An Efficient Functional Verification Method for Quantum Boolean Circuits

Shiou-An Wang, Chin-Yung Lu, and Sy-Yen Kuo

Department of Electrical Engineering and Graduate Institute of Electronic Engineering
National Taiwan University, Taipei, sykuo@cc.ee.ntu.edu.tw, Taiwan

Abstract — **Functional verification is an important design method for verifying functional equivalence between a simplified quantum Boolean circuit and original one. During the design process, checking the equivalence of two quantum Boolean circuits is necessary. In this paper, we present an algorithm that can efficiently and easily verify two quantum Boolean circuits by using the back propagation method. For a set of input vectors, the idea of the algorithm is to find the checking vectors that output vectors are different from input vectors by a backward tracking process.**

Index Terms — **Back propagation, functional verification, quantum Boolean circuit.**

I. Introduction

As the development of a VLSI design grows, the gate size becomes the bottleneck in the design process. Quantum computing gives another solution to conquer this kind of limitation. There are several quantum algorithms, such as Shor's quantum factoring [1] and Grover's fast database search algorithm [2], have been introduced. They are much faster than their best classical counterparts.

To implement a quantum computer, we need to construct quantum Boolean circuits with quantum gates [3]. In general, the classical Boolean circuits are assembled by NOT, OR, and AND gates. However, the quantum Boolean circuits are based on NOT, CN, and CCN gates. The only difference of the functionality between the two kinds of circuits is that the CN gate is substituted for the OR gate.

Because the number of quantum bits which can be used to implement a quantum computer is still small. Therefore, constructing a quantum circuit is always limited by the number of quantum bits. This leads to the equivalence checking [4] of the two given design. A simplified quantum Boolean circuit is functionally equivalent to its original version.

II. Functional Verification

For most quantum Boolean circuits, functional verification involves proving that the circuit is functionally equivalent to other circuits. To prove that two given designs have the same functionality is called *equivalence checking*. This means that an optimized design is functionally equivalent to its earlier design.

After observing the truth table of a circuit, we find part of input vectors keep unchanged, and these vectors can be discarded to reduce the complexity of the verification algorithm. For truth tables under verification, we only consider those that input vectors are different from corresponding output vectors. These vectors are called *changed vectors*, and *checking vectors* are made up of all changed vectors. Those that input vectors are equal to corresponding output vectors are called *unchanged vectors*, and the unchanged vectors don't verify.

Definition 1: Let S be a universal gate set which consists of three kinds of the basic quantum gates, NOT, CN, and CCN gates, and the corresponding functions of these gates are NOT, XOR, and AND functions, respectively.

For the CCN gate of the universal gate set, it can have more than one control qubits and only one target qubit.

Definition 2: Let Q with n qubits be a quantum Boolean circuit which consists of m gates. x_i^j represents the i-th qubit of the j-th quantum gate.

In Fig. 4, the gate {1} has two control qubits, x_2^1 and x_3^1, and a target qubit, x_4^1.

III. Back Propagation Rules

The concept of the back propagation method is to determine a set of checking vectors by a backward tracking process starting from the primary output and tracking toward the primary input.

Definition 3: Let the i-th qubit of a quantum Boolean circuit with n qubits be checked. x_i and y_i represent the input and output value of the i-th qubit, respectively. We call $(x_1, x_2, \ldots x_i, \ldots x_n) = (y_1, y_2, \ldots y_{i-1}, \bar{x}_i, y_{i+1}, \ldots y_n)$ a checking vector of the i-th qubit if the output value of x_i is changed into $\bar{x}_i$ in the case of $x_1=y_1, \ldots, x_{i-1}=y_{i-1}, \ldots, x_n=y_n$.

In this section, several back propagation rules are defined for the functional verification. These rules are shown as follows:

Rule 1: There is no any target bit of the gates in a qubit from the primary output to the primary input. The qubit doesn't need to verify because the value of this qubit keeps unchanged.

According to Rule 1, the qubit x_l in Fig. 3 can be skipped in the operation of the verification algorithm.

Rule 2: There are two CN gates, g and h, with the same target bit x_i and control bit x_j as shown in Fig. 2(a). These two CN gates can be skipped in the backward tracking.

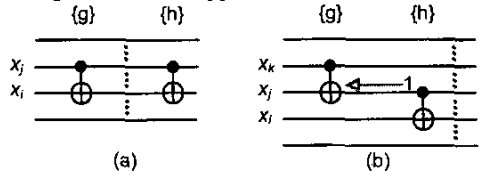

Fig. 1. (a) An example for back propagation rule 2 (b) An example for back propagation rule 3.

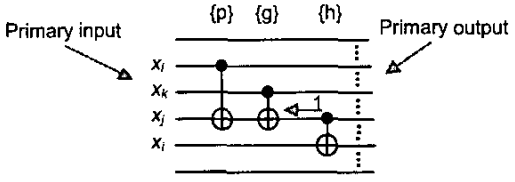

Fig. 2. An example for back propagation rule 4

Rule 3: The qubit x_i is selected to generate the checking vectors as shown in Fig. 2(b). The control bit of the gate h lies on the qubit x_j and there is only one target bit of other gate on the qubit x_j between the gate h and the primary input. The output value of the qubit x_i can be changed into $\bar{x}_i$ according to the following equation:

$$(x_j^h)=1 \xrightarrow{Backward\_tracking} (x_j^g \oplus x_k^g)=1 \xrightarrow{Backward\_tracking} (x_j \oplus x_k)=1 \quad (1)$$

From (1), we can get an equation, $x_j \oplus x_k=1$. If $x_j=1$, $x_k=0$ or $x_j=0$, $x_k=1$, the value of x_i is changed into $\bar{x}_i$. So the checking vectors of the qubit x_i can be defined as

$$(x_k, x_j, x_i)=(0,1,\bar{x}_i),(1,0,\bar{x}_i) \quad (2)$$

Rule 4: The qubit x_i is selected to produce the checking vectors as shown in Fig. 3. The control bit of the gate h lies on the qubit x_j and there are more than one target bit of other gates on the qubit x_j between the gate h and the primary input. The output value of the qubit x_i can be changed into $\bar{x}_i$ according to the following equation:

$$(x_j^h)=1 \xrightarrow{Backward\_tracking} (x_j^g \oplus x_k^g)=1 \quad (3)$$

$$\xrightarrow{Backward\_tracking} ((x_j^p \oplus x_l^p) \oplus x_k^p)=1 \quad (4)$$

$$\xrightarrow{Backward\_tracking} ((x_j \oplus x_l) \oplus x_k)=1 \quad (5)$$

Note that the expressions of the gates p and g are combined by using XOR function in (3) and (4). From (5), we can get a equation, $x_j \oplus x_l \oplus x_k=1$. The checking vectors of the qubit x_i can be defined as

$$(x_l, x_k, x_j, x_i)=(0,0,1,\bar{x}_i),(0,1,0,\bar{x}_i),(1,0,0,\bar{x}_i),(1,1,1,\bar{x}_i) \quad (6)$$

IV. Algorithm

A. Back Propagation

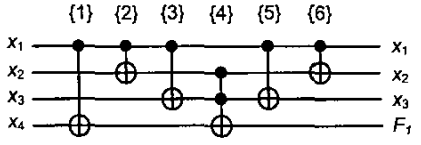

Fig. 3. An example of back propagation

In Fig. 3, we find the only qubit x_4 need to verify and other qubits keep unchanged. The first gate which can affect the value of x_4 is the gate {4}. If $x_2^4=1$ and $x_3^4=1$, the value of x_4 can be changed into $\bar{x}_4$. Then we backwardly track the quantum circuit from the control qubits of the gate {4} toward the primary input. Steps of the back propagation is as follows:

$$(x_2^4)\cdot(x_3^4)=1 \xrightarrow{Backward\_tracking} (x_2^3)\cdot(x_1^3 \oplus x_3^3)=1 \xrightarrow{Backward\_tracking} (x_1^2 \oplus x_2^2)\cdot(x_1^2 \oplus x_3^2)=1 \xrightarrow{Backward\_tracking} (x_1 \oplus x_2)\cdot(x_1 \oplus x_3)=1 \quad (7)$$

From (7), we can get two equations, $x_1 \oplus x_2=1$ and $x_1 \oplus x_3=1$, Note that these two equations are combined by using AND function. If $(x_1,x_2,x_3)=(1,0,0)$ or $(0,1,1)$, then the qubit x_4 can be changed into $\bar{x}_4$. We can get the first checking vectors:

$$\begin{cases}(x_1,x_2,x_3,x_4)=(1,0,0,\bar{x}_4)\\(x_1,x_2,x_3,x_4)=(0,1,1,\bar{x}_4)\end{cases} \quad (8)$$

After tracking the gate {4}, we find the gate {1} can also affect the value of x_4. If $x_1^1=1$, the value of x_4 can be changed into $\bar{x}_4$. The backward tracking is as shown below:

$$(x_1^1)=1 \xrightarrow{backward\_tracking} (x_1)=1 \quad (9)$$

From (9), we can get the equation, $x_1=1$. If $(x_1,x_2,x_3)=(1,x,x)$, where x means $x_i=0$ or 1, then the qubit x_4 can be changed into $\bar{x}_4$. We can get the second checking vector:

$$(x_1,x_2,x_3,x_4)=(1,x,x,\bar{x}_4) \quad (10)$$

Then two checking vectors, (8) and (10), can be combined by XOR function. If any input vectors exist simultaneously in both checking vectors, these input vectors can be deleted because of XOR function. The checking vector $(x_1, x_2, x_3, x_4) = (1,0,0,\bar{x}_4)$ exists in (8) and (10), so it is deleted. Then $(x_1, x_2, x_3, x_4) = (0,1,1,\bar{x}_4)$ and $(1,1,1,\bar{x}_4)$ can be combined into $(x_1, x_2, x_3, x_4) = (x,1,1,\bar{x}_4)$. The final checking vectors are shown as follows:

$$\begin{cases} (x_1, x_2, x_3, x_4) = (x,1,1,\bar{x}_4) \\ (x_1, x_2, x_3, x_4) = (1,0,1,\bar{x}_4) \\ (x_1, x_2, x_3, x_4) = (1,1,0,\bar{x}_4) \end{cases} \quad (11)$$

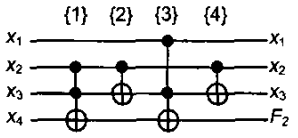

Fig. 4. An example of back propagation

Then we check another circuit to generate its checking vectors. In Fig. 4, we find the only qubit x_4 need to verify and other qubits keep unchanged. The gate {3} can affect the value of x_4. Steps of the back propagation are as follows:

$$(x_3^3)\cdot(x_1^3) = 1 \xrightarrow{Backward\_tracking} (x_2 \oplus x_3)\cdot(x_1) = 1 \quad (12)$$

From (12), the first checking vectors are shown below:

$$\begin{cases} (x_1, x_2, x_3, x_4) = (1,0,1,\bar{x}_4) \\ (x_1, x_2, x_3, x_4) = (1,1,0,\bar{x}_4) \end{cases} \quad (13)$$

The gate {1} can also modify the value of x_4, the back propagation is shown as follows:

$$(x_2^1)\cdot(x_3^1) = 1 \xrightarrow{backward\_tracking} (x_2)\cdot(x_3) = 1 \quad (14)$$

From (14), the second checking vector is shown below:

$$(x_1, x_2, x_3, x_4) = (x,1,1,\bar{x}_4) \quad (15)$$

Then both checking vectors, (13) and (15), are combined by using XOR function. The final checking vectors are shown in (11). Two set of the checking vectors are equal, so two quantum Boolean circuits (Fig. 3 and Fig. 4) have the same functionality.

B. Algorithm

Suppose that two quantum Boolean circuits with n qubits have m_1 and m_2 quantum gates, respectively. The algorithm is described as follows:

Step 1. Select an unmarked quantum Boolean circuit and then mark this circuit.

Step 2. Scan all of the gates and record the position of the target bit of every gate.

Step 3. Select an unmarked qubit which has target bits of quantum gates on it and then mark this qubit.

Step 4. Apply the back propagation for this qubit, then a set of checking vectors can be obtained.

Step 5. If there exist any unmarked qubits, go to Step 3.

Step 6. Combine all of checking vectors from Step 4 to get a final set of checking vectors of the circuit.

Step 7. If there exists an unmarked quantum Boolean circuit, then go to Step 1.

Step 8. Compare the two final sets of checking vectors from Step 6. If their checking vectors are equal, then two circuits are functionally equivalent.

C. Complexity

From the previous subsection, we propose an algorithm that can efficiently verify the equivalence of two quantum Boolean circuits. Assume that two circuits with n qubits have m_1 and m_2 quantum gates, respectively. Using our algorithm to verify the functional equivalence of two circuits, we need to scan the circuit to find the position of the target bit for every gate. If m is equal to $(m_1+m_2)/2$, the complexity of the algorithm is $O(nm)$. Therefore, our algorithm can apply the back propagation method to prove the functional equivalence in the polynomial time.

V. CONCLUSION

In this paper, we focus on a general approach using the back propagation method to find all changed vectors for a quantum Boolean circuit. If two quantum Boolean circuits have the same changed vectors, then these two circuits are functionally equivalent to each other. If two circuits with n qubits have m_1 and m_2 quantum gates, the complexity of the algorithm is $O(nm)$.

REFERENCES

[1] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring" *Proc. of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, pp. 124-134, 1994.

[2] L. Grover, "A fast quantum mechanical algorithm for database search" *Proc. of the 28th Annual ACM symposium on the Theory of Computing*, pp. 212-219, 1996.

[3] I-Ming Tsai and Sy-Yen Kuo, "Quantum Boolean Circuit Construction and Layout under Locality Constraint" *Proc. of the 1st IEEE Conference on Nanotechnology*, pp. 111-116, 2001.

[4] Shi-Yu Huang and Kwang-Ting(Tim) Cheng, *Formal equivalence checking and design debugging*, Boston: Kluwer Academic Publishers, 1998.