Quantum Inf Process (2008) 7:175–192
DOI 10.1007/s11128-008-0081-x

# Heuristic methods to use *don't cares* in automated design of reversible and quantum logic circuits

**Majid Mohammadi · Mohammad Eshghi**

Published online: 23 August 2008

**Abstract** This paper introduces a broad concept of *don't cares* in reversible and quantum logic circuits. *Don't cares* are classified into three categories: inputs, outputs, and conditions. Some heuristic methods to use these *don't cares*, when an optimization algorithm such as genetic algorithm is used, are also presented. We show that, these methods decrease the quantum cost of the reversible or quantum logic circuit, as well as the design time of the resulting circuit. Some examples are also synthesized and optimized using the *don't care* concept and genetic algorithms.

**Keywords** Synthesis · Quantum circuit · Reversible logic · Optimization · Genetic algorithm

**PACS** 03.67.Lx · 03.67.Ta

## 1 Introduction

In 1961, Landauer [8] proved that irreversible processing of information in a logic circuit can cause energy dissipations. This energy dissipation is equal to kTln 2 joules per bit of information loss. Bennet [2] introduced reversible processing and showed if the hardware and software of a digital computer are designed using the reversible scheme, then the energy dissipation due to the irreversibility can be arbitrarily decreased. Therefore, there are forceful reasons to consider circuits composed of reversible

M. Mohammadi (✉) · M. Eshghi
Shahid Beheshti University, Tehran, Iran
e-mail: m_mohamadi@sbu.ac.ir

M. Eshghi
e-mail: m-eshghi@sbu.ac.ir

gates. Reversible circuits are of particular interest in low power CMOS design, optical computing, quantum computing, and nanotechnology based systems.

Synthesis of reversible circuits differs significantly from synthesis using traditional irreversible gates. Two restrictions are added to reversible circuits, namely fanout and feedback are not allowed. Many algorithms have been proposed for synthesis of reversible circuits [5,7,10,13]. Miller et al. [13] proposed a bidirectional, transformation-based algorithm in 2003. Kerntopf [7] proposed a synthesis algorithm based on binary decision diagram (BDD) for logical reversible circuits. Gupta et al. [5] presented an algorithm using the positive polarity Reed–Muller expansion of a reversible function to synthesize the function as a network of Toffoli gates. They used generalized Toffoli gates (TOFn) to synthesize the reversible logic circuits.

Some automated reversible logic synthesis methods using optimization algorithms, such as genetic algorithms (GAs) are also presented [10,11,14]. In this approach, the global optimization characteristic of GA [3] is used to synthesize reversible circuits, to obtain optimized or near optimized circuits.

When defining logical functions, there are inputs whose corresponding outputs are not determined. That is, the truth table of the function is not completely defined. Conventionally, these patterns are called “don’t care conditions”. In reversible or quantum logic circuits, there are also some inputs or outputs whose values are not important. These extra inputs or outputs are usually added to the circuit to maintain the reversibility condition. Traditionally, these added inputs and outputs are called “constant inputs” and “garbage outputs”, respectively [12].

Although, *don’t cares* (*DC*s) in reversible logic circuits are referenced in some papers [4,15], the usage of *DC*s to obtain a minimal reversible circuit has not been covered. In [14] we have proposed a method for using *DC* conditions to optimize a reversible full adder. In this paper a broad concept of *DC*s is introduced and they are classified to three categories. Some new heuristic methods to optimize a reversible or quantum circuit using *DC*s are also proposed. We also present a behavioral description of quantum V and V+ gates which allows us to construct a truth table for these quantum gates.

The organization of this paper is as follows. Section 2 covers the background including reversible logic circuits, genetic algorithms and their application in the synthesis of reversible circuits. The main topic of the paper, *DC*s in reversible and quantum logic circuits, is presented in Sect. 3. In Sect. 4, the proposed method is applied to some examples and their results are presented.

## 2 Background

### 2.1 Reversible *gates* and *circuits*

A function or a circuit is called reversible if there is a one-to-one correspondence between its input and output assignments. If a reversible function is shown by a truth table, then its output patterns must be the permutation of its input patterns. Conventional AND, OR, and XOR gates are not reversible. A reversible gate has equal number of inputs and outputs. Generally, there are $2^n!$ reversible gates for $n$ Inputs.

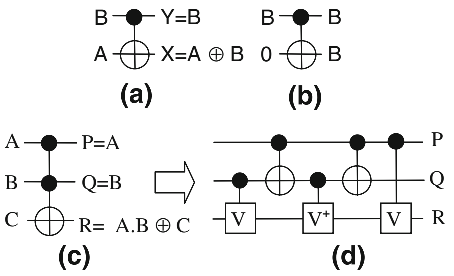

**Fig. 1** (**a**) Feynman gate. (**b**) Copying a signal using Feynman gate. (**c**) Toffoli gate. (**d**) Quantum implementation of Toffoli gate

The well-known 2 × 2 Feynman gate, also known as CNOT, is shown in Fig. 1a. It operates as a controlled NOT. If its control input (B) is '0', then the gate acts as a BUFFER gate. If its control input is '1' the gate acts as a NOT gate. In reversible circuits, a signal can be copied using the Feynman gate, if needed (Fig. 1b).

Figure 1c shows Toffoli 3 × 3 gate. This gate is universal, i.e. any reversible logic circuit can be implemented using this gate.

Toffoli gate is generalized to $n$ inputs and $n$ outputs, named TOFn gate [13]. This gate has $n - 1$ control inputs and one target (or main) input/output. If all control inputs are '1', then the target output is the NOT of the target input; otherwise the target output is equal to the target input. Using TOFn representation, The TOF2 and TOF1 gates can be used instead of Feynman and NOT gates in the circuits, respectively. Other specialized reversible logic gates are also proposed in papers such as [6] in which a special reversible gate is proposed to design a reversible full adder.

## 2.2 Quantum gates and circuits

A quantum logic gate is a basic quantum circuit operating on a small number of qubits. Quantum logic gates are reversible, unlike many classical logic gates. They are represented by unitary matrices. The most common quantum gates operate on spaces of one or two qubits. Quantum gates can be described by 2 × 2 or 4 × 4 matrices with orthonormal rows. Some examples of quantum gates are Feynman, Toffoli, Fredkin, Hadamard, and Phase Shifter gates. Some reversible logic gates, such as the Feynman, Toffoli, and Fredkin gates are directly mapped onto quantum logic gates [1].

The controlled-U gate is a gate that operates on two qubits in such a way that the first qubit serves as a control (Eq. 1).

$$C(U) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & U_{10} & U_{11} \end{bmatrix} \tag{1}$$

Figure 1d shows a quantum implementation of a 3 × 3 Toffoli gate. In this figure, a V gate indicates the state of the qubits is multiplied by a 2 × 2 matrix (Eq. 2).

$$V = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^{0.5} = \frac{1+i}{2}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \tag{2}$$

The V gate is also named square root of NOT gate ($\sqrt{\text{NOT}}$). $\text{V}^+$ gate is the complex conjugate transpose of V. The V and $\text{V}^+$ quantum gates have some properties that are shown in Eq. 3.

$$\begin{cases} \text{V} \times \text{V} = \text{NOT} \\ \text{V} \times \text{V}^+ = \text{V}^+ \times \text{V} = \text{I} \\ \text{V}^+ \times \text{V}^+ = \text{NOT} \end{cases} \tag{3}$$

These equations show that two V gates in series or two $\text{V}^+$ gates in series are equivalent to the NOT gate; and two V and $\text{V}^+$ in series, are equivalent to an identity or a BUFFER gate.

A quantum circuit consists of quantum gates which are interconnected without fanout or feedback by quantum wires. It can be seen as a classical circuit along with quantum data [1]. Classic reversible logic circuits can be realized using quantum gates. This is useful whenever a classical process is needed in a quantum computer. The set of quantum V, $\text{V}^+$, and Feynman gates is a universal set for reversible logic circuits [1].

Quantum cost (QC) is used to measure the complexity of a reversible or quantum circuit [9]. The QC of a reversible circuit is defined as the number of $1 \times 1$ or $2 \times 2$ reversible quantum or logic gates that are needed to realize the circuit. $1 \times 1$ and $2 \times 2$ quantum or logic gates are primitives of quantum circuits. The QC of a primitive gate is one, regardless of its internal structure. The Toffoli gate is realized by a minimum of five $2 \times 2$ gates (Fig. 1d); therefore, its QC is five.

## 2.3 Automated synthesis of reversible or quantum logic circuits using genetic algorithm

The automated synthesis of a reversible circuit using a searching algorithm, such as GA, is a complex problem having a huge searching space. The number of combinations for placing one Toffoli $r \times r$ gate in an $n \times n$ circuit $(0 < r < n - 1)$, to synthesize a reversible $n \times n$ circuit is expressed by Eq. 4.

$$\rho = n \cdot \sum_{r=0}^{n-1} \binom{n-1}{r} = n \cdot 2^{n-1} \tag{4}$$

If the number of gates required to design a circuit is $m$, then the number of possible circuits is $\rho^m$. If the quantum controlled gates V and $\text{V}^+$ are added to the set of Toffoli $r \times r$ gates, then the number of possible circuits is $(3\rho)^m$.

Genetic algorithms (GAs) have shown their efficiency in optimization and finding the global minimum or maximum of a function, in an extensive searching space [3]. They are also used in the synthesis of reversible logic circuits [10,11,14]. In the next subsections, a review of GA, the circuit coding, and the fitness function are presented.

### *2.3.1 Genetic algorithm*

Genetic algorithm is one of the well-known evolutionary algorithms in Soft Computing. Although GA is an optimization algorithm, it is also used in the optimal synthesis of reversible logic circuits.

First, the variables of the search space have to be coded to a string of bits, named the chromosome. Then, a population of chromosomes is produced and three basic GA operators, as follows, are applied to them:

(a) *The crossover operator* selects two chromosomes randomly and exchanges some of their corresponding segments.
(b) *The mutation operator* applies random changes to the selected chromosome, with a specific probability, by randomly inverting some of its bits.
(c) *The selection operator* selects some of the *good* chromosomes for reproduction of the next population. Selection directs the algorithm toward the optimum.

The genetic algorithm for reversible logic synthesis is shown in Fig. 2. First, a population of chromosomes (circuits) is generated and initialized randomly. This population has $\mu$ (between 20 and 100) members. Two operators of GA, mutation and crossover, are used to generate $\lambda$ new members, named offspring. Now, the population has $\mu + \lambda$ members.

Then the fitness function is calculated for any chromosome. The selection operator selects $\mu$ from $\mu + \lambda$ individuals. These $\mu$ selected individuals are considered the parent set of the next generation. The algorithm continues until the fitness function reaches an acceptable value.

### *2.3.2 Circuit coding*

In the synthesis of a circuit using GA, each chromosome is a coded circuit. Choosing a proper style to represent a circuit is an essential step in coding. To represent a reversible circuit, two styles are available. One style is to show the circuit using some individual gates connected together by wires. Another style shows the circuit using a series of connected gates on some parallel lines similar to the music lines. These lines are the

**Fig. 2** The synthesis genetic algorithm

```
Initiate a random population of μ individuals;
//Each individual is a quantum or reversible circuit
while termination condition not met
  {
  Produce λ new individuals {circuits} by crossover
          and mutation
  Evaluate the fitness of each of λ+μ individuals
  Select μ better {less error} individuals for
          reproduction
  }
```

**Fig. 3** Coding of a reversible gate

| Gate Type | Main I/O | Ctrl. I/O 1 | Ctrl. I/O 2 | ... | Ctrl. I/O r |
|---|---|---|---|---|---|

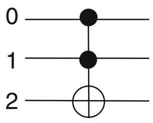

**Fig. 4** An example for coding of a Toffoli gate

**Fig. 5** A chromosome of a circuit with *m* gates

| Gate1 | Gate2 | ... | Gate *m* |
|---|---|---|---|

inputs/outputs of the circuit. The reversible gates are placed on these parallel lines. Using this style, designing of a reversible circuit is similar to composing a music piece. Figure 7a–d are samples of the music line style presentation of circuits.

To code a circuit, it is presumed as an $n \times n$ circuit with $m$ gates, where $n$ is the number of inputs/outputs (number of parallel lines) and $m$ is number of columns or gates are placed on the parallel lines. We assume that only one reversible gate can be placed on each column.

As shown in Fig. 3 each gate is coded with some fields in it. The first field indicates the type of gate. The second field shows the binary number of the location of its main input/output. If the music lines are numbered from 0 to $n - 1$, this code shows the line number of the main output.

The next $r$ fields show the binary number of the location of gate inputs. For example if a $3 \times 3$ Toffoli gate is placed on three music lines as shown in Fig. 4, its code is 01,10,00,01 (01 is assumed the code of Toffoli gate).

One chromosome which contains $m$ gates, can completely code a circuit (Fig. 5).

### 2.3.3 Fitness function

The Fitness function (*FF*) has many aspects of the circuits' optimality. To achieve an optimum circuit, the *FF* should be maximized. Since the *FF* has to be maximized, the reciprocal of the error is used in some papers [10,11] as *FF* (Eq. 5).

$$FF = \frac{1}{1 + \text{Error}} \tag{5}$$

In this paper, GA is designed to minimize a desired function, such as the *error function* (*EF*). The Hamming distance (*HD*), shown in Eq. 6, is a good candidate for the error function,

$$EF = HD = \sum_i \sum_j \left| O_{ij} - S_{ij} \right|, \tag{6}$$

where $O_i$ is a row vector of the desired truth table and $S_i$ is a row vector of the truth table of the synthesized circuit. $O_{ij}$ is the $j$th element of the vector $O_i$ and $S_{ij}$ is the

$j$th element of the vector $S_i$. As suggested in some papers [10,11], the number of gates, used to synthesize the circuit, can be added to the error function (Eq. 7).

$$EF = \alpha \cdot HD + \beta \cdot m, \tag{7}$$

where $m$ is the number of gates and $\alpha$ and $\beta$ are weighting factors, specifying the significance of each item. Note that for a valid circuit, the *HD* must be zero; thus, adding $m$ to the error function may result in an invalid circuit. We present a solution for this problem by presenting a modified genetic algorithm in Sect. 3.5.

## 3 Using *DC*s in designing reversible and quantum logic circuits

### 3.1 Classifying the *DC*s in the reversible logic circuits

Among the input patterns of a logic function, there are some patterns whose corresponding outputs are insignificant. This also occurs in traditional irreversible logic functions. The designer can use these conditions to achieve a better design. These conditions appear in reversible logic circuits, but can not be handled in the same manner as the irreversible case. Since the desired function must be one-to-one, all function values have to be defined at the start of the synthesis process. As a result, the designers assign some values to the *DC conditions* using their experience or a trial and error process [13].

Three types of *DC*s can be recognized in a reversible logic circuit: *DC* inputs, *DC* conditions and *DC* outputs. The *DC* inputs are additional bits that are added to the input part of the truth table to keep the reversibility of the function. Traditionally, they are named “constant inputs” because their values can not be varied in the circuit. When GA assumes these inputs as *DC* inputs, it has permission to specify their values to achieve the minimum circuit. Table 5 illustrates this type of *DC*s. In this table R1, R2, R3, and R4 are *DC* inputs.

The *DC* conditions correspond to the rows of the truth table whose main (not garbage) outputs are *DC*s. When the values of some rows of the truth table are not specified or are not important, they are assumed as *DC* conditions. These conditions occur when some *DC* inputs for the function are assumed; however, this is not the whole case. In other words, a circuit may have *DC* conditions while it doesn't have any *DC* inputs. Example 2 shows this type of *DC*s.

The *DC* outputs are additional outputs whose values are not important in any row of the truth table. They may be added to the circuit to maintain its reversibility. Conventionally, they are called garbage outputs. Table 4 illustrates this type of *DC*s. In this table G1 and G2 are *DC* outputs.

### 3.2 Heuristic method for using *DC* outputs to obtain an optimum circuit

Any $n \times n$ reversible function can be considered as $n$ functions of $n$ input variables (Fig. 6) with reversibility restrictions. The goal of synthesis is to design a circuit whose outputs have the same functions as the desired functions, using the reversible gates.

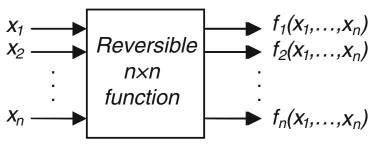

**Fig. 6** An $n \times n$ reversible function can be considered as $n$ functions $f_1, \ldots, f_n$ of inputs $x_1, \ldots, x_n$

The functions $f_1$ to $f_n$ have to satisfy the reversibility of the desired functions for all input vectors and the synthesized circuit has to have the fanout and loop limitations for reversible circuits.

When a reversible function has some *DC* outputs, it means that some of these $f_1$ to $f_n$ are not specified or their values are not significant. Without loss of generality we assume that the reversible function has only one main output ($f_1$) and $n - 1$ *DC* outputs ($f_2, \ldots, f_n$). In this case, it is enough to design one function ($f_1$) using the synthesis tool. The designed circuit for $f_1$ is simpler than a circuit which implements all $f_1$ to $f_n$ functions. The circuit for $f_1$ is not more complex than the circuit which implements whole functions; because $f_1 - f_n$ circuit is also an answer to the $f_1$ circuit. Actually, the worst case occurs when the obtained circuit for $f_1$ alone is the same as the circuit which implements whole functions.

If we restrict the synthesis algorithm to implement $f_1$, can we guarantee the reversibility of the obtained circuit? The answer is affirmative, because our circuits are composed of reversible gates which are placed on music line style (Sect. 2.3.2). In these circuits, designed by GA, the reversibility restrictions, namely fanout and loop limitations, are also guaranteed.

Therefore to obtain a simpler circuit for a function which has *DC* outputs, we limit the synthesis algorithm to implement the care output functions and discard the *DC* outputs. To do this, we ignore the values of *DC* outputs in "Hamming Distance" as the error function (Sect. 2.3.3). A simple way to ignore the error for *DC* outputs is using an n-bit string, named *mask*. We consider that the mask pattern has '0' value in *DC* output positions and '1' in the *care* output positions. Then, we apply this mask pattern to the outputs of desired function, using the AND operation. At any iteration of GA, we also apply the mask to outputs of the designed circuit. Consequently, the values for all *DC* outputs become zero and hamming distance for these outputs is also zero. For example if mask pattern is "000, . . . , 01" then just the $f_1$ value will appear in the masked output.

*Example 1* As a synthesis example, consider a $4 \times 4$ function F1 whose output values are:

$$F1 = \{0, 5, 2, 13, 4, 1, 6, 9, 7, 8, 15, 10, 3, 12, 11, 14\}.$$

When there is no *DC* output in the circuit the mask pattern is "1111". The obtained circuit is shown in Fig. 7a. This circuit has 5 gates with QC of 17. With mask="0111" (i.e. one *DC* output), mask="0011" (two *DC* outputs), and mask="0001" (three *DC* outputs), we obtain circuits in Fig. 7b, c, and d, respectively. QCs for these circuits are 8, 7, and 1, respectively. Note that the values of corresponding *DC* outputs in circuits 1a–d are not the same but the $Q_0$ in all circuits has the same function.

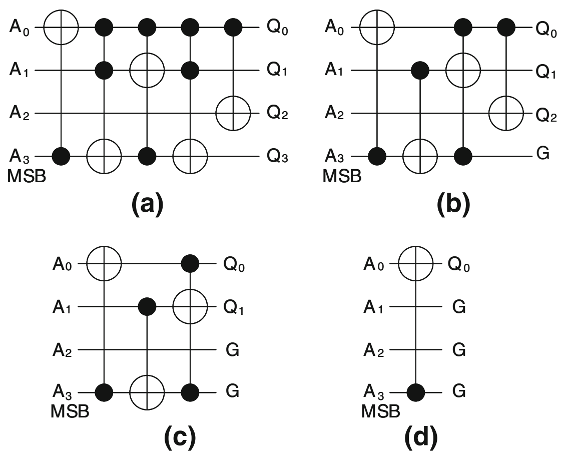

**Fig. 7** Synthesis of Example 1 with (**a**) mask="1111", (**b**) mask="0111", (**c**) mask="0011", and (**d**) mask="0001"

## 3.3 Using *DC* conditions to optimize the circuit

In two cases the *DC* conditions may occur in reversible logic functions. The first case is when the *DC* conditions are embedded in the function of the circuit. The second case is when the circuit has constant inputs (called *DC* inputs in this paper). This case is described in Sect. 3.4.

In the first case, the *DC* conditions are embedded in the function. For example, a BCD adder has defined outputs for input digits 0–9. The outputs for input values of 10–15 are embedded *DC* conditions; because the decimal digits are not bigger than 9. The important problem with this type of functions is that the *DC* conditions have to be assigned in such a way that the resulting function be reversible. In previously proposed synthesis methods [5,7,13] that can not handle the *DC* conditions, these conditions have to be assigned by a trial and error process or designer's experience. This is not applicable for functions with large number of *DC* conditions because for $r$ *DC* conditions, there are $r!$ combinations to select them. It is important to note that the problem of *DC* assignment becomes more complicated when the number of *DC* conditions in the truth table increases.

Assume that the truth table of an $n \times n$ function, having $2^n$ rows, has one care condition for input "00, . . . , 0" (first row) and the rest of $2^n - 1$ rows as *DC* conditions. This function can be simply implemented using some NOT gates. For example, consider a $4 \times 4$ function which has "1111" output pattern for the first row of its truth table, and all other rows are *DC* conditions. This function can be implemented by only four NOT gates (one NOT on each line). If we assume two care conditions, the synthesis problem becomes slightly more complex and a bigger circuit is needed to implement the function. Increasing the care conditions incorporates more complexity to the circuit. Therefore, if we can restrict the synthesis tool to implement the care conditions and ignore the *DC* conditions, then we can obtain a simpler circuit. The worst case is when the obtained circuit for care conditions of truth table is the same as the circuit which implements the whole truth table. The circuit for only care conditions is not more complex than the circuit which implements all rows of truth table because this circuit is also an answer to the first case. As stated before, the music line style presentation of circuit not only guarantees the reversibility of the circuit but also assures the fanout and loop conditions for reversible circuit.

In order to obtain the optimum circuit we restrict the synthesis algorithm to design a circuit which satisfies only the care conditions of the truth table. To do this, we ignore the values of *DC* conditions in “Hamming Distance” as the error function. We design the synthesis algorithm such that it jumps over the *DC* rows of truth table. We insert ‘x’ characters in *DC* rows to indicate that these rows are *DC* conditions. Using this method, we not only obtain a simpler circuit, but also obtain it in less synthesis time. Since the synthesis algorithm jumps over the *DC* rows, it performs each of iterations faster. As a result, the total time needed to synthesize a circuit is decreased.

Note that, in the trial and error method to assign the *DC* conditions, the more *DC* conditions, the more time needed to design the circuit, and the more quantum cost. However, in our proposed method, the more *DC* conditions, the less time needed to synthesis the circuit, and the fewer quantum cost.

*Example 2* As a synthesis example, consider the 4 × 4 function F2 defined as:

F2 = {0, 2, 6, 4, 9, 11, 1, 8, 12, 13, 10, 3, 15, 14, 7, 5}

The designed circuit for this function is shown in Fig. 8a. This circuit has 8 gates with QC of 24. Now, consider the above function with 8 *DC* conditions as

F3 = {0, 2, 6, 4, 9, 11, x, x, x, x, x, x, x, x, 7, 5}.

The synthesized circuit of F3 is shown in Fig. 8b. This circuit has only two gates with QC of 2.

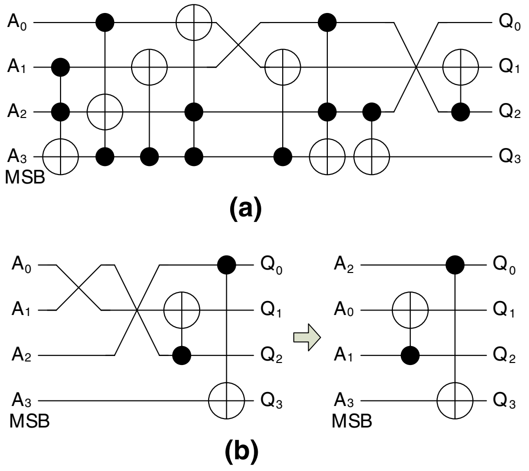

**Fig. 8** Synthesis of Example 2: (**a**) Without *DC* conditions and (**b**) With eight *DC* conditions

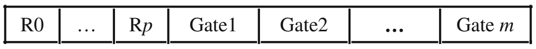

**Fig. 9** Modified chromosome for *DC* inputs

## 3.4 Constant inputs

*Constant inputs* or *DC inputs* are constant; and they can not be masked or skipped as the other types of *DC*s. The values of these inputs affect on the generated circuit; therefore, choosing the best values for these inputs is important. In this research, to obtain the optimum values for these *DC*s, they are inserted in the chromosomes of genetic algorithm. Figure 9 shows a chromosome that codes a circuit and includes the $p$ *DC* inputs.

Initial values of these inputs are random, as the other bits of the chromosome. During the synthesis process, these values are affected by GA operators: crossover, mutation, and selection. Thus, in evolution of the circuit, the best values for R0, . . . , R$p$ are also achieved.

## 3.5 Modified synthesis algorithm

As mentioned in Sect. 2.3.3, if the number of gates ($m$) is added to the error function, GA may converge to an invalid result. On the other hand, to obtain a circuit with minimum number of gates, $m$ must be included in the synthesis algorithm. To solve this problem the synthesis algorithm is modified and shown in Fig. 10.

In this algorithm *Error* function is hamming distance as defined in Sect. 2.3.3. We also modified the algorithm to handle the *DC* outputs and conditions. In this algorithm des_row[$i$] is the $i$th row of desired truth table, and syn_row[$i$] is the $i$th row of truth table of synthesized circuit. The '&' sign represents the bitwise AND operation.

Initially, $m$ is set to 1, i.e. synthesis algorithm starts with one gate. In the *while* loop the GA tries to find a valid circuit for the desired truth table, by using one gate. If GA can find a valid circuit (Error = 0), it is the answer. Otherwise, $m$ is incremented and the *while* loop starts again, using two gates. This process continues until a valid circuit (Error = 0), is obtained. Since the synthesis algorithm starts with $m = 1$, the answer is a circuit with the minimum possible number of gates. The minimum number of gates is not necessarily the minimum quantum cost (QC). To obtain a circuit with minimum QC, the algorithm have to be restricted to the $1 \times 1$ and $2 \times 2$ quantum gates.

The *while* loop is limited to a maximum number of iterations, which depends on the complexity of the circuit; and it can be obtained by a trial and error process. Since the optimization is offline, this is not a sever dilemma.

## 3.6 Behavioral description of V and V+ gates

Quantum gates have been specified by their unitary matrices, often including complex numbers. Defining the quantum gates using a truth table is not common. On the other hand, the quantum V and V$^+$ gates are necessary to construct a set of universal $2 \times 2$ quantum gates in synthesis process.

**Fig. 10** Modified synthesis algorithm

```
Set m := 1  {m is the number of gates being used for
        synthesis}
Loop1:
Initiate a random population of μ circuits using m gates;
        //each circuit is a Quantum or Reversible circuit
While  ( up to max-iteration)
   {
   Produce λ new circuits using crossover and mutation
   For all of λ + μ circuits
      {
      For i-th row of truth table
          {
          If  i-th row is not "don't care condition"then
          Error = Error + Abs ( HD ( (des_row [ i ]  &
             mask), ( syn_row [ i ] & mask ) ) );
           // mask bits are '1' for a care output
           // and ' 0' for a don't care output
          }
      }
   Select μ better {less error} circuits for reproduction
   } // End of while
If Minimum Error is not 0 then
    { Increment m; goto loop1; }
Print the best circuit
```

To construct a truth table for V and V$^+$ gates, we use properties of these gates, Eq. 3. This equation shows that two V gates, or two V$^+$ gates, in series are equivalent to a NOT gate. One V gate and one V$^+$ gate in series are equal to identity. We define two intermediate signals $v$ and $V$ for V gate output values and two intermediate signals $w$ and $W$ for V$^+$ gate output values. We assign $v$ value to the V gate output if its input is $|0\rangle$ and $V$ value to the V gate output if its input is $|1\rangle$. In the same way, we assign $w$ and $W$ values to the output values of a V$^+$ gate, when input values are $|0\rangle$ and $|1\rangle$, respectively. If the input of V gate is $v$ then the output is $|1\rangle$ because the $v$ value is the result of applying input $|0\rangle$ to a V gate and two series V gates is equal to a NOT gate. In the same way, applying $V$ input to a V gate results a $|0\rangle$ output. If the input of a V gate is $w$ then its output is $|0\rangle$ because a $w$ signal is a result of applying $|0\rangle$ to a V$^+$ gate and two V and V$^+$ series gates are equal to an identity or buffer gate. These results can also be stated for a V$^+$ gate. Based on these results we can construct a truth table for V and V$^+$ gates, shown in Table 1. In this table, for simplicity, we use '0' and '1' characters instead of $|0\rangle$ and $|1\rangle$ vectors, respectively.

Controlled V and V$^+$ gates (Fig. 11) are also represented by truth tables, shown in Table 2. Note that intermediate states ($v$, $V$, $w$, and $W$) are not valid for control inputs.

**Table 1** Truth table of V and $V^+$ gates

| A (input) | Q (V gate) | Q ($V^+$ gate) |
|---|---|---|
| 0 | $v$ | $w$ |
| 1 | $V$ | $W$ |
| $v$ | 1 | 0 |
| $V$ | 0 | 1 |
| $w$ | 0 | 1 |
| $W$ | 1 | 0 |

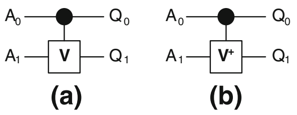

**Fig. 11** Controlled V and $V^+$ gates

**Table 2** Truth table of controlled V and $V^+$ gates

| $A_0$ (control) | $A_1$ | $Q_0$ | $Q_1$ (V gate) | $Q_1$ ($V^+$ gate) |
|---|---|---|---|---|
| 0 | X | 0 | X | X |
| 1 | 0 | 1 | $v$ | $w$ |
| 1 | 1 | 1 | $V$ | $W$ |
| 1 | $v$ | 1 | 1 | 0 |
| 1 | $V$ | 1 | 0 | 1 |
| 1 | $w$ | 1 | 0 | 1 |
| 1 | $W$ | 1 | 1 | 0 |

We use this truth table of V and $V^+$ gates in simulation and synthesis of quantum logic circuits.

## 3.7 Optimum location of *DC* outputs and *DC* inputs

In Sects. 3.2 and 3.4 we introduced the heuristic methods in order to handle *DC* outputs and inputs. An important consideration about these inputs and outputs is their position in the truth table. For example, consider the truth table of a full adder, depicted in Table 3. In this table, Gs are garbage outputs, or *DC* outputs, which are placed in the most significant positions. The result of synthesis of this table is shown in Fig. 12a. If we place these *DC* outputs in the least significant positions, then the result is a simpler circuit, shown in Fig. 12b. Thus, not only the value but also the location of a *DC* input and output is important in synthesis process. We can automatically specify the best location of *DC* outputs and inputs by inserting a SWAP gate in the gate set of the synthesis tool. The SWAP gate exchanges the location of two wires in the circuit which is represented in music line style.

When GA searches the best circuit, it inserts SWAP gates in different locations in the circuit. This is equivalent to swap the position of *DC* inputs and outputs.

**Table 3** Truth table of full adder

| Gin | a | b | c | G | G | C | S |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | x | x | 0 | 0 |
| 0 | 0 | 0 | 1 | x | x | 0 | 1 |
| 0 | 0 | 1 | 0 | x | x | 0 | 1 |
| 0 | 0 | 1 | 1 | x | x | 1 | 0 |
| 0 | 1 | 0 | 0 | x | x | 0 | 1 |
| 0 | 1 | 0 | 1 | x | x | 1 | 0 |
| 0 | 1 | 1 | 0 | x | x | 1 | 0 |
| 0 | 1 | 1 | 1 | x | x | 1 | 1 |

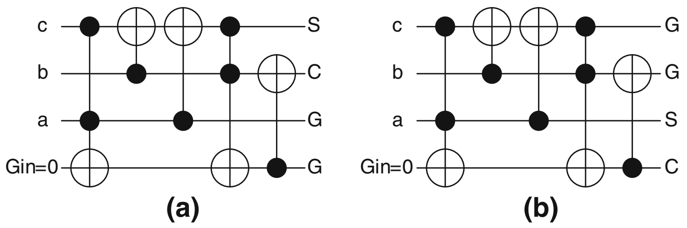

**Fig. 12** Synthesis results for different positions of *DC* outputs as: (**a**) Higher significant bits and (**b**) Lower significant bits

## 4 Tests and results

In this section, four examples which have different combinations of *DC* inputs, conditions and outputs are presented. We have used the QC as a measure to compare the designs with and without *DC*. Another measure which we consider is the real time of synthesis using an ordinary computer (a Pentium IV, 2.8 GHz, 512 MB RAM ) to show that *DC* concept also affects this time.

*Example 3* A circuit with *DC* outputs

As an example for a circuit with *DC* outputs, consider the *majority gate*. A majority irreversible gate has $n$ inputs and one output. Parameter $n$ has to be an odd number. The output of the majority gate is ‘1’ if the number of ‘1’s in the input is more than the number of ‘0’s; otherwise the output is ‘0’. On the other hand, a reversible majority gate has $n$ inputs and $n$ outputs. Only one of its outputs represents the majority function and the other $n - 1$ outputs are considered as *DC* outputs. For example consider a $3 \times 3$ reversible majority gate. Table 4 shows the truth table of this gate. G1 and G2 are the garbage outputs. These are masked in the proposed synthesis algorithm.

At the first attempt the values of 0, 2, 4, 1, 6, 3, 5, and 7 for the eight output rows are considered, respectively. The synthesized circuit for these values is shown in Fig. 13a. At the second attempt the *DC* outputs are masked as stated in Sect. 3.2. The generated circuit is shown in Fig. 13b. The QC for generated circuits is 9 and 7, respectively.

*Example 4* A circuit with *DC* outputs (the quantum implementation)

The *DC* concept can also be used in quantum reversible circuits. In quantum circuits the data bits are vectors named qubits. In this paper we consider qubits as bits,

**Table 4** The truth table of a $3 \times 3$ reversible majority gate

| a | b | c | G1 | G2 | p |
|---|---|---|---|---|---|
| 0 | 0 | 0 | x | x | 0 |
| 0 | 0 | 1 | x | x | 0 |
| 0 | 1 | 0 | x | x | 0 |
| 0 | 1 | 1 | x | x | 1 |
| 1 | 0 | 0 | x | x | 0 |
| 1 | 0 | 1 | x | x | 1 |
| 1 | 1 | 0 | x | x | 1 |
| 1 | 1 | 1 | x | x | 1 |

*Note:* Entries in columns G1 and G2 are *DC* outputs

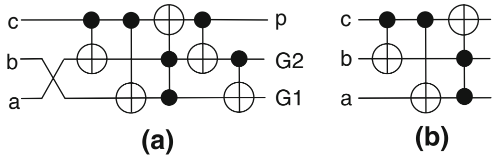

**Fig. 13** A majority $3 \times 3$ gate: (**a**) Without *DC*s and (**b**) With *DC*s

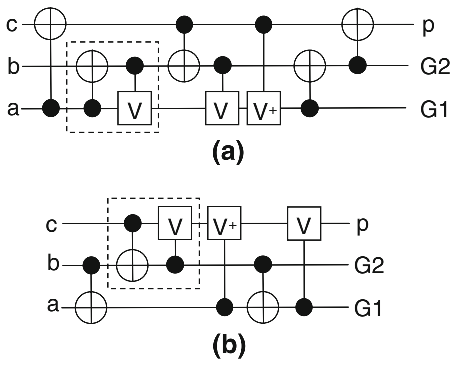

**Fig. 14** A quantum majority $3 \times 3$ gate: (**a**) Without *DC*s and (**b**) With *DC*s

wherever it does not make a mistake. We also use the truth table form of V and V+ gates in simulations and synthesis procedure. For instance, consider the quantum implementation of the majority $3 \times 3$ gate (Table 4) using $2 \times 2$ quantum gates. Two cases are considered; (i) with predefined values (as the first attempt in the Example 3), and (ii) with *DC*s. Figure 14a and b show the generated circuits, respectively. The QCs of these circuits are 7 and 5, respectively.

*Example 5* A circuit with *DC* inputs, outputs and conditions

Consider a $2 \times 2$ *binary multiplier*. This multiplier has two 2-bit inputs and one 4-bit output. The truth table of this multiplier is shown in Table 5. As shown in this table, there are 7 repeated patterns of "0000" outputs. Based on the Maslov's formula [12] at least $\lceil Log_2 r \rceil = 3$ garbage outputs have to be added to the output part of truth table ($r$ is maximum number of repeated patterns in the output). Although, this is the minimum number of *DC* outputs, sometimes it is better to consider more *DC*s to increase the flexibility in the synthesis procedure. As shown in Table 5, four *DC*

**Table 5** Truth table of a 2 × 2 bit multiplier

| *DC* inputs | | | | Care inputs | | | | Care outputs | | | | *DC* outputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R4 | R3 | R2 | R1 | a | b | c | d | p | q | r | s | G1 | G2 | G3 | G4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | x | x | x | x |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | x | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | x | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | x | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | x | x | x | x |
| 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | x | x | x | x |
| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . |
| 1 | 1 | 1 | 1 | x | x | x | x | x | x | x | x | x | x | x | x |
| *DC* conditions (240 rows) | | | | | | | | | | | | | | | |

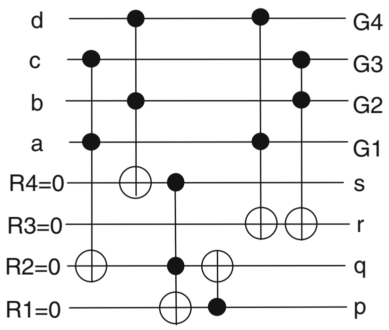

**Fig. 15** Implementation of a 2 × 2 bit multiplier

outputs are considered for this example. The function which is shown in Table 5 has four *DC* inputs ($R_1$ to $R_4$), four *DC* outputs ($G_1$ to $G_4$) and 240 *DC* conditions.

The result of the synthesis is shown in Fig. 15. The circuit has six Toffoli and Feynman gates with a total QC of $5 \times 5 + 1 = 26$.

*Example 6 DC* input value

In the previous examples the optimum value for all *DC* inputs is ‘0’. However, sometimes the value of ‘0’ for *DC* inputs is not optimum. Consider the irreversible NAND gate which has two inputs and one output. The truth table of this gate shows that adding one *DC* input and two *DC* outputs make it reversible. In the proposed synthesis algorithm, *DC* input is inserted in chromosomes. The *DC* outputs are also masked.

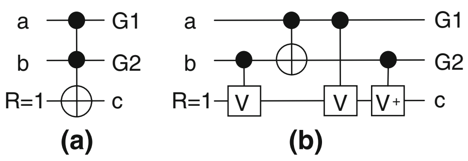

**Fig. 16** Synthesized circuits for NAND2 function (**a**) using Toffoli gates and (**b**) using quantum gates (c is output, G1 and G2 are garbage outputs)

The generated circuit is shown in Fig. 16a. The optimum value for *DC* input (R) is '1'. The optimum quantum implementation of the circuit is also achieved for R = 1. The generated quantum circuit is shown in Fig. 16b. The QCs for two implementations are 5 and 4, respectively.

## 5 Conclusions

In this paper, we introduced a broad concept of *DC*s in reversible logic circuits and quantum circuits. We categorized *DC*s in reversible logic circuits to three types: *DC* inputs, *DC* outputs, and *DC* conditions.

*DC* inputs are additional garbage input bits in the truth table of a logical circuit. They are assumed constant since their values can not be varied in the circuit (Example 5 in Sect. 4). *DC* conditions correspond to the rows of the truth table whose main outputs (not garbage) are *DC*s. *DC* conditions occur when some *DC* inputs for the function are assumed; however, a circuit may have *DC* conditions but not *DC* inputs (Example 2 in Sect. 3). *DC* outputs, called garbage outputs, are additional output bits whose values are not important in any of the truth table rows (Examples 3–5 in Sect. 4).

In this paper, also, some heuristic methods to use these *DC*s, when an optimization algorithm such as the genetic algorithm is used as a synthesis tool, were presented. Table 6 shows the comparison between the designs with and without *DC* concepts, in some examples. Examples 3–5 are basic circuits to design other reversible circuits. This is the first time that these reversible circuits are synthesized efficiently, since the *DC*s are widely used in their design procedure.

**Table 6** Comparison between the AVT and QC of the designs with and without *DC*, in five examples

| Example | Condition | AVT[a] (s) | QC |
|---|---|---|---|
| 1 | Mask="1111" | 10.5 | 17 |
| | Mask="0111" | 2 | 8 |
| | Mask="0011" | 1.3 | 7 |
| | Mask="0001" | 0.12 | 1 |
| 2 | Without DC | 74 | 24 |
| | With DC | 0.1 | 2 |
| 3 | Without DC | 0.63 | 9 |
| | With DC | 0.00185 | 7 |
| 4 | Without DC | 100 | 7 |
| | With DC | 16 | 5 |
| 5 | Without DC | – | – |
| | With DC | 120 | 26 |

[a] AVT is the average synthesis time

## References

1. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Phys. Rev. A **52**(5), 3457–3467 (1995)
2. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**(6), 525–532 (1973)
3. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA (1989). ISBN 0201157675
4. Grobe, D., Dueck, G.W., Chen, X., Drechsler, R.: Exact SAT-based Toffoli network synthesis. In: Proceedings of the 17th ACM Great Lakes Symposium on VLSI (GLSVLSI'07), Stresa-Lago Maggiore, Italy, pp. 96–101 (2007)
5. Gupta, P., Agrawal, A., Jha, N.K.: An algorithm for synthesis of reversible logic circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **25**(11), 2317–2330 (2006)
6. Haghprast, M., Navi, K.: A novel reversible BCD adder for nanotechnology based systems. Am. J. Appl. Sci. **5**(3), 282–288 (2008)
7. Kerntopf, P.: A new heuristic algorithm for reversible logic synthesis. In: Annual ACM IEEE Design Automation Conference. Proceedings of the 41st Annual Conference on Design Automation, San Diego, CA, USA, pp. 834–837 (2004). ISBN 1-58113-828-8
8. Landauer, R.: Irreversibility and heat generation in the computing processes. IBM J. Res. Dev. **5**, 183–191 (1961)
9. Lee, S., Lee, S.J., Kim, T., Lee, J.-S., Biamonte, J., Perkowski, M.: The cost of quantum gate primitives. J. Multi-Valued Logic Soft Comput. **12**(5–6) (2006)
10. Lukac, M., Perkowski, M., Gol, H.: Evolutionary approach to quantum and reversible circuits synthesis. Artif. Intell. Rev. **20**(3–4), 361–417 (2003)
11. Lukac, M., Pivtoraiko, M., Mishchenko, A., Perkowski, M.: Automated synthesis of generalized reversible cascades using genetic algorithms. In: Proceedings of the Fifth International Workshop on Boolean Problems, Freiberg, Sachsen, Germany, 19–20 September, pp. 33–45 (2006)
12. Maslov, D., Dueck, G.W.: Garbage in reversible design of multiple output functions. In: 6th International Symposium on Representations and Methodology of Future Computing Technologies, Trier, Germany, pp. 162–170 (2003)
13. Miller, D.M., Dueck, G.W., Maslov, D.: A transformation based algorithm for reversible logic synthesis. In: Proceedings of the 40th Design Automation Conference, Anaheim, CA, pp. 318–323 (2003)
14. Mohamadi, M., Eshghi, M., Navi, K.: Optimizing the reversible full adder circuit. In: Proceedings of the IEEE East-West on Design and Tests (EWDTS), Yerevan, Armenia, 7–10 September, pp. 312–315 (2007)
15. Shende, V.V., Markov, I.L.: Quantum circuits for incompletely specified two-qubit operators. Quantum Inf. Comput. **5**(1), 49–57 (2005)