IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

DOI:10.1587/transfun.2023EAP1165

Publicized:2024/07/16

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Reducing T-Count in Quantum Circuits Using Alternate Forms of the Relative Phase Toffoli Gate

David CLARINO[†a)], **Shohei KURODA**[†b)], *Nonmembers*, *and* **Shigeru YAMASHITA**[†c)], *Senior Member*

SUMMARY Toffoli gates are an important primitive in reversible Boolean logic. In quantum computation, these Toffoli gates are composed using other elementary gates, most notably the Clifford+T basis. However, in fault-tolerant implementations of quantum circuits, the T-gate incurs extra cost relative to Clifford gates like the S-gate and CNOT gate. Relative-phase Toffoli Gates (RTOFs) have been proposed as a way to reduce this T-count at the cost of incurring a relative phase that could skew the final quantum states. In this paper, we utilize an observation that the relative phase which RTOFs introduce can be canceled by the appropriate application of less expensive S-gates instead of T-gates. It leverages alternate forms of the RTOF including incorporating S-gates into it or moving around its input bits in order to simplify the logic to erase the relative phase. We find experimentally that our method has a clear advantage in most cases, and identify several types of circuits that it could be synergistic with.

key words: *relative-phase Toffoli gates (RTOF), S gate, circuit optimization*

1. Introduction

Boolean functions are often used as essential components in quantum algorithms. These are calculated using the circuit model called *quantum Boolean circuits* [1], which are quantum circuits that realize Boolean functions. To realize such quantum Boolean circuits, a Toffoli gate is often used as a logic primitive. However, as three-qubit operations do not occur naturally in the two-level systems that enable modern quantum computers, these Toffoli gates have to be in turn, composed of physically realizable gates, such as the S-gate, CNOT gate, and T-gate. (e.g., the Clifford + T [2] basis gate set).

As qubits are famously prone to error and perturbation, it is often useful to consider a *fault-tolerant* implementation of quantum computation, one which could detect and correct errors [3]. Unfortunately, in the fault-tolerant paradigm, T-gates and their inverse the $T^\dagger$-gates incur much higher cost than other gate types. Therefore, decreasing the number of T-gates is a major design consideration when implementing physically realizable quantum computing. The number of these T-gates, in addition to the number of their inverse the $T^\dagger$-gates, is called the *T-count*.

Relative-phase Toffoli gates (RTOF) [4] have been proposed as an alternative implementation of a Toffoli gate in the decomposition of a Toffoli gate. An RTOF can calculate the same logic function as a Toffoli gate but with a lower T-count by three. However, it introduces a *relative phase* among its output quantum states, which means that the phase introduced is dependent on the component of the quantum state on which it acts. Because of the relative phase, we cannot simply replace a Toffoli gate with an RTOF in general. In [4], we see an application of this principle in the decomposition of multiple control Toffoli gates using pairs of RTOFs that are inverses of each other.

In this work[†], we observe that because RTOFs introduce phases of $\{\frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$, their phases can be erased by applying S-gates and $S^\dagger$-gates appropriately. These can drive phases without increasing the T-count. We propose a post-synthesis step that replaces all Toffoli gates in a quantum Boolean circuit with RTOFs, and then try to erase the added relative phases by adding logic to drive the appropriate S-gates and $S^\dagger$-gates. We identify several cases where Toffoli gates can be replaced by RTOFs without increasing T-count in this manner. We list the main contributions of the paper below.

Our Contribution.

- A post-synthesis framework to replace Toffoli gates with RTOF gates in quantum Boolean circuits and generate logic to correct their phase
- Introduction of a variant of the RTOF, which introduces only a relative phase of π on the $|101\rangle$ state. This simplifies the procedures to correct the phase.
- Introduction of a heuristic method of assigning inputs to the RTOF in order to simplify the phase function

The paper is structured as follows. Section 2 provides the necessary information to understand this paper. Then, Section 3 explains our idea on how to utilize RTOFs to reduce the T-count of quantum Boolean circuits [5] and subsequently correct the phase. Section 4 proposes our enhancements, including a new variant of the RTOF as well as an algorithm to reduce the T-count of quantum Boolean circuits by simplifying the phase correction logic. We explain our experimental methodology and detail our experimental results in Section 5. Finally, Section 6 concludes the paper with our future work.

2. Preliminary

2.1 Quantum Gates and Quantum Circuits

Quantum computers internally represent data as *qubits*,

[†]The author is with the Graduate School of Information Science and Engineering, Ritsumeikan University, Osaka, 567-8570 Japan.
a) E-mail: dizzy@ngc.is.ritsumei.ac.jp
b) E-mail: sousai@ngc.is.ritsumei.ac.jp
c) E-mail: ger@cs.ritsumei.ac.jp

[†]Previous versions of this paper were presented at ISMVL 2023 and RC 2022

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

$|x\rangle$ — $|x\rangle$
$|y\rangle$ — $|x \oplus y\rangle$

(a) CNOT-gate

$$\begin{bmatrix}1&0\\0&e^{i\frac{\pi}{4}}\end{bmatrix}$$

$|x\rangle$ — T — $e^{i\frac{\pi}{4}x}|x\rangle$

(b) T-gate

$$\begin{bmatrix}1&0\\0&e^{i\frac{\pi}{2}}\end{bmatrix}$$

$|x\rangle$ — S — $e^{i\frac{\pi}{2}x}|x\rangle$

(c) S-gate

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix}$$

$|x\rangle$ — H — $\frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle)$

(d) H-gate

$$\begin{bmatrix}1&0\\0&e^{-i\frac{\pi}{4}}\end{bmatrix}$$

$|x\rangle$ — $T^\dagger$ — $e^{-i\frac{\pi}{4}x}|x\rangle$

(e) $T^\dagger$-gate

$$\begin{bmatrix}1&0\\0&e^{-i\frac{\pi}{2}}\end{bmatrix}$$

$|x\rangle$ — $S^\dagger$ — $e^{-i\frac{\pi}{2}x}|x\rangle$

(f) $S^\dagger$-gate

Fig. 1: The H, CNOT, T , S (Clifford+T) basis gate set

which are quantum systems that can take on the states $|0\rangle$ and $|1\rangle$. We can combine these qubits into bit strings to create multi-qubit states. In particular, there exists a specific set of these called *computational basis states* which are of the form $|x_0\rangle \otimes |x_1\rangle \otimes \cdots \otimes |x_{n-1}\rangle = |\mathbf{x}\rangle$ where $x_0x_1\cdots x_{n-1} = \mathbf{x}$, for $\mathbf{x} \in \{0,1\}^n$, where n is the number of qubits in the system, and $\otimes$ is tensor multiplication between quantum states [3]. These computational basis states can in turn be used in linear combination to express a generalized quantum state $|\psi\rangle = \sum_{k\in[0,1]^n} e^{i\theta(\mathbf{k})}|\mathbf{k}\rangle, \mathbf{k} \in \{0,1\}^n$, where $\theta(\mathbf{k}) \in [-\pi, \pi]$ is an arbitrary phase that is a function of k.

Quantum gates map quantum states $|\psi\rangle = \sum_{k\in[0,1]^n} e^{i\theta(k)}|k\rangle$ to other quantum states $|\phi\rangle = \sum_{j\in[0,1]^n} e^{i\theta(j)}|j\rangle$.

In this paper, we consider the elementary gate set consisting of the H-gate, $CNOT$, T-gate, $T^\dagger$-gate (the inverse of the T-gate), S-gate, and $S^\dagger$-gate (the inverse of the S-gate), known as the *Clifford+T basis gate set*. We describe their behavior in Fig. 1. In the fault-tolerant paradigm, T-gates and $T^\dagger$-gates are much more expensive to implement than H-gate, $CNOT$, S-gate, and $S^\dagger$-gates. This means that the number of T-gates is key to lowering the cost of the implementation. Hereafter, we will refer to the set of the T-gate and the $T^\dagger$-gate as "T-gates", and the set of the S-gate and the $S^\dagger$-gate "S-gates", unless otherwise noted.

When this set of gates is assembled into a network, we call the result a *quantum circuit*. In a quantum circuit, inputs come in from the left, gates are applied in order from left to right, and outputs go out from the right. Quantum circuits also have *output qubits*, which are the qubits that output quantum states to be measured or used by other quantum circuits.

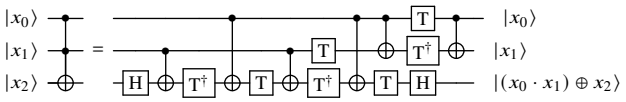
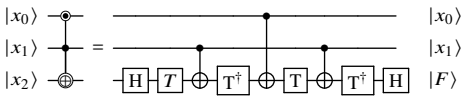


Fig. 2: A Toffoli gate in Clifford+T basis gate set



$$|F\rangle = e^{i\theta(\mathbf{x})}|(x_0 \cdot x_1) \oplus x_2\rangle$$
$$\theta(\mathbf{x}) = \tfrac{\pi}{2}(x_0x_1\overline{x_2}) + \pi(x_0\overline{x_1}x_2) + \tfrac{3\pi}{2}(x_0x_1x_2)$$

Fig. 3: An $RTOF$ gate in Clifford+T basis gate set

$\mathbf{x}$	$\theta(\mathbf{x})$
000	0
001	0
010	0
011	0
100	0
101	$+\pi$
110	$+\pi/2$
111	$-\pi/2$

Table 1: Added phases by an $RTOF$.

2.2 Quantum Boolean Circuits

In this paper, we consider *quantum Boolean circuits* [1]. These are a special class of quantum circuits that map computational basis states to other computational basis states. These are equivalent to classical Boolean functions since manipulating the computational basis states is equivalent to manipulating bit-strings.

Recall that to implement arbitrary Boolean functions, we need both a NOT and either an OR or an AND operation. The sequence of H-gate, S-gate, S-gate, H-gate, gives us the mapping $|0\rangle \rightarrow |1\rangle, |1\rangle \rightarrow |0\rangle$. This thus gives us a way to realize the NOT operation. The Toffoli gate, depicted in Fig. 2, provides a way to realize the AND operation by implementing the mapping $|x_0\rangle|x_1\rangle|x_2\rangle \rightarrow |x_0\rangle|x_1\rangle|(x_0 \cdot x_1) \oplus x_2\rangle$. If we set the third qubit to $x_2 = 0$, then the third qubit realizes $x_0 \cdot x_1$, which is the AND operation. As we can see from Fig. 2, Toffoli gates can also be implemented using the Clifford+T gate set. Given these sets of gates, we can therefore realize Boolean operations using the quantum gate set described in Sec. 2.1. We will call Boolean operations that use the Toffoli gate depicted in Fig. 2 the *All-Toffoli case*.

Finally, when quantum Boolean circuits use qubits to store values temporarily during the computation, these values can provide undesirable effects on the rest of the computation due to quantum entanglement. There is thus a need to restore these qubits to their original values in order to use the output qubits as input to other quantum circuit elements. We call the action of restoring these non-output qubits to their original states *uncomputation*.

$mathbf{x}$	$F_{\pi/2}$	F_{π}	$F_{3\pi/2}$
000	0	0	0
001	0	0	0
010	0	0	0
011	0	0	0
100	0	0	0
101	0	1	0
110	1	0	0
111	0	0	1

Table 2: An example of phase functions.

2.3 Relative-Phase Toffoli Gates ($RTOFs$)

The Relative-Phase Toffoli gate ($RTOF$) [4] is a relative phase version of the Toffoli gate, depicted in Fig. 3. Just like the Toffoli gate, it also realizes the Boolean function $(x_0 \cdot x_1) \oplus x_2$. However, it does this at the expense of a relative phase, such that the whole map is $|x_0\rangle |x_1\rangle |x_2\rangle \rightarrow e^{i\theta(x_0,x_1,x_2)} |x_0\rangle |x_1\rangle |(x_0 \cdot x_1) \oplus x_2\rangle$ where $\theta(x_0, x_1, x_2) = \frac{\pi}{2}(x_0x_1\overline{x_2}) + \pi(x_0\overline{x_1}x_2) + \frac{3\pi}{2}(x_0x_1x_2)$. We express $\theta(x_0, x_1, x_2)$ as a truth table in Fig. 1. In exchange for this relative phase, the same Boolean mapping that cost 7 T-gates in Fig. 3 can now be implemented using 4 T-gates in Fig. 3. The $RTOF$'s inverse, the $RTOF^{\dagger}$, implements the same Boolean mapping, but with a relative phase with the opposite sign. It can be easily checked that the $RTOF^{\dagger}$ may be implemented using the inverse of the gates in Fig. 3, applied in reverse order. Throughout this work, when we talk about $RTOF$s, we also include $RTOF^{\dagger}$s, unless otherwise noted. Note the double circle control on the input $|x_0\rangle$. This denotes which input is connected to the center CNOT gate. We will come back to this in Section 4 later.

Because of this added phase, $RTOF$s cannot be used naively in place of Toffoli gates. In the next section, we propose a method to erase the relative phase that accumulates from using these gates.

3. Reducing T-Count by Replacing with the $RTOF$

To explain our method, first we need to introduce some terminology to analyze the added phases by T- and S- gates.

3.1 Phase Functions

Definition 1: For an n-input quantum Boolean circuit, an **added phase function** is defined as a mapping from one specific pattern of n inputs, $\mathbf{x}$, to the resulting phase to the input state corresponding to $\mathbf{x}$ by the circuit. In the following, we use the following notation $P(\mathbf{x})$ to denote an added phase function:

$$P(\mathbf{x}) = \theta \ (0 \leq \theta < 2\pi). \tag{1}$$

For example, recall again the function for the relative phase for the $RTOF$, which is $\theta(\mathbf{x})$, illustrated in Table 1. This is the phase as a function of its inputs, and so we can consider this the added phase function of the $RTOF$. We can also express it as $\theta(\mathbf{x}) = \frac{\pi}{2}(x_0x_1\overline{x_2}) + \pi(x_0\overline{x_1}x_2) + \frac{3\pi}{2}(x_0x_1x_2)$, which will be useful in our next definition. We see it outlined in the truth table in Table 1. Here we can see that, for example, the input $|101\rangle$ is mapped to $e^{i(3\pi/2)} |101\rangle$ (that is, an identity for the state, with an added relative phase of $3\pi/2$). We use this in the following definition that is used in our proposed method.

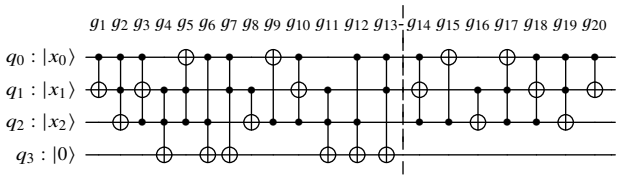


Fig. 4: Algorithm input: a circuit consisting of Toffoli gates.

Definition 2: For an added phase function $P(\mathbf{x})$ of a quantum circuit, we define a **phase function** which is a Boolean function $F_{\theta}(\mathbf{x})$ with respect to the input variables $\mathbf{x}$ of the circuit such that:

$$P(\mathbf{x}) = \theta \text{ if } (F_{\theta}(\mathbf{x}) = 1)$$

For legibility reasons, we drop F_{θ}'s dependence on $(\mathbf{x})$ from the notation from here on out.

As an example, recall the added phase function for the $RTOF$ $\theta(\mathbf{x}) = \frac{\pi}{2}(x_0x_1\overline{x_2}) + \pi(x_0\overline{x_1}x_2) + \frac{3\pi}{2}(x_0x_1x_2)$. We can take $F_{\frac{\pi}{2}} = x_0x_1\overline{x_2}$, $F_{\pi} = x_0\overline{x_1}x_2$, and $F_{\frac{3\pi}{2}} = x_0x_1x_2$. With these $F_{\frac{\pi}{2}}, F_{\pi}, F_{\frac{3\pi}{2}}$, we can then express the added phase function as $\theta(\mathbf{x}) = (\pi/2)F_{\frac{\pi}{2}} + \pi F_{\pi} + (3\pi/2)F_{\frac{3\pi}{2}}$. These $F_{\frac{\pi}{2}}, F_{\pi}, F_{\frac{3\pi}{2}}$ are thus phase functions that will be used to calculate our corrections later. We show these values as a truth table in Table 2. As in the example in Definition 1, we can see that for input $|101\rangle$, $F_{3\pi/2}(101) = 1$, which means that $P(101) = \frac{3\pi}{2}$, in turn meaning that $|101\rangle \rightarrow e^{i3\pi/2} |101\rangle$.

Hereafter, we will use the phrase "phase function/added phase function *at* g_i" to denote the phase function/added phase function on the target qubit after applying gates $g_0, \cdots, g_i$ to the input qubit. Consequently, we use the qualifier *total* to denote the phase function/added phase function of the entire circuit, as opposed to the phase function/added phase function at g_i.

3.2 Erasing Relative Phases

We now present an outline of how to go about replacing the Toffoli gates with RTOFs and erase the relative phases. Because it has a lot in common with our proposed method later, we only present its rough outline and then illustrate in the example that follows how it is executed. We subsequently demonstrate what are some inherent issues that will need to be addressed before proposing the full algorithm.

1 Each Toffoli gate among a quantum Boolean circuit's gates $g_1, \cdots, g_n$ is replaced with an $RTOF$, unless it is part of uncomputation logic, in which case it is replaced with an $RTOF^{\dagger}$.
2 Calculate $F_{\frac{\pi}{2}}, F_{\pi}, F_{\frac{3\pi}{2}}$, the phase functions for the relative phases introduced by the previous step.

Table 3: Calculating the Phase Function for Fig. 4. using the Naive Method

input	init	g_1	g_2	g_3	g_4	g_5	g_6
q_0	00001111	00001111	00001111	00001111	00001111	00011110	00010111
q_1	00110011	00111100	00111100	00110101	00110101	00110101	00110101
q_2	01010101	01010101	01011001	01011001	01011001	01011001	01011001
q_3	00000000	00000000	00000000	00000000	00010001	00011000	00001001
Additional P(x)					$000\frac{\pi}{2}000\frac{\pi}{2}$		$000\frac{3\pi}{2}\frac{\pi}{2}000$
Total P(x)					$000\frac{\pi}{2}000\frac{\pi}{2}$		$0000\frac{\pi}{2}00\frac{\pi}{2}$

input	g_7	g_8	g_9	g_{10}	g_{11}	g_{12}	g_{13}
q_0	00010111	00010111	01110010	01110100	01110100	01110100	01110100
q_1	00111010	00111010	00111010	01010101	01011010	01011010	01011010
q_2	01011001	01101100	01100011	01100011	01100011	01100011	01100011
q_3	00011101	00011011	00011011	00011011	01011001	00111001	01101001
Additional P(x)	$000\frac{\pi}{2}\pi\frac{\pi}{2}00$				$0\frac{\pi}{2}0\pi0\frac{3\pi}{2}0\pi$	$0\frac{3\pi}{2}\frac{\pi}{2}\pi0000$	$0\frac{\pi}{2}\pi\frac{3\pi}{2}0000$
Total P(x)	$000\frac{\pi}{2}\frac{3\pi}{2}\frac{\pi}{2}0\frac{\pi}{2}$				$0\frac{\pi}{2}0\frac{3\pi}{2}\frac{3\pi}{2}00\frac{3\pi}{2}$	$00\frac{\pi}{2}\frac{\pi}{2}\frac{3\pi}{2}00\frac{3\pi}{2}$	$0\frac{\pi}{2}\frac{3\pi}{2}0\frac{3\pi}{2}00\frac{3\pi}{2}$

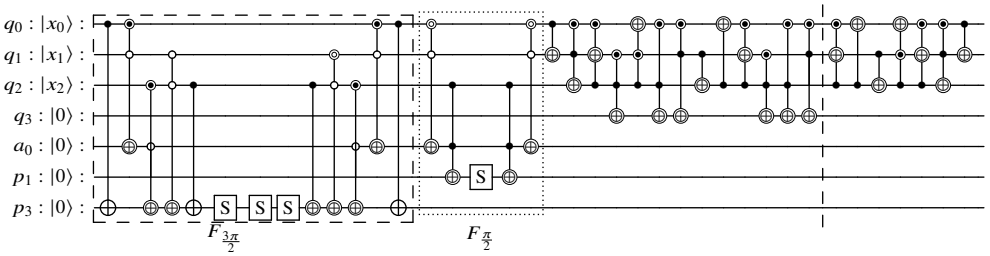


Fig. 5: The transformed Fig. 4 using the Naive Method

3 For each gate g_i, calculate the Boolean logic function $f_{i,j}(\mathbf{x})$ at its target bit q_j after applying g_i, where i is an index we use to iterate through the gates, $x \in \{0,1\}^n$, j is an index for the qubit, and n is the number of input qubits. We then save $f_{i,j}(\mathbf{x})$ for later use.

4 Once we have calculated all $f_{i,j}(\mathbf{x})$, search the circuit for any $f_{i,j}(\mathbf{x})$ that is identical to one of $F_{\frac{\pi}{2}}, F_{\pi}, F_{\frac{3\pi}{2}}$.

5 For every such $f_{i,j}(\mathbf{x})$ found, the appropriate number of S-gates are then applied to the location where the F_θ $f_{i,j}(\mathbf{x})$ is associated with is realized, e.g. if $f_{1,j}(\mathbf{x}) = F_{\frac{\pi}{2}}$, then a single S-gate is applied to q_j, the target bit of g_1, which then drives the S-gate according to $F_{\frac{\pi}{2}}$. We subsequently remove this F_θ from further searches.

6 If no $f_{i,j}(\mathbf{x})$ is found in Step 5, a circuit implementing the remaining $F_{\frac{\pi}{2}}, F_{\pi}, F_{\frac{3\pi}{2}}$ which are not found is synthesized to drive the appropriate S-gates on an ancilla qubit and prepended to the circuit. These act on ancilla bits p_1, p_2, p_3, respectively, such that their final state values are $|0\rangle$. Collectively, these circuits create the state $e^{-i(P(\mathbf{x}))}|000\rangle$ for the added phase function $P(\mathbf{x})$. When these states are introduced to the original set of qubits $e^{iP(\mathbf{x})}|\psi\rangle$ to create the system $e^{-i(P(\mathbf{x}))}|000\rangle \otimes e^{iP(\mathbf{x})}|\psi\rangle$, they erase the relative phase $e^{iP(\mathbf{x})}$ and we are left with $|000\rangle \otimes |\psi\rangle$., where $\otimes$ is the tensor product we used in Sec. 2.1

Hereafter we will refer to this method as the *Naive method*. We illustrate it using Example 1.

Example 1: We transform the circuit in Fig. 4 using the Naive method. We first replace all the Toffoli gates in $g_1 - g_{13}$ (up to the dashed line barrier) with RTOFs because they comprise the computation logic. We then replace the Toffoli gates in $g_{14} - g_{20}$, the uncomputation logic, with RTOF†s to cancel out the phases of the corresponding compute RTOFs in $g_1 - g_{13}$ that were replaced earlier. Table 3 shows the process of calculating the Boolean and phase functions from these RTOFs. Each of the cells shows the truth values of the functions $f_{i,j}(\mathbf{x})$ at the labeled gate g_i, as a one hot string, denoting the value $f_{i,j}(\mathbf{x} = 000), f_{i,j}(\mathbf{x} = 001), \cdots, f_{i,j}(\mathbf{x} = 111)$, where $x_0, x_1, x_2 = \mathbf{x}$, from left to right. We start from *init*, where $f_{0,j}(\mathbf{x}) = x_j$ for $j \in \{0,1,2\}$ and $f_{0,3} = 0$. We denote the added phase function $P(\mathbf{x})$ and show its value at every g_i.

We start by calculating the action of g_1. g_1 acts on q_1 and now implements the function $f_{1,1}(\mathbf{x}) = x_0 \oplus x_1$ such that $q_1 = |x_0 \oplus x_1\rangle$ at g_1. We adjust the q_1 line in the g_1 cell to reflect this function. We do the same for g_2 and g_3, which are Toffoli gates. Note that we do not calculate the added phase here; this is because their contribution to the total added phase function will be canceled by its corresponding uncomputation gates g_{19} and g_{18} respectively. We thus calculate the phase function only up to g_{13}. We assume that the uncomputation logic similarly also reverses the phase because we replaced them with RTOF† earlier.

Now for g_4, we see that it acts on the output bit q_3. Just like our calculation for g_1, we now calculate the function $f_{4,3}(\mathbf{x}) = x_1 x_2$, and update the q_3 line in the g_4 cell accordingly. Additionally, we calculate the added phase function at g_4, which is $P(\mathbf{x}) = \frac{\pi}{2}(x_1 \cdot x_2)$. We depict it in an analogous one hot encoding to the Boolean functions, this time with each digit having possible values $\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$.

We repeat the process for all the gates until we end up with a total added phase function of $P(\mathbf{x}) = \frac{\pi}{2}(\overline{x_0}x_1x_2) + \frac{3\pi}{2}(\overline{x_0}x_1\overline{x_2} + x_0\overline{x_1x_2}) + x_0x_1x_2$ giving us $F_{\frac{\pi}{2}} = \overline{x_0}x_1x_2$ and $F_{\frac{3\pi}{2}} = \overline{x_0}x_1\overline{x_2} + x_0\overline{x_1x_2} + x_0x_1x_2$.

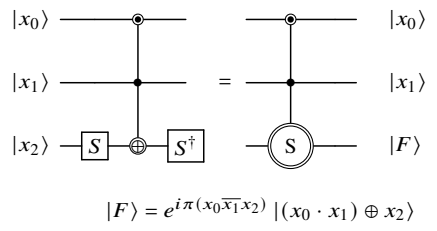


Fig. 6: The $RTOF_S$

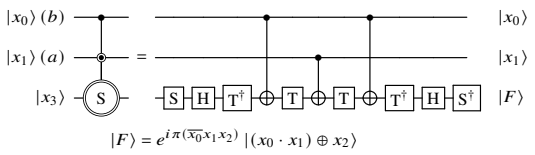


Fig. 7: The $RTOF_S$ with inputs swapped

input	$\theta(\mathbf{x})$
000	0
001	0
010	0
011	0
100	0
101	$+\pi$
110	0
111	0

Table 4: Added phases by an $RTOF_S$.

We then search the circuit for the phase function for $F_{\frac{\pi}{2}}$, comparing the function in q_3 in each column of Table 3, and fail to find it there. Therefore we generate a phase correction element. The phase correction element is generated and set to act on p_1, while using an ancilla a_0. We then apply an S-gate here and then the same set of gates' inverses in reverse order in order to uncompute the output.

We repeat the search again for $F_{\frac{3\pi}{2}}$, fail to find it again, and again prepend an appropriate phase correction element, this time acting on p_3 and applying three S-gates.

The generated circuit is shown in Fig. 5. The phase correction element for $F_{\frac{\pi}{2}}$ is denoted by the dotted box labeled $F_{\frac{\pi}{2}}$. Similarly, the phase correction element for $F_{\frac{3\pi}{2}}$ is denoted by the dashed box labeled $F_{\frac{3\pi}{2}}$. The two of them together use 12 Toffoli gates. The original circuit uses 14 Toffoli gates. Each of these takes 7 T-gates to implement, which gives us a T-count of 98. With the circuit in Figure 5, we instead have 26 RTOFs. Since it takes 4 T-gates to implement an RTOF, the transformation thus has a T-count of 104, resulting in a gain of 6 T-gates.

We can see from Table 3 that the phase functions are complicated, and on top of that, both $F_{\frac{\pi}{2}}$ and $F_{\frac{3\pi}{2}}$ depend on three variables. As Fig 5 shows, the logic to correct this phase can match or even become much bigger than the circuit itself. In the next section, we can simplify this significantly with the introduction of different forms of the $RTOF$.

4. Proposed Algorithm

4.1 Simplifying the Phase Function

As we will see in Section 5, correcting for three different added phase functions results in highly complicated phase correction logic. To remedy this shortcoming, in lieu of using the $RTOF$ as a base element, we instead propose the use of the $RTOF_S$, depicted in Fig. 6. This gate incorporates the an S-gate before the target bit and an $S^\dagger$-gate after the target bit to erase the relative phases on the $|110\rangle$ and $|111\rangle$ states, creating a version of the Margolus gate [6] that only has a π relative phase on the $|101\rangle$ state (note that in this notation, $RTOF_S^\dagger$ is denoted by $S^\dagger$). We illustrate this with Table 4. This makes calculating the phase function easier and possibly reduces the cost of the logic to return the phase.

We can immediately see the advantage of the $RTOF_S$ here. It means each individual $RTOF_S$ contributes a phase of π on the $|101\rangle$ state and subsequent phase-correcting quantum Boolean circuit would only need to act on the π phase.

Another observation is that swapping the CNOT connections of the $RTOF_S$ function, as depicted in Fig. 7, moves the relative phase to different states without affecting the final Boolean function. It is therefore possible to manipulate the final phase function just by an appropriate choice of $RTOF_S$. We observe that a Boolean function f with n inputs generally requires more gates to implement as its ON-set (the number of fulfilling assignments) approaches 2^{n-1}, or half the total number of possibilities, are in the ON-set. We therefore assign the inputs according to whichever assignment pushes the ON-set closer to either 0 or 2^n. If the ON-set is the same with either assignment, then the input with a "lower index" (defined in Algorithm 1 is chosen as "a". We describe this process using Algorithm 1.

We can see that the above manipulation is made much easier by using the $RTOF_S$. Because all the $RTOF_S$ only add π to the total added phase function, this limits our comparison to one phase function F_π.

Finally, we observe that adding two S-gates at arbitrary points on the input qubits can change the phase function but maintain the same Boolean mapping. We can thus search for an appropriate point to insert the S-gates on the output qubits to possibly affect the final phase function. Searching for this appears to be difficult at first glance. However, suppose a circuit has a set of gates $g_1, \cdots, g_n$, defined as in Section 3.2. The original total phase function of the circuit is $F_\pi = F_{\pi_{g_1}} \oplus \cdots F_{\pi_{g_{i-1}}} \oplus F_{\pi_{g_i}} \cdots \oplus F_{\pi_{g_n}}$, where each $F_{\pi_{g_i}}$ denotes the phase function from g_i.

We select a gate g_i that implements the function $f_{i,j}$ from that set. If we then apply two S-gates after its target bit q_j, the added phase function of the whole circuit thus becomes $F_{\pi_{new}} = F_{\pi_{g_1}} \oplus \cdots F_{\pi_{g_{i-1}}} \oplus F_{\pi_{g_i}} \oplus f_{i,j} \oplus \cdots \oplus F_{\pi_{g_n}}$. By commutativity of the XOR, we can also calculate it as $F_{\pi_{new}} = F_{\pi_{g_1}} \oplus \cdots F_{\pi_{g_i}} \cdots \oplus F_{\pi_{g_n}} \oplus f_{i,j} = F_\pi \oplus f_{i,j}$. Again, using the same ON-set heuristic we used for the input assignments three paragraphs above, we thus have a way to

Algorithm 1 Picking Assignments

```
1: function PickAssign(g_i)
2:     /* g_i : A Toffoli gate
3:     q_a,q_b : the control qubits of g_i
4:     ON(f) : counts the number of fulfilling assignments of a Boolean
5:     function f
6:     lower_index(q_a,q_b) : returns q_b if b < a. Otherwise, return q_b
       */
7:     F_a ← F_π phase function of g_i with lower_index(q_a,q_b) as a input
8:     F_b ← F_π phase function of g_i with lower_index(q_a,q_b) as b input
9:     if ON(F_b) is closer to 2^n or 0 than ON(F_a) is then
10:        g_i is changed to an RTOF using lower_index(q_a,q_b) as b input
11:        return F_b
12:    else
13:        g_i is changed to an RTOF using lower_index(q_a,q_b) as a input
14:        return F_a
15:    end if
16: end function
```

reduce the T-count of correcting the resulting total added phase function. For any such $f_{i,j}$, if $F_\pi \oplus f_{i,j}$ has an ON-set closer to either 0 or 2^n than F_π, we add two S-gates after it. Otherwise, we leave the circuit as is and move on to g_{i+1}.

4.2 Proposed Algorithm

The proposed method then replaces all Toffoli gates with $RTOF_S$ and $RTOF_S^\dagger$ instead of $RTOF$ and $RTOF^\dagger$ respectively, while assigning their inputs according to the heuristic in Sec. 4.1. In this way, we seek to simplify the phase correction logic.

Our proposed algorithm is described in Algorithm 2. Line 1 assumes that the input QC has already gone through the process of replacement in Step 1 of the method from Section 3.2.

Lines 16-22 describe the process for calculating the phase functions. Here, we first calculate the Boolean function $f_{i,j}(\mathbf{x})$ implemented on the target bit q_j by the gate g_i. We then execute Algorithm 1, which was explained in Section 4.1. Algorithm 1 returns the phase function of the better assignment, and this is added to the total phase function. We do this for every gate g_i to get the total added phase functions $P(\mathbf{x})$. Then, Lines 24-29 describe the process of finding Boolean functions inside the circuit itself that can be used to drive a phase correction described in Section 4.1. Finally, if $P(\mathbf{x})$ is still non-zero, Lines 31-38 describe the process by which phase correction logic is generated and appended to the quantum circuit, which is a simplified version of Step 6 in Section 3.2. We demonstrate this using Example 2.

Example 2: We transform the circuit in Fig. 4 this time using the proposed method. Table 5 outlines the actions of Lines 16-22. This time there are two lines for added phase for each Toffoli: (a) denotes when the (a) input is connected to the qubit with a lower index and (b) denotes when the (b) input is connected to the qubit with a lower index. "Total Phase" denotes the resulting phase, with the assignment chosen for the Toffoli gate indicated in parentheses after the bit

Algorithm 2 Erasing Relative Phases.

```
1: function EraseRelativePhase(QC, x)
2:     QC : A quantum Boolean circuit of gates g_1, ··· , g_m
3:     containing RTOFs in place of Toffoli gates.
4:     x : an n-bit string x_n x_{n-1} ··· x_1*/
5:     for every f_{0,j} do
6:         f_{0,j} ← x_j //input qubits initialized to x_j
7:     end for
8:     P ← 0
9:     /*
10:    f_{i,j}(x) : is the same variable defined in Sec. 3.2
11:    P(x) : the total added phase function for QC,
12:    initialized to 0
13:    ON(f) : counts the number of fulfilling assignments of
14:    the Boolean function f
15:    */
16:    for every g_i in QC do
17:        /* q_j is the target bit of g_i */
18:        f_{i,j} ← Boolean function obtained from applying g_i to f_{i-1,j}
19:        if q_j is an output qubit and g_i is an RTOF then
20:            P ← P + PickAssign(g_i)
21:        end if
22:    end for
23:    /* Search for places to insert 2 S-gates */
24:    for every f_{i,j} do
25:        if ON((f_{i,j} ⊕ P)) < ON(P) such that P ≠ 0 then
26:            Apply 2 S-gates after g_i on q_j
27:            P ← (f_{i,j} ⊕ P)
28:        end if
29:    end for
30:    /* compose the circuit */
31:    if P ≠ 0 then
32:        /* p is the ancilla qubit to drive the phase. */
33:        appQC ← quantum Boolean circuit that implements F_π on
             p using RTOF.
34:        inv_appQC ← the inverse of appQC
35:        QC ← quantum Boolean circuit with inv_appQC prepended to
             QC
36:        QC ← quantum Boolean circuit with two S-gates on qubit p
             prepended to QC
37:        QC ← quantum Boolean circuit with appQC prepended to QC
38:    end if
39:    return (QC with no relative phases)
40: end function
```

string.

Walking through the example, we can already see the advantages of our method in the first few steps. Unlike in Example 1, when we get to g_4, because the gates are acting on the state $|0\rangle$ there is no phase added. When we get to g_6, we see that both the (a) assignment adds no phase while the (b) assignment adds a phase of π to the $|111\rangle$ state, therefore we choose the (a) assignment and the calculation stays free of relative phases. Moving on to g_7, we see that both (a) and (b) assignments create a phase function with the same size ON-set, so we choose (a).

Moving on to the next step, we take the final F_π and XOR it with each line of columns $g_1 - g_{13}$. We find that XORing each of these with F_π produces a Boolean function with an ON-set which is closer to 2^{n-1} than F_π and so we leave the circuit as it is.

Completing the example, we can see that the resulting phase function $F_\pi = \overline{x_0} \cdot x_1$ can be implemented with an RTOF/RTOF† pair with a negated input acting on p. We

Table 5: Calculating the Phase Function for Fig. 4. using the Proposed Method

input		g_1	g_2	g_3	g_4	g_5	g_6
q_0	00001111	00001111	00001111	00001111	00001111	00011110	00010111
q_1	00110011	00111100	00111100	00110101	00111010	00111010	00111010
q_2	01010101	01010101	01011001	01011001	01011001	01011001	01011001
q_3	00000000	00000000	00000000	00000000	00010001	00011000	00001001
Additional P(x) (a)					00000000		00000000
Additional P(x) (b)					00000000		0000000π
Total P(x)					00000000		00000000(a)

input	g_7	g_8	g_9	g_{10}	g_{11}	g_{12}	g_{13}
q_0	00010111	00010111	01110010	01110100	01110100	01110100	01110100
q_1	00111010	00111010	00111010	01010101	01011010	01011010	01011010
q_2	01011001	01101100	01100011	01100011	01100011	01100011	01100011
q_3	00011101	00011011	00011011	00011011	01011001	00111001	01101001
Additional P(x) (a)	0000π000				000π000π	000π0000	00π00000
Additional P(x) (b)	0000000π				0000π000	0000π000	0000000π
Total P(x)	0000π000(a)				0000000(b)	000π0000(a)	00$\pi\pi$0000(a)

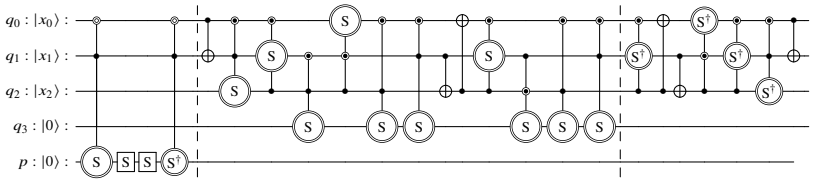
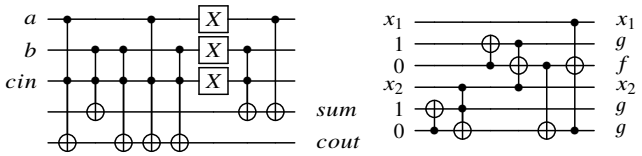


Fig. 8: The circuit from Fig. 4 transformed using the Proposed Method

thus synthesize the $RTOF$ with negated $\overline{x_0}$ input and set its target bit to the ancilla p. We then apply two S gates on p and repeat the same RTOF's inverse RTOF† to uncompute the states.

With 16 total RTOF, the T-count is 64. This is 40 less than the Naive method's T-count of 104, and 34 less than the All-Toffoli case's T-count of 98.



(a) FTR 2007 (b) WD 2009

Fig. 9: Synthesis algorithms for the benchmarks

5. Experimental Results

5.1 Methodology

Our algorithm was implemented using Qiskit[7] and Python. Calculations for the output logic as well as the phase functions were done by operating on Binary Decision Diagram (BDD) representations of Boolean functions in the quantum circuit using the CUDD interface in the TuLiP DD tool[8]. The Qiskit classicalfunction functionality was used to generate the relative-phase correction Boolean logic circuits from the calculated BDDs by converting the latter into the appropriate Boolean function in Python.

A selection of circuits from RevLib[9] were then read from REAL format into Qiskit QuantumCircuit format†. However, some benchmarks do not contain uncomputation gates as required by quantum computation. We can see this in from Fig. 9b with the garbage state $|g\rangle$, which are different from their initial states. We therefore modify them from their normal forms by appending any gates that only act on input qubits and ancilla qubits in the reverse order that they appear in the circuit.

To have a good distribution of circuits, we take benchmarks from RevLib generated with the methods from [10] (WD:2009), [11] (FTR:2007), and [12] (MDS:2005). FTR:2007 is an Exclusive Sum of Products (ESOP) based method that cascades the ESOP terms on the output qubits, and implements each product term as an MPMCT gate. This is shown in Fig 9a. As such, it needs no uncompute logic and every single gate contributes to the phase function. These are chosen to measure the contribution of the phase correction logic, as well as its performance under reasonably complicated phase functions. WD:2009 is a BDD-based method that cascades gates to the output using input bits and ancilla. It creates circuits like Fig. 9b with logic that acts on ancilla that have corresponding uncompute logic. MDS:2005 is a database of mostly hand-designed circuits. Each of them has a unique set of characteristics and provides a mix of different types of logic. Table 6 indicates the source of each benchmark in the column "Type".

Additionally, while we used examples in this paper consisting only of NOT, CNOT, and Toffoli gates (NCT), very few circuits in our benchmark set limit themselves to this set of gates. Within our benchmarks, we see many that use Multi-Polarity Multiple Control Toffoli (MPMCT) gates. While these are not in our original set, we can nonetheless get a T-count for them by using the clean-ancilla MPMCT decomposition method in [4]. For MPMCT gates implemented without a relative phase, we get a T-count of $8n-17$ for inputs $n \geq 4$ (note that the three-input case is the Tof-

†Documentation for all benchmarks used are available at the following URL http://www.revlib.org/realizations.php

Table 6: Experimental Results

Circuit_Name	Type	AllToffoli	TGates	Naive	Just$RTOF_S$	Pairwise	Proposed	vsJust$RTOF_S$%	vsNaive%	vsPairwise%	vsAllToffoli%
cycle10_2_110	MDS:2005	1561	29	948	892	976	892	0	-2.95	8.61	42.86
decod24-enable_126	MDS:2005	357	25	212	204	276	204	0	-30.19	26.09	42.86
hwb6_56	MDS:2005	4039	215	2748	2308	2950	2692	-16.64	-7.35	8.75	33.35
hwb8_115	MDS:2005	30191	919	20444	17252	20006	20292	-17.62	2.14	-1.43	32.79
rd53_130	MDS:2005	665	25	964	564	446	556	1.42	53.73	-24.66	16.39
mod5adder_127	WD:2009	413	21	396	236	296	404	-71.19	25.25	-36.49	2.18
sym9_317	WD:2009	693	43	588	484	432	396	18.18	26.53	8.33	42.86
rd73_312	WD:2009	679	45	1388	732	436	420	42.62	68.59	3.67	38.14
mlp4_245	FTR:2007	9695	191	10140	8556	6110	8692	-1.59	39.74	-42.26	10.35
rd73_252	FTR:2007	2835	111	5860	3044	1950	2940	3.42	66.72	-50.77	-3.7
sao2_257	FTR:2007	3759	43	2652	2148	2274	2148	0	14.25	5.54	42.86
sqn_258	FTR:2007	4347	91	3540	3676	2754	3708	-0.87	22.2	-34.64	14.7
sym9_148	FTR:2007	12509	421	7148	7148	8408	7148	0	-17.63	14.99	42.86
Average								-3.25	20.08	-8.80	27.57

foli gate that we have already studied here). For MPMCT gates implemented up to a relative phase, we get a T-count of $8n - 14$.

The numbers from the experiments in Table 6 are as follows:

- *AllToffoli*, which describe the base T-count of the Clifford+T Toffoli implementation i.e. the All-Toffoli case
- *Naive*, which is the method described in Sec. 3.2
- *Proposed*, which uses the method described in Algorithm 2 in full.
- *Pairwise*, which describes a version of the Proposed method where only the Toffoli gates that act on non-output qubits are replaced by $RTOF_S$ in the circuit (i.e., only the ones that can be canceled out by a balancing $RTOF_S^\dagger$)
- *Just*$RTOF_S$, which is a version of the Proposed method, where instead of calling the PickAssign method from Algorithm 1, it fixes the input assignments to use the lower indexed input as (a).

In addition, there are four sets of comparison numbers:

- *vsToffoli* is $(AllToffoli - Proposed)/AllToffoli$ i.e. this measures the effect of doing the Proposed method vs. not doing anything at all.
- *vsNaive* is $(Naive - Proposed)/Naive$ and measures the effectiveness of introducing the $RTOF_S$ and its corresponding input reassignments to the Naive method described in Sec. 3.2.
- *vsJust*$RTOF_S$ is $(JustRTOF_S - Proposed)/JustRTOF_S$ and is meant to measure the effectiveness of the heuristic introduced in Sec. 4.1 by comparing the Proposed method to merely introducing the $RTOF_S$.
- *vsPairwise* is the result of the equation $(Pairwise - Proposed)/Pairwise$. One may notice that the Pairwise method merely comprises the Toffoli replacement procedure in Algorithm 2. This means that the comparison is to measure the impact of the generated phase correction elements vs. merely performing the replacement of symmetric Toffoli gates.

At the bottom of the columns for vsToffoli, vsNaive, vsJust$RTOF_S$ and vsPairwise, we take the arithmetic average of each column and list it in the "Average" row.

5.2 Analysis

The first thing that we can see is that because of the phase correction logic, the Naive method often performs worse than Pairwise. Many times it performs worse than the AllToffoli method. This is because phase functions get overly complicated. This is seen in the substantial differences between the T-counts of the Naive method and the Pairwise method. The Naive method will need significant improvements such as introducing the $RTOF_S$ in order to become practical. Indeed the vsNaive numbers show an overwhelming advantage in favor of the Proposed method.

The numbers for vsNaive should be considered against the vsToffoli numbers, which show that the Proposed method has, on average, 27.57% lower T-count than the AllToffoli case. While the advantage of the Naive method is unclear in many cases, the Proposed method is much more unambiguous in its advantage over the AllToffoli method.

That being said, while for most cases the Proposed method provides a modest advantage in T-count over the Pairwise method, there still remain cases where it performs much poorer than the Pairwise method. This means the Proposed method has on average 8.80% *higher* T-count in the benchmarks we tested than the Pairwise method. We can see however that the performance differs greatly between WD:2009, FTR:2007, and MDS:2005. Because FTR:2007 does not have any uncomputation logic and every gate contributes to the overall phase function, this means that its T-count savings is dominated by implementing the Toffoli gates that comprise each MPMCT in a relative phase manner. However, this is balanced out by the T-count of the phase correction logic. We see that the benchmarks for FTR:2007 have vastly higher T-counts, on average, than the benchmarks taken from the other two sources under the Proposed method. Finally, vsJust$RTOF_S$ shows only a modest contribution by the heuristic input reassignment, and is many times actually detrimental to the overall T-count. This could be due to the highly complicated Boolean function being implemented, and the heuristic could be failing as the Boolean functions' ON-set get closer to the 2^{n-1} region where it is unclear whether making the ON-set closer to 0 or 2^n will

make it simpler. This suggests that it needs a more sophisticated heuristic to decide which assignment to take. Despite the present disadvantages, however, we still see that several cases, such as rd73_312, still have lower T-count using the Proposed method over the vsJust$RTOF_S$ method. Taken all together, this means that most of the optimization is dominated by the $RTOF$ replacement procedure for gates and their uncomputation logic. However, as we can see, there still remain cases where the Proposed method generates circuits with lower T-count than the Pairwise method. This opens up an interesting further topic of research of how to analytically decide *a priori* whether to use the Proposed method, Pairwise method, or the Just$RTOF_S$ method, before running any generation. In the absence of such a method, however, a practical implementation could simply generate separate circuits using the Proposed method, Pairwise method, and Just$RTOF_S$ method, and then compare their T-counts to decide which implementation to use after the fact, at the cost of additional runtime.

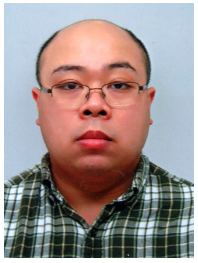
6. Conclusion

This paper proposed a method to convert a quantum circuit to use $RTOF$s instead of Toffoli gates, and then erase the resulting relative phase by adding extra logic. We demonstrate that while there are cases where this could reduce the T-count, there are also many other cases where the correcting logic adds too much to the T-count. We propose several enhancements to mitigate this increase in T-count, including the $RTOF_S$ and a heuristic method to assign inputs in order to simplify the overall phase function.

We find experimentally that with use of mitigation we find more cases where replacing with $RTOF$s shows an advantage. However, there remain many cases where even the simplistic Pairwise method outperforms the Proposed method. Looking closer at these cases, it is apparent that cases which use uncompute logic are better suited for using this method. Additionally, even among the cases we tested, there were some cases where we can get a lower T-count by using the heuristic input assignment in the Proposed method. This suggests that a practical implementation could use some combination of all of these proposals. We conjecture that this can either be done analytically before performing the transformation, or after the fact by selecting the implementation that has the lowest T-count. Future topic of research that have resulted from this study include: development of a better heuristic to select input assignments, and an analytical method to decide which method to use before doing any synthesis.

References

[1] S. Yamashita, S. Minato, and D.M. Miller, "Ddmf: An efficient decision diagram structure for design verification of quantum circuits under a practical restriction," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E91-A, no.2, pp.3793–3802, dec 2008.
[2] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," Phys. Rev. A, vol.52, pp.3457–3467, Nov 1995.
[3] M.A. Nielsen and I.L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, Cambridge University Press, 2010.
[4] D. Maslov, "Advantages of using relative-phase toffoli gates with an application to multiple control toffoli optimization," Physical Review A, vol.93, no.1, p.022311, 2016.
[5] S. Kuroda and S. Yamashita, "Optimization of quantum boolean circuits by relative-phase toffoli gates," RC, vol.14, no.2, 2022.
[6] G. Song and A. Klappenecker, "The simplified toffoli gate implementation by margolus is optimal," 2003.
[7] "Qiskit: An open-source framework for quantum computing," 2021.
[8] "Binary decision diagrams (bdds) in pure python and cython wrappers of cudd, sylvan, and buddy." https://github.com/tulip-control/dd, 2022.
[9] R. Wille, D. Große, L. Teuber, G.W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits," Int'l Symp. on Multi-Valued Logic, pp.220–225, 2008. RevLib is available at http://www.revlib.org.
[10] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," Design Automation Conf., pp.270–275, 2009.
[11] K. Fazel, M. Thornton, and J. Rice, "ESOP-based Toffoli Gate Cascade Generation," Proceedings of the IEEE Pacific Rim Conference on Communications, pp.206–209, 2007.
[12] D. Maslov, G.W. Dueck, and N. Scott, "Reversible Logic Synthesis Benchmarks Page," 2005. http://webhome.cs.uvic.ca/ dmaslov.



David Clarino received his BS EECS and BA Physics from UC Berkeley and MS from Santa Clara University. He is currently a student at the Graduate School of Information Science and Engineering, Ritsumeikan University, Shiga, Japan. His research interests include quantum circuit design and quantum circuit verification and has previously worked on SoC design verification

Shohei Kuroda received his bachelor's and master's degree from Ritsumeikan University. His research interests include quantum circuit design.

Shigeru Yamashita is a professor at the Department of Computer Science, College of Information Science and Engineering, Ritsumeikan University. He received his B.E., M.E. and Ph.D. degrees in Information Science from Kyoto University, Kyoto, Japan, in 1993, 1995 and 2001, respectively. His research interests include new types of computation and logic synthesis for them. He received the 2000 IEEE Circuits and Systems Society Transactions on Computer-Aided Design of Integrated Circuits and Systems Best Paper Award, SASIMI 2010 Best Paper Award, 2010 IPSJ Yamashita SIG Research Award, and 2010 Marubun Academic Achievement Award of the Marubun Research Promotion Foundation. He is a senior member of IEEE and a member of IPSJ.