Calculating the population count of a binary string on a Quantum Computer

Exam ACIT4321 Quantum Information Technology

Bernt Moritz Schmid Olsen (s341528), Rolf Alexander Klæboe (rokla9659)

November 2024, OsloMet

Abstract

In this work we explore different ways to solve the popcount problem on a Quantum Computer. The popcount problem, is an important problem in Computer Science and finding efficient ways to solve this problem on a Quantum Computer is valuable. We focus specifically on constructing circuits that find the j-th bit of the output in binary form. This means we only use n+1 qubits where n is the number of input bits. Through our work, we were able to find a general approach to solve the problem for any problem size for $j \in \{1, 2, 3\}$ values. This involves using controlled not gates with 2^{j-1} control bits. For the problem instance where $j = 2$ we were also able to find a more efficient circuit by utilizing the Quantum Fourier Transform (QFT).

Contents

1 Introduction

The popcount problem is the problem of counting the number of set bits in a binary string. It is short for population count problem and can also be described as the bit-counting problem [1] or in more general terms: finding the hamming weight of a bit string. The hamming weight is the function that counts the number of non-zero elements among a string of elements from an alphabet that has a well defined zero-symbol. [2]. Hamming weight is significant in quantum computing for a number of different applications such as quantum cryptography, quantum algorithms such as Grover's Search algorithm[3], and finding energy states in Ising representations.

1.1 Problem definition

In this work, we worked with a specific popcount problem with some constraints. For a given string $z = z_0z_1z_2...z_n$ of n bits, the popcount problem is to find the number of set bits in the string. The output is a positive integer that can be expressed in binary. The popcount function of a bit string z is the same as hamming weight function on z, which is sometimes denoted $\omega(z)$.

$$\omega(z) = z_0 + z_1 + ... + z_n = y, y = y_0y_1y_2y_3...y_m$$

The function produces a bit string as output with length $m = \lfloor log_2(n) \rfloor + 1$. We try to implement the hamming weight function for a bit string of size n for one of the output bits y_j where $0 \leq y_j \leq m$. For that we can only use $n + 1$ qubits (qubit constraint) and the gates allowed to be used is the CNOT gate and any single qubit gate (gate type constraint).

In the subsequent sections we walk trough circuits that solve specific popcount problem instances. First we work with the $j = 1$ with $n = 3$ and $n = 4$. Next, we present solutions for the $j = 2$ cases for $n_{=}3$ and $n = 4$. Finally we also explain how the $j = 3$ can be solved for $n = 4$. We also explain how the solutions can generalize to any n for the specific j.

2 Solutions

2.1 Solutions for $j = 1$ popcount problems

For an input of $n = 3$ qubits, the largest popcount is 3 corresponding to $|111\rangle$. The number of output bits is given by $m = \lfloor log_2(3) \rfloor + 1 = 1 + 1 = 2$. The number of different input bit strings is 2^3. With this information we can create the truth table for the problem instance (see Table 1). From the truth table it becomes clear that to find the popcount for $j = 1$, which is the least significant bit we need a circuit that produces $y_1 = 1$ when $\omega(z)$ is even and $y_1 = 0$ when $\omega(z)$ is odd. This is trivial to do with n number of CNOT gates where each input is assigned as one of the control qubits of the n qubits and the target is set to the output bit (see Figure 1). In this setup the X gate is applied as many times as there are set bits in the output. Thus, if $\omega(z)$ is odd we get $X^{2n+1} = (X^2)^n \cdot X = X$ and for when $\omega(z)$ is even, we get Identity: $X^{2n} = (X^2)^n = I^n = I$.

The exact same approach can be used to find the solution to the $n = 4$ problem instance. note however that this time 3 bits are needed to represent y entirely: $m = \lfloor log_2(4) \rfloor + 1 = 2 + 1 = 3$. The truth table is found in Table 2. Note that also in this case $|y_1\rangle = |1\rangle$ if $\omega(z)$ is odd and $|y_1\rangle = |0\rangle$ when $\omega(z)$. We can thus extend the same concept with CNOT gates to this problem instance (see Figure 2)

From the two solutions derived above, the questions of generalizing this solution to $n \geq 1$ naturally arises. We can observe that for any $\omega(z)$ function value ($V_\omega \in \mathbb{N}$), the least significant bit in the binary representation is always 1 if the total entire binary number is odd and 0 if it is even. This arises from the binary representation of an integer where the least significant bit is contributing the term $2^0 = 1$ if it is set. All other terms are a multiple of 2:

$$(z_{n-1}...z_2z_11)_2 = z_{n-1} \cdot 2^{n-1} + ... + z_1 \cdot 2^1 + 1 = 2(z_n \cdot 2^{n-2} + ... + z_1 \cdot 2^0) + 1 = 2N + 1$$

$$(z_{n-1}...z_2z_10)_2 = z_{n-1} \cdot 2^{n-1} + ... + z_1 \cdot 2^1 + 0 = 2(z_n \cdot 2^{n-2} + ... + z_1 \cdot 2^0) = 2N$$

where N is some positive integer.

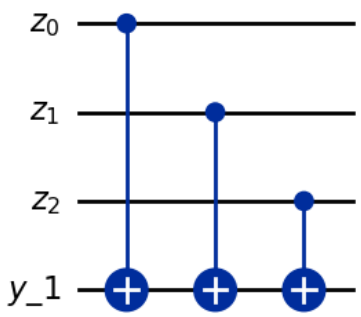

Figure 1: Circuit diagram for n=3, j=1 solution

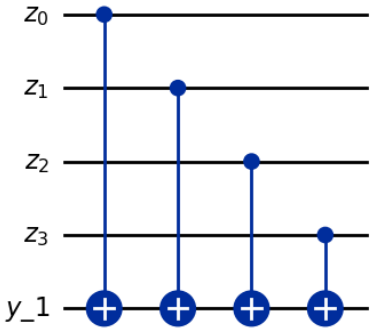

Figure 2: Circuit diagram for n=4, j=1 solution

$\|z_1, z_2, z_3\rangle$	$\omega(\|z\rangle) = \|y\rangle$	$\|y_2\rangle$	$\|y_1\rangle$
$\|000\rangle$	$\|00\rangle$	$\|0\rangle$	$\|0\rangle$
$\|001\rangle$	$\|01\rangle$	$\|0\rangle$	$\|1\rangle$
$\|010\rangle$	$\|01\rangle$	$\|0\rangle$	$\|1\rangle$
$\|011\rangle$	$\|10\rangle$	$\|1\rangle$	$\|0\rangle$
$\|100\rangle$	$\|01\rangle$	$\|0\rangle$	$\|1\rangle$
$\|101\rangle$	$\|10\rangle$	$\|1\rangle$	$\|0\rangle$
$\|110\rangle$	$\|10\rangle$	$\|1\rangle$	$\|0\rangle$
$\|111\rangle$	$\|11\rangle$	$\|1\rangle$	$\|1\rangle$

Table 1: Truth table for $n = 3$, $j = 2, j = 1$

It is therefore clear that the solution for $n = 4$ and $n = 3$ can be extended to used for all $n \geq 1$ with $j = 1$.

2.2 Solutions to the $j = 2$ popcount problems

In our work with this project, we found two different approaches for solving the popcount problem for $j = 2$ for any $n \geq 2$. First we present an approach which involves using the Toffoli gate to test every possible pair of input combinations and apply the X gate as many times as there are active pairs of inputs. The other approach uses the Quantum Fourier Transform (QFT) to solve the problem.

$\|z_1, z_2, z_3, z_4\rangle$	$\omega(\|z\rangle) = \|y\rangle$	$\|y_3\rangle$	$\|y_2\rangle$	$\|y_1\rangle$
$\|0000\rangle$	$\|000\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0001\rangle$	$\|001\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$
$\|0010\rangle$	$\|001\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$
$\|0011\rangle$	$\|010\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$
$\|0100\rangle$	$\|001\rangle$	$\|0\rangle$	$\|0\rangle$	$\|1\rangle$
$\|0101\rangle$	$\|010\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$
$\|0110\rangle$	$\|010\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$
$\|0111\rangle$	$\|011\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1000\rangle$	$\|001\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|1001\rangle$	$\|010\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1010\rangle$	$\|010\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1011\rangle$	$\|011\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$
$\|1100\rangle$	$\|010\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1101\rangle$	$\|011\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$
$\|1110\rangle$	$\|011\rangle$	$\|0\rangle$	$\|1\rangle$	$\|0\rangle$
$\|1111\rangle$	$\|100\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$

Table 2: Truth table for $n = 4$, $j = 3, j = 2, j = 1$

2.2.1 Solving the n=3 and n=4 problems using Toffoli gates

In order to find the popcount for $j = 2$, we simply count if two or more bits are $|1\rangle$. For an input of 3 qubits, we have 4 such cases as seen in Table 1. From the table we see that we want to apply X on the $y_{j=2}$ qubit when two or more bits are $|1\rangle$, this can be achieved with three Toffoli gates arranged as given by Figure 3.

The same concept applies to the case where $n = 4$. However, this time it is not enough to just check if there are at least two set bits because for the case $\omega(1111_2) = 4 = 100_2$ the output should be 0. We should therefore analyse how the similar circuit for $n = 4$, where all possible combinations of input pairs are connected to a unique Toffoli (see Figure 4), handles the different input. Henceforth, we call this approach the Exhaustive Toffoli (ET) approach.

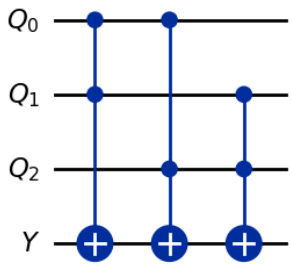

Figure 3: Circuit diagram for the $n = 3$, $j = 2$ popcount problem

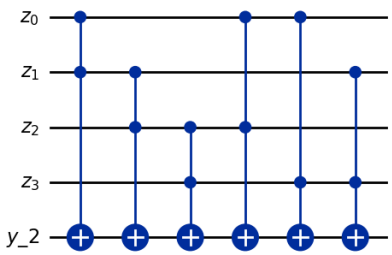

Figure 4: Circuit diagram for the $n = 4$, $j = 2$ popcount problem

There are four possible hamming weights that the algorithm needs to handle: $\omega(z) \in 0, 1, 2, 3, 4$. When $z = 0000_2$, the expected output is 0 and the proposed circuit (see Figure 4) produces no effect on the output. The same applies for any cases where $\omega(z) = 1$ $(1000_2, 0100_2, 0010_2, 0001_2)$: no Toffoli gate will apply the X gate in the target if not both bits are set. For the case where two qubits are in the $|1\rangle$ state, we can clearly deduce that only one Toffoli will be active. This is because each Toffoli is controlled by a unique pair of input bits. If there are three set bits, it is less clear because there must be more than one activated Toffoli gate. Since $3_{10} = 11_2$, we know that $y_2 = 1$ for $\omega(z) = 3$. If we now look at the ways those three inputs can be combined to input a unique Toffoli, we see that it is a combination problem without repetition. The number of unique combinations of two input qubits and thus Toffoli activations, are given by the binomial coefficient:

$$\binom{3}{2} = \frac{3 \cdot 2 \cdot 1}{2 \cdot 1(3-2)!} = 3$$

This means the X gate is applied 3 times on the target qubit for any z such that $\omega(z) = 3$:

$$|z_n^{\omega(z)=3}\rangle \otimes |0\rangle \xrightarrow{ET_4} |z_n^{\omega(z)=3}\rangle \otimes X^3 |0\rangle = |z_n^{\omega(z)=3}\rangle \otimes X^2 \cdot X |0\rangle = |z_n^{\omega(z)=3}\rangle \otimes |1\rangle$$

Where $z_n^{\omega(z)=3}$ is one of the input bit strings with three set bits and ET_4 denotes the Exhaustive Toffoli circuit for four input qubits.

The last case is when all four bits are set. But in that case all Toffoli gates must be activated. The total number of X gates applied is in this case $\binom{4}{2} = 3 \cdot 2 = 6$. This is an even number, which means the effect of the X gates is canceled out because $X^6 = (X^2)^3 = I^3 = I$. This shows that the Exhaustive Toffoli approach solves the problem. However, the original task was to solve the popcount problem with only single qubit gates or CNOT gates. For this solution to be feasible, it needs to be shown that the Toffoli

gate can be decomposed with CNOT and single qubit gates. In the next section we will show how this solution fulfills the gate type constraint.

2.2.2 Decomposition of the Toffoli gate

Toffoli gates can be decomposed by CNOT and single qubit gates[4] as illustrated by Figure 5, where H is the Hadamard transform and T is the phase gate $P(\varphi)$ where $\varphi = \frac{\pi}{4}$, and $T^\dagger$ is the phase gate with $\varphi = -\frac{\pi}{4}$. To see that this is indeed a decomposition of a Toffoli gate we evaluate it for the possible inputs. For input $|000\rangle$ we can ignore the CNOTs making each qubit independent leaving us with $T|0\rangle \otimes TT^\dagger|0\rangle \otimes HT^\dagger TT^\dagger TH|0\rangle$ where T and $T^\dagger$ are unitary matrices which cancel out while and subsequently since the Hadamard transform is involutory HH cancel out as well giving us back $|000\rangle$
For inputs $|100\rangle$ & $|010\rangle$ we see that the $TT^\dagger$ still cancel out and the input is returned.

For an input of $|110\rangle$ we evaluate the decomposed Toffoli gate in Table 3.

$H_3\|110\rangle$	$\rightarrow$	$\|11\rangle \otimes \frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$
$Cnot(1,2)\|11\rangle \otimes \frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$	$\rightarrow$	$\|11\rangle \otimes \frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$
$\|11\rangle \otimes T^\dagger \frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$	$\rightarrow$	$\|11\rangle \otimes \frac{\|0\rangle+e^{-i\pi/4}\|1\rangle}{\sqrt{2}}$
$Cnot(0,2)\|11\rangle \otimes \frac{\|0\rangle+e^{-i\pi/4}\|1\rangle}{\sqrt{2}}$	$\rightarrow$	$\|11\rangle \otimes \frac{e^{-i\pi/4}\|0\rangle+e^{i\pi/4}\|1\rangle}{\sqrt{2}}$
$\|11\rangle \otimes T\frac{\|0\rangle+e^{-i\pi/4}\|1\rangle}{\sqrt{2}}$	$\rightarrow$	$\|11\rangle \otimes \frac{e^{-i\pi/4}\|0\rangle+e^{i\pi/4}\|1\rangle}{\sqrt{2}}$
$Cnot(1,2)\|11\rangle \otimes \frac{e^{-i\pi/4}\|0\rangle+e^{i\pi/4}\|1\rangle}{\sqrt{2}}\sqrt{2}$	$\rightarrow$	$\|11\rangle \otimes \frac{e^{i\pi/4}\|0\rangle+e^{-i\pi/4}\|1\rangle}{\sqrt{2}}$
$\|1\rangle \otimes T\|1\rangle \otimes T^\dagger \frac{e^{i\pi/4}\|0\rangle+e^{-i\pi/4}\|1\rangle}{\sqrt{2}}\sqrt{2}$	$\rightarrow$	$\|1\rangle \otimes e^{i\pi/4}\|1\rangle \otimes \frac{e^{i\pi/4}\|0\rangle+e^{-i\pi/2}\|1\rangle}{\sqrt{2}}$
$Cnot(0,2)\|1\rangle \otimes e^{i\pi/4}\|1\rangle \otimes \frac{e^{i\pi/4}\|0\rangle+e^{-i\pi/2}\|1\rangle}{\sqrt{2}}$	$\rightarrow$	$\|1\rangle \otimes e^{i\pi/4}\|1\rangle \otimes \frac{e^{-i\pi/2}\|0\rangle+e^{i\pi/4}\|1\rangle}{\sqrt{2}}$
$Cnot(0,1)\|1\rangle \otimes e^{i\pi/4}\|1\rangle \otimes T\frac{e^{-i\pi/2}\|0\rangle+e^{i\pi/4}\|1\rangle}{\sqrt{2}}$	$\rightarrow$	$\|1\rangle \otimes e^{i\pi/4}\|0\rangle \otimes \frac{e^{-i\pi/2}\|0\rangle+e^{i\pi/2}\|1\rangle}{\sqrt{2}}$
$T\|1\rangle \otimes T^\dagger e^{i\pi/4}\|0\rangle \otimes H\frac{e^{-i\pi/2}\|0\rangle+e^{i\pi/2}\|1\rangle}{\sqrt{2}}$	$\rightarrow$	
$e^{i\pi/4}\|1\rangle \otimes e^{i\pi/4}\|0\rangle \otimes \frac{\frac{e^{-i\pi/2}\|0\rangle+e^{-i\pi/2}\|1\rangle}{\sqrt{2}}+\frac{e^{i\pi/2}\|0\rangle-e^{i\pi/2}\|1\rangle}{\sqrt{2}}}{\sqrt{2}}$		
$= e^{i\pi/4}\|1\rangle \otimes e^{i\pi/4}\|0\rangle \otimes \frac{e^{-i\pi/2}\|0\rangle+e^{-i\pi/2}\|1\rangle+e^{i\pi/2}\|0\rangle-e^{i\pi/2}\|1\rangle}{2}$	$=$	$e^{i\pi/4}\|1\rangle \otimes e^{i\pi/4}\|0\rangle \otimes \frac{-i\|0\rangle-i\|1\rangle+i\|0\rangle-i\|1\rangle}{2}$
$Cnot(0,1)e^{i\pi/4}\|1\rangle \otimes e^{i\pi/4}\|0\rangle \otimes \frac{-2i\|1\rangle}{2}$	$\rightarrow$	$e^{i\pi/4}\|1\rangle \otimes e^{i\pi/4}\|1\rangle \otimes -i\|1\rangle$
$= -ie^{i\pi/4}e^{i\pi/4}\|111\rangle$	$=$	$\|111\rangle$

Table 3: Confirming Toffoli decomposition

We see in Table 3, that the decomposed Toffoli gate will map $|110\rangle \rightarrow |111\rangle$ using only

Cnot and T gates confirming the solutions given for $j = 2$ using an exhaustive Toffoli approach in section 2.2.1.

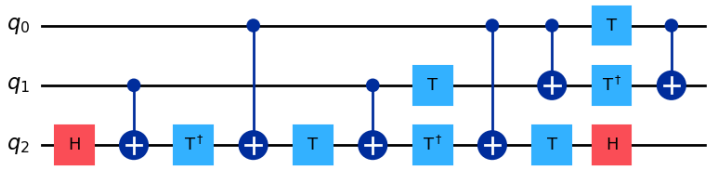

Figure 5: Toffoli definition

2.2.3 Simplified Toffoli

Aside from the definition of a Toffoli gate given by Figure 5 we can utilize the simplified Toffoli gate also known as the Margolis gate [4] illustrated in Figure 6, which was shown to be optimal for a single Toffoli gate [5] with a minimum of 3 Cnots and 4 single qubit gates.

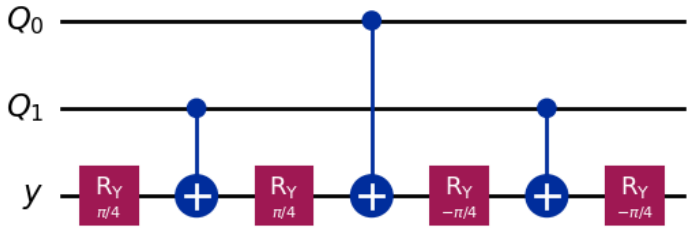

Figure 6: Margolis Gate

We are able to simplify this further since the computational basis is not impacted by relative phase. When multiple Margolis gates act sequentially together we can further reduce the gate number by combining redundant gates as suggested in this blogpost [6].

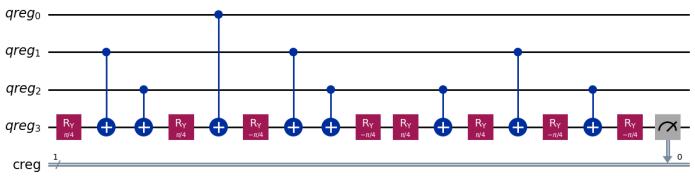

Figure 7: Combining two margolis gates

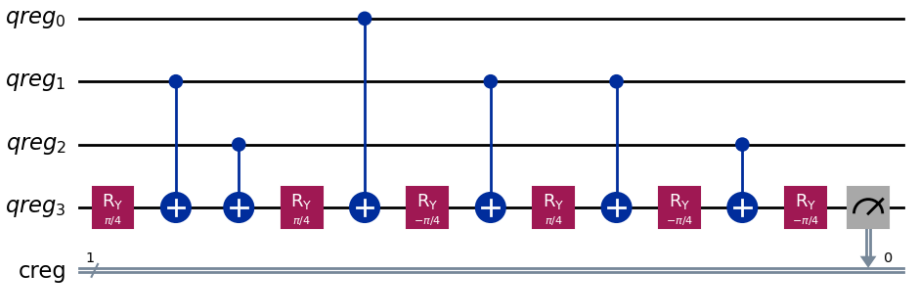

Figure 8: Simplified circuit diagram for $n = 3, j = 2$ popcount problem

Since $Toffoli(q_0, q_1, q_3)$ & $Toffoli(q_0, q_2, q_3)$ both involve q_0 as a control bit, instead of recomputing the operations for q_0 we can instead add two $Cnot(q_2, q_3)$ into $Toffoli(q_0, q_1, q_3)$ as shown in Figure 7. As we can see the two central $Cnot(q_2, q_3)$ and R_y gates cancel out allowing us to remove them giving us the resulting circuit illustrated in Figure 8. To verify this solution for the $j = 2, n = 3$ problem we used Qiskit to simulate the circuit. The code can be examined in appendix4.0.1, except the `create_QPE_popcount_circuit_rcx()` is exchanged for 4.0.2.

2.2.4 Generalizing the Exhaustive Toffoli approach to $n \geq 2$

We can clearly see a pattern in the specific solutions for the $n = 3$ and $n = 4$ problem instances. In both cases all possible combinations of input qubit pairs are covered by one Toffoli gate. The number of Toffoli gates is given by the binomial coefficient $\binom{n}{2}$ each time: $\binom{3}{2} = 3, \binom{4}{2} = 6$. An obvious question is thus if this approach can be used for any $n \in \mathbb{N}$ with $j = 2$.

To answer that question we can analyse how the y_2 output bit behaves for any $\omega(z) \in \mathbb{N}$ value. The y_2 qubit is the second least significant bit in a binary number. The decimal value that this bit corresponds to when set to 1 is 2. If it is set to 0, it is 0 in decimal value. The contribution from this bit can be interpreted as periodic starting from 0_{10} with a period of 4_{10} where the bit is set to 0 in the first two cases and 1 in the last two. For y_1 we showed that the circuit generalizes because the circuit flips the output qubit only if the number of activated CNOTs is odd. This could be expressed with modular arithmetic where we use that the decimal value is divisible by 2 (even) if the bit should be 0 and not divisible by 2 (odd) if it should be 1:

$$|z_n, y_2\rangle = |z_n\rangle \otimes X^{\omega(z_n) \bmod 2} |0\rangle$$

The general circuit for $j = 2$ should therefore implement a function that given the number of set bits $\omega(z)$ gives an output alternating between two odd decimal values and two even

integer numbers. If we now look at the binomial coefficient which determines the number of times the X gate is applied to y_2 and the cases where $k = 2$, we can see that exactly that pattern of even and odd integers occur in the output.

$$\binom{n}{k} = \frac{\prod_{i=n-k}^{n} i}{k!}$$

For $k = 2$ we can observe that the numerator consists of the two factors $n - 1$ and n. Every other integer in the lexicographically ordered $\mathbb{Z}$ must be an even number, thus there is always at least one occurrence of the factor 2 in the numerator. So for all n such that n or $n - 1$ only contains one factor 2, the result will be an odd number. In all other cases the, coefficient is even. The binomial coefficient is in other words even every time n or $n - 1$ is a multiple of $4 = 2 \cdot 2$. A multiple of 4 occurs each 4-th integer in the lexicographic ordered series of integers. Every such integer also accounts for two even binomial coefficients: once for when it is equal to n and once for when it is equal to $n-1$.

We can illustrate this by looking at the first four integers. The first integer is 2, which is not a multiple of 4 and results in an odd binomial coefficient. The same applies to $n = 3$. The next two are even as the first multiple is reached for $n = 4$ and for $n = 5$ we have $n - 1 = 4$. Note that the binomial coefficient $\binom{n}{2}$ is not defined for $n \in \{0, 1\}$, which is fine because for those cases the y_2 output bit is not defined. It is therefore clear that the binomial coefficient provides the output pattern we are looking for to express the function in modular arithmetic like for y_1. We can write:

$$y_2 = \binom{\omega(z_n)}{2} \; mod \; 2$$

Since the number of activated Tofflo gates in the Exhaustive Toffoli circuit, we can express the effect of the circuit ET_n for any $n \geq 2$ like this:

$$|z_n, y_2\rangle = |z_n\rangle \otimes X^{\binom{\omega(z_n)}{2} \; mod \; 2} |0\rangle$$

2.2.5 Finding y_2 with the Quantum Fourier Transform

From the previous section we know that using the ET circuit does generalize to all $n \geq 2$ for the $j = 2$ problem configuration. Another approach to solve the popcount problem on a Quantum Computer can be found in [7]. In this case they used the Quantum Fourier Transform (QFT) to solve the problem for any $n \in \mathbb{N}$ for all y_j. This is thus an interesting starting point to create a general solution for the problem configuration $n \geq 2$ with $j = 2$. Table 4 shows the general circuit with:

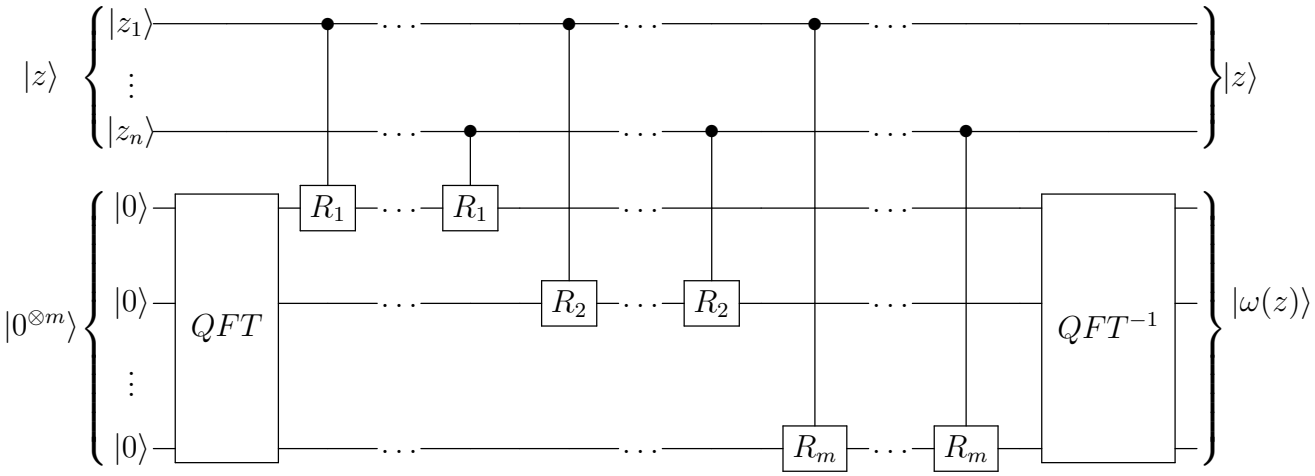

Table 4: The solution presented by [7]

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$$

One problem with this solution is that it uses m qubits to represent the output. In our case, we are limited to using one qubit (qubit constraint). However, the QFT algorithm depends on access to the less significant bits of the most significant bit. This can be observed from the definition of the QFT [8, p. 363]:

$$QFT_n(|\varphi\rangle) = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} \sum_{k=0}^{2^n-1} a_k e^{\frac{2\pi ijk}{2^n}} |j\rangle_n$$

Where n is the number of input bits and $|\varphi\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} a_j |j\rangle_n$. The phase of each input configuration j is the sum of the amplitudes a_k from the input state times $e^{\frac{2\pi ijk}{2^n}}$. In the exponent $\frac{2\pi ijk}{2^n}$ the value of j (current standard basis) determines the phase. Thus the phase of the output depends on the value of the input. To find the correct phase of each qubit, the algorithm needs information from the other qubits. One can better observe this in the circuit diagram (see Figure 9) where it is clear that the less significant bit controls the phase shift of all the more significant bits.

Another important observation is that the input state of the input to the QFT is $|0\rangle^{\otimes m}$. Since all the controlled operators only effect the circuit for the $|1\rangle$ state, only the $H^{\otimes m}$ is applied, resulting in a state $\frac{1}{2^n} \sum_{j=0}^{2^n-1} |j\rangle$. It is therefore only the QFT^{-1} where the dependency is for the less significant bits lies.

When explicitly looking at the $j = 2$ case, the QFT^{-1} is only depending on the least significant bit $y_{j=1}$. It is also important to note that we can change the state of the n input bits after the CR_z gates have been applied, because the needed information is

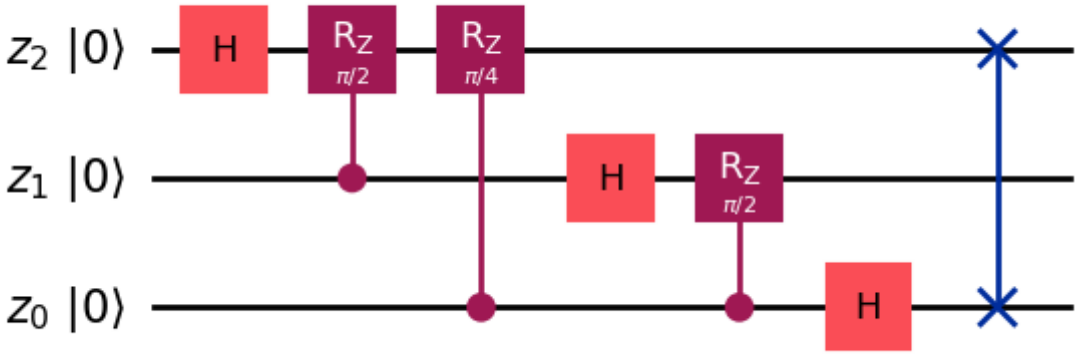

Figure 9: The circuit diagram for QFT_3

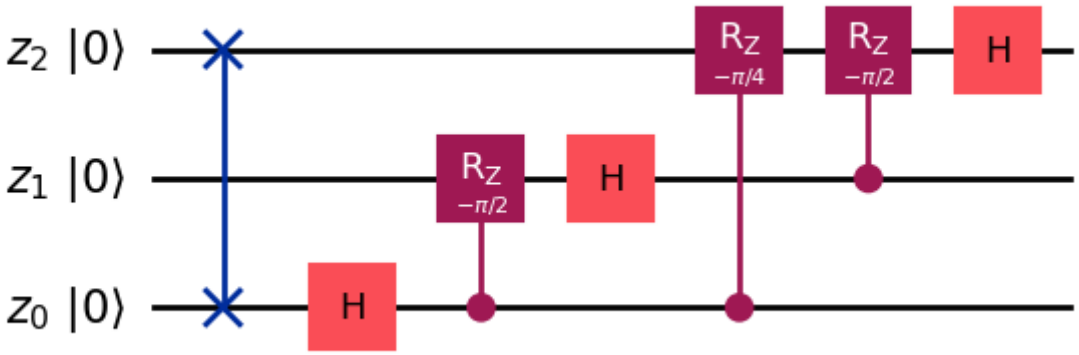

Figure 10: The circuit diagram for ${QFT^{-1}}_3$

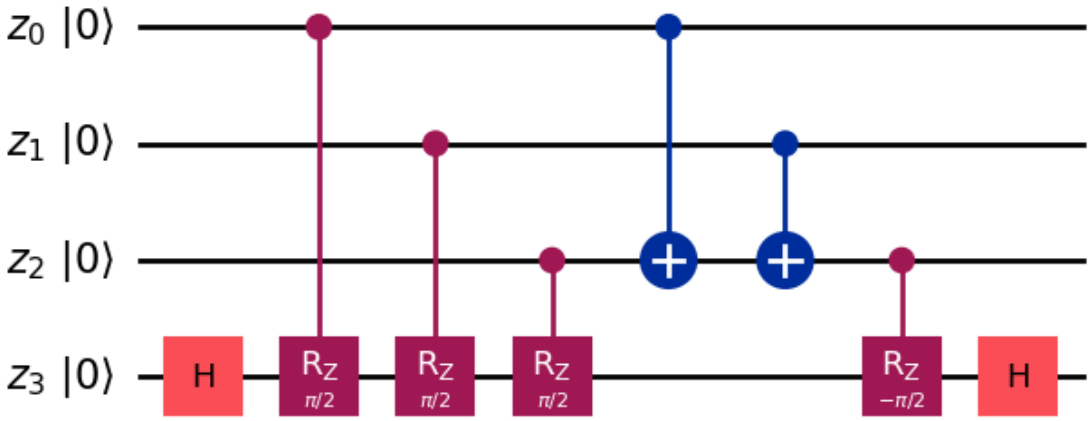

Figure 11: The circuit diagram for the QFT solution to n=3 j=2 popcount problem

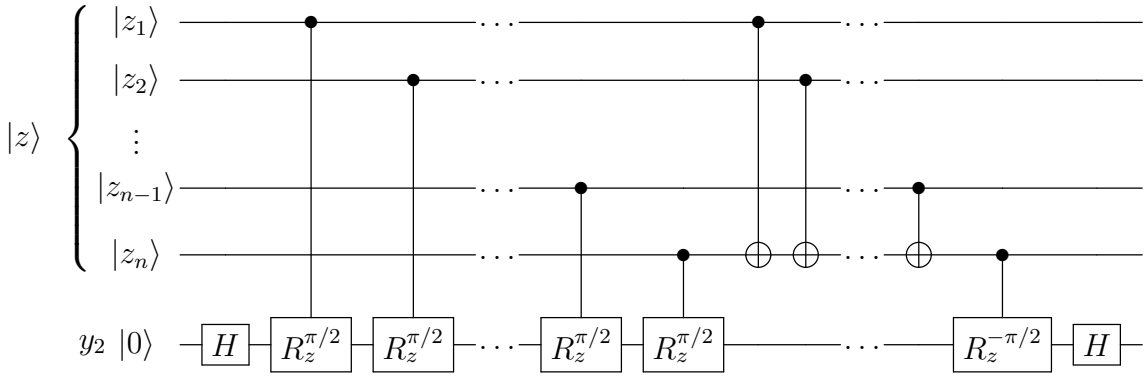

Figure 12: The general solution to the $j = 2$ popcount problem

encoded into $y_{j=2}$ by that point. The question is thus if it is possible to obtain the state of $y_{j=1}$ with the n qubits.

Looking at the solutions for $j = 1$ problems, we see the pattern of applying the X gate to the $y_{j=1}$ qubit as many times as there are $|1\rangle$ qubits in the input. This is done by applying n CNOT gates where each input is set to be control bit to one of them and the target is always the $y_{j=1}$ qubit. If we now reduce the number of CNOT gates by one and set one of the inputs to be the target qibut for all the CNOT gates without being a control qubit for any of the CNOT gates. Since this input qubit is either $|0\rangle$ or $|1\rangle$, setting the input to $|1\rangle$ will be the equivalent of applying the X gate once more. Thus we have a solution to the $j = 1$ configuration with n qubits used and $n - 1$ CNOT gates used. With the $y_{j=1}$ information encoded in another qubit, we are able to apply the controlled $S^\dagger$ gate (i.e. $CR_z(-\frac{\pi}{2})$) with control bit set to the $y_{j=1}$ and the target being the $y_{j=2}$ qubit. What remains of the QFT_2^{-1} algorithm is to change the bit back to the $|0\rangle, |1\rangle$ basis with the Hadamard gate. The described solution is presented as a circuit in Figure 12.

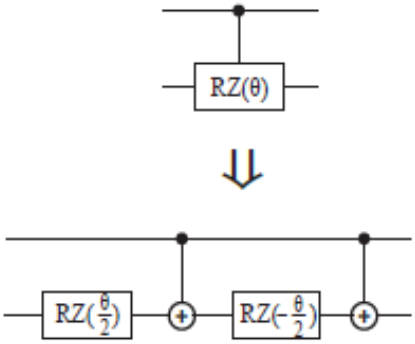

Figure 13: The crz decomposition from [9]

Another limitation is that we can only use the CNOT operator and any single qubit operators. The CR_z gate needs therefore to be decomposed for this solution to be valid. Finding a decomposition that is exact is however difficult. The matrix of the CR_z gate is known [8, p. 293]:

$$CR_z(\varphi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\varphi i} \end{bmatrix}$$

It was difficult to find a circuit that matches that transformation. One example is the decomposition mentioned in [9] (see 13). That approach produces a slightly different matrix. The matrix for the circuit can be found by doing the following calculation:

$$CR_z(\varphi)' = \left(I_2 \otimes R_z\left(\frac{\varphi}{2}\right)\right) \cdot CNOT \cdot \left(I_2 \otimes R_z\left(-\frac{\varphi}{2}\right)\right) \cdot CNOT$$

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

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

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

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

From the matrices, we can see that the operations only affect the target qubit with the $2x2$ matrix in the lower right corner. The difference is that $CR_z(\varphi)'$, in addition to state $|11\rangle$ also has an effect on the $|10\rangle$ state. The lower right $2x2$ matrices for $CR_z(\varphi)$ and $CR_z(\varphi)'$, are representations of the $R_z(\varphi)$ single qubit gate with a difference of a complex factor $e^{\frac{\varphi}{2}i}$ [8, p. 261]:

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

This common factor is a complex number with an absolute value of 1 and does not affect the amplitudes of the basis states. The difference should therefore not have an effect in this case. This can be observed by analysing how the two operations are applied to a qubit in the $|+\rangle$ state:

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{\varphi i} \end{bmatrix} \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 & 0 \\ 0 & e^{\varphi i} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & e^{\varphi i} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} |0\rangle + \frac{e^{\varphi i}}{\sqrt{2}} |1\rangle$$

If we now do the same calculation with the decomposed matrix, we can see that the relative phase is equal [8, p. 248-249]:

$$\begin{bmatrix} e^{-\frac{\varphi}{2}i} & 0 \\ 0 & e^{\frac{\varphi}{2}i} \end{bmatrix} \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} e^{-\frac{\varphi}{2}i} & 0 \\ 0 & e^{\frac{\varphi}{2}i} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} e^{-\frac{\varphi}{2}i} & 0 \\ 0 & e^{\frac{\varphi}{2}i} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{e^{-\frac{\varphi}{2}i}}{\sqrt{2}} |0\rangle + \frac{e^{\frac{\varphi}{2}i}}{\sqrt{2}} |1\rangle$$

$$= e^{-\frac{\varphi}{2}i}\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{e^{\left(\frac{\varphi}{2}+\frac{\varphi}{2}\right)i}}{\sqrt{2}}|1\rangle\right), \quad where \;\; \varphi = \frac{\varphi}{2} + \frac{\varphi}{2}$$

When using the decomposition mentioned above we can see that the operation performed on the target is given by the following matrix:

$$S' = CR_z(\pi/2) = \begin{bmatrix} e^{-\frac{\pi}{4}i} & 0 \\ 0 & e^{\frac{\pi}{4}i} \end{bmatrix}$$

Now we can substitute S' into the function expression and use that it is a diagonal matrix:

$$(S')^{\omega(z)-y_1} = \begin{bmatrix} e^{-\frac{\pi}{4}i(\omega(z)-y_1)} & 0 \\ 0 & e^{\frac{\pi}{4}i(\omega(z)-y_1)} \end{bmatrix}$$

The target qubit starts in the $|0\rangle$ state and is transformed into the $|+\rangle$ state after the first Hadamard gate. Next the S' operators are applied (note that the inverse S gate is applied by subtracting the exponent with y_1):

$$(S')^{\omega(z)-y_1}|0\rangle = \begin{bmatrix} e^{-\frac{\pi}{4}i(\omega(z)-y_1)} & 0 \\ 0 & e^{\frac{\pi}{4}i(\omega(z)-y_1)} \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} e^{-\frac{\pi}{4}i(\omega(z)-y_1)} \\ 0 \end{bmatrix} = e^{-\frac{\pi}{4}i(\omega(z)-y_1)}|0\rangle$$

$$(S')^{\omega(z)-y_1}|1\rangle = \begin{bmatrix} e^{-\frac{\pi}{4}i(\omega(z)-y_1)} & 0 \\ 0 & e^{\frac{\pi}{4}i(\omega(z)-y_1)} \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ e^{\frac{\pi}{4}i(\omega(z)-y_1)} \end{bmatrix} = e^{\frac{\pi}{4}i(\omega(z)-y_1)}|1\rangle$$

With that we can find:

$$(S')^{\omega(z)-y_1}|+\rangle = \frac{\left((S')^{\omega(z)-y_1}|0\rangle + (S')^{\omega(z)-y_1}|0\rangle\right)}{\sqrt{2}}$$

$$= \frac{e^{-\frac{\pi}{4}i(\omega(z)-y_1)}|0\rangle + e^{\frac{\pi}{4}i(\omega(z)-y_1)}|1\rangle}{\sqrt{2}} \xrightarrow{H} \frac{e^{-\frac{\pi}{4}i(\omega(z)-y_1)}H|0\rangle + e^{\frac{\pi}{4}i(\omega(z)-y_1)}H|1\rangle}{\sqrt{2}}$$

$$= \frac{e^{-\frac{\pi}{4}i(\omega(z)-y_1)}(|0\rangle + |1\rangle) + e^{\frac{\pi}{4}i(\omega(z)-y_1)}(|0\rangle - |1\rangle)}{\sqrt{2}^2}$$

$$= \frac{|0\rangle\left(e^{-\frac{\pi}{4}i(\omega(z)-y_1)} + e^{\frac{\pi}{4}i(\omega(z)-y_1)}\right) + |1\rangle\left(e^{-\frac{\pi}{4}i(\omega(z)-y_1)} - e^{\frac{\pi}{4}i(\omega(z)-y_1)}\right)}{2}$$

The amplitudes for the two standard basis have now been separated and we can calculate the probability of measuring each of the states:

$$p_{|0\rangle} = \left|\frac{e^{-\frac{\pi}{4}i(\omega(z)-y_1)} + e^{\frac{\pi}{4}i(\omega(z)-y_1)}}{2}\right|^2$$

$$p_{|1\rangle=}\left|\frac{e^{-\frac{\pi}{4}i(\omega(z)-y_1)}-e^{\frac{\pi}{4}i(\omega(z)-y_1)}}{2}\right|^2$$

where

$$e^{-\frac{\pi}{4}i(\omega(z)-y_1)} = cos(-\frac{\pi}{4}\cdot(\omega(z)-y_1)) + sin(-\frac{\pi}{4}\cdot(\omega(z)-y_1))i$$
$$= cos(\frac{\pi}{4}\cdot(\omega(z)-y_1)) - sin(\frac{\pi}{4}\cdot(\omega(z)-y_1))i$$
$$e^{\frac{\pi}{4}i(\omega(z)-y_1)} = cos(\frac{\pi}{4}\cdot(\omega(z)-y_1)) + sin(\frac{\pi}{4}\cdot(\omega(z)-y_1))i$$
$$= cos(\frac{\pi}{4}\cdot(\omega(z)-y_1)) + sin(\frac{\pi}{4}\cdot(\omega(z)-y_1))i$$

We can identify two common terms: $a := cos(\frac{\pi}{4}\cdot(\omega(z)-y_1))$ and $b := sin(\frac{\pi}{4}\cdot(\omega(z)-y_1))$. With that we can rewrite:

$$e^{-\frac{\pi}{4}i(\omega(z)-y_1)} + e^{\frac{\pi}{4}i(\omega(z)-y_1)} = a - bi + a + bi = 2a$$
$$e^{-\frac{\pi}{4}i(\omega(z)-y_1)} - e^{\frac{\pi}{4}i(\omega(z)-y_1)} = a - bi - (a + bi) = a - a - bi - bi = -2bi$$

The probabilities can now be simplified

$$p_{|0\rangle=}\left|\frac{2a}{2}\right|^2=\left|cos\left(\frac{\pi}{4}\cdot(\omega(z)-y_1)\right)\right|^2$$

$$p_{|1\rangle=}\left|\frac{-2bi}{2}\right|^2=|b|^2=\left|sin\left(\frac{\pi}{4}\cdot(\omega(z)-y_1)\right)\right|^2$$

We see that the measured result depends on the periodic trigonometric functions. Because they are periodic, we priomarily need to concider the the cases where the angle is between 0 and 2π.

$$\frac{\pi}{4}\cdot x = 2\pi \Leftrightarrow x = \frac{2\pi\cdot 4}{\pi} = 8$$

We can now examine the effect of the circuit with a truth table where all possible cases within the first period is listed. Those cases are the 8 first possible function values of $\omega(z) \in \{a \in \mathbb{Z} | 0 \leq a < 8\}$ (see table 5). From the table we can observe that the state of the qubit, always ends up in a deterministic state that matches the expected values of y_2. From this we can see that the QFT solution can be applied to $j = 2$ popcount problem with $n \geq 2$.

To verify this solution for the $j = 2, n = 6$ problem instance, we used Qiskit to simulate the circuit (see Figure 14). The code can be reviewed in Appendix A.

2.2.6 Comparing the two solutions

Since we have found two different circuits that can be used to solve the $j = 2$ popcount problem, it is interesting to compare them in how efficient they are solving the problem. The number of gates needed with regards to the number of input bits, is a metric that can be used to get insight on how effective the algorithms are. It is also interesting to

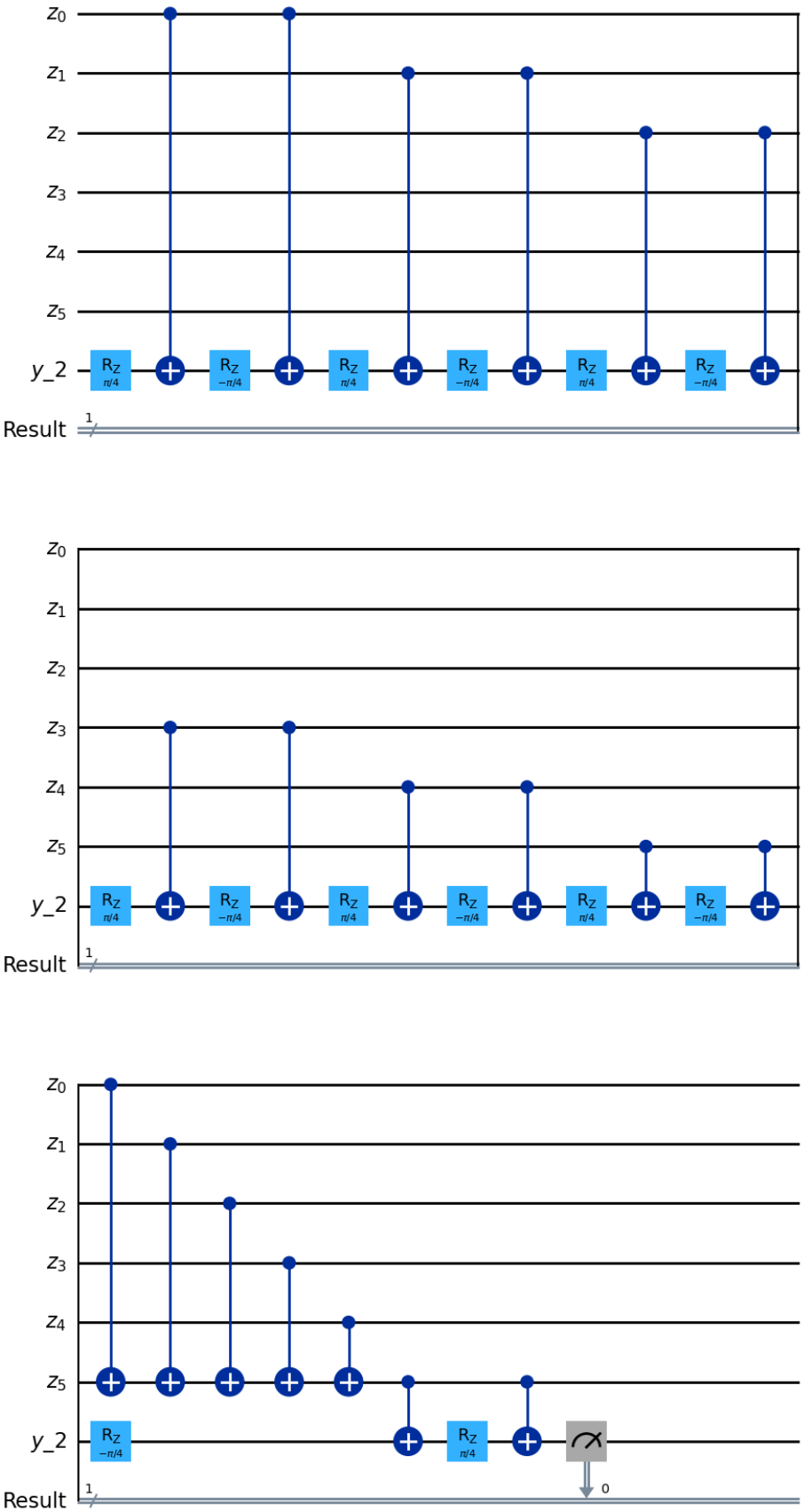

Figure 14: Circuit diagram for $n = 6$, $j = 2$ with CR_z decomposition.

Table 5: The truth table for the QFT solution for the $j = 2$ popcount problem. The truth table contains all special cases within the first period

$\omega(z)$	y_1	expected y_2	$\frac{\pi}{4} \cdot (\omega(z) - y_1)$	$p_{\|0\rangle}$	$p_{\|1\rangle}$
0	0	0	0	$\|cos(0)\|^2 = 1$	$\|sin(0)\|^2 = 0$
1	1	0	$\frac{\pi}{4}(1-1) = 0$	$\|cos(0)\|^2 = 1$	$\|sin(0)\|^2 = 0$
2	0	1	$\frac{\pi}{4}(2-0) = \frac{\pi}{2}$	$\left\|cos\left(\frac{\pi}{2}\right)\right\|^2 = 0$	$\left\|sin\left(\frac{\pi}{2}\right)\right\|^2 = 1$
3	1	1	$\frac{\pi}{4}(3-1) = \frac{\pi}{2}$	$\left\|cos\left(\frac{\pi}{2}\right)\right\|^2 = 0$	$\left\|sin\left(\frac{\pi}{2}\right)\right\|^2 = 1$
4	0	0	$\frac{\pi}{4}(4-0) = \pi$	$\|cos(\pi)\|^2 = 1$	$\|sin(\pi)\|^2 = 0$
5	1	0	$\frac{\pi}{4}(5-1) = \pi$	$\|cos(\pi)\|^2 = 1$	$\|sin(\pi)\|^2 = 0$
6	0	1	$\frac{\pi}{4}(6-0) = \frac{3\pi}{2}$	$\left\|cos\left(\frac{3\pi}{2}\right)\right\|^2 = 0$	$\left\|sin\left(\frac{3\pi}{2}\right)\right\|^2 = 1$
7	1	1	$\frac{\pi}{4}(7-1) = \frac{3\pi}{2}$	$\left\|cos\left(\frac{3\pi}{2}\right)\right\|^2 = 0$	$\left\|sin\left(\frac{3\pi}{2}\right)\right\|^2 = 1$

count the number of specific gates since some gates are more expensive in the amount of resources needed. In [10] it is shown that the CNOT gate has a cost that is 5 times that of the X gate on an ideal Quantum Computer.

From the analysis about the Exhaustive Toffoli approach we know that the number of Toffoli gates is given by the binomial coefficient. Furthermore, the Toffoli gate can be decomposed by using 6 CNOT gates and 9 single qubit gates.

Number of CNOT gates:

$$N_{CNOT} = \binom{n}{2} \cdot 6 = \frac{6n(n-1)}{2} = 3(n^2 - n) = O(n^2)$$

Total number of gates:

$$N_{total} = \binom{n}{2} \cdot 15 = \frac{15n(n-1)}{2} = \frac{15}{2}(n^2 - n) = O(n^2)$$

For the QFT solution tot he $j = 2$ popcount problem, we can count the number of CRZ gates as n (each input corresponds to one CRZ gate). Further more there are two Hadamard gates and one CRZ gate in the QFT and $QFT^{-}1$. The CRZ gate can be decomposed into two RZ gates and two CNOT gates.

Number of CNOT gates:

$$N_{CNOT} = 2 + n \cdot 2 = O(n)$$

Total number of gates:

$$N_{total} = 6 + 4n = O(n)$$

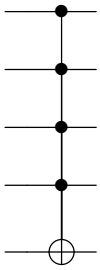

Figure 15: A five qubit gate which is applying the X gate on a target if four control qubits are in state $|1\rangle$.

From this we see that the QFT solution is significantly less costly as it is done with a linear amount of gates compared to the Toffoli approach where the number of gates grows quadratically.

2.3 Solution to the $j = 3$ popcount problem

To solve the $j = 3$ popcount problem, we can use the same thought process as for the Exhaustive Toffoli approach $j = 2$ where all possible input pairs are used as control bits for a unique Toffoli gate. We were able to show that this approach generalizes to problems with any input size because the binomial coefficient with $k = 2$ produces two odd and two even numbers periodically when $\omega(z)$ increases.

The expected output y_3 can be observed to be zero for four values and one for four subsequent values when we iterate over the lexicographically ordered sequence of Integers for $\omega(z)$. In other words we need a circuit that produces four even numbers sequentially and four odd numbers sequentially such that the X gate is canceled out for the four even numbers and applied once for the odd numbers.

If we would be to create a gate that is controlled by four qubits, we would be able to find $\binom{n}{4}$ different input combinations. The number of times the X gate is applied is thus also given by $\binom{\omega(z)}{4}$ where $\omega(z), n \geq 4$. By using the same reasoning as for the $j = 2$ problem instance, we can observe that the binomial coefficient produces first four even numbers and then four odd numbers and repeats this pattern periodically if we sequentially calculate it for the lexicographically ordered set of positive integers greater than 4.

$$\binom{\omega}{4} = \frac{\omega(\omega-1)(\omega-2)(\omega-3)}{4!}$$

The numerator consists of four factors which form a sequence of difference 1. The denominator contains three factors of 2: $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 2^3 \cdot 3$. In the cases where the four factors in the denominator contain only 3 factors of two, the result will be odd. If there are four factors of two $2^4 = 16$, then the result is even. Furthermore, it is important to note that the a multiple of $8 = 2^3$, is also a multiple of four and has always a distance of 4 integers to the next multiple of four. This means that if one of the four factors in

the denominator is a multiple of 8, then none of the others can be a multiple of 4. The distance to the next multiple of 2 is 2, which means that if any of the four factors is a multiple of 8 there must be another factor 2 among the tree other factors. This means that every time one of the factors is a multiple of 8, we get an even number. This scenario is encountered four times in a row because the multiple of 8 is shifted through all four factors. Furthermore, there are four encounters of odd numbers in a row because once a multiple of 8 is passed all four factors, it takes four new integers to reach the next multiple of 8.

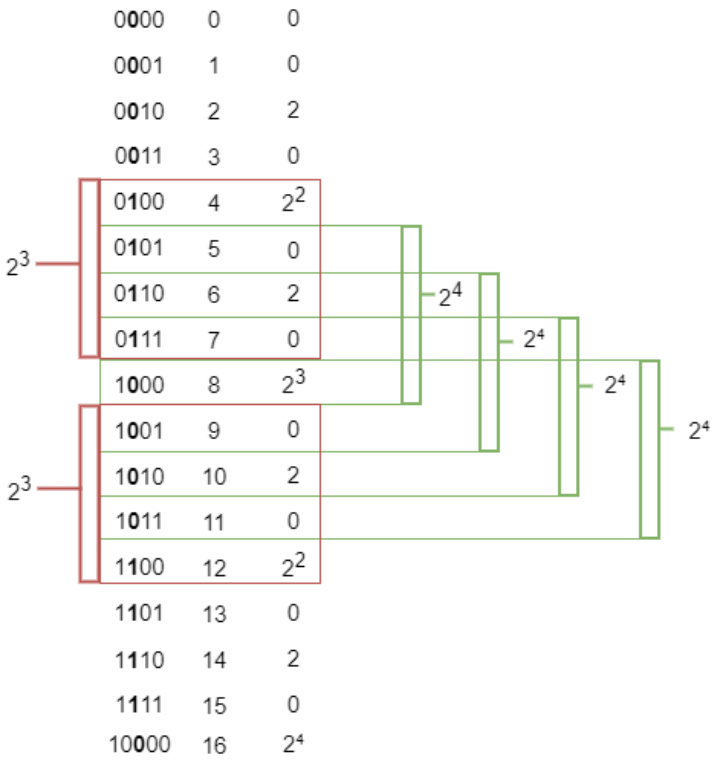

Figure 16: This figure illustrates how the four factors in the denominator contains 2^3 and 2^4 or more 2 factors.

Note that the four first numbers ($\{0, 1, 2, 3\}$) will be zero because non of the gates can be active, and thus the approach also holds for those cases.

With this we have shown that using the five qubit gate (see Figure 15 to cover all possible combinations of four inputs, we can solve the $j = 3$ problem for any $n \geq 4$. To complete the solution we also need to show the decomposition of the gate with four control qubits. For this we applied the general framework for decomposing any multi-qubit gates with n control bits presented in [4, p.17-18]. The core of the method is the following identity:

$$\sum_{k_1} z_{k_1} - \sum_{k_1<k_2} (z_{k_1} \oplus z_{k_2}) + \sum_{k_1<k_2<k_3} (z_{k_1} \oplus z_{k_2} \oplus z_{k_3}) - \ \ldots \ + (-1)^{m-1}(z_1 \oplus z_2 \oplus \ \ldots \ \oplus z_m)$$

$$= 2^{m-1}(x_1 \ \wedge \ x_1 \ \wedge \ ... \ \wedge \ x_m)$$

Where z_k is the k-th input qubit. For the case where $m = 4$) we get the following equation:

$$z_0 + z_1 + z_2 + z_4 - (z_0 \otimes z_1) - (z_0 \otimes z_2) - (z_0 \otimes z_3) - (z_1 \otimes z_2) - (z_1 \otimes z_3) - (z_2 \otimes z_3)$$

$$+(z_0 \otimes z_1 \otimes z_2) + (z_0 \otimes z_1 \otimes z_3) + (z_1 \otimes z_2 \otimes z_3) - (z_0 \otimes z_1 \otimes z_2 \otimes z_3)$$

$$= 2^3(z_0 \ \wedge \ z_1 \ \wedge \ z_2 \ \wedge \ z_3)$$

The circuit can with this be decomposed with CNOT gates and controlled rotation gates of the single qubit gate that the circuit is applying in the target qubit. For a unitary single qubit operation U, the single qubit operator we need is the gate V such that $U = V^{2m}$. In our case we need the $X^{\frac{1}{2 \cdot 4}} = X^{\frac{1}{8}}$. Since CRX is not one of the elementary gates (CNOT or single qubit gate), we need to decompose the CRX gate. However, the decomposition of the CRX gate requires more gates than the CRZ decomposition [9]. An idea is to change the standard basis of the target qubit to $\{|+\rangle, |-\rangle\}$ with a Hadamard gate. After that, applying the CRZ gates has the same effect as CRX in the $\{|0\rangle, |1\rangle\}$ basis. The basis is transformed back with another Hadamard gate to finish the algorithm. In that case $V = Z^{\frac{1}{8}} = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{\pi}{8}i} \end{bmatrix}$.

The complete circuit can now be implemented by adding the V gate each time a positive term in the above identity is True and adding the $V^\dagger$ when a negative term is True. The circuit is constructed by following the grey code sequence. The order can be reviewed in Table 6. A core idea is that CNOT is equvalent with the xor operator and it is unitary, menaing that applying the same CNOT a second time will recreate the previous state. The Finished circuit can be examined in Figure 17.

Table 6: The order in which the cnot (xor) gates are applied can be derived from the grey code sequence as shown in this table

Selected inputs	Boolean expression	Effect on target
0000	-	-
1000	z_0	$R_z(\frac{\pi}{8})$
1100	$x_0 \otimes z_1$	$R_z(-\frac{\pi}{8})$
0100	z_1	$R_z(\frac{\pi}{8})$
0110	$z_1 \otimes z_2$	$R_z(-\frac{\pi}{8})$
1110	$z_0 \otimes z_1 \otimes z_2$	$R_z(\frac{\pi}{8})$
1010	$z_0 \otimes z_2$	$R_z(-\frac{\pi}{8})$
0010	z_2	$R_z(\frac{\pi}{8})$
0011	$z_2 \otimes z_3$	$R_z(-\frac{\pi}{8})$
1011	$z_0 \otimes z_2 \otimes z_3$	$R_z(\frac{\pi}{8})$
1111	$z_0 \otimes z_1 \otimes z_2 \otimes z_3$	$R_z(-\frac{\pi}{8})$
0111	$z_1 \otimes z_2 \otimes z_3$	$R_z(\frac{\pi}{8})$
0101	$z_1 \otimes z_3$	$R_z(-\frac{\pi}{8})$
1101	$z_0 \otimes z_1 \otimes z_3$	$R_z(\frac{\pi}{8})$
1001	$z_0 \otimes z_3$	$R_z(-\frac{\pi}{8})$
0001	z_3	$R_z(\frac{\pi}{8})$

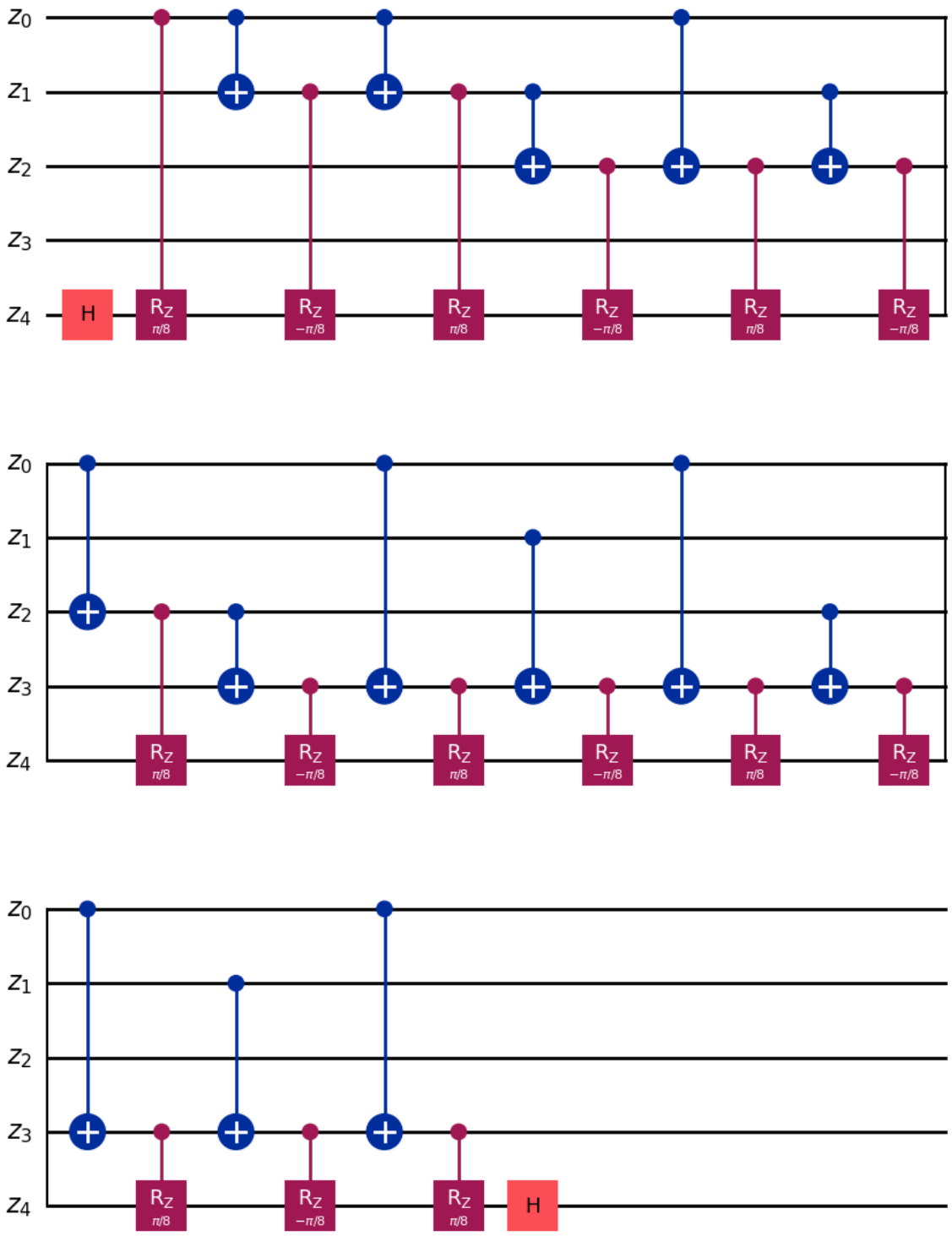

Figure 17: The decomposition of the quadruple CNOT gate.

3 References

[1] E. El-Qawasmeh, "Beating the popcount," vol. 9, 01 2003.

[2] R. Hamming, "Chapter 4 - classical error-correcting codes: Machines should work. people should think." in *Classical and Quantum Information*, D. C. Marinescu and G. M. Marinescu, Eds. Boston: Academic Press, 2012, pp. 345–454. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123838742000047

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[4] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," Mar. 1995, arXiv:quant-ph/9503016. [Online]. Available: http://arxiv.org/abs/quant-ph/9503016

[5] G. Song and A. Klappenecker, "The simplified toffoli gate implementation by margolus is optimal," 2003. [Online]. Available: https://arxiv.org/abs/quant-ph/0312225

[6] P. Downing, "Qiskit Quantum Challenge - ICPC 2021," Apr. 2021. [Online]. Available: https://padraignix.github.io/quantum-computing/2021/04/06/ibm-quantum-challenge-icpc2021/

[7] MIT, "Problem Set #4 Solution," Sep. 2010. [Online]. Available: https://web.mit.edu/2.111/www/2010/ps4solution.pdf

[8] R. S. Sutor, *Dancing with Qubits: How Quantum computing works and how it cna change teh world*. Birmingham: Packt Publishing, 2019.

[9] Y. Liu, K. Baba, K. Kaneko, N. Takeda, J. Koyama, and K. Kimura, "Analysis of Parameterized Quantum Circuits: on The Connection Between Expressibility and Types of Quantum Gates," Aug. 2024, arXiv:2408.01036 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2408.01036

[10] S. Lee, S.-J. Lee, T. Kim, J.-S. Lee, J. Biamonte, and M. Perkowski, "The Cost of Quantum Gate Primitives," 2006.

4 Appendix

4.0.1 Appendix A

For running the simulatin with Qiskit a jupyter notebook was created. In this appendix the exported version of the notebook can be reviewed. The following libraries were imported:

[19]:
```
import qiskit
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
import numpy as np
import matplotlib.pyplot as plt
from qiskit_aer.primitives import Sampler
import copy
import pandas as pd
```

The spesific problem instance is defined:

[20]:
```
# problem definition
n = 6
```

Function to prepare the initial states of the circuit. It takes a numpy array (input_boolean) containing booleans representing the binary string that should be prepared:

[21]:
```
def prepare_input_state(input_boolean: np.ndarray, qreg:␣
↪QuantumRegister, qc: QuantumCircuit):
    for i, set_bit in enumerate(input_boolean):
        # For each bit in the bit string
        if set_bit:
            # If the bit is set (True), then apply the bit flip to the␣
↪corresponding qubit
            qc.x(qreg[i])
```

The following functions create the different circuits that solve the popcount j=2 problem for $n > 0, n \in \mathbb{Z}$.

[22]:
```
def create_QPE_popcount_circuit_rcx(qreg: QuantumRegister, qreg_result:␣
↪QuantumRegister, creg: ClassicalRegister, qc: QuantumCircuit, n:␣
↪int):
    # This function takes the quantum circuit as argument and applies␣
↪the QFT solution using the CR_x(pi/2)
    # Note that the hadamard gates are not used in this case
```

```
    for control_q in range(n):
        # Apply the crx gate with pi/2 as rotation angle (S gate) with␣
↪each input qubit as control bit and the y_2 qubit as target.
        qc.crx(np.pi/2, qreg[control_q], qreg_result)

    for control_q in range(n-1):
        # Add the CNOT operators for each input qubit as controll␣
↪except one which serves as target to get the y_1 value.
        qc.cx(qreg[control_q], qreg[n-1])

    # Add the inverse controlled operation once qwith y_1 as controll␣
↪and y_2 as target.
    qc.crx(-np.pi/2, qreg[n-1], qreg_result)
    # The y_2 qubit is measured
    qc.measure(qreg_result, creg)

def create_QPE_popcount_circuit_rcz(qreg: QuantumRegister, qreg_result:␣
↪QuantumRegister, creg: ClassicalRegister, qc: QuantumCircuit, n:␣
↪int):
    # This function takes the quantum circuit as argument and applies␣
↪the QFT solution using the decomposition of CR_z(pi/2)
    for control_q in range(n):
        # Apply the crz gate with pi/2 as rotation angle (S gate) with␣
↪each input qubit as control bit and the y_2 qubit as target.
        qc.rz(np.pi/4, qreg_result)
        qc.cx(qreg[control_q], qreg_result)
        qc.rz(-np.pi/4, qreg_result)
        qc.cx(qreg[control_q], qreg_result)


    for control_q in range(n-1):
        # Add the CNOT operators for each input qubit as controll␣
↪except one which serves as target to get the y_1 value.
        qc.cx(qreg[control_q], qreg[n-1])

    # Add the inverse controlled operation once qwith y_1 as controll␣
↪and y_2 as target.
    qc.rz(-np.pi/4, qreg_result)
    qc.cx(qreg[n-1], qreg_result)
```

```
    qc.rz(np.pi/4, qreg_result)
    qc.cx(qreg[n-1], qreg_result)
    # The y_2 qubit is measured
    qc.measure(qreg_result, creg)
```

Instantiate the registers and circuit objects and prepare the circuit. Note that the qubits are divided into two registers: One for the n inputs and one for the single qubit output

[29]:
```
qreg = QuantumRegister(n, name="z")
qreg_result = QuantumRegister(1, name="y_2")
creg = ClassicalRegister(1, name="Result")
qc = QuantumCircuit(qreg, qreg_result, creg, name="QFT popcount␣
 ↪solution with crz decomposition")

create_QPE_popcount_circuit_rcz(qreg, qreg_result, creg, qc, n)
qc.draw("mpl")
```

See figure 14 for the output image from this cell.

Next, all different 2^n input combinations are generated an stored as numpy array

[ ]:
```
# The following lines generates all 2**n input combinations
input_states = np.empty((2**n, n), dtype=bool)
for i in range(2**n):
    input_states[i, :] = np.array(list(bin(i)[2:].zfill(n)), dtype=int).
 ↪astype(bool)
```

This cell verifies that there are no redundant inputs in the list

[7]:
```
test_list = []
for input_state in input_states:
    test_list.append(int("".join(input_state.astype(int).astype(str)),␣
 ↪2))

cur_length = len(test_list)
actual_unique_lenght = len(set(test_list))
if not (cur_length == actual_unique_lenght):
    print("There is a mistake in the generator function")
```

Finally the circuit is tested by running the simulation for every possible input combination. The results are stored in a pandas dataframe and saved to a csv file in the current working directory with filename "popcount_QPE_analysis_n{n}.csv" ({n} depends on n)

[10]:
```
results = {"Input_dec" : [], "Input_bin" : [], "Hamming_weight" : [],␣
↪"Expected_output" : [], "Measured_result" : []}
for input_state in input_states:
    cur_input_decimal = int("".join(input_state.astype(int).
↪astype(str)), 2)
    cur_hamming_weight = np.sum(input_state.astype(int))
    # Here the output bit is determined by checking if the hamming␣
↪weight value has the second bit set to 1
    expected_output = int(bin(np.sum(input_state.astype(int)))[2:].
↪zfill(n)[-2])

    cur_qreg = QuantumRegister(n+1, name="qreg")
    cur_creg = ClassicalRegister(1, name="creg")
    cur_qc = QuantumCircuit(qreg, creg)

    prepare_input_state(input_state, cur_qreg, cur_qc)

    create_QPE_popcount_circuit(cur_qreg, cur_creg, cur_qc, n)
    result = Sampler().run([cur_qc], shots=500).result()

    results["Input_dec"].append(cur_input_decimal)
    results["Input_bin"].append("".join(input_state.astype(int).
↪astype(str)))
    results["Hamming_weight"].append(cur_hamming_weight)
    results["Expected_output"].append(expected_output)
    measured_keys = result.quasi_dists[0].keys()
    if len(measured_keys) > 1:
        results["Measured_result"].append(1/2)
    else:
        results["Measured_result"].append(list(measured_keys)[0])

results_df = pd.DataFrame(results)
results_df.to_csv(f"popcount_QPE_analysis_n{n}.csv")
results_df
```

[10]:
```
    Input_dec Input_bin  Hamming_weight  Expected_output ␣
↪Measured_result
0           0    000000               0                0
↪0
```

```
1             1    000001                1                 0                ␣
 ↪0
2             2    000010                1                 0                ␣
 ↪0
3             3    000011                2                 1                ␣
 ↪1
4             4    000100                1                 0                ␣
 ↪0
..          ...       ...              ...               ...               ␣
 ↪...
59           59    111011                5                 0                ␣
 ↪0
60           60    111100                4                 0                ␣
 ↪0
61           61    111101                5                 0                ␣
 ↪0
62           62    111110                5                 0                ␣
 ↪0
63           63    111111                6                 1                ␣
 ↪1

[64 rows x 5 columns]
```

[12]:
```
results_df.groupby(["Hamming_weight"])[["Expected_output",␣
 ↪"Measured_result"]].mean()
```

[12]:
```
                Expected_output  Measured_result
Hamming_weight
0                           0.0              0.0
1                           0.0              0.0
2                           1.0              1.0
3                           1.0              1.0
4                           0.0              0.0
5                           0.0              0.0
6                           1.0              1.0
```

[12]:
```
results_df.groupby(["Hamming_weight"])[["Expected_output",␣
 ↪"Measured_result"]].mean()
```

4.0.2 Appendix B

Code for the compact margolis solution8:

[13]:
```
def create_QPE_popcount_circuit(qreg: QuantumRegister, creg:␣
↪ClassicalRegister, qc: QuantumCircuit, n: int):
    qc.ry(np.pi/4,qreg[3])
    qc.cx(qreg[1],qreg[3])
    qc.cx(qreg[2],qreg[3])
    qc.ry(np.pi/4,qreg[3])
    qc.cx(qreg[0],qreg[3])
    qc.ry(-np.pi/4,qreg[3])
    qc.cx(qreg[1],qreg[3])
    qc.ry(np.pi/4,qreg[3])
    qc.cx(qreg[1],qreg[3])
    qc.ry(-np.pi/4,qreg[3])
    qc.cx(qreg[2],qreg[3])
    qc.ry(-np.pi/4,qreg[3])
    qc.measure(qreg[n], creg)
```

Results for the compact margolis solution:

[13]:

Hamming_weight	Expected_output	Measured_result
0	0.0	0.0
1	0.0	0.0
2	1.0	1.0
3	1.0	1.0