

---

# Q-Map: Quantum Circuit Implementation of Boolean Functions

Hassan Hajjdiab<sup>1,\*</sup>, Ashraf Khalil<sup>2</sup>, Hichem Eleuch<sup>3,4,5</sup>,

<sup>1</sup> Computer Science and Software Engineering Department, Concordia University, Montreal, Quebec, Canada; Email: hassan.hajjdiab@ieee.org

<sup>2</sup> College of Technological Innovation, Zayed University, Abu Dhabi, UAE; Email: ashraf.khalil@zu.ae

<sup>3</sup> Department of applied physics and astronomy, University of Sharjah, Sharjah, UAE; Email: heleuch@sharjah.ac.ae

<sup>4</sup> College of Arts and Sciences, Abu Dhabi University, Abu Dhabi 59911, UAE

<sup>5</sup> Institute for Quantum Science and Engineering, Texas A&M University, College Station, TX 77843, USA

\*Author to whom correspondence should be addressed

## Abstract

Quantum computing has gained attention in recent years due to the significant progress in quantum computing technology. Today many companies like IBM, Google and Microsoft have developed quantum computers and simulators for research and commercial use. The development of quantum techniques and algorithms is essential to exploit the full power of quantum computers. In this paper we propose a simple visual technique (we call Q-Map) for quantum realisation of classical Boolean logic circuits. The proposed method utilises concepts from Boolean algebra to produce a quantum circuit with minimal number of quantum gates.

## Introduction

The advancement of quantum computing hardware and software intrigues researchers to develop quantum algorithms in areas such as cryptography, image processing, algorithms, finance [6, 10, 13, 15] and many other areas. One main advantage of quantum computers compared to classical computers is the processing power. The quantum computer can process computationally expensive tasks exponentially faster than the classical computer. While classical algorithms are limited in complexity to  $O(n)$ , a quantum search algorithm proposed by Grover [8] uses  $O(\sqrt{n})$  for unsorted list of  $n$  items. Shor [23] proposed a quantum algorithm to factor an integer  $n$  in polynomial of  $\log n$  time complexity. At this point, there is no classical algorithm that can solve number factorisation in polynomial time. The RSA cryptographic system [20] is based on prime number factorisation, and thus with quantum computers an RSA encrypted message can be decrypted in polynomial time complexity. Hallgen [9] presented a polynomial-time quantum algorithm to solve the Pell-Fermat equation [3] (also known as the Pells equation<sup>1</sup>). In classical

---

<sup>1</sup>Pell-Fermat equation is:  $x^2 - dy^2 = 1$  and the goal is to find pairs of integers  $(x, y)$  to satisfy the equation

algorithm there is no known polynomial time solution and the problem is known to be NP complete [4]. Recently, a group of scientists at Google AI Quantum [2] used 53 qubit quantum computer to sample the output of a pseudo-random quantum circuit [18]. The results were compared with the state-of-the-art super computer that needs 10,000 years while the 53 qubit quantum computer needs 200 seconds.

Another advantage of quantum computers is low energy consumption. In computation, energy consumption is correlated with reversibility of the computation. Irreversibility is equivalent to information erasure. For the case of an **AND** gate of output **0**, we may say that we cannot uniquely identify the input, the **AND** operation resulted in erasure of information and thus consumed energy [14]. As demonstrated by Landauer [14], classical binary computers mainly dissipate energy during information erasure at the rate of  $KT \ln 2$  per bit erased, where  $K$  is the Boltzmann constant and  $T$  is the temperature in Kelvins. At room temperature (300 Kelvin), each bit erasure will cost around  $3 \times 10^{-21}$  Joules. This number appears to be too small, however the digital binary computer is composed of huge number of Boolean logic gate operations at the hardware level which results in significant consumption of energy. On the other hand, quantum computers utilise quantum gates which are all reversible gates and thus quantum computation do not result in energy consumption.

In this paper we propose a simple visual technique (we call Q-Map) for quantum realisation of classical Boolean logic circuits. Classical boolean computation can be described in terms of classical boolean functions. To perform classical computations using a quantum computer, the classical boolean function need to be synthesised using reversible functions.

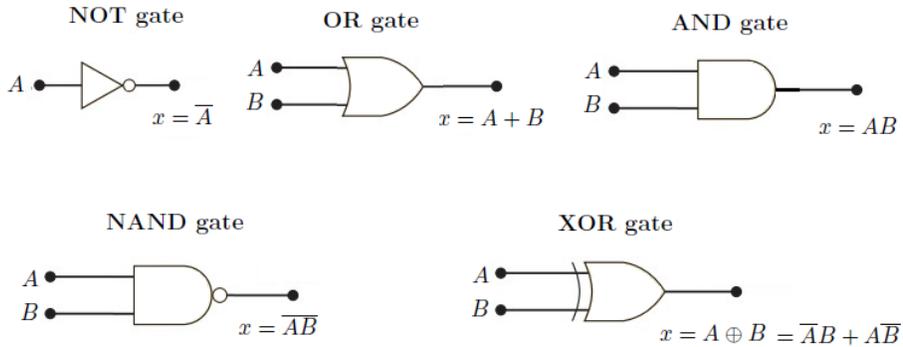
## Overview of classical and Quantum gates

A Boolean function with  $n$  inputs and  $m$  outputs is defined as:

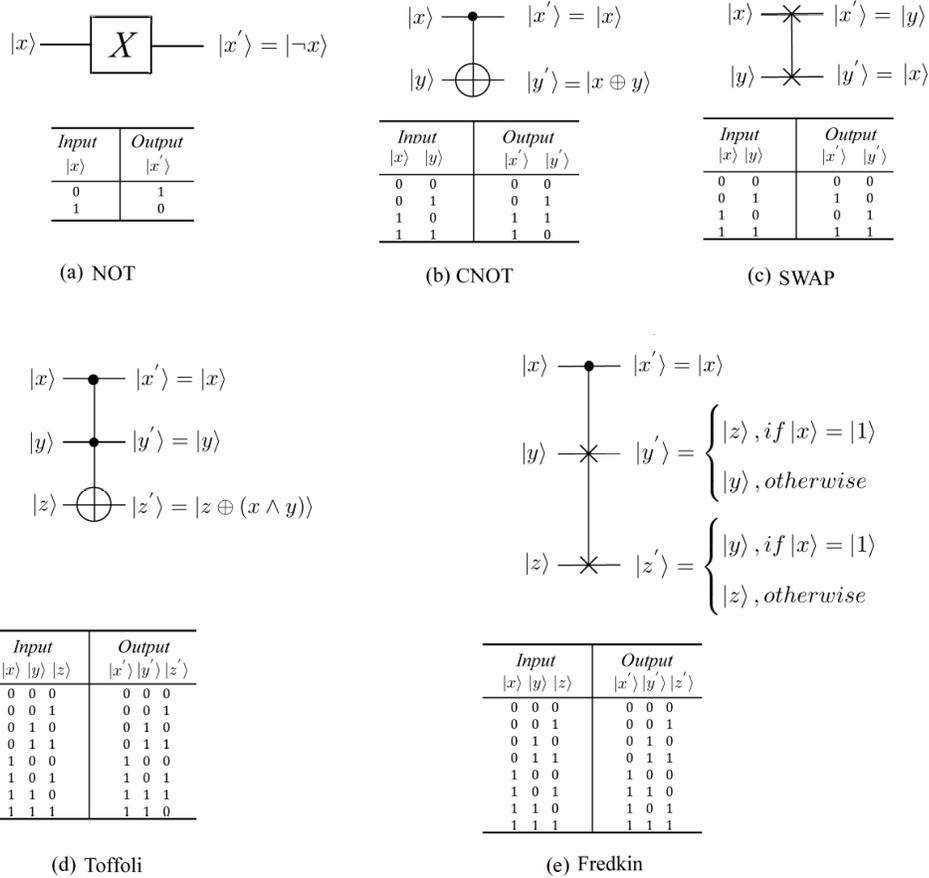
$\mathcal{B}_{n,m} \stackrel{\text{def}}{=} \{f | f : \mathbb{B}^n \rightarrow \mathbb{B}^m\}$ , where the *Boolean* values are denoted by  $\mathbb{B} \stackrel{\text{def}}{=} \{0, 1\}$ .

A function  $f \in \mathcal{B}_{n,n}$  is reversible if it is bijective [21], each input map exactly to one output and the number of inputs is equal to the number of outputs.

Classical Binary computers are, in essence, composed of irreversible logic gates. Other than the **NOT** gate, the binary logic gates are irreversible gates ( see Figure 1 ). For example the classic **AND** gate is irreversible, for an output of **0** the input could be **00**, **01** or **10** and cannot be uniquely identified.



**Figure 1.** Basic Boolean gates, only the **NOT** gate is reversible



**Figure 2.** Commonly used quantum gates

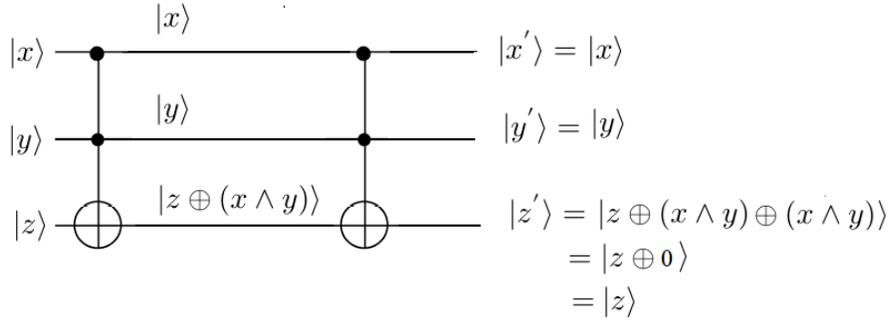
On the other hand, a quantum bit (or qubit)  $\mathbf{x}$  represents a unit of information and can be described in a two dimensional quantum system as follows:

$$|x\rangle = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \text{ where } \sqrt{|c_0|^2 + |c_1|^2} = 1$$

The quantum states of  $|0\rangle$  and  $|1\rangle$  are represented by the vectors  $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

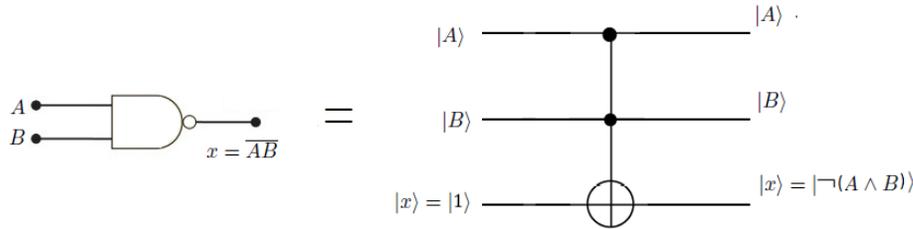
and  $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

The qubit can be in an "on" or "off" states as in the classical Boolean computers or in any combination of the "on-off" states :  $\mathbf{x} = \alpha|0\rangle + \beta|1\rangle$ , where  $\sqrt{\alpha^2 + \beta^2} = 1$ . To represent a classical digital system the qubit  $|0\rangle$  and  $|1\rangle$  are sufficient. Figure 2 shows the commonly used quantum gates, all quantum gates are reversible and the input can be reconstructed from the output by applying the gate twice. Figure 3 shows an example using the Toffoli gate, the input can be reconstructed by applying the Toffoli gate twice.



**Figure 3.** All quantum gates are reversible gates, the input can be recovered by applying the same gate twice. The figure above shows an example using the Toffoli gate, we can reconstruct the input by applying the Toffoli gate twice.

In this paper we propose a technique to implement a binary logic circuit using a quantum gates, and thus only the  $|0\rangle$  and  $|1\rangle$  states of the qubit are utilised. In classical Boolean circuits, the **NAND** gate is a universal gate and all other Boolean gates could be constructed using one or more **NAND** gates [27]. Any Boolean logic circuit can be designed using reversible quantum gates, the logic gates presented in Figure 1 can be constructed using a combination of the **NOT**, **CNOT** and **Toffoli** gates, thus the **NOT-CNOT-Toffoli** gates form a universal basis for quantum circuit implementation [5, 19, 31]. Figure 4 shows quantum reconstruction of the **NAND** gate using a Toffoli gate, the quantum equivalent requires an extra bit ( i.e ancillary bit ).The rest of the boolean gates can be synthesised using the **NOT-CNOT-Toffoli** bases.



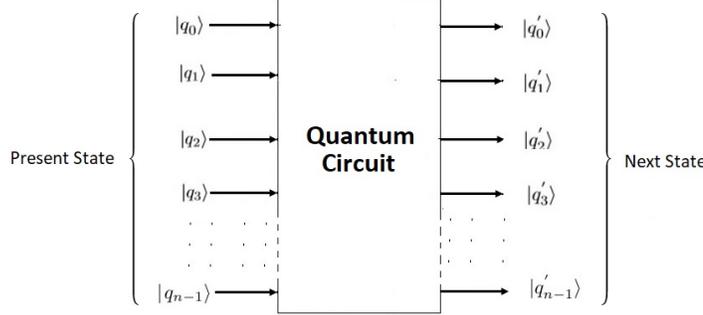
**Figure 4.** Reconstruction of the logic **NAND** gate using the Toffoli gate, all Boolean logic gates can be reconstructed using quantum gates.

Thus any Boolean function can be synthesised by simply replacing each Boolean gate by its quantum counterpart. However this approach is hardly efficient and leads to a significant number of ancillary bits [17, 30]. In literature several approaches have been proposed [7, 12, 24, 25, 28, 29] to synthesise a given Boolean function with minimal number of ancillary bits. Most of the approaches rely on heuristic methods to minimise the costs of the resulting circuits using complex function manipulation [11]. In this paper we present an exact method to realise the quantum implementation of any classical binary system. The technique (we call Q-Map) is analogous to the Karnaugh Map [27] for classical logic gate minimisation technique. The main contribution of this paper is as follows:

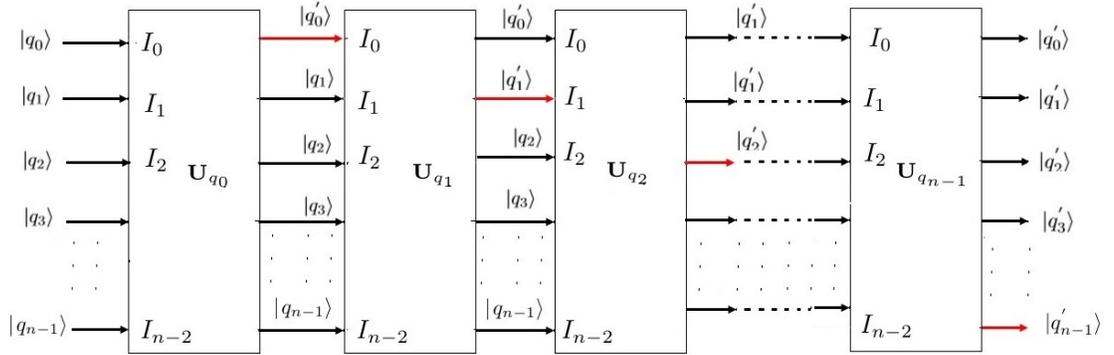
- We propose a visual method to synthesise any Boolean functions without having to resort to complex function decomposition and manipulation.
- We demonstrate the algorithm by implementing the 4-bit Gray Code Encoder using QISKIT.

## Quantum-Map technique

In our proposed approach, the problem is modelled as a quantum circuit with  $n$  input quantum bits ( $|q_0, q_1, \dots, q_{n-1}\rangle$ ) that represents the initial state of every qubit and  $n$  output quantum bits ( $|q'_0, q'_1, \dots, q'_{n-1}\rangle$ ) that represent the final state of each bit (see Fig. 5 (a)). The quantum circuit is further decomposed into a series of  $n$  cascaded stages. Each stage is a quantum circuit with  $n$  input qubits represented by the vector  $\mathbf{V}_i$  and  $n$  output qubits represented by the vector  $\mathbf{V}'_i$  where qubit  $|q_i\rangle$  is altered and the rest of the qubits are unaltered, the stages are presented in Fig. 5 (b).



(a) Quantum Circuit



(b) The Quantum circuit is decomposed into  $n$  stages.

**Figure 5.** (a) The quantum circuit is composed of the present state vector  $|q_0q_1q_2q_3 \dots q_{n-1}\rangle$  and the next state vector  $|q'_0q'_1q'_2 \dots q'_{n-1}\rangle$  (b) The circuit is decomposed into  $n$  cascaded stages, for stage  $\mathbf{U}_{q_i}$  only qubit  $q_i$  is changed to its next state  $q'_i$  all other qubits are unaltered.

The relationship between the input vector and the output qubit at stage  $i$  is defined by a control function  $\mathbf{U}_{q_i} : \mathbb{B}^n \rightarrow \mathbb{B}^n$  as follows:

$$\mathbf{V}_i \xrightarrow{\mathbf{U}_{q_i}} \mathbf{V}'_i \quad (1)$$

Where  $\mathbf{U}_{q_i}$  is a control function that computes a new value at the target output  $q_i$  and leaves all other variables unaltered. Equation 1 can be expanded as:

$$\begin{array}{ccc} |q_0 q_1 q_2 q_3 \dots q_{n-1}\rangle & \xrightarrow{\mathbf{U}_{q_0}} & |q'_0 q_1 q_2 q_3 \dots q_{n-1}\rangle \\ |q_0 q_1 q_2 q_3 \dots q_{n-1}\rangle & \xrightarrow{\mathbf{U}_{q_1}} & |q_0 q'_1 q_2 q_3 \dots q_{n-1}\rangle \\ |q_0 q_1 q_2 q_3 \dots q_{n-1}\rangle & \xrightarrow{\mathbf{U}_{q_2}} & |q_0 q_1 q'_2 q_3 \dots q_{n-1}\rangle \\ & \vdots & \vdots \\ |q'_0 q'_1 \dots q'_{n-2} q_{n-1}\rangle & \xrightarrow{\mathbf{U}_{q_{n-1}}} & |q'_0 q'_1 \dots q'_{n-2} q'_{n-1}\rangle \end{array} \quad (2)$$

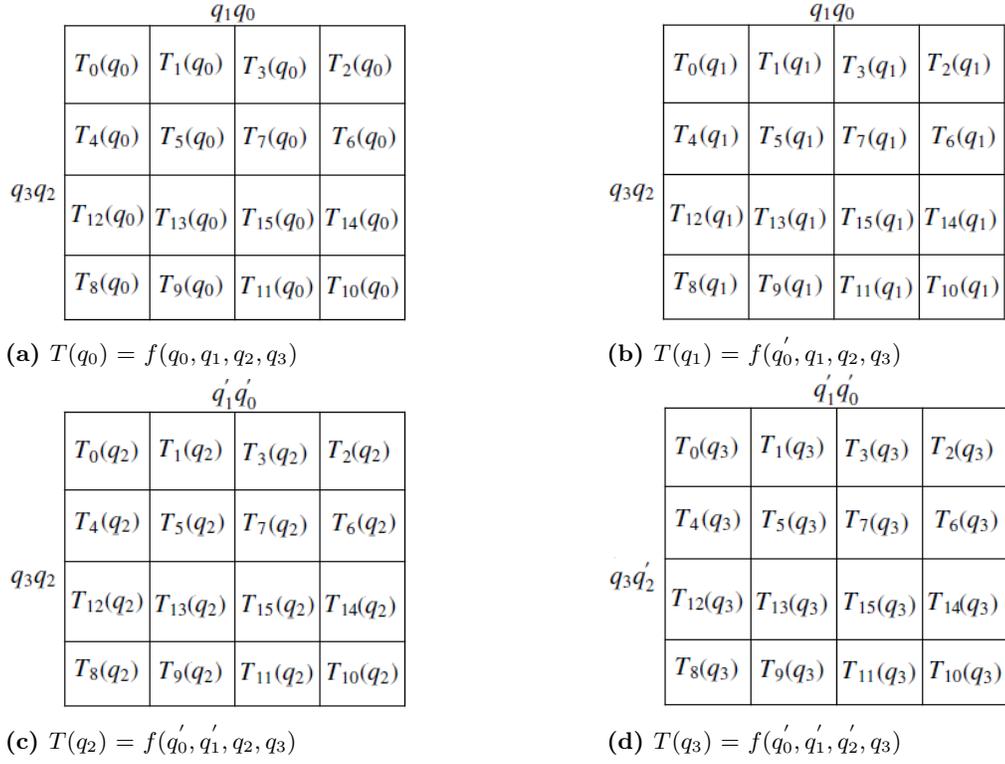
As presented in Equation 2, the input vector  $\mathbf{V}_0 = |q_0 q_1 q_2 q_3 \dots q_{n-1}\rangle$  includes all qubits, thus the quantum circuit  $\mathbf{U}_{q_0}$  will calculate  $q'_0$  (i.e the next state of  $q_0$ ) as a function of  $n$  qubits. The output vector  $\mathbf{V}'_0 = |q'_0 q_1 q_2 q_3 \dots q_{n-1}\rangle$  is an  $n$  qubit vector where only qubit  $q_0$  is changed to  $q'_0$  based on value of the control function  $\mathbf{U}_{q_0}$  and the rest of the qubits are unchanged. To calculate  $q'_1$ , the output qubit from the previous stage will be used. In general, the input vector for any stage  $k$  is the vector  $\mathbf{V}_k = |q_0 q_1 \dots q_{k-1} q_k \dots q_{n-1}\rangle$  and the output is the vector  $\mathbf{V}'_k = |q'_0 q'_1 \dots q'_{k-1} q'_k \dots q_{n-1}\rangle$ .

To calculate  $\mathbf{U}_{q_i}$  for each stage, we follow a function minimisation approach inspired by the Karnaugh Map technique [27] for Boolean function minimisation. Our proposed approach starts by building a logic map for the function to be minimised we call it the Quantum Map (Q-Map). The Q-Map is a two-dimensional array of cells used to represent a switching function. The switching function  $T(q_i)$ , presented in Table 1, represents the toggle state of a qubit from the present state  $q_i$  to the next state  $q'_i$  (i.e toggle from  $|0\rangle$  to  $|1\rangle$  or from  $|1\rangle$  to  $|0\rangle$ ). The function  $T(q_i)$  can be represented as the logical XOR of the current state  $q_i$  with the next state  $q'_i$  as presented in Table 1.

**Table 1.** Toggle function to represent change of state

| Present State<br>$q_i$ | Next State<br>$q'_i$ | Toggle function<br>$T(q_i) = q_i \oplus q'_i$ |
|------------------------|----------------------|---|
| $ 0\rangle$            | $ 0\rangle$          | $ 0\rangle$                                   |
| $ 0\rangle$            | $ 1\rangle$          | $ 1\rangle$                                   |
| $ 1\rangle$            | $ 0\rangle$          | $ 1\rangle$                                   |
| $ 1\rangle$            | $ 1\rangle$          | $ 0\rangle$                                   |

$T(q_i)$  is a function of  $n$  qubits denoted by  $|q_0 q_1 \dots q_{n-1}\rangle$ , the Q-Map of the stage  $\mathbf{U}_{q_i}$  is composed of two vectors  $\mathbf{S}_1 = |q_{n-1} \dots q_k\rangle$  and  $\mathbf{S}_2 = |q_{k-1} \dots q_0\rangle$  with a row for each assignment of  $\mathbf{S}_1$  for a total of  $2^{n-k-1}$  and with a column for each assignment of  $\mathbf{S}_2$  for a total of  $2^k$  columns. Similar to the Karnaugh Map, the adjacency condition is established by labelling the rows and the columns such that for any  $2^r$  adjacent rows ( or columns) differ only in  $r$  variables. Fig. 6 shows the Q-Map to find the quantum circuit with four variables. In each cell the corresponding value of the switching function  $T(q_i)$  is inscribed. Using the Q-Map, a minimal expression  $T(q_i)$  is calculated for each qubit. And finally the quantum circuit is implemented using the **NOT – CNOT – Toffoli** quantum gate basis.



**Figure 6.** Quantum maps with four variables. In each cell the corresponding value of  $T(q_i)$  is inscribed.

To demonstrate our proposed approach, we present a quantum circuit implementation of the Gray Code to Binary converter. The details of proposed algorithm with the reversible circuit implementation are presented in the next section.

# Implementation of the Gray Code Encoder

In this section we demonstrate the technique by implementing the Gray Code to Binary converter. The Gray Code is used in many applications such as position control systems, communications and many other areas [16]. The Gray code provides a binary code that changes by one bit only when it changes from one state to the next. The Gray code and the corresponding decimal unsigned binary equivalent is shown in Table 2.

**Table 2.** Gray code to Binary converter truth table.

| Present State |       |       |       | Next State |        |        |        |
|---------------|-------|-------|-------|------------|--------|--------|--------|
| $q_3$         | $q_2$ | $q_1$ | $q_0$ | $q'_3$     | $q'_2$ | $q'_1$ | $q'_0$ |
| 0             | 0     | 0     | 0     | 0          | 0      | 0      | 0      |
| 0             | 0     | 0     | 1     | 0          | 0      | 0      | 1      |
| 0             | 0     | 1     | 1     | 0          | 0      | 1      | 0      |
| 0             | 0     | 1     | 0     | 0          | 0      | 1      | 1      |
| 0             | 1     | 1     | 0     | 0          | 1      | 0      | 0      |
| 0             | 1     | 1     | 1     | 0          | 1      | 0      | 1      |
| 0             | 1     | 0     | 1     | 0          | 1      | 1      | 0      |
| 0             | 1     | 0     | 0     | 0          | 1      | 1      | 1      |
| 1             | 1     | 0     | 0     | 1          | 0      | 0      | 0      |
| 1             | 1     | 0     | 1     | 1          | 0      | 0      | 1      |
| 1             | 1     | 1     | 1     | 1          | 0      | 1      | 0      |
| 1             | 1     | 1     | 0     | 1          | 0      | 1      | 1      |
| 1             | 0     | 1     | 0     | 1          | 1      | 0      | 0      |
| 1             | 0     | 1     | 1     | 1          | 1      | 0      | 1      |
| 1             | 0     | 0     | 1     | 1          | 1      | 1      | 0      |
| 1             | 0     | 0     | 0     | 1          | 1      | 1      | 1      |

The first step starts by building the switching function  $T(q_i)$  for each quantum bit as described in Table 1. The function  $T(q_i)$  for each qubit is calculated as:

$$T(q_0) = q_0 \oplus q'_0 \tag{3}$$

$$T(q_1) = q_1 \oplus q'_1 \tag{4}$$

$$T(q_2) = q_2 \oplus q'_2 \tag{5}$$

$$T(q_3) = q_3 \oplus q'_3 \tag{6}$$

The result is presented in Table 3

**Table 3.** Gray Code to binary converter: Q-Map functions are calculated as the XOR of the present and next states.

| Present State |       |       |       | Next State |        |        |        | Toggle Functions |          |          |          |
|---------------|-------|-------|-------|------------|--------|--------|--------|------------------|----------|----------|----------|
| $q_3$         | $q_2$ | $q_1$ | $q_0$ | $q'_3$     | $q'_2$ | $q'_1$ | $q'_0$ | $T(q_3)$         | $T(q_2)$ | $T(q_1)$ | $T(q_0)$ |
| 0             | 0     | 0     | 0     | 0          | 0      | 0      | 0      | 0                | 0        | 0        | 0        |
| 0             | 0     | 0     | 1     | 0          | 0      | 0      | 1      | 0                | 0        | 0        | 0        |
| 0             | 0     | 1     | 1     | 0          | 0      | 1      | 0      | 0                | 0        | 0        | 1        |
| 0             | 0     | 1     | 0     | 0          | 0      | 1      | 1      | 0                | 0        | 0        | 1        |
| 0             | 1     | 1     | 0     | 0          | 1      | 0      | 0      | 0                | 0        | 1        | 0        |
| 0             | 1     | 1     | 1     | 0          | 1      | 0      | 1      | 0                | 0        | 1        | 0        |
| 0             | 1     | 0     | 1     | 0          | 1      | 1      | 0      | 0                | 0        | 1        | 1        |
| 0             | 1     | 0     | 0     | 0          | 1      | 1      | 1      | 0                | 0        | 1        | 1        |
| 1             | 1     | 0     | 0     | 1          | 0      | 0      | 0      | 0                | 1        | 0        | 0        |
| 1             | 1     | 0     | 1     | 1          | 0      | 0      | 1      | 0                | 1        | 0        | 0        |
| 1             | 1     | 1     | 1     | 1          | 0      | 1      | 0      | 0                | 1        | 0        | 1        |
| 1             | 1     | 1     | 0     | 1          | 0      | 1      | 1      | 0                | 1        | 0        | 1        |
| 1             | 0     | 1     | 0     | 1          | 1      | 0      | 0      | 0                | 1        | 1        | 0        |
| 1             | 0     | 1     | 1     | 1          | 1      | 0      | 1      | 0                | 1        | 1        | 0        |
| 1             | 0     | 0     | 1     | 1          | 1      | 1      | 0      | 0                | 1        | 1        | 1        |
| 1             | 0     | 0     | 0     | 1          | 1      | 1      | 1      | 0                | 1        | 1        | 1        |

The second step is to establish the Q-Map for each qubit. Figure 7 (a) shows the Q-Map to evaluate  $q_0$  given the values of the input  $q_1$ ,  $q_2$  and  $q_3$ . A value of 1 in the Q-Map represents the state of  $q_1$ ,  $q_2$  and  $q_3$  when  $q_0$  toggles its state. Thus  $q_0$  will toggle its state when the following expression is true:

$$T(q_0) = \bar{q}_3\bar{q}_2q_1 \oplus \bar{q}_3q_2\bar{q}_1 \oplus q_3\bar{q}_2\bar{q}_1 \oplus q_3q_2q_1 \quad (7)$$

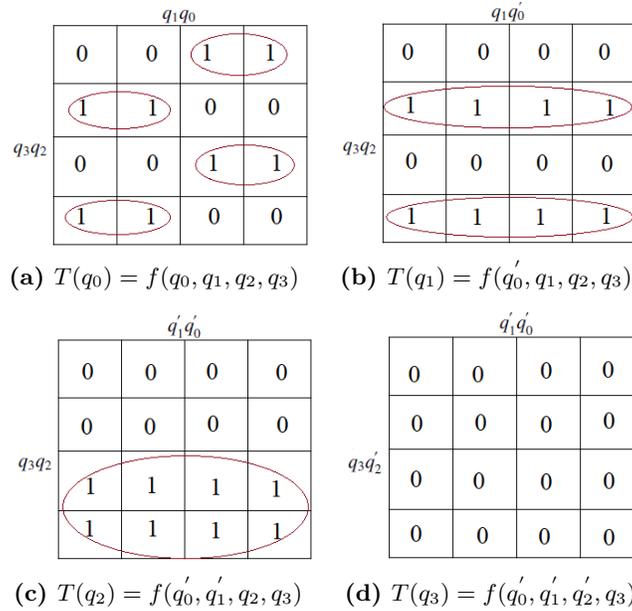
Figure 7 (b) shows the Q-Map to find  $T(q_1)$ . The expression is calculated based on inputs  $q_3$ ,  $q_2$  and  $q'_0$  as follows:

$$T(q_1) = q_3\bar{q}_2 \oplus \bar{q}_3q_2 \quad (8)$$

Figure 7 (c) shows the Q-Map to find  $T(q_2)$ . The expression is calculated based on inputs  $q_3$ ,  $q'_1$  and  $q'_0$  as follows:

$$T(q_2) = q_3 \quad (9)$$

Finally, figure 7 (d) shows the Q-Map to find  $T(q_3)$ . The expression is calculated based on the inputs  $q_2$ ,  $q'_1$  and  $q'_0$  as follows:



**Figure 7.** Q-Map representing the switching function for qubits  $q_0, q_1, q_2$  and  $q_3$ ,  $T(q_i)$  is written in Exclusive-OR Sum-of-Product form (ESOP)

Notice here we use the XOR ( $\oplus$ ) operation and the function  $T(q_i)$  is represented in Exclusive-OR Sum-of-Product form (ESOP) form since the qubit will toggle only if we have an odd number of true terms. For even number of true terms the qubit will retain its initial state. In classical Boolean function minimisation using Karnaugh Map the logic OR (+) operation is used and the function is expressed in Sum of Product (SOP) form since the Boolean function will be true if any of the terms is true and so group overlap is allowed.

In our proposed Q-Map approach group overlap is not allowed, this makes all terms in every function  $T(q_i)$  mutually exclusive and only one term can be true at one instant of time. Thus the XOR ( $\oplus$ ) operation can be replaced by the OR (+) operation and the function  $T(q_i)$  can be represented in the Sum of Product (SOP) form. Since each Q-Map in Figure 7 contains no overlapping groups, the functions can be represented in Sum of Product (SOP) form as follows:

$$\begin{array}{l}
 T(q_0) = \bar{q}_3\bar{q}_2q_1 + \bar{q}_3q_2\bar{q}_1 + q_3\bar{q}_2\bar{q}_1 + q_3q_2q_1 \\
 T(q_1) = q_3\bar{q}_2 + \bar{q}_3q_2 \\
 T(q_2) = q_3 \\
 T(q_3) = 0
 \end{array} \tag{10}$$

The functions presented in Equation 10 can be realised by a reversible circuit with only four lines (i.e.  $O(n)$  lines) using the **NOT-CNOT-Toffoli** bases with Multi-Control Toffoli gates [22]. This means that the implementation is efficient and no temporary lines (also referred as ancilla) are needed. The quantum circuit can be implemented using CNOT and two and three input Toffoli gates as shown in Figure 8.

The quantum circuit is also simulated using QISKIT open-source framework simulator for quantum circuit [1, 26]. In QISKIT, the Toffoli gate is composed of two control inputs and one output. To implement the circuit in Figure 8, we redesigned the quantum circuit to include 2-input Toffoli gates; however an additional ancillary qubit is needed to store the intermediate values. The design with 2-input Toffoli gates is presented in Figure 9. The code to simulate the quantum gray to binary converter is presented in Figure 11.

The Q-Map algorithm can be summarised as follows:

**Step 1:** For each qubit  $q_i$  build the toggle function  $T(q_i) = q_i \oplus q'_i$  as the logical Exclusive-OR of the present state  $q_i$  and the final state  $q'_i$ .

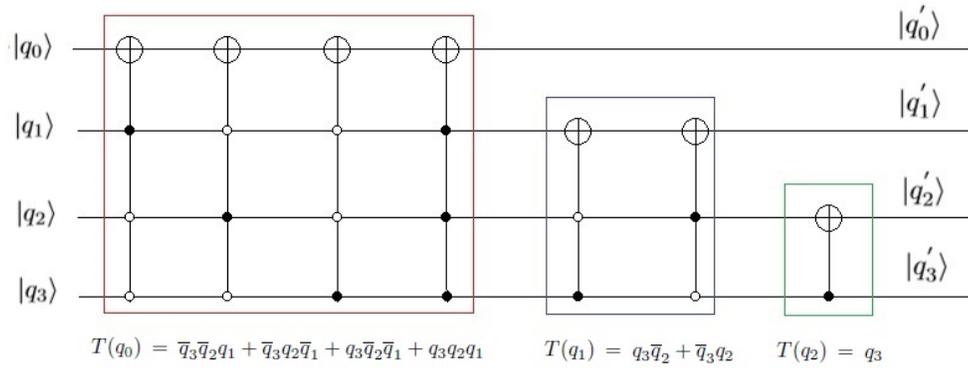
**Step 2:** Establish the Q-Map of the switching function  $T(q_i)$  as a function of the  $n$  qubits  $(q'_0, q'_1, \dots, q'_{i-1}, q_i, \dots, q_{n-1})$ .

**Step 3:** In each cell inscribe the value of  $T(q_i)$  in the Q-Map.

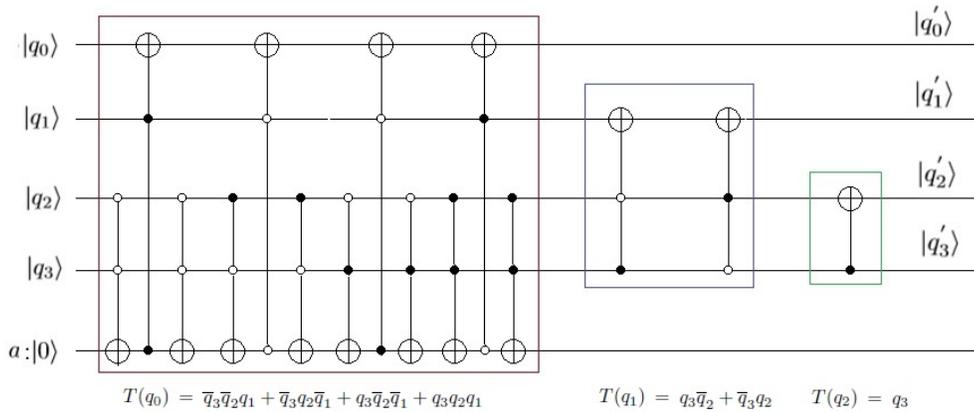
**Step 4:** Find the expression of  $T(q_i)$  in Sum-Of-Product form using the Q-Map such that:

1. Groups should be as large as possible
2. Group Overlapping is not allowed

**Step 5:** For each expression  $T(q_i)$  use the **CNOT-NOT-Toffoli** bases to implement the corresponding quantum circuit.



**Figure 8.** Implementation of the Quantum circuit using 2 and 3 input Toffoli and CNOT gates



**Figure 9.** Implementation of the Quantum circuit using 2-input Toffoli and CNOT gates



## Q-Map optimisation

In classical boolean function minimisation using the Karnaugh map, the entry inscribed in every cell indicates the value of the boolean function at the corresponding state. However in our proposed approach, the entry inscribed in each cell in the Q-map indicates change of state of the function. An entry of 1 in the Q-map indicates that the function must change state (i.e toggle) and a value of 0 indicates the function must remain in the same state. Thus including the 1's in the Q-map in an odd number of groups will result in one change of state and including the 0's in even number of groups will result in no change of state. This property of the Q-map could be used in our advantage to maximise the number of Q-map cells in each group and produce a more simplified expression that minimises the quantum cost of the design.

The optimised Q-Map algorithm can be summarised as follows:

**Step 1:** For each qubit  $q_i$  build the toggle function  $T(q_i) = q_i \oplus q'_i$  as the logical Exclusive-OR of the present state  $q_i$  and the final state  $q'_i$ .

**Step 2:** Establish the Q-Map of the switching function  $T(q_i)$  as a function of the  $n$  qubits  $(q'_0, q'_1, \dots, q'_{i-1}, q_i, \dots, q_{n-1})$ .

**Step 3:** In each cell inscribe the value of  $T(q_i)$  in the Q-Map.

**Step 4:** Find the expression of  $T(q_i)$  in Sum-Of-Product form using the Q-Map such that:

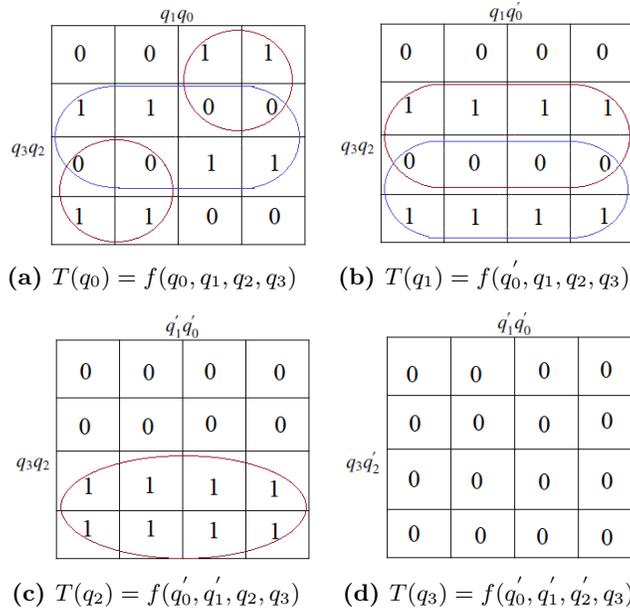
1. Groups should be as large as possible and may include 1's and 0's.
2. Every 1 must be included in an odd number of groups
3. If a 0 is included, it must be included in an even number of groups

**Step 5:** For each expression  $T(q_i)$  use the **CNOT-NOT-Toffoli** bases to implement the corresponding quantum circuit.

To demonstrate the idea, we re-evaluate the functions produces in the Q-map of Figure 7 as shown in Figure 12 and the functions presented in Eq. 10 can be replaced by :

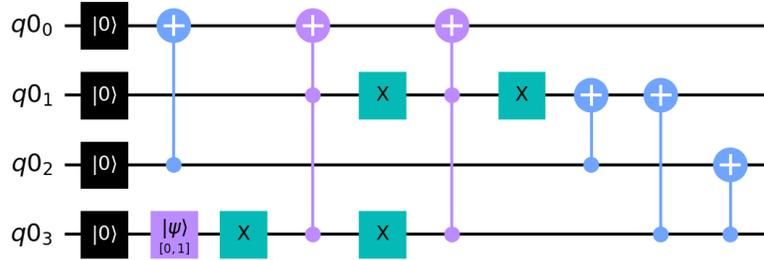
$$\begin{array}{l}
 T(q_0) = \bar{q}_3 q_1 + q_3 \bar{q}_1 + q_2 \\
 T(q_1) = q_3 + q_2 \\
 T(q_2) = q_3 \\
 T(q_3) = 0
 \end{array} \tag{11}$$

Other alternative designs are possible, for example,  $T(q_1)$  can be also written as  $\bar{q}_2 + \bar{q}_3$ .



**Figure 12.** Optimised Q-Map representing the switching function for qubits  $q_0, q_1, q_2$  and  $q_3, T(q_i)$ . Every entry of 1 must be included in an odd number of groups and an entry of 0 must be included in an even number of groups.

Based on the functions in Eq.11, the quantum circuit can be designed as presented in Figure 13 and the corresponding QISKIT code is shown in Figure 14. The optimised approach re-designed the quantum circuit with no ancillary bits and significantly reduced the number of quantum gates.



**Figure 13.** Implementation of the optimised Quantum circuit using QISKIT

## Conclusion

In this paper we propose a visual method for quantum realisation of classical Boolean logic functions. The proposed method utilise concepts from Boolean algebra to produce a quantum circuit with minimal number of quantum gates. The proposed technique is composed of three steps: (1) for each quantum bit  $q_i$  build a switching function  $T(q_i)$ , (2) establish the Q-Map for  $T(q_i)$  and (3) using the Q-Map find the quantum expression to implement the switching function using the **NOT-CNOT-Toffoli** quantum gate basis. The proposed method is demonstrated by implementing the Gray-Code Encoder.

```

q = QuantumRegister(4)
c = ClassicalRegister(4)
grayEncoder = QuantumCircuit(q,c)

# Reset input qbit q[0], q[1], q[2],q[3]
grayEncoder.reset(q[0])
grayEncoder.reset(q[1])
grayEncoder.reset(q[2])
grayEncoder.reset(q[3])

#===== calculate T(q0) =====
grayEncoder.cx(q[2],q[0]) # CNOT gate (T(q0) = q2)
grayEncoder.x(q[3]) # NOT gate
grayEncoder.ccx(q[1],q[3],q[0]) # TOFOLLI gate (T(q0) = q1.!q3)
grayEncoder.x(q[3]) # NOT gate
grayEncoder.x(q[1]) # NOT gate
grayEncoder.ccx(q[1],q[3],q[0]) # TOFOLLI gate T(q0) = !q1. q3
grayEncoder.x(q[1]) # NOT gate

#===== calculate T(q1) =====
grayEncoder.cx(q[2],q[1]) # TOFOLLI gate T(q1) = q2
grayEncoder.cx(q[3],q[1]) # TOFOLLI gate T(q1) = q3

#===== calculate T(q2) =====
grayEncoder.cx(q[3],q[2]) # CNOT gate

```

Figure 14. Code to implement of the Quantum circuit in Fig. 13

## References

1. Learn quantum computation using qiskit, May 2023.
2. F. Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 2019.
3. E. Barbeau. *Pell's Equation*. Problem Books in Mathematics. Springer, 2003.
4. J. Buchmann and H. C. Williams. On the existence of a short proof for the value of the class number and regulator of a real quadratic field. *in: Richard A. Mollin (ed.), Number Theory and Applications, (NATO — Advanced Study Institute, Banff, 1988 ) Dordrecht: Kluwer*, pages 327–345, 1989.
5. T. T. E. Fredkin. Conservative logic. *International Journal of Theoretical Physics*, 21:219–253, 1982.
6. D. J. Egger, C. Gambella, J. Marecek, S. McFaddin, M. Mevissen, R. Raymond, A. Simonetto, S. Woerner, and E. Yndurain. Quantum computing for finance: State-of-the-art and future prospects. *IEEE Transactions on Quantum Engineering*, 1:1–24, 2020.
7. K. Fazel, M. A. Thornton, and J. E. Rice. Esop-based toffoli gate cascade generation. *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 206–209, 2007.
8. L. K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325–328, Jul 1997.
9. S. Hallgren. Polynomial-time quantum algorithms for pell's equation and the principal ideal problem. *J. ACM*, 54(1):4:1–4:19, Mar. 2007.
10. H. Iqbal and W. O. Krawec. High-dimensional semiquantum cryptography. *IEEE Transactions on Quantum Engineering*, 1:1–17, 2020.
11. G. D. James. *The representation theory of the symmetric groups*, volume 682. Springer, 2006.

12. P. Kerntopf. A new heuristic algorithm for reversible logic synthesis. In *Proceedings of the 41st Annual Design Automation Conference, DAC '04*, page 834–837, New York, NY, USA, 2004. Association for Computing Machinery.
13. T. Krauss and J. McCollum. Solving the network shortest path problem on a quantum annealer. *IEEE Transactions on Quantum Engineering*, 1:1–12, 2020.
14. R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 44(1.2):261–269, Jan 2000.
15. S.-Y. Ma, A. Khalil, H. Hajjdiab, and H. Eleuch. Quantum dilation and erosion. *Applied Sciences*, 10(11), 2020.
16. T. K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms, 2nd Edition*.
17. R. D. Nabila Abdessaied. *Reversible and Quantum Circuits*.
18. C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. V. Isakov, V. Smelyanskiy, A. Megrant, B. Chiaro, A. Dunsworth, K. Arya, R. Barends, B. Burkett, Y. Chen, Z. Chen, A. Fowler, B. Foxen, M. Giustina, R. Graff, E. Jeffrey, T. Huang, J. Kelly, P. Klimov, E. Lucero, J. Mutus, M. Neeley, C. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, H. Neven, and J. M. Martinis. A blueprint for demonstrating quantum supremacy with superconducting qubits. *Science*, 360(6385):195–199, 2018.
19. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.
20. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, Feb. 1978.
21. K. Rosen. *Discrete Mathematics and Its Applications*.
22. V. Shende, A. Prasad, I. Markov, and J. Hayes. Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, 2003.
23. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, Oct. 1997.
24. M. Soeken, L. Tague, G. W. Dueck, and R. Drechsler. Ancilla-free synthesis of large reversible functions using binary decision diagrams. *Journal of Symbolic Computation*, 73:1–26, 2016.
25. M. Soeken, R. Wille, C. Otterstedt, and R. Drechsler. A synthesis flow for sequential reversible circuits. In *Proceedings of the 2012 IEEE 42nd International Symposium on Multiple-Valued Logic, ISMVL '12*, page 299–304, USA, 2012. IEEE Computer Society.
26. A. tA v, M. S. ANIS, Abby-Mitchell, H. Abraham, AduOffei, R. Agarwal, G. Agliardi, M. Aharoni, V. Ajith, I. Y. Akhalwaya, and et. al. Qiskit: An open-source framework for quantum computing, 2021.
27. R. J. Tocci and N. S. Widmer. *Digital systems.*. Prentice Hall, Upper Saddle River, N.J., 9th edition edition, 2004.
28. A. D. Vos and Y. V. Rentergem. Young subgroups for reversible computers. *Advances in Mathematics of Communications*, 2(2):183–200, 2008.
29. R. Wille and R. Drechsler. Bdd-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference, DAC '09*, page 270–275, New York, NY, USA, 2009. Association for Computing Machinery.
30. C. P. Williams. *Explorations in Quantum Computing, Second Edition*. Texts in Computer Science. Springer, 2011.

- 
31. N. S. Yanofsky and M. A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.