Design and Implementation of the Deutsch-Jozsa Algorithm Based Quantum Multiplier with a Reduced T-count


Department of Electronics and Communication Engineering, Vardhaman College of Engineering, Hyderabad, Telangana, India

Abstract

To create a more effective and efficient multiplier based on an algorithm called the Deutsch-Jozsa algorithm. Quantum circuits for fundamental operations like multiplication are needed. In this paper, we suggested a design for quantum-efficient integer multiplication. The models, based on the quantum Clifford+T gates, are T-depth and T-count optimized. Gates may be used to create quantum circuits that are leakage, however, the T-gate is exceedingly expensive to put into practice. Therefore, lowering both T-count and T-depth has emerged as crucial optimization targets. The quantity of qubits that can be used in existing quantum hardware is constrained. A unique quantum conditional adder circuit is used in quantum inspire to implement the suggested quantum multiplier architecture, which lowers the T-count. The conditional adder is also changed to a Toffoli gate when one of its operands is zero.

Keywords

Toffoli gate, Control adder, Array multiplier, Quantum multiplier, Deutsch-Jozsa algorithm, Grover’s algorithm, Nanotechnology

Introduction

Quantum computing and nanotechnology appears to be one of the most promising new computer paradigms due to its applications in research processing, discovery, encryption, and numerical methods [1-4]. Quantum computing can provide computational power and algorithms to simulate and optimize nanoscale phenomena and systems, such as molecular dynamics, nanoelectronics, and nanomaterials. Together, they can enable new functionalities and applications that are impossible or impractical with conventional technologies. In several of these domains' quantum computations, integer mathematical operations like add, reduction, and multiplication are performed using quantum circuits. For usage with quantum computer languages like Quipper [5] plus LIQI [6] and cQASM, python, Qiskit, and Git, scientists have developed specific packages of core quantum numerical arithmetic methods, as well as tools for designing quantum computers like those suggested in [6] and [7]. Quantum gates are devices that can be used to do quantum computation. There is a 1-to-1 mapping between the output and input vectors of every quantum circuit. Ancillae have been any stable inputs in a quantum circuit. Every output that would be necessary for the quantum circuit to maintain 1-to-1 mapping but does not constitute a significant input nor a desirable output is referred to as garbage output. The results that resulted from the inputs are not thought of as wasteful results [8]. Circuit overhead should be maintained to a minimum, including ancillae and garbage outputs. Quantum nanoscience is the basic research area at the intersection of nanoscale science and quantum science that creates the understanding that enables development of nanotechnologies. It uses quantum mechanics to explore and use coherent quantum effects in engineered nanostructures.
Researchers are becoming more interested in the construction of quantum circuits with fault tolerance as a result of the noise faults that can occur in physical quantum computers [8, 9]. To get beyond the restrictions imposed, it is possible to use failure quantum error-correcting codes and quantum gates.

The quantum T-gate’s increased noise tolerance, however, is accompanied by an increase in implementation costs [10–12]. Due to the greater cost to implement the T-count, T-gate has evolved as a crucial performance parameter for the error-resistant design of quantum circuitry.

In the papers, the designing of quantum numeric multiplication circuits has received a lot of interest. Garbage-free architectures, similar to those in [13] are built on several examples of highly T-counted circuits for quantum condition adders. The recurrent use of the conditional adder circuit in these designs will increase the multiplication circuit’s T-gate cost. Other publications, like the one in [14], show T-gate efficiency architectures but do not factor in the extra ancillae plus T-gate expenses associated with removing trash outputs from the cost estimates.

Although the circuits for integer multiplication reported in earlier publications like [7] and [13] are attractive concepts, their fault-tolerant implementations are unworkable due to a high T-count expense.

This study includes proposing the design of a garbage-free multiplier with 4.n+1 Qubits, where n = number of bits for multiplying. The suggested multiplier can be designed using the Toffoli gate which contains 4-T gates and a control adder that has no input carry and is garbage-less T-count optimization is the foundation of a quantized numeric multiplication circuit and the design is implemented by an algorithm using cQASM programming language in Quantum Inspire Tool. Comparing the proposed design with other ones, it can be shown that it performs better in terms of T-count and algorithm level.

**Performance evaluation of quantum circuits**

Since the T-gate’s error-tolerant implementation costs significantly more than those of the various Clifford+T gates’ failure implementations, it is interesting to assess quantum network effectiveness in terms of T-count [10–12]. The overall number of T-gates or symmetric transposes of a T-gate within a quantum circuit is known as the T-count. The count of the T-gate for the Clifford+T design is 4.

Researchers are looking at techniques for integer multiplication. Numerous multiplication methods, including Booth’s method, shifting and add, as well as Karatsuba’s algorithm, have been devised by researchers. In this research, we are showing a quantum implementation of the T-count-optimized shifting and introduce the quantum inspire multiplication approach.

Let’s consider multiplying 2n-bit values, X and Y. The shifting and adding multiplication method gives the product R of multiplying the two integers X and Y at the conclusion of processing. Regarding the X-digit number multiplied by the number Y, the stages of the shifting and add algorithm for multiplying are shown.

**Procedure**

```
R = 0
N = number of bits
for P in range(0, N):
    R = R + (X^Y)’(2^P)
print(R)
```

**Literature review**

Inside the literature, the architecture of quantum circuits for multiplying integers has drawn a lot of interest. The majority of efforts, however, focus on versatile computing and have high production costs for garbage. Clifford+T gate prices in works like [10] and [15] that provide in the study of the quantum Fourier series, multiplication circuits are unreasonably expensive.

For example, the T-gate value of the quantum multiplier design in [13] is on the order O(n3). The characteristics of the quantum multipliers in [12] and [13] are shown in table 1. Quantum multiplier circuits are shown in designs like those in [6, 9, 10] that can be realized with far fewer quantum gates. The characteristics of the quantum multipliers in [9] and are shown in table 1. The Square Root of Not, Pauli gates, and Toffoli gates may be attained using this technique.

These known quantum arithmetic multiplication circuits employ the Fredkin, Toffoli. The current quantum conditional addition circuits provide the foundation for quantum integer multiplication circuits described in works of [12] and [15]. A conditional addition circuit based on the architecture described in [19] is applied to create the multiplication circuit presented in [8]. The quantum multiplier circuit architecture is described in [20] and consists of a unique quantum conditional adder circuit.

Based on a two-step method, the multiplication circuits in [10] and [15] multiply the numbers: Create all partial products, and then use adders to merge them all. A novel quantum plus circuit and a complete quantum adder circuit makes up the architecture shown in [8]. CNOT gate and the square of not gates are used to build the quantum AND circuit and quantum full adder. To reduce the depth of a circuit, the intermediate products are produced simultaneously. Toffoli gates and other devices are used in the design shown in [13]. To decrease circuit depth, the intermediate products are built concurrently with a series of Toffoli gates. Examples of quantum ripple carryover adders are shown in [15] and are employed to complete the product addition partially depth of the intermediate product addition is lowered using a design strategy formed on a bit’s addition tree, which is similar to the idea displayed in [13]. Despite being depth-optimized, the systems have large ancillae and garbage output expenses. The multiplication circuits which are only for integers described in articles [11] and [13] have interesting designs, but they have a high T-count cost (Table 1).

**Theory about Deutsch-Jozsa algorithm**

In the Deutsch-Jozsa method, we use an oracle code to determine whether a binary function is correct. A basic quan-
A polynomial or even exponential speedup over conventional algorithms may be achieved by other quantum algorithms that use the same quantum mechanical phenomena, despite the fact that the speedup of this particular technique is just a factor of two. Figure 1 presents circuit diagram of Deutsch-Jozsa algorithm.

**Experimentation**

**Design of a proposed method**

P = A * B’s multiplication matrix. In this diagram, the 2n-bit product is created by multiplying the n-bit multiplicand, A, by the n-bit multiplier, B [6, 9].

A = A0 to An-1  
B = B0 to Bn-1  
P = P0 to P2n-1

In figure 2, it’s clearly, we deduced a generic formula to calculate the value of its products when A and B’s input values are known. We next examined other examples of the process of multiplying different values of n. that are shown in table 2.

**Algorithm for a Toffoli gate**

To write oracle code for Toffoli gate (Figure 3) by using a Deutsch-Jozsa algorithm.

- VERSION 1.0  
- #INITIALIZE THE NUMBER OF QUBITS REQUIRED

**Table 1:** Overview of the quantum multipliers currently in use.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Add and rotate</td>
<td>Shift and add</td>
<td>2 step algorithm</td>
<td>2 step algorithm</td>
</tr>
<tr>
<td>Modules</td>
<td>Conditional adder, swap gates</td>
<td>Conditional adder, swap gates</td>
<td>Partial product circuit, adder tree</td>
<td>Partial product circuit, adder tree</td>
</tr>
<tr>
<td>Quantum gates</td>
<td>CNOT, Toffolo gate, Fredkin gate</td>
<td>CNOT, Toffolo gate</td>
<td>CNOT, Toffolo gate, Peres gate</td>
<td>CNOT, square root of not gate</td>
</tr>
<tr>
<td>Qubit gates</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>NA</td>
</tr>
<tr>
<td>T-count</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
</tbody>
</table>

**Table 2:** Procedure for multiplication (n = 6).

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
<td>B5</td>
<td>B6</td>
</tr>
<tr>
<td>B7</td>
<td>B8</td>
<td>B9</td>
<td>B10</td>
<td>B11</td>
<td>B12</td>
</tr>
<tr>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
<td>P5</td>
<td>P6</td>
</tr>
<tr>
<td>P7</td>
<td>P8</td>
<td>P9</td>
<td>P10</td>
<td>P11</td>
<td>P12</td>
</tr>
<tr>
<td>P13</td>
<td>P14</td>
<td>P15</td>
<td>P16</td>
<td>P17</td>
<td>P18</td>
</tr>
<tr>
<td>P19</td>
<td>P20</td>
<td>P21</td>
<td>P22</td>
<td>P23</td>
<td>P24</td>
</tr>
<tr>
<td>P25</td>
<td>P26</td>
<td>P27</td>
<td>P28</td>
<td>P29</td>
<td>P30</td>
</tr>
<tr>
<td>P31</td>
<td>P32</td>
<td>P33</td>
<td>P34</td>
<td>P35</td>
<td>P36</td>
</tr>
<tr>
<td>P37</td>
<td>P38</td>
<td>P39</td>
<td>P40</td>
<td>P41</td>
<td>P42</td>
</tr>
<tr>
<td>P43</td>
<td>P44</td>
<td>P45</td>
<td>P46</td>
<td>P47</td>
<td>P48</td>
</tr>
<tr>
<td>P49</td>
<td>P50</td>
<td>P51</td>
<td>P52</td>
<td>P53</td>
<td>P54</td>
</tr>
<tr>
<td>P55</td>
<td>P56</td>
<td>P57</td>
<td>P58</td>
<td>P59</td>
<td>P60</td>
</tr>
<tr>
<td>P61</td>
<td>P62</td>
<td>P63</td>
<td>P64</td>
<td>P65</td>
<td>P66</td>
</tr>
<tr>
<td>P67</td>
<td>P68</td>
<td>P69</td>
<td>P70</td>
<td>P71</td>
<td>P72</td>
</tr>
<tr>
<td>P73</td>
<td>P74</td>
<td>P75</td>
<td>P76</td>
<td>P77</td>
<td>P78</td>
</tr>
<tr>
<td>P79</td>
<td>P80</td>
<td>P81</td>
<td>P82</td>
<td>P83</td>
<td>P84</td>
</tr>
<tr>
<td>P85</td>
<td>P86</td>
<td>P87</td>
<td>P88</td>
<td>P89</td>
<td>P90</td>
</tr>
<tr>
<td>P91</td>
<td>P92</td>
<td>P93</td>
<td>P94</td>
<td>P95</td>
<td>P96</td>
</tr>
<tr>
<td>P97</td>
<td>P98</td>
<td>P99</td>
<td>P100</td>
<td>P101</td>
<td>P102</td>
</tr>
<tr>
<td>P103</td>
<td>P104</td>
<td>P105</td>
<td>P106</td>
<td>P107</td>
<td>P108</td>
</tr>
</tbody>
</table>

**Figure 1:** Circuit diagram of Deutsch-Jozsa algorithm.

**Figure 2:** Block diagram of a multiplier.

**Figure 3:** Implementation of Toffoli gate with Clifford+T gates.
• QUBITS 3
• #SET ALL QUBITS TO 0
• PREP_Z Q[0:2]
• [X Q[0] | X Q[1]]
• [H Q[2]]
• ADD
• CNOT Q[2], Q[1]
• CNOT Q[1], Q[0]
• T Q[0]
• TDAG Q[1]
• T Q[2]
• CNOT Q[2], Q[1]
• CNOT Q[1], Q[0]
• {TDAG Q[0]}
• CNOT Q[2], Q[0]
• H Q[2]
• .MEASUREMENT
• MEASURE Q[0,1,2]

Circuit design for quantum conditional addition without input carry

Considering the outline, the layout of the suggested input-free quantum conditional adder circuit (Figure 4). The design doesn’t produce any trash. Comparing the suggested design to existing design strategies that don’t produce garbage, the suggested design has a less T-count. The paper [21] provided a detailed explanation of the design stages.

Algorithm for a control adder

• VERSIONi 1.0
• QUBITS 15
• PREP_Z Q[0:14]
• .INIT
• .ADD
• CNOT1 Q[4], Q[3]
• CNOT2 Q[6], Q[5]
• CNOT3 Q[8], Q[7]
• CNOT4 Q[10], Q[9]
• CNOT5 Q[12], Q[11]
• TOFFOLI1 Q[0], Q[12], Q[13]
• CNOT6 Q[10], Q[12]
• CNOT7 Q[8], Q[10]
• CNOT8 Q[6], Q[8]
• CNOT9 Q[4], Q[6]
• TOFFOLI2 Q[1], Q[2], Q[4]
• TOFFOLI3 Q[3], Q[4], Q[6]
• TOFFOLI4 Q[5], Q[6], Q[8]
• TOFFOLI5 Q[7], Q[8], Q[9]
• TOFFOLI6 Q[10], Q[11], Q[13]
• TOFFOLI7 Q[11], Q[12], Q[14]
• TOFFOLI8 Q[0], Q[14], Q[13]
• TOFFOLI9 Q[11], Q[12], Q[14]
• TOFFOLI10 Q[0], Q[12], Q[11]
• TOFFOLI11 Q[9], Q[10], Q[12]
• TOFFOLI12 Q[0], Q[10], Q[9]
• TOFFOLI13 Q[7], Q[8], Q[10]
• TOFFOLI14 Q[0], Q[8], Q[7]
• TOFFOLI15 Q[5], Q[6], Q[7]
• TOFFOLI16 Q[0], Q[6], Q[5]
• TOFFOLI17 Q[3], Q[4], Q[6]
• TOFFOLI18 Q[0], Q[4], Q[3]
• TOFFOLI19 Q[1], Q[2], Q[4]
• TOFFOLI20 Q[0], Q[2], Q[1]
• CNOT10 Q[4], Q[6]
• CNOT11 Q[6], Q[8]
• CNOT12 Q[8], Q[10]
• CNOT13 Q[10], Q[12]
• CNOT14 Q[4], Q[3]
• CNOT15 Q[6], Q[5]
• CNOT16 Q[8], Q[7]
• CNOT17 Q[10], Q[9]
• CNOT18 Q[12], Q[11]
• .MEASUREMENT
• MEASURE Q[1,3,5,7,9,11,13]

Note: Here we had given the numbering for CNOT and Toffoli gates for readability purposes.

T-count

The suggested quantum Ctrl_Add circuit for an n-bit size has the following total T-count: 12.n8.

Ancillae cost

For each phase of the suggested design process, a brief
illustration of the necessary ancillae is offered for the proposed quantum control adder circuit. By adding the ancillae needed for each stage of the suggested design technique, we determine the total expenses associated with the projected control adder circuit. An n-bit suggested quantum Ctrl_Add circuit has a total of two ancillae (Table 3).

**Design of a suggested quantum arithmetic multiplication circuit**

Compared to the design methodologies currently used, which do not contain garbage outputs, the suggested quantum multiplication circuit for integers is constructed lacking garbage outputs plus a reduced T-count. The suggested quantum control adder. The quantum arithmetic multiplication circuit is according to the Toffoli gate array and the circuit and without any input carry, the total circuit operation explained in [21].

- **Step - 1:** In this step, we will get each Toffoli gate output which is input to the control adder as shown in step 2.
- **Step - 2:** In this step, the outputs of Toffoli gates are given to the control adder circuit. The operation is clearly explained in [21].
- **Step - 3:** Similarly, by keeping side-by-side control adders we will get the output of the multiplier.

The following steps are shown in the form of a circuit (Figure 5). The code will be split into five blocks and initialized using the Toffoli gate and control adder technique for this multiplier circuit. The multiplier code operates similarly and is based on the following process initialized.

**Algorithm for a multiplier circuit**

- Set up the necessary qubits.
- Next, use Toffoli gates in accordance with the logic.
- Use the control adder algorithm next.
- Carry out this step five more times.
- Evaluate the output of the multiplier.

**Cost analysis**

**T-gate cost**

For each phase of the suggested design process, a brief illustration of the T-count of the suggested quantum arithmetic multiplication circuit is provided (Table 4). By adding the T-counts for every stage of the suggested design process, we determine the overall account for the suggested quantum arithmetic multiplication circuit (Table 5). The suggested quantum arithmetic multiplication circuit’s overall T-count is as follows.

\[ 12n^2 - 24n + 8 \]

**Ancillae cost**

The suggested quantum arithmetic multiplication circuit’s necessary ancillae are briefly presented for each stage of the suggested design process. We add the ancillae needed for each phase of the suggested design process to determine the sum of ancillae for the suggested quantum arithmetic multiplication circuit. The suggested quantum arithmetic multiplication circuit’s whole ancillae are listed as follows.

\[ 2n + 1 \]

**Results and Discussion**

Here, presenting a single result circuit diagram, in which we have included numerous metrics concerning the efficiency of the quantum circuit following the use of the provided approach. In this article, we offer one design approach for a high-speed multiplier that is fault tolerant. The results were verified in quantum inspire, Cadence tool, and Verilog simulations.

**Toffoli gate output**

The inputs are \( Q[0]=1, Q[1]=0, Q[2]=1 \) (Figure 6).

Then Toffoli gate output is \( Q[0]=1, Q[2]=0, Q[2]=0 \) (Figure 7).

Here we tested all eight combinations. The outputs were verified.

---

Table 3: Comparison of quantum ctrl_add circuit.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_Count</td>
<td>56N</td>
<td>28N+7</td>
<td>21N+7</td>
<td>28N+7</td>
<td>12N+8</td>
</tr>
<tr>
<td>Qubits</td>
<td>4N+2</td>
<td>2N+3</td>
<td>2N+3</td>
<td>2N+3</td>
<td>2N+3</td>
</tr>
<tr>
<td>Ancillae</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Note: Lin et al. creation is number 1 [11]. Jayashree et al. design number 2 [16], Thapliyal et al. design number 3 [21], Markov and Shi design is number 4 [22].

Table 4: Circuits for quantum arithmetic multiplication are compared.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_Count</td>
<td>56N^2</td>
<td>28N^2+7N</td>
<td>42N^2</td>
<td>21N^2-14</td>
<td>12N^2-24N+8</td>
</tr>
<tr>
<td>Qubits</td>
<td>5N+1</td>
<td>4N+1</td>
<td>NA</td>
<td>4N+1</td>
<td>4N+1</td>
</tr>
<tr>
<td>Ancillae</td>
<td>3N+1</td>
<td>2N+1</td>
<td>NA</td>
<td>2N+1</td>
<td>2N+1</td>
</tr>
</tbody>
</table>

Note: The design by Lin et al. is number 1 [11], The design by Jayashree et al. is number 2 [16], 3 is a modified version of Babu’s [18] design that doesn’t produce waste, 4 is a modified version of Thapliyal et al. [21] design that doesn’t produce garbage.
Control adder

Let’s have a look at a 6 x 6 multiplication matrix (n = 6) where the input comes from two different references A and B.

We should give an illustration of how the outcomes of the generalized formula are obtained. The expression that would be consistent with results produced by computers or other quantitative approaches like \[7\] would then be acquired after crossing the carry, we would proceed as before.

- Keep the ctrl bit set to one.
- As stated, A and B have the following values:
  - \(A = 0101010 = 10\)
  - \(B = 0101001 = 20\)
  - The output given by the control adder is 30.
  - i.e., \(0 \times 0 \times 1 \times 1 \times 1 \times 0\)

After writing the code by using Deutsch-Jozsa algorithm, quantum inspire generated the below circuit and the outputs are verified (Figure 8 and figure 9).

Multiplier

Let’s have a look at a 6 x 6 multiplication matrix (n = 6) where the input comes from two different sources, A and B.

We would provide an example to demonstrate how the generalized formula was obtained and how to calculate the product values. After that, we would proceed with passing the carry until we had the product term that matched the numbers produced by arithmetic or software algorithms like \[8\]. As stated, A and B have the following values:

\[
A = 010001 = 17
\]

\[
B = 011100 = 28
\]
P = 00011101110

The output given by the multiplier is 476, which will be given in the form of hexadecimal i.e., 0x1DC. The circuit was given in figure 10 and figure 11.

The outputs were verified in Verilog simulations also for writing a Verilog code that will be dumped in the cadence tool and power, delay, and area calculations were calculated. The performance of the 6 x 6 multiplier for signed bits was determined in this work and implemented. With Verilog serving as the hardware description language, we synthesized these multipliers utilizing cadence software. The analysis is done on the radix6 schematic circuits and waveforms [6]. The transistor-level circuit is created by cadence software in the accurate multiplier of radix-6, which has 12 input bits and 12 output bits (Figure 12).

The window for the 6-bit quantum multiplier. The cadence software generated the power (Figure 13), delay (Figure 14), and area (Figure 15) reports.

Here we are not comparing with any other multipliers because there are no previous multipliers that are done in quantum IBM or quantum inspire.

We also noted that the kind of multiplier being employed will have an impact on multiplier performance. Although rapid adders will operate well, they may be used to speed up the adding process by positioning them close to the center, while slower adders can be employed close to the regions with the least and greatest significance. As a result, the speed will be preserved, and less hardware and power will be consumed. So here we are using an efficient control adder which already existing adders implemented in Deutsch-Jozsa algorithm. When compared to classical algorithms, quantum algorithms were developed to solve classical problems with fewer steps the Shor algorithm, the Grover algorithm, and others are a few of the highly well-known algorithms.

Grover’s method is essentially a search technique that is used to discover the input in a collection of functions in order to produce a specific output. It has several uses while looking over a big database.

The Deutsch-Jozsa algorithm is an exponentially quicker quantum deterministic algorithm than the conventional algorithm (Figure 16). There is a drastic change in speed and performance (Figure 17).
Acknowledgements
None.

Conflict of Interest
None.

References


Conclusion
In this paper, we offer two different forms of enhancements for integer quantum circuit multiplication, one that utilizes Clifford+T gates and the other on a Deutsch-Jozsa algorithm, which both increase the multiplier’s speed and efficiency. It also illustrates the architecture of the quantum conditional add-on circuit functionality and the quantum adder, two subcomponents utilized in the suggested quantum integer multiplier circuits. In count of T-depth and T-count, the suggested quantum multiplier for integers circuits is demonstrated to outperform already existing models. It is also a quick multiplier. We get to the conclusion that the suggested multiplier circuit may be incorporated into a wider quantum data route system design, where creating a Deutsch-Jozsa algorithm and focusing on T-depth and T-count are very important parameters.


