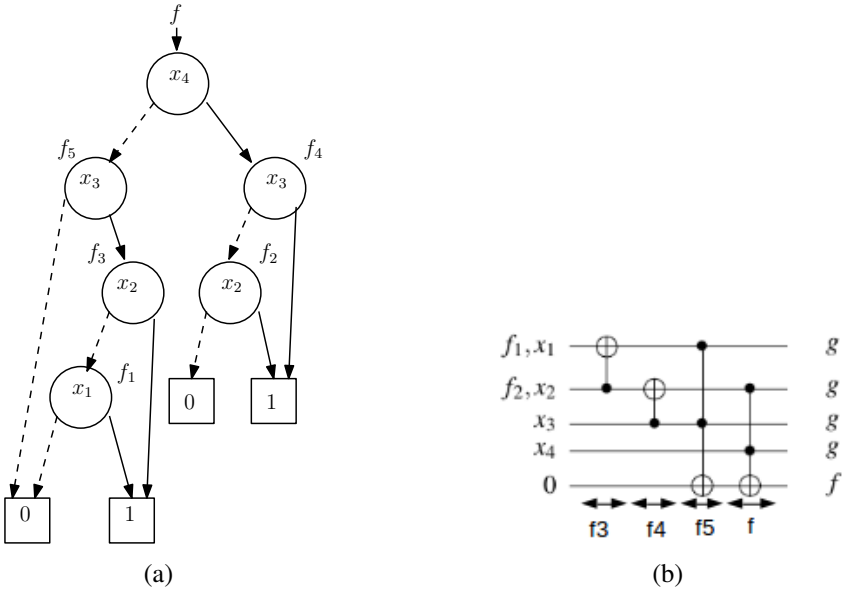
each position is checked for the best results the algorithm is a brute force algorithm. Each variable is swapped until no significant decrease in the size of the DD occurs.

Considering the fact that the variable ordering reduces the size of the DD, we can further investigate the changes in a DD by comparing the variable ordering to favor a particular set of node structures. In Figure 5.1 an OKFDD for a function $f = x_1(x_3 \oplus x_4) \oplus \overline{x_3}x_4 \oplus x_2x_3 \oplus x_2x_4$ is given with the variable order of $x_1 < x_2 < x_3 < x_4$ and pD decomposition on all the nodes. This variable ordering for the function is best according to the sifting algorithm due to the minimum number of nodes. Randomly testing the other variable orders an OKFDD with the structure shown in Figure 5.1 is obtained. The quantum cost is 17 while the line count is 4. This is a case where the number of nodes is increased compared to the original OKFDD to favor the pD node structure #3 from Table 5.2. Comparing the resultant circuits from both OKFDDs Figure 5.2 gives a more compact and less expensive circuit with a quantum cost of 12 and line count 4. The reason behind the cost reduction in spite of an increase in the number of nodes is that the increased number of nodes represents the variable itself as a function such as x_1 and x_2. Therefore, these nodes require a single input line (no additional line) to represent the function in the circuit. There may be other different cases where an increase in the number of nodes may increase the cost of the circuit.



(a) (b)

Figure 5.1: (a) OKFDD with the variable order $x_1 < x_2 < x_3 < x_4$ and (b) its equivalent circuit.



(a) (b)

Figure 5.2: (a) OKFDD with the variable order $x_4 < x_3 < x_2 < x_1$ and (b) its equivalent circuit.

Table 5.1: Experimental Results for the Algorithm

Function Name	**#in**	**#out**	**Our approach**			**OKFDD approach [42]**						**BDD approach [45]**					
			L	**QC**	**GC**	**L**	**QC**	**GC**	**Δ L**	**Δ QC**	**Δ GC**	**L**	**QC**	**GC**	**Δ L**	**Δ QC**	**Δ GC**
Rd_32	3	2	6	16	8	8	20	12	-2	-4	-4	6	22	10	0	-6	-2
mod10	4	4	12	60	24	14	64	28	-2	-4	-4	13	80	28	-1	-20	-4
one_two_three	3	3	9	35	15	10	37	17	-1	-2	-2	9	44	16	0	-9	-1
plus127	13	13	26	85	41	37	97	53	-11	-12	-12	25	98	54	1	-13	-13
plus63	12	12	24	78	38	34	89	49	-10	-12	-11	23	89	49	1	-11	-11
radd	8	5	17	59	27	21	68	36	-4	-9	-9	28	217	73	-11	-158	-46
rd53	5	3	14	64	32	15	69	37	-1	-5	-5	-	-	-	-	-	-
rd73	7	3	23	107	51	25	115	59	-2	-8	-8	25	217	73	-2	-110	-22
rd84	8	4	30	153	69	33	161	77	-3	-8	-8	34	304	104	-4	-151	-35
root	8	5	51	413	153	52	415	159	-1	-2	-6	45	444	140	6	-31	13
sqr6	6	12	48	304	112	50	315	123	-2	-11	-11	49	486	154	-1	-182	-42
sqrt8	8	4	28	166	62	28	170	66	0	-4	-4	-	-	-	-	-	-
z4	7	4	14	46	22	20	56	32	-6	-10	-10	14	66	30	0	-20	-8
z4ml	7	4	14	46	22	20	56	32	-6	-10	-10	14	66	30	0	-20	-8
add6	12	7	40	183	79	39	184	80	1	-1	-1	54	499	159	-14	-316	-80
adr4	8	5	16	54	26	23	65	37	-7	-11	-11	16	74	34	0	-20	-8
bw	5	28	78	581	237	81	593	249	-3	-12	-12	87	943	307	-9	-362	-70
cm82a	5	3	10	31	15	12	36	20	-2	-5	-5	13	82	30	-3	-51	-15
con1	7	2	15	94	30	17	100	36	-2	-6	-6	-	-	-	-	-	-
cycle	12	12	28	97	49	34	104	56	-6	-7	-7	39	202	78	-11	-105	-29
dc1	4	7	21	141	45	22	146	50	-1	-5	-5	20	160	56	1	-19	-11
inc	7	9	56	442	158	58	447	171	-2	-5	-13	53	579	187	3	-137	-29
xor195	5	1	6	6	6	10	10	10	-4	-4	-4	6	8	8	0	-2	-2
sym9	9	1	30	150	70	28	154	74	2	-4	-4	27	206	62	3	-56	8
ex1	5	1	6	6	6	10	10	10	-4	-4	-4	6	8	8	0	-2	-2
ex2	5	1	11	50	18	11	48	20	0	2	-2	11	73	25	0	-23	-7
max46	9	1	62	664	216	67	684	228	-5	-20	-12	54	598	190	8	66	26
sym10	10	1	38	259	103	40	266	110	-2	-7	-7	32	253	77	6	6	26
life175	9	1	26	159	67	31	168	76	-5	-9	-9	27	204	64	-1	-45	3
9syml	9	1	30	150	70	28	154	74	2	-4	-4	27	206	62	3	-56	8
sao	10	4	74	562	186	73	568	192	1	-6	-6	74	667	211	0	-105	-25
tial	14	8	410	4185	1681	419	4179	1703	-9	6	-22	578	7609	2253	-168	-3424	-572
urf1	9	9	384	4320	1628	390	4372	1672	-6	-52	-44	374	6080	1848	10	-1760	-220
urf2	8	8	206	2276	920	209	2304	948	-3	-28	-28	209	3187	983	-3	-911	-63
wim	4	7	18	93	37	19	95	39	-1	-2	-2	18	107	39	0	-14	-2
mjority	5	1	10	37	17	10	38	18	0	-1	-1	10	41	13	0	-4	4
sym6	6	1	16	76	28	16	77	29	0	-1	-1	14	93	29	2	-17	-1
cordic	23	2	52	264	104	53	264	108	-1	0	-4	-	-	-	-	-	-
cm85a	11	3	35	161	61	35	164	64	0	-3	-3	36	275	87	-1	-114	-26
clip	9	5	66	546	214	69	566	226	-3	-20	-12	66	704	228	0	-158	-14
e64	64	64	195	897	385	195	899	387	0	-2	-2	-	-	-	-	-	-
cps	24	109	619	5677	2305	625	5668	2332	-6	-9	-27	-	-	-	-	-	-

Table 5.2: Lower bound Table

Function name	#inputs (n)	#outputs (m)	Lower Bound $LB = \lceil log_2(\mu) \rceil$	#Total Lines (LB + m)	Our Approach #Lines
Rd_32	3	2	2	4	6
mod10	4	4	4	8	12
one_two_three	3	3	2	5	9
plus127	13	13	0	13	26
plus63	12	12	0	12	24
radd	8	5	9	14	17
rd53	5	3	5	8	14
rd73	7	3	9	12	23
rd84	8	4	7	11	30
root	8	5	8	13	51
sqn	7	3	7	10	41
sqr6	6	12	7	19	48
sqrt8	8	4	9	13	28
z4	7	4	8	12	14
z4ml	7	4	8	12	14
add6	12	7	13	20	40
adr4	8	5	9	14	16
bw	5	28	8	36	78
cm82a	5	3	5	8	10
con1	7	2	7	9	15
cycle	12	12	0	12	28
dc1	4	7	2	9	21
inc	7	9	3	12	56
xor195	5	1	4	5	6
sym9	9	1	10	11	30
ex1	5	1	4	5	6
ex2	5	1	4	5	11
max46	9	1	7	8	62
sym10	10	1	10	11	38
life175	9	1	9	10	26
9syml	9	1	10	11	30
sao	10	4	10	14	74
tial	14	8	15	23	410
urf1	9	9	0	9	384
urf2	8	8	0	8	206
wim	4	7	4	11	18
mjority	5	1	5	6	10
sym6	6	1	6	7	16
cordic	23	2	26	28	52
cm85a	11	3	11	14	35
clip	9	5	10	15	66
e64	64	64	63	128	195
cps	24	109	28	137	619

Bibliography

[1] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. Minimizing DNF Formulas and AC Circuits given a Truth Table. In *Computational Complexity, 2006. CCC 2006. Twenty-First Annual IEEE Conference on*, pages 15–pp. IEEE, 2006.

[2] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary Gates for Quantum Computation. *Physical Review A*, 52(5):34–57, 1995.

[3] Bernd Becker, Rolf Drechsler, and Michael Theobald. OKFDDs versus OBDDs and OFDDs. In *Automata, Languages and Programming*, pages 475–486. Springer, 1995.

[4] Felix Bloch. Nuclear Induction. *Physical review*, 70(7-8):460–463, 1946.

[5] Alex Bocharov, Krysta M Svore, Y Gurevich, et al. From Reversible Logic Gates to Universal Quantum Bases. *Bulletin of the European Association for Theoretical Computer Science*, (110):79–85, 2013.

[6] Beate Bollig and Ingo Wegener. Improving the Variable Ordering of OBDDs is NP-complete. *Computers, IEEE Transactions on*, 45(9):993–1002, 1996.

[7] Karl S Brace, Richard L Rudell, and Randal E Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 40–45. ACM, 1991.

[8] Anupam Chattopadhyay, Chander Chandak, and Kaushik Chakraborty. Complexity Analysis of Reversible Logic Synthesis. *arXiv preprint arXiv:1402.0491*, 2014.

[9] Gerhard W Dueck and Dmitri Maslov. Reversible Function Synthesis with Minimum Garbage Outputs. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 154–161, 2003.

[10] K Fazel, M Thornton, and JE Rice. ESOP-based Toffoli Gate Cascade Generation. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 206–209, 2007.

[11] David Y Feinstein, Mitchell A Thornton, and D Michael Miller. Partially Redundant Logic Detection using Symbolic Equivalence Checking in Reversible and Irreversible Logic Circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1378–1381. ACM, 2008.

[12] Michael P Frank. Introduction to Reversible Computing: Motivation, Progress, and Challenges. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 385–390. ACM, 2005.

[13] Pallav Gupta, Abhinav Agrawal, and Niraj K Jha. An Algorithm for Synthesis of Reversible Logic Circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(11):2317–2330, 2006.

[14] Julio Gea-Banacloche Jared Ostmeyer. Visualizing Quantum Dynamics: Bloch Spheres and Q-Functions. `http://comp.uark.edu/~jgeabana/blochapps/bloch.html`, 2006.

[15] Jackie Rice Jayati. Line Reduction in Reversible Circuits Using KFDDs. In *Proceedings of the 2015 IEEE/Pacific Rim Conference on Communications, Computers and Signal Processing*. IEEE Computer Society Press, 2015.

[16] Pawel Kerntopf. A new heuristic algorithm for reversible logic synthesis. In *Proceedings of the 41st annual Design Automation Conference*, pages 834–837. ACM, 2004.

[17] Andrei B Khlopotine, Marek A Perkowski, and Pawel Kerntopf. Reversible Logic Synthesis by Iterative Compositions. In *International Workshop on Logic and Synthesis*, pages 261–266. Citeseer, 2002.

[18] Rolf Landauer. Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[19] D Maslov. Reversible Benchmarks. `http://www.cs.uric.ca/-dmaslov/`, 2013.

[20] Dmitri Maslov and Gerhard W Dueck. Reversible Cascades with Minimal Garbage. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(11):1497–1509, 2004.

[21] Dmitri Maslov, Gerhard W Dueck, and D Michael Miller. Toffoli Network Synthesis with Templates. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(6):807–817, 2005.

[22] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*. Springer Science & Business Media, 1998.

[23] Ralph C Merkle. Reversible Electronic Logic Using Switches. *Nanotechnology*, 4(1):21–27, 1993.

[24] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. A Transformation Based Algorithm for Reversible Logic Synthesis. In *Design Automation Conference, 2003. Proceedings*, pages 318–323. IEEE, 2003.

[25] D Michael Miller and Zahra Sasanian. Lowering the Quantum Gate Cost of Reversible Circuits. In *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, pages 260–263. IEEE, 2010.

[26] D Michael Miller, Robert Wille, and Rolf Drechsler. Reducing Reversible Circuit Cost by Adding Lines. In *International Symposium on Multi-Valued Logic*, pages 217–222, 2010.

[27] Alan Mishchenko and Marek Perkowski. Fast Heuristic Minimization of Exclusive-Sums-Of-Products.

[28] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge university press, 2010.

[29] W David Pan and Mahesh Nalasani. Reversible Logic. *Potentials, IEEE*, 24(1):38–41, 2005.

[30] Marek Perkowski and Malgorzata Chrzanowska-Jeske. An Exact Algorithm to Minimize Mixed-radix Exclusive Sums of Products for Incompletely Specified Boolean Functions. In *Circuits and Systems, 1990., IEEE International Symposium on*, pages 1652–1655. IEEE, 1990.

[31] Marek Perkowski, Lech Jozwiak, Pawel Kerntopf, Alan Mishchenko, Anas Al-Rabadi, Alan Coppola, Andrzej Buller, Xiaoyu Song, Svetlana Yanushkevich, Vlad P Shmerko, et al. A General Decomposition for Reversible Logic. 2001.

[32] Aditya K Prasad, Vivek V Shende, Igor L Markov, John P Hayes, and Ketan N Patel. Data Structures and Algorithms for Simplifying Reversible Circuits. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2(4):277–293, 2006.

[33] Richard Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, pages 42–47. IEEE Computer Society Press, 1993.

[34] Mehdi Saeedi and Igor L Markov. Synthesis and Optimization of Reversible Circuits–A Survey. *ACM Computing Surveys (CSUR)*, 45(2):21–27, 2013.

[35] Tsutomu Sasao. An Exact Minimization of AND-EXOR Expressions using Reduced Covering Functions. In *Proc. of the Synthesis and Simulation Meeting and International Interchange*, pages 374–383, 1993.

[36] Tsutomu Sasao and Philipp Besslich. On the Complexity of Mod-2l Sum PLA's. *Computers, IEEE Transactions on*, 39(2):262–266, 1990.

[37] Robert R Schaller. Moore's Law: Past, Present and Future. *Spectrum, IEEE*, 34(6):52–59, 1997.

[38] Aleksandr Sergeevich Semenov. International Journal of Optical Computing. *Quantum Electronics*, 20(5):581–581, 1990.

[39] Vivek V Shende, Aditya K Prasad, Igor L Markov, and John P Hayes. Reversible Logic Circuit Synthesis. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pages 353–360. ACM, 2002.

[40] Vivek V Shende, Aditya K Prasad, Igor L Markov, and John P Hayes. Synthesis of Reversible Logic Circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(6):710–722, 2003.

[41] Mathias Soeken, Stefan Frehse, Robert Wille, and Rolf Drechsler. RevKit: An open source toolkit for the design of reversible circuits. In *Reversible Computation 2011*, volume 7165 of *Lecture Notes in Computer Science*, pages 64–76, 2012. RevKit is available at www.revkit.org.

[42] Mathias Soeken, Robert Wille, and Rolf Drechsler. Hierarchical Synthesis of Reversible Circuits using Positive and Negative Davio Decomposition. In *Design and Test Workshop (IDT), 2010 5th International*, pages 143–148. IEEE, 2010.

[43] Fabio Somenzi. CUDD: CU Decision Diagram package release 2.3. 0. `http://vlsi.colorado.edu/~fabio/CUDD/`, 1998.

[44] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[45] Robert Wille and Rolf Drechsler. Bdd-based Synthesis of Reversible Logic for Large Functions. In *Proceedings of the 46th Annual Design Automation Conference*, pages 270–275. ACM, 2009.

[46] Robert Wille and Rolf Drechsler. Embedding of Irreversible Functions. In *Towards a Design Flow for Reversible Logic*, pages 93–111. Springer, 2010.

[47] Robert Wille and Rolf Drechsler. Synthesis of Boolean Functions in Reversible Logic. *Synthesis Lectures on Digital Circuits and Systems*, pages 1079–1087, 2010.

[48] Robert Wille, Oliver Keszocze, and Rolf Drechsler. Determining the Minimal Number of Lines for Large Reversible Circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–4. IEEE, 2011.

[49] Robert Wille, Mathias Soeken, and Rolf Drechsler. Reducing the number of Lines in Reversible Circuits. In *Proceedings of the 47th Design Automation Conference*, pages 647–652. ACM, 2010.

[50] Robert Wille, Mathias Soeken, D Michael Miller, and Rolf Drechsler. Trading Off Circuit Lines and Gate Costs in the Synthesis of Reversible Logic. *Integration, the VLSI Journal*, 47(2):284–294, 2014.

[51] Jing Zhong and Jon C Muzio. Using Crosspoint Faults in Simplifying Toffoli Networks. In *Circuits and Systems, 2006 IEEE North-East Workshop on*, pages 129–132. IEEE, 2006.

[52] LLC Ziff Davis. Computing Commercial Quantum Computer. http://www.extremetech.com/computing/84228-first-ever-commercial-quantum-computer-now-available-for-10-million, 2011. [Online; accessed 19-Nov-2014].