Author:
Scott, John R

Title:
Classical Control Systems for Photonic Quantum Computing

CLASSICAL CONTROL SYSTEMS FOR PHOTONIC QUANTUM COMPUTING

John Scott

A dissertation submitted to the University of Bristol in accordance with the requirements for award of the degree of Doctor of Philosophy (PhD) in the Faculty of Engineering, School of Computer Science, Electrical and Electronic Engineering, and Engineering Maths

October, 2022

Word count: 43,000

Abstract

Photonic quantum computing is based on the use of single photons as qubits. There are many proposed architectures for the realisation of photonic quantum computers; however, the design of electronic control systems for these architectures is substantially less well understood. Furthermore, most of these proposals for photonic quantum computing use measurement-based quantum computing (MBQC), or similar schemes that avoid the necessity for long-lived photons in the system. These schemes can appear quite abstract and theoretical, and cause a barrier to entry for electronic engineers hoping to design control systems for photonic quantum computing.

This thesis has two primary objectives. First, to present a methodology for evaluating prospective control system designs for photonic quantum computers based on MBQC, based on the analysis of timing constraints that these implementations impose on the full quantum computing system. These timing constraints are derived by analysing a concrete design, targeting a simple model for photonic quantum computing, in a case where the control system is simple enough to design without needing hardware emulation to evaluate design trade-offs. Constraints in a more complicated (and more realistic) setting, involving photonic quantum computing using incomplete cluster states, are addressed by emulating one possible choice of algorithms that could be used as the basis for a control system in this case. This latter emulation provides a framework for analysing other prospective algorithms, and forms the basis for an analysis system that could be modified to investigate other photonic quantum computing models.

The second objective is to present MBQC-based photonic quantum computing in a simple diagrammatic form, which focuses on the control system specification rather than the mathematics of MBQC. It is hoped that this lowers the barrier to entry for engineers interested in photonic quantum computing control system design.

Acknowledgements

This thesis would not have been possible without the support from a great many of my colleagues, friends, and family.

I would like to thank my supervisor, Krishna Coimbatore Balram, for providing constant support throughout my PhD, and for many interesting and varied discussions during our regular weekly meetings. His ideas and suggestions have materially improved this thesis, and I will miss our discussions surrounding the future development of control systems in photonic quantum computing.

I would also like to thank the staff and students in the Quantum Engineering Centre for Doctoral Training, who have provided a supportive and friendly environment in which to conduct a PhD. In particular, I would like to thank Alex Qiu, Naomi Solomons and Oliver Thomas, for our entertaining weekly discussions involving our PhD projects and other miscellany! I would like to especially thank Oliver Thomas, for all the other discussions we have had over the last four years, and all his helpful suggestions relating directly to my PhD.

I would like to extend my warm thanks to Lana Mineh, who has not only been a constant source of support, but also helped me solve programming problems, aided me in checking calculations, and discussed ideas with me relating to many aspects of this thesis. Her contribution cannot be overstated.

Finally, my sincere gratitude goes to my family, for providing constant help and support to me over the last four years, and for their help checking and offering suggestions relating to this thesis.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE:.....

Contents

List of Figures

List of Tables

List of Abbreviations

ADC Analog-to-digital converter.

ASIC Application-specific integrated circuit.

BFS Breadth-first search.

CSV Comma-separated variable.

DAC Digital-to-analog converter.

DRAM Dynamic random-access memory (for larger main memory).

FIFO First-in first-out (data structure).

FPGA Field-programmable gate array.

GBFS Global breadth-first search.

I/O Input/output.

IBFS Incremental breadth-first search.

IMBQC Incomplete-cluster-state measurement-based quantum computing.

KLM Knill-Laflamme-Milburn (scheme for photonic quantum computing).

LUT Look-up table.

MBQC Measurement-based quantum computing.

MMCM Mixed-mode clock manager.

QSL Quantum simulation library.

RAM Random-access memory.

SNSPD Superconducting-nanowire single-photon detector.

SOI Silicon-on-insulator (platform for photonic integrated circuits).

SRAM Static random-access memory (for fast on-chip memory).

VHDL Very high-speed integrated circuit hardware description language.

Chapter 1

Introduction

With the invention and subsequent development of computers, we have become accustomed to a rate of technological improvement unequalled by any other human invention. In the last 75 years, electronic computers have risen from relatively simple arithmetic computing machines to become the basis for an infrastructure layer that touches nearly every aspect of our lives. The incredibly rapid rate of improvement is due to the ability to miniaturise computers, and scale up their computing power, at a rate that has been exponential with time – a fact predicted by Moore in 1965 [1]: from the year 1965 to nearly the present day, the number of transistors on a chip has doubled every 18 months. The layered complexity of computing services that now exist makes it nearly impossible to comprehend how the whole system works, from the services provided by cloud computing all the way down to the functioning of the individual transistors.

Quantum computing is a fundamentally different approach to problem solving using physical devices. Instead of using a classical logical building block such as the transistor, and then encoding every problem in digital terms, quantum computers seek to use the behaviour of quantum systems themselves to provide the solution to problems. An important motivation for this is that quantum systems are difficult for classical computers to simulate – a fact that Feynman noted in one of the earliest discussions of quantum computing [2] – meaning that quantum computers may be capable of solving problems that classical computers cannot. On the other hand, to make use of a quantum computer, it is necessary to turn a problem into something that is equivalent to the evolution of a quantum system. The difficulty of this task is the reason that

only relatively few quantum algorithms exist today, and it is an active area of research to find practical problems that quantum computers can solve [3].

Comparisons between quantum and classical computers are inevitable. So too are efforts to cast quantum computers in a framework similar to classical computing. For example, the qubit, or quantum bit, is named by analogy with the classical bit; and the quantum operations that a quantum computer performs are called quantum gates, by analogy with classical logic gates. There is also a desire to "front-run" the ideas that made classical computers successful, such as quantum software frameworks and cloud-based access to quantum computers [4]. However, there is an apparent contradiction between the state of quantum hardware, which is analogous to the state of early classical computing hardware, and the plethora of higher-level quantum tools, which, to an outside observer, give quantum computing the impression of being "done". A glance at the various competing quantum computing technologies, all with their own unique problems, shows that this is very far from the case. It has only recently been shown that current quantum computers can do something that a classical computer cannot [5], and the validity of this demonstration is disputed [6, 7]. Scalability, in a sense analogous to Moore's law, is not currently achieved by any quantum computing technology [8].

It is universally agreed that the ideal end-goal of quantum computing would be the construction of a large-scale, integrated, quantum computer, analogous to the high performance classical processors that exist today.

However, the incredible performance achieved by microprocessors is due to a very complicated mix of design choices – often trade-offs between non-ideal systems – which have been optimised by a long process of trial-and-error lasting decades. For example, the use of the memory hierarchy[1] [9] to hide the extremely long latency of main memory (computer RAM) is not an intrinsic feature of computing systems; it arises from the particular memory technology (DRAM) used as the basis for main memory. The (unlikely) discovery of a new type of memory that has the capacity of DRAM but the speed capabilities of SRAM (which is used for caches and other on-chip memory) would entail an entire redesign of modern microprocessors. Current microprocessor design

[1]The memory hierarchy is the use of several different memory stages that optimise for high speed and low latency (e.g. cache levels) near the processor, and optimise for large capacity and error-correction far from the processor (e.g. main memory, and disk-based storage). The objective is to mimic a single quite-large and quite-fast memory technology, from the point of view of software.

involves a very complicated balancing of different competing factors to achieve a design with improved performance compared to its predecessors [10]. None of these factors have anything to do with the high-level design of computers; for example, the use of the stored-program architecture, or the use of classical logic gates or transistors in its construction[2].

It is possible that the success of quantum computing will similarly rest on a combination of specific device characteristics, and trade-offs in device design and construction. At a very high level, a prospective quantum computing system is considered to be made up of a physical system that represents the qubits, and an associated control system, made from digital and analog electronics and classical computing systems. In each of the computing quantum computing technologies, there are many different proposals[3] for how to realise large-scale quantum computers; those with thousands or millions of qubits, which are able to compete with classical computers at solving problems. In these proposals, the quantum aspects of the system are often discussed at length and worked out in detail. For example:

- How to make/manufacture the qubits?
- How to perform high-fidelity quantum operations (gates)?
- How to perform error correction, or build it in, so that the output from the qubits is reliable?

What is often missing from these proposals is similarly detailed information about how the classical control system will work. **This thesis is focused entirely on this control system design, for the case of photonic quantum computers based on measurement-based quantum computation.** The main purpose of this investigation is to establish whether the control systems constrain the design of photonic quantum computers.

1.1 Why investigate control system design?

Analog and digital electronic circuits, and classical computing systems required to control quantum computers are often seen as an implementation

[2]Nowadays, different transistor technologies are being explored as a method to increase processor performance. However, in the past, the primary technique enabling Moore's law was the gradual miniaturisation of transistors, not any fundamental changes in their design.

[3]Often called quantum computer architectures.

detail in quantum computing architecture design, to be worked out after the qubits have been fully understood. The purpose of this thesis is to promote the design of the control system as equally important, or more important, than the design of the qubits. The reason for this is threefold:

1. Classical control system design is starting to be seen as a real engineering problem in relation to scaling quantum computers. For example, in superconducting qubit systems (the current leading qubit technology by number of qubits), the control system is already a serious bottleneck.

Google's 54-qubit superconducting qubit processor uses 277 digital-to-analog converters (DACs) and two stages of cryogenic analog signal processing (one at 3 K and one at 10 K) to interface to their qubits [5]. Outside the cryostat, the pulse-generating hardware is mounted in four 6U-chassis (about 27 cm tall), each of which controls approximately 15 qubits.

The use of external rack-mounted electronics is infeasible for large-scale quantum computer, containing thousands of qubits. Although custom-made electronics can alleviate some of the problems relating to size and power consumption, there still remains a physical limit to the number of wires that can be routed into a cryostat [11], from both a mechanical and thermal point of view. As a result, there is a large amount of current research into the development of cryogenic electronics that can be integrated much closer to the superconducting qubits [12].

2. Analog and digital electronics, particularly classical computing systems, are very highly developed disciplines, where there is limited room for improvement on the state of the art.

This means that if the electronic control system imposes a limit on a quantum computing architecture, it may be very difficult to remove the imposed limitation by trying to improve the electronics. In the context of photonic quantum computing, the most important limitation relates to the timing characteristics of the control system.

On the other hand, quantum technologies are relatively new, and may be expected to see a great deal of development. It may therefore be much more important to prioritise the limitations arising from the control system over considerations relating purely to the quantum architecture of the system.

3. Like classical computers, the success of large-scale quantum computers may come down to highly specific trade-offs between different parts of the qubit technologies and the associated control system. Analysis of these trade-offs requires the implementation details of the control system and quantum system to be known in detail; these systems must be designed in tandem, on an equal footing.

This thesis is about control system design for photonic quantum computers, where the physical system used as the qubit is a single photon. This subject has a particularly elaborate set of proposals for realising large-scale quantum computers [13–16], compared to other quantum computing platforms, mainly arising from the limited interaction between different photons. Despite the breadth of research into photonic quantum computing architectures, there is almost no research on the subject of control systems for photonic quantum computers. This may lead newcomers to the subject to believe that the control systems are not an important problem for photonic quantum computers.

This is partially true, because even simple building blocks required for the use of photonic qubits do not yet exist. For example, on-demand single-photon sources, with the high purity required for any implementation of (discrete-variable) photonic quantum computing, have been a subject of research for the last 20 years, but an ideal source does not yet exist [17]. Very good quality single-photon detectors are well established [18], but it is currently unknown how to integrate them into a prospective large-scale photonic quantum computer, and manage the cryogenic constraints they impose on the rest of the system [19].

However, many simple details of these control systems can be worked out now, based on available prospective architectures for photonic quantum computers, without needing all the prerequisite components to exist. Based on this analysis, it is clear that there may be substantial implementation problems relating to the design of electronic systems for photonic quantum computing. These problems essentially arise from the need for the speed of the electronics to compete with the speed of light. The characteristic timescale for photons moving through a photonic quantum computer is hundreds of picoseconds, which imposes a serious timing constraint on how fast the electronic control system must be able to operate. Based on point two listed above, it is very

important to understand what limitations control system electronics may impose, because these kind of limitations may render one architecture or another unviable.

Another feature of photonic quantum computing architectures is that they are relatively mathematically complicated. This may be a significant barrier to entry for electronic engineers, who would be able to provide the most assistance with the difficult control system implementation problems described above. The complexity is due to the use of measurement-based quantum computing (MBQC) as the basis of most architectures for photonic quantum computing. MBQC is used in photonic quantum computing because it allows photons to be used in a relatively short-lived manner, where each individual photon is produced and then measured (and destroyed) quickly. This is primarily necessary to mitigate the effect of propagation loss[4]. No single photon corresponds to a qubit in the quantum computer, but together, the photons interact in such a way as to simulate a set of qubits. By measuring the individual qubits according to particular rules, it is possible to realise quantum gates on these qubits. However, the rules are not as intuitive as the gate-based model of quantum computation, which makes photonic quantum computing more abstract than other quantum computing platforms (for example, superconducting qubits or trapped ions, where each qubit is easily mapped to a physical device).

Therefore, this thesis seeks to achieve two goals:

- **Provide a practical description of MBQC.** Throughout this thesis, the focus is on practical discussions of the implementation of MBQC algorithms in hardware and software, rather than mathematically rigorous analysis of measurement patterns, which are treated in great detail elsewhere [20, 21].

- **Investigate constraints imposed by electronic control systems on photonic quantum computing architectures.** We consider two simple models for photonic quantum computing based on MBQC, one involving ideal (fully-connected) cluster states, and one where the cluster state may contain missing edges. Chapters 2, 3 and 4 contain hardware design and verification for a control system for the first model. Chap-

[4]Where a photon is absorbed by its transmitting medium: an optical fibre, or a waveguide.

ters 5 and 6 contain the design of an emulation and simulation library, MBQCSIM, whose purpose is to analyse implementation constraints involved in the second model.

Even though it is not realistic, we consider the simple fully-connected model of photonic quantum computing first, because it leads to a control system which: does not involve complicated algorithms; does not involve difficult design trade-offs; and provides a clear indication of where control system constraints may appear in more complicated (and more realistic) models for photonic quantum computing. At the same time, a concrete hardware design targeting a high-performance field-programmable gate array (FPGA) provides a strong starting point for analysing design-imposed constraints, and offers an introductory system for others wishing to learn about control-system implementation for photonic quantum computing.

The simplest realistic model of photonic quantum computing, based on MBQC, uses an incomplete cluster state – one with missing edges[5]. These missing edges arise because there is no known deterministic mechanism for generating entanglement between photons, of the kind required for the cluster state which is the basis for MBQC. Instead, fusion gates (the generators of entanglement between photons) only succeed with some edge probability p (e.g. 50 % [13] or 75 % [15, 22]). Therefore, any realistic system of photonic quantum computing must be based on incomplete cluster states. For brevity, we refer to this model as IMBQC (for incomplete-cluster-state MBQC).

The use of an incomplete cluster state introduces many new algorithmic complications, such as the need to map measurement patterns onto random cluster states [14], and the need to implement this mapping in real time by searching for paths through the cluster state [23, 24]. This increased complexity makes it infeasible to design a control system for photonic quantum computing without first emulating the control system, in order to analyse design trade-offs between different algorithm choices and other implementation questions (such as the memory model for algorithm data storage). This kind of emulation is analogous to the techniques used in microprocessor architecture design, to evaluate whether one design choice or another will lead to a better overall system [10]. This emulation is not necessary for the model based on the fully-connected cluster state discussed above, because measurement pat-

[5] A cluster state can be thought of as a graph, where the nodes correspond to photons, and the edges correspond to entanglement between photons.

terns are static in that case, and the algorithms which must be implemented reduce to simple fixed arithmetic operations.

Hardware emulation is able to provide concrete answers to questions relating to how a particular implementation will behave. Timing constraints may be obtained from the emulation by making reasonable assumptions on the hardware that may be used to realise the emulated system. However, this does not replace a full system design, which is the single valid method to establish what constraints the control system will really impose on the quantum computer.

A side-effect of the verification of the designs discussed in this thesis is the development of an MBQC simulator, which is capable of simulating measurement patterns on cluster states of arbitrary width, and height at most 14[6]. In addition to verification of measurement patterns and algorithms, this tool can be used as the basis for an analysis of how noise in the analog components in the control system affects the quantum output states from the system. This is discussed in detail in Chapter 6.3.

This thesis aims to advocate an "engineering-focused" approach for photonic quantum computing research: by completely isolating the control system specification from the quantum computing architecture, and approaching it entirely as an isolated electronic engineering problem, it may be brought within the scope of highly developed design methodologies for high-performance electronic systems [25, 26]. The current landscape of theoretical photonic quantum computing architectures provides almost unlimited material for this kind of investigation. It requires, as a prerequisite, that the control system is entirely extricated from the quantum architecture, and is presented in a form amenable to electronic circuit design. We have tried to achieve this separation for the models we discuss in this thesis. We also present some simple designs that satisfy these specifications. However, the true advantage of this approach is that others may now take the same decoupled specification (or modify it as they see fit), and improve upon and optimise the designs. Here, we take a simple minimalist approach, and focus on the digital aspects of the control system. However, it is the analog parts of the system (especially if the cryogenic requirement is incorporated) that offer the real design challenge, and will impose the toughest constraints on the overall system. The problems involved in this latter case are simply too complicated to address without first

[6] On an ordinary laptop.

obtaining a rigorously specified behaviour for all parts of the control system, which is completely devoid of quantum mechanical considerations.

The remainder of this introduction is organised as follows. Section 1.2 contains a brief historical summary of the development of quantum computers, specifically highlighting relationships with the development of classical computers, and provides some comparison between the two different approaches to solving problems. A goal of this thesis is to point out similarities and differences between the implementation of quantum and classical computers, and draw on the development of classical computers to inform the development of control systems for quantum computing.

Section 1.3 provides a brief overview of the gate-based model for quantum computing, and other aspects of quantum information theory that are necessary for understanding this thesis. Section 1.4 contains an overview of the basics of (discrete-variable) photonic quantum computing, using single photons as qubits. It does not cover in detail the substantial information available in the literature on the various components required for the realisation of photonic quantum computers, including the underlying photonic platform for waveguides (e.g. silicon-on-insulator or lithium niobate); heralded [27] or multiplexed [28, 29] single-photon sources; single-photon detectors [30]; modulators [31]; and architectural components such as fusion gates [13], or the generation of cluster states [15]. This information is covered in great detail in the references provided, and much of the development of photonic devices is ongoing research which is outside the scope of this thesis. Instead, it provides a simple operational overview of the aspects of photonic quantum computing which are directly relevant for control system implementation. Finally, Section 1.5 contains a guide for the reader about how to approach this thesis.

1.2 Comparison of classical and quantum computers

Quantum computing is a relatively recent idea in the history of quantum physics. While quantum mechanics originated near the turn of the 1900s, the first ideas of quantum computing began to develop in the 1970s and 80s – about 20 years after quantum effects had been used for the invention of the transistor, and the subsequent development of classical computers. The idea of quantum computers arose as the intersection of three different strands of

thinking:

- Investigation of the boundaries of theoretical computational models such as the Turing machine.
- Research into the possibility of extremely low-energy (classical) computers, via reversible computation.
- The question whether physics can be simulated using computers.

There was a great deal of interest in the 1970s and 80s in the question whether are any fundamental energy limits inherent in the process of classical computation [32]. Such energy limits arise if a computational process is irreversible, because thermodynamics guarantees that energy will be lost in that process. Following the work of Landauer in 1961 on intrinsic energy dissipation in computers [33], it was widely thought that computation was not a reversible process. By designing a scheme of classical computation based on ideal billiard balls, Edward Fredkin showed that this was not true [34]. The key insight was the use of a reversible classical logic gate known as the Fredkin gate, or controlled-SWAP. Being somewhat outside academia, Fredkin did not publish his results, and Bennett (independently) showed in 1973 that reversible computation is possible [35], leading to the theoretical possibility of an extremely low energy computer. Feynman, with an interest in physical computation inspired by Fredkin, found that there are essentially no lower energy limits due to quantum mechanics either [36]. In doing so, he created a model of classical computation using reversible quantum mechanical elements; a type of quantum computer.

Although presented in the context of reversible computation, Feynman's true interest in the problem lay in the question whether the study of the computer simulation of physics could cast a new light on quantum theory [2]. Feynman argued that the simulation of physics could only be achieved by using computers based on the laws of quantum physics. The opposite point of view, held by Edward Fredkin due to his strongly-held belief in finite physical laws [34], was that digital electronic computers could exactly simulate physical phenomena. In his keynote address [2], Feynman laid out the important concept of the exponential scaling of the classical simulation of physics, and suggested the goal of finding a "universal" quantum computer, analogous

to the universal Turing machine. This talk is seen by many as the birth of quantum computing.

The concept of the Turing machine has been a cornerstone of theoretical computer science since it was introduced in 1936 [37]. A Turing machine is a simple abstract type of (classical) computer which consists of an infinite memory, and a processor that can read and write to the tape according to special rules[7]. The importance of the Turing machine is that there exist universal Turing machines, which are able to simulate any other Turing machine. This is achieved by using the tape to store a "program" for the computer, in addition to its role as a scratch pad for data processing. The revolutionary concept of universality – via the stored-program concept – formed the abstract basis for the design of the EDVAC, one of the first large-scale general purpose computers. The design, laid out by von Neumann in his famous draft [39], described the physical realisation of a universal Turing machine, with the addition of practical details, such as the presence of higher level arithmetic processing operations, and a mechanism for fetching instructions[8]. Although the stored-program concept is often described as the crucial characteristics of post-World War II computers, the really revolutionary aspect of these computers was their universality [40].

The importance of the universal computer is cemented by the Church-Turing thesis, which states, approximately, that any real-world problem can be solved using a universal Turing machine. With the vast improvement in technology, particularly the increase in the size of memory, computers have approached the ideal universal Turing machine, implying that they can solve any problem at all – provided they are fast and large enough. The enormous array of applications of classical computers shows that this is certainly very nearly the case. However, the Church-Turing thesis has received continuous scrutiny to attempt to discover whether any computations lie outside the scope of traditional universal digital computers.

In 1985, David Deutsch reformulated the Church-Turing thesis to specify the simulation of finitely-realisable physical systems by finite computing systems [41]. A Turing machine does not satisfy this stronger variant of the Church-Turing thesis, because a finite physical system (being continuous) can-

[7] Rules such as "move one step to the left" and "write zero to the tape" [38].

[8] The origins of the design of the stored-program concept and the design of the all-purpose computer caused a great deal of controversy. See [40] for a much more complete account.

not be simulated using a discrete Turing machine based on classical physics. Within this framework, Deutsch introduced the concept of the universal quantum computer, a machine that contained a finite processing quantum system, and an infinite "tape" of quantum memory (like the traditional Turing machine), where quantum operations can be performed with unitary dynamics. For Deutsch, one of the critical aspects of the quantum computer was its "programmability"; it must be possible to reconfigure the same fixed computing system to realise different universal quantum computers, in contrast to Feynman's quantum computer. Deutsch and Jozsa introduced a simple algorithm to exploit quantum effects in this new type of computer [42]. Despite the contrived nature of the problem, the Deutsch-Jozsa algorithm is simple proof that a quantum computer can solve some problems "faster" than an ordinary classical computer[9].

A much more important quantum algorithm, that kick-started widespread interest in quantum computing, is Shor's algorithm for factoring numbers into their prime factors [43]. The algorithm has undoubted significance, due to the possibility that it could undermine RSA, one of the most widely used cryptographic algorithms. Later, Grover introduced an algorithm for performing certain types of search tasks faster than ordinary computers [44]. Although the speed-up from using this algorithm is less impressive than Shor's algorithm, Grover's algorithm attracts interest due to its applicability to a more general type of problem.

The development of other quantum algorithms has turned out to be a very difficult problem; so much so that today, more than 25 years after their invention, Shor's algorithm and Grover's algorithm are still the highest profile quantum algorithms[10]. This shows that, in practice, the universality achieved by quantum computers is significantly less than that of classical computers (where any algorithm based on arithmetic and conditional logic is within the scope of implementation). It also leads to significant differences between classical and quantum computers. For example, a quantum analogue for the stored-program concept does not exist; the data in a quantum computer are quantum states, whereas the program (the specification of a quantum algo-

[9] In this context, "faster" means "using fewer operations". Quantum computers are not well-enough developed yet to decide whether they can solve any problem faster than classical computers, in the sense of wall-clock time.

[10] There are, however, many envisioned applications of variations of these algorithms to real-world problems [45].

rithm in terms of gates) remains in classical form[11]. As a result, quantum computers will remain fundamentally under the control of classical devices.

Despite this, many aspects of classical computing have been imported wholesale into the quantum ecosystem, under the assumption that they will eventually be useful. For example, many quantum programming languages[12] already exist, and big companies such as Microsoft and IBM have already rolled out large-scale software stacks for quantum computing infrastructure that does not yet exist [47, 48]. The disconnect between the apparent infrastructure in place surrounding quantum computing, and the actual readiness of real quantum computers has lead to concern about the level of "quantum hype" [49], which may negatively impact research into quantum computing. In order to meet the high expectations laid out for quantum computers, it is necessary that large-scale quantum computers are available at some point relatively soon. Even if large-scale quantum computers cannot be created immediately, it is certainly feasible to perform a detailed analysis of the type of analog and digital electronics necessary to control them. This information may be used to accelerate the process of large-scale quantum computer development, by ruling out as quickly as possible designs that will fail due to classical electronics-imposed constraints. This thesis is an attempt to offer some methodologies and tools for this analysis in the case of photonic quantum computing.

1.3 Gate-based quantum computing

The basic unit of quantum computation is the qubit, which is a two-state system, analogous to a bit, except complex linear combinations of the zero-state (denoted $|0\rangle$) and the one-state (denoted $|1\rangle$) are also valid states. The states $|\psi\rangle$ of a qubit are elements of the complex vector space $\mathbb{C}^2$ spanned by $|0\rangle$ and $|1\rangle$, which are expressed as

$$|\psi\rangle = a|0\rangle + b|1\rangle = \begin{bmatrix} a \\ b \end{bmatrix}, \quad a, b \in \mathbb{C}; \tag{1.1}$$

the coefficients a and b are called amplitudes.

[11] It has been shown that there is no benefit to generalising the program to include quantum elements. Specifically, it is not possible to create a fixed quantum gate array that can implement universal quantum operations controlled by a finite quantum program, unless the quantum computer is allowed to be non-deterministic [46].

[12] Most of which amount to gate listings.

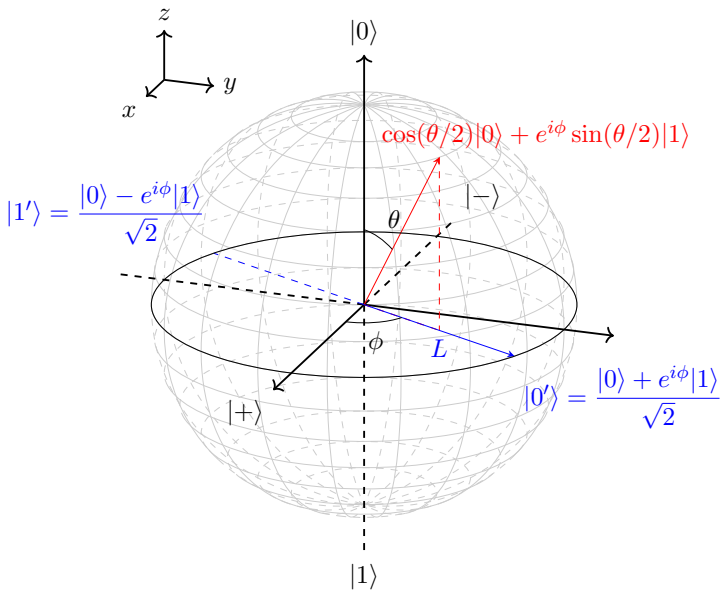

Figure 1.1: The state of a single qubit can be represented as a point on the Bloch Sphere. A measurement of a single qubit can be made along any straight line through the Bloch sphere. Measurements along lines L in the xy-plane of the Bloch sphere are particularly important in MBQC; these are parametrised by a single angle ϕ. Computational basis measurements are made along the vertical line through $|0\rangle$ and $|1\rangle$.

For any particular physical realisation, the qubit can only ever be observed (measured) in the basis states $|0\rangle$ or $|1\rangle$, with probabilities given by the ratio of $|a|^2$ to $|b|^2$. These states correspond to the natural measurable states of the physical system[13]. The absolute values of a and b have no independent physical meaning, so the condition $|a|^2 + |b|^2 = 1$ is imposed so that the probabilities of each outcome are $|a|^2$ and $|b|^2$. Likewise, only the difference between arguments of the complex numbers a and b has physical meaning, so it is possible to impose $a \in \mathbb{R}$ without loss of generality. The argument of b is then the relative phase between $|0\rangle$ and $|1\rangle$.

The states of a single qubit can be identified with points on the surface of a sphere, called the Bloch sphere, as shown in Figure 1.1. The mapping between the coefficients a and b and the angles θ and ϕ is given by the identity:

[13]For the purposes of this discussion, we have dual-rail-encoded photonic qubits in mind (see Section 1.4). As explained in that section, measurements correspond to clicks of photon detectors, which can only result in the $|0\rangle$ or $|1\rangle$ state.

$$a|0\rangle + b|1\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle.$$

The angle ϕ in the equator of the Bloch sphere is the relative phase between $|0\rangle$ and $|1\rangle$, and the angle θ controls the probability of observing $|0\rangle$ or $|1\rangle$ upon measurement.

The states of two qubits can be expressed analogously to Equation (1.1) as

$$|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \tag{1.2}$$

where $a, b, c, d \in \mathbb{C}$. The sum is taken over all four possible states that the two qubits could be observed in, which are the basis states of the tensor product $\mathbb{C}^2 \times \mathbb{C}^2$. As each new qubit is added, the number of amplitudes required to express the state doubles. This leads to the inability of classical computers to simulate quantum circuits containing large numbers of qubits. As with the single qubit case, the condition $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$ is imposed, and the probability of obtaining, for example, $|01\rangle$, is given by $|b|^2$. There is no equivalent of the Bloch sphere for graphically presenting the states of two qubits.

In computer simulations of quantum circuits, the state of the N-qubit system is stored in a vector of complex numbers (amplitudes), of size 2^N:

$$|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{2^N-1} \end{bmatrix}. \tag{1.3}$$

The vector is normalised, so that the $\sum |a_i|^2 = 1$. We discuss the efficient computer simulation of MBQC measurement patterns at length in Chapter 4.

1.3.1 Quantum gates

The state of a multi-qubit system can be changed by applying a quantum gate, which is the term used in quantum computing for a unitary transformation[14].

[14]The word gate is used to draw an analogy between the operations that can be performed on a qubit and the classical logic gates used in digital circuit design.

The gates U on a single qubit, called one-qubit gates, are 2×2 unitary matrices U, which satisfy $U^\dagger U = I$. It can be shown that they correspond to a rotation of the points on the Bloch sphere about any axis, by any angle. The gates which perform rotations of the state about the x, y and z axes are denoted $R_x(\alpha)$, $R_y(\alpha)$ and $R_z(\alpha)$, where α is the angle of rotation according to the right-hand rule. They are given by

$$R_x(\alpha) = \begin{bmatrix} \cos(\alpha/2) & -i\sin(\alpha/2) \\ -i\sin(\alpha/2) & \cos(\alpha/2) \end{bmatrix}, \tag{1.4}$$

$$R_y(\alpha) = \begin{bmatrix} \cos(\alpha/2) & -\sin(\alpha/2) \\ \sin(\alpha/2) & \cos(\alpha/2) \end{bmatrix}, \tag{1.5}$$

$$R_z(\alpha) = \begin{bmatrix} e^{-i\alpha/2} & 0 \\ 0 & e^{i\alpha/2} \end{bmatrix}. \tag{1.6}$$

An arbitrary one-qubit rotation can be formed by applying R_x- and R_z-rotations in sequence as $R_x(\zeta)R_z(\eta)R_x(\xi)$ (applied from right to left). This follows from the decomposition using Euler angles of an arbitrary rotation into R_x- and R_z-rotations. Alternatively, it may be verified by direct matrix multiplication of the gates in Equations (1.4), (1.5) and (1.6) [50].

The Pauli matrices are important special cases of the rotation matrices defined above, given by $X = iR_x(\pi)$, $Y = iR_y(\pi)$ and $Z = iR_z(\pi)$ (the global phase i has no effect on the quantum gate that is performed). The Pauli operators are also important for representing measurement, as described in Section 1.3.2.

Two-qubit gates cannot be visualised as rotations; instead, they are expressed as 4×4 matrices. An example of a two-qubit gate is the controlled-NOT (CNOT) gate, which is described by the following matrix:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{1.7}$$

If the state given by Equation (1.2) is multiplied on the left by the matrix for the CNOT gate, then the amplitudes for the states $|10\rangle$ and $|11\rangle$ are reversed. The interpretation of this gate is that qubit one controls whether an X-gate

(which has the effect of a NOT gate) is applied to qubit zero[15]. Qubit one is therefore called the control qubit, and qubit zero is called the target[16].

Analogously to the way that a NAND gate is universal for digital logic, the CNOT gate combined with the basic rotations $R_x(\alpha)$, $R_y(\alpha)$ and $R_z(\alpha)$ are universal for quantum computation. To build up any complicated computation, all that is required is to apply the correct string of one- and two-qubit gates, one after the other, to a set of qubits. For example, in Figure 2.1c, an arbitrary one-qubit gate $U = R_x(\zeta)R_z(\eta)R_x(\xi)$ is applied to the top qubit, and a CNOT gate is applied between the bottom two qubits.

In general, arbitrarily large multi-qubit gates may be applied to multi-qubit systems. For example, a five-qubit gate would be described by a $2^5 \times 2^5$ unitary matrix. Often, these larger gates are broken down into smaller gates, because it is expected that quantum hardware will not be able to perform arbitrary multi-qubit gates on more than two qubits.

1.3.2 Measurement

When a qubit is measured, it always collapses to either the state $|0\rangle$, with probability $|a|^2$, or the state $|1\rangle$, with probability $|b|^2$. This is called a computational basis measurement.

However, it is possible to generalise the concept of measurement so that an "observation" causes the qubit to collapse into the state $|0'\rangle$ or the state $|1'\rangle$, which are any two antipodal points on the Bloch sphere, joined by a line L. This generalised observation is made by using one-qubit gates to transform the line L to the line through $|0\rangle$ and $|1\rangle$, and then making a computational basis measurement. For example, to measure along the line denoted L in Figure 1.1, it is necessary to apply a z-rotation $R_z(-\phi + \pi/2)$ to align the state $|0'\rangle$ with the positive y axis, followed by an x-rotation $R_x(\pi/2)$ to obtain $|0\rangle$. It is important to realise that these measurements involve the application of a one-qubit gate before making a computational basis measurement.

[15] In this thesis, we will follow a little-endian convention and describe the rightmost qubit as qubit zero (indexing from zero rather than one). For example, in an expression such as $|0011\rangle$, qubit zero is in the state $|1\rangle$ and qubit three is in the state $|0\rangle$

[16] The (classical) CNOT gate originated in the study of reversible classical computers, as part of a wider investigation into the minimum energy required for classical computation [36]. Since all quantum gates are reversible (being unitary), reversible classical gates such as the CNOT naturally give rise to quantum gates as well.

It is possible to measure along any (arbitrary) line by applying an arbitrary one-qubit gate $R_z(\alpha)R_x(\beta)R_z(\gamma)$ and then measuring in the computational basis. However, measurements that lie in the xy-plane are very important for MBQC, and form the basis for arbitrary one-qubit measurement patterns (see Sections 2.1.1 and 6.2.2).

Measurements with two outcomes can be characterised in terms of observables. These are Hermitian operators whose eigenvalues represent the outcomes from the measurement, and whose associated orthogonal eigenvectors are the states that result when each outcome is observed. The Pauli operators are all examples of observables. For example,

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{1.8}$$

is already diagonal. Its eigenvalues are $+1$ and -1, and the corresponding eigenvectors are $|0\rangle$ and $|1\rangle$, showing that Z represents a computational basis measurement[17]. All the other Pauli operators (and all other measurements in this thesis) have outcomes ± 1. We represent the two outcomes in the form $(-1)^m$, where $m \in \{0, 1\}$. This allows the outcome to map more intuitively onto the state produced ($m = 0$ when the outcome is $|0\rangle$ and $m = 1$ when the outcome is $|1\rangle$). We use m exclusively throughout this thesis to refer to measurement outcomes.

The measurement represented by

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.9}$$

is an xy-plane measurement along the x-axis (see Figure 1.1), because its eigenvectors are $|0\rangle + |1\rangle$ and $|0\rangle - |1\rangle$, which[18] are denoted $|+\rangle$ and $|-\rangle$ respectively.

The observable representing a general xy-plane measurement at an angle ϕ to the x-axis is given by $R_z(\phi)XR_z(-\phi)$, because its eigenvectors lie along the blue line L in Figure 1.1 (this is because the R_z rotations rotate this line to the x-axis, which are the eigenvectors of X).

[17] As a result, we use the terms Z-measurement and computational basis measurement interchangeably throughout this thesis.

[18] We will omit normalisation in expressions such as these (here, division by $\sqrt{2}$) to simplify the expression.

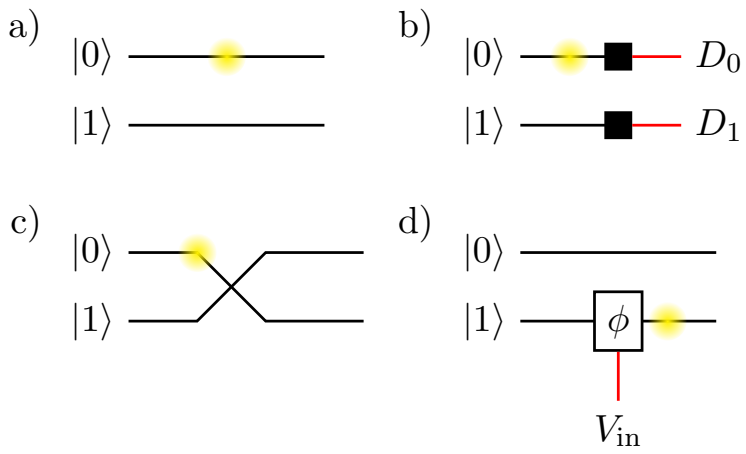

Figure 1.2: a) A single photon in two waveguides can be used as a qubit. If the photon is in the top waveguide, then the qubit is in the $|0\rangle$ state, whereas if it is in the bottom waveguide, the qubit is in the $|1\rangle$ state. b) Computational basis measurements can be performed by placing a single-photon detector at the end of the waveguides. Basic one-qubit operations can be realised using linear optical elements such as c) beamsplitters and d) modulators. Complex operations can be realised by placing the elements one after the other.

1.4 Photonic qubits

In (discrete-variable) photonic quantum computing, a qubit is realised using a single photon. In the dual-rail encoding, a single photon passes through one waveguide or another depending on whether the qubit it represents is in the state $|0\rangle$ or $|1\rangle$, as shown in Figure 1.2. A qubit encoded like this can be measured in the computational basis by placing a single-photon detector at the end of the pair of waveguides. It is important to realise that this process destroys the qubit (by absorbing the photon), unlike a matter-based qubit which can be re-used after measurement.

The advantage of photonic quantum computing is that passive linear optical elements (modulators and beamsplitters) can be used to realise an arbitrary one-qubit gate, as follows. First, a modulator in the $|1\rangle$ waveguide realises an arbitrary R_z-rotation, shown in Figure 1.2d. Then, the variable beamsplitter shown in Figure 1.3 realises an arbitrary R_x-rotation. Finally, a second modulator in the $|1\rangle$ waveguide realises another arbitrary R_z-rotation, which completes the decomposition $R_z(\alpha)R_x(\beta)R_z(\gamma)$.

We consider a simple model for modulators in this thesis, where the phase

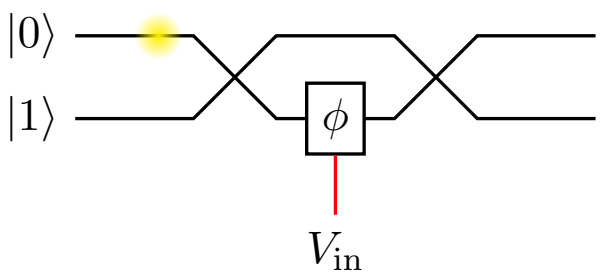

Figure 1.3: A variable beamsplitter, which realises an $R_x(\phi)$ rotation, is formed by placing two fixed beamsplitters on either side of a modulator.

ϕ realised by the modulator is a linear function of the applied voltage V_{in}: $\phi = \pi V_{\text{in}}/V_\pi$. The magnitude of the voltage required to drive the modulator is defined by V_π: the voltage required to realise a $R_z(\pi)$-rotation. There are other important electronic attributes of modulators (for example, small-signal characteristics [51]) which we do not consider here, because we are primarily interested in the digital aspects of electronic control systems for photonic MBQC. A good review of high-speed modulator design for lithium-niobate platforms is [31].

Similarly, we consider single-photon detectors to be devices that output a voltage pulse, compatible with digital logic levels (e.g. $\sim 1.5\,\text{V}$), when a photon arrives. We have high-speed detectors such as superconducting-nanowire single-photon detectors (SNSPDs) in mind [30]; however, we ignore the cryogenic amplification of the detector signal, and how to design reset circuitry [52]. These investigations are outside the scope of the thesis, which contains primarily digital control system analysis.

In contrast to many other physical realisations of quantum computing, including superconducting qubits and trapped ions, that have a natural way to implement two-qubit operations [53], there is no way to implement a deterministic two-qubit photonic entangling gate using passive linear optical elements [54]. However, it was shown that one can implement an artificial non-linear gate that works probabilistically by using additional auxiliary photons and photodetection [55], giving rise to the Knill-Laflamme-Milburn (KLM) scheme for photonic quantum computing. This approach is not robust in the face of photon loss; as a result, modern approaches to photonic quantum computing are based on photonic cluster states [56].

We describe a simple model for photonic quantum computing using a fully-

connected cluster state in Section 2.2. We treat the generation of a photonic cluster state as a black box, and consider the system inputs and outputs to be modulator voltages and detector output pulses, respectively. In Chapter 5, where we consider incomplete cluster states, the success/failure signal for each edge in the cluster state is also assumed to be an input to the system. These models are described fully in the main body of the thesis.

1.5 Reading this thesis

This thesis is partially intended to make the subject of control system design for photonic MBQC easier to understand. In order to help achieve this goal, important information throughout the text is highlighted in boxes, shown below.

Key points

Key takeaways in the main text are highlighted in purple boxes. These are intended to make it easier to find the important information contained in each section.

The "key points" contain a running summary of the material covered in each chapter. An overview of the contents of this document may be obtained by skimming through each section looking for the purple boxes.

Diagrams have been used as a tool in this thesis to attempt to portray complicated mathematical concepts in a simple form that can be easily implemented, without needing to fully understand the underlying mathematics. The primary target audience of the thesis is electronic engineers; consequently, diagrams relating to quantum information have been simplified as much as possible, and those relating to hardware implementations or algorithms are pitched at a higher level. Detailed mathematical treatment of MBQC is available from several excellent resources [20, 21].

Many of the implementations discussed in this thesis can be extended in various ways. These are shown in boxes like the one below:

Next steps

These boxes contain short descriptions of how to extend the algorithms, designs, or results presented in the main text. These are designed to make the suggested "further work" easier to understand.

In order to make the implementation discussions in this thesis more concrete, specific source-code references are included showing which part of the source code relates to the current discussion in the text. C++, VHDL, and python source code is contained in two repositories, shown below. All the code is documented, both inline in the source code, and in specific documentation pages. The code is work in progress, and may continue to be updated. To prevent references in this thesis becoming outdated, the specific (git) commits used for all the source-code references in each repository are shown in the box below.

Source-code reference: mbqc-fpga, MBQCSIM, QSL

Pointers to relevant parts of the source code are listed in boxes like these. Three repositories are referenced in this thesis:

- **mbqc-fpga** (`https://gitlab.com/johnrscott/mbqc-fpga`). All references refer to commit `fde787d174` of the `master` branch.
- **MBQCSIM** (`https://gitlab.com/johnrscott/mbqcsim`). All references refer to commit `5f2ecb32e3` of `master` branch.
- **QSL** (`https://github.com/lanamineh/qsl`). All references refer to commit `7e339d2345` of `master` branch.

The repository for the reference is listed after the colon in the title line of the box. It is hoped that these source-code pointers may make the specifics of the designs and programs discussed here more approachable.

Material from Chapters 2 and 3 was adapted from the paper "Timing constraints imposed by classical digital control systems on photonic implementations of measurement-based quantum computing" [57]. The C++ library QSL (Quantum Simulation Library) was developed in collaboration with Lana Mineh.

Chapter 2

Overview of photonic MBQC

MBQC is a suitable method for quantum computing in systems where one-qubit gates and computational basis measurements are easy to perform, but multi-qubit gates are hard [20]. Photonic quantum computing is one example of such a system, because it is not possible to realise deterministic two-qubit gates using photonic qubits [54].

Even though multi-qubit gates are not required for the main computational step of MBQC, they are not removed entirely. What is required instead is a special state called a cluster state, which is built up using two-qubit entangling gates called fusion gates [13]. The primary advantage of using MBQC is that it is only necessary for a photon to travel through an (approximately) fixed length circuit [14], as opposed to the KLM scheme, where a single-photon qubit must travel a distance proportional to the length of the quantum circuit being implemented [55].

A cluster state on N qubits may be generated by placing all the qubits in the $|+\rangle$ state, and applying CZ gates between pairs of qubits. The cluster state is represented by a graph, whose nodes are the qubits and whose edges are the locations where CZ gates were applied. Throughout this thesis, we consider 2D rectangular cluster states, where the nodes are laid out in a rectangular array, and the edges are horizontal or vertical. It is common to generalise these cluster states to three dimensions [58], or different lattice geometries [59], for reasons mainly relating to fault-tolerance. All the results in this thesis can be extended to arbitrary-shaped cluster states, at the expense of increased complexity of control system design. In this chapter, and Chapters 3 and 4, we consider an ideal cluster state, where edges can be deterministically created

between qubits. In Chapters 5 and 6, we consider incomplete 2D rectangular cluster states, which may be missing edges. These missing edges represent the failure of the probabilistic fusion gates which generate entanglement between the nodes of the cluster state [54].

Despite only using passive linear optical elements, the computational part of MBQC is not particularly simple. The fundamental building blocks of MBQC circuits – called measurement patterns – consist of schemes of one-qubit operations and measurements which can have quite complicated relationships. For example, the outcome of measurements often affect subsequent measurement settings; the outcomes of measurements must be tracked to correct for systematic errors; and the patterns themselves may introduce additional implementation complexities due to backwards-in-time measurement dependencies. In addition, some of these calculations must be performed in real time, extremely fast, to keep up with the throughput of photons through the system.

Often, the investigation of photonic MBQC has lacked information about the control system implementations that are necessary to realise the proposals for photonic quantum computing [13–15]. Accompanying classical control systems are often assumed to feasible due to the "polynomial scaling" of the algorithms required [14], or simply due to the hope that the electronic systems required will not be too hard to build [56].

The scaling of algorithms is an argument used at a high level to justify the development of quantum computers; for example, Grover's algorithm searches a set of size n in time given by $O(\sqrt{n})$, which is (polynomially) faster than the classical solution to the problem (visiting each element of the set one by one), which uses $O(n)$ time steps [50]. Classical algorithms that scale exponentially are often seen as a target for speed-ups for quantum computing (for example, Shor's algorithm achieves this for factoring integers). On the other hand, algorithms that scale polynomially are seen as efficiently solved using classical computers. However, this efficiency does not by itself justify the feasibility of a particular algorithm for solving a particular problem in the face of design constraints. In that case, it is necessary to know (in absolute terms) whether hard timing or other constraints are satisfied, in the particular instance of the problem under consideration; not in the asymptotic limit of generic large problem sizes. For example, in the implementation of the breadth-first search algorithm, required for the use of incomplete cluster states [14], the relevant

question is whether it can be implemented fast enough to meet the requirements of photonic quantum computers. We address this question in detail in Chapter 5.

The practical feasibility of photonic MBQC is often see as resting on the ability to generate cluster states, not performing the computational aspect of MBQC once the cluster state is available. In this and the following three chapters, the practical implementation of this latter MBQC is discussed in detail, by designing and analysing a system for performing one-qubit gates and CNOT gates, based on an ideal cluster state. The results show that a great deal of attention should be paid to the classical control aspects in the problem, which may become bottlenecks in the system, both from the point of view of speed and the ability to scale up the system.

This chapter contains a review of the practical aspects of MBQC, that are relevant to control system design. A certain amount of mathematics is necessary to make the subject of MBQC clear. However, the primary output of the chapter is Figures 2.2 and 2.3, which are the measurement patterns for the one-qubit gate and CNOT gate. These diagrams summarise what must be implemented in a control system for MBQC using ideal cluster states. This chapter may also be seen as a simpler version of Chapter 6, which generalises the diagram for the one-qubit gate to a version appropriate for use with an incomplete cluster state (see Figure 6.3). This chapter also introduces byproduct operators and adaptive measurement settings, which are important implementation details in photonic quantum computing. Finally, it concludes with a simple model of photonic MBQC which will form the basis for subsequent analysis in Chapters 3, 4 and 6

Key points

This chapter introduces the main features required to implement photonic MBQC. The measurement patterns for the one-qubit gate and CNOT gate are summarised in Figures 2.2 and 2.3. Material here is generalised to the more complicated incomplete cluster state in Chapter 6 (which is more mathematical). Section 2.2 contains a simple model of the control system necessary for the ideal cluster state, which is implemented using an FPGA in Chapter 3.

The material from this chapter was adapted from Sections II and III of

the paper "Timing constraints imposed by classical digital control systems on photonic implementations of measurement-based quantum computing" [57].

2.1 Measurement-based quantum computing

Quantum computing in the gate-based model (described in Section 1.3) consists of the following steps:

1. An initial quantum state $|\phi\rangle$ is prepared on N qubits;
2. Quantum gates are applied to the qubits;
3. The resulting state $|\psi\rangle$ is measured, which constitutes the output from the quantum circuit.

MBQC is a different way to obtain the same resulting output state $|\psi\rangle$, by performing single-qubit measurements on a more complicated initial state called a cluster state. It consists of the following steps:

1. Prepare a special quantum state, called a cluster state, on a larger number $M > N$ of qubits. The main feature of the cluster state is that adjacent qubits are entangled together, which is represented using line segments in Figure 2.1a;
2. Measure qubits from the cluster state one at a time, according to rules that correspond to the quantum circuit, until all but N have been measured;
3. Finally, the resulting state $|\psi'\rangle$ on the N remaining qubits is measured in the computational basis, which constitutes the output from the circuit.

In the gate-based model, if each qubit is initially prepared in the $|+\rangle$ state, then the output states $|\psi\rangle$ and $|\psi'\rangle$ from the gate-based model and MBQC are the same[1]. This means that any algorithm expressed in the gate-based model can be equally well performed using MBQC.

Figure 2.1 shows an example rectangular cluster state, along with a schematic representation of the measurement patterns for a quantum circuit shown in

[1] Up to some systematic corrections called byproduct operators, which are described in Section 2.1.4

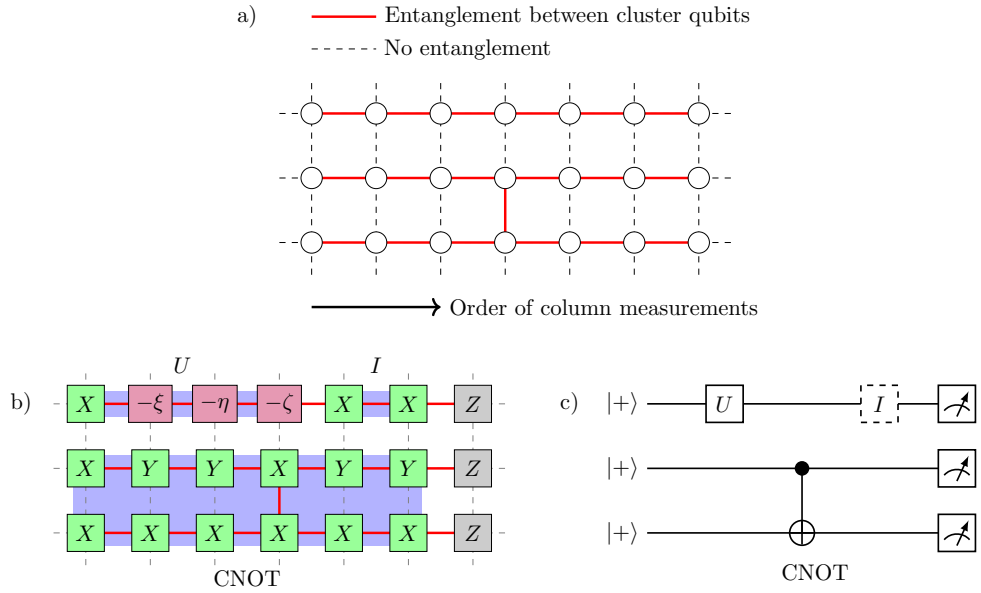

Figure 2.1: a) A cluster state is made from a rectangular array of qubits (the white dots), each of which may be entangled with its four nearest neighbours. When a computation is performed, a specific pattern of entanglement is required that matches the shape of the circuit. b) The quantum computation is performed by measuring the cluster qubits in bases derived from the measurement pattern. The shaded blue regions show which cluster qubits are involved in implementing which gates. The identity gate is included to pad the length of the one-qubit gate $U = R_x(\zeta)R_z(\eta)R_x(\xi)$ so it matches the CNOT. c) The quantum circuit that is performed by the measurement pattern in b).

the gate-based model. The subsequent sections describes what is required to implement these types of quantum circuits in detail.

For a comprehensive overview of MBQC, see [20]. A short pedagogical introduction is contained in [21]. What follows is a brief description of the main features of MBQC which are relevant to the implementation of photonic MBQC in this thesis.

2.1.1 Logical qubits and measurement patterns in MBQC

Each horizontal line of entanglement in the cluster state corresponds to a single qubit in the gate-based model, which we call a logical qubit, to distinguish it from the cluster qubits that make up the cluster state. One-qubit gates in the gate-based model are realised by measuring cluster qubits along a logical-qubit

row according to rules that determine the basis settings of each measurement, and define what to do with the measurement outcomes. These rules are called measurement patterns. Two-qubit gates require vertical lines of entanglement which join the logical qubit rows together, as shown in Figure 2.1a between the second and third row.

Each gate G that is implemented in MBQC is defined by a measurement pattern, which is a set of rules describing:

- How many cluster qubits are needed to realise the gate G and what pattern of entanglement is necessary between those cluster qubits;
- Which basis to use for each cluster qubit measurement;
- How to process the outcomes from the cluster qubit measurements.

A given computation involving multiple gates, such as the one shown in the gate-based model in Figure 2.1c, can be performed using MBQC by concatenating the measurement patterns for each gate (the blue shaded regions in Figure 2.1b)[2]. The resulting pattern contains one row for each qubit in the gate-based model (here, $N = 3$), and a number of columns defined by the length of the concatenated measurement patterns (the total number of cluster qubits is $M = 21$).

In making the measurements defined by the measurement patterns, each cluster qubit is removed one by one until only the rightmost column remains unmeasured. The final column of the cluster state is measured in the computational basis as shown in Figure 2.1b, and these measurement outcomes can be used to obtain the output from the quantum circuit.

The arbitrary one-qubit gate U in Figure 2.1c is realised using a measurement pattern of four cluster qubits in the top row of Figure 2.1b, and the CNOT gate[3] is realised using a measurement pattern of 12 cluster qubits

[2] In [20], measurement patterns are taken to include the "output" qubits, which is the first column of qubits directly to the right of the measurement pattern. In this scheme, measurement patterns must overlap (because the output qubit column is also the input qubit column for the next gate pattern). Here, we associate the output qubit with the next measurement pattern, so that patterns can be simply concatenated.

[3] The symbol for a CNOT gate shown in Figure 2.1 is the same as a classical XOR applied to the target qubit. This is because the CNOT gate can be thought of as adding the value of the control qubit to the target qubit modulo 2. We also make use of the classical XOR operation in subsequent figures in this thesis. It should be clear from the context whether a CNOT gate or an XOR operation is meant.

spanning the bottom two rows of Figure 2.1b (note the vertical entanglement link).

All measurements shown in green and purple boxes in the figure are performed along lines L that lie in the equator of the Bloch sphere (Figure 1.1). Green boxes containing X or Y are measurements along the x- or y-axis, respectively. Purple boxes are measured along a line L with an angle ϕ derived from the value in the box and measurement outcomes of other cluster qubits. The grey boxes represent computational basis measurements, which are made along the z-axis of the Bloch sphere.

Key points

Measurement patterns are the rules which allow quantum circuits to be implemented on cluster states. The control system must be able to set measurement bases, and process measurement outcomes from the cluster qubits. As described in the following sections, measurement bases may depend on previous measurement outcomes, and all measurement outcomes contribute to systematic correctable errors, called byproduct operators.

2.1.2 Performing the cluster qubit measurements

As described in Section 1.3.2, the only physical measurements that can be performed are computational basis measurements (realised by measuring the $|0\rangle$ or $|1\rangle$ waveguide in the dual-rail encoding). All the other measurements relevant for MBQC are performed in the xy-plane of the Bloch sphere, by applying a one-qubit gate to the given cluster qubit and then measuring it in the computational basis.

It is important to understand that the one-qubit gates that set the measurement bases in the measurement patterns are different from the one-qubit gates implemented by MBQC, such as U in Figure 2.1. The former are basic operations that, together with computational basis measurements, are required for implementation of MBQC. They are analogous to the physical layer in a communication system, because they must be realised by some physical mechanism; for example, using photonic qubits, as we discuss in Section 1.4. The logical one-qubit gates U do not correspond to any basic physical opera-

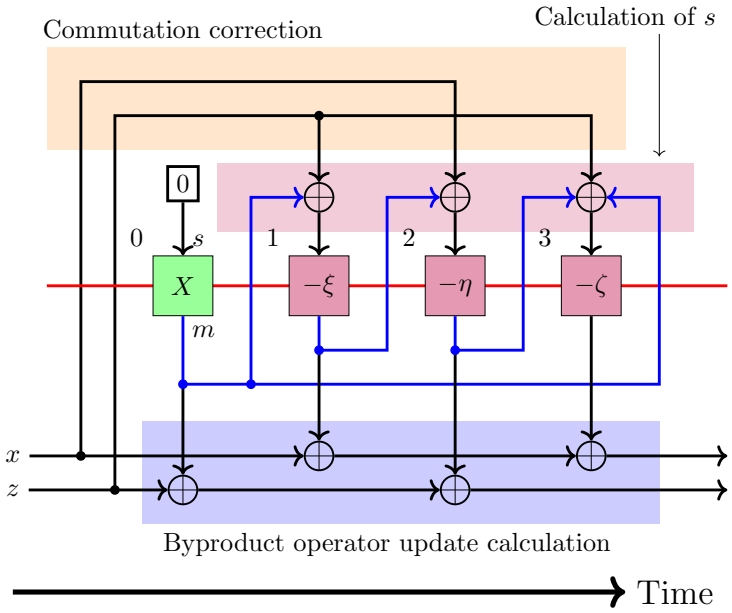

Figure 2.2: The arbitrary one-qubit gate $U = R_x(\zeta)R_z(\eta)R_x(\xi)$. The black line connected to the top of the box is the adaptive measurement setting s. The line connected to the bottom of each box is the measurement outcome m. The byproduct operator calculation (shaded blue) is simple, but the adaptive measurement settings depend on previous measurement outcomes (shaded purple). The byproduct operators must be stored because they are used in the adaptive measurement setting calculation (shaded in orange). In Section 2.2.1, the condition is imposed that columns are measured from left to right, so as to be compatible with photonic MBQC.

tion, and instead arise as a result of applying the measurement pattern to the cluster qubits. They are analogous to the logical data layers in a communication network, which use the resources of the physical layer to transmit logical information.

Figures 2.2 and 2.3 summarise all measurement pattern rules that are required to implement the arbitrary one-qubit and CNOT gate [20]. In the following sections we describe in detail how to interpret these diagrams, and provide a basic overview of the mathematics underpinning them.

2.1.3 Measurement basis angles and adaptive measurements

Every xy-plane measurement that is part of a measurement pattern is measured along a line L in the equator of the Bloch sphere, as shown in Figure 1.1.

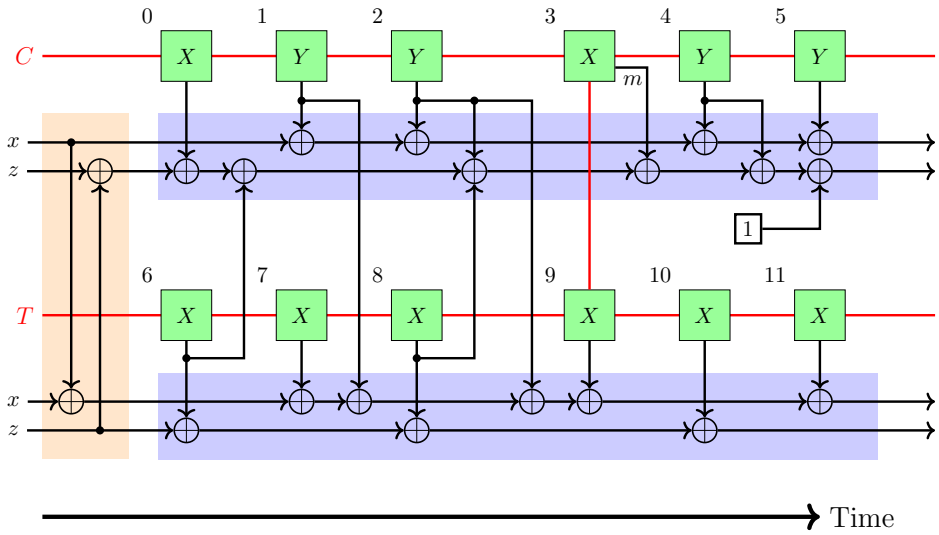

Figure 2.3: The measurement pattern for the CNOT gate. The line connected to the bottom of each box is the measurement outcome m. There are no adaptive measurement settings, because all the measurement bases are X or Y. However, the computation of the byproduct operators (shaded in blue) is more complicated, and involves mixing outcomes from the control C and target T rows. The commutation correction is shaded in orange. It involves mixing the byproduct operators before applying the pattern.

It is therefore specified by one real angle ϕ. In MBQC measurement patterns, this angle is made up of a base value θ, and a sign bit s, such that $\phi = (-1)^s\theta$. The value of θ is shown in the purple boxes in Figure 2.1b. Note that θ may be negative. In the case of the one-qubit gate, these values encode the logical rotations that are performed by the measurement pattern.

For the green boxes in Figure 2.1, the value of θ is 0 for X and $\pi/2$ for Y, and correspond to measurements of the X and Y Pauli operators (as described in Section 1.3.2). In the first case, the value of s does not affect the basis angle ϕ at all. In the second case, the roles of $|0'\rangle$ and $|1'\rangle$ are swapped because L is reversed; however, since the outcome of the measurement is recorded, the control system can correct for the effect of the swap (in the calculation of byproduct operators – see Section 2.1.4). Therefore, the X and Y measurements are not affected by the value of s.

The value of θ is a characteristic of the quantum circuit being implemented, and sets what logical gate is performed. The value s, however, depends on the outcomes of other (prior) measurements in the measurement pattern. The

measurement bases in MBQC are therefore adaptive, because the basis (the angle ψ) in which a cluster qubit is measured may depend on the outcomes of measurements of other cluster qubits which have been measured before it. We will refer to s henceforth as the adaptive measurement setting.

Any measurement pattern, such as the CNOT gate, which contains only X and Y measurements, does not involve adaptive basis settings, because the value of s has no effect. It can be shown that the set of gates implementable with these non-adaptive patterns is the Clifford gate set [20], which is not universal [50]. For universal quantum computing, it is necessary to include a gate such as the one-qubit gate which does require adaptive measurement settings.

In Figure 2.2, the measurement pattern for the arbitrary one-qubit gate (corresponding to a rotation of the Bloch sphere) is shown in detail [20]. The shaded purple region (particularly the blue wires) shows how the adaptive measurement setting for each measurement is computed from previous measurement outcomes. The dependence between s and measurement outcomes implies that the measurements must be made from left to right, which is also indicated by the arrow of time at the bottom of the figure.

The measurement pattern for the CNOT gate is shown in Figure 2.3. This is not the same pattern presented in the original MBQC paper [20], which uses three logical qubit rows. The derivation of the CNOT pattern in Figure 2.3 is contained in Appendix A. We use this modified CNOT measurement pattern because it considerably simplifies our example digital implementation in Section 3.2, which only supports nearest-neighbour connectivity of logical qubits (where the control and target are on adjacent cluster-state rows).

Key points

The xy-plane measurement bases for measurement patterns are made from a base value θ, defined by the quantum gate, and an adaptive measurement setting s. As described above and shown in Figure 2.2, s is calculated by combining the measurement outcomes m from other cluster qubits, and commutation corrections, described in Section 2.1.5 below. The CNOT pattern (Figure 2.3) does not require adaptive measurement settings.

2.1.4 Byproduct operator calculations

As the measurement pattern proceeds, the random outcomes of the measurements introduce correctable errors in the computation. These errors are known as byproduct operators, because they are unintended logical operations which occur as a byproduct of the MBQC measurements [20].

Specifically, after any N-qubit gate G has been applied to a state $|\psi\rangle$ using its measurement pattern, the resulting state is actually $BG|\psi\rangle$, rather than simply $G|\psi\rangle$, where B is a gate (called the byproduct operator) given by

$$B = \prod_{i=1}^{N} Z_i^{z_i} X_i^{x_i}, \qquad x_i, z_i \in \{0, 1\}.$$

The byproduct operator for the logical qubit i is specified by two bits x_i and z_i, which are updated as the computation proceeds. By an abuse of notation, we will refer to the pair (x_i, z_i) as the byproduct operator as well. For N logical qubits (N rows of the cluster state), $2N$ bits are needed to store the byproduct operators. At the start of the computation, they are all initialised to zero, because no gate has been performed so no errors have been introduced. As the computation proceeds, the outcomes of the measurements in the pattern are XORed into the x_i and z_i according to prescribed rules, described below.

For the one-qubit gate in Figure 2.2, the new byproduct operators (x', z') are calculated according to the rule

$$\begin{aligned} z' &= z \oplus m_0 \oplus m_2 \\ x' &= x \oplus m_1 \oplus m_3, \end{aligned}$$

where m_k is the measurement outcome from the k^{th} qubit, numbered according to Figure 2.2.

For the CNOT pattern shown in Figure 2.3, two byproduct operators are involved, one for the control qubit row (x_c, z_c) and one for the target qubit row (x_t, z_t). The new byproduct operators (x'_c, z'_c) and (x'_t, z'_t) are calculated using

$$\begin{aligned} z'_c &= z_c \oplus m_0 \oplus m_2 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_8 \oplus 1 \\ x'_c &= x_c \oplus m_1 \oplus m_2 \oplus m_4 \oplus m_5, \end{aligned} \tag{2.1}$$

and

$$\begin{aligned} z'_t &= z_t \oplus m_6 \oplus m_8 \oplus m_{10} \\ x'_t &= x_t \oplus m_1 \oplus m_2 \oplus m_7 \oplus m_9 \oplus m_{11}. \end{aligned} \tag{2.2}$$

Unlike for the one-qubit gate, the byproduct operators for a given logical qubit row are calculated using measurements from other rows. Note the addition of the constant 1 in the control qubit byproduct operator.

On the face of it, byproduct operators appear to introduce errors into the computation, because the gate BG is performed instead of the desired gate G. However, the effect of this error can be corrected after the final column of Z-measurements in the MBQC process has been performed: the outcome from any logical qubit row i where $x_i = 1$ has its outcome flipped from a zero to a one or vice versa [20]. This action undoes the effect of the byproduct operators, leaving a circuit that effectively only implements the gate G as desired. The z_i components are not used because they correspond to a phase shift which does not affect the probability of measuring a zero or one in a computational basis measurement. However, as we describe in the next section, it is necessary to keep track of their values because they can affect the value of the x_i, through the process of commutation corrections.

Key points

Byproduct operators are two bits (x, z) per logical qubit that account for the randomness of measurement outcomes in MBQC. All measurement outcomes contribute to byproduct operators. These bits are shown along the bottom of diagrams such as Figure 2.2 and 2.3. Even though the CNOT gate does not use adaptive measurement settings, its byproduct operators are more complicated, involving mixing outcomes from both logical qubit rows. This connectivity between measurement outcomes and byproduct operators must be accounted for in the control system design.

2.1.5 Commutation corrections

The byproduct operators are used to correct the outcomes obtained after the MBQC circuit is finished. However, the correction only works if the byproduct operators are the last operation before the final column computational basis measurement, which is only the case if a single gate G is performed.

If multiple gates G_k are performed on a state $|\psi\rangle$, then the resulting state

$|\phi\rangle$ will be

$$|\phi\rangle = (B_K G_K) \dots (B_1 G_1)(B_0 G_0)|\psi\rangle. \tag{2.3}$$

These interleaved byproduct operators cannot be corrected at the end of the circuit. Instead, it is necessary to move all the byproduct operators to the end (the leftmost side of the equation). To do that, after each new gate G_{k+1} is applied, it is necessary to commute the current byproduct operators B_k and the gate G_{k+1}, so that the byproduct operators are always on the leftmost side of the equation. This is illustrated below for the application of the second gate G_1:

$$B_0 G_0|\psi\rangle \mapsto B_1 G_1 B_0 G_0|\psi\rangle \mapsto B_1 B_0' G_1' G_0|\psi\rangle \mapsto B_r G_1' G_0|\psi\rangle, \tag{2.4}$$

where $G_1 B_0 = B_0' G_1'$, and the prime indicates the change that may occur in either gate. The byproduct operators B_1 and B_0' can be combined into a resulting byproduct operator B_r by adding together the values of (x_i, z_i) bitwise modulo 2 for each operator. The state on the far right of Equation (2.4) is therefore in the same form as the state on the far left, so that on the application of the next gate G_2, the process can be repeated and the byproduct operators are always kept on the left. We refer to the process of commuting B through G as a commutation correction.

In practical terms, the commutation correction is an operation that is performed before a gate is applied, by manipulating the current value of the byproduct operators and the upcoming gate so as to have the effect of Equation (2.4). For the measurement patterns we consider in Figures 2.2 and 2.3, the commutation corrections are quite simple. In the case of the CNOT gate $G = \text{CNOT}$, there is no modification necessary for the gate itself ($G' = G$), and only the byproduct operator B changes to B', according to the rule

$$\begin{aligned} z_c' &= z_c \oplus z_t \\ x_c' &= x_c \\ z_t' &= z_t \\ x_t' &= x_t \oplus x_c. \end{aligned} \tag{2.5}$$

For the one-qubit gate $G = U$, the byproduct operators remain the same ($B' = B$) but the gate itself G must be modified. The modification is made by using the values of the byproduct operators to affect the adaptive measurement settings, by XORing the byproduct operators with previous measurement outcomes to form the values of s for each cluster qubit [20], as shown in

Figure 2.2. The calculation of the adaptive measurement settings s_j for each cluster qubit j is shown in the following equations

$$\begin{aligned} s_0 &= 0 \\ s_1 &= m_0 \oplus z \\ s_2 &= m_1 \oplus x \\ s_3 &= m_0 \oplus m_2 \oplus z. \end{aligned} \tag{2.6}$$

It is necessary to make a copy of the byproduct operators (x, z) before measuring the cluster qubits, because otherwise they will be overwritten during the calculations described in the previous paragraph. For example, after the measurement of cluster qubit 1 in the arbitrary one-qubit gate in Figure 2.2, both the x and z values have been updated by measurement outcomes from cluster qubits 0 and 1. However, the old values of x and z are necessary in the measurement settings for cluster qubits 2 and 3.

Key points

Commutation corrections are really an artefact introduced by concatenating measurement patterns, instead of considering one long pattern. In Chapter 6, where we consider measurement patterns laid out along paths through cluster states, the commutation corrections are absorbed into a more accurate representation of byproduct operators. The reader may like to compare Figure 2.2, for the one-qubit gate, with Figure 6.2, which is a generalisation that covers any length pattern, and does not need commutation corrections.

To illustrate the concepts in these sections, the reader may refer to the example two-logical-qubit calculation in Table 3.3, which illustrates a two-qubit measurement pattern comprising a randomly chosen one-qubit gate $U = R_x(0.3)R_z(0.2)R_x(0.1)$ on qubit 0, followed by a CNOT gate between qubits 0 (the target) and 1 (the control). The values of s are always zero on the CNOT gate, whereas the values of s depend on the measurement outcomes as shown in Figure 2.2.

In the remainder of the chapter, we discuss how to map these measurement patterns to a simple implementation model, which treats the cluster-state generation as a black box that is able to deterministically produce cluster

states with edges in the correct places for the measurement pattern, and where the components required for the control system are specifically identified.

2.2 Simplified model of photonic quantum computing

In this section we describe how to implement MBQC, in the ideal-cluster-state model described above, using photons as qubits. We assume an ideal photonic cluster-state generator, which can generate arbitrarily connected rectangular cluster states, with the appropriate edges for the desired measurement pattern.

The main feature of the model, shown in Figure 2.4, is that the cluster state is generated one column at a time, and each row of the cluster state is processed by a measurement block (responsible for setting measurement bases), an analog interface (for controlling the measurement block and amplifying measurement outcomes), and a digital system (which is shared across all rows of the cluster state).

Key points

In this thesis, we focus on the simplest part of the control system: the digital system design. This component places timing constraints in the form of lower-bounds on the processing time of the control system. However, in this section we discuss the full control system model, including an overview of what analog building blocks are necessary. This is intended to show how the digital subsystems would fit into an overall control system implementation.

2.2.1 Photonic MBQC

For matter-based implementations of MBQC, the grid of cluster qubits directly corresponds to a two-dimensional array of physical systems, such as atoms. However, for photonic quantum computing, it is not feasible to maintain a static array of qubits for long enough to perform the measurements. This is because a photon is always moving, so the only way to store it is to place it in a long waveguide, called a delay line, or keep it circulating in an on-chip cavity, such as a microresonator [60]. Both of these approaches eventually

lead to photon decay, primarily due to scattering and absorption loss in the waveguide.

Instead, the cluster state can be generated one column at a time, and each column is measured one after the other. This is opposite to the original presentation of MBQC [20], where the goal was to separate the processes of generating the cluster state and making the measurements. The original motivation for generating the cluster state all at once in matter-based systems was also due to physical considerations: a cluster state can be generated using a tunable Ising interaction that acts globally on the system [21]. However, it can be shown that the two approaches are equivalent [20, Section II.D][4].

When the cluster-state generation and the photon measurement is alternated, a single photon only has to travel from its source, through the cluster-state generator, through a fixed-length waveguide, and finish at the measurement block. This is the definitive improvement enabled by MBQC, which is not possible using the KLM scheme.

For photonic MBQC, the horizontal axis in Figures 2.2 and 2.3 can be interpreted as time, and the vertical axis as space. Using this approach introduces a restriction which is not present in the matter-based realisation of MBQC: the scheme is only viable if the measurement settings for the currently measured block only depend on the outcomes of previously measured columns. This is quite a severe restriction, ruling out many of the measurement patterns originally proposed in [20] (for example the CPhase gate, a two-qubit gate that depends on a continuous parameter). However, this requirement is satisfied for the one-qubit gate and the CNOT gate described here. In the case of the CNOT gate, there are no measurement dependencies. For the one-qubit gate, all the measurement dependencies (the blue lines in Figure 2.2) point from left to right[5].

2.2.2 Timing constraints on the cluster state

We do not consider the generation of the photonic cluster state, apart from making the following remark about the choice of time delay between the generation of columns, which is crucial for our timing analysis.

[4] There, the successive column approach is used as tool for verifying measurement patterns.

[5] In the context of photonic quantum computing, this is sometimes referred to as feed-forward of measurement results.

In order to entangle photons P_n and P_{n+1} from two adjacent columns n and $n+1$ of the cluster state, they must be brought to the same location (for example, a beamsplitter) at the same time. However, when performing the cluster qubit measurement for the MBQC measurement pattern, P_n (from column n) must arrive at the detector a finite time before P_{n+1} (from column $n+1$), to allow time for the processing of measurement settings, byproduct operators and commutation corrections. Therefore, P_{n+1} must experience a delay T_p (realised using an on-chip delay line or optical fibre) after the entangling operation of adjacent columns and the measurement block. The inverse of this delay $X_p = 1/T_p$ is the photonic clock frequency, which is the rate at which columns are produced and measured, and which determines the speed at which the quantum computation progresses. We do not consider how this value arises from the detailed construction of the cluster-state generator. Instead, we take it as a design parameter, and use it to derive timing constraints on the overall system.

Two distinct physical mechanisms provide upper and lower bounds for this delay. An upper bound is given by the loss of the on-chip delay line, optical fibre, or routing system involved in the delay of the photon. The lower bound is given by the time required to process the measurement outcomes. The object of the design and timing analysis in Chapter 3 is to estimate the lower bound. We do not consider the other constraints on T_p which arise from the construction of the cluster-state generator, of which there are many. We take $T_p = 1\,\text{ns}$ as an estimate for the column-generation delay, for the purpose of making concrete calculations about timing constraints. It is difficult to justify a particular choice for T_p without having to hand a particular design for the cluster-state generator; however, a value in the GHz is often chosen as a ballpark for the expected speed of photonic quantum computing systems [61, 62]. The calculations performed in this thesis can easily be modified to suit any other value of T_p.

2.2.3 The full MBQC system

Figure 2.4 shows the full system required for processing one row of the MBQC measurement pattern, which corresponds to one logical qubit. It consists of the following five parts:

- **Cluster-state generator**, which outputs the dual-rail encoded photon

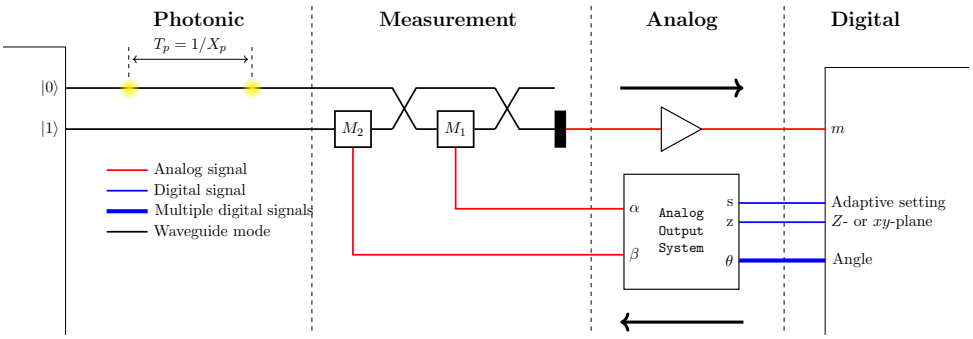

Figure 2.4: One row of the system diagram of the classical control required to implement photonic MBQC. The cluster-state generator is assumed to be ideal, outputting columns of photons at the photon clock frequency X_p. The cluster qubits represented by these photons are measured in bases specified by the measurement pattern in the measurement block, which is controlled by the voltages α and β from the analog output system. The measurement results are amplified by an analog system, and processed by the digital system into subsequent adaptive measurement settings s and byproduct operators. A copy of the system shown is required for each logical qubit, but each block is independent apart from the cluster-state generator and the digital system.

in each column of the cluster state one after the other. The photon has been entangled with the previous photon in the same row, and with the photons in the rows above and below as necessary for the measurement pattern.

- **Delay line**, described in the previous section, which is necessary to temporally separate the photons in adjacent columns after they have been entangled.

- **Measurement block**, which consists of passive linear optical elements that apply a configurable one-qubit operation, followed by a computational basis measurement.

- **Photon detector amplifier**, which converts the output from a single-photon detector to a logic level suitable for input to a digital system.

- **Analog output system**, controlled by the digital system, which produces the analog voltage levels needed to drive the modulators in the measurement block.

z	Rotation	Measurement basis
1	$R_z\left(\frac{\pi}{2}-\phi\right)R_x\left(\frac{\pi}{2}\right)$	xy-plane measurement at an angle ϕ
0	(None)	Computational basis measurement

Table 2.1: The table shows the one-qubit rotations generated by analog system modulator voltages. When $z = 1$, a regular xy-plane measurement is performed, which accounts for the majority of cluster-state measurements. A computational basis measurement is made at the end of the computation by setting $z = 0$.

In addition, there is the **digital system**, which processes measurement outcomes into adaptive measurement settings and keeps track of byproduct operators. This is shared across all the logical qubit rows, because it is necessary to account for the non-locality of byproduct operator calculations, as described in Section 2.1.4.

The input to the digital system is the output pulse from the photon detector amplifier. This may be, for example, an SNSPD [63] followed by a low-noise amplifier [52]. We do not consider the design of these analog stages in this thesis; the digital system is a more straightforward first step, which is able to impose timing bounds on the system that may form part of the specification of the analog systems.

The output from the digital system includes the digital form of the angle θ, the adaptive measurement setting output s, and a signal z which determines whether the measurement is in the xy-plane of the Bloch sphere, or if it is a computational basis measurement.

The analog output system is responsible for generating the voltages that control the modulators in the measurement block. It may be implemented using a combination of fast DACs and modulator drivers. Two modulators are necessary: one (M_1 in Figure 2.4) chooses between an xy-plane measurement and a computational basis measurement; and another (M_2) controls the basis angle ϕ for the xy-plane measurement. They are controlled by the voltages α

and β respectively, defined as follows[6]

$$\begin{aligned} \alpha &= \frac{\pi}{2} z, \\ \beta &= \frac{\pi}{2} - \phi = \frac{\pi}{2} - (-1)^s \theta. \end{aligned} \tag{2.7}$$

These modulator voltages realise the one-qubit rotation $R_x(\alpha)R_z(\beta)$, which sets the basis for the measurement. The one-qubit rotations are summarised in Table 2.1.

The voltage α controls the R_x rotation portion of the measurement setting, which determines whether the measurement is a computational basis measurement ($z = 0$) or an xy-plane measurement ($z = 1$). The voltage β controls the angle of the xy-plane measurement ϕ, which is itself determined by the fixed value θ and the adaptive measurement setting s.

Key points

All the timing constraints discussed in this thesis arise from the design parameter T_p, in combination with the control system model shown in Figure 2.4. The time T_p places an upper bound on the execution time of the feedback loop starting at the single-photon detector and finishing on the modulator input voltages. By designing the digital system, part of this execution time may be determined to be T_d. This leaves the remaining time, $T_a = T_p - T_d$, for the latency inherent in the analog stages. If $T_d > T_p$, then either the delay line length must be increased, or the digital part of the control system must be improved.

2.3 Summary

This chapter provides a practical overview of MBQC, based on Figures 2.2 and 2.3, which show what logic must be implemented by the digital part of the control system for photonic quantum computing. We assume access to an ideal cluster state, which can deterministically entangle the cluster qubits according to the measurement pattern. This device does not exist, because deterministic two-qubit gates between photonic qubits are not possible [54].

[6] Voltages are expressed in modulator-phase units, where $V = 1$ is chosen such that the modulator applies a 1 rad phase shift.

However, it provides a simple setting in which to explore the implementation of many features that do occur in all implementations of MBQC; for example, the calculation of adaptive measurement settings and byproduct operators.

In Chapter 3, we focus on implementing the digital control system shown in Figure 2.4, and present a design capable of performing the one-qubit gate and CNOT gate shown in Figures 2.2 and 2.3. We analyse the timing behaviour of this design by implementing it with an FPGA and performing static timing analysis, based on the timing parameter T_p. The main objective of this analysis is to place timing constraints on the input and output analog systems, and therefore on the overall quantum photonic clock rate of the system.

In Chapter 6, we extend the ideas in this chapter to measurement patterns compatible with cluster states containing random missing entanglement links.

Chapter 3

Control system design for photonic MBQC

This chapter contains an example digital system design for implementing the arbitrary one-qubit gate and CNOT gate described in the previous chapter. The digital system takes measurement pulses from single-photon detectors as input, along with a program for specifying the measurement pattern, and outputs adaptive measurement settings and byproduct operators for each measurement round.

The design and timing analysis data is available in a public repository [64]. This chapter was adapted from Sections IV and VI of the paper "Timing constraints imposed by classical digital control systems on photonic implementations of measurement-based quantum computing" [57].

3.1 Overall system design

The object of the design presented here is to provide a concrete implementation of a digital control system capable of realising one-qubit gates and CNOT gates, in a manner that also lends itself to the analysis of timing constraints relating to the system. The use of an FPGA to prototype the design is appropriate for three reasons:

- **Rapid prototyping.** When a system is being prototyped for exploratory reasons (for example, to assess ballpark constraints and test out implementation strategies), the ability to rapidly generate working designs is very important. Proven designs can then easily be converted to custom

platforms, such as application-specific integrated circuits (ASICs), which may even amount to simply porting the hardware description language source code.

- **Automatic timing constraint analysis.** FPGA designs are automatically optimised and analysed for timing closure, which greatly simplifies the process of exploring design constraints.

- **Reasonable high-performance design.** Although FPGA designs will not provide an optimal digital design (a custom platform is necessary for that), the performance of an FPGA-based design still provides a sound indication of what is possible in modern high-speed digital design. For example, critical path delays have been found to decrease by 3-4 times in standard-cell ASIC designs, but this does not imply that unlimited improvement is available [65].

We use a high-end Xilinx FPGA (7 series) for our design [66] – part number xc7k70tfbg484-2. This series was chosen to maximise the clock frequency of the global clock tree (710.00 MHz [67]) while offering dedicated level-sensitive input latches [68] in order to minimise the input delay of the measurement signal. In addition, the device and required synthesis and implementation tools are supported by Xilinx under a free license, allowing the design to be used and modified by the widest possible group.

The overarching intent with this design was to maximise the maximum clock frequency of the system by fitting the computational aspects of each measurement round into one clock cycle. To facilitate this, the clocking facilities built into the FPGA were used to generate several out-of-phase clocks, to initiate different parts of the computation process. We discuss this design choice further in Section 3.3.

Key points

Targeting an FPGA is a good starting point for obtaining ballpark timing information, and rapidly testing different implementations of the logic required for photonic MBQC. Design optimisation and analysis is performed automatically by synthesis tools, and the source code for the design may form the basis for an improved system targeting a custom hardware platform (e.g. an ASIC).

Source-code reference: mbqc-fpga

The design is written in VHDL, contained in the folder `design/mbqc.srcs/sources_1/new`. Of these files, the most interesting from the point of view of implementing MBQC are `adapt.vhd` and `byproduct.vhd`, which contain the logic for computing adaptive measurement settings and byproduct operators, described in Section 3.2. The full design should be viewed and modified using Xilinx Vivado [69], by running `vivado design/mbqc.xpr` from the top level of the repository.

3.1.1 Logical qubit unit cells

The digital control system consists of several digital "unit cells", each of which is responsible for one logical qubit row in the cluster state. These unit cells are shaded in green in Figure 3.1, which shows the overall system design. The majority of the complexity in the system is contained in the design of the computational system (see Section 3.2), which is responsible for computing adaptive measurement settings s and byproduct operators b. The unit cells are connected together in such a way that each is able to access the measurement outcomes and byproduct operators from the unit cells above and below. This is necessary to support the byproduct operator calculations in the CNOT pattern in Figure 2.3.

The photon input to the system is considered to be a voltage pulse that is compatible with digital logic levels (here, the logic levels of the digital inputs in the target FPGA). This signal must be converted to a constant digital level so that it is compatible with the sequential logic used in the design. A latch performs this conversion: when the pulse arrives at the input to the latch, the output is held at constant logic one until the latch is reset. If no pulse arrives, the output remains at zero.

The digital design supports arbitrary measurement patterns made by concatenating one-qubit gates and CNOT gates. These patterns are stored in a simple program format (consisting of one two-byte word per clock cycle, per unit cell), which controls what operations the computational system should perform. The program word specifies how to calculate adaptive measurement

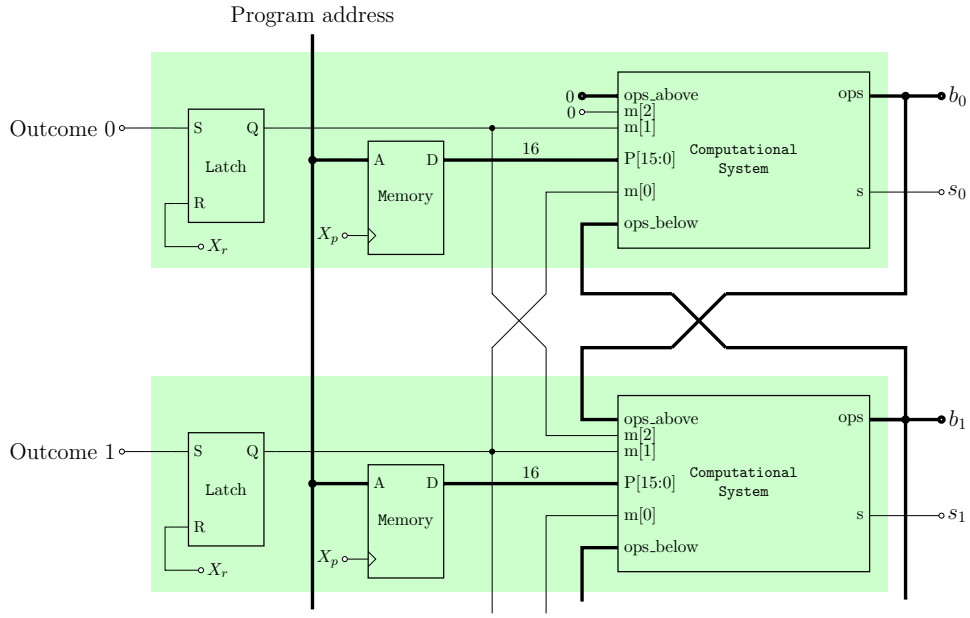

Figure 3.1: The digital system diagram for multiple qubits. The unit cell for each qubit (shaded green) has a measurement latch, a program memory, and a control system for calculating measurement settings and byproduct operators.

settings, how to calculate byproduct operators, and whether a commutation correction is necessary.

A summary of the notation used in this chapter is provided in Table 3.1.

3.1.2 Clock planning

We present a design that can process measurements within a single clock cycle, by using three out-of-phase clocks, as shown in Figure 3.2. We consider a system synchronous design, with the photonic clock X_p the common (master) clock in the system.

On the rising edge of X_p, the photon arrives in the measurement block, causing a pulse at the output of the single-photon detector. This measurement outcome is amplified and triggers a latch which provides a constant digital signal to the digital system.

The other two clocks, X_s and X_r, are internal to the digital system. On the rising edge of the measurement sample clock X_s, the measurement latch is sampled by the digital system. The rising edge of X_s must be sufficiently offset from the rising edge of X_p so that the output from the latch has settled

Signal	Meaning
X_p	The photonic cycle clock
X_s	The measurement sample clock
X_r	The latch reset clock
`ops_above`	Byproduct operator values from the logical qubit above
`ops_below`	Byproduct operator values from the logical qubit below
`P[15:0]`	The two-byte program word input
`m[2:0]`	Latched measurement inputs to `control system`
`m[2]`	Measurement outcome from logical qubit above
`m[1]`	Measurement outcome from current logical qubit
`m[0]`	Measurement outcome from logical qubit below
`s`	`control system` adaptive measurement setting output
`ops`	`control system` byproduct operator output
`ops[0]`	X byproduct operator bit
`ops[1]`	Y byproduct operator bit

Table 3.1: The table contains a summary of the notation used in Figures 3.1 and 3.3. When a signal `x` is a bus (a bold line in the figures), consisting of multiple parallel signals), the signals are indexed from 0 upwards, and the range is expressed using square brackets after the signal name. For example, `x[3:0]` is a bus of four signals. Subsets of the signals use the same notation (`x[2:1]` is signal 2 and 1), and a single signal is identified using one index (`x[0]` is signal 0).

to a steady state. This delay must include the time required to amplify the photon detector output.

On the rising edge of the reset clock X_r, the latch is returned to its initial state (untriggered) ready for the next measurement round. This event must occur after the rising edge of X_s, but before the rising edge of the next photon clock cycle X_p, to satisfy the hold time requirement of the sampling logic.

The computation of the adaptive measurement setting is performed using combinational logic at the earliest possible time that the latch output is valid, on the rising edge of X_s. The measurement setting for the next measurement is then computed and becomes available a short amount of time after the rising edge of X_s, corresponding to the combinational logic delay.

In addition, the byproduct operators are also computed on the rising edge of X_s using combinational logic. The commutation correction, which must be applied at the boundary of a quantum gate, is then computed on the rising edge of X_r, because it requires the value of the byproduct operators computed

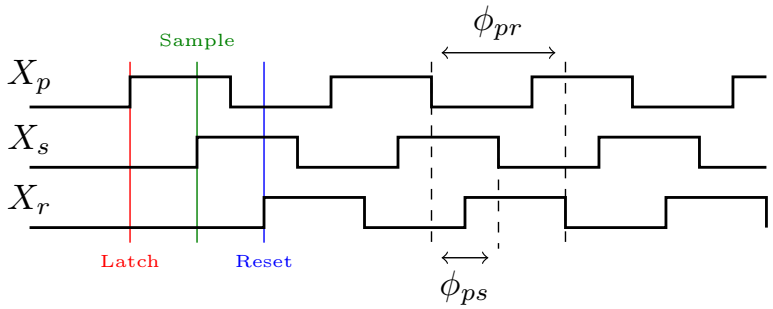

Figure 3.2: The diagram showing the clocks used in the FPGA design. The photon arrives on the rising edge of X_p, which triggers the latch (converting the pulse to a persistent level). The level is sampled on the rising edge of X_s, which represents the start of processing by the system. The latch is reset on the rising edge of X_r, which returns the latch to the reset state, ready for the next photon to arrive. The phase differences ϕ_{ps} and ϕ_{pr} are design parameters of the system. They may be adjusted to maximise the operational frequency, or provide as much timing slack as possible for the inputs and outputs.

on X_s. The program which controls the measurement pattern is loaded from memory on X_p so that it is ready for the computations that take place on X_s and X_r.

The design of each computational subsystem is described in detail below.

Key points

The overall system is made from copies of a basic digital unit cell, which latches measurement input pulses, provides a computational system that calculates adaptive measurement settings and byproduct operators, and stores a program that controls the calculations. The system is clocked using three out-of-phase clocks, in order that only one clock cycle of X_p is required per measurement round (of a column of the cluster state).

3.2 Computational system design

The computational system is the main subsystem that performs the computations relating to MBQC shown in Figures 2.2 and 2.3 of the previous chapter. The inputs to the system are measurement outputs (from the latch), current

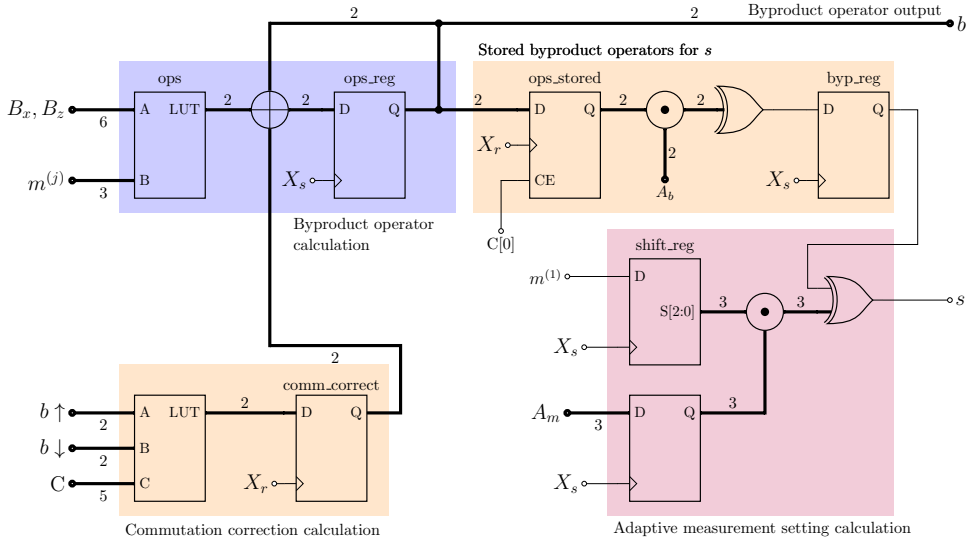

Figure 3.3: The design of the control system. In the high level schematic diagram of the control system, buses are denoted with bold lines, and the bus width is written next to the wire. The circles apply bitwise operations between their inputs: the cross stands for XOR and the dot stands for AND. The right port of the circle is the output, and all other ports are inputs. The logic gates are multi-input, with inputs from all the buses and wires connected on their left (i.e. wires inside a bus will be combined in the logic operation). Each part of the diagram is shaded according to its function, using the same colouring as in Figures 2.2 and 2.3. Flip-flops are clocked on the rising edge of their clock input, and elements whose output is `LUT` represent combinational logic. Reset signalling is omitted from the diagram for simplicity.

values of byproduct operators, and a program word stored in memory. This section describes how these calculations are performed, and how the program is defined.

3.2.1 Adaptive measurement setting generation

The most important feature of the adaptive measurement setting s is that it must be present as soon as possible, ready for the next measurement round. The earliest possible time that s can be computed is on the rising edge of X_s. From Figure 2.2, the value of s can depend on previous measurement settings and stored byproduct operator values from the current qubit.

A shift register is used to store the past three measurement values[1], m_0,

[1]For more complicated measurement patterns it may be necessary to store more than

m_1 and m_2, where m_0 is the most recent measurement outcome. The shift register is loaded sequentially with the next measurement on the rising edge of X_s. The output s is then obtained using a combinational circuit from the shift register, so it is present soon after the rising edge of X_s.

The outputs from the shift register are combined bitwise with a 3-bit mask A_m and XORed together to produce the measurement contribution to s. The stored byproduct operators (x_s, z_s) are masked using a two-bit value A_b and XORed to produce a second contribution to s. These two contributions are XORed to produce s itself. Putting together these two contributions gives the following expression for s:

$$s = \left(\bigoplus_{i=0}^{2} A_m[i] m_i \right) \oplus \left(A_b[1] x_s \oplus A_b[0] z_s \right), \tag{3.1}$$

where square brackets denote bitwise access.

The masks A_m and A_b for each measurement round are chosen in such a way that they combine past measurement outcomes and byproduct operators correctly to realise the one-qubit gate, as shown in Figure 2.2. The CNOT gate has no adaptive measurement settings, so $A_m = A_b = 0$ in that case.

The mask A_m must remain valid through the rising edge of X_p, so it is registered on the rising edge of X_s. The byproduct operator contribution due to A_b is also registered on X_s, so that the byproduct term persists through X_p. These registers are necessary because the program word P, which contains the masks (see Section 3.2.4 below), is updated on the rising edge X_p.

A disadvantage of this design is that the output s may contain function hazards [70], due to the propagation delays from each of the flip-flops to the output s. These hazards do not affect the digital function of the (synchronous) digital system; however, they may contribute to the power dissipation of the system and/or noise in the analog output, depending on how it is implemented. In order to avoid the hazards, the output s could be registered; however, this would require another clock edge soon after X_s to preserve the setup time of the analog output stage.

The adaptive system is shaded in purple in the control system schematic diagram shown in Figure 3.3.

three measurements. However, for the arbitrary one-qubit gate and CNOT gate, three measurements are sufficient.

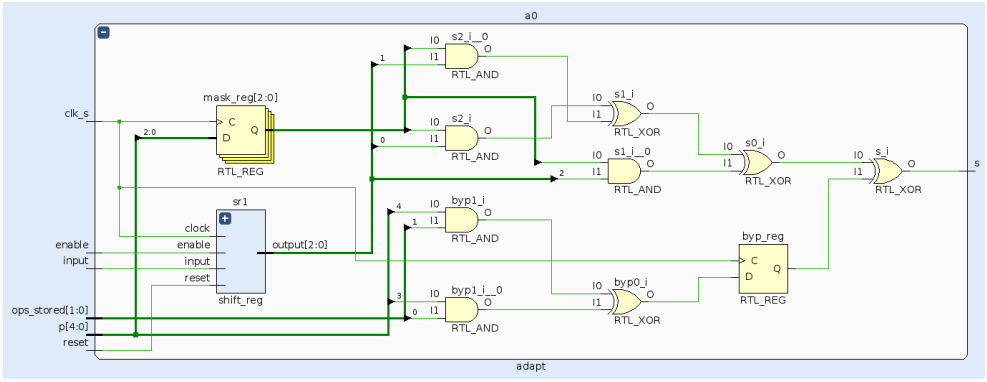

Figure 3.4: Elaborated design of the `adapt` entity, responsible for calculating the adaptive measurement settings. This corresponds to the adaptive measurement calculation, shown schematically in Figure 3.3. The `mask_reg` is used to store A_m, which is combined with the measurement outcomes `output` via the AND-XOR network at the top, representing the first term in Equation (3.1). The lower part of the diagram corresponds to the second term in Equation (3.1), and the stored byproduct operator contribution to s shown in Figure 3.3.

Source-code reference: mbqc-fpga

The file `design/mbqc.srcs/sources_1/new/adapt.vhd` contains the VHDL entity `adapt`, which implements the logic described above. The elaborated design (produced by Vivado) is shown in Figure 3.4.

3.2.2 Byproduct operator calculation

The byproduct operators must be updated after each measurement round. Since they only depend on the measurement outcomes, they can also be computed on the rising edge of X_s.

The byproduct operators comprise two bits (x, z), which are updated according to the measurement outcomes from the current logical qubit, $m_0^{(1)}$, and the two neighbouring logical qubits, $m_0^{(2)}$ above and $m_0^{(0)}$ below. Any of these three measurements may be XORed in any combination, together with the old byproduct operator values (x, z), to produce new (x', z'). Two three-bit masks B_x and B_z control which of the three measurements outcomes should be XORed together to produce the updated x and z, so that the byproduct

operators are obtained using the following equations:

$$x' = x \oplus \left(\bigoplus_{j=0}^{2} B_x[j] m_0^{(j)} \right) \tag{3.2}$$

$$z' = z \oplus \left(\bigoplus_{j=0}^{2} B_z[j] m_0^{(j)} \right). \tag{3.3}$$

The masks B_x and B_z for each measurement round are chosen in such a way that they combine measurement outcomes from the current and surrounding logical qubit rows to form the updates to the byproduct operators that are shown in Figures 2.2 and 2.3.

It is sometimes necessary to add a constant (the 1 in Equation (2.1) for z'_c) to the byproduct operators, as in the case of the CNOT pattern. This constant addition is controlled by the commutation correction program, as described in Section 3.2.3 below.

The main byproduct operator calculation is shaded blue in the top left of Figure 3.3.

Source-code reference: mbqc-fpga

The file `design/mbqc.srcs/sources_1/new/byproduct.vhd` contains the VHDL entity `byproduct`, which implements the logic for the byproduct operator calculation. The elaborated design is not shown here, because it does not fit on the page. The complexity is mainly due to the logic for the incorporated commutation correction, described in the next section.

3.2.3 Commutation corrections

For the CNOT gate, the commutation correction is performed by mixing the values of the byproduct operators between the control and target logical qubits, as described in Equation (2.5).

For an arbitrary one-qubit gate, the correction is more complicated, requiring the use of the byproduct operators in the calculation of the measurement settings. However, in order to avoid overwriting these correctional byproduct operators prematurely, it is necessary to store them in a separate register,

Bit	Meaning
0	If high then store the byproduct operators
1	If high then a commutation correction is necessary, in which case:
2	If high then current logical qubit is the control
3	If high then other qubit in CNOT is above
4	If high then add ones to the byproduct operators, in which case:
2	Constant value to add to z
3	Constant value to add to x

Table 3.2: The table contains the interpretation of the bit fields of C, which controls the commutation correction for the arbitrary one-qubit gate and CNOT gate, and also controls the addition of constants to the byproduct operator. The meaning of bits 2 and 3 depend on whether bits 1 or 4 are set, which are mutually exclusive. If $C = 0$ then no operation is performed.

called the stored byproduct operator register. The correction for the one-qubit gate then amounts to loading this register from the current byproduct operators.

Both these corrections, for the CNOT and the one-qubit gate, require the byproduct operator values and must therefore be calculated on the rising edge of X_r rather than X_s. The behaviour of this correction is controlled by a five-bit value C, whose interpretation is shown in Table 3.2.

Most of the time $C = 0$ and the commutation correction does nothing. It is only directly before gate boundaries that a commutation correction must be performed. The commutation corrections are shaded in orange in Figure 3.3.

Next steps

The most complicated logic in the design is contained within the commutation correction subsystem. This complexity is artificial, because it corresponds to gate boundaries (which are removed in the patterns discussed in Chapter 6). The design could likely be improved and simplified by incorporating a better scheme for byproduct operator calculation, based on the material discussed there. This would also entail reviewing the design of the program word, discussed in the next section.

		Qubit 0					Qubit 1				
Gate	A	m_0	P_0	θ_0	s_0	b_0	m_1	P_1	θ_1	s_1	b_1
U	0	0	0302	0	0	00	0	0002	0	0	00
	1	1	0510	-0.1	1	10	1	0010	0	0	10
	2	1	0342	-0.2	1	11	0	0002	0	0	10
	3	0	3010	-0.3	0	11	1	5010	0	0	00
CNOT	4	1	0003	0	0	10	0	0002	0	0	10
	5	0	0010	$\pi/2$	0	10	1	0030	0	0	00
	6	0	a013	$\pi/2$	0	10	0	0022	0	0	00
	7	1	0002	0	0	10	1	0010	0	0	10
	8	1	0012	$\pi/2$	0	01	0	0002	0	0	10
	9	1	0010	$\pi/2$	0	11	0	0010	0	0	10

Table 3.3: Example two-qubit computation comprising a one-qubit gate $U = R_x(0.3)R_z(0.2)R_x(0.1)$ on qubit 0, followed by a CNOT between qubits 0 and 1 (qubit 0 is the control). The program P_i (written in hexadecimal in the table) combines the measurement outcomes m_i (randomly generated) to produce the adaptive measurement setting s_i and the byproduct operators b_i (the least significant bit is z) for the i^{th} qubit. The basis measurement angles θ_i are included for completeness (s_i is combined with θ_i to produce the measurement angle ϕ_i).

3.2.4 Program word

The digital system is controlled using a 16-bit program word P which is formed by concatenating the masks and control bits in the previous sections as follows:

$$P = CA_bA_mB_xB_z. \tag{3.4}$$

Each logical qubit requires its own set of program words, one per measurement round.

Table 3.3 shows an example calculation for the two qubit circuit containing an arbitrary one-qubit gate $U = R_x(0.3)R_z(0.2)R_x(0.1)$ on the first qubit, followed by a CNOT gate between the first and second qubit. The table contains randomly chosen measurement outcomes, and the adaptive measurement settings and byproduct operators that result from the measurement pattern, including the program word that is used to make the calculations.

It is clear that the program word could be compressed to save on memory usage. In our example design, we have prioritised program simplicity over memory usage.

Key points

The computational system implements the logic required to realise the one-qubit gate and CNOT measurement patterns. The adaptive measurement settings arise predominantly from the one-qubit gate, and lead to a simple implementation involving a shift register to store measurement outcomes (the `adapt` entity). On the other hand, the CNOT gate entails more complicated byproduct operator calculations (the `byproduct` entity), and commutation corrections that must be stored in the program word. These aspects of the program could likely be simplified and optimised, by considering better measurement patterns, or refactoring the byproduct operator calculations.

3.3 FPGA implementation of the design

The previous section laid out a control system design, written in VHDL, targeting a 7-series Xilinx FPGA device. In this section, we elaborate on several technical details of the design.

We used the mixed-mode clock manager (MMCM) [71] to generate the two out-of-phase clocks X_s and X_r from the (external) system clock X_p, in order to fit the computation inside a single clock cycle. The use of out-of-phase clocks in a digital design complicates timing closure (the requirement to meet specification-imposed timing constraints) [25, 72].

Next steps

As we show in Section 3.4, the design choice to use out-of-phase clocks does not pay off, because the maximum frequency achievable by the system is quite low ($\sim 150\,\text{MHz}$), which is much lower than the maximum supported frequency of the FPGA (710.00 MHz). An improved design might be obtained by removing the out-of-phase clocks, and refactoring the design to split the calculations over multiple clock cycles.

The program word is stored in memory generated by an instance of the distributed memory generator IP [73], configured as ROM. This enabled stor-

age of the program in a coefficients file for the purpose of verifying the design (the subject of Chapter 4).

The utilisation of logic and input/output (I/O) pads in the design is provided for one logical qubit and 20 logical qubits in Table 3.4. The data was obtained from the utilisation report generated by Vivado after implementing the system for each number of logical qubits. The number of logic elements scales more than linearly between one and 20 logical qubits because the synthesis tool optimises away logical qubit interconnects in the single logical qubit case. However, the overall utilisation of flip-flops and look-up tables in the design is very low ($< 1\,\%$ of device resources), because the calculations involved in the design are quite simple.

The use of I/O pads is quite high, due to the need for one measurement input m, one adaptive measurement setting s and two byproduct operator lines per logical qubit. In our design, the total number of I/O pads required is

$$K = 4N + 4,$$

where N is the number of logical qubits. This includes four common signals: the input clock X_p; the clock-is-locked output signal from the MMCM; a reset signal; and an enable signal. By accessing the byproduct operators via a low speed serial interface, it would be possible to reduce this pin count to

$$K \sim 2N,$$

which includes only the measurement inputs m and adaptive measurement setting outputs s. On the largest FPGA in the 7-series family [66], the Virtex-7 xc7v2000t device (which has 1200 user I/O pads), this provides an upper bound on the number of logical qubits (cluster-state rows) of $N \sim 600$.

I/O delays are also a bottleneck for performance in the FPGA design, as we show in Section 3.4. The Xilinx 7-series devices were chosen because they have a level-sensitive latch built into their input logic slice (LDCE) [68], which forms the first stage of the digital system.

A disadvantage of the design is that it is not possible to place the output s in the output logic slice, because there is combinational logic between the final register and the output port [68]. It is also not possible to place the byproduct operator registers in output logic slices, because the output is rerouted to the internal FPGA fabric for use in updating the byproduct operators (see the feedback loop in Figure 3.3).

	Flip-flops			Look-up tables			I/O	
N	CS	Full	Util.	CS	Full	Util.	Full	Util.
1	10	24	0.03 %	5	11	0.03 %	8	2.8 %
20	237	476	0.89 %	137	364	0.58 %	84	29.5 %

Table 3.4: Utilisation of flip-flops, look-up-tables and I/O pads in the design, for $N = 1$ logical qubit and $N = 20$ logical qubits, for the computational system (CS) in Figure 3.3 and the full design in Figure 3.1. The proportion of device resources is included in the utilisation (Util.) columns.

As we show in Section 3.4, the clock frequency is not a bottleneck in the system, so it may be possible to create another design with multi-cycle latency, where the outputs are stored in separate registers and eligible for placing in the output logic slice. This may remove some of the output delay and allow a slightly higher clock frequency. It would also remove the logic hazards present in the output s.

Before performing timing analysis of the design, we performed functional verification of the design, in order to establish that the logic described above is correct. This is described fully in Chapter 4.

Key points

The use of out-of-phase clocks is only viable because the device contains a clock manager that can generate these clocks. The design does not make significant use of the logic slices in the FPGA, because the algorithms are very simple (in comparison to typical embedded applications for FPGAs). However, a large number of inputs and outputs are used, due to the high number of measurement inputs, adaptive measurement settings, and byproduct operators.

Next steps

There is no reason to route byproduct operators to FPGA outputs, because they are not required by the measurement block (see Figure 2.4). It may be possible to simplify the system and improve its timing behaviour by removing these outputs.

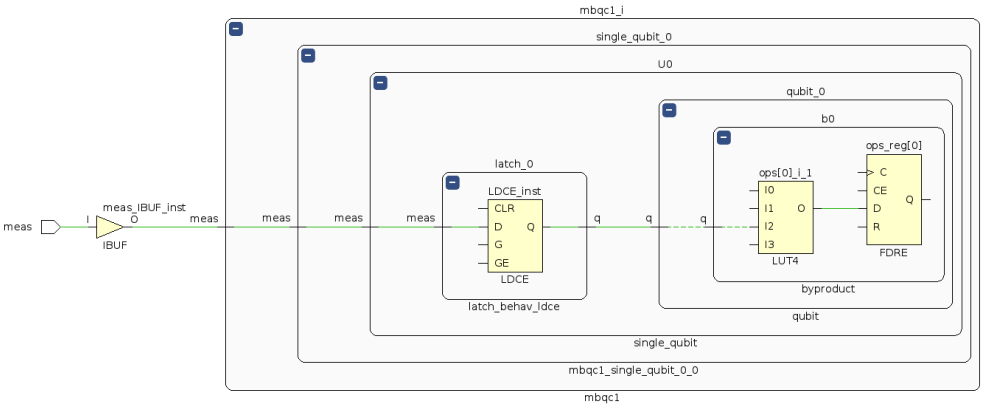

Figure 3.5: The diagram shows the critical path on the input side of the design, which is the path through the logic which creates the timing bottleneck between clocks X_p to X_s. The measurement pulse at the FPGA input m coincides with the rising edge of X_p, and this signal must propagate though the latch to the byproduct operator system before the signal is sampled on the rising edge of X_s.

3.4 Timing analysis

We used static timing analysis, performed automatically in Vivado, to establish the maximum operating frequency of the design and to obtain the I/O delays associated with the system. The critical timing path is made up of two components:

- **Input path** from the input port m (clocked on the rising edge of X_p) to the byproduct operator register (loaded on the rising edge of X_s). This path is shown in Figure 3.5.
- **Output path** from the shift register output (loaded on the rising edge of X_s) to the output port s (clocked on the rising edge of X_p). This path is shown in Figure 3.6.

By modifying the phase shift of X_s relative to X_p (ϕ_{ps} in Figure 3.2), it is possible to allocate more time to one path or the other. The phase of X_r (ϕ_{pr} in Figure 3.2) must also be adjusted to allow timing closure of paths between the X_s and X_r clock domains. We established the maximum operating frequency $F_{\max}$ of the system by manually adjusting the phase of X_s and X_r to balance the worst negative setup slack between the critical paths,

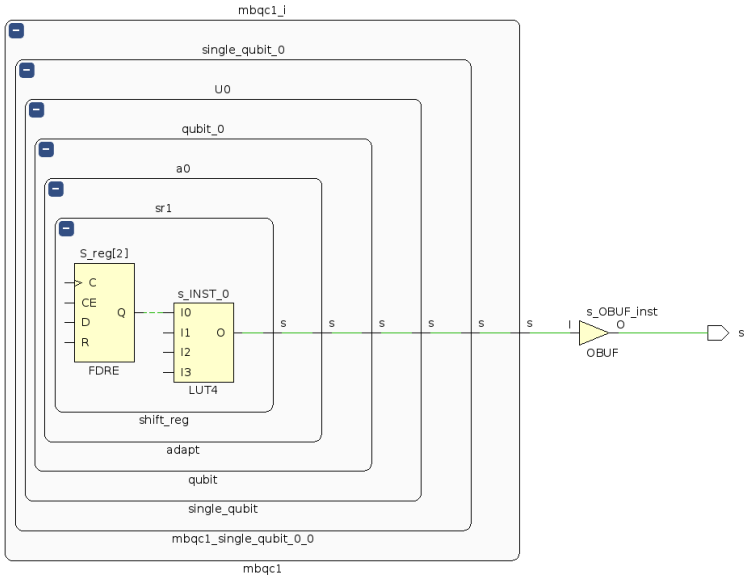

Figure 3.6: On the output side of the design, the critical path begins on the output from the shift register (clocked on the rising edge of X_s) and ends on the FPGA output s (the adaptive measurement setting). This output must be ready for the next photon to arrive at the next rising edge of X_p. The setup time for this output is the total time available for the analog output system to prepare the phase shifts on modulators M_1 and M_2 (see Figure 2.4) before the next photon arrives.

while increasing the frequency of the design, until both paths fail to meet timing. Using this method, we obtained $F_{\max} = 190\,\text{MHz}$ using $\phi_{ps} = 220°$ and $\phi_{pr} = 300°$. The phase difference 80° between X_s and X_r represents the amount of the time taken for the internal FPGA logic to process the latched measurement outcome before it is reset.

We then performed the timing analysis at each frequency between 10 MHz and 190 MHz, in steps of 10 MHz, to establish the most generous input and output constraints that still allow timing closure at each frequency. All I/O constraints are expressed with respect to the external clock X_p (the system clock).

The input constraint is specified by the clock-to-out time t_{co} of the input signal m, which is equal to the time delay between the rising edge of X_p and the pulse generated by the input analog system at m. This time constrains the analog characteristics of the single-photon detector amplifier.

The output constraint is the setup time t_{su} of the output signal s with

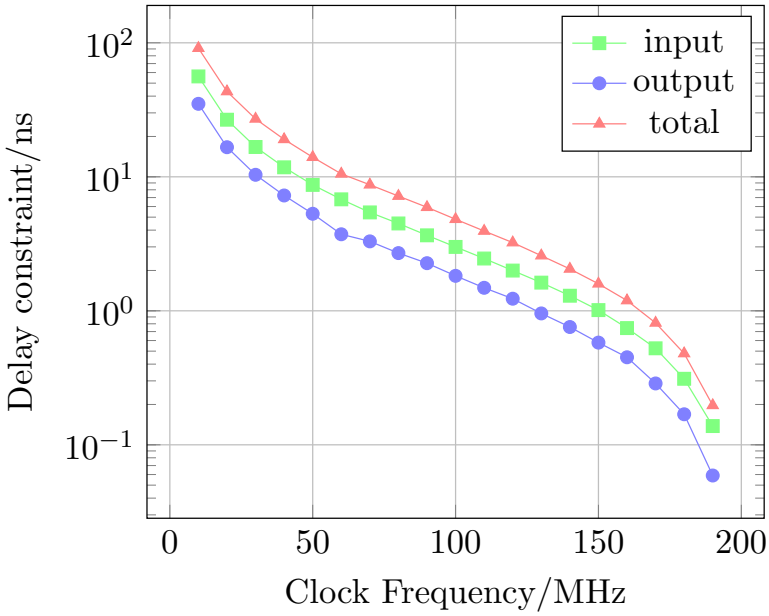

Figure 3.7: The most generous input and output delay constraints that allow implementation of the design at each frequency. The total delay, which can be apportioned between input and output analog systems by adjusting the phase of X_s, represents the maximum amount of time available to the analog system shown in Figure 2.4.

respect to the system clock X_p, which is the delay between the time that s transitions at the boundary of the FPGA and the next rising edge of X_p. This time determines the required operating speed of the output DAC system and modulator drivers, which must be able to set the voltages of the modulators before the next photon arrives on the rising edge of X_p.

The I/O timing constraints are plotted as a function of frequency in Figure 3.7. The input constraint is systematically more generous than the output constraint, because of the choice of phase of X_s. The sum of the input and output constraints must be less than the total I/O slack, also shown in the figure.

Figure 3.8 shows a graph of the proportion of the clock cycle X_p taken up with digital processing, as a function of frequency. It is clear that at higher frequencies, the digital processing dominates the clock cycle, leaving very little time for the analog amplifier systems.

At a representative clock frequency of 150 MHz, the photons would need to be delayed for 6.67 ns in either an optical fibre or a waveguide delay line.

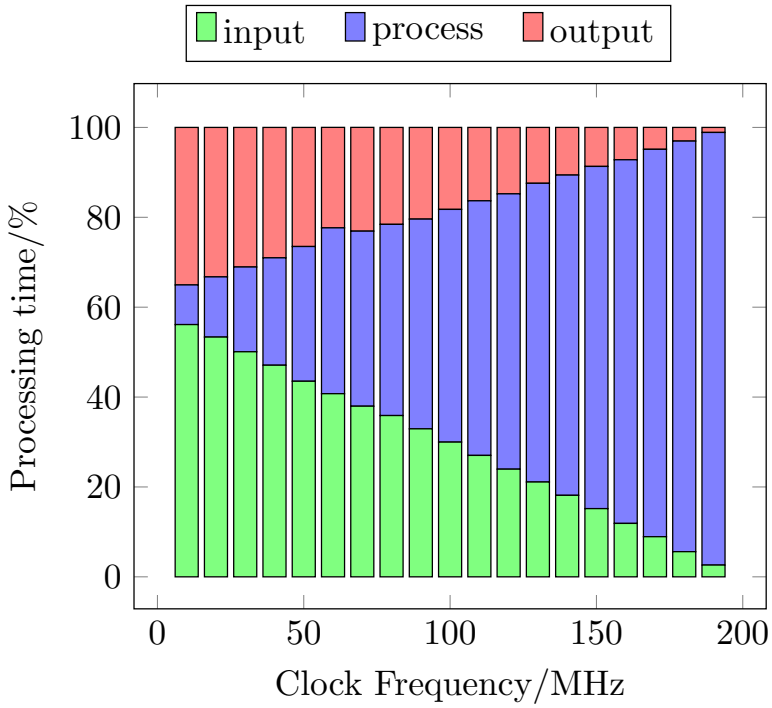

Figure 3.8: The proportion of the clock cycle devoted to processing the adaptive measurement settings and the byproduct operators, as a function of photon clock frequency. At the higher frequencies, nearly all of the cycle is spent processing the measurements, leaving almost no time for the analog amplification at the input and output (shown in green and red).

Assuming a standard silicon-on-insulator (SOI) platform, the delay line must be approximately 83 cm, assuming a mode index of ~ 2.4 [74].

Key points

By adjusting the phases ϕ_{ps} and ϕ_{pr} of the out-of-phase clocks, it is possible to maximise the operating frequency of the FPGA design, up to a maximum of $X_p = 190\,\text{MHz}$. As the frequency is increased, the proportion of this clock cycle devoted to digital processing increases, leaving less time for the analog systems at the input and output of the control system, as shown in Figure 3.8. This imposes a timing constraint on the design of the analog and photonic systems, if the control system is realised in the FPGA considered here.

We re-implemented the design targeting a higher end FPGA (Xilinx Kintex Ultrascale+, part no. xcku5p-ffvd900-3-e), to see whether the maximum clock frequency could be improved. We found that the maximum clock frequency increased to $F_{\max} = 220\,\text{MHz}$ using $\phi_{ps} = 140°$ and $\phi_{pr} = 230°$. In this case, at the maximum clock frequency, less time is allocated to the input analog system compared to the 7-series FPGA. The phase difference of 90° between X_s and X_r indicates that approximately the same time (1.125 ns) is taken by the internal digital system compared to the 7-series FPGA (1.152 ns).

3.5 Discussion of extensions to the design

There are several improvements that could be made to the simple digital system presented in this chapter. It is likely that some performance improvement could be obtained by implementing the digital design using an ASIC, due to the average reduction in critical path delays [65]. However, this may not translate to a performance improvement in this design because the majority of the critical path delays come from the I/O buffers, not the logic. To improve this, it may be possible to utilise very high speed latches and output buffer designs, with delays on the order of 100 ps [75]. A full analysis of the I/O buffer delays should be performed in tandem with the design of the I/O analog systems, to ensure compatibility between the two systems. At this point, the requirement for absolute synchronisation between the cluster-state generator and the digital control system, using a system synchronous architecture [76] may become the bottleneck to the design. Such schemes are often limited to speeds up to 200 MHz–300 MHz, due to clock skew and data path delays [26].

The design could be extended to support a greater degree of non-locality in the byproduct operator calculation. In the design discussed here, the byproduct operators depend only on measurement outcomes from adjacent logical qubits. However, there are measurement patterns for which byproduct operators for a given logical qubit may depend on cluster qubits that are further away [20, Section IV.C.]. This may lead to a routing problem in FPGA and ASIC designs, especially as the number of qubits increases, which are important to quantify.

The design presented here shows that, even in the ideal (unrealistic) scenario where a deterministic cluster-state generator is available, the design of a control system that does not impose significant timing constraints may present

a significant challenge. We have used an FPGA for prototyping the design, which favours rapid prototyping of the design over thorough optimisation of the logic. On the other hand, the design presented here could only reasonably be expected to operate at 150 MHz, more than a factor of six less than the target photonic clock rate of 1 ns highlighted in Section 2.2.2. At 150 MHz, the total time available for the input and output analog processing is 1.59 ns, out of the total period 6.67 ns. The remaining 5.08 ns is consumed by logic delays inside the FPGA design. At the same time, a photon clock period of 6.67 ns corresponds to a long delay line ($\sim$ 83 cm), that will occupy quite a large footprint in an integrated implementation of photonic MBQC. A certain amount of work is necessary to show that these timing constraints can be managed and accounted for in a better electronic control system design.

Key points

The design presented here cannot operate much faster than 150 MHz. This is despite implementing algorithms that only involve fixed, small amounts of arithmetic, and target a high-performance FPGA with a maximum clock frequency of 710.00 MHz. The problem is the simultaneous requirement for low-latency in the adaptive measurement setting calculation, and high-throughput of measurement outcomes – conflicting requirements in digital system design. We argue in this thesis that this highlights the necessity to investigate electronic control systems for photonic quantum computing, as an optimisation problem in its own right.

Next steps

Although some improvements may be made by using custom ASICs, a better approach would be to extend the analysis to the analog parts of the control system. This would make it possible to address the main component of the critical path delays, due to the I/O buffers. This investigation could be performed in a high-speed mixed-signal process, using the control system design presented in this chapter as a basis, to assess what timing constraints arise from the analog/digital interface.

3.6 Summary

We have presented a simple, concrete, control system design, for a model of photonic quantum computing that uses a deterministic cluster state. Even though this model of photonic quantum computing is not possible to realise in practice, it forms a useful entry-level model in which to explore control system implementations. However, even in this simple case, the control system design is not a trivial exercise, because of the simultaneous high-throughput and low-latency requirements of the photonic system. This control system is freely available, and may be used as a practical basis for learning about control systems for photonic quantum computing, or optimising the designs for custom hardware. We extend the investigation of control system design to the case of non-deterministic cluster states in Chapter 5.

Before beginning there, we address in detail (in Chapter 4) how the system described in this chapter was verified for correctness. This functional verification is very important in any digital design, where the resulting system (for example, Figure 3.3) is complicated enough that it may contain logic errors. In the case of photonic MBQC, it is also important to bring the measurement patterns themselves within the scope of the verification, so that there is some assurance that the overall quantum computer based on the control system would realise the correct quantum gates. In the development of the verification of measurement patterns, we develop a quantum simulator for MBQC, which is re-used in Chapter 6 for the investigation of voltage noise on the operation of the control systems models discussed in Chapter 5.

Chapter 4

Design simulation and verification

In any complicated digital system, functional verification is very important, to ensure that hard-to-find bugs are not present in the synthesised design. However, in the case of MBQC, where quantum gates are performed using non-intuitive measurement patterns, it is also important to check that the measurement patterns themselves do not contain errors. This is less important for well established patterns such as the arbitrary one-qubit gate [20]. However, non-standard patterns, for example the reduced CNOT gate derived in Appendix A.1, may contain errors (e.g. resulting from mistakes in the derivation) and must therefore be checked for correctness[1].

In the previous chapter, we presented an FPGA design for the digital part of the control system for photonic MBQC, written in VHDL. However, the design may contain errors, due to simple (typographical) mistakes in the source code, or more fundamental logic errors in its design. One established method for reducing the chance of these type of errors is to write VHDL testbenches – source files that probe the digital design in a simulated environment, and verify that the outputs from the digital system are correct [77]. This type of verification is fully supported by Vivado as part of the recommended design flow [78].

In this chapter, we present a verification system which consists of the following components:

[1] In an earlier iteration of the hardware design, the reduced CNOT pattern did contain an error which was discovered using the methods presented in this chapter.

- **MBQC simulator**. This is used for verifying measurement patterns and outputting data that can be used as reliable inputs/outputs for the digital control system verification.
- **VHDL testbenches**. These use the simulation data to verify that the digital control system produces the correct outputs (adaptive measurement settings and byproduct operators) for a given set of inputs (random measurement outcomes and program words corresponding to the quantum circuit).

Section 4.1 describes the design and verification of the MBQC simulator, including the underlying resizeable quantum computing simulator for the simulation of cluster states and measurement patterns. The use of the output data from this program in VHDL testbenches is described in Section 4.2.

The simulation program and testbenches are available in the digital system design repository `mbqc-fpga` [64].

Key points

Hardware verification of the design described in Chapter 3 is necessary to attempt to eliminate mistakes in the design. Three steps are required for the verification: checking that the measurement patterns themselves are correct; generating valid input and output data based on the measurement-pattern simulation; and using this data to probe the digital system in a simulated environment. This chapter describes how these steps are performed.

4.1 Measurement pattern verification

An important component of the verification system is an MBQC simulator, written is C++, which performs two operations:

- Verification that a given measurement pattern performs the correct quantum operation.
- Generation of valid input and output data in a format that can be used in VHDL testbenches for the functional verification of the digital system described in Chapter 3.

The design of the MBQC simulation program is quite specific to the hardware design: the subsystems described in Section 3.2 correspond quite closely to subsystems in the C++ program. The advantage of this approach is that writing and debugging the VHDL testbenches is easier, because the internal state of the verification program corresponds more closely to the internal state of the digital design. However, as a result, the program cannot be easily modified to cover MBQC simulations in other contexts.

The core of the program is the quantum simulator and cluster-state simulator described in Sections 4.1.2 and 4.1.1. The main novelty of this simulator is the efficient mapping of a cluster state of bounded height (≤ 14) and arbitrary width onto a simulation containing a finite number of qubits, which can be simulated on an ordinary laptop.

Source-code reference: mbqc-fpga

The C++ code for the simulator is contained in the folder `simulator/src/`. The files `cluster.hpp` and `cluster.cpp` contain the cluster-state simulator used for verifying measurement patterns. This simulator is based on QSL, which is discussed in more detail in the sections below. Most other files are highly specific to the FPGA design under discussion here.

4.1.1 Cluster-state simulation by recycling the state vector

Direct simulation (simulation of the entire state vector) of a quantum computer is a memory intensive task. For M qubits, the state vector (containing 2^M complex amplitudes) has length $2k2^M$ bytes, where k is the number of bytes required to store a floating point real number. For the purposes of the simulator discussed here, we assume that each floating point value is stored using double precision, where $k = 8$. Therefore, $M = 28$ qubits (using 4 GiB) is the practical upper limit to the number of qubits that can be simulated on a computer with 8 GiB of main memory.

It is therefore clearly not feasible to simulate a cluster-state computation by simulating all the cluster qubits directly. The number of cluster qubits required for the simple circuit shown in Figure 2.1 is 27, which is one qubit short of the maximum. Adding another logical qubit (a total of four logical qubits) would increase the number of cluster qubits to 36, beyond what is possible

to simulate with a normal laptop. Furthermore, the number of cluster qubits depends on the length of the circuit, which is an undesirable feature; even a circuit containing one logical qubit could contain at most seven concatenated arbitrary one-qubit gates before it is too large to simulate.

Instead, it is possible to simulate a cluster-state computation one column at a time. The justification of this method is the same as the method of gate verification [20]. Specifically, there is no difference between generating the cluster state all at once at the start of a computation, and generating the cluster state "patch-by-patch" (interleaving the entangling of a new patch of the cluster state, with performing the measurement pattern on that patch).

As a result, it is only necessary to store two columns of cluster qubits in memory at any one time[2]. Therefore, a quantum simulation of $2N$ cluster qubits (where N is the number of logical qubit rows) is sufficient to simulate an MBQC-based computation of height N and arbitrary width. Using this method, it is possible to simulate arbitrarily long MBQC circuits containing up to 14 logical qubits on a computer with 8 GiB of memory. The state vector is recycled in the sense that the same set of simulated qubits is used to simulate each column of the measurement pattern one by one.

The steps that are required to perform an MBQC simulation of N logical qubits using this method are as follows. The process is shown in Figure 4.1 for $N = 5$.

1. **Begin with the left-most column.** Create a state vector of N qubits, all in the $|0\rangle$ state, representing the left-most column of cluster qubits in Figure 2.1b. Prepare it in the equal-superposition state by applying Hadamard gates to all the qubits.

2. **Add a new column of qubits to the right,** by extending the state vector with N qubits (all in the state $|0\rangle$), resulting in a new state vector of $2N$ qubits. These new qubits represent the column immediately to the right of the current column in the state vector.

3. **Generate the cluster state in the new column,** by applying Hadamard gates to the new qubits, and apply CZ gates anywhere that an entanglement link is desired between the two columns.

[2]It is not possible to store just one column; in that case, there is no way to generate entanglement between columns, which is necessary for MBQC.

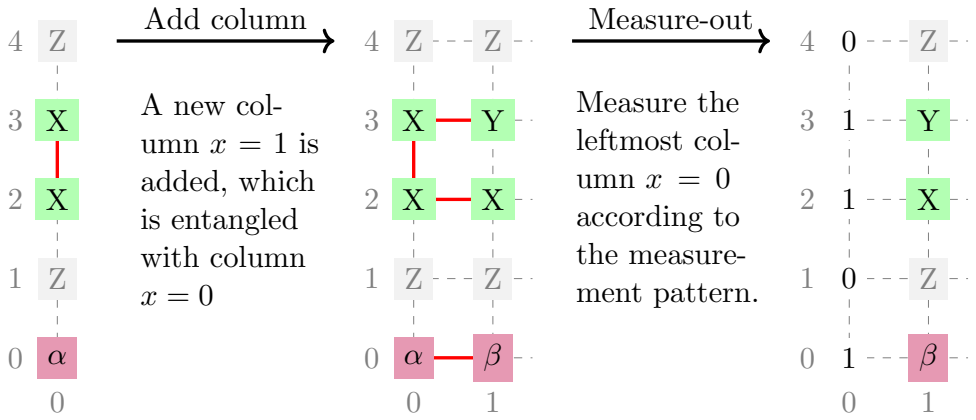

Figure 4.1: Diagram showing how column-wise simulation of measurement patterns is performed, for the case $N = 5$. Each column is entangled with the column to the right, before being measured out according to the measurement pattern. In the example, the CNOT pattern (see Figure 2.3), starting from the column with the vertical entanglement link, is simulated in rows 2 and 3. An arbitrary one-qubit gate is laid out along row 0. Any qubits not involved in a measurement pattern are measured in the Z-basis and the results discarded. In an implementation of this scheme, both columns are stored in the same state vector, which alternates in size between 2^N and 2^{2N}.

4. **Measure out the left-most of the two columns,** according to the measurement pattern given by the quantum algorithm. Store the measurement outcomes for processing into adaptive measurement settings and byproduct operators. After this step, the state vector is reduced to a single column, which now becomes the left-most column. If there are any more columns to the right in the measurement pattern, go back to step 2.

5. **Output state.** If there are no more columns to add, then the current state vector of N qubits contains the result of the quantum computation (up to byproduct operators).

The algorithm above consists of the repeated process of appending a new column of qubits to the right, measuring out qubits to the left, and shifting the remaining column to the left-most position. A description of an efficient program capable of performing these operations is described in the next section.

4.1.2 A resizeable quantum computer simulator

A quantum simulator that is capable of performing the algorithm described in the previous section must have the following features:

- **Ability to add qubits.** Adding a new column of photons to the right in Figure 4.1 is achieved by adding qubits one at a time to the state vector.
- **Ability to measure-and-remove qubits.** This is required when the left-most column is measured according to the measurement pattern.
- **Ability to perform one- and two-qubit gates.** CZ gates are necessary to add entanglement links, and one-qubit gates are required to set the bases for the measurement pattern.

In addition, the simulator must be efficient enough that the simulations finish in a timely fashion.

The special-purpose resizeable simulator in QSL [79] was designed to satisfy these criteria. The efficient implementation of gates and measurement in QSL is described in [80, Chapter 6]. Here, we focus specifically on the implementation of the addition and removal of qubits from the state vector.

Key points

When the scheme shown in Figure 4.1 is implemented, the state vector may alternate between two very different sizes; for example, when $N = 5$, one cluster-state column contains 32 amplitudes, and two columns contains 1024 amplitudes. Care is required to implement this system in a manner that limits the number of expensive memory-related operations that may be involved, as we discuss in the sections below.

Source-code reference: QSL

The relevant source code for the resizeable quantum simulation, required for the implementation of the scheme shown in Figure 4.1, is contained in the folder `src/qubits/resize`. In particular, the file `measure.cpp` contains the implementation of the efficient measurement-and-removal of qubits, described in the following sections.

Efficient measurement and removal of qubits from a state vector

Consider a state vector $|\psi\rangle$ for the simulation of N qubits, given by

$$|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{2^N-1} \end{bmatrix}. \tag{4.1}$$

If the index i of a given amplitude a_i is represented in binary as

$$x_{N-1}x_{N-2}\ldots x_1x_0, \qquad x_k \in \{0,1\}, \tag{4.2}$$

then that amplitude corresponds to the state of the system where qubit k is in the computational basis state $|x_k\rangle$. This uses a little-endian convention (where lower-index qubits correspond to less significant bit positions). From now on, the k^{th} bit of an index i will be written using square brackets $i[k]$. The set of valid indices in the state vector will be written $I = \{0, 1, 2, \ldots, 2^N - 1\}$.

To measure and remove a qubit from the state vector, it is first necessary to measure it (without removing it) according to the methods described in [80, Chapter 6]. Briefly, a random measurement outcome is chosen based on the probability p of measuring zero on that qubit, which is obtained by summing the squared amplitudes a_i where the k^{th} bit of i is zero:

$$p = \sum_{\substack{i \in I \\ i[k]=0}} |a_i|^2 . \tag{4.3}$$

After obtaining the random measurement outcome $z \in \{0, 1\}$, all the amplitudes a_i of the state vector where $i[k] \neq z$ are set to zero, and the state vector is renormalised.

In order to then remove the qubit k from the state vector, all that is required is to delete the amplitudes a_i from the state vector where $i[k] = 0$. The amplitudes are completely removed, in the sense that amplitudes a_{i-1} and a_{i+1} surrounding a removed amplitude a_i will be adjacent in the new state vector. The state vector will halve in size as a result of this operation (corresponding to the difference between 2^N and 2^{N-1}).

Assuming that the state vector is stored as a `std::vector` in C++, a naïve implementation of the procedure above would involve calling `std::erase` repeatedly to remove amplitudes from the vector. However, each time `std::erase`

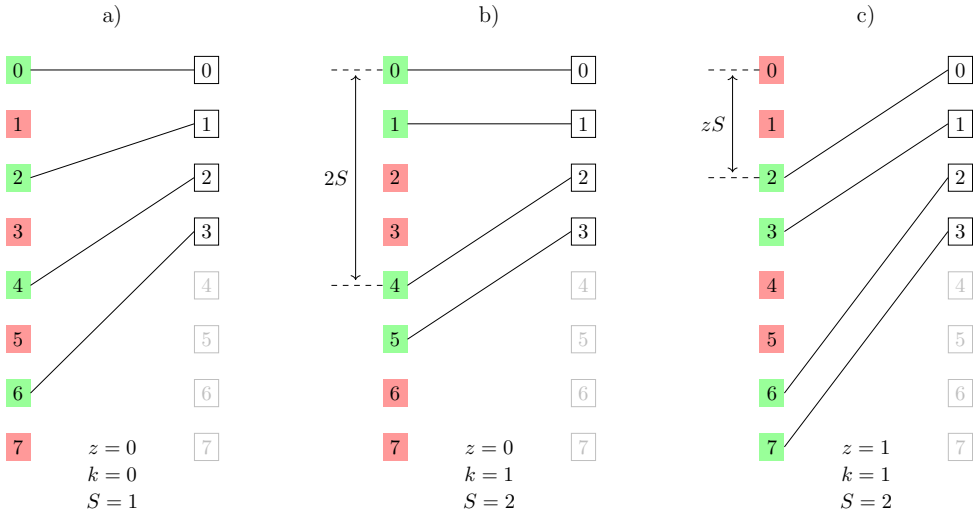

Figure 4.2: The method for removing an arbitrary qubit k in one pass of the state vector, for different values of k and outcome z. The distance $S = 2^k$ is the stride length, which is the size of the (green) blocks that must be copied. The red blocks are discarded (these are the amplitudes that are deleted). If the outcome of the measurement of the qubit is one, then the first block to be copied starts at offset S, rather than 0. The result of the copying operation is that only half of the state vector is used (the other half is greyed out).

Algorithm 1 Removal of qubit k from the state vector

```
for n ← 0 to 2^{N−k−1} − 1 do          ▷ Loop over all the blocks
    for m ← 0 to S − 1 do               ▷ Loop inside a green block
        i ← 2Sn + Sz + m                ▷ Source index
        j ← Sn + m                      ▷ Destination index
        a_j ← a_i                       ▷ Copy the amplitude a_i to the index j
    end for
end for
```

is called on the index i, all the elements of the vector at indices $j > i$ must be copied so as to maintain the contiguous storage of the vector required by the C++ standard [81]. In addition, the resizing of the vector may trigger memory reallocation, especially when the vector is increased in size with the addition of new qubits, outlined in the next section. Using `std::erase` to remove qubits was found to limit the overall MBQC simulator to $N = 6$ logical qubits, due to the prohibitive amount of time spent removing and adding qubits to the state vector.

A much more efficient scheme is to perform the required deletion operations

manually, by copying the amplitudes that are retained to their new (lower-index) positions in the state vector, and leaving the state vector the same size as before[3]. Figure 4.2 shows how this copying operation can be done using one pass of the state vector. The algorithm is summarised in Algorithm 1.

The important feature of this algorithm is that the state vector is traversed from low to high indices (top to bottom in the diagram), and each amplitude is only read or written once. In addition, higher-index amplitudes end up overwriting lower index positions, so that the end result is a state vector whose lower half is correct, and whose higher half contains random values. These random values can simply be ignored by the program. For a cluster state of height N, only 2^{N-1} memory reads and 2^{N-1} memory writes are required to remove one qubit.

Adding a qubit to the state vector

To add a qubit to the state vector, the procedure described in Algorithm 1 is essentially reversed, with some modifications; the goal of this algorithm is to insert zeros into the correct locations in the state vector. This operation corresponds to taking the tensor product of the state with a single qubit in the $|0\rangle$ state, in the desired index position. This may be interpreted as copying amplitudes from the right to the left in Figure 4.2, instead of left to right.

Firstly, it is necessary to loop over the blocks of green (destination) indices backwards, otherwise valid data at higher indices in the state vector would be overwritten by data that is copied from lower indices. For example, in Figure 4.2c, copying amplitudes 0 and 1 would overwrite amplitudes 2 and 3, unless those amplitudes are copied first. Secondly, it is possible to always set $z = 0$, which sets the new qubit in the $|0\rangle$ state. It is possible to obtain any other state after inserting the qubit by applying a one-qubit gate to the new qubit. Finally, it is necessary to set the red indices to zero – these are the amplitudes corresponding to the $|1\rangle$ state of the newly added qubit ($z = 1$).

The procedure for adding a new qubit is shown in Algorithm 2. It corresponds to the copying operations shown in Figure 4.2, with the modification that amplitudes are copied from left to right instead of right to left.

[3] In terms of free-store-allocated memory; the logical size of the state vector has still halved, because the program simply ignores the higher half of the vector.

Algorithm 2 Adding a qubit of state $|0\rangle$ at position k to the state vector

```
for n ← 2^{N−k−1} − 1 down to 0 do          ▷ Loop over all the blocks
    for m ← 0 to S − 1 do                    ▷ Loop inside a green block
        i ← Sn + m                           ▷ Source index
        j ← 2Sn + m                          ▷ Destination index
        a_j ← a_i                            ▷ Copy the amplitude a_i to the index j
        j ← 2Sn + S + m                      ▷ Index to zero-out (corresponds to z = 1)
        a_j ← 0                              ▷ Set the |1⟩-amplitudes to zero
    end for
end for
```

Key points

By using Algorithms 1 and 2 the transition between the one-column state vector and the two-column state vector is achieved in one pass of the state vector, without needing to write to any amplitude in the state vector more than once. This is a substantial improvement based on a naïve use of C++ standard library functions, and makes it possible to use the algorithm with $N = 14$ on a regular laptop.

4.1.3 Performing measurement patterns

The measurement pattern consists of two parts:

- The pattern of entanglement between cluster qubits. Logical qubit rows are always entangled in simple patterns, but vertical entanglement is only necessary when a two-qubit gate is implemented.
- What bases to use for cluster qubit measurements, and how to process the measurement outcomes.

The first of these requirements is achieved by applying CZ gates in the correct locations when a new column of qubits is added to the state vector in step 3 of the main recycling algorithm in Section 4.1.1. More specifically, a CZ gate is applied between qubits i and $i + N$, due to the column-major ordering of the qubits in the columns shown in Figure 4.1. If a vertical link is required, a CZ gate is applied between qubits i and $i + 1$, where $i \geq N$ (i.e. in the newly added column).

Algorithm 3 Processing measurement outcomes from the n^{th} round into adaptive measurement settings and byproduct operators for the k^{th} logical qubit. The variable M stores all the past measurement outcomes from all logical qubit rows. The program P controls how the byproduct operators, adaptive measurement settings, and commutation corrections are computed.

procedure STOREOUTCOME(n,k,M,P)
  $b[n,k] \leftarrow b[n-1,k] \oplus$ BYPUPDATE(n,k,M,P)
  $b_s[n,k] \leftarrow b_s[n-1,k]$ ▷ By default, b_s does not change
  $s[n] \leftarrow$ ADAPTIVESETTING(n,k,M,P,b_s)
  if STOREBYPRODUCT$(P[n,k])$ **then**
    $b_s[n,k] \leftarrow b[n,k]$ ▷ Store byproduct operators for one-qubit gates
  end if
  $b[n,k] \leftarrow b[n,k] \oplus$ CNOTCOMMCORRECT(n,k,P,b)
end procedure

The main processing step required by the scheme described in Section 4.1.1 is the measurement of the left-most column in appropriate bases, and calculations of adaptive measurement settings and byproduct operators from the resulting outcomes. For this step, we use the same program word described in Section 3.2.4, augmented with measurement basis information, to specify how to measure the left-most column of cluster qubits in each measurement round.

The subroutines that process measurement outcomes are written to emulate the operation of the FPGA control system in Figure 3.3. This has the benefit that the internal state of the control system maps closely to the internal state of the verification program, which aided debugging both systems.

Source-code reference: mbqc-fpga

The program word P controlling the measurement pattern is defined in the file `simulator/src/program.hpp`. The calculation of the adaptive measurement settings and byproduct operators based on this program are performed analogously to the FPGA design as part of the `storeOutcome` method of the `LogicalQubit` class, defined in `qubit.hpp`. This implementation of MBQC simulation is highly specific to the FPGA design, and is not easily generalised. Part of the motivation for MBQCSIM (discussed in Chapter 5) was to extend the simple program discussed here to a wider variety of scenarios.

The main subroutine `storeOutcome` in the program is shown in Algorithm 3. The internal state of the program consists of the set of measurement outcomes M, the program P, the byproduct operators b and the stored byproduct operators b_s. All these variables are indexed with two variables k (the logical qubit index) and n (the measurement round).

At each newly completed measurement round n, the byproduct operators are updated using the most recent measurement outcomes according to Equations (3.2) and (3.3), via the function `bypUpdate`. This corresponds to the `byproduct` VHDL entity in the FPGA design. Then the next adaptive measurement setting is calculated using the function `adaptiveSetting` based on Equation (3.1), which corresponds to the `adapt` VHDL entity. By default, it is not necessary to change the stored byproduct operators b_s, unless the program P indicates (see Table 3.2) that the byproduct operators should be stored. Similarly, if a CNOT commutation correction is required, the function `cnotCommCorrect` provides the necessary correction. If a correction is not required, it returns zero (so as not to affect the byproduct operators).

Key points

The implementation of the column-measurement step of the simulation scheme is designed to map as closely as possible to the FPGA design, to make verification of the design easier. However, this has the disadvantage that the simulator is not easily modified for other situations. This issue is partially addressed in MBQCSIM, which contains a more general-purpose MBQC simulator.

4.1.4 Verification of the simulator

It is important to determine that the program is producing valid simulations of MBQC measurement patterns, before using it to verify the digital control system design. This is achieved by simulating the target quantum circuit in the gate-based model alongside the MBQC simulation, and checking that the two simulations agree.

At the end of the MBQC simulation (see step 5 of the recycling algorithm in Section 4.1.1), the final column of the measurement pattern remains unmeasured. This column of qubits is the outcome from the quantum circuit

(up to byproduct operator corrections). After applying these corrections, the resulting state $|\psi\rangle$ should be the same as the state $|\phi\rangle$ that is obtained using a direct gate-based simulation of the quantum circuit.

These two states are compared in the simulator using the Fubini-Study distance, defined as follows:

$$d(|\psi\rangle, |\phi\rangle) = \arccos\left(\frac{|\langle\phi|\psi\rangle|}{\|\phi\|\|\psi\|}\right). \tag{4.4}$$

If the states are normalised, then d reduces to the arc cosine of the fidelity [50].

The distance $d(|\psi\rangle, |\phi\rangle) = 0$ if and only if the states $|\psi\rangle$ and $|\phi\rangle$ represent the same state. This is used as verification that the measurement patterns used in the MBQC simulation perform the correct quantum operation[4].

Key points

This verification forms the basis for the noise simulations described in Chapter 6. By introducing errors into the measurement process (for example, due to the noise in modulator voltages), it is possible to use the fidelity between the simulation state $|\psi\rangle$ and the true state $|\phi\rangle$ to quantify the errors that occurred due to the noise.

4.1.5 Simulator input and output

The input to the simulation program consists of the set of program words $P[n, k]$ (see Equation (3.4)) for each logical qubit k and each measurement round n. To simplify this process, the program contains some utilities for automatically producing the correct set of program words on the correct logical qubits for one-qubit gates and CNOT gates, which can be read from a file. The format of the "circuit file" is very simple, as follows:

```
N=3
u 0 0.1 0.2 0.3
cnot 0 1
```

[4]Due to the finite-precision of the computer simulation, the distance results are not exactly zero even for a correctly working measurement pattern. In practice, on the x86_64 architecture used for to develop and run the program, distance values of 1×10^{-8} were typically obtained. Errors often caused distances on the order 1, making it easy to identify problems.

Option	Meaning
`-h`	Print a brief summary of command line options
`-c`	The file name of the input circuit file to be simulated
`-o`	The file name of the output log file (optional)
`-p`	The output program file base name (optional)
`-s`	The single logical qubit testbench data file base name (optional)
`-m`	The multiple logical qubit testbench data file base name (optional)
`-q`	Specify which qubit to store (optional)

Table 4.1: The command line options for the MBQC simulator program. If the optional arguments are omitted, then those files will not be written. The "base name" refers to the beginning of the filename – the qubit index will be appended to the end.

```
u 1 -0.3 -0.2 -0.1
cnot 1 2
u 2 0 0 1
...
```

The first line of the file is the line `N=n`, which specifies that there are n logical qubits in the circuit. After that, the quantum gates are listed in the order that they should be applied. The lines `u k xi eta zeta` apply one-qubit gates $U = R_x(\zeta)R_z(\eta)R_x(\xi)$ to the k^{th} logical qubit. Lines such as `cnot c t` apply a CNOT gate between the control qubit c and the target qubit t.

After the simulation has completed, the measurement outcomes, byproduct operators and adaptive measurement settings are written to a file in a format that can be used in VHDL testbenches. The program is packaged into a simple command line utility called `mbqcsim`, with command line options shown in Table 4.1. The options `-s` and `-m` produce data files for use in the VHDL testbenches, described in Section 4.2.

Key points

The `mbqcsim` program reads an input gate listing, simulates the MBQC realisation of this circuit (checking the patterns for correctness), and outputs a file of control system inputs and outputs suitable for use with the VHDL testbenches discussed in the next section.

4.2 Verification of the FPGA design

Section 4.1 contains the design of an MBQC simulator for verifying MBQC measurement patterns and generating known-valid inputs and outputs for the control system at each measurement round. The use of this data in verifying the digital control system for a single logical qubit and for multiple logical qubits is described in the following sections.

4.2.1 One logical qubit

Before verifying the hardware in the most general case of multiple logical qubits (i.e. using multiple rows of the unit cell in Figure 3.1), it is important to check that the computational system for a single logical qubit shown in Figure 3.3 works as intended. For the purpose of this test, the measurements and byproduct operators associated to the logical qubits above and below the unit under test are tied to zero. This means that it is not possible to test the CNOT gate, which uses these inputs. If the test passes, then the calculation of adaptive measurement settings works, along with the storing and processing of byproduct operators associated to the one-qubit gate.

Source-code reference: mbqc-fpga

The testbench file for the single-qubit system is called `mbqc1_tb.vhd`. It may be run using Vivado's behavioural simulation [78], and outputs information to the Vivado console about whether the test passed or failed. The behavioural simulation also produces traces of the internal state of the control system during execution, shown in Figure 4.3.

On the rising edge of X_p (the photon cycle clock), the testbench reads comma separated variable (CSV) input data from a file obtained from the `mbqcsim` program using the `-s` option. Each row of the file corresponds to a measurement round, which contains the following fields:

- m, the random measurement outcome for that round. The measurement is written to the input of the control system immediately (on the rising edge of X_p)
- b', the correct value that the byproduct operators should take after the rising edge of X_s in the current measurement round.

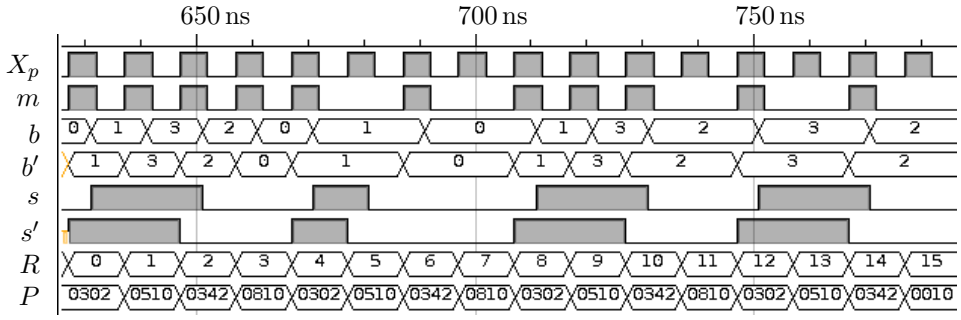

Figure 4.3: Functional simulation of the single qubit control system using data generated from the MBQC simulation program. At each measurement round R, the testbench compares s and b, generated by the control system under test, with s' and b' which are read from a file. The program word P for each measurement round is shown at the bottom.

- s', the correct value that the adaptive measurement setting should take after the rising edge of X_s in the current measurement round

There are also other fields that are useful for debugging the internal state of the control system, such as the stored byproduct operators.

The program is loaded into the control system via the distributed memory generator IP block in Vivado, using a memory coefficient file [73] which is generated using the `-p` option of the `mbqcsim` program.

On the rising edge of X_s, the control system computes the byproduct operators and adaptive measurement settings for the current measurement round. On the rising edge of X_r, the testbench reads these outputs and compares them with the true values taken from the file. If there is a discrepancy, an error is recorded in the simulation log file. If there are no errors at the end of the simulation, then the values of the byproduct operators and adaptive measurement settings produced by the control system in each measurement round are correct.

An example of the output from the functional simulation based on the single qubit testbench is shown in Figure 4.3. Data for the simulation was generated using the following circuit file, which consists of four one-qubit gates:

```
N=1
u 0 0.1 0.2 0.3
u 0 -0.3 -0.2 -0.1
```

```
u 0 0 0 1
u 0 0 1 0
```

The sequence of program words that represent the one-qubit gate is 0302, 0510, 0342, 0810 (written in hexadecimal). This sequence is repeated four times in the P row of Figure 4.3. In each measurement round, the correct values b' and s' slightly lead the control system outputs b and s, which are produced on the rising edge of X_s. However, it is clear from Figure 4.3 that the control system is correctly calculating adaptive measurement settings and byproduct operators.

The simulation was run using the "behavioural simulation" in Vivado. As a result, no timing information is present in the simulation. A frequency of 100 MHz was arbitrarily chosen for the functional verification.

Key points

Functional verification, of the type performed here, is normally the first step of verification performed in FPGA design, followed by static timing analysis of the kind discussed in Chapter 3, to ensure that the (functionally correct) design also meets its timing specification. Here, we perform the steps in reverse, because the primary goal of this thesis is to obtain exploratory timing constraints on control systems for quantum computing systems. However, functional verification is still important, even if the system is never built, to ensure that the design being analysed for timing is a fair representation of what is required in the real system.

4.2.2 Multiple logical qubits

Verification of the full system (see Figure 3.1) is required to check that the connections between logical qubit modules do not contain errors. The purpose of the multiple qubit testbench is to ensure that interactions between the byproduct operators on different logical qubits are correct when arbitrary CNOT gates and one-qubit gates are performed.

Similarly to the single-qubit case, the multiple qubit testbench reads a file produced using the `-m` option of the `mbqcsim` program, resulting in a CSV

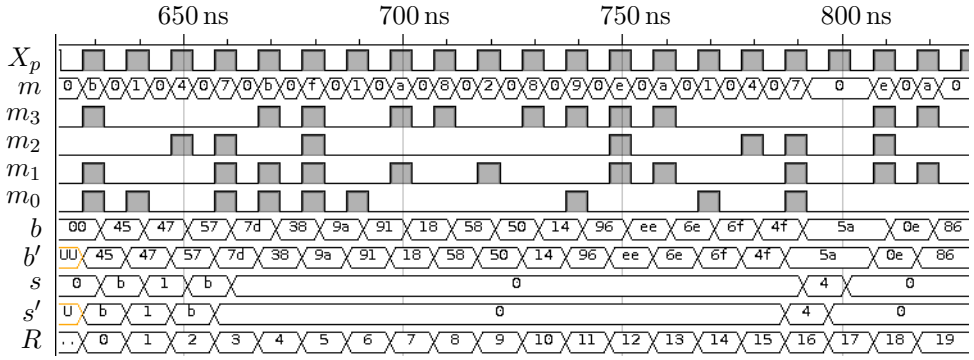

Figure 4.4: The functional simulation of the multiple qubit system with four logical qubits. The inputs and outputs are concatenated and expressed in hexadecimal. For example, the input m consists of four bits (shown in the four rows below m), and is therefore represented as a single hexadecimal character. The output b contains two bits per logical qubit, and is therefore represented by two hexadecimal characters. The equality between the primed and unprimed variables indicates that the control system performs correctly.

file where each row (corresponding to a measurement round) contains the following fields:

- m, the random measurement outcome for that round. This time, the field is an N-bit value, where each bit corresponds to the measurement outcome from a different logical qubit row.
- b', the correct value of the byproduct operators in the current measurement round. The byproduct operators from each logical qubit row are concatenated, leading to a $2N$-bit field.
- s', the correct value of the adaptive measurement setting in the current measurement round. The measurements settings for each logical qubit row are concatenated to form an N-bit field.

An example functional simulation based on the multiple qubit testbench with four logical qubits is shown in Figure 4.4. The circuit file used to test the multiple qubit system was:

```
N=4
u 0 0.1 0.2 0.3
u 1 -0.3 -0.2 -0.1
```

```
cnot 0 1
cnot 1 2
u 2 0 0 1
u 3 0 1 0
```

In the simulation results presented in Figure 4.4, inputs and outputs over multiple logical qubits are expressed by concatenating the corresponding fields for each logical qubit, placing lower-index logical qubits at less significant positions. For example, the measurement inputs are m_i for each logical qubit i are concatenated to form the four bit value m, which is expressed in hexadecimal in the figure. The byproduct operators are expressed using 8 bits, because each logical qubit contributes two bits. Similarly to the single qubit case, the control system is verified by observing that the byproduct operators b and b' agree, as do the adaptive measurement settings s and s'.

Although the simulation shows a relatively small circuit, it is possible to generate data for arbitrary long circuits with up to 14 logical qubits, to increase the chance of catching edge cases. In this case, it is not practical to check manually that all the outputs agree between the simulation and the control system. However, the testbench automatically checks for equality in each clock cycle, and reports the results at the end of the simulation.

Next steps

Functional verification via behavioural simulation is one route to checking that a digital design is correct. However, a stronger form of verification could be obtained by using an approach based on formal verification (proof-based verification) [77]. It is likely that MBQC control systems would be amenable to this type of verification technique, due to the simple fixed rules that must be applied to realise measurement patterns. This type of formal verification is compatible with VHDL [82]. By using formal verification, it would not be necessary to generate (pseudo-random) inputs and outputs to test the system – the formal properties would establish that the system works for all combinations of inputs and outputs.

4.3 Summary

The verification of the control system described in Chapter 3 is necessary to ensure that the implemented design is free of inaccuracies. There are two main places that bugs can occur: in the measurement patterns themselves; or in the implementation of the control system.

In Section 4.1, we described the design of an MBQC simulator capable of checking that the measurement patterns perform the correct quantum operations. Due to the efficient recycling of the state vector, the simulator is capable of simulating arbitrarily long quantum circuits up to a cluster-state height of 14 logical qubits.

The control system was verified by comparing its outputs (the byproduct operators and adaptive measurement settings) with known-true outputs from the simulator, using VHDL testbenches described in Section 4.2. We showed the results of the testbenches for MBQC circuits containing one and four logical qubits. The simulation and verification framework provides evidence that the digital control system is free from errors.

This chapter marks the end of the analysis of control systems for photonic MBQC using deterministic cluster states. Chapters 5 and 6 extend the control system ideas and simulation techniques developed so far to the realm of MBQC based on incomplete cluster states (IMBQC). Although many of the ideas relating to measurement patterns carry over to IMBQC in principle, there is a substantial increase in algorithmic complexity, due to the need to map measurement patterns dynamically onto a cluster state where random edges may be missing. This means that it is not possible to progress immediately to designing a prototype FPGA-based control system to analyse timing constraints in this case. Instead, an additional emulation step is required, to establish what performance and timing implications follow from making the various algorithmic choices involved in searching incomplete cluster states.

Even though the simple FPGA-based control system designed here cannot be used as the basis for photonic quantum computing, it does provide a basic illustration of the methodologies required for the design of the electronic control system. The approach of using static timing analysis to investigate the feasibility of electronic control systems for use in photonic quantum computing can be applied to any prospective control system designs, including more advanced implementations based on Chapters 5 and 6.

Chapter 5

Algorithmic overheads due to incomplete cluster states

Chapters 2 and 3 contain a description of photonic MBQC in the presence of an ideal (deterministic) cluster state, and present a timing analysis of a simple control system implementation of one-qubit gates and CNOT gates targeting an FPGA. This chapter and Chapter 6 consider the implementation of control systems for photonic quantum computers based on incomplete cluster states – those which may have random missing entanglement links between cluster qubits. The randomness is caused by the inability to deterministically entangle photonic qubits [54].

Accounting for randomness in the structure of the cluster state causes a significant increase in the complexity of a digital control system designed to realise a measurement-based photonic quantum computer. This increase in complexity is primarily due to the need to dynamically generate and perform a suitable measurement pattern, compatible with the edges present in the cluster state.

The design discussed in Chapter 3 was based on a scheme of photonic MBQC containing almost no algorithmic complexity. However, even in that case, producing and analysing a prototype control system design for simple fixed measurement patterns was a non-trivial task, involving choices relating to the clocking structure of the design, the optimal choice of I/O logic, and the relative placement of different computational processes throughout the photonic clock cycle. This level of detailed analysis was only possible because decisions relating to the measurement patterns and implementation of MBQC

were not required. For example, the two measurement patterns considered were both fixed, and involved only simple Boolean logic calculations.

IMBQC, on the other hand, does not involve the same level of concreteness in its specification. Initial proposals for MBQC-based photonic quantum computing do not contain details regarding what control systems must be implemented, in a form that electronic engineers can easily understand [13, 14]. Much subsequent work has focused on photonic aspects of the problem, particularly the generation of cluster states and architectural requirements [83]. Even recent works do not present results in a form that are easily amenable to control system design, lacking information about exactly what operations the classical control system is expected to perform, and what target electronic hardware is required [16, 61]. Other research into "realistic" aspects of photonic quantum computing often adopt a software-focused approach[1] [24], or provide a variant of the scaling argument discussed in Chapter 2 [23]. There are no specific conclusions relating to how the algorithms should be implemented in hardware, or what concrete constraints these implementations may impose on photonic quantum computing implementations.

The object of this chapter is to offer an alternative route to the investigation of control-system feasibility in these more complicated regimes. We adopt the search-type strategies described in the context of cluster-state renormalisation by way of example [14, 23, 24], and present a concrete, fully specified, set of algorithms for implementing one-qubit gates in the presence of an incomplete cluster state. In doing so, we define the problem in a simple diagrammatic manner, which should make the scheme understandable to electronic engineers, or others who do not wish to become familiar with the details of MBQC. In addition, we provide an open-source and verified library, MBQCSIM, to emulate the system, and derive high-level timing constraints that may follow from control systems that implement these algorithms.

In deriving timing constraints from emulation, the results strongly depend on the implementation model under consideration, which describes what control system architecture is under investigation. An implementation model is

[1] By software focused, we mean that the feasibility of the algorithms discussed is justified by efficient implementation on microprocessors. This is not valid, because microprocessors are an entirely different type of device from the hardware which must be considered for photonic control systems. Very simplistically, microprocessors are heavily optimised for high-throughput of general purpose instructions, at the expense of exceptionally long interface-latency. However, both throughput and latency must be simultaneously optimised in control system design.

necessary to define exactly what behaviour is within the scope of the emulation, and specify the region of validity of the timing results. Here, we choose to analyse the memory operations required by the algorithms we implement, based on a simple implementation scheme that uses a ring buffer to store cluster state and algorithm data. The model and assumptions underlying the emulation are laid out in Appendix 5.1.3.

By taking this chapter as a starting point, it may be possible to design a digital system in the manner presented in Chapters 3 and 4. However, there are also many other variants of photonic quantum computing that may be considered. As with earlier chapters, we aim to lay out an approach to the analysis of control systems that may be applicable to other proposals for photonic quantum computing.

Key points

Control systems for IMBQC are substantially complicated by the need to dynamically generate measurement patterns onto a random cluster state. This extra complexity must be emulated to assess the performance of alternative algorithmic solutions to the problem. By providing an implementation model defining certain aspects of the control system design, this emulation can be used to derive timing constraints on control systems designed within the scope of the model.

In Chapter 6, we extend the simulation techniques developed in Chapter 4 to obtain estimates for the logical qubit errors introduced by modulator voltage noise for the emulated implementation we consider in this chapter.

5.1 Photonic MBQC using incomplete cluster states

The simplified model of photonic MBQC (using an ideal cluster state) considered in Chapters 2 and 3 contains essential elements of any control system design, for example, the need to encode measurement patterns, and implement them by setting modulator voltages and interpreting photon detector outcomes. However, it is not possible to realise this simple system in practice, due to the inability to deterministically perform a CZ gate between dual-rail encoded photonic qubits – one of the factors that prevents the realisation of a

simple gate-based photonic quantum computer in the first place [54]. Instead, various mechanisms have been introduced to non-deterministically entangle two photons; for example, fusion gates [13], or boosted fusion gates [15]. The use of these gates is heralded, meaning that it is known whether the entanglement operation has succeeded or not.

Compared to Chapter 2, this means that it is not possible to lay out logical qubits along rows in the cluster state; instead, they must be mapped to paths, which track successful edges in the cluster state. This approach is called renormalisation [83], where a large incomplete cluster state is course-grained into blocks, each of which contains at least one "accessible" cluster qubit that is connected to the accessible cluster qubits in surrounding blocks, via winding paths. It has been shown that, depending on the randomness in the cluster state and the size of the blocks, this process will always succeed in providing a higher-level deterministic cluster state (where all edges – now paths – are present) between accessible qubits [14].

Although of theoretical importance in establishing the possibility of using incomplete cluster states in photonic quantum computing, the conceptual framework of renormalisation is not helpful for control system design. This is because the primary processes – finding paths through cluster states, and dynamically generating the measurement pattern – are not explicitly emphasised. A much more suitable approach is based on the explicit consideration of algorithms used for pathfinding [23]. We extend these methods here to a framework that can provide concrete timing constraints on hardware implementations of the algorithms required.

Therefore, the model of photonic quantum computing considered in the remainder of this thesis is the following:

- **Random cluster states.** MBQC is realised using an incomplete 2D cluster state, meaning one with random missing entanglement links between some qubits.

- **Column-wise photon entanglement.** Photons are produced one column at a time, as described in Chapter 2, using a black-box cluster-state generator that may fail to entangle photons.

- **Heralded entanglement.** The mechanism that produces the entanglement is heralded, meaning that it is known to the control system when an edge is present and when it is not.

- **Fixed edge probability p.** The probability of successful entanglement is given by p, which is the same across the cluster state. Furthermore, the presence or absence of the edges are independent from one another.
- **No photon propagation loss.** This means that the control system may assume that heralded edges are valid, and will not change with time[2].
- **Realise one-qubit gate paths.** We are specifically interested in how to realise arbitrary one-qubit gates, by dynamically generating a measurement pattern that is mapped onto a path through the cluster state.

The primary addition to a control system in the model above is that there are a new set of inputs, corresponding to which edges are present in the cluster state. This may be seen as a new set of photon detectors outputs feeding into the digital system in Figure 2.4 of Chapter 2, corresponding to the heralded outputs from fusion gates. This data is produced when each column is produced, and is stored for use by the digital implementation. We will refer to the model of photonic MBQC described above as IMBQC (for incomplete-cluster-state MBQC).

Key points

Renormalisation [14] is an important theoretical framework for analysing the effect of non-deterministic entangling gates in MBQC. However, the pertinent information relating to control system design is what algorithms should be used to identify paths through cluster states, and what algorithms should generate measurement patterns. An overview of how cluster-state edge information can be used to generate a one-qubit measurement patterns is described in Section 5.1.1.

5.1.1 Steps involved in implementing IMBQC

A high level summary of what is required for the implementation of the scheme is shown in Figure 5.1. One immediate consequence of having an incomplete cluster state is that multiple columns of photons must remain unmeasured

[2] If photons were lost, then this would remove any entanglement edges connected to that photon, and remove the possibility of using these edges in paths.

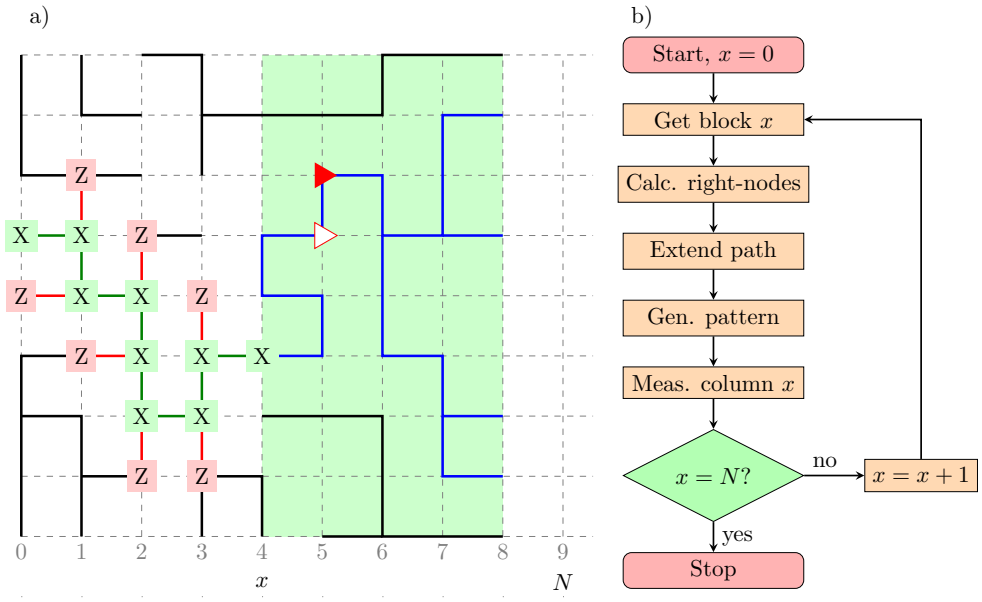

Figure 5.1: Diagram of the overall algorithm for implementing a single logical qubit in photonic IMBQC, which is mapped to a winding path through the cluster state. The incomplete cluster state is searched one block at a time (shaded green), to establish a path for the qubit. The search process is decoupled from the path extension through the establishment of right-nodes on the path (see Section 5.3). Once the path is established, the measurement pattern is generated, taking account of qubits which must be cut out around the path; then the left-most column of the block is measured out. In the implementation considered here, data relating to the nodes in the shaded green region is stored in a ring buffer, as described in Appendix 5.1.3.

simultaneously [23], shown as the shaded green region in Figure 5.1a. On the right of this region, columns are generated, and the corresponding edge data is recorded. On the left side, the column is measured in the dynamically generated measurement basis that is required for the realisation of the quantum circuit. This may be seen as a generalisation of the control system presented in Chapter 3, which only kept one column of photons in delay lines at any one time, as described in Section 2.2.

The buffer window is necessary to ensure that the dynamic measurement pattern, mapped to a path, can be propagated to the right as the cluster state is generated, without hitting dead-ends. Throughout this chapter, this buffer will be called the "block subwindow", or just "block" for short[3]. As each new

[3]The "subwindow" terminology arises from how this aspect of the system is realised in

column of photons is measured on the left, and a new column is added on the right, the block moves one step to the right. Blocks are characterised by their width B and starting column x. This block width is a very important implementation parameter, because it determines how long a photon must be delayed in the system before being measured. The block width can be related directly to the timing constraints of the digital system implementation, for example the memory latency, as we describe in Section 5.5.2.

Figure 5.1b shows the steps required to implement IMBQC using the block-search method. In order to find a path, a search algorithm is required in the current block. In order to separate the algorithm which performs the search from the algorithm which extends the path based on the search results, we make use of the concept of right-nodes, defined in Section 5.3. From this point of view, the search algorithm may be replaced by any algorithm which outputs right-nodes.

Many search algorithms have been considered for this purpose [14, 23, 24]. The main objective of this chapter is to take the simplest proposed search algorithm – the breadth-first search (BFS) – and emulate a full implementation of IMBQC based on this algorithmic choice. We consider two variants of BFS, described in detail in Sections 5.4.1 and 5.4.2, and show that one has extremely high algorithmic overhead, and the other does not succeed in finding paths through the cluster state. In the process, we lay out a framework for analysing algorithmic overheads in IMBQC, which may be used to analyse other proposed algorithms for finding paths through incomplete cluster states.

Once a path extension has been obtained, the measurement pattern for the one-qubit gate can be laid out along the path. A one-qubit measurement pattern consists of a set of measurement bases along a path P through the cluster state, together with a set of adaptive measurement setting dependencies and byproduct operator update rules; and a set of Z-measurements on cluster qubits which are connected to P via an edge. For example, the identity pattern (the pattern that would perform the identity gate on the logical qubit) consists of a set of X measurements along the path P (shown in green in Figure 5.1), and a set of Z-measurements on adjacent cluster qubits (shown in red in Figure 5.1). There are no adaptive measurement setting dependencies in this case; however, there are byproduct operator update rules which

the program described in Section 5.2; the terminology is kept throughout for the sake of consistency.

are described fully in Chapter 6.

In more complicated measurement patterns, for example, in a full implementation of renormalisation, consideration must also be paid to the interaction between different logical qubit paths (or even more general measurement-pattern constructs). In this chapter, only the implementation of one-qubit paths is discussed. However, the same approach presented here is applicable to other more advanced implementations.

Key points

A control system for IMBQC must use the edge data of the cluster state to find a path, and map a measurement pattern onto it, in real time. The steps to do this are shown in Figure 5.1. Compared to Chapter 2, it is now necessary to store multiple columns of photons in delay lines. The number of stored columns, B, is the origin of the timing constraints we discuss in Section 5.5.2.

5.1.2 The need for hardware emulation of IMBQC

The complexity involved in implementing and analysing the trade-offs in all the steps shown in Figure 5.1b, and how they relate to the timing characteristics of the photonic system, means that the design of a prototype system for IMBQC, of the kind presented in Chapter 3, is not immediately feasible. However, it is possible to undertake a prerequisite intermediate step, and analyse what constraints may arise from generic classes of control system implementation. The object of this analysis is to narrow down the set of algorithmic choices by ruling out those which impose more significant timing constraints on the system. Two key components are required as the basis for this analysis:

- **Implementation model.** A specification defining how certain aspects of the implementation will work, but without providing implementation details such as a digital design of the subcomponents.

- **Software emulation of the model.** A simulation of the internal behaviour of the implementation model, as a function of various design choices, for the purpose of analysing how the design choices affect the overall performance of the system.

Emulation provides three key benefits. Firstly, it shows whether a particular proposal for a control system implementation of IMBQC will work, in the sense of the verification described in Chapter 4. This verification is much more complicated in the case of IMBQC based on Figure 5.1b than it was in the simple model presented in Chapter 2, because of the diversity and complexity of the algorithms involved.

Secondly, the emulation shows how the algorithms will perform in realistic operating conditions. We use this to derive timing constraints on the behaviour of control system implementations in Section 5.5.2.

Finally, the design and implementation of the emulation[4] provides a concrete written draft of all the steps involved in all the algorithms that must be implemented in hardware. This may be taken as the starting point for a hardware design written in a specific language such as Verilog or VHDL. If the design is open source, it may serve as the starting point for others to better appreciate the details involved in IMBQC implementation. Furthermore, once a hardware design is in place, the emulator may form the basis of the functional verification of the hardware, using the techniques discussed in Chapter 4.

In the next section, we describe explicitly the implementation model, based on a ring buffer for storing the block, and provide an overview of how the algorithms in Figure 5.1b map onto this model.

Key points

Emulation of IMBQC algorithms provides a framework for understanding timing constraints due to various algorithmic choices, without needing to implement the hardware realising the algorithms. To define the scope of this analysis, an implementation model is necessary, which we describe in the next section. The emulation of the model also provides evidence that the algorithms work, and the source code may serve as the basis for hardware design and functional verification.

[4]Note that this refers to the design of the emulator itself, not the design of the implementation model.

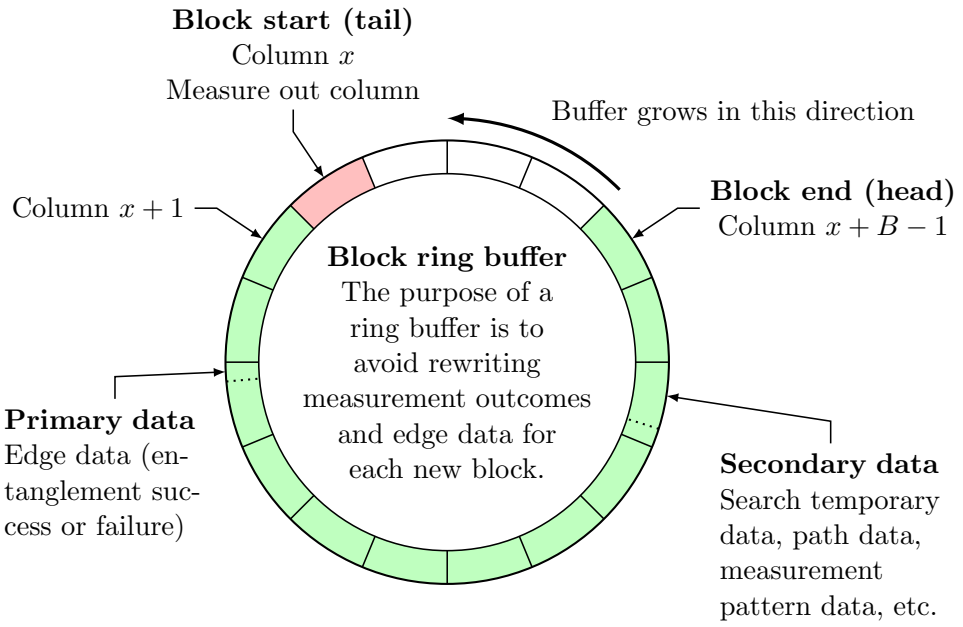

Figure 5.2: Schematic representation of the ring-buffer implementation model for IMBQC. Each entry in the ring buffer stores information relating to a column of cluster qubit data. At the buffer head, edge data comprising entanglement success or failure is recorded as a new column of photons is generated. The advantage of a ring buffer is that this data is only written once. At the tail end, the buffer is reduced in size by one when a column is photons is measured out. The ring buffer also stores implementation specific secondary data that may be required by the pathfinding and pattern generation algorithms that are used.

5.1.3 The ring-buffer model of control system implementation

This section contains an implementation model of IMBQC based on the use of a ring buffer[5] for storing block information. We focus here on the details of the underlying data structure of the implementation, which provides the basis for the analysis of memory related timing constraints in this chapter, and the memory layout for the local measurement-pattern rules discussed in Chapter 6.

A ring buffer is an implementation of a first in, first out (FIFO) data structure [84] which consists of a bounded buffer region whose ends are logically

[5]Also called a circular buffer.

connected, as shown in Figure 5.2. The advantage of this structure is that it may be implemented simply in hardware or software by utilising a contiguous block of memory, and storing the next available location for writing (one past the `head` in Figure 5.2), and the last valid location for reading (the `tail`). When data is added to the buffer, the `head` is incremented once (advances one position anticlockwise in Figure 5.2), and when data is read, `tail` is incremented once. This way, old data is continually overwritten by new data, and no error occurs provided that `tail` is always strictly in front of `head`.

The advantage of block-based IMBQC as shown in Figure 5.1a is that the block is a fixed size, so the buffer need only be as large as the block width ($B+1$, to account for the possibility of write-before-read). In addition, even though the block subwindow logically moves to the right in Figure 5.1a, appearing to require the rewriting of all data at each new block, the ring-buffer model means that each column is only written once. Instead of moving the data, the `head` and `tail` pointers are moved, and old column data is overwritten by new column data as the `head` pointer moves anti-clockwise around the buffer.

Each entry in the buffer shown in Figure 5.2 stores a column of block information, and its associated data. This includes the vertical edge data for that column, and the horizontal edge data connecting one column to the next. We refer to this data as primary data, because any implementation of IMBQC in this model must store this information[6].

In addition to primary edge data, the buffer must also store local information required by the implementation of the various algorithms required by IMBQC. For example, path information may be stored locally as an offset from one cluster-qubit node in the buffer to the cluster-qubit node which follows it on the path. Or, the search algorithms may store information relating to the search process, for example, cluster-qubit predecessors and distance data. This information is algorithm specific, and we refer to it as secondary data.

Secondary data is also stored in the ring buffer, and consequently we impose the following two conditions on the data:

- **Locality.** The data must be storable in the entries of the ring buffer, which correspond to qubit positions in the block. Secondary data must

[6]Exactly how the data in the buffer is laid out, including alignment and data size, is an implementation detail outside the scope of this implementation model. These kind of details would be specified by a full hardware design of the kind discussed in Chapter 3.

therefore be cast in a form that is local to cluster qubits.

- **Homogeneity.** The data structure must be the same at all cluster-qubit positions in the ring buffer.

Both requirements are intended to simplify a hardware realisation of the implementation model as much as possible. The first requirement removes the need to consider another data structure in addition to the ring buffer for the storage of secondary data. The second requirement guarantees straightforward alignment of the ring buffer in memory (by requiring that each buffer location be the same size), which ensures that hardware logic for processing buffer entries does not have to depend on which entry is being read.

Key points

The ring buffer contains primary data consisting of cluster-state edge data, which is required for all algorithms. Algorithm specific data (arising from the search process or the pattern generation process) is contained in secondary data. Conditions are imposed on this secondary data to ensure that it fits in the ring buffer and does not impose unwieldy implementation constraints.

5.2 Overall design of MBQCSIM

This section describes the design of MBQCSIM, which is a C++ library together with associated analysis software for the ring-buffer implementation model described in Appendix 5.1.3. The design goals of MBQCSIM are as follows:

- **Open source**, so that the design is available for others to analyse and use.
- **Modular**, so that it is straightforward to plug in different algorithms for analysis.
- **Tested and verified**, so that there is some assurance that the program works correctly and the results are valid.

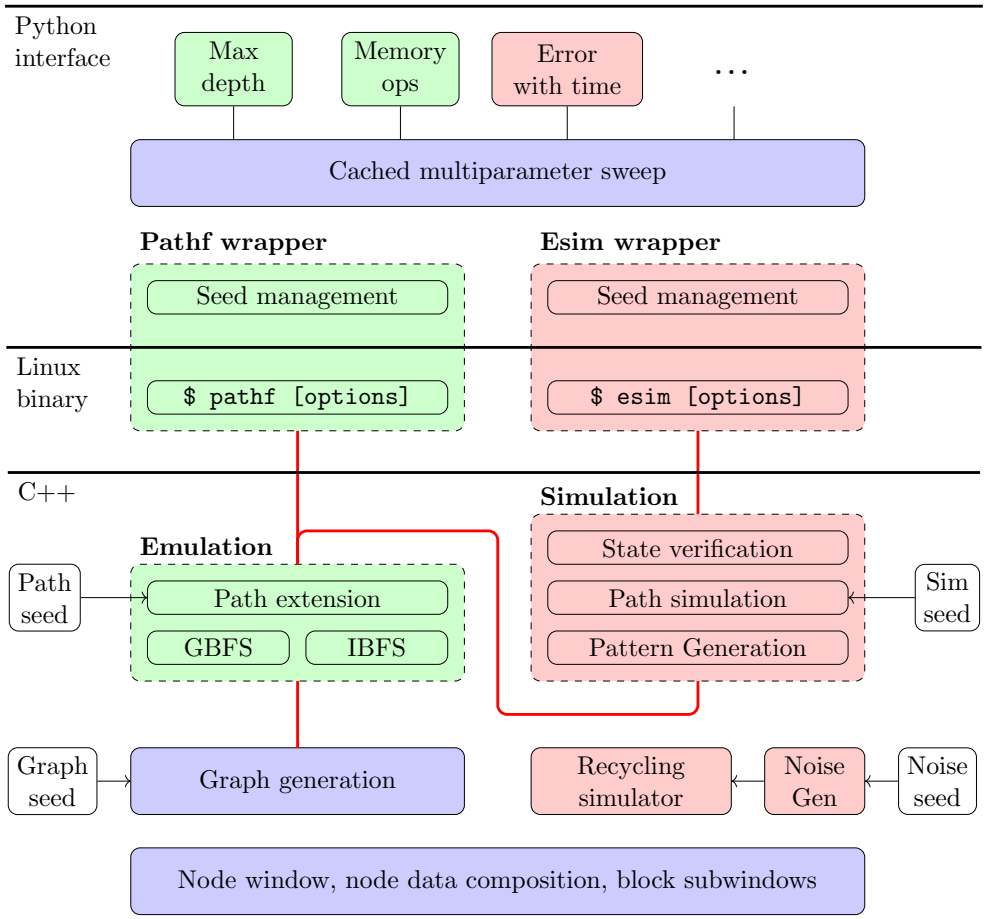

Figure 5.3: Block diagram of the design of MBQCSIM. The library contains two parts: a C++ library, which contains the main framework for emulating and simulating MBQC systems; and a python interface for easily analysing the data produced by two example executables: `pathf`, and `esim` (described in this chapter and the next). Regions shaded in green predominantly relate to the emulation of pathfinding algorithms, and regions shaded red relate to the simulation of the quantum aspects of the MBQC system. Regions shaded in blue are generic aspects of the program. The locations where randomness is used in the program is indicated by the location of the four seeds. The thick red line shows the sequence of operations that are performed on a given block.

- **Reproducible**, so that it produces the same results in different environments, and edge cases in program behaviour can be isolated and analysed.

The design of this system is provided schematically in Figure 5.3. The

diagram shows a high-level overview of how the program is constructed. At the lower levels is a C++ library that contains utilities for modelling a rectangular window of cluster qubits, and provides a facility for obtaining block subwindows that model the data in the ring buffer at any one time. Based on these components, the C++ library contains an emulation of the search and path extension algorithms (shaded green in the diagram), which can be run via the program `pathf`. For the purpose of analysing the effect of noise in the system, the library provides a full quantum simulation of the resulting measurement pattern for a one-qubit path (shaded in red), which is exposed through the program `esim`. This latter aspect of the program is discussed fully in Chapter 6.

In order to make output from the library reproducible, every element of the program that models randomness is controlled using a seed, which uniquely defines the random numbers that it will generate. The programs `pathf` and `esim` are accessed via python wrappers, which manage the seeds necessary for running the library components that involve randomness, which is described fully in Appendix B.2. These wrappers take arguments that configure various parameters that control the library components, such as the block width B, the search algorithm used, and a root seed which uniquely defines the results from the experiment.

Instead of running the program wrappers directly with fixed parameters, it is more useful to automatically be able to run multiple versions of the program, sweeping over various parameters (such as edge probability p, or block width B). This functionality is provided by the multiparameter sweep near the top of Figure 5.3, which aggregates the results from multiple calls to `pathf` or `esim` into one dataset. Based on this sweeping function, it is possible to define various experiments that answer specific questions relating to the implementation model (for example, how many memory operations occur, as a function of algorithm and parameter choices). The ellipsis is meant to indicate that the user of the library can write their own experiments.

MBQCSIM has been designed with correctness in mind, rather than performance[7]. Partly as a result of this, and partly due to the intrinsic complexity of the program (especially the quantum simulation), the running time of

[7]High performance is a long term goal, and the reason for choosing C++ as the language for library implementation. However, many decisions designed to enforce correctness (for example, the often redundant use of internal program checks) harm performance. These will be removed once correctness is fully established.

`pathf` and `esim` can be quite long. To mitigate this, the multiparameter sweep caches results that have been previously performed. This is possible because the random behaviour of an instance of the program is uniquely defined by a seed, which can be used to reproduce exactly the same results. Providing this caching function makes it easy to decouple the running of the experiment from the subsequent data analysis.

Source-code reference: MBQCSIM

All the source code for the C++ library of MBQCSIM is contained in the `src/` folder. Throughout the following sections, we will point out where important parts of the program are implemented by referencing files in this folder. The Python library is located in `scripts/py_mbqcsim/`.

5.3 Path extension using right-nodes

An important design feature of MBQCSIM is the potential to easily implement different algorithms in order to see how they perform. For this to make sense, some kind of separation must be imposed between different algorithms. One such separation is between the search process, which establishes potential paths through a block, and the algorithm which extends the path. This separation is enforced by the calculation of right-nodes, discussed in Section 5.4. Here, we discuss how right-nodes are used to extend the path. In addition, right-nodes constitute one way to mitigate against problems caused by backtracking (when the path doubles back on itself).

Given a starting point for the path in column x, the path must be extended to column $x+1$. After the path has been extended, column x is measured out, meaning the path may not re-enter column x (which no longer exists). In order to avoid this re-entry, the path must be advanced to a right-node in column $x+1$, defined below:

Definition 1 (Right-node)**.** *Given a path P comprising a sequence of edge-connected nodes $((x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N))$ in a 2D cluster state, a right-node in column x is a node (x_n, y_n) in P which satisfies two conditions:*

1. $x_n = x$;

2. $x_m \geq x$ *for all* $m > n$.

The (unique) minimal right-node in column x *is the right-node* (x_n, y_n) *that minimises* n.

A right-node in column x has the property that all path successors lie in or to the right of column x. This condition is what is needed to ensure that a path does not backtrack into the region of the graph that has been measured out. Right-nodes have been considered implicitly in the context of pathfinding [23]. In MBQCSIM, the calculation is made explicit, and the cost of the calculation is explicitly quantified.

The importance of avoiding backtracking is not related to the requirement to time-order adaptive measurement settings and outcomes in the measurement pattern. For example, the identity pattern, which consists entirely of X and Z measurements, does not have any adaptive measurement dependencies, and therefore may be measured in any order (including an order which splits the path in two due to backtracking). Instead, backtracking must be avoided in order to prevent the "false dead-end" that would arise if an undiscovered part of the future path happens to backtrack into the measured region of the window. In this case, because the path has not yet been uncovered, it would not have been possible to assign X and Z measurements at all before the measurement occurs.

A path can be extended to a right-node by following the algorithm laid out in Figure 5.4. The algorithm is quite simple, and involves stepping along the path until a right-node is reached in the target column, making random choices at any potential branch points in the path. (There are many alternative approaches to random choice which we do not consider here [23].) The procedure requires that the search algorithms, described in Section 5.4, store potential path successors for each qubit, and compute right nodes along each of these potential paths.

The separation of the path extension from the calculation search process (or equivalent algorithm) places a restriction on the types of algorithms which may be modelled using MBQCSIM. For example, an algorithm which non-trivially combines the search process with the path extension does not fit within this framework. Analysing these more complex scenarios falls outside the scope of this investigation into the implementation of IMBQC.

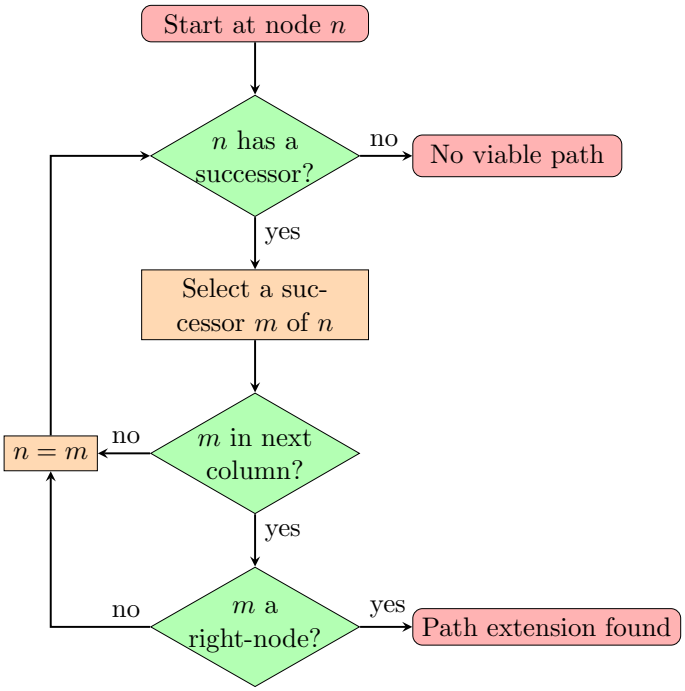

Figure 5.4: Algorithm for extending the path to a right node in the next column.

Figure 5.5 shows example output from the program, where right-nodes are highlighted on the diagram as arrows pointing to the right (instead of dots). Minimal right-nodes are highlighted in red. The utility of right-nodes in preventing backtracking is highlighted in the path extension from $(6, 6)$ to $(7, 3)$, where the path progresses two columns forward, before returning to column $x = 6$ and eventually terminating on the right node $(7, 3)$.

Key points

When a one-qubit path is advanced forward one column, it is important to progress to a right-node in the target column, so that all of the forward path lies in the region that will remain after the column to the left is measured out. The calculation of right nodes is the output from the search algorithm, and the input to the path extension algorithm (see Figure 5.4). This separation is important for the modular design of MBQCSIM, and enables the independent modification of both these algorithms.

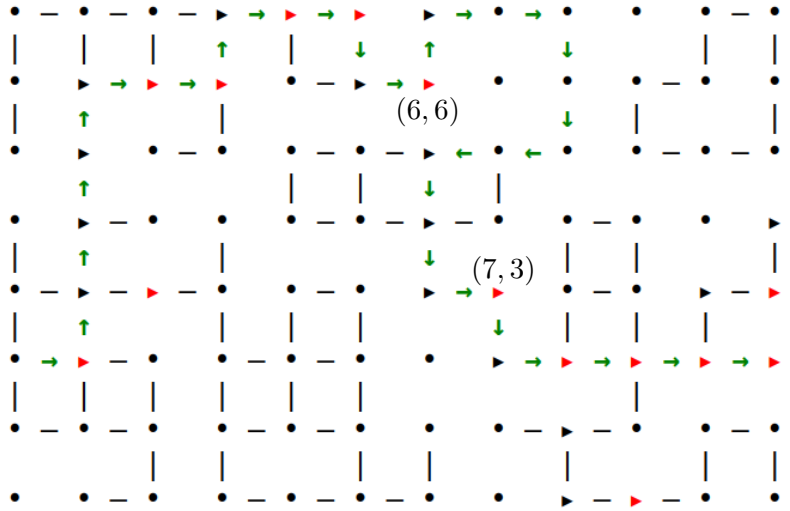

Figure 5.5: Program output showing example right-nodes along a path. The path is extended one column at a time by progressing from one right node (the red highlighted nodes) to the next. Sometimes this involves backtracking, as shown in the path extension from $(6, 6)$ to $(7, 3)$.

5.4 Search algorithm implementation

This section contains a detailed description of the two block-search algorithms implemented in MBQCSIM. The purpose of the algorithms is to write path successors and calculate right-nodes for use in the path extension algorithm.

The GBFS algorithm is a standard proposal for how to perform pathfinding in the context of photonic MBQC [14, 23]. It comprises a fresh BFS of the block subwindow at each photonic clock cycle, for the purpose of establishing viable path extensions[8]. The number of operations involved in the algorithm is large, because most of the block must be visited by the algorithm during each cycle. In addition, most of the search appears to be repeated, because the block does not change much from one photonic cycle to the next.

The simplest possible improvement that can be made to GBFS is to attempt to re-use some of the search data that is already present in the ring-buffer after the previous search has completed. This is the spirit of the IBFS algorithm (the use of "incremental" is supposed to indicate that each search builds upon the previous one). Although the operation count is significantly reduced compared to GBFS, the algorithm is more complicated, and a number

[8]The use of the word "global" is meant to indicate this full search process.

of defects in the implementation presented here cause it to fail. The resolution of these defects, along with a thorough analysis of this class of algorithms, may help to find an optimal solution for the block-search approach to IMBQC.

Key points

There are many other potential methods for finding paths through cluster states, other than using search algorithms based on the BFS. For example, one option would be to simply use a heuristic-based guess to establish path extensions. This multiplicity of algorithmic choices is part of the reason why a digital control system design cannot be immediately presented for IMBQC. The goal of MBQCSIM is to provide concrete algorithms for the search-based approach, described in the following sections, which is an established proposal for dealing with incomplete cluster state [14, 23].

5.4.1 Global breadth-first search

The GBFS algorithm is shown in Figure 5.6. The algorithm begins after a new column of photons has been generated, and edge data has been recorded in the ring buffer. It comprises a forward search over the nodes in the ring buffer, which calculates distance and predecessor information (described below), followed by a reverse pass which calculates the successors (that form candidates for the path extensions) and the right-nodes. This information constitutes the secondary data, which must be stored at each node in the ring buffer. The data used by GBFS is summarised in Table 5.1.

Source-code reference: MBQCSIM

The implementation of GBFS is contained in `global-bfs.hpp`. In the class `GBFS`, the `search` method performs the forward pass, and the `makeSPT` (make shortest-path tree) method performs the reverse pass. The class counts the memory operations that are performed as the algorithm is executed, which forms the basis for the timing analysis in Section 5.5.

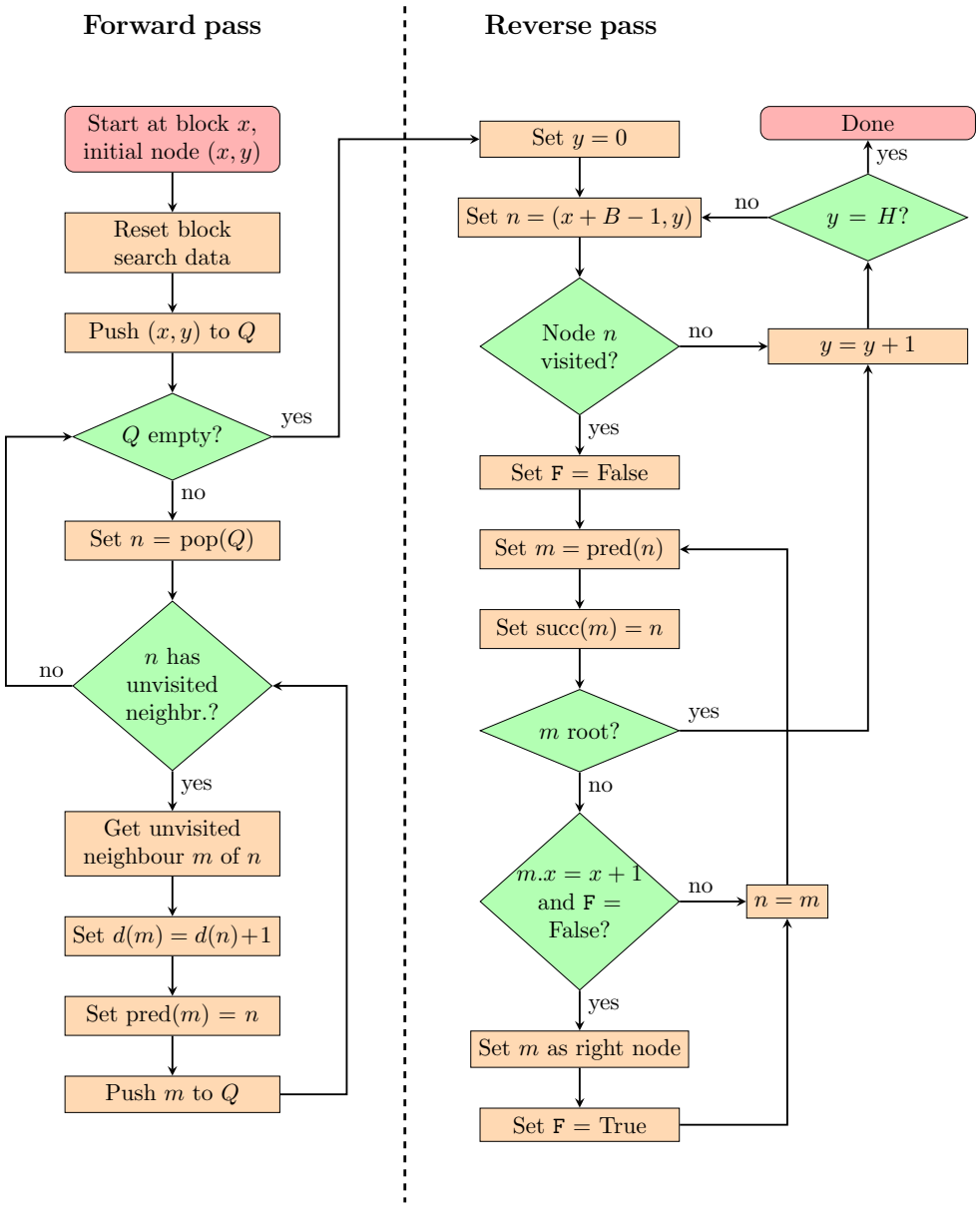

Figure 5.6: Flowchart outlining the GBFS algorithm.

Breadth-first search

The first step of the algorithm is a BFS. The BFS begins at the last node on the path (x, y). First, the algorithm must reset all the secondary data (see Table 5.1) in the ring buffer, which constitutes temporary information from one photonic cycle to the next.

Data	Meaning
Distance	The distance from the root node to node n
Predecessor	The unique BFS predecessor of n
Inaccessible flag (IBFS only)	A Boolean flag to indicate whether n is an inaccessible exit node
Right-node flag	A Boolean flag to indicate whether n is a right-node
Successors	The set of successors of n

Table 5.1: Secondary data required for the implementation of GBFS and IBFS. In traversing the graph forwards from the starting node, the distance to each node is recorded. In addition, each node apart from the starting node stores a predecessor, the node from which it was visited. These predecessors are reversed to generate a successor set for each node, which is used in the path extension algorithm (see Figure 5.4). Finally, a flag is used to indicate whether a given node is a right node. For IBFS, an additional flag is required as part of the failed-path pruning step.

Next, a standard implementation of the BFS algorithm [85] is used to construct a tree of predecessors, and each node is assigned a distance d that is one greater than its predecessor. Although distance is not important in this context (any path is desired, not only shortest-paths), the distance cannot be removed because it is a proxy for whether or not a node has been visited: initially, all nodes have distance -1, and as the algorithm progresses, nodes whose distances are not -1 are taken to be visited. (The distance may, however, be replaced with a visited/un-visited flag.) A queue (FIFO) structure Q is used to maintain the breadth-first order of traversal of the nodes.

Key points

GBFS uses a standard implementation of the BFS algorithm to generate a tree of predecessors. However, these predecessors cannot be used by the path extension algorithm, because they "point the wrong way". Instead, a reverse pass over this tree is necessary to obtain potential path successors, as described below.

Reverse pass and right-node calculation

The tree of predecessors is used in the reverse pass of the algorithm to iterate from exit nodes (visited nodes in the right-most column of the block) back to the root node. At each node, the predecessor relationship is recast as a successor relationship, which forms the basis for path extensions. It is important to note that this step cannot be optimised away – it is not possible to obtain path extensions using local predecessor information, because there is no (local) way to obtain viable successors from a given node, based only on the predecessors information. The right-node along each potential path is computed by marking the first node encountered in column $x + 1$ during the reverse pass. Since the potential path is being traversed backwards, the first node encountered in a given column is guaranteed to be a right-node for that path.

Performance problems in GBFS

The performance issue involved in GBFS is immediately apparent from the algorithm. First, secondary data for all the nodes must be unconditionally cleared at the beginning of the algorithm, resulting in a lower bound of HB writes to those memory locations. Then, with high probability (depending on the edge probability p), a high proportion of the block nodes are visited again and assigned predecessor and successor information, much of which likely duplicates the data that was already there before it was cleared. All these writes have to happen in the timescale of a single photonic clock cycle.

This is the primary motivation for developing an alternative, such as the IBFS algorithm in the next section. However, as we show, such an attempt is fraught with problems, that will likely require detailed analysis of these types of algorithms to overcome.

5.4.2 Incremental breadth-first search

The IBFS algorithm is the simplest possible attempt to remove the main defect of GBFS – the resetting of all the search data at the beginning of each new clock cycle. The algorithm begins after a new column of photons has been generated, and edge data has been recorded. However, this time, only the region between the penultimate column and the right-most column of the block is searched. This represents a significant reduction in search complexity

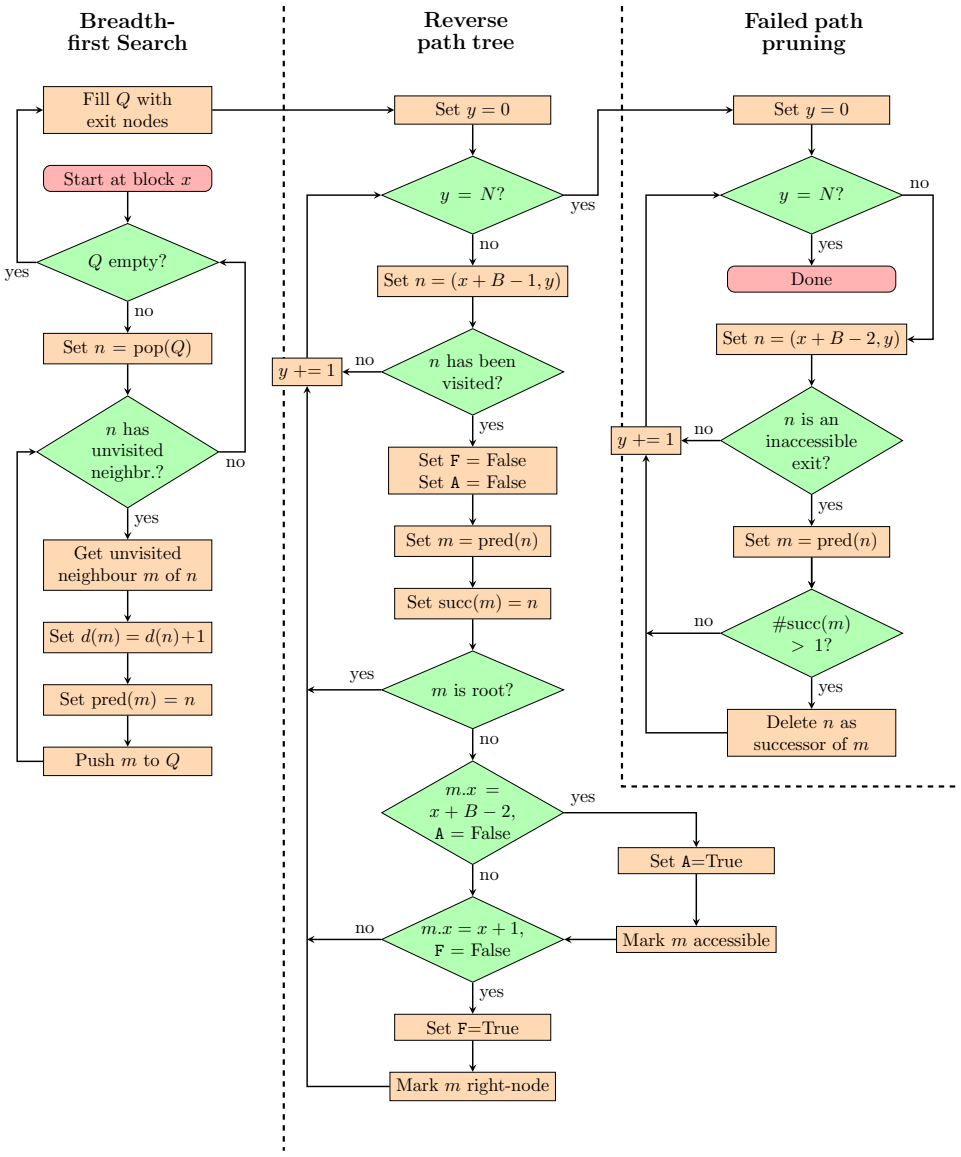

Figure 5.7: Flowchart outlining the IBFS algorithm.

compared to the full search in the GBFS algorithm. However, as a result of keeping the data from previous searches, several changes must be made in the reverse pass of the algorithm.

The IBFS algorithm is shown in Figure 5.7, and described in the sections below. Secondary data used in IBFS is shown in Table 5.1. This data includes a new flag, to mark when an exit node becomes inaccessible. This relates to the main new feature of IBFS compared to GBFS – the need to prune failed

paths.

Key points

By not clearing the secondary data at the start of each iteration, IBFS encounters a problem relating to paths that become invalid when a new column of photons is added. In the simplest possible case, this occurs when a path is not connected to the final (new) column of photons, and is therefore a dead-end. This type of path is called a failed path. In order to prevent the path extension algorithm from using one of these failed paths, it is necessary to remove them – this pruning process must be incorporated into the reverse pass of the algorithm, shown in Figure 5.7.

Source-code reference: MBQCSIM

The implementation of IBFS is contained in `iterative-bfs.hpp`[a]. The `search` of method of the `IBFS` class performs the forward pass, and `makeSPT` performs the reverse pass, including pruning failed paths.

[a]The name "iterative" instead of "incremental" is a misnomer that will be corrected in a future version of the source code.

Incremental breadth-first search

The implementation of BFS in this algorithm is quite similar to the version in GBFS, with a few differences. Firstly, it is not necessary to reset the secondary data in the ring buffer, because the main purpose of IBFS is to use this data.

Secondly, the BFS does not start with just one root node; instead, it begins with all the exit nodes from the previous block. These are the visited nodes that were in the right-most column of the previous block, and are now in the penultimate column due to the newly added column of photons. These nodes are already assumed to be in the queue from the previous iteration of the algorithm[9]. This behaviour best exhibits the incremental nature of

[9]For simplicity, we do not consider in detail how all the edge-cases for these algorithms are implemented (for example, the initial block and final block of the window). The interested reader should consult the code for MBQCSIM, which is publicly available.

the algorithm – the starting state for this algorithm would only occur in the middle of the regular BFS algorithm.

As a result of the need to begin each iteration with the queue populated by exit nodes, it is necessary to fill Q with the exit nodes at the end of the search process. This is achieved by looping over the right-most column of the block and pushing any visited node to the queue.

Although it would appear that IBFS performs an identical search process to GBFS, albeit over several photonic clock cycles instead of one, the two algorithms are not equivalent. Not only may they produce different predecessor relationships, it is not even necessarily the case that they will assign the same distances to nodes[10]. This is because a newly added column on the right may expose a shorter path to an *already visited* node inside the block. Because nodes are only ever visited once, the distance data is not rewritten. This is not a problem, because shortest paths are not an important criterion for this implementation of IMBQC. However, it exhibits a type of effect that occurs because secondary data is not being overwritten in the block.

What is a much more of a problem is the possibility that a path may be invalidated when a new column is added – for example, if it turns out that path leads to a dead-end. This problem is addressed in the section on failed-path pruning below.

Key points

Although it appears that the IBFS and GBFS algorithm perform equivalent BFS search processes, this is not the case: IBFS may not output shortest paths, because of the requirement that each node is only ever visited once (in contrast, in GBFS, nodes are visited at most once per photonic clock cycle).

Reverse pass and right-node calculation

Like the GBFS algorithm, it is necessary to perform a reverse pass over the block in order to establish right nodes. Due to the incremental nature of the

[10] In the regular BFS, the distances to the nodes are invariant, whereas the predecessor relationships are implementation defined. This is because there is only one minimum distance to each node, and BFS finds it. However, there may be multiple shortest paths, and BFS finds one valid shortest-path tree.

algorithm, it is only necessary to traverse the predecessor paths up to an exit node in column $x + B - 2$ (the penultimate column), on the grounds that a previous iteration of IBFS will have established successor information before that point.

However, it is not possible to compute right-nodes in this way. Although one could try to establish right-nodes in column $x+B-2$, by marking the first node in column $x + B - 2$ a right node, this will not work, because it is highly likely the path may backtrack into the left region of the block via a path not yet visible to the algorithm (because those photonic columns have not been created yet). As a general rule, it is best to calculate right-nodes at the left side of the block, because this maximises the forward path length on which the right node is based. Therefore, it is still necessary to make at least one reverse pass over the entire block, even though no full forward pass is necessary. This is still a substantial saving compared to GBFS, because the reverse pass only involves checking for right nodes, not writing all the successor information.

Key points

Even though IBFS only requires a relatively short forward pass, it is still necessary to perform a full reverse pass (to the left side of the block), in order to compute right-nodes correctly.

Failed-path pruning

The most important new part of the algorithm is the need to prune failed paths. Failed paths arise because a string of successors established during the searching of block x may become invalid when block $x + 1$ is searched, if the path leads to a dead-end. This cannot happen in the GBFS algorithm, because all data is reset at the start of each block search.

To establish failed paths, it is necessary to establish failed exit nodes. These are exit nodes in the penultimate column that have not lead to exit nodes in the right-most column. These exit nodes are easily established as part of the reverse pass. First, any exit node in the penultimate column encountered during the reverse pass is marked as accessible. Then, after the reverse pass is complete, one loop over the penultimate column can be used to check which exit nodes have not been marked as accessible – these are the

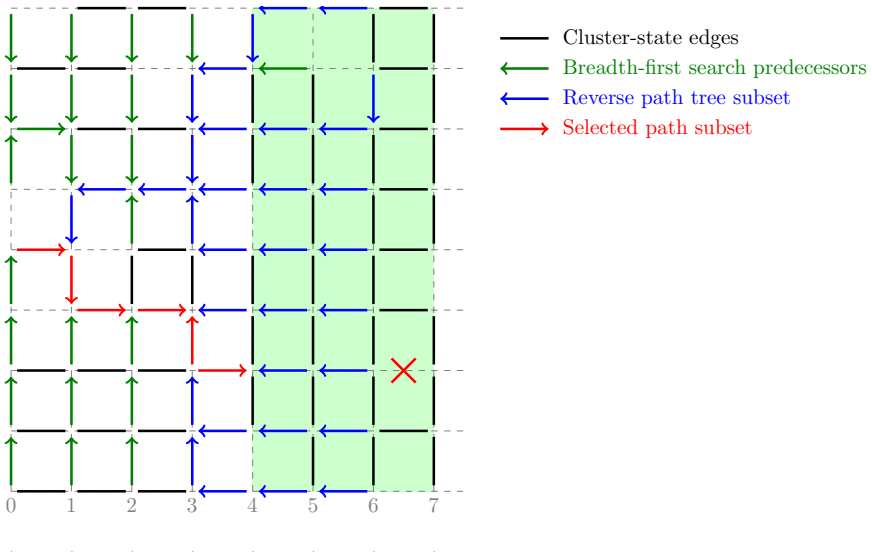

Figure 5.8: Diagram showing the most common failure case in the IBFS algorithm. Even though the path can clearly be extended, the algorithm is not able to extend the path because it cannot "see" a way around the dead-end (the reverse-path tree is missing valid edges that could be used in the path). The problem is due to the inability of IBFS to re-write the path predecessors more than once.

failed exit nodes.

Once failed exit nodes have been established, a final reverse pass of the block can be used to prune any successor paths that lead to these failed exit nodes. This is achieved by deleting the successor from the root of any tree which only leads to successor nodes. In MBQCSIM, this is achieved by iterating through predecessors until a node is discovered with more than one successor. The successor pointing to the failed exit is then removed, meaning that the path extension process will not incorrectly follow this path.

Other failure cases

There are a number of other subtle issues that arise in the implementation of IBFS. The one that ultimately causes the version of the algorithm presented here to fail is shown in Figure 5.8.

This failure case occurs when BFS finds a horizontal path through a fully connected region of cluster state. In this case, along this section, all path predecessors point backwards to the left. If a column is reached that is missing

a horizontal segment to extend this path, then IBFS will fail, even though the path could extend up or down in order to circumvent the missing horizontal link. This failure occurs because the BFS algorithm does not "know" about edges above and below the horizontal line, that it could use to avoid this missing edge, because it cannot revisit the nodes along the path from different directions.

This deficiency dramatically reduces the effectiveness of the IBFS algorithm, as we show in Section 5.5.2. A solution to this problem would require a modification to the BFS process. Whatever modification is necessary may increase the algorithmic complexity of the solution to a level comparable with GBFS.

Key points

The IBFS algorithm is less costly than GBFS, in the sense that it uses fewer memory operations. However, it is more complicated, due to the need to incorporate failed path pruning, and contains some subtle failure modes which render it ineffective for pathfinding in IMBQC. However, it may be possible to modify IBFS in order to fix these deficiencies. Given the extremely high overhead of GBFS (shown in Section 5.5), algorithms such as IBFS are more promising contenders for search-based approaches to IMBQC.

Next steps

A potential starting point for improving IBFS may be to add a limited depth- or breadth-first search process just after the detection of an inaccessible exit, in the right-hand column of Figure 5.7, to discover if there are any local edges that could be used to reconnect the node to a successful path. This might eliminate, or mitigate, the main issue that causes IBFS to fail.

5.5 Analysing algorithm performance using `pathf`

In this section, we use results from the program `pathf` to draw conclusions about the behaviour of the GBFS and IBFS algorithms, as a function of

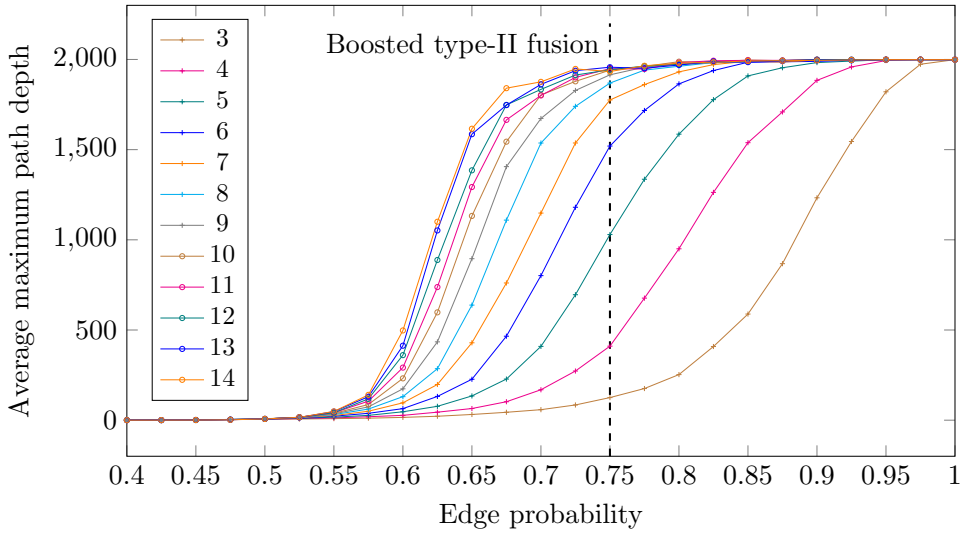

Figure 5.9: Average maximum path depth achieved using GBFS, as a function of the block width parameter (given in the legend), for cluster states with varying edge probabilities. The graph shows that the block width has a significant impact of the achievable depth, but there is limited benefit available from arbitrarily increasing the block width. The upper bound depth 2000 is due to only simulating cluster states of width 2000. The vertical dashed line shows the edge probability achieved by using boosted type-II fusion gates to generate the cluster state. This line may be used to establish what block width is necessary to achieve a particular target depth.

implementation parameters such as the block width B and edge probability p. The primary objective is to show whether the algorithms presented in the previous sections work, and derive timing constraints on implementations based on the ring-buffer model discussed in Appendix 5.1.3.

5.5.1 Average maximum path depth

A straightforward metric for whether a pathfinding algorithm works is whether it is able to find paths to a requisite depth. We show here that GBFS is very successful at finding these paths, and IBFS is not. These results support the conclusions drawn in a previous numerical simulation of pathfinding algorithms [23].

Figure 5.9 shows the maximum path depth (obtained as the average of 1000 repeated simulations for each parameter combination) using GBFS, as a function of cluster-state edge probability, and block width B (shown in the

legend), in a cluster state of width 2000 and height 20. It is clear that both the block width and the edge probability strongly affect the maximum path depth. As the edge probability increases, the average path length increases. Similarly, as the block width increases, the average path length also increases strongly, but limited improvement is obtained at much higher block widths > 10. The initial vertical position of the path was not found to affect the probability of finding paths substantially, provided it is near the middle. When it is near the edges, there is slightly less freedom for finding paths in one direction or the other. For all experiments, the path was initiated at row 10, roughly halfway up the 2D grid. For a working algorithm like GBFS, the probability of finding paths through any given block is very high when the percolation threshold ($p = 0.5$) is exceeded, in line with general results on percolation through incomplete graphs [23]. However, the slight chance of failing to find a path becomes more pronounced as more blocks are searched, as shown in Figure 5.9.

Key points

GBFS is substantially better than IBFS at finding paths through incomplete cluster states. However, the ability to find paths also strongly depends on the edge probability p (a property of the cluster state), and the block width B. For GBFS, a block width of $B = 5$ is sufficient to obtain an average maximum path length of 1000, for a cluster state with edge probability $p = 0.75$ (arising from the use of boosted type-II fusion gates).

Figure 5.10 shows the same analysis performed using IBFS. It is clear that IBFS is significantly worse at finding long paths, although it does not fail altogether. There is a slight dependence of the maximum path length on the block size, but this is much less marked than with GBFS. As the edge probability approaches one, the average path length increases substantially as the search process enters a degenerate state that succeeds with unit probability.

5.5.2 Algorithmic overhead of pathfinding

The main purpose of MBQCSIM is to emulate the block-search algorithms discussed in Section 5.4, for the purpose of placing timing constraints on the

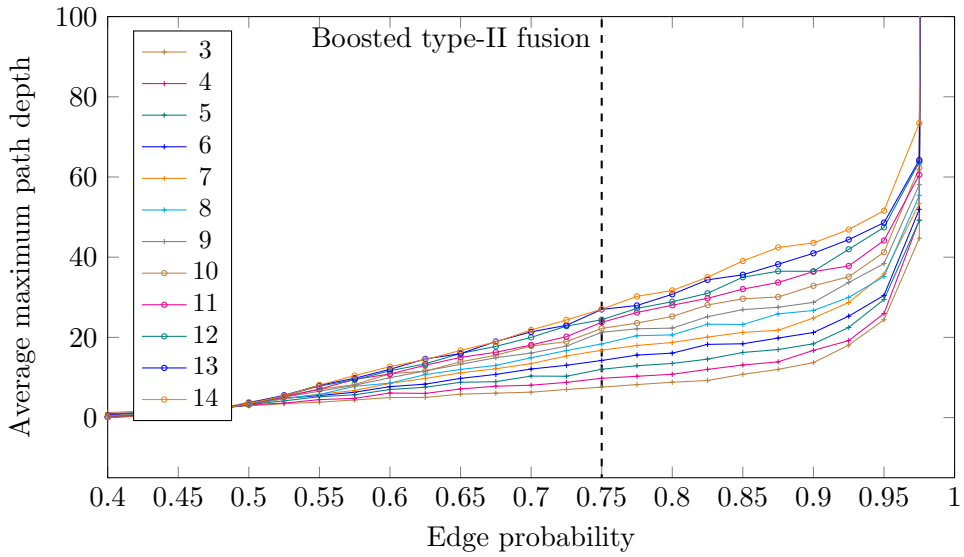

Figure 5.10: Average maximum path depth achieved using IBFS, as a function of the block width parameter (given in the legend), for cluster states with varying edge probabilities. The graph shows that the IBFS algorithm performs substantially worse than GBFS for nearly all edge probabilities, due to the limitations outlined in Section 5.4.2. (For a correctly working algorithm, the expected behaviour is a family of phase-transition-like curves showing longer paths above the percolation threshold $p = 0.5$, as shown in Figure 5.9.) This experiment was performed alongside GBFS using the same cluster state width 2000.

ring-buffer implementation model described in Appendix 5.1.3. This section contains the results of this analysis.

Figure 5.11 shows the average number of predecessor writes that are made during each block search using the GBFS algorithm, while searching a cluster state of height 20, and varying block width B and edge probability p. This number includes the initial reset of the block, which contributes HB predecessor writes. As the edge probability approaches 1, the number of predecessors approaches the maximum $2HB$, where all nodes in the block are visited by the BFS algorithm.

The y-axis of Figure 5.11 may be interpreted in the context of memory latency for a target memory technology used to implement the ring buffer. In the implementation discussed here, the memory accesses are performed sequentially, and must all be completed within the photonic clock cycle. If the photonic clock cycle is T_p, and the average number of predecessor writes

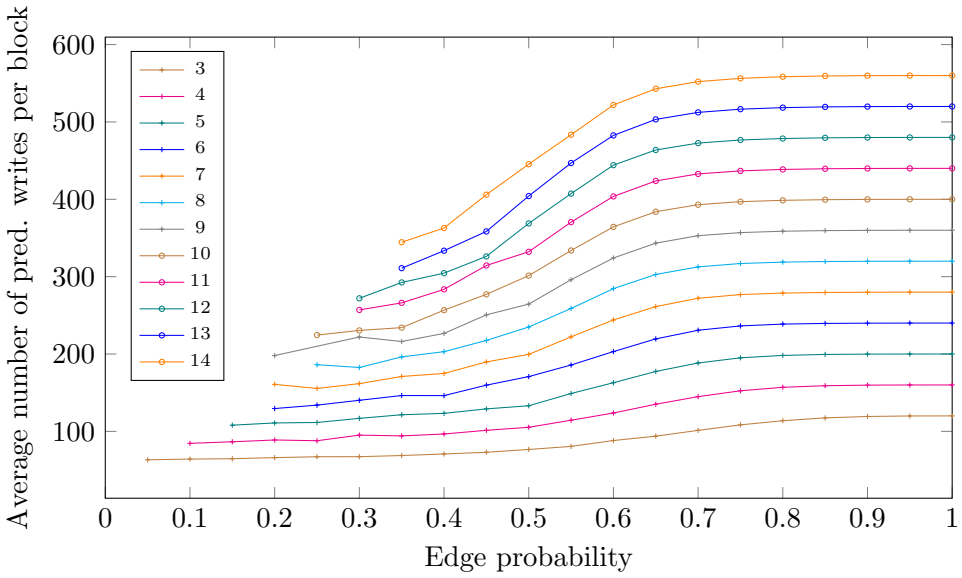

Figure 5.11: Average number of predecessors written to memory during each block search process when using GBFS. At lower edge probabilities, fewer predecessors are written because less of the block may be accessible. As the edge probability increases, the number of predecessors written approaches $2HB$, where H is the cluster height and B is the block width. The factor of two is due to the need to clear the ring buffer at the start of the search process.

per block-search is W_{pred}, then the average maximum acceptable memory write time t_{write} is given by

$$t_{\text{write}} = \frac{T_p}{W_{\text{pred}}} \tag{5.1}$$

For example, if the photonic cycle time is 1 ns (the same as that considered in Chapter 3), and the edge probability p is taken as the type-II fusion probability (75 %), then a block width of $B = 5$ (required to achieve a path depth of approximately 1000, from Figure 5.9) would lead to a maximum acceptable write time of $t_{\text{write}} = 5$ ps. This is an extremely tight timescale in which to achieve a memory write in a digital system.

Taking the FPGA used for the design in Chapter 3 as an example, memory switching times for distributed RAM are on the order of 0.5 ns [67] – two orders of magnitude too slow for the implementation of the GBFS algorithm discussed here. This means that it would likely not be feasible to implement

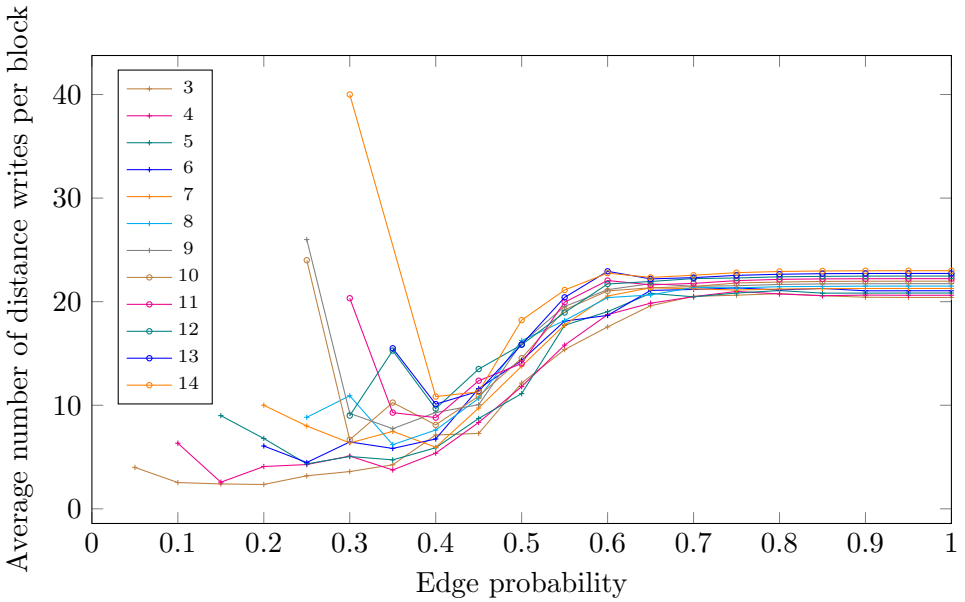

Figure 5.12: Average number of predecessors written to memory during each block search process when using IBFS. Compared to GBFS, the predecessor-write-overhead is substantially reduced. At higher edge probabilities, the average number of distance writes is equal to H (the cluster height, fixed at 20 in this experiment), and does not scale with the block width. This is because, on average, only the final column of the block is searched in each IBFS block search process. As the edge probability decreases, the memory overhead increases, because there is a chance that the search process will have to visit previously inaccessible columns in the inner part of the block, that has been made available by the addition of new edges in the final column.

the ring-buffer-based control system using this FPGA; a higher performance device would be required.

For the GBFS algorithm, in the limit of high edge probabilities (the asymptote visible at each block width in Figure 5.11), it is possible to provide a specific formula for the maximum acceptable latency, in terms of implementation parameters of the system:

$$t_{\text{write}} \approx \frac{T_p}{2BH}, \tag{5.2}$$

where H is the cluster-state height and B is the block width.

Figure 5.12 shows the number of predecessor writes for the IBFS algorithm. It is immediately clear that the number of memory writes is significantly reduced, and does not scale with the block width. If the IBFS can be modified

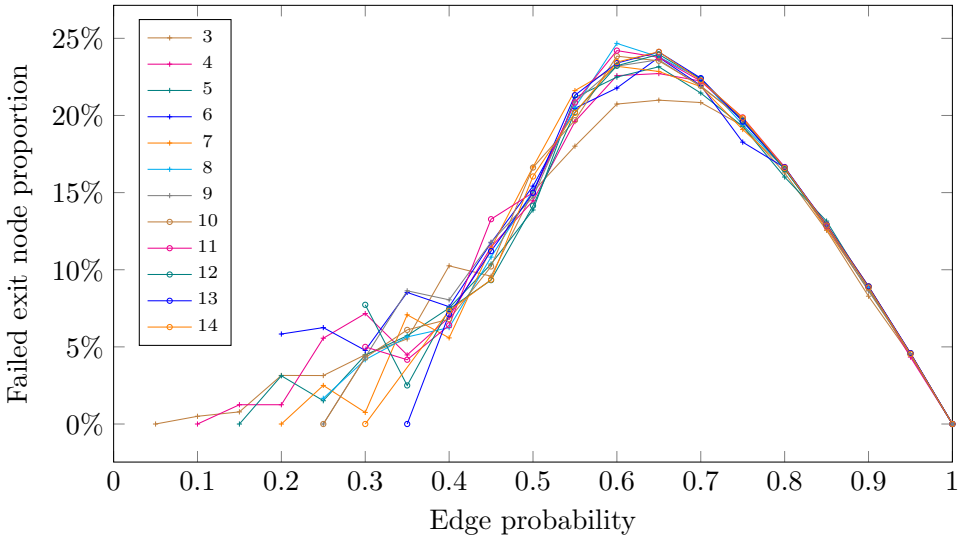

Figure 5.13: Average number of failed exit nodes per column using IBFS, as a proportion of the cluster height H, which determines how many paths must be pruned from the reverse path tree.

such that it also produces paths with an acceptable average depth, then this algorithm would be a substantial improvement upon GBFS.

Performing the same calculation as before, the IBFS algorithm would lead to a maximum acceptable memory write time of $t_{\text{write}} = 50\,\text{ps}$. This is still an extremely tight timing constraint, but a substantial improvement compared to the 5 ps of GBFS.

Key points

For the GBFS algorithm, a reasonable choice of algorithmic parameters combined with a photonic cycle time $T_p = 1\,\text{ns}$ leads to a target memory latency for writing ring-buffer entries of 5 ps. However, for IBFS, the corresponding memory latency is 50 ps. This order of magnitude improvement shows that algorithms like IBFS are much better candidates for control systems for IMBQC. The speed at which these control systems may have to operate underlines the importance of carefully analysing these timing constraints in the presence of a concrete specification for the algorithms involved in photonic quantum computing.

Figure 5.13 shows a further example of the type of analysis that can be

performed using an emulation system such as MBQCSIM. The graph shows the number of failed exit nodes, as a proportion of the cluster-state height H, which directly translates to the number of paths that need to be pruned from the set of potential path extensions. At high edge probabilities, very few of the exit nodes fail, because there are a very high proportion of edges, meaning that path extensions can be easily found. However, as the edge probability decreases below approximately $p = 0.6$, the decrease in the proportion of failed nodes is likely caused by a lack of paths through the block. These two effects compete to produce the maximum in Figure 5.13. At very low edge probability, near $p = 0.5$, the high variability is caused by the relatively few number of search processes which results in any paths at all (near the percolation threshold).

5.6 Conclusion

In the case of IMBQC, which involves significant algorithmic complexity, control system emulation is necessary before hardware design in order to investigate trade-offs between the performance of different types of algorithms. For example, algorithms such as IBFS are better candidates for implementation that GBFS, relaxing timing constraints on the photonic quantum computer by approximately an order of magnitude. However, thorough analysis is required to remove subtle issues from the behaviour of this algorithm in order to achieve the same performance as GBFS.

In emulating the system, we specified an implementation model based on a ring buffer. The main feature of this model is that it minimises copying of the data in the buffer when new columns of photons are added. It also provides a simple framework to begin the study of photonic control system emulation. An implementation model is necessary in order to define the scope of validity of the emulation, and provide a starting point for control system architecture design in the more complicated setting involving incomplete cluster states.

The emulation conducted in this chapter was achieved through the use of MBQCSIM, a modular, open source, reproducible, and tested C++ and python library for the investigation of control system implementations for photonic quantum computing. The implementations in MBQCSIM may be taken as the starting point for hardware designs, written in VHDL or Verilog, that implement the block-search based algorithms discussed in this chapter.

As we describe in the next chapter, MBQCSIM also performs the dynamic pattern generation and quantum simulation of one-qubit gates. The dynamic pattern generation is an important component of the control system, and must also be emulated. We leave this emulation as future work.

The FPGA used for the design in Chapter 3 is approximately an order of magnitude too slow for the implementation of the IBFS algorithm discussed here. True timing constraints relating to these algorithms can only be obtained using static timing analysis from an actual prototype hardware design, likely using a higher performance device. This chapter is intended to lay some of the groundwork for this kind of design.

Chapter 6

Dynamic measurement pattern generation and analysis of analog voltage noise

In the previous chapter, a number of algorithms for finding logical one-qubit paths through incomplete cluster states were presented and emulated, for the purpose of placing timing constraints on digital implementations of these algorithms. This analysis was performed by the library MBQCSIM, which was designed for the purpose of analysing implementations of IMBQC (MBQC on incomplete cluster states).

In this chapter, we describe the other half of the system required for the simulation of one-qubit gates in IMBQC: how the measurement pattern along a path is dynamically generated, and how this pattern may be implemented within the context of the ring-buffer model (see Appendix 5.1.3). We describe local rules for storing the measurement pattern in the secondary data of the ring buffer, and describe how to simulate and verify the functionality of the measurement pattern. This lays the groundwork for an emulation of these algorithms for the purpose of deriving timing constraints. However, we do not perform this emulation here; instead, we show how the simulation and verification can be used to investigate how modulator voltage noise affects the fidelity of a one-qubit identity pattern, found using GBFS, as a function of time (path depth).

This kind of simulation may be seen as the emulation of a different realistic aspect of the system. Throughout the thesis, we have focused on how the algorithms involved in MBQC impose timing constraints on the photonic quantum computer. However, in this chapter, we show how non-idealities of the analog electronics in the system may impose other kinds of constraints on the performance of the quantum computer; in particular, limiting the fidelity of one-qubit operations. The techniques presented here may be extended to the analysis of other proposed architectures for photonic quantum computing, particularly involving error correction [16], in order to investigate how realistic errors in the control system electronics affects the performance of the system.

6.1 Effect of non-idealities in MBQC circuits

The advantage of having a working pattern verification system, which is the basis of the results in Section 6.6, is the ability to investigate how various non-idealities in the system affect the operation of the quantum computer. These non-idealities may be introduced by modifying parts of the system to behave slightly incorrectly, and then seeing how this affects the fidelity of the output state from the system. Example non-idealities include:

- **Analog voltage noise** that causes slightly wrong basis settings in modulators. This type of noise may arise from electrical interference or crosstalk in the analog electronics driving the modulators. If the modulators cause slightly inaccurate measurement bases, then the measurement pattern will not quite produce the right result.

- **Photon loss** that may lead to loss of entanglement in the cluster state, or an inaccurate record of the entanglement present in the ring buffer.

- **Detector errors** that lead to inaccurate measurement outcome information, and consequently errors in the processing of adaptive measurement settings and byproduct operators.

We take the study of the first non-ideality, modulator noise, as the goal for this chapter. The other two sources of errors are beyond the scope of this introductory analysis, because they introduce the need for error correction in either the measurement patterns, or fault tolerance in the cluster-state generation. These effects require a redesign of the search algorithms presented

in Chapter 5, and potentially require the consideration of a different type of architecture for photonic quantum computing [16]. However, these non-idealities can be addressed using the same methods we present in this chapter – by designing an emulation system for the implementation in question, simulating the quantum output resulting from the implementation, and comparing this output with the known-true quantum state in the face of various different non-idealities.

6.2 One-qubit measurement patterns in incomplete cluster states

We begin by showing the measurement patterns which are implemented in MBQCSIM. Compared to the simple measurement pattern for the one-qubit gate in Figure 2.2, it is necessary to take account of two new features.

Firstly, the pattern is laid out on a path through a cluster state (found using an algorithm such as those described in the previous chapter). However, this path may still be connected to other cluster qubits around the path, via entanglement edges. These qubits must be removed as part of the measurement pattern, and we refer to them as cut-out qubits. The removal entails measuring the qubits in the Z-basis, and using the measurement outcome to appropriately update byproduct operators for the path.

Secondly, the pattern is considered to be arbitrarily long, rather than having a fixed length of four. This means that it is no longer necessary to consider commutation corrections, which account for the effect of placing the fixed length patterns one after the other. Instead, the commutation correction is incorporated into the running calculation of the byproduct operators. As part of this process, the adaptive measurement setting calculation is refactored so that it depends only on byproduct operators, and not directly on measurement outcomes. This leads to a pattern, summarised diagrammatically in Figure 6.3, that is in a form suitable for implementation by a control system designed to realise one-qubit in IMBQC.

We discuss the effect of cut-out qubits in Section 6.2.1. In Section 6.2.2, we describe the measurement pattern for the one-qubit gate in detail, including how to calculate byproduct operators and adaptive measurement settings.

6.2.1 Cutting out qubits around the path

Before deriving generalised measurement patterns in the presence of cut-out qubits, it is important to understand how cutting out qubits affects a cluster state. As described in Appendix A.1, a cluster state $|\phi_C\rangle$ is described by a set of eigenvalue equations, one for each cluster qubit a:

$$\left(X_a \prod_{b\sim a} Z_b\right) |\phi_C\rangle = (-1)^{\kappa_a} |\phi_C\rangle, \tag{6.1}$$

The crucial term of the equation for the purposes of this section is $\kappa_a \in \{0, 1\}$, which is a number assigned to each cluster qubit. For a cluster state on N qubits, there are 2^N choices for κ_a across all the cluster qubits. All these states are equally usable as the basis of MBQC, but in order to simplify the analysis, it is usual to take the state where $\kappa_a = 0$ everywhere, so as to avoid κ_a appearing in all the equations [20].

However, if the analysis involves cutting out cluster qubits, it is not possible to ignore κ_a. This is because measuring a cluster qubit a in the Z-basis, resulting in outcome m, has the effect of flipping the value of κ_b on neighbouring qubits $b \sim a$ when $m = 1$ [20]. As a result, a cluster state where $\kappa = 0$ everywhere will become a cluster state with some non-zero κ_a, when the Z-measurements corresponding to these cut-outs are performed. More precisely, from the point of view of the remaining qubit a, its value of κ_a is given by the following expression:

$$\kappa_a = \left(\sum_{b\sim a,\, b\in\mathcal{Z}} m_b\right) \text{ modulo } 2 \tag{6.2}$$

where $\mathcal{Z}$ is the set of cluster qubits that have been measured out. The formula may be interpreted as stating that κ_a is obtained by XORing together the measurement outcomes m_b of all its neighbouring cut-out qubits b.

Key points

As we show in detail in Appendix A.2, the effect of cutting out qubits around an on-path qubit a is accounted for by incorporating the parameter κ_a into the (byproduct operator calculations of the) measurement pattern in all locations where the measurement outcome from a is used.

The following examples illustrate how this applies to simple situations

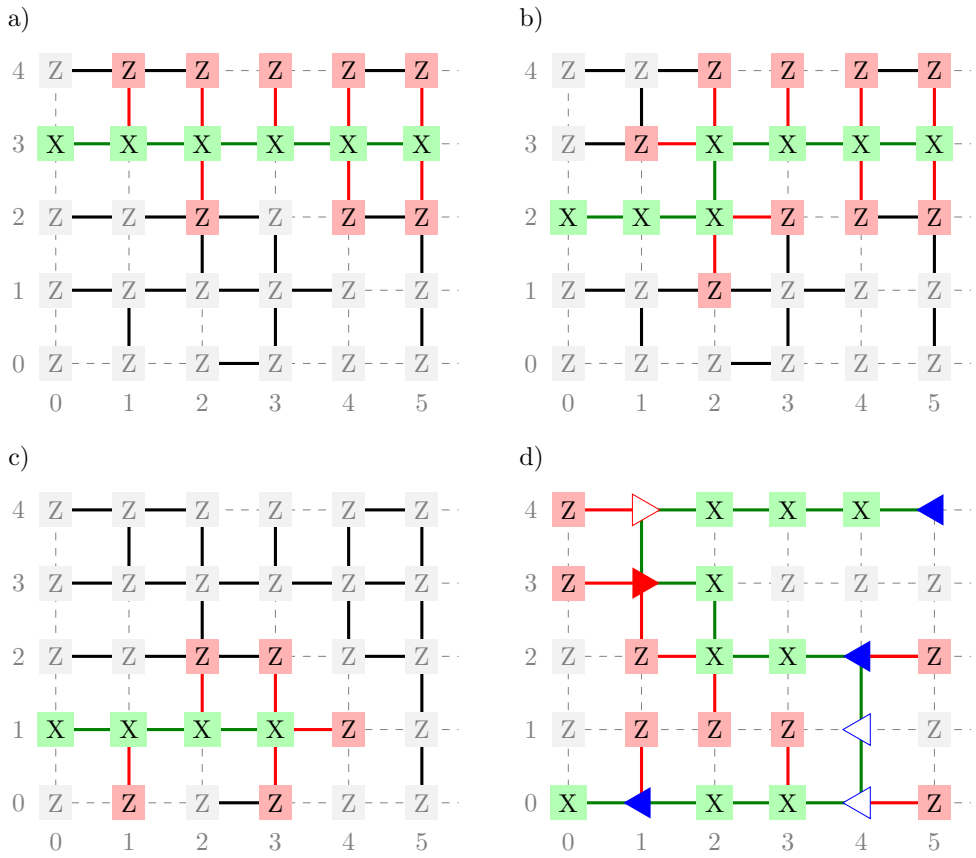

Figure 6.1: Diagram showing different types of cluster qubit cut-outs that occur along a path pattern. The path is shown in green, and cut-out qubits are shown in red. a) A straight path is realised by cutting out at most two cluster qubits (above and below) by measuring them in the Z-basis. Their measurement outcomes are XORed with the κ_a value of the on-path qubit. b) A corner is realised by cutting out at most two neighbouring cluster qubits (above or below and to the left or right). c) A path is terminated by cutting out qubits on three out of four sides. d) An example of a general path which backtracks and involves the different kinds of cut-outs shown in a) and b). In addition, some example right-nodes and left-nodes are depicted using arrows pointing right and left respectively. (These are still X-measurements.)

- **Straight path.** If the intention is to cut out a straight path (for a single logical qubit), then a computational basis measurement needs to be made above and below each cluster qubit in the wire. The results of these measurements are XORed together, and the result is used to compute the κ_a on the remaining cluster qubit. This is shown in Figure 6.1a.

- **Corner.** To make a wire turn a corner (for example, from horizontal to downwards), there are still only two adjacent measurements (one above and one to the right or left), as shown in Figure 6.1b. These measurement outcomes should be XORed together to produce κ_a.

- **Path terminations.** There is limited need for three- and four-input XOR operations. Three inputs would correspond to a wire that terminates (which might occur at the end of a circuit). Four-input XOR corrections would only be needed to cut out a single isolated qubit (where a qubit has been removed on all four sides). An example is shown in Figure 6.1c.

Important subtleties arise in the simulation and verification of measurement patterns involving cut-out qubits. For example, Figure 6.1d shows an example of a path that backtracks on itself. Since each column is measured one at a time, column zero is measured first, and contains both a path qubit $(0, 0)$, and two cut-out qubits $(0, 3)$ and $(0, 4)$. As we will describe in Section 6.2.2, the cut-out qubits contribute to the byproduct operators – however, they only contribute to a "future" part of the path, starting at path index 10[1]. In order to verify the measurement pattern up to cluster qubit $(1, 0)$, it is necessary to ignore these contributions. The method for doing this is described in Section 6.3.2.

In addition, Figure 6.1d shows some example maximal left-nodes, which are used as the verification points for the measurement pattern (these are shown in the diagram as filled blue arrows pointing left). There is one maximal left-node per column; however, only the left-nodes in columns one, four and five are shown, to avoid cluttering the diagram. The function of left-nodes is to ensure that a valid verification of the measurement pattern is possible. This is described in detail in Section 6.5.

6.2.2 One-qubit gate along an arbitrary path

Figure 6.2 shows the generalisation of the arbitrary one-qubit gate to a measurement pattern that can be arbitrarily long, does not include commutation corrections, and derives the adaptive measurement settings directly from the byproduct operators. This pattern is derived in Appendix A.2.4.

[1]Throughout this chapter, we use the term "path index" to refer to the zero-indexed position of a cluster qubit along a path.

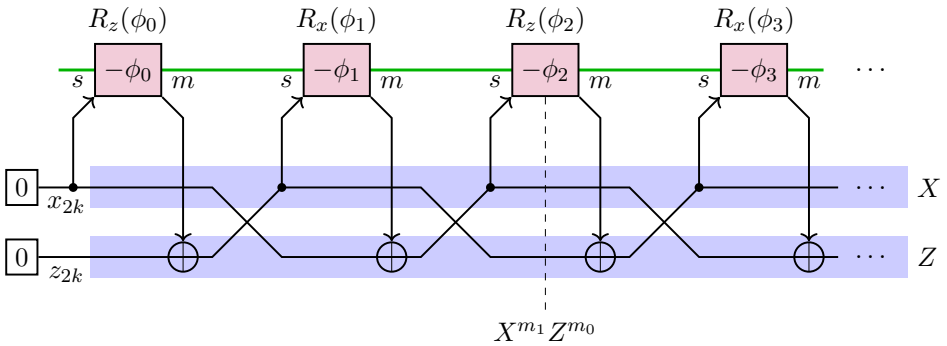

Figure 6.2: Diagram showing the arbitrary-length one-qubit-gate measurement pattern on a linear cluster state. The figure is a generalisation of Figure 2.2, showing how the adaptive measurement setting is derived from the byproduct operators at each measurement round (column). There is no gate boundary, and therefore no commutation correction. The byproduct operator lines z_{2k} and x_{2k} are arranged so that byproduct operators valid for both the odd and even terms may be read by drawing a vertical dashed line through the desired final (unmeasured) qubit, and reading off the values that fall in the blue regions, as shown in the example for $n = 2$. This is to facilitate the simple implementation of the verification scheme discussed in Appendix C, which must be valid at an arbitrary point along the path.

The most important feature of this pattern is that each term in the byproduct operator pair (x, z) is only modified every other cluster qubit. As a result, we refer to it as (x_{2k}, z_{2k}), following the notation used in Appendix A.2.4. The pattern realises a string of alternating R_z and R_x rotations, that may be used to realise any one-qubit gate. For example, to realise the identity gate, all the ϕ_n are set to zero (in this case, the adaptive measurement settings are not important, because they flip the sign of ϕ_n).

The pattern is only "valid" when it terminates on even values of n, for example, if qubit $-\phi_2$ remained unmeasured. These are the locations where the x_{2k}, z_{2k} lines on the left coincide with the X, Z horizontal shaded regions. If the pattern terminates on an odd qubit, it is still possible to interpret the pattern, by using the shaded X and Z regions to obtain the byproduct operators. In this case, the pattern realises an additional H gate, as described in Appendix A.2.4 (see Equations (A.27) and (A.28)). The ability to obtain the known-true state of the pattern at any qubit position is important for the verification discussed in Section 6.5.

As discussed in the previous section, the measurements from cut-out qubits

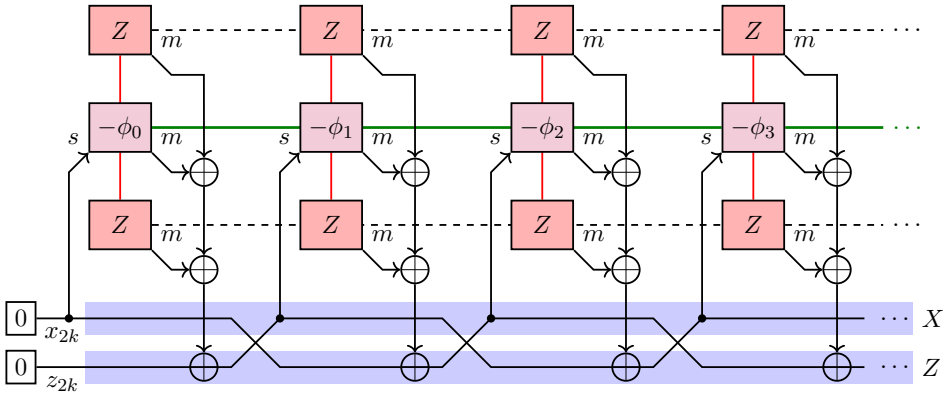

Figure 6.3: Diagram showing the generalisation of Figure 6.2 to the case where a straight path is embedded in a 2D cluster state. As in the previous case, the byproduct operators swap every other column, showing how to interpret the byproduct operators for measurement patterns whose final unmeasured qubit lies in an odd column. This figure is also representative of an arbitrary path through an arbitrarily connected cluster state. The byproduct operators display the same behaviour, alternating with incrementally increasing path index n. Measurement outcomes for any cut-out cluster qubits adjacent to a path qubit q_n are added to the measurement outcome from q_n, before this is added to the byproduct operator z_{2k}.

are incorporated into the measurement pattern in locations where the outcome from the corresponding on-path qubit is used. Figure 6.3 shows how this generalisation is made for the straight pattern discussed in the previous section (with cut-out qubits above and below). In general, all that is required is to add κ_a to the corresponding byproduct operators which are modified by the outcome from a. This diagram forms the basis for all one-qubit measurement patterns that can be implemented using MBQCSIM.

Key points

Figure 6.2 shows the measurement pattern that forms the basis for any one-qubit gate realised along a path through a cluster state, and Figure 6.3 shows what modifications are necessary to account for cut-out qubits. The alternating byproduct operators lines make it possible to read out at each position what byproduct operator values should be used for interpreting the state of the measurement pattern.

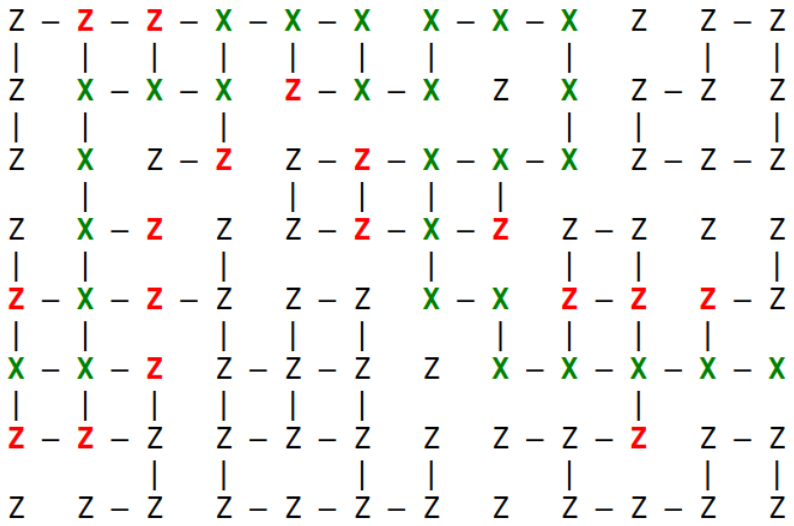

Figure 6.4: Program output showing the generated measurement pattern for the identity gate along the same path shown in Figure 5.5. All measurements along the path are performed in the X basis (these would be replaced by general xy-plane measurements for an arbitrary one-qubit gate). Any qubits connected to the path by edges must be removed by cutting them out, and incorporating those outcomes into the byproduct operators (shown in red). Other cluster qubits (shown in black) do not contribute to the measurement pattern, and may be measured in the Z basis (their outcomes are discarded).

In Section 6.3, we discuss how this measurement pattern is stored in the ring-buffer model, discussed in Appendix 5.1.3. We present a scheme of local measurement-pattern rules which enables the encoding of the pattern locally with the logical qubits, in a form that could easily be implemented in a control system design.

6.3 Dynamic measurement pattern generation in MBQCSIM

This section describes the system used by MBQCSIM for dynamically generating a measurement pattern for a path extension defined in Section 5.3. Before showing how the measurement pattern is generated and encoded, we discuss some specific restrictions that implementations may impose on the set of valid measurement patterns, including some which may affect the search process described in the previous chapter.

We will then discuss in detail how the measurement pattern is generated along the path extension, and how the rules for bases, adaptive measurement settings, and byproduct operators are stored in secondary node data. This data is analogous to the program word introduced in Section 3.2.4 as part of the FPGA design for one-qubit gates and CNOT gates.

6.3.1 Restrictions imposed by implementation considerations

Not all paths through an incomplete cluster state may be used as the basis for one-qubit-gate measurement patterns, in the model that we are considering (the block-search algorithm shown in Figure 5.1). In this model, a new column of photons is generated every photonic clock cycle and added to the right side of the block, and a column of photons is measured out according to the dynamically-generated measurement pattern on the left side of the block. In addition, we impose the restriction that all the measurements in the left-most column happen at the same time, to minimise the timing overhead associated with the measurement process in each photonic clock cycle.

It follows that there cannot be any (vertical) measurement setting dependencies within a single column[2], because this would entail multiple rounds of measurement per column. (The dependent cluster qubits must be measured after the outcomes from the independent qubits have been obtained and processed.) This type of multi-round measurement would require an increase in the clock rate of the digital system, due to the need to process measurement settings within a particular fraction of the photonic clock period.

In a similar manner, it is not possible to allow patterns where the basis setting of a cluster qubit a depends on the outcome from a cluster qubit b that is strictly to the right of a, because a will be measured before b. This can happen for paths which backtrack into previous columns, as shown in Figure 6.1d. To cope with this, it would be necessary to dynamically re-order column measurements based on the measurement pattern being implemented, which would introduce new algorithmic overheads in the implementation of the control system.

If strict column ordering is maintained, it is necessary to lay measurement patterns with non-trivial adaptive measurement settings on straight horizontal sections of path, and ensure that vertical sections and backtracks are used

[2]That is, a pair of qubits a and b in the same column, where the measurement outcome from a is used in the calculation of the adaptive measurement setting for b.

only for Clifford operations (which do not require adaptive measurement settings [58]). However, depending on the characteristics of the path (in particular, in cluster states with a lower edge probability p), there may not be long horizontal sections of path available, or they may occur in the wrong places. In that case, it may be necessary to incorporate the requirements of the measurement pattern generation into the search process itself, which would complicate the algorithms described in Chapter 5.

We will avoid considering any of these issues by considering only the identity pattern, where all on-path measurements are in the X basis, and there are no adaptive measurement settings. Currently, only the identity pattern is implemented in MBQCSIM. However, we will still describe general local measurement-pattern rules that cover any one-qubit gate pattern. That way, all that remains is a choice for how to deal with the implementation constraints described in this section.

Key points

Not all one-qubit measurement patterns are possible in the model shown in Figure 5.1. In particular, adaptive measurement settings must only depend on measurements from columns strictly to the left. This means that non-trivial one-qubit gates must not be laid on vertical sections of path, or path segments that backtrack. We focus on the identity gate in this chapter, so as to avoid this additional complexity.

Next steps

MBQCSIM could be extended to support arbitrary one-qubit gate patterns by laying out non-trivial pattern sections on horizontal sections of path of a suitable length. This is one possible scheme for avoiding the issues described in this section.

6.3.2 Local measurement-pattern rules

The measurement pattern is encoded in a set of local pattern rules, which are designed to be stored in the ring buffer. The need for local pattern rules follows from the desire to simplify the column measurement implementation as much as possible. In making the measurements, it is necessary to step

Data	Meaning
Basis flag FB	Indicates either Z-measurement or xy-plane measurement
θ	Base angle (only for xy-plane measurement)
R_s	Adaptive measurement setting rule, (r, s)
R_b	Byproduct operator update rule, (r, s)

Table 6.1: Table showing the secondary data required for storing the measurement pattern. The pattern is stored as a set of local rules (one per cluster qubit), which completely specify how each cluster qubit should be measured, and what should be done with the measurement outcome. The pattern rules are generated after a path extension has been found (see Figure 6.5). This dynamic pattern generation should be emulated following the example of Chapter 5, to find how many ring-buffer memory operations are necessary for a hardware implementation of the system.

through each cluster qubit in the column, set its measurement basis (including adaptive measurement setting), make the measurement, and then use the outcome to update byproduct operators. This process is simplified if all these measurements can be made in parallel, and each is fully controlled by information that is local to the cluster qubit being measured (and does not involve, for example, the collection and processing of information stored at multiple nodes). By making each measurement use identical information, the speed of the measurement process in a digital implementation may be maximised, by ensuring that no measurement takes longer than any of the others.

Key points

The measurement pattern is defined by local pattern rules, whose purpose is to provide a simple, parallelisable, means for digital hardware to realise the measurement pattern. By keeping the rules simple, and excluding complex algorithms (involving traversal of multiple nodes in the ring buffer), the speed of the measurement step may be maximised, and the measurement control system implementation may be separated as much as possible from the dynamic pattern generation system.

The secondary data required for storing local pattern information is summarised in Table 6.1, and described in the following sections.

Byproduct operator update rules

The measurement pattern rules make reference to the pair (x, z), which is the running value of the byproduct operators (x_{2k}, z_{2k}). It is updated as the measurement pattern is evaluated by XORing measurement outcomes into either the x or z term in the pair, using rules defined here.

As shown in Figure 6.3, the measurement outcome m from an on-path qubit a_n is XORed into x or z depending on whether n is even or odd:

$$(x, z) \mapsto \begin{cases} (x, z \oplus m) & \text{if } n \text{ is even,} \\ (x \oplus m, z) & \text{if } n \text{ is odd.} \end{cases} \tag{6.3}$$

This rule is stored as a pair $R_b = (r, s)$ which is either $(1, 0)$ or $(0, 1)$, depending on whether the outcome should be added to x or z respectively. The value of this pair for a qubit a_n is denoted $R_b(a_n)$.

For each on-path qubit a_n whose measurement outcome m is added to a term in (x, z), the measurement outcome from any adjacent cut-out qubit must also be added to that same term. This leads to the rule that the outcome m from a cut-out qubit b may be added to either of the terms (x, z) multiple times, because b may be adjacent to multiple on-path qubits (see, for example, cut-out qubit $(1, 2)$ in Figure 6.1d). To account for this, each cut-out qubit stores a pair $R_b = (r, s)$, which expresses the net effect of this cut-out qubit on the byproduct operators, when the measurement outcome from this cut-out is m:

$$(x, z) \mapsto (x \oplus mr, z \oplus ms). \tag{6.4}$$

This pair is obtained for a particular cut-out qubit b by adding (pairwise modulo-2) all the values $R_b(a)$ for on-path qubits a ($a \in P$) that are adjacent to b ($a \sim b$):

$$R_b(b) = \bigoplus_{\substack{a \sim b \\ a \in P}} R_b(a). \tag{6.5}$$

Key points

The byproduct operator rules $R_b(a)$ define how the measurement outcome m from a affects the running value of the byproduct operators (x, y). Only qubits on the path, and cut-out qubits around the path, have non-trivial byproduct operator rules – other qubits use $R_b = (0, 0)$.

Measurement basis angle and dependency rules

Each qubit in the cluster state is either measured in the Z-basis (if it is a cut-out qubit, or if it is not directly connected to the measurement pattern), or the xy-plane, for all other measurements. A flag FB is stored in each cluster qubit node to specify in which basis it is measured.

The measurement basis angle for each xy-plane measurement is stored as a base angle θ, and an adaptive measurement setting s. The base angle is shown inside the filled boxes in Figure 6.3, and the measurement setting is shown as the input in the bottom-left corner of each square. For example, the second on-path qubit in Figure 6.3 has $\theta = -\phi_1$, and s is the current value of the byproduct operator z_{2k} just before the measurement of that qubit. The base angle is a static property of the pattern (it does not depend on any measurement outcomes or byproduct operators), and relates to what R_x and R_z rotations are realised by the pattern.

The adaptive measurement setting rule $R_s(a_n)$ is stored as a pair (r, s), which describes how to calculate the adaptive measurement setting s from the current values of the byproduct operators (x, z):

$$s = R_s(a_n) \cdot (x, z) = rx \oplus sz. \tag{6.6}$$

Key points

The rule $R_s(a)$ describes how to calculate what adaptive measurement setting s to use for the measurement of the qubit a from the current value of the byproduct operators, (x, z). Compared to the one-qubit pattern in Figure 2.2 in Chapter 2, the calculation is simplified, because it does not involve commutation corrections or measurement outcomes.

Generating the pattern rules

The pattern rules described above are generated along a path extension (see Section 5.3) by traversing the path extension once, from the starting right-node to the ending right-node, visiting all the cut-out qubits adjacent to each on-path qubit as it goes along. At each on-path qubit node a_n, the local rules for this qubit are calculated, and written to the ring buffer . Then, the algorithm iterates over all the adjacent cut-out qubits b surrounding a_n, and

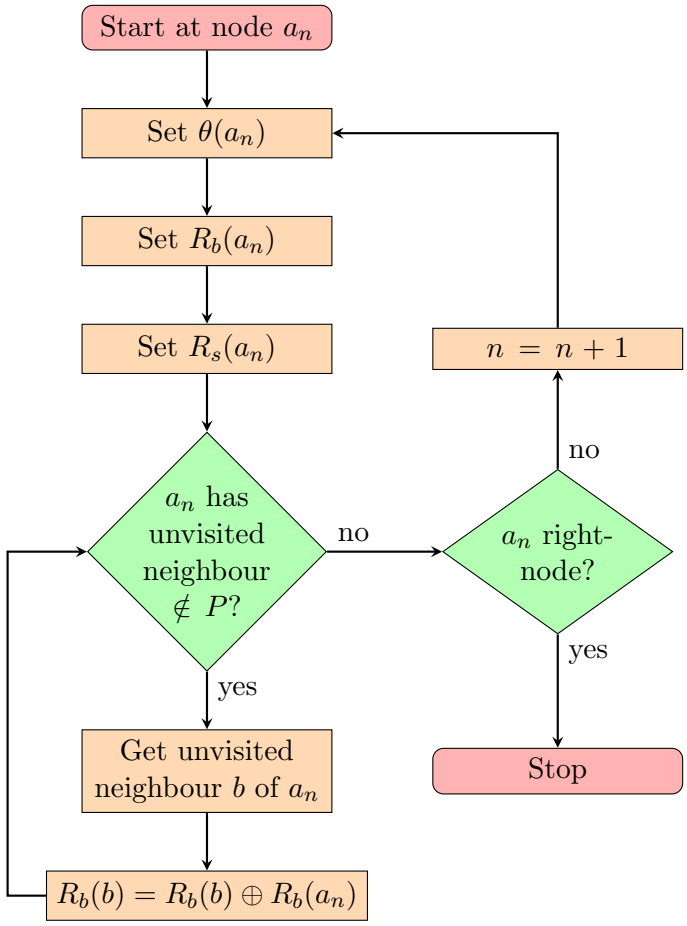

Figure 6.5: Algorithm for generating local pattern rules on a path extension (up to the next right-node). The path extension is traversed once, forwards. For the identity pattern, all on-path measurements are X: $\theta(a_n) = 0$, $R_s(a_n) = (0, 0)$, and $R_b(a_n) = (n+1 \bmod 2, n \bmod 2)$. (Note that the "unvisited" status of the cut-out qubits only applies to the innermost loop; any given cut-out qubit may be visited more than once from different on-path qubits.)

accumulates the same byproduct operator rules used for a_n to this (r, s) pair for the current cut-out qubit b. The algorithm is shown in Figure 6.5.

The algorithm described here must be performed as part of the implementation of IMBQC, and should therefore be emulated to understand its contribution to the overall cost of the block search process in Figure 5.1. It is likely that this algorithm could be incorporated into the path extension algorithm (Figure 5.4), and the number of operations involved could be easily counted in MBQCSIM. The scheme presented here is compatible with the

ring-buffer model. As a result, MBQCSIM could easily be extended to count how many ring-buffer memory operations would be involved in a hardware implementation of this part of IMBQC.

Key points

The algorithm to generate the pattern rules involves traversing all the nodes in a path extension, and visiting all the cut-out qubits connected to each on-path qubit. The algorithm calculates the local pattern rules and stores them in secondary data in the ring buffer, for use during the measurement round. This algorithm constitutes the main process in the dynamic measurement pattern generation.

Next steps

The generation of the pattern rules must be implemented as part of a control system digital design, and should therefore be emulated, following the example set out in Chapter 5, to estimate any timing constraints that it imposes on the operation of the photonic quantum computer.

In the remainder of the chapter, we discuss the simulation of the rules discussed above, how the simulation can be verified, and how this may be used to analyse the effect of analog voltage noise in the implementation of IMBQC.

6.4 Simulating generated measurement patterns

Cluster-state simulation replaces the quantum part of the system: the entanglement of a set of dual-rail encoded photonic qubits in a 2D arrangement, and the measurement of these photons one column at a time. The advantage of writing a simulator is that it provides a concrete test of the interpretation of the local pattern rules described in the previous section, without needing a physical quantum computer. Ultimately, the part of the simulator responsible for interpreting the pattern could form the basis for the digital system which controls the photon measurement settings via modulator voltages. This system could also be emulated along the lines described in Chapter 5.

Key points

Writing a simulator for dynamically generated measurement patterns provides a concrete way to check the validity of the pattern rules, and forms a basis for control system implementation of the measurement process.

6.4.1 Adapting the cluster-state simulator to IMBQC

As discussed in Chapter 4, it is not possible to store the quantum state of the entire cluster state all at once in a simulation, because that involves more entangled qubits than can be reasonably simulated[3]. However, we showed how to construct a recycling quantum simulator in Section 4.1.1 that only ever holds two columns of the cluster state in memory, thereby enabling the simulation of a cluster state that is limited in height but not limited in width. On a computer that can simulate a state vector of 28 qubits, the maximum cluster-state height that can be simulated using this method is 14.

In this case, some simple modifications must be made to the simulation procedure outlined in Section 4.1.1. Firstly, in step three, where the entanglement links are added, CZ gates are now only applied where there are edges in the cluster state. Secondly, in step four of Section 4.1.1, measurement bases were set and outcomes interpreted by the program word; now the column measurement is performed by interpreting and executing the local pattern rules.

The MBQCSIM simulator keeps a running total of the byproduct operators (x, z) for the measurement pattern being simulated, and measures a column by traversing the column from bottom to top, performing the following three steps at each cluster qubit a_n:

1. **Construct the measurement basis.** First, check whether a computational basis measurement or an xy-plane measurement is required using the flag FB (Table 6.1). In the latter case, form the angle from the x-axis $\phi = (-1)^{R_s(a_n)\cdot(x,z)}\theta(a_n)$ using the pattern rules $\theta(a_n)$ and $R_s(a_n)$.

[3] A rule of thumb is that the maximum number of qubits simulable on a laptop is 28. This would limit the cluster-state size to, e.g. 4x7.

2. **Measure out the cluster qubit,** in the basis defined in the previous step, recording the measurement outcome m.

3. **Update the byproduct operators,** as $(x, z) \mapsto (x \oplus mr, z \oplus ms)$, using the outcome m and the byproduct update rule $R_b(a_n) = (r, s)$.

The output from the simulation of a one-qubit gate described above is a one-qubit state $|\psi_o\rangle$, from the final unmeasured qubit[4], along with a pair of byproduct operators (x, z) which determine how to obtain the corrected output state $|\psi_c\rangle = (X^x Z^z)|\psi_o\rangle$. This can be compared with the true (known) output state $|\psi_t\rangle$, obtained from a one-qubit simulation of the logical operation being realised by the measurement pattern, to establish that the overall measurement pattern worked.

This scheme allows the output from the measurement pattern to be verified. However, it does not allow the measurement pattern to be verified column-by-column. This finer-grained verification is important for understanding how errors (due to noise) accumulate throughout the measurement pattern, and also for isolating and correcting bugs in the execution of the measurement pattern simulation. For example, in a cluster state of width 100 whose output state is wrong, it is hard to find exactly where the error occurred without being able to step through the simulation one column at a time. In Section 6.5, we describe how column-verification is implemented in MBQCSIM.

Key points

Simulating one-qubit gate patterns along paths in IMBQC requires minimal changes to the simulator described in Chapter 4; the program word which specifies the measurement patterns is replaced by local pattern rules. However, verification of the simulation column-by-column is not trivial, and forms the subject of Section 6.5. However, this type of verification is necessary in order to aid the debugging of long measurement patterns, and provide a means to analyse errors due to noise as the simulation progresses.

[4]The final qubit is obtained by measuring all but one qubit in the final column of the cluster state. The unmeasured qubit is the highest-index on-path qubit in that column.

In the sections below, we describe how errors due to noise in the modulator voltages are modelled in the simulation process.

6.4.2 Simulating analog voltage noise in modulators

There are many sources of potential errors inherent in prospective implementations of photonic MBQC. For example, there are (discrete) errors introduced by photon loss, which are often discussed. In addition, there are continuous sources of errors, such as spectral impurity in photon sources (where identical photons are in fact slightly distinguishable) [86]. The type of continuous noise of interest here is due to the inability of an analog electronic control system to set exactly the right measurement settings during execution of the measurement pattern.

As discussed in Section 2.2.3 (see Figure 2.4), we assume that two voltage-controlled modulators in the measurement block (M_1 and M_2) set the xy-plane or Z-measurements necessary to realise the measurement pattern. If noise is present in the analog input signal to the modulator, then they will generate the wrong phase shift, which will cause a slight inaccuracy in the basis measurement.

We assume that the input voltage is subject to additive Gaussian white noise, with mean $\mu = 0$, and a standard deviation σ (Volts). White noise is characterised by constant power spectral density [87], meaning that the noise is not correlated in time. Each sample of the noise (one for each modulator and cluster qubit measurement) is an independent random variable $e \sim N(\mu, \sigma)$. The modulator voltage is given by $v = v_d + e$, where v_d is the desired (exact) voltage.

This noise may model general electrical interference in the analog system, with a magnitude that can be set using σ. MBQCSIM also supports settings non-zero μ, which represents a systematic error in the voltages. This may correspond to a true systematic offset, or may also arise from the combination of ringing in the modulator voltage and the precise time at which the photon passes through the modulator.

Like all other random behaviour in MBQCSIM, the voltage noise e generated by the program is based on a noise seed S_n, shown in Figure 5.3 and described in Section B.2. Due to the non-guaranteed repeatability of the `std::normal_distribution` in C++, MBQCSIM uses a custom implemen-

tation of the normal distribution, which is described in more detail in Appendix B.4.

Key points

Voltage noise in the modulators that set the measurement basis is modelled by adding Gaussian white noise with a particular standard deviation σ to the intended (exact) voltage. This causes slight errors in the realisation measurement pattern, which can be quantified by the verification procedure laid out in Section 6.5 below.

6.5 Verification of the simulation

Column verification is a desirable feature in a complex MBQC simulation, because it enables finding the precise location of subtle bugs (which may be due to a combination of pathfinding errors, pattern generation issues, or simulation problems), and also provides the ability to view the evolution of errors (due to noise) as a function of time in the simulation. This column verification involves comparison of the state of the measurement pattern with the known-true state once per column, instead of once in the final column of the pattern.

However, this column verification is not trivial to achieve, because there is not necessarily any "state of the measurement pattern" available at an intermediate point in the realisation of the pattern, which is compatible with the full measurement pattern M, that does not involve modifying the full the measurement pattern. This incompatibility is due to the need to consider the column in question, x, to be the end of the pattern, which involves removing all its entanglement edges with the next column $x+1$. This removal of edges constitutes a change in the measurement pattern M, resulting in a new pattern M' (which terminates at column x). In this new "final column", a terminating qubit T is selected, which plays the role of the final (unmeasured) qubit in the full measurement pattern M. The state after measuring out all but this final qubit is taken as the output from column x.

It would be convenient if this change to the measurement pattern could be ignored; but it cannot be, due to the incompatibility between measurement outcomes from measurements in M and measurements in M'. This incompat-

ibility arises from differing correlations in the cluster state due to the different entanglement edges.

In this section, we describe the trade-off made by MBQCSIM in order to realise column verification, in the face of the issue described above. The method results in a scheme that offers verification of the measurement pattern M', which is a proxy for the measurement pattern M, based on the compromise that the value of the byproduct operators obtained in the measurement of column x in M' will be different from the byproduct operators obtained when column x is measured in M. We call these differing byproduct operators "verification byproduct operators". The verification is valid, because the verification byproduct operators at column x are consistent with the byproduct operators and measurement outcomes from the realisation of M at all columns strictly to the left of x. The compromise is therefore that the verification scheme checks all columns to the left of x (not including x), so that the verification is one step behind the simulation itself.

First, we describe how to choose the terminating qubit T, in terms of the left-nodes of the path P along which the one-qubit measurement pattern is defined. Then, we describe the necessity for verification byproduct operators in detail, and show how they are calculated.

Key points

To verify the simulation at column x, it is necessary to treat x as the final column, be removing entanglement links to column $x + 1$. This change to the entanglement causes a discrepancy in the measured outcomes between the main simulation and the verification, which is addressed below by allowing the byproduct operators in the verification to be different from the main simulation.

6.5.1 Calculating left-nodes

The section describes how the terminating qubit T is selected in each column of the verification scheme: it is selected to be a maximal left-node. Left-nodes are defined analogously to right-nodes (see Section 5.3) as follows:

Definition 2 (Left-node)**.** *Given a path P comprising a sequence of edge-connected nodes $((x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N))$ in a 2D cluster state, a left-*

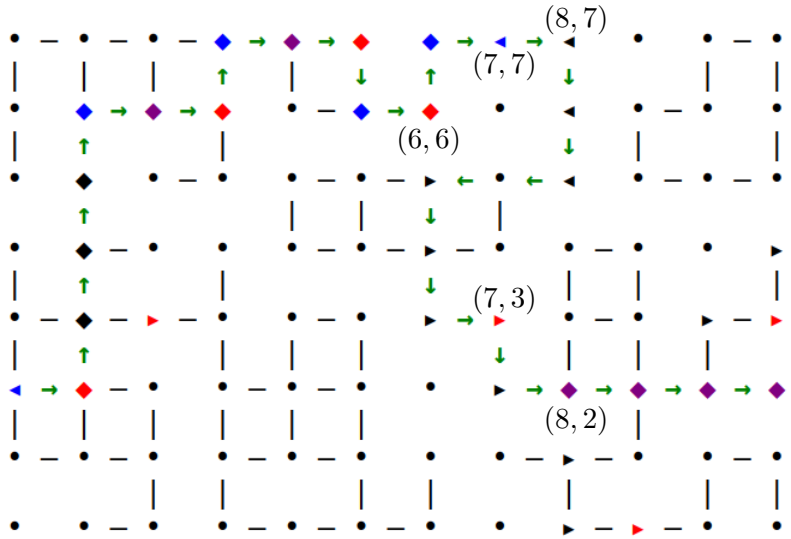

Figure 6.6: Program output showing the left-nodes in addition to the right-nodes shown in Figure 5.5. Left nodes are depicted using arrows that point to the left, and right nodes point to the right. Maximal right-nodes are coloured red, and minimal left-nodes are coloured blue. A node which is both a left- and a right-node is a diamond shape (an arrow pointing both ways), which is coloured red, blue, purple or black depending on whether the left- and right-nodes are minimal and maximal.

node in column x is a node (x_n, y_n) in P which satisfies two conditions:

1. $x_n = x$;
2. $x_m \leq x$ *for all* $m < n$.

The (unique) maximal left-node in column x is the left-node (x_n, y_n) that maximises n.

The main property of a left-node n is that all path predecessors of n lie in the same column as n, or to the left of n. The maximal left-node m has the property that the immediate path successor is to the right of m (rather than above or below). On diagrams, left-nodes are depicted as triangles pointing to the left, and maximal left nodes are filled blue.

The importance of using a left-node for T in column x is that it ensures that there is a continuous path completely contained in the verification region (columns $\leq x$) which terminates at T. For example, consider column seven

in Figure 6.6. Suppose that the right-node $(7, 3)$ is selected for T. Since only the region $x \leq 7$ is under consideration, this choice will disconnect T from the path (which ends at $(7, 7)$). Since no measurement bases and local pattern rules are changed, then measuring out all but T in this case will result in an incorrect state. The situation is corrected by using the maximal left-node $(7, 7)$, which is the highest node that is still connected to the region $x \leq 7$.

As a further example, suppose that the (non-maximal) left-node $(8, 7)$ is selected for T in column $x = 8$. Even though this is connected to the path, it has the undesirable property that the path successor $(8, 6)$ is also in the same column. Since no measurement bases change, qubit $(8, 7)$ will be measured in the X basis (for the identity pattern), which will lead to an invalid state at T (because it will realise a measurement pattern with a dangling X cluster qubit connected to the terminating qubit). This may be alternatively viewed as choosing a T from the middle of the path, not the end. This is not valid, because the terminating qubit must lie at the end of the one-qubit gate measurement pattern. By choosing a maximal left-node, it is guaranteed that the immediate path successor is always to the right, which guarantees that T is at the end of the path (in the verification region).

Key points

While right-nodes must be implemented in a control system for IMBQC, because they are directly involved in the calculation of path extensions, left-nodes are not required by the control system. Instead, the purpose of left-nodes is only in the verification of the simulation state, by providing a definite means to select a terminating qubit in each column x, that is connected to the verification region in (columns $\leq x$), and that does not require any changes to the local pattern rules.

6.5.2 Verification byproduct operators

The previous section contains a description of maximal left-nodes, and why they are an appropriate choice for the terminating qubit T in each column. In this section, we discuss the verification byproduct operators, which follow from the potential discrepancy in measurement outcomes between the main simulation and the verification. The verification byproduct operators:

- **Are based on the main byproduct operators.** These are the true byproduct operators that were computed during the main simulation of column $x - 1$. This agreement justifies the verification scheme: if the verification succeeds, it provides evidence that the main byproduct operators from column $x - 1$ were correct.

- **Differ from the main byproduct operators at column x.** This is the compromise involved in using this verification scheme, which means the verification lags the main simulation by one column.

- **Are discarded after the verification of column x.** The verification byproduct operators do not persist across columns. Their only purpose is to correct the state obtained during the verification step, for comparison with the known-true state.

The need for verification byproduct operators is explained fully in Appendix C.

A simple problem that occurs in the application of the byproduct operator rules R_b in a column such as $x = 7$ in Figure 6.6. Since the measurement pattern is generated up to the right-node $(7, 3)$, there are several (on-path) nodes whose contribution to byproduct operators must be ignored, because they occur in the future of the path. To make this point clearer, consider the measurement pattern obtained from Figure 6.3 by choosing the qubit two $(-\phi_2)$ as the terminating qubit, and removing the entanglement link to qubit three. Despite terminating on qubit two, all qubits in the diagram are still measured. It is obvious that byproduct operator contributions to the right of qubit two must not be applied, because those qubits are no longer part of the measurement pattern (they are simply measured and discarded).

This issue is solved by storing an index with each byproduct operator rule R_b, which records what path index the rule should be applied to. Then, when applying the measurement pattern to a column, byproduct rules should only be applied if their corresponding index is less than or equal to the path index of the terminating qubit. A consequence of this rule is that it is not possible to combine the cut-out rules using Equation (6.5). Instead, each term must be stored separately with its index, and then combined into the sum only if its index is less than or equal to the terminating qubit index. This requirement does not have to be implemented in an actual design; it is purely a feature of the column verification method.

In the next section we intentionally introduce errors into the simulation, by adding noise to the modulator voltages that set the measurement bases in the simulation. In this case, the fidelity of the verification state may be interpreted as an indication of how well the MBQC system is able to realise one-qubit gates.

6.6 Analysing fidelity as a function of time using `esim`

Figure 6.7 shows the result of using MBQCSIM (`esim`) to simulate the effect of modulator voltage errors in an implementation of the identity pattern in IMBQC. In the simulation, Gaussian white noise defined by σ in the range 0 mV–5 mV was applied to each modulator voltage setting (two per measurement of each cluster qubit), and the verification scheme described in Section 6.5 was used to establish the fidelity of the output logical qubit state at each column. A photonic cycle time of 1 ns has been used to interpret the column index x as time, which is plotted on the x-axis of the graph. Each modulator is assumed to have $V_\pi = 1\,\mathrm{V}$. The results show that even quite a low noise level in the modulator voltage leads to quite a large error in the quantum state (fidelity = 0.95) after about 1 µs. Simulations like these may be used to establish specifications for analog electronics suitable for driving the modulators, maximum circuit depths before a particular error is reached, or provide information relating to what type of quantum error correction is necessary in implementations of IMBQC.

Key points

The results show that logical qubits in photonic quantum computing may display a kind of "decoherence" that is similar to that observed in matter-based qubits. This arises from the inability to perform exactly the right measurements in MBQC. While quantum error correction schemes often consider qubit errors [50], these continuous errors due to non-idealities in the analog electronics are also important, and must be fully quantified as part of the design of a control system for photonic quantum computing.

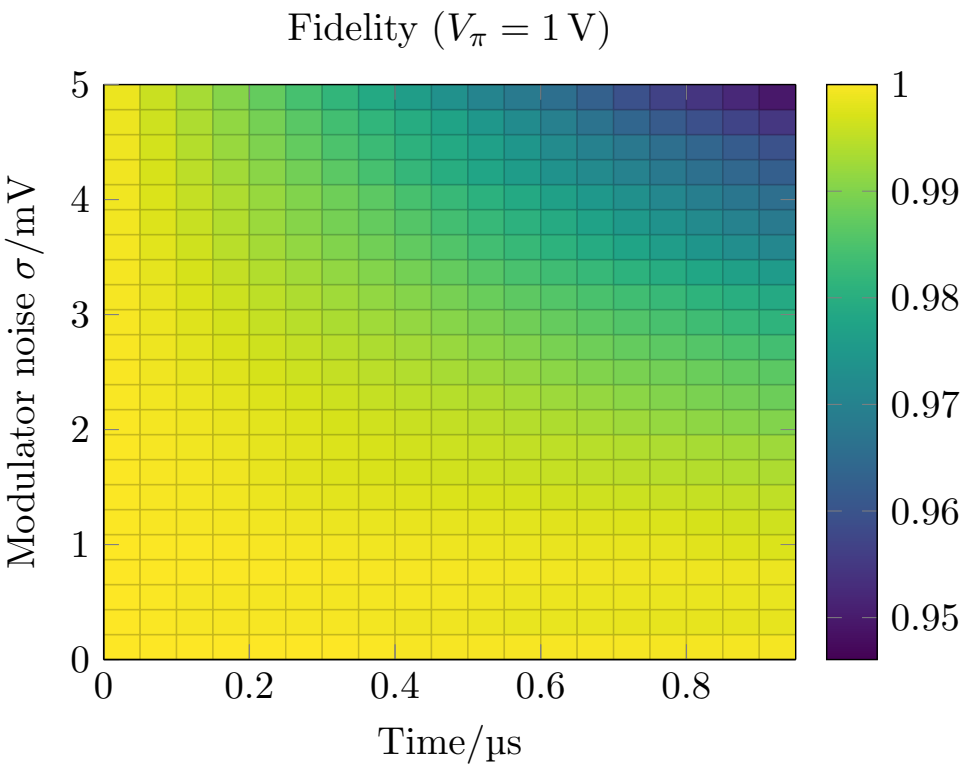

Figure 6.7: The heatmap shows the average fidelity of the logical qubit realised using the identity path pattern, as a function of elapsed time (derived from a photonic cycle time of 1 ns), and the standard deviation of the Gaussian white noise in the modulator voltage. The experiment was conducted by simulating a pathfinding process using GBFS in cluster state of height $H = 7$, for a range of noise levels. Each experiment was repeated 1000 times and the results averaged.

Next steps

It is possible to simulate any other noise model in MBQCSIM. For example, the Gaussian noise model also supports the setting of μ, which represents a DC bias in all the modulator settings. This type of error may represent effects due to ringing in the modulator voltage, if the photon always arrives in a particular phase relationship with the timing of the analog electronic control system. A full analysis of how noise affects the fidelity of quantum operations is an important prerequisite for the design of the analog parts of the system.

6.7 Summary and conclusion

Chapters 5 and 6 presented a framework for analysing and simulating photonic MBQC using incomplete cluster states (IMBQC), realised using the C++ and python library MBQCSIM. The purpose of this analysis has been to lay the ground work for an initial (idealised) implementation of a control system for IMBQC.

In creating such a control system, it is necessary to know the implementation cost of the algorithms involved in the cluster-state search process, the pathfinding process, and the pattern generation process. The ring-buffer implementation model enabled a concrete discussion of what memory operations are required for the particular search algorithms presented in Chapter 5 (GBFS and IBFS), and formed the basis for the emulation of hardware implementations using the MBQCSIM program `pathf`.

Chapter 6 focused on the dynamic generation and storage of the measurement pattern for arbitrary one-qubit gates, in a manner suitable for emulation along the lines discussed in Chapter 5, although MBQCSIM does not perform this emulation. Instead, the focus of `esim` is to simulate the measurement patterns laid out along paths through an incomplete cluster state, both to verify that the pattern (and pattern rules) are correct, and also to investigate how voltage errors in the modulators affect the fidelity of the output logical-qubit state from the IMBQC system. These simulations may form the basis for analog control system specifications for the modulators, or restrictions on the depths of MBQC patterns that can be realised using IMBQC implementations.

The purpose of this work is to recast some of the theoretical aspects of photonic MBQC in a form that is free of complicated mathematics, by providing measurement patterns in the form of simple diagrams. It is hoped that these diagrams somewhat separate the implementation of the schemes from the need to understand why they work; a necessary feature if control system design for photonic quantum computing is to be treated in a serious way. At the same time, the implementation of the algorithms in MBQCSIM could provide a practical starting point for the further development of this subject, without requiring as a prerequisite a full understanding of all the mathematics underpinning the subject.

On the other hand, many theoretical aspects of the implementation of IM-

BQC remain unaddressed. Most importantly, only arbitrary one-qubit gates are addressed here, and only the identity pattern is currently implemented in MBQCSIM (although the additional local pattern rules and simulation could be incorporated into MBQCSIM). The need to address multi-qubit gates would entail algorithms that could find multiple paths (representing separate logical qubits), and join them appropriately in order to realise, for example, the CNOT gate.

The source code of MBQCSIM, which is freely available, may be taken as a first draft of the algorithms required for a hardware implementation of IMBQC, along the lines discussed in Chapter 3. MBQCSIM is tested and verified, and may provide the basis of a functional simulation of a control system design. Ultimately, a hardware design is necessary to establish what constraints are present in this prospective realisation of quantum computers.

Chapter 7

Conclusions

We close this thesis by drawing two key conclusions. Firstly, we stress the importance of presenting photonic quantum computing architectures in a simple non-mathematical form. Secondly, we emphasise that the analysis of a fully-specified control system design is better than using emulation techniques.

Finally, we provide an outlook on the subject of control system design for photonic quantum computing, and offer some suggestions for next steps.

7.1 The need for a non-mathematical approach to photonic MBQC

We advocate a non-mathematical approach to photonic MBQC in this thesis. This lowers the barrier to entry for engineers who may be able to contribute significantly to the problem of control system design for photonic quantum computing, but who do not wish to familiarise themselves with all the mathematical details of MBQC and related subjects.

As we showed, detailed mathematics is not necessary for the discussion of control system implementations for photonic MBQC. We presented all key measurement patterns in terms of simple diagrams, which specified what operations must be performed by the control system, but not why.

The question "why" is not relevant for control system design. Instead, proposals for photonic quantum computing should be fully specified, by writing all classical operations required in a simple descriptive form that constitutes a specification for the control system. This way, the problem is cast in a form familiar to electronic designers, who expect to implement a specification defin-

ing what the control system must do. Most research on photonic quantum computing to date has not been presented in this form [13–16, 61].

Often, however, it is not possible to represent a given proposal for photonic quantum computing in the form described above, because too many aspects of the classical control system remain unspecified. This leads to the need for control system emulation, to investigate what is the best way to realise the classical algorithms required by the quantum computer. However, the analysis of a fully-worked-out control system design is preferable to this emulation, as we discuss in the next section.

7.2 Control system design is better than emulation

In Chapter 3, we designed a digital system targeting an FPGA, whereas in Chapter 5, we emulated a class of control system implementations based on a particular model.

Providing a full control system design is preferable to using emulation, because a detailed design provides a concrete basis for the analysis of constraints on the photonic quantum computer. Although emulation may be used to estimate these constraints, there is always the possibility that implementation details may render the estimates invalid.

However, immediately designing the control system is very difficult when many parts of the design are unspecified. For example, in the renormalisation approach to photonic MBQC [14], only the high-level goals of the relevant classical algorithms are presented. We showed in Chapter 5 that substantial work is required to hone these algorithms into a form that would not impose very stringent timing constraints on the photonic quantum computer.

Photonic quantum computing proposals should lay out what classical processing steps are required in detail, in a non-mathematical form; this would greatly simplify the process of analysing control system implementations, enabling the in-depth optimisation that is likely required to achieve designs that do not impose heavy limitations on the quantum computer.

In addition to the greater validity of constraints, a detailed hardware design also provides a full draft of all the classical algorithms that are required for the implementation of photonic quantum computing. Even though software emulation is a good starting point, it is still possible to "hide details under the rug" in the emulation system; it is very important to expose all the classical

control elements in the system, so as to begin the task of optimising the design of this aspect of photonic quantum computers.

To date, control system design for photonic quantum computing is an under-researched discipline; partly because many architectural aspects of the quantum computer are still being worked out, and some critical photonic components for the realisation of quantum computers are being actively investigated. In the next section, we highlight some important next steps in the analysis of classical control systems for photonic quantum computing.

7.3 What to investigate next?

This thesis focused almost exclusively on the design of the digital parts of the control system for photonic quantum computing. The same type of analysis could be conducted for the analog parts of the system, such as the single-photon detector amplifiers, the ADC systems for producing measurement basis voltages, and the driving circuitry for interfacing to modulators. All these systems must meet similarly high timing constraints, and some or all of them may need to be cryogenic, for compatibility with SNSPDs.

Another direction lies in converting the wealth of theoretical information about photonic quantum computing architectures into a practical descriptive form that lends itself to control system analysis. This type of work may not be very hard for theorists and mathematicians to perform, but would pay for itself immeasurably in easing the work of engineers wishing to investigate design questions relating to control systems for photonic quantum computers.

Finally, there is the question of optimising the specific algorithms discussed in this thesis to the point where they are feasible for use in photonic quantum computing systems. Even though these algorithms may look efficient on paper, the necessity for the implementation to compete with the speed of light places very high standards on the level of optimisation that must be performed. Unless very significant improvements are made in the speed of these algorithms, which will be present in any architecture for photonic quantum computing, the use of photonic qubits does not represent a realistic proposal for the realisation of large-scale quantum computers. This problem pushes the boundaries of what is possible in high-speed electronic design, and forms an attractive subject of study for engineers and scientists in the fields of electronic engineering, computer science and quantum physics.

Appendices

Appendix A

Mathematics of MBQC

This appendix contains an overview of the mathematics used to derive the measurement patterns presented in this thesis. Appendix A.1 shows how the reduced CNOT gate, introduced in Section 2.1.1, is derived. Appendix A.2 derives the effect of cut-out qubits discussed in Section 6.2.1 on the arbitrary one-qubit-gate measurement pattern presented in Section 2.1.

In line with the goal of this thesis, to make MBQC as accessible as possible, we try to present the mathematics in an introductory manner. This is designed to complement more rigorous approaches to the subject [20, 21], which provide more detailed background on the information presented here.

A.1 CNOT measurement pattern

In the control system design in Chapter 3, we implement a reduced measurement pattern for the CNOT gate that only uses two rows of cluster qubits, so that only nearest-neighbour connectivity is required between logical qubits. This section is adapted from [57, Appendix B].

The reduced CNOT pattern is derived here using the same method used for the calculation of the three-row CNOT gate [20, Section II.G.7]. In order to explain the derivation, we begin by discussing some technical aspects of cluster states, and describe what it means for a measurement pattern to realise a gate.

A cluster state $|\phi_C\rangle$ on N qubits is created by placing all the qubits in the $|+\rangle$ state, and then applying CZ gates between each pair of qubits that should have an entanglement link (shown as red line segments in Figure A.1). It can

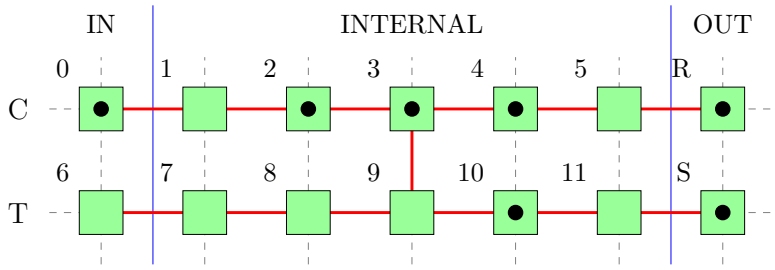

Figure A.1: The labelling of the cluster qubits for the purpose of deriving the CNOT measurement pattern. When a gate is realised in MBQC, the input state starts on the IN column and is teleported to the OUT column R and S by applying the measurement pattern. The black dots show the location of the correlation operators K_a in Equation (A.8) below.

be shown [20] that cluster states satisfy the eigenvalue equations

$$K_a|\phi_C\rangle = \left(X_a \prod_{b\sim a} Z_b\right)|\phi_C\rangle = |\phi_C\rangle, \tag{A.1}$$

where the first equality defines the correlation operator K_a on the cluster qubit a. There is one such equation for each cluster qubit a, and in each equation, the product is over all other neighbouring cluster qubits b joined by red line segments to a (denoted $b \sim a$).

To state what it means for a measurement pattern to realise a gate G, we use the arrangement of qubits shown in Figure A.1, on which the CNOT measurement pattern is defined. Instead of placing all the qubits in the $|+\rangle$ state, assume qubits 0 and 6 (the IN qubits) are in an arbitrary state $|\phi\rangle$. As before, place all the other qubits (including the OUT qubits) in the $|+\rangle$ state, and apply CZ gates wherever there are red line segments in the Figure A.1. Now, after the measurement pattern for the CNOT gate has been applied, meaning that all the IN and INTERNAL qubits have been measured out, there remains a two-qubit state $|\psi\rangle$ on the OUT qubits R and S. The sense in which the measurement pattern has realised the gate G is that the input and output states are related by

$$|\phi\rangle = BG|\psi\rangle, \tag{A.2}$$

where B is the byproduct operator for the measurement pattern. In other words, the measurement pattern has the effect of moving the state of the IN column to the OUT column, and transforming it according to the gate which is being realised by the measurement pattern.

The measurement pattern for the CNOT gate is obtained by using a theorem [20, Theorem 1] that relates eigenvalue equations derived from Equation (A.1) and a given measurement pattern, to the gate G which that measurement pattern realises. The content of the theorem is that it is only necessary to check how a cluster state $|\phi_C\rangle$ is affected by the measurement pattern (where the state of qubits 0 and 6 are $|+\rangle$) in order to establish that the measurement pattern works for any other IN state $|\phi\rangle$. In the interest of simplicity, We state the theorem for the case of a two-qubit gate G like the CNOT gate:

Theorem 1. *Suppose that a cluster state $|\phi_C\rangle$ is prepared on the pattern of 14 qubits shown in Figure A.1, for the purpose of realising a two-qubit gate G acting on logical qubits labelled C and T. Suppose that a set of measurements M is performed on the INTERNAL cluster qubits 1 to 5 and 7 to 11, resulting in a state $|\psi_C\rangle$ on the remaining qubits (0, 6, R and S), which satisfies the following sets of eigenvalue equations:*

$$\begin{aligned} X_0 \left[GX_CG^\dagger\right]_{R,S} |\psi_C\rangle &= (-1)^{\lambda_x} |\psi_C\rangle \\ Z_0 \left[GZ_CG^\dagger\right]_{R,S} |\psi_C\rangle &= (-1)^{\lambda_z} |\psi_C\rangle \end{aligned} \tag{A.3}$$

and

$$\begin{aligned} X_6 \left[GX_TG^\dagger\right]_{R,S} |\psi_C\rangle &= (-1)^{\mu_x} |\psi_C\rangle \\ Z_6 \left[GZ_TG^\dagger\right]_{R,S} |\psi_C\rangle &= (-1)^{\mu_z} |\psi_C\rangle \end{aligned} \tag{A.4}$$

Then the measurement pattern in which the inner qubits are measured according to M, and the IN cluster qubits 0 and 6 are measured in the X-basis, realises the gate GB, where the byproduct operators B for the logical qubits C and T are given by

$$\begin{aligned} (x_C, z_C) &= (\lambda_z, m_0 + \lambda_x) \\ (x_T, z_T) &= (\mu_z, m_6 + \mu_x), \end{aligned} \tag{A.5}$$

where m_a is the outcome of the measurement of the a^{th} cluster qubit. □

The square bracketed terms in Equations (A.3) and (A.4) are computed in terms of the logical qubits C and T, without reference to cluster qubits. Any terms involving C and T are then interpreted as applying to the cluster qubits R and S. For example, when $G = \text{CNOT}$,

$$\left[GX_TG^\dagger\right]_{R,S} = [X_CX_T]_{R,S} = X_RX_S.$$

To apply the theorem to the CNOT gate, it is therefore necessary to obtain the following eigenvalue equations

$$\begin{aligned} X_0 \left(X_R X_S\right) |\psi_C\rangle &= (-1)^{\lambda_x} |\psi_C\rangle \\ Z_0 \left(Z_R\right) |\psi_C\rangle &= (-1)^{\lambda_z} |\psi_C\rangle \end{aligned} \tag{A.6}$$

and

$$\begin{aligned} X_6 \left(X_S\right) |\psi_C\rangle &= (-1)^{\mu_x} |\psi_C\rangle \\ Z_6 \left(Z_R Z_S\right) |\psi_C\rangle &= (-1)^{\mu_z} |\psi_C\rangle \end{aligned} \tag{A.7}$$

To obtain these equations, begin with the cluster state $|\phi_C\rangle$ on the two-row CNOT shape shown in Figure A.1, and multiply together the correlation operators in Equation (A.1) so as to obtain the following four equations:

$$\begin{aligned} |\phi_C\rangle &= K_0 K_2 K_3 K_4 K_R K_{10} K_S |\phi_C\rangle \\ &= -X_0 Y_2 X_3 Y_4 X_R X_{10} X_S |\phi_C\rangle \\ |\phi_C\rangle &= K_1 K_2 K_4 K_5 |\phi_C\rangle \\ &= Z_0 Y_1 Y_2 Y_4 Y_5 Z_R |\phi_C\rangle \\ |\phi_C\rangle &= K_6 K_8 K_{10} K_S |\phi_C\rangle \\ &= X_6 X_8 X_{10} X_S |\phi_C\rangle \\ |\phi_C\rangle &= K_4 K_5 K_7 K_9 K_{11} |\phi_C\rangle \\ &= Y_4 Y_5 Z_R Z_6 X_7 X_9 X_{11} Z_S |\phi_C\rangle. \end{aligned} \tag{A.8}$$

The right hand sides are obtained by repeated application of the equation $X_a Z_a = iY_a = -Z_a X_a$. Note that Pauli operators on different qubits commute.

As with any pattern derived using this method, the choice of operators K_a in the above equations is motivated by two goals:

- The equations must contain the correct IN and OUT terms in Equations (A.6) and (A.7). These terms are coloured red in the equations.
- The Pauli operators on the INTERNAL cluster qubits agree between all the equations. That is, for each cluster qubit a, only X_a or Y_a appears across all the equations. For example, when $a = 4$, only Y_4 appears (three times, shown in blue), and there are no instances of X_4. It is these operators that define the measurement bases M for each qubit a in the INTERNAL group of cluster qubits.

When the INTERNAL qubits are measured according to M, the Pauli terms disappear [50, Section 10.5.3], and each one contributes a sign according to its measurement outcome m_a, to give the following equations on the reduced state $|\psi_C\rangle$:

$$\begin{aligned} X_0 X_R X_S |\psi_C\rangle &= (-1)^{1+m_2+m_3+m_4+m_{10}} |\psi_C\rangle \\ Z_0 Z_R |\psi_C\rangle &= (-1)^{m_1+m_2+m_4+m_5} |\psi_C\rangle \\ X_6 X_S |\psi_C\rangle &= (-1)^{m_8+m_{10}} |\psi_C\rangle \\ Z_6 Z_R Z_S |\psi_C\rangle &= (-1)^{m_4+m_5+m_7+m_9+m_{11}} |\psi_C\rangle \end{aligned}$$

These equations are in the form of Equations (A.6) and (A.7), and define the values of $\lambda_x, \lambda_x, \mu_x, \mu_z$ in terms of the measurement outcomes m_a. As a result, it follows from the theorem above that the measurement pattern consisting of M, plus X measurements on the IN qubits, realises the gate (CNOT)B, where the byproduct operator B found using Equation (A.5) to be

$$\begin{aligned} (x_C, z_C) = (&m_1 + m_2 + m_4 + m_5, \\ &1 + m_0 + m_2 + m_3 + m_4 + m_{10}) \\ (x_T, z_T) = (&m_4 + m_5 + m_7 + m_9 + m_{11}, \\ &m_6 + m_8 + m_{10}). \end{aligned} \tag{A.9}$$

Finally, the byproduct operator can be commuted past the CNOT gate to obtain

$$\begin{aligned} (Z_C^{1+m_0+m_2+m_3+m_4+m_6+m_8} X_C^{m_1+m_2+m_4+m_5} \\ Z_T^{m_6+m_8+m_{10}} X_T^{m_1+m_2+m_7+m_9+m_{11}}) \text{CNOT}. \end{aligned} \tag{A.10}$$

The contributions to the byproduct operators given in this formula are depicted in Figure 2.3, and stated in Equations (2.1) and (2.2).

A.2 One-qubit gates in incomplete cluster states

The derivation of the measurement pattern for the one-qubit gate, incorporating cut-out qubits, follows a plan analogous to the derivation of the ordinary one-qubit gate pattern [20, Section II.G.3-5]. The derivation is performed using a generalisation of the theorem used in Appendix A.1 [20, Theorem 1], extended to the situation where some $\kappa_a \neq 0$ (see Section 6.2.1):

Theorem 2. *Suppose that a cluster state $|\phi_C\rangle$ (which may have $\kappa_a \neq 0$ for some cluster qubits) is prepared on a linear pattern of N qubits (labelled 0 through $N-1$), for the purpose of realising a one-qubit gate G acting on logical qubits labelled on one logical qubit T. Suppose that a set of measurements M is performed on the INTERNAL cluster qubits 1 to $N-2$, resulting in a state $|\psi_C\rangle$ on the remaining qubits (0 and $N-1$), which satisfies the following sets of eigenvalue equations:*

$$X_0 \left[GX_T G^\dagger\right]_{N-1} |\psi_C\rangle = (-1)^{\lambda_x} |\psi_C\rangle$$
$$Z_0 \left[GZ_T G^\dagger\right]_{N-1} |\psi_C\rangle = (-1)^{\lambda_z} |\psi_C\rangle$$

Then the measurement pattern in which the inner qubits are measured according to M, and the IN cluster qubit 0 is measured in the X-basis, realises the gate GB, where the byproduct operators B for the logical qubit T are given by

$$(x, z) = (\lambda_z, m_0 + \lambda_x)$$

where m_0 is the outcome of the measurement of cluster qubit 0. □

The theorem states that the general cluster state $|\phi_C\rangle$, where some $\kappa_a \neq 0$, may also be used for the realisation of measurement patterns. The theorem may be proven by making bookkeeping-style modifications to the proof of the original theorem [20, Theorem 1]. By accounting for the effects of cut-out qubits in setting particular values for κ_a, as described in Section 6.2.1, this theorem may then be used to establish the functioning of the measurement pattern on the resulting state.

The arbitrary R_x gate and arbitrary R_z gates are derived first, and then concatenated in the combination $R_x(\zeta)R_z(\eta)R_x(\xi)$. In each case, the eigenvalue equations of the cluster state (Equation (6.1)) on a line of appropriate length N are manipulated into a form that satisfies the conditions of Theorem 2, by making particular measurements according to a measurement pattern. The conclusion of the theorem is that this measurement pattern realises the correct one-qubit operation G, along with a byproduct operator. The types of calculations required for the R_x and R_z rotations are summarised in the sections below, so as to provide direct comparison with the same methods used to derive the ordinary one-qubit gates[1] [20]. In addition, we show in

[1] In performing this comparison, note that measurement patterns here are indexed from zero, rather than one [20].

detail how the stabiliser calculations are performed, which may be of interest to those unfamiliar to the subject.

A.2.1 Arbitrary X-rotation

The R_x rotation is achieved on a measurement pattern of three cluster qubits ($N = 3$, labelled zero through two, from left to right). However, we consider here a pattern of nine cluster qubits, where the top and bottom rows consist of computational basis measurements, whose outcomes define the values of κ_a in the middle row according to Equation (6.2). As a result, the eigenvalue equations satisfied by the middle row once the cut-out qubits have been measured[2] is given by

$$(-1)^{\kappa_0} X_0 Z_1 |\phi_C\rangle = |\phi_C\rangle \tag{A.11}$$

$$(-1)^{\kappa_1} Z_0 X_1 Z_2 |\phi_C\rangle = |\phi_C\rangle \tag{A.12}$$

$$(-1)^{\kappa_2} Z_1 X_2 |\phi_C\rangle = |\phi_C\rangle \tag{A.13}$$

Following the method used for the derivation of the ordinary one-qubit gate [20, 21], take the third equation and rearrange it to $Z_1|\phi_C\rangle = (-1)^{\kappa_2} X_2|\phi_C\rangle$, which implies $(Z_1 - (-1)^{\kappa_2} X_2)|\phi_C\rangle = 0$. Multiply by $-i\eta/2$ and exponentiate both sides to get

$$\exp\left(-\frac{i\eta}{2}\left[Z_1 - (-1)^{\kappa_2} X_2\right]\right)|\phi_C\rangle = |\phi_C\rangle.$$

Given that the terms in the exponent commute, this equation may be split up and rewritten in terms of R_z and R_x rotations as follows:

$$R_z^{(1)}[\eta] R_x^{(2)}\left[(-1)^{\kappa_2}(-\eta)\right]|\phi_C\rangle = |\phi_C\rangle. \tag{A.14}$$

In this equation, the superscript indices in the rotations indicate which qubit is rotated. Call the operator on the left hand side A, so that $A|\phi_C\rangle = |\phi_C\rangle$. Combine this with the Equation (A.12) above, to get $A(Z_0 X_1 Z_2)A^{-1}|\phi_C\rangle = (-1)^{\kappa_1}|\phi_C\rangle$, or

$$Z_0 \left(R_z^{(1)}[\eta] X_1 R_z^{(1)}[-\eta]\right)\left(R_x^{(2)}[(-1)^{\kappa_2}(-\eta)] Z_2 R_x^{(2)}[(-1)^{\kappa_2}(-\eta)]\right)|\phi_C\rangle = (-1)^{\kappa_1}|\phi_C\rangle \tag{A.15}$$

[2] Recall that, in analysis of MBQC circuits, the order that the cluster qubits are measured does not matter, provided that any adaptive measurement setting dependencies are satisfied. In the analysis, it is assumed that all the cut-out qubits are measured first, before the middle row, even though the measurements would actually occur one column at a time in a real photonic implementation.

The purpose of this equation is that a measurement of the observable $M = R_z^{(1)}[\eta]X_1R_z^{(1)}[-\eta]$ on the middle qubit (qubit one) will replace the leftmost term in parentheses with a constant $(-1)^{m_1}$, where m_1 is the result of the measurement. This may be verified algebraically by multiplying Equation (A.15) by the projector

$$P_M = \frac{1 + (-1)^{m_1}M}{2},$$

resulting in the equation

$$Z_0\left(R_x^{(2)}[(-1)^{\kappa_2}(-\eta)]Z_2R_x^{(2)}[(-1)^{\kappa_2}\eta]\right)|\phi'_C\rangle = (-1)^{m_1+\kappa_1}P_M|\phi'_C\rangle, \quad \text{(A.16)}$$

where $|\phi'_C\rangle = P_M|\phi_C\rangle$. This is one of the equations that goes into the assumptions of Theorem 2: it is an eigenvalue equation that applies to the IN and OUT qubits of the measurement pattern (qubit zero and two), which is of the general form

$$Z_0\left[GZG^\dagger\right]_2|\phi'_C\rangle = (-1)^{m_1+\kappa_1}P_M|\phi'_C\rangle. \quad \text{(A.17)}$$

In this equation, the G term represents the logical one-qubit unitary operation that is achieved using the measurement pattern; in this case, $G = R_x[(-1)^{\kappa_2}(-\eta)]$.

The second equation that is necessary for the application of the theorem may be derived by combining Equations (A.11) and (A.13) to obtain

$$(-1)^{\kappa_0+\kappa_2}X_0X_2|\phi_C\rangle = |\phi_C\rangle.$$

Applying the projector P_M on the left (which commutes with this operator, and therefore has no effect apart from to replace ϕ_C by ϕ'_C), and inserting a redundant term corresponding to G above, results in

$$X_0\left[GXG^\dagger\right]_2|\phi'_C\rangle = (-1)^{\kappa_0+\kappa_2}|\phi'_C\rangle. \quad \text{(A.18)}$$

Equations (A.16) and (A.18) satisfy the assumptions of Theorem 2 with $G = R_x[(-1)^{\kappa_2}(-\eta)]$, $\lambda_z = m_1 + \kappa_1$ and $\lambda_x = \kappa_0 + \kappa_2$. It follows that the byproduct operator is $B = Z^{m_0+\kappa_0+\kappa_2}X^{m_1+\kappa_1}$, and the measurement pattern realises the gate GB.

Swapping the order of the gates, using the fact that X commutes with R_x and $R_x(\theta)Z = ZR_x(-\theta)$, results in the following gate being realised by the measurement pattern:

$$X^{m_1+\kappa_1}Z^{m_0+\kappa_0+\kappa_2}R_x[(-1)^{m_0+\kappa_0}(-\eta)]. \quad \text{(A.19)}$$

Note that in equations like this, it is always possible to swap the order of the X and Z contributions to the byproduct operators on the left, because this introduces a minus sign, which amounts to an (unimportant) global phase. To deterministically realise an X rotation angle of η, measure qubit one at an angle ϕ to the x-axis that satisfies $\alpha = (-1)^{m_0+\kappa_0}(-\eta)$, that is,

$$\phi = (-1)^{m_o+\kappa_0}(-\eta).$$

A.2.2 Arbitrary Z-rotation

The calculation of the R_z rotation is also a simple generalisation of the method presented in [20]. The R_z is achieved using the identity $R_z = HR_xH$. The conjugation by the Hadamard gate is achieved in the measurement pattern by placing an X measurement on either side of the measurement pattern for R_x; hence, the pattern involves a row of five cluster qubits ($N = 5$, numbered starting from zero on the left). As before, the pattern begins on a cluster state of 15 qubits arranged in three rows, where the top and bottom row are cut-out, leaving the following cluster-state equations on the middle row:

$$\begin{aligned}
(-1)^{\kappa_0} X_0 Z_1 |\phi_C\rangle &= |\phi_C\rangle \\
(-1)^{\kappa_1} Z_0 X_1 Z_2 |\phi_C\rangle &= |\phi_C\rangle \\
(-1)^{\kappa_2} Z_1 X_2 Z_3 |\phi_C\rangle &= |\phi_C\rangle \\
(-1)^{\kappa_3} Z_2 X_3 Z_4 |\phi_C\rangle &= |\phi_C\rangle \\
(-1)^{\kappa_4} Z_3 X_4 |\phi_C\rangle &= |\phi_C\rangle
\end{aligned}$$

As before, the κ_a are defined through Equation (6.2). To bring these equations into a form compatible with the use of Theorem 2, it is necessary to obtain the result of performing X measurements on qubits one and three. The process for measuring qubit one based on stabiliser manipulations is described in [21], and consists of the following steps:

1. **Obtain the stabiliser group S.** The stabiliser of the state $|\phi_C\rangle$ is given by

$$\begin{aligned}
S = \big\langle &(-1)^{\kappa_0} X_0 Z_1, (-1)^{\kappa_1} Z_0 X_1 Z_2, (-1)^{\kappa_2} Z_1 X_2 Z_3, \\
&(-1)^{\kappa_3} Z_2 X_3 Z_4, (-1)^{\kappa_4} Z_3 X_4 \big\rangle \\
= \langle &k_0, k_1, k_2, k_3, k_4 \rangle
\end{aligned}$$

The measurement is given by $\Sigma = X_1$, which anti-commutes with k_0 and k_2. Replace k_2 by $k_0 k_2$ to get

$$S = \langle (-1)^{\kappa_0} X_0 Z_1, (-1)^{\kappa_1} Z_0 X_1 Z_2, (-1)^{\kappa_0+\kappa_2} X_0 X_2 Z_3, \\ (-1)^{\kappa_3} Z_2 X_3 Z_4, (-1)^{\kappa_4} Z_3 X_4 \rangle.$$

Now, Σ commutes with all the generators except k_0.

2. **Measure qubit one in the X basis.** The stabiliser S' of the state after the measurement of Σ is obtained by replacing k_0 by $(-1)^{m_1}\Sigma$, depending on the measurement outcome m_1:

$$\begin{aligned} S' &= \big\langle (-1)^{m_1} X_1, (-1)^{\kappa_1} Z_0 X_1 Z_2, (-1)^{\kappa_0+\kappa_2} X_0 X_2 Z_3, \\ &\qquad (-1)^{\kappa_3} Z_2 X_3 Z_4, (-1)^{\kappa_4} Z_3 X_4 \big\rangle \\ &= \langle (-1)^{m_1}\Sigma, k_1, k_0 k_2, k_3, k_4 \rangle. \end{aligned}$$

3. **Remove references to qubit one.** Finally, rewrite the stabiliser generators so that all but $(-1)^{m_1}\Sigma$ act trivially on the measurement qubit (i.e. do not contain a Pauli operator acting on that qubit):

$$S' = \langle (-1)^{m_1} X_1, (-1)^{m_1+\kappa_1} Z_0 Z_2, (-1)^{\kappa_0+\kappa_2} X_0 X_2 Z_3, \\ (-1)^{\kappa_3} Z_2 X_3 Z_4, (-1)^{\kappa_4} Z_3 X_4 \rangle.$$

This was obtained by multiplying the second generator by $(-1)^{m_1}\Sigma$. Now, the term $(-1)^{m_1}\Sigma$ may be removed from the group generators, leaving the stabiliser of the state on the unmeasured qubits.

The above process can now be repeated to obtain the result of measuring qubit three in the X basis. After this is done, the stabiliser group contains three generators, corresponding to the following eigenvalue equations for the state $|\phi'_C\rangle$ on the remaining three qubits:

$$\begin{aligned} (-1)^{m_1+\kappa_1} Z_0 Z_2 |\phi'_C\rangle &= |\phi'_C\rangle \\ (-1)^{\kappa_0+\kappa_2+\kappa_4} X_0 X_2 X_4 |\phi'_C\rangle &= |\phi'_C\rangle \\ (-1)^{m_3+\kappa_3} Z_2 Z_4 |\phi'_C\rangle &= |\phi'_C\rangle \end{aligned}$$

These equations are analogous to Equations (A.11), (A.12) and (A.13) used in the derivation of the R_x rotation. The same method can be applied, measuring

qubit two using the observable $M = R_z^{(2)}[\eta]X_2R_z^{(2)}[-\eta]$ to obtain a new state $|\psi_C\rangle = P_M|\phi'_C\rangle$ which satisfies the following two eigenvalue equations:

$$X_1\left[GXG^\dagger\right]_4|\psi_C\rangle = (-1)^{\kappa_0+m_2+\kappa_2+\kappa_4}|\psi_C\rangle \tag{A.20}$$

$$Z_1\left[GZG^\dagger\right]_4|\psi_C\rangle = (-1)^{m_1+\kappa_1+m_3+\kappa_3}|\psi_C\rangle, \tag{A.21}$$

where $G = R_z[(-1)^{m_3+\kappa_3}(-\eta)]$, which are suitable for use with Theorem 2. As a result, the measurement pattern implements the following unitary operation

$$\begin{aligned} U &= R_z[(-1)^{m_3+\kappa_3}(-\eta)]Z^{m_0+\kappa_0+m_2+\kappa_2+\kappa_4}X^{m_1+\kappa_1+m_3+\kappa_3} \\ &= X^{m_1+\kappa_1+m_3+\kappa_3}Z^{m_0+\kappa_0+m_2+\kappa_2+\kappa_4}R_z[(-1)^{m_1+\kappa_1}(-\eta)], \end{aligned}$$

using the identity $R_z[\theta]X = XR_z[-\theta]$. In order to deterministically realise a Z-rotation angle of η, measure qubit 2 in a basis ϕ that satisfies $\eta = (-1)^{m_1+\kappa_1}(-\phi)$.

A.2.3 Arbitrary one-qubit gate

The arbitrary one-qubit gate is constructed by placing arbitrary R_x rotations around an arbitrary R_z rotation:

$$\begin{aligned} U =& Z^{m_6+\kappa_6+\kappa_8}X^{m_7+\kappa_7}R_x[(-1)^{m_6+\kappa_6}(-\zeta)] \\ & Z^{m_2+\kappa_2+m_4+\kappa_4+\kappa_6}X^{m_3+\kappa_3+m_5+\kappa_5}R_z[(-1)^{m_3+\kappa_3}(-\eta)] \\ & Z^{m_0+\kappa_0+\kappa_2}X^{m_1+\kappa_1}R_x[(-1)^{m_0+\kappa_0}(-\xi)] \end{aligned}$$

In doing this (by overlapping the input and output qubits from each pattern), there exist two blocks of repeated X measurements. These measurements may be removed for the same reason they are removed in the simple non-cut out case. In that case, if the X measurements are made first (before any other measurements), the resulting equations define a cluster state on the remaining qubits, as if the X qubits were never there in the first place. If the measurement results are chosen so that there is no effect on the remaining measurement pattern, then those qubits can be treated as if they were never there [20].

In the above equations, the X measurements are on qubits 2, 3, 5 and 6. If the m_n and κ_n are chosen to be zero, then the equations reduce to:

$$U = Z^{\kappa_8}X^{m_7+\kappa_7}R_x[-\zeta]Z^{m_4+\kappa_4}R_z[-\eta]Z^{m_0+\kappa_0}X^{m_1+\kappa_1}R_x[-(-1)^{m_0+\kappa_0}\xi]$$

After relabelling the qubits 0 through 4, the equation becomes:

$$\begin{aligned} U &= Z^{\kappa_4} X^{m_3+\kappa_3} R_x[-\zeta] Z^{m_2+\kappa_2} R_z[-\eta] Z^{m_0+\kappa_0} X^{m_1+\kappa_1} R_x[(-1)^{m_0+\kappa_0}(-\xi)] \\ &= X^{m_1+\kappa_1+m_3+\kappa_3} Z^{m_0+\kappa_0+m_2+\kappa_2+\kappa_4} R_x[(-1)^{m_0+\kappa_0+m_2+\kappa_2}(-\zeta)] \\ &\qquad R_z[(-1)^{m_1+\kappa_1}(-\eta)] R_x[(-1)^{m_0+\kappa_0}(-\xi)] \end{aligned} \tag{A.22}$$

Key points

Equation (A.22) shows the logical operation U achieved by the generalised one-qubit measurement pattern, including the effect of cut-out qubits above and below the measurement pattern line. Each time the measurement outcome m_a is present in the formula, it is accompanied by κ_a (apart from κ_4, because qubit four remains unmeasured). We generalise this pattern in the next section to measurement patterns of arbitrary length.

A.2.4 One-qubit gate along a linear cluster

The simplest way to obtain the one-qubit pattern along an arbitrarily-long line is to follow an inductive strategy by adding one cluster qubit to the line at a time [21]. This is equivalent to generating a full cluster state and then performing all the measurements at once [20], and also corresponds to the simulation method outlined in Chapter 4. First, suppose that the state of the first qubit a_0 in the line is $|\psi_0\rangle$. After a new qubit a_1 in the $|+\rangle$ state has been added and entangled with a_0 using a CZ gate, the first qubit a_0 is measured in the equator of the Bloch sphere, at an angle $-\phi_0$ from the x-axis, with outcome m_0. This results in the following state on a_1 [21][3]:

$$|\psi_1\rangle = X^{m_0} M_x(\phi_0, 0) H |\psi_0\rangle. \tag{A.23}$$

In writing this equation, we are using the notation $M_d(\theta, s) = R_d((-1)^s\theta)$, where $d \in \{x, z\}$, in order to simplify the subsequent formulas.

By repeating the process, each time adding a new $|+\rangle$ qubit a_n, entangling with CZ, and then measuring at an angle $-\phi_n$ to the x-axis, the following

[3] There is a sign error in that tutorial [21]: Equation (6) should read $X^m H U_z(-\phi)|\psi\rangle$. This sign is required to implement rotations with the correct sign. Here, we have moved the minus sign to the xy-plane angle, $-\phi_0$

states are obtained:

$$|\psi_2\rangle = (X^{m_1} Z^{m_0})\, M_x(\phi_1, m_0) M_z(\phi_0, 0)|\psi_0\rangle \tag{A.24}$$

$$|\psi_3\rangle = \left(X^{m_0+m_2} Z^{m_1}\right) M_x(\phi_2, m_1) H M_x(\phi_1, m_0) M_z(\phi_0, 0)|\psi_0\rangle \tag{A.25}$$

$$|\psi_4\rangle = \left(X^{m_1+m_3} Z^{m_0+m_2}\right) M_x(\phi_3, m_2 + m_0) M_z(\phi_2, m_1) \\ M_x(\phi_1, m_0) M_z(\phi_0, 0)|\psi_0\rangle \tag{A.26}$$

$$\ldots$$

The following general expression is obtained by induction on n, where the inductive step is multiplying on the left by $X^n R_x(\phi_n) H$:

$$|\psi_{2k-1}\rangle = (X^{z_{2k}} Z^{x_{2k-2}})\, M_x(\phi_{2k-2}, x_{2k-2}) H \\ \prod_{r=k-1}^{1} M_x(\phi_{2r-1}, z_{2r}) M_z(\phi_{2r-2}, x_{2r-2})|\psi_0\rangle \\ \text{for } n = 2k-1 \tag{A.27}$$

$$|\psi_{2k}\rangle = (X^{x_{2k}} Z^{z_{2k}}) \prod_{r=k}^{1} M_x(\phi_{2r-1}, z_{2r}) M_z(\phi_{2r-2}, x_{2r-2})|\psi_0\rangle \\ \text{for } n = 2k, \tag{A.28}$$

where the byproduct operator bits are defined by the following equations:

$$z_{2k} = \left(\sum_{r=0}^{k-1} m_{2r}\right) \text{ modulo } 2 \tag{A.29}$$

$$x_{2k} = \left(\sum_{r=0}^{k-1} m_{2r-1}\right) \text{ modulo } 2. \tag{A.30}$$

One-qubit measurement patterns are typically based on Equation (A.28) (for $|\psi_{2k}\rangle$), because then the H term is absorbed into the product and the unitary operation that is realised is a simple product of R_x and R_z rotations. As a result of this convention, the byproduct operators are only defined on even integers. The byproduct operators can easily be extended to the odd integers too using $x_{2k-1} = z_{2k}$ and $z_{2k-1} = x_{2k-2}$ (see the byproduct operator term in Equation (A.27)).

The M_x and M_z terms in the products of Equations (A.27) and (A.28) show why it is necessary to choose adaptive measurement bases, depending on past byproduct operators, if a particular string of R_x and R_z rotations in desired: the adaptive measurement settings are required to correct the signs

of the rotations. For example, the term $M_x(\phi_{2r-1}, z_{2r})$ means that an $R_x(\theta)$ rotation can be deterministically realised by choosing $\phi_{2r-1} = \theta$, setting the adaptive measurement setting to $s = z_{2r}$, and measuring the cluster qubit a_{2r-1} in the xy-plane at an angle of $(-1)^s(-\phi_{2r-1})$ from the x-axis.

Figure 6.2 (in Chapter 6) shows a diagrammatic representation of Equations (A.27) and (A.28). The diagram is designed to express the alternating nature of the byproduct operators, by swapping the x_{2k} and z_{2k} lines every other qubit. However, the regions shaded in blue show how to use these lines to obtain the byproduct operators compatible with either Equation (A.27) or Equation (A.28). The one-qubit logical operation that is realised can be read off by combining all the rotations listed above each qubit. However, for measurement patterns terminating on an odd qubit, it is important to remember the H term present in Equation (A.27).

In contrast to the arbitrary one-qubit gate presented in Chapter 2 (in Figure 2.2), where a commutation correction is made at the boundary of each measurement pattern, Figure 6.2 shows how to perform a running calculation of byproduct operators; since the pattern can be any length, there is no need to consider gate boundaries and commutation corrections.

The measurement pattern to achieve the identity gate may be obtained from Equations (A.27) and (A.28) by setting $\theta_n = 0$ everywhere (so that no R_x or R_z rotation is performed). In that case, the formulas reduce to $|\psi_{2k-1}\rangle = X^{z_{2k}} Z^{x_{2k-2}} H|\psi_0\rangle$ and $|\psi_{2k}\rangle = X^{x_{2k}} Z^{z_{2k}}|\psi_0\rangle$. This example shows why the identity pattern must be considered only on even cluster qubits: in the odd case, the state $|\psi_{2k-1}\rangle$ realised by the measurement pattern is H (up to byproduct operators), not I.

A.2.5 One-qubit gate through a 2D cluster state

To embed the linear measurement pattern shown in Figure 6.2 into an arbitrary 2D cluster state, it is necessary to account for the effect of cut-out qubits above and below the line, as calculated in Appendix A.2.3. The outcome from any cut-out Z-measurement that is connected to a path qubit a_n is XORed into z_n if n is even, and into x_n if n is odd, leading to the following updated

equations for the byproduct operators

$$z_{2k} = \left(\sum_{r=0}^{k-1} m_{2r} + \kappa_{2r} \right) \text{ modulo 2,} \tag{A.31}$$

$$x_{2k} = \left(\sum_{r=0}^{k-1} m_{2r-1} + \kappa_{2r-1} \right) \text{ modulo 2,} \tag{A.32}$$

where κ_n is the sum of Z-measurement outcomes from the cut-out qubits surrounding a_n.

The effect of the measurement pattern is then given by Equations (A.27) and (A.28), with the updated values for z_{2k} and x_{2k}. The calculation of the byproduct operators and adaptive measurement settings for a straight horizontal path is shown in Figure 6.3.

This figure may be generalised to the measurement pattern for a general path through an incomplete cluster state. The only difference is which cut-out measurement outcomes are added to the byproduct operators at each path index n. Path index in the general case corresponds to column index in the special case shown in Figure 6.3.

Appendix B

Implementation details of MBQCSIM

The primary reference for the algorithms implemented in Chapters 5 and 6 is the source code for MBQCSIM [88]. This appendix describes some details regarding how the library was implemented.

B.1 Modelling incomplete cluster states

The most important data structure in the C++ library is the full node window, which models a two-dimensional cluster state of width W and height H. This node window is stored as a column-major vector of primary data (edge information), along with secondary data relating to algorithms (see Appendix 5.1.3 for an overview of the ring buffer model implemented in MBQCSIM). Column-major format is used so that new columns can easily be appended to the right of the node window without disturbing data to the left.

Source-code reference: MBQCSIM

The `NodeWindow` class, which models the cluster state, and includes primary and secondary data in the ring buffer, is defined in `node-window.hpp`. The class is configurable by specifying a `Node` template parameter, which controls the format of the primary and secondary data – this models the entry in the ring buffer.

An important feature of this node window is that the secondary data struc-

ture is defined by the algorithms that are used as part of IMBQC. In order to satisfy the modular requirement of the program, which allows different algorithms to be developed and tested, the node type is represented as a class that is generated from its constituent substructures using a policy-based approach to node characteristics [89]. This allows the compile-time customisation of the secondary data characteristics that are required for the implementation of any algorithm in the ring-buffer model, without needing to hard code those characteristics in the node window.

Source-code reference: MBQCSIM

The node data is built up from the classes defined in `distance-node.hpp`, `pattern-node.hpp`, `qubit-node.hpp`, etc., using the utility `MakeNode` contained in `make-node.hpp`. This aggregates all the data from each node type into a single `Node` class, that represents the primary and secondary data in the node window.

An important function provided by the full node window is access to a particular block subwindow of width B, extending from column x to column $x + W$, to any subroutine in the program that needs it. This block models the ring buffer described in Appendix 5.1.3. The block contains BH entries in total, each of which is a data structure containing the primary data (edge information) and secondary data (which is algorithm specific) relating to a cluster qubit at coordinates (x, y) within the block.

Next steps

Currently, MBQCSIM only supports 2D cluster states. In principle, it could be extended to support any cluster-state geometry, for example, a regular 3D lattice. This would require the replacement of the column-major format used in the node window with another storage pattern supporting the new geometry. It would also complicate the printed output from the program (e.g. Figure 5.5), which currently uses Unicode characters printed to the console. However, the ability to emulate different cluster geometries would enable the investigation of a wider variety of proposals for photonic quantum computing.

B.2 Reproducibility and seeding in MBQCSIM

Reproducibility is very important in a program where randomness can directly affect the results, for two reasons. Firstly, if a run of the program exposes a bug, it is important to be able to be able to reproduce that bug so that it can be analysed and fixed. Secondly, if the program is to be thoroughly tested, it must be able to produce known-correct results.

There are four uses of randomness in the program, which are shown explicitly in Figure 5.3:

- **Random graph generation**, for generating the random edges in the node window, which models the cluster state. Each edge in the graph is a independent binomial random variable, with probability p (the edge probability). This source of this randomness is controlled by the graph seed, S_g.

- **Path branch selection**, when a random choice is made between two path successors for a given node. The choice is made randomly, and each such choice is independent. This randomness is controlled by the path seed, S_p.

- **Simulated measurement outcomes**, which are required when cluster qubit measurements are made of the qubits in the embedded quantum simulator. The probability of getting one outcome or the other, and their correlations, is a property of the evolving quantum state of the system. However, the underlying source of randomness used is controlled by the simulation seed, S_s.

- **Modulator voltage noise**, which is used to introduce realism into the quantum simulation process. The source of this randomness is controlled by the noise seed, S_n.

B.3 Generation of seeds

Providing the same values for all four seeds will guarantee that the program produces the same output (provided all other configuration parameters are equal). However, any one of the seeds in isolation may not produce the same results for that part of the program, unless the other seeds on which it depends

are also equal. For example, specifying S_p will not guarantee that the same path is produced unless S_g is also specified. Similarly, simulated measurement outcomes (which depend on the measurement pattern) are not entirely determined by S_s, because the graph S_g and the path S_p are also important. In general, each seed listed above only guarantees the same program output if all the other seeds listed before it are specified. All seeds also depend on the configuration parameters of the program to guarantee repeatability.

In C++, randomness is created by using a "generator", which acts as a source of randomness, and a "distribution", which converts that random source into samples from the desired distribution. In MBQCSIM, the generator is `std::mt19937_64`, which produces numbers from the (64-bit) Mersenne Twister pseudo-random number generator. However, C++ standard distributions such as `std::uniform_int_distribution` are not guaranteed to provide the same samples even if the generator is the same [81]. As a result, MBQCSIM implements a simple distribution for generating uniform random real numbers and integers. The seeds described above are used to set the starting point for the Mersenne Twister engine.

Key points

Reproducibility of program results is very important in MBQCSIM, because much of the behaviour of the program is governed by random processes. Rare edge cases are very hard to reproduce unless the same string of random numbers can be generated each time the program runs. All experiments performed using MBQCSIM are defined using a root seed, which completely specifies the output data from the program.

Source-code reference: MBQCSIM

The custom implementation of random number generation, used to enforce reproducibility in MBQCSIM, is contained in the files `random.hpp` and `seed.hpp`. The test suite (in the folder `tests/`) contains checks that particular seeds give rise to the same random numbers, and that the program output is uniquely defined by the seeds.

The locations of the randomness and the seeds are shown in Figure 5.3. The emulation program `pathf` uses two seeds (S_g and S_p), and the noise sim-

ulation program uses all four seeds. When the programs are called with the parameter n (number of repeats), a different seed is needed for each run of the program. It is very important that these seeds are not generated randomly, because of the possibility of accidentally using the same seed twice, and thereby introducing correlations into the program [90]. Instead, the only property required is that the seeds are unique. This is obtained simply by using the following formulas to generate the seeds in `pathf`:

$$S_g^{(n)} = S + 2n,$$
$$S_p^{(n)} = S + 2n + 1,$$

and for `esim`:

$$S_g^{(n)} = S + 4n,$$
$$S_p^{(n)} = S + 4n + 1,$$
$$S_s^{(n)} = S + 4n + 2,$$
$$S_n^{(n)} = S + 4n + 3.$$

The superscript (n) indicates that the seed is used for the n^{th} run of the experiment, and S is the root seed supplied to the program. Thus, the value of S, along with the configuration parameters of the program, uniquely specifies its output.

An error can occur if the root seed is improperly chosen in a subsequent call to `pathf` or `esim`. For example, if $S = 0$ is initially used for an experiment where $n = 10$, then `pathf` will use all the seeds in the range $[0, 19]$, and `esim` will use the seeds $[0, 40]$. If either program is run again with a seed in this range, then the results will be identical – incorrectly reducing the variability in an experiment designed to assess the variance of some quantity of interest. To mitigate this problem, the python wrappers for `pathf` and `esim` automatically manage the seed generation so that subsequent calls to the program use different seed ranges.

A similar error can occur in choosing the root seeds for a particular experiment using the multiparameter sweep. These experiments may consume very large seed ranges in the calls to the `pathf` and `esim`. A simple technique to avoid use of repeated seeds is to begin with a completely random seed, for example 6903564118784409788, and then choose subsequent seeds by incrementing the digit fourth from the left: 6904564118784409788,

6905564118784409788, etc. This will ensure that the seeds are far enough apart, but simultaneously provide enough headroom for many adjacent experiments.

B.4 Custom implementation of the normal distribution

Samples from a normal distribution are required to generate the noise values used in the analysis of modulator voltage noise described in Section 6.4.2. However, for the same reason discussed in Section B.3, it is not possible to use the C++ `std::normal_distribution`.

The custom normal distribution used in MBQCSIM is based on the polar form of the Box-Muller transform [91]. This method converts samples from the seed-based uniform random number generators in MBQCSIM into samples from a normal distribution. The method is implemented efficiently as follows:

1. **Generate two random real numbers** $x, y \in [-1, 1]$**.** This represents a coordinate (x, y) in the square of side length two, centred on the origin.

2. **Compute** $c = x^2 + y^2$**.** If it is greater than 1, discard (x, y), and go back to step one. The object is to obtain coordinates (x, y) in the unit circle.

3. **Compute the value** $d = \sqrt{-(2/c)\log c}$**.** Use it to obtain two samples $s_0 = d\sigma x + \mu$ and $s_1 = d\sigma y + \mu$, where μ and σ are the mean and standard deviation of the desired normal distribution.

The samples s_0 and s_1 are then normally distributed. Since the generation of x and y was based on a seed, so are the samples from the normal distribution. Since each instance of the algorithm generates two samples, one is stored for the next time a sample is required.

B.5 Verification of MBQCSIM programs

Verification is very important in a program such as MBQCSIM, which is designed to provide information on realistic behaviour of an emulated digital system that is affected by several different sources of randomness. The primary

benefit of using reproducible randomness is that the output of the program can be compared with known-true results. Two different testing strategies have been applied to MBQCSIM in order to provide some assurance that the results are correct.

Fundamental utilities in the code, such as the node window, have been tested using a bottom-up method designed to cover many or all possible uses of those components. This bottom-up testing is used for all basic components of the code until its use becomes impractical (there are too many uses of a component, or it is not possible to easily isolate the component in the code).

Higher level constructions, such as the search algorithms, have been tested by hand-calculating the output from the algorithms and comparing the results with the output from the program. This is a form of top-down testing that can be used to establish the functioning of a large block of code at once. On the other hand, if these tests fail, they do not pinpoint the exact location of the problem.

Many rare edge cases in the algorithms have been established through the method of testing the code, establishing the seeds of failing cases, and then analysing the programs behaviour in those cases. Sometimes the bugs arise due to defects in the algorithm. In those cases, the bug has been added to the test suite in order to highlight it for correction in the next version of the program.

Source-code reference: MBQCSIM

The test suite contained in the `tests/` folder of MBQCSIM executes $> 95\,\%$ of the lines in the C++ source code. This does not guarantee that this proportion of the code is free of errors, but the high coverage reduces the chance of bugs in the library.

Appendix C

The need for verification byproduct operators

In this appendix, we describe why verification byproduct operators are necessary in the simulation of measurement patterns discussed in Chapter 6. Figure C.1 shows a summary of the steps involved in performing the cluster-state simulation discussed in Section 6.4. In Figure C.1a, the main simulation steps are outlined: the simulation progresses by successively adding columns to the right, and measuring out columns to the left, so that at most two columns are ever present. In the verification step (Figure C.1b), the left-most column is measured directly, without first adding a new column, as if it were the final column in the pattern. In the ideal verification scheme, the same measurement outcomes produced by the simulation would be used in the verification step, so that the byproduct operators agree between the verification and the main simulation.

However, it is not necessarily possible to postselect the verification column using the measurement outcomes from the main simulation, as shown in Figure C.1d. In that case, the verification column a_1a_0 is the state $|\psi\rangle = |0+\rangle + |1-\rangle$ (normalisation is omitted). If qubit a_1 is measured in the Z-basis, and a_0 is measured in the X-basis, then the only possible outcomes are 00 or 11. However, when a_2 and a_3 are entangled as shown (all the lines in the figure are CZ gates), the state on all four qubits $a_3a_2a_1a_0$ becomes

$$|\psi\rangle = |+00+\rangle + |-01-\rangle + |-10-\rangle + |+11+\rangle. \tag{C.1}$$

When a_0 and a_1 are measured in the same bases as before, all four outcomes 00,

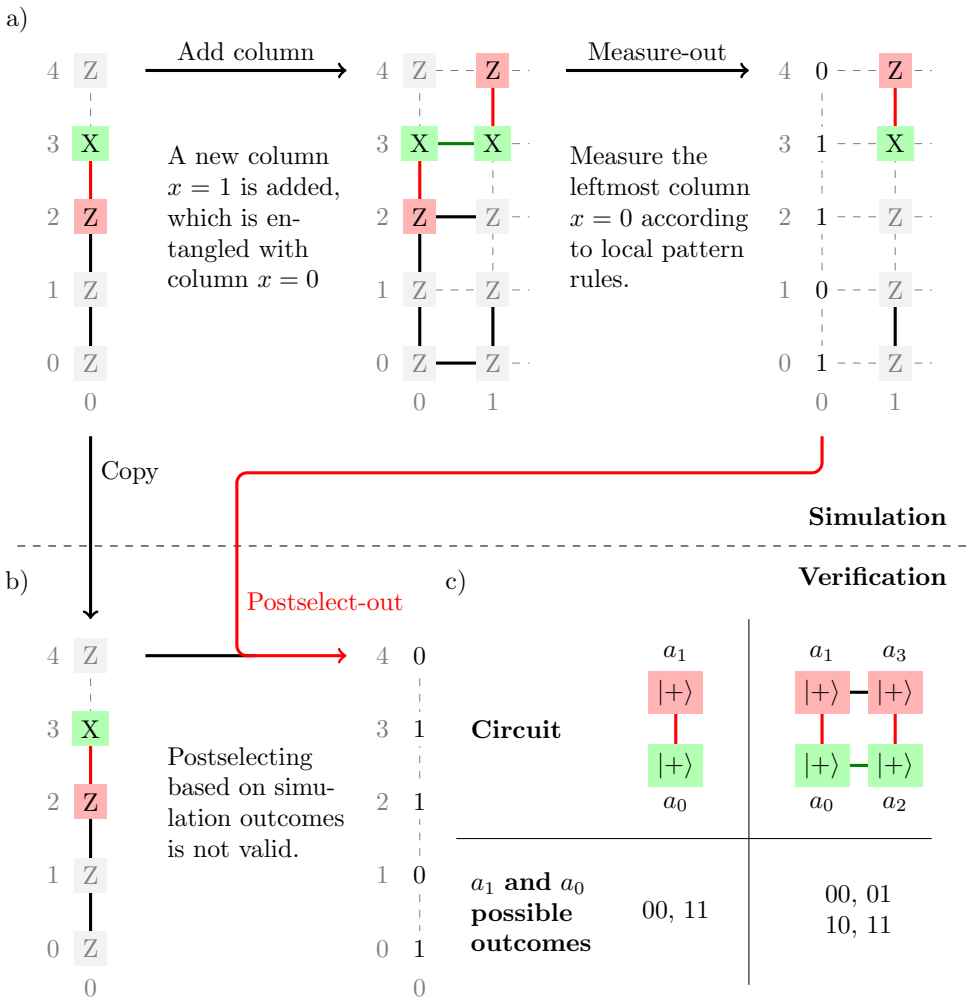

Figure C.1: a) Review of the main simulation procedure, where a column is added to the right, and then the leftmost column is measured-out according to local pattern rules. b) The "ideal" method of verification, where outcomes are made to agree between the simulation and verification by postselecting the verification column. c) A minimal example showing how the incompatibility arises in a simple case. In both circuits shown, a_1 and a_0 are measured in the Z- and X-bases, and the possible outcomes are shown below the circuits.

01, 10, and 11 are possible. This latter set of possibilities reflects the potential outcomes from the main simulation. However, if (for example) the outcome 01 is measured, the verification column cannot be postselected, because 01 is not a valid outcome for that state. As a result, it is necessary to measure the verification column, rather than postselect it. This can introduce discrepancies

between the main simulation and the verification, which is the origin of the verification byproduct operators.

Key points

As shown in Figure C.1c, the presence of entanglement with the next column can affect measurement outcomes in the column that is being verified. This means that it is not possible to enforce the same byproduct operators between the simulation and verification using postselection, which is why verification byproduct operators are necessary.

It is very important to recognise the adverse numerical effect of postselecting an invalid set of measurement outcomes. Postselection of the state vector involves retaining all those amplitudes corresponding to the postselected outcome, and zeroing out all the amplitudes corresponding to the opposite outcome, as described in Section 4.1.2. Since the postselected outcome has probability zero, these amplitudes will have very small numerical values in the simulation (e.g. $1.532 \times 10^{-15} - 1.023 \times 10^{-15} i$). However, when the state vector is normalised, these small values will be amplified, resulting in a random output state. Therefore, the effect of incorrectly attempting to postselect during the simulation is that no error will appear to have occurred, but the state will be randomised as a result.

On the other hand, errors due to the implementation of the local pattern rules (particularly the byproduct operator calculations) lead to discrete errors in the output state (in the case of the identity pattern), which are very easy to recognise. This property makes the identity pattern a useful place to start when implementing an MBQC pattern simulator, because the presence of truly random states normally indicates a problem in either the logic or the implementation of the underlying simulation infrastructure (rather than a measurement pattern error).

Once the verification byproduct operators have been obtained, and the output state is determined, this state (corrected using the verification byproduct operators) may be compared with the known state of the measurement pattern at this index (which may be determined from Figure 6.3 and Equations (A.27) and (A.28)). The expected outcome is a fidelity of one, if the simulation is correct. A fidelity less than one indicates that the simulation disagrees with the expected state of the pattern.

Key points

Invalid postselection introduces a random state into the simulation process. However, errors due to incorrect byproduct operators introduce discrete errors into the simulation. Therefore, the nature of error in fidelity (continuous or discrete) can be used to identify the kind of error that has occurred in the simulation.

References

[1] G. E. Moore et al. "Cramming more components onto integrated circuits". *Electronics* 38.8 (1965), pp. 114–117.

[2] R. P. Feynman. "Simulating physics with computers". *International Journal of Theoretical Physics* 21.6-7 (1982), pp. 467–488.

[3] B. Bauer, S. Bravyi, M. Motta, and G. K.-L. Chan. "Quantum Algorithms for Quantum Chemistry and Quantum Materials Science". *Chemical Reviews* 120.22 (2020), pp. 12685–12717.

[4] A. Cross. "The IBM Q experience and QISKit open-source quantum computing software". *APS March Meeting Abstracts*. Vol. 2018. 2018, pp. L58–003.

[5] F. Arute et al. "Quantum supremacy using a programmable superconducting processor". *Nature* 574.7779 (2019), pp. 505–510.

[6] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, and R. Wisnieff. "Leveraging Secondary Storage to Simulate Deep 54-qubit Sycamore Circuits". arXiv:1910.09534. 2019.

[7] F. Pan and P. Zhang. "Simulating the Sycamore quantum supremacy circuits". arXiv:2103.03074. 2021.

[8] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage. "Trapped-ion quantum computing: Progress and challenges". *Applied Physics Reviews* 6.2 (2019), p. 021314.

[9] D. Patterson and J. L. Hennessy. "Computer Organization and Design RISC-V Edition: The Hardware Software Interface". Morgan Kaufmann, 2017.

[10] J. P. Shen and M. H. Lipasti. "Modern processor design: fundamentals of superscalar processors". Waveland Press, 2013.

[11] J. van Dijk, E. Charbon, and F. Sebastiano. "The electronic interface for quantum processors". *Microprocessors and Microsystems* 66 (2019), pp. 90–101.

[12] E. Charbon, M. Babaie, A. Vladimirescu, and F. Sebastiano. "Cryogenic CMOS Circuits and Systems: Challenges and Opportunities in Designing the Electronic Interface for Quantum Processors". *IEEE Microwave Magazine* 22.1 (2021), pp. 60–78.

[13] D. E. Browne and T. Rudolph. "Resource-Efficient Linear Optical Quantum Computation". *Physical Review Letters* 95.1 (2005), p. 010501.

[14] K. Kieling, T. Rudolph, and J. Eisert. "Percolation, Renormalization, and Quantum Computing with Nondeterministic Gates". *Physical Review Letters* 99.13 (2007), p. 130501.

[15] M. Gimeno-Segovia, P. Shadbolt, D. E. Browne, and T. Rudolph. "From Three-Photon Greenberger-Horne-Zeilinger States to Ballistic Universal Quantum Computation". *Physical Review Letters* 115.2 (2015), p. 020502.

[16] S. Bartolucci et al. "Fusion-based quantum computation". arXiv:2101.09310. 2021.

[17] S. Thomas and P. Senellart. "The race for the ideal single-photon source is on". *Nature Nanotechnology* 16.4 (2021), pp. 367–368.

[18] B. Korzh, Q.-Y. Zhao, J. P. Allmaras, S. Frasca, T. M. Autry, E. A. Bersin, A. D. Beyer, R. M. Briggs, B. Bumble, M. Colangelo, et al. "Demonstration of sub-3 ps temporal resolution with a superconducting nanowire single-photon detector". *Nature Photonics* 14.4 (2020), pp. 250–255.

[19] S. Gyger, J. Zichi, L. Schweickert, A. W. Elshaari, S. Steinhauer, S. F. Covre da Silva, A. Rastelli, V. Zwiller, K. D. Jöns, and C. Errando-Herranz. "Reconfigurable photonics with on-chip single-photon detectors". *Nature Communications* 12.1 (2021), p. 1408.

[20] R. Raussendorf, D. E. Browne, and H. J. Briegel. "Measurement-based quantum computation on cluster states". *Physical Review A* 68.2 (2003), p. 022312.

[21] D. E. Browne and H. J. Briegel. "One-way Quantum Computation - a tutorial introduction". arXiv:quant-ph/0603226. 2006.

[22] F. Ewert and P. van Loock. "3/4-efficient Bell measurement with passive linear optics and unentangled ancillae". *Physical Review Letters* 113.14 (2014), p. 140403.

[23] S. Morley-Short, S. Bartolucci, M. Gimeno-Segovia, P. Shadbolt, H. Cable, and T. Rudolph. "Physical-depth architectural requirements for generating universal photonic cluster states". *Quantum Science and Technology* 3.1 (2017), p. 015005.

[24] D. Herr, A. Paler, S. J. Devitt, and F. Nori. "A local and scalable lattice renormalization method for ballistic quantum computation". *npj Quantum Information* 4.1 (2018), p. 27.

[25] Xilinx. "UltraFast Design Methodology Guide for the Vivado Design Suite". UG949 (v2020.1). 2020.

[26] S. H. Hall, G. W. Hall, and J. A. McCall. "High speed digital system design : a handbook of interconnect theory and design practices". New York: Wiley, 2000.

[27] S. Paesani, M. Borghi, S. Signorini, A. Maïnos, L. Pavesi, and A. Laing. "Near-ideal spontaneous photon sources in silicon quantum photonics". *Nature Communications* 11.1 (2020), p. 2505.

[28] P. Laferriere, E. Yeung, L. Giner, S. Haffouz, J. Lapointe, G. C. Aers, P. J. Poole, R. L. Williams, and D. Dalacu. "Multiplexed single-photon source based on multiple quantum dots embedded within a single nanowire". *Nano Letters* 20.5 (2020), pp. 3688–3693.

[29] E. Meyer-Scott, C. Silberhorn, and A. Migdall. "Single-photon sources: Approaching the ideal through multiplexing". *Review of Scientific Instruments* 91.4 (2020), p. 041101.

[30] I. Esmaeil Zadeh, J Chang, J. W. Los, S. Gyger, A. W. Elshaari, S. Steinhauer, S. N. Dorenbos, and V. Zwiller. "Superconducting nanowire single-photon detectors: A perspective on evolution, state-of-the-art, future developments, and applications". *Applied Physics Letters* 118.19 (2021), p. 190502.

[31] M. Zhang, C. Wang, P. Kharel, D. Zhu, and M. Lončar. "Integrated lithium niobate electro-optic modulators: when performance meets scalability". *Optica* 8.5 (2021), pp. 652–667.

[32] C. H. Bennett and R. Landauer. "The fundamental physical limits of computation". *Scientific American* 253.1 (1985), pp. 48–57.

[33] R. Landauer. "Irreversibility and Heat Generation in the Computing Process". *IBM Journal of Research and Development* 5.3 (1961), pp. 183–191.

[34] A. Hagar. "Ed Fredkin and the Physics of Information: An Inside Story of an Outsider Scientist". *Information & Culture* 51.3 (2016), pp. 419–443.

[35] C. H. Bennett. "Logical Reversibility of Computation". *IBM Journal of Research and Development* 17.6 (1973), pp. 525–532.

[36] R. P. Feynman. "Quantum Mechanical Computers". *Optics News* 11.2 (1985), pp. 11–20.

[37] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265.

[38] R. Penrose. "The emperor's new mind : concerning computers, minds, and the laws of physics". Oxford: Oxford University Press, 1989.

[39] J. von Neumann. "First Draft of a Report on the EDVAC" (1945).

[40] M. Davis. "Engines of Logic: Mathematicians and the Origin of the Computer". WW Norton & Co., Inc., 2001.

[41] D. Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (1985), pp. 97–117.

[42] D. Deutsch and R. Jozsa. "Rapid solution of problems by quantum computation". *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), pp. 553–558.

[43] P. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press, 1994, pp. 124–134.

[44] L. K. Grover. "A fast quantum mechanical algorithm for database search". *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM Press, 1996, pp. 212–219.

[45] S. Jordan. "Quantum Algorithm Zoo". URL: https://quantumalgorithmzoo.org/ (visited on 07/03/2022).

[46] M. A. Nielsen and I. L. Chuang. "Programmable quantum gate arrays". *Physical Review Letters* 79.2 (1997), p. 321.

[47] Microsoft. "Azure quantum documentation (preview)". URL: https://docs.microsoft.com/en-gb/azure/quantum/ (visited on 07/03/2022).

[48] IBM. "QISKIT: open-source quantum development". URL: https://qiskit.org (visited on 07/03/2022).

[49] O. Ezratty. "Mitigating the quantum hype". arXiv:2202.01925. 2022.

[50] M. Nielsen and I. Chuang. "Quantum computation and quantum information". 10th. Cambridge New York: Cambridge University Press, 2010.

[51] H. Zwickel, S. Singer, C. Kieninger, Y. Kutuvantavida, N. Muradyan, T. Wahlbrink, S. Yokoyama, S. Randel, W. Freude, and C. Koos. "Verified equivalent-circuit model for slot-waveguide modulators". *Optics Express* 28.9 (2020), pp. 12951–12976.

[52] C. Cahall, D. J. Gauthier, and J. Kim. "Scalable cryogenic readout circuit for a superconducting nanowire single-photon detector system". *Review of Scientific Instruments* 89.6 (2018), p. 063117.

[53] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. "A quantum engineer's guide to superconducting qubits". *Applied Physics Reviews* 6.2 (2019), p. 021318.

[54] P. Kok and B. W. Lovett. "Introduction to optical quantum information processing". Cambridge University Press, 2010.

[55] E. Knill, R. Laflamme, and G. J. Milburn. "A scheme for efficient quantum computation with linear optics". *Nature* 409.6816 (2001), pp. 46–52.

[56] T. Rudolph. "Why I am optimistic about the silicon-photonic route to quantum computing". *APL Photonics* 2.3 (2017), p. 030901.

[57] J. R. Scott and K. C. Balram. "Timing constraints imposed by classical digital control systems on photonic implementations of measurement-based quantum computing". arXiv:2109.04792. 2021.

[58] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest. "Measurement-based quantum computation". *Nature Physics* 5.1 (2009), pp. 19–26.

[59] B. J. Brown and S. Roberts. "Universal fault-tolerant measurement-based quantum computation". *Physical Review Research* 2.3 (2020), p. 033305.

[60] F. Flamini, N. Spagnolo, and F. Sciarrino. "Photonic quantum information processing: a review". *Reports on Progress in Physics* 82.1 (2018), p. 016001.

[61] H. Bombin, I. H. Kim, D. Litinski, N. Nickerson, M. Pant, F. Pastawski, S. Roberts, and T. Rudolph. "Interleaving: Modular architectures for fault-tolerant photonic quantum computing". arXiv:2103.08612. 2021.

[62] J. E. Bourassa et al. "Blueprint for a Scalable Photonic Fault-Tolerant Quantum Computer". *Quantum* 5 (2021), p. 392.

[63] C. M. Natarajan, M. G. Tanner, and R. H. Hadfield. "Superconducting nanowire single-photon detectors: physics and applications". *Superconductor Science and Technology* 25.6 (2012), p. 063001.

[64] J. R. Scott. "A digital control system for photonic MBQC". `https://gitlab.com/johnrscott/mbqc-fpga`. 2021.

[65] I. Kuon and J. Rose. "Measuring the Gap Between FPGAs and ASICs". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.2 (2007), pp. 203–215.

[66] Xilinx. "7 Series FPGAs Data Sheet: Overview". v2.6.1. DS180. 2020.

[67] Xilinx. "Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics". v2.19. DS182. 2021.

[68] Xilinx. "7 Series FPGAs SelectIO Resources". v1.10. UG471. 2018.

[69] Xilinx. "Vivado Design Suite User Guide: Getting Started". UG910 (v2021.1). 2021.

[70] E. B. Eichelberger. "Hazard Detection in Combinational and Sequential Switching Circuits". *IBM Journal of Research and Development* 9.2 (1965), pp. 90–99.

[71] Xilinx. "7 Series FPGAs Clocking Resources". v1.14. UG472. 2018.

[72] Xilinx. "Vivado Design Suite Tutorial: Design Analysis and Closure Techniques". UG938 (v2021.2). 2021.

[73] Xilinx. "Distributed Memory Generator v8.0". PG063. 2015.

[74] L. Chrostowski. "Silicon photonics design". Cambridge, United Kingdom: Cambridge University Press, 2015.

[75] P. Heydari and R. Mohavavelu. "Design of ultra high-speed CMOS CML buffers and latches". *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.* IEEE, 2003.

[76] A. Athavale. "High-Speed Serial I/O Made Simple A Designers' Guide, with FPGA Applications". 1.0. Xilinx, 2005.

[77] P. J. Ashenden. "The designer's guide to VHDL". Morgan Kaufmann, 2010.

[78] Xilinx. "Vivado Design Suite Tutorial: Design Flows Overview". UG888 (v2021.1). 2021.

[79] L. Mineh and J. Scott. "Quantum Simulation Library (QSL)". `https://github.com/lanamineh/qsl`. 2021.

[80] L. Mineh. "Solving the Hubbard model using the variational quantum eigensolver". PhD thesis. University of Bristol, 2021.

[81] "Programming Languages – C++". Standard. International Organization for Standardization, 2020.

[82] C. Eisner and D. Fisman. "A practical introduction to PSL". Springer Science & Business Media, 2007.

[83] M. Gimeno-Segovia. "Towards practical linear optical quantum computing". PhD thesis. Imperial College London, 2015.

[84] R. L. Kruse and A. J. Ryba. "Data structures and program design in C++". Prentice-Hall, Inc., 2000.

[85] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. "Introduction to algorithms". MIT press, 2009.

[86] O. F. Thomas, W. McCutcheon, and D. P. S. McCutcheon. "A general framework for multimode Gaussian quantum optics and photodetection: Application to Hong–Ou–Mandel interference with filtered heralded single photon sources". *APL Photonics* 6.4 (2021), p. 040801.

[87] M. H. Hayes. "Statistical digital signal processing and modeling". John Wiley & Sons, 2009.

[88] J. R. Scott. "MBQCSIM: C++ library for MBQC simulation". https://gitlab.com/johnrscott/mbqcsim. 2021.

[89] A. Alexandrescu. "Modern C++ design: generic programming and design patterns applied". Addison-Wesley, 2001.

[90] J. Cook. "Random number generator seed mistakes". URL: https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/ (visited on 14/02/2022).

[91] P. Occil. "More Random Sampling Methods: Normal (Gaussian) Distribution". URL: https://peteroupc.github.io/randomnotes.html (visited on 14/02/2022).