THE UNIVERSITY
of EDINBURGH

Parallel Quantum Computing From Theory to Practice

Einar Pius

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy
to the
University of Edinburgh

August 2014

Abstract

The term quantum parallelism is commonly used to refer to a property of quantum computations where an algorithm can act simultaneously on a superposition of states. However, this is not the only aspect of parallelism in quantum computing. Analogously to the classical computing model, every algorithm consists of elementary quantum operations and the application of them could be parallelised itself. This kind of parallelism is explored in this thesis in the *one way quantum computing* (1WQC) and the *quantum circuit* model.

In the quantum circuit model we explore arithmetic circuits and circuit complexity theory. Two new arithmetic circuits for quantum computers are introduced in this work: an adder and a multiply-adder. The latter is especially interesting because its depth (*i.e.* the number of parallel steps required to finish the computation) is smaller than for any known classical circuit when applied sequentially. From the complexity theoretical perspective we concentrate on the classes QAC^0 and $\mathsf{QAC}^0[2]$, the quantum counterparts of AC^0 and $\mathsf{AC}^0[2]$. The class AC^0 comprises of constant depth circuits with unbounded fan-in AND and OR gates and $\mathsf{AC}^0[2]$ is obtained when unbounded fan-in parity gates are added to AC^0 circuits. We prove that QAC^0 circuits with two layers of multi-qubit gates cannot compute parity exactly. This is a step towards proving $\mathsf{QAC}^0 \neq \mathsf{QAC}^0[2]$, a relation known to hold for AC^0 and $\mathsf{AC}^0[2]$.

In 1WQC, computation is done through measurements on an entangled state called the resource state. Two well known parallelisation methods exist in this model: *signal shifting* and *finding the maximally delayed general flow*. The first one uses the measurement calculus formalism to rewrite the dependencies of an existing computation, whereas the second technique exploits the geometry of the resource state to find the optimal ordering of measurements. We prove that the aforementioned methods result in same depth computations when the input and output sizes are equal. Through showing this equivalence we reveal new properties of 1WQC computations and design a new algorithm for the above mentioned parallelisations.

Declaration

Except where otherwise stated, the research undertaken in this thesis was the unaided work of the author. Where the work was done in collaboration with others, a significant contribution was made by the author.

Parts of this work have been published in the Journal of Quantum Information Processing [1] and other parts submitted to the Journal of Quantum Information and Computation. An earlier archived version of the latter can be found at [2].

E. Pius

August 2014

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisors Elham Kashefi, Chris Johnson, and Chris Maynard. I thank Elham for getting me interested in quantum computing in the first place, and for asking the right questions during the PhD. She guided me through my PhD while being both critical and encouraging at the same time and I am grateful for that. I also thank Chris Maynard, who is responsible for me pursuing a topic that eventually led me to the quantum arithmetic circuits. And many thanks to Chris Johnson who took up my supervision from Chris Maynard midway through my PhD and has taken care of me since then.

I would like to thank Alain Tapp who kindly accepted me as a visiting student at the University of Montreal in autumn 2012. The discussions I had with Alain ultimately led to the two results in the final part of the thesis. I would like to thank my colleagues Vedran Dunjko, Raphael Dias da Silva, and Theodoros Kapourniotis for the good times we had together and for the discussions we had.

This PhD was made possible through the funding from the Archimedes Foundation in Estonia and for that I am very grateful.

Finally, but by no means least, I am grateful to my lovely girlfriend Riinu, who not only supported me while I was spending long hours in the evenings working on the thesis, but also helped me proofread my work. I could not have done it without her.

Contents

List of Figures

List of Tables

Introduction

The quantum computing model has two concepts of parallelism. First, quantum computations can act on a superposition of states, modifying all of them simultaneously. This effect has for example been used in the famous Shor's [2] and Grover's [3] algorithms and does not exist in classical computers. Second, the parallelism achieved through the application of multiple quantum gates simultaneously. This corresponds to the classical approach of parallelisation and has been studied in both the circuit [4, 5, 6, 7, 8] and one way quantum computing (1WQC) model [9, 10, 11]. The latter kind of parallelism, which is the focus of this work, is important in at least two aspects. First, parallel computations could be executed faster (that is the main motivation for parallelising), thus reducing the time that quantum states need to be coherent for. Second, the quantum model could allow the implementation of some algorithms in a more parallel manner than is possible classically, thus computations could possibly run faster on parallel quantum computers. This has motivated the search for new parallel quantum algorithms [12] and parallelisation methods [9, 10, 11]. The problem addressed in this work is twofold: do there exist any arithmetic operations benefitting from quantum parallelism, and how much can we expect to parallelise computations in the quantum model?

The first part of this thesis presents a new quantum arithmetic circuit, which is more parallel than any of its known classical counterparts. The later chapters try to establish limits on parallelism in quantum computing by first clarifying that two common methods in 1WQC model produce equivalent results (Part II) and proving a lower bound on the parallel circuits computing parity (Part III).

This work introduces two new quantum Fourier transform (QFT) based arithmetic circuits. The first of these circuits, the QFT multiply-accumulator (QMAC), is introduced in Chapter 2 and has been published in the Journal of Quantum Information processing [1]. A multiply-accumulator (MAC) is a circuit performing the operation $z = z+xy$ on numbers and is an important operation in modern digital signal processors;

thus accelerating this operation has significant practical uses. Our QMAC is the first published quantum MAC circuit and the first basic quantum arithmetic circuit that exhibits a lower depth than any of its classical counterparts. The defining feature of the QMAC is its depth on sequential application, which is asymptotically smaller than in any previously known classical or quantum MAC. This property makes the QMAC not only a suitable candidate for a MAC unit in future quantum processors, but it could be beneficial to implement it as an accelerator or co-processor in classical computers. The second arithmetic circuit, the QFT adder presented in Chapter 3, has not yet been published. It was initially left out of the QMAC paper [1] since it does not exhibit an asymptotically lower depth than its classical counterparts, as the QMAC does. However, later analysis showed that in sequential application its implementation could have as much as 24 times smaller depth than any other quantum adder. Therefore, possible future developments of quantum processors should consider the use of the QFT adder.

In August 2013, the QMAC was presented to the physicists at the Centre for Quantum Photonics in the University of Bristol. There was interest in the QMAC and questions about simplifying the circuit for implementation with the technology available to their group. Their interest and questions motivated the inclusion of Chapter 4, which contains modifications that could simplify the experimental realisation of a QMAC. One request from the Bristol group was the simplified circuit schema of a QMAC with an explicit proof that it would perform the multiply-accumulation of a two-bit integer. This circuit is included towards the end of Chapter 4 and could be used as a starting point for experimentalists wishing to implement the QMAC.

The second part of the thesis focuses on parallelisation methods in the one way quantum computing (1WQC) model. Computations in the 1WQC model are performed by doing measurements on entangled qubits. These measurement outcomes are in general probabilistic, but can be used to correct subsequent measurements to obtain computations performing unitary operations [13]. The main result of Part II states that two well known parallelisation techniques in 1WQC result in equal depth when applied to computations translated from quantum circuits. These two methods work on distinct representations of measurement based computation: one on the *measurement patterns* [14] and the other on the *flows* [15] and *general flows* [16] of the underlying graph. The description of flows and of the 1WQC is included in Chapter 6, signal shifting is covered separately in Chapter 7. First of those methods, *signal shifting* [14], comprises of a set of rewriting rules for measurement patterns. The second technique, *finding the maximally delayed general flow* [9], uses only the structure of the graph representation of 1WQC to provide a low depth dependency

structure for the measurements. Through the construction of the proof, many new properties of 1WQC which could be used in future research of flows and signal shifting have been discovered. One of the main techniques from the proof has already been successfully applied to construct a translation method from 1WQC to quantum circuits which does not increase the number of qubits in the computation [17]. In the course of constructing the main proof, a new algorithm for signal shifting and finding maximally delayed general flows is created. This new algorithm works on computations derived from quantum circuits and requires $O(n^2)$ operations to complete. This is smaller than the operations required for the best previously known algorithms for signal shifting and finding the maximally delayed general flow, $O(n^6)$ [14] and $O(n^2)$ [9] correspondingly. The work in Part II is available on arXiv.org [18] and has been submitted to the Journal of Quantum Information and Computation.

The final part of this thesis focuses on quantum circuit complexity. Proving lower bounds in complexity theory for general circuits is very hard, hence numerous circuit families with various restrictions have been created. One of those restrictions limits the depth of the circuits. Depth restricted circuits can also be seen as parallel circuits, since the number of gates and bits/qubits allowed is polynomial in input, *i.e.* a larger number of resources can be used to simultaneously perform a computation in fewer sequential steps. The only lower bound known in quantum parallel complexity is between the classes QNC^0 and QAC^0. The class QNC^0 consists of constant depth quantum circuits containing only quantum gates that act on a constant number of qubits. The class QAC^0 is the quantum equivalent to the classical AC^0 class: constant depth Boolean circuits with unbounded fan-in AND, OR and NOT gates. Unbounded fan-in means that the gates can have an arbitrary number of inputs. The inequality $\mathsf{QNC}^0 \neq \mathsf{QAC}^0$ holds because circuits in the QNC^0 class cannot compute functions that depend on all the input qubits [6].

Although not many lower bounds have been found, research in quantum circuit complexity has revealed surprising differences in the relations between quantum circuit classes and their classical counterparts. First, the quantum circuit classes $\mathsf{QAC}^0[n]$ and $\mathsf{QAC}^0[m]$ have been proven to be equal for every m and n [5], whereas in classical circuit complexity $\mathsf{AC}^0[p] \neq \mathsf{AC}^0[q]$ for distinct primes p and q [19, 20]. $\mathsf{QAC}^0[m]$ is the quantum counterpart to the $\mathsf{AC}^0[m]$ class, which is the class AC^0 with the addition of Mod m gates acting on any number of inputs. Second, the relation $\mathsf{AC}^0[p] \subset \mathsf{TC}^0$ [21] (where p is a prime) between the classical classes does not hold for their quantum counterparts, where $\mathsf{QTC}^0 \subseteq \mathsf{QAC}^0[m]$ (for every m) [7, 22]. Here TC^0 notes constant depth threshold circuits, *i.e.* circuits with unbounded threshold gates and QTC^0 is its quantum counterpart where quantum threshold gates are used. Since in [7] it was

also established that $\mathsf{TC}^0 \subseteq \mathsf{QAC}[m]^0$, these results show that the quantum parallel complexity classes $\mathsf{QAC}^0[m]$ are strictly more powerful than their classical counterparts.

We present two new results in low depth quantum circuit complexity theory. These two results are a step towards proving the inequality of two quantum circuit classes: QAC^0 and $\mathsf{QAC}^0[2]$. Admittedly, the initial goal was to prove this inequality in this thesis, but this was not possible under the time constraints imposed by this PhD. The relation $\mathsf{AC}^0 \neq \mathsf{AC}^0[2]$ is known to hold for classical circuits [23]. The approach in proving that $\mathsf{QAC}^0 \neq \mathsf{QAC}^0[2]$ is to show that computing the parity of input qubits, possible in $\mathsf{QAC}^0[2]$, cannot be done in QAC^0. It has already been shown that this is impossible exactly and cleanly if the number of auxiliary qubits in the circuit is less than the input size [8]. However, before this work nothing was known for the case when the number of auxiliary qubits is polynomial in the input size. First, we prove that QAC circuits with one multi-qubit gate layer cannot compute parity even probabilistically. Second, we show that when the number of multi-qubit gate layers is two, parity cannot be computed exactly and cleanly. During these two proofs, a number of properties for QAC circuits are discovered. These could be useful in proving the inequality of the QAC^0 and $\mathsf{QAC}^0[2]$ classes.

There exist computational problems that exhibit more parallelism in the quantum than classical computing mode. This thesis expands the number of problems benefiting from quantum computers by introducing a new parallel quantum arithmetic circuit (Part I), while establishing boundaries through proving the equivalence of two parallelisation methods (Part II) and giving a new lower bound of the parity function (Part III).

Part I

Quantum Arithmetic Circuits

Quantum computing has the potential to dramatically change the nature of computing, but has mostly been a theoretical subject partly due to the difficulties in building physical quantum circuits. However, recent progress has enabled the first, albeit small, quantum devices to be constructed, for example utilising photonics [24]. These devices are not complete quantum computers, but consist of simple quantum circuits capable of processing information to solve specific problems. These devices can be made in silicon [24] which could lead to their integration with conventional microelectronics. How would such a hybrid of conventional and quantum microprocessor be used? Co-processor architectures have been developed in the past but perhaps the most promising context would be to consider the quantum device as an accelerator.

There are several examples of modern heterogeneous computer architectures. For example, Graphical Processing Units (GPUs) have been used extensively in the field of scientific numerical computing to accelerate specific aspects of these calculations, where some suitably defined kernel, *i.e.* the core of the computation, is offloaded from the CPU and executed faster on the GPU. Another analogy can be drawn with field programmable gate arrays (FPGAs) where particular computational patterns in software can be instantiated in hardware using the reprogrammable logic of these devices, see for example [25, 26]. Rather than accelerating an entire kernel as would be required for a GPU, a quantum device could be employed to accelerate a specific computational pattern. Moreover, as this device would function as an accelerator, a complete quantum computer would not be required. Furthermore, the effects of quantum decoherence which destroys quantum information can be mitigated because such quantum circuits need only to be in an entangled state for a brief period compared to a full quantum computer.

The main result of this chapter is a quantum multiply-adder (QMAC) circuit, which could potentially be implemented as a quantum accelerator for classical computers. It is the first quantum multiply-adder design and, more importantly, the first quantum arithmetic circuit that has a smaller depth than its classical counterparts. This is achieved through the use of the Quantum Fourier Transform (QFT) and entangled quantum states combined with the ability to easily copy classical bits. This new quantum-classical hybrid circuit is presented in Chapter 2. A MAC is an important hardware module, *i.e.* electronic sub-circuit, in digital signal processors (DSPs), which are used, for example, in audio and video processing, encryption, pattern recognition, *etc* [27]. Since DSPs have a very wide area of application, improving the performance of MACs would be immensely useful. There exist two types of DSPs: fixed point and floating point DSPs. The integer QMAC introduced in this work can be adapted for fixed point arithmetics and could thus be used instead of classical MAC circuits in fixed

point DSPs.

This part starts with a brief introduction to classical and quantum arithmetic circuits in Chapter 1 and contains some basic concepts and definitions referred to throughout this thesis (not just from Part I). After the introduction of the QMAC circuit in Chapter 2, the same techniques are applied in Chapter 3 to create a parallel quantum adder, corresponding to a highly parallel Draper adder [28]. Chapter 4 contains implementation optimisations applicable to both the QMAC and the new adder, followed by the discussion about the results and impact of the new arithmetic circuits in Chapter 5.

Chapter 1

Preliminaries

1.1 Classical Circuits

The classical arithmetic circuits in this work are represented as *Boolean circuits*.

Definition 1.1 (Boolean circuit [29])**.** *A Boolean circuit is a directed graph with a set of source nodes called the inputs, and one or more sink nodes called the outputs. Each internal node, or "gate," is labelled AND, OR, NOT, and produces the corresponding function of its inputs. This graph is acyclic, meaning that there are no loops - information flows in one direction from the inputs to the outputs.*

In our work we also allow the use of the XOR gate in the Boolean circuits, since the gate set consisting of consisting of NOT, AND, OR, and XOR is the usual gate set used in the literature on arithmetic circuits. We use $a \cdot b$ for representing the AND of two bits, $a + b$ for the OR, $a \oplus b$ for the XOR, and $\neg$ for the NOT. The XOR an be replaced with two AND, one OR, and one NOT gate (Equation 1.1), thus the number of gates in our circuits is at most four times smaller than when using a gate set without the XOR.

$$a \oplus b = (a + b) \cdot \neg(a \cdot b) \tag{1.1}$$

Some estimates on the number of gates used in circuits require rounding of values. We use $\lceil n \rceil$ to denote the value of n rounded up to the nearest integer and $\lfloor n \rfloor$ the value of n rounded down to the nearest integer. The *fan-in* of a gate is its number of inputs [30]. Logic gates usually have a constant fan-in, *i.e.* they act on a fixed (usually small) number of bits. This number is often, as in this thesis, chosen to be 2. The *fan-out* of a circuit is the number of outputs a gate has [30]. This value is usually considered to

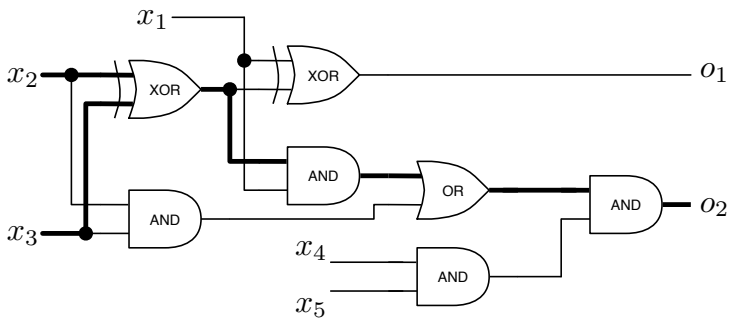


Figure 1.1: An example illustrating the concept of depth in Boolean circuits. The longest directed paths from inputs to outputs is highlighted in bold. In this example, there exist two longest paths (from x_2 to o_2 and from x_3 to o_2). The length of the path is the number of gates it passes through. In this circuit the longest path, and thus the depth, of this circuit is 4.

be unbounded, *i.e.* there can be any number of outputs. The main parameter we use in comparing parallelism in classical and quantum circuits is the *depth* of the circuit.

Definition 1.2 (The depth of a Boolean circuit [31])**.** *The depth of a Boolean circuit is the longest directed path from an input to the output.*

An example illustrating the depth of a Boolean circuit is in Figure 1.1. Each circuit can only process inputs of a specific size, but often it is useful to estimate how the circuits' parameters like number of gates or depth depend on the input size. This can be done by using *circuit families*.

Definition 1.3 (Circuit families [31])**.** *Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A $T(n)$-size (depth) circuit family is a sequence $\{C_n\}_{n\in\mathbb{N}}$ of Boolean circuits, where C_n has n inputs and a single output, and its (depth) size is $|C_n| \leq T(n)$ for every n.*

Throughout this thesis, if it is mentioned that there exists a $O(n)$ depth (or size, etc.) circuit solving a problem, it is implicitly meant that there exists a family of $O(n)$ depth (or size, etc.) circuits.

Obviously the time it takes to execute any circuit depends very much on the implementation but it is proportional to the depth of the circuit. In addition to the number of gates on the longest path, there are many other parameters that contribute to the execution time of a digital circuit: wiring, fan-out, *etc.* The depth is thus not the most accurate measure of performance, but it is an implementation independent way to compare the performance of various circuits. This is especially desirable when comparing classical circuits with quantum circuits, since there has not yet emerged a quantum architecture from where the execution times of different components could be taken. The classical arithmetic circuits presented here are often represented as digital circuits, where the term *delay* [30] is used instead of depth to represent the performance

of a circuit. There, each type of gate is assigned a fixed gate delay and the overall delay of the circuit is the longest delay from inputs to outputs. This work uses the simpler notion of depth, because it is the concept used in circuit complexity theory, which is the focus of Part III.

1.1.1 Numerical Representation

Although the arithmetic circuits presented in this thesis are shown to work for unsigned integers, they can be generalised to signed fixed point numbers. Unsigned integers are used since this simplifies the notation. The adaption to use signed fixed point numbers does not require any changes to the circuits as will be explained in the following sections. The concepts following can be found in most of the textbooks on digital signal processing or on computer arithmetic circuits [27, 30, 32, 33].

Fixed Point Representation

There are two main representations for real numbers in digital circits: the *floating point* and the *fixed point* representations [27, 30, 32, 33]. This thesis uses the fixed point representation since it is not clear if and how the new arithmetic circuits designed in the later chapters can be adapted to floating point representation. Although the floating point representation has superior precision and ease of use when programming, the fixed point system is still widely used. The main advantages of fixed point arithmetic circuits are its simple and cost efficient implementation and performance compared to the floating point systems. When large value ranges are not needed, fixed point representation is often suitable and provides a speed-up over floating points [30]. Thus, many Digital Signal Processors (DSPs) still use fixed point systems (for example the Texas Instruments TMS320C64xx [34] and the Analog Devices Blackfin [35] DSP families).

In this work the binary representation of integers is used. In a conventional number system every k-digit number x can be written as [30, 32]

$$\sum_{i=1}^{k} w_i x_i, \tag{1.2}$$

where x_i is the i-th digit of x and w_i is the weight associated with the digit. In the binary system w_i is always a power of two and $x_i \in \{0, 1\}$. Generally, a number can have both an integer and a fractional part. In the fixed point system this can be

achieved by interpreting the m rightmost digits as the integer part and $k-m$ leftmost digits as the fractional part. These two parts are separated by the radix point (.). For example in the number $z_9z_8z_7z_6z_5.z_4z_3z_2z_1$ the 4 rightmost digits comprise the fractional part and the digits z_5 to z_9 make up the integer part. In general the fixed point representation of a k digit number z with an m-digit fractional part is written as $z_k \cdots z_{m-2}z_{m-1}.z_m \cdots z_2z_1$. The weights of the bits in the integer part are always non-negative powers of two and the weights of the fractional parts are negative powers of two. The value of a k-bit fixed point number with m bit fractional part is therefore

$$\sum_{i=1}^{k} 2^{i-1-m} x_i. \tag{1.3}$$

As an example, the binary number $z_9z_8z_7z_6z_5.z_4z_3z_2z_1$ can be written as

$$z_9 2^4 + z_8 2^3 + z_7 2^2 + z_6 2^1 + z_5 2^0 + z_4 2^{-1} + z_3 2^{-2} + z_2 2^{-3} + z_1 2^{-4}. \tag{1.4}$$

The fixed point numbers are stored as integers, with the position of the radix point stored separately. To add or subtract two fixed point integers with the radix point at the same position, it is enough to add or subtract the underlying integers and keep the radix point at the same location. To multiply two fixed point integers with the radix point at the same position, the underlying integers can be multiplied together and the radix point will be positioned to twice as many digits from the right as it was before. This way it is also easy to obtain the new fixed point value with the radix point at the same position as the multiplicands: namely the last digits can be just discarded or not even computed. This allows us to use the arithmetic circuits presented in this thesis on integers without having to consider whether the integers represent fixed point numbers or not. The downside of using the fixed point representation is the loss of precision. Namely, when multiplying two fixed-point numbers, the result can have as many bits as the sum of the number of bits in the multiplicands. To fit the result in a fixed size register of the size of the inputs, some of the bits might have to be be discarded. In the worst case, half of the bits need to be discarded resulting in a significant loss of precision.

Two's Complement Representation

It is possible to perform arithmetic operations on signed integers (and thus also on signed fixed point values as explained in the previous section) as if they were all positive and just interpret the numbers as having a sign. This greatly simplifies the

implementation of digital arithmetic circuits by eliminating the need for considering signs independently. One way of achieving this is to use the two's complement representation, which is also the most common representation for signed integers in modern computers. In two's complement representation, the negative number is represented by [30]:

1. taking the binary representation of its positive counterpart,
2. flipping the bits of the positive counterpart,
3. adding one to the result.

For example the two's complement representation of a six bit number -13:

1. the binary representation of 13 is 001101,
2. flipping these bits results in 110010,
3. and finally, adding 1 to it gives the two's complement representation of -13: 110011.

The sign of integers in the two's complement representation is determined by the leftmost bit: 0 for unsigned and 1 for signed. Adding negative numbers in this representation does not require any overhead and subtraction can be performed by adding the two's complement of the positive number. An example showing how adding a negative number results in a correct answer is shown in Figure 1.2. Multiplication

1	1	0	1	1	0		(carry)
	0	1	0	1	1	0	(22)
+	1	1	0	0	1	1	(-13)
	0	0	1	0	0	1	(9)

Figure 1.2: Adding a negative number in the two's complement representation using columnar addition. The values in the carry row correspond to the carry from the added bits in the previous column, *i.e.* the carry bit is 0 if the sum of the previous column above the line is 01 or 00, and 1 if the sum is 10 or 11.

works similarly, with the additional constraint that the result must fit in the number of bits available, but this needs to be considered even when regular binary representation is used. The intermediate results need only to hold as many bits as is in the result, the rest can be discarded. An example of how multiplication in two's complement notation works is presented in Figure 1.3 as noted by d in the example below:

Thus addition and multiplication of signed two's complement numbers can be done using exactly the same method as for unsigned integers. Therefore, although unsigned

	0	0	0	0	1	1	(3)
×	1	1	1	0	1	1	(-5)
	0	0	0	0	1	1	
	0	0	0	1	1	0	
	0	0	0	0	0	0	
	0	1	1	0	0	0	
	1	1	0	0	0	0	
d	1	0	0	0	0	0	
d	1	1	0	0	0	1	(-15)

Figure 1.3: Multiplying two integers in the two's complement representation. The multiplication is done in two stages. First, the partial products are found by multiplying each bit in the multiplier (-5) with the multiplicand (3). In the second stage, the partial products are summed together. Here the result should be a 6-bit number, hence the 7th (leftmost) bit is discarded.

integers are used in this thesis, the circuits would work for signed numbers in two's complement representation.

1.2 Quantum Circuits

We assume that the reader of this thesis is familiar with the basics of quantum computing. A good overview can be found in the textbooks of Nielsen and Chuang [36], and Kaye, Laflamme and Mosca [37]. Nevertheless, we provide a very brief introduction to quantum circuits and highlight some of the definitions, concepts, terms and techniques most often used in this thesis. We also give the four postulates of quantum mechanics which define the underlying the mathematical framework required for this thesis.

The analogue of the classical bit in quantum computing is the *qubit*. A qubit corresponds to a two-dimensional quantum mechanical system.

Postulate 1.1 (State Space Postulate [37])**.** *The state of a quantum system is described by a unit vector in a Hilbert space $\mathcal{H}$.*

We can choose an orthonormal basis in the two-dimensional Hilbert space and label the basis vectors as $|0\rangle$ and $|1\rangle$. Then the general state of a qubit is:

$$\alpha|0\rangle + \beta|1\rangle, \tag{1.5}$$

where α and β are complex coefficients and $|\alpha|^2 + |\beta|^2 = 1$. The $\{|0\rangle, |1\rangle\}$ basis for the state of a qubit is called the *computational basis*. Physical systems can be combined

to form a larger *composite system*, the following postulate explains how these systems can be described.

Postulate 1.2 (Composition of Systems Postulate [36])**.** *The state space of a composite physical system is the tensor product of the component physical systems. Moreover, if we have systems numbered from i through n and the system number i is prepared in the stat* $|\psi_i\rangle$*, then the joint state of the total system is* $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$*.*

As a shorthand the tensor product $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$ is commonly written as $|\psi_1\psi_2\rangle \cdots |\psi_n\rangle$, where $\psi_i \in \{0, 1\}$. An n-qubit state is thus a unit vector in the n-fold tensor product space $\mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_n$. The 2^n basis states of this space are the n-fold tensor products of the states $|0\rangle$ and $|1\rangle$. With these basis states, a n-qubit state $|\phi\rangle$ is a 2^n dimensional complex unit vector

$$|\phi\rangle = \sum_{i \in \{0,1\}^n} \alpha_i |i\rangle. \tag{1.6}$$

Not all multi-qubit states can be written as tensor products of their components, the ones that cannot be represented as products are *entangled states.*

Definition 1.4 (Entangled states [36])**.** *A multi-qubit state is* entangled *if it cannot be written as a product of its component states.*

A quantum system whose state is known exactly is said to be in a *pure state*, otherwise the system is said to be in a *mixed state*. A mixed state is described by a *density operator*:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|, \tag{1.7}$$

where p_i is the probability that the system is in the pure state $|\psi_i\rangle$. The density operator of a pure state $|\psi\rangle$ is defined as

$$\rho = |\psi\rangle\langle\psi|. \tag{1.8}$$

To be able to perform computations with qubits, we need to be able to change their state. The following postulate describes, how the state of qubits changes over time.

Postulate 1.3 (Evolution Postulate [37])**.** *The time-evolution of a closed quantum system is described by a unitary operator. That is, for any evolution of the closed system there exists a unitary operator U such that if the initial state of the system is*

$|\psi\rangle$, then after the evolution the state of the system will be

$$|\psi'\rangle = U|\psi\rangle. \tag{1.9}$$

Thus one way to change the state of a quantum state is to apply a unitary operator to it. Another operation that can change the state of qubits is the measurement, which also gives us the way to observe quantum systems.

Postulate 1.4 (Measurement Postulate [36])**.** *Quantum measurements are described by a collection $\{M_m\}$ of* measurement operators*. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may result in the experiment. If the state space of the system is $|\psi\rangle$ immediately before the measurement, then the probability that result m occurs is given by*

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle, \tag{1.10}$$

and the state of the system after the measurement is

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}. \tag{1.11}$$

The measurement operators satisfy the completeness equation*,*

$$\sum_m M_m^\dagger M_m = I. \tag{1.12}$$

An important measurement, used in this thesis, is the computational basis measurement, defined by measurement operators $M_0 = |0\rangle\langle 0|$, $M_1 = |1\rangle\langle 1|$. It is easy to see, that performing a computational basis measurement on a general quantum state $\alpha|0\rangle + \beta|1\rangle$ results in 0 with probability $|\alpha|^2$ and in 1 with probability $|\beta|^2$. All measurements in this work are computational basis measurements unless they are explicitly defined.

In the quantum circuit model the qubits are represented as horizontal *wires* and the unitary operators are represented as *gates* acting on a number of wires [37]. A generic quantum circuit can be seen in Figure 1.4. Unless explicitly stated, the circuits used in this thesis are only allowed to contain one and two qubit gates. Computations represented by quantum circuits are executed by applying quantum gates from left to right until all the gates have been applied. Due to the following principle, all the quantum wires at the end of the computation will be assumed to be measured in this

thesis.

Principle 1.1 (Principle of implicit measurement [36])**.** *Without loss of generality, any unterminated quantum wires (qubits which are not measured) at the end of a quantum circuit may be assumed to be measured.*

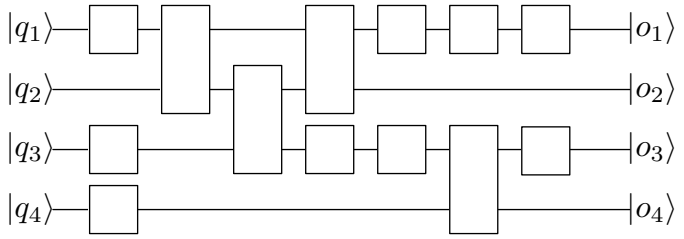


Figure 1.4: A generic quantum circuit. Each horizontal line represents a "wire". Each wire represents a qubit in the computation performed by the quantum circuit. $|q_1\rangle$, $|q_2\rangle$, $|q_3\rangle$, and $|q_4\rangle$ are the initial states of the qubits represented by the wires. $|o_1\rangle$, $|o_2\rangle$, $|o_3\rangle$, $|o_4\rangle$ are the output states of those qubits. The rectangles on the wires represent quantum gates. Gates can be applied to any number of wires.

Assume that applying a gate to the qubits in the circuit takes one discrete time step, and that gates acting on distinct qubits can be applied in parallel. Then we can divide quantum circuits to a number of layers, so that executing each layer takes exactly one time step. This allows us to define the central concept thesis: the *depth* of a quantum circuit. The definition used throughout this work is adapted from [4] by removing the restriction on using only one and two qubit gates, thereby allowing to compute the depth of circuits containing unbounded gates in Part III.

Definition 1.5 (Quantum circuit depth, adapted from [4])**.** *A* one layer circuit *is a unitary operator consisting of a tensor product of gates where each gate couples a disjoint set of gates. A* quantum circuit of depth d *is a unitary operator written as a product of d one layer circuits.*

An example of how the depth of a circuit can be found is shown in Figure 1.5. Parallel circuit complexity teory uses asymptotic complexity of the circuit depth, instead of execution time to compare algorithms. Rather than comparing exact execution times, the number of parallel steps required to finish the computation is used. This simplification allows to compare algorithms without the need to consider the underlying architecture (by architecture we mean the technology used to implement the quantum gates, *i.e.* photonics, ion traps, *etc.*). As long as the execution time of a quantum gate does not depend on the input size, the exact execution time will differ by at most a constant factor from the depth of the circuit. Note that although the asymptotic circuit depth does not depend on the architecture, it depends on the computational model, *i.e.* whether the computation is represented in the adiabatic, circuit, 1WQC, or any other

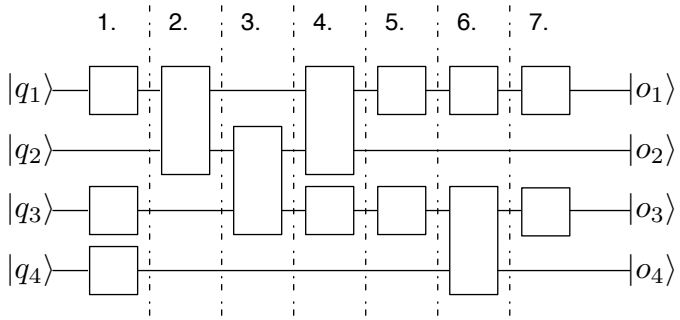


Figure 1.5: An example of how the depth of a quantum circuit can be calculated by dividing the circuit into layers. The depth of this particular circuit is 7.

model. This dependency on the model is due to translating an algorithm to a different models, which can change the depth of the computation. For example, it is known how to translate computations from circuit model to 1WQC without increasing the depth [10], but it is not know how to perform the opposite depth preserving translation in the general case (see Part II for more detailed description of depth in 1WQC).

Often, instead of computing the depth of a single circuit, a uniform family of quantum circuits is used to estimate a it function of the number of input qubits. The definition is very similar to the definition of Boolean circuit families in Definition 1.3.

Definition 1.6 (Quantum circuit families [6])**.** *A quantum circuit family is a sequence* $\{C_n\}_{n\in\mathbb{N}}$ *of quantum circuits, where each* C_n *has* n *inputs. We say that* $\{C_n\}$ *is uniform if there is a classical polynomial-time algorithm that outputs* C_n *on input* 0^n*.*

For example, if we say that a quantum circuit has logarithmic depth, then we mean that the number of discrete time steps it takes to evaluate the circuit increases logarithmically corresponding to the problem size, and the depth of the circuit is $O(\log n)$.

Implementing a specific quantum circuit exactly is not always possible with a finite set of gates. It not necessary in practice to compute a circuit exactly, it is enough to *approximate* it to some specific accuracy.

Definition 1.7 (Approximate unitary operators)**.** *A unitary operator* V approximates *another unitary operator* U *with with error* $E(U,V)$*, if*

$$E(U,V) = \max \|(U-V)|\psi\rangle\|. \tag{1.13}$$

An operator U *can be* approximated to arbitrary precision *if for every* $\epsilon > 0$ *there exists another unitary* V *such that* $E(U,V) < \epsilon$*.*

The gate name	The gate symbol and its unitary matrix	Symbol used in quantum circuits
The identity gate	$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	
The Pauli X gate	$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	X
The Pauli Y gate	$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	Y
The Pauli Z gate	$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	Z
The arbitrary phase rotation gate	$Z(\alpha) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$	$Z(\alpha)$
The phase rotation gate	$P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	P
The Hadamard gate	$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	H
The J gate	$J(\alpha) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{i\alpha} \\ 1 & -e^{i\alpha} \end{pmatrix}$	$J(\alpha)$
The square root of NOT gate	$V = \frac{1+i}{2} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$	V
The CNOT gate	$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	
The control-Z gate	$\wedge Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$	
The control-phase gate	$R_n = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^n}} \end{pmatrix}$	R_n

Figure 1.6: Some of the most common one and two-qubit quantum gates.

Every computation realisable by quantum circuits can be implemented approximately by using only a small set of different gates. We call such set a *universal* set of gates.

Definition 1.8 (Universal set of gates [37])**.** *A set of gates is said to be* universal*, if for any integer $n \geq 1$, any n-qubit unitary operator can be approximated to arbitrary accuracy by a quantum circuit using gates from that set.*

Some of the most common one and two-qubit quantum gates are presented in Figure 1.6. Most of the gates used in this work are present in this figure. The remaining gates are either introduced immediately before they are used or are defined in Section 1.2.1. The gates in Figure 1.6 can be used to create multiple distinct universal sets of gates, for example $\{Z(\frac{\pi}{8}), H, CNOT\}$ [36] and $\{J(\frac{\pi}{8}), H, \wedge Z\}$ [38].

Many of the circuits presented in this work are *hybrid circuits*, *i.e.* circuits which contain both classical and quantum bits and gates. Sometimes we also need to represent multiple qubits or bits compactly in the figure. The notation used to display all this information is shown in Figure 1.7.

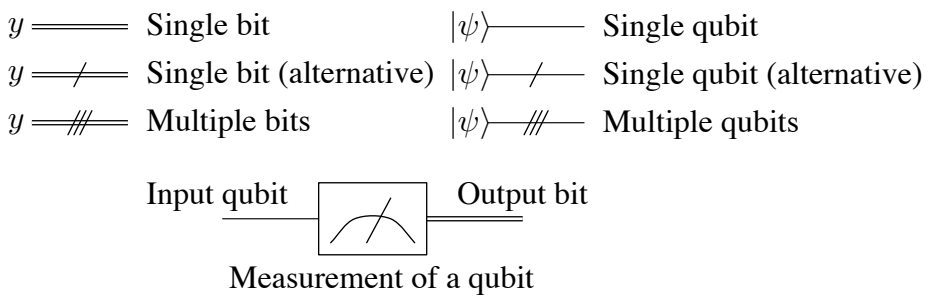


Figure 1.7: The notation used to distinguish between qubits and bits and multiple instances of them. Double lines for bits are only used when there is the possibility ambiguity in the figures, otherwise single lines are used for both bits and qubits. Likewise, the alternative single qubit lines with a single dash are used to avoid confusion over whether multiple qubits or a single qubit is meant.

Given a 1-qubit quantum gate U, the corresponding 2-qubit *controlled-U* gate, denoted as $\wedge U$, performs the following operation [36, 37]:

$$\wedge U|0\rangle|\psi\rangle = |0\rangle|\psi\rangle, \tag{1.14}$$

$$\wedge U|1\rangle|\psi\rangle = |1\rangle U|\psi\rangle. \tag{1.15}$$

The qubit that controls the application of U is called the *control qubit* and the qubit on which U acts is called the *target qubit*. It is possible to replace a controlled quantum gate at the end of a quantum circuit with a measurement of the control qubit in the computational basis followed by the application of U to the target qubit if and only if the measurement outcome is $|1\rangle$ [36, 37]. This is due to the principle of deferred measurement.

Principle 1.2 (Principle of deferred measurement [36])**.** *Measurements can always be moved from an intermediate stage of a quantum circuit to the end of the circuit. If the measurement results are used at any stage of the circuit then the classically controlled operations can be replaced by conditional quantum operations.*

The principle of deferred measurement is illustrated in Figure 1.8. This useful property of controlled gates is used in Chapter 4 to reduce the number of two-qubit gates in our quantum multiply adder circuit and in Chapter 12 to analyse the probabilities of the target qubit being modified.

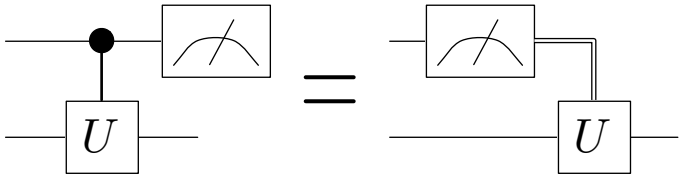


Figure 1.8: A controlled U gate at the end of a circuit can be replaced with a measurement and a single-qubit gate.

The following trivial lemma is included since it is often used in both Part I and III of this thesis.

Lemma 1.1. *Quantum gates commuting with the measurement of the output qubit can be removed from the circuit without affecting the measurement outcome of output qubit.*

Proof. Let U be any quantum gate commuting with the measurement of the output qubit o. Since U commutes with the measurement of o we can measure o and then apply U. Obviously the application of U cannot influence the already measured value, hence we can remove it from the circuit without affecting the measurement outcome on o. □

1.2.1 Unbounded Quantum Gates

In some instances gates that act on an unlimited number of qubits are used, these quantum gates are called *unbounded quantum gates.*

Definition 1.9 (Unbounded quantum gates)**.** *A quantum gate U is unbounded if it can act on unlimited number of qubits.*

Unbounded quantum gates are used in quantum circuit complexity (See Part III) where adding an unbounded gate to the set of gates allowed in a circuit can increase the number of solvable problems in a complexity class. The unbounded quantum gates are

also used in regular quantum circuits as a shorthand to denote their decomposition to two-qubit and single-qubit gates, *i.e.* instead of writing out the full decomposition the symbol of an unbounded gate is used. In what follows, the definitions of the unbounded quantum gates used throughout this thesis are given.

Definition 1.10 (The unbounded Toffoli gate [8])**.** *The unbounded Toffoli gate T is the unitary operator implementing the following map*

$$T|x_1, x_2, \ldots, x_n, b\rangle = |x_1, x_2, \ldots, x_n, b \oplus \prod_{i=1}^{n} x_i\rangle. \tag{1.16}$$

The symbol representing the unbounded Toffoli gates in quantum circuits is shown in Figure 1.11(a).

Definition 1.11 (The unbounded $\wedge Z$ gate [8])**.** *The unbounded $\wedge Z$ (controlled-Z) gate is the unitary operator implementing the following map*

$$\wedge Z|x_1, x_2, \ldots, x_n\rangle = (-1)^{\prod_{i=1}^{n} x_i}|x_1, x_2, \ldots, x_n\rangle. \tag{1.17}$$

The symbol representing the unbounded $\wedge Z$ gates in quantum circuits is shown in Figure 1.11(b).

The unbounded Toffoli gates can be turned into $\wedge Z$ gates by applying Hadamard gates to to the target bit of the Toffoli gate [8] as is shown in Figure 1.9.

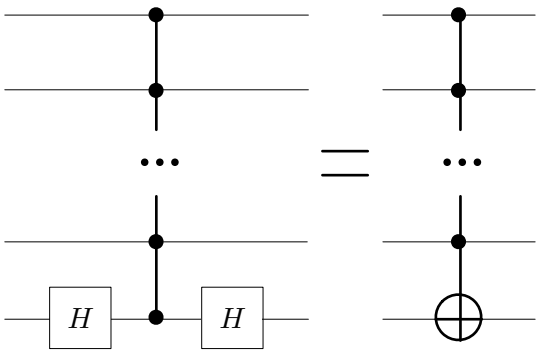


Figure 1.9: The $\wedge Z$ gate can be turned to a Toffoli gate with two Hadamard gates and vice versa [8].

Definition 1.12 (The unbounded fan-out gate [8])**.** *The unbounded fan-out gate F is the unitary operator implementing the following map*

$$F|b, x_1, x_2, \ldots, x_n\rangle = |b \oplus x_1, b \oplus x_2, \ldots, b \oplus x_n\rangle. \tag{1.18}$$

The symbol representing the unbounded fan-out gates in quantum circuits is shown in

Figure 1.11(c).

Definition 1.13 (The unbounded parity gate [8])**.** *The unbounded parity gate P is the unitary operator implementing the following map*

$$P|x_1, x_2, \ldots, x_n\rangle = |x_1, x_2, \ldots, x_n, b \oplus \bigoplus_{i=1}^{n} x_i\rangle. \tag{1.19}$$

The symbol representing the unbounded parity gates in quantum circuits is shown in Figure 1.11(d).

The unbounded parity gate can be turned into an unbounded fan-out gate via layers of Hadamard gates before and after the gate [5] as is shown in Figure 1.10.

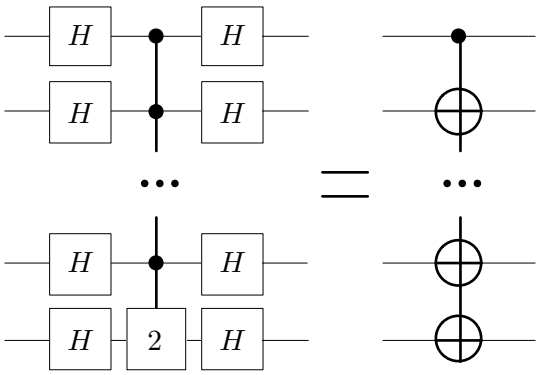


Figure 1.10: The parity gate can be turned to a fan-out gate with two Hadamard layers and vice versa [5].

The quantum MOD_q gate is a generalisation of the unbounded quantum parity gate.

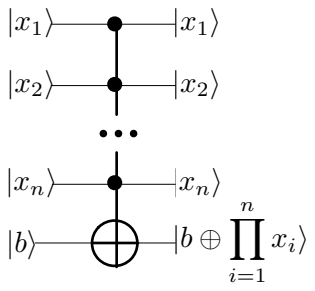
Definition 1.14 (The unbounded MOD_q gate [5])**.** *The unbounded MOD_q gate is the unitary operator implementing the following map*

$$MOD_q|x_1, x_2, \ldots, x_n\rangle = |x_1, x_2, \ldots, x_n, b \oplus Mod_q(x_1, x_2, \ldots, x_n)\rangle, \tag{1.20}$$

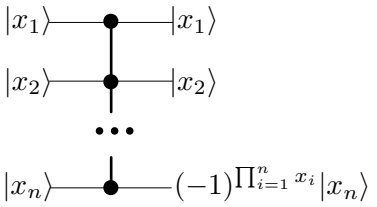
where $Mod_q(x_1, x_2, \ldots, x_n) = 1$ if and only if $\sum_{i=1}^{n} x_i \not\equiv 0 \bmod q$.

1.2.2 Translating Boolean circuits to quantum circuits

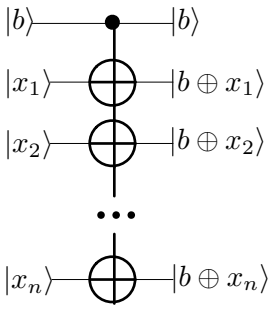
Boolean circuits can be translated to quantum circuits on a gate-by-gate basis. This can be done by replacing the Boolean gates with quantum gates, which on computational basis input will output a single qubit computational basis state corresponding to the



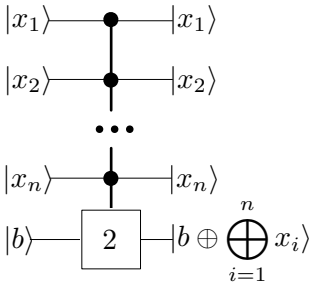
(a) The unbounded Toffoli gate



(b) The unbounded $\wedge Z$ gate. The phase change resulting by the application of the $\wedge Z$ gate can be written before any of the n qubits, since it is not associated with any particular qubit but with the state $|11\cdots1\rangle$.



(c) The unbounded fan-out gate



(d) The unbounded Parity gate

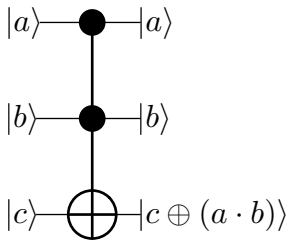
Figure 1.11: The four unbounded gates used in this thesis.

output of the Boolean gate. First, the NOT gate is translated to a Pauli X gate since:

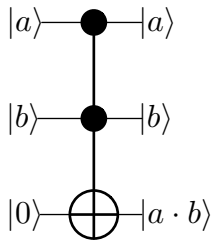
$$X|0\rangle = |1\rangle \tag{1.21}$$

$$X|1\rangle = |0\rangle \tag{1.22}$$

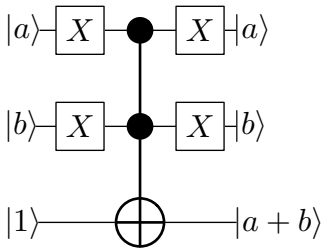
Second, the AND gate is replaced with a Toffoli gate and the input to the target bit is fixed to $|0\rangle$ as shown in Figure 1.12(b). Third, the OR gate is replaced with one Toffoli and four Pauli X gates as shown in Figure 1.12(c). Finally, we must consider the fan-out in classical circuits. Since copying of qubits is not possible in general [36], this is not a trivial operation as in Boolean circuits. Note that the intermediate states of a gate by gate translation of a Boolean circuit will be computational basis states if the input is a computational basis state, *i.e.* the input to the quantum circuit correspond to classical bit-strings. It is possible to copy the value of qubits if they are in a computational basis state, which is the case of a translation from Boolean circuits. Namely, the unbounded fan-out-gate (Definition 1.12 and Figure 1.11(c)) copies the state of the control qubit (b in Figure 1.11(c)) to target qubits (x_1 to x_n in Figure 1.11(c)) if they are initialised to $|0\rangle$ [5]. Thus, every Boolean circuit has a corresponding quantum circuit. The unbounded fan-out gates can be removed from the quantum translation of a Boolean



(a) The Toffoli gate. The qubits a and b are called control qubits and the c is called the target qubit.s



(b) The Toffoli gate can be used to compute the AND of $|a\rangle$ and $|b\rangle$ by setting the target bit to $|0\rangle$.



s

(c) Using Pauli X gates and the De Morgan's laws the Toffoli gate can be used to compute the OR gate.

Figure 1.12: Using the Toffoli gate to replace classical Boolean gates.

circuit by replacing them with $O(\log n)$ depth sub-circuits (where n is the number of qubits the unbounded gate acts on) consisting of two-qubit $CONT$ gates [4].

1.2.3 The Quantum Fourier Transform

The *quantum Fourier transform* (QFT) is the quantum analogue of the *discrete Fourier transform* (DFT) algorithm. For a given dimension n, the DFT is a linear function mapping the vector $(a_0, a_1, \ldots a_{N-1})$ in $\mathbb{C}^N$ to the vector $(b_0, b_1, \ldots, b_{N-1})$, where

$$b_x = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i x y}{N}} a_y. \tag{1.23}$$

Let $\{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$ be an orthonormal basis of $\mathcal{H}^{\otimes 2^n}$. QFT is the unitary operator that maps the n qubit quantum state $\sum_{x=0}^{2^n-1} a_x|x\rangle$ to the state $\sum_{x=0}^{2^n-1} b_x|x\rangle$, where the amplitudes b_x are the DFT values from equation 1.23. The QFT of a k-qubit

computational basis state $|z\rangle$ has the following useful representation [36]:

$$\begin{aligned} QFT|z\rangle = & \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.z_1}|1\rangle) \\ & \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.z_2 z_1}|1\rangle) \\ & \otimes \cdots \\ & \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.z_k \cdots z_2 z_1}|1\rangle). \end{aligned} \tag{1.24}$$

The circuit computing the QFT is given in Figure 1.13. Throughout this thesis we use the following notation for the individual qubits in the quantum state resulting from the application of QFT on a computational basis state:

$$|QFT(z)\rangle_b = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.z_b \cdots z_2 z_1}|1\rangle\right). \tag{1.25}$$

The QFT in Figure 1.13 will be used as a sub-circuit in the quantum arithmetic circuits,

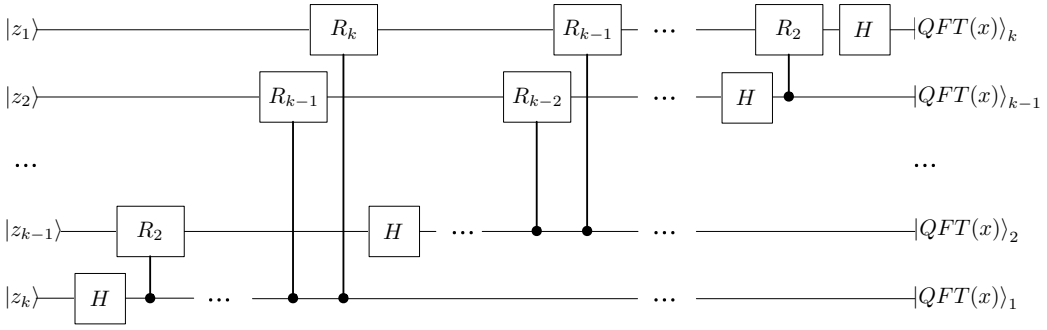


Figure 1.13: The quantum circuit computing the QFT of up to a permutation of the output qubits, *i.e.* the output qubits need to be rearranged by reversing their order. This circuit is the QFT circuit presented in textbooks [36, 37] and has $O(k)$ depth [12], which is conjectured to be the best possible depth for the exact QFT [4]

thus the parameters of the circuit are given in the following lemma:

Lemma 1.2. *The QFT of k qubits in Figure 1.13 requires k qubits, k single-qubit gates, and $(k^2 - k)/2$ two qubit gates.*

Proof. The circuit acts only on the input qubits, without requiring any auxiliary qubits; hence the number of qubits needed is k. The only single-qubit gates are the Hadamard gates, of which there is exactly one applied to each qubit; thus the number of one qubit gates is k. It can be seen from Figure 1.13 that every qubit z_i is the control qubit for exactly $i - 1$ two qubit controlled R_j gates. Thus the total number of two qubit gates

is

$$\sum_{i=1}^{k} i - 1 = k(k-1)/2 = (k^2 - k)/2. \tag{1.26}$$

□

As a summary, the parameters of the circuit in Figure 1.13 are following:

- **Depth**: $2k - 1$ [12] (Conjectured to be the best depth for the exact QFT [4]),
- **Qubits required**: k (Lemma 1.2),
- **Single-qubit gates**: k (Lemma 1.2),
- **Two qubit gates**: $(k^2 - k)/2$ (Lemma 1.2),
- **Total gates**: $k + (k^2 - k)/2 = (k^2 + k)/2$.

As can be seen from equation 1.24, the result of applying QFT on a k-qubit computational basis state can be written as a tensor product of k qubits; thus QFT acting on a computational basis state does not create any entanglement. Indeed, the quantum algorithms utilising QFT to obtain quantum speedup use the QFT on entangled states (Shor's factorising algorithm being one of them [2]). This allows the transformation of 2^k complex values in the amplitudes of a quantum state by using only k qubits — something not feasible classically. This might leave the impression that the QFT could be used to perform extremely fast DFT, but this is not the case. The transformations are done on the amplitudes and in general it is not possible to measure all the amplitudes of a quantum state.

1.3 Arithmetic Circuits

There are two basic arithmetic circuits relevant to this thesis: the adder and the multiplier. These can be combined to to create a more complex circuit, the multiply-accumulator, as will be shown in Section 1.3.3. There exists many different classical arithmetic circuits for both adders and multipliers, each of them having its own advantages and disadvantages. Many good textbooks exist describing them [27, 30, 32, 33], thus a review of all of them is not included in this thesis. Included are the descriptions of some of the more parallel arithmetic circuits, with the purpose to give an overview of the existing quantum and classical circuits and to provide a comparison with the arithmetic circuits designed in the later chapters.

1.3.1 Adders

The focus of this work is parallelism and this is reflected in the adders reviewed in this section, these are the *ripple-carry adder*, the *carry-lookahead adder*, the *carry-save adder*, and the *Draper adder* [28]. The full definitions of these adders is given in following subsections. The ripple-carry adder is included since it is one of the simplest adders and although not exhibiting much parallelism, it is worthwhile to include it since the first quantum adder proposed was a ripple-carry adder [39]. The carry-lookahead adder on the other hand is the most parallel classical adder with depth $O(\log n)$ and has also a quantum counterpart [40]. The third adder, the carry-save adder, differs from most classical adders by using a redundant representation. This makes this adder particularly efficient in calculating sums of multiple numbers, allowing sequential addition of numbers in constant depth, but requiring a final $O(\log n)$ addition using a non-redundant adder. The final adder reviewed, the Draper adder [28], does not have a classical counterpart, since it acts on the amplitudes of a quantum state in a superposition. Although initially not exhibiting much parallelism, a new parallel adder based on the Draper adder, exhibiting similar depth to the carry-save adder is created in Chapter 3. Before reviewing the aforementioned adders, it is necessary to describe the basic building blocks in classical arithmetic circuits — the half and full adder.

Half and Full Adders

The basic construction blocks of classical adders are the *half adder* and *full adder*. These adders add together two and three input bits respectively.

Definition 1.15 (Half adder [30])**.** *A half adder is a Boolean circuit which performs the addition of two one bit input numbers. Given two input bits X and Y the output bits S (Sum) and C (Carry) of the half adder are*

$$S = X + Y, \tag{1.27}$$

$$C = X \cdot Y. \tag{1.28}$$

Since the maximum result of adding two one bit numbers is 2, with binary representation 10, the half adder needs two output bits. A circuit depicting the half adder is shown together with its truth table in Figure 1.14. The circuits used in this work are the common constructions used in textbooks [27, 30, 32, 33], but there exists more than one way to implement the half and full adders. The full adder adds three input bits, but since sum of three bits can be at most 3, with binary representation 11, it is

enough to have two outputs.

Definition 1.16 (Full adder [30])**.** *A full adder is a Boolean circuit which performs the addition of three one bit input numbers. Given three input bits X, Y, C_{in} the output bits S and C_{out} of the full adder are*

$$S = A \oplus X \oplus C_{in}, \tag{1.29}$$

$$C_{out} = (X \cdot Y) + C_{in}(X \oplus Y). \tag{1.30}$$

A Boolean circuit for the full adder is shown together with its truth table in Figure 1.14. Since multiple inputs are mapped to the same output values, neither the full nor the half adder is a unitary operator and therefore cannot be directly translated to quantum operators. The unitary counterparts of half and full adder require 3 and 4 qubits correspondingly. The truth table and quantum circuit of the full adder is shown in Figure 1.16.

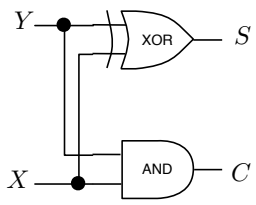
Definition 1.17 (Quantum full adder [41])**.** *The quantum full adder is the unitary operator mapping a computational basis input state to an output state according to the truth table in Figure 1.16(a).*

Analogously to the half and full adders, which are the main building blocks for classical arithmetic circuits, the quantum full adder is often used to construct quantum arithmetic circuits, but the quantum half adder is rarely used. There exist multiple different decompositions of the quantum full adder, two of which are used in this thesis. The first, allowing only one and two-qubit gates and the other allowing additionally three qubit Toffoli gates. The latter was presented in [41] and is the lowest depth decomposition into one, two, and three qubit gates known at the time of writing this work. Since many of the circuits presented in this work use only one and two-qubit gates, the full adder decomposition into two-qubit gates is given in the next lemma.

Lemma 1.3 (Quantum full adder decomposition)**.** *The quantum full adder can be*

X	Y	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

(a)

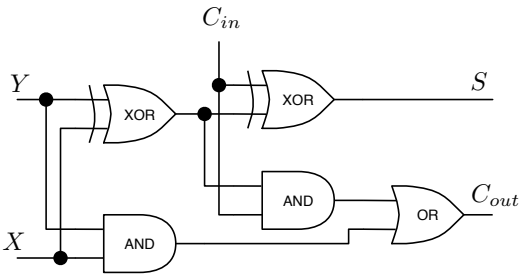


(b)

Figure 1.14: The truth table (a) and Boolean circuit (b) of the half adder. [30]

X	Y	C_{in}	S	C_{out}
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

(a)



(b)

Figure 1.15: The truth table (a) and Boolean circuit (b) of the full adder. [30]

implemented using:

1. *Two Toffoli gates, two CNOT gates, and depth 4 [41].*
2. *Twelve two-qubit gates and depth 12.*

Proof. The first of these decompositions is due to [41] and is shown in Figure 1.16(b). The second decomposition is obtained by replacing the Toffoli gates in Figure 1.16(b) with the Toffoli decomposition from [42], depicted in Figure 1.17. This Toffoli decomposition has the lowest known depth and gate count, which are both 5. Since the full adder in Figure 1.16(b) consists of two CNOT and two Toffoli gates applied in a sequence, both the depth and the number of two-qubit gates in the decomposed full adder is $2 \cdot 1 + 2 \cdot 5 = 12$. □

As a comparison, the depth and size of classical full and half adders used in this thesis is stated below. These are formulated as lemmas to emphasise the properties of the half and full adders used in this thesis.

Lemma 1.4 (Half adder decomposition)**.** *There exists a depth one Boolean circuit for a half adder consisting of two Boolean gates.*

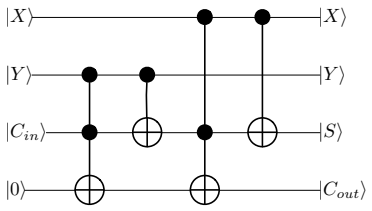
Proof. The depth and size of the half adder circuit, obvious from Figure 1.14(b) taken from [30], satisfy this lemma. □

Lemma 1.5 (Full adder decomposition)**.** *There exists a depth three Boolean circuit for a full adder consisting of five Boolean gates.*

Proof. The depth and size of the half adder circuit, obvious from Figure 1.15(b) taken from [30], satisfy this lemma. □

Input				Output			
X	Y	C_{in}	C'_{out}	X	Y	S	C_{out}
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	0	1	0
1	0	0	1	1	0	1	1
1	0	1	0	1	0	0	1
1	0	1	1	1	0	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	0	0
1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	0

(a)



(b)

Figure 1.16: The truth table (a) and quantum circuit (b) of the quantum full adder. It can be seen from the truth table that when the fourth qubit ($|C'_{out}\rangle$) is initialised to $|0\rangle$, its final value $|C_{out}\rangle$ will correspond to the C_{out} bit in a classical full adder. Setting $|C'_{out}\rangle$ to $|1\rangle$ will flip this output value.

Note that by translating a full adder to a quantum adder, the depth and two-qubit gate count will increase. This is one of the reasons why some of the quantum arithmetic circuits encountered later have a larger depth and size than their classical counterparts.

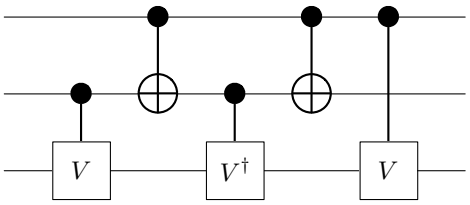


Figure 1.17: The decomposition of the Toffoli gate into two-qubit gates [42].

Ripple-Carry Adder

One of the simplest adders is the ripple-carry adder shown on Figure 1.18. A k-bit ripple-carry adder (allowing to add two k-bit integers) consists of an initial half adder followed by $k-1$ sequentially applied full adders [30]. The total number of gates used can be estimated by adding together the Boolean gates in the full adders and half adder.

Lemma 1.6. *The ripple-carry adder for two k-bit integers can be constructed by using $5k-3$ Boolean gates.*

Proof. The ripple-carry adder consists of $k-1$ full adders and one half adder, whereby the number of gates in a half adder is two (Lemma 1.4) and in a full adder is five (Lemma 1.5). Thus the total number of gates is $5(k-1)+2 = 5k-3$. □

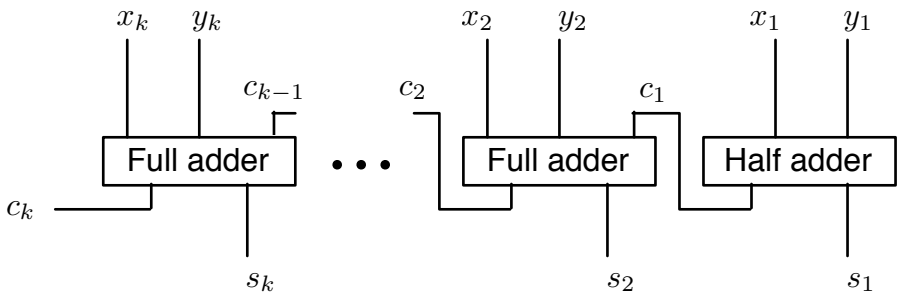


Figure 1.18: The ripple-carry adder [30]. The full and half adders need to be applied sequentially from right to left.

The parameters for the ripple-carry adder are following:

- **Depth**: $2k-1$ [30],
- **Gates**: $5k-3$ (Lemma 1.6).

Since the ripple-carry adder does not require any fan-out, it is one of the most natural adders to be translated into the quantum circuit model (fan-out is not possible in the quantum circuit model) and indeed, the first proposed quantum adder was a ripple-carry adder [39]. Since the unitary operators used as the replacement for the classical full adder required an auxiliary qubit, the initial quantum ripple-carry adder needed k auxiliary qubits. This adder was improved by Cuccaro *et.al.* [43] to use just one auxiliary qubit. Their adder has the following parameters:

- **Depth**: $2k-1$ [43],
- **Qubits required**: $2k+2$ [43],

- **CNOT gates**: $5k - 3$ [43],
- **Toffoli gates**: $2k - 1$ [43],
- **Total gates**: $(5k - 3) + (2k - 1) = 7k - 4$.

Note that in their paper Cuccaro *et.al.* did not estimate then number of quantum NOT gates (X gates), thus these figures also exclude them. Both the depth of the quantum ripple-carry adder and the number of qubits used is almost equal to the classical ripple-carry adder as can be seen in Table 1.1. Nevertheless, the quantum

Table 1.1: Comparison of classical and quantum ripple-carry adders.

	Depth	Total Gates	Qubits
Classical	$2k - 1$	$5k - 3$	-
Quantum	$2k - 1$	$7k - 4$	$2k - 1$

ripple-carry adder allows quantum states to be added in superpositions and thus can be used as a subroutine in quantum algorithms. For example, it can be used to construct the quantum modular exponentiation circuits required for the Shor's factorizing algorithm. [39]

Carry-Lookahead Adder

In the ripple-carry adder, every full adder (after the first one) needs to wait for the output carry bit from the previous full adder, thus limiting the depth of the circuit to $\Omega(k)$. Carry-lookahead adders address this problem by computing the carry bits before they are needed in the full adders. There are various designs to do this. Some adders compute blocks of carries simultaneously, and chain up the carry-lookahead blocks similarly to the ripple-carry adder, others use unbounded fan-in to compute the carry values. The carry-lookahead adder with constant fan-in, which has the lowest depth is the Kogge-Stone adder [44]. In the rest of the thesis when a carry-lookahead adder is used, the Koegg-Stone adder is implicitly meant unless otherwise mentioned. From the truth table of the full adder (Figure 1.15(a)) the relation shown in Table 1.2 between the input bits and the C_{out} can be seen.

When $X \neq Y$ the C_{out} will always equal C_{in} and the carry is then *propagated*. When $X = Y = 0$ the incoming carry gets *killed* and if $X = Y = 1$ a new carry is *generated* regardless of C_{in}. Consider an interval $[i, j]$. A bit $G[i, j]$ is used to mark whether this interval generates a carry, and a bit $P[i, j]$ to mark if it propagates a carry. Here

Table 1.2: The value of C_{out} in a carry-lookahead adder.

X	Y	C_{out}
0	0	0
0	1	C_{in}
1	0	C_{in}
1	1	1

the interval is used in an unconventional way, where the end is specified first and the beginning last. This notation is used so that the beginning and end of the intervals would correspond to how we interpret bits of a binary number, where the rightmost is the first bit and the leftmost the k-th bit. Based on the above observation the carry and propagate bits for the interval $[i, i]$ can be computed as follows

$$G[i, i] = X_i \cdot Y_i \tag{1.31}$$

$$P[i, i] = X_i \oplus Y_i \tag{1.32}$$

Given two carry intervals $[i, j]$ and $[j - 1, k]$, the combined interval $[i, k]$

- generates a carry if and only if either $[i, j]$ generates a carry or $[j-1, k]$ generates and $[i, j]$ propagates a carry, *i.e.*

$$G[i, k] = G[i, j] + G[j - 1, k] \cdot P[i, j]; \tag{1.33}$$

- propagates a carry if and only if both $[i, j]$ and $[j - 1, k]$ propagate, *i.e.*

$$P[i, k] = P[i, j] \cdot P[j - 1, k]. \tag{1.34}$$

The value of $G[i, 1]$ is 1 if and only if there will be a carry generated that reaches bit i in the interval $[i, 1]$, *i.e.* it is the carry bit required to compute the bit i in the sum of two numbers. Using a tree structure it is possible to compute the value $G[i, 1]$ in $\lceil \log i \rceil$ depth. When combining this tree with the generation of the $G[i, i]$ and $P[i, i]$ bits and final addition of the carry and sum bits gives the Koegg-Stone adder, of which an 8-bit example is depicted in Figure 1.19.

The first quantum carry-lookahead adder was designed in [45], but their adder had depth $O(k)$ while using unbounded Toffoli gates. Although having smaller depth than the quantum ripple-carry adder when the constants in the depth are taken into account, this depth benefit would be obtainable only if unbounded fan-out gates could be easily implementable. Usually, only constant size (and thus fan-out) gates are allowed in

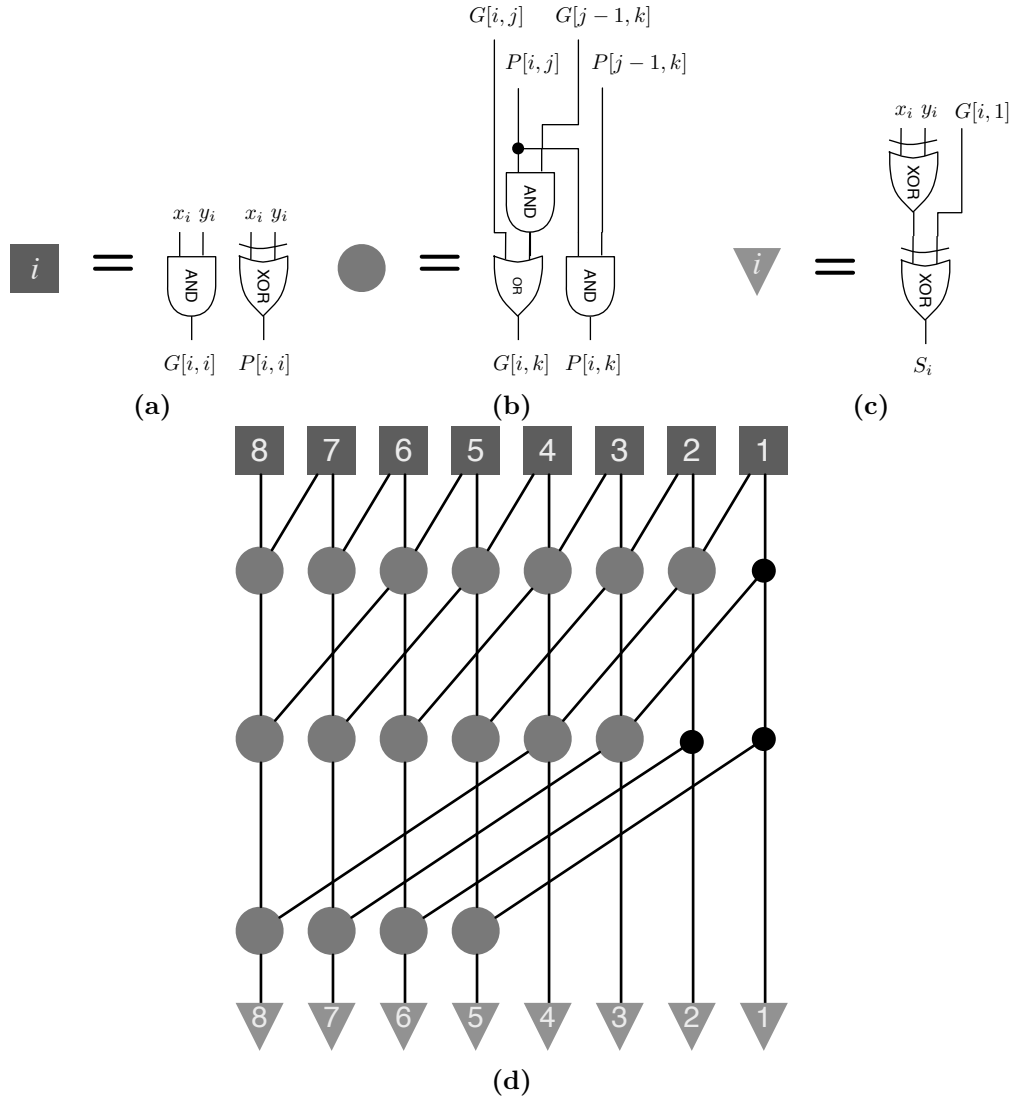


Figure 1.19: The 8-bit Kogge-Stone adder (d) with its various components shown in (a), (b), and (c)

quantum circuits, and if constant size gates would be used instead of the unbounded fan-out gates in [45], the depth would increase to $O(k \log k)$. A logarithmic depth carry-lookahead adder design, based on the Kogge-Stone adder, was proposed in [40]. In addition to considerably smaller depth, this adder used only constant sized gates, which is in general a desirable feature of quantum circuits. Table 1.3 gives a summary of the properties of the quantum and classical carry-lookahead adders.

Carry-Save Adder

When multiple numbers need to be added together, the adder most suitable for the task is the carry-save adder [46], which uses redundancy to add two numbers in constant depth. The principle of the carry-save adder is to compute the sum and carry generated by adding every two corresponding bit, but instead of propagating the carry, it is saved until the next addition is performed. Obviously this system is redundant, since the result consists of k sum bits and k carry bits. When the next number is added, the carry from the last addition is propagated to the next position by adding it with the new sum bits. The operation of the carry-save addition is illustrated in Figure 1.20. As can be seen from Figure 1.20, the carry-save adders have constant depth since they consist of full adders, which are applied in parallel. Thus the sequence of n carry-save adders has depth $O(n)$.

After the numbers are added together, a final conventional adder must be used to convert the result from the redundant representation to a non-redundant by adding the two resulting numbers (the carry bits and the sum bits). This could be done with carry-save adders, but this requires k applications of the adder for the rightmost carry bit to propagate to the left. A better solution is to use an $O(\log n)$ depth adder, such as the Kogge-Stone adder, resulting in a total depth of $O(n + \log k)$ when adding n numbers of k-bits. Since a final non-redundant adder is needed when using carry-save adders,

Table 1.3: Comparison of classical and quantum carry-lookahead adders. Here it is assumed that k is a power of two.

	Classical	Quantum
Depth	$2 \log k + 2$	$2 \log k + 2$
CNOT Gates	-	$3k - 1$
Toffoli Gates	-	$2k - 3 \log k - 3$
Total Gates	$5k \log k + 3k + 5$	$5k - 3 \log k - 4$
Qubits	-	$4k - \log k$

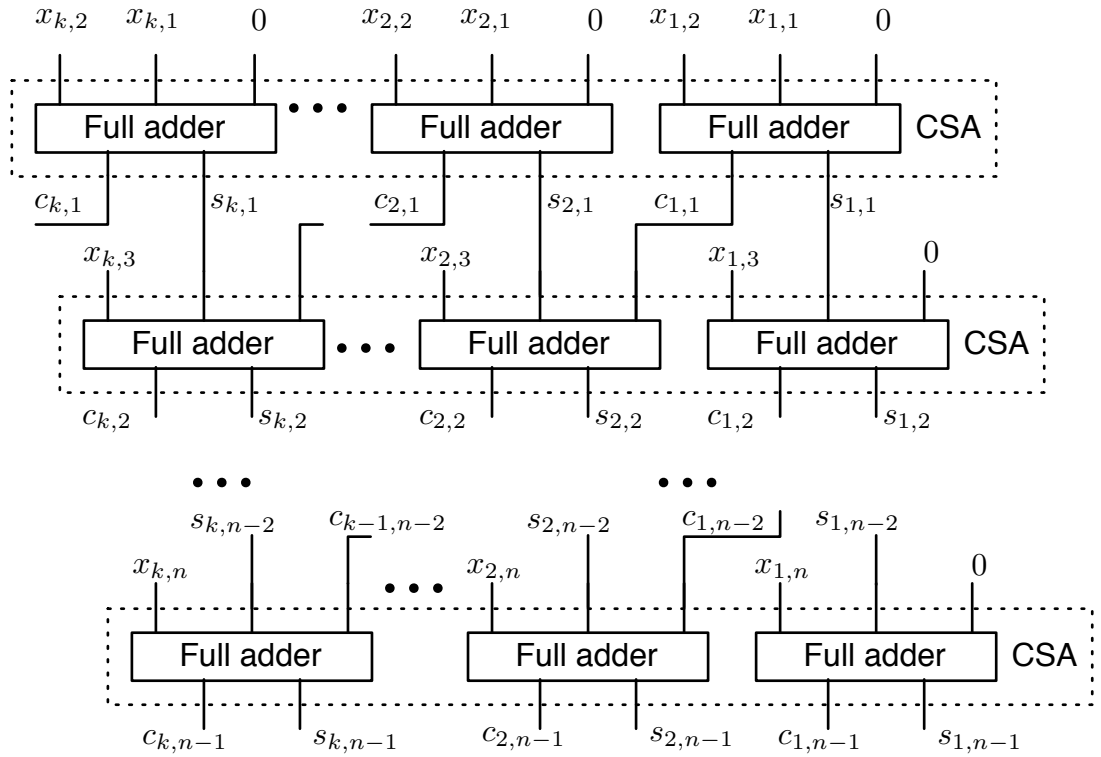


Figure 1.20: The application of n carry-save adders of k-bits in a sequence (CSA is used to denote a carry-save adder). To get the final result an additional non redundant adder, like the carry-lookahead adder, needs to be used to add the carry and sum outputs of the last carry-save adder.

this adder is not suitable to use when only two numbers are added. On the other hand, it will give a considerable improvement in computational depth when multiple numbers are added in a sequence, thus the carry-save adder is the most commonly used adder in multipliers.

Interestingly, it is possible to add n numbers of k bits in depth $O(\log n \log k)$ using k-bit adders. This can be done by arranging the adders for two numbers (with $O(\log k)$ depth) into a $O(\log n)$ depth tree and thereby summing the numbers recursively. This solution, although having low depth, is unpractical since to compute the sums in the first layer of the tree, $O(n)$ adders are needed. The number of available adders in modern electronic circuits is a fixed, and usually a small, constant and does not scale with the problem size. Thus in this work we consider the case where there is only a constant number of available adders. This number is usually one in the analysis, since adding a constant number of adders would reduce the overall depth only by a constant, when adding n numbers.

A straightforward translation (by making the classical circuit reversible and cleaning

Table 1.4: Comparison of classical and quantum carry-save adders. The values for the classical adder are taken from circuits presented in [30] and the quantum adder parameters are taken the quantum carry-save adder presented in [41]. We used Lemma 1.3 to estimate the number of two qubit-gates, whereas the original circuit consisted of $2k$ quantum full adders. Since the decomposition of Lemma 1.3 does not contain any single-qubit gates, neither does the quantum carry-save adder.

	Classical	Quantum
Depth	3	24
Single-Qubit Gates	-	-
Two-Qubit Gates	-	$24k$
Total Gates	$5k$	$24k$
Qubits	-	$4k$

up the auxiliary qubits) from the classical to quantum carry-save adder was shown in [41]. The parameters of the resulting quantum circuit are compared with the classical couterparts in Table 1.4. Since in the quantum circuit model the registers are cleaned by applying the inverse of the quantum full adders at the end of the computation, every full adder in the classical carry-save adder has two corresponding quantum full adders in its quantum counterpart. This is one of the contributors to the larger depth and number of gates in the quantum circuit — one quantum full adder consists of 12 sequential two-qubit gates and acts on four qubits.

The Draper Adder

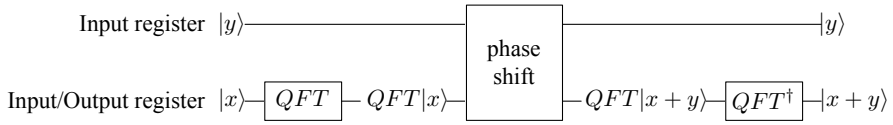


Figure 1.21: The Draper adder.

Most of the quantum arithmetic circuits do not contain anything inherently quantum. They consist of reversible Boolean gates [39, 41, 45, 47, 43, 40, 48, 49, 50], usually Toffoli and CNOT gates, and therefore would work without modifications for classical reversible computations. Hence it is unlikely that studying these circuits would reveal any differences between the classical and quantum computing models. To explore the dissimilarities between the two models and find lower depth circuits some sort of "quantumness" has to be used in the construction of the quantum circuits. The only known arithmetic circuit that does this is the Draper adder [28]. The structure of the

Draper adder is shown in Figure 1.21, from where it can be seen that adding two k-bit integers x and y modulo k has the following steps:

1. The integers are stored in two k-qubit quantum registers.
2. QFT is applied to the second register.
3. A phase shift sub-circuit, shown in Figure 1.22, is applied to the two registers.
4. QFT† is applied to the second register.
5. The second register is measured in the computational basis given as the output.

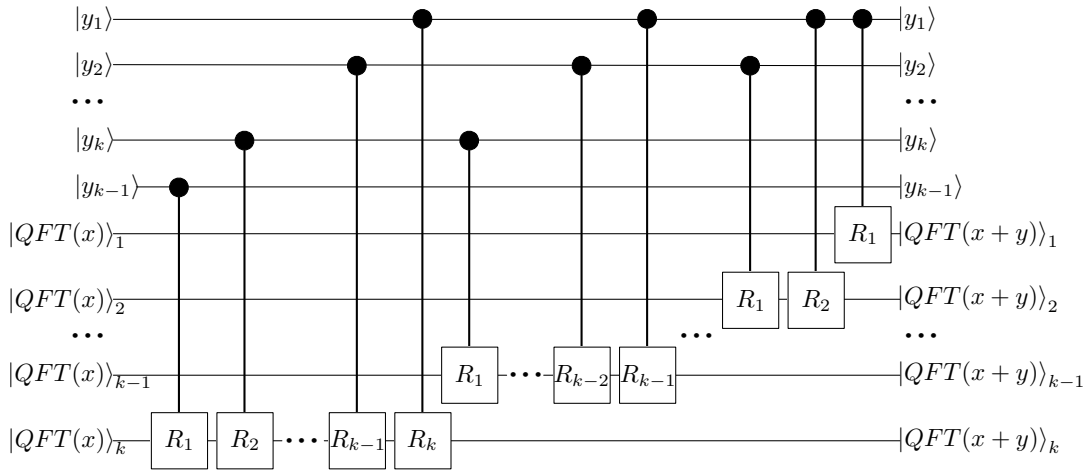


Figure 1.22: The phase shift block of a k-qubit Draper adder.

After the first QFT, the second quantum register is in state

$$\begin{aligned} QFT|x\rangle = & \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.x_1}|1\rangle) \\ & \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.x_2 x_1}|1\rangle) \\ & \otimes \cdots \\ & \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.x_k \cdots x_2 x_1}|1\rangle). \end{aligned} \tag{1.35}$$

Thus applying the phase shift block in Figure 1.22 results in the state

$$\begin{aligned}
&\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.x_1+0.y_1)}|1\rangle) \\
\otimes&\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.x_2y_1+0.y_2y_1)}|1\rangle) \\
\otimes&\cdots \\
\otimes&\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.x_k\cdots x_2x_1+0.y_k\cdots y_2y_1)}|1\rangle),
\end{aligned} \tag{1.36}$$

which can be rewritten as

$$\begin{aligned}
&\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i0.(x+y)_1}|1\rangle) \\
\otimes&\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i0.(x+y)_2(x+y)_1}|1\rangle) \\
\otimes&\cdots \\
\otimes&\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i0.(x+y)_k\cdots(x+y)_2(x+y)_1}|1\rangle) = QFT|x+y\rangle.
\end{aligned} \tag{1.37}$$

Finally, undoing the QFT by applying the adjoint results in $|x+y\rangle$.

The Draper adder utilises quantum effects to perform the addition: the integers are stored and manipulated in the relative phases of the quantum state — something that is not possible classically, since bits cannot be in a superposition. One could of course simulate the Draper adder hoping it might reveal a new classical addition algorithm. It is possible to use integers for storing and modifying the required powers of e like the Draper adder does on the amplitudes. Note that in this case the simulation of the last bit of the Draper requires the computation of $x + y$, the final value of the circuit itself, removing the need to simulate the Draper adder at all and providing no new and insightful classical algorithm.

As can be seen from Table 1.5, the depth of the Draper adder is larger than for the classical adders reviewed so far (although smaller than the depth of the quantum ripple-carry adder). The Draper adder needs $2k$ qubits for its two quantum registers and has

Table 1.5: The resources required to implement a k-qubit Draper adder.

Depth	Single-Qubit Gates	Double-Qubit Gates	Total Gates	Qubits
$5k-2$	$2k$	$(3k^2-k)/2$	$3(k^2-k)/2$	$2k$

depth $5k-2$ (the QFT and $QFT^\dagger$ have $2k-1$ depth and the phase shift block has depth k). The total number of gates used is $3(k^2+k)/2$ (the QFT, $QFT^\dagger$, and the phase shift block consisting of $(k^2+k)/2$ gates each) out of which $2k$ are Hadamard gates and $(3k^2-k)/2$ are control-phase gates.

1.3.2 Multipliers

The main operation in classical multiplier circuits is addition. To multiply two k-bit numbers (possibly in the two's complement representation) x and y, first the partial products $xy_1, xy_2, \ldots, xy_k$ are produced. The main difference between multipliers lies in when and how these are computed and added together. In our work we are interested in *parallel multipliers*, which have the lowest depth. In these multipliers the partial products can be found in parallel with one step using AND gates. The most well-known parallel multipliers are the Wallace-tree [51] and Dadda-tree [52] based multipliers. The aforementioned multipliers have $O(\log n)$ depth, which puts the multiplication into complexity class NC^1 and is the smallest depth possible without using unbounded fan-in gates (see Part III of this thesis for an overview of low depth complexity classes). An high-level diagram of the Wallace and Dadda multipliers for 12 bit integers is shown in Figure 1.23.

Since the Dadda and Wallace multipliers use the carry-save adder to add together the partial products, they need a non-redundant adder (carry-lookahead adder for example) to perform the final addition. The Wallace multiplier differs from the Dadda multiplier by the partial sum bits added together at each step. The Wallace multiplier adds together as many bits as possible, whereby the Dadda multiplier minimises the use of half adders. For example, consider the addition of three partial sums, obtained in multiplication of $x_6x_5x_4x_3x_2x_1$ with $y_3y_2y_1$:

$$\begin{array}{cccccccccc} & & & x_6y_1 & x_5y_1 & x_4y_1 & x_3y_1 & x_2y_1 & x_1y_1 & \\ & & x_6y_2 & x_5y_2 & x_4y_2 & x_3y_2 & x_2y_2 & x_1y_2 & & \\ + & x_6y_3 & x_5y_3 & x_4y_3 & x_3y_3 & x_2y_3 & x_1y_3 & & & \end{array} \tag{1.38}$$

Note that since in the final step a non-redundant adder needs to sum the resulting two partial sums, it is not needed that the sum of the bits x_2y_1 and x_1y_2 are computed in the example above, this could be left for the last adder. In general, the full adder has three inputs and two outputs, thus adding three bits reduces the number of bits needed to add in the following steps by one. On the other hand, when two bits are added using the half adder, the number of output bits is still two. Hence to minimise

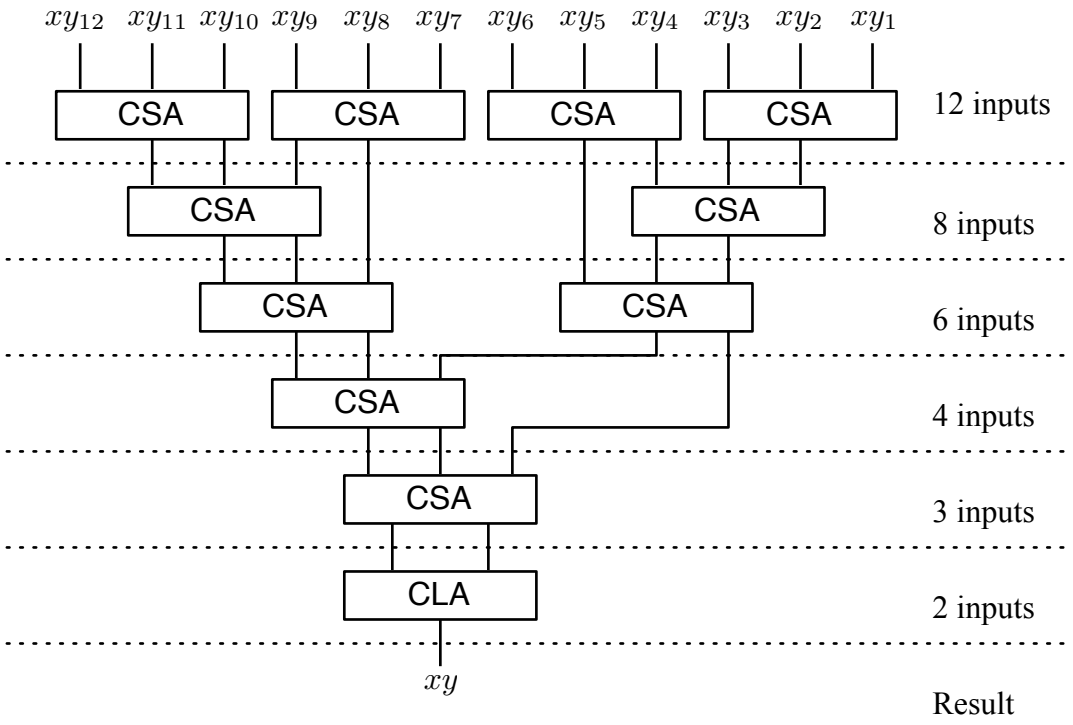


Figure 1.23: A high-level overview of the Wallace and Dadda multiplier for 12 bit integers. On each level, the number of inputs to the carry-save adders will decrease by a factor of 1.5. The final adder needs to be a non-redundant adder like the carry-lookahead adder.

the gate count it is beneficial to use as few half-adders as possible, as is done in the Dadda multiplier. The Table 1.6 summarises the parameters of the Dadda multiplier, was chosen over the Wallace multiplier because its lower gate count.

Table 1.6: The resources required to implement a k-qubit Dadda multiplier. The estimates on the number of half and full adders is taken from [53].

	Depth	Full Adders	Half Adders	Total Gates
Partial Products	1	-	-	k^2
Intermediate CSAs	$3\lceil \log_{1.5} k/2 \rceil$	$k^2 - 4k + 3$	$k - 1$	$5k^2 - 22k + 13$
Final CLA	$2 \log k + 2$	-	-	$5k \log k + 3k + 5$
Total	$3\lceil \log_{1.5} k/2 \rceil + 2 \log k + 3$	$2k - 4$	$15k - 8$	$6k^2 - 5k \log k - 19k + 18$

Several quantum multiplication circuits have been proposed, most of which are reversible counterparts to classical circuits: sequential multipliers using the ripple-carry adder [39, 54] (depth $O(k^2)$), array multiplier [55] (depth $O(k \log k)$), the Booth multiplier [56] (depth $O(\log^2 k)$). The only multiplier having no classical counterpart is the sequential multiplier utilising using the Draper adder [57]. Out of the currently published quantum multipliers, the lowest depth one is the Booth multiplier with depth $O(\log^2 k)$ [56], which is larger than the $O(\log k)$ depth of the classical Dadda and Wallace adders.

1.3.3 Multiply-Adders

Table 1.7: The depth and gate count of the MAC depicted in Figure 1.24. We assume that the MAC is reused for every application, hence the gate count does not increase per application.

	Depth	Gates
Multiplication (Dadda)	$3\lceil \log_{1.5} k/2 \rceil + 1$	$6k^2 + 22k + 13$
Addition (CSA)	2	$5k$
Final Addition (CLA)	$2 \log k + 2$	$5k \log k + 3k + 5$
Total for n applications	$3n(\lceil \log_{1.5} k/2 \rceil + 1) + 2 \log k + 2$	$6k^2 + 5k \log k + 30k + 18$

A *multiply-adder*, a.k.a. *multiply-accumulator* (MAC) performs the multiplication of

two integers x and y and adds the result to a register z, *i.e.* it performs the operation

$$z = z + xy. \tag{1.39}$$

The MAC is usually implemented using a multiplier and an adder, thus its characteristics depend on the chosen multiplier and adder. Depthwise, using the Dadda multiplier and carry-save adder will give the best result — carry-save adder in particular since the MAC is usually applied in a sequence to many numbers. If the Dadda adder is used in a MAC where the addition to the output register is done with a carry-save adder, it is possible omit the final carry-lookahead adder since the result of the multiplication would be used as an input to a carry-save adder. Instead of the carry-lookahead adder we could use a second carry-save adder in adding the product to the output register as shown in figure 1.24. Of course a final non-redundant adder must be used to transform the output to the usual binary representation. Taking this into account, the properties of a low depth MAC are presented in Table 1.7.

Although quantum arithmetic logic units (ALUs) have been proposed in several papers [58, 59, 60, 61], none of them analyse if the addition and multiplication could be merged into a single, more efficient multiply-add operation. Thus this work presents the first quantum MAC.

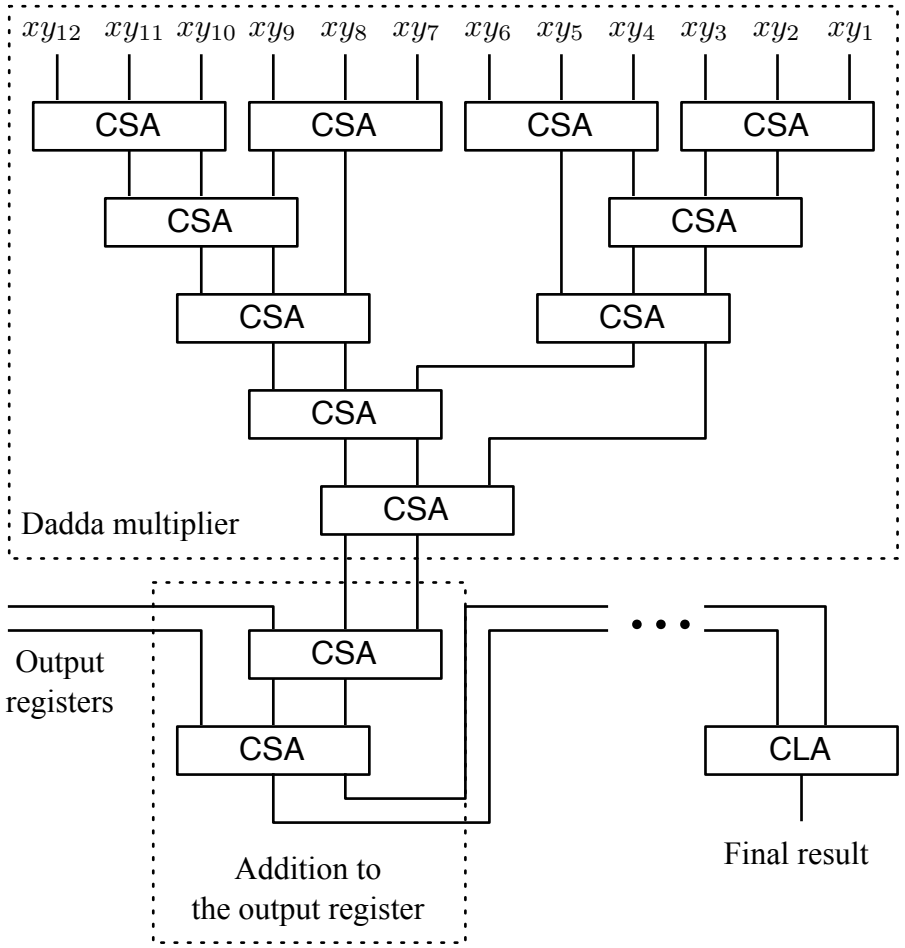


Figure 1.24: A high-level overview of a MAC using the Dadda multiplier for multiplication and the carry-save adder to add to the output register. Note that it is possible to delay the conversion to a non-reduntant representation until all the MACs have been applied as is done here.

Chapter 2

The QFT Multiply-Adder

We start this chapter by introducing a unitary operator which, when combined with the QFT, can be used to compute the action of a classical integer MAC: $z = z + y \cdot x$, where $z, y, x \in \mathbb{Z}$. This, like any unitary operator, can be decomposed into one and two-qubit gates. The decomposition presented in this section is particularly useful, since it allows for the construction of a highly parallel quantum circuit. For application on multiple numbers, this new circuit has lower depth than any known classical MAC. For the sake of notational simplicity, only unsigned integers are considered but the presented circuits work with signed integers if the two's complement representation is used and can be adapted to work with fixed point arithmetic as shown in Section 1.1.1.

2.1 The QMAC Circuit

Let $M_j(y, x)$ be a single-qubit unitary operator defined as follows:

$$M_j(y, x)|0\rangle \rightarrow |0\rangle, \tag{2.1}$$

$$M_j(y, x)|1\rangle \rightarrow e^{2i\pi 0.y_j \cdots y_1 \cdot x}|1\rangle, \tag{2.2}$$

where $x, y \in \mathbb{Z}$ are k-bit integers. The effect of applying $M_j(y, x)$ to a state which has the first j bits of a k-bit integer z encoded in its relative phase is

$$M_j(y, x)\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.z_j \cdots z_2 z_1}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_j \cdots z_2 z_1 + 0.y_j \cdots y_2 y_1 \cdot x)}|1\rangle). \tag{2.3}$$

The above equation shows that the action of $M_j(y, x)$ is similar to applying a MAC operator to the binary fraction encoded in the relative phase, *i.e.* it multiplies the

binary fraction $0.y_j \cdots y_2 y_1$ with x and adds it to $0.z_j \cdots z_2 z_1$. Furthermore, we define the k qubit quantum operator $M(y, x)$

$$M(y,x) = M_1(y,x) \otimes M_2(y,x) \otimes \cdots \otimes M_k(y,x). \tag{2.4}$$

The application of $M(y, x)$ to $QFT|z\rangle$ will result in the state

$$\begin{aligned} M(y,x)QFT|z\rangle = & \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_1 + 0.y_1 \cdot x)}|1\rangle) \\ & \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_2 z_1 + 0.y_2 y_1 \cdot x)}|1\rangle) \\ & \otimes \cdots \\ & \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_k \cdots z_2 z_1 + 0.y_k \cdots y_2 y_1 \cdot x)}|1\rangle) = QFT|z + y \cdot x\rangle. \end{aligned} \tag{2.5}$$

Applying the $QFT^\dagger$ operator to the result and measuring the result in the computational basis gives the output $z + y \cdot x$, which is also the result of a classical MAC applied to x, y, z. Thus $M(y, x)$ can be seen as a quantum operator on a quantum Fourier transformed state corresponding to a classical MAC. Note that for any $l \leq k$ the l-th qubit of $M(y, x)QFT|z\rangle$ will be in the form

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_l \cdots z_2 z_1 + 0.y_l \cdots y_2 y_1 \cdot x)}|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(m.w_l \cdots w_2 w_1)}|1\rangle), \tag{2.6}$$

where $m \in \mathbb{Z}$ and $w \in \{0,1\}^l$. Since

$$e^{2\pi i(m.w_l \cdots w_2 w_1)} = e^{2\pi i(m + 0.w_l \cdots w_2 w_1)} = e^{2\pi i 0.w_l \cdots w_2 w_1} \tag{2.7}$$

and $l \leq k$ all the bits of the result $z + y \cdot x$ after the k-th will be lost, which corresponds to computation modulo $2^k - 1$.

Any realistic quantum device would have to be built using quantum gates which act on a limited number of qubits (*i.e.* the number of qubits the gates act on should be fixed to a small constant), thus the $M(y, x)$ operator needs to be decomposed into one- and two-qubit quantum gates. To obtain a performance that surpasses classical MACs the $M(y, x)$ operation will be constructed in a way that allows every gate in its circuit to be applied in one simultaneous step. The following gates are used in the circuit construction:

$$R_j = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2i\pi}{2^j}} \end{bmatrix}, \quad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{2.8}$$

where R_j is a phase shift gate and $CNOT$ is the controlled NOT gate. Note that the operator R_j has the following properties:

$$\forall j \in \mathbb{Z} < 1 \quad R_j = I, \tag{2.9}$$

$$R_j^{2^m} = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2i\pi 2^m}{2^j}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2i\pi}{2^{j-m}}} \end{bmatrix} = R_{j-m}. \tag{2.10}$$

The j-qubit *fan-out* operator F_j which maps $|a\rangle|b_1\rangle \cdots |b_{j-1}\rangle \rightarrow |a\rangle|b_1 \oplus a\rangle \cdots |b_{j-1} \oplus a\rangle$. It is trivial to see that $F^\dagger = F$. The operator $Q_j(y) = R_{j \cdot y_1} \cdots R_{2 \cdot y_{j-1}} R_{1 \cdot y_j}$ is used as a sub-circuit in the $M(y, x)$ construction. The effect of $Q_j(y)$ on the one qubit computational basis is:

$$Q_j(y)|0\rangle \rightarrow |0\rangle, \tag{2.11}$$

$$Q_j(y)|1\rangle \rightarrow e^{2\pi i \cdot 0.y_j \cdots y_2 y_1}|1\rangle. \tag{2.12}$$

Note that since y_m, where $m \in \{1, 2, \ldots, j\}$, is a binary value and $R_0 = R_l^0 = I$ for every $l \in \mathbb{Z}$, the operator $Q_j(y)$ can be written as follows:

$$Q_j(y) = R_j^{y_1} \cdots R_2^{y_{j-1}} R_1^{y_j}. \tag{2.13}$$

Furthermore, from the equalities 2.9 and 2.10 it follows that:

$$\begin{aligned} Q_j(y)^{2^m} &= R_j^{y_1 \cdot 2^m} \cdots R_2^{y_{j-1} \cdot 2^m} R_1^{y_j \cdot 2^m} \\ &= R_{j-m}^{y_1} \cdots R_{2-m}^{y_{j-1}} R_{1-m}^{y_j} \\ &= Q_{j-m}(y). \end{aligned} \tag{2.14}$$

The above equation implies that $Q_{j-m} = I$ if $j - m < 1$, therefore the $M_j(y, x)$ operator

can be written as:

$$\begin{aligned} M_j(y,x)|1\rangle &= e^{2\pi i \cdot 0.y_j \cdots y_1 \cdot x}|1\rangle \\ &= e^{2\pi i \cdot 0.y_j \cdots y_1 \cdot (x_1 \cdot 2^0 + x_2 \cdot 2^1 + \cdots + x_k \cdot 2^{k-1})}|1\rangle \\ &= Q_j(y)^{x_k \cdot 2^{k-1}} \cdots Q_j(y)^{x_2 \cdot 2^1} Q_j(y)^{x_1 \cdot 2^0}|1\rangle \\ &= Q_{j-k+1}(y)^{x_k} \cdots Q_{j-1}(y)^{x_2} Q_j(y)^{x_1}|1\rangle \\ &= Q_1(y)^{x_j} \cdots Q_{j-1}(y)^{x_2} Q_j(y)^{x_1}|1\rangle \qquad (2.15) \\ M_j(y,x)|0\rangle &= Q_1(y)^{x_j} \cdots Q_{j-1}(y)^{x_2} Q_j(y)^{x_1}|0\rangle = |0\rangle. \qquad (2.16) \end{aligned}$$

The decomposition of $M_j(y,x)$ into $Q_j(y)$ (Eq. 2.15 and 2.16) operators and $Q_j(y)$ into R_j operators (Eq. 2.13) will be used to construct a parallel quantum circuit for $M(y,x)$. Note that the descriptions of $M(y,x)$, $M_j(y,x)$, and $Q_j(y)$ contain the arguments x and y. This is undesired for implementations of a circuit, since a circuit cannot in general change depending on the input. In the design below, this problem is resolved by using the bits of the arguments as controls for quantum gates, *i.e.* the value of classical bits is used to determine if a particular quantum gate should be applied or not. First, the parallel hybrid circuit for the operator $Q_j(y)$ is constructed. Since $R_j^0 = I$ and $R_j^1 = R_j$, the effect of an input bit y_m in Eq. 2.13, where $m \in \{1, 2, \ldots, j\}$, is to control the application of the gate R_{j+1-m}. Thus the quantum circuit of $Q_j(y)$ can be constructed using only single-qubit R_j gates controlled by classical bits y_m. All of the R_j gates in $Q_j(y)$ can be applied in parallel using auxiliary qubits and the F_j gate [4]. Thus the parallel hybrid circuit $F_j Q_j F_j$ of $Q_j(y)$ can be constructed as shown in Figure 2.1. Note that we move y out of the argument of the operator and use a fixed circuit for every y — the y will be the input to the circuit.

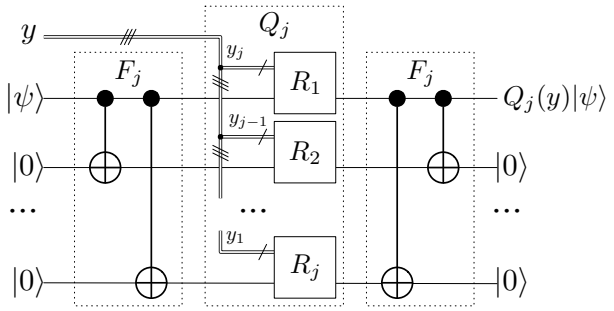


Figure 2.1: The parallel version of the operator $Q_j(y)$. The F_j blocks can be applied in $O(\log j)$ steps [4]. $|\psi\rangle$ is an arbitary 1 qubit state. See Figure 1.7 for circuit notation.

Since $Q_j(y)^0 = Q_j(0) = I$ and x_i is a binary value, M_j can be written as $M_j(y,x) = Q_1(y \cdot x_j) \cdots Q_{j-1}(y \cdot x_2) Q_j(y \cdot x_1)$. The values of both y and x are classical bit-strings,

hence the operation $y \cdot x_i$ can be performed classically in one parallel computational step using an AND operator between x_i and every bit of y. Since $M_j(y,x)$ can be decomposed into diagonal operators $Q_j(y)$, there exists a parallel hybrid circuit where all the $Q_j(y)$ operators are applied simultaneously [4]. In this circuit's construction the sub-circuit Q_j is used as seen in Figure 2.2. Since $M(y,x)$ is a tensor product of

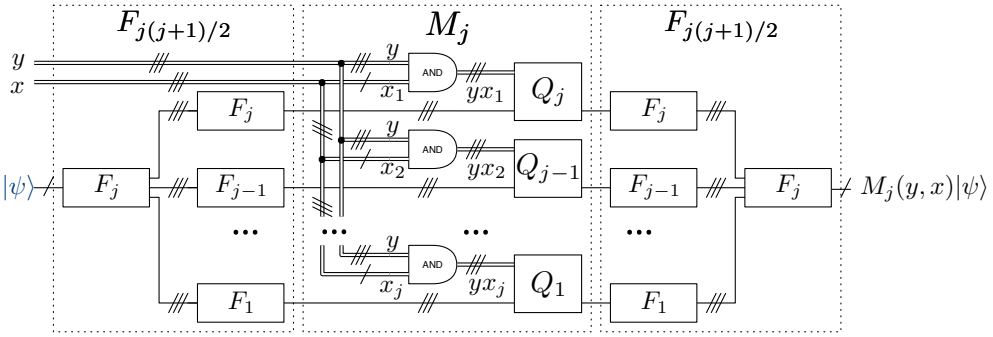


Figure 2.2: The parallel hybrid circuit of the $M_j(y,x)$ operator.

the operators $M_j(y,x)$, where $j \in \{1, \cdots k\}$, the circuit of $M(y,x)$ can be created by simply applying an appropriate M_j sub-circuit to each of the input qubits as shown in Figure 2.3. The circuit FMF in the aforementioned figure corresponds to the operator $M(y,x)$ and together with the QFT comprises the quantum MAC circuit.

2.2 Analysis of the Circuit

The main result of this work concerns the depth of the QMAC circuit in the case of sequential application. When the circuit FMF in Figure 2.3 is applied repeatedly, then the only F gates having a non-trivial effect will be at the beginning and the end of the computation. This is due to the fact that $FF = FF^\dagger = I$ and thus

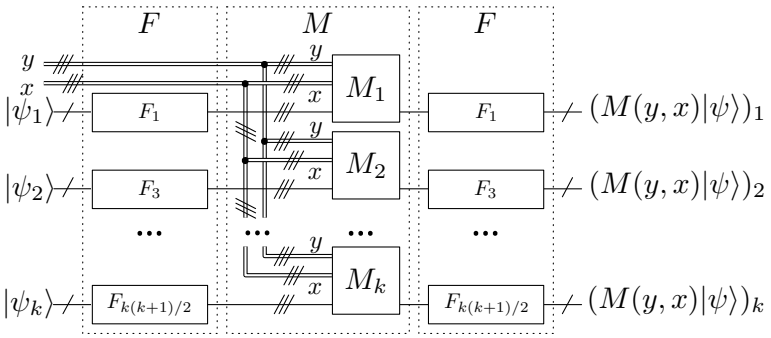


Figure 2.3: The parallel hybrid circuit of the $M(y,x)$ operator.

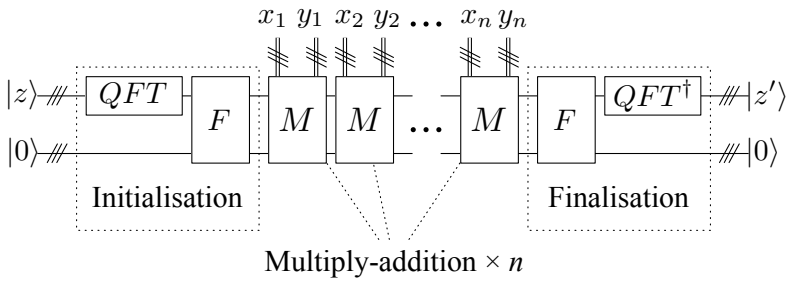


Figure 2.4: The hybrid quantum circuit computing the MAC operation n times in a sequence with multiplicand pairs $(x_1, y_1), \ldots, (x_n, y_n)$, where $z, x_i, y_i \in \mathbb{Z}$. Here $z' = z + \sum_{i=1}^{n} x_i \cdot y_i$.

$(FMF)(FMF) = FMMF$. Combining the circuit in Figure 2.3 with the QFT and using it to perform the multiply-addition operation of n integers results in the circuit depicted in Figure 2.4. We divide the circuit into three parts:

- The initialisation consisting of the QFT and the initial fan-out.
- The multiply-addition consisting of the sub-circuit M and which is reapplied for every pair of integers needed to be multiply-accumulated.
- The finalisation part consisting of the final $QFT^\dagger$ and the final fan-out.

We analyse each of these parts separately, starting with the multiply-addition. As can be seen from the figure, the overall depth will depend on the depth of M, which according to the next lemma is constant.

Lemma 2.1. *The quantum circuit M has depth 2 and requires $(k^3 + 3k^2 + 2k)/6$ each of AND gates, classically controlled phase shift gates, qubits.*

Proof. It can be seen from Figure 2.3 that the depth of the M circuit has to be equal to the maximum depth of any M_j sub-circuits, where $j \in \{1, \ldots, k\}$. By substituting the Q_j circuits in M_j, shown in Figure 2.2, with the one described in Figure 2.1, a circuit with one layer of classical AND gates and one layer of single-qubit R_m gates can be constructed. Thus the combined depth of any M_j and hence the M, circuit is 2.

Let $size(C)$ be the size of a quantum circuit C, *i.e.* the number of one- and two-qubit quantum gates in the decomposition of C. Every M_j sub-circuit in M corresponds to one $M_j(y, x)$ operator in the definition of $M(y, x)$. Furthermore, every Q_l sub-circuit in M_j corresponds to a $Q_l(y)$ operator in the definition of $M_j(y, x)$ (Eq. 2.15 and 2.16) and each R_m gate in Q_l corresponds to a R_m operator in the definition of $Q_l(y)$ (Eq. 2.13).

It can be seen from Eq. 2.13 that $size(Q_l) = l$ and the size of the circuit M is therefore

$$size(M) = \sum_{j=1}^{k} size(M_j) = \sum_{j=1}^{k} \sum_{l=1}^{j} size(Q_l) \tag{2.17}$$

$$= \sum_{j=1}^{k} \sum_{l=1}^{j} l = \sum_{j=1}^{k} \frac{j(j+1)}{2} = \frac{k^3 + 3k^2 + 2k}{6}. \tag{2.18}$$

The R_m gates are the only quantum gates in this decomposition and controlled by a classical bit. This classical bit needs to be computed via an AND gate as can be seen in Figure 2.2, thus we need an additional $(k^3 + 3k^2 + 2k)/6$ AND gates. Every one of the R_m gates acts on a distinct qubit and there is no need for qubits without a gate acting on it, thus the number of qubits equals the number of quantum gates in M. □

When determining the depth of a circuit, gates of variable size, such as the F gate have to be decomposed into one- and two-qubit quantum gates. An F_m gate can be written as an $O(\log m)$ depth circuit consisting of only $CNOT$ gates, where m is the number of qubits F_m acts upon. From Figure 2.3 it can be seen that the number of qubits F acts upon, is equal to the number of qubits M acts upon. This in turn is equal to the number of quantum gates in M since according to Lemma 2.2 there is only one layer of quantum gates. Thus the depth of the initialisation and finalisation parts depends on the number of gates in M as is proven in the next lemma.

Lemma 2.2. *The initialisation and finalisation sub-circuits in Figure 2.4 both have depth* $2k + \lceil \log(k^2 + k - 2) \rceil - 2$ *and require*

- $(k^3 + 3k^2 + 2k)/6$ *qubits,*
- $(k^3 + 3k^2 - 4k)/6$ *CNOT gates,*
- k *Hadamard gates,*
- $(k^2 - k)/2$ *two-qubit control-phase shift gates.*

Proof. First, the finalisation circuit is the conjugate transpose of the initialisation circuit and can be created by reversing the gate order of the latter and replacing every gate with its conjugate transpose. Hence the gate counts, depth and qubit requirement of these two sub-circuits is the same. The fan-out gate acting on a qubit j needs to be able to fan-out the state such that every phase shift gate in M_j can act on a distinct qubit. The fan-out circuit is comprised of CNOT gates and needs one CNOT gate per phase shift gate in M to achieve this, minus k since the k values from the QFT already have the correct state, *i.e.* $size(F) = size(M) - k = (k^3 + 3k^2 - 4k)/6$. The

depth of the fan out circuit depends on the largest fan-out operation in the circuit, which is $F_{(k^2+k)/2-1}$ for qubit k. This operator has a decomposition into CNOT gates which has depth logarithmic in its size [4], *i.e.* $\lceil \log(k^2+k-2) \rceil - 1$. The QFT contains k Hadamard gates and $(k^2-k)/2$ controlled phase-shift gates and has depth $2k-1$ (Section 1.2.3) resulting in a total depth of

$$2k + \lceil \log(k^2+k-2) \rceil - 2. \tag{2.19}$$

□

The overall properties of a hybrid circuit performing n MAC operations on k-bit integers can now be estimated.

Theorem 2.1. *There exists a hybrid quantum circuit with depth* $2n + 4k + 2\lceil \log(k^2 + k - 2) \rceil - 4$ *which performs* n *multiply additions of* k*-bit integers using*

- $(k^3 + 3k^2 + 2k)/6$ *qubits,*
- $n(k^3 + 3k^2 + 2k)/6$ *AND gates,*
- $n(k^3 + 3k^2 + 2k)/6$ *classically controlled phase-shift gates,*
- $2k$ *Hadamard gates,*
- $(k^3 + 6k^2 - 7k)/3$ *two-qubit gates.*

Proof. The circuit in Figure 2.4 consisting of the initialisation, finalisation and M sub-circuits performs the n multiply additions of k-bit integers. We can add together the requirements for each of these parts specified in Lemmas 2.1 and 2.2. The qubit requirement is $(k^3 + 3k^2 + 2k)/6$ — all of the used operations act on the same qubits, none of them require any new ones. The Hadamard gates exist only in the initialisation and finalisation parts, k in each. The initialisation and finalisation also contain the only two-qubit quantum gates, $(k^3 + 3k^2 - 4k)/6 + (k^2 - k)/2 = (k^3 + 6k^2 - 7k)/6$ each. The $(k^3 + 3k^2 + 2k)/6$ AND and classically controlled phase-shift gates will be required for each of the n multiply additions. The total depth will be the sum of the initialisation and finalisation ($2k + \lceil \log(k^2 + k - 2) \rceil - 2$ each) and n times the depth of M: $2n + 4k + 2\lceil \log(k^2 + k - 2) \rceil - 4$. □

Since the MAC part of this circuit acts on quantum Fourier transformed states we shall call this circuit the QFT multiply-accumulator (QMAC). In Table 2.1 we compare the QMAC with a classical MAC which uses the carry-save adder for addition, Kogge-Stone carry-lookahead adder for finalisation and the Dadda multiplier. We assume that the

Table 2.1: Comparison of the MAC and QMAC circuits. Note that the multiply-add part of the circuits can be reused with each new input pair, thus the number of gates used would not increase with n applications integers.

	Depth of adding n integers	Classical Gates	Quantum Gates	Total Gates	Qubits
MAC	$O(n \log k)$	$O(k^2)$	-	$O(k^2)$	-
QMAC	$O(n + k)$	$O(k^3)$	$O(k^3)$	$O(k^3)$	$O(k^3)$

Table 2.2: A detailed comparison of the MAC and QMAC circuits. Here we assume that k is a power of two.

	Depth of adding n integers	Total Gates
MAC	$3n(\lceil \log_{1.5} k/2 \rceil + 1) + 2 \log k + 2$	$6k^2 + 5k \log k + 30k + 18$
Initialisation	-	-
Multiply-add	$3\lceil \log_{1.5} k/2 \rceil + 3$	$6k^2 + 27k + 13$
Finalisation	$2 \log k + 2$	$5k \log k + 3k + 5$
QMAC	$2n + 4k + 2 \log(k^2 + k - 2) - 4$	$k^3 + 4k^2 + k$
Initialisation	$2k + \log(k^2 + k - 2) - 2$	$(2k^3 + 9k^2 + k)/6$
Multiply-add	2	$(k^3 + 3k^2 + 2k)/3$
Finalisation	$2k + \log(k^2 + k - 2) - 2$	$(2k^3 + 9k^2 + k)/6$

M circuit is reused for each input pair — a scenario which mimics the real world use of MAC circuits. It can be seen, that the QMAC's depth is $O(\log k)$ times smaller as for the classical MAC. A more detailed breakdown of the compared parameters is shown in Table 2.2. Note that the constants in the depth of the circuits depend on the fan-in of the individual gates, which for quantum circuits is the number of qubits they act on. For example, by allowing l qubit quantum gates, the decomposition of fan-out described in [4] can be done with depth $\log_l k$. This would reduce the overall depth of the QMAC circuit and in the case when non-constant fan-in gates of size n would be allowed, the depth of the fan-out would be reduced to 1.

2.2.1 Pipelining: a classical alternative

One well known and widely used method of speeding up computations in digital circuits is a *temporal parallelisation* (parallelising in time) method called *pipelining*, as is explained in [30]. Consider a Boolean circuit of depth d, which produces a result on each execution. The execution of this circuit would take d time steps, where a time step is the application time of a gate in the circuit. The gates in a single layer are applied only once and they would be idle for the rest of the $d-1$ time steps of the computation. Instead they can be utilised to process a new set of input values, coming from the previous layer each step. This way the last layer of the circuit could produce a new result every time step after it received its initial input and thus any circuit of depth d can output a new result each time step after the first d steps. [30]

The use of pipelining allows a depth $O(\log k)$ MAC circuit for k-bit integers to multiply-add n integer pairs in $O(n + \log k)$ time steps. The $\log k$ time steps are required for the first result and the n remaining results will be produced in the $O(n)$ remaining time steps. This implies that a classical pipelined MAC can produce the output of n MAC applications in less time steps than a QMAC (which has depth $O(n + k)$). This is regardless of the fact that the depth of the QMAC circuit is smaller than the $O(n \log k + \log k)$ depth of a MAC as proven previously.

As a comparison, let us consider how to pipeline a QMAC. The sub-circuit used for multiply-adding the inputs to the output register (shown in Figure 2.3) has depth two. The first layer consisting of classical AND gates and the second of classically controlled R_j gates. Thus the execution time of the QMAC can be reduced by at most two times through pipelining, which is negligible compared to the theoretical benefits of pipelining to a classical MAC, which was discussed above.

A pipelined circuit is more difficult to implement than a regular Boolean circuit due to the need of synchronising the pipelined gates [30]. Since the QMAC does not need pipelining, its implementation could be faster than that of a conventional MAC. This comparison between a pipelined MAC and QMAC could be explored in a possible continuation of this work. Even if the performance of QMAC would be inferior to a MAC, there exists at least one utilisation of the QMAC where the use of classical circuits would be impossible: quantum computers. Namely, the QMAC uses a quantum register for output, which allows its use as a sub-circuit in quantum algorithms where the product of two integers needs to be added to states in a superposition. Thus the QMAC circuit has future applications regardless of its performance compared to its classical counterparts.

Chapter 3

The QFT Adder

The QMAC introduced in the previous section could be used as an adder by choosing either x or y to be 1. Since one of the inputs is then constant, many of the gates will not have any effect regardless of the other input value. Indeed, as will be shown in this chapter it is possible to construct a QFT adder that requires $O(k^2)$ qubits and $O(k^2)$ gates compared to the $O(k^3)$ qubits and $O(k^3)$ gates in the QMAC while having roughly the same depth. Perhaps even more importantly, the size of entangled states used in this circuit is $O(k)$ compared to the $O(k^2)$ of the QMAC. In this construction, the same techniques as utilised in Chapter 2 are used to construct a QFT adder. This adder can be seen as a parallel out-of-place version of the Draper adder, optimised for sequential addition. The required modifications have not been considered before and result in a adder with distinct properties from the Draper adder — the QFT adder. From the point of view of parallelism, the most important characteristic of the new adder is the depth, which when applied to multiple numbers will be similar to the one obtained by using the carry-save adder.

3.1 The QFT Adder Circuit

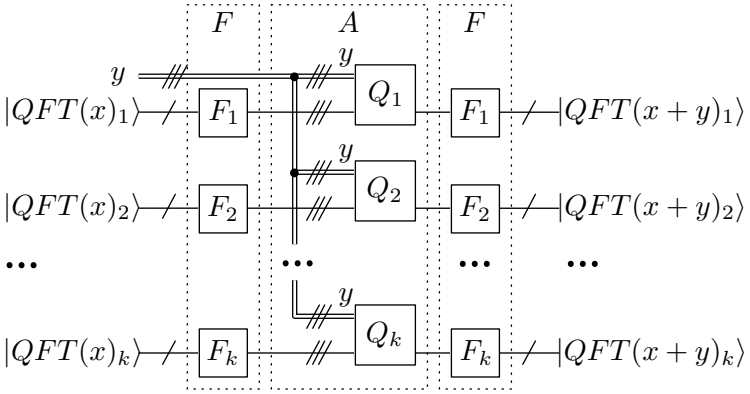


Figure 3.1: The parallelised Draper phase shift block. Note that by using the Q_j subcircuits we have replaced one quantum register with a classical one. See Figure 1.7 for circuit notation.

Consider the single-qubit quantum operator $Q_j(y)$ as defined in Equation 2.13 and the circuit Q_j in Figure 2.1 used to compute this operator. This is the operation performed in the Draper adder phase shift block (Figure 1.22) on qubit j if the $|y\rangle$ register is a computational basis state. Hence the phase shift gates in the Draper phase shift block acting on any single qubit j could be replaced with the parallelised $Q_j(y)$ operators shown in Figure 2.1. We will use A to denote the sub-circuit performing all these phase shifts in the parallelised adder:

$$A = Q_1 \otimes Q_2 \otimes \cdots \otimes O_k. \tag{3.1}$$

The resulting parallel Draper phase shift block is shown in Figure 3.1 and the whole adder in Figure 3.2.

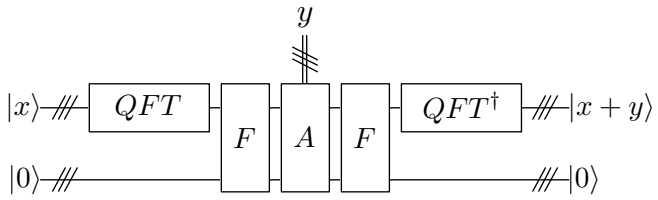


Figure 3.2: Step 1 of constructing the QFT adder: Replacing the quantum register $|y\rangle$ with a classical register y and parallelising the phase shift block.

Note that by using the $Q_j(y)$ operations we have changed one of the input registers from quantum to classical. This has two desirable side effects. First, the number of two-qubit gates is reduced by replacing the two-qubit gates with single-qubit gates controlled by classical input, which are simpler to implement than the two-qubit gates. Second, the

use of classical bits means that unbounded fan-out can be used — something that would not be possible for qubits. To add two k-bit integers x and y using the QFT adder, one of the summands needs to be transformed using the QFT. That the same result would be obtained by adding $0 + x + y$ using the circuit in Figure 3.3.

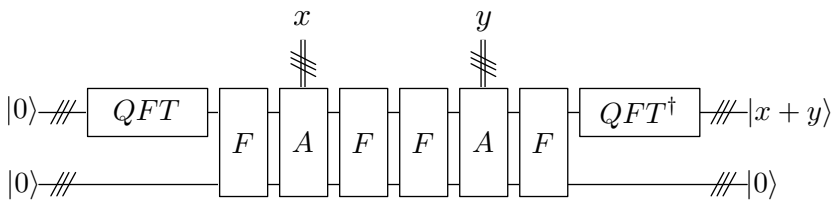


Figure 3.3: Step 2 of constructing the QFT adder: changing the adder from and in-place adder to an out-of-place adder.

Since the output register is distinct from the input registers, the adder in Figure 3.3 is an out-of-place version of the parallelised Draper adder in Figure 3.2. An important consequence of starting with the output register in state $|0\rangle$ is that the circuit can be further simplified. In particular, the state $QFT|0\rangle$ is equivalent to applying a single Hadamard gate to each qubit in the state $|0\rangle$ (see Section 1.2.3), leading to the simplified circuit in Figure 3.4.

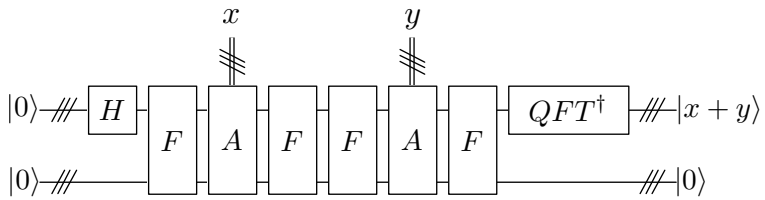


Figure 3.4: Step 3 of constructing the QFT adder: replacing the QFT with a single layer of Hadamard gates.

Finally, since $FF = FF^\dagger = I$ it is possible to remove the fan-out operations between the two phase shift blocks resulting in the final structure of the QFT adder shown in Figure 3.5.

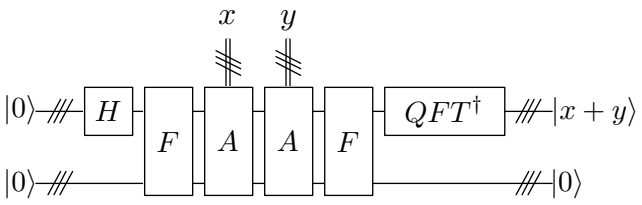


Figure 3.5: Step 4 of constructing the QFT adder: removing the cancelling fan-out gates.

Before following with the analysis of the QFT adder we point out the main differences

of the QFT and Draper adder:

- The inputs to the QFT adder are bitstrings whereas the Draper adder has quantum inputs and one of the input registers has to be quantum for the QFT to be applicable.
- The QFT adder is an out-of-place adder, whereby the Draper adder is an in-place adder.
- The QFT adders phase shift block is parallelised.

These differences are significant enough to result in distinct resource requirements and depth in these two adders.

3.2 Analysis of the Circuit

In additional to analysing the parallelism in the new adder through the depth of the circuit, the number of qubits and gates required and the size of the entangled states is also examined. We start with the physical resources needed to implement the proposed QFT adder, of which there are two main types: qubits to preserve and move the information and quantum gates to modify it. The use of classical bits as inputs reduces the number of qubits used, but to be able to parallelise the computation we need to add extra qubits. Overall, the number of qubits is larger than in the Draper adder, as is proven in the following lemma.

Lemma 3.1. *The QFT adder for k-bit integers requires $k(k+1)/2$ qubits.*

Proof. The number of qubits that are used in the QFT adder depends on the phase shift gates that are applied in the phase shift block — every one of these phase shift gates needs to be applied to a distinct qubit. Otherwise it would not be possible to apply them all in one step. The fan-outs are used to create the entanglement such that the phase shift gates can be applied simultaneously and hence don't introduce any additional qubits. The final $QFT^\dagger$ will only act on the k initial qubits after the entanglement is removed by the final fan-out layer and will also not require extra qubits. As can be seen from figure 2.1, there are exactly j phase shift gates applied for qubit j in the QFT adder, each to a separate qubit. Thus the total number of required qubits is

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}. \tag{3.2}$$

□

To perform any computations on the qubits, quantum gates must be applied to them. Since two-qubit gates are usually much more difficult to implement than single-qubit ones, the requirement of one, two, and classically controlled one qubit gates is pointed out individually.

Lemma 3.2. *The QFT adder requires* $2k$ *one qubit quantum gates,* $k(k+1)/2$ *classically controlled one qubit quantum gates, and* $3k(k-1)/2$ *two-qubit quantum gates. These gates are used in the initialisation, phase shift, and finalisation blocks as follows:*

- *Initialisation — k one qubit gates and* $k(k-1)/2$ *two-qubit gates.*
- *Phase shift —* $k(k+1)$ *classically controlled one qubit gates.*
- *Finalisation — k one qubit gates and* $k(k-1)$ *two-qubit gates.*

Proof. The initialisation step consists of a single layer of Hadamard gate followed by the fan-out operation. The Hadamard gates being the only single-qubit gates and the fan-out consisting of two-qubit CNOT gates. The number of Hadamard gates is equal to k, the number of bits in the integers we are adding together. Each of the Hadamard transformed qubits will then be fanned out using the CNOT gates for the simultaneous application of phase shift gates. The number of CNOT gates used for the fan-out of qubit j is equal to the number of phase shift gates in the operator $Q_j(y)$ minus one (since one gate can be applied to the initial qubit):

$$\sum_{i=1}^{k-1} i = \frac{k(k-1)}{2}. \tag{3.3}$$

The phase shift block in the QFT adder consists of gates $Q_1, Q_2, \ldots Q_k$ (Figure 3.1), consisting of $1, 2, \ldots k$ phase shift gates respectively (Figure 2.1). Thus the number of quantum gates in each of the two phase shift blocks, all of which are classically controlled single-qubit gates, is

$$\sum_{i=0}^{k} i = \frac{k(k+1)}{2}. \tag{3.4}$$

The finalisation block consists of the initialisation fan-out plus the final $QFT^\dagger$ operator to transform the result to the computational basis. As mentioned above, the fan-out operator needs $k(k-1)/2$ two-qubit gates. From Section 1.2.3, we know that $QFT^\dagger$

uses k Hadamard gates and $k(k-1)/2$ two-qubit controlled phase shift gates. Thus in total the finalisation block required k single-qubit and $k(k-1)$ two-qubit gates. □

We have already estimated the number of gates and qubits required for the QFT adder. Two-qubit gates can be used to create entanglement between qubits, which is another important measure of quantum circuits. In many quantum computation implementations it is difficult to create and/or keep entanglement. The parallelism of the QFT adder helps to counter the problem of keeping entangled states coherent, by allowing the computation to finish sooner.

Lemma 3.3. *The largest entangled state in the QFT adder consists of k qubits.*

Proof. As can be seen in Figure 3.1, no entanglement will be created between the fanned out output qubits — there are no two-qubit gates acting between the distinct fanned out parts. The only entangling operators are the fan-out operators F_j. Since the largest number of qubits a fan-out operator in the QFT adder acts on is k, the largest entangled state in the circuit consists of k qubits. □

To compute the depth of the QFT adder, first the depth of the sub-circuit A needs to be estimated.

Lemma 3.4. *The depth of the phase shift block A in the QFT adder is 1.*

Proof. As can be seen in Figure 3.1, the phase shift operation consists of only simultaneously applicable $Q_j(y)$ blocks. Since each of the $Q_j(y)$ blocks has depth is 1 (Figure 2.1), the overall depth of A is also 1. □

Lemma 3.5. *The QFT adder can be used to add two k-bit integers in $2k+2\lceil\log k\rceil+2$ depth.*

Proof. As can be seen in Figure 3.5, the QFT adder for adding two k-bit integers consists of one layer of Hadamard gates (depth 1), two fan-out operators (with the largest one having $\lceil\log k\rceil$ each), two phase-shift blocks (depth 1 each) and a final $QFT^\dagger$ circuit (depth $2k-1$). Thus the total depth will be

$$1 + 2\lceil\log k\rceil + 2 + 2k - 1 = 2k + 2\lceil\log k\rceil + 2. \tag{3.5}$$

□

The depth required to compute the sum of two k-bit integers is $O(k)$ for both Draper adder and the new QFT adder. The depth of the Draper adder's phase shift block is one

of the dominating terms in the overall depth. In the QFT adder the depth of the phase shift block is 1 (Lemma 3.4) and the depth is dominated by the $O(k)$ depth finalisation step. When two QFT adders are applied in sequence, the last F_j gates of the first, and the first F_j gates of the second adder will cancel out since $F_j F_j = F_j F_j^\dagger = I$. This makes the QFT adder especially suitable for numerous sequential applications — each subsequent application of the QFT adder increases the overall depth of the quantum circuit by only 1. This is shown in Figure 3.6 and is formalised in the following theorem.

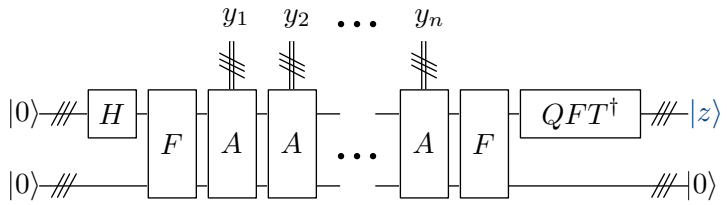


Figure 3.6: Applying the QFT adder n times in a sequence. Here $z' = z + \sum_{i=1}^{n} y_i$.

Theorem 3.1. *The QFT adder can be used to construct an $n + 2k + 2\lceil \log k \rceil$ depth circuit for adding n integers of k bits.*

Proof. As can be seen in Figure 3.6, the circuit for the n time application of the QFT adder simplifies to one layer of Hadamard gates (depth 1), n phase shift blocks (each having depth 1), an initial and final fan-out (the depth of the largest F_j being $\lceil \log k \rceil$ for both), and a final $QFT^\dagger$ (depth $2k - 1$) operation. Thus the total depth will be

$$1 + n + 2\lceil \log k \rceil + 2k - 1 = n + 2k + 2\lceil \log k \rceil. \tag{3.6}$$

□

Although the depth increases with the application of n QFT adders, since multiple application of the QFT adder requires only the repetition of the phase shift block, we do not need any extra qubits — the number of qubits used adding any number of integers is always $O(k^2)$. Moreover, we could reuse the phase-shift block A and thus the implementation size would only depend on k. Another adder having small depth when adding multiple numbers is the carry-save adder. In Table 3.1 we compare the QFT adder with the Draper and classical and quantum carry-save adders. From Table 3.1 we see that the QFT Adder does not have any benefit over the quantum carry-save adder when adding n integers — It uses more qubits and gates ($O(k)$ vs $O(k^2)$) and has a larger depth ($O(n + \log k)$ vs $O(n + k)$). When we look at the more detailed

Table 3.1: Comparison of the QFT adder with carry-save and Draper adders. The acronym CSA is used to denote a CSA in the table. The addition part of the circuits can be reused with each new integer, thus the number of gates used would not increase with adding multiple integers.

	Depth of adding n integers	Single-Qubit Gates	Double Qubit Gates	Total Gates	Qubits
CSA	$O(n+\log k)$	-	-	$O(k \log k)$	-
Quantum CSA	$O(n+\log k)$	-	$O(k)$	$O(k)$	$O(k)$
Draper Adder	$O(kn)$	$O(k)$	$O(k^2)$	$O(k^2)$	$O(k)$
QFT Adder	$O(n+k)$	$O(k^2)$	$O(k^2)$	$O(k^2)$	$O(k^2)$

breakdown of the circuits (Table 3.2) we see that there are reasons for choosing the QFT adder over the quantum carry-save adder. Most importantly, Table 3.2 shows the depth for each individual addition is considerably smaller in the QFT adder, even if Toffoli gates are allowed in the circuit. In particular, when adding n integers, then for each integer the depth required increases by 1 for the QFT adder, by 8 for the quantum carry-save adder with Toffoli gates, and by 24 for quantum carry-save adder with only one and two-qubit gates. The execution time of a circuit, and thus also the time the quantum states need to be coherent, depends on the depth of the circuit, hence it might in some implementations be beneficial to use the QFT adder. It should also be noted that for the integer sizes used commonly in current processors (64 bit), the QFT adder uses about four times more gates than the quantum carry-save adder (8256 versus 2442). Although both of these numbers are too large to be implementable using current technology, the four fold difference is not as large as it could be since the constants in the quantum carry-save adder are rather large. In the end the choice of adder will depend largely on the limitations set by the architecture, much like in the current classical digital circuits.

Table 3.2: A detailed breakdown of the properties of various adders when used on multiple integers. CSA is used to denote a carry save adder and QCSA a quantum carry-save adder. The QCSA (Toffoli) is the QSCA where Toffoli gates are allowed in addition to two-qubit gates. The finalisation in the carry-save adder is performed by a carry-lookahead adder.

	Depth of adding n integers	Total Gates	Qubits
CSA	$3n + 2\log k + 2$	$5k\log k + 8k + 5$	-
Initialisation	-	-	-
Addition	3	$5k$	-
Finalisation	$2\log k + 2$	$5k\log k + 3k + 5$	-
QCSA	$24n + 2\log k + 2$	$37k - 15\log k - 16$	$6k - \log k$
Initialisation	-	-	-
Addition	24	$24k$	$4k$
Finalisation	$2\log k + 2$	$13k - 15\log k - 16$	$4k - \log k$
QCSA (Toffoli)	$8n + 2\log k + 2$	$13k - 3\log k - 4$	$6k - \log k$
Initialisation	-	-	-
Addition	8	$8k$	$4k$
Finalisation	$2\log k + 2$	$5k - 3\log k - 4$	$4k - \log k$
Draper Adder	$kn + 3k - 1$	$3(k^2 + k)/2$	$2k$
Initialisation	$2k - 1$	$(k^2 + k)/2$	$2k$
Addition	k	$(k^2 + k)/2$	$2k$
Finalisation	$2k - 1$	$(k^2 + k)/2$	$2k$
QFT Adder	$n + 2k + 2\log k$	$2k^2 + k$	$(k^2 + k)/2$
Initialisation	$1 + \log k$	$(k^2 + k)/2$	$(k^2 + k)/2$
Addition	1	$(k^2 + k)/2$	$(k^2 + k)/2$
Finalisation	$2k - 1 + \log k$	k^2	$(k^2 + k)/2$

Chapter 4

Implementing the QFT Arithmetic Circuits

In this chapter we analyse the various implementation issues of the QFT arithmetic circuits proposed in the previous two chapters and present optimisation that could alleviate some of them. We do not consider any particular architecture, but instead present a number of optimisations that can be applied to different parts of the QFT Arithmetic circuits. Each of these optimisations has some disadvantages along with the advantages and might not be always suitable to implement, thus they were not included as part of the main QFT arithmetic circuits. The techniques themselves are presented in Sections 4.1 through Section 4.3. After we introduce the implementation optimisations, we'll also show in Section 4.4 how these can be applied to the two-qubit version of the QMAC circuit and give the fully optimised quantum circuit for it. The two-qubit implementation is also the minimal proof of concept implementation that utilises the parallelism of the QMAC.

Implementing quantum circuits is not easy for a number of various reasons. The QFT arithmetic circuits have four major implementation concerns:

1. The size of the entangled states used.
2. The number of auxiliary qubits required.
3. The number of two-qubit gates required.
4. The number of different different phase shift gates.

Parallelising computations usually requires adding resources to be able to perform a number of steps simultaneously. In the QFT arithmetic circuits, the parallelism is

obtained by dispersing the data into entangled states and working on this new state. Thus the entangled states are the core resources of the QFT arithmetic circuits. We can reduce the size of the entangled states used, and therefore the amount of resources we used for parallelisation, by sacrificing parallelism. Instead of applying all the phase shift gates in one step, we could apply them in 2, 3, 4, etc. steps. By doing so we reduce the number of qubits (and the size of entangled states) we need. It is easy to see, that for k-bit QFT arithmetic circuits, executing the phase shift sub circuits in n steps reduces the size of entangled states used to $O(k/n)$ for the adder and $O(k^2/n)$ for the QMAC. Since the auxiliary qubits are required only for these entangled states, the number of total qubits in the circuit also changes to $O(k^2/n)$ and $O(k^3/n)$ correspondingly. Thus it is possible to reduce the number of qubits and the size of entangled states in the circuit by sacrificing parallelism.

Considering the number of two-qubit gates in the circuit, there exist optimisations that allow us to remove some of them from the circuit. A crucial part of the QMAC circuit is the QFT at the beginning and QFT† at end of the computation. Implementing a QFT is not an easy task itself and might be a hinderance in implementing the proposed circuit. The circuit of QFT requires two-qubit controlled R_i gates which like most two-qubit quantum gates are more difficult to implement than single-qubit gates. However, the QMAC circuit introduced in this work does not require a full QFT implementation. In Section 4.1 we show how the initial QFT can be replaced with the creation of *Greenberger—Horne—Zeilinger* (GHZ) states (usually a much simpler operation) and in Section 4.2 we describe how the final QFT can be replaced with a semiclassical QFT. By doing so we remove the need of two-qubit controlled R_i gates and thereby reduce the total number of two-qubit gates in the circuit.

Although the two-qubit controlled R_i gates can be avoided, the QFT arithmetic circuits still require R_i gates for i values in $\{1, 2, \ldots, k\}$. Note that $R_{i+1}R_{i+1} = R_i$. Hence if the R_k gates are implementable, it is possible to decompose all the different R_i gates into sequences of R_k gates. Unfortunately, this replacement sacrifices the depth of the circuit and it is not certain that R_k gates are more easily implementable than R_i gates for $i < k$.

4.1 Initialisation

Consider the initial QFT of the QFT arithmetic circuits in Figures 2.4 and 3.6. Instead of setting the quantum input register to $|z\rangle$, its is advantageous to set the register to to $|0\rangle^{\otimes k}$. Then computing the $QFT|0\rangle^{\otimes k}$ and performing an $M(z, 1)$ or $A(z)$ operation

will result in the states $QFT|0 + z \cdot 1\rangle$ and $QFT|0 + z\rangle$ correspondingly. Hence, for the initialisation it is enough to be able to compute $QFT|0\rangle^{\otimes k}$. This can be done by applying a Hadamard gate to all of the k qubits since

$$QFT|0\rangle^{\otimes k} = 2^{k/2} \sum_{i=0}^{2^k-1} |i\rangle = H^{\otimes k}|0\rangle^{\otimes k}, \tag{4.1}$$

which is a much simpler to implement than a full QFT circuit. The only downside is that this optimisation allows only to add to computational basis states. *I.e* the output register cannot be in a superposition, as is possible without this optimisation. Having the output register in a superposition might be desirable when the QMAC is used as a sub-circuit in a larger algorithm. If it is known that the numbers are never added to a superposition state, this optimisation should always be performed.

There might be an additional benefit in applying the above optimisation. The QFT arithmetic circuits require relatively large entangled states. Considering what is possible using current technology, creating entanglement is still a very difficult task. One of the most well studied and often experimentally realised entangled state is the GHZ state [62]. The GHZ state $|GHZ_l\rangle$ is a l qubit entangled quantum state

$$|GHZ_l\rangle = \frac{|0\rangle^{\otimes l} + |1\rangle^{\otimes l}}{\sqrt{2}}. \tag{4.2}$$

It is easy to see, that the initial state $QFT|0\rangle^{\otimes k}$ followed by the fan-out operation in the QFT arithmetic circuits creates k GHZ states. The state of the quantum registers in the QFT adder after the initial fan-out is therefore

$$|GHZ_1\rangle \otimes |GHZ_2\rangle \otimes \cdots \otimes |GHZ_k\rangle, \tag{4.3}$$

and the state of the quantum registers in the QMAC after the fan-out is

$$|GHZ_1\rangle \otimes |GHZ_3\rangle \otimes \cdots \otimes |GHZ_{k(k+1)/2}\rangle. \tag{4.4}$$

There have been successful experiments in creating multi-qubit GHZ states. In particular a 8 qubit GHZ has been was created using photonics [63] and 14 qubit state with trapped ions [64]. Since GHZ states have been experimentally realised on multiple architectures, the fact that GHZ states are the entangled states required in the QFT arithmetic circuits could lend to the possibility of implementing these circuits (or at least the initialisation part of them).

4.2 The Semiclassical QFT

Consider the QFT† at the end of the computation in the QFT Adder (Figure 3.6) and QMAC (Figure 2.4). As was shown in [65] the QFT can be implemented semiclassically using only measurements and single-qubit gates that depend on the classical outcomes of these measurement. This simplification cannot be used in the most general case where the output of the QFT would be used as input to another quantum circuit since the measurements will destroy the quantum state. It can be used if the outcome of the QFT arithmetic circuits is measured at the end of the computation or is always a computational basis state. This allows for a much simple implementation of the final QFT by removing the need for two-qubit quantum gates. The circuit of the semiclassical QFT is depicted in Figure 4.1. Note that since the semiclassical QFT

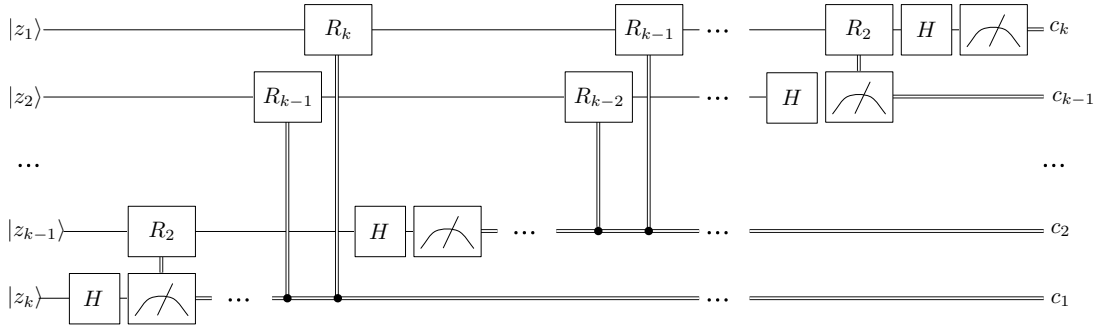


Figure 4.1: The circuit of a semiclassical QFT. The outputs are classical bits c_i, with $P(c_i = 1) = \|\langle 1|QFT(z)\rangle_i\|^2$. These classical bits control the application of the quantum phase shift gates R_i. Contrary to the general QFT (Figure 1.13), the semiclassical QFT does not contain any two-qubit quantum gates.

is just the general QFT where the two-qubit gates replaced with measurement and classically controlled single-qubit gate, the depth of the semiclassical QFT is equal to the depth of the general QFT. Thus, if the output of the QFT arithmetic circuits is a computational basis state, the semiclassical QFT should be used instead of the general QFT. For example, this would be the case if the optimisations ofae Section 4.1 were applied.

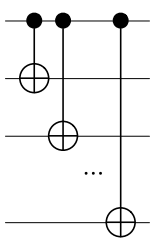
4.3 The Final Fan-Out

The finalisation part of the QFT Arithmetic circuits can be optimised even further. Namely, it is possible to replace the final fan-outs with measurements and controlled NOT gates depending on these measurement outcomes similarly to the replacement

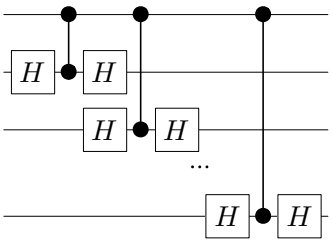
of QFT with semiclassical QFT described in the previous section. Since in this fan-out optimisation the auxiliary qubits are measured, it is applicable only if the auxiliary qubits would be discarded after the computation. This is a reasonable assumption since creating a new $|0\rangle$ state for the auxiliary qubits might be more efficient than reusing the existing one by cleaning it up using the fan-out. Thus this optimisation would be desirable on most future implementations. Figure 4.2 depicts each intermediate step of replacing the fan-out. We start with a regular fan-out circuit decomposed into two-qubit CNOT gates 4.2(f). Each of these CNOT gates can be replaced with one $\wedge Z$ gate and two Hadamard gates (Figure 4.2(b)). Since we are not interested in auxiliary qubits, we can just discard them at the end of the computation without needing them to be cleaned up to the $|0\rangle$ state. The final Hadamard gates can therefore be removed from the circuit since, albeit being necessary for cleaning up the auxiliary, they don't have any effect on the measurement of the non-auxiliary qubit we are interested in (Figure 4.2(d)). Now the $\wedge Z$ gates can be replaced with measurements and single-qubit gates based on the outcome of this measurement as explained in Section 1.2. Since $Z = Z^{\dagger}$, we have that $Z^{x_1}Z^{x_2}\cdots Z^{x_i} = Z^{x_1 \oplus x_2 \oplus \cdots \oplus x_i}$, where $x_1, x2, \ldots, x_i \in \{0, 1\}$, and can replace the controlled-Z gates with just one Z gate controlled by the XOR of all the measurements (Figure 4.2(f)). This optimisation will not increase the overall depth of the circuit, since the XOR of k bits can be computed using a $O(\log n)$ tree of XOR gates with fan-in two. This depth is the same as the the depth of the initial fan-out circuit [4].

4.4 The Optimised Two-Qubit QMAC

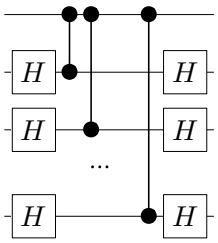
The full circuit of the two bit QMAC on Figure 4.3 is the smallest implementation that could demonstrate the parallelism in QMAC. We have applied all the optimisations described in this chapter, thus the circuit requires a pair of two-qubit gates to create the initial GHZ state. Since this is a small circuit, we can verify its correctness by computing the values of each output qubit. This verification is motivated by discussion with experimentalists who were interested in implementing the two qubit QMAC (in addition to the general proofs presented in the previous chapters). They were interested in an explicit proof that the circuit after all these optimisations would compute the MAC as claimed. This section could be of interest to other experimentalists considering implementing the QMAC. The two classically controlled Z gates acting at the end of the circuit on qubit 2 can be replaced by one Z gate controlled by the XOR of the measurement outcomes of qubits 3 and 4 (Section 4.3). We used two Z gates to simplify the verification.



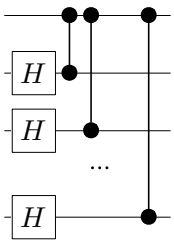
(a) A fan-out circuit.



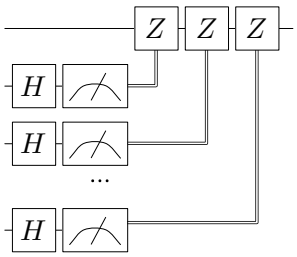
(b) Replacing the $CNOT$ gates with $\wedge Z$ and Hadamard gates.



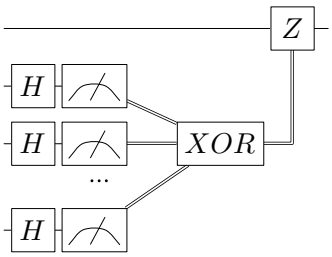
(c) The Hadamard gates can be applied in one layer at the beginning and end of the circuit.



(d) Discarding gates that don't have an effect to the first qubit.



(e) Replacing the controlled quantum gates with measurements and classical control.



(f) It is enough to have a single classically controlled Z gate.

Figure 4.2: Replacing the fan-out circuit with measurements and controlled single-qubit gates.

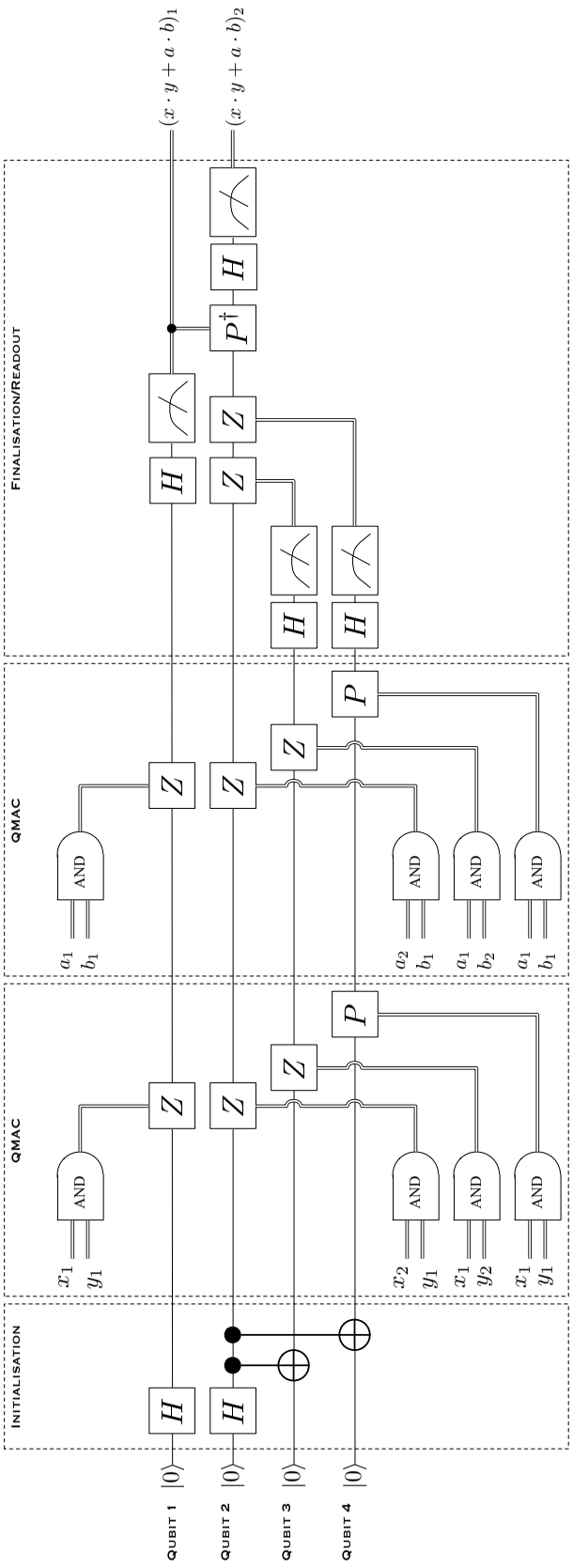


Figure 4.3: The QMAC circuit with all the optimisations from this chapter applied.

4.4.1 Verifying the circuit

Following is the proof that the circuit in Figure 4.3 implements the multiply-addition operation (MAC) of a pair of two-bit integers integers x and y. Note that after applying the first Hadamard gates in Figure 4.3 the state of the output qubits is $QFT|0\rangle$. The final output of the circuit should be a two bit number $0+x\cdot y$. For generality we assume that the state is instead initialised to

$$QFT|z\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.z_1}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.z_2 z_1}|1\rangle), \tag{4.5}$$

where z is a 2 bit integer and in Figure 4.3 the value of both z_1 and z_2 is 0. We show that then the application of the QMAC and Finalisation parts of the circuit in Figure 4.3 results in the application of a MAC, *i.e.* the first and second output bits will be correspondingly

$$(z + x \cdot y)_1 = z_1 \oplus x_1 \cdot y_1, \tag{4.6}$$

$$(z + x \cdot y)_2 = z_2 \oplus z_1 \cdot x_1 \cdot y_1 \oplus x_1 \cdot y_2 \oplus x_2 \cdot y_1. \tag{4.7}$$

Lets first consider the operations on qubit 1.

$$\begin{aligned} HZ^{y_1 x_1}\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.z_1}|1\rangle) &= H\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{y_1\cdot x_1}e^{2\pi i 0.z_1}|1\rangle) \\ &= H\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.z_1}e^{2\pi i 0.y_1\cdot x_1}|1\rangle) \\ &= H\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i (0.z_1 + 0.y_1\cdot x_1)}|1\rangle) \\ &= H\frac{1}{\sqrt{2}}(|0\rangle + e^{\pi i (z_1 + y_1\cdot x_1)}|1\rangle) \\ &= H\frac{1}{\sqrt{2}}(|0\rangle + e^{\pi i (z_1 \oplus y_1\cdot x_1)}|1\rangle) \\ &= H\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{z_1 \oplus y_1\cdot x_1}|1\rangle) \\ &= \neg z_1 \oplus y_1 \cdot x_1|0\rangle + z_1 \oplus y_1 \cdot x_1|1\rangle. \end{aligned} \tag{4.8}$$

Measuring qubit 1 gives thus $|1\rangle$ if and only if $z_1 \oplus y_1 \cdot x_1 = 1$, which is equal to $(z + x \cdot y)_1$ just as required. Now lets consider the operations on qubits 2, 3 and 4 which before the QMAC block are in the state $1/\sqrt{2}(|000\rangle + e^{2\pi i 0.z_2 z_1}|111\rangle)$. The QMAC block itself is the operator $P^{y_1 x_1} \otimes Z^{y_2 x_1} \otimes Z^{y_1 x_2}$, hence after the QMAC block the state is

$$
\begin{aligned}
& P^{y_1x_1} \otimes Z^{y_2x_1} \otimes Z^{y_1x_2} \frac{1}{\sqrt{2}}(|000\rangle + e^{2\pi i 0.z_2z_1}|111\rangle) \qquad (4.9)\\
=&\frac{1}{\sqrt{2}}(|000\rangle + e^{2\pi i 0.z_2z_1} e^{2\pi i 0.0y_1\cdot x_1} e^{2\pi i 0.y_2\cdot x_1} e^{2\pi i 0.y_1\cdot x_2}|111\rangle)\\
=&\frac{1}{\sqrt{2}}(|000\rangle + e^{2\pi i 0.z_2z_1} e^{2\pi i 0.y_2y_1\cdot x_1} e^{2\pi i 0.y_1\cdot x_2}|111\rangle)\\
=&\frac{1}{\sqrt{2}}(|000\rangle + e^{2\pi(i 0.z_2z_1+0.y_2y_1\cdot x_1+0.y_1\cdot x_2)}|111\rangle).
\end{aligned}
$$

Let $p = e^{2\pi i(0.z_2z_1+0.y_2y_1\cdot x_1+0.y_1\cdot x_2)}$, then the application of $I \otimes H \otimes H$ before the measurement of qubits 3 and 4 results in

$$
\begin{aligned}
& I \otimes H \otimes H \frac{1}{\sqrt{2}}(|000\rangle + p|111\rangle) \qquad (4.10)\\
=&I \otimes I \otimes H \frac{1}{2}(|000\rangle + |010\rangle + p|101\rangle - p|111\rangle)\\
=&\frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + p|100\rangle - p|101\rangle - p|110\rangle + p|111\rangle).
\end{aligned}
$$

When qubit 4 is measured and, based on the outcome, a Z gate is applied to the first qubit, the resulting state will be

$$
\begin{aligned}
& \frac{1}{4}(|00\rangle + |00\rangle + |01\rangle + |01\rangle + p|10\rangle + p|10\rangle - p|11\rangle - p|11\rangle) \qquad (4.11)\\
=&\frac{1}{2}(|00\rangle + |01\rangle + p|10\rangle - p|11\rangle).
\end{aligned}
$$

Now measuring qubit 3 and based on the result applying a Z gate to the first qubit results in the state

$$
=\frac{1}{2\sqrt{2}}(|0\rangle + |0\rangle + p|1\rangle + p|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + p|1\rangle). \qquad (4.12)
$$

Finally the $P^\dagger$ gate is applied depending on the result of the measurement on the first

qubit $(z_1 + x_1 \cdot y_1)$:

$$
\begin{aligned}
& P^{\dagger z_1 \oplus x_1 \cdot y_1} \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_2 z_1 + 0.y_2 y_1 \cdot x_1 + 0.y_1 \cdot x_2)}|1\rangle) \qquad (4.13)\\
=& \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_2 z_1 + 0.y_2 y_1 \cdot x_1 + 0.y_1 \cdot x_2)} e^{-2\pi i(0.0 z_1 \oplus 0.0 y_1 \cdot x_1)}|1\rangle)\\
=& \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_2 + 0.y_2 \cdot x_1 + 0.y_1 \cdot x_2)} e^{2\pi i(0.0 z_1 + 0.0 y_1 \cdot x_1)} e^{-2\pi i(0.0 z_1 \oplus 0.0 y_1 \cdot x_1)}|1\rangle)\\
=& \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_2 + 0.y_2 \cdot x_1 + 0.y_1 \cdot x_2)} e^{2\pi i 0.z_1 \cdot y_1 \cdot x_1} e^{2\pi i(0.0 z_1 \oplus 0.0 y_1 \cdot x_1)} e^{-2\pi i(0.0 z_1 \oplus 0.0 y_1 \cdot x_1)}|1\rangle)\\
=& \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_2 + 0.y_2 \cdot x_1 + 0.y_1 \cdot x_2 + 0.z_1 \cdot y_1 \cdot x_1)}|1\rangle).
\end{aligned}
$$

The application of the last gate of the circuit (a Hadamard gate) to qubit 2 in the above state results in

$$
\begin{aligned}
& H \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.z_2 + 0.y_2 \cdot x_1 + 0.y_1 \cdot x_2 + 0.z_1 \cdot y_1 \cdot x_1)}|1\rangle) \qquad (4.14)\\
=& \neg z_2 \oplus z_1 \cdot x_1 \cdot y_1 \oplus x_1 \cdot y_2 \oplus x_2 \cdot y_1 |0\rangle + z_2 \oplus z_1 \cdot x_1 \cdot y_1 \oplus x_1 \cdot y_2 \oplus x_2 \cdot y_1 |1\rangle,
\end{aligned}
$$

which when measured gives the outcome $|1\rangle$ exactly if and only if $z_2 \oplus z_1 \cdot x_1 \cdot y_1 \oplus x_1 \cdot y_2 \oplus x_2 \cdot y_1 = 1$. Therefore the outcome of the measurement of qubit 2 corresponds to the output of the second qubit of a MAC circuit (Equation 4.7) and thus the circuit in Figure 4.3 is correct.

Chapter 5

Discussion and Results

We presented two new quantum arithmetic circuits utilising the QFT — the QMAC (Section 2.1) and the QFT adder (Section 3.1). The former is the first MAC designed for quantum computers and enables the application of n MAC operations on k-bit integers in $O(n + k)$ parallel steps (Theorem 2.1). Classically, the depth of a MAC operation is $O(n \log k)$. This is the first time that an elementary arithmetic operation, which is very widely used, could benefit from being executed on a quantum device. However, it is possible to achieve the same theoretical performance classically using pipelining as discussed in Section 2.2.1. How exactly the performance of the QMAC compares with a pipelined MAC will depend on its implementation.

The QMAC circuit realises a very specific computational pattern, the MAC. This makes it suitable for use as an execution unit in a hybrid CPU, DSP, or even as a separate Quantum accelerator device. Moreover, the fact that implementing this circuit does not require a full quantum computer makes it more likely to be realisable in the near future. The small depth of the QMAC is a consequence of using the QFT, a highly entangled quantum state and classical fan-out, that is, copying of bits. First, since the MAC operation is performed on the QFT state, only diagonal gates are necessary. This makes it possible to entangle the quantum register with auxiliary qubits in a way that allows the simultaneous application of every single-qubit quantum gate. Second, the states of a bit can be copied by using multiple output wires to more than two registers for the next computational step. Thus the information propagates in one step to all the quantum gates controlled by these bits. This can be interpreted as influencing the state of an unbounded number of qubits with just one fan-out operation.

The QFT adder we introduced has the same structure than the QMAC — it consists of initialisation and finalisation QFT and fan-out sub-circuits and an intermediate

addition block, which can be repeated for addition of multiple numbers. The adder has depth $O(n+k)$, as does the QMAC, when applied to n inputs and requires $O(k^2)$ quantum gates (Theorem 2.1), but unlike the MAC operation there exists a classical adder, the carry-save adder, that has a lower depth $O(n+\log k)$ (Section 1.3.1). The quantum carry-save adder has depth $O(n+\log k)$ and requires $O(k)$ gates [41] (see also Section 1.3.1). At first, it seems that using the quantum carry-save adder is a better choice compared to the QFT adder. However, since the constants in these estimates are larger for the quantum carry-save adder (see Table 3.2), it can be beneficial to use a QFT adder in some instances, for example, with small input sizes (including the 32 and 64 bits used in current processors). A comparison of the properties of the QFT and carry-save adder (Table 3.2) reveals that in a sum of multiple integers, the depth of QFT adder increases by 1 for each integer, whereas for the quantum carry-save adder the increase in depth is 8 (if Toffoli gates are allowed) or 24 (if only two-qubit gates can be used). Thus when the number of summed integers is large enough that the $O(k)$ and $O(\log k)$ terms in the total depth are negligible, the QFT adder would have roughly 24 times smaller depth than the quantum carry-save adder (or 8 times smaller if Toffoli gates can be used).

For classical circuits, low depth is desirable for accelerating computations, but a higher depth circuit is still relisable. For quantum circuits, the depth of the circuit affects also its implementability due to decoherence, *i.e.* over time quantum states lose the information stored in them. The benefit of low-depth circuits is that the operations can be performed in parallel, thus reducing the total time required to execute the circuit (compared to circuits with higher depth). Thus low depth parallel circuits, such as the QMAC and QFT adder, can mitigate the loss of information due to decoherence since the quantum states need to be coherent for a shorter time.

Future work is needed to study how and if the hybrid QFT arithmetic circuits could be adopted to floating point operations, which are used in most of the time-intensive computations. This would greatly increase the number of problems which would benefit from quantum devices. Another direction would be to consider hybrid circuits for other arithmetic operations, for example division, and examine at how the different circuits can be combined together. The QMAC introduced in this paper has a lower depth than a classical MAC only if it is applied in a sequence. Hence combining different quantum arithmetic operators could result in an improved depth compared with classical circuits.

In addition of providing a comparison with existing classical circuits and other quantum circuits we provided some guidelines how they could be optimised with the intent of simplifying the decision of which adder or MAC to choose. The implementation

optimisations presented is Chapter 4 (and summarised in Table 5.1) focus on removing two-qubit gates from the QFT arithmetic circuits. Note that if GHZ states could be created in constant depth, then the optimised QFT arithmetic circuits wouldn't need any two-qubit quantum gates. This is a very desirable property, since two-qubit gates are considerably more difficult to implement than single-qubit ones. From implementation point of view, there are still at least three possible obstacles in implementing QFT arithmetic circuits of useful (32 or 64 bits) size. The most obvious ones are the number of required qubits (QFT adder $O(k^2)$ an and QMAC $O(k^3)$) and the number of qubits needed to be entangled at a time: $O(k)$ and $O(k^2)$ for the QFT adder and QMAC respectively. In addition, implementing the single-qubit phase shift gates might be an issue. Namely for k-bit QFT arithmetic circuits phase shift gates performing phase shifts with k different magnitudes are required. On the other hand approximating the phase gates (for example using the Solovay-Kitaev [66] or more recent algorithms from [67, 68, 69]), would increase the depth of the circuits and nullify the main benefit of our circuits. The choice of which arithmetic circuits to implement will eventually depend on the architecture which is used to implement it.

One possible continuation of this work would is a collaboration with experimentalists to implement the proof of concept circuit presented in Section 4.4. First, to explore the possibility to implement the QFT arithmetic circuits for further larger implementations in full quantum computers. Second, such experiment would give some initial, albeit very rough, estimates about the possible performance of the QMAC allowing detailed comparisons with classical circuits. It would give more isight into whether this quantum circuit could outperform a classical pipelined MAC (see discussion on pipelining in Section 2.2.1) and thus reveal its potential an accelerator in classical computers.

Table 5.1: Summary of possible optimisations to the QFT Arithmetic circuits.

		Initialisation using Hadamard gates (Section 4.1)	Semiclassical QFT (Section 4.2)	Measurements instead of fan-out (Section 4.3)
Removed Gates	QFT adder	$(k^2-k)/2$ two qubit controlled R_i gates.	$(k^2-k)/2$ two qubit controlled R_i gates.	$(k^2-k)/2$ CNOT gates.
	QMAC	$(k^2-k)/2$ two qubit controlled R_i gates.	$(k^2-k)/2$ two qubit controlled R_i gates.	$(k^3-k)/6$ CNOT gates.
Added Gates	QFT adder	-	(k^2-k) classically controlled R_i gates.	$(k^2-k)/2$ Hadamard gates, k classically controlled Z gates, $(k^2-k)/2$ classical XOR gates.
	QMAC	-	$(k^2-k)/2$ classically controlled R_i gates.	$(k^3-k)/6$ Hadamard gates, k classically controlled Z gates, $(k^3-k)/2$ classical XOR gates.
Change in depth		Reduced by k.	-	-
Additional benefits		The quantum register can be initialised to a GHZ state.	-	-
Limitations		The accumulation can only be done only by adding to a classical value.	The output is a classical bit string.	The ancillae cannot be reused.

Part II

Measurement Based Quantum Computing

The quantum circuit model used in Part I is one of many models for quantum computing. In this chapter we concentrate on a model where the computation is driven by adaptive measurements of a quantum state. Such models of computation are called the Measurement Based Quantum Computing (MBQC) models and in this thesis we focus on the One Way Quantum Computing (1WQC) model described in Chapter 6. From parallel computing perspective, this model is particularly interesting since it has been proven to be more parallel than the circuit model [11] (this will be explained in more detail in Part III). This has raised the question, whether it is possible to utilise the parallelism in 1WQC to optimise computations in other classical and quantum computation models. Indeed, the 1WQC model can be used to parallelise quantum circuits through back-and-forth translation to it from the circuit model [10]. The main idea in this kind of parallelisation is to apply 1WQC specific optimisations to the computation in the 1WQC model. Currently there exist two different optimisation techniques that can be used: signal shifting [16] and finding the maximally delayed gflow [9]. Until this work, the relationship between these two methods was unknown.

The main result of this part is the proof of equality in depth resulting in signal shifting and finding the maximally delayed gflow, but in the process of obtaining this proof we have found several important side results. First, to construct a flow corresponding to a signal shifted measurement pattern, we created a new algorithm for signal shiftings in Section 7.1.3. This new algorithm is more efficient in the number of steps required than any previously known one. Second, by showing that signal shifting results in a maximally delayed gflow we created a new method for finding maximally delayed gflows of open graphs with flows using our new signal shifting algorithm. This new method of finding maximally delayed gflows is again more efficient, in the number of elementary steps required, than any previous algorithm that can be used for this purpose. Finally, perhaps one of the most important contribution of this work is a number of new lemmas and techniques regarding the structure of the signal shifted gflows. Although initially used to prove our main theorem, they can be used to obtain new results about 1WQC as has already been proven by [17], who have successfully applied the lemmas in Chapter 8 to create a new method of translating computations from 1WQC to the circuit model in a way that does not increase neither the number of auxiliary qubits nor the depth of the computation.

Chapter 6

Preliminaries

In 1999 Chuang and Gottesman showed how quantum teleportation could be used to implement arbitrary quantum gates [70]. This approach was further developed by other researchers [71, 72, 73, 74], enabling one in principle to perform arbitrary computations given a few primitives: preparation of maximally entangled systems of fixed, small dimension; multi-qubit measurements on arbitrary set of qubits; and the possibility of adapting the measurement bases depending on earlier measurement outcomes.

These models draw on measurements to implement the dynamics of a computation, and as such are called measurement-based quantum computation (MBQC) models. For an overview see the paper by Jozsa [75]. An MBQC model using only single-qubit measurements on special types of entangled states, the so-called *cluster states*, was proposed by Raussendorf and Briegel in 2001, which became known as the one-way quantum computing (1WQC) model [13]. Although two-dimensional cluster states can be used as a resource for universal quantum computation in the one-way model [76], arbitrary graph states may, or may not, serve the same purpose; investigating which kinds of entangled states are useful resources for MBQC is an active area of research [77, 78, 79, 80, 81].

We review the basic ideas behind the one way quantum computing, with special attention to its description in terms of the formal language known as Measurement Calculus [14], and the flow theorems [15, 16].

6.1 The measurement calculus

A formal language describing the computations in the one-way model was developed in [14]. In this framework every 1WQC algorithm (referred to as a *measurement pattern*) consists of a finite sequence of five different types of commands: preparation, entangling, measurement and two types of correction commands. These commands act on a set of working qubits V, out of which some are identified as input and some as output qubits, denoted by I and O correspondingly.

The *preparation command* N_i prepares a qubit i in the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and has to be applied to every non-input qubit before any other command. Since it always has to be applied to the non-input qubits, it is common to omit these commands when a pattern is written out. This is also done in this thesis, the N_i commands are always implicit in the pattern.

The *entangling command* $E_{i,j}$ corresponds to applying the unitary operator $\wedge Z$ to qubits i and j, where

$$\wedge Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{6.1}$$

Through the process of *standardisation* (Section 6.1.2) it is possible to rewrite any measurement pattern such that all the entanglement commands are applied just after the preparation commands. The preparation and entanglement commands together define therefore the resources on which the computation is performed through the application of the subsequent measurement pattern commands. These resources are often represented as *open graphs*.

Definition 6.1 (**open graph**). *An open graph is a triplet (G, I, O), where $G = (V, E)$ is an undirected graph, and $I, O \subseteq V$ are respectively called input and output vertices.*

The qubits of an 1WQC are represented as vertices of an open graph and the entanglement commands as edges. An example of an open graph and description of the notation used in their graphical representation is given in Figure 6.1.

The *measurement command* M_i^θ corresponds to a measurement of qubit i in the basis $|\pm_\theta\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm e^{i\theta}|1\rangle)$, with outcome $s_i = 0$ associated with $|+_\theta\rangle$, and outcome 1 with $|-_\theta\rangle$. The measurement outcomes s_i are usually referred as *signals*.

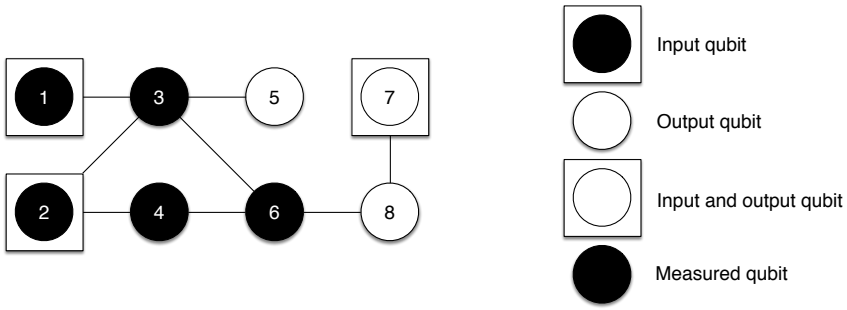


Figure 6.1: An example open graph corresponding to the following measurement pattern: $E_{7,8}E_{6,8}E_{3,5}E_{3,6}E_{4,6}E_{1,3}E_{2,3}E_{2,4}N_3N_4N_5N_6N_8$. Note that a preparation command is not applied to qubit 7, since it is both an input and an output qubit.

The final two types of commands are the *correction commands*. Corrections may be of two types, either Pauli X or Pauli Z, and they may depend on any prior measurement results through signals, denoted by $s = \oplus_{j \in J \subset V} s_j$ ($s_j = 0$ or 1 and the summation is done modulo two). This dependency can be summarised as correction commands: X_i^s and Z_i^s denoting Pauli X and Z corrections on qubit i which must be applied only when the parity of the measurement outcomes on qubits $j \in J \subset V$ equals one (as $Z^0 = X^0 = I$). A characteristic of the 1WQC model is that the choice of measurement bases may depend on earlier measurement outcomes. These dependent measurements can be conveniently written as ${}_t[M_i^\theta]^s$, where

$$ {}_t[M_i^\theta]^s = M_i^\theta X_i^s Z_i^t = M_i^{(-1)^s\theta + t\pi}, \tag{6.2}$$

where it is understood that the operations are performed in the order from right to left in the sequence. The left (t) and right (s) dependencies of the measurement M_i are called its Z and X *dependencies*, respectively.

The commands in a measurement pattern are always applied from right to left, such that they satisfy the following *definiteness* conditions [14]:

- (D0) no command depends on an outcome not yet measured;
- (D1) no command acts on a qubit already measured;
- (D2) no command acts on a qubit not yet prepared, unless it is an input qubit;
- (D3) a qubit i is measured if and only if it is not an output qubit.

6.1.1 An example measurement pattern

As an example, consider the pattern consisting of the qubits $V = \{1, 2\}, I = \{1\}, O = \{2\}$ and the sequence of commands:

$$X_2^{s_1} M_1^{-\theta} E_{1,2} N_2^0. \tag{6.3}$$

This sequence of operations does the following: first it initialises the output qubit 2 in the state $|+\rangle$; then it applies $\wedge Z$ on qubits 1 and 2; followed by a measurement of input qubit 1 onto the basis $\{1/\sqrt{2}(|0\rangle + e^{-i\theta}|1\rangle), 1/\sqrt{2}(|0\rangle - e^{-i\theta}|1\rangle)\}$. If the result is the latter vector then the one-bit outcome is $s_1 = 1$ and there is a correction on the second qubit ($X_2^1 = X_2$), otherwise no correction is necessary. The open graph corresponding the the above sequence is given in Figure 6.2. A simple calculation shows that this pattern implements the unitary J_θ on the state prepared in qubit 1, outputting the result on qubit 2, where

$$J_\theta \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{i\theta} \\ 1 & -e^{i\theta} \end{pmatrix}. \tag{6.4}$$

The simple sequence above is a convenient building block for more complicated computations in the 1WQC model. This is because the set consisting of the single-qubit operator J_θ ($\forall\theta$) together with the $\wedge Z$ operator acting on arbitrary pairs of qubits can be shown to be a universal set of gates for quantum computation [82].

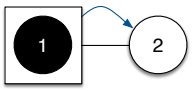


Figure 6.2: The open graph corresponding to the measurement pattern in equation 6.3. We can conclude the following about the pattern by examining the graph: (1) the blue arrow represents the command X^{s_1}; (2) the edge between the two vertices the entanglement command $E_{1,2}$; (3) the square around the first qubit that it is an output qubit; qubit 1 coloured black implies that it is measured (the angle $-\omega$ is not represented on the graph); (4) qubit 2 is left uncoloured to show that it is an output qubit and hence needs to have the command N_2 applied to it at the beginning of a measurement pattern.

6.1.2 Rewriting patterns

The following rewrite rules ([14]) put the command sequence in the *standard* form, where preparation is done first followed by the entanglement, measurements and

corrections:

$$E_{ij}X_i^s \Rightarrow X_i^s Z_j^s E_{ij}, \tag{6.5}$$

$$E_{ij}Z_i^s \Rightarrow Z_i^s E_{ij}, \tag{6.6}$$

$${}_t[M_i^\theta]^s X_i^r \Rightarrow {}_t[M_i^\theta]^{s+r}, \tag{6.7}$$

$${}_t[M_i^\theta]^s Z_i^r \Rightarrow {}_{r+t}[M_i^\theta]^s. \tag{6.8}$$

This procedure is called *standardisation* and can directly change the dependency structure commands, possibly reducing the computational depth, without breaking the causality ordering [10]. There exist more extensive rewrite rules such as *signal shifting* and Pauli optimisation [14], which could be used for parallelising measurement patterns [10]. The former will be described in Section 7.1, where a new algorithm for performing it is presented. The latter is out of the scope of this work; a description of how it can be utilised in with signal shifting for parallelisation can be found in [10].

6.2 Determinism in 1WQC

Due to the probabilistic nature of quantum measurement, not every measurement pattern implements a *deterministic* computation — a completely positive, trace-preserving (cptp) map that sends pure states to pure states. We will refer to the collection of possible measurement outcomes as a *branch* of the computation. In this work, we consider deterministic patterns which satisfies three conditions: (1) the probability of obtaining each branch is the same, called *strong determinism*; (2) for any measurement angle we have determinism, called *uniform determinism*; and (3) which are deterministic after each single measurement, called *stepwise determinism*. We will call those patterns simply *deterministic patterns*. As explained earlier, measurement patterns can act on resource states called open graphs. Similarly to measurement patterns, which not always implement deterministic computations, there exist open graphs which cannot be used for deterministic computation. Identification and characterisation of the graphs that could be used has been done in [15, 16, 83]. Following sections summarise the results most relevant to this work.

6.2.1 Flow

Sufficient conditions (known as the *flow*) for open graphs which can be used for deterministic computation were presented in [15]. Flow is the basis on which we construct the *signal shifted flow* in Chapter 7 and is a central concept in this thesis. In

what follows, we denote non-input vertices as I^C (complement of I in the graph) and non-output vertices as O^C (complement of O in the graph).

Definition 6.2 (**Flow** [15])**.** *We say that an open graph (G, I, O) has flow if and only if there exists a map $f : O^C \to I^C$ and a strict partial order $\prec_f$ over all vertices in the graph such that for all $i \in O^C$*

- *(F1) $i \prec_f f(i)$;*
- *(F2) if $j \in N(f(i))$, then $j = i$ or $i \prec_f j$, where $N(v)$ is the neighbourhood of v;*
- *(F3) $i \in N(f(i))$.*

An example of a graph with flow is shown in Figure 6.3(a). Given a flow $(f, \prec_f)$ of an open graph (G, I, O) it is possible to write a deterministic measurement pattern on the graph [15]:

$$P = \prod_{i \in O^C}^{\prec_f} \left(X^{s_i}_{f(i)} Z^{s_i}_{N(f(i)) \setminus \{i\}} M_i^{\alpha_i} \right) E_G N_{I^C}. \tag{6.9}$$

Note that since this pattern represents a uniformly deterministic computation, it implements a unitary operator irrespective of the measurement angles α_i. From equation 6.9 we see that a Z-correction on a vertex j depending on the measurement outcome of another vertex i appears only if j is a neighbour of $f(i)$. This is formally stated in the next corollary, which we refer to in several places.

Corollary 6.1. *If (G, I, O) is an open graph with a flow $(f, \prec_f)$, then there exists a Z-correction from vertex i to another vertex j if and only if $j \in N(f(i)) \setminus \{i\}$.*

The flow function f is a one-to-one function. The proof is trivial, but as this property is extensively used in this work we will present it here.

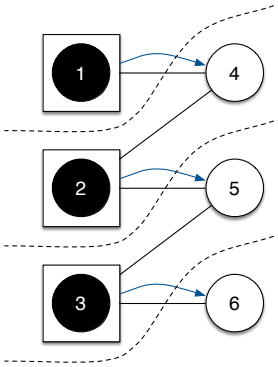
Lemma 6.1. *Let $(f, \prec_f)$ be a flow on an open graph (G, I, O). The function f is an injective function, i.e. for every $i \in O^C$, $f(i)$ is unique.*

Proof. Let us assume that for some $i \in O^C$, $f(i)$ is not unique, *i.e.* there exists $j \in O^C$ such that $i \neq j$ but $f(i) = f(j)$. Then according to the flow definition:

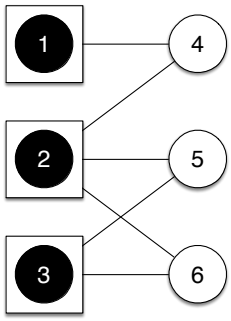
$$j \in N(f(j)) = N(f(i)) \Rightarrow i \prec_f j, \tag{6.10}$$

$$i \in N(f(i)) = N(f(j)) \Rightarrow j \prec_f i, \tag{6.11}$$

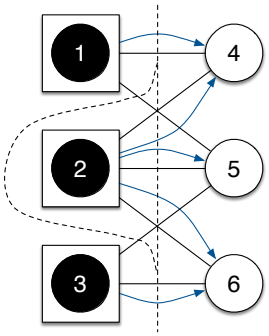
and we arrive at a contradiction because $i \prec_f j$ and $j \prec_f i$ cannot be true at the same time. Hence $f(i)$ has to be unique. □



(a) An open graph with flow and gflow.



(b) An open graph with no flow and no gflow.



(c) An open graph with gflow and no flow.

Figure 6.3: Examples of open graphs with flow (a), gflow (b) and without either (c). Blue lines represent the flow/gflow functions and dashed lines group together measurements that can be performed simultaneously according to the flow/gflow partial order.

6.2.2 General flow

Flow provides only a sufficient condition for determinism but one can generalise the above definition to obtain a condition that is both necessary and sufficient. This generalisation allows correcting sets with more than one element. In those cases, we say that the graph has *generalised flow* (or simply *gflow*). In what follows we define $Odd(K) = \{k\,, |N_G(k) \cap K| = 1 \mod 2\}$ to be the set of vertices where each element is connected with the set K by an odd number of edges.

Definition 6.3 (**Generalised flow** [16])**.** *We say (G, I, O) has generalised flow if there exists a map $g : O^C \to 2^{I^C}$ (the set of all subsets of non-input qubits) and a partial order $\prec_g$ over all vertices in the graph such that for all $i \in O^C$,*

- *(G1) if $j \in g(i)$ then $i \prec_g j$;*
- *(G2) if $j \in Odd(g(i))$ then $j = i$ or $i \prec_g j$;*
- *(G3) $i \in Odd(g(i))$.*

The set $g(i)$ is often referred to as the *correcting set* for qubit i. Flow is a special case of gflow, where $g(i)$ contains exactly one element. An example of a gflow is given in Figure 6.3(c). Interestingly, adding one edge to the flow in Figure 6.3(a) as done in Figure 6.3(b) can remove both the flow and gflow from the open graph, but adding two edges as in Figure 6.3(c) will remove only the flow from the graph. Such graphical representation of underlying entanglement and their link to flow and gflow is fully explained in [84]. Similar to the flow scenario, a deterministic pattern P for an open

graph (G, I, O) can be derived given a gflow $(g, \prec_g)$ on the graph:

$$P = \prod_{i \in O^C}^{\prec_g} \left(X_{g(i)}^{s_i} Z_{Odd(s(i))}^{g_i} M_i^{\alpha_i} \right) E_G N_{I^C}. \tag{6.12}$$

The gflow partial order leads to an arrangement of the vertices into layers (see below), in which all the corresponding measurements can be performed simultaneously. The number of layers corresponds to the number of parallel steps in which a computation could be finished, known as the *depth* of the pattern.

Definition 6.4 (**Depth of a gflow** [9])**.** *For a given open graph (G, I, O) and a gflow $(g, \prec_g)$ of (G, I, O), let*

$$V_k^{\prec_g} = \begin{cases} \max_{\prec_g}(V(G)) & \text{if } k = 0 \\ \max_{\prec_g}(V(G) \setminus (\cup_{i<k} V_i^{\prec_g})) & \text{if } k > 0 \end{cases}, \tag{6.13}$$

where $\max(X)_{\prec_g} = \{u \in X \text{ s.t. } \forall v \in X, \neg(u \prec_g v)\}$ *is the set of maximal elements of X according to $\prec_g$. The* depth $d^{\prec_g}$ *of the gflow is the smallest d such that $V_{d+1}^{\prec_g} = \emptyset$, $(V_k)_{k=0\ldots d^{\prec_g}}$ is a partition of $V(G)$ into $d^{\prec_g} + 1$ layers.*

We define the *layering function* of a gflow based on the above distribution of vertices into layers.

Definition 6.5 (**Layering function**)**.** *Given a gflow $(g, \prec_g)$ on an open graph (G, I, O) we define its* layering function $L_g : V(G) \to \mathbb{N}$ *to be the natural number k such that $L(i) = k$ if and only if $i \in V_k^{\prec_g}$.*

There is another useful way to understand the depth of a gflow. A gflow can be represented as a directed graph on top of an open graph as shown in Figure 6.3. The longest path from inputs to outputs over those directed edges corresponds to the depth of the gflow. In [9] it was shown, that a special type of gflow, called a *maximally delayed gflow*, has minimal depth.

Definition 6.6 (**Maximally delayed gflow** [9])**.** *For a given open graph (G, I, O) and two given gflows $(g, \prec_g)$ and $(g', \prec_{g'})$ of (G, I, O), $(g, \prec_g)$ is more delayed than $(g', \prec_{g'})$ if $\forall k$, $|\cup_{i=0\ldots k} V_i^{\prec_g}| \geq |\cup_{i=0\ldots k} V_i^{\prec_{g'}}|$ and there exists a k such that the inequality is strict. A gflow $(g, \prec_g)$ is* maximally delayed *if there exists no gflow of the same graph that is more delayed.*

Note that in [9] it was proven that the layering of the vertices imposed by a maximally delayed gflow is always unique, however the gflow itself might not be unique. This is

an important property, which together with the following lemmas is exploited later in linking gflow to other known structures of 1WQC.

Lemma 6.2 (Lemma 1 from [9]). *If $(g, \prec)$ is a maximally delayed gflow of (G, I, O) then $V_0^{\prec} = O$.*

Lemma 6.3 (Lemma 2 from [9]). *If $(g, \prec)$ is a maximally delayed gflow of (G, I, O) then $(\tilde{g}, \prec_{\tilde{g}})$ is a maximally delayed gflow of $(G, I, O \cup V_1^{\prec})$ where $\tilde{g}$ is the restriction of g to $V(G) \setminus (V_1^{\prec} \cup V_0^{\prec})$ and $\prec_{\tilde{g}} = \prec \setminus V_1^{\prec} \times V_0^{\prec}$.*

6.2.3 Focused flow

A simpler characterisation of stepwise strong determinism called *focused gflow* was introduced in [83].

Definition 6.7 (Focused gflow [83]). *$g : O^C \to 2^{I^C}$ is a focused gflow of (G, I, O) if*

- *(FG1) g is extensive i.e. the transitive closure of the relation $\{(i, j)$ s.t. $j \in g(i)\}$ is a partial order over $V(G)$;*
- *(FG2) $\forall i \in O^C$, $Odd(g(i)) \cap O^C = \{i\}$.*

Both flow and focused gflow have been shown to be unique for open graphs where input size is equal to output size [85, 83]. One way of explaining the uniqueness of focused gflow is through its relation with the flow. Namely, signal shifting a flow results in a focused gflow as will be shown in Chapter 7.

Chapter 7

Signal shifted flow

The parallel power of 1WQC is proven to be equivalent to quantum circuits augmented with unbounded fan-out [11]. This motivates us to use 1WQC as an automated tool for circuit parallelisation, using the process of signal shifting explained in Section 7.1, as it was first presented in [10]. Another way to obtain parallel 1WQC structure is to use the open graph of the pattern to obtain the optimal gflow of the graph [9]. From this maximally delayed gflow, it is possible to create a measurement pattern (using Formula 6.12 given in previous chapter). Our first main result is to show the equivalence between these two seemingly very different techniques for the patterns obtained from a quantum circuit, that is those with flow. More precisely we show how the effect of performing signal shifting optimisation (that is the core idea in [10]) result in a maximally delayed gflow. This is done by first creating a new type of flow from the signal shifted measurement pattern (Section 7.2), proving that this new flow is actually a gflow (Section 7.3) and proving its equivalence to maximally delayed gflow in Chapter 8. This structural link sheds further light on the complicated structure of maximally delayed gflow and permits us to find a new efficient algorithm for finding it for the large class of patterns obtained from a circuit.

7.1 Signal shifting

7.1.1 The definition of signal shifting

We have already described how finding an optimal gflow could be used as an optimisation tool for one way quantum computation. Next we will explain another known depth reducing technique used for measurement patterns. The core of this

procedure consists of three rewrite rules which manipulate the signals (see section 6.1) of individual commands [14]:

$$ {}_t[M_i^\alpha]^s \Rightarrow S_i^t \; [M_i^\alpha]^s, \tag{7.1} $$

$$ {}_t[M_j^\alpha]^s \; S_i^{t'} \Rightarrow S_i^{t'} \; {}_{t[(t'+s_i)/s_i]}[M_j^\alpha]^s, \tag{7.2} $$

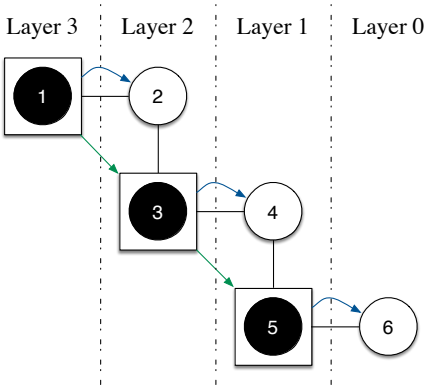
$$ X_j^s \; S_i^t \Rightarrow S_i^t \; X_j^{s[(t+s_i)/s_i]}, \tag{7.3} $$

$$ Z_j^s \; S_i^t \Rightarrow S_i^t \; Z_j^{s[(t+s_i)/s_i]}, \tag{7.4} $$

where S_i^t is the signal shifting command (adding t to s_i) and $s[t/s_i]$ denotes the substitution of s_i with t in s. We call the process of applying rewrite rules 7.1 - 7.4 to a measurement pattern until none of them can be further applied *signal shifting* and the obtained pattern a *signal shifted pattern*. This procedure is interesting in the context of current work, since it can be utilised to parallelise measurement patterns and quantum circuits. This is due to the work of Broadbent and Kashefi, who in [10] showed that signal shifting will never increase the depth of a pattern, whereas it can decrease it.

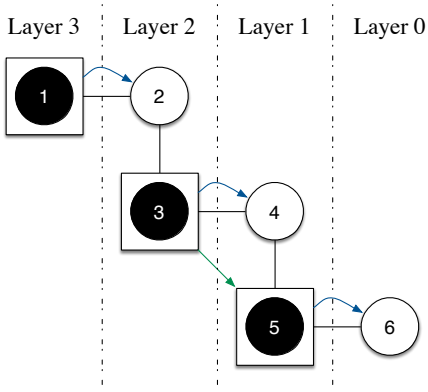
7.1.2 Understanding signal shifting

As can be seen from Rules 7.1 - 7.4, signal shifting is a method for rewriting the X- and Z-corrections of a measurement pattern. An example illustrating that is given in Figure 7.1. Note that we do not have a way to represent the presence of a signal command in the graph representation of a measurement pattern. Thus the graphs of the pattern where the signal commands are not at the end of the computation (Figures 7.1(b) and 7.1(d)) are not actually valid representations, but are included to give a rough idea what happens to the graph during signal shifting. The whole process of signal shifting can be interpreted in the following way: signal shifting takes a signal from a Z-correction on a measured qubit i (Rule 7.1, Figures 7.1(b) and 7.1(d)) and adds it to the corrections that depend on the outcome of the measurement of i (Rules 7.2 - 7.4, Figures 7.1(c) and 7.1(e)). When the signal is added to an X-correction command, it won't propagate any further. On the other hand, if the signal is added to another Z-correction of a measured vertex, then Rule 7.1 can be applied again. This process can be repeated until no Z-corrections are left on non-output vertices. Note, that since Rule 7.1 can only be applied to measured qubits, the process of signal shifting will have the effect of moving all the Z-corrections in the pattern to the output qubits. This removal of the Z-corrections from the measured qubits is the reason why signal shifting can be used to optimise measurement patterns. Note however, that not all measurement patterns will have their depth reduced, as is shown in Figure 7.1.



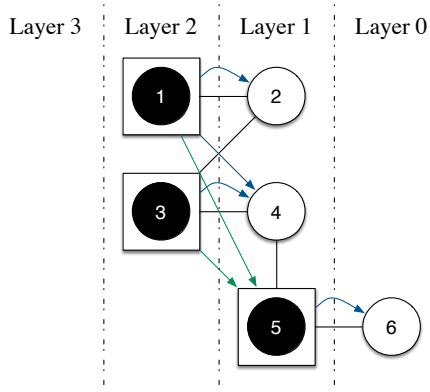
$$X_6^{s_5}\ {}_{s_3}[M_5^{\alpha_5}]X_4^{s_3}\ {}_{s_1}[M_3^{\alpha_3}]X_2^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

(a) The measurement pattern before signal shifting.



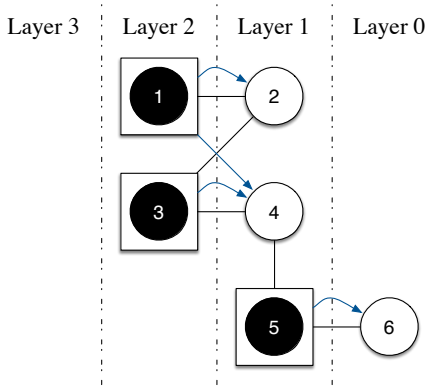
$$X_6^{s_5}\ {}_{s_3}[M_5^{\alpha_5}]X_4^{s_3}S_3^{s_1}\ [M_3^{\alpha_3}]X_2^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

(b) Creating the signal command from qubit 3.



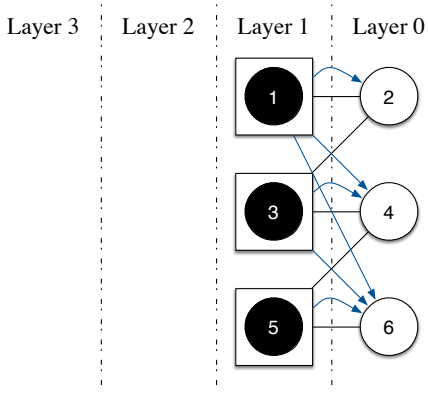
$$S_3^{s_1}X_6^{s_5}\ {}_{s_3+s_1}[M_5^{\alpha_5}]X_4^{s_3+s_1}\ [M_3^{\alpha_3}]X_2^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

(c) Moving the signal from qubit 3 to the end of the pattern.



$$S_3^{s_1}X_6^{s_5}\ S_5^{s_3+s_1}[M_5^{\alpha_5}]X_4^{s_3+s_1}\ [M_3^{\alpha_3}]X_2^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

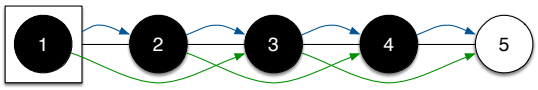
(d) Creating the signal command from qubit 5.



$$S_3^{s_1}S_5^{s_3+s_1}X_6^{s_5+s_3+s_1}\ [M_5^{\alpha_5}]X_4^{s_3+s_1}\ [M_3^{\alpha_3}]X_2^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

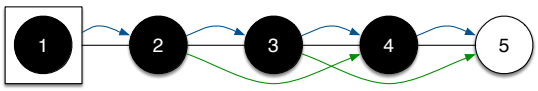
(e) Moving the signal from qubit 5 to the end of the pattern.

Figure 7.1: Signal shifting a measurement pattern. In this particular example, the depth of the computation will be reduced from 4 to 2 through signal shifting.



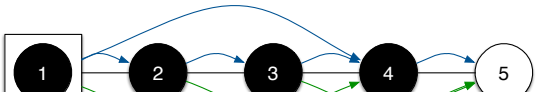
$$X^{s_4}Z^{s_3}\ {}_{s_2}[M_4^{\alpha_3}]^{s_3}\ {}_{s_1}[M_3^{\alpha_3}]^{s_2}\ [M_2^{\alpha_2}]^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

(a) The measurement pattern before signal shifting.



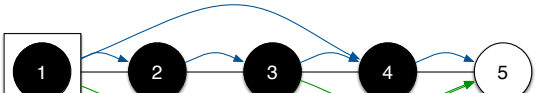
$$X^{s_4}Z^{s_3}\ {}_{s_2}[M_4^{\alpha_3}]^{s_3}\ S_3^{s_1}[M_3^{\alpha_3}]^{s_2}\ [M_2^{\alpha_2}]^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

(b) Creating the signal command from qubit 3.



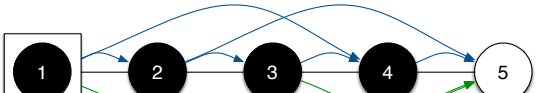
$$S_3^{s_1}X^{s_4}Z^{s_3+s_1}\ {}_{s_2}[M_4^{\alpha_3}]^{s_3+s_1}\ [M_3^{\alpha_3}]^{s_2}\ [M_2^{\alpha_2}]^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

(c) Moving the signal from qubit 3 to the end of the pattern.



$$S_3^{s_1}X^{s_4}Z^{s_3+s_1}\ S_4^{s_2}[M_4^{\alpha_3}]^{s_3+s_1}\ [M_3^{\alpha_3}]^{s_2}\ [M_2^{\alpha_2}]^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

(d) Creating the signal command from qubit 4.



$$S_3^{s_1}S_4^{s_2}X^{s_4+s_2}Z^{s_3+s_1}\ [M_4^{\alpha_3}]^{s_3+s_1}\ [M_3^{\alpha_3}]^{s_2}\ [M_2^{\alpha_2}]^{s_1}\ [M_1^{\alpha_1}]E_G N_{I^C}$$

(e) Moving the signal from qubit 4 to the end of the pattern.

Figure 7.2: An example of a measurement pattern where signal shifting will not reduce the dept of the computation.

There is one important observation to be made from the signal shifting examples in Figures 7.1 and 7.2. If we define a function $g : O^C \to 2^{I^C}$ as $j \in g(i) \Leftrightarrow \exists X_j^{s_i} \in P$, then together with a partial order $\prec_g$, where the partial order respects the measurement order, it is easy to show that $(g, \prec_g)$ is an optimal gflow. This raises the question, wether for every signal shifted computation we can use the X-correction structure to represent an optimal gflow. Note that if that would be true, signal shifting would result in the same computational depth as finding the optimal gflow of the graph. We will show in the following chapters that this is the case if we start with a measurement pattern with flow where $|I| = |O|$. If $|I| = |O|$ we can construct examples where the signal shifting does not give rise to an optimal gflow.

7.1.3 An algorithm for signal shifting

As described above, using signal shifting the signals will move from measured vertices to vertices which have a Z-correction from it, *i.e.* they move along a path created by the Z-corrections. This leads to the description of propagation of signals in a measurement pattern through the *Z-path* and *Z-graph* as defined below.

Definition 7.1 (Z-graph)**.** *Let M be a measurement pattern on an open graph (G, I, O). The* Z dependency graph *(Z-graph) of M, denoted by G_Z, is a directed acyclic graph on the vertices of G, such that there exists a directed edge from i to j if and only if there exists a correction command $Z_j^{s_i}$ in M, i.e:*

- $V(G_Z) = V(G)$,
- $E(G) = \{(i, j) \mid Z_j^{s_i} \in M\}$.

Definition 7.2 (Z-path)**.** *Let M be a measurement pattern on an open graph (G, I, O) and G_Z its Z dependency graph. A path in G_Z between two vertices v and u is called a* Z-path.

When we say that two vertices are connected, we always mean connected *via* an edge. When paths between vertices are considered it will be noted explicitly to avoid confusion. The above definitions allows us to state a simple observation about the connectivity of a graph with flow.

Lemma 7.1. *If $(f, \prec_f)$ is a flow on an open graph (G, I, O), and there exists a Z-path from vertex i to vertex j, then the vertices i and $f(j)$ cannot be connected.*

Proof. The existence of a Z-path from i to j implies that $i \prec_f j$. The Z dependency graph is an acyclic graph, thus $i \neq j$. If i would be connected to $f(j)$, then according

to the flow property (F2):

$$i \in N(f(j)) \quad \wedge \quad i \neq j \quad \Rightarrow \quad j \prec_f i. \tag{7.5}$$

Now we have two contradicting strict partial order relations $i \prec_f j$ and $j \prec_f i$. Therefore i cannot be connected to $f(j)$. □

Recall that the addition of signals is done modulo 2, therefore, if an even number of signals from a measured vertex i is added to a correction command on vertex j, the signals will cancel out (since $Z^2 = X^2 = I$). Furthermore, it is evident from the rewrite Rules of 7.1 - 7.4 that after signal shifting, the measurement result of vertex i will create a new X-correction over vertex j if there exists an odd number of Z-paths from i to a vertex k such that j is X-dependent k in the original pattern. Similarly a new Z-correction from i to j will be created if there exists an odd number of Z-paths from i to j. Either way, *the number of Z-paths from a vertex i to another vertex j*, denoted as $\zeta_i(j)$, can be used to determine if the signal from i should be added to a correction. We define $\zeta_i(i)$ to be 1 to simplify further calculations and definitions in this paper. The importance of the number of Z-paths will manifest itself in subsequent sections, where the relation between signal shifting and gflows is studied.

We define a new structure called the *signal shifted flow* (SSF), and show that it satisfies the three gflow properties in Definition 6.3. Before constructing the SSF, some definitions and lemmas are needed to justify our definition. We define the *Z-dependency neighbourhood* of a vertex j to be the set of vertices from which j is receiving a Z-correction. This set has an explicit form given as $N_Z(j) = \{k \in O^C | f(k) \in N(j) \setminus \{f(j)\}\}$. This is due to the following facts: first, $f(k)$ has to exist for all vertices $k \in O^C$ because of the flow definition; second, since $f(k) \neq f(j)$ the vertex k cannot be equal to j. Because $f(k) \in N(j) \Rightarrow j \in N(f(k))$ and Corollary 6.1 there exists a Z-correction from k to j. It is easy to see, that $\zeta_i(j)$ can be written as:

$$\zeta_i(j) = \sum_{k \in N_Z(j)} \zeta_i(k). \tag{7.6}$$

There exists a Z-correction from every $k \in N_Z(j)$ to j. These Z-corrections can be used to extend every such Z-path to k to reach j. If i is in the sum, then because $\zeta_i(i) = 1$ the correct number of Z-paths is obtained with equation 7.6.

All the properties proved so far allow us to present a new algorithm (Algorithm 1) for performing the signal shifting rules over pattern with flow in the form of equation 6.9. We keep in mind that the order in which we apply the signal shifting rules does not

matter [15]. This algorithm as we discuss later leads to a more efficient gflow finding algorithm.

Algorithm 1: *SignalShift*

Input: A measurement pattern P of a flow $(f, \prec_f)$ as defined in equation 6.9.
Output: The signal shifted pattern P_{SS} of P.

begin
  $toShift = O^C$;
  $P_{SS} = P$;
  while $toShift \neq \emptyset$ **do**
    select any vertex $i \in toShift$ which is the smallest according to $\prec_f$;
    $toShift = toShift \setminus \{i\}$;
    while $\exists k \in toShift$ *s.t.* $Z_k^{s_i} \in P_{SS}$ **do**
      Select the smallest $k \in toShift$ s.t. $Z_k^{s_i} \in P_{SS}$ according to $\prec_f$;
      In P_{SS}, move the $Z_k^{s_i}$ command next to the $M_k^{\alpha_k}$ command;
      Use Rule 7.1 on P_{SS} to create the signal command $S_k^{s_i}$;
      `// Removes the` $Z_k^{s_i}$ `command from` P_{SS};
      Use Rule 7.3 on P_{SS} to create a new $X_{f(k)}^{s_i}$ command;
      foreach $j \in N(f(k)) \setminus \{k\}$ **do**
        Use Rule 7.4 on P_{SS} to create a new $Z_j^{s_i}$ command.
      In P_{SS}, move $S_k^{s_i}$ to the end of the pattern and remove it.

Proposition 7.1. *Given as input the measurement pattern P of a flow $(f, \prec_f)$ defined in equation 6.9, Algorithm 1 outputs the signal shifted measurement pattern of P.*

Proof. We will prove this proposition by showing that:

- Algorithm 1 always terminates.
- Every step in Algorithm 1 that modifies the pattern P_{SS} is a valid application of a signal shifting rewrite rule.
- The output of Algorithm 1, the pattern P_{SS}, is signal shifted.

Note that the "for" loop in the algorithm terminates because the underlying open graph is finite. The first "while" loop will terminate since we decrease the number of elements in $toShift$ on each loop iteration and never add anything to it. As we'll explain now, the second "while" loop will terminate since each k is selected only once and the open graph is finite. Note that since $j \in N(f(k)) \setminus \{k\}$, then according to the flow definition $k \prec_f j$. Every new $Z_j^{s_i}$ command added to P_{SS} is such that $k \prec j$ and hence no removed $Z_k^{s_i}$ command can be added to P_{SS} again. This is true also on on subsequent

loop iterations since we choose k to be the smallest according to $\prec_f$. Hence the second "while" loop and the whole algorithm terminates.

For Algorithm 1 to actually perform the signal shifting, its operations have to be either trivial commuting rules or the four signal shifting Rules 7.1 - 7.4. As can be easily seen from the algorithm, the operations done are indeed the signal shifting rewrite Rules 7.1 - 7.4. We still need to prove, that these rules can be applied in the order shown in the algorithm. Obviously we can use Rule 7.1 on line 8 to create the signal command due to the fact that $k \in toShift \subseteq O^C$ and that every non-output qubit is measured. Hence we have the measurement required for the creation of the signal command in the pattern. We know that $Z_k^{s_i}$ has to be in the pattern after the command $M_i^{\alpha_i}$ and before $M_k^{\alpha_k}$. The entanglement and creation commands are the first commands in the pattern and we do not need to move the $Z_k^{s_i}$ command past them. Hence we only need to move $Z_k^{s_i}$ past measurement commands on qubits that are not i and k and other correction commands. These can be done trivially and hence we can always move the $Z_k^{s_i}$ command next to $M_k^{\alpha_k}$ to apply Rule 7.1.

Next we want to move the newly created S_k^i command to the end of the measurement pattern. To do that we need to commute it past the commands that appear after it. The only commands S_k^i commutes non-trivially with are the ones that depend on the measurement of qubit k as can be seen from Rules 7.1 - 7.4. Those are the X- and Z-corrections depending on the measurement outcome of qubit k. According to equation 6.9 there is exactly one such X-correction in the pattern P, namely $X_{f(k)}^k$. The previous steps of the algorithm could not have created any dependencies from qubit k — the Z-correction commands have only been created depending on vertices that we already moved from $toShift$. Therefore we need to create exactly one new X-correction command using Rule 7.3. We also look at the Z-corrections depending on k and from equation 6.9 we see that in the original pattern these are on vertices from the set $N(f(k)) \setminus \{k\}$. Just like for the X-corrections, we have not created any new Z-corrections from k in the previous steps of the algorithm. Hence this is exactly the set of corrections we need to commute with and apply Rule 7.4. We are only left with commands after S_k^i in the pattern that commute trivially with S_k^i. We can move the command to the end of the pattern. The signal command at the end of the pattern does not influence the computation and we will not add any new commands to the end of the pattern. Hence we can remove the S_i^k command.

Finally we show that no more signal shifting rules can be applied after the completion of Algorithm 1, *i.e.* the pattern P_{SS} is signal shifted. We eliminate all Z-corrections acting on a non-output qubit depending on a vertex i after removing it from the set

$toShift$ and will afterwards never create any new Z-corrections depending on that vertex. At the end of the algorithm the set $toShift$ is empty, hence there cannot exist any non-output qubit that has a Z-correction acting on it and Rule 7.1 cannot be applied anymore. Moreover, since every signal command is at the end of the pattern, we cannot apply the Rules 7.2 to 7.4 either. This completes the proof. □

Proposition 7.2. *Algorithm 1 completes in $O(n^3)$ steps, where n is the number of qubits the input pattern acts on.*

Proof. The number of times the outermost and innermost loops are executed is easy to estimate. The first `while` loop is entered $|O^C| = O(n)$ times and the inner `foreach` loop $\deg(G) = O(n)$ times, where $\deg(G)$ is the degree of the graph. The number of times the second `while` loop is entered depend on the number of times a $Z_j^{s_i}$ command is added on line 15. Adding new Z-correction commands cannot create any loops, as otherwise the algorithm would not halt as proven in Proposition 7.1. Since there cannot be created any loops, the number of new $Z_j^{s_i}$ corrections added can only be $O(n)$ and the second `while` loop can only be entered $O(n)$ times. Therefore the Algorithm 1 completes in $O(n) \cdot O(n) \cdot O(n) = O(n^3)$ steps. □

We consider any trivial commutation of the commands of a pattern resulting in an equivalent pattern. Therefore given the measurement pattern P of a flow $(f, \prec_f)$ as defined in Eq. 6.9 as input to Algorithm 1, the output will be the unique signal shifted measurement pattern of P. Note that Algorithm 1 works almost like a directed graph traversal, where there is a directed edge from vertex i to k if and only if there exists the command $Z_k^{s_i}$ in the measurement pattern. The only difference from a classical directed graph traversal is that we allow visiting of a vertex more than once. Hence we will traverse through every different path in the graph. However we do that exactly once.

7.2 Constructing the SSF

As mentioned before, the evenness of the number of Z-paths can be used to determine if a signal is added to a correction command. If an open graph has a flow, the oddness of $\zeta_i(j)$ can be found as described in the following lemma.

Lemma 7.2. *For every two vertices i and j in an open graph (G, I, O) with flow $(f, \prec_f)$*

$$\zeta_i(j) \bmod 2 = |\{k \in N_Z(j) \mid \zeta_i(k) = 1 \bmod 2\}| \bmod 2, \tag{7.7}$$

i.e. the oddness of $\zeta_i(j)$ depends only on the number of vertices in the Z-dependency neighbourhood which have an odd number of Z-paths from i.

Proof. $\zeta_i(j) \bmod n$ can be written as

$$\begin{aligned}\zeta_i(j) \bmod 2 &= \left(\sum_{k\in N_Z(j)} \zeta_i(k)\right) \bmod 2 \\ &= \sum_{k\in N_Z(j)} (\zeta_i(k) \bmod 2) \bmod 2 \\ &= \sum_{\{k\in N_Z(j) | \zeta_i(k) \bmod 2=1\}} (\zeta_i(k) \bmod 2) \bmod 2 \\ &= |\{k \in N_Z(j) \mid \zeta_i(k) \bmod 2 = 1\}| \bmod 2.\end{aligned} \tag{7.8}$$

□

All these notions will allow us to define the structure of the pattern after signal shifting is performed.

Proposition 7.3. *Given a flow $(f, \prec_f)$ on an open graph (G, I, O), let s be a function from $O^C \mapsto P^{I^C}$ such that $j \in s(i)$ if and only if $\zeta_i(f^{-1}(j)) \mod 2 = 1$. Also define L_s to be a layering function from $V(G)$ into a natural number:*

$$L_s(i) = 0 \qquad \forall i \in O, \tag{7.9}$$

$$L_s(i) = \max_{j\in s(i)}(L_s(j) + 1) \qquad \forall i \notin O. \tag{7.10}$$

Define the strict partial order $\prec_s$ with:

$$i \prec_s j \quad \Leftrightarrow \quad L_s(i) > L_s(j). \tag{7.11}$$

Then, the application of signal shifting Rules 7.1 - 7.4 over a measurement pattern with flow $(f, \prec_f)$ will lead to the following pattern:

$$P = \prod_{j\in O, i\in I^C} Z_j^{s_i\zeta_i(j) \mod 2} \prod_{i\in O^C}^{\prec_s} \left(X_{s(i)}^{s_i} M_i^{\alpha_i}\right) E_G N_{I^C}. \tag{7.12}$$

Proof. The proof is divided into three parts. First we will show that signal shifting creates exactly the commands comprising the pattern shown in equation 7.12. These commands do not have to be in the same order as in equation 7.12. We proceed by showing that the layering function L_s is defined for every $i \in V(G)$. Lastly, we need to

prove that using the partial order $\prec_s$ derived from L_s for ordering the commands as in equation 7.12 gives a valid measurement pattern.

Note that the preparation commands (N_I^C), entanglement commands (E_G) and measurement commands ($M_i^{\alpha_i}$) are the same for equations 6.9 and 7.12. Because signal shifting would not change these commands (Rules 7.1 - 7.3) these are as required for a signal shifted pattern. Hence we need only to consider the correction commands.

We will look at the correction commands that would appear in a signal shifted pattern. We do this by examining the signal shifting algorithm (Algorithm 1). As mentioned before, the algorithm works as a directed graph traversal, in a way that every distinct path is traversed. As seen in the algorithm every $Z_k^{s_i}$ correction acting on a non-output qubit is removed from the pattern. This is in accordance with the proposed pattern in equation 7.12. Let us examine which new corrections are created.

The number of newly created $X_j^{s_i}$ depends on the number of times we enter the first loop with command $Z_{f^{-1}(j)}^{s_i}$. As the algorithm is a directed graph traversal algorithm, this happens as many times as there are different paths over the Z-dependency graph from i to $f^{-1}(j)$. Because the same two $X_l^{s_i}$ corrections cancel each other, a new X-correction appears in a signal shifted pattern only if $\zeta_i(f^{-1}(j)) \mod 2 = 1$. We also note that no new $X_{f(i)}^{s_i}$ correction is created since there exists no Z-path between i and $f^{-1}(f(i))$. On the other hand Algorithm 1 leaves the already existing X- corrections unchanged and moreover since we have defined $\zeta_i(i) = 1$, we have $f(i) \in s(i)$. This implies that the set $s(i)$ does indeed contain all the vertices that have an X-correction depending on s_i after signal shifting is performed.

The number of newly created Z-corrections on an output vertex j depending on a vertex i appearing in the signal shifted pattern is equal to the number of different paths from i to j. The difference with non-output qubits is that these will not be removed through the process of signal shifting. As with X-corrections, two Z-correction commands on the same qubit will cancel each other out and hence the existence of a $Z_j^{s_i}$ in the final pattern depends on the parity of the number of paths from i to j. This can be written in short as

$$Z_j^{s_i \zeta_i(j) \mod 2}. \tag{7.13}$$

Hence the measurement pattern in equation 7.12 has exactly the same commands as the signal shifted pattern in equation 6.9.

Another thing we need to prove is that the layering function L_s is defined for every $i \in V(G)$. As proven above, the X-corrections depending on the measurement of

qubit v correspond to the set $s(v)$. Hence we can interpret the definition of $L_s(v)$ as finding the maximum value of L_s for every vertex that has an X-correction from v and adding 1 to it. The recursive definition of $L_s(v)$ is well defined, if for every non-output qubit we can find a path over X-corrections ending at an output qubit. We know that signal shifting of a valid pattern creates another valid pattern. This implies that the X-corrections cannot create a cyclic dependency structure and hence every path over the X-corrections has an endpoint. Moreover such a path cannot end on a non-output qubit k since $f(k) \in s(k)$ and one could always extend that path with $f(k)$. Therefore $L_s(v)$ is well defined.

Finally, it is easy to show that the partial order $\prec_s$ as used in equation 7.12 gives a valid ordering of the commands. Every vertex j that has an X-correction depending on the measurement of qubit i has a smaller L_s number and hence $i \prec_s j$. This way no X-correction command acts on an already measured qubit and because the Z-corrections are applied only on output qubits, the correction ordering is valid. Every other command is applied before the measurement command and hence the pattern in equation 7.12 is a valid measurement pattern. □

Given an open graph with a flow, we refer to the construction of the above proposition as its corresponding *signal shifted flow* (SSF). Before stating the first major result of this part, it is necessary to highlight the following property of SSF. The usefulness of this property will manifest itself in later sections.

Corollary 7.1. *If (G, I, O) is an open graph with flow $(f, \prec_f)$ and SSF $(s, \prec_s)$ then for every vertex i and j such that $f(j) \in s(i) \setminus \{f(i)\}$, we can find another vertex k, such that $f(k) \in s(i) \cap N(j)$.*

Proof. If $f(j) \in s(i)$, then from the Proposition 7.3 of SSF we can conclude that $\zeta_i(j) \bmod 2 = 1$. We know that $j \neq i$ from the assumptions. Lemma 7.2 says that there must exist at least one other vertex k from which j has a Z-correction, such that $\zeta_i(k) \bmod 2 = 1$. The flow definition says that j must therefore be a neighbour of $f(k)$. Definition 7.3 of SSF states that $f(k)$ must therefore be in $s(i)$, hence $f(k) \in s(i) \cap N(j)$. □

7.3 SSF and gflow

As mentioned before, gflow is a sufficient and necessary condition for determinism while flow is only a sufficient condition. At first it seems that the simple local rewriting rules

of signal shifting could not upgrade a flow to the more powerful gflow construction. Indeed the proof of this statement is not trivial either and is based on discovering various properties of flow of information in an SSF pattern.

Theorem 7.1. *Given any open graph (G, I, O) with flow $(f, \prec_f)$, the corresponding signal shifted flow $(s, \prec_s)$ is a gflow.*

The proof is based on the following lemmas, demonstrating that s is a gflow by satisfying all the properties of Definition 6.3. The first property (G1) of gflow is satisfied by SFF implicitly from Definition 7.3, *i.e.* for every $i \in V(G)$ it holds that $i \prec_s j$ if $j \in s(i)$. Consider the second gflow property (G2), *i.e.* if $j \in Odd(s(i))$ then $j = i$ or $i \prec_s j$. We will prove an even stronger property for the SSF, namely that every vertex with an odd number of connections to $s(i)$ has to be either i itself or an output qubit.

Lemma 7.3. *If $(s, \prec_s)$ is an SSF then every non-output vertex $v \neq i$ connected to $s(i)$ has an even number of connections to $s(i)$, i.e.*

$$\forall v \in N(s(i)) \setminus O \quad \wedge \quad v \neq i \quad \Rightarrow \quad v \notin Odd(s(i)). \tag{7.14}$$

Proof. Let $v \neq i$ be a vertex connected to $s(i)$, we show that the following two sets have the same number of elements.

$$\{k \in N_Z(v) \mid \zeta_i(k) \bmod 2 = 1\} \quad \text{and} \quad s(i) \cap N(v) \setminus \{f(v)\}. \tag{7.15}$$

For every $j \in s(i) \cap N(v) \setminus \{f(v)\}$, we prove $f^{-1}(j)$ is the unique element in

$$\{k \in N_Z(v) \mid \zeta_i(k) \bmod 2 = 1\}. \tag{7.16}$$

Because $j \in s(i)$, from Proposition 7.3 there must exist $f^{-1}(j)$. Also, since $j \in N(v)$ therefore $v \in N(j) = N(f(f^{-1}(j)))$. Moreover since $j \neq f(v)$, Corollary 6.1 implies the existence of a Z-correction from $f^{-1}(j)$ to v, *i.e.* $f^{-1}(j) \in N_Z(v)$. Proposition 7.3 says that because $j \in s(i)$, it must hold that $\zeta_i(f^{-1}(j)) \bmod 2 = 1$. Therefore $f^{-1}(j) \in \{k \in N_Z(v) \mid \zeta_i(k) \bmod 2 = 1\}$.

On the other hand, for every vertex $u \in \{k \in N_Z(v) \mid \zeta_i(k) \bmod 2 = 1\}$, as $\zeta_i(u) \bmod = 1$ then from Proposition 7.3 we have $f(u) \in s(i)$. Also $f(u) \in N(v)$ because of Corollary 6.1 and finally, $f(u) \neq f(v)$ because v cannot have a Z-correction from itself, *i.e.* $v \notin N_Z$. Hence it holds that $f(u) \in s(i) \cap N(v) \setminus \{f(v)\}$ and

$$|s(i) \cap N(v) \setminus \{f(v)\}| = |\{k \in N_Z(v) \mid \zeta_i(k) \bmod = 1\}|. \tag{7.17}$$

According to Lemma 7.2 $\zeta_i(v) \bmod 2 = |\{k \in N_Z(v) \mid \zeta_i(k) \bmod 2 = 1\}| \mod 2$. If $\zeta_i(v) \bmod 2 = 0$, then $s(i) \cap N(v) \setminus \{f(v)\}$ must have an even number of elements. Proposition 7.3 says that $f(v)$ cannot be in $s(i)$ and therefore v can have only even number of connections to $s(i)$. If $\zeta_i(v) \bmod = 1$, then we know that $s(i) \cap N(v) \setminus \{f(v)\}$ must have an odd number of elements. If $f(v)$ exists, it must be in $s(i)$ because of Proposition 7.3. In the case of $f(v) \in s(i)$, we can conclude that $|s(i) \cap N(v)| \bmod 2 = 0$ and v has an even number of connections to $s(i)$. On the other hand if $f(v)$ does not exist, v has to be an output qubit because the flow function f is defined for every non-output vertex. The only possibility of k having odd many connections to $s(i)$ is therefore if k is an output vertex, which proves the lemma. □

The next lemma directly proves that an SSF also satisfies the last gflow property (G3) which states that $i \in Odd(s(i))$.

Lemma 7.4. *If $(s, \prec_s)$ is an SSF, then for every $i \in O^C$ it holds that $i \in Odd(s(i))$.*

Proof. First we show that, performing signal shifting creates new X-corrections only between unconnected vertices. Recall that signal shifting creates a new X-correction between vertices i and j if and only if there exists a Z-path from i to $f^{-1}(j)$ and an X correction from $f^{-1}(j)$ to j, therefore from the Flow definition we have:

$$i \prec_f f^{-1}(j) \prec_f j. \tag{7.18}$$

Let us assume that there exists an edge between i and j. According to the Flow definition

$$\left.\begin{aligned} i \in N(j) \\ j = f(f^{-1}(j)) \end{aligned}\right\} \quad \Rightarrow \quad i \in N(f(f^{-1}(j))) \quad \Rightarrow \quad f^{-1}(j) \prec_f i. \tag{7.19}$$

This contradicts the partial order $i \prec_f f^{-1}(j) \prec_f j$ of the Flow $(f, \prec_f)$ and therefore there cannot be an edge between vertices i and j.

Next we claim that there is exactly one edge between i and $s(i)$. According to Definition 7.3 of SSF, the set $s(i)$ consists only of the vertex $f(i)$ and the vertices to which signal shifting created a new X dependency from i. We showed that signal shifting does not create X dependencies between connected edges. Hence, $f(i)$ is the only vertex in $s(i)$ that can be connected to i, and there must be an edge between i and $f(i)$ because of the flow property (F3) ($i \in N(f(i))$). □

Proof of Theorem 7.1

Proof. To obtain the proof of Theorem 7.1, we note that the definition of SSF implies the gflow property (G1). Lemma 7.3 implies that every SSF satisfies the gflow condition (G2). As the third and last gflow condition is satisfied by SSF according to Lemma 7.4, SSF is indeed a gflow and Theorem 7.1 holds. □

The above theorem for the first time presents a structural link between two seemingly different approach for parallelisation, gflow and signal shifting, for patterns with flow. The next section explores further links between SSF and gflow, showing the usefulness of SSF in parallelisation. We conclude with a simple corollary showing that SSF is in fact a special case of focused flow.

Corollary 7.2. *Given any open graph (G, I, O) with flow $(f, \prec_f)$ and corresponding signal shifted flow $(s, \prec_s)$ the function s is a focused flow.*

Proof. For $(s, \prec_s)$ to be a focused gflow, s must satisfy the two properties (FG1) and (FG2) in Definition 6.7. First, it is clear from Proposition 7.3 that the transitive closure of the relation $\{(i, j)\ s.t.\ j \in g(i)\}$ is a strict partial order over $V(G)$ and hence property (FG1) is satisfied. Second, Lemma 7.3 proves that $Odd(s(i)) \cap O^C \setminus \{i\} = \emptyset$ and Lemma 7.4 states that $i \in Odd(s(i))$. Combining these two lemmas creates exactly the second focused flow property (FG2) $\forall i \in O^C, Odd(s(i)) = \{i\}$. □

Note that although SSF is a focused flow, a focused flow does not always have to be a SSF. Focused flow exists for every open graph with gflow, whereas a SSF by definition exists only for open graphs with flow.

7.4 Properties of SSF

The notions of *influencing walks* and *partial influencing walks* on open graphs with flow was introduced in [10] to describe the set of all vertices that a measurement depends on. An influencing walk starts with an input and ends with an output vertex, a partial influencing walk starts with an input vertex but can end with a non-output vertex. We will use a modified definition of influencing walks that can start from any non-output vertex i and end at any vertex $j \in s(i)$ and call it a *stepwise influencing path*. This will allow us to conveniently explore the dependency structure of a pattern with SSF.

Definition 7.3. *Let $(s, \prec_s)$ be an SSF that is obtained from a flow $(f, \prec_f)$ of an open graph (G, I, O) and vertices i and j in $V(G)$ such that $j \in s(i)$. We say that a path between vertices i and j is an* stepwise influencing path*, noted as $\wp_i(j)$, iff*

- *The path is over the edges of G.*
- *The first two elements on the path are i and $f(i)$.*
- *Every even-placed vertex k on the path $\wp_i(j)$, starting from $f(i)$, is in $s(i)$.*
- *Every odd-placed vertex on the path $\wp_i(j)$ is the unique vertex $f^{-1}(k)$ of some $k \in s(i)$ such that k is the next vertex on the path $\wp_i(j)$.*

It is easy to see that every second edge, in particular the edges between $f^{-1}(k)$ and $k \in s(i)$, in the stepwise influencing path is a flow edge. Hence the path contains no consecutive non-flow edges. If we restrict the first vertices of the stepwise influencing path to be input vertices, the stepwise influencing path would be a partial influencing path, but not *vice versa*. Stepwise influencing paths are useful because of their appearance in the SSF as proven by the following lemma.

Lemma 7.5. *Let $(s, \prec_s)$ be an SSF obtained from a flow $(f, \prec_f)$ of an open graph (G, I, O) and vertices i and j in $V(G)$ such that $j \in s(i)$. Then there always exists a stepwise influencing path $\wp_i(j)$.*

Proof. We start by constructing such a path backward from j to i. We select j and $f^{-1}(j)$ as the last two vertices on the path and apply Corollary 7.1 to find the vertices on the path, until we reach i. The formation of cycles is impossible, as this would imply a cyclic dependency structure, impossible for a flow. We have to reach i as the set of vertices we choose from is finite. □

Note that there might be more than one stepwise influencing path from i to j. We conclude the section about influencing paths with the following two lemmas which will be used to prove the optimality of SSF. First, the structure of stepwise influencing paths imposes a strict restriction on the way a vertex on the stepwise influencing path can be connected.

Lemma 7.6. *Let $\wp_i(j)$ be a stepwise influencing path from i to j in an open graph (G, I, O) with flow $(f, \prec_f)$ and corresponding SSF $(s, \prec_s)$. Then $f^{-1}(j)$ is the only odd-placed vertex in $\wp_i(j)$ that j is connected to.*

Proof. According to the definition of stepwise influencing path, for every three consecutive vertices v_1, v_2, v_3 in $\wp_i(j)$ such that v_1 and v_3 are odd-placed we have

that $v_2 = f(v_1)$ and $v_3 \in N(v_2) = N(f(v_1))$. According to Corollary 6.1 there must exist a Z-correction from v_1 to v_3. Therefore the odd-placed vertices in $\wp_i(j)$ are on a Z-path from i to $f^{-1}(j)$ and obviously from every odd-placed vertex in $\wp_i(j)$ there exists a Z-path to $f^{-1}(j)$. Lemma 7.1 says that j cannot be connected to any of the odd-placed vertices in $\wp_i(j)$. □

The previous lemma shows, that the stepwise influencing paths can be used to describe some properties of the connectivity in open graphs with SSF. The next lemma (illustrated in Figure 7.3) will explain how a stepwise influencing path can be extended.

Lemma 7.7. *Let (G, I, O) be an open graph with flow $(f, \prec_f)$ and corresponding SSF $(s, \prec_s)$ and let i and j be two non-output vertices of the open graph such that $f(j) \in s(i)$. If $v \in N(j) \cap s(i) \setminus \{f(j)\}$ then every stepwise influencing path $\wp_i(v)$ can be extended by the vertices j and $f(j)$ to create another stepwise influencing path $\wp_i(f(j))$.*

Proof. Adding j and $f(j)$ to $\wp_i(v)$ satisfies the conditions for stepwise influencing paths. There exists an edge between vertices j and v and vertices j and $f(j)$, hence it is a valid path. Moreover, $f(j) \in s(i)$ would be an even-placed vertex on the extended path, and j would be the unique oddly-placed vertex with $f(j) \in s(i)$. □

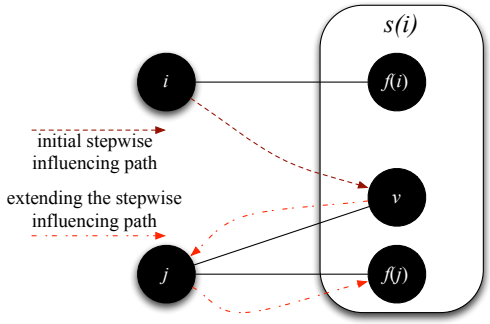


Figure 7.3: Extending a stepwise influencing path ending at vertex v according to Lemma 7.7.

Chapter 8

Computational Depth of SSF

Given an MBQC pattern with gflow, finding the maximally delayed gflow of its underlying graph could potentially further reduce the depth of the computation [9]. A natural question that arises is how SSF is linked with the optimal gflow. In this section, we prove that if the input and output sizes of the pattern are equal, then SSF is indeed the optimal gflow. Hence we can conclude the most optimal parallelisation that one could obtain via translation of a quantum circuit into an MBQC pattern is achieved by the simple rewriting rules of SSF. This will also lead to a more efficient algorithm than the one presented in [9] for finding the maximally delayed gflow of a graph as we discuss later.

Theorem 8.1. *Let (G, I, O) be an open graph with flow $(f, \prec_f)$ such that $|I| = |O|$. Let $(s, \prec_s)$ be the SSF obtained from $(f, \prec_f)$. Then $(s, \prec_s)$ is the optimal gflow of (G, I, O).*

The proof of the theorem is rather long, an outline is presented below. A general reader could omit the next subsections, however various novel constructions have been introduced in the proof that could be explored for other MBQC results and hence could be valuable for an MBQC expert. In Section 8.1 we show that the penultimate layers of an optimal gflow and an SSF of an open graph where $|I| = |O|$, are equal. Next we introduce the concept of a *reduced open graph* in Section 8.2. We prove two key properties of the optimal gflow and SSF of the reduced open graph. This highlights the recursive structures of the gflow and SSF leading to the possibility of extending these notions to new domains[1]. In Section 8.3 we put the pieces together, by showing that the previous properties imply that reduced gflow (implicitly also optimal gflow and SSF) layers are equal to the original gflow layers from layer 1 onward. This allows

[1]For example, the authors are currently exploring this structure to define the concept of partial flow, for patterns with no deterministic computation.

us to construct a recursive proof for Theorem 8.1, which we present in Section 8.4.

8.1 The last two layers

The equality of the last layers of an SSF and maximally delayed gflow follows from Lemma 6.2 and Proposition 7.3 — the last layer of a maximally delayed gflow and an SSF is always the set of output vertices. To prove Theorem 8.1 we need to show that the penultimate layers of SSF and maximally delayed gflow have the same size. One might think that because the penultimate layer of a gflow contains all the vertices that can be corrected by the vertices in the output layer, surely when $|I| = |O| = n$ the number of elements in the penultimate layer would also be n. If that would be true we could omit the proofs in this section and skip to section 8.2, but this is not the case as demonstrated in Figure 8.1. To prove that the penultimate layers of SSF and maximally delayed gflow are equal we need the following properties of open graphs with SSF. An illustration of the property proven in the first of the two lemmas is shown in Figure 8.2.

Lemma 8.1. *Let (G, I, O) be an open graph with flow $(f, \prec_f)$ and corresponding SSF $(s, \prec_s)$. If $i \in O^C$ then for every strict subset S of $s(i)$ containing $f(i)$ there must exist a non-output vertex v that is oddly connected to S such that $f(v) \in s(i) \setminus S$, i.e.*

$$\forall i \in O^C \quad \forall S \subset s(i) \quad s.t. \quad f(i) \in S \quad \exists v \in Odd(S) \quad s.t. \quad f(v) \in s(i) \setminus S. \tag{8.1}$$

Proof. If $s(i) = \{f(i)\}$ the lemma holds trivially, as there does not exist any nonempty strict subsets of $s(i)$. Consider the case where $s(i)$ contains more than one element and S is a strict subset of $s(i)$. Then we select any vertex $j \notin S$ from $s(i)$ and look at the stepwise influencing paths from i to j. Note that there might be more than one such path. We move backwards from j towards i over the stepwise influencing paths in the following way:

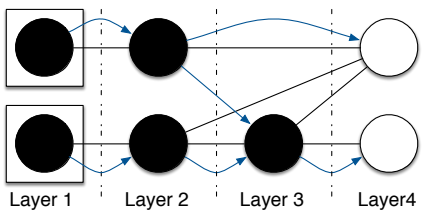


Figure 8.1: An example of a gflow where $|I| = |O|$ and the penultimate layer has less element than the last one.

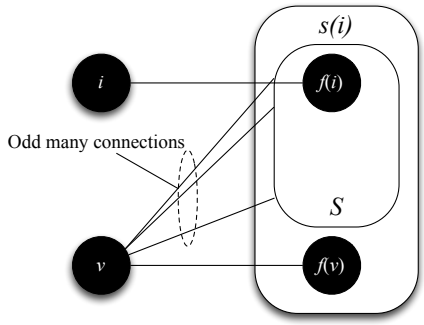


Figure 8.2: For every strict subset S of $s(i)$ containing $f(i)$ we can find a vertex v in the odd neighbourhood of S such that $f(v)$ is not contained in S. This is proven in Lemma 8.1.

1. Move by two vertices

   1.1 If possible, choose any stepwise influencing path where the previous even-placed element is not in S and move to that element.

   1.2 If the previous even-placed elements in all the stepwise influencing paths from i to j are in S, then stop.

2. Repeat step 1.

Let u be the vertex to where we moved using the above process, u has to exist because of the way we initially selected j. There are a couple of other observations that we can make about u. First, $u \in s(i) \setminus S$, because of the selection of j and the way we moved on the paths. Second, u cannot be the first even placed vertex on a stepwise influencing path from i to u because the first element is $f(i) \in S$ (according to Definition 7.3). Third, for every stepwise influencing path ending in u, the previous even-placed vertex has to be in S as otherwise we could have moved one more step towards i.

Considering the previous three observations we can show that the vertex $v = f^{-1}(u)$ must be oddly connected to S. We begin by noting that v cannot be connected to any vertex $k \in s(i) \setminus (S \cup \{f(v)\})$. Otherwise, according to Lemma 7.7, we could extend any stepwise influencing path ending at k with v and $f(v)$. Hence $k \notin S \cup \{f(v)\}$ would then be an even-placed vertex on a stepwise influencing path from i to $f(v)$. In particular, k would be the second to last even-placed vertex on a stepwise influencing path from i to $f(v) = u$ Every such vertex, except $f(v)$ itself, is in S as mentioned before. Because, according to Lemma 7.3, v has to be evenly connected to $s(i)$, it has to be oddly connected to S and Lemma 8.1 holds. □

Next we need to show that every non-input vertex i has a corresponding unique vertex $f^{-1}(i)$, this is only true for those graphs with $|I| = |O|$.

Lemma 8.2. *If $(f, \prec_f)$ is a flow on an open graph (G,I,O), then $|I| = |O|$ if and only if for every $j \in I^C$ there exists $f^{-1}(j)$.*

Proof. First, if $|I| = |O|$ then also $|I^C| = |O^C|$. The flow definition uniquely defines $f(i)$ for every $i \in O^C$ and therefore $f^{-1}(j)$ is uniquely defined for some, but not necessarily for all, vertices $j \in I^C$. The number of vertices for which f is defined must equal the number of vertices for which f^{-1} is defined and because $|I^C| = |O^C|$, f^{-1} must be defined for every element in I^C.

Second, Let us consider the case when for every $j \in I^C$ there exists $f^{-1}(j)$. The number of elements for which f^{-1} is defined equals the number of elements f is defined for. f is by Definition 6.2 defined for every element in O^C. Hence $|I^C| = |O^C|$ which implies that $|I| = |O|$. □

Note that the above requirement, *i.e.* the existence of $f^{-1}(i)$, is the only reason why our proof of Theorem 8.1 fails if $|I| \neq |O|$. We conjecture that by padding the input with necessary ancilla qubits without changing the underlying computation we could extend the above theorem to the general graphs. However the proof of such result is outside of the scope of this thesis and not relevant for the optimisation of quantum circuit.

Note that because of Definition 6.6 if a gflow is not optimal, its penultimate layer has to either be equal to the penultimate layer of the optimal gflow or there exists a vertex in the penultimate layer of optimal gflow that is not included in the penultimate layer of the other gflow. In the proof of the main result we assume that the penultimate layers are not equal, hence we could choose a vertex with particular properties (described in the next two lemmas) to derive a contradiction.

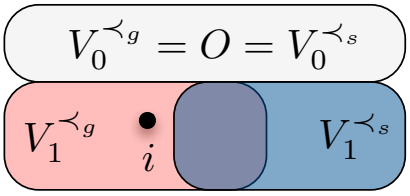


Figure 8.3: The initial conditions required for Lemma 8.3.

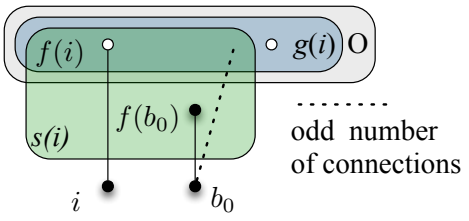


Figure 8.4: The final conditions proved in Lemma 8.3.

Lemma 8.3. *Let (G, I, O) be an open graph where $|I| = |O|$ with flow $(f, \prec_f)$, corresponding SSF $(s, \prec_s)$ and a gflow $(g, \prec_g)$ such that $V_0^{\prec_g} = O$. Assume there exists a vertex $i \in V_1^{\prec_g} \setminus V_1^{\prec_s}$, then*

- $g(i) \subseteq O$,
- $g(i) \cap s(i) \subset s(i)$,
- $f(i) \in g(i)$,

and there exists a vertex b_0 *such that*

- $b_0 \in Odd(g(i) \cap s(i))$,
- $f(b_0) \in s(i) \setminus g(i)$.

Proof. Because i is in $V_1^{\prec_g}$ the set $g(i)$ must be a subset of $V_0^{\prec_g} = O$ according to Definition 6.4. Proposition 7.3 implies that $V_0^{\prec_s} = O$. This and the fact that $i \notin V_1^{\prec_s}$ implies that $s(i)$ is not a subset of the output vertices $O = V_0^{\prec_s}$. Therefore there must exist a non-output vertex in $s(i)$ and, because $g(i) \subseteq O$, this vertex cannot be contained in $g(i)$. Thus the intersection of $s(i)$ and $g(i)$ cannot be equal to $s(i)$ and $g(i) \cap s(i) \subset s(i)$.

We now show that $f(i) \in g(i)$. Let us assume that $f(i) \notin g(i)$, and choose a vertex $a_1 \in g(i)$ connected to i, such a vertex has to exist because the gflow definition says that i is oddly connected to $g(i)$. As $a_1 \in g(i)$ then by the gflow definition a_1 cannot be an input qubit. According to Lemma 8.2, there must exist a vertex $f^{-1}(a_1)$ to which a_1 is connected to. By the definition of flow, $f^{-1}(a_1)$ cannot be an output vertex and thus is not in layer $V_0^{\prec_g}$. As $g(i) \subseteq O$ this also means $f^{-1}(a_1) \notin g(i)$. On the other hand $f^{-1}(a_1)$ is connected to $a_1 \in g(i)$. Because $i \in V_1^{\prec_g}$ and $f^{-1}(a_1) \notin V_0^{\prec_g}$ we know from Definition 6.4 that $i \not\prec_g f^{-1}(a_1)$. As $f^{-1}(a_1)$ is connected to $g(i)$ we can conclude from the gflow definition that $f^{-1}(a_1)$ has to be evenly connected to $g(i)$ and therefore has at least one more connection to a vertex $a_2 \in g(i)$.

Using the same argument for a_2 as for a_1 we can say that there must exist $f^{-1}(a_2) \notin g(i)$ to which a_2 is connected to. Let us assume that $f^{-1}(a_2)$ is not connected to a_1. This means it has only one connection to the set $A_2 = \{a_1, a_2\} \subseteq g(i)$ and is therefore oddly connected to it. We can continue this procedure of selecting vertices from $g(i)$ until we select a vertex a_n such that $f^{-1}(a_n)$ is connected to at least one vertex a_j in $A_{n-1} = \{a_1, \ldots a_{n-1}\} \subseteq g(i)$. If this happens we can no longer say with certainty that $f^{-1}(a_n)$ is oddly connected to $A_n \subseteq g(i)$, which means we cannot select any more elements from $g(i)$ using this method. Because (G, I, O) is a finite open graph we must find this a_n in finite number of steps.

We created the set A_n in such a way that:

$$\forall j \in \{1, 2, \ldots, n-1\} \quad f^{-1}(a_j) \in N(a_{j+1}) = N(f(f^{-1}(a_{j+1}))) \tag{8.2}$$

Hence we have a Z-correction from every $f^{-1}(a_{j+1})$ to $f^{-1}(a_j)$ and thus there exists a Z-path from $f^{-1}(a_n)$ to every $f^{-1}(a_j)$ such that $a_j \in A_{n-1}$ and, because of Lemma 7.5, $f^{-1}(a_n)$ cannot be connected to any vertex in A_{n-1}. This leads to a contradiction with the assumption that it is connected to at least one vertex in A_{n-1}. Therefore our initial assumption that $f(i) \notin g(i)$ must be false and $g(i)$ must contain $f(i)$.

From the definition of SSF we have that $f(i) \in s(i)$ and therefore also $f(i) \in g(i) \cap s(i)$. Now we know that $g(i) \cap s(i)$ is a strict subset of $s(i)$ containing $f(i)$; the existence of b_0 follows from Lemma 8.1. □

Now we prove that if we have a vertex with the same properties as b_0 in Lemma 8.3 and a (possibly empty) subset A of vertices with particular properties (which will be defined in the next lemma) we can always increase the size of A and find another vertex with properties of b_0. This would imply the possibility of increasing the size of A to infinity and will give us the contradiction we need.

Lemma 8.4. *Let (G, I, O) be an open graph where $|I| = |O|$ with flow $(f, \prec_f)$, corresponding SSF $(s, \prec_s)$ and a gflow $(g, \prec_g)$. If we have a vertex i in the open graph such that*

- $g(i) \subseteq O$,
- $g(i) \cap s(i) \subset s(i)$,
- $f(i) \in g(i)$,

and if we have a subset $A \subseteq g(i)$ and another vertex b_0 such that

- $b_0 \in Odd(g(i) \cap s(i))$,
- $f(b_0) \in s(i) \setminus g(i)$,
- $\forall j \in A \quad \exists \quad b_0 \xrightarrow{Z} f^{-1}(j)$,

then there exists another vertex c_o and a non empty set $B \subseteq g(i)$ such that

- $B \neq \emptyset$,
- $B \cap A = \emptyset$,
- $c_0 \in Odd(g(i) \cap s(i))$,

- $f(c_0) \in s(i) \setminus g(i)$,
- $\forall j \in A \cup B \quad \exists \quad c_0 \xrightarrow{Z} f^{-1}(j)$.

Proof. The proof consists of three steps: we start by constructing the set B; we proceed with finding the vertex c_0; and finally we prove that c_0 has the required properties.

Define $S = g(i) \cap s(i)$, since $f(b_0)$ exists hence b_0 cannot be an output vertex. Also since $g(i) \subseteq O$ therefore b_0 is not in $g(i)$. As $b_0 \notin O = V_0^{\prec_g}$ and $g(i) \subseteq O$ we can conclude from Definition 6.4 that $i \in V_1^{\prec_g}$ and $i \not\prec_g b_0$. Therefore according to the gflow definition, b_0 must be in the even neighbourhood of $g(i)$. We also know from the initial conditions of this lemma that b_0 is in the odd neighbourhood of $g(i) \cap s(i)$. Thus there has to exist a vertex v_1 in $g(i)$ to which b_0 is connected to, but which is not included in $g(i) \cap s(i)$, *i.e.* $v_1 \in g(i) \setminus s(i)$. As $g : O^C \to P^{I^C}$, $v_1 \in g(i)$ cannot be an input qubit and because f^{-1} exists for every non-input qubit according to Lemma 8.2, there must exist a vertex $f^{-1}(v_1) = b_1$. It is also important for the later part of the proof to note that $f(b_1) = v_1 \notin A$. This is due to Lemma 7.1, which implies that b_0 cannot be connected to any vertex in A.

Define $B_0 = S$ and consider the case when b_1 is evenly connected to B_0. Remember that the flow property (F3) says that there is always an edge between b_1 and $f(b_1)$. This means that b_1 is oddly connected to $B_1 = \{f(b_1)\} \cup B_0$ which is a subset of $g(i)$. But again because of the gflow property (G2) we have that b_1 must be evenly connected to $g(i)$. Thus there must exist another vertex b_2 such that b_1 is connected to $f(b_2) \in g(i) \setminus B_1$, otherwise b_1 could not be in the even neighbourhood of $g(i)$. If b_2 is evenly connected to B_1, it must be oddly connected to $B_2 = \{f(b_2)\} \cup B_1$ which is again a subset of $g(i)$. If b_2 is oddly connected to B_2 there must exist a vertex b_3 such that b_3 is connected to $f(b_3) \in g(i) \setminus B_2$, otherwise b_2 could not be in the even neighbourhood of $g(i)$. We can continue this scheme until we get to vertex b_n that is oddly connected to B_{n-1}. As $B_n = \{f(b_n)\} \cup B_{n-1}$ and there exists an edge between b_n and $f(b_n)$ we get that b_n must be evenly connected to B_n. Such vertex b_n must exist, otherwise we could continue selecting elements from $g(i)$ infinitely, but (G, I, O) is a finite open graph. We select $B = B_n \setminus S$. Recall that $f(b_1)$ must exist, therefore B must have at least on element.

Next we show b_n is oddly connected to S. We note that we have the following:

$$\forall j \in \{1, 2, \ldots, n\} \quad b_j \in N(f(b_j)) \quad \wedge \quad b_{j-1} \in N(f(b_j)). \tag{8.3}$$

Corollary 6.1 implies that for every $j > 0$ there exists a Z-correction from b_j to b_{j-1}. Thus we have a Z-path from b_n to every other b_j where $j < n$, hence from Lemma 7.1 we

conclude b_n cannot be connected to any vertex $f(b_j) \in B_{n-1}$ where $j < n$. The number of edges that connect the vertices in B_{n-1} to vertex b_n has to be the same as the number of edges between vertices of S and b_n, because $B_{n-1} = \{f(b_1), f(b_2), \ldots, f(b_{n-1})\} \cup S$. As b_n was oddly connected to B_{n-1}, it must also be oddly connected to S. Note that however b_n does not have the required properties for c_0, but will be used to find such a vertex.

The gflow definition says that b_n must be evenly connected to $s(i)$. It is also oddly connected to $s(i) \cap g(i)$ hence there must exist a vertex $c \in s(i) \setminus g(i)$ to which b_n is connected to. According to Lemma 7.5 there exists a stepwise influencing path $\wp_i(c)$ and due to Definition 7.3, $f(i)$ has to be on on this path. Therefore there exists at least one element in $\wp_i(c)$ that is in S. Let $f(a_0)$ be the last element of the path $\wp_i(c)$ in S.

Define a_1 to be the vertex in $\wp_i(c)$ that comes after $f(a_0)$. We know that a_1 has odd many Z-paths from i because Definition 7.3 implies that $f(a_1) \in s(i)$. If a_1 is already oddly connected to S, then we are done and $a_1 = c_0$. If a_1 is evenly connected to $S \subset s(i)$, then we know that it must be oddly connected to $S \cup \{f(a_1)\} \subseteq s(i)$. There must exist another vertex $f(a_2) \in s(i) \setminus (S \cup \{f(a_1)\})$ to which a_1 is connected to for it to be evenly connected to $s(i)$ as is required by Lemma 7.3. Because $f(a_2) \in s(i)$ we know there exists a stepwise influencing path $\wp_i(f(a_2))$ (Lemma 7.5) and we can extend that path by a_1 and $f(a_1)$ as was proven in Lemma 7.7. We move backward on this path and find the element a_2. If a_2 is oddly connected to S, we are done and set $c_0 = a_2$. Otherwise we can continue as was the case for a_1 until we find an element a_m that is oddly connected to S. This element must exist since graph is finite and the Z corrections do not create any loops. We select $c_0 = a_m$. Note that a_m cannot be i because $f(i) \in S = s(i) \cap g(i)$ but $f(a_m) \notin g(i)$.

There is a Z-path from $a_m = c_0$ to a_1 (we moved backwards along this path to find a_m) and from a_1 to b_n because of the way we selected a_1. There also exists a Z-path from b_n to every other b_j such that $0 \leq j < n$, thus a_m will also have a Z-path to every b_j in $\{b_1, b_2, \ldots, b_n\}$. Moreover, because:

$$\begin{aligned} & a_m \xrightarrow{Z} b_n \quad \wedge \quad b_n \xrightarrow{Z} b_0 \quad \wedge \quad \forall j \in A \qquad b_0 \xrightarrow{Z} f^{-1}(j) \quad \Rightarrow \\ \Rightarrow \quad & \forall j \in A \qquad a_m \xrightarrow{Z} f^{-1}(j). \end{aligned} \tag{8.4}$$

This completes the proof. □

Finally we could put together Lemmas 8.3 and 8.4.

Lemma 8.5 (**Equality of the penultimate SSF and optimal gflow layer**). *Let*

(G, I, O) be an open graph with flow $(f, \prec_f)$, corresponding SSF $(s, \prec_s)$ and optimal gflow $(g, \prec_g)$ such that $|I| = |O|$. Then $V_1^{\prec_s} = V_1^{\prec_g}$.

Proof. Assume $V_1^{\prec_s} \neq V_1^{\prec_g}$ we show how we can choose infinitely many different vertices from $V(G)$. Due to Definition 6.6 we have $|V_1^{\prec_s}| \leq |V_1^{\prec_g}|$ and since $V_1^{\prec_s} \neq V_1^{\prec_g}$ hence trivially $V_1^{\prec_g} \not\subset V_1^{\prec_s}$ and there must exist a vertex i in $V_1^{\prec_g} \setminus V_1^{\prec_s}$. Then from Lemma 6.2 we have $V_g^0 = O$ and using Lemma 8.3 we obtain the following:

- $g(i) \subseteq O$,
- $f(i) \in g(i)$,
- $g(i) \cap s(i) \subset s(i)$,

and that there exists another vertex b_0 such that

- $b_0 \in Odd(g(i) \cap s(i))$,
- $f(b_0) \in s(i) \setminus g(i)$.

These constraints together with an empty set A allow us to apply Lemma 8.4. Lemma 8.4 is constructed in such a way that whenever we can apply it to a (possibly empty) set A, it proves the existence of another set B such that that $|A| < |A \cup B|$ and Lemma 8.4 is applicable to the new set $A \cup B$. Thus it is possible to apply Lemma 8.4 infinitely many times and construct a subset of $V(G)$ containing infinitely many vertices. This leads to a contradiction as G is a finite graph. □

8.2 Reducing the open graph

The equality of penultimate layers of SSF and gflow might suggest that one could prove the equality of other layers simply by removing the last layer from the open graph and reapply the lemmas from the last section. However this would fail as the vertices in any layers can also use the output vertices in their correcting sets. Therefore we need to be careful which vertices we remove such that the reduced graph still have a gflow.

Definition 8.1. *If (G, I, O) is an open graph with flow $(f, \prec_f)$ and corresponding SSF $(s, \prec_s)$ then we call the open graph (G', I, O') a* reduced open graph according to $(s, \prec_s)$, *where*

- $R = \{v \in O \mid f^{-1}(v) \in V_1^{\prec_s}\}$ *is the set of removed vertices;*
- $G' = (V', E')$ *where*

$$V' = V \setminus R,$$

$$E' = E \setminus (V \times R);$$

- $O' = (V_1^{\prec_s} \cup O) \setminus R$.

We will omit "according to ..." and call (G', I, O') just reduced open graph when it is clear from the text which SSF is used for constructing it. An example of a reduced open graph is shown in Figure 8.5

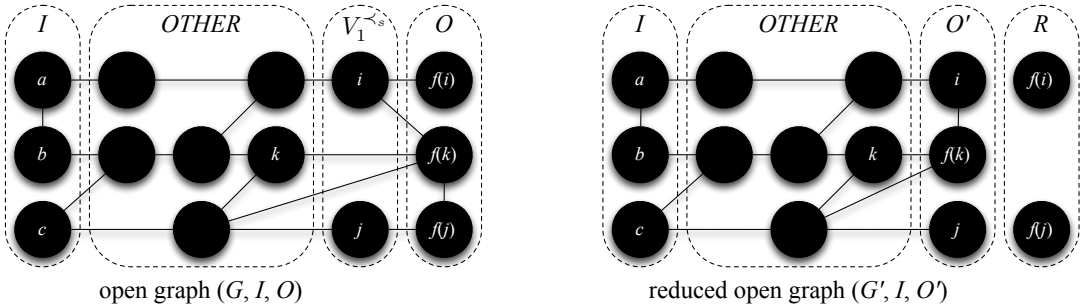


Figure 8.5: An example of an SSF reduced open graph (right) together with the original open graph (left).

As we saw in the previous section, we needed the fact that $|I| = |O|$ to be able to prove that the penultimate layers of SSF and optimal gflow are equal. If we want to apply the same lemmas to the new reduced open graph, we need to guarantee that if we start with a graph where input size equals output size, the same holds for the reduced open graph.

Lemma 8.6. *Let (G', I, O') be a reduced open graph of the open graph (G, I, O), then $|O| = |O'|$.*

Proof. Let R be the set of vertices removed from G, then for every vertex $i \in V_1^{\prec_s}$ we have a corresponding unique vertex $f(i)$ in R since Proposition 7.3 implies that $s(i) \subseteq O$ and $f(i) \in s(i)$. On the other hand, for every vertex in R there exists a corresponding vertex in $V_1^{\prec_s}$ from the definition of R. Therefore for every vertex $v \in R$ that we remove from O when constructing $O' = (V_1^{\prec_s} \cup O) \setminus R$ we add another vertex $f^{-1}(v) \in V_1^{\prec_s}$ and it must hold that $|O| = |O'|$. □

The next lemma is used later to construct a gflow of the reduced open graph from the gflow of the original open graph.

Lemma 8.7. *Let (G, I, O) be an open graph and A and B two sets in O such that $Odd(B) \cap O^C = \emptyset$. Then $Odd((A \cup B) \setminus (A \cap B)) \cap O^C = Odd(A) \cap O^C$.*

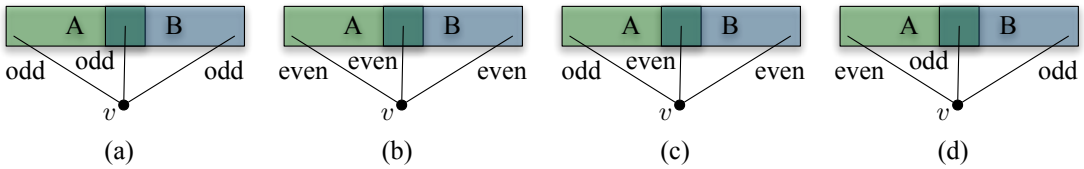


Figure 8.6: The four possibilities for a vertex $v \in O^C$ to be connected connected to the sets A and B. The vertex v can be either oddly (a) or evenly (b) connected the sets $A \setminus B$, $B \setminus A$ and $A \cap B$; oddly connected to $A \setminus B$ and evenly to $B \setminus A$ and $A \cap B$ (c); or evenly connected to $A \setminus B$ and oddly to $B \setminus A$ and $A \cap B$ (d).

Proof. There are altogether four different possibilities for a vertex $v \in O^C$ to be connected to the sets A and B satisfying $Odd(B) \cap O^C = \emptyset$ as shown in Figure 8.6:

$$\begin{aligned}
& v \in Even(A) \cap Odd(A \setminus B) \Rightarrow v \in Odd(A \cap B) \Rightarrow v \in Odd(B \setminus A) \Rightarrow \\
\Rightarrow \quad & v \in Even((A \setminus B) \cup (B \setminus A)), && (8.5) \\
& v \in Even(A) \cap Even(A \setminus B) \Rightarrow v \in Even(A \cap B) \Rightarrow v \in Even(B \setminus A) \Rightarrow \\
\Rightarrow \quad & v \in Even((A \setminus B) \cup (B \setminus A)), && (8.6) \\
& v \in Odd(A) \cap Odd(A \setminus B) \Rightarrow v \in Even(A \cap B) \Rightarrow v \in Even(B \setminus A) \Rightarrow \\
\Rightarrow \quad & v \in Odd((A \setminus B) \cup (B \setminus A)), && (8.7) \\
& v \in Odd(A) \cap Even(A \setminus B) \Rightarrow v \in Odd(A \cap B) \Rightarrow v \in Odd(B \setminus A) \Rightarrow \\
\Rightarrow \quad & v \in Odd((A \setminus B) \cup (B \setminus A)). && (8.8)
\end{aligned}$$

We see that every time v is evenly connected to A it is also evenly connected to $(A \setminus B) \cup (B \setminus A)$ and every time v is oddly connected to A it is also oddly connected to $(A \setminus B) \cup (B \setminus A)$. Because $(A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$ and v is in O^C it must hold that $Odd((A \cup B) \setminus (A \cap B)) \cap O^C = Odd(A) \cap O^C$. □

We start by creating a function that will be proven to have the required properties of the gflow.

Lemma 8.8 (**Finding the reduced gflow function**). *Let (G, I, O) be an open graph with flow $(f, \prec_f)$, SSF $(s, \prec_s)$ and optimal gflow $(g, \prec_g)$ such that $|I| = |O|$. Let (G', I, O') be the SSF reduced open graph of (G, I, O) with the removed vertices set R, then there exists a function $g' : O'^C \to P^{I^C \cap V(G')}$ such that:*

1. $\forall i \in O'^C \quad g'(i) \cap O'^C = g(i) \cap O'^C$,
2. $\forall i \in O'^C \quad Odd(g'(i)) \cap O'^C = Odd(g(i)) \cap O'^C$.

Proof. We start by noting that according to Lemma 6.3 we can create an optimal

gflow $(\tilde{g}, \prec_{\tilde{g}})$ of the open graph $(G, I, O \cup V_1^{\prec_g}) = (G, I, O' \cup R)$ by restricting g to $V(G) \setminus (V_1^{\prec_g} \cup V_0^{\prec_g}) = O'^C$ and setting $\prec_{\tilde{g}} = \prec_g \setminus V_1^{\prec_g} \times V_0^{\prec_g}$. We construct our desired g' function from $\tilde{g}$.

We consider $i \in O'^C$, if there exists a vertex $j \in R \cap \tilde{g}(i)$ then from the reduced open graph definition we have $f^{-1}(j) \in V_1^{\prec_g}$. Also from Lemma 8.5 we have $V_1^{\prec_g} = V_1^{\prec_s}$ and thus $f^{-1}(j) \in V_1^{\prec_s}$. According to Proposition 7.3 this means that $s(f^{-1}(j)) \subseteq O$. We have $Odd(s(f^{-1}(j))) \cap O'^C = \emptyset$ since the only odd neighbours of $s(f^{-1}(j))$ are either output vertices or the vertex $f^{-1}(j) \in V_1^{\prec_g} \subseteq O'$.

Now we define $g'(i) = (\tilde{g}(i) \cup s(f^{-1}(j))) \setminus (\tilde{g}(i) \cap s(f^{-1}(j)))$, hence $j \notin g'(i)$. Moreover Lemma 8.7 implies that $Odd(g'(i)) \cap O'^C = Odd(\tilde{g}(i)) \cap O'^C$. Also $g'(i) \cap O'^C = \tilde{g}(i) \cap O'^C$ since $s(f^{-1}(j)) \subseteq O'$. Note that, since the new set will be constructed via a union of two sets we might add another vertex $k \in R$ to the set $g'(i)$. However, we can remove any such vertex k added to $g'(i)$ by applying the same procedure recursively. For every such vertex k, it must hold that $f^{-1}(j) \prec_f f^{-1}(k)$ since $k \in s(f^{-1}(j))$ and Proposition 7.3 implies the existence of a Z-path from $f^{-1}(j)$ to $f^{-1}(k)$. Now we remove k via the above procedure *i.e.* defining $g'(i) = (g'(i) \cup s(f^{-1}(k))) \setminus (g'(i) \cap s(f^{-1}(k)))$. If this would add vertex j again to $g'(i)$, hence there exists a Z-path from $f^{-1}(k)$ to $f^{-1}(j)$ and $f^{-1}(k) \prec_f f^{-1}(j)$ which contradicts the previous relation. This procedure will eventually terminate and remove all undesired vertices $k \in R$ since in the above procedure we never create any Z-path loops. □

We call a function which satisfies properties (1) and (2) of Lemma 8.8 the *reduced gflow function* of g. We can interpret these properties as saying that the gflow function g' differs from the gflow function g only by the vertices in O', *i.e.* the other elements in the correcting set are left unchanged. As a gflow consists of a function and a partial order, we still need to define a valid partial order. The one that is most useful to us is such that it preserves as much relations as possible from the original gflow, hence the layering structures remain similar.

Lemma 8.9 (Constructing the reduced gflow). *Let (G, I, O) be an open graph with SSF $(s, \prec_s)$, gflow $(g, \prec_g)$ and (G', I, O') a reduced open graph of (G, I, O). If g' is a reduced gflow function of $(g, \prec_g)$, then $(g', \prec_{g'})$ is a gflow of (G', I, O'), where*

$$\forall i, j \in O'^C \quad i \prec_g j \Leftrightarrow i \prec_{g'} j, \tag{8.9}$$

$$\forall i \in O'^C, \forall j \in O' \cap g'(i) \quad \Rightarrow \quad i \prec_{g'} j. \tag{8.10}$$

Proof. We will show that $(g', \prec_{g'})$ satisfies the three gflow properties (G1) — (G3) in Definition 6.3. First property requires that if $j \in g'(i)$, then $i \prec_{g'} j$. This is obviously

true if $j \in O'$. If $j \in g'(i) \cap O'^C$, from Lemma 8.8 we have $j \in g(i)$ which implies that $i \prec_g j$ because $(g, \prec_g)$ is a gflow. Now according the definition of $\prec_{g'}$ it must also hold, that $i \prec_{g'} j$.

Now we consider the gflow property (G2). For every $j \in Odd(g'(i))$ it must be that $j = i$ or $i \prec_{g'} j$. If $j \in O'$, then again this is obviously true because of the definition of $\prec_{g'}$. If $j \in Odd(g'(i)) \cap O'^C$ then we know that $j \in Odd(g(i))$ and $j = i$ or $i \prec_g j$. According to the definition of $\prec_{g'}$, $i \prec_g j$ implies that $i \prec_{g'} j$ and we have that if $j \in Odd(g'(i))$ then either $i = j$ or $i \prec_{g'} j$. Thus the gflow property (G2) is satisfied.

Finally, we require for gflow property (G3) that $i \in Odd(g'(i))$ and as $i \in O'^C$ this is true because of the properties of g'. □

We call the gflow $(g', \prec_{g'})$ from Lemma 8.9 the *reduced gflow* of $(g, \prec_g)$. Similarly we can construct the SSF of the reduced open graph. Note that an SSF can only exist if the reduced open graph has flow. Thus arises the need to prove the existence of a flow on the reduced open graph, as is done in the next lemma.

Lemma 8.10. *If (G, I, O) is an open graph with flow $(f, \prec_f)$ and if (G', I, O') is the reduced open graph described in Definition 8.1, then $(f', \prec_{f'})$, where*

- $\forall i \in O'^C \quad f'(i) = f(i)$
- $\prec_{f'} = \prec_f \setminus [(V \times R) \cup (V_1^{\prec_s} \times O)]$

is a flow of (G', I, O').

Proof. It is sufficient to show that $(f', \prec_{f'})$ satisfies the flow properties (F1) — (F3) in Definition 6.2 and that f' is a function from O'^C to I^C. It is easy to see that $f' : O'^C \to V' \setminus I$. f' acts by definition on O'^C and

$$\forall i \in O'^C \quad f'(i) = f(i) \in V \setminus I. \tag{8.11}$$

The graph G' has fewer vertices than G, therefore we need to show that all the vertices required according to the flow function f' are included in G', *i.e.* $\forall i \in O'^C$ it must hold that $f'(i) \in V'$. According to Definition 8.1 every vertex v removed from the initial open graph (G, I, O) is chosen such that $f^{-1}(v) \in O'$. Therefore it must be that every vertex $j \in V'$ such that $f(j) \notin V'$ must be an output vertex in (G', I, O'^C). Because $f'(j)$ is not defined for outputs the vertices removed from the original graph G are not needed for f' and $f' : O'^C \to V' \setminus I$. Hence we have that $f(i) = f'(i) \in I^C$ for every vertex $i \in O'^C$ and $f' : O'^C \to I^C$.

Let R be the set of removed vertices as defined in Definition 8.1. The flow property (F1) states that $i \prec_{f'} f'(i)$ and holds because:

$$\begin{aligned} & \forall i \in O'^C \subseteq O^C \quad i \prec f(i) = f'(i) \quad \Rightarrow \\ \Rightarrow \quad & f'(i) \notin R \quad \wedge \quad (i, f'(i)) \in \prec_f \quad \Rightarrow \\ \Rightarrow \quad & (i, f'(i)) \in \prec_f \setminus V \times R = \prec_{f'} \quad \Rightarrow \\ \Rightarrow \quad & i \prec_{f'} f'(i). \end{aligned} \tag{8.12}$$

To prove that $(f', \prec_{f'})$ satisfied flow property (F2) we need to show that for every $j \in V'$ if $j \in N(f'(i))$ then either $j = i$ or $i \prec_{f'} j$.

$$\begin{aligned} & j \in N(f'(i)) \quad \Rightarrow \quad j \in N(f(i)) \quad \Rightarrow \\ \Rightarrow \quad & j = i \quad \vee \quad i \prec_f j \quad \Rightarrow \\ \Rightarrow \quad & j = i \quad \vee \quad (i,j) \in \prec_f \quad \wedge \quad j \in V' = V \setminus R \quad \Rightarrow \\ \Rightarrow \quad & j = i \quad \vee \quad (i,j) \in \prec_f \setminus V \times R = \prec_{f'} \quad \Rightarrow \\ \Rightarrow \quad & j = i \quad \vee \quad i \prec_{f'} j. \end{aligned} \tag{8.13}$$

Finally the flow property (F3) $i \in N(f'(i))$ holds almost trivially:

$$i \in O'^C \subseteq O^C \quad \Rightarrow \quad i \in N(f(i)) = N(f'(i)). \tag{8.14}$$

□

Next we prove that the reduced gflow of an SSF is also an SFF.

Lemma 8.11 (**Constructing the reduced SSF**). *Let (G, I, O) be an open graph with flow $(f, \prec_f)$ and SSF $(s, \prec_s)$. If (G', I, O') is the reduced open graph, according to $(s, \prec_s)$, then there exists an SSF $(s', \prec_{s'})$ of (G', I, O') such that $(s', \prec_{s'})$ is the reduced gflow of $(s, \prec_s)$.*

Proof. Let R be the set of vertices removed from (G, I, O) to get (G', I, O'). The reduced flow $(f', \prec_{f'})$ exists because of Lemma 8.10. Define $(s', \prec_{s'})$ to be the the SSF derived from this reduced flow. Assume $(s', \prec_{s'})$ is not a reduced gflow of $(s, \prec_s)$, then one of the properties of Lemma 8.8 should not hold, We show a contradiction in both cases.

If the first property does not hold then

$$\begin{aligned}
&\exists i \in O'^C \quad s.t. \quad s'(i) \cap O'^C \neq s(i) \cap O'^C \Rightarrow \\
&\exists j \in O'^C \cap [(s(i) \setminus s'(i)) \cup (s'(i) \setminus s(i))] \Rightarrow \\
&(\zeta_i^s(f^{-1}(j)) \bmod 2 = 1 \wedge \zeta_i^{s'}(f^{-1}(j)) \bmod 2 = 0) \vee \\
&(\zeta_i^{s'}(f^{-1}(j)) \bmod 2 = 1 \wedge \zeta_i^{g}(f^{-1}(j)) \bmod 2 = 0) \Rightarrow \\
&\zeta_i^s(j) \bmod \neq \zeta_i^{s'}(j) \bmod 2.
\end{aligned} \tag{8.15}$$

Hence by removing vertices and edges from the open graph (G, I, O) we must have changed $\zeta_i^s(f^{-1}(j))$ by an odd number to get $\zeta_i^{s'}(f^{-1}(j))$.

We look at how removing the vertices in R from the open graph (G, I, O) changes $\zeta_i^s(f^{-1}(j))$. Removing a vertex v changes the number of Z-paths from i to $f^{-1}(j)$ if by removing it we also remove an edge in the Z-correction graph G_Z. Let this removed edge be (k, l), then Corollary 6.1 implies that $l \in N(f(k))$ and v has to be either k, l or $f(k)$. Corollary 6.1 also implies that for v to have an outgoing edge in G_Z, $f(v)$ has to be defined. Since f is not defined for output vertices and $v \in R \subseteq O$, there cannot be any outgoing edges from v. Therefore v cannot be k as there is an edge from k to l in G_Z. Also v cannot be l since again v cannot have an outgoing edge in G_Z, hence v would have to be the last element on the Z-path from i to $f^{-1}(j)$, which is $f^{-1}(j)$. This is impossible, as $f^{-1}(j)$ cannot be an output vertex. Therefore the only possibility is that $v = f(k)$.

Let v be the first vertex removed from G, such that $\zeta_i^s(f^{-1}(j))$ changes by an odd number. Hence all the paths from i to $f^{-1}(j)$ that disappear due to removal of v have to go through $f^{-1}(v)$. Therefore there must also exist an odd number of paths from $f^{-1}(v)$ to $f^{-1}(j)$. We know that because of Proposition 7.3, $j \in s(f^{-1}(v))$ and $f^{-1}(v) \prec_s j$. On the other hand because of Definition 8.1 it must also hold that $f^{-1}(v) \in V_1^{\prec_s}$, which together with Definition 6.4 implies that $j \in V_0^{\prec_s} = O$. This leads to a contradiction, because j has to be in $O'^C \subseteq O^C$ and cannot be in O. Therefore property (1) must be true for $(s', \prec_{s'})$.

Now we show that property (2) has to hold. According to Lemma 7.3:

$$Odd(s'(i)) \cap O'^C = \{i\} = Odd(s(i)) \cap O^C. \tag{8.16}$$

Because $i \in O'^C \subseteq O^C$, it must also hold that

$$Odd(s'(i)) \cap O'^C = \{i\} = Odd(s(i)) \cap O'^C, \tag{8.17}$$

and property (2) has to be true for $(s', \prec_{s'})$. □

In the specific case where $|I| = |O|$, it will follow from Lemma 8.11 that the unique SSF of a reduced open graph is the reduced gflow of the original SSF.

Corollary 8.1. *Let (G, I, O) be an open graph with an SSF $(s, \prec_s)$ such that $|I| = |O|$. If (G', I, O') is the reduced open graph, according to $(s, \prec_s)$, of (G, I, O), then the unique SSF $(s', \prec_{s'})$ of (G', I, O') has the following properties:*

1. $\forall i \in O'^C \quad s'(i) \cap O'^C = s(i) \cap O'^C$,
2. $\forall i \in O'^C \quad Odd(s'(i)) \cap O'^C = Odd(s(i)) \cap O'^C$.

Proof. Because of Lemma 8.10 (G', I, O') has a flow. Since $|I| = |O|$, then according to Lemma 8.6 $|I| = |O'|$ and hence $(G'.I, O')$ has a unique flow [85]. Flow is required for the existence of an SSF according to Proposition 7.3, therefore there can exist only one SSF and because of Lemma 8.1 this SSF has to satisfy properties (1) and (2). □

8.3 Moving back

We have proven that the penultimate layers of SSF and optimal gflow are equal if $|I| = |O|$. Then we showed how to remove some vertices from the open graph and construct an SSF and optimal gflow on the new reduced graph. Both of them are reduced gflows, a property which we will use in this section to show that they preserve the layering of the gflows they were derived from.

Lemma 8.12. *Let (G, I, O) be an open graph with SSF $(s, \prec_s)$ and gflow $(g, \prec_g)$ such that (G', I, O') is the reduced open graph of (G, I, O) with the removed vertices set R. For every reduced gflow $(g', \prec_{g'})$ of (G', I, O') such that $V_0^{\prec_s} = V_0^{\prec_g} = O$ and $V_1^{\prec_s} = V_1^{\prec_g}$ it must hold that*

$$\forall n \geq 0 \quad \cup_{k=0}^{n} V_k^{\prec_{g'}} = \cup_{k=0}^{n+1} V_k^{\prec_g} \setminus R. \tag{8.18}$$

Proof. We prove Lemma 8.12 by induction and showing first that equation 8.18 holds if $n = 0$, *i.e.* we need to prove that

$$V_0^{\prec_{g'}} = (V_1^{\prec_g} \cup V_0^{\prec_g}) \setminus R. \tag{8.19}$$

Lemma 6.2 tells that $V_0^{\prec_{g'}} = O'$ and $V_0^{\prec_g} = O$. Because the penultimate layers of SSF $(s, \prec_s)$ and gflow $(g, \prec_g)$ are equal we also have that $V_1^{\prec_s} = V_1^{\prec_g}$. Now we take

the definition of O' from Definition 8.1 of the reduced open graph and substitute the appropriate sets:

$$O' = (V_1^{\prec_s} \cup O) \setminus R \quad \Rightarrow V_0^{\prec_{g'}} = (V_1^{\prec_g} \cup V_0^{\prec_g}) \setminus R. \tag{8.20}$$

Thus equation 8.18 holds for $n = 0$. For the induction step we assume that equation 8.18 holds for $n = m - 1$, *i.e*

$$\cup_{k=0}^{m-1} V_k^{\prec_{g'}} = \cup_{k=0}^{m} V_k^{\prec_g} \setminus R, \tag{8.21}$$

and show that it holds for $n = m$. We use contradiction and assume that

$$\cup_{k=0}^{m} V_k^{\prec_{g'}} \neq \cup_{k=0}^{m+1} V_k^{\prec_g} \setminus R. \tag{8.22}$$

There are two possibilities: either $\exists i \in V_m^{\prec_{g'}} \setminus V_{m+1}^{\prec_g}$ or $\exists i \in V_{m+1}^{\prec_g} \setminus V_m^{\prec'_g}$. We note that according to Lemma 8.9 $i \prec_g j \Leftrightarrow i \prec_{g'} j$ if $i \in O'^C$ and $j \in O'^C$. Because $V_m^{\prec_{g'}} \subseteq O'^C$ for every $m > 0$ we have that

$$\begin{aligned} \exists i \in V_m^{\prec_{g'}} \setminus V_{m+1}^{\prec_g} \Rightarrow \quad & \exists j \in V_{m+1}^{\prec_g} \cap g'(i) \quad s.t. \quad i \prec_g j \quad \Rightarrow \qquad (8.23) \\ \Rightarrow \quad & i \prec_{g'} j \quad \Rightarrow \quad j \in \cup_{k=0}^{m-1} V_k^{\prec_{g'}} = \cup_{k=0}^{m} V_k^{\prec_g}, \\ \exists i \in V_{m+1}^{\prec_g} \setminus V_m^{\prec_{g'}} \quad \Rightarrow \quad & \exists j \in V_m^{\prec_{g'}} \cap g(i) \quad s.t. \quad i \prec_{g'} j \quad \Rightarrow \qquad (8.24) \\ \Rightarrow \quad & i \prec_g j \quad \Rightarrow \quad j \in \cup_{k=0}^{m} V_k^{\prec_g} = \cup_{k=0}^{m-1} V_k^{\prec_{g'}}. \end{aligned}$$

As can be seen above, both of the possible cases lead to a contradiction and hence it must hold that

$$\cup_{k=0}^{m} V_k^{\prec_{g'}} = \cup_{k=0}^{m+1} V_k^{\prec_g} \setminus R. \tag{8.25}$$

This completes the induction step and the proof itself. □

From the previous lemma we can construct a proof saying that every layer of a reduced gflow starting from the second to last one is equal to a layer of the original gflow.

Corollary 8.2. *Let (G, I, O) be an open graph with SSF $(s, \prec_s)$ and gflow $(g, \prec_g)$ such that (G', I, O') is the reduced open graph of (G, I, O) with the removed vertices set R. For every reduced gflow $(g', \prec_{g'})$ of (G', I, O') such that $V_0^{\prec_s} = V_0^{\prec_g} = O$ and $V_1^{\prec_s} = V_1^{\prec_g}$ it must hold that*

$$\forall n > 0 \quad V_n^{\prec_{g'}} = V_{n+1}^{\prec_g}. \tag{8.26}$$

Proof. This follows trivially from Lemma 8.12. We have that if $n > 0$:

$$\cup_{k=0}^{n} V_k^{\prec_{g'}} = \cup_{k=0}^{n+1} V_k^{\prec_g} \setminus R, \tag{8.27}$$

$$\cup_{k=0}^{n-1} V_k^{\prec_{g'}} = \cup_{k=0}^{n} V_k^{\prec_g} \setminus R. \tag{8.28}$$

We can now subtract the elements of the second set from the first.

$$\cup_{k=0}^{n} V_k^{\prec_{g'}} \setminus \cup_{k=0}^{n-1} V_k^{\prec_{g'}} = (\cup_{k=0}^{n+1} V_k^{\prec_g} \setminus R) \setminus (\cup_{k=0}^{n} V_k^{\prec_g} \setminus R) \quad \Rightarrow \quad V_n^{\prec_{g'}} = V_{n+1}^{\prec_g} \setminus R. \tag{8.29}$$

Because Definition 8.1 of SSF reduced open graph we know that $R \subseteq O$. Lemma 6.2 says that $O = V_0^{\prec_g}$, hence we know that no element in R can be included in $V_{n+1}^{\prec_g}$ for $n \geq 0$ and $V_n^{\prec_{g'}} = V_{n+1}^{\prec_g}$. □

It turns out, that if the gflow we have for the original open graph is the optimal one, then the reduced gflow will be optimal for the reduced open graph.

Lemma 8.13 (**Constructing the optimal reduced gflow**). *Let (G, I, O) be an open graph with SSF $(s, \prec_s)$ and optimal gflow $(g, \prec_g)$. If (G', I, O') is the reduced open graph of (G, I, O) then the reduced gflow $(g', \prec_{g'})$ of $(g, \prec_g)$ is the optimal gflow of (G', I, O').*

Proof. First, because of Lemma 8.9 $(g', \prec\ g')$ has to be a gflow of (G', I, O'). Let us assume that $(g', \prec_{g'})$ is not the optimal gflow of (G', I, O') and let $(d, \prec_d)$ be the optimal one. Then according to Definition 6.6

$$\exists n > 0, i \in O'^C \quad s.t. \quad i \in V_n^{\prec_d} \setminus V_n^{\prec_{g'}} \quad \wedge \quad \forall k < n \quad V_k^{\prec_{g'}} = V_k^{\prec_d}, \tag{8.30}$$

and since $d(i) \in \cup_{k=0}^{n-1} V_k^{\prec_d} = \cup_{k=0}^{n-1} V_k^{\prec_{g'}}$ from Lemma 8.12 we obtain $d(i) \in \cup_{k=0}^{n} V_k^{\prec_g} \setminus R$, where the R is the set of vertices removed from the original graph. Now we know from Definition 6.6 that i is in $V_{n+1}^{\prec_g}$ which according to Corollary 8.2 must be equal to $V_n^{\prec_{g'}}$. This leads to a contradiction because $i \in V_n^{\prec_d} \setminus V_n^{\prec_{g'}}$ and thus $(g', \prec_{g'})$ has to be the optimal gflow of (G', I, O'). □

8.4 Proof of the optimality theorem

We can now prove Theorem 8.1 by showing that the vertex layering of any SSF and an optimal gflow is exactly the same. Let (G, I, O) be an open graph with flow $(f, \prec_f$) such that $|I| = |O|$. Let $(s, \prec_s)$ be the SSF obtained from $(f, \prec_f)$ according to Proposition 7.3 and $(g, \prec_g)$ the optimal gflow of (G, I, O). According to Proposition 7.3

the last layer of any SSF is the set of output vertices. Lemma 6.2 says that this is also true for the last layer of an optimal gflow, therefore $V_0^{\prec_s} = V_0^{\prec_g} = O$. The layers $V_1^{\prec_s}$ and $V_1^{\prec_g}$ are equal because of Lemma 8.5.

Now we need to show that layers $V_n^{\prec_s}$ and $V_n^{\prec_g}$ are equal for $n > 1$. We can construct a reduced open graph (Definition 8.1) (G', I, O') from (G, I, O). We now consider the unique SSF $(s', \prec_{s'})$ and reduced gflow $(g', \prec_{g'})$ of (G', I, O'), which according to Lemma 8.13 is optimal. According to Lemma 8.12, $V_n^{\prec_{g'}} = V_{n+1}^{\prec_g}$ for every $n > 0$ and because SSF is by Theorem 7.1 a gflow the same lemma also implies that $V_n^{\prec_{s'}} = V_{n+1}^{\prec_s}$. Because of the way a reduced open graph is defined, we know that $|I| = |O'|$ (Lemma 8.6). Thus we can again use Lemma 8.5 to say that $V_2^{\prec_g} = V_1^{\prec_{g'}} = V_1^{\prec_{s'}} = V_2^{\prec_s}$. We can now take (G', I, O') and find its reduced open graph to show using the same technique that $V_3^{\prec_g} = V_3^{\prec_s}$. This can be continued until we reach the empty layers, in which case we have considered all the layers according to $(s, \prec_s)$ and $(g, \prec_g)$. As every layer of $(s, \prec_s)$ and $(g, \prec_g)$ will be equal and $(g, \prec_g)$ is the optimal gflow, the SSF of a flow of an open graph (G, I, O) is an optimal gflow if $|I| = |O|$, which proves Theorem 8.1.

The above theorem together with Algorithm 1 for signal shifting implies a new method for finding maximally delayed flows on graphs with flow. In particular, if an open graph has flow and its input size equals output size, the maximally delayed gflow can be found in $O(n^3)$ steps. This is more efficient in the number of operations than previous best known algorithm which completes in $O(n^4)$ steps [9], but can find the maximally delayed gflow (if it exists) of arbitrary open graphs.

Corollary 8.3. *The maximally delayed gflow on an open graph (G, I, O) with flow $(f, \prec_f)$ and $|I| = |O|$ can be found in $O(n^3)$ computational steps, where n is the number of vertices in G.*

Proof. We prove that the following process gives us the maximally delayed gflow of (G, I, O) in $O(n^3)$ steps, by showing that each step in the process takes at most $O(n^3)$ elementary operations.

1. Find the unique flow $(f, \prec_f)$ using $O(n^2)$ operations.
2. Create the flow pattern (equation 6.9) of $(f, \prec_f)$ using $O(n^2)$ operations.
3. Signal shift the resulting pattern using $O(n^3)$ operations.
4. Construct the SSF from the signal shifted pattern suing $O(n^2)$ operations.

Step (1) can be completed using $O(n^2)$ operations with the algorithm from [9] and Step (3) takes no more than $O(n^3)$ operations if Algorithm 1 is used. To estimate

the number of operations required for steps (2 and 4) we first estimate the number of commands in a gflow pattern, since flow is a stricter version of gflow this upper bound holds also for flow.

- One preparation command N_i for every vertex $i \in I^C$ — total of $O(n)$ preparation commands.
- One entanglement command $E_{i,j}$ for for every edge $E_{i,j} \in E(G)$ — total of $O(n^2)$ entanglement commands.
- One measurement command $M_i^{\alpha_i}$ per vertex $i \in V(G)$ — total of $O(n)$ measurement commands.
- Up to n X- and Z-correction commands per vertex in $V(G)$ — total of $O(n^2)$ correction commands.

The non-correction commands for the pattern in flow Step (2) can be created by processing the open graph and the correction commands by parsing a description of the flow. Since the open graph has at most $O(n^2)$ elements (n vertices and $O(n^2)$ edges) the non correction commands can be created in $O(n^2)$ steps. The measurement commands should be added to the pattern respecting the flow partial ordering. This can be done in $O(n \log n)$ steps by first sorting the vertices using any reasonable sorting algorithm. After the non-correction commands are added to the pattern, the correction commands can be inserted before the measurement command acting on the vertex they act on. These can be read from the description of the flow (equation 6.9) and since there is a total of $O(n^2)$ of them the whole Step (2) takes $o(n^2)$ operations.

The gflow $(g, \prec_g)$ in Step (4) can be created with one read over the measurement pattern (equation 7.12) resulting from Step (3). This can be done by taking $j \in s(i)$ if and only if there exists an $X_j^{s_i}$ command in the pattern and $i \prec_g j$ if and only if there exists a $X_j^{s_i}$ or $Z_j^{s_i}$ command in the pattern. Since there are at most $O(n^2)$ commands in the pattern, the gflow can be constructed in $O(n^2)$ operations. Finally, because of Theorem 8.1 the constructed SSF is a maximally delayed gflow

Chapter 9

Discussion and Results

Initially, 1WQC was proposed as an alternative architecture for the implementation of quantum computing. However, from early on the distinct parallel power of the model attracted researchers to explore further this unique feature of the model. In a series of results the key concepts of flow, signal shifted flow, gflow, maximally delayed gflow, focused gflow, and information preserving flow were introduced [15, 14, 16, 9, 83]. They address the general question of determinism in 1WQC, while shedding light on the parallelism as well. Although it is now known that the parallel power of 1WQC is equivalent to the quantum circuit with unbounded fan-out [11], further investigation is required to fully take advantage of this extra power. In this part of the thesis we continued this line of research by presenting a surprising link between signal shifted flow and maximally delayed gflow. The surprise comes from the fact that the former is obtained via a simple rewrite rules of pushing the Z-dependencies of a pattern to the end of the computation, while the latter is constructed directly from the structure of the underlying graph. This leads to a new efficient procedure for finding the maximally delayed gflow of graphs with flow, as was discussed in the last section. The structure of the flows for which our algorithm produces a maximally delayed gflow if $|I| \neq |O|$ remains an open question.

Moreover, the link between signal shifted flow and maximally delayed gflow opens a new direction to unify the "flow" structure further and fully characterise the constructions behind the parallel power of 1WQC. The techniques presented in this work have already been used to translate computations in the 1WQC model to circuit model witho neither increasing the depth of the computation nor introducing any auxiliary qubits [17]. Further application of the different techniques introduced in this paper to known quantum algorithms, as well as a full comparison with other known optimisation

methods beyond 1WQC constitute an interesting subject to be explored in the future. We end this part of the thesis with a few concrete open questions that could benefit from the 1WQC properties proven in this thesis.

1. **Removal of auxiliary qubits from quantum circuits corresponding to parallelised 1WQC patterns.** The usual translation of 1WQC patterns to quantum circuits introduces a number of auxiliary qubits to the circuits. The method for translating from 1WQC to quantum circuits without those auxiliary qubits has existed for a while, but the depth of the computation was not considered. [86] To benefit from the 1WQC parallelisation techniques, a method which preserves the depth while not adding any auxiliary qubits is required. One such method has recently been proposed by Miyazaki *et al.* [17] using the techniques introduced in a preprint of the work in Part II of this thesis. There is another approach to solving this problem, namely da Silva *et al.* proposed in [87] a set of rewrite rules for quantum circuits that can in some instances be used to remove these auxiliary qubits. We worked together with da Silva on this question, using the rigid structure of SSF [18]. The goal of the collaboration, which has not been achieved yet, was to show that these rewrite procedures result in a circuit with no auxiliary qubits if the underlying open graph has SSF.

2. **Extending graphs to stepwise strongly uniform determinism by adding and/or removing edges.** Consider a scenario where a group of experimentalists can create a specific 1WQC resource state corresponding to an open graph. The structure of the graph is limited by their experiment and they would like to implement deterministic computations with it. Assume that the graph does not have gflow, hence stepwise strongly uniformly deterministic computations are impossible. A graph with gflow could be creatable through minor modifications to their experiment, which they can do, as opposed to using a completely new set-up. It would be beneficial if they could find out what kind of modifications they could do to obtain a resource with gflow. This motivates the following question: does there exist an algorithm which can in polynomial time give the minimum number of changes for any open graph to be transformed to an open graph with gflow by adding and removing edges?

Part III

Quantum Circuit Complexity

When designing new parallel algorithms it is beneficial to know how much an algorithm can be parallelised, *i.e.* what is the lowest possible depth of a circuit implementing it. If this is not known, one might be searching for a parallel algorithm for a problem which does not have one, or trying to reduce the depth of a parallel algorithm which is already as parallel as possible. We address this problem through examining the parity function on quantum computers and establishing new bounds on the depth of circuits implementing it in the quantum computing model. This is done from a complexity theoretical point of view, starting with a review of the classical and quantum circuit complexity classes in Chapter 10. It is known that in the classical computing model parity cannot be computed with constant depth circuits with unbounded fan-in AND and OR gates [88, 23, 89, 90]. This class of circuits is called *constant depth alternating circuits* and is denoted as AC^0. On the other hand, computing parity is possible when additional unbounded modulo 2 gates are allowed, resulting in the AC^0 with modulo 2 complexity class, denoted with $\mathsf{AC}^0[2]$. This result is not known to hold for the analogous quantum classes QAC^0 and $\mathsf{QAC}^0[2]$ and proving a similar result $\mathsf{QAC}^0 \neq \mathsf{QAC}^0[2]$ is the motivation of this part. In Chapter 12, two new results are proven regarding the relationship between these two classes. First, it is shown that the parity of more than two qubits cannot be computed probabilistically with QAC circuits having only one layer of multi-qubit gates. Second, the QAC^0 circuits that have two layers of multi-qubit gates cannot compute parity of more than five qubits exactly. These two results are a step towards our ultimate goal, which we intend to pursue beyond this thesis — the proof of $\mathsf{QAC}^0 \neq \mathsf{QAC}^0[2]$.

Chapter 10

Preliminaries

10.1 Classical Low-Depth Complexity Classes

The results of this part focus on quantum complexity classes, but it is useful to have an overview of the classical counterparts to see where exactly the quantum classes are in the complexity classes' hierarchy. This section gives the definitions and relations between the classical circuit complexity classes, followed by their quantum counterparts' descriptions in the next section. Most of the definitions of the classical classes here are taken from [31], with minor differences in formulation and notation for consistency with the rest of this thesis, especially considering the definitions of the quantum complexity classes in the next section. Important concepts in these definitions are the depth of a circuit and the fan-in of gates, defined in Section 1.1. The depth of a Boolean circuit is the longest path from an input to the output and fan-in is the number of inputs to a gate. The number of inputs in an unbounded fan-in gate can be arbitrarily large, contrary to a bounded fan-in gate, where this is fixed to a constant (usually two). In circuit complexity, inputs of different size are usually processed by separate circuits. Computational problems are hence associated with families of circuits (Definition 1.3), which contain a distinct circuit for each input size. Sometimes a uniformity constraint is imposed on the circuit families, the most common are the polynomial-time and logspace uniform families of circuits.

Definition 10.1 (polynomial-time uniform circuit families [31])**.** *A circuit family* $\{C_n\}$ *is polynomial-time uniform if there exists a polynomial-time turing machine that on input* 1^n *outputs the description of the circuit* C_n*.*

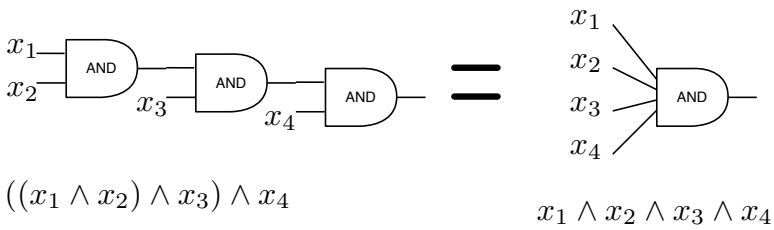


Figure 10.1: Subsequent AND gates can be merged into one AND with a larger input size., but this is obviously not possible if the fan-in of gates is limited. Similar equivalence holds for the OR gates.

Definition 10.2 (logspace uniform circuit families)**.** *A circuit family $\{C_n\}$ is logspace uniform if there exists a turing machine that uses logarithmic space and on input 1^n outputs the description of the circuit C_n.*

We start with the most basic circuit complexity class, the class which does not allow any unbounded fan-in gates: the Nick's Class (NC), named after Nick Pippenger, who first defined this class [31].

Definition 10.3 (Nick's Class (NC) [31])**.** *For every d, a language L is in NC^d if L can be decided by a logspace uniform family of circuits $\{C_n\}$ where C_n has poly(n) size, depth $O(\log^d n)$, and the gates have bounded fan-in. The class $\mathsf{NC} = \cup_{i\geq 0}\mathsf{NC}^i$.*

When the restriction of bounded fan-in is lifted from NC, we get the next major circuit class: the Alternating Circuits AC, the name referring to alternations between AND and OR gates in the circuit.

Definition 10.4 (Alternating Circuits (AC) [31])**.** *For every d, a language L is in AC^d if L can be decided by a family of circuits $\{C_n\}$ where C_n has poly(n) size, depth $O(\log^d n)$, and the gates are allowed to have unbounded fan-in. The class $\mathsf{AC} = \cup_{i\geq 0}\mathsf{AC}^i$.*

Since unbounded fan-in is allowed in AC, sequential gates of the same type can always be merged into one larger gate (demonstrated in Figure 10.1), the gates therefore always alternate between AND and OR and hence the name of the class. It is known that for each $i \in \mathbb{N}$ [31]

$$\mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}, \tag{10.1}$$

and the only known strict inclusions in the above sequence are $\mathsf{NC}^0 \subset \mathsf{AC}^0$ and $\mathsf{AC}^0 \subset \mathsf{NC}^1$ [31]. The next complexity classes considered in this work are obtained by adding MOD_q gates to the set of gates allowed in the AC^0 class.

Definition 10.5 (The MOD_m gate [31])**.** *For any integer q, the MOD_q gate outputs 0 if the sum of its inputs is 0 modulo q, and 1 otherwise.*

The Mod_q gates can be interpreted as counting gates since they return true if and only if the number of input bits equals a fixed count q, thus the name of the next complexity class: AC with Counters (ACC).

Definition 10.6 (AC with Counters (ACC^0) [31])**.** *For every $q > 1$, a language L is in $\mathsf{AC}^0[q]$ if L can be decided by a family of circuits $\{C_n\}$ where C_n has poly(n) size, constant depth, and consists of unbounded fan-in AND, OR, NOT, and MOD_q gates. The class $\mathsf{ACC}^0 = \cup_{i>1}\mathsf{AC}^0[i]$.*

For two distinct primes p and q, $\mathsf{AC}[q] \neq \mathsf{AC}[m]$, where m is a power of p [19, 20]. It is also known that $PARITY$, which is a MOD_q gate with $q = 2$ cannot be computed in AC^0, thus the classes AC^0 and $\mathsf{AC}^0[2]$ cannot be equal [23]. The final classical complexity class included in this work is obtained by adding the unbounded $MAJORITY$ gates to the allowed set of gates in AC.

Definition 10.7 (The $MAJORITY$ gate [20])**.** *The majority gate outputs one if and only if at least half of the inputs are ones.*

Definition 10.8 (Threshold Circuits (TC) [91])**.** *For every d, a language L is in TC^d if L can be decided by a family of circuits $\{C_n\}$ where C_n has poly(n) size, depth $O(\log^d n)$, and consisting of unbounded fan-in AND, OR, NOT, and majority gates. The class $\mathsf{TC} = \cup_{i\geq 0}\mathsf{TC}^i$.*

The class TC^0 is known to include ACC^0 [21] and to be a subset of NC^1 [21], but neither of these inclusions is known to be strict. We thus have the following hierarchy of classical low-depth complexity classes

$$\mathsf{NC}^0 \subset \mathsf{AC}^0 \subset \mathsf{AC}^0[2] \subset \mathsf{ACC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1. \tag{10.2}$$

10.2 Quantum Low-Depth Complexity Classes

Before defining the quantum circuit complexity classes, the concept of *clean computations* needs to be explained. Some of the known results described later and one of the main results of this work apply only to clean computations.

Definition 10.9 (Clean computations [8])**.** *We say that a quantum circuit C cleanly computes a unitary operator U if for any $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$*

$$\langle y_1 \cdots y_n 0 \cdots 0 | C | x_1 \cdots x_n 0 \cdots 0 \rangle = \langle y_1 \cdots y_n 0 \cdots 0 | U \otimes I | x_1 \cdots x_n 0 \cdots 0 \rangle. \tag{10.3}$$

The benefit of clean computations can be seen when considering a quantum algorithm and its circuit A, which consists of multiple sub-circuits $C_1, C_2, \ldots, C_n$ applied in sequence. Consider the case when all of the sub-circuits perform clean computations. Then the final state of these qubits after each sub-circuit is the all zero computational basis state $|0 \cdots 0\rangle$ and the auxiliary qubits of one sub-circuit can be used as the auxiliary qubits of the next one. Thus, if $aux(C)$ is the number of auxiliary qubits used by a quantum circuit C, the total number of auxiliary qubits required for A is

$$aux(A) = \max_{i=1}^{n} aux(C_i). \tag{10.4}$$

On the other hand, if none of the circuits $C_1, C_2, \ldots, C_n$ performs a clean computation, the auxiliary qubits cannot be reused since quantum circuits in general expect the auxiliary qubits to be initialised in the all zero computational basis state $|0 \cdots 0\rangle$ [4]. The total number of auxiliary qubits in A then becomes

$$aux(A) = \sum_{i=1}^{n} aux(C_i). \tag{10.5}$$

It is of course possible that only some of the sub-circuits are clean. Then the auxiliary qubits of the clean circuits can be reused, whereby the auxiliary qubits of non-clean circuits needs cannot. Thus the benefit of clean computations comes from the reusability of the auxiliary qubits. From complexity theoretical point of view, the restriction on the final state of the auxiliary qubits can simplify the analysis of the circuits, as can be seen in the later parts of this thesis.

The quantum language classes are defined in two parts. First, several classes of quantum circuit families (Definition 1.6) are defined. These classes impose restrictions on the allowed gates and the depth of the circuits. Whereby the depth of Boolean circuits is defined as the longest path from inputs to outputs, the depth of quantum circuits (Definition 1.5) is the number of layers (containing simultaneously applicable gates) in the circuit. Second, the classes of languages are defined on these circuit families classes. These classes differ in the probabilities by which an input is accepted or rejected. This approach was also used in [5] and offers a succinct way to describe of a large number of complexity classes with different acceptance probabilities and restrictions on the circuit.

The first class of quantum circuit families, initially defined in [4], is the analogue to the NC class. The definition included in this work is taken from [6].

Definition 10.10 (Quantum NC (QNC) [6])**.** QNC^d *is the class of quantum circuit families* $\{C_n\}_{n\in\mathbb{N}}$ *for which there exists a polynomial* p *such that each* C_n *contains at most* n *input qubits and at most* $p(n)$ *auxiliary qubits. Each* C_n *has depth* $O(\log^d n)$ *and uses only single-qubit and CNOT gates. The single qubit gates must be from a fixed finite set. The class of quantum circuit families* QNC *is* $\cup_{i\geq 0}\mathsf{QNC}^i$*.*

The definition of the quantum counterpart of the AC class requires the quantum equivalent of the classical unbounded AND (or OR) gate. From the Definition 1.10 of the unbounded Toffoli gate it can be seen that if the target qubit b is initialised to 0, the final state of b will contain the AND of all the remaining qubits. Thus the Toffoli gate is used as the quantum counterpart of the classical AND gate in the definition of the quantum AC (QAC) class. The definition included here is from [5] and adapted to match the rest of the thesis.

Definition 10.11 (Quantum AC (QAC) [5])**.** QAC^d *is the class of quantum circuit families* $\{C_n\}_{n\geq 0}$ *for which there exists a polynomial* p *such that each* C_n *contains at most* n *input qubits and at most* $p(n)$ *auxiliary qubits. Each* C_n *has depth* $O(\log^d n)$ *and uses only single qubit and unbounded Toffoli gates. The single qubit gates must be from a fixed finite set. The class of quantum circuit families* QAC *is* $\cup_{i\geq 0}\mathsf{QAC}^i$*.*

Note that the QAC class does not contain alternating layers AND and OR classes like its classical counterpart. Instead, it consists of Toffoli gates, interlaced with single qubit gates. By adding unbounded MOD_q gates to the class QAC we get the quantum counterparts to the $\mathsf{AC}[q]$ and ACC classes.

Definition 10.12 (QACC [5])**.** $\mathsf{QAC}^d[q]$ *is the class of quantum circuit families* $\{C_n\}_{n\geq 0}$ *for which there exists a polynomial* p *such that each* C_n *contains at most* n *input qubits and at most* $p(n)$ *auxiliary qubits. Each* C_n *has depth* $O(\log^d n)$ *and uses only single qubit, unbounded Toffoli and unbounded quantum* MOD_q *gates (as defined in Definition 1.14). The single qubit gates must be from a fixed finite set. The class of quantum circuit families* $\mathsf{QACC}^k = \cup_{i\geq 0}\mathsf{QAC}^k[i]$ *and* $\mathsf{QACC} = \cup_{i\geq 0}\mathsf{QACC}^i$*.*

Surprisingly, Green *et al.* proved that for any $p, q > 1$, MOD_p can be constructed from MOD_q gates for any q in constant depth, which implies that for any $q, k \in \mathbb{N}, q > 1$, $\mathsf{QACC}^k = \mathsf{QACC}^k[q]$ [5]. This is in contrast to the classical classes, where it is known that $\mathsf{AC}[p] \neq \mathsf{AC}[q]$ when p and q are distinct primes [20]. We can also extend the QAC

class with unbounded quantum threshold gates to derive the QTC class, the quantum counterpart to TC.

Definition 10.13 (QTC [7])**.** QTC^d *is the class of quantum circuit families* $\{C_n\}_{n\geq 0}$ *for which there exists a polynomial* p *such that each* C_n *contains at most* n *input qubits and at most* $p(n)$ *auxiliary qubits. Each* C_n *has depth* $O(\log^d n)$ *and uses only single qubit, unbounded Toffoli and unbounded threshold gates. The single qubit gates must be from a fixed finite set. The class of quantum circuit families* QTC *is* $\cup_{i\geq 0}\mathsf{QTC}^i$*.*

Given any of the above defined classes of quantum circuit families, QNC^k, QAC^k, $\mathsf{QAC}^k[q]$, QACC^k, QTC^k, the classes of languages are defined as follows:

Definition 10.14 (Classes of languages)**.** *Let* F *be a class of quantum circuit families.*

- EF *(Exact* F*) is the class of languages* L *such that there exists* $\{C_n\} \in \mathsf{F}$ *such that for every* x

$$x \in L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] = 1, \tag{10.6}$$
$$x \notin L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] = 0. \tag{10.7}$$

- BF *(Bounded error* F*) is the class of languages* L *such that there exists* $\{C_n\} \in \mathsf{F}$ *such that for every* x

$$x \in L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] \geq \frac{2}{3}, \tag{10.8}$$
$$x \notin L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] \leq \frac{1}{3}. \tag{10.9}$$

- RF *(One-sided bounded error* F*) is the class of languages* L *such that there exists* $\{C_n\} \in \mathsf{F}$ *such that for every* x

$$x \in L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] \geq \frac{1}{2}, \tag{10.10}$$
$$x \notin L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] = 0. \tag{10.11}$$

- NF *(Non-deterministic* F*) is the class of languages* L *such that there exists* $\{C_n\} \in \mathsf{F}$ *such that for every* x

$$x \in L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] > 0, \tag{10.12}$$
$$x \notin L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] = 0. \tag{10.13}$$

- PrF *(Probabilistic* F*) is the class of languages* L *such that there exists* $\{C_n\} \in \mathsf{F}$ *such that for every* x

$$x \in L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] > \frac{1}{2}, \tag{10.14}$$

$$x \notin L \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] \leq \frac{1}{2}. \tag{10.15}$$

Note the difference between classes of languages, classes of circuit families, and circuit families. For example, for every input size n, there exists a $O(\log n)$ depth quantum circuit C_n, which uses only CNOT gates, 0 auxiliary qubits, and whose output qubit is $|1\rangle$ if and only if the number of qubits in state $|1\rangle$ in the input is equal to 1 (mod 2) [8]. The circuits $C_1, C_2, \ldots, C_n$ comprise the family of quantum circuits $\{C_n\}$. According to Definition 10.10, this family of quantum circuits belongs to the QNC^d class of quantum circuit families for each $d \geq 1$. Let the language $PARITY$ be defined as follows:

Definition 10.15. $PARITY = \{x \in \{0,1\}^n : \sum_{i=1}^{n} x_1 \pmod 2 = 1\}$

Definition 10.14 can now be used to determine if and to which language class $PARITY$ belongs to. Since the class of circuit families $\{C_n\} \in \mathsf{QNC}^1$ is such that

$$x \in PARITY \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] = 1, \tag{10.16}$$

$$x \notin PARITY \quad \Leftrightarrow \quad \Pr[C_n(x) = 1] = 0, \tag{10.17}$$

it is obvious that $PARITY \in \mathsf{EQNC}^1$. But this is not the only suitable language class, it is easy to see that $PARITY \in \mathsf{BQNC}^1$, $PARITY \in \mathsf{RQNC}^1$, $PARITY \in \mathsf{NQNC}^1$, and $PARITY \in \mathsf{PrQNC}^1$.

In additional to comparing language classes, it is possible to compare classes of circuit families. Since the definitions of the language classes depend on the underlying circuit family classes (Definition 10.14), an equality in between two classes of circuit families A and B implies that all the corresponding language classes are also equal to each other, *e.g* $EA = EB$, $PrA = PrB$, *etc.*

Definition 10.16. *We say that two classes of circuit families* F *ad* G *are equal if for every* $\{C_n\}_{n\in\mathbb{N}} \in F$ *there exists* $\{G_n\}_{n\in\mathbb{N}} \in G$ *such that for every* $i \in \mathbb{N}$ *circuits* C_i *and* G_i *implement the same unitary operator, and vice versa.*

Due to the impossibility of copying quantum states, the quantum complexity classes defined above cannot have unbounded fan-out. This is contrary to their classical counterparts, where unbounded fan-out is allowed. When fan-out in Boolean circuits is

translated to quantum circuits, it is replaced with a $O(\log k)$ depth sub-circuit (where k is the size of the fan-out) consisting of a tree of two qubit CNOT gates [4]. Thus the depth of Boolean circuits increases by $O(\log n)$ when translating to quantum circuits.

It is possible to rewrite Boolean circuits such that the depth is preserved, all the gates have fan-out one, and the unbounded fan-out is performed at the beginning of the circuit on the input bits. This can be done recursively by starting at the first layer and replacing the inputs to a Boolean gate, which are results of fan-out, with a copy of the sub-circuit computing the fanned out value. If c is the maximum fan-in of the gates in a Boolean circuit, then the total number of gates in the circuit and the number of input bits to the first layer is $O(c^d)$, where d is the depth of the circuit. For NC^0, this results in $O(1)$ size circuit acting on $O(1)$ bits. Thus the fan-out required in the beginning of the circuit has a maximum size of $O(1)$, which can be computed in constant depth with a tree of CNOT gates and thus $\mathsf{NC}^0 \subset \mathsf{QNC}^0$. For NC^1 circuits this results in $O(2^{\log n}) = O(n)$ size circuits acting on $O(n)$ bits with $O(n)$ size fan-out at the beginning of the circuit. The $O(n)$ size fan-out can be performed by a CNOT circuit of depth $O(\log n)$, thus the NC^1 circuit with fan-out at the beginning of the circuit can be translated to a $O(\log n)$ depth quantum circuit and $\mathsf{NC}^1 \subset \mathsf{QNC}^1$. For NC^d classes for $d > 1$ this construction does not work, since moving the fan-out to the beginning of the circuit will result in super-polynomial circuit sizes of $O(2^{\log^d n})$, which are not allowed by definition of neither NC^d nor QNC^d. Therefore it is not known whether $\mathsf{NC}^d \subset \mathsf{QNC}^d$ for $d > 1$. For unbounded fan-in circuits AC^0, TC^0, moving fan-out to the beginning of the circuit results in a circuit where the initial fan-out is of size $O(poly(n))$. Translating this to a quantum fan-out consisting of two qubit CNOT gates results in $O(\log n)$ depth quantum circuit, which cannot be computed in neither QAC^0 nor QTC^0. For unbounded fan-in circuits AC^d, TC^d, where $d > 0$ moving fan-out to the beginning of the circuit results in $O(n^{\log^d n})$ size circuits, which is not allowed according to the definitions of the circuit complexity classes, hence it is not known whether $\mathsf{AC}^d \subset \mathsf{QAC}^d$ and $\mathsf{TC}^d \subset \mathsf{QTC}^d$. In classes $\mathsf{QAC}^d[q]$ and QACC, it has been proven that constant-depth quantum fan-out is possible [5] and thus $\mathsf{AC}^d[q] \subseteq \mathsf{QAC}^d[q]$ and $\mathsf{ACC} \subset \mathsf{QACC}$. Augmenting the quantum classes with unbounded quantum fan-out gates, described in Definition 1.12, gives rise to new, surprisingly powerful, complexity classes.

Definition 10.17 (Unbounded fan-out classes [5, 7])**.** *Let F be a class of quantum circuit families. The class F_f, called the unbounded fan-out F, is defined as the class F with addition of the unbounded fan-out gates to the set of allowed gates.*

The power of unbounded fan-out in quantum circuits was first pointed out by Green,

et al. in [5], where they showed that $\mathsf{QAC}^d_f = \mathsf{QAC}^d[2] = \mathsf{QACC}^d$. The first relation ($\mathsf{QAC}^d_f = \mathsf{QAC}^d[2]$) is a consequence of the fact that a parity gate can be implemented with one fan-out gate and two layers of Hadamard gates (see Section 1.2.1). Thus adding unbounded fan-out to the QAC^d class results in the class $\mathsf{QAC}^d[2]$, which was proven to equal QACC^d in the same paper [5]. Their work was the first to highlight an important difference between quantum and classical complexity classes — allowing a quantum operator which is able to fan-out the classical states, *i.e.* computational basis states, in quantum circuits results in a new complexity class. In contrast, unbounded fan-out is allowed for the classical complexity classes. Notably, the classical counterparts, as explained in the previous section, relate to each other as $\mathsf{AC}^0_f \subset \mathsf{AC}^0[2] \subset \mathsf{ACC}^0$, where $\mathsf{AC}^0_f = \mathsf{AC}^0$ since unbounded fan-out is allowed for the AC^0 class.

More surprisingly, Hoyer *et al.* showed that $\mathsf{BQNC}^d_f = \mathsf{BQAC}^d_f$ and $\mathsf{QAC}^d_f = \mathsf{QTC}^d_f$ [7]. Their work left open the question wether every QAC^d_f circuit, regardless of the probability of accepting an input, can be implemented in QNC^d_f. Recently, Takahashi and Tani, solved this problem by proving that it is possible to implement the unbounded OR (and thus also AND) gate exactly with constant depth using only unbounded fan-out and single qubit gates [22], thus $\mathsf{QAC}^d_f = \mathsf{QAC}^d[2] = \mathsf{QNC}^d_f = \mathsf{QTC}^d_f$.

The numerous complexity classes and the relationships between them presented in this chapter are summarised in Figure 10.2. In can be seen from Figure 10.2 that biggest difference of the quantum hierarchy compared to the classical is between the constant depth quantum classes with unbounded fan-out and their classical counterparts:

$$\mathsf{NC}^0_f \subset \mathsf{AC}^0_f \subset \mathsf{AC}^0_f[2] \subset \mathsf{ACC}_f \subseteq \mathsf{TC}^0_f, \tag{10.18}$$

$$\mathsf{QNC}^0_f = \mathsf{QAC}^0_f = \mathsf{QAC}^0_f[2] = \mathsf{QACC}_f = \mathsf{QTC}^0_f, \tag{10.19}$$

where the suffix $_f$ is added to the classical classes to highlight the fact that unbounded fan-out is allowed implicitly.

The unbounded fan-out class QNC^0_f is also the focus of this work. Since parity can be transformed into fan-out with two layers of Hadamard gates, the circuits with unbounded fan-out also contain unbounded parity. Thus one approach to proving that $\mathsf{QAC}^0 \neq \mathsf{QNC}^0_f$ is to show that QAC^0 circuits cannot compute parity. It has been proven, that when the number of auxiliary qubits is less than the size of the input, it is not possible to compute parity exactly and cleanly using QAC^0 circuits, *i.e.* $PARITY$ is not in cleanly computing EQAC^0 with less than n auxiliary qubits [8]. It remains an open question, addressed in this thesis, whether this also holds when the number of auxiliary qubits is polynomial in the number of inputs. Both in the

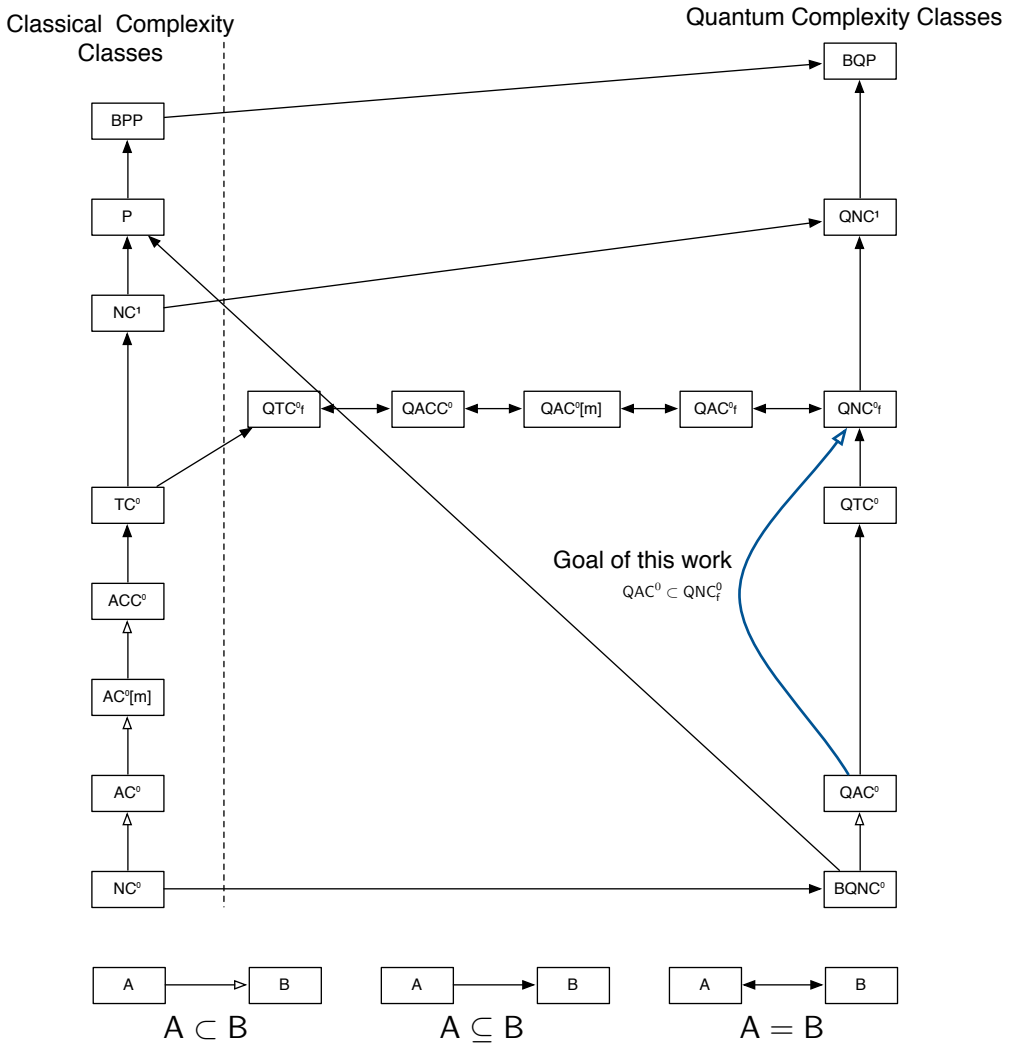


Figure 10.2: The hierarchy of classical and quantum low-depth complexity classes.

proofs of $\mathsf{BQAC}^k \subseteq \mathsf{BQNC}_f^k$ [7] and $\mathsf{QAC}^k \subseteq \mathsf{QNC}_f^k$ [22], the number of auxiliary qubits needed is greater than the input size. Therefore the availability of auxiliary qubits is an important factor in creating parallel quantum circuits. The work in this part of the thesis is pushing towards the proof of $\mathsf{QAC}^0 \neq \mathsf{QNC}_f^0$ by proving that $PARITY$ is not in PrQAC^0 with one layer of multi-qubit gates and not in cleanly computing EQAC^0 with two layers of multi-qubit gates.

Chapter 11

Properties of QAC circuits

For the purpose of this thesis we define a modified version of QAC circuits, the class QAC_a.

Definition 11.1 (QAC_a)**.** QAC_a^d *is the class of quantum circuit families* $\{C_n\}_{n\geq 0}$ *for which there exists a polynomial* p *such that each* C_n *contains at most* n *input qubits and at most* $p(n)$ *auxiliary qubits. Each* C_n *has depth* $O(\log^d n)$ *and contains only single qubit and unbounded* $\wedge Z$ *gates. Arbitrary single qubit gates can be used. The single qubit layers are always interlaced with* $\wedge Z$ *layers, and the first and last layer in a circuit is a single qubit gate layer. The class of quantum circuit families* $\mathsf{QAC}_a = \cup_{i\geq 0}\mathsf{QAC}^i$.

As can be seen from the above definition, the class QAC_a has three main differences from QAC. First, the $\wedge Z$ gates are used instead of Toffoli gates since they do not have any target qubit. Each qubit a $\wedge Z$ gate acts on is treated equally, hence when analysing QAC_a circuits we do not have to consider separately target and control qubits. Second, arbitrary single qubit gates are allowed. This is a more general condition than in QAC, but for our purposes, limiting to fixed gate set does not give any benefit. Third, the single qubit layers are interlaced with two-qubit gates. This rigid structure allows for an easier analysis of the circuits as will be clear in the following sections.

Definition 11.2 (The multi-gate depth)**.** *The multi-gate depth of a quantum circuit is the number of layers containing multi-qubit gates.*

There could be problems solvable by QAC_a circuits with lower depth than is possible using QAC circuits. This is possible since all subsequent single qubit gates in QAC can be replaced by one gate in QAC_a (where arbitrary single qubit gates are allowed), thereby possibly reducing depth of the circuit. Since the depths of QAC and QAC_a

circuits implementing the same unitary operators cannot be assumed to equal, proofs concerning the depth of QAC_a might not be adaptable to QAC. Therefore, we define the *multi-gate* depth, which is more persistent between the two circuit family classes.

Lemma 11.1. *For each* QAC *circuit* C *with depth* d *and multi-gate depth* m *there exists a* QAC_a *circuit with depth* $d' \leq 2d+1$ *and the same multi-gate depth.*

Proof. Let C be a QAC circuit with multi-gate depth d. To rewrite it to a QAC_a circuit we need to

- replace the Toffoli gates with $\wedge Z$,
- guarantee that the circuit starts and ends with a single qubit layer,
- guarantee that single qubit layers are always next to $\wedge Z$ layers and vice versa.
- separate the $\wedge Z$ and single qubit gates to distinct layers.

First, we replace every two sequential single qubit gates U_1U_2 with the gate $V = U_1U_2$. This is done until there are no sequential single qubit gates in the circuit and is allowed since QAC_a circuits can contain arbitrary single qubit gates. Obviously, this does not increase neither the multi-gate nor the regular depth of the circuit. Second, we add a new empty single qubit layer after each existing layer and move all the single qubit gates from the previous layer to it. This doubles the number of layers and hence the depth of the circuit, but separates the Toffoli layers from the single qubit layers and does not increase the multi-qubit depth. Since we add a single qubit layer after each existing layer, we also guarantee that the circuit ends with a single qubit layer. Third, if the first layer of the circuit is not a single qubit layer, we will add a single qubit layer with identity operators to the beginning of the circuit, increasing the depth by one to $2d+1$ and keeping the multi-gate depth the same. Finally, we replace the Toffoli gates with two Hadamard and one $\wedge Z$ gates as shown in Figure 1.9. The H gates introduced can be merged with the single qubit gates in the layers before and after the $\wedge Z$ gate. Thus the replacement of Toffoli with $\wedge Z$ gates will not increase neither the depth nor the multi-gate depth. The circuit now corresponds to the definition of a QAC_a circuit and the overall depth has increased to $2d+1$ and multi-gate depth has stayed the same as in the original circuit C. □

If a problem can be shown to be not computable in multi-gate depth d QAC_a circuit, then the above lemma can be used to show that it is also impossible in multi-gate depth d QAC circuits. This is formalised in the following corollary:

Corollary 11.1. *If a decision problem is not solvable with* QAC_a *circuits of multi-gate depth d, it is also not solvable with* QAC *circuits of multi-gate depth d.*

Proof. Lets assume that a decision problem is not solvable with multi-gate depth d QAC_a circuits but there exists a multi-gate depth d QAC circuit solving it. Then according to Lemma 11.1 there exists a multi-gate depth d QAC_a circuit solving it, contradicting with the initial assumption. □

The above lemma allows us to use the more rigid structure of QAC_a circuits in the later proofs ot prove properties of QAC circuits. In general, the qubits can be either in a computational basis or superposition state before the first $\wedge Z$ gate is applied to them. The following lemma proves that the auxiliary qubits can always be assumed to be in a superposition state. This property simplifies the later analysis of QAC_a circuits.

Lemma 11.2. *Every* QAC *circuit can be rewritten such that all the auxiliary qubits are in a superposition before the first* $\wedge Z$ *gate is applied to them. This rewrite can be done by either removing* $\wedge Z$ *gates or by replacing* $\wedge Z$ *gates with smaller* $\wedge Z$ *gates and removing auxiliary qubits from the circuit.*

Proof. We can simplify the circuit such that every auxiliary qubit is in a non computational basis state before a $\wedge Z$ gate is applied to them. If the auxiliary qubit is in the $|0\rangle$ state, then the $\wedge Z$ gate would never be applied and we can remove the $\wedge Z$ gate from the circuit by using circuit identity in Figure 11.1. Since the $\wedge Z$ gate layers are interlaced with single qubit gate layers, this can be repeated until the auxiliary qubit is in some state other than $|0\rangle$ before a $\wedge Z$ gate. If the auxiliary qubit is in the state $|1\rangle$ before a $\wedge Z$ gate is applied, we can simply replace the $\wedge Z$ with one that acts on the same qubits except for the auxiliary qubits in state $|1\rangle$ (Figure 11.2). This can be done since $\wedge Z$ is a symmetric controlled gate and if a control qubit is in the state $|1\rangle$ independently of the input, we can remove the control from the gate. □

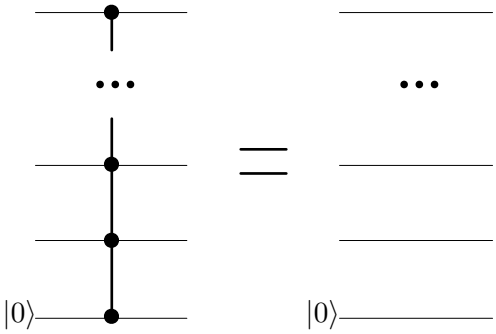


Figure 11.1: If an input to the $\wedge Z_n$ gate is $|0\rangle$, the gate can be removed from the circuit.

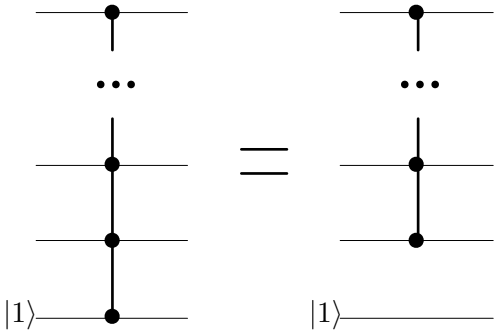


Figure 11.2: If an input to the $\wedge Z_n$ gate is $|1\rangle$, the gate can be replaced it with a $\wedge Z_{n-1}$ gate, which does not act on the qubit in state $|1\rangle$.

Chapter 12

Lower Bounds on Parity

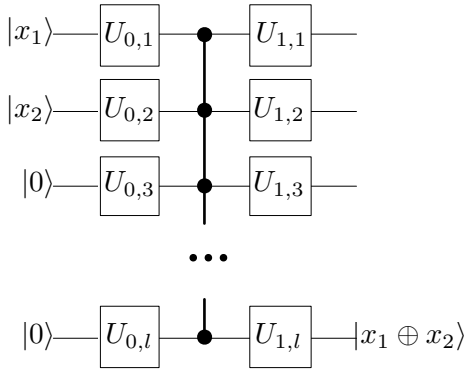
It has been conjectured in [8] that a QAC circuit cleanly computing parity of n inputs needs to have a depth greater than or equal to $2 \log n$. We propose a slightly different conjecture, highlighting the importance of multi-gate layers in a circuit and omitting the requirement for clean computation.

Conjecture 12.1. *A multi-gate depth d* QAC *circuit can compute the parity of at most 2^d inputs exactly.*

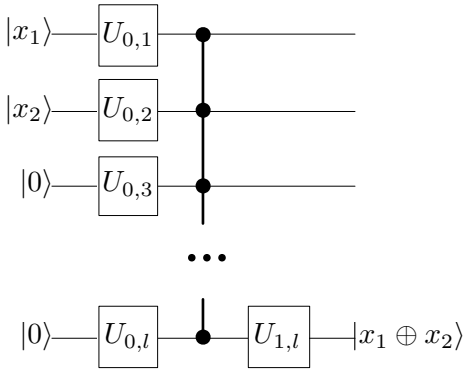
If the above conjecture would be true, it would imply the inequality of the circuit classes QAC^0 and $\mathsf{QAC}^0[2]$, since parity can be computed using one unbounded MOD_2 gate in $\mathsf{QAC}^0[2]$. This chapter presents the two main results of Part III supporting our conjecture. First, we will prove that multi-gate depth one is not enough to even probabilistically compute parity with QAC circuits. This property is initially proven for the QAC_a class, which has a more rigid structure, but later it will be shown that this also holds for the QAC class.

Theorem 12.1. *Multi-gate depth one* PQAC_a *circuits cannot compute parity of more than two qubits.*

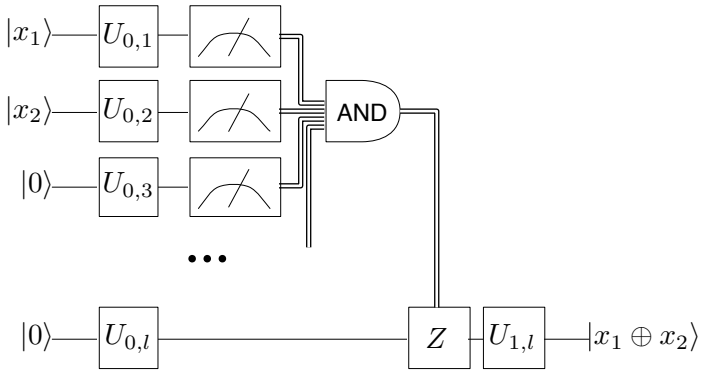
Proof. Assume that it is possible to compute the parity of three qubits x_1, x_2, and x_3 probabilistically with unbounded error using one $\wedge Z$ gate. First, one of the input qubits is fixed to $|0\rangle$ and treated as an auxiliary qubit, thus reducing the circuit to compute the parity of just two qubits. If one of the input qubits is the output qubit, this will be the fixed qubit, otherwise the fixed qubit can be chosen randomly. Without loss of generality it can be assumed that the fixed qubit was x_3 (the qubits can always be relabeled). The resulting circuit will have the structure showed in Figure 12.1(a).



(a) If there exists a depth one probabilistic QAC^0 circuit computing parity, it is possible to construct the above circuit.



(b) Since the gates on the non-output qubits cannot affect the measurement outcome of it, they can be discard them from the circuit.



(c) By applying the principle of delayed measurement, the $\wedge Z$ gate can be replaced with measurements controlling one Z gate.

Figure 12.1: Simplifying a multi-gate depth one QAC_a circuit.

The single-qubit operations in the final layer acting on the non-output qubits can be removed (Figure 12.1(b)) — these operators can not change the measurement outcome of the output qubit (Lemma 1.1). Since a $\wedge Z$ operation is a controlled Z gate, it can be replaced with measurements controlling a Z gate on the output qubit Z (see the principle of delayed measurement in Section 1.2). The simplified circuit will have the structure depicted in Figure 12.1(c).

Let unitary operators $U_{0,1}$ and $U_{0,2}$ be the first unitary operators acting on x_1 and x_2

correspondingly:

$$U_{0,1}|0\rangle = \alpha_1|0\rangle + \beta_1|1\rangle, \tag{12.1}$$
$$U_{0,1}|1\rangle = \alpha_1'|0\rangle + \beta_1'|1\rangle, \tag{12.2}$$
$$U_{0,2}|0\rangle = \alpha_2|0\rangle + \beta_2|1\rangle, \tag{12.3}$$
$$U_{0,2}|1\rangle = \alpha_2'|0\rangle + \beta_2'|1\rangle. \tag{12.4}$$

Since only single-qubit gates are acting on the non-output qubits, they do not get entangled and the probability of each qubit being measured in state $|1\rangle$ is independent of other qubits. The probability of either the qubits $I \cup A \setminus \{x_1, x_2, o\}$ all being measured in state $|1\rangle$ is constant for all of the values of x_1 and x_2; we will denote this probability as a. Taking this into account, the probability that all non-output qubits are measured in state $|1\rangle$, and therefore also the probability of the Z gate being applied, is

$$\Pr(Z|x_1, x_2) = a|\langle 1|U_{0,1}|x_1\rangle|^2|\langle 1|U_{0,2}|x_2\rangle|^2. \tag{12.5}$$

There are only two possible final states for the output qubit o: $U_{1,o}ZU_{0,o}|0\rangle$ and $U_{1,o}U_{0,o}|0\rangle$. The probabilities either of these two states being $|1\rangle$ is $|\langle 1|U_{1,o}ZU_{0,o}|0\rangle|^2 = C_Z$ and $|\langle 1|U_{1,o}U_{0,o}|0\rangle|^2 = C$ correspondingly. The probability of the output qubit being measured in state $|1\rangle$ is therefore

$$(1 - \Pr(Z|x_1, x_2))C + \Pr(Z|x_1, x_2)C_Z = C + \Pr(Z|x_1, x_2)(C_Z - C). \tag{12.6}$$

Since the circuit computes parity of x_1 and x_2 probabilistically, this probability needs to be greater than $1/2$ for $x_1 \neq x_2$ and less than or equal to $1/2$ otherwise. Thus if we consider all possible values for x_1 and x_2 the following inequalities must hold:

$$C + a|\beta_1'|^2|\beta_2|^2(C_Z - C) > 1/2, \tag{12.7}$$
$$C + a|\beta_1|^2|\beta_2'|^2(C_Z - C) > 1/2, \tag{12.8}$$
$$C + a|\beta_1|^2|\beta_2|^2(C_Z - C) \leq 1/2, \tag{12.9}$$
$$C + a|\beta_1'|^2|\beta_2'|^2(C_Z - C) \leq 1/2. \tag{12.10}$$

By adding inequality 12.9 to 12.7 and 12.10 to 12.8 we get the relations:

$$C + a|\beta_1'|^2|\beta_2|^2(C_Z - C) + 1/2 < 1/2 + C + a|\beta_1|^2|\beta_2|^2(C_Z - C), \tag{12.11}$$
$$C + a|\beta_1|^2|\beta_2'|^2(C_Z - C) + 1/2 < 1/2 + C + a|\beta_1'|^2|\beta_2'|^2(C_Z - C), \tag{12.12}$$

which when simplified will lead to a contradiction:

$$|\beta_1'|^2 < |\beta_1|^2, \tag{12.13}$$

$$|\beta_1|^2 < |\beta_1'|^2. \tag{12.14}$$

Thus it cannot be possible to compute parity with $\mathsf{PQAC_a}$ circuits having only one $\wedge Z$ layer. □

The lower bound for parity proven in the above theorem holds for the PQAC_a circuits, which have a more rigid structure than QAC circuits. However, the result can easily be extended to the QAC circuits.

Corollary 12.1. *Parity of more than two qubits cannot be computed probabilistically with multi-gate depth one* QAC *circuits.*

Proof. Corollary 11.1 implies that if a multi-depth one QAC circuit could compute parity of more than two qubits, there would exits a QAC_a circuit having the same multi-gate depth. This is not possible because of Theorem 12.1, hence the corollary must be true. □

The proof ofTheorem 12.1 relies on the qubits being not entangled before the final $\wedge Z$ gate. This allowed to analyse the probability of the $\wedge Z$ gate affecting the final measurement and through that the probability of the final outcome being $|1\rangle$. In multi-gate depth two circuits the $\wedge Z$ gates in the first multi-gate layer can create entanglement, thus this assumption is no longer valid. To show that parity cannot be computed in multi-gate depth two circuits we needed additional restrictions on the circuits: the exactness and cleanness of computations. Note that we conjectured at the beginning of this chapter that the cleanness might not be necessary, removing the cleanness constraint is a possible continuation of this work.

Theorem 12.2. *Parity of more than* 5 *qubits cannot be computed exactly cleanly with multi-gate depth two* QAC *circuits.*

This theorem is proven for multi-gate depth two QAC_a circuits, which through Corollary 11.1 extends to QAC circuits. We start by simplifying the multi-gate depth two QAC_a circuits, whose general structure can be seen in Figure 12.2. First, the single-qubit gates acting on non-output qubits in the final layer can be removed since they cannot influence the measurement outcome of the output qubit (Lemma 1.1). Second, due to the principle of implicit measurement it can be assumed that all the

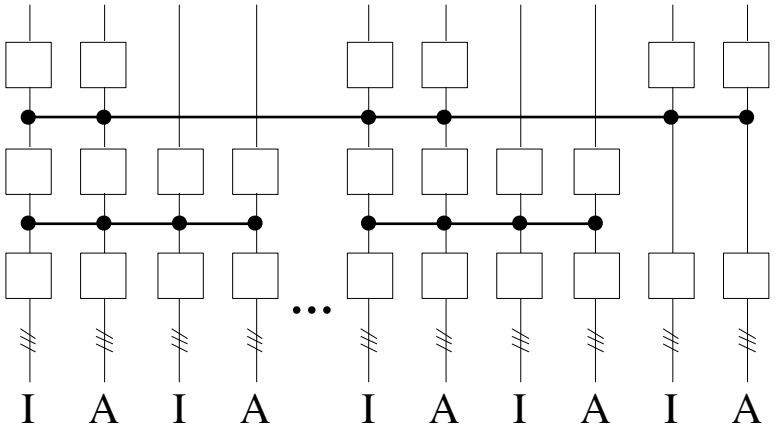


Figure 12.2: The general structure of multi-gate depth two QAC_a circuits. The empty rectangles denote arbitrary single-qubit gates.

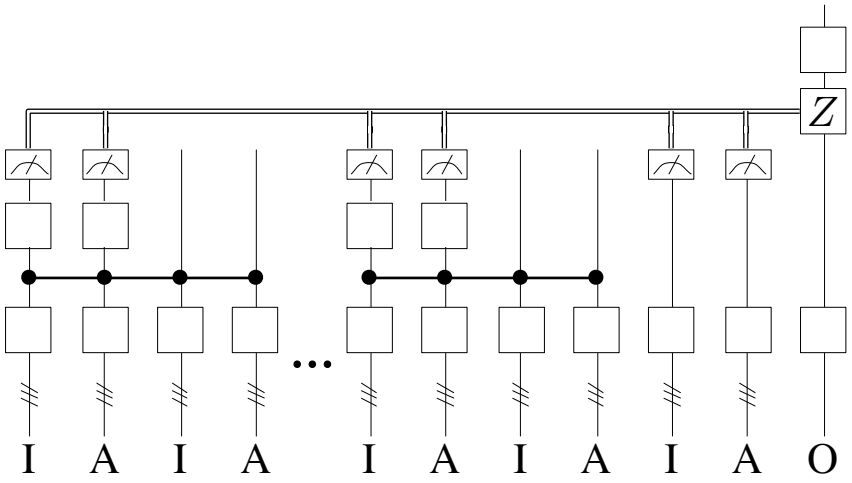


Figure 12.3: The simplified structure of multi-gate depth two QAC_a circuits. All of the non-output qubits acted on by the final $\wedge Z$ in the original circuit are measured, and a Z gate is applied to the output qubit if and only if all the measured qubits were in state $|1\rangle$. The output qubit can be any one of the qubits, but in this figure it is one of the qubits to which no $\wedge Z$ gate is applied in the first multi-gate layer.

non-output qubits are measured. By applying the principle of deferred measurement it is now possible to replace the final $\wedge Z$ with a Z on the output qubit depending on the outcomes of all the other qubits this $\wedge Z$ acts on. After these simplifications we get the structure shown in Figure 12.3, which is used throughout the rest of this thesis. Now, to help us prove Theorem 12.2 we will reveal some properties of multi-gate depth 2 QAC_a circuits.

Lemma 12.1. *In a* EQAC_a *circuit, the final* $\wedge Z$ *gate can be discarded if there exists an input value for which the application of the final* $\wedge Z$ *gate does not influence the final measurement outcome of the output qubit* o*.*

Proof. Let U_o be the final single-qubit gate applied to o. The circuit computes an exact output value and thus the output qubit cannot be entangled. As discussed above, the $\wedge Z$ gate can be seen as a Z gate, which is applied based on the measurement outcome of some non-output qubits. Thus there are exactly four states in which o can be before the final $\wedge Z$:

$$|\psi_1\rangle = U_o^\dagger|1\rangle, \tag{12.15}$$
$$|\psi_2\rangle = ZU_o^\dagger|1\rangle, \tag{12.16}$$
$$|\psi_3\rangle = U_o^\dagger|0\rangle, \tag{12.17}$$
$$|\psi_4\rangle = ZU_o^\dagger|0\rangle. \tag{12.18}$$

If there exists a value for which the application of the last $\wedge Z$ does not affect the final measurement of o, then since unitary operators map orthogonal states to orthogonal states and $|\psi_2\rangle$ and $|\psi_4\rangle$ are orthogonal, the following will hold:

$$|\langle 1|U_oZ|\psi_2\rangle|^2 = |\langle 1|U_o|\psi_2\rangle|^2 \quad \Leftrightarrow \quad |\langle 1|U_oZ|\psi_4\rangle|^2 = |\langle 1|U_o|\psi_4\rangle|^2. \tag{12.19}$$

Since applying Z to $|\psi_2\rangle$ and $|\psi_4\rangle$ does not change the final measurement outcome, we can just discard the final $\wedge Z$ gate altogether. □

The proof works on any depth of EQAC_a circuits, not only depth two ones, which are the focus of this section. To be able to remove the final $\wedge Z$, it is enough to find one input value for which the application of the last $\wedge Z$ does not matter. This allows to further restrict the structure of the EQAC in the following lemmas. Next the the qubits are divided into blocks based on the $\wedge Z$ gates applied in the first layer.

Definition 12.1 ($\wedge Z$ block)**.** *Two qubits in a multi-gate depth two* QAC_a *circuit belong to the same* $\wedge Z$ block*, if and only if there exists a* $\wedge Z$ *in the first multi-gate layer that*

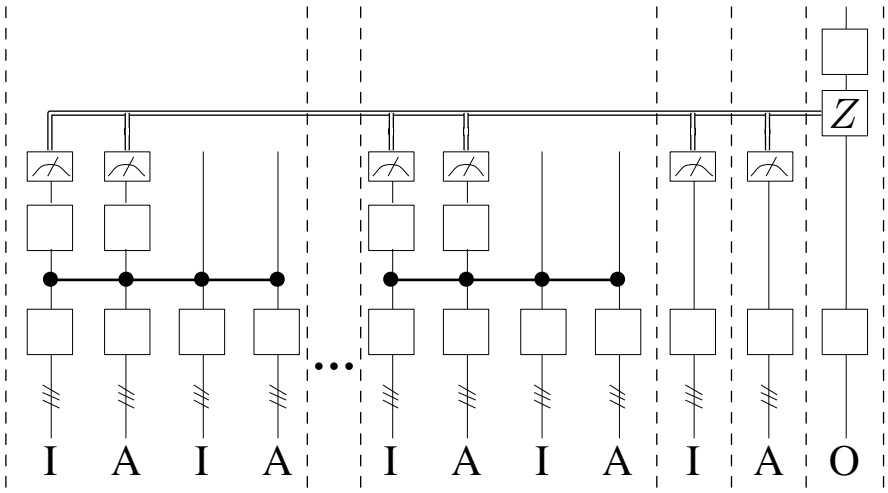


Figure 12.4: An example of a multi-gate depth two circuit divided into blocks as per Definition 12.1. The dashed lines represent borders between distinct blocks.

acts on both of them. The block to which the output qubit belongs to is called the output block. We define the number of blocks in a quantum circuit C as $B_n(C)$ and the number of qubits in the same block as the output qubit as $B_o(C)$.

The $\wedge Z$ blocks are called *blocks* for short in this thesis. The division into blocks, demonstrated in Figure 12.4, and the values of $B_n(C)$ and $B_o(C)$ for a circuit are central to our proofs in the forthcoming lemmas.

Lemma 12.2. *For every input in a multi-gate depth two* EQAC_a *circuit, all the qubits not in the output block must be in a computational basis state before the last* $\wedge Z$*.*

Proof. Lets assume that there exists an input for which one of the non-output block qubits x_i is not in a computational basis state before the last $\wedge Z$. We use the simplified multi-gate depth two QAC_a circuit shown in Figure 12.3. Then the application of Z on the output qubit has a probability of being applied, which depends the measurement outcome of x_i. Lemma 12.1 tells that the application of this Z influences the final outcome, which will be probabilistic since x_i is not in a computational basis state. If x_i would be in the same block as the output qubit, the output could depend on the state of x_i before the last $\wedge Z$ and the probabilistic application of the Z gate could result in a non-probabilistic result. This is not the case, and the probabilistic application of Z results in the final output being probabilistic, which is impossible in EQAC_a circuits. □

Lemma 12.3. *Let C be a multi-gate depth two* EQAC_a *circuit. If the number of input*

qubits acted on by two $\wedge Z$ gates in the output block of C is less than two, then only the output qubit can be in a superposition before the first $\wedge Z$.

Proof. The the final $\wedge Z$ will always be applied to the output qubit o, otherwise we can discard the gate from the circuit because of Lemma 1.1. Assume that there exists a qubit x_i in the output block such that it is only acted on by the first $\wedge Z$ and x_i is in a superposition before this gate is applied. If o is in a computational basis state before the first $\wedge Z$, we can choose a value for it that kills the gate and the final output cannot then depend on x_i. The value of x_i could be flipped, changing the parity of the inputs, whereas the output of the circuit would stay the same. On the other hand, if o is in a superposition, x_i will be entangled to it after the first $\wedge Z$ gate. The entanglement between the two qubits cannot be destroyed with the final $\wedge Z$ because we cannot destroy entanglement without acting on both of the qubits. Now we need to consider the possibility of transferring the entanglement between x_i and o from o to another qubit. All the qubits in the output block will be auxiliary qubits which according to Lemma 11.2 are also in a superposition before the first $\wedge Z$ and thus already entangled with x_i, thus we cannot transfer the entanglement with x_i from o to them. The non-output block qubits on the other hand must be in a computational basis state before the final $\wedge Z$ because of Lemma 12.2. Thus the final $\wedge Z$ cannot change their state and transfer of entanglement from o is impossible. For the final measurement the output qubit will therefore be entangled and have a probabilistic outcome which is impossible for QAC circuits. □

The following three lemmas can be combined to prove Theorem 12.2. First, we will prove that in a multi-gate depth two EQAC_a circuit, there can be only one non-output block with input qubits. Then we prove that the non-output block can have at most two input qubits and the output block no more than three, thus limiting the maximum number of possible inputs to five.

Lemma 12.4. *A multi-gate depth two* EQAC *circuit C cannot compute parity of all its input qubits if it contains more than one non-output block with input qubits.*

Proof. Assume that there are at least two non-output blocks with input qubits. We can choose inputs to one of the non-output blocks such that one of the qubits acted on by the last $\wedge Z$ gate is $|0\rangle$. This is possible because of Lemma 12.2. If we now flip a input to the other non-output block, the output of the circuit will stay the same, since the output qubit can't depend on any value in this block, but the parity of the input changes. Thus the circuit C cannot compute parity of all its inputs. □

If $B_n(C) \leq 2$, it would not be possible to kill the output qubit's dependency on the inputs by simply killing the last $\wedge Z$. If $B_n(C) = 2$, no matter how we fix the one non-output block, since the output block qubits all have a common $\wedge Z$ gate acting on them, the final outcome will depend on all of the qubits in that block. On the other hand, if $B_n(C) = 1$, we cannot kill even the last $\wedge Z$ (for now) since Lemma 12.2 states that only the non-output block qubits need to be measured in the computational basis. Thus a different approach is needed for the case $B_n \leq 2$. We prove separately that the non-output and output blocks cannot contain more than 2 and 3 input qubits correspondingly.

Lemma 12.5. *A multi-gate depth two* EQAC_a *circuit C computing parity cannot contain a non-output qubit block with more than two input qubits.*

Proof. Assume that there exists a non-output block with at least three input qubits. There must be at least one qubit on which both the first and second level $\wedge Z$ gate is applied to, otherwise the output qubit could not depend on all of the inputs in this block. Out of the qubits that are acted on by both of the $\wedge Z$ gates, there needs to be one, which we label as x_1, that is in a superposition before the first $\wedge Z$ gate. Otherwise, the first $\wedge Z$ gate would not affect any qubits which are used for the second $\wedge Z$ and through that the output qubit, and thus the the first $\wedge Z$ could be removed. Removing the first $\wedge Z$ would mean that the qubits we assumed were in the same block actually are not. Thus this is impossible. The qubit x_1 has to be the only qubit in a superposition before the first $\wedge Z$, otherwise it would get entangled which cannot happen because of Lemma 12.2. Assuming that x_1 is an input qubit, there are at least two more input qubits x_2 and x_3 in the non-output block. They need to be in a computational basis state before the first $\wedge Z$ because of the above argument and also before the second $\wedge Z$ because of Lemma 12.2. We will fix x_2 to kill the first $\wedge Z$ which leaves two possibilities for x_3. First, if the second $\wedge Z$ does not act on x_3, then the output cannot depend on this qubit and if we flip it, the parity of the inputs changes but the output of the circuit does not. Second, if the last $\wedge Z$ acts on x_3, we will fix it to kill the last $\wedge Z$ gate. Thus no other qubit in the same non-output block can influence the final outcome of the circuit, but there is at least one more input qubit in that block which we can flip to change the parity of the inputs without affecting the outcome of the circuit. We have fixed only qubits x_2 and x_3. Therefore if any of the non-output blocks in a depth two QAC circuit contains at least three input qubits, C cannot compute parity of all its input qubits. □

Finally, we prove the maximum number of input qubits in the output block by using

the technique presented in [8]. In general, their method will not work if the number of auxiliary qubits is not bounded. However, it can be adapted to the very limited structure of the multi-gate depth two circuit we have derived, as is shown in the next lemma.

Lemma 12.6 (The technique in this proof is adapted from [8])**.** *A multi-gate depth two* EQAC *circuit C computing parity cleanly cannot contain more than three input qubits in the output block.*

Proof. This proof focuses on the output block and fixes three qubits in it to a specific value. These qubits are input qubits q_1, q_2, and the output qubit o. The requirement of clean computation is imposed on C so that the output qubit could not be an auxiliary qubit and thus it is possible for us to select its initial value. Let q_2 be one of the non-output qubits acted on by both $\wedge Z$ gates. If no such qubit exists, then according to Lemma 12.3 all of the non-output qubits will be in a computational basis before the first $\wedge Z$. Thus we can fix one of the input qubits to a value that kills the first $\wedge Z$. Since $B_o(C) > 2$, there must exist another qubit such that the second $\wedge Z$ is not applied to it. After the first $\wedge Z$ is killed, this qubit cannot affect the outcome of the final measurement of the output qubit o. We can change the parity of inputs by flipping this qubit, but the output of the circuit will not change. Now that we have established that there exists a qubit q_2 such that both of the $\wedge Z$ gates are applied to it, we will fix the qubits q_1, q_2, and o as follows:

$$q_1 = U_{0,q_1}^\dagger |0\rangle, \tag{12.20}$$

$$q_2 = U_{0,q_2}^\dagger U_{1,q_2}^\dagger |0\rangle, \tag{12.21}$$

$$o = U_{0,o}^\dagger U_{1,o}^\dagger U_{1,o}^\dagger |0\rangle. \tag{12.22}$$

Qubit q_1 will kill the first $\wedge Z$, since $U_{0,q_1} U_{0,q_1}^\dagger |0\rangle = |0\rangle$. The state of qubit q_2 before the second $\wedge Z$ will therefore be $U_{1,q_2} U_{0,q_2} U_{0,q_2}^\dagger U_{1,q_2}^\dagger |0\rangle = |0\rangle$, which kills the second $\wedge Z$. Since both of the $\wedge Z$ gates are killed, the output qubit will always be $U_{2,o} U_{1,o} U_{0,o} U_{0,o}^\dagger U_{1,o}^\dagger U_{1,o}^\dagger |0\rangle = |0\rangle$. Since C computes parity, it must hold, that parity for the same assignment of values to q_1, q_2, and o must also be 0. On the other hand, the output of a parity computation can be 0 if and only if its input is a computational basis state, hence the values assigned to q_1, q_2, and o are computational basis states and thus valid inputs. Since $B_o(C) > 3$ and we fixed only three inputs, there must exist at least one more input qubit in the output block, but the output qubit cannot depend on this qubit. Thus C cannot compute the parity of all of its input qubits. □

Proof of Theorem 12.2. Lemma 12.4 proves that there cannot be more than one non-output block in a multi-gate depth two QAC_a circuit computing parity exactly. This non-output block can contain at most two input qubits, as is proven in Lemma 12.5. When we add this to the maximum number of input qubits in the output block, which is three as proven in Lemma 12.6, we get the maximum possible number of inputs which is five. Note that the condition of clean computation was imposed in Lemma 12.6. Finally, by using Corollary 11.1 we can conclude that a multi-gate depth two QAC circuit can not compute parity exactly cleanly.

Chapter 13

Discussion and Results

The final part of this thesis resulted in two complexity theoretical results. First, it was shown that multi-gate depth one QAC circuits cannot compute parity probabilistically. Second, we proved that multi-gate depth two QAC circuits cannot compute parity exactly cleanly. These results are a step towards proving that parity cannot be computed with EQAC^0 circuits. This was previously proven to be true only if the number of auxiliary qubits in the circuit is limited to $O(n)$ [8], whereas we impose not restrictions on the number of auxiliary qubits. The motivation for this problem lies in creating a distinction between the quantum complexity classes QAC^0 and QAC^0[2]. The latter class having access to unbounded parity gates and hence naturally being able to compute parity of all input qubits.

The lemmas leading to the proof about multi-gate depth two circuits (Theorem 12.2) use two techniques: First, reducing the circuit to multi-gate depth one circuits by killing the final $\wedge Z$. Second, using the exactness of the computation and thus the impossibility of the qubits to be in a non-computational basis state before the last $\wedge Z$ gate. The proof of [8], which we also use, does not fit into these two categories, their approach is to kill of all the gates acting on the output qubit. In one of our proofs we need to apply the technique from [8] to the specific circuit structure where it is possible that the output qubit is entangled in the first layer and disentangled in the last layer. Interestingly, this is also the main problem in extending our techniques to depth three and beyond. Namely, in depth three circuits we cannot easily deduce whether the qubits are entangled or not in the last layer. The larger depth allows for the possibility of the entanglement being destroyed, whereby one of the techniques we used relied on the fact that we could identify cases in which the qubits are entangled at the end of the computation. One approach to extending the result is to try to find

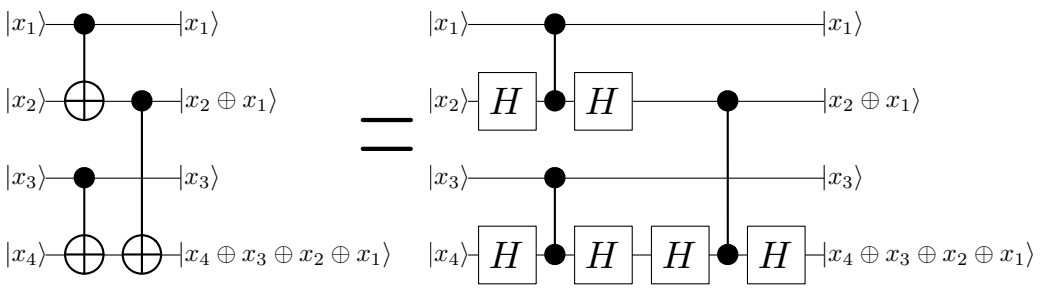


Figure 13.1: A multi-gate depth two circuit computing parity of four qubits.

an alternative solution to the conditions that required the use of [8] techniques. Since their method was required in a situation that arises with depth three circuits, perhaps this would allow to solve the problem initialy to depth three and then for the general case of arbitrary constant depth. Another benefit in replacing the [8] technique is the possibility of removing the requirement of the circuits to perform clean computations, which would make the result presented in this thesis more general. The method in [8] can only work for non-clean computations if there are no auxiliary qubits.

There are some open questions left even for multi-gate depth two EQAC circuits. We proved that multi-gate depth two circuits cannot compute parity exactly cleanly for more than 5 qubits. Conjecture 12.1 on the other hand implies that the parity of maximally four qubits could be computed. We also conjectured that the cleanness restriction is unnecessary. It thus remains an open question whether we can improve the proofs presented here to remove the cleanness condition and reduce the number of qubits for which parity can be computed. Note that while a circuit computing parity of four qubits is trivial (Figure 13.1), we could not construct a multi-gate depth two QAC circuit for computing parity of five qubits.

It remains an open question whether multi-gate depth two QAC circuits can compute parity with bounded error. Before this thesis, it was only known that it cannot be computed with QNC^0 circuits. This lower bound existed since the output of QNC^0 circuits can only depend on a constant number of the input qubits [6]. Theorem 12.1 improves this bound by proving that parity cannot be computed probabilistically by QAC^0 circuits of multi-gate depth one, whose output can depend on every input qubit. It is unlikely that the techniques in this work allow to improve this bound. The proofs of Theorem 12.2 is not extendable to bounded error circuits since it relies on the output qubit being disentangled from the rest of the qubits at the end of the computation. The proof of Theorem 12.1, is also unlikely to be extendable to arbitrary constant depth, since it depends on some of the non-output qubits being disentangled before the $\wedge Z$ gate. This assumption only holds for multi-gate depth one circuits.

Bibliography

[1] Christopher M Maynard and Einar Pius. A quantum multiply-accumulator. *Quantum Information Processing*, 13(5):1127–1138, May 2014.

[2] Peter W Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484, 1997.

[3] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th annual ACM symposium on Theory of computing (STOC'96)*, pages 212–219, New York, New York, USA, 1996. ACM Press.

[4] Cristopher Moore and Martin Nilsson. Parallel Quantum Computation and Quantum Codes. *SIAM Journal on Computing*, 31(3):799–815, January 2001.

[5] Frederic Green, Steven Homer, Cristopher Moore, and Christopher Pollett. Counting, Fanout, And The Complexity Of Quantum ACC. *Quantum Information and Computation*, 2(1):35–65, January 2002.

[6] Stephen Fenner, Frederic Green, Steven Homer, and Yong Zhang. Bounds on the Power of Constant-Depth Quantum Circuits. In *Proceedings of the 15th International Symposium on on Fundamentals of Computation Theory (FCT 2005)*, pages 44–55, August 2005.

[7] Peter Høyer and Robert Spalek. Quantum Circuits with Unbounded Fan-out. *Theory of Computing*, 1(5):81–103, August 2002.

[8] Maosen Fang, Stephen Fenner, Frederic Green, Steven Homer, and Yong Zhang. Quantum Lower Bounds for Fanout. *Quantum Information and Computation*, 6(1):46–57, January 2006.

[9] Mehdi Mhalla and Simon Perdrix. Finding Optimal Flows Efficiently. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I (ICALP'08)*, pages 857–868, Berlin, Heidelberg, 2008.

[10] Anne Broadbent and Elham Kashefi. Parallelizing quantum circuits. *Theoretical Computer Science*, 410(26):2489–2510, 2009.

[11] Dan E Browne, Elham Kashefi, and Simon Perdrix. Computational depth complexity of measurement-based quantum computation. In *Proceeding of the 5th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC'10)*, pages 35–46, December 2010.

[12] Richard Cleve and John Watrous. Fast Parallel Circuits for the quantum Fourier transform. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000)*, pages 526–536, Rendo Beach, CA, USA, 2000. IEEE Comput. Soc.

[13] Robert Raussendorf, Hans Briegel, and Daniel Browne. The one-way quantum computer - a non-network model of quantum computation. *Journal of Modern Optics*, 49(8):1299–1306, 2002.

[14] Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus. *Journal of the ACM*, 54(2), April 2007.

[15] Vincent Danos and Elham Kashefi. Determinism in the one-way model. *Physical Review A*, 74(5):052310, 2006.

[16] Daniel Browne, Elham Kashefi, Mehdi Mhalla, and Simon Perdrix. Generalized flow and determinism in measurement-based quantum computation. *New Journal of Physics*, 9:250, August 2007.

[17] Jisho Miyazaki, Michal Hajdušek, and Mio Murao. Translating measurement-based quantum computation with gflow into quantum circuit. *arXiv:1310.4043v2*, November 2013.

[18] Raphael Dias da Silva, Einar Pius, and Elham Kashefi. Global Quantum Circuit Optimization. *arXiv:1301.0351*, January 2013.

[19] A A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, April 1987.

[20] R Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. *Proceedings of the nineteenth annual ACM symposium on Theory of computing (STOC '87)*, pages 77–82, January 1987.

[21] David A Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC1. *Journal of Computer and System Sciences*, 41(3):274–306, December 1990.

[22] Yasuhiro Takahashi and Seiichiro Tani. Constant-Depth Exact Quantum Circuits for the OR and Threshold Functions. *arXiv:1112.6063v2*, December 2011.

[23] Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, 1984.

[24] D Bonneau, E Engin, K Ohira, N Suzuki, H Yoshida, N Iizuka, M Ezaki, C M Natarajan, M G Tanner, R H Hadfield, S N Dorenbos, V Zwiller, J L O'Brien, and M G Thompson. Quantum interference and manipulation of entanglement in silicon wire waveguide quantum circuits. *Journal of Physics A: Mathematical and Theoretical*, 14(4):045003, April 2012.

[25] R Baxter, S Booth, M Bull, G Cawood, J Perry, M Parsons, A Simpson, A Trew, A McCormick, G Smart, R Smart, A Cantle, R Chamberlain, and G Genest. Maxwell - a 64 FPGA Supercomputer. In *2nd NASA/ESA Conference on Adaptive Hardware and Systems*, pages 287–294, August 2007.

[26] O Almer, R V Bennett, I Böhm, A C Murray, X Qu, M Zuluaga, B Franke, and N P Topham. An End-to-End Design Flow for Automated Instruction Set Extension and Complex Instruction Selection based on GCC. In *Proceedings of 1st International Workshop on GCC Research Opportunities*, 2009.

[27] Dake Liu. *Embedded DSP Processor Design: Application Specific Instruction Set Processors*. Systems on Silicon. Morgan Kaufmann, Linkoping University, May 2008.

[28] Thomas G Draper. Addition on a Quantum Computer. *arXiv:quant-ph/0008033v1*, August 2000.

[29] Cristopher Moore and Stephan Mertens. *The Nature of Computation*. Oxford University Press, October 2011.

[30] Amos R Omondi. *Computer arithmetic systems*. Algorithms, Architecture, and Implementation. Prentice Hall International (UK) Limited, 1994.

[31] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, April 2009.

[32] Lars Wanhammar. *DSP Integrated Circuits*. Academic Press, February 1999.

[33] Milos D Ercegovac and Tomas Lang. *Digital Arithmetic*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann Publishers, September 2003.

[34] Texas Instruments. Texas Instruments C64x DSP [online]. Avaliable at: http://www.ti.com/product/TMS320C6455 [Accessed: 17 August 2014].

[35] Blackfin ADSP-BF50x Processors [online]. Avaliable at: http://www.analog.com/static/imported-files/white_papers/Blackfin_BF50x_Next_Innovation.pdf [Accessed: 17 August 2014].

[36] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, October 2000.

[37] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, USA, 1 edition, January 2007.

[38] Vincent Danos, Elham Kashefi, and Prakash Panangaden. Parsimonious and robust realizations of unitary maps in the one-way model. *Physical Review A*, 72(6):064301, December 2005.

[39] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147–153, July 1996.

[40] Thomas G Draper, Samuel A Kutin, Eric M Rains, and Krysta M Svore. A Logarithmic-Depth Quantum Carry-Lookahead Adder. *Quantum Information and Computation*, 6(4&5):351–369, July 2006.

[41] Phil Gossett. Quantum Carry-Save Arithmetic. *arXiv:quant-ph/9808061v2*, August 1998.

[42] Adriano Barenco, Charles Bennett, Richard Cleve, David Divincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, November 1995.

[43] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv:quant-ph/0410184v1*, October 2004.

[44] Peter M Kogge, Harold S Stone, and Harold S. A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Transactions on Computers*, C-22(8):786–793, August 1973.

[45] Kai-Wen Cheng and Chien-Cheng Tseng. Quantum Plain and Carry Look-Ahead Adders. *arXiv:quant-ph/0206028v1*, June 2002.

[46] John G Earle. Latched carry save adder circuit for multipliers. Google Patents, September 1967.

[47] Kai-Wen Cheng and Chien-Cheng Tseng. Quantum full adder and subtractor. *Electronics Letters*, 38(22):1343–1344, October 2002.

[48] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. Quantum Addition Circuits and Unbounded Fan-Out. *Quantum Information and Computation*, 10(9&10):872–890, 2010.

[49] Agung Trisetyarso and Rodney Van Meter. Circuit Design for A Measurement-Based Quantum Carry-Lookahead Adder. *International Journal of Quantum Information*, 8(5):843–867, August 2010.

[50] Lafifa Jamal, Md Shamsujjoha, and Hafiz Md Hasan Babu. Design of Optimal Reversible Carry Look-Ahead Adder with Optimal Garbage and Quantum Cost. *International Journal of Engineering and Technology*, 2(1):44–50, January 2012.

[51] C S Wallace. A Suggestion for a Fast Multiplier. *IEEE Transactions on Electronic Computers*, EC-13(1):14–17, 1964.

[52] L Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, 1965.

[53] Whitney J Townsend, Jr Swartzlander Earl E, and Jacob A Abraham. A comparison of Dadda and Wallace multiplier delays. In *Proceeding so the Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, pages 552–560, December 2003.

[54] David Beckman, Amalavoyal Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Physical Review A*, 54(2):1034–1063, August 1996.

[55] Himanshu Thapliyal and M.B Srinivas. Novel Reversible Multiplier Architecture Using Reversible TSG Gate. *IEEE International Conference on Computer Systems and Applications*, pages 100–103, May 2006.

[56] J J Álvarez-Sánchez, J V Álvarez-Bravo, and L M Nieto. A quantum architecture for multiplying signed integers. *Journal of Physics: Conference Series*, 128(1):1–9, October 2008.

[57] Giuseppe Florio and Domenico Picca. Quantum implementation of elementary arithmetic operations. *arXiv.org*, March 2004.

[58] H Thapliyal and M.B Srinivas. Novel Reversible 'TSG' Gate and Its Application for Designing Components of Primitive Reversible/Quantum ALU. In *Information, Communications and Signal Processing, 2005 Fifth International Conference on*, pages 1425–1429, 2005.

[59] Moayad A Fahdil, Ali Foud Al-Azawi, and Sammer Said. Operations Algorithms on Quantum Computer. *International Journal of Computer Science and Network Security*, 10(1):85–95, January 2010.

[60] Michael Kirkedal Thomsen, Robert Glück, and Holger Bock Axelsen. Reversible arithmetic logic unit for quantum arithmetic. *Journal of Physics A: Mathematical and Theoretical*, 43(38):382002, August 2010.

[61] Y Syamala and A.V.N Tilak. Reversible Arithmetic Logic Unit. In *3rd International Conference on Electronics Computer Technology*, pages 207–211, 2011.

[62] Daniel M Greenberger, Michael A Horne, and Anton Zeilinger. Going Beyond Bell's Theorem. December 2007.

[63] Thomas Monz, Philipp Schindler, Julio T Barreiro, Michael Chwalla, Daniel Nigg, William A Coish, Maximilian Harlander, Wolfgang Hänsel, Markus Hennrich, and Rainer Blatt. 14-Qubit Entanglement: Creation and Coherence. *Physical Review Letters*, 106(13):130506, March 2011.

[64] Xing-Can Yao, Tian-Xiong Wang, Ping Xu, He Lu, Ge-Sheng Pan, Xiao-Hui Bao, Cheng-Zhi Peng, Chao-Yang Lu, Yu-Ao Chen, and Jian-Wei Pan. Observation of eight-photon entanglement. *Nature Photonics*, 6(4):225–228, February 2012.

[65] Robert Griffiths and Chi-Sheng Niu. Semiclassical Fourier Transform for Quantum Computation. *Physical Review Letters*, 76(17):3228–3231, April 1996.

[66] Christopher M Dawson and Michael A Nielsen. The Solovay-Kitaev algorithm. *Quantum Information and Computation*, 6(1):81–95, January 2006.

[67] Efficient Clifford+T approximation of single-qubit operators. *arXiv:1212.6253*, December 2012.

[68] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Asymptotically Optimal Approximation of Single Qubit Unitaries by Clifford and T Circuits Using a Constant Number of Ancillary Qubits. *Physical Review Letters*, 110(19):190502, May 2013.

[69] Optimal ancilla-free Clifford+T approximation of z-rotations. *arXiv:1403.2975*, March 2014.

[70] Daniel Gottesman and Isaac Chuang. Quantum Teleportation is a Universal Computational Primitive. *Nature*, 402:390–393, 1999.

[71] Susana F Huelga, Joan A Vaccaro, Anthony Chefles, and Martin B Plenio. Quantum Remote Control: Teleportation of Unitary Operations. *Physical Review A*, 63(042303):5, 2001.

[72] Susana F Huelga, Martin B Plenio, and Joan A Vaccaro. Remote control of restricted sets of operations: Teleportation of Angles. *Physical Review A*, 65(042316), 2002.

[73] Debbie Leung. Two-qubit Projective Measurements are Universal for Quantum Computation. *arXiv:quant-ph/0111122v2*, 2002.

[74] Michael Nielsen. Universal quantum computation using only projective measurement, quantum memory, and preparation of the 0 state. *Physics Letters A*, 2-3(308):96–100, February 2003.

[75] Richard Jozsa. An introduction to measurement based quantum computation. *arXiv:quant-ph/0508124v2*, page 22, January 2006.

[76] Robert Raussendorf, Daniel Browne, and Hans Briegel. Measurement-based quantum computation on cluster states. *Physical Review A*, 68(022312), August 2003.

[77] Marc Hein, Wolfgang Dür, Jens Eisert, Robert Raussendorf, Marteen Van den Nest, and Hans J Briegel. Entanglement in Graph States and its Applications. In *Proceedings of the International School of Physics "Enrico Fermi"*, pages 115–218, February 2006.

[78] Maarten Van Den Nest, Akimasa Miyake, Wolfgang Dür, and Hans Briegel. Universal Resources for Measurement-Based Quantum Computation. *Physical Review Letters*, 97(15):150504, October 2006.

[79] David Gross, Jens Eisert, and Steven T Flammia. Most quantum states are too entangled to be useful as computational resources. *Physical Review Letters*, 102(190501):5, 2009.

[80] Caterina E Mora, Marco Piani, Akimasa Miyake, Marteen Van den Nest, Wolfgang Dür, and H J Briegel. Universal resources for approximate and stochastic measurement-based quantum computation. *Physical Review A*, 81(4):042315, April 2010.

[81] Tzu-Chieh Wei, Ian Affleck, and Robert Raussendorf. Affleck-Kennedy-Lieb-Tasaki State on a Honeycomb Lattice is a Universal Quantum Computational Resource. *Physical Review Letters*, 106(7):070501, February 2011.

[82] Vincent Danos and Elham Kashefi. Pauli Measurements are Universal. *Electronic Notes in Theoretical Computer Science*, 170:95–100, 2007.

[83] Mehdi Mhalla, Mio Murao, Simon Perdrix, Masato Someya, and Peter S Turner. Which graph states are useful for quantum information processing? In *Proceeding of the 6th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2011)*, 2010.

[84] Damian Markham and Elham Kashefi. Entanglement, Flow and Classical Simulatability in Measurement Based Quantum Computation. In Jan Rutten, editor, *Horizons of the Mind. A Tribute to Prakash Panangaden*, pages 427–453. Springer International Publishing, 2014.

[85] Niel de Beaudrap. Finding flows in the one-way measurement model. *Physical Review A*, 77(2):022328, January 2008.

[86] Ross Duncan and Simon Perdrix. Rewriting Measurement-Based Quantum Computations with Generalised Flow. *Automata, Languages and Programming*, pages 285–296, 2010.

[87] Raphael Dias da Silva and Ernesto F Galvão. Compact quantum circuits from one-way quantum computation. *Physical Review A*, 88(1):012319, July 2013.

[88] Miklós Ajtai. $\sum_1^1$-Formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.

[89] Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles. In *26th Annual Symposium on Foundations of Computer Science*, pages 1–10. IEEE, October 1985.

[90] Johan Hastad. Almost optimal lower bounds for small depth circuits. *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, November 1986.

[91] William Hesse. Division Is in Uniform TC0. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, pages 104–114. Springer-Verlag, July 2001.

Publications

Christopher M Maynard and Einar Pius. A quantum multiply-accumulator. In *Quantum Information Processing*, 13(5):1127-1138, May 2014.

Einar Pius, Elham Kashefi, and Raphael Dias da Silva. Global quantum circuit optimisation. Accepted to be published in *Quantum Information and Computation*, 2015.