

Chapter 2

Background

To keep the book self-contained, this chapter briefly introduces the basics on reversible logic, quantum circuits, and other required concepts. The chapter consists of seven parts: the first section introduces the basic definitions and notations of Boolean functions while the second section reviews existing Boolean decomposition. The third and the fourth sections give an overview of exclusive sum of products and Boolean satisfiability, respectively. The fifth section provides a summary on the principles of reversible logic required later in this book. Similarly, the sixth section gives an introduction on quantum computation and quantum circuits. The later section details the different metrics used as a quality-measure of reversible and quantum realizations. Finally, decision diagrams are reviewed.

2.1 Boolean Functions

A *Boolean function* is defined as follows.

Definition 2.1 (Boolean Function). Let $\mathbb{B} \stackrel{\text{def}}{=} \{0, 1\}$ denote the *Boolean values*. Then we refer to

$$\mathcal{B}_{n,m} \stackrel{\text{def}}{=} \{f \mid f: \mathbb{B}^n \rightarrow \mathbb{B}^m\}$$

as the set of all *Boolean functions* with n inputs and m outputs. There are 2^{m2^n} such Boolean functions. We write $\mathcal{B}_n \stackrel{\text{def}}{=} \mathcal{B}_{n,1}$ to denote the set of Boolean 1-output functions.

Assume that each multiple-output function $f \in \mathcal{B}_{n,m}$ is represented as a tuple $f = (f_1, \dots, f_m)$ where $f_i \in \mathcal{B}_n$ for each $i \in \{1, \dots, m\}$ and hence $f(X) = (f_1(X), \dots, f_m(X))$ for each $X \in \mathbb{B}^n$. The functions $f_i(X)$ are called *primary outputs*.

Fig. 2.1 Boolean functions.(a) ($\vee$) Function. (b) 1-bit full adder function

a			b				
x_1	x_2	y	x_1	x_2	C_{in}	C_{out}	S
0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	1
1	0	1	0	1	0	0	1
1	1	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

The behavior of a Boolean function can be specified by enumerating the outputs respond to the inputs values. This input/output relationship is commonly enumerated in a tabular form, called a *truth table* in which all permutations of the inputs are listed on the left, and the outputs of the function are listed on the right.

Example 2.1. Figure 2.1a shows the truth table of a function with 2 inputs and 1 output. This function represents the OR function ($\vee$). Figure 2.1b presents the truth table of a multi-output function with 3 inputs and 2 outputs. This specification presents the behavior of the 1-bit full adder.

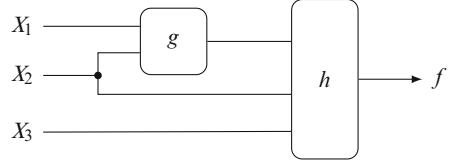
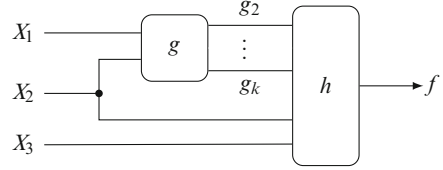
2.2 Boolean Function Decomposition

Boolean function decomposition describes the problem of expressing a Boolean function of n variables as a functionally equivalent composition of simpler Boolean functions of less than n variables. Several types of Boolean function decomposition that have been proposed in the last decades are outlined in the following.

2.2.1 Ashenhurst Decomposition

When a function $f \in \mathcal{B}_n$ can be decomposed (see Fig. 2.2) into

$$f(X) = h(g(X_1, X_2), X_2, X_3)$$

Fig. 2.2 Ashenhurst decomposition**Fig. 2.3** Curtis decomposition

with $g \in \mathcal{B}_{|X_1|+|X_2|}$, $h \in \mathcal{B}_{|X_2|+|X_3|+1}$, $X = X_1 \cup X_2 \cup X_3$, and $X_1 \cap X_2 = X_1 \cap X_3 = X_2 \cap X_3 = \emptyset$, then it is referred to as *Ashenhurst decomposition*. If $X_2 = \emptyset$, then the decomposition is called *disjoint*, otherwise it is called a *non-disjoint* decomposition. The set X_1 is called *bound set* and the set X_3 is called *free set*.

Example 2.2 ([155]). Consider the Boolean function

$$f = x_1\bar{x}_2 \vee x_1x_3 \vee x_1\bar{x}_4 \vee \bar{x}_2\bar{x}_3x_4$$

f can be decomposed using the Ashenhurst decomposition. One possible disjoint decomposition of f is as follows.

$$f(x_1, x_2, x_3, x_4) = h(x_1, x_2, g)$$

$$g(x_3, x_4) = \bar{x}_3x_4$$

$$h(x_1, x_2, g) = \bar{x}_2g \vee x_1\bar{g}$$

2.2.2 Curtis Decomposition

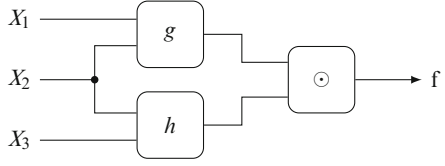
A generalization of the Ashenhurst decomposition was suggested by Curtis [31] where g can have multiple outputs (see Fig. 2.3), i.e.,

$$f(X) = h(g_1(X_{1_1}), g_2(X_{1_2}), \dots, g_k(X_k), X_2, X_3)$$

with $g_i \in \mathcal{B}_{|X_{j_i}|}$ (where $i = 1 \dots k$ and $j = 1, 2$), $h \in \mathcal{B}_{|X_2|+|X_3|+k}$, $X = X_1 \cup X_2 \cup X_3$, and $X_1 \cap X_2 \cap X_3 = \emptyset$.

Example 2.3 ([155]). Consider the Boolean function

$$f = \bar{x}_1x_2x_3x_4\bar{x}_5 \vee \bar{x}_1\bar{x}_4(\bar{x}_2 \vee \bar{x}_3) \vee x_1x_2x_3x_4 \vee x_1x_4(\bar{x}_2 \vee \bar{x}_3) \vee x_1x_5 \vee \bar{x}_4x_5.$$

Fig. 2.4 Bi-decomposition

One possible disjoint Curtis decomposition associated with f is as follows.

$$f(x_1, x_2, x_4, x_5) = h(g_1, g_2, x_4, x_5)$$

$$g_1(x_1, x_2, x_3) = x_1$$

$$g_2(x_1, x_2, x_3) = x_1 \bar{x}_2 \vee x_2 x_3$$

$$h(g_1, g_2, x_4, x_5) = \bar{g}_1 \bar{g}_2 \bar{x}_4 \vee \bar{g}_1 g_2 (x_4 \oplus x_5) \vee g_1 \bar{g}_2 x_4 \vee g_1 g_2 \bar{x}_4$$

2.2.3 Bi-decomposition

When a Boolean function can be decomposed into two sub-functions [117] (see Fig. 2.4) $f(X) = g(X_1) \odot h(X_2)$, with $g \in \mathcal{B}_{|X_1|}$, $h \in \mathcal{B}_{|X_2|}$, $X = X_1 \cup X_2$, and $X_1 \cap X_2 = \emptyset$, then the decomposition is called *bi-decomposition*. The ‘ $\odot$ ’ can be any binary Boolean operation (typically $\vee$, $\wedge$, $\oplus$, or $\otimes$). This decomposition is also known as the *simple decomposition*.

Example 2.4 ([155]). Consider the Boolean function

$$f = \bar{x}_1 x_2 \bar{x}_3 \vee x_1 x_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3.$$

f can be decomposed into the following non-disjoint bi-decomposition:

$$f(x_1, x_2, x_3) = g(x_1, x_2, x_3) \wedge h(x_2, x_3)$$

$$g(x_1, x_2, x_3) = x_1 x_3 \vee x_2$$

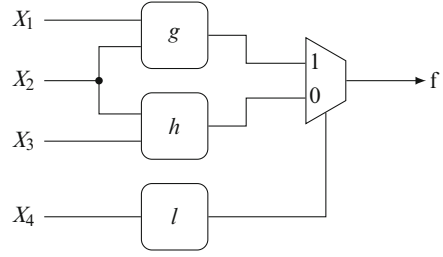
$$h(x_2, x_3) = \bar{x}_2 \vee \bar{x}_3$$

2.2.4 Multiplexer Decomposition

If a Boolean function can be decomposed into three sub-functions (see Fig. 2.5)

$$f(X) = l(X_4) \wedge g(X_1, X_3) \oplus \bar{l}(X_4) \wedge h(X_2, X_3)$$

Fig. 2.5 Multiplexer decomposition



with $g \in \mathcal{B}_{|X_1|+|X_2|}$, $h \in \mathcal{B}_{|X_2|+|X_3|}$, $l \in \mathcal{B}_{|X_4|}$, $X = X_1 \cup X_2 \cup X_3 \cup X_4$, and $X_1 \cap X_2 = X_1 \cap X_3 = X_2 \cap X_3 = \emptyset$, then it is referred to as *multiplexer decomposition*.

Example 2.5 ([155]). Consider the Boolean function $f(x_1, x_2, x_3) = \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3$. In the following, a possible multiplexer decomposition for f is given:

$$\begin{aligned} f(x_1, x_2, x_3) &= l(x_1)g(x_2, x_3) \oplus \bar{l}(x_1)h(x_2, x_3) \\ h(x_2, x_3) &= x_2 x_3 \\ g(x_2, x_3) &= x_2 \oplus x_3 \\ l(x_1) &= x_1 \end{aligned}$$

2.3 Exclusive-OR Sum of Products

Exclusive-OR sum-of-products (ESOPs, [113])

$$f = \bigoplus_{i=1}^k x_{i_1}^{p_{i_1}} \wedge \cdots \wedge x_{i_{l_i}}^{p_{i_{l_i}}} \quad (2.1)$$

are two-level descriptions for Boolean functions in which a function is composed of k *product terms* that are combined using the exclusive-OR (EXOR, $\oplus$) operation. A product term is the conjunction of l_i literals where a *literal* is either a propositional variable $x^1 = x$ or its negation $x^0 = \bar{x}$. ESOPs are the most general form of two-level AND-EXOR expressions.

Each n -variable function has an infinite number of ESOP representations [113]. As a result, many heuristic minimization approaches have been proposed (e.g., Exmin [116] and EXORCISM [93]) to find the ESOP expression with a small number of product terms. In addition, exact ESOP minimization algorithms have been developed (e.g., [60, 114, 115, 137]), which are able to find the ESOP expression with the smallest number of product terms. However, they require a large computation time and are therefore only applicable to small functions.

Several restricted subclasses have been considered in the past, e.g., *positive polarity Reed-Muller expressions* (PPRM [113]), in which all literals are positive, or *fixed polarity Reed-Muller expressions* (FPRMs, [113]), in which each variable must have the same polarity in all product terms. There are further subclasses (e.g., *kronecker expressions* (KROs) [34] and *pseudo kronecker expressions* [71] (PSDKROs)) and most of them can be defined based on applying the following decomposition rules. An arbitrary Boolean function $f(x_1, x_2, \dots, x_n)$ can be expanded as

$$f = \bar{x}_i f_{\bar{x}_i} \oplus x_i f_{x_i} \quad (\text{Shannon})$$

$$f = \bar{f}_{\bar{x}_i} \oplus x_i (\bar{f}_{\bar{x}_i} \oplus f_{x_i}) \quad (\text{positive Davio})$$

$$f = f_{x_i} \oplus \bar{x}_i (\bar{f}_{\bar{x}_i} \oplus f_{x_i}) \quad (\text{negative Davio})$$

with *co-factors* $\bar{f}_{\bar{x}_i} = f(x_1, \dots, x_{i-1}, 0, \dots, x_n)$ and $f_{x_i} = f(x_1, \dots, x_{i-1}, 1, \dots, x_n)$.

The inclusion relationships between the subclasses of AND-EXOR expressions can be stated as the following: $PPRM \subseteq FPRM \subseteq KRO \subseteq PSDKRO \subseteq ESOP$.

Example 2.6. Consider the truth table of the 1-bit full adder specified in Fig. 2.1b, the output carry function is defined as follows.

$$C_{\text{out}}(x_0, x_1, C_{\text{in}}) = x_0 C_{\text{in}} \vee x_1 C_{\text{in}} \vee x_0 x_1.$$

The PPRM representation is obtained by applying only the positive Davio decomposition for each variable.

$$C_{\text{out}}(x_0, x_1, C_{\text{in}}) = x_0 C_{\text{in}} \oplus x_1 C_{\text{in}} \oplus x_0 x_1 C_{\text{in}} \oplus x_0 x_1.$$

The FPRM representation is obtained by applying either the positive or the negative Davio decomposition for each variable. In the following, we apply the positive Davio decomposition to the variables x_0 and C_{in} while for the variable x_1 , we apply the negative Davio decomposition.

$$C_{\text{out}}(x_0, x_1, C_{\text{in}}) = C_{\text{in}} \oplus x_0 \oplus x_0 \bar{x}_1 \oplus C_{\text{in}} \wedge \bar{x}_1 \oplus C_{\text{in}} \wedge x_0 \bar{x}_1.$$

The optimal ESOP expression with minimal product terms is either

$$C_{\text{out}}(x_0, x_1, C_{\text{in}}) = C_{\text{in}} x_0 \oplus \bar{C}_{\text{in}} x_1 \oplus \bar{x}_0 x_1$$

or

$$C_{\text{out}}(x_0, x_1, C_{\text{in}}) = C_{\text{in}} x_0 \oplus C_{\text{in}} x_1 \oplus x_0 x_1.$$

2.4 Boolean Satisfiability and SAT Modulo Theory

The *conjunctive normal form* (CNF) consists of a conjunction of clauses. A clause is a disjunction of literals and each literal is a propositional variable or its negation.

The *Boolean satisfiability* (SAT) problem is defined as follows.

Definition 2.2 (Boolean Satisfiability). Let h be a Boolean function in CNF, i.e., a product-of-sum representation. Then, the SAT problem is to find an assignment for the variables of h such that h evaluates to 1 or to prove that such an assignment does not exist.

Example 2.7. Let $F = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_4)$, then $x_1 = 1$, $x_2 = 0$, $x_3 = 0$, and $x_4 = 0$ is a satisfying assignment for F . The value of x_1 ensures that the first two clauses become satisfied while the value of x_2 ensures that the remaining clauses become satisfied.

SAT is one of the central $\mathcal{NP}$ -complete problems. It was the first known $\mathcal{NP}$ -complete problem that was proven by Cook in 1971 [30]. Despite this proven complexity today there are SAT algorithms which solve many practical problem instances very fast. Application domains are for example automatic test pattern generation [70, 123], logic synthesis [156], diagnosis [126], and verification [19, 29, 102]. For SAT solving several (backtracking) algorithms have been proposed [35, 36, 45, 72, 95].

The basic search procedure to find a satisfying assignment is shown in Fig. 2.6 and follows the structure of the *Davis Putnam Logemann Loveland* (DPLL) algorithm [35, 36]. The description follows the implementation of the procedure in modern SAT solvers. While there are free variables left (c), a decision is made (e) to assign a value to one of these variables. Then, implications are determined due to the last assignment by BCP (f). This may cause a conflict (g) that is analyzed. If the conflict can be resolved by undoing assignments from previous decisions, backtracking is done (h). Otherwise, the instance is unsatisfiable (i). If no further decision can be done, i.e., a value is assigned to all variables and this assignment did not cause a conflict, the CNF is satisfied (d).

Due to the tremendous improvements in the recent past, several researchers investigated the combination of SAT solvers with decision procedures for decidable theories resulting in *SAT modulo theories* (SMT) [21, 44]. An SMT solver integrates a Boolean SAT solver with other solvers for specialized theories. SMT is about checking the satisfiability of first-order formulas containing operations from various theories such as bit-vectors, arithmetic, arrays, uninterpreted functions, and recursive data-types. Here the SAT solver works on an abstract representation (also in CNF) of the problem and steers the overall search process, while each (partial) assignment of this representation has to be validated by the theory solver for the theory constraints. Thus, advanced SAT techniques together with specialized theory solvers can be utilized.

SMT solvers are available to handle complex formulas such as Z3 [37], MathSAT [24], Boolector [23], and Yices [44].

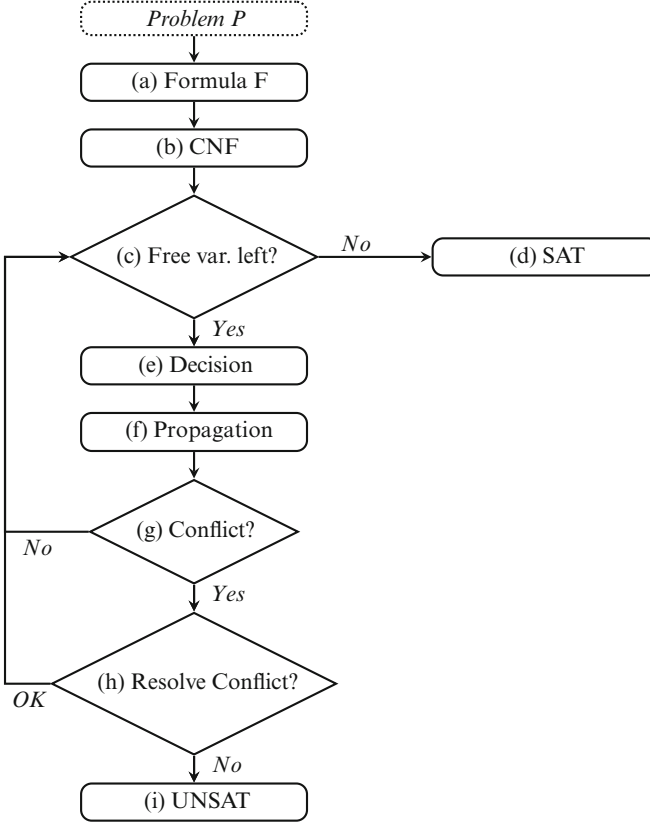


Fig. 2.6 DPLL algorithm in modern SAT solvers

Example 2.8. Let $F = (x_2 + x_3 = -1) \wedge (x_3 > 3x_4) \wedge (x_1 + x_3 < 1) \wedge (x_1 + x_2 > 0)$ be a conjunction of four clauses that contain formulas over theories. Then, $x_1 = 1$, $x_2 = 1$, $x_3 = -2$, and $x_4 = -1$ is a satisfying assignment for F . The values of x_2 and x_3 ensure that the first clause becomes satisfied while x_4 ensures this for the second clause and the assigned value of x_1 asserts the satisfiability of the remaining two clauses.

2.5 Reversible Logic

In contrast to classical design, reversible computation is restricted to operations which can be performed in an invertible fashion. This reversibility builds the basis for many emerging technologies that may replace or at least enhance classical computer chips. Examples include applications in the domain of

Fig. 2.7 Examples of Boolean functions. (a) f_1 : Irreversible function. (b) f_2 : Reversible function

a							b						
x_1	x_2	x_3	y_1	y_2	y_3		x_1	x_2	x_3	y_1	y_2	y_3	
0	0	0	0	0	0		0	0	0	0	0	0	
0	0	1	1	0	0		0	0	1	0	1	0	
0	1	0	0	1	0		0	1	0	1	0	0	
0	1	1	0	1	1		0	1	1	1	0	1	
1	0	0	0	1	1		1	0	0	0	0	1	
1	0	1	1	0	1		1	0	1	0	1	1	
1	1	0	1	1	1		1	1	0	1	1	0	
1	1	1	1	1	0		1	1	1	1	1	1	

- *Low power computation*, where the fact that no information is lost in reversible computation can be exploited (cf. [17, 18, 69]),
- *Adiabatic circuits*, a special low power technology that reversible circuits are particularly suited for (cf. [100]), and
- *Quantum computation*, which enables the solution of many relevant problems significantly faster than with classical circuits (cf. [98]).

To let the reader be familiar with the concept of reversibility, we introduce in the following the basics on reversible logic. The first part presents the properties of reversible functions. The second part defines the common reversible gates that are used in this book. Finally, reversible circuits and their characteristics are given with examples.

2.5.1 Reversible Function

Definition 2.3 (Reversible Function). A function $f \in \mathcal{B}_{n,n}$ is called *reversible* if f is bijective, i.e., if each input pattern is uniquely mapped to a corresponding output pattern and vice versa. Otherwise, it is called *irreversible*. Clearly, if f is reversible, then its number of inputs is equal to the number of outputs.

In other words, each reversible function $f \in \mathcal{B}_{n,n}$ is a bijection that performs a permutation of the set of input patterns. Therefore, a reversible function $f \in \mathcal{B}_{n,n}$ can be represented by a *permutation* of $\{0, 1, \dots, 2^n - 1\}$, i.e.

$$\pi_f \stackrel{\text{def}}{=} (\text{nat}(f(0, \dots, 0, 0)), \dots, \text{nat}(f(1, \dots, 1, 1))) \quad (2.2)$$

where $\text{nat} : \mathbb{B}^n \rightarrow \{0, 1, \dots, 2^n - 1\}$ maps a bit-vector to its natural number representation. Further, f can be represented as a $2^n \times 2^n$ Boolean *permutation matrix* Π_f where

$$(\Pi_f)_{r,c} \stackrel{\text{def}}{=} (\pi_f(c) \equiv r). \quad (2.3)$$

Example 2.9. The function f_1 presented in Fig. 2.7a is irreversible. Despite that the number of inputs is equal to the number of outputs, there is no unique input-output mapping (e.g., both inputs 011 and 100 map to the same output 011). In contrast, the function f_2 outlined in Fig. 2.7b is reversible, since each input pattern maps to a unique output pattern and the number of inputs is equal to the number of outputs.

The output permutation after applying f_2 is computed in the following:

$$\begin{aligned}\pi_{f_2} &= (\text{nat}(f(000)), \text{nat}(f(001)), \dots, \text{nat}(f(110)), \text{nat}(f(111))) \\ &= (\text{nat}(000), \text{nat}(010), \dots, \text{nat}(110), \text{nat}(111)) \\ &= (0, 2, 4, 5, 1, 3, 6, 7)\end{aligned}$$

The permutation matrix is in the following:

$$\Pi_{f_2} = \begin{array}{c} \begin{array}{c} x_3/y_3 \\ x_2/y_2 \\ x_1/y_1 \end{array} \begin{array}{c} \text{Inputs} \\ 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \end{array} \left[\begin{array}{c} \text{Outputs} \\ 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \right]$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

An irreversible function can be embedded into a reversible specification by adding extra variables to achieve a bijective function. An embedding is not unique and the choice of embedding can have a very significant effect on the number of the variables of the resulting function [89, 136].

Assume $f = (f_1, \dots, f_m) \in \mathcal{B}_{n,m}$ and $g = (g_1, \dots, g_{m'}) \in \mathcal{B}_{n,m'}$, where $m' \geq m$. We write $f = g|_m$ in case $f_i(X) = g_i(X)$ for each $X \in \mathbb{B}^n$ and each $i \in \{1, \dots, m\}$ and say that f is the m -projection of g .

Definition 2.4 (Embedding [136]). A function $g \in \mathcal{B}_{n,m+k}$ embeds $f \in \mathcal{B}_{n,m}$, if g is injective and $f \equiv g|_m$. The function g is called an *embedding* and the additional k outputs of g are referred to as **garbage outputs**. Furthermore, c additional input variables might be added such that $n + c = m + k$ in order to obtain a reversible function for the embedding g . The additional c inputs are referred to as **constant inputs**.

More precisely, given an m -output irreversible function f on n variables, a reversible function g with $m + k$ outputs is determined such that g agrees with f on the first m components. Then, bijectivity can readily be achieved, e.g., by adding additional inputs such that f evaluates to its original values in case these inputs are

assigned the constant value 0 and each output pattern that is not in the image of g is arbitrarily distributed among the new input patterns.

Let $\mu(f)$ denote the number of occurrences of the most frequent output pattern in the truth table of a . Then, $\ell(f) \stackrel{\text{def}}{=} \lceil \log_2 \mu(f) \rceil$ is the minimum number of garbage outputs (denoted by k) required to convert an irreversible function to a reversible function. Thus, if $k = \ell(f)$, then the embedding g is called *optimal*. Different algorithms that perform an embedding of irreversible functions based on their truth table description have been proposed in the past, e.g., [76, 89, 136].

Example 2.10. The 1-bit full adder specified in Fig. 2.8a is a Boolean function $f \in \mathcal{B}_{3,2}$. It can be embedded into a function $g \in \mathcal{B}_{3,4}$ which is illustrated in Fig. 2.8b. The most frequent output pattern in f is 01. This pattern is repeated three times in rows 2, 3, and 5. Then, $\mu(f) = 3$. As can be seen, the number of additional garbage outputs are $k = 2$ and $\ell(f) = \lceil \log_2 \mu(f) \rceil = 2 = k$, hence the embedding g is optimal. Two additional constant inputs are added to ensure the reversibility of the final embedded functional, i.e., the same number of inputs and outputs. The result of adding a constant input for the embedding function g shown in Fig. 2.8b is given in Fig. 2.8c.

a

x_1	x_2	C_{in}	C_{out}	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

b

x_1	x_2	C_{in}	C_{out}	S	g_1	g_2
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	1	0	0	1
1	0	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	0	1	0
1	1	1	1	1	1	0

c

C	x_1	x_2	C_{in}	C_{out}	S	g_1	g_2
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	1
0	1	0	0	0	1	1	1
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	1	1	0
1	0	0	0	1	0	0	0
1	0	0	1	1	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	0	0	1
1	1	0	0	1	1	1	1
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	1	1	0

Fig. 2.8 Embedding of the 1-bit full adder function. (a) 1-bit full adder function. (b) Garbage outputs. (c) Constant input

2.5.2 Reversible Gates

Reversible functions in $\mathcal{B}_{n,n}$ can be realized by reversible circuits that consist of at least n lines and are constructed as circuits of reversible gates that belong to a certain gate library. Prominent reversible logic gates are Toffoli [139], Fredkin [49], and Peres [101]. In the case where a reversible gate library contains a smallest complete set of reversible gates which can be used to implement any reversible function, the library is called a *universal* gate library. The most common universal gate library consists of *Toffoli gates* [139], and *single-target gates* [40, 140]. In the following, we define each of these libraries.

Definition 2.5 (Single-Target Gate). Given a set of variables $X = \{x_1, \dots, x_n\}$, a *single-target* (ST) gate $T_g(C, t)$ with *control lines* $C = \{x_{i_1}, \dots, x_{i_k}\} \subset X$, a *target line* $t \in X \setminus C$, and a *control function* $g \in \mathcal{B}_k$ inverts the variable on the target line, if and only if $g(x_{i_1}, \dots, x_{i_k})$ evaluates to true. All other variables remain unchanged. If the definition of g is obvious from the context, it can be omitted from the notation $T_g(C, t)$.

Example 2.11. Figure 2.9 shows the diagrammatic representation of a single-target gate based on Feynman's notation. The target line is denoted with the symbol $\oplus$.

Given n Boolean variables, the maximum number of variables that can be involved in a controlling function of an ST gate is $n - 1$ and the remaining variable would be used to compute the target for that gate. A controlling function g of an ST gate denotes an arbitrary Boolean function of $n - 1$ Boolean variables. It is well known that the number of possible Boolean functions with $n - 1$ variables is $2^{2^{n-1}}$, hence with n different variables taken each as a target line we get $n \cdot 2^{2^{n-1}}$ possible ST gates as depicted in Table 2.1.

Definition 2.6 (Toffoli Gate). *Mixed-polarity multiple-control Toffoli* (MPMCT) gates are a subset of the single-target gates in which the control function g can be represented with one product term $g : C = (x_{i_1}, x_{i_2}, \dots, x_{i_k}) \mapsto \bigwedge_{k=i}^j x_i^p$. *Multiple-control Toffoli* gates (MCT) are a subset from MPMCT gates in which the product terms can only consist of positive literals.

For drawing circuits, we follow the established conventions of using the symbol solid black circles ($\bullet$) to indicate positive controls and white circles ($\circ$) to indicate

Fig. 2.9 Single-target gate representation

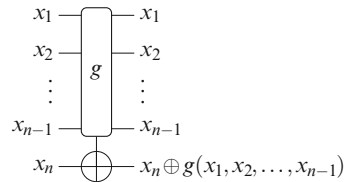
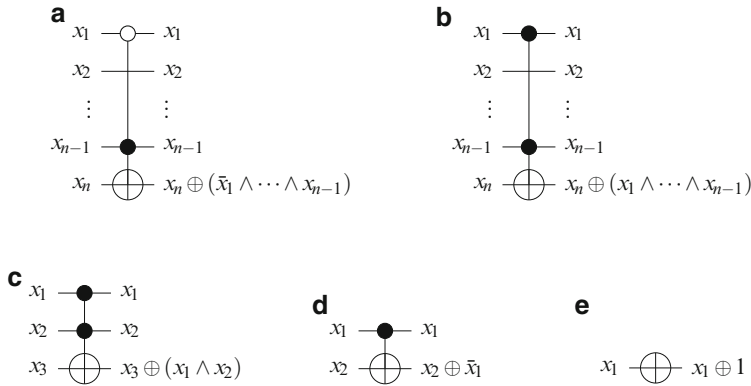


Table 2.1 Number of reversible gates using n lines

Gate type	Formula	Number of gates
NOT	$n \cdot \binom{n-1}{0}$	n
CNOT	$n \cdot \binom{n-1}{1}$	$n \cdot (n-1)$
Toffoli	$n \cdot \binom{n-1}{2}$	$\frac{n \cdot (n-1) \cdot (n-2)}{2}$
NCT	$n \cdot \sum_{i=0}^2 \binom{n-1}{i}$	$n \cdot \left(\frac{n \cdot (n-1)}{2} + 1 \right)$
MCT	$n \cdot \sum_{i=0}^{n-1} \binom{n-1}{i}$	$n \cdot 2^{n-1}$
MPMCT	$n \cdot \sum_{i=1}^{n-1} \binom{n-1}{i} \cdot 2^i$	$n \cdot 3^{n-1}$
ST	$n \cdot \mathcal{B}_{n-1} $	$n \cdot 2^{2^{n-1}}$
Arbitrary	$P_{2^n}^{2^n}$	$(2^n)!$

**Fig. 2.10** Toffoli library. (a) MPMCT gate. (b) MCT gate. (c) Toffoli gate. (d) CNOT gate. (e) NOT gate

negative controls. Also, MPMCT gates can be represented as a diagrammatic representation as a single-target gate representation.

Example 2.12. Figure 2.10a shows an MPMCT gate with mixed polarity control lines over the set of variables $X = \{x_1, \dots, x_n\}$, while Fig. 2.10b shows an MCT gate with positive control lines. The remaining figures present the Toffoli gate, the CNOT gate, and the NOT gate.

An MCT gate with two controls is called *Toffoli* gate, while when it has only one control it is called a *CNOT* gate, and in case it has no control it is named a *NOT* gate. When a library contains only gates from the gate set type $\{\text{NOT}, \text{CNOT}, \text{Toffoli}\}$, then this library is referred as *NCT library*. It is called the *MCT library* when only MCT gates are used, otherwise it is called the *MPMCT library*.

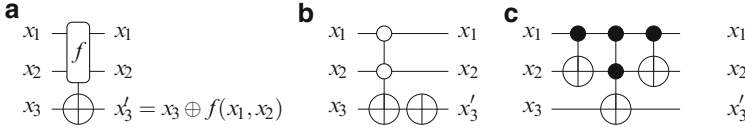


Fig. 2.11 ST gate with a controlling function (∇) given by Fig. 2.1b. (a) ST gate. (b) MPMCT circuit. (c) MCT circuit gate

With n different variables, the total number of the conceivable NOT gates is equal to n as depicted in Table 2.1 since each variable can have a NOT gate. Furthermore, the number of all the possible CNOT and Toffoli gates are $n \cdot (n - 1)$ and $n \cdot \binom{n-1}{2}$, respectively. This is calculated as the following: in a set X , the total number of subset having k elements is the sum of all the combinations with k elements from that set. If the set has $n - 1$ elements, the number of k -combinations is equal to the binomial coefficient $\binom{n-1}{k}$. Assuming that we have as a starting point n different sets, hence we have n multiplied by the total number of combinations. Therefore the total number of gates that an NCT library and MCT library is the total number of binomial coefficients with $k = 0, 1, 2$ and $k = 0, 1, 2, \dots, n - 1$, respectively where k denotes the number of controls. Consider the possible number of MPMCT gates in an MPMCT library over n variables. It is the same as the number of MCT gates except that we add the possibility of negative controls. Hence, the number of MPMCT gates is $n \cdot \sum_{i=1}^{n-1} \binom{n-1}{i} \cdot 2^i$ and given $\sum_{i=1}^n \binom{n}{i} \cdot x^i \cdot y^{n-i} = (x + y)^n$, one can deduct that this the total number of possible MPMCT gates is $n \cdot 3^{n-1}$.

Since its controlling function is Boolean, an ST gate can be expressed in terms of a circuit of MPMCT or MCT gates, which can be obtained from an ESOP or PPRM expression [113], respectively.

Example 2.13. Figure 2.11a shows a single-target gate. The controlling function of this gate is the (∇) function specified in Fig. 2.1b. The equivalent MPMCT gate circuit is given in Fig. 2.11b, while Fig. 2.11c shows its correspondent MCT gate circuit.

2.5.3 Reversible Circuits

A reversible circuit is defined as follows.

Definition 2.7 (Reversible Circuit [107]). A combinational reversible circuit is an acyclic combinational logic circuit in which all gates are reversible, and are interconnected without explicit fan-outs and loops.

Boolean functions can be synthesized to a reversible circuit after embedding them to reversible functions. Therefore, in general a reversible circuit contains n inputs with p primary inputs and c constant inputs with $p + c = n$. At the output side, there are m primary outputs and k garbage outputs with $k + m = n$. Figure 2.12

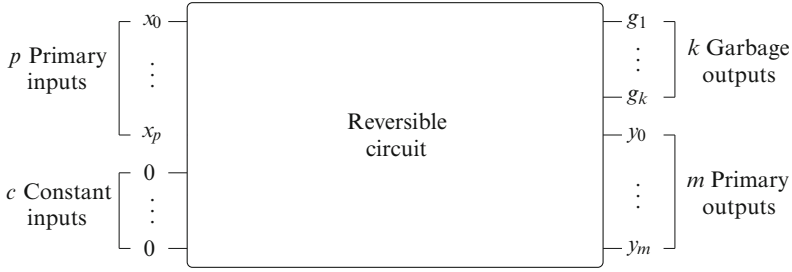


Fig. 2.12 Reversible circuit structure inspired from [107]

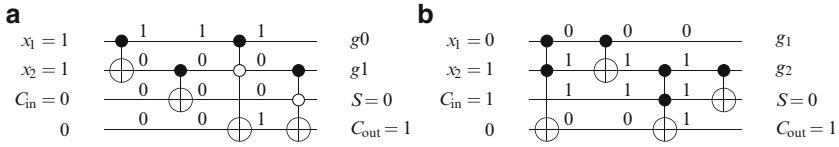


Fig. 2.13 Reversible circuits for 1-bit full adder as specified in Fig. 2.8. (a) MPMCT library based circuit. (b) MCT library based circuit

depicts the general structure of a reversible circuit. Note that when the function is bijective, there are neither constant inputs nor garbage outputs. This follows from [122], where it has been shown that any reversible function $f \in \mathcal{B}_{n,n}$ can be realized by a reversible circuit with n lines when using MCT gates. This means that it is not necessary to add any temporary lines (referred to as *ancilla*) to realize the circuit.

Definition 2.8 (Ancilla). Let G be a reversible or quantum circuit. An *ancilla* is an additional line whose input is arbitrary or set to a constant value and is used in a way throughout the circuit to compute intermediate results.

Reversible functions can be realized by reversible circuits that consist of at least n lines and are constructed as circuits of reversible gates that belong to a certain gate library (e.g., NCT, MCT, or MPMCT library [139]).

Example 2.14. Figure 2.13a shows different MPMCT gates in a circuit forming a reversible circuit that realizes the embedded 1-bit full adder specified in Fig. 2.8b. The annotated values demonstrate the computation of the gates for a given input assignment, while Fig. 2.13b depicts the equivalent circuit based on MCT gates.

2.6 Quantum Computation

In this section, we present briefly the fundamental concepts of quantum states and systems. Then, we introduce quantum libraries and several examples of quantum physical realizations.

2.6.1 Quantum Systems

Quantum computation [98] provides a new paradigm of computation based on the so-called *qubits* instead of bits. In contrast to Boolean logic, *qubits* do not only represent Boolean 0's and Boolean 1's, but also the superposition of both.

Definition 2.9 (Qubit). A *qubit* $|\varphi\rangle$ (bra-ket notation introduced by Paul Dirac) is a vector $\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$ where $a_0, a_1 \in \mathbb{C}$ such that

$$|a_0|^2 + |a_1|^2 = 1. \quad (2.4)$$

If $a_0 = 1$, then $|\varphi\rangle$ represents the classical 0, denoted $|0\rangle$ with $|0\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, and if $a_1 = 1$, then $|\varphi\rangle$ represents the classical 1, denoted $|1\rangle$, with $|1\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Any state of a *qubit* may also be written as

$$|\varphi\rangle = a_0|0\rangle + a_1|1\rangle. \quad (2.5)$$

Using quantum computation and *qubits* in superposition, functions can be evaluated with different possible input assignments in parallel. It is not possible to obtain the current state of a *qubit*. Instead, if a *qubit* is measured, either 0 or 1 is returned depending on a respective probability, i.e., 0 is returned with a probability of $|a_0|^2$ and 1 is returned with a probability of $|a_1|^2$. After the measurement, the state of the qubit is destroyed.

The state of a quantum system with $n \geq 1$ qubits is given by an element of the tensor product of the single qubit spaces, i.e., a linear combination of the 2^n tensor states $|0 \cdots 0\rangle, |0 \cdots 1\rangle, \dots$, and $|1 \cdots 1\rangle$, which are the tensor products of basis states. Consequently, a quantum state is represented as a normalized vector of length 2^n (called the state vector), whose components denote the amplitude for each tensor state. The state of an n -qubit quantum system can be represented as the following equation:

$$|\psi\rangle = \sum_{X \in \{0,1\}^n} a_X |X\rangle \quad (2.6)$$

with

$$\sum_{X \in \{0,1\}^n} |a_X|^2 = 1 \quad (2.7)$$

Example 2.15. Consider two states of a 1-qubit quantum system $|\psi\rangle = a|0\rangle + b|1\rangle$ and $|\phi\rangle = c|0\rangle + d|1\rangle$. The state space of the composite system $|\Phi\rangle$ is the tensor product of the state spaces of the component systems $|\psi\rangle$ and $|\phi\rangle$. Then:

$$\begin{aligned} |\Phi\rangle &= |\psi\rangle \otimes |\phi\rangle \\ &= (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle) \\ &= ac(|0\rangle \otimes |0\rangle) + ad(|0\rangle \otimes |1\rangle) + bc(|1\rangle \otimes |0\rangle) + bd(|1\rangle \otimes |1\rangle) \end{aligned}$$

with $|0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = |00\rangle$. Similarly calculated,

the tensor products $|0\rangle \otimes |1\rangle$, $|0\rangle \otimes |1\rangle$, and $|0\rangle \otimes |1\rangle$ are equal to $|01\rangle$, $|10\rangle$, and $|11\rangle$, respectively. Hence,

$$|\Phi\rangle = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle.$$

A state of a n -qubit quantum system is referred to as an entangled state when it cannot be written as any combination of the states of the individual qubits. In other words, the state cannot be represented as the tensor product of n single qubits.

Definition 2.10 (Entanglement). Let $|\psi\rangle$ denote a composite state. Then $|\psi\rangle$ is said to be entangled if $|\psi\rangle$ cannot be expressed in the form:

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \quad (2.8)$$

otherwise is said to be separable.

Example 2.16. The bell state [16] $|\beta\rangle = \frac{1}{\sqrt{2}}|00\rangle + |11\rangle$ is an entangled state. Also any state $|\phi\rangle = a|00\rangle + b|11\rangle$ is an entangled state provided that $a \neq 0$ and $b \neq 0$. While the state $|\psi\rangle = a|10\rangle + b|11\rangle$ is a separable state since $|\psi\rangle = a|10\rangle + b|11\rangle = |1\rangle \otimes (a|0\rangle + b|1\rangle)$.

Definition 2.11 (Unitary Matrix). A matrix U is *unitary* if $U^\dagger U = UU^\dagger = I$ where $U^\dagger = (U^*)^T$ is the conjugate transpose of U .

Example 2.17. Consider the matrix $U = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$.

Its conjugate transpose is $U^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$. Since $UU^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $U^\dagger U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, hence U is a unitary matrix.

Following the theory of quantum mechanics, the evolution of a quantum system due to a quantum operation can be described by a unitary transformation matrix U . Here, the columns correspond to the output state vectors that result when applying

the respective operation to the tensor states as inputs. Thus, the entry u_{ij} of the matrix describes the mapping from the input tensor state $|j_i\rangle$ to the output tensor state $|i_j\rangle$.

Example 2.18. Consider the bell state [16] which is a 2-qubit quantum system defined by $|\beta\rangle = \frac{1}{\sqrt{2}}|00\rangle + |11\rangle$. In the following, we determine the unitary transformation matrix U that represents the 2-qubit quantum system $|\beta\rangle$:

$$\begin{aligned}
 |\beta\rangle &= \frac{1}{\sqrt{2}}|00\rangle + |11\rangle \\
 &= \frac{1}{\sqrt{2}}|00\rangle + 0|01\rangle + 0|10\rangle + 1|11\rangle \\
 &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + 0 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + 0 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \text{ hence,} \\
 U &= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

2.6.2 Quantum Libraries

Since most of the known quantum algorithms include a large Boolean component (e.g., the database in Grover's search algorithm and the modulo exponentiation in Shor's algorithm), the design of these components is often conducted by a two-stage approach: (1) realizing the desired functionality as a reversible circuit and (2) mapping the resulting circuit to a functionally equivalent quantum circuit.

The *technology mapping* is the process where a reversible circuit is converted to a quantum circuit using a selected set of gates from a technology library. The technology mapping is also called decomposition or transformation. The currently known mappings of MCT/MPMCT gates into quantum circuits using ancillae have been introduced in [15, 91, 98]. In the scope of this book, we are considering circuits that realize pure Boolean functionality but still need to be realized using quantum gates in order to embed them into quantum algorithms such as Deutsch-Josza [41], Grover [56], or Shor [125, 141].

Definition 2.12 (Quantum Gate). In general, a quantum gate acting on n qubits represents a $2^n \times 2^n$ unitary matrix [98]. A *single qubit gate* $U(t)$ over the inputs $X = \{x_1, \dots, x_n\}$ consists of a single-target line $t \in X$, while a *two qubit gate* (also denoted as *single-control gate*) $U(c, t)$ comprises, in addition, a single control line $c \in X$ with $t \neq c$. Quantum gates are also called *elementary* gates.

Different libraries of quantum gates have thereby been introduced for this purpose. In the literature, the *NCV library* [15] is frequently applied. This library is defined in the following:

Definition 2.13 (NCV Library). We consider the gate library $\{NOT, CNOT, V, V^\dagger\}$ as a universal quantum gate set for implementing any reversible function. The V and $V^\dagger$ gate are both the square root of the $CNOT$ gate since two adjacent identical V , or $V^\dagger$, gates are equivalent to a $CNOT$ gate. The $V^\dagger$ gate is the inverse of the V gate (Table 2.2).

Table 2.2 Gate definitions

Type	Symbol	Matrix	Diagram
NOT	N	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	
CNOT	C	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	
Controlled V	V	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{bmatrix}$	
Controlled V ⁻¹	$V^\dagger$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \end{bmatrix}$	
Hadamard	H	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	
Z-gate	Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	
Phase	S	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	
Phase ⁻¹	$S^\dagger$	$\begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$	
T-gate	T	$\begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix}$	
T-gate ⁻¹	$T^\dagger$	$\begin{bmatrix} 1 & 0 \\ 0 & e^{-\frac{i\pi}{4}} \end{bmatrix}$	

Table 2.3 Possible qubit values for NCV gates

x	$NOT(x)$	$V(x)$	$V^+(x)$
0	1	v_0	v_1
v_0	v_1	1	0
1	0	v_1	v_0
v_1	v_0	0	1

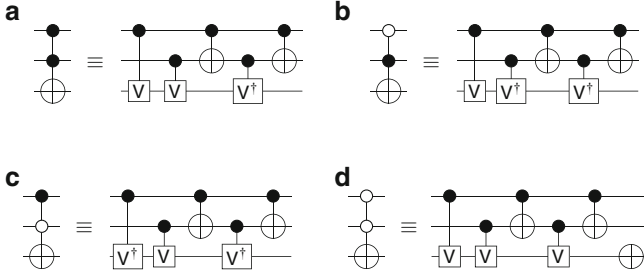


Fig. 2.14 NCV quantum mapping of different Toffoli gates. (a) Toffoli with two positive controls. (b) Toffoli with a negative and a positive control. (c) Toffoli with a positive and a negative control. (d) Toffoli with two negative controls

If circuits with Boolean inputs use NCV gates only, the value of each qubit at each stage of the circuit is restricted to one of $\{0, v_0, 1, v_1\}$, where $v_0 = \frac{1+i}{2} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $v_1 = \frac{1+i}{2} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$. The NOT , V , and V^+ operations over these four-valued logic are given in Table 2.3. As shown, NOT is a complement operation, V is a cycle, and V^+ is the inverse cycle.

Example 2.19. Figure 2.14 shows quantum circuits consisting of NCV gates. Each of these circuits depicts one of the minimal realizations of a Toffoli gate with respect to the polarity assignments for its controls.

Recently, there has been particular interest in quantum circuits composed of Clifford+ T gates. This is motivated by the importance of fault tolerance in quantum computations [144]. Unlike the NCV library, the Clifford+ T library is universal for quantum computation, i.e., every quantum system can be realized by it [22] as well as its gates can be implemented in a fault-tolerant way [26, 61].

Definition 2.14 (Clifford+ T Library). We consider the library $\{H, Z, S, T, CNOT\}$ as the universal gate set for implementing any quantum system. Note that the $S^\dagger$, $T^\dagger$, and NOT gates can be implemented with ZS , ZST , and HZH , respectively. The S and $S^\dagger$ gates are square roots of the Z gate. Similarly, the T and $T^\dagger$ gates are given by matrices that are the fourth root of the Z gate (See Table 2.2).

A single qubit gate $G(t)$ over the inputs $X = \{x_1, \dots, x_n\}$ consists of a single-target line $t \in X$, while a CNOT gate $G(c, t)$ comprises, in addition, a single control line $c \in X$ with $t \neq c$.

Fig. 2.15 Clifford + T quantum mapping of different Toffoli gates. **(a)** Toffoli with two positive controls. **(b)** Toffoli with a negative and a positive control. **(c)** Toffoli with a positive and a negative control. **(d)** Toffoli with a negative controls

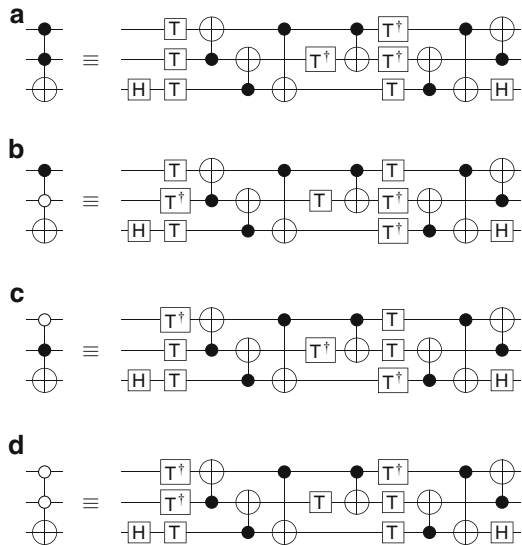


Table 2.4 Number of quantum gates using n lines

Gate/library type	Formula	Number of gates
Noncontrolled gate ^a	$n \cdot \binom{n-1}{0}$	n
Single-control gate ^a	$n \cdot \binom{n-1}{1}$	$n \cdot (n-1)$
NCV	$n \cdot \binom{0}{n-1} + 3n \cdot \binom{n-1}{1}$	$n \cdot (3n-2)$
Clifford + T	$6n \cdot \binom{0}{n-1} + n \cdot \binom{n-1}{1}$	$n \cdot (n+6)$
Arbitrary	∞	∞

^aNCV or Clifford + T gate

Example 2.20. Figure 2.15 shows the quantum realizations consisting of Clifford + T gates. These circuits represent the optimal realizations of Toffoli gates with all possible polarities that its controls might take.

The major cost-aware objective in Clifford + T circuits is to minimize the number of T gates and particularly the T -depth of the circuit due to the fact that the cost of the fault tolerant implementation of a T gate can exceed the cost of implementing a Clifford gate by a factor of 100 or more [10].

Table 2.4 outlines the total number of elementary gates that can be implemented over n variables. For example the NCV library has $n \cdot (3n - 2)$ different quantum gates while the Clifford + T library contains $n \cdot (n + 6)$ gates, where n is the number of possible qubits. The number of quantum arbitrary gates is infinite if we consider the infinite roots of the Z , NOT , or $CNOT$ gates.

2.6.3 Quantum Circuits

Definition 2.15 (Quantum Circuit). A quantum circuit is formulated as circuits of quantum gates over one or two qubits from a particular library. Unary quantum gates apply its operations at the respective qubit, while two-qubit gates have a *control qubit* in addition. The respective operation at the target qubit is performed if the control qubit is 1. A quantum circuit can be represented by a unitary matrix that is computed as the product of the matrices representing the individual gates.

A quantum circuit can be obtained through two different strategies: (1) a reversible circuit is synthesized, then each MPMCT gate is mapped to its quantum circuits [15, 91], or (2) direct synthesis finds the suitable circuit of quantum gates that can represent a given unitary matrix based on decomposition or exhaustive search. Many algorithms have been presented for NCV library [55, 82] as well as for the Clifford + T library [51, 65].

Example 2.21. The circuits depicted in Fig. 2.16a, b are the NCV and Clifford + T quantum circuits, respectively, for the reversible circuit shown in Fig. 2.13b realizing a 1-bit full adder.

When a resulting mapped circuit does not produce any entangled state, it is called a *Semi-Classical Quantum Circuit*.

Definition 2.16 (Semi-Classical Quantum Circuit). A Semi-Classical Quantum Circuit (SCQC) is a quantum circuit in which if all the initial input quantum states of the circuit are in the base states $|0\rangle$ or $|1\rangle$ (classical values), the quantum states of all gate controls in the circuit are also in the base states $|0\rangle$ or $|1\rangle$ [154]. Entanglement does not occur in SCQCs as long as their inputs are initialized to classical values.

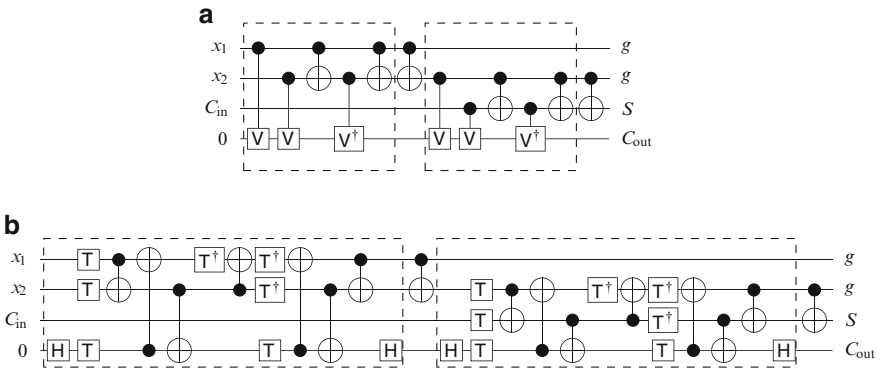


Fig. 2.16 Mapping of the 1-bit full adder reversible circuit in Fig. 2.13b. (a) NCV based quantum circuit. (b) Clifford + T based quantum circuit

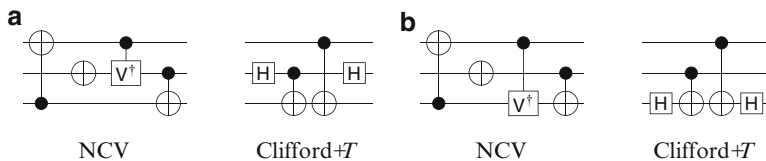


Fig. 2.17 Examples of entangled and SCQC quantum circuits. (a) Entangled circuits. (b) SCQC circuits

A quantum circuit is called an entangled circuit when it generates an entangled state for some given binary input state. If the state of a qubit is either v_0 or v_1 , and is applied to the control of a two-qubit gate, an entangled state may occur. Consequently, the resulting output state cannot be represented by the combination of individual qubit states.

Example 2.22. The circuits shown in Fig. 2.17a are entangled circuits. Consider the circuit in Fig. 2.17a. (1), it generates an entangled state for the input state vector $\langle 0, 0, 1 \rangle$ and the resulting output state vector cannot be separable into 3 single-qubit state vectors. The circuits sketched in Fig. 2.17b present two SCQC quantum circuits.

2.7 Cost Metrics for Reversible and Quantum Circuits

To compare the efficiency of different synthesis approaches it is important to evaluate the resulting circuits. Depending on the target application, different metrics are applied to measure the complexity of a given circuit.

2.7.1 Quantum Cost

The *quantum cost* denotes the required effort to transform a reversible circuit to a quantum circuit. The quantum cost depends on the used quantum library, and it is denoted as NCV quantum cost (*NCV-cost*) when the NCV library is used and *T-depth* when the Clifford+*T* library is used.

Definition 2.17 (NCV-Cost). The *NCV-cost* of a quantum circuit is the total number of quantum gates that form the circuit, while it denotes the number of required elementary gates to decompose that circuit to a quantum circuit.

Example 2.23. Consider the NCV based quantum circuit that implements a 1-bit full adder shown in Fig. 2.18a. This circuit is an optimal realization with respect to the NCV-cost. The quantum circuit has an NCV-cost of 6. An equivalent circuit outlined in Fig. 2.16a has an NCV-cost of 12.

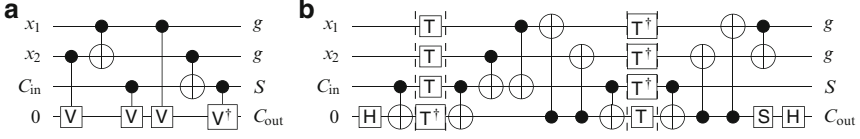


Fig. 2.18 Quantum circuits realizing a 1-bit full adder. (a) Optimal NCV circuit. (b) Optimal Clifford + T circuit

Table 2.5 Quantum cost for MPMCT gates with c controls

Number of controls (c)	NCT gate-count	NCV-cost (+2) ^a	T -depth
0	1	1	0
1	1	1	0
2	1	5	3
3	4	20	12
4	10	40	30
≥ 5	$8(c - 3)$ [15]	$40(c - 3)$	$24(c - 3)$

^aIn case all the controls are negative, the NCV-cost is incremented by 2

Definition 2.18 (T -Depth). The T -depth is the number of T -stages in a quantum circuit where each stage consists of one or more T or $T^\dagger$ gates performed in parallel on separate qubits.

The motivation for that cost measure originates from the fact that the T gate is significantly larger compared to the other gates in the circuit. The T -count of a Clifford + T circuit is the total number of T and $T^\dagger$ gates in the circuit, while the H -count denotes the total number of H gates in a quantum circuit.

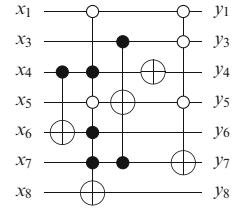
Example 2.24. The Clifford + T optimal circuit with respect to the T -depth of a 1-bit full adder is depicted in Fig. 2.18b. This circuit has a T -depth of 2, T -count of 8, and H -count of 2, while its equivalent realization sketched in Fig. 2.16b has a T -depth of 6, T -count of 14, and H -count of 4.

The quantum cost can be directly calculated from a reversible circuit. It is calculated based on the principles proposed originally in [15]. First, each MPMCT gate in a given circuit is mapped to an NCT circuit, then each Toffoli gate in the later circuit is evaluated by its quantum cost. Finally, one can obtain the quantum cost of a reversible circuit by computing the sum of quantum cost for each gate. For more details, the reader is referred to Sect. 4.1.1.

Table 2.5 reports all the different quantum costs as well as the gate-count of an NCT circuit for each MPMCT gate.

Example 2.25. In Fig. 2.19, the MPMCT based reversible circuit has an NCV-cost of 107 and a T -depth of 63. Based on Table 2.5, we will calculate the quantum cost for each gate, and then the cost for the whole circuit.

Fig. 2.19 MPMCT based reversible circuit



The NCV-cost is obtained in the following:

$$\text{NCV-cost}_{g_1} = 1$$

$$\text{NCV-cost}_{g_2} = 40 \cdot (5 - 3) = 80$$

$$\text{NCV-cost}_{g_3} = 5$$

$$\text{NCV-cost}_{g_4} = 1$$

$$\text{NCV-cost}_{g_5} = 20$$

$$\text{NCV-cost}_G = \sum_{i=1}^5 \text{NCV-cost}_{g_i} = 1 + 80 + 5 + 1 + 20 = 107$$

The T -depth is calculated with the same way:

$$T\text{-depth}_{g_1} = 0$$

$$T\text{-depth}_{g_2} = 24 \cdot (5 - 3) = 48$$

$$T\text{-depth}_{g_3} = 3$$

$$T\text{-depth}_{g_4} = 0$$

$$T\text{-depth}_{g_5} = 12$$

$$T\text{-depth}_G = \sum_{i=1}^5 T\text{-depth}_{g_i} = 0 + 48 + 3 + 0 + 12 = 63$$

2.7.2 Number of Gates

Definition 2.19 (Number of Gates). This metric is related to reversible circuits. It is referred to the total number of gates in a reversible circuit. It is also denoted as *gate-count*.

Example 2.26. The reversible circuit shown in Fig. 2.19 has a number of gates of 5. Figure 2.20a illustrates a reversible circuit with a number of gates of 6.

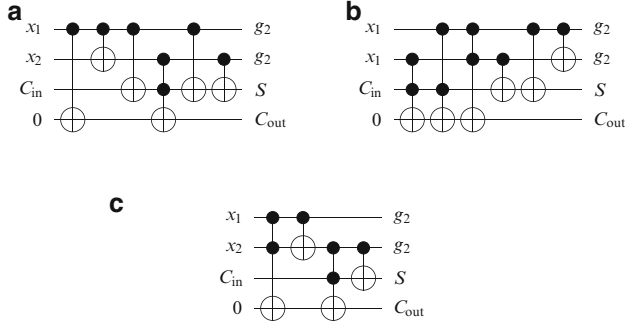


Fig. 2.20 Reversible circuits realizing a 1-bit full adder. (a) First realization. (b) Second realization. (c) Optimal circuit regarding the gate-count

This metric is used to evaluate a given realizations for a reversible function. However, this metric does not give any information about the quantum cost of the corresponding quantum circuit. The minimization based on this metric does not guarantee a minimization of the quantum cost and the inverse is also not true. In fact it is possible that two different realizations for the same reversible function have different gate-count in a strictly increasing order, but the quantum cost for the first realization is more expensive than the second.

Example 2.27. Figure 2.20 shows two different realizations of a 1-bit full adder and a third optimized circuit for both in terms of the number of gates. Although, the first reversible circuit depicted in Fig. 2.20a has more gates (6 gates) than the circuit presented in Fig. 2.20c (4 gates), the first realization is cheaper in terms of quantum cost than the second circuit. More precisely, the NCV-cost and the T -depth of the first circuit are 10 and 3, respectively. The second circuit has an NCV-cost of 12 and a T -depth of 6. Consider the circuits given in Figs. 2.22b and 2.20c. The first circuit has more gates as well as higher quantum cost than the second. Therefore, as mentioned above, minimizing a circuit with respect to the gate-count does not guarantee a minimization on the quantum cost.

2.7.3 Number of Lines

Definition 2.20 (Number of Lines). The number of lines refers to the total number of the input variables or the output variables used in a reversible or a quantum circuit.

Note that if the function to be synthesized is reversible, the number of circuit lines can be equal to the number of the inputs. However, in the case where the Boolean function to be synthesized is irreversible, then additional variables (i.e., constant

inputs and garbage outputs) are unavoidable [76]. Therefore, there are several optimization approaches that target the reduction of the number of lines in reversible circuits [148].

Example 2.28. Figure 2.22a shows a reversible circuit with a number of lines of 4, where the number of primary inputs, constants, primary outputs, and garbage outputs are 3, 1, 2, and 2, respectively.

2.7.4 Depth

This metric recognizes whether gates can be applied in parallel which can lead to a reduction in the execution time of a circuit. This cost metric is related to quantum circuits composed by the NCV library or the Clifford + T library.

Definition 2.21 (Depth). Let $U_i(C_i, t_i)$ and $U_{i+1}(C_{i+1}, t_{j+1})$ be two consecutive quantum gates. These gates can be applied *concurrently* if

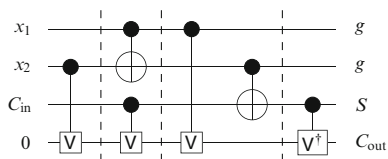
$$|C_i \cup C_{i+1} \cup \{t_i, t_{i+1}\}| = |C_i| + |C_{i+1}| + 2.$$

In other words, if the lines used by each gate (both control and target line) are disjoint. Let G be a quantum circuit with k elementary quantum gates, then G can be partitioned into $d \leq k$ subcircuits whose gates can be pairwise applied concurrently. We refer to the minimal d as the *depth* of the circuit.

It is defined as a sub-sequence of gates in a circuit that can be applied in parallel. We assume in this book that two or more gates can operate at the same level if they operate on disjoint qubits and can be grouped together in the circuit.

Example 2.29. The quantum circuit that realizes a 1-bit full adder illustrated in Fig. 2.21 has an NCV-cost of 6. Gates in position 2 and 3 can be applied *concurrently* because they don't share any line together. The same for gates in positions 4 and 5. Hence, the depth of this circuit is 4.

Fig. 2.21 Quantum depth for a 1-bit full adder circuit



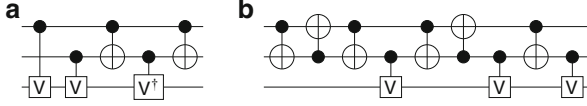


Fig. 2.22 NCV quantum circuits realizing the Toffoli gate given in Fig. 2.10c. (a) First realization. (b) Second realization

2.7.5 Nearest Neighbor Cost

Many physical realizations of quantum circuits permit limited interaction distance between gate qubits [47]. For example, trapped ions (cf. [58]), liquid nuclear magnetic resonance (cf. [68]), and architectures based on the original Kane model [62] assume that the computations are only performed between adjacent (i.e., nearest neighbor) qubits [47].

Hence, synthesis of quantum circuit with nearest neighbor quantum gates has become an important research branch [27, 59, 63, 96, 107, 151]. This cost is referred to *nearest neighbor cost*.

Definition 2.22 (Nearest Neighbor Cost). Let $U_i(C_i, t_i)$ be quantum gate. The *nearest neighbor cost* (NNC) of U_i is defined $|C_i - t_i - 1|$, i.e., the distance between the control and the target lines of an elementary gate. Let G be a quantum circuit with k elementary quantum gates. Then, the NNC of G is defined as the sum of the NNCs of its gates:

$$\sum_{i=0}^k |C_i - t_i - 1| \quad (2.9)$$

Optimal NNC for a circuit is 0, where all quantum gates are either 1-qubit or 2-qubit gates performed on adjacent qubits. The NNC is also denoted by the interaction cost.

Example 2.30. The quantum circuits given in Fig. 2.22 are possible quantum realizations for a Toffoli gate. The first implementation is an optimal circuit regarding the quantum cost. It has an NCV-cost of 5 but an NNC of 1, while the second circuit represents an optimal circuit with respect to the NNC since it has an NCV-cost of 9 but an NNC of 0.

2.8 Decision Diagrams

2.8.1 Binary Decision Diagrams

A *binary decision diagram* (BDD) is a standard data structure for representing Boolean functions. The general concepts for BDDs are briefly outlined in this section, nevertheless, the reader is referred to the literature for a comprehensive overview [25, 67].

Let $X = x_1, \dots, x_n$ be the variables of a Boolean function $f \in \mathcal{B}_n$. A BDD representing the function f is a directed acyclic graph with nonterminal vertices N and terminal vertices $T \subseteq \{\perp, \top\}$ where $N \cap T = \emptyset$ and $T \neq \emptyset$. Each nonterminal vertex $v \in N$ is labeled by a variable from X and has exactly two children, $\text{low } v$ and $\text{high } v$. The directed edges to these children are called *low-edge* and *high-edge* and are drawn dashed and solid, respectively. A nonterminal vertex v labeled x_i represents a function denoted $\sigma(v)$ given by the *Shannon decomposition* [121]

$$\sigma(v) = \bar{x}_i \sigma(\text{low } v) + x_i \sigma(\text{high } v) \quad (2.10)$$

where $\sigma(\text{low } v)$ and $\sigma(\text{high } v)$ are the functions represented by the children of v with $\sigma(\perp) = 0$

$$\sum_{i=0}^k |C_i - t_i - 1|$$

and $\sigma(\top) = 1$.

$$T \subseteq \{\perp, \top\}$$

The BDD has a single start vertex s with $\sigma(s) = f$.

A BDD is *ordered* if the variables of the vertices on every path from the start vertex to a terminal vertex adhere to a specific ordering. Not all of the variables need to appear on a particular path, but a variable can appear at most once on any path. A BDD is *reduced* if there are no two nonterminal vertices representing the same function. Hence, the representation of common sub-functions is shared. In the following, only reduced and ordered BDDs are considered and for brevity referred to as BDDs.

Multiple-output functions can be represented by a single BDD that has more than one start vertex. Common sub-functions that can be shared among the functions decrease the overall size of the BDD. In fact, many practical Boolean functions can efficiently be represented using BDDs, and efficient manipulations and evaluations are possible.

Example 2.31. Figure 2.23a illustrates the Shannon decomposition from (2.10). A binary decision diagram for the 1-bit full adder Boolean function, described by the truth table outlined in Fig. 2.8a, is given in Fig. 2.23b.

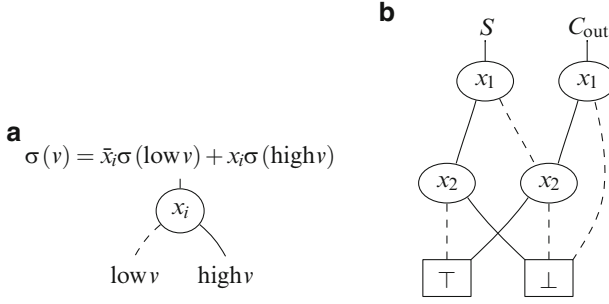


Fig. 2.23 Binary decision diagrams. (a) Shannon decomposition. (b) BDD for a 1-bit full adder

2.8.2 Quantum Multiple-Valued Decision Diagrams

A *quantum multiple-valued decision diagrams* (QMDD) [87] is a directed acyclic graph representing and manipulating $r^n \times r^n$ complex-valued matrices including the unitary matrices required to represent n -qubit quantum gates and circuits with r pure logic states. The fundamental idea is a recursive partitioning of the respective transformation matrix and the use of edge and vertex weights to represent various complex-valued matrix entries. More precisely, a transformation matrix of dimension $r^n \times r^n$ is successively partitioned into r^2 sub-matrices of dimension $r^{n-1} \times r^{n-1}$. This partitioning is represented by a directed acyclic graph (QMDD) with the following properties:

- There is a single terminal vertex representing the complex number 1 without any outgoing edge.
- Nonterminal vertices are labeled by an r^2 valued selection variable and have r^2 outgoing edges designated $e_1, e_2, \dots, e_{r^2}$.
- There is a single root vertex which has a single incoming edge (the root edge) that itself has no source vertex.
- Every edge (including the root edge) has an associated complex-valued weight and edges with a weight of 0 (0-edges) point to the terminal vertex.
- There are no redundant vertices, i.e., no nonterminal vertex has r^2 identical outgoing edges (destinations and weights).
- Nonterminal vertices are unique, i.e., no two nonterminal vertices labeled by the same selection variable have the same set of outgoing edges (destinations and weights).
- Nonterminal vertices are normalized (see details in the following section).

Each assignment to the selection variables corresponds to choosing the respective sub-matrices in the partitioning process, and as a result, to some matrix entry. Thus, for any entry of the $r^n \times r^n$ matrix the QMDD can be evaluated in at most n steps by multiplying the weights on the path from the root vertex to the terminal vertex that is determined by the respective assignment.

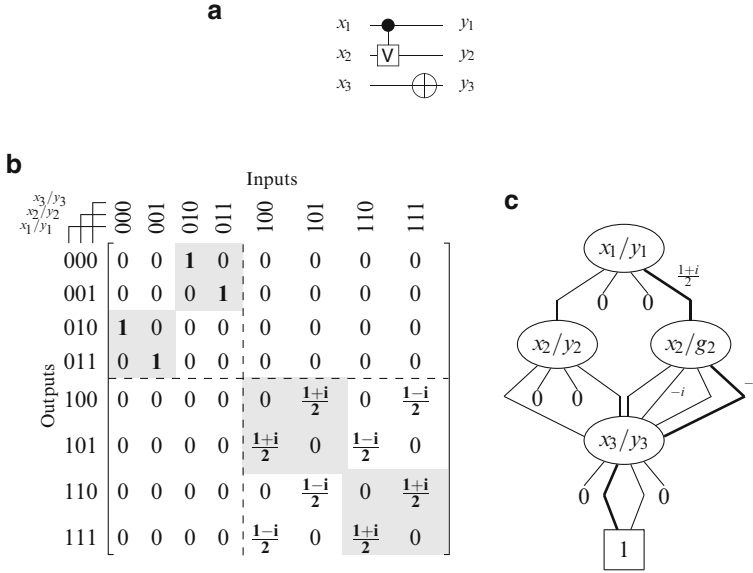


Fig. 2.24 Unitary matrix representations. (a) Quantum circuit based on the NCV library. (b) Transformation matrix. (c) QMDD

Example 2.32. Consider the NCV based circuit G of Fig. 2.24a. The transformation matrix, U_G , of the circuit is obtained by the multiplication of the unitary matrices of its gates in a reversed order.

The transformation matrix for the first gate, U_{g_1} , is the tensor product of the matrix representing a controlled V gate and a line which acts as an identity gate:

$$\begin{aligned}
 U_{g_1} &= U_v \otimes U_I \\
 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} \\ 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} \end{bmatrix}
 \end{aligned}$$

The matrix of the NOT gate, U_{g_2} , is calculated by the Kronecker product of the identity matrix, the NOT matrix, and the identity matrix:

$$\begin{aligned}
 U_{g_2} &= U_I \otimes U_I \otimes U_{NOT} \\
 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}
 \end{aligned}$$

The transformation matrix, U_G , that represents the circuit G is obtained by multiplying the matrices U_{g_2} and U_{g_1} :

$$\begin{aligned}
 U_G &= \prod_{i=n}^1 U_{g_i} = U_{g_2} \times U_{g_1} \\
 &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} \\ 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} \end{bmatrix} \\
 &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} \\ 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} \\ 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} & 0 \end{bmatrix}
 \end{aligned}$$

Hence, the unitary matrix describing the behavior of this circuit depicted in Fig. 2.24a is given in Fig. 2.24b. The QMDD for this matrix is given in Fig. 2.24c. The edges from each nonterminal vertex point to four sub-matrices. Each edge has a complex-valued weight. For clarity, edges with weights equal to 1 are omitted and edges with weight 0 are indicated as stubs. In fact, they point to the terminal vertex.

Here, the unique root vertex labeled by the pairing x_0/y_0 represents the whole matrix and has four outgoing edges targeting child vertices representing from left to right:

- the top-left sub-matrix where $x_1 = 0$ maps to $y_1 = 0$,
- the top-right sub-matrix where $x_1 = 1$ maps to $y_1 = 0$,
- the bottom-left sub-matrix where $x_1 = 0$ maps to $y_1 = 1$, and
- the bottom-right sub-matrix where $x_1 = 1$ maps to $y_1 = 1$.

This decomposition is repeated at each partitioning level until the terminal vertex that represents a single matrix entry $\boxed{1}$ is reached. The vertices are normalized by dividing the weights of all outgoing edges by a normalization factor (here: such that the leftmost edge with a nonzero weight has weight 1). This factor is propagated to referencing edges, e.g., the factor $\frac{1+i}{2}$ is propagated downwards from the x_1/y_1 -level to the x_3/y_3 -level in Fig. 2.24b. By this, structurally equivalent sub-matrices are compressed to a shared vertex (highlighted in same color in Fig. 2.24b). This procedure is repeated for each level until a single terminal vertex labeled by $\boxed{1}$ is created in the bottom. To obtain the value of a particular matrix entry, one has to follow the corresponding path from the root to the terminal vertex and multiply all edge weights on this path. For example, the matrix entry $\frac{1+i}{2}$ from the bottom “rightmost” sub-matrix of Fig. 2.24b can be determined as the product of the weights on the highlighted path of the QMDD in Fig. 2.24c.

A reversible function has the same number of inputs and outputs and maps each input pattern to a unique output pattern. In particular, a reversible function with n variables describes a permutation π_{2^n} of the set $\{0, 1, \dots, 2^n - 1\}$. This permutation can also be described using a *permutation matrix*, i.e., a $2^n \times 2^n$ matrix Π_f as defined in Sect. 2.5.1. Since the QMDD [87] is a data structure that can represent the $r^n \times r^n$ unitary matrices used to describe quantum operations over r -valued logic. In the case where $r = 2$, a unitary matrix is in fact a permutation matrix. Hence, QMDDs are also suitable for representing and manipulating reversible functions.

Example 2.33. Figure 2.25a shows the truth table for the reversible function of a 1-bit full adder. Its permutation matrix is depicted in Fig. 2.25b. Its output pattern has only one input pattern. Therefore, each row has only one 1. The QMDD for the permutation matrix in Fig. 2.25b is shown in Fig. 2.25c. Two out of the sixteen 1-paths (paths from the root vertex to a $\boxed{1}$ -terminal) have been randomly chosen and are highlighted in bold representing the mappings $0000 \mapsto 0000$ and $1110 \mapsto 1011$.

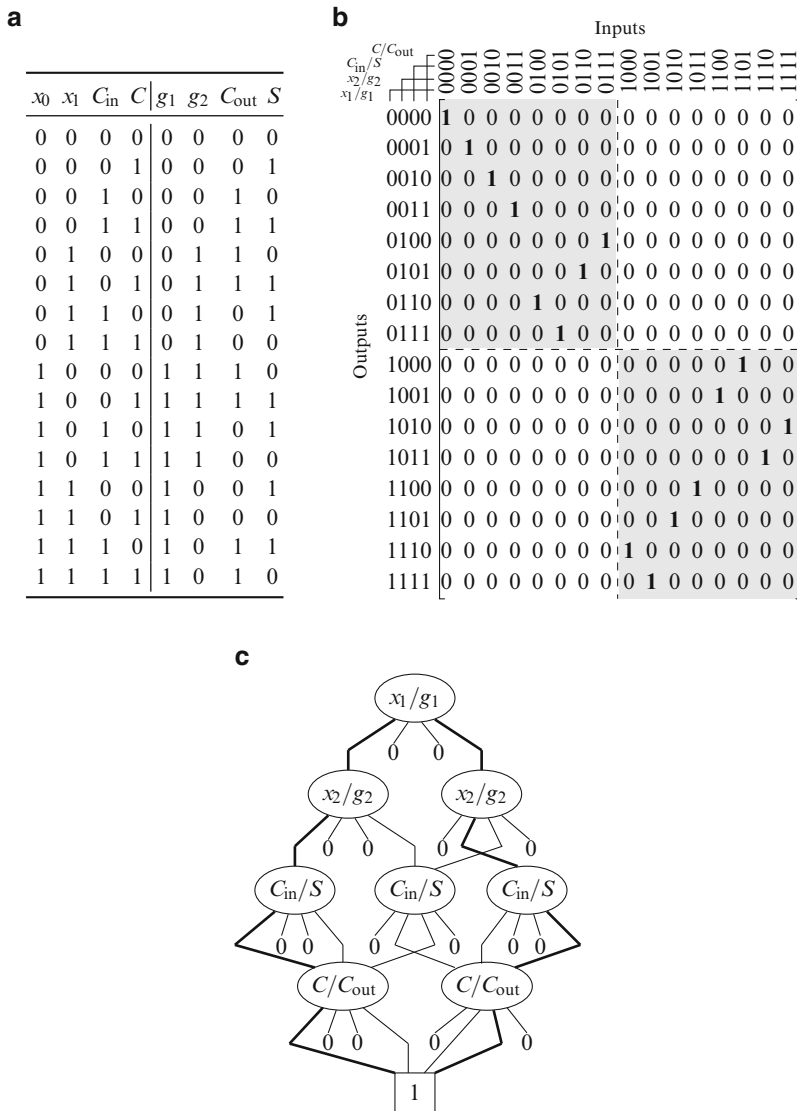


Fig. 2.25 1-bit full adder reversible function and its different representations. (a) Truth table. (b) Permutation matrix. (c) QMDD

Since QMDDs involve multiple edges from vertices and are applicable to both binary and multiple-valued problems, the QMDD package is not built using a standard decision diagram package, but the implementation employs well-known decision diagram techniques. QMDDs are ordered in a manner analogous to BDDs.

Hence, the vertex labels along any path from the start vertex adhere to a fixed ordering and no label appears more than once on any path. QMDDs are also reduced in a manner analogous to BDDs so each vertex represents a unique sub-matrix and the representation of common sub-matrices is shared.

The size of a unitary transformation matrix grows exponentially with respect to the number of input/output variables. However, QMDDs [87] provide an efficient data-structure which enables a much more compact representation of unitary matrices. QMDDs are canonical representations, if normalization of edge weights (as described in [87]) is performed. Thus, they are very convenient for equivalence checking.

Reversible and Quantum Circuits

Optimization and Complexity Analysis

Abdessaied, N.; Drechsler, R.

2016, XXII, 186 p. 105 illus., 3 illus. in color., Hardcover

ISBN: 978-3-319-31935-3