Indian Statistical Institute

Master's Thesis

Quantum Computing: How to estimate error probability in logic synthesis

Author:
Priyanka Mukhopadhyay

Supervisor:
Prof. Susmita Sur-Kolay

A thesis submitted in fulfilment of the requirements for the degree of Master of Technology

in

Computer Science

INDIAN STATISTICAL INSTITUTE

Abstract

Master of Technology

Quantum Computing: How to estimate error probability in logic synthesis

by Priyanka MUKHOPADHYAY

The interest in quantum computing began because it showed the potential to solve many classically intractable problems, as was evident from Shor's discovery of an algorithm to factor large numbers in polynomial time and Grover's algorithm to find a single object in an unassorted database. The main strength of quantum computers lay in the phenomenon of superposition, which gives it enormous information storing capability as compared to classical computers. But this does not come free of cost.

Real quantum systems are open systems which can couple in an unwanted manner to an environment or control system and lose their intrinsic quantum nature through the process of decoherence, quantum noise, and imprecise measurement, preparation and control. Fortunately, it was discovered that under reasonable physical assumptions, that is, if the worst error probability of any component is below a certain threshold, a fault-tolerant quantum computation can be built.

Ideally, an error-correcting circuit must be placed after every encoded component for error detection and recovery. But this entails a huge amount of physical resources, that is, additional gates and ancilla qubits. So we trace the error propagation in quantum circuits and place the error-correction sub-circuit only when the probability of errors exceed a certain cut-off.

For encoding we have considered three quantum error correcting codes, namely Bacon-Shor, Steane and Knill code. To calculate the error probability for different encoded gates at various levels of concatenation, we have analysed and designed an error analysis model for the physically realizable tile architecture that uses SWAP gates for movement of qubits locally. We have also designed a model for error propagation in quantum circuits and have tested it on benchmark circuits.

Acknowledgements

The two years M.Tech course at ISI, has been one of the most rewarding experiences in my life. It would not have been so wonderful without some people, whom I would like to acknowledge here.

Firstly, I am deeply grateful to my guide, Prof. Susmita Sur-Kolay, without whose active support, this thesis would not have been possible. The long hours of discussion of ideas and results, are some of the most fruitful and enjoyable moments that I have spent.

I thank Prof Niraj Jha, Chia-Chun and Amlan Chakrabarti of Princeton University, for the hours of discussion and exchange of ideas. The benchmark circuits that I have simulated my results on, were provided by them and I am grateful for that.

I thank Prof. G. Kar of PAMU, ISI, Kolkata for helping us with problems that we had. I have been lucky enough to get some wonderful teachers in ISI. No words can express my gratitude for all those things that I have learnt from them.

My parents and my brother have always been the main strength for me. And last but not the least, I thank all my friends for all our hours of fun and enjoyment.

Contents

List of Figures

List of Tables

Abbreviations

FT	**F**ault **T**olerant
EC	**E**rror **C**orrection
QD	**Q**uantum **D**ot
SC	**S**uper **C**onductor
NA	**N**eutral **A**tom
LP	**L**inear **P**hotonics
NLP	**N**on **L**inear **P**hotonics
IT	**I**on **T**rap
PMD	**P**hysical **M**achine **D**escription
QECC	**Q**uantum **E**rror **C**orrecting **C**ode

Dedicated to my parents . . .

Chapter 1

Introduction

Computer science is undoubtedly one of the greatest intellectual triumphs of the 20^{th} century. Computers are ubiquitous - from medicine to agriculture, space to road traffic control - they have tremendous impact in nearly every sphere of modern life.

The origins of computer science are lost in the depths of history. Alan Turing [1] pioneered the theoretical formulation of the modern incarnation of computer science. He developed an abstract model of computation, Turing machine (named in his honour), what is now known as a programmable computer. Few years after that, hardware development took pace when in 1947, John Bardeen, Walter Brattain and Will Shockley developed the transistor. Ever since then, computer hardware has grown in power at an astonishing rate. In 1965 Gordon Moore codified in what is famously known as Moore's law, that computer power will double for constant cost roughly once every two years. However, this dream run is expected to end because conventional approaches to the fabrication of computer technology are beginning to run up against fundamental difficulties in size. Quantum effects are beginning to interfere in the functioning of electronic devices as they are made smaller and smaller.

One possible solution is to move to a different computing paradigm. One such paradigm is provided by the theory of quantum computation, which is based on the idea of using quantum mechanics to perform computations, instead of classical physics.

Since the landmark Nature paper in 2001 [26] describing a technology for scalable quantum computation (part of the work for which Wineland received the Nobel prize in physics in 2012), interest in practical computation has grown significantly. Recent announcements by commercial effort D-Wave of a 128-quantum bit (qubit) adiabatic system [54], although controversial in terms of its quantum properties, illustrate the engineering progress that has been made.

1.1 Quantum Computers

It is tempting to say that a quantum computer is one whose operation is governed by the laws of quantum mechanics. But quantum mechanics being a complete natural theory, its laws govern the behaviour of all physical phenomena, even classical computers. So this temptation must be resisted. What distinguishes quantum from classical computers is that operation of the former is based on the two distinctively quantum-mechanical effects of interference and entanglement that do not appear in classical physics. Thus a quantum computer is an interference device of many entangled computation paths. Just as an interference pattern can appear by preparing a particle in a superposition of different geometric paths which are then combined to interfere, the output of a quantum computer is obtained by preparing the quantum bits in a superposition of different classical computation states which are also combined to interfere producing the final computation answer.

A classically intractable problem is simulating quantum systems. A single spin - 1/2 particle, such as an electron trapped in a quantum dot, has a 2-D space of states, which can be considered to describe the direction of its spin. A similar classical particle such as a Heisenberg spin will also have a 2-D state space. However, n quantum particles have a 2^n - D state space, while n classical Heisenberg spins will only have a $2n$ - D space of states. Thus quantum systems are difficult to simulate classically because they generically utilize the full 2^n - D Hilbert space as they evolve, requiring exponential classical resources. This led Feynman [3] to conjecture that a quantum computer, which used quantum mechanics intrinsically might be more powerful than a computer mired in the classical world.

One of the most spectacular instances of the power of quantum computers to solve classically intractable problems is Shor's discovery of an algorithm to factor numbers on quantum computer in polynomial time in the number of bits [5, 21]. This could be used to break the RSA cryptosystem, which is one of the most widely deployed public key cryptosystems. Another impressive algorithm is Grover's algorithm [11], which can find a single object in an unassorted database of N objects in $O(\sqrt{N})$ time on a quantum computer, while the same task would require an exhaustive search on a classical computer, taking $O(N)$ time. It has been shown that $O(\sqrt{N})$ time is the best possible speed for this task [18].

1.2 Physical Limitations of Quantum Computers : Decoherence

While the power of quantum superposition enables a lot of interesting computing, it comes at a cost. In a quantum computer the physical systems that encode the individual logical bits must have no physical interactions with whatever that are not under the complete control of the program. All other interactions, however irrelevant they might be in a classical computer, introduce potentially catastrophic disruptions into the operations of a quantum computer. Such damaging encounters can include interactions with the external environment, such as air molecules bouncing off the physical systems that represent bits, or the absorption of minute amounts of ambient radiant thermal energy. There can even be disruptive interactions between the computationally relevant features of the physical systems that represent bits and other features of those same systems that are associated with computationally irrelevant aspects of their internal structure. Such destructive interactions between what matters for the computation and what does not, result in *decoherence*, which is fatal to a quantum computer. Quantum decoherence is the loss of coherence or ordering of the phase angles between the components of a system in a quantum superposition. These problems serve as an obstacle towards the eventual construction of a robust large scale quantum computer [6, 7, 9]. If left unchecked, these problems turn a quantum computer into a classical information processing device, or even worse, into a machine which can enact no computation at all.

For this reason, error rates for all quantum operations are higher than classical ones. Error rates to do anything in any quantum computing technology in the lab right now are in the range of 10^{-2} - 0.1 errors per operation. This includes having qubits wait around, not doing anything. Realistic estimates for error rates in the foreseeable future are said to be around 10^{-5} - 10^{-2} errors per operation. In contrast, CMOS transistor error rates range from 10^{-20} to 10^{-15} errors per gate.

To avoid decoherence individual bits cannot in general be encoded in physical systems of macroscopic size, because such systems (except under very special circumstances) cannot be isolated from their own irrelevant internal properties. Such isolation can be achieved if the bits are encoded in a small number of states of a system of atomic size, where extra internal features do not matter, either because they do not exist, or because they require unavailably high energies to come into play. Such atomic scale systems must also be decoupled from their surroundings except for the completely controlled interactions that are associated with the computation process itself.

Two things keep the situation from being hopeless. First, because the separation between the discrete energy levels of a system on the atomic scale can be enormously larger than

the separation between the levels of a large system, the dynamical isolation of an atomic system is easier to achieve. It can take a substantial kick to knock an atom out of its state of lowest energy. The second reason for hope is the discovery that errors introduced by extraneous interactions can actually be corrected if they occur at a sufficiently low rate, that is, under reasonable physical assumptions, a fault-tolerant quantum computer can be built.

1.2.1 Error Correction and Fault Tolerance

A central insight used in the theory of fault-tolerant quantum computation is that quantum information can be encoded into quantum error-correcting codes. In classical computation, error-correcting codes encode a message by adding enough redundancy such that even if the encoded message is corrupted with noise, it is possible to decode or recover the information in the original message. Somewhat analogously, quantum error-correcting codes spread information across many physical qubits of a system and protect the quantum information from undesired effects which cause a lot of quantum decoherence.

However, there are some fundamental differences between classical and quantum information that must be kept in mind while developing these codes. [51]

- *No cloning:* This forbids the implementation of repetition codes quantum mechanically.
- *Continuous errors:* Since a continuum of different errors can occur on a single qubit, determining which particular error occured would require infinite precision and hence infinite resources.
- *Measurement destroys quantum information:* In classical error-correction output from the channel is observed and accordingly different error-correcting steps can be applied. Observation or measurement in quantum mechanics generally irreversibly destroys quantum states.

A simple two-stage process can be used to recover the correct quantum state.

1. **Error-detection or Syndrome diagnosis:** A measurement is performed which tells which error, if any, occurred on the quantum state. The measurement result is called the *error syndrome*. The syndrome contains only information about what error has occurred and no information about the state being protected, because to obtain the latter it is in general necessary to perturb the state.

2. **Recovery:** The value of the error syndrome tells what procedure to use to recover the initial state.

An important result of quantum error correction is *discretization of errors*, that gives us some relief against the continuous errors. This states that, to fight the continuum of errors possible on a single qubit it is sufficient merely to win the war against a finite set of errors, the four Pauli matrices (described in Chapter 2). Similar results hold for higher dimensional quantum systems. This stands in remarkable contrast to the theory of error correction for analog classical systems. Error correction in these systems is extremely difficult because in principle there are an infinite number of different error syndromes. Digital error-correction for classical information processing is much more successful because it involves a finite number of error syndromes. Thus quantum error-correction seems more similar to classical digital error-correction than it is to classical analog error-correction.

Some of the popular quantum error correcting codes (QECCs) are Shor code, Steane code and Knill code. These are described in Chapter 2.

Fault tolerant quantum computing aims at computing directly on encoded quantum states in such a manner that decoding is never required. Fault tolerance of a procedure is defined [51] to be the property that if only one component in the procedure fails then the failure causes at most one error in each encoded block of qubits output from the procedure. For example, the failure of a single component in a fault-tolerant recovery procedure for quantum error-correction results in the recovery procedure being performed correctly, up to an error on a single qubit of the output. 'Component' means any of the elementary operations used in the encoded gate, which might include noisy gates, noisy measurements, noisy quantum wires and noisy state preparations. Analogously, a procedure for measuring an observable on a set of encoded qubits is said to be fault-tolerant if the failure of any single component in the procedure results in an error in at most one qubit in each encoded block of qubits at the output of the procedure. Furthermore, we require that if only one component fails then the measurement result reported must have probability of error of $O(p^2)$, where p is the (maximum) probability of a failure in any one of the components used to implement the measurement procedure. Similarly, a procedure for preparing a fixed encoded sate is said to be fault-tolerant if, given that a single component failed during the procedure, there is at most a single qubit in error in each encoded block of qubits output from the procedure.

1.3 Motivation of Work and Scope

Ideally, quantum error correction must be carried out after every component. But that entails a tremendous increase of resources. So we do an error analysis of quantum circuits. Our aim is to reduce the resources by placing the error correction sub-circuit judicially. For this, we trace the propagation of error probability in a quantum circuit at the logical level. We build the tile structures of fault-tolerant gates encoded with different QECCs. We propose a model to calculate the error probability of these gates at the logical level and use these values in error tracing. When the probability of error in the circuit crosses a certain threshold, we place an error correction sub-circuit.

1.4 Contribution of this Thesis

Our contributions can be summarized as follows:

- We have developed the tile structure of fault-tolerant gates, encoded with different QECCs and designed a model to calculate the error probability of each such gate at the physical level as well as at the logical level.
- We have built an algorithm to trace the propagation of error probability in a logical quantum circuit.
- We have designed a methodology of fixing thresholds and applying quantum error correcting sub-circuits, which results in significant reduction of resources.
- We have implemented our algorithm in C and developed a tool for calculating error probability of gates at physical and logical level, tracing error probability in a logical quantum circuit and placing error-correction sub-circuit selectively.

1.5 Organization

In Chapter 2 we give some background knowledge about quantum mechanics, quantum computing, error correction and fault tolerance and then we proceed to describe our methodology for tracing error probability in quantum circuits in Chapter 3. The model for calculating error probability of the gates at physical and logical level has been detailed in Chapter 4. Results have been given in Chapter 5 and we end with some concluding remarks in Chapter 6.

Chapter 2

Preliminary Knowledge

Quantum computing is an interdesciplinary field, encompassing physics, mathematics and computer science. The background knowledge which people from one field takes for granted, may be unfamiliar to others. So in this chapter we provide a brief preliminary knowledge of quantum mechanics, quantum computing, error correcting codes and fault tolerance.

2.1 Quantum Mechanics

2.1.1 Qubits

The fundamental concept of classical computation and classical information is the *bit*, which has can be in two states - 0 or 1. Analogously, the simplest quantum mechanical system is the *qubit*, which has a 2-D state space and is represented by a unit state vector. Let $|0\rangle$ and $|1\rangle$ form an orthonormal basis for that state space. Then an arbitrary state vector in that state space can be written as :

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle \qquad \text{where } a \text{ and } b \text{ are complex numbers} \tag{2.1}$$

For $|\psi\rangle$ to be a unit vector, $\langle\psi\,|\,\psi\rangle = 1$, where $\langle\alpha\,|\,\beta\rangle$ represents *inner product* of two vectors α and β. This impies $|a|^2 + |b|^2 = 1$. Similarly an n - qubit system has 2^n *computational basis states* and can have a state which is a linear combination of these basis states. This gives rise to the continuum of quantum states.

This way a qubit differs from a classical bit, that is, it can exist in *superposition* of states. Any linear combination $\sum_i \alpha_i\,|\psi_i\rangle$ is a superposition of the states $|\psi_i\rangle$ with amplitude α_i. The probability that the state of the qubit after measurement happens to be $|\psi_i\rangle$ is

$|\alpha_i|^2$. The condition that the probabilities sum to 1 is expressed by the *normalization condition* $\sum_i |\alpha_i|^2 = 1$. Example of superposed states are $\frac{|0\rangle \pm |1\rangle}{\sqrt{2}}$, which are often called the *Hadamard basis states*.

There are multiple qubit states that cannot be written as product of single qubit states. These are called *entangled states*. An important two qubit entangled state is the *Bell state* or *EPR pair*, $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$

2.1.2 Bloch Sphere Representation

An useful way of thinking about qubits is the geometric representation of Bloch sphere. Since normalization conditions hold in equation(2.1), $|a|^2 + |b|^2 = 1$ and it maybe re-written as:

$$|\psi\rangle = e^{i\gamma}(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle) \qquad \text{where } \theta, \varphi, \gamma \in \mathbf{R} \tag{2.2}$$

The factor $e^{i\gamma}$ can be ignored because it has no observable effects. Thus equation(2.2) can be effectively written as:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \tag{2.3}$$

The numbers θ and φ define a point on an unit 3-D sphere, called the *Bloch sphere* (figure(2.1)). It offers an useful way of visualizing the state of a single qubit. But the intuition is limited because there is no simple generalization of the Bloch sphere for multiple qubits.

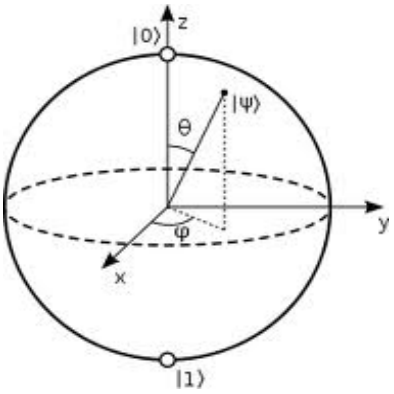


FIGURE 2.1: Bloch sphere representation of a qubit

2.1.3 Postulates of Quantum Mechanics

The postulates provide a connection between the physical world and the mathematical formalism of quantum mechanics. These are stated as follows with brief explainations:

Postulate 1**:** Associated to any isolated physical system is a complex vector space with inner product (that is, Hilbert space) known as the *state space* of the system. The system is completely described by its *state vector*, which is a unit vector in the system's state space.

Postulate 2**:** The evolution of a *closed* quantum system is described by a *unitary transformation*. That is, the state $|\psi\rangle$ of the system at time t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2,

$$|\psi'\rangle = U\,|\psi\rangle \tag{2.4}$$

Postulate 2′**:** The time evolution of the state of a closed quantum system is described by the Schrödinger equation,

$$i\hbar\frac{d\,|\psi\rangle}{dt} = H\,|\psi\rangle \qquad \text{where } \hbar \text{ is Planck's constant} \tag{2.5}$$

H is a fixed Hermitian operator known as the *Hamiltonian* of the closed system. Since the Hamiltonian is a Hermitian operator it has a spectral decomposition

$$H = \sum_E E\,|E\rangle\langle E| \qquad \text{with eigenvalues } E \text{ and corresponding normalized eigenvectors } |E\rangle \tag{2.6}$$

The states $|E\rangle$ are conventionally referred to as *energy eigenstates*, or *stationary states* and E is the energy of the state $|E\rangle$.

For simplicity, let H is independent of time, then equation(2.5) can be solved as:

$$|\psi(t)\rangle = e^{-iH(t-t_0)/\hbar}\,|\psi(t_0)\rangle = U\,|\psi(t_0)\rangle \tag{2.7}$$

where U is a unitary operator.

Postulate 3: Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that the result m occurs is given by :

$$p(m) = \langle\psi|\,|M_m^\dagger M_m|\,|\psi\rangle \tag{2.8}$$

and the state of the system after measurement is

$$\frac{M_m\,|\psi\rangle}{\sqrt{\langle\psi|\,|M_m^\dagger M_m|\,|\psi\rangle}} \tag{2.9}$$

The measurement operators satisfy the *completeness equation*,

$$\sum_m M_m^\dagger M_m = I \tag{2.10}$$

The completeness equation expresses the fact that probabilities sum to one:

$$1 = \sum_m p(m) = \sum_m \langle\psi| \, |M_m^\dagger M_m| \, |\psi\rangle \tag{2.11}$$

2.1.4 Indistinguishibility of non-orthogonal quantum states

A proof by construction shows that no measurement distinguishing the orthogonal states $|\psi_1\rangle$ and $|\psi_2\rangle$ is possible.
Let such a mesurement is possible. Let $f(.)$ is a rule that determines the outcome of the measurement. If the state $|\psi_1\rangle$ ($|\psi_2\rangle$) is prepared then the probability of measuring j such that $f(j) = 1 (f(j) = 2)$ must be 1.
Defining $E_i \equiv \sum_{j:f(j)=i} M_j^\dagger M_j$, these observations may be written as:

$$\langle\psi_1 \,|\, E_1 \,|\, \psi_1\rangle = 1; \qquad \langle\psi_2 \,|\, E_2 \,|\, \psi_2\rangle = 1 \tag{2.12}$$

Since $\sum_i E_i = I$ it follows that $\sum_i \langle\psi_1 \,|\, E_i \,|\, \psi_1\rangle = 1$, and since $\langle\psi_1 \,|\, E_1 \,|\, \psi_1\rangle = 1$ we must have $\langle\psi_1 \,|\, E_2 \,|\, \psi_1\rangle = 0$, and thus $\sqrt{E_2} \,|\psi_1\rangle = 0$.
Let we decompose $|\psi_2\rangle = \alpha \,|\psi_1\rangle + \beta \,|\varphi\rangle$, where $|\varphi\rangle$ is orthonormal to $|\psi_1\rangle$, $|\alpha|^2 + |\beta|^2 = 1$, and $|\beta| < 1$ since $|\psi_1\rangle$ and $|\psi_2\rangle$ are not orthogonal.
Then $\sqrt{E_2} \,|\psi_2\rangle = \beta\sqrt{E_2} \,|\varphi\rangle$, which implies a contradiction with (2.12), as

$$\langle\psi_2 \,|\, E_2 \,|\, \psi_2\rangle = |\beta|^2 \langle\varphi \,|\, E_2 \,|\, \varphi\rangle \leq |\beta|^2 < 1$$

where the second inequality follows from the observation that

$$\langle\varphi \,|\, E_2 \,|\, \varphi\rangle \leq \sum_i \langle\varphi \,|\, E_i \,|\, \varphi\rangle = \langle\varphi \,|\, \varphi\rangle = 1$$

2.1.5 No Cloning Theorem

The No Cloning theorem [4] states that it is impossible to make a copy of an arbitrary quantum state. The proof is straightforward [19] :
Let we wish to have an operation that maps an arbitrary state $|\psi\rangle \to |\psi\rangle \otimes |\psi\rangle$.
Then arbitrary $|\phi\rangle$ is mapped by $|\phi\rangle \to |\phi\rangle \otimes |\phi\rangle$.

Because the transformation must be linear it follows that

$$|\psi\rangle + |\phi\rangle \rightarrow |\psi\rangle \otimes |\psi\rangle + |\phi\rangle \otimes |\phi\rangle$$

However

$$|\psi\rangle \otimes |\psi\rangle + |\phi\rangle \otimes |\phi\rangle \neq (|\psi\rangle + |\phi\rangle) \otimes (|\psi\rangle + |\phi\rangle)$$

So we have failed to copy $|\psi\rangle + |\phi\rangle$.
In general, if we pick an orthonormal basis, we can copy the basis states but we will not have correctly copied superpositions of those basis states.

2.2 Introduction to Quantum Computing

Changes occuring to a quantum state can be described using the language of quantum computation. From equation(2.7), all valid quantum operations are unitary. The evolution of an isolated quantum system with a finite number of states can be described by a unitary matrix and thus is reversible. *Reversibility is a necessary condition for qunatum computing.*

Quantum circuits can be represented by space-time diagrams. In these diagrams, time usually progresses from left to right. The circuit comprises of a sequence of quantum gates, either in series or parallel. An n - qubit gate or operation is represented by a $2^n \times 2^n$ unitary matrix. The net unitary transformation performed is computed by composing the unitary matrices of the corresponding quantum gates. If several gates act on the same subset of qubits, then they must be applied in series and their overall effect is computed by the dot product. If adjacent gates within a quantum circuit act on independent subset of qubits, then they can be applied in parallel and the net effect is the tensor product of the unitary matrices.

2.2.1 Quantum Gates

In this subsection we present definitions and discuss properties of some fundamental one-qubit, two-qubit and three-qubit operations and the corresponding quantum gates [58], that are used to build quantum circuits for information processing. It must be borne in mind, that due to no-cloning principle quantum circuits do not have any fanout or feedback mechanism and thus can be represented by an acyclic graph.

2.2.1.1 One-qubit Gates

A one-qubit gate can be represented by a 2×2 unitary matrix. Some one-qubit gates are listed below.

- **Global Phase Gate:** The global phase gate, P, is defined as:

$$P(\theta) = e^{i\theta}\dot{I} \tag{2.13}$$

  where I denotes the identity matrix, which indicates that no operation is performed.
  Remark: The global phase gate is physically indistinguishable and hence is not physically implemented. But it is useful to match circuit identities.

- **Pauli Gates:** The Pauli spin matrices for the x, y and z axes, corresponding to the Pauli Gates X, Y and Z are respectively:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.14}$$

- **Rotation Gates:** From equation(2.7), evolution of a quantum operation depends on the exponentiation of the Hermitian matrix. This leads to the definition of rotation gates, which represent rotation around the different axes.

$$R_x(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} R_y(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & \sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} R_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} \tag{2.15}$$

  X, Y and Z can be regarded as special cases of R_x, R_y and R_z respectively with rotation angles of π. The rotation gates can be defined in terms of the Pauli gates as follows:

$$R_j(\theta) = e^{\frac{-i\theta A}{2}} = \cos(\frac{\theta}{2})I - i\sin(\frac{\theta}{2})A, \quad j \in \{x, y, z\}, \quad A \in \{X, Y, Z\} \tag{2.16}$$

  The periods of R_x, R_y and R_z are 4π.

$$R_i(\theta) = R_i(\theta \pm 4\pi) = -R_i(\theta \pm 2\pi), \qquad i \in \{x, y, z\} \tag{2.17}$$

- **H Gate:** The Hadamard or H gate is defined as follows:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.18}$$

- **S and T Gates:** The S and T gates are defined as follows:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}, T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \tag{2.19}$$

  S and T are special cases of R_z with rotation angles of $\frac{\pi}{2}$ and $\frac{\pi}{4}$ respectively. The following relations hold true:

$$Z = S^2; \qquad S = T^2 \tag{2.20}$$

- **R_{xy} and Asqu Gates:** Along with single-axis rotation operators, some multi-axis one-qubit rotation operators also exist.

$$R_{xy}(\theta, \phi) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -e^{-i(\phi+\frac{\pi}{2})}\sin(\frac{\theta}{2}) \\ e^{i(\phi+\frac{\pi}{2})}\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} \tag{2.21}$$

$$Asqu(\phi_1, \theta, \phi_2) = \begin{pmatrix} e^{i(\frac{\phi_1+\phi_2}{2})}\cos(\frac{\theta}{2}) & -e^{-i(\frac{\phi_1-\phi_2}{2})}\sin(\frac{\theta}{2}) \\ e^{i(\frac{\phi_1-\phi_2}{2})}\sin(\frac{\theta}{2}) & e^{-i(\frac{\phi_1+\phi_2}{2})}\cos(\frac{\theta}{2}) \end{pmatrix} \tag{2.22}$$

2.2.1.2 Two-qubit Gates

The physical interactions available within different types of quantum systems can give rise to different two-qubit operations and corresponding gates. These are described by 4×4 unitary matrices.

One of the most useful operations for both classical and quantum computing are the *controlled operations*. They act on two qubits - a *control* qubit and a *target* qubit. Suppose U is an arbitrary single-qubit operation. For the controlled-U (CU) operation, if the control qubit c is set, then U is applied to the target qubit t, else the target qubit t is left alone. That is,

$$|c\rangle |t\rangle \rightarrow |c\rangle U^c |t\rangle \tag{2.23}$$

In matrix notation,

$$CU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & U_{10} & U_{11} \end{pmatrix} \tag{2.24}$$

- **CNOT Gate:** This is a specific type of CU gate with U = X. It is defined as:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.25}$$

  In terms of the computational basis, the action of the CNOT is given by $|c\rangle |t\rangle \rightarrow |c\rangle |t \otimes c\rangle$

- **CP Gate:** It leads to a relative phase rotation $e^{i\theta}$ between the two qubits when the control qubit is $|1\rangle$. In fact, it is symmetric with respect to both the qubits. So there is no need to distinguish between the control and target qubit.

$$CP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix} \tag{2.26}$$

- **CZ Gate:** It is a special type of CP gate with $\theta = \pi$.

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{2.27}$$

- **iSWAP Gate:** iSWAP or iSW is not a controlled operation. It is also symmetric.

$$iSW(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -i\sin(\theta) & 0 \\ 0 & -i\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.28}$$

- **ZENO Gate:** ZENO is a special kind of iSW with $\theta = \frac{\pi}{2}$.

$$ZENO = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.29}$$

- **SWAP Gate:** This is a symmetric gate and is defined as follows.

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.30}$$

- **Geometric (G) Gate:** This symmetric gate applies a geometric phase to two qubits.

$$G(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.31}$$

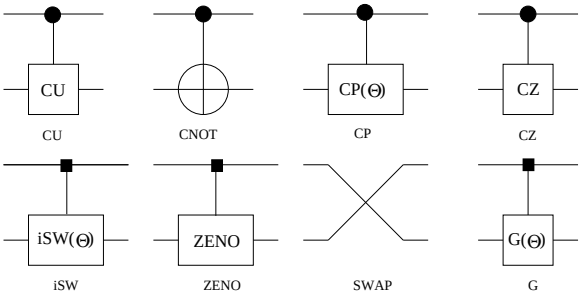


FIGURE 2.2: Two qubit gates

2.2.1.3 Three-qubit Gates

Three-qubit reversible gates provide a higher abstraction level for circuit description. Because interactions among more than two qubits are difficult to implement, the three-qubit gates must be decomposed into two-qubit and one-qubit gates. Some commonly used three-qubit gates are **Toffoli, Fredkin** and **Peres gates** (Figure 2.3).

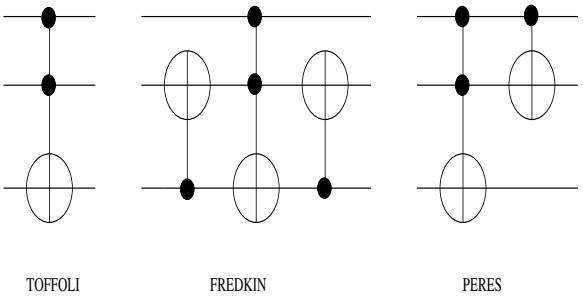


FIGURE 2.3: Three qubit gates

2.2.2 Universal Quantum Gate Library

A small set of gates (e.g. AND, OR, NOT) can be used to compute an arbitrary classical function and are thus called universal for classical computation. A similar universality result is true for quantum computation, where a set of gates is said to be *universal for quantum computation* if any unitary operation may be approximated to arbitrary accuracy by a quantum circuit involving only those gates [51]. Since the number of possible quantum gates is uncountable, whereas number of finite sequences of elements from a finite set is a countable set, exact realization of all unitary operations is not possible. The **Solovay-Kitaev theorem** states that efficient approximations of an arbitrary unitary with elements from a finite set is possible. Specifically, it implies that an arbitrary single qubit gate may be approximated to an accuracy ϵ using $O(log^c(1/\epsilon))$ gates from any discrete set [36].

A number of universal gate sets have been introduced in literature like NCT (NOT, CNOT, Toffoli). But the most popular universal gate set with provable fault tolerant implementation is the CTL gate set. The *Clifford* gate library includes H, S, X and Z gates for one-qubit circuits. For multiple-qubit circuits CNOT gate is included. To make the gate set universal the T gate has been added.

2.3 Physcial Machine Description (PMD)

Physical realization of quantum computers not only requires a robust physical representation of qubits, but also the enabling of their time evolutionas desired. In quantum mechanics, the time evolution of a closed quantum system is described by a unitary operator determined by its Hamiltonians (Equation 2.5). Since different quantum systems have different Hamiltonians, they also have different Physical Machine Descriptions (PMD) [58]. To perform a quantum operation, one must be able to control the Hamiltonians of the system. However, an operation may be easily performed in one system but with difficulty in another. Hence, one PMD may be more suitsble for implementing a quantum logic gate than another.

We have considered the PMDs of the following six promising quantum systems.

1. **Quantum Dot (QD):** Here, a qubit is represented by the spin states of two electrons ina double electrostatically defined quantum dot, whcih has two potential wells with a tunneling barrier between them. [43]

2. **Neutral Atom (NA):** This is a system of trapped neutral atoms that can be isolated from the environment and whose simple quantum-level structure can be exploited. [39, 52, 55]

3. **Linear Photonics (LP):** In this quantum system, a probabilistic two-photon gate is teleported into a quantum circuit with high probability. [25, 32, 42, 44]

4. **Non-Linear Photonics (NLP):** This quantum system is based on weak cross-Kerr non-linearities. [29, 38]

5. **Super-Conducting (SC):** In this system, charged carriers are used to represent qubits. [30, 35, 47, 48, 50, 53]. At low temperatures in certain metals, two electrons can bind together to form a Cooper pair. Such a pair can be confined within an electrostatic box and used to represent quantum information.

6. **Ion Trap (IT):** This quantum system is based on a 2-D lattice of confined ions, each of which represents a physical qubit that can be moved within the lattice to accomodate local interactions. [28, 34]

The primitive quantum operations realized in each of the six PMDs is shown in Table 2.1. Since for a particular PMD, the non-primitive gates have a higher cost than the primitive ones, it will be useful if after chosing a particular PMD, the non-primitive gates of a quantum circuit is realized with the primitive ones.

TABLE 2.1: Supported operations in different PMDs

PMD	One-qubit operation	Two-qubit operation
QD	R_x, R_z, X, Z, S, T	CZ
NA	R_{xy}	CZ
LP	$R_x, R_y, R_z, X, Y, Z, S, T, H$	$CNOT, CZ, SWAP, ZENO$
NLP	$Asqu, R_x, R_y, R_z, H$	$CNOT$
SC	R_x, R_y, R_z	$iSWAP, CP$
IT	R_{xy}, R_z	G

2.4 Quantum Error Correcting Code (QECC)

Noise is one of the important banes of information processing, be it classical or quantum. Quantum systems, as already discussed in Chapter 1 are more prone to errors due to decoherence. In classical computing, this problem is countered with the help of error correcting codes. Such codes are also possible in the quantum domain but we have to be careful about some aspects like no-cloning, continuum of errors and measurements, destroying information (Chapter 1). In this section we introduce three very popular

quantum error correcting codes (QECCs) that we have used intensively throughout or work.

Let us consider the properties of more general codes [19]. A code to encode k qubits in n qubits will have 2^k basis codewords corresponding to the basis of the original states. For linear codes, any linear combination of these basis codewords is also a valid codeword, corresponding to the same linear combination of the unencoded basis states. The space T of valid codewords (the *coding space*) is therefore a Hilbert space in its own right, a subspace of the full $2^n - D$ Hilbert space. One convenient basis to use is the set of tensor products of X, Y, Z and I. The *weight* of an operator of this form is the number of qubits on which it differs from the identity. The set of all these tensor products with a possible overall factor of $\pm 1, \pm i$ forms tne *Pauli group* under multiplication. For n qubits, we denote this group by G_n. Thus:

$$G_1 \equiv \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\} \tag{2.32}$$

We note that since $X^2 = Y^2 = Z^2 = I$, every element in G_1 square to $\pm I$. Also X, Y and Z on the same qubit anti-commute, while they commute on different qubits. Therefore, any two elements of G_1 either commute or they anti-commute. X, Y and Z are all Hermitian but of course $(iI)^\dagger = -iI$, so elements of G_1 can be either Hermitian or anti- Hermitian. In either case, if $A \in G_1, A^\dagger \in G_1$ also. Similarly, X, Y and Z are all unitary, so every element of G_1 is unitary.

We begin by describing a wide class of error-correcting codes, called the stabilizer codes, sometimes known as *additive quantum codes*, whose construction is analogous to classical linear codes.

Stabilizer Codes: [19, 51] Let S is a subgroup of G_n and define V_S to be the state of n qubit states which are fixed by every element of S. V_S is the *vector space stabilized by* S, and S is said to be the *stabilizer* of the space V_S, since every element of V_S is stable under the action of elements in S. For example, consider the case with $n = 3$ qubits and $S \equiv \{I, Z_1Z_2, Z_2Z_3, Z_3Z_1\}$. The subspace fixed by Z_1Z_2 is spanned by $|000\rangle, |001\rangle, |110\rangle$ and $|111\rangle$, and the subspace fixed by Z_2Z_3 is spanned by $|000\rangle, |100\rangle, |011\rangle$ and $|111\rangle$. Similarly considering the subspaces spanned by all the operator in S, we find $|000\rangle$ and $|111\rangle$ are common elements and thus V_S must be the subspace spanned by the states $|000\rangle$ and $|111\rangle$.

Calderbank-Shor-Steane (CSS) Code: CSS codes [10, 13, 14, 51](named after the initials of the inventors) are an important subclass of stabilizer codes. Let C_1 and C_2 are $[n, k_1]$ and $[n, k_2]$ classical linear codes such that $C_2 \subset C_1$ and C_1 and $C_2^\perp$ both correct t errors. We define an $[n, k_1 - k_2]$ quantum code $CSS(C_1, C_2)$ capable of correcting errors

on t qubits, the *CSS code of* C_1 *over* C_2, via the following construction. Let $x \in C_1$ is any codeword in the code C_1. Then we define the quantum state $|x + C_2\rangle$ by

$$|x + C_2\rangle \equiv \frac{1}{\sqrt{|C_2|}} \sum_{y \in C_2} |x + y\rangle \tag{2.33}$$

where $+$ is bitwise addition modulo 2.

In the following subsections we describe three important QECCs and corresponding logic level implementation.

2.4.1 [9,1,3] Shor Code

The [9,1,3] Shor Code [8, 19, 51] is a stabilizer code that can protect against the effects of an arbitrary error on a single qubit. The stabilizers for Shor code has been given in Table 2.2. It is a combination of three qubit phase flip and bit flip codes. First a qubit is encoded using the phase flip code: $|0\rangle \rightarrow |+++\rangle, |1\rangle \rightarrow |---\rangle$. Next, each of these qubits is encoded using the three qubit bit flip code: $|+\rangle \rightarrow \frac{(|000\rangle+|111\rangle)}{\sqrt{2}}, |+\rangle \rightarrow \frac{(|000\rangle+|111\rangle)}{\sqrt{2}}$. The result is a nine qubit code, with codewords given by:

$$\begin{aligned} |0\rangle \rightarrow |0_L\rangle &\equiv \frac{(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)}{2\sqrt{2}} \\ |1\rangle \rightarrow |1_L\rangle &\equiv \frac{(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)}{2\sqrt{2}} \end{aligned} \tag{2.34}$$

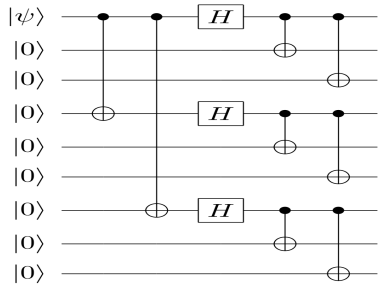


FIGURE 2.4: Quantum encoding circuit for Shor [9,1,3] code

Let a bit flip occurs on the first qubit. We compare the first two qubits by measuring the eigenvalue of Z_1Z_2. If qubits are same then eigenvalue is $+1$, else it is -1. In this case the outcome should be -1. We conclude that a bit flip occurred on the first or second qubit. Next we compare the second and third qubit by measuring the eigenvalue of Z_2Z_3. We find it is $+1$, so it could not have been the second qubit which flipped. We conclude that the first qubit must have flipped, and recover from the error by flipping

M_1	Z	Z	I	I	I	I	I	I	I
M_2	Z	I	Z	I	I	I	I	I	I
M_3	I	I	I	Z	Z	I	I	I	I
M_4	I	I	I	Z	I	Z	I	I	I
M_5	I	I	I	I	I	I	Z	Z	I
M_6	I	I	I	I	I	I	Z	I	Z
M_7	X	X	X	X	X	X	I	I	I
M_8	X	X	X	I	I	I	X	X	X

TABLE 2.2: Stabilizers for Shor's 9-qubit code

the first qubit again back to its original state, i.e. applying a X operator. In a similar way we can detect and recover from the effects of bit flip errors on any of the nine qubits in the code.

In a similar way phase flip errors are detected and recovered. Let a phase flip changes the sign of the first block of qubits: $(|000\rangle + |111\rangle) \rightarrow (|000\rangle - |111\rangle)$ and vice versa. Syndrome measurement begins by comparing the sign of the first and second blocks of three qubits, and then the sign of the second and third block of qubits, i.e. measuring the eigenvalue of observables $X_1X_2X_3X_4X_5X_6$ and $X_4X_5X_6X_7X_8X_9$. In this case, the first two blocks differ in sign and we conclude that either the first or second block has changed sign. The last two blocks are same in sign and we confirm that it is the first block that has changed sign. We recover from this by flipping the sign of the erroneous block, in this case, applying a $Z_1Z_2Z_3$ operator.

Let both bit and phase flips occur on the first qubit, i.e. the operator Z_1X_1 is applied to that qubit. Then it is easy to see that the procedure for detecting a bit flip error will detect a bit flip on the first qubit, and correct it, and the procedure for detecting a phase flip error will detect a phase flip on the first block of three qubits, and correct it. Thus, the Shor code also enables the correction of combined bit and phase flip errors on a single qubit.

2.4.2 [7,1,3] Steane Code

The Steane code (C) [13, 19, 23, 51], known after its inventor, is an example of CSS code were $C_1 \equiv C$ and $C_2 \equiv C^{\perp}$. It is based on the classical [7,4,3] Hamming code, which is self-dual. The stabilizers of Steane code have been listed in Table 2.3. Therefore, the

logical states $|0_L\rangle$ and $|1_L\rangle$ can be written as follows:

$$\begin{aligned} |0\rangle \rightarrow |0_L\rangle &\equiv \frac{1}{\sqrt{8}}(|0000000\rangle + |1111000\rangle + |1100110\rangle + |1010101\rangle \\ &+ |0011110\rangle + |0101101\rangle + |0110011\rangle + |1001011\rangle) \\ |1\rangle \rightarrow |1_L\rangle &\equiv \frac{1}{\sqrt{8}}(|1111111\rangle + |0000111\rangle + |0011001\rangle + |0101010\rangle \\ &+ |1100001\rangle + |1010010\rangle + |1001100\rangle + |0110100\rangle) \end{aligned} \tag{2.35}$$

The encoded $|0_L\rangle$ state is the superposition of the even codewords in the Hamming code and the encoded $|1_L\rangle$ state is the superposition of the odd codewords in the Hamming codeword. The quantum encoding circuit for Steane code has been shown in Figure 2.5

g_1	I	I	I	X	X	X	X
g_2	I	X	X	I	I	X	X
g_3	X	I	X	I	X	I	X
g_4	I	I	I	Z	Z	Z	Z
g_5	I	Z	Z	I	I	Z	Z
g_6	Z	I	Z	I	Z	I	Z

TABLE 2.3: Stabilizers for Steane's 7-qubit code

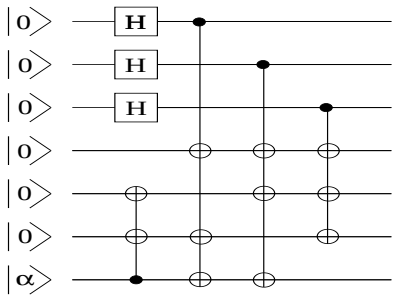


FIGURE 2.5: Quantum encoding circuit for Steane [7,1,3] code

2.4.3 [9,1,3] Bacon-Shor Code

The Shor and Steane code described above, are subspace codes, that is, quantum information is encoded into a subspace of the Hilbert space of many quantum systems. However, a more general approach to perform quantum error correction is to use quantum error correcting subsystems. While every quantum error correcting subsystem can be turned into a subspace code, subsystems codes differ significantly in how quantum error correction is performed on the encoded quantum information. Of particular significance is that quantum error correcting subsystems can significantly reduce the number of stabilizer measurements needed during their quantum error recovery routine. This, in turn, can lead to significantly improved thresholds for fault-tolerant quantum computation.

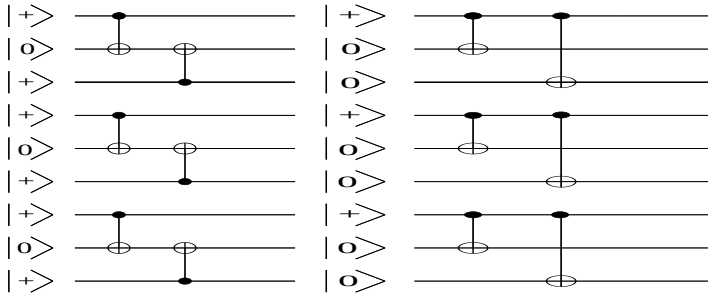
The [9,1,3] Bacon-Shor code [40] is the subsystem version of the class of codes arising from generalizing Shor's code. In the latter, a bit flip error-correcting code is concatenated with a phase flip error-correcting code or vice versa. The order of these concatenations present an asymmetry in the recovery routing for these codes. If, for example, a bit flip code is used on the lowest level, then the recovery procedure for the bit flip code must be enacted for every lowest level code, whereas the phase flip code recovery routine needs only be enacted on the next level of the code. In Bacon-Shor code this asymmetry between the bit flips and the phase flips in the recovery routine is removed. This simplifies the recovery routine but maintains the protection of the Shor code.

The stabilizers can be better viewed as a 3×3 array, as shown in Table 2.4

$$g_1 = \begin{matrix} X & X & X \\ X & X & X \\ I & I & I \end{matrix} \qquad g_2 = \begin{matrix} I & I & I \\ X & X & X \\ X & X & X \end{matrix}$$

$$g_3 = \begin{matrix} Z & Z & I \\ Z & Z & I \\ Z & Z & I \end{matrix} \qquad g_4 = \begin{matrix} I & Z & Z \\ I & Z & Z \\ I & Z & Z \end{matrix}$$

TABLE 2.4: Stabilizers for Bacon-Shor's 9-qubit code

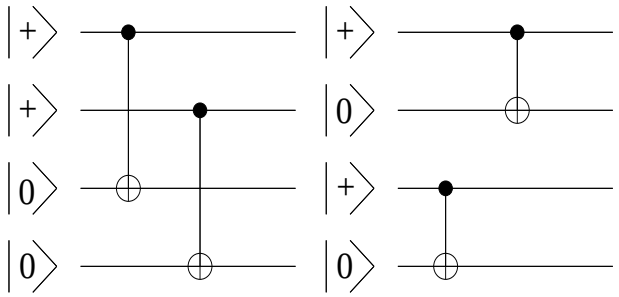


(a) Encoding $|0\rangle$ state (b) Encoding $|+\rangle$ state

FIGURE 2.6: Quantum encoding circuit for Bacon-Shor [9,1,3] code

2.4.4 [4,2,2] Knill Code

In the original scheme, Knill [37] concatenated two error-detecting codes C_4 and C_6 which alternate. We follow the simpler version, using only the C_4 code as in [33], which has a higher error threshold. More details can be found in [59]. It encodes two qubits and can simultaneously detect any single qubit X error and any single qubit Z error. This code has a stabilizer group generated by $XXXX$ and $ZZZZ$



(a) Encoding $|0\rangle$ state (b) Encoding $|+\rangle$ state

FIGURE 2.7: Quantum encoding circuit for Knill [4,2,2] code

2.5 Fault Tolerance

The analogue question of protecting classical computation from errors was already studied by von Neumann in 1956 [2], when he showed that classical computation can be made robust to noise with constant error probability per gate. This was done by using computation on redundant information, encoded by a repetition code.

The existence of QECCs is not itself sufficient to ensure the possibility of quantum computation in presence of noise. Even if error correction is performed frequently in between the computational steps to prevent accumulation of errors, additional problems occur when dealing with noisy computations that did not exist in case of noisy transmission.

- The error correction itself can introduce errors on the encoded qubits.
- The encoded gates can cause errors to propagate.

One must therefore be able to compute on encoded states by using procedures and error corrections which do not allow the errors to propagate too much. Such procedures which limit the propagation of errors are called *fault tolerant*.

Fortunately, it was discovered that under specific reasonable physical assumptions, a fault-tolerant quantum computation can be built. In particular, a set of *threshold theorems* for fault tolerant computation have been established. These theorems prove (or give a heuristic proof) that if decoherence, quantum noise and lack of control were all small enough in comparison to the ability to control the quantum system (below some threshold or thresholds), then the noisy imprecise bare devices could be efficiently put together in a fashion which decreased the failure probability of a quantum computer to any desired level.

Shor [15] showed how to design fault-tolerant procedures for a universal set of quantum gates. In order to use these fault-tolerant constructions so as to improve the reliability of the quantum circuit, one would first encode the qubits in the original circuit by using a QECC and apply quantum error correction and recovery after every fault-tolerant computation (Figure 2.8). Shor showed the resulting quantum circuit performs desired computation reliably when the error rate, or fault probability at each time step, per qubit or gate decays polylogarithmically with the size of the quantum circuit. The result is a major improvement on the performance of quantum circuits without error corrections, in which the error probability is required to decay as one over the size of the circuit in order for the computation to succeed [18]. But the assumption that the error probability decays polylogarithmically with the size of the computer, is physically unrealistic.

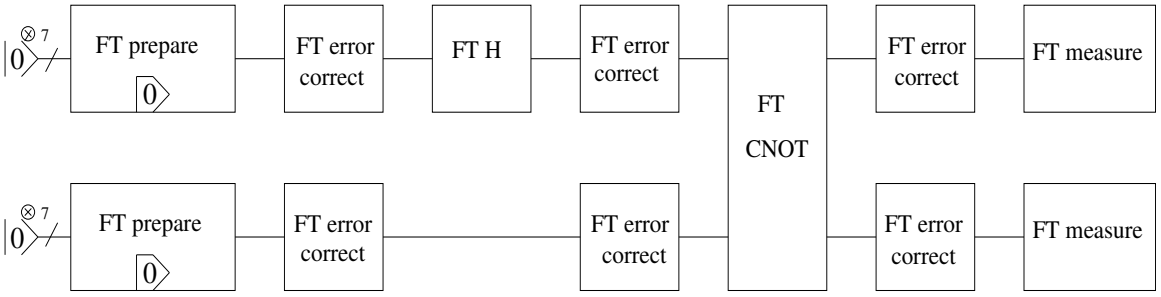


FIGURE 2.8: Fault-Tolerant implementation of a simple circuit

Concatenated codes : Construction based on concatenated codes [51] can be used to reduce the effective error rate achieved by the computation even further. The idea is to recursively apply the scheme discussed above for simulating a circuit using an encoded circuit, constructing a hierarchy of quantum circuits $C_0, C_1, C_2, \ldots$ In the first stage of this construction, each qubit in the original circuit is encoded in a quantum code whose qubits are themselves encoded in a quantum code, whose own qubits are encoded yet again, and so forth *ad infinitum*.

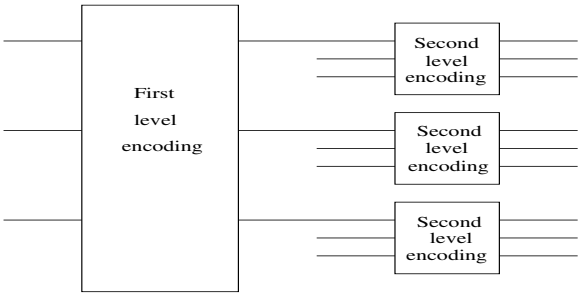


FIGURE 2.9: A two-level concatenated code

Different variants of the threshold result for probabilistic noise were independently discovered by Knill, Laflamme and Zurek, [12, 22], who used Steane's 7-qubit code and

Kitaev [20], who used toric code. All of these works are based on the same idea of applying one scheme recursively to get a hierarchical structure and achieve approximately the same estimated threshold value of 10^{-6}. These also work under the assumption that intermediate measurements are allowed and classical computation can be performed infinitely fast. This makes the problem easier, because under this assumption parts of the computation can be assumed error-free. The proof of Aharonov and Ben-Or [17, 45] does not require intermediate measurements and classical operations during quantum computation.

However, the main drawback in these models is that error correction must be done after every gate periodically. This imposes a considerable overhead of resources. So we propose a methodology whereby, the placement of error-correction circuit after every gate or time step can be avoided. These have been detailed in the next chapter.

Chapter 3

Overview and Synthesis

Different PMDs support various primitive operations and thus a given quantum circuit can have different implementations reducing cost or number of operations. For the encoded circuit, the tile architecture for various QECCs differ and accordingly the probability of failure of an encoded quantum gate differ with QECC used and PMD considered. We start this chapter by describing briefly the FTQLS tool that shows PMD dependency of a circuit implementation and then we give an overview of the methodology used for estimating probability of error in a given quantum circuit.

3.1 Fault Tolerant Quantum Logic Synthesis (FTQLS)

The goal of FTQLS is to synthesize efficiently a PMD specific optimum circuit with FT gates. We first discuss the flow of FTQLS before proceeding to discuss about the FT gates the algorithm considers.

3.1.1 FTQLS flow

The input to FTQLS [60] is an unoptimized quantum circuit realized using a set of commonly used gates and its output is an optimized FT quantum circuit that only comprises primitive quantum operations supported by the given PMD. FTQLS operates in three domains: (1) behaviour domain, (2) technology domain and (3) FT domain. We describe each of these domains briefly.

1. **Behaviour domain:** No PMD information is contained in this domain and thus the synthesized circuits may not be implementable on a quantum machine.

2. **Technology domain:** Using the PMD type and optimized gate library [58], all the gates in the behaviour domain are decomposed into primitive operations for a specific PMD. This is also called *Technology Mapping*.

3. **FT domain:** Using quantum compilers, non-FT circuits are converted into FT circuits within a reasonable CPU time. A quantum compiler converts an arbitrary unitary gate into a cascade of FT gates. For FT synthesis, two tools based on SKA [36] and STA [56] were integrated into FTQLS. Since these two tools can only compile one-qubit gates, initially the non-FT two-qubit gates are converted into FT two-qubit gates. Then all the non-FT one-qubit gates are compiled to FT cascades.

At each domain optimizations are performed according to the rules given in [58, 60].

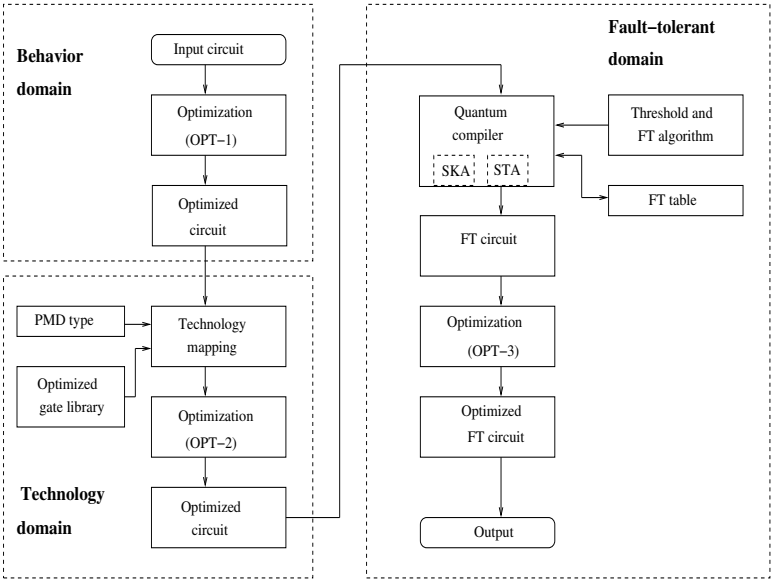


FIGURE 3.1: FTQLS flow

3.1.2 Fault-Tolerant Set (FTS) of gates

Every gate does not have a fault-tolerant (FT) implementation. The implementations of the QECCs described in the previous chapter - Shor, Steane, Bacon-Shor and Knill code are based on the Clifford-plus-T library (CTL) [57], described in Chapter 2. Unfortunately, not every quantum system directly supports the CTL (Table 2.1). This makes the implementation of FT circuits with CTL inefficient. Therefore, in [60] the authors extended the CTL library to a larger set of gates, the fault-tolerant set (FTS), to bridge the gap between CTL and FTS. All the operations in FTS are primitive to each PMD and thus can be efficiently implemented.

TABLE 3.1: Conversion between one-qubit FTS and CTL

k	R_z	R_x	R_y
1	T	HTH	$SHTHS^\dagger$
2	S	HSH	HZ
3	$ZT^\dagger$	$HZT^\dagger H$	$SHZT^\dagger HS^\dagger$
4	Z	X	ZX
5	ZT	$HZTH$	$SHZTHS^\dagger$
6	$S^\dagger$	$HS^\dagger H$	ZH
7	$T^\dagger$	$HT^\dagger H$	$SHT^\dagger HS^\dagger$

FTS for one qubit gates is defined as follows:

$$FTS(1) = \{R_A(k.\frac{\pi}{4}), H\}, \qquad A \in \{x, y, z\}, \quad k \in \{0, 1, \dots 7\} \tag{3.1}$$

The elements of FTS can be trivially obtained from CTL (Table 3.1).

In the case of two-qubit gates, CZ (supported in QD, SC, NA and LP systems), $G(\frac{\pi}{2})$ and $G(\frac{3\pi}{2})$ (supported in the IT system) are FT because:

$$CZ = [I \otimes H]\dot{C}NOT\dot{[}I \otimes H] \tag{3.2}$$

$$G(\frac{\pi}{2}) = [S \otimes SH]\dot{C}NOT\dot{[}I \otimes H] \tag{3.3}$$

$$G(\frac{3\pi}{2}) = [I \otimes H]\dot{C}NOT\dot{[}S^\dagger \otimes HS^\dagger] \tag{3.4}$$

In addition, since SWAP and ZENO (a special case of iSWAP) gates can be constructed from $CNOT$ and some FT one-qubit gates [58], they are also FT. Thus, the two-qubit FTS is defined as follows:

$$FTS(2) = \{CNOT, CZ, G(\frac{\pi}{2}), G(\frac{3\pi}{2}), SWAP, ZENO\} \tag{3.5}$$

3.2 Error Models

In all quantum computing technologies, errors are abstracted into 3 different sources [49]:

1. **Gate errors:** Depending on the physical technology, gates could involve complex sequences of applications of electrical and/or magnetic fields, current, and/or EMI radiation applied to one or more co-located qubits. These gate processes can introduce errors from apparatus imprecision or tunneling effects between qubits. The abstraction of this error type is that each qubit involved in a gate has some

probability of an error being introduced immediately after the gate is finished. Additionally, multi-qubit gates can propagate existing errors from one qubit to another.

2. **Movement/communication errors:** Qubit communication can involve either physical movement of coherent particles or gate-like operations to transfer state across fixed physical resources. In the former case, kinetic motion of particles can introduce motional heating and even particle loss. The abstraction often used is some amount of distance moved and it introduces a single qubit error with some probability.

3. **Memory/idle errors:** Even when a qubit is sitting stationary, interaction with the environment, either through coupling with stray EMI fields or contact with stray particles, can cause errors. Since there is no physical action the qubit is performing, memory errors are abstracted to a probability of error per unit of time it is stationary.

While estimating probability of error in a given quantum logical circuit we consider only the gate error.

3.3 Error Propagation

Before we proceed to describe the methodology, it would be helpful if we discuss briefly about how gate error propagates in a quantum circuit. For single qubit gates, the error can only propagate on that qubit the gate is working on.

For multi-qubit gates, the error can propagate in two ways. Since we are working with circuits that are output from FTQLS, it will have maximum two-qubit gates. So we describe error propagation in two-qubit gates. Let us consider the CNOT gate. It is obvious that if a bit flip occurs in one qubit, and that qubit is then used as the sourse qubit of CNOT, then the bit flip will propagate forward to the target qubit. Now, if we perform a rotation of basis with a Hadamard gate on both qubits, then the source and the target of the CNOT gate are interchanged. Since this change of basis also interchanges a bit flip error with a phase flip error, we infer that if a phase flip occurs in one qubit, and that qubit is then used as the target qubit of a CNOT gate, then the error will propagate "backward" to the source qubit.

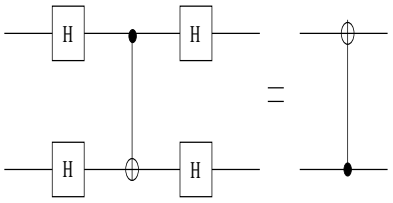


FIGURE 3.2: The source and the target of CNOT gate is interchanged if a change of basis is performed with Hadamard rotations

3.4 Methodology: Estimating error probability in quantum circuits

Our aim, as already stated, is to estimate the error probability in a quantum circuit, so that placing syndrome detection and error correction circuit after every component, can be avoided, thus saving some resources. The input to our algorithm is a file containing the quantum circuit in QASM format. QASM is a simple text-format language for describing acyclic quantum circuits composed from single as well as multiple qubut gates. The circuit given to us is at logical level, that is it is encoded and the user specifies the number of levels of encoding. We calculate offline the error probability for the different quantum gates at the logical level, encoded at that specific level of encoding. We describe the calculation of gate error probability at the logical level in the next chapter.

3.4.1 Data Structure

The data structure for representing a quantum circuit is a bidirectional graph.

We have an array of structures where each qubit is represented by a structure which must have the following fields:

- Name of the qubit as given in the input file.
- Pointer to the first gate operating on that qubit.
- The error probability on that qubit.

Each qubit structure points to a link list, where each node represents a gate and it must have the following fields:

- Type of the gate, that is, X, Y, CNOT, etc.

- Number of qubit lines it is operating on, that is whether single or double-qubit gate.
- Name of the qubit lines it is operating on.
- Forward pointers to the gate (gates for two-qubit gate) operating just after its time slice.
- Backward pointers to gate (gates for two-qubit gate) operating just before its time slice.
- Initial error probability on the qubits it is operating on.
- Error probability on the qubits after its operation is over.

A pictorial representation of this implementation on a toy circuit has been shown in Figure 3.3.

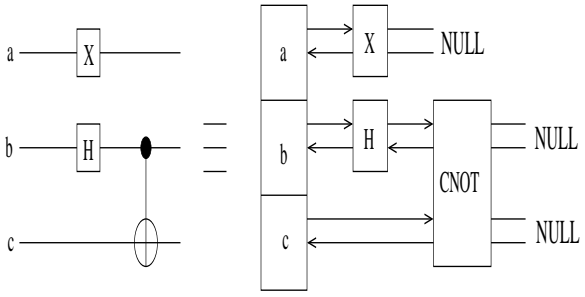


FIGURE 3.3: Pictorial representation showing how a toy circuit is implemented

3.4.2 Methodology

In our analysis we have made no difference between a bit flip error and a phase flip error. We follow these rules for error probability estimation.

- For single qubits, we propagate the error forward, that is we add the existing error probability on the qubit with the gate error probability and update the error probability on the qubit.
- For multi-qubit gate we propagate the maximum of the error probabilities in the input qubits to all the output qubits. That is, we add the maximum of the input error probabilities with the gate error probability and update the error probability on all the outgoing qubits.
- While adding error probabilities, if the value exceeds 1, we update the error probability as 1.

- With the help of the information stored in the structures we can trace the critical path, that is the path propagating the maximum error probability. This might be of interest in other applications.
- We fix a threshold error probability and place the error correcting subcircuit only when the error probability after a gate exceeds the threshold. After an error correcting block is placed, the error probability on the qubit is updated to the error probability of the error correcting block.

A flow diagram of the proposed algorithm has been shown in Figure 3.4, for better understanding. Square blocks in the diagram indicate input or output in file, while rounded blocks indicate procedures.

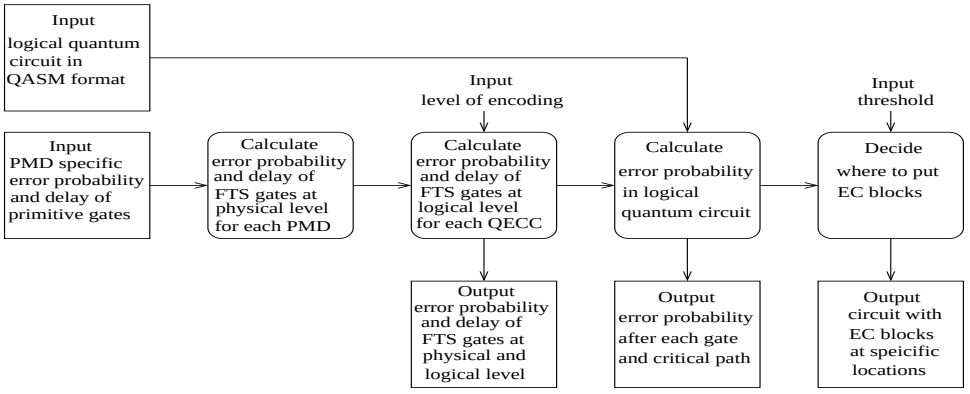


FIGURE 3.4: Flow diagram describing the proposed algorithm. (Square blocks indicate input or output in file, while rounded blocks indicate procedures)

3.4.3 Time Complexity

Since no-cloning theory holds in quantum computing, a quantum circuit has no fan-out or feedback. It is an acyclic graph. Also, the calculation of error probabilities of the gates both at the physical and logical level, is done offline. So doing an error tracing in the quantum logical circuit requires a single pass of the whole circuit. Thus the time complexity of the proposed algorithm is $O(n)$, where n is the size of the circuit, in terms of number of gates.

3.4.4 Comparison with previous works

In [61] the authors have developed the tile structures for the three QECCs we have considered and have done calculation of number of operations and time required by each encoded gate. They have used these values to calculate the resources required by popular quantum algorithms. However our work is markedly different from them because:

- We have developed the tile structures independently of them.
- We have calculated the error probability of the gates at the physical and logical level, which they have not considered at all.
- We have proposed an algorithm to trace the error probability in a given quantum logical circuit. We do not estimate its resources.
- We have devised a methodology to reduce the number of error correcting sub-circuits for fault-tolerant implementation, thus reducing the resources considerably.

The method of tracing errors in quantum circuits that has been developed in [49] has some similarity with our method. But there are a number of disparities between these two methods.

- We have considered error probabilities. In [49] a metric for error has been taken, but the physical significance of that value is not clear as details are not given.
- We have done a detailed error analysis of the gates at the physical and logical level, which is missing in [49].
- Our way of calculating error probabilities is more involved and much more rigorous than in [49].

Chapter 4

Estimation of Gate Error Probability at Logical Level

We model the qubit layout after the micro-architecture of Svore et al [41] and Spedalieri et al [46]. They have considered a 2-D nearest neighbour lattice architecture using logical noisy SWAP gates as the basic qubit transport mechanism. Clearly, any viable layout for stationary qubits has to be in a 2-D plane so that classical control fields can access the qubits from the third dimension. In a concatenated architecture, after first level of encoding each physical qubit is replaced by a 2-D block or cell or tile, that forms the logical qubit. After the second level of encoding each such cell is replaced by a 2-D block of such logical qubit cells, and so on. A high level picture of the architectural organization is shown in Figure 4.1.

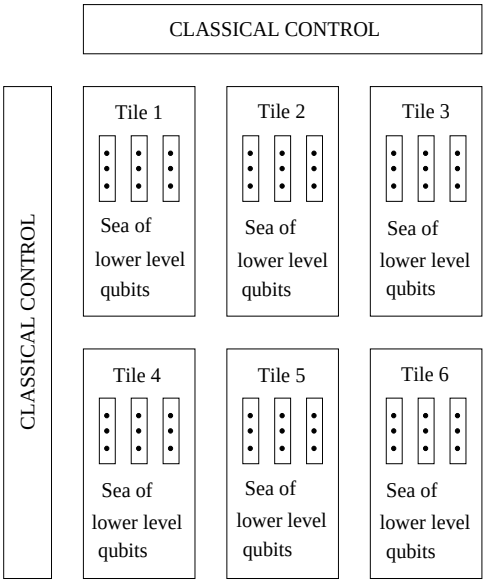


FIGURE 4.1: Qubit layout consisting of tiles that represent one logical qubit

In the next three sections we consider one-by-one each of Bacon Shor, Steane and Knill code and we describe the physical layout of the tiles and the different encoded gate operations. This is important for estimating the error probability of encoded operations. In section 4.4 we describe our methodology for calculating the error probability of gates at the physical and logical level. In section 4.5 and 4.6 we formulate the error probability at physical and logical level respectively for gates in the FTS set.

In all the diagrams, data qubits are represented by d_1, d_2, etc., and ancilla qubits are noted as a_1, a_2, etc.The dummy qubits used to prepare and measure the ancilla as well as for qubit transport are represented by O.

4.1 Bacon Shor Code

For implementing Bacon Shor code, we use the implementation in [46] which consists of embedding the nine data (physical) qubits corresponding to one logical qubit, in a 7×7 array of physical qubits (Figure 4.2).

FIGURE 4.2: Tile structure of the Bacon Shor code in a 7×7 lattice

4.1.1 Encoded gates

The simplest FT operation is the application of physical gates *transversally* across the codewords, meaning that the j^{th} gate is applied to the j^{th} qubits of the codewords, for every j. The advantage of transversal implementation is that since two or more qubits

within the same block do not interact, so there is no possibility of error propagation to other qubits in the same block. This makes the implementation fault-tolerant.

As already stated the Bacon Shor code belongs to the family of CSS codes, a family of codes with transversal implementation of most gates, including the CNOT gate. Transversal CNOT gate at level m can be obtained by applying nine CNOT gates to the corresponding control and target qubits at level $m-1$. Figure 4.3 shows a FT implementation of the CNOT gate. Thus we only need to move the data qubits so that the corresponding qubits of the two neighbouring blocks are next to each other, and then apply a single-qubit CNOT between each pair. To do this, we first move all the data qubits in one block up one row (or left one column), while the data qubits on the other block start moving laterally (or vertically) towards the first block. Then we keep moving the data qubits towards each other, interleaving the rows (columns) until the corresponding data qubits are next top each other. Gates that can be performed transversally also include the single qubit Pauli gates. The H gate is transversal modulo a π rotation of the 3×3 array. The encoded SWAP gate can be implemented simply by moving the data qubits of adjacent blocks towards each other.

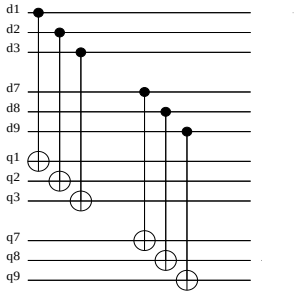


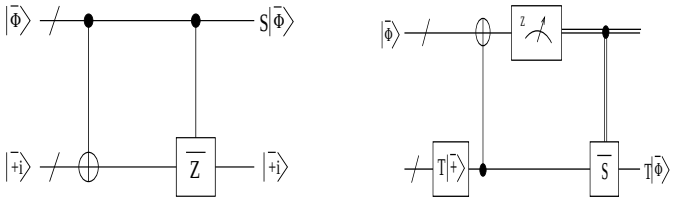
FIGURE 4.3: Fault tolerant transversal implementation of CNOT gate

But not all FT gates have transversal implementation for a given code, for e.g. the S and T gates in this case. The S gate can be implemented using the circuit in Figure 4.4(a). It uses an ancilla in the state $|+i\rangle = \frac{|0\rangle+i|1\rangle}{\sqrt{2}}$ as a resource to generate the required gate. A FT version of the T gate cannot be constructed transversally. A FT construction of the T gate has been shown in Figure 4.4(b).

A detailed pictorial representation for the implementation of the different encoded gates at the tile level for the Bacon Shor code has been given in Appendix A.

4.2 Steane Code

For implementing Steane code, we use the tile structure in [41], designed to minimize the amount of SWAP operations used during error-correction routines, and thus preserve a



(a) S gate (b) T gate

FIGURE 4.4: Fault-tolerant (non-transversal) implementation of S and T gate

high error threshold. The tile, representing one logical qubit consists of a 6×8 lattice of physical qubits (Figure 4.5), in which are embedded the seven data (physical) qubits.

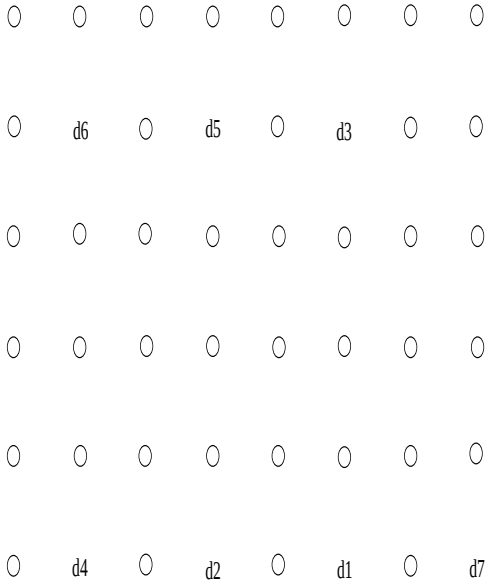


FIGURE 4.5: Tile structure of the Steane code in a 6×8 lattice

4.2.1 Encoded Gates

Just like the Bacon Shor code, the Steane code belongs to the family of CSS codes and has fault-tolerant transversal implementation for the Pauli gates, H and CNOT gates. But unlike the Bacon Shor code, it has also fault-tolerant implementation for the S gate. This can be done by applying the operation ZS to each qubit in a logical block. Fault-tolerant implementation of the T gate can be done by the circuit in Figure 4.4(b).

A detailed pictorial representation for the implementation of the different encoded gates at the tile level for the Steane code has been given in Appendix B.

4.3 Knill Code

For implementing Knill code, we followed [61] and design a 2-D 5×5 lattice architecture of physical qubits to represent a logical qubit of the Knill or C_4 code. Embedded within this tile are the four data (physical)qubits. We initialize the tile as the structure in Figure 4.6.

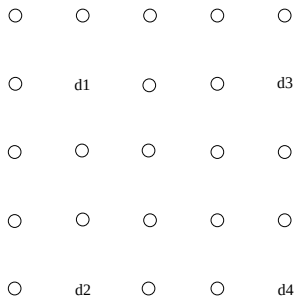


FIGURE 4.6: Tile structure of the Knill code in a 5×5 lattice

4.3.1 Encoded Gates

Just like the Bacon Shor code, the Knill code has fault-tolerant transversal implementation for the Pauli gates, H and CNOT gates. Fault-tolerant implementation of the T gate can be done by the circuits in Figure 4.4.

A detailed pictorial representation for the implementation of the different encoded gates at the tile level for the Steane code has been given in Appendix C.

Having discussed about the tile structures of the different encoded gates, we now explain our method of calculating the error probability of the gates at the physical level and at the logical level.

4.4 Calculating the error probability and delay of gates

To calculate the error probability of logical gates, we must first calculate the error probability of the gates at the physical level. The minimum a priori knowledge that we require is the error probability of the primitive gates for each PMD at the physical level and the error probability of an idle qubit. But to the best of our knowledge, no such information is available. In [61] a table summarizing the error probability after applying the worst gate and the probability of a bit flip per nanosecond on an idle qubit, has been given (Table 4.1). Also, the operation set that the authors have considered include

the following: Pauli gates, H, S, T, CNOT, SWAP, measurement gates and the state preparation. Since we have no other information, we assume that the probability of error of the primitive gates at the physical level for each PMD is the value given in this table for primitive control. So in a way our results are pessimistic, but it must be noted that the analysis can be applied with change of input, once the correct information regarding primitive gate errors at physical level is obtained.

TABLE 4.1: The probability of error of the worst gate at the physical level and the probability of error occurring on an idle qubit for each PMD [61]

Technology	Probability of Gate error	Memory error (per ns)
QD	9.89×10^{-1}	3.47×10^{-2}
NA	8.12×10^{-3}	0.00
LP	1.01×10^{-1}	9.80×10^{-4}
NLP	5.20×10^{-3}	$9.80 \times 10^{-}5$
SC	1.00×10^{-5}	1.00×10^{-5}
IT	3.19×10^{-9}	2.52×10^{-12}

The idle/memory error probability for a single qubit is defined as in [49].

$$1 - p_{idle} = (1 - p)^t \tag{4.1}$$

where p_{idle} is the total idle error probability for that time, p is the probability of idle error per unit time and t is the total time the qubit is lying idle. So to calculate the memory error we must also keep track of the time required for each gate. A table for the time required by different gates for each PMD has been given in [61]. We use these values (Table 4.2).

TABLE 4.2: Gate time (in ns) (at physical level)[61]

Technology	CNOT	SWAP	H	M_X	M_Z	X	Y	Z	S	T
QD	27	81	12	100	112	10	11	1	1	1
NA	2533	7599	781	80457	80000	457	457	915	915	915
LP	10	10	1	2	1	1	1	1	1	1
NLP	12	36	1 51	50	1	1	1	1	1	1
SC	26	13	16	10	26	10	10	1	1	1
IT	120000	10000	6000	106000	100000	5000	5000	3000	2000	1000

We describe our method of calculating the error probabilities first at the physical level and then at the logical level.

4.4.1 Calculation of gate error probability at physical level

We assume that the gate error probability of the primitive gates for each PMD is given and we have already mentioned the source we have considered. Thus given a particular PMD, our methodology for calculating the physical error of gates is as follows:

- For the primitive gates of that PMD (Table 2.1), the error probability is taken from the input.
- The non-primitive gates of that PMD are realized with the primitive gates of that PMD. These implementations have been given in [58] and we give them later in section 4.5. So these non-primitive gates are just like a black box within which the components are made of primitive gates.
- We consider such a non-primitive gate erroneous if any of the primitive components is at error. Thus the probability of error for non-primitive gates is calculated as the probability that any one of its constituent gates fail.

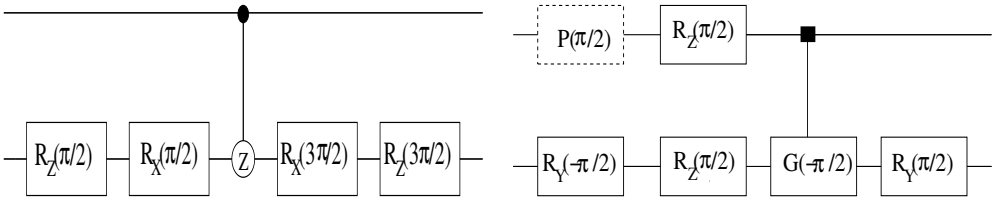
For example, consider the CNOT gate, which is a non-primitive gate in QD. It can be realized with the primitive gates of QD, as shown in Figure 4.7(a).

Let g_0 is the gate error probability at the physical level, that is 0^{th} level of encoding, g_n is the gate error at the n^{th} level of encoding, g_{k_A} is the error probability of gate A at k^{th} level of encoding. Similarly let t_0 is the gate delay at the physical level, that is 0^{th} level of encoding, t_n is the gate delay at the n^{th} level of encoding, t_{A_k} is the delay of gate A at k^{th} level of encoding.

Thus the error probability of CNOT in QD at physical level is:

$$g_0 = 1 - (1 - g_{0_{Rz}})^2 (1 - g_{0_{Rx}})^2 (1 - g_{0_{CZ}}) \tag{4.2}$$

The time at physical level is given in Table 4.2.



(a) QD (b) IT

FIGURE 4.7: CNOT gate construction in QD and IT

4.4.2 Calculation of gate probability and time at logical level

For calculating the error probability at different levels of encoding, we have to consider the tile structures of encoded gates for Bacon Shor, Steane and Knill code, shown respectively in Appendix A , Appendix B and Appendix C. To the best of our knowledge,

the tile structures are possible for the gates in the CTL library or any gate which can be obtained in a compound way from one or more operations in the CTL library. Thus our analysis only applies to gates which can be implemented int this tile architecture. Since the QECCs we have considered are single error correcting, we say that a logical qubit is faulty if at least two of its constituent data (physical) qubits is erroneous.

Our methodology for calculating the error probability at logical level for a particular encoded gate, given a specific QECC, is as follows:

- We first calculate error probability of encoded CTL gates. For the first level of encoding, from the tile structures of that gate for that QECC, we find out the number of distinct physical gates it uses. By distinct, we mean the gates must be located at distinct locations.
- We find out the gates which cause a single error per logical block. Let this gate set be S.
- Then the probability of getting at least two errors per logical block is:

$$\begin{aligned} Pr(\text{at least two errors per logical block}) &= 1 \\ &- Pr(\text{no gates fail}) \\ &- Pr(\text{exactly one gate in S fails}) \end{aligned}$$

- For higher levels of encoding, we consider the failure probability of gates at the next lower level, giving a recursive formula.
- For the non-CTL gates, but which can be implemented with a sequence of CTL gates, we have found after similar analysis, that the error probability at a particular level of encoding, say n is approximately equal to the probability that any one of the constituent CTL gates at n^{th} level of encoding fails.

For example, consider a CNOT gate, encoded with the Bacon Shor code, as shown in Appendix A (Figure A.4). We can see that the logical CNOT at n^{th} level requires 54 distinct SWAP and 9 distinct CNOT operations at $(n-1)^{th}$ level. Of these 36 SWAPs and 9 CNOT s are there such that failure of any one of them causes a single error. Thus the probability of error at the n^{th} level of encoding is:

$$\begin{aligned} g_n &= 1 - (1 - g_{n-1_{CNOT}})^9 (1 - g_{n-1_{SWAP}})^{54} - \binom{36}{1} g_{n-1_{SWAP}} (1 - g_{n-1_{SWAP}})^{53} (1 - g_{n-1_{CNOT}})^9 \\ &- \binom{9}{1} g_{n-1_{CNOT}} (1 - g_{n-1_{CNOT}})^8 (1 - g_{n-1_{SWAP}})^{54} \end{aligned} \tag{4.3}$$

The delay is simply the sum of the delay required for each constituent operations at the tile level. In this case, for CNOT, delay at n^{th} level of encoding can be formulated as :

$$t_n = 8t_{n-1_{SWAP}} + t_{n-1} \tag{4.4}$$

This methodology will be more clear as we formulate the error probabilities of each gate for each PMD, considering each of the three QECC in the next sections. Let w and t is the worst gate error probability and gate delay for the particular PMD, taken from Table 4.1 and Table 4.2 respectively. We assume that the error probability and delay of gates A and $A^\dagger$ is same.

4.5 Error probability and delay of gates at physical level

The error probability at physical level for each gate only depends on the PMD and not on the QECC (since encoding has not been done yet). Thus we first formulate the error probability of the gates in the FTS set (given in Chapter 3) at the physical level. The primitive gates for each PMD has been given in Table 2.1. We enlist the gates in FTS set here for convenience of the reader:

$$\begin{aligned} FTS(1) &= \{R_A(k.\frac{\pi}{4}), H\}, \qquad A \in \{x, y, z\}, \quad k \in \{0, 1, \ldots 7\} \\ FTS(2) &= \{CNOT, CZ, G(\frac{\pi}{2}), G(\frac{3\pi}{2}), SWAP, ZENO\} \end{aligned}$$

4.5.1 QD

The primitive gates in QD are R_x, R_z, X, Z, S, T.. We enlist the error probabilities of the different gates.

$\mathbf{R_x}, \mathbf{R_y}, \mathbf{R_z}$

Since R_x and R_z are primitive gates, the error probability for the physical gate is $g_0 = w$ for each of them. R_y can be expressed in terms of the other two gates in the following way:

$$R_y(\theta) = -R_z(\frac{\pi}{2}).R_x(\theta).R_z(\frac{3\pi}{2}) \tag{4.5}$$

Thus for R_y the error probability at physical level is :

$$g_0 = 1 - (1 - g_{0_{Rz}})^2(1 - g_{0_{Rx}}) \tag{4.6}$$

We assume that the delay at physical level for R_x, R_y and R_z is the same as for X, Y and Z respectively and is given in Table 4.2. Thus in this case, for each gate $t_0 = t$.

X, Y, Z

X, Y and Z are special cases of R_x, R_y and R_z respectively. So we can say that X and Z are available to us as primitive gates in this PMD. Thus the physical error probability for each of these gates is $g_0 = w$. The Y gate can be realized in terms of X and Z as:

$$Y = iZX \tag{4.7}$$

Thus for Y gate the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_X})(1 - g_{0_Z}) \tag{4.8}$$

The delay at physical level is given in Table 4.2.

H

In QD, H can be realized in terms of the primitive gates using the following circuit identity:

$$H = P(\frac{\pi}{2})R_z(\frac{\pi}{2})R_x(\frac{\pi}{2})R_z(\frac{\pi}{2}) \tag{4.9}$$

Keeping in mind that the global phase gate is not physically implementable, the error probability of H at the physical level is:

$$g_0 = 1 - (1 - g_{0_{Rz}})^2(1 - g_{0_{Rx}}) \tag{4.10}$$

The delay at physical level is given in Table 4.2.

S

S is a primitive gate for PMD and thus the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

T

T is a primitive gate for PMD and thus the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

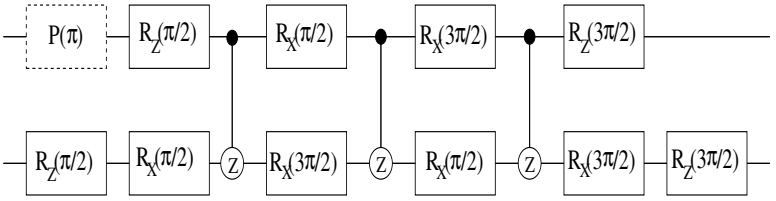
SWAP

The SWAP gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.8(a).

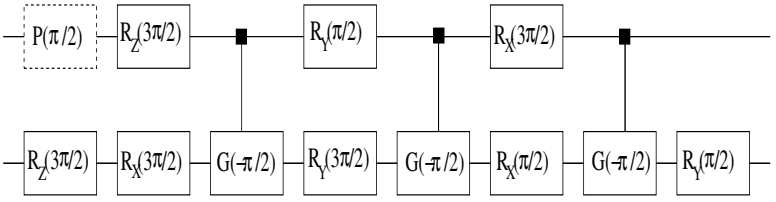
Thus, by previous logic, the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_{Rz}})^4 (1 - g_{0_{Rx}})^6 (1 - g_{0_{CZ}})^3 \tag{4.11}$$

The delay at physical level is given in Table 4.2.



(a) QD



(b) IT

FIGURE 4.8: SWAP gate construction in QD and IT

The delay at physical level is given in Table 4.2.

CNOT

It has already been discussed in subsection 4.4.1.

CZ

The CZ gate, being a primitive gate, the error probability at the physical level is $g_0 = w$. Using Table 4.2 and Figure 4.12(b) delay for CZ at the physical level can be calculated as follows:

$$t_{0_{CZ}} = t_{0_{CNOT}} - 2t_{0_{Rz}} - 2t_{0_{Rx}} \tag{4.12}$$

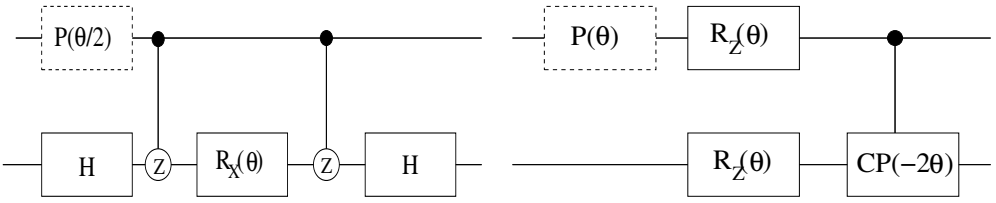
G

The G gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.9(a). Thus, by previous logic, the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_H})^2 (1 - g_{0_{CZ}})^2 (1 - g_{0_{Rx}}) \tag{4.13}$$

Using Table 4.2 and Figure 4.9(a) delay for G at the physical level can be calculated as follows:

$$t_{0_G} = 2t_{0_{CZ}} + 2t_{0_H} + t_{0_{Rx}} \tag{4.14}$$



(a) QD (b) SC

FIGURE 4.9: G gate construction in QD and SC

ZENO

The ZENO gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.10. Thus, by previous logic, the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_{CZ}})^4 (1 - g_{0_{Rz}})^2 (1 - g_{0_{Rx}})^6 \tag{4.15}$$

Using Table 4.2, Figure 4.10 and keeping in mind parallel operation is possible in quantum technology, delay for ZENO at the physical level can be calculated as follows:

$$t_{0_{ZENO}} = 4t_{0_{CZ}} + t_{0_{Rz}} + 4t_{0_{Rx}} \tag{4.16}$$

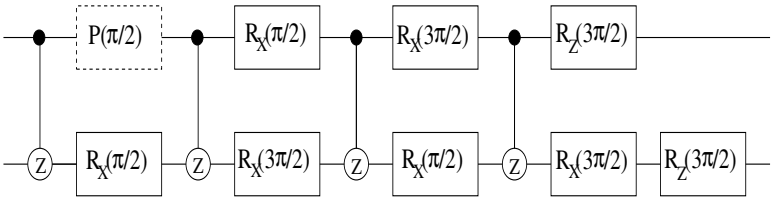


FIGURE 4.10: ZENO gate construction in QD

4.5.2 SC

The primitive gates in SC are $R_x, R_y, R_z, iSWAP, CP$.. We enlist the error probabilities of the different gates. The error probability of CP at the physical level is $g_0 = w$.

$\mathbf{R_x}, \mathbf{R_y}, \mathbf{R_z}$

Since R_x, R_y and R_z are primitive gates, the error probability for the physical gate is $g_0 = w$ for each of them.

We assume that the delay at physical level for R_x, R_y and R_z is the same as for X, Y and Z respectively and is given in Table 4.2. Thus in this case, for each gate $t_0 = t$.

X, Y, Z

X, Y and Z are special cases of R_x, R_y and R_z respectively. So we can say that X, Y and Z are available to us as primitive gates in this PMD. Thus the physical error probability for each of these gates is $g_0 = w$. The delay at physical level is given in Table 4.2.

H

In SC, H can be realized in terms of the primitive gates as in QD and the error probability formula at physical level is Equation 4.10. The delay at physical level is given in Table 4.2.

S

S is a special case of R_z since $R_z(\frac{\pi}{2}) = S$ and thus this can be treated as primitive gate in SC. So the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

T

T is a special case of R_z since $R_z(\frac{\pi}{4}) = T$ and thus this can be treated as primitive gate in SC. So the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

CZ

The CZ gate, being a special case of CP gate, is primitive gate, and thus the error probability at the physical level is $g_0 = w$. delay can be calculated like in Equation 4.12

SWAP

The SWAP gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown is Figure 4.11(c).

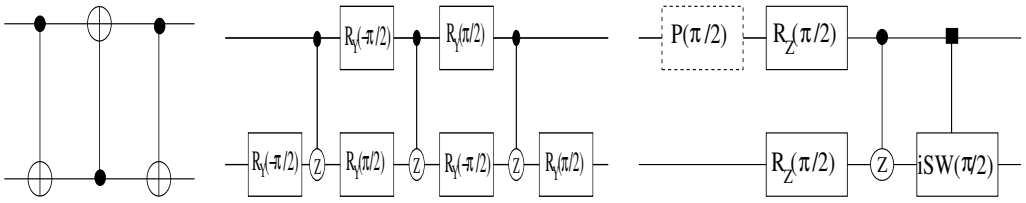
Thus, by previous logic, the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_{Rz}})^2(1 - g_{0_{iSW}})(1 - g_{0_{CZ}}) \tag{4.17}$$

The delay at physical level is given in Table 4.2.

CNOT

The CNOT gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.12(a).



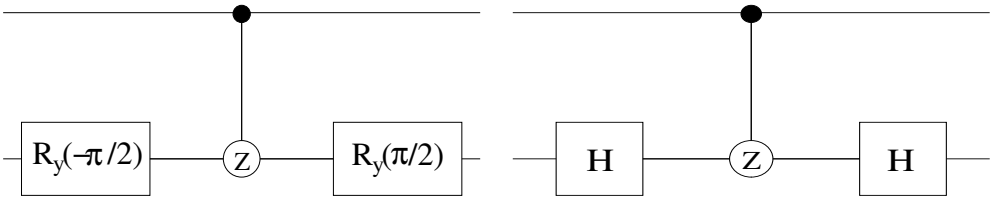
(a) NLP (b) NA (c) SC

FIGURE 4.11: SWAP gate construction in NLP, NA and SC

Thus, by previous logic, the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_H})^2(1 - g_{0_{CZ}}) \tag{4.18}$$

The delay at physical level is given in Table 4.2.



(a) from H gate (b) SC, NA

FIGURE 4.12: CNOT gate construction from H gate and in SC, NA.

G

The G gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.9(b). Thus, by previous logic, the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_{CP}})(1 - g_{0_{Rz}})^2 \tag{4.19}$$

Delay can be calculated in this case because we do not know the delay for CP. Anyway, it does not matter, since this gate does not come in the picture when calculating the memory error probability for any gate.

ZENO

The ZENO gate, is a special case of iSW and thus can be treated as a primitive gate. Hence the error probability at physical level is $g_0 = w$. Again, we do not know the delay at the physical level. Since this gate also does not come into the picture while calculating the memory error probability of any gate, thus we can safely put this aside for the time being.

4.5.3 LP

The primitive gates in SC are $R_x, R_y, R_z, X, Y, Z, S, T, H, CNOT, CZ, SWAP, ZENO$. We enlist the error probabilities of the different gates. The error probability of CP at the physical level is $g_0 = w$.

$\mathbf{R_x}, \mathbf{R_y}, \mathbf{R_z}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{S}, \mathbf{T}, \mathbf{H}$

Since all these are primitive gates, the error probability for the physical level is $g_0 = w$ for each of them.

We assume that the delay at physical level for R_x, R_y and R_z is the same as for X, Y and Z respectively and is given in Table 4.2. Thus in this case, for each gate $t_0 = t$.

CZ

The CZ gate, being a primitive gate, the error probability at the physical level is $g_0 = w$. Delay can be calculated like in Equation 4.12

SWAP

The SWAP gate, being a primitive gate, the error probability at the physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

CNOT

The CNOT gate, being a primitive gate, the error probability at the physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

G

The G gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.13(a). Thus, by previous logic, the error probability at physical level is:

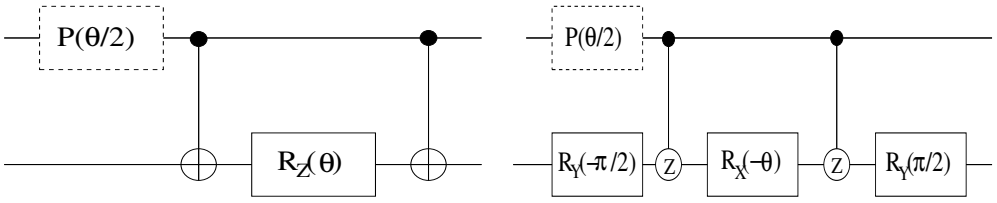
$$g_0 = 1 - (1 - g_{0_{CNOT}})^2(1 - g_{0_{Rz}}) \tag{4.20}$$

Delay can be calculated using the following equation:

$$t_0 = 2t_{0_{CNOT}} + t_{0_{Rz}} \tag{4.21}$$

ZENO

The ZENO gate, being a primitive gate, the error probability at physical level is $g_0 = w$. Again, we do not know the delay at the physical level. Since this gate does not come



(a) LP, NLP (b) NA

FIGURE 4.13: G gate construction in LP, NLP and NA

into the picture while calculating the memory error probability of any gate, thus we can safely put this aside for the time being.

4.5.4 NLP

The primitive gates in SC are $R_x, R_y, R_z, H, CNOT$. We enlist the error probabilities of the different gates. The error probability of CP at the physical level is $g_0 = w$.

$\mathbf{R_x}, \mathbf{R_y}, \mathbf{R_z}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{H}$

Since all of them are primitive gates, the error probability at the physical level is $g_0 = w$ for each of them.

We assume that the delay at physical level for R_x, R_y and R_z is the same as for X, Y and Z respectively and is given in Table 4.2. Thus in this case, for each gate $t_0 = t$.

S

S is a special case of R_z since $R_z(\frac{\pi}{2}) = S$ and thus this can be treated as primitive gate in NLP. So the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

T

T is a special case of R_z since $R_z(\frac{\pi}{4}) = T$ and thus this can be treated as primitive gate in NLP. So the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

CZ

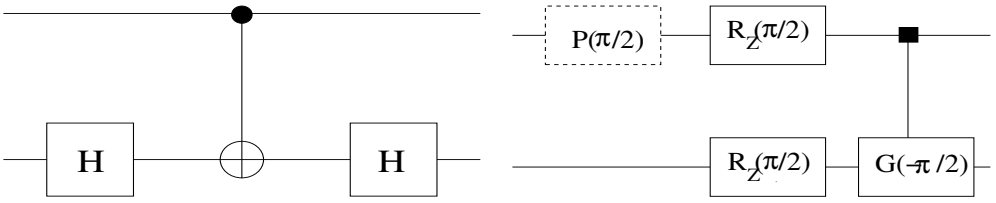
The realization of CZ with the help of the primitive gates in NLP is shown in Figure 4.14(a)

Thus the error probability at the physical level is:

$$g_0 = 1 - (1 - g_{0_H})^2(1 - g_{0_{CNOT}}) \tag{4.22}$$

The delay at the physical level can be calculated using the following equation:

$$t_0 = 2t_{0_H} + t_{0_{CNOT}} \tag{4.23}$$



(a) NLP (b) IT

FIGURE 4.14: CZ gate construction in NLP and IT.

SWAP

The SWAP gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.11(a)

Thus, by previous logic, the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_{CNOT}})^3 \tag{4.24}$$

The delay at physical level is given in Table 4.2.

CNOT

The CNOT gate, being a primitive gate, the error probability at the physical level is given by $g_0 = w$.

The delay at physical level is given in Table 4.2.

G

The G gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.13(a). The error probability at physical level is given by Equation 4.20. The delay at the physical level is given by Equation 4.21.

ZENO

The ZENO gate can be realized as in Figure 4.15.

Thus the error probability at the physical level is:

$$g_0 = 1 - (1 - g_{0_{Rz}})^2(1 - g_{0_{CZ}})(1 - g_{0_{SWAP}}) \tag{4.25}$$

The delay can be calculated as:

$$t_0 = t_{0_{Rz}} + t_{0_{CNOT}} + t_{0_{SWAP}} \tag{4.26}$$

In our analysis we have used the non-primitive SWAP gate, but it must be borne in mind that it can be realized with the primitive gates, as already shown.

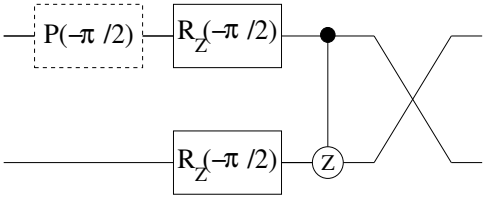


FIGURE 4.15: ZENO gate construction with SWAP gate

4.5.5 IT

The primitive gates in IT are R_{xy}, R_z, G. We enlist the error probabilities of the different gates.

$\mathbf{R_x}, \mathbf{R_y}, \mathbf{R_z}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}$

R_x and R_y are special cases of R_{xy}. Thus R_x, R_y and R_z, and hence X, Y, Z can be considered to have primitive implementations. Hence the error probability at the physical level is $g_0 = w$ for each of them.

We assume that the delay at physical level for R_x, R_y and R_z is the same as for X, Y and Z respectively and is given in Table 4.2. Thus in this case, for each gate $t_0 = t$.

S

S is a special case of R_z since $R_z(\frac{\pi}{2}) = S$ and thus this can be treated as primitive gate in NLP. So the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

T

T is a special case of R_z since $R_z(\frac{\pi}{4}) = T$ and thus this can be treated as primitive gate in NLP. So the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

H

In IT, H can be realized in terms of the primitive gates as in QD and the error probability formula at physical level is Equation 4.10. The delay at physical level is given in Table 4.2.

CNOT

The CNOT gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.7(b). The error probability at physical level is given by:

$$g_0 = 1 - (1 - g_{0_{Rz}})^2(1 - g_{0_{Ry}})^2(1 - g_{0_G}) \tag{4.27}$$

The delay at the physical level is given in Table 4.2.

G

The G gate, being a primitive gate, the probability of error at the physical level is given by $g_0 = w$. The delay at the physical level can be calculated using the circuit in fig..

$$t_0 = t_{0_{CNOT}} - 2t_{0_{Ry}} - t_{0_{Rz}} \tag{4.28}$$

CZ

The CZ gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.14(b). The error probability at physical level is given by:

$$g_0 = 1 - (1 - g_{0_{Rz}})^2(1 - g_{0_G}) \tag{4.29}$$

The delay at the physical level is given by:

$$t_0 = t_{0_G} + t_{0_{Rz}} \tag{4.30}$$

SWAP

The SWAP gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.8(b). The error probability at physical level is given by:

$$g_0 = 1 - (1 - g_{0_{Rx}})^3(1 - g_{0_{Ry}})^3(1 - g_{0_{Rz}})^2(1 - g_{0_G})^3 \tag{4.31}$$

The delay at the physical level is given in Table 4.2.

ZENO

The ZENO gate, being a non-primitive gate, can be realized with primitive gate using the circuit shown in Figure 4.15. Thus the error probability at the physical level is given by Equation 4.25

The delay for the physical gate is given by Equation 4.26

4.5.6 NA

The primitive gates in NA are R_{xy}, CZ. We enlist the error probabilities of the different gates.

$\mathbf{R_x}, \mathbf{R_y}, \mathbf{R_z}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}$

R_x and R_y are special cases of R_{xy}. R_z can be realized by cascading two R_{xy} gates. Thus the error probability of R_x, R_y and hence X, Y at the physical level is $g_0 = w$ for each gate. For R_z and Z the error probability at the physical level is $g_0 = 1 - (1 - w)^2$.

We assume that the delay at physical level for R_x, R_y and R_z is the same as for X, Y and Z respectively and is given in Table 4.2. Thus in this case, for each gate $t_0 = t$.

S

S is a special case of R_z since $R_z(\frac{\pi}{2}) = S$ and thus this can be treated as primitive gate in NLP. So the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

T

T is a special case of R_z since $R_z(\frac{\pi}{4}) = T$ and thus this can be treated as primitive gate in NLP. So the error probability at physical level is $g_0 = w$. The delay at physical level is given in Table 4.2.

H

In IT, H can be realized in terms of the primitive gates as in QD and the error probability formula at physical level is Equation 4.10. The delay at physical level is given in Table 4.2.

CNOT

The CNOT gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.12(a). The error probability at physical level is given by:

$$g_0 = 1 - (1 - g_{0_{Ry}})^2 (1 - g_{0_{CZ}}) \tag{4.32}$$

The delay at the physical level is given in Table 4.2.

CZ

The CZ gate, primitive gate, the error probability at physical level is given by $g_0 = w$.

The delay at the physical level is given by:

$$t_0 = t_{0_{CNOT}} - 2t_{0_{Ry}} \tag{4.33}$$

SWAP

The SWAP gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.11(b). The error probability at physical level is given by:

$$g_0 = 1 - (1 - g_{0_{Ry}})^6 (1 - g_{0_{CZ}})^3 \tag{4.34}$$

The delay at the physical level is given in Table 4.2.

G

The G gate, being a non-primitive gate, can be realized with primitive gates using the circuit shown in Figure 4.13(b). Thus, by previous logic, the error probability at physical level is:

$$g_0 = 1 - (1 - g_{0_{CZ}})^2 (1 - g_{0_{Rx}})(1 - g_{0_{Ry}})^2 \tag{4.35}$$

Delay can be calculated using the following equation:

$$t_0 = 2t_{0_{CZ}} + 2t_{0_{Ry}} + t_{0_{Rx}} \tag{4.36}$$

ZENO

The ZENO gate, being a non-primitive gate, can be realized with primitive gate using the circuit shown in Figure 4.15. Thus the error probability at the physical level is given by Equation 4.25

The delay for the physical gate is given by Equation 4.26

4.6 Error probability and delay of gates at logical level

The error probability at the logical level depends on the QECC used and not on the PMD. Hence we deduce the error probability of the encoded gates for the three different QECCs considered.

TABLE 4.3: Error probability and delay of R_z gate at the logical level (k as in Equation 3.1)

k	R_z	$g_{n_{Rz}}$	$t_{n_{Rz}}$
1	T	g_{n_T}	t_{n_T}
2	S	g_{n_S}	t_{n_S}
3	$ZT^\dagger$	$1-(1-g_{n_Z})(1-g_{n_T})$	$t_{n_T}+t_{n_Z}$
4	Z	g_{n_Z}	t_{n_Z}
5	ZT	$1-(1-g_{n_Z})(1-g_{n_T})$	$t_{n_T}+t_{n_Z}$
6	$S^\dagger$	g_{n_S}	t_{n_S}
7	$T^\dagger$	g_{n_T}	t_{n_T}

4.6.1 Bacon Shor Code

The tile structure for Bacon Shor code for the various CTL library gates has been given in Appendix A. Using this code, we formulate the error probabilities and the time for each gate operation for each PMD.

X, Y, Z

From the tiles shown in Appendix A (Figure A.1), we can see that for the logical encoding of each of these gates we need 9 distinct gates of respective type. Thus a logical gate is at error if more than 2 gates at the next lower level is at error. So for each of X, Y and Z, the error probability at the logical level at n^{th} level of encoding is:

$$g_n = 1-(1-g_{n-1})^9 - \binom{9}{1} g_{n-1}(1-g_{n-1})^8 \tag{4.37}$$

The delay for these gates at the logical level is:

$$t_n = t_{n-1} \tag{4.38}$$

$\mathbf{R_x}, \mathbf{R_y}, \mathbf{R_z}$

To calculate the error probability for the rotation gates at different levels of encoding we use Table 3.1 to express them in terms of the CTL library. Using the logic as discussed earlier, the error probabilities and delay for the rotation gates at the logical level have been calculated and listed in Table 4.3, Table 4.4 and Table 4.5.

Since all of the remaining gates SWAP is used we start with the error probability of the SWAP gate.

SWAP

TABLE 4.4: Error probability and delay of R_x gate at the logical level (k as in Equation 3.1)

k	R_x	$g_{n_{Rx}}$	$t_{n_{Rx}}$
1	HTH	$1-(1-g_{n_H})^2(1-g_{n_T})$	$t_{n_T}+2t_{n_H}$
2	HSH	$1-(1-g_{n_H})^2(1-g_{n_S})$	$t_{n_S}+2t_{n_H}$
3	$HZT^\dagger H$	$1-(1-g_{n_H})^2(1-g_{n_T})(1-g_{n_Z})$	$t_{n_T}+2t_{n_H}+t_{n_Z}$
4	X	g_{n_X}	t_{n_X}
5	$HZTH$	$1-(1-g_{n_H})^2(1-g_{n_T})(1-g_{n_Z})$	$t_{n_T}+2t_{n_H}+t_{n_Z}$
6	$HS^\dagger H$	$1-(1-g_{n_H})^2(1-g_{n_S})$	$t_{n_S}+2t_{n_H}$
7	$HT^\dagger H$	$1-(1-g_{n_H})^2(1-g_{n_T})$	$t_{n_T}+2t_{n_H}$

TABLE 4.5: Error probability and delay of R_y gate at the logical level (k as in Equation 3.1)

k	R_y	$g_{n_{Ry}}$	$t_{n_{Ry}}$
1	$SHTHS^\dagger$	$1-(1-g_{n_H})^2(1-g_{n_T})(1-g_{n_S})^2$	$t_{n_T}+2t_{n_H}+2t_{n_S}$
2	HZ	$1-(1-g_{n_H})(1-g_{n_Z})$	$t_{n_H}+t_{n_Z}$
3	$SHZT^\dagger HS^\dagger$	$1-(1-g_{n_H})^2(1-g_{n_T})(1-g_{n_Z})(1-g_{n_S})^2$	$t_{n_T}+2t_{n_H}+2t_{n_S}+t_{n_Z}$
4	ZX	$1-(1-g_{n_X})(1-g_{n_Z})$	$t_{n_X}+t_{n_Z}$
5	$SHZTHS^\dagger$	$1-(1-g_{n_H})^2(1-g_{n_T})(1-g_{n_Z})(1-g_{n_S})^2$	$t_{n_T}+2t_{n_H}+2t_{n_S}+t_{n_Z}$
6	ZH	$1-(1-g_{n_H})(1-g_{n_Z})$	$t_{n_H}+t_{n_Z}$
7	$SHT^\dagger HS^\dagger$	$1-(1-g_{n_H})^2(1-g_{n_T})(1-g_{n_S})^2$	$t_{n_T}+2t_{n_H}+2t_{n_S}$

The logical SWAP (Figure C.3 in Appendix A) requires 33 distinct SWAP operations at the next lower level. Of these 12 SWAP operations are such that failure of any one of them cause a single error. Thus the probability of error at the n^{th} level of encoding is:

$$g_n = 1-(1-g_{n-1})^{33} - \binom{12}{1} g_{n-1}(1-g_{n-1})^{32} \tag{4.39}$$

The delay is formulated as:

$$t_n = 7t_{n-1} \tag{4.40}$$

CNOT

It has already been discussed in subsection 4.4.2.

H

The logical H (Figure A.2 in Appendix A) requires 16 distinct SWAP gates and 9 distinct H gates at the next lower level. Also we can see that one of the qubits is idle for 4 timestamps. Only the 9 H if fails, cause one error. Failure of all other gates cause

at least two errors per block. Thus the probability of error at the logical level is:

$$\begin{aligned} g_n &= 1-(1-g_{n-1_H})^9(1-g_{n-1_{SWAP}})^{16}(1-M_{n_H}) \\ &- \binom{9}{1} g_{n-1_H}(1-g_{n-1_H})^8(1-g_{n-1_{SWAP}})^{16}(1-M_{n_H}) \\ &- M_{n_H}(1-g_{n-1_H})^9(1-g_{n-1_{SWAP}})^{16} \end{aligned} \tag{4.41}$$

where M_n is the memory error probability at the n^{th} level of encoding. Let m is the memory error probability obtained from Table 4.1 for a particular PMD. Then the memory error at different levels of encoding can be deduced to be the following:

$$\begin{aligned} M_{1_H} &= 1-(1-m)^{4t} \quad \text{where } t = t_{0_{SWAP}} \\ M_{n_H} &= 1-(1-m)^{9^{n-1}t} - 9M_{n-1_H}(1-m)^{8.9^{n-2}t} \quad \text{where } t = t_{n-1_{SWAP}} \end{aligned} \tag{4.42}$$

The delay is formulated as:

$$t_n = t_{n-1} + 4t_{n-1_{SWAP}} \tag{4.43}$$

S

The logical S (Figure A.5 in Appendix A) requires 42 distinct SWAP, 9 H and 9 CNOT. Of these 21 SWAP, 9 H and 9 CNOT are such that their failure causes one error per data block. Also, as can be seen from the figure, the data block of 9 qubits lies idle for 8 timestamps. So we have to take care of memory errors. Thus the error probability for the encoded S gate can be formulated as:

$$\begin{aligned} g_n &= 1-[(1-g_{n-1_H})^9(1-g_{n-1_{CNOT}})^9(1-g_{n-1_{SWAP}})^{42} \\ &+ \binom{21}{1} g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{41}(1-g_{n-1_H})^9(1-g_{n-1_{CNOT}})^9 \\ &+ \binom{9}{1} g_{n-1_H}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_H})^8(1-g_{n-1_{CNOT}})^9 \\ &+ \binom{9}{1} g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_H})^9(1-g_{n-1_{CNOT}})^8](1-M_{n_S}) \\ &- M_{n_S}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_H})^9(1-g_{n-1_{CNOT}})^9 \end{aligned} \tag{4.44}$$

The memory error M_n in this case is defined as:

$$\begin{aligned} M_{1_S} &= 9[1-(1-m)^{8t_{0_{SWAP}}}](1-m)^{64t_{n-1_{SWAP}}} \\ M_{n_S} &= 9[1-(1-M_{n-1_S})^{8t_{n-1_{SWAP}}}](1-M_{n-1_S})^{64t_{n-1_{SWAP}}} \end{aligned} \tag{4.45}$$

The delay is formulated as:

$$t_n = 8t_{n-1_{SWAP}} + 2(t_{n-1_{CNOT}} + t_{n-1_H}) \tag{4.46}$$

T

The logical T (Figure A.6 in Appendix A) requires 42 distince SWAP, 18 CNOT, 9 H and 9 measurement (M_z) gates. Of these 21 SWAP, 18 CNOT, 9 H and 9 M_z gates are capable of causing one error per data block. Also the data block of 9 qubits lies idle for 8 timestamps, giving rise to the possibility of memory error. We assume the error probability of measurement gate is equal to the worst gate error probability for a specific PMD.

$$\begin{aligned}
g_n &= 1 - [(1 - g_{n-1_{SWAP}})^{42}(1 - g_{n-1_{CNOT}})^{18}(1 - g_{n-1_H})^9(1 - g_{n-1_{M_z}})^9 \\
&+ \binom{21}{1} g_{n-1_{SWAP}}(1 - g_{n-1_{SWAP}})^{41}(1 - g_{n-1_{CNOT}})^{18}(1 - g_{n-1_H})^9(1 - g_{n-1_{M_z}})^9 \\
&+ \binom{18}{1} g_{n-1_{CNOT}}(1 - g_{n-1_{SWAP}})^{42}(1 - g_{n-1_{CNOT}})^{17}(1 - g_{n-1_H})^9(1 - g_{n-1_{M_z}})^9 \\
&+ \binom{9}{1} g_{n-1_H}(1 - g_{n-1_{SWAP}})^{42}(1 - g_{n-1_{CNOT}})^{18}(1 - g_{n-1_H})^8(1 - g_{n-1_{M_z}})^9 \\
&+ \binom{9}{1} g_{n-1_{SWAP}}(1 - g_{n-1_{SWAP}})^{42}(1 - g_{n-1_{CNOT}})^{18}(1 - g_{n-1_H})^9(1 - g_{n-1_{M_z}})^8](1 - M_{n_S}) \\
&- M_{n_S}(1 - g_{n-1_{SWAP}})^{42}(1 - g_{n-1_{CNOT}})^{18}(1 - g_{n-1_H})^9(1 - g_{n-1_{M_z}})^9
\end{aligned} \tag{4.47}$$

The memory error M_{n_S} is as defined in Equation 4.45.

The delay is formulated as:

$$t_n = 9t_{n-1_{SWAP}} + 3t_{n-1_{CNOT}} + 2t_{n-1_H} + t_M \tag{4.48}$$

t_M is the time required for measurement. The value has been given in Table 4.2.

CZ

The FT implementation of CZ is given by:

$$CZ = [I \otimes H] \cdot CNOT \cdot [I \otimes H] \tag{4.49}$$

Using the logic already explained before, the error probability of CNOT at logical level is:

$$g_n = 1 - (1 - g_{n_H})(1 - g_{n_{CNOT}}) \tag{4.50}$$

The delay is formulated as:

$$t_n = 2t_{n_H} + t_{n_{CNOT}} \tag{4.51}$$

G

In the FTS set we have $G(\frac{\pi}{2})$ and $G(\frac{3\pi}{2})$. The FT implementatin of these two gates is as follows:

$$\begin{aligned} G(\frac{\pi}{2}) &= [S \otimes SH] \cdot CNOT \cdot [I \otimes H] \\ G(\frac{3\pi}{2}) &= [I \otimes H] \cdot CNOT \cdot [S^\dagger \otimes HS^\dagger] \end{aligned} \tag{4.52}$$

Thus, the error probability of these two gates at the logical level is:

$$g_n = 1 - (1 - g_{n_S})^2(1 - g_{n_H})(1 - g_{n_{CNOT}}) \tag{4.53}$$

The delay is formulated as:

$$t_n = 2(t_{n_H} + t_{n_S}) + t_{n_{CNOT}} \tag{4.54}$$

ZENO

Since $R_z(-\frac{\pi}{2}) = S^\dagger$, so from fig.. the error probability of ZENO at logical level is:

$$g_n = 1 - (1 - g_{n_S})^2(1 - g_{n_{SWAP}})(1 - g_{n_{CZ}}) \tag{4.55}$$

The delay is formulated as:

$$t_n = 2t_{n_S}) + t_{n_{SWAP}} + t_{n_{CZ}} \tag{4.56}$$

For the remaining two QECCs the analyses of the rotation gates, CZ, G and ZENO remains same. So we give the analyses for the remaining ones.

4.6.2 Steane Code

The tile structure for Steane code for the various CTL library gates has been given in Appendix B. Using this code, we formulate the error probabilities and the time for each gate operation for each PMD.

X, Y, Z, H

From the tiles shown in Appendix B (Figure B.1 and Figure B.2), we can see that for the logical encoding of each of these gates we need 7 distinct gates of respective type. Thus a logical gate is at error if more than 2 gates at the next lower level is at error. So for each of X, Y, Z and H, the error probability at the logical level at n^{th} level of encoding is:

$$g_n = 1 - (1 - g_{n-1})^7 - \binom{7}{1} g_{n-1}(1 - g_{n-1})^6 \tag{4.57}$$

The delay for these gates at the logical level is:

$$t_n = t_{n-1} \tag{4.58}$$

S

Unlike Bacon Shor, the S gate has a transversal implementation in Steane code, as has been shown in Figure B.3 of Appendix B. Thus the error probability at the logical level is:

$$\begin{aligned} g_n &= 1 - (1 - g_{n-1})^7 (1 - g_{n-1_z})^7 - \binom{7}{1} g_{n-1}(1 - g_{n-1})^6 (1 - g_{n-1_z})^7 \\ &- \binom{7}{1} g_{n-1_z}(1 - g_{n-1})^7 (1 - g_{n-1_z})^6 \end{aligned} \tag{4.59}$$

The delay at the logical level is:

$$t_n = t_{n-1} + t_{n-1_Z} \tag{4.60}$$

SWAP

The tile structure of the Steane code is asymmetrical, thus it has slightly different complexities for SWAP and CNOT gates, depending on the alignment of the adjacent tiles. Accordingle there exists horizontal and vertical SWAP (hSWAP and vSWAP), horizontal and vertical CNOT (hCNOT and vCNOT). In our analyses we found that with increasing levels of encding the difference in error probabilities and time of the horizontal and vertical gates is very small. So we take the worst of the two in our analyses. For hSWAP we need 26 distinct SWAP and 8 such gates exist that causes 1 error per data block (Figure B.4 in Appendix B). Also, for each block, each qubit is idle for 1 timestamp. Thus, the error probability at the logical level is:

$$g_n = 1 - [(1 - g_{n-1})^{26} + 8 g_{n-1}(1 - g_{n-1})^{25}](1 - M_{n_{SWAP}}) - M_{n_{SWAP}}(1 - g_{n-1})^{26} \tag{4.61}$$

The memory error is given by:

$$\begin{aligned} M_{0_{SWAP}} &= m \\ M_{n_{SWAP}} &= 7[1-(1-M_{n-1_{SWAP}})^{t_{n-1_{SWAP}}}](1-M_{n-1_{SWAP}})^{6t_{n-1_{SWAP}}} \end{aligned} \tag{4.62}$$

The delay at the logical level is:

$$t_n = 9t_{n-1} \tag{4.63}$$

CNOT

We give the analysis of the vertical CNOT (vCNOT). From Figure B.7 in Appendix B, we see that it requires 43 distinct SWAP and 7 CNOT. Of these, 26 SWAPs and the 7 CNOTs are capable of causing one error per block. Also, each qubit in the block is idle for 2 timestamps. Thus, the error probability at the logical level is given by:

$$\begin{aligned} g_n &= 1-[(1-g_{n-1_{SWAP}})^{43}(1-g_{n-1})^7 \\ &+ \binom{26}{1} g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1})^7 \\ &+ \binom{7}{1} g_{n-1}(1-g_{n-1_{SWAP}})^{43}(1-g_{n-1})^6](1-M_{n_{CNOT}}) \\ &- M_{n_{CNOT}}(1-g_{n-1_{SWAP}})^{43}(1-g_{n-1})^7 \end{aligned} \tag{4.64}$$

The memory error is given by:

$$\begin{aligned} M_{0_{CNOT}} &= m \\ M_{n_{CNOT}} &= 7[1-(1-M_{n-1_{CNOT}})^{2t_{n-1_{CNOT}}}](1-M_{n-1_{CNOT}})^{12t_{n-1_{CNOT}}} \end{aligned} \tag{4.65}$$

The delay at the logical level is:

$$t_n = t_{n-1} + 10t_{n-1_{SWAP}} \tag{4.66}$$

T

From Figure B.8 in Appendix B, we find that for encoding the T gate we require 40 distinct SWAP, 7 CNOT, 7 X, 7 S and 7 M_z. Of these 22 SWAP, 7 CNOT, 7 X, 7 S and 7 M_z are capable of causing 1 error per block. Each data qubit per block lies idle

for 9 time stamps. Thus, the error probability at logical level is:

$$\begin{aligned} g_n &= 1-[(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7 \\ &+ \binom{22}{1} g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{39}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7 \\ &+ \binom{7}{1} g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^6(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7 \\ &+ \binom{7}{1} g_{n-1_{Mz}}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^6 \\ &+ \binom{7}{1} g_{n-1_X}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^6(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7 \\ &+ \binom{7}{1} g_{n-1_S}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^6(1-g_{n-1_{Mz}})^7](1-M_{n_T}) \\ &- M_{n_T}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7 \end{aligned} \tag{4.67}$$

The memory error is given by:

$$\begin{aligned} M_{0_T} &= m \\ M_{n_T} &= 7[1-(1-M_{n-1_T})^{9t_{n-1_T}}](1-M_{n-1_T})^{54t_{n-1_T}} \end{aligned} \tag{4.68}$$

The delay at the logical level is:

$$t_n = 10t_{n-1_{SWAP}} + t_{n-1_{CNOT}} + t_{n-1_X} + t_{n-1_S} + t_M \tag{4.69}$$

t_M is the time required for measurement. The value has been given in Table 4.2.

4.6.3 Knill Code

The tile structure for Knill code for the various CTL library gates has been given in Appendix C. Using this code, we formulate the error probabilities and the time for each gate operation for each PMD.

X, Y, Z

From the tiles shown in Figure C.1 of Appendix C, we can see that for the logical encoding of each of X and Z we need 2 distinct gates of respective types. Thus the error probability at logical level is:

$$g_n = 1-(1-g_{n-1})^2 - \binom{2}{1} g_{n-1}(1-g_{n-1}) \tag{4.70}$$

The delay at the logical level is:

$$t_n = t_{n-1} \tag{4.71}$$

Y gate uses one each of X, Y and Z gates for encoding. Thus, the error probability at logical level is:

$$\begin{aligned} g_n &= 1-(1-g_{n-1})(1-g_{n-1_X})(1-g_{n-1_Z}) - g_{n-1}(1-g_{n-1_X})(1-g_{n-1_Z}) \\ &- g_{n-1_X}(1-g_{n-1})(1-g_{n-1_Z}) - g_{n-1_Z}(1-g_{n-1_X})(1-g_{n-1}) \end{aligned} \tag{4.72}$$

The delay at the logical level is:

$$t_n = t_{n-1} + t_{n-1_X} + t_{n-1_Z} \tag{4.73}$$

H

As can be seen from Figure C.2 in Appendix C, the logical H gate uses 4 distinct H gates at the next lower level, the failure of any one of which causes one error per block. Thus, the error probability at the logical level is:

$$g_n = 1-(1-g_{n-1})^4 - \binom{4}{1} g_{n-1}(1-g_{n-1})^3 \tag{4.74}$$

The delay at the logical level is:

$$t_n = t_{n-1} \tag{4.75}$$

SWAP

The logical SWAP (Figure C.3 in Appendix C) uses 40 distinct SWAPS, out of which 32 are such that the failure of any one causes one error per block. Also, the four qubits per block lie idle for 2 timestamps. Thus the error probability at the logical level is:

$$\begin{aligned} g_n &= 1-[(1-g_{n-1})^{40} + \binom{32}{1} g_{n-1}(1-g_{n-1})^{39}](1-M_{n_{SWAP}}) \\ &- M_{n_{SWAP}}(1-g_{n-1})^{40} \end{aligned} \tag{4.76}$$

The memory error probability is given by:

$$\begin{aligned} M_{0_{SWAP}} &= m \\ M_{n_{SWAP}} &= 4[1-(1-M_{n-1_{SWAP}})^{2t_{n-1_{SWAP}}}](1-M_{n-1_{SWAP}})^{6t_{n-1_{SWAP}}} \end{aligned} \tag{4.77}$$

The delay at the logical level is:

$$t_n = 7t_{n-1} \tag{4.78}$$

CNOT

The logical CNOT (Figure C.4 in Appendix C) requires 24 distinct SWAP and 4 distinct CNOT at the next lower level, out of which failure of 22 SWAPs and 4 CNOT s cause one error per block. Thus, the error probability at the logical level is:

$$\begin{aligned} g_n &= 1-(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1})^4 - \binom{4}{1} g_{n-1}(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1})^3 \\ &- \binom{22}{1} g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{23}(1-g_{n-1})^4 \end{aligned} \tag{4.79}$$

The delay at the logical level is:

$$t_n = t_{n-1} + 3t_{n-1_{SWAP}} \tag{4.80}$$

S

The encoded S gate (Figure C.5 in Appendix C) uses 20 distinct SWAP, 4 CNOT and 4 H, out of which there are 16 CNOT, 4 CNOT and 4 H, such that the failure of any one of them causes one error per block. Also, the four qubits per block sit idle for 6 timestamps. Thus, the error probability at the logical level is given by:

$$\begin{aligned} g_n &= 1-[(1-g_{n-1_{SWAP}})^{20}(1-g_{n-1_{CNOT}})^4(1-g_{n-1_H})^4 \\ &+ \binom{16}{1} g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{19}(1-g_{n-1_{CNOT}})^4(1-g_{n-1_H})^4 \\ &+ \binom{4}{1} g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{20}(1-g_{n-1_{CNOT}})^3(1-g_{n-1_H})^4 \\ &+ \binom{4}{1} g_{n-1_H}(1-g_{n-1_{SWAP}})^{20}(1-g_{n-1_{CNOT}})^4(1-g_{n-1_H})^3](1-M_{n_S}) \\ &- M_{n_S}(1-g_{n-1_{SWAP}})^{20}(1-g_{n-1_{CNOT}})^4(1-g_{n-1_H})^4 \end{aligned} \tag{4.81}$$

The memory error is given by:

$$\begin{aligned} M_{0_S} &= m \\ M_{n_S} &= 4[1-(1-M_{n-1_S})^{6t_{n-1_{SWAP}}}](1-M_{n-1_S})^{18t_{n-1_{SWAP}}} \end{aligned} \tag{4.82}$$

The delay at the logical level is:

$$t_n = 2t_{n-1_H} + 6t_{n-1_{SWAP}} + 2t_{n-1_{CNOT}} \tag{4.83}$$

T

The encoded T (Figure C.6 in Appendix C) gate uses 24 distinct SWAP, 8 CNOT, 4 H, 4 M_z, out of which there are 20 SWAP, 8 CNOT, 4 H and 4 M_z such that failure of any one of them causes one error per block. Also, the four qubits per block lie idle for 6 timestamps. Thus, the error probability at the logical level is:

$$\begin{aligned}
g_n &= 1 - [(1 - g_{n-1_{SWAP}})^{24}(1 - g_{n-1_{CNOT}})^{8}(1 - g_{n-1_H})^{4}(1 - g_{n-1_{Mz}})^{4} \\
&+ \binom{20}{1} g_{n-1_{SWAP}}(1 - g_{n-1_{SWAP}})^{23}(1 - g_{n-1_{CNOT}})^{8}(1 - g_{n-1_H})^{4}(1 - g_{n-1_{Mz}})^{4} \\
&+ \binom{8}{1} g_{n-1_{CNOT}}(1 - g_{n-1_{SWAP}})^{24}(1 - g_{n-1_{CNOT}})^{7}(1 - g_{n-1_H})^{4}(1 - g_{n-1_{Mz}})^{4} \\
&+ \binom{4}{1} g_{n-1_H}(1 - g_{n-1_{SWAP}})^{24}(1 - g_{n-1_{CNOT}})^{8}(1 - g_{n-1_H})^{3}(1 - g_{n-1_{Mz}})^{4} \\
&+ \binom{4}{1} g_{n-1_{Mz}}(1 - g_{n-1_{SWAP}})^{24}(1 - g_{n-1_{CNOT}})^{8}(1 - g_{n-1_H})^{4}(1 - g_{n-1_{Mz}})^{3}](1 - M_{n_T}) \\
&- M_{n_T}(1 - g_{n-1_{SWAP}})^{24}(1 - g_{n-1_{CNOT}})^{8}(1 - g_{n-1_H})^{4}(1 - g_{n-1_{Mz}})^{4}
\end{aligned} \tag{4.84}$$

The memory error is given by:

$$\begin{aligned}
M_{0_T} &= m \\
M_{n_T} &= 4[1 - (1 - M_{n-1_T})^{6t_{n-1_{SWAP}}}](1 - M_{n-1_T})^{18t_{n-1_{SWAP}}}
\end{aligned} \tag{4.85}$$

The delay at the logical level is:

$$t_n = 2t_{n-1_H} + 7t_{n-1_{SWAP}} + 3t_{n-1_{CNOT}} + t_M \tag{4.86}$$

t_M is the time required for measurement. The value has been given in Table 4.2.

Chapter 5

Results and Observations

We have considered few benchmark circuits and applied our error probability tracing algorithm. We have implemented our algorithm in C, developing a tool that takes as input the error probability of gates at the physical level, the quantum circuit in QASM format, the number of levels of encoding and the threshold value. The tool outputs the gate error probabilities at the physical and logical level for each PMD and per QECC. With the help of these values the tool calculates and prints the error probabilities after each component in the quantum circuit and depending on the threshold considered it places the EC blocks at specific locations. We have obtained significant reduction in the number of error correction (EC) blocks, thus reducing considerable resources. We begin this chapter by briefly discussing about the benchmark circuits we have considered and then we tabulate our results and enlist our observations and inferences.

5.1 Benchmark Circuits

We have considered three arithmetic circuits - 4-bit adder, 8-bit adder, multiplier circuit and circuit for Grover's search algorithm.

Since Shor's algorithm can break the RSA cryptosystem, building quantum circuits for Shor's algorithm has gained much attention. An important part of Shor's algorithm is to find the order r of a modulo N. The order r of a mod N is the least positive integer such that a^r mod $N \equiv 1$.A systematic way to build the modular exponentiation circuit is to hierarchically decompose it into a controlled modulo multiplier , controlled modulo adder, modular adder and full adder. [62]

The **quantum adder** is the fundamental module of many circuits, including circuit for Shor's algorithm. In quantum adders, the addition of two registers is written as:

$$|a, b\rangle \rightarrow |a, a + b\rangle \tag{5.1}$$

Many different types of adders have been presented in literature [31], [27], [16].

The **quantum multiplier** of two input variables can be written as

$$|b, a, 0\rangle \rightarrow |b, a, a \cdot b\rangle \tag{5.2}$$

It can be implemented by cascading several quantum adders because:

$$a \cdot b = (2^0 a)b_0 + (2^1 a)b_1 + \ldots + (2^{n-1} a)b_{n-1} \tag{5.3}$$

where $b = b_{n-1}b_{n-2}\ldots b_1 b_0$

5.1.1 Circuit for Grover's Search algorithm

Grover's search [11] is perhaps the second most important algorithm after Shor's algorithm. It is used in many quantum algorithms. Consider a function:

$$\begin{aligned} f_a(x) &= 0, \quad x \neq a \\ f_a(x) &= 1, \quad x = a \end{aligned} \tag{5.4}$$

$$\tag{5.5}$$

where is x is an n-bit integer and f_a is a black box routine, which outputs 1 if $x = a$, else it outputs 0. On a classical computer, we can do no better than applying f_a repeatedly to different random numbers until we find a. We must try $N/2$ times, where $N = 2^n$, for a 50% chance of success. On the other hand, in quantum computing, Grover's search algorithm can find a with a probability p by calling f_a no more than $\frac{\pi}{4}\sqrt{N}$ times [11], [51].

The circuit structure is shown in Figure 5.1 [62]. The upper line is an n-qubit quantum register while the lower one is a 1-qubit quantum register. It contains r iterations of a sub-module, that has three parts: oracle function (f_a) routine, H^n block, and the diffusion operator D. The oracle function changes the state of the bottom register if the state of the upper integer equals a special integer.

We give an example of a QASM file format that we have used. Below is given a small Grover's circuit, having two data qubits and one ancilla qubit and it is searching the

state $|11\rangle$.

$$
\begin{array}{rll}
 & .\text{qubit3} & \\
 & \text{qubit} & q_0 \\
 & \text{qubit} & q_1 \\
 & \text{qubit} & q_2 \\
 & .\text{begin} & \\
 & H & q_0 \\
 & H & q_1 \\
 & H & q_2 \\
 & T & q_0 \\
 & T & q_1 \\
 & H & q_2 \\
CNOT & q_2 & q_1 \\
 & T & q_1 \\
CNOT & q_0 & q_2 \\
CNOT & q_0 & q_1 \\
 & T & q_1 \\
 & T & q_2 \\
CNOT & q_0 & q_2 \\
CNOT & q_2 & q_1 \\
 & T & q_1 \\
 & T & q_2 \\
CNOT & q_0 & q_1 \\
 & H & q_2 \\
 & H & q_0 \\
 & H & q_1 \\
 & X & q_0 \\
 & X & q_1 \\
CZ & q_0 & q_1 \\
 & X & q_0 \\
 & X & q_1 \\
 & H & q_0 \\
 & H & q_1
\end{array}
\tag{5.6}
$$

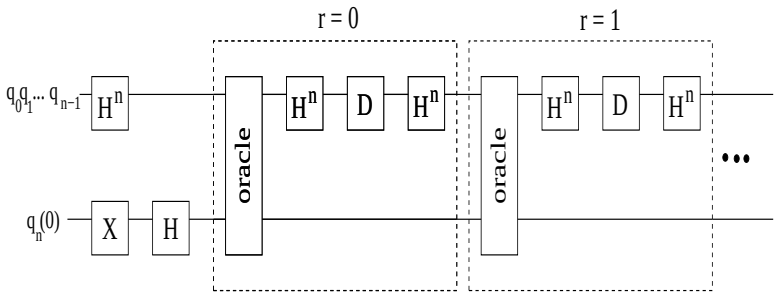


FIGURE 5.1: Circuit structure for Grover's search algorithm

5.2 Results

We have applied our error probability analysis on the above-mentioned four benchmarks circuits: 4-bit adder, 8-bit adder, multiplier and circuit for Grover's algorithm. We have calculated the number of EC blocks required by our algorithm and compared it with the original number of EC blocks required, had they been placed after each component.

For convenience, we enlist the assumptions we made in our methodology:

- Only gate errors and no memory or communication errors have been taken into account while tracing error probabilities in quantum logic circuits.
- No difference between bit flip and phase flip errors have been made.
- State preparation and EC blocks have 0 probability.
- Error probability of primitive gate for a particular PMD at physical level is same.

We tabulate the results in eight tables from Table 5.4, to Table 5.11. In all the tables, we have used certain symbols: Th stands for threshold considered, Concat denotes the level of encoding, Orig. denotes the original number of EC blocks, % save denotes the saving our algorithm has in terms of EC blocks. In the columns of Th, the number of EC blocks obtained by our algorithm has been enlisted. BS, S and K respectively denote Bacon Shor, Steane and Knill codes.

For convenience of the readers, we are tabulating the error probability of the gates in FTS library at the physical and logical level in Table 5.1, Table 5.2 and Table 5.3. It has been assumed that all the primitive gates specific to a particular PMD has error probability w at physical level and this value is given as input(Table 4.1). All other error probabilities at physical level are derived as functions of w. It must be noted that the error probabilities at logical level for R_z, R_x and R_y have already been tabulated in

TABLE 5.1: Error probability and delay of FTS gates at physical level for each PMD

Gates	QD	SC	LP	NLP	IT	NA
R_x	w	w	w	w	w	w
R_y	$1-(1-w)^3$	w	w	w	w	w
R_z	w	w	w	w	w	w
X	w	w	w	w	w	w
Y	$1-(1-w)^2$	w	w	w	w	w
Z	w	w	w	w	w	w
H	$1-(1-w)^3$	$1-(1-w)^3$	w	w	$1-(1-w)^3$	$1-(1-w)^3$
S	w	w	w	w	w	w
T	w	w	w	w	w	w
SWAP	$1-(1-w)^{13}$	$1-(1-w)^4$	w	$1-(1-w)^3$	$1-(1-w)^{11}$	$1-(1-w)^9$
CNOT	$1-(1-w)^5$	$1-(1-w)^7$	w	w	$1-(1-w)^5$	$1-(1-w)^3$
CZ	w	w	w	$1-(1-w)^3$	$1-(1-w)^3$	w
G	$1-(1-w)^9$	$1-(1-w)^3$	$1-(1-w)^3$	$1-(1-w)^3$	w	$1-(1-w)^5$
ZENO	$1-(1-w)^{12}$	w	w	$1-(1-w)^8$	$1-(1-w)^{16}$	$1-(1-w)^{12}$

Table 4.3, Table 4.4 and Table 4.5 respectively. The notations are as defined before in Chapter 4.

5.3 Observations and Inference

From the results we have obtained, we make certain interesting observations and have deduced certain interferences:

- The number of EC blocks obtained by using our algorithm is significantly less than that required if we would have placed EC block after every circuit component. Thus considerable reduction in resources can be obtained.
- Clearly, not every PMD is equally efficient from the aspect of component error probability. PMD s like QD score much less in this regard, while IT and SC are preferred ones.
- Even with same PMD, number of EC blocks required also depends upon the QECC used.
- We observe that usually with increasing level of concatenation, number of EC is reduced, but then increasing the concatenation level itself increases the number of circuit components. So there is a trade-off between these two.
- With increasing concatenation level, memory error increases considerably, so the decrease in error probability with increased concatenation is not as rapid as expected.

TABLE 5.2: Error probability and delay of FTS(1) gates at logical level for each QECC

Gates	Bacon Shor	Steane	Knill
X	$1-(1-g_{n-1_X})^9-\binom{9}{1}g_{n-1_X}(1-g_{n-1_X})^8$	$1-(1-g_{n-1_X})^7-\binom{7}{1}g_{n-1_X}(1-g_{n-1_X})^6$	$1-(1-g_{n-1_X})^2-\binom{2}{1}g_{n-1_X}(1-g_{n-1_X})$
Y	$1-(1-g_{n-1_Y})^9-\binom{9}{1}g_{n-1_Y}(1-g_{n-1_Y})^8$	$1-(1-g_{n-1_Y})^7-\binom{7}{1}g_{n-1_Y}(1-g_{n-1_Y})^6$	$1-(1-g_{n-1_Y})(1-g_{n-1_X})(1-g_{n-1_Z})-g_{n-1_Y}(1-g_{n-1_X})(1-g_{n-1_Z})-g_{n-1_X}(1-g_{n-1_Y})(1-g_{n-1_Z})-g_{n-1_Z}(1-g_{n-1_X})(1-g_{n-1_Y})$
Z	$1-(1-g_{n-1_Z})^9-\binom{9}{1}g_{n-1_Z}(1-g_{n-1_Z})^8$	$1-(1-g_{n-1_Z})^7-\binom{7}{1}g_{n-1_Z}(1-g_{n-1_Z})^6$	$1-(1-g_{n-1_Z})^2-\binom{2}{1}g_{n-1_Z}(1-g_{n-1_Z})$
H	$1-(1-g_{n-1_H})^9(1-g_{n-1_{SWAP}})^{16}(1-M_{n_H})-\binom{9}{1}g_{n-1_H}(1-g_{n-1_H})^8(1-g_{n-1_{SWAP}})^{16}(1-M_{n_H})-M_{n_H}(1-g_{n-1_H})^9(1-g_{n-1_{SWAP}})^{16}$	$1-(1-g_{n-1_H})^7-\binom{7}{1}g_{n-1_H}(1-g_{n-1_H})^6$	$1-(1-g_{n-1_H})^4-\binom{4}{1}g_{n-1_H}(1-g_{n-1_H})^3$
S	$1-[(1-g_{n-1_H})^9(1-g_{n-1_{CNOT}})^9(1-g_{n-1_{SWAP}})^{42}+\binom{21}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{41}(1-g_{n-1_H})^9(1-g_{n-1_{CNOT}})^9+\binom{9}{1}g_{n-1_H}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_H})^8(1-g_{n-1_{CNOT}})^9+\binom{9}{1}g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_H})^9(1-g_{n-1_{CNOT}})^8](1-M_{n_S})-M_{n_S}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_H})^9(1-g_{n-1_{CNOT}})^9$	$1-(1-g_{n-1_S})^7(1-g_{n-1_Z})^7-\binom{7}{1}g_{n-1_S}(1-g_{n-1_S})^6(1-g_{n-1_Z})^7$	$1-[(1-g_{n-1_{SWAP}})^{20}(1-g_{n-1_{CNOT}})^4(1-g_{n-1_H})^4+\binom{16}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{19}(1-g_{n-1_{CNOT}})^4(1-g_{n-1_H})^4+\binom{4}{1}g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{20}(1-g_{n-1_{CNOT}})^3(1-g_{n-1_H})^4+\binom{4}{1}g_{n-1_H}(1-g_{n-1_{SWAP}})^{20}(1-g_{n-1_{CNOT}})^4(1-g_{n-1_H})^3](1-M_{n_S})-M_{n_S}(1-g_{n-1_{SWAP}})^{20}(1-g_{n-1_{CNOT}})^4(1-g_{n-1_H})^4$
T	$1-[(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_{CNOT}})^{18}(1-g_{n-1_H})^9(1-g_{n-1_{Mz}})^9+\binom{21}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{41}(1-g_{n-1_{CNOT}})^{18}(1-g_{n-1_H})^9(1-g_{n-1_{Mz}})^9+\binom{18}{1}g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_{CNOT}})^{17}(1-g_{n-1_H})^9(1-g_{n-1_{Mz}})^9+\binom{9}{1}g_{n-1_H}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_{CNOT}})^{18}(1-g_{n-1_H})^8(1-g_{n-1_{Mz}})^9+\binom{9}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_{CNOT}})^{18}(1-g_{n-1_H})^9(1-g_{n-1_{Mz}})^8](1-M_{n_S})-M_{n_S}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_{CNOT}})^{18}(1-g_{n-1_H})^9(1-g_{n-1_{Mz}})^9$	$1-[(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7+\binom{22}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{39}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7+\binom{7}{1}g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^6(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7+\binom{7}{1}g_{n-1_{Mz}}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^6+\binom{7}{1}g_{n-1_X}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^6(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7](1-M_{n_T})-M_{n_T}(1-g_{n-1_{SWAP}})^{40}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_X})^7(1-g_{n-1_S})^7(1-g_{n-1_{Mz}})^7$	$1-[(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1_{CNOT}})^8(1-g_{n-1_H})^4(1-g_{n-1_{Mz}})^4+\binom{20}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{23}(1-g_{n-1_{CNOT}})^8(1-g_{n-1_H})^4(1-g_{n-1_{Mz}})^4+\binom{8}{1}g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1_{CNOT}})^7(1-g_{n-1_H})^4(1-g_{n-1_{Mz}})^4+\binom{4}{1}g_{n-1_H}(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1_{CNOT}})^8(1-g_{n-1_H})^3(1-g_{n-1_{Mz}})^4+\binom{4}{1}g_{n-1_{Mz}}(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1_{CNOT}})^8(1-g_{n-1_H})^4(1-g_{n-1_{Mz}})^3](1-M_{n_T})-M_{n_T}(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1_{CNOT}})^8(1-g_{n-1_H})^4(1-g_{n-1_{Mz}})^4$

TABLE 5.3: Error probability and delay of FTS(2) gates at logical level for each QECC

Gates	Bacon Shor	Steane	Knill
SWAP	$1-(1-g_{n-1_{SWAP}})^{33}-\binom{12}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{32}$	$1-[(1-g_{n-1_{SWAP}})^{26}+8g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{25}](1-M_{n_{SWAP}})-M_{n_{SWAP}}(1-g_{n-1_{SWAP}})^{26}$	$1-[(1-g_{n-1_{SWAP}})^{40}+\binom{32}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{39}](1-M_{n_{SWAP}})-M_{n_{SWAP}}(1-g_{n-1_{SWAP}})^{40}$
CNOT	$1-(1-g_{n-1_{CNOT}})^{9}(1-g_{n-1_{SWAP}})^{54}-\binom{36}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{53}(1-g_{n-1_{CNOT}})^{9}-\binom{9}{1}g_{n-1_{CNOT}}(1-g_{n-1_{CNOT}})^{8}(1-g_{n-1_{SWAP}})^{54}$	$1-[(1-g_{n-1_{SWAP}})^{43}(1-g_{n-1_{CNOT}})^{7}+\binom{26}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{42}(1-g_{n-1_{CNOT}})^{7}+\binom{7}{1}g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{43}(1-g_{n-1_{CNOT}})^{6}](1-M_{n_{CNOT}})-M_{n_{CNOT}}(1-g_{n-1_{SWAP}})^{43}(1-g_{n-1_{CNOT}})^{7}$	$1-(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1_{CNOT}})^{4}-\binom{4}{1}g_{n-1_{CNOT}}(1-g_{n-1_{SWAP}})^{24}(1-g_{n-1_{CNOT}})^{3}-\binom{22}{1}g_{n-1_{SWAP}}(1-g_{n-1_{SWAP}})^{23}(1-g_{n-1_{CNOT}})^{4}$
CZ	$1-(1-g_{n_H})(1-g_{n_{CNOT}})$	$1-(1-g_{n_H})(1-g_{n_{CNOT}})$	$1-(1-g_{n_H})(1-g_{n_{CNOT}})$
G	$1-(1-g_{n_S})^2(1-g_{n_H})(1-g_{n_{CNOT}})$	$1-(1-g_{n_S})^2(1-g_{n_H})(1-g_{n_{CNOT}})$	$1-(1-g_{n_S})^2(1-g_{n_H})(1-g_{n_{CNOT}})$
ZENO	$1-(1-g_{n_S})^2(1-g_{n_{SWAP}})(1-g_{n_{CZ}})$	$1-(1-g_{n_S})^2(1-g_{n_{SWAP}})(1-g_{n_{CZ}})$	$1-(1-g_{n_S})^2(1-g_{n_{SWAP}})(1-g_{n_{CZ}})$

TABLE 5.4: Savings (in %) on EC blocks for 4-bit Adder circuit on IT, SC and NA

PMD	QECC	Concat	Orig.	Th=0.7	% save	Th=0.8	%save	Th=0.9	% save
IT	BS	1	190	0	100.000	0	100.000	0	100.000
		2	1710	139	91.871	139	91.871	139	91.871
		3	15390	139	90.097	139	90.097	139	90.097
		4	138510	4	99.997	6	99.996	2	99.999
		5	1246590	55	99.995	50	99.996	45	99.996
IT	S	1	190	0	100.000	0	100.000	0	100.000
		2	1330	42	96.842	42	96.842	41	96.917
		3	9310	90	99.033	90	99.033	90	99.033
		4	65170	90	99.862	90	99.862	90	99.862
		5	456190	90	99.980	90	99.980	90	99.980
IT	K	1	190	0	100.000	0	100.000	0	100.000
		2	760	47	93.816	47	93.816	37	95.132
		3	3040	139	95.428	139	95.428	139	95.428
		4	12160	190	98.437	190	98.437	190	98.437
		5	48640	190	99.609	190	99.609	190	99.609
SC	BS	1	190	0	100.000	0	100.000	0	100.000
		2	1710	139	91.871	139	91.871	139	91.871
		3	15390	45	99.708	42	99.727	39	99.747
		4	138510	190	99.863	190	99.863	190	99.863
		5	1246590	190	99.985	190	99.985	190	99.985
SC	S	1	190	0	100.000	0	100.000	0	100.000
		2	1330	90	93.233	90	93.233	90	93.233
		3	9310	90	99.033	90	99.033	90	99.033
		4	65170	90	99.862	90	99.862	90	99.862
		5	456190	90	99.980	90	99.980	90	99.980
SC	K	1	190	0	100.000	0	100.000	0	100.000
		2	760	139	81.711	139	81.711	139	81.711
		3	3040	190	93.750	190	93.750	190	93.750
		4	12160	190	98.437	190	98.437	190	98.437
		5	48640	190	99.609	190	99.609	190	99.609
NA	BS	1	190	190	00.000	190	00.000	190	00.000
		2	1710	190	88.889	190	88.889	190	88.889
		3	15390	190	98.765	190	98.765	190	98.765
		4	138510	190	99.863	190	99.863	190	99.863
		5	1246590	190	99.985	190	99.985	190	99.985
NA	S	1	190	90	52.632	90	52.632	60	68.421
		2	1330	90	93.233	90	93.233	90	93.233
		3	9310	90	99.033	90	99.033	90	99.033
		4	65170	90	99.862	90	99.862	90	99.862
		5	456190	90	99.980	90	99.980	90	99.980
NA	K	1	190	92	51.579	92	51.579	92	51.579
		2	760	190	75.000	190	75.000	190	75.000
		3	3040	190	93.750	190	93.750	190	93.750
		4	12160	190	98.437	190	98.437	190	98.437
		5	48640	190	99.609	190	99.609	190	99.609

TABLE 5.5: Savings (in %) on EC blocks for 4-bit Adder circuit on LP, NLP and QD

PMD	QECC	Concat	Orig.	Th=0.7	% save	Th=0.8	%save	Th=0.9	% save
LP	BS	1	105	101	03.810	101	03.810	101	03.810
		2	945	101	89.312	101	89.312	101	89.312
		3	8505	105	98.765	105	98.765	105	98.765
		4	76545	105	99.863	105	99.863	105	99.863
		5	688905	105	99.985	105	99.985	105	99.985
LP	S	1	105	98	06.667	98	06.667	98	06.667
		2	735	101	86.258	101	86.258	101	86.258
		3	5145	101	98.037	101	98.037	101	98.037
		4	36015	101	99.720	101	99.720	101	99.720
		5	252105	105	99.958	105	99.958	105	99.958
LP	K	1	105	101	03.810	101	03.810	70	33.333
		2	420	101	75.952	101	75.952	101	75.952
		3	1680	101	93.988	101	93.988	101	93.988
		4	6720	101	98.497	101	98.497	101	98.497
		5	26880	101	99.624	101	99.624	101	99.624
NLP	BS	1	105	44	58.095	43	59.048	46	56.191
		2	945	101	89.312	101	89.312	101	89.312
		3	8505	101	98.812	101	98.812	101	98.812
		4	76545	101	99.868	101	99.868	101	99.868
		5	688905	101	99.985	101	99.985	101	99.985
NLP	S	1	105	27	74.286	23	78.095	22	79.048
		2	735	98	86.667	98	86.667	98	86.667
		3	5145	98	98.095	98	98.095	98	98.095
		4	36015	98	99.728	98	99.728	98	99.728
		5	252105	98	99.961	98	99.961	98	99.961
NLP	K	1	105	25	76.190	21	80.000	19	81.905
		2	420	101	75.952	70	83.333	70	83.333
		3	1680	101	93.988	101	93.988	101	93.988
		4	6720	101	98.497	101	98.497	101	98.497
		5	26880	101	99.624	101	99.624	101	99.624
QD	BS	1	190	190	00.000	190	00.000	190	00.000
		2	1710	190	88.889	190	88.889	190	88.889
		3	15390	190	98.765	190	98.765	190	98.765
		4	138510	190	99.863	190	99.863	190	99.863
		5	1246590	190	99.985	190	99.985	190	99.985
QD	S	1	190	190	00.000	190	00.000	190	00.000
		2	1330	190	85.714	190	85.714	190	85.714
		3	9310	190	95.759	190	95.759	190	95.759
		4	65170	190	99.708	190	99.708	190	99.708
		5	456190	190	99.958	190	99.958	190	99.958
QD	K	1	190	25	76.190	21	80.000	19	81.905
		2	760	190	75.000	190	75.000	190	75.000
		3	3040	190	93.750	190	93.750	190	93.750
		4	12160	190	98.437	190	98.437	190	98.437
		5	48640	190	99.609	190	99.609	190	99.609

TABLE 5.6: Savings (in %) on EC blocks for 8-bit Adder circuit on IT, SC and NA

PMD	QECC	Concat	Orig.	Th=0.7	% save	Th=0.8	%save	Th=0.9	% save
IT	BS	1	274	0	100.000	0	100.000	0	100.000
		2	2466	105	95.742	105	95.742	105	95.742
		3	22194	105	99.527	105	99.527	105	99.527
		4	199746	3	99.998	0	100.000	0	100.000
		5	1797714	58	99.997	55	99.997	53	99.997
IT	S	1	274	0	100.000	0	100.000	0	100.000
		2	1918	116	93.952	113	94.108	112	94.161
		3	13426	232	98.272	232	98.272	232	98.272
		4	93982	232	99.753	232	99.753	232	99.753
		5	657874	232	99.965	232	99.965	232	99.965
IT	K	1	274	0	100.000	0	100.000	0	100.000
		2	1096	51	95.347	51	95.347	56	94.891
		3	4384	105	97.605	105	97.605	105	97.605
		4	17536	232	98.677	232	98.677	232	98.677
		5	70144	232	99.669	232	99.669	232	99.669
SC	BS	1	274	0	100.000	0	100.000	0	100.000
		2	2466	105	95.742	105	95.742	105	95.742
		3	22194	40	99.820	38	99.829	33	99.851
		4	199746	262	99.869	262	99.869	262	99.869
		5	1797714	262	99.985	262	99.985	262	99.985
SC	S	1	274	0	100.000	0	100.000	0	100.000
		2	1918	232	87.904	232	87.904	232	87.904
		3	13426	232	98.272	232	98.272	232	98.272
		4	93982	232	99.753	232	99.753	232	99.753
		5	657874	232	99.965	232	99.965	232	99.965
SC	K	1	274	0	100.000	0	100.000	0	100.000
		2	1096	107	90.237	106	90.328	105	90.420
		3	4384	232	94.708	232	99.965	232	99.965
		4	17536	232	98.677	232	98.677	232	98.677
		5	70144	232	99.669	232	99.669	232	99.669
NA	BS	1	274	232	15.328	232	15.328	232	15.328
		2	2466	262	89.376	262	89.376	262	89.376
		3	22194	262	98.820	262	98.820	262	98.820
		4	199746	262	99.869	262	99.869	262	99.869
		5	1797714	262	99.985	262	99.985	262	99.985
NA	S	1	274	232	15.328	232	15.328	119	56.569
		2	1918	232	87.904	232	87.904	232	87.904
		3	13426	232	98.272	232	98.272	232	98.272
		4	93982	232	99.753	232	99.753	232	99.753
		5	657874	232	99.965	232	99.965	232	99.965
NA	K	1	274	119	56.569	119	56.569	119	56.569
		2	1096	232	78.832	232	78.832	232	78.832
		3	4384	232	94.708	232	94.708	232	94.708
		4	17536	232	98.677	232	98.677	232	98.677
		5	70144	232	99.669	232	99.669	232	99.669

TABLE 5.7: Savings (in %) on EC blocks for 8-bit Adder circuit on LP, NLP and QD

PMD	QECC	Concat	Orig.	Th=0.7	% save	Th=0.8	%save	Th=0.9	% save
LP	BS	1	274	262	04.380	262	04.380	232	15.328
		2	2466	262	89.376	262	89.376	262	89.376
		3	22194	274	98.765	274	98.765	274	98.765
		4	199746	274	99.863	274	99.863	274	99.863
		5	1797714	274	99.985	274	99.985	274	99.985
LP	S	1	274	232	15.328	232	15.328	232	15.328
		2	1918	232	87.904	232	87.904	232	87.904
		3	13426	232	98.272	232	98.272	232	98.272
		4	93982	274	99.708	274	99.708	274	99.708
		5	657874	274	99.958	274	99.958	274	99.958
LP	K	1	274	1283	12.543	1283	12.543	977	33.401
		2	1096	1283	78.136	1283	78.136	1283	78.136
		3	4384	1283	94.534	1283	94.534	1283	94.534
		4	17536	1283	98.633	1283	98.633	1283	98.633
		5	70144	1283	99.658	1283	99.658	1283	99.658
NLP	BS	1	274	124	54.745	96	64.963	93	66.058
		2	2466	262	89.376	262	89.376	262	89.376
		3	22194	262	98.820	262	98.820	262	98.820
		4	199746	262	99.869	262	99.869	262	99.869
		5	1797714	262	99.985	262	99.985	262	99.985
NLP	S	1	274	45	83.577	43	84.307	45	83.577
		2	1918	232	87.904	232	87.904	232	87.904
		3	13426	232	98.272	232	98.272	232	98.272
		4	93982	232	99.753	232	99.753	232	99.753
		5	657874	232	99.965	232	99.965	232	99.965
NLP	K	1	274	49	82.117	48	82.481	42	84.672
		2	1096	232	78.832	169	84.580	169	84.580
		3	4384	232	94.708	232	94.708	232	94.708
		4	17536	232	98.677	232	98.677	232	98.677
		5	70144	232	99.669	232	99.669	232	99.669
QD	BS	1	274	274	00.000	274	00.000	274	00.000
		2	2466	232	87.904	232	87.904	232	87.904
		3	22194	232	98.272	232	98.272	232	98.272
		4	199746	232	99.753	232	99.753	232	99.753
		5	1797714	232	99.965	232	99.965	232	99.965
QD	S	1	274	274	00.000	274	00.000	274	00.000
		2	1918	274	85.714	274	85.714	274	85.714
		3	13426	274	97.959	274	97.959	274	97.959
		4	93982	274	99.708	274	99.708	274	99.708
		5	657874	274	99.958	274	99.958	274	99.958
QD	K	1	274	274	00.000	274	00.000	274	00.000
		2	1096	274	75.000	274	75.000	274	75.000
		3	4384	274	93.750	274	93.750	274	93.750
		4	17536	274	98.437	274	98.437	262	98.506
		5	70144	274	99.609	262	99.626	262	99.626

TABLE 5.8: Savings (in %) on EC blocks for Multiplier circuit on IT, SC and NA

PMD	QECC	Concat	Orig.	Th=0.7	% save	Th=0.8	%save	Th=0.9	% save
IT	BS	1	1467	0	100.000	0	100.000	0	100.000
		2	13203	645	95.115	645	95.115	645	95.115
		3	118827	645	99.457	645	99.457	645	99.457
		4	1069443	54	99.995	36	99.997	43	99.996
		5	9624987	360	99.996	309	99.997	299	99.997
IT	S	1	1467	0	100.000	0	100.000	0	100.000
		2	10269	711	93.076	666	93.514	663	93.544
		3	71883	1283	98.215	1283	98.215	1283	98.215
		4	503181	1283	99.745	1283	99.745	1283	99.745
		5	3522267	1283	99.964	1283	99.964	1283	99.964
IT	K	1	1467	0	100.000	0	100.000	0	100.000
		2	5868	506	91.377	506	91.377	507	91.360
		3	23472	687	97.073	687	97.073	687	97.073
		4	93888	1283	98.633	1283	98.633	1283	98.633
		5	375552	1283	99.658	1283	99.658	1283	99.658
SC	BS	1	1467	0	100.000	0	100.000	0	100.000
		2	13203	645	95.115	645	95.115	645	95.115
		3	118827	245	99.794	209	99.824	198	99.833
		4	1069443	1467	99.863	1467	99.863	1467	99.863
		5	9624987	1467	99.985	1467	99.985	1467	99.985
SC	S	1	1467	0	100.000	0	100.000	0	100.000
		2	10269	1248	87.847	1248	87.847	1248	87.847
		3	71883	1283	98.215	1283	98.215	1283	98.215
		4	503181	1283	99.745	1283	99.745	1283	99.745
		5	3522267	1283	99.963	1283	99.963	1283	99.963
SC	K	1	1467	0	100.000	0	100.000	0	100.000
		2	5868	667	88.633	665	88.667	663	88.701
		3	23472	1283	94.534	1283	94.534	1283	94.534
		4	93888	1283	98.633	1283	98.633	1283	98.633
		5	375552	1283	99.658	1283	99.658	1283	99.658
NA	BS	1	1467	1301	11.316	1301	11.316	1269	13.497
		2	13203	1467	88.889	1467	88.889	1467	88.889
		3	118827	1467	98.765	1467	98.765	1467	98.765
		4	1069443	1467	99.863	1467	99.863	1467	99.863
		5	9624987	1467	99.985	1467	99.985	1467	99.985
NA	S	1	1467	1283	12.543	1251	14.724	668	54.465
		2	10269	1283	87.506	1283	87.506	1283	87.506
		3	71883	1283	98.215	1283	98.215	1283	98.215
		4	503181	1283	99.745	1283	99.745	1283	99.745
		5	3522267	1283	99.964	1283	99.964	1283	99.964
NA	K	1	1467	670	54.329	670	54.329	654	55.419
		2	5868	1283	78.136	1283	78.136	1283	78.136
		3	23472	1283	94.534	1283	94.534	1283	94.534
		4	93888	1283	98.633	1283	98.633	1283	98.633
		5	375552	1283	99.658	1283	99.658	1283	99.658

TABLE 5.9: Savings (in %) on EC blocks for Multiplier circuit on LP, NLP and QD

PMD	QECC	Concat	Orig.	Th=0.7	% save	Th=0.8	%save	Th=0.9	% save
LP	BS	1	1467	1467	00.000	1467	00.000	1301	11.316
		2	13203	1467	88.889	1467	88.889	1467	88.889
		3	118827	1467	98.765	1467	98.765	1467	98.765
		4	1069443	1467	99.863	1467	99.863	1467	99.863
		5	9624987	1467	99.985	1467	99.985	1467	99.985
LP	S	1	1467	1283	12.543	1283	12.543	1251	14.724
		2	10269	1283	87.506	1283	87.506	1283	87.506
		3	71883	1301	98.190	1301	98.190	1301	98.190
		4	503181	1467	99.708	1467	99.708	1467	99.708
		5	3522267	1467	99.958	1467	99.958	1467	99.958
LP	K	1	1467	1283	12.543	1283	12.543	977	33.401
		2	5868	1283	78.136	1283	78.136	1283	78.136
		3	23472	1283	94.534	1283	94.534	1283	94.534
		4	93888	1283	98.633	1283	98.633	1283	98.633
		5	375552	1283	99.658	1283	99.658	1283	99.658
NLP	BS	1	1467	669	54.397	507	65.440	536	63.463
		2	13203	1467	88.889	1467	88.889	1467	88.889
		3	118827	1467	98.765	1467	98.765	1467	98.765
		4	1069443	1467	99.863	1467	99.863	1467	99.863
		5	9624987	1467	99.985	1467	99.985	1467	99.985
NLP	S	1	1467	275	81.254	243	83.436	238	83.776
		2	10269	1283	87.506	1283	87.506	1283	87.506
		3	71883	1283	98.215	1283	98.215	1283	98.215
		4	503181	1283	99.745	1283	99.745	1283	99.745
		5	3522267	1283	99.964	1283	99.964	1283	99.964
NLP	K	1	1467	242	83.504	234	84.049	210	85.685
		2	5868	1283	78.136	942	83.947	942	83.947
		3	23472	1283	94.534	1283	94.534	1283	94.534
		4	93888	1283	98.633	1283	98.633	1283	98.633
		5	375552	1283	99.658	1283	99.658	1283	99.658
QD	BS	1	1467	1467	00.000	1467	00.000	1467	00.000
		2	13203	1467	88.889	1467	88.889	1467	88.889
		3	118827	1467	98.765	1467	98.765	1467	98.765
		4	1069443	1467	99.863	1467	99.863	1467	99.863
		5	9624987	1467	99.985	1467	99.985	1467	99.985
QD	S	1	1467	1467	00.000	1467	00.000	1467	00.000
		2	10269	1467	85.714	1467	85.714	1467	85.714
		3	71883	1467	97.959	1467	97.959	1467	97.959
		4	503181	1467	99.708	1467	99.708	1467	99.708
		5	3522267	1467	99.958	1467	99.958	1467	99.958
QD	K	1	1467	1467	00.000	1467	00.000	1467	00.000
		2	5868	1467	75.000	1467	75.000	1467	75.000
		3	23472	1467	93.75	1467	93.75	1467	93.75
		4	93888	1467	98.437	1467	98.437	1467	98.437
		5	375552	1467	99.609	1467	99.609	1467	99.609

TABLE 5.10: Comparative results considering Grover's search circuit for IT, SC and NA

PMD	QECC	Concat	Orig.	Th=0.7	% save	Th=0.8	%save	Th=0.9	% save
IT	BS	1	27	0	100.000	0	100.000	0	100.000
		2	243	7	97.119	7	97.119	7	97.119
		3	2187	7	99.680	7	99.680	7	99.680
		4	19683	0	100.000	0	100.000	0	100.000
		5	177147	4	99.998	3	99.998	4	99.998
IT	S	1	27	0	100.000	0	100.000	0	100.000
		2	189	7	96.296	7	96.296	7	96.296
		3	1323	14	98.942	14	98.942	14	98.942
		4	9261	14	99.849	14	99.849	14	99.849
		5	64827	14	99.978	14	99.978	14	99.978
IT	K	1	27	0	100.000	0	100.000	0	100.000
		2	108	2	98.148	2	98.148	0	100.000
		3	432	7	98.380	7	98.380	7	98.380
		4	1728	14	99.190	14	99.190	14	99.190
		5	6912	14	99.797	14	99.797	14	99.797
SC	BS	1	27	0	100.000	0	100.000	0	100.000
		2	243	7	97.119	7	97.119	7	97.119
		3	2187	4	99.817	3	99.862	3	99.862
		4	19683	23	99.883	23	99.883	23	99.883
		5	177147	23	99.987	23	99.987	23	99.987
SC	S	1	27	0	100.000	0	100.000	0	100.000
		2	189	14	92.593	14	92.593	14	92.593
		3	1323	14	98.942	14	98.942	14	98.942
		4	9261	14	99.849	14	99.849	14	99.849
		5	64827	14	99.978	14	99.978	14	99.978
SC	K	1	27	0	100.000	0	100.000	0	100.000
		2	108	7	93.519	7	93.519	7	93.519
		3	432	14	96.759	14	96.759	14	96.759
		4	1728	14	99.190	14	99.190	14	99.190
		5	6912	14	99.797	14	99.797	14	99.797
NA	BS	1	27	15	44.444	15	44.444	15	44.444
		2	243	23	90.535	23	90.535	23	90.535
		3	2187	23	98.948	23	98.948	23	98.948
		4	19683	23	99.883	23	99.883	23	99.883
		5	177147	23	99.987	23	99.987	23	99.987
NA	S	1	27	14	48.148	14	48.148	7	74.074
		2	189	14	92.592	14	92.592	14	92.592
		3	1323	14	98.942	14	98.942	14	98.942
		4	9261	14	99.849	14	99.849	14	99.849
		5	64827	14	99.978	14	99.978	14	99.978
NA	K	1	27	6	77.778	6	77.778	6	77.778
		2	108	14	87.037	14	87.037	14	87.037
		3	432	14	96.759	14	96.759	14	96.759
		4	1728	14	99.190	14	99.190	14	99.190
		5	6912	14	99.797	14	99.797	14	99.797

TABLE 5.11: Comparative results considering Grover's search circuit for LP, NLP and QD

PMD	QECC	Concat	Orig.	Th=0.7	% save	Th=0.8	%save	Th=0.9	% save
LP	BS	1	27	23	14.815	23	14.815	19	29.630
		2	243	23	90.535	23	90.535	23	90.535
		3	2187	27	98.765	27	98.765	27	98.765
		4	19683	27	99.863	27	99.863	27	99.863
		5	177147	27	99.985	27	99.985	27	99.985
LP	S	1	27	14	48.148	14	48.148	14	48.148
		2	189	14	92.592	14	92.592	14	92.592
		3	1323	19	98.564	19	98.564	19	98.564
		4	9261	27	99.708	27	99.708	27	99.708
		5	64827	27	99.958	27	99.958	27	99.958
LP	K	1	27	14	48.148	14	48.148	13	51.852
		2	108	14	87.037	14	87.037	14	87.037
		3	432	14	96.759	14	96.759	14	96.759
		4	1728	14	99.190	14	99.190	14	99.190
		5	6912	14	99.797	14	99.797	14	99.797
NLP	BS	1	27	7	74.074	8	70.370	8	70.370
		2	243	23	90.535	23	90.535	23	90.535
		3	2187	23	98.948	23	98.948	23	98.948
		4	19683	23	99.883	23	99.883	23	99.883
		5	177147	23	99.987	23	99.987	23	99.987
NLP	S	1	27	2	92.592	3	88.889	2	92.592
		2	189	14	92.592	14	92.592	14	92.592
		3	1323	14	98.942	14	98.942	14	98.942
		4	9261	14	99.849	14	99.849	14	99.849
		5	64827	14	99.978	14	99.978	14	99.978
NLP	K	1	27	2	92.592	2	92.592	2	92.592
		2	108	14	87.037	10	90.741	10	90.741
		3	432	14	96.759	14	96.759	14	96.759
		4	1728	14	99.190	14	99.190	14	99.190
		5	6912	14	99.797	14	99.797	14	99.797
QD	BS	1	27	27	00.000	27	00.000	27	00.000
		2	243	27	88.889	27	88.889	27	88.889
		3	2187	27	98.765	27	98.765	27	98.765
		4	19683	27	99.863	27	99.863	27	99.863
		5	177147	27	99.985	27	99.985	27	99.985
QD	S	1	27	27	00.000	27	00.000	27	00.000
		2	189	27	85.714	27	85.714	27	85.714
		3	1323	27	97.959	27	97.959	27	97.959
		4	9261	27	99.708	27	99.708	27	99.708
		5	64827	27	99.958	27	99.958	27	99.958
QD	K	1	27	27	00.000	27	00.000	27	00.000
		2	108	27	75.000	27	75.000	27	75.000
		3	432	27	93.750	27	93.750	27	93.750
		4	1728	27	98.437	27	98.437	23	98.669
		5	6912	27	99.609	23	99.667	23	99.667

Chapter 6

Conclusion

Quantum computers have displayed superior computing power compared to classical computers by being able to solve certain classically intractable problems. Thus they come with the promise of promoting to a different computing paradigm. The main strength of quantum computing lies in the phenomena of superposition and entanglement, but decoherence is the main obstacle in the realization of a practical quantum computer. To protect quantum information from noise, different QECCs have been developed. The threshold theorem stating that as long as the error probability of each component is below a certain threshold, fault-tolerant quantum computing is possible, avoids the situation to become hopeless.

Ideally error correction blocks should be placed after every component in a quantum circuit and thus these constitute a significant portion of encoded fault-tolerant quantum circuits. To overcome this problem, we have devised an algorithm to trace error probability in a logical quantum circuit and place the error-correction sub-circuit only when error probability exceeds a certain pre-defined threshold. We have also developed an error model, by which we have calculated the error probability of gates both at the physical level and logical level. To the best of our knowledge, such analysis has not been done before. Our algorithm takes care of the QECC used for encoding as well as the PMD used.

We have developed a tool in C, that takes as input the error probability of the primitive gates of each PMD, the threshold considered, the number of levels of encoding and the quantum logic circuit in QASM format. The tool prints the error probability of each gate in the FTS set both at physical and logical level and also the gate delays, that are very helpful in calculating memory error probability. It also prints the error probability after each component in the input logical circuit and the the critical path. According to threshold considered, the tool places error-correction blocks at specific locations

and modifies the error probabilities after each component in the circuit. Applying this method on some benchmark circuits, we have shown our algorithm can cause significant reduction in the number of error-correction blocks in a logical quantum circuit.

We would like to emphasize again that we had very limited access to error probability of primitive gates at the physical level for each PMD. We have already mentioned the source from which we have taken our values and the assumptions we made. The results depend a lot on these values. We chose the threshold according to the values given to us. So, with change in input error probabilities of primitive gates, all these are highly likely to change. However, the error model, calculated formulae for error probabilities, the underlying algorithm and the tool developed will still hold.

6.1 Future Scope

The future scope of this work are:

- We plan to include other codes like surface code within our model and calculate the error probabilities.
- The analysis we did is post-processing. We plan to integrate it with some logic synthesis algorithms like FTQLS.
- In our error model and error analyis technique we have not taken care of layuot level errors. The placement of tiles is also an important factor in the transport of qubits and hence contribute in the erorr probability arising from transportation. Except for local SWAP based transport of qubits, we have not taken care of transport over some significantly long distance. This important aspect is worth analysing and including in the error calculations.
- The analysis we made can be made more rigorous by considering error probability of preparation circuits, ancilla verification circuits and error-correction blocks. We assumed all these are without error.
- We have done the calculation of error probability for the gates in the FTS set. It would be helpful as well as interesting to calculate error probabilities for other non-FTS gates like CP which is extensively used in quantum circuits.

Appendix A

Tile operations for CTL gates using the Bacon Shor code

In all the diagrams, data qubits are represented by d_1, d_2, etc., and ancilla qubits are noted as a_1, a_2, etc.The dummy qubits used to prepare and measure the ancilla as well as for qubit transport are represented by O. The preparation of ancilla qubits is noted by $P_{X,Z}(a_j)$, which represents the preparation of the qubit in the eigenstate of either X or Z, with eigenvalue 1. The single-headed thin arrows represent CNOT gates, pointing from source to target qubits; the double-headed thick arrows represent SWAP gates; and qubit measurements in the X or Z bases are represented by $M_{X,Z}()$. It must be noted that though SWAP is not a CTL gate, but the tile structure has been drawn because in our architecture SWAP is the only mode of local transport.

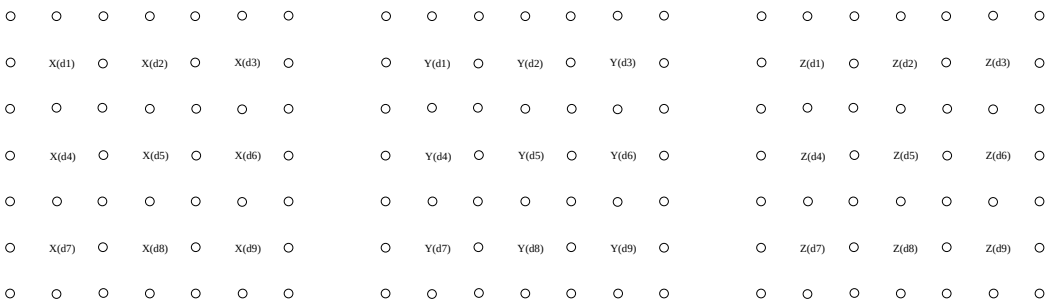
The encoding of gates have been shown through different time steps. One time step is the maximum time taken when each physical qubit undergoes at most one operation. It must be noted that in no way can more than one operation be done on a single qubit within a single time step. The total number of time steps required for the encoding of a gate gives the latency. The latencies of CTL gates encoded with Bacon Shor code has been summarized in Table C.1.

TABLE A.1: Latency of CTL gates encoded with the Bacon Shor code

Gate	Latency
X	1
Y	1
Z	1
H	5
S	12
T	15
SWAP	7
CNOT	9

A.1 Pauli Gates

Time step 1:

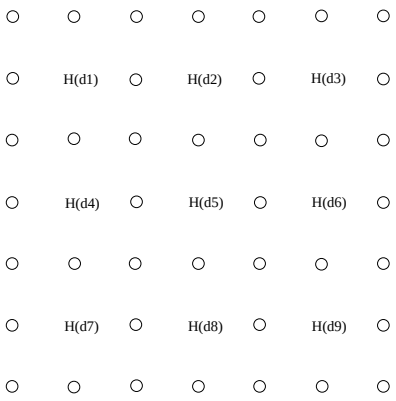


(a) Encoded X gate (b) Encoded Y gate (c) Encoded Z gate

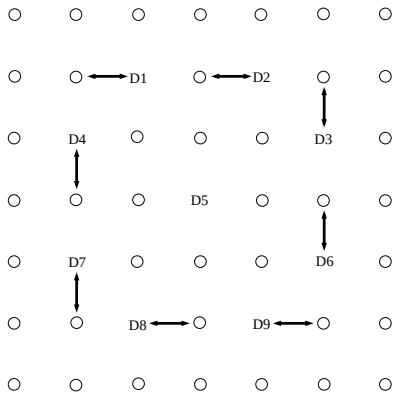
FIGURE A.1: Encoded Pauli Gates for the Bacon Shor code

A.2 H

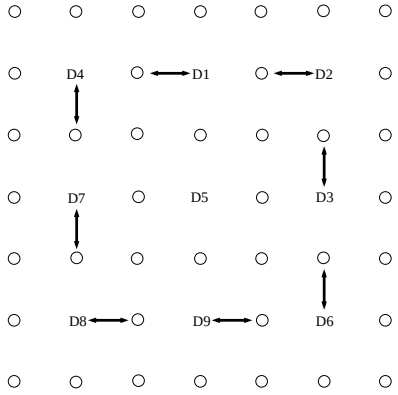
Time step 1:



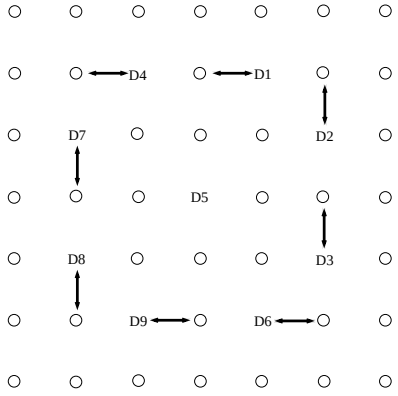
Time step 2:



Time step 3:



Time step 4:



Time step 5:

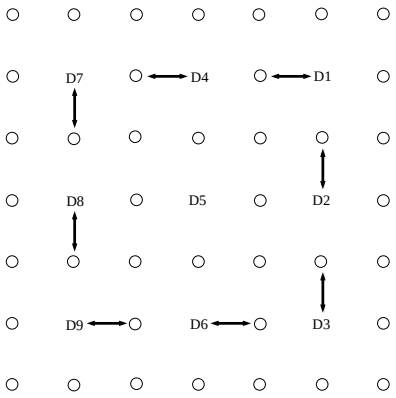
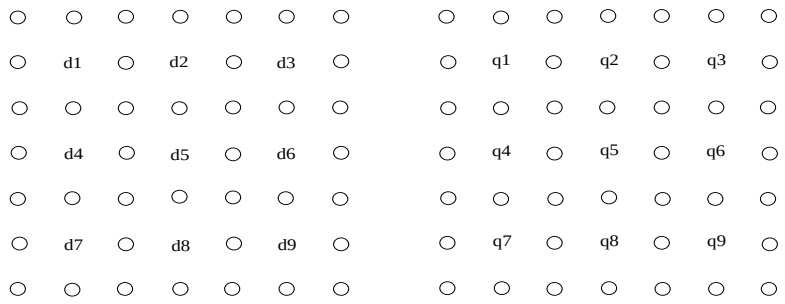


FIGURE A.2: Encoded H gate for the Bacon Shor code

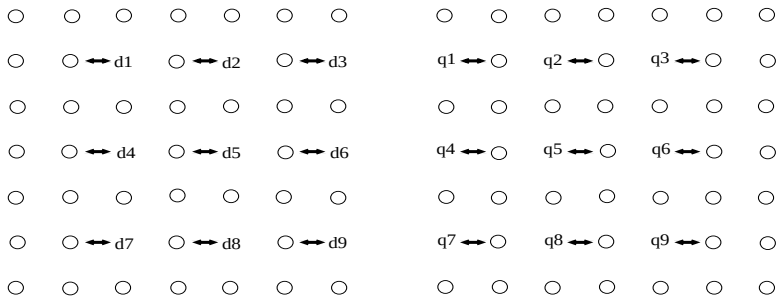
A.3 SWAP

An encoded SWAP operation between $|d_1 d_2 \dots d_9\rangle$ and $|q_1 q_2 \dots q_9\rangle$.

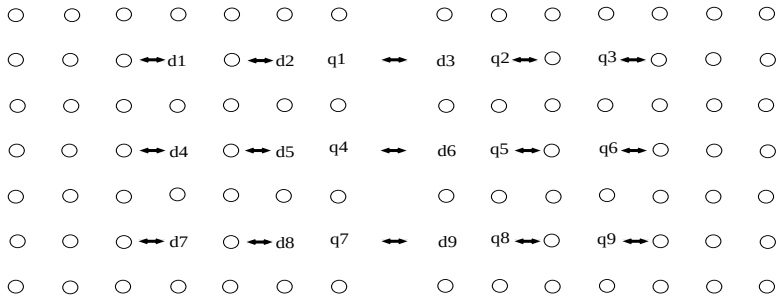
Time step 0:



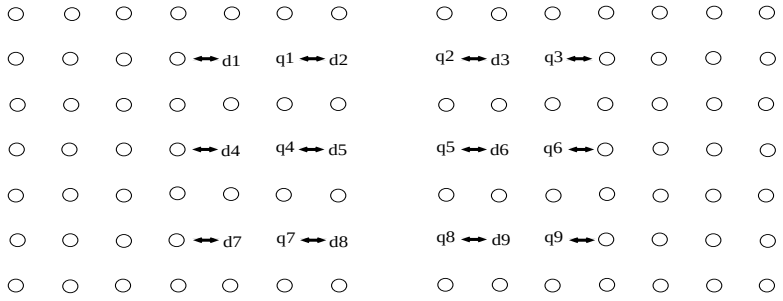
Time step 1:



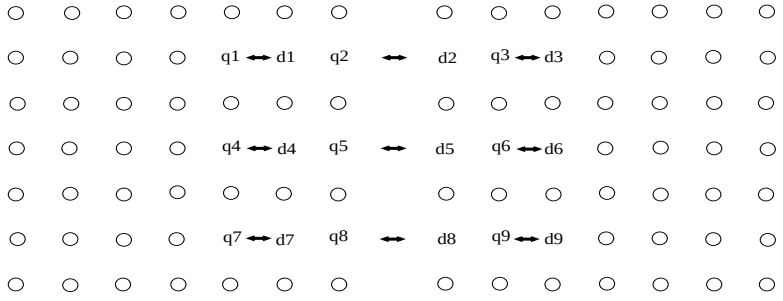
Time step 2:



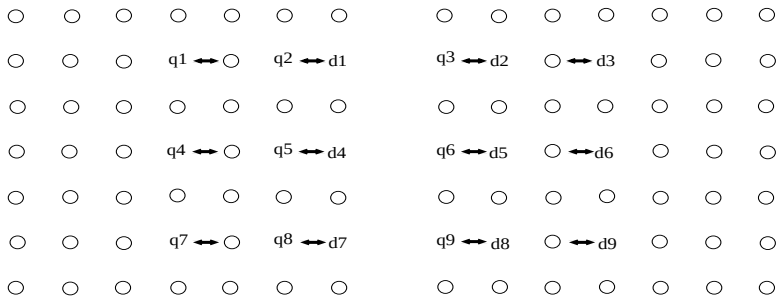
Time step 3:



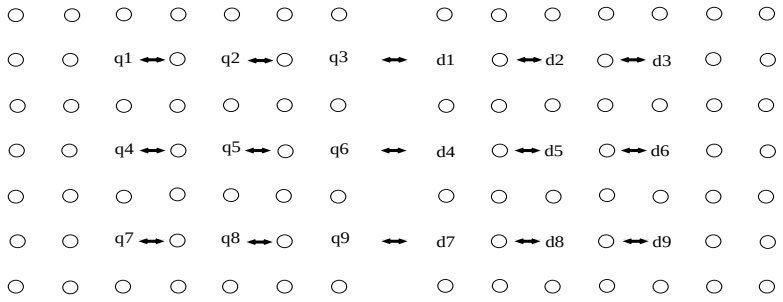
Time step 4:



Time step 5:



Time step 6:



Time step 7:

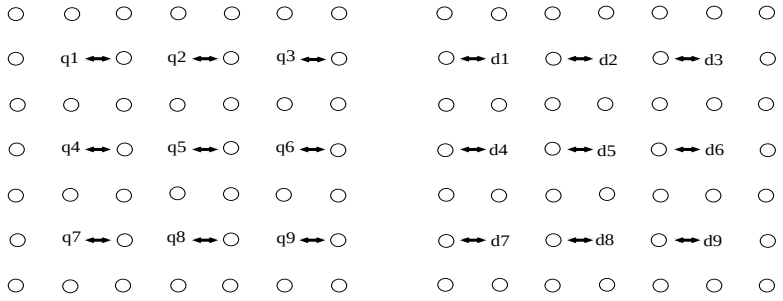
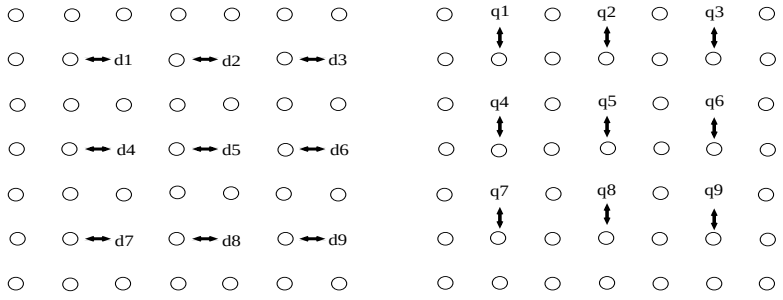


FIGURE A.3: Encoded SWAP gate for the Bacon Shor code

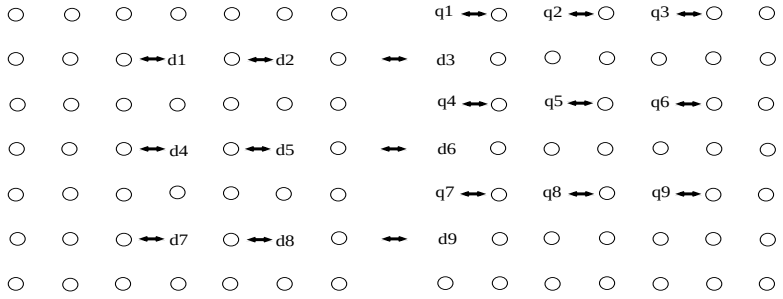
A.4 CNOT

An encoded CNOT operation between $|d_1 d_2 \dots d_9\rangle$ (control) and $|q_1 q_2 \dots q_9\rangle$ (target)

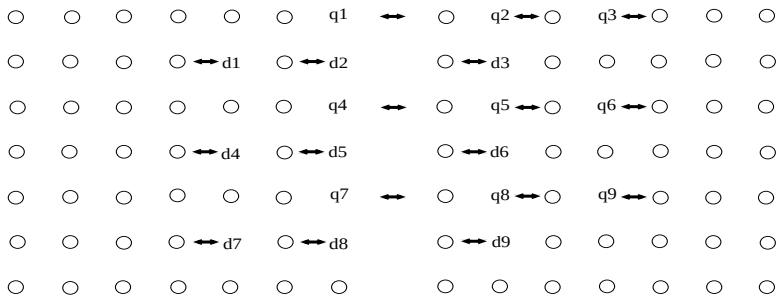
Time step 1:



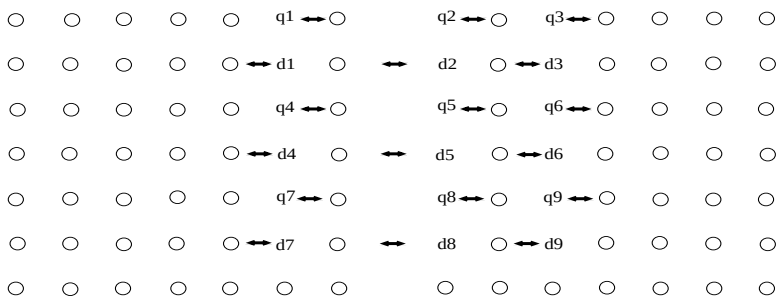
Time step 2:



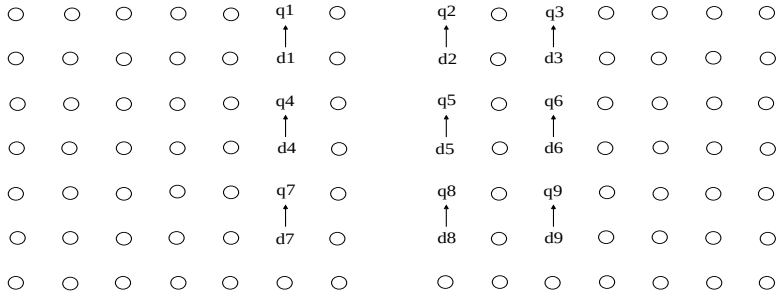
Time step 3:



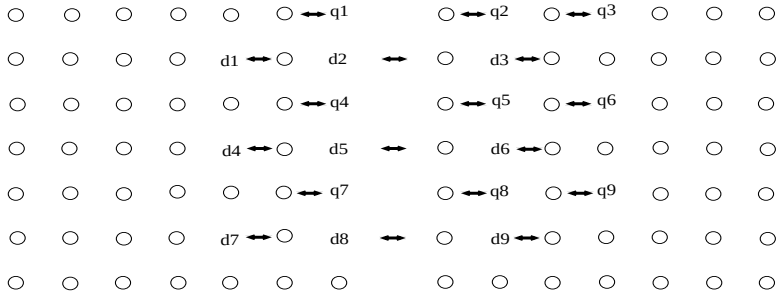
Time step 4:



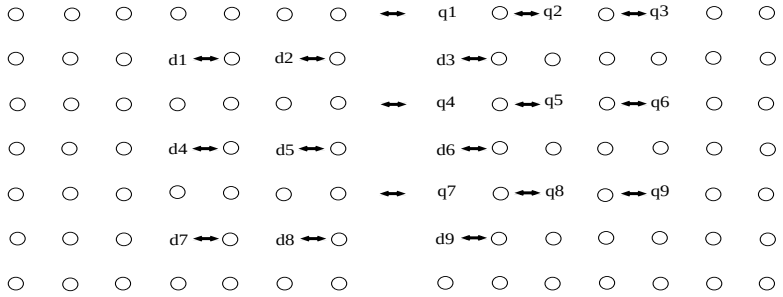
Time step 5:



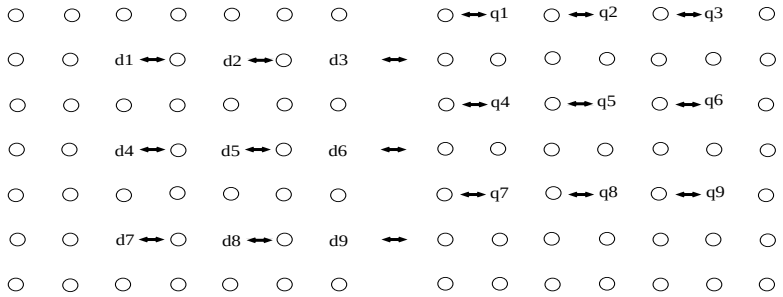
Time step 6:



Time step 7:



Time step 8:



Time step 9:

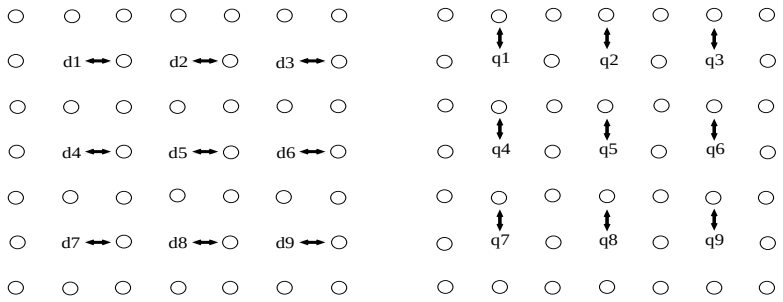
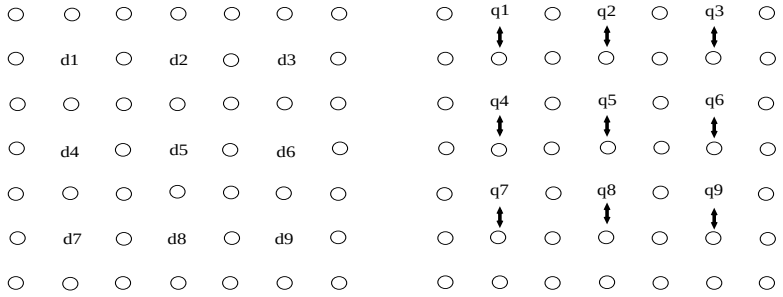


FIGURE A.4: Encoded CNOT gate for the Bacon Shor code

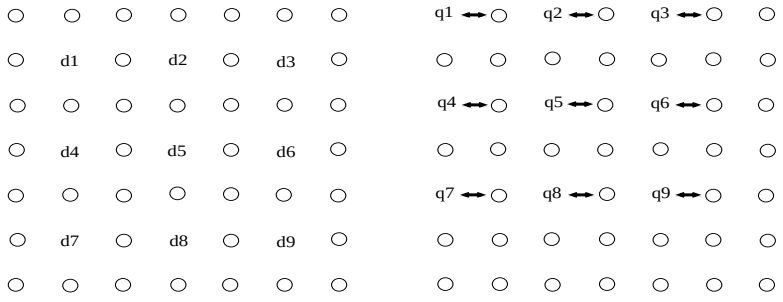
A.5 S

An encoded S operation on $|d_1 d_2 \dots d_9\rangle$. We assume $|q_1 q_2 \dots q_9\rangle$ is in $\left|\overline{+i}\right\rangle$ state.

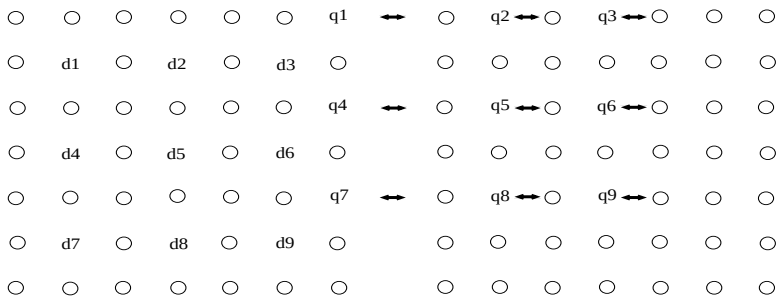
Time step 1:



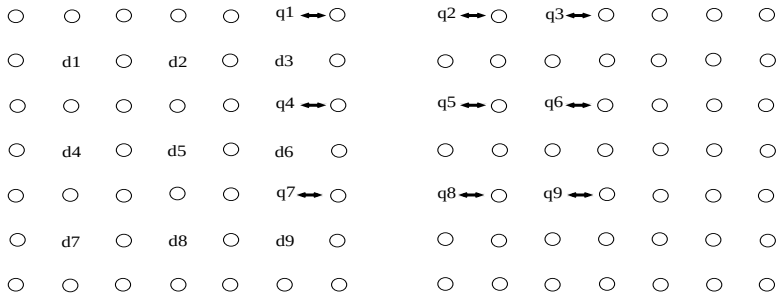
Time step 2:



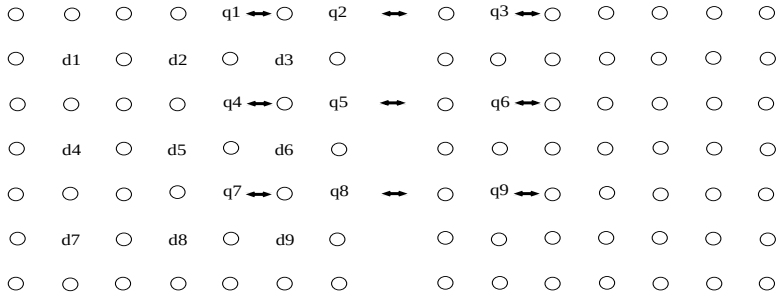
Time step 3:



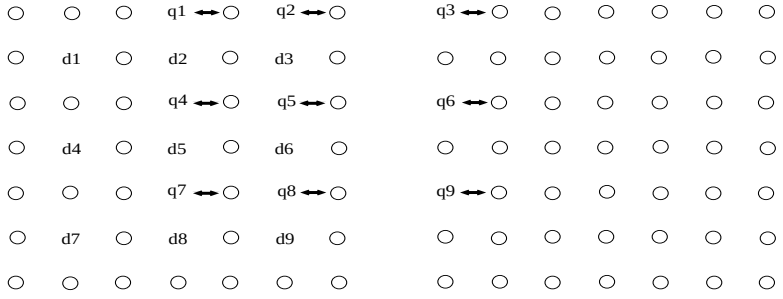
Time step 4:



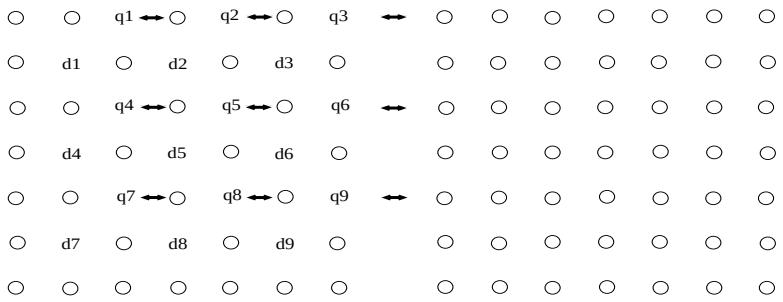
Time step 5:



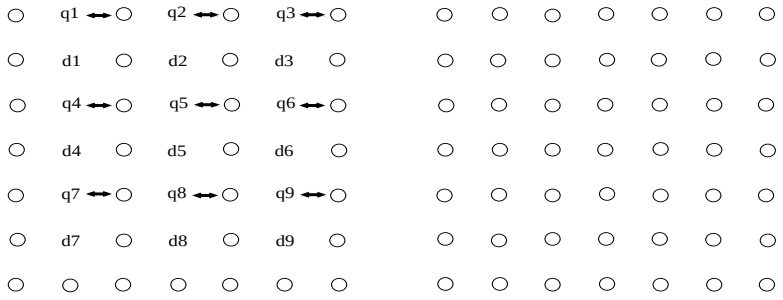
Time step 6:



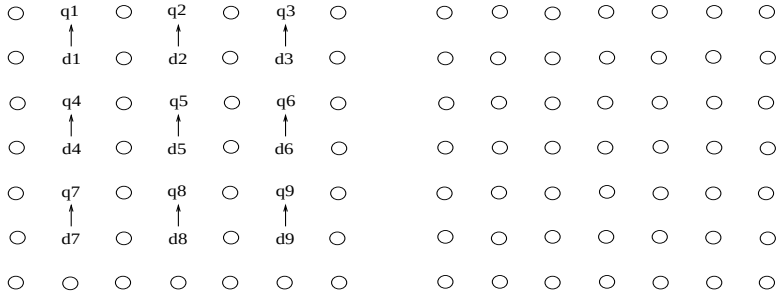
Time step 7:



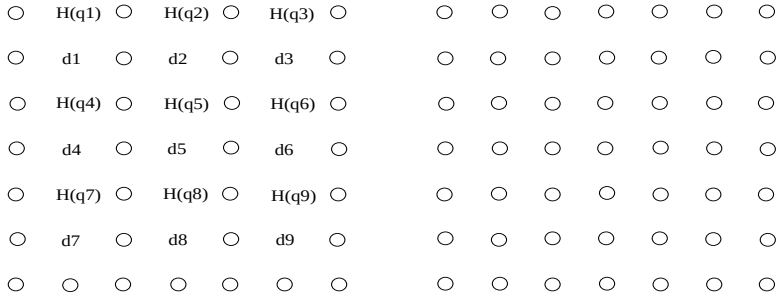
Time step 8:



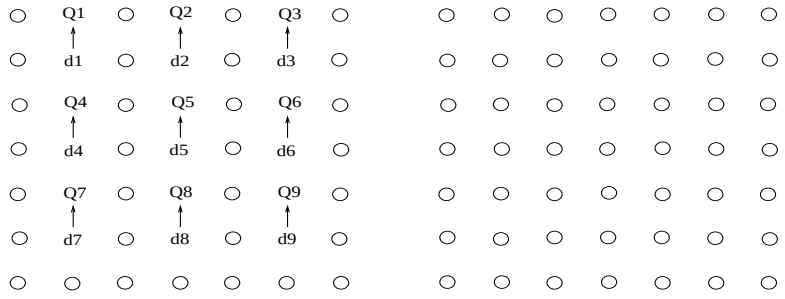
Time step 9:



Time step 10:



Time step 11:



Time step 12:

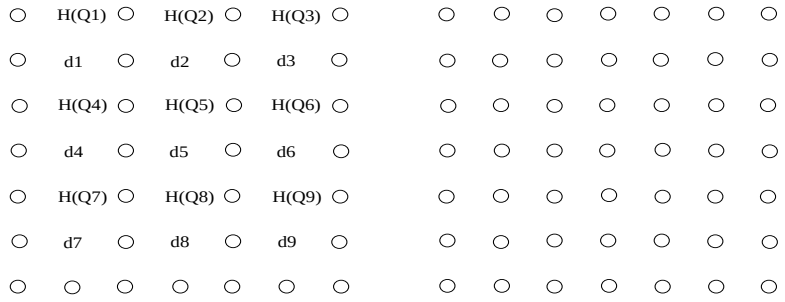
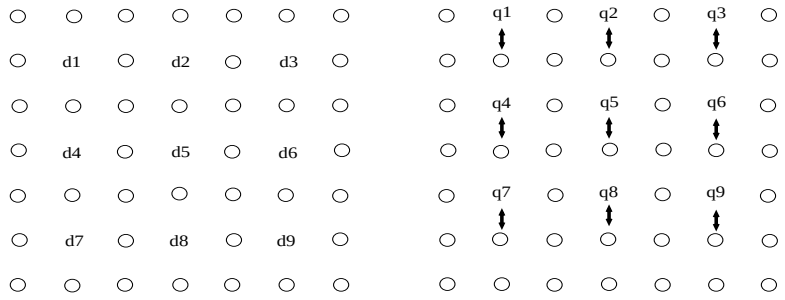


FIGURE A.5: Encoded S gate for the Bacon Shor code

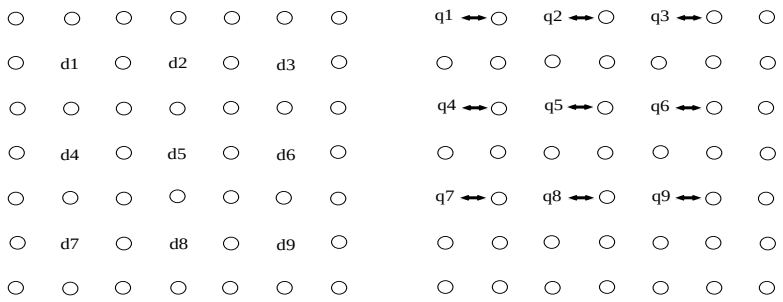
A.6 T

An encoded T operation on $|d_1 d_2 \dots d_9\rangle$. We assume $|q_1 q_2 \dots q_9\rangle$ is in $T|\overline{+}\rangle$ state.

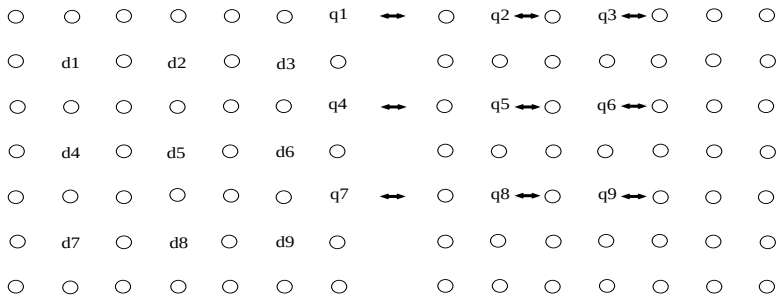
Time step 1:



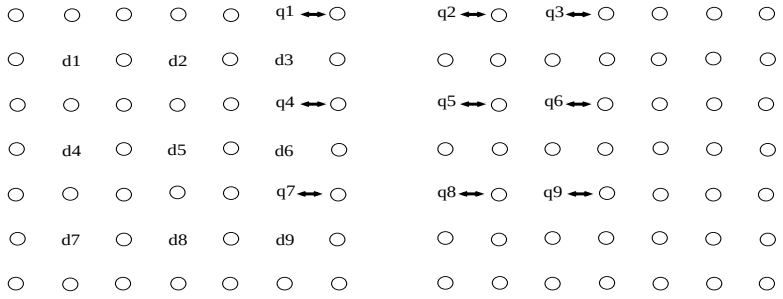
Time step 2:



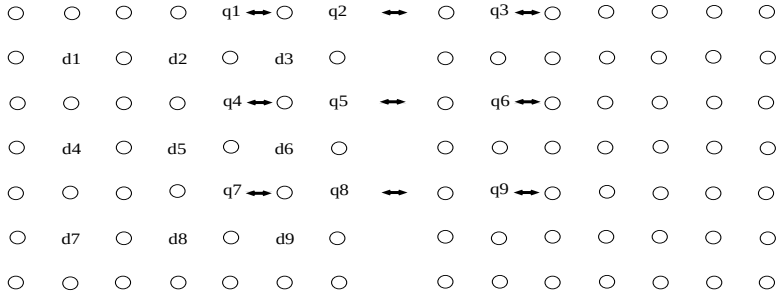
Time step 3:



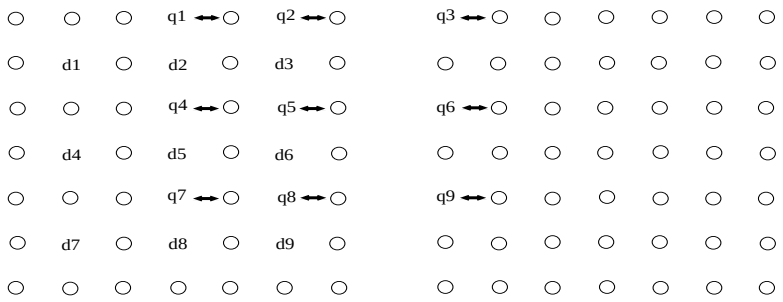
Time step 4:



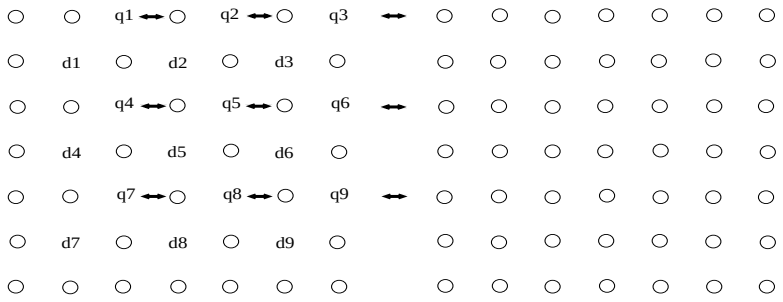
Time step 5:



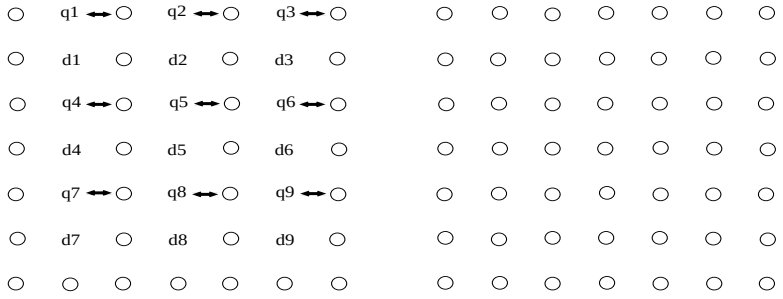
Time step 6:



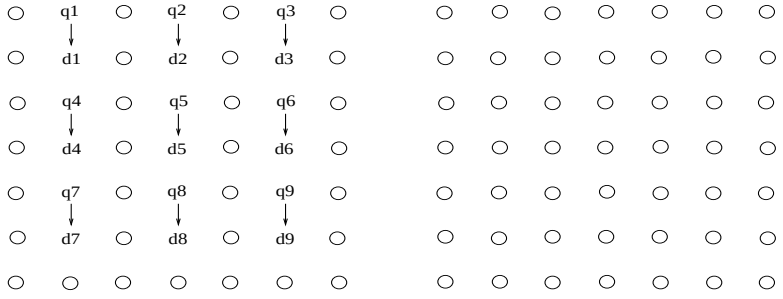
Time step 7:



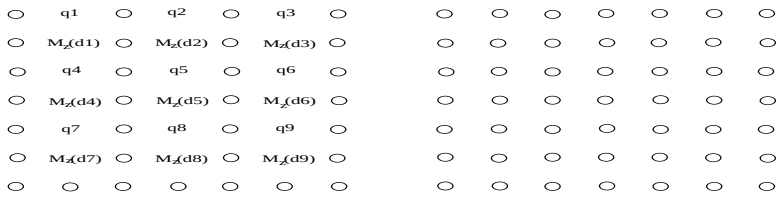
Time step 8:



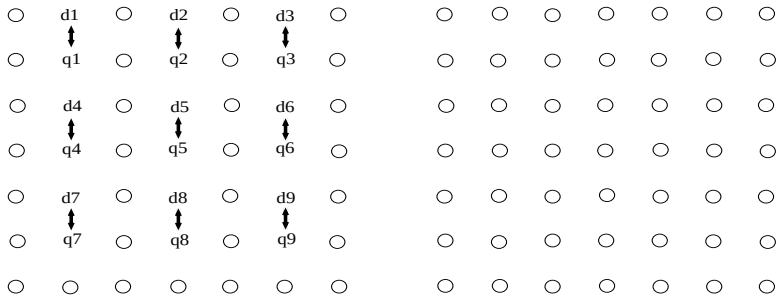
Time step 9:



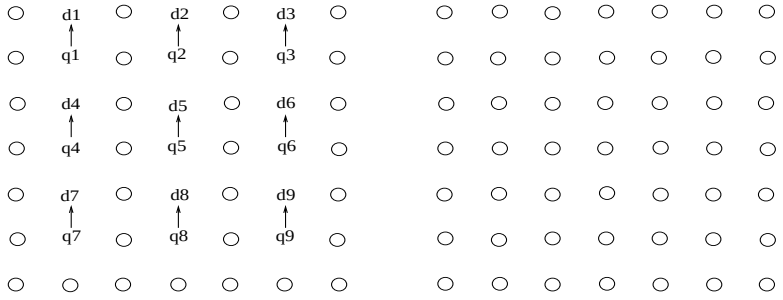
Time step 10:



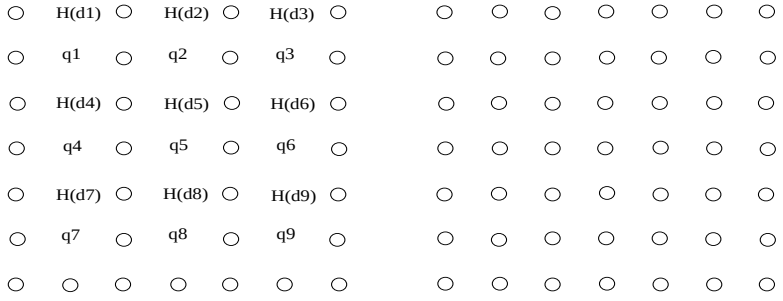
Time step 11:



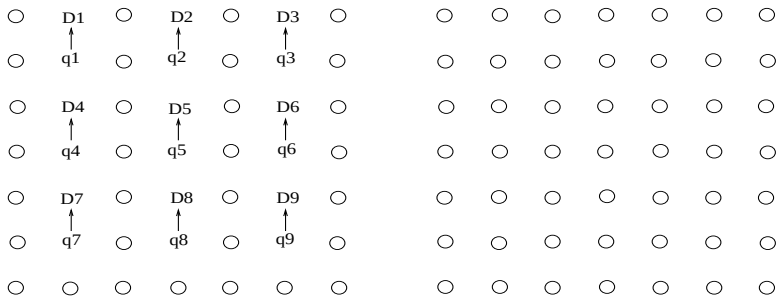
Time step 12:



Time step 13:



Time step 14:



Time step 15:

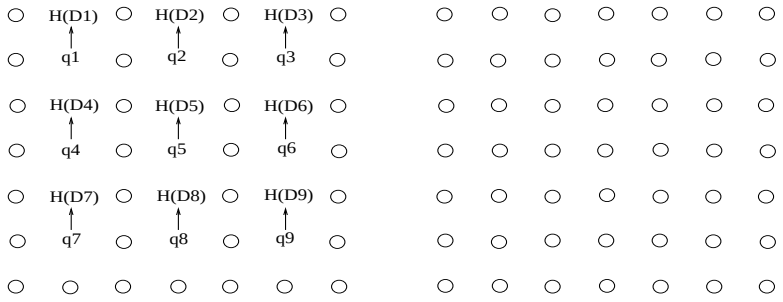


FIGURE A.6: Encoded T gate for the Bacon Shor code

Appendix B

Tile operations for CTL gates using the Steane code

The latencies of CTL gates encoded with Steane code has been summarized in Table B.1.

TABLE B.1: Latency of CTL gates encoded with the Bacon Shor code

Gate	Latency
X	1
Y	1
Z	1
H	1
S	2
T	14
hSWAP	9
vSWAP	7
vCNOT	11
hCNOT	9

B.1 Pauli Gates

Time step 1:

(a) Encoded X gate (b) Encoded Y gate (c) Encoded Z gate

FIGURE B.1: Encoded Pauli Gates for the Steane code

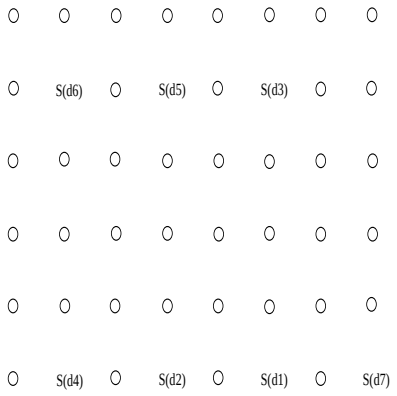
B.2 H

Time step 1:

FIGURE B.2: Encoded H gate for the Steane code

B.3 S

Time step 1:



Time step 2:

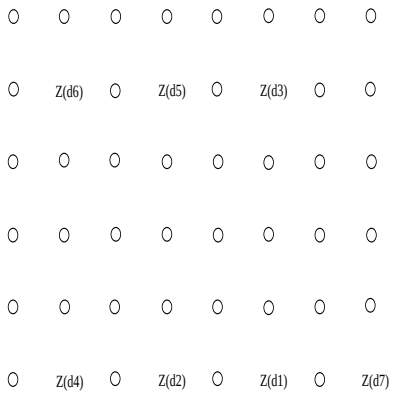
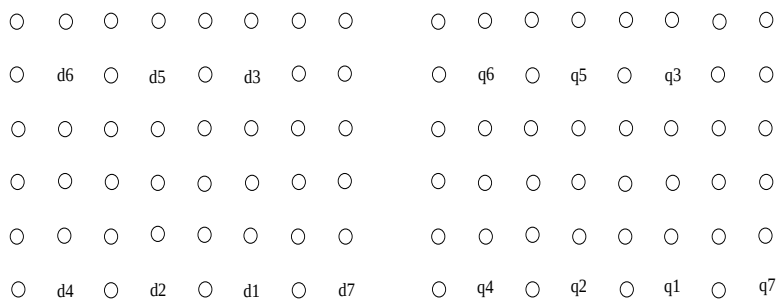


FIGURE B.3: Encoded S gate for the Steane code

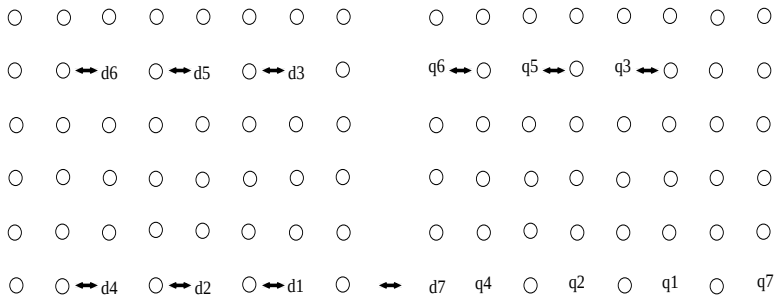
B.4 hSWAP

An encoded horizontal SWAP (hSWAP) operation between $|d_1 d_2 \dots d_7\rangle$ and $|q_1 q_2 \dots q_7\rangle$.

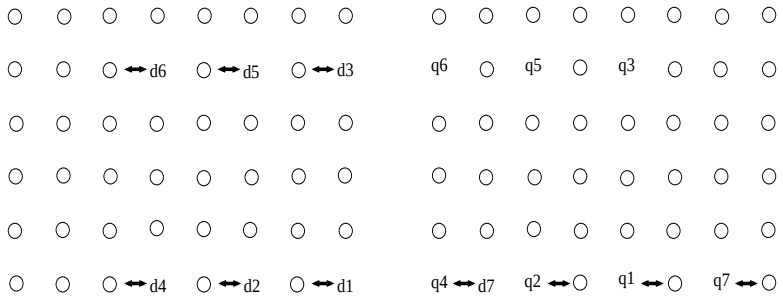
Time step 0:



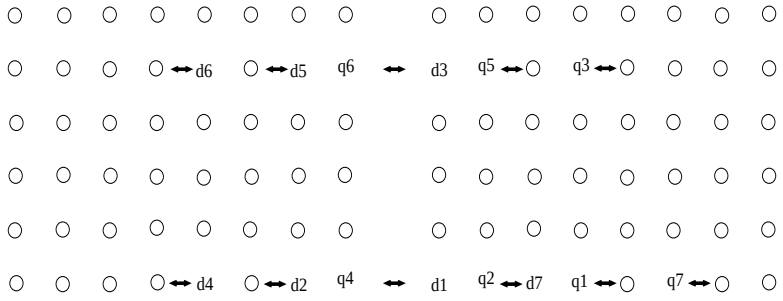
Time step 1:



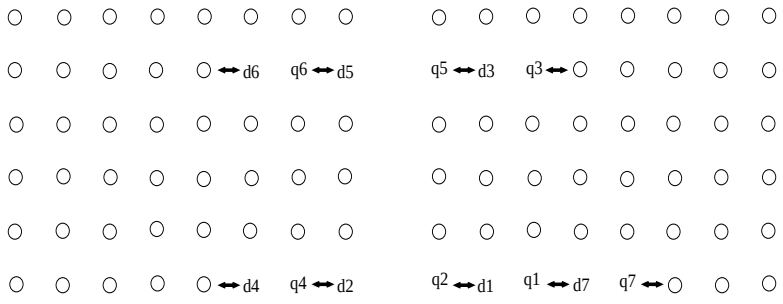
Time step 2:



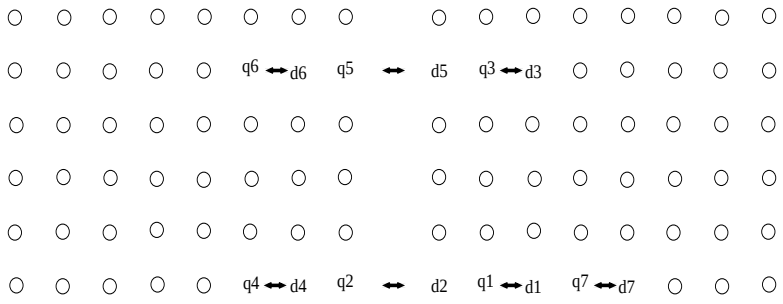
Time step 3:



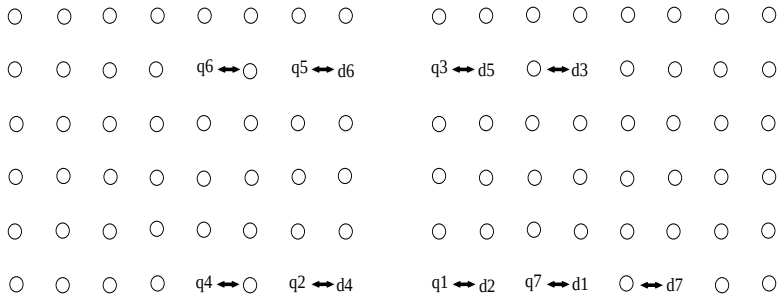
Time step 4:



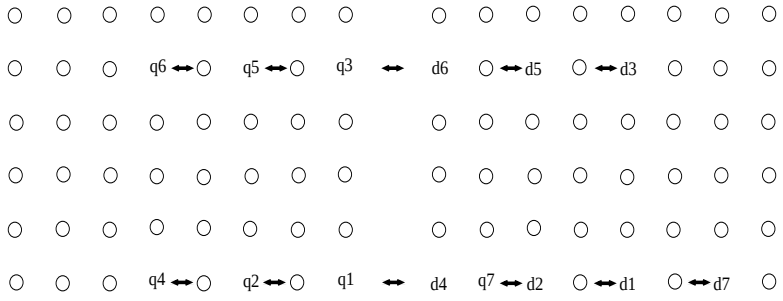
Time step 5:



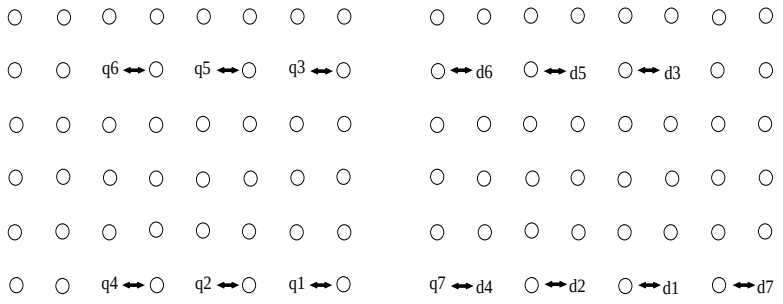
Time step 6:



Time step 7:



Time step 8:



Time step 9:

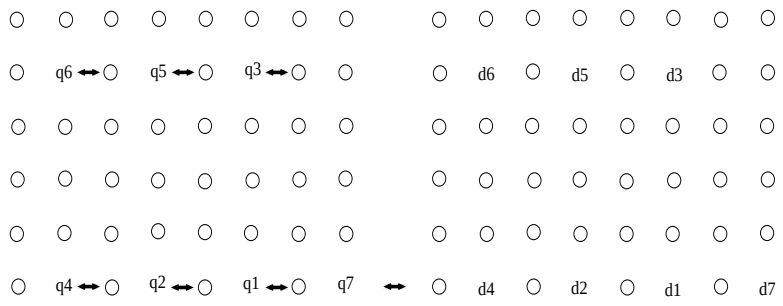
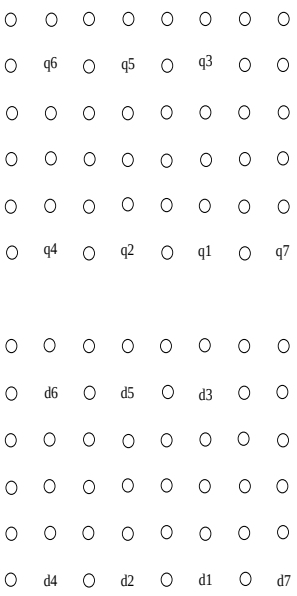


FIGURE B.4: Encoded hSWAP gate for the Steane code

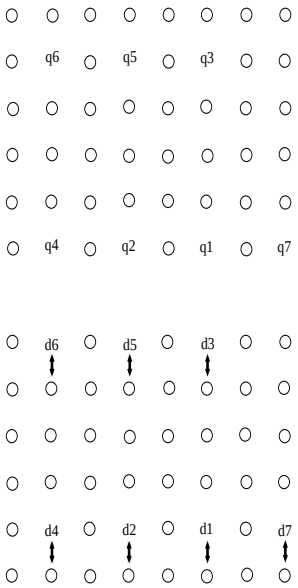
B.5 vSWAP

An encoded vertical SWAP (vSWAP) operation between $|d_1 d_2 \dots d_7\rangle$ and $|q_1 q_2 \dots q_7\rangle$.

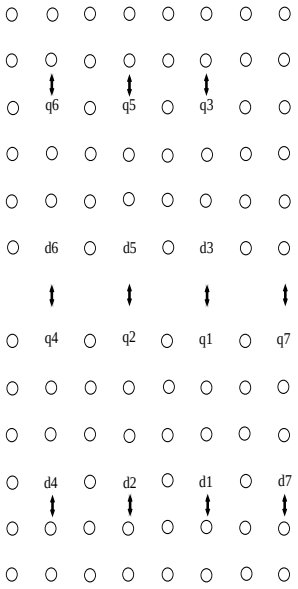
Time step 0:



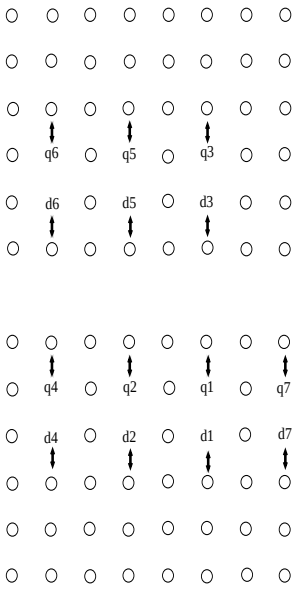
Time step 1:



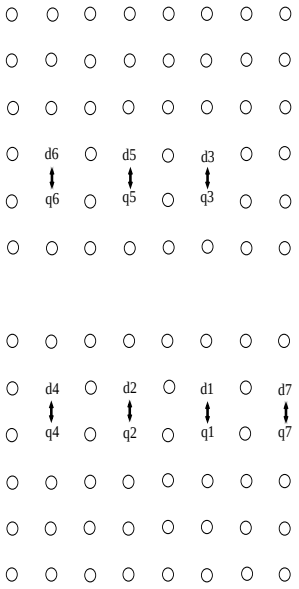
Time step 2:



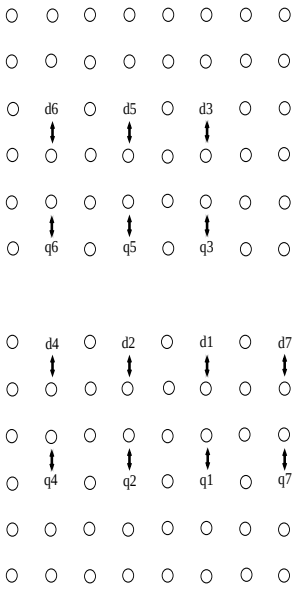
Time step 3:



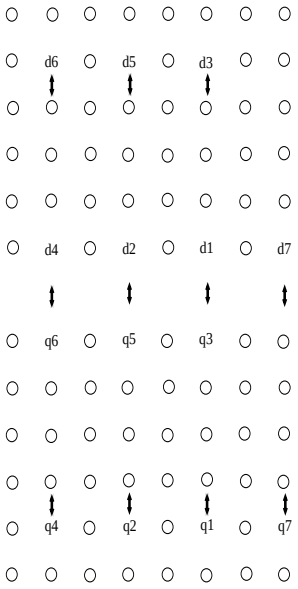
Time step 4:



Time step 5:



Time step 6:



Time step 7:

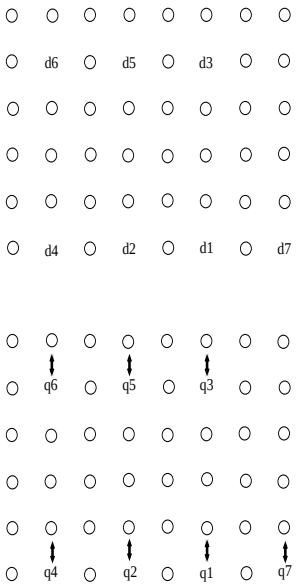
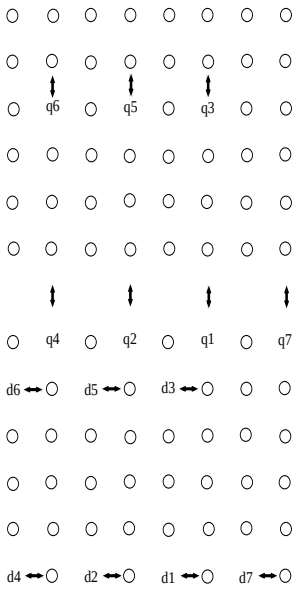


FIGURE B.5: Encoded vSWAP gate for the Steane code

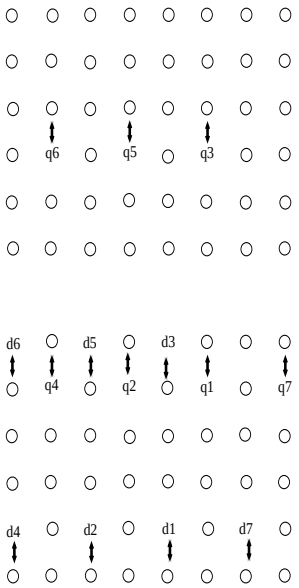
B.6 hCNOT

An encoded horizontal CNOT (hCNOT) operation between $|d_1 d_2 \dots d_7\rangle$ (control) and $|q_1 q_2 \dots q_7\rangle$ (target).

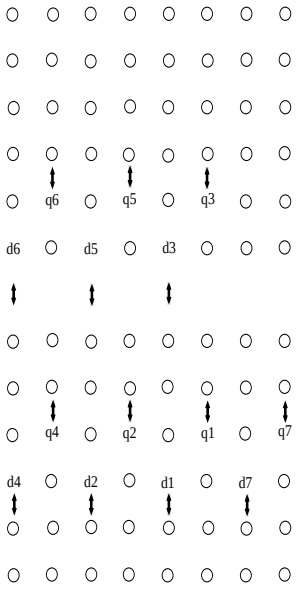
Time step 1:



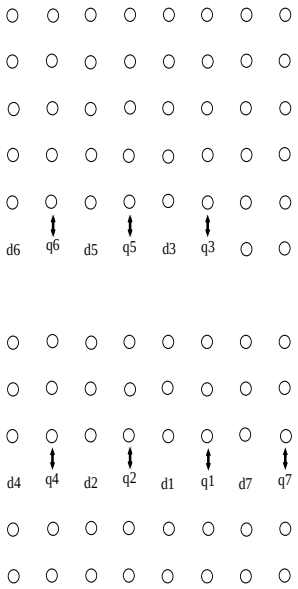
Time step 2:



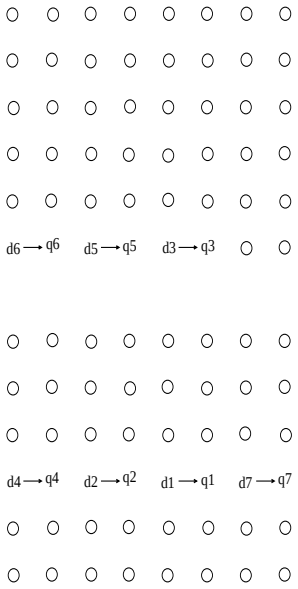
Time step 3:



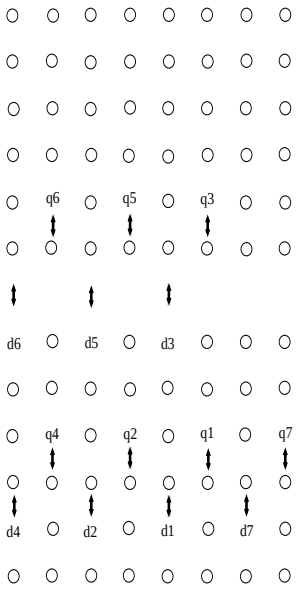
Time step 4:



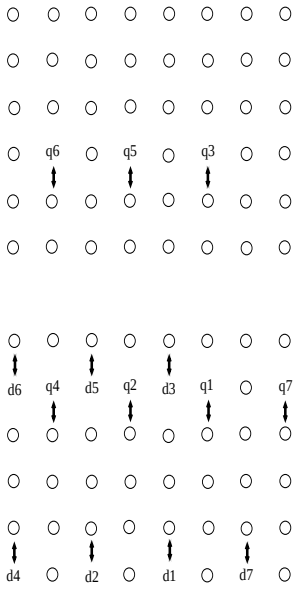
Time step 5:



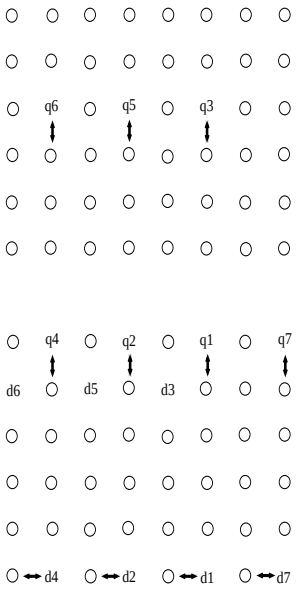
Time step 6:



Time step 7:



Time step 8:



Time step 9:

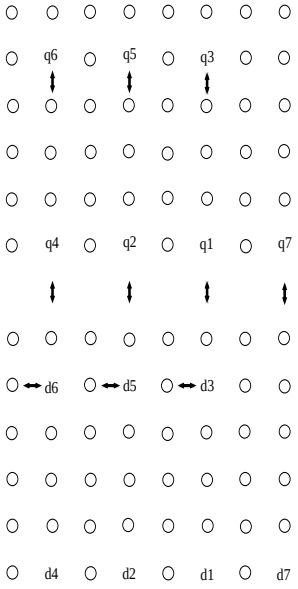
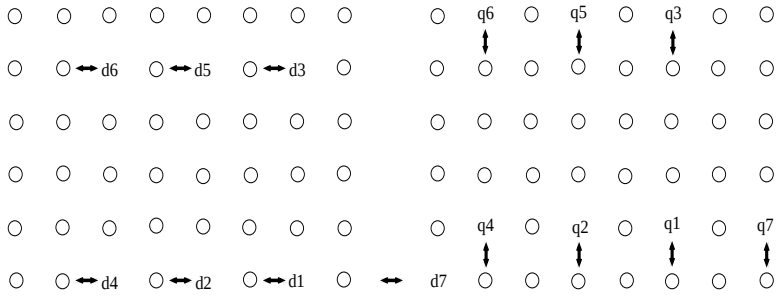


FIGURE B.6: Encoded hCNOT gate for the Steane code

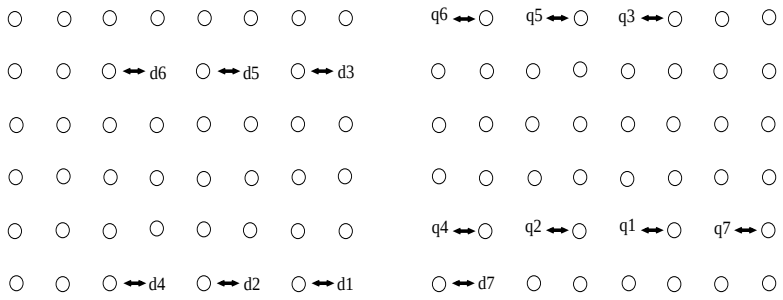
B.7 vCNOT

An encoded vertical CNOT (vCNOT) operation between $|d_1 d_2 \dots d_7\rangle$ (control) and $|q_1 q_2 \dots q_7\rangle$ (target).

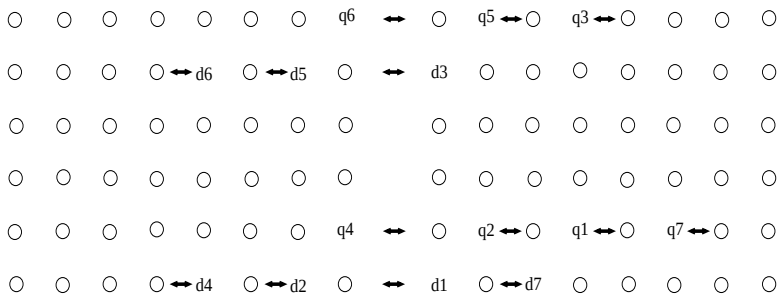
Time step 1:



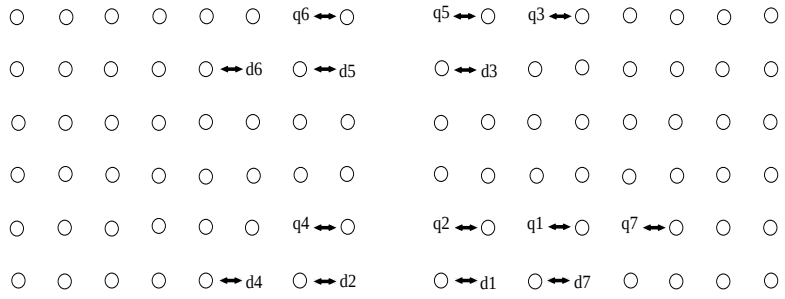
Time step 2:



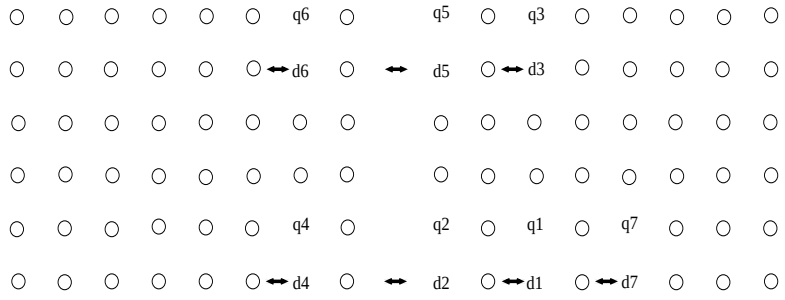
Time step 3:



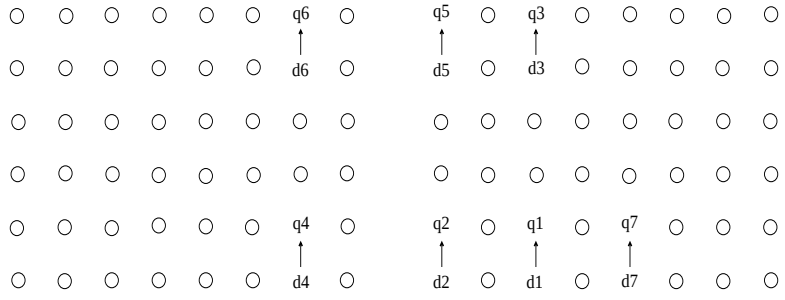
Time step 4:



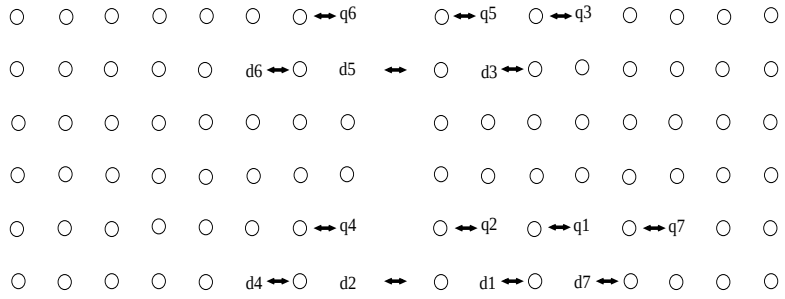
Time step 5:



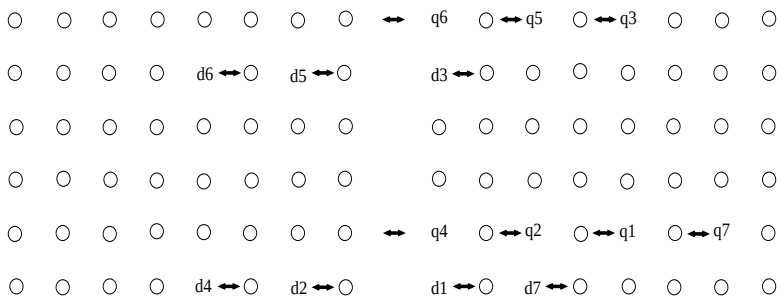
Time step 6:



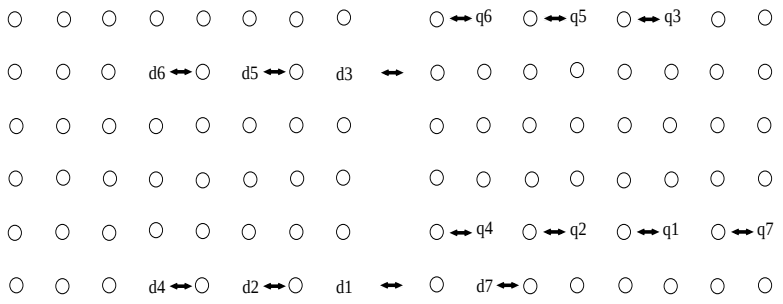
Time step 7:



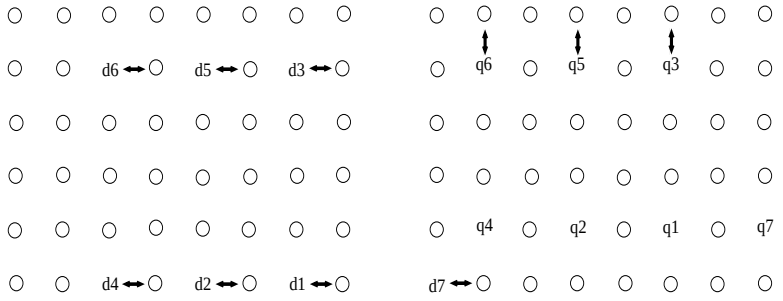
Time step 8:



Time step 9:



Time step 10:



Time step 11:

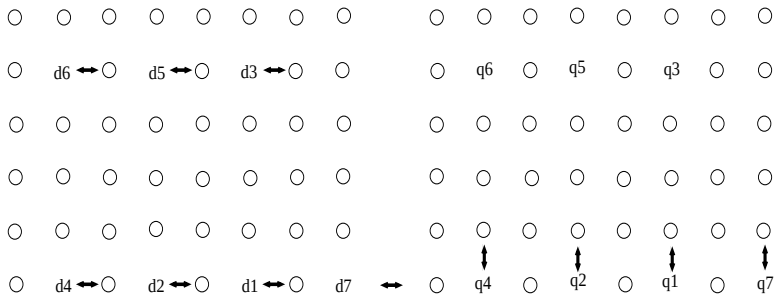
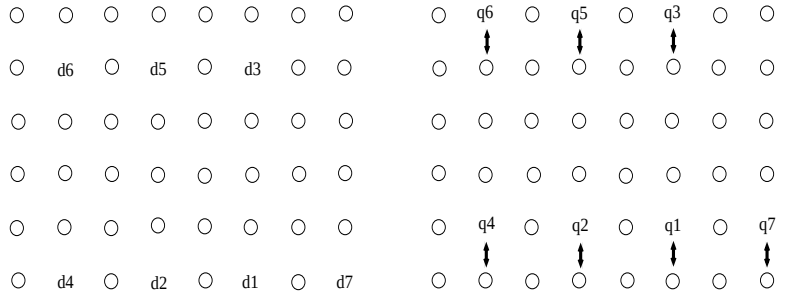


FIGURE B.7: Encoded vCNOT gate for the Steane code

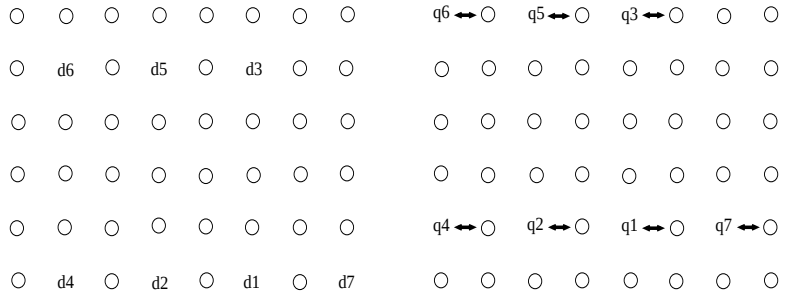
B.8 T

An encoded T operation on $|d_1 d_2 \dots d_7\rangle$. We assume $|q_1 q_2 \dots q_7\rangle$ is in $T\,|\overline{+}\rangle$ state.

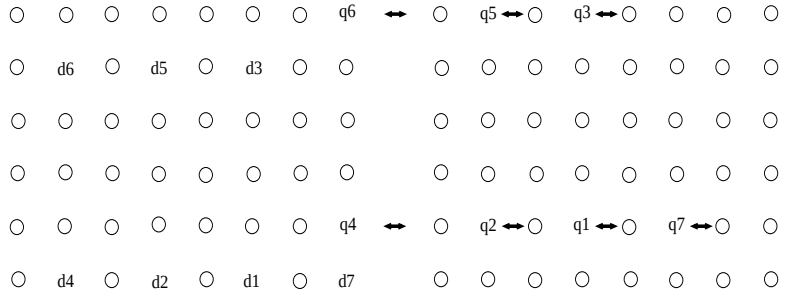
Time step 1:



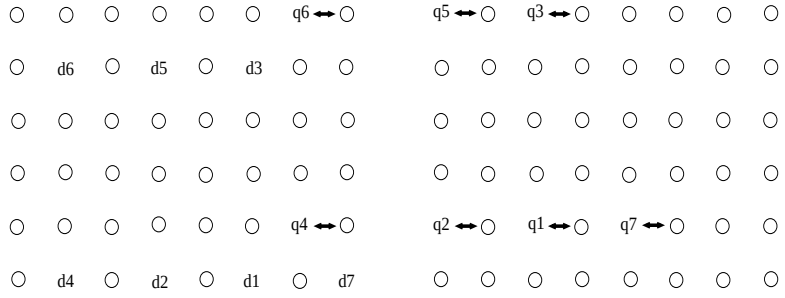
Time step 2:



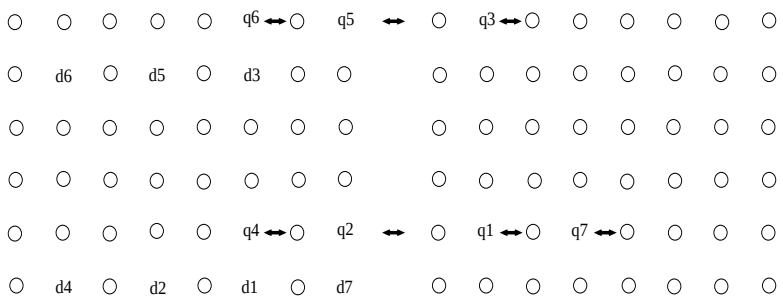
Time step 3:



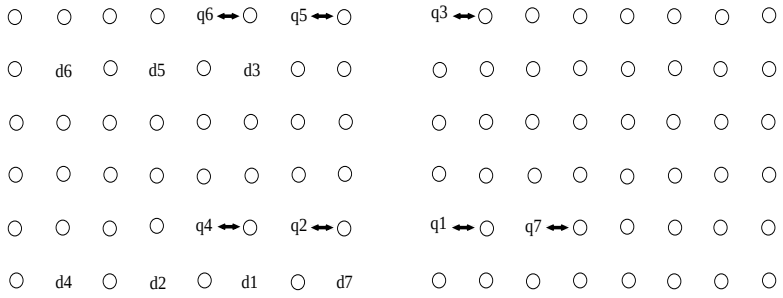
Time step 4:



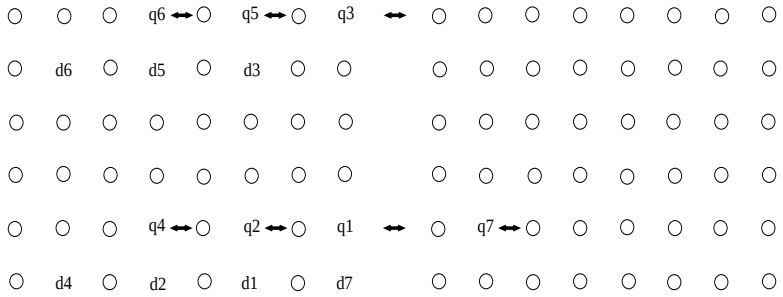
Time step 5:



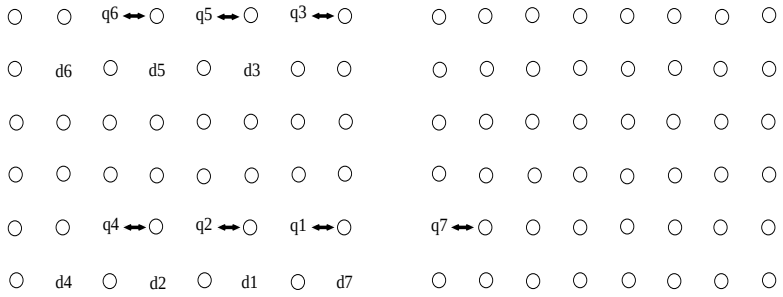
Time step 6:



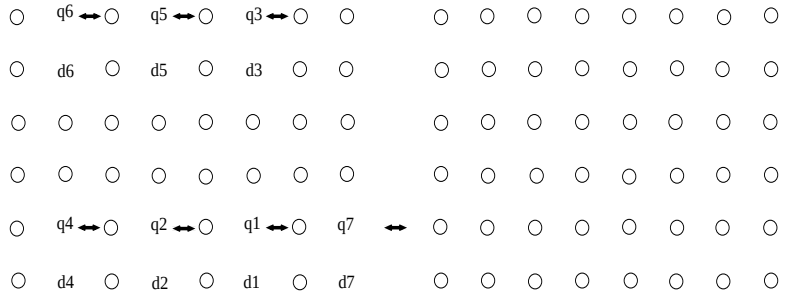
Time step 7:



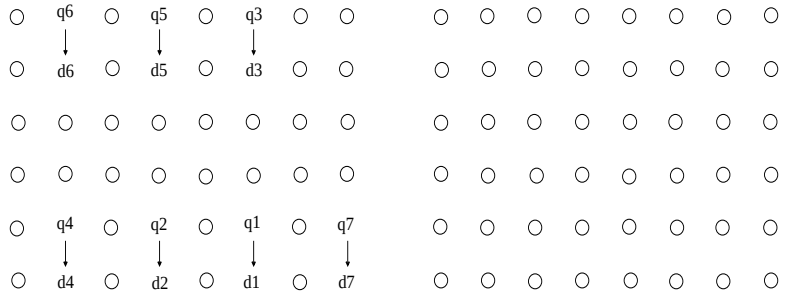
Time step 8:



Time step 9:

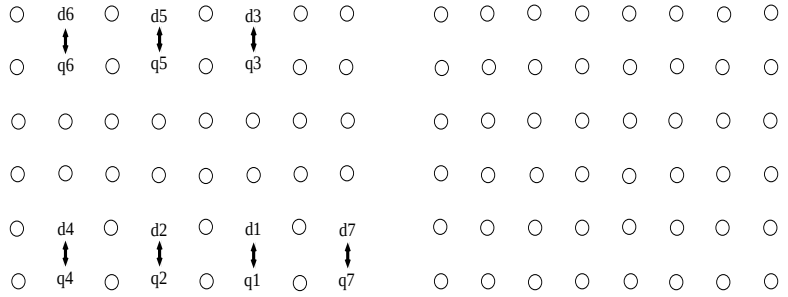


Time step 10:

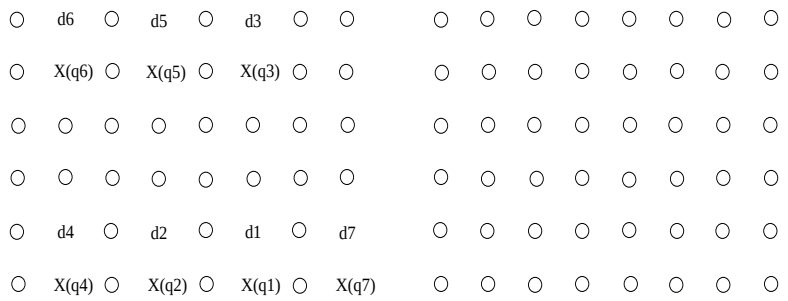


Time step 11:

Time step 12:



Time step 13:



Time step 14:

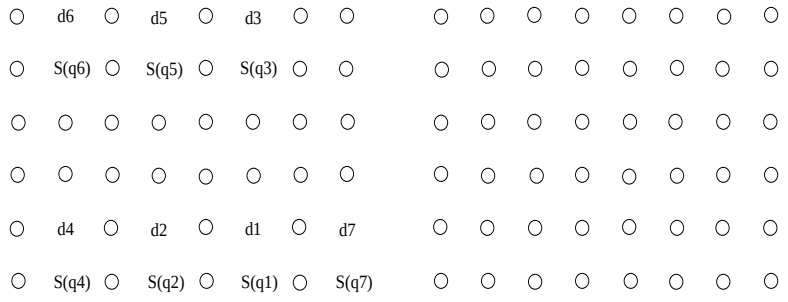


FIGURE B.8: Encoded T gate for the Steane code

Appendix C

Tile operations for CTL gates using the Knill code

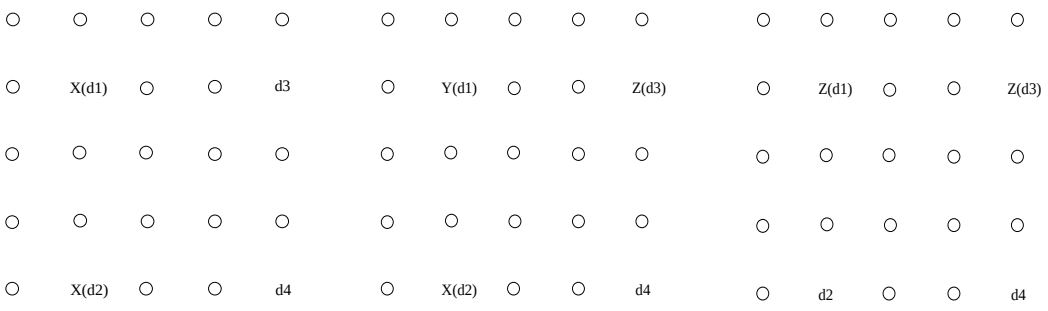
The latencies of CTL gates encoded with Knill code has been summarized in Table C.1.

TABLE C.1: Latency of CTL gates encoded with the Bacon Shor code

Gate	Latency
X	1
Y	1
Z	1
H	1
S	10
T	13
SWAP	7
CNOT	7

C.1 Pauli Gates

Time step 1:



(a) Encoded X gate (b) Encoded Y gate (c) Encoded Z gate

FIGURE C.1: Encoded Pauli Gates for the Knill code

C.2 H

Time step 1:

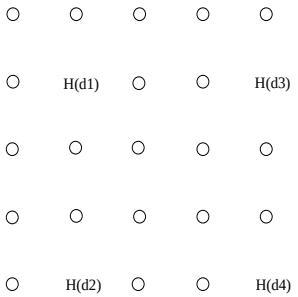
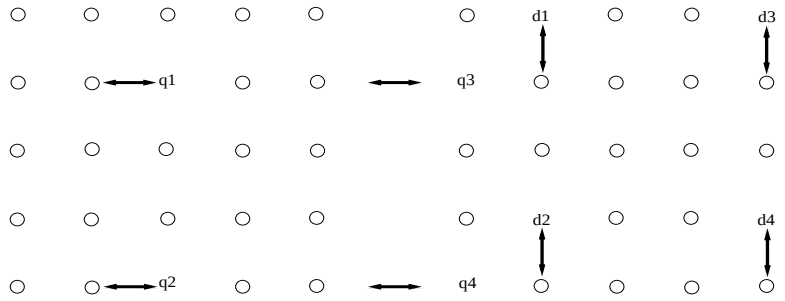


FIGURE C.2: Encoded H gate for the Knill code

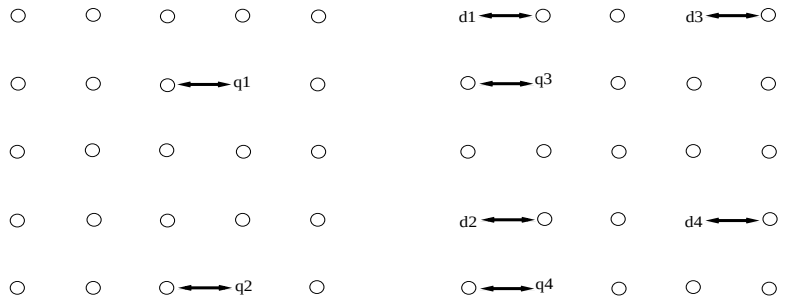
C.3 SWAP

An encoded SWAP operation between $|d_1 d_2 \dots d_9\rangle$ and $|q_1 q_2 \dots q_9\rangle$.

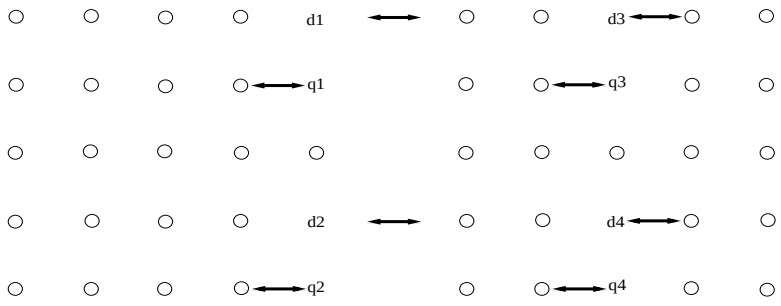
Time step 1:



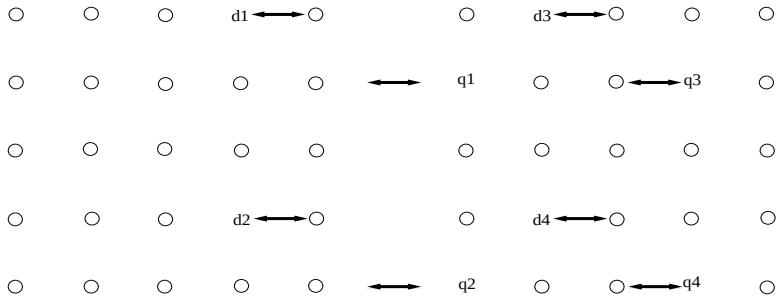
Time step 2:



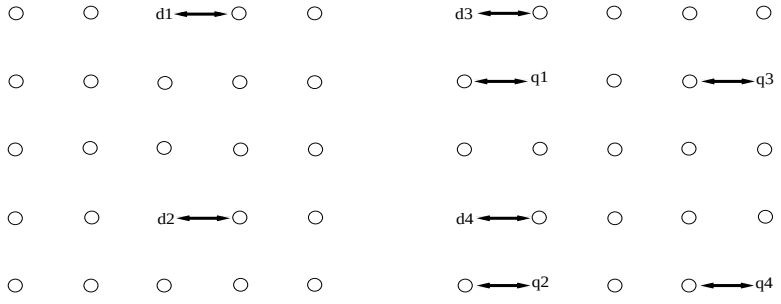
Time step 3:



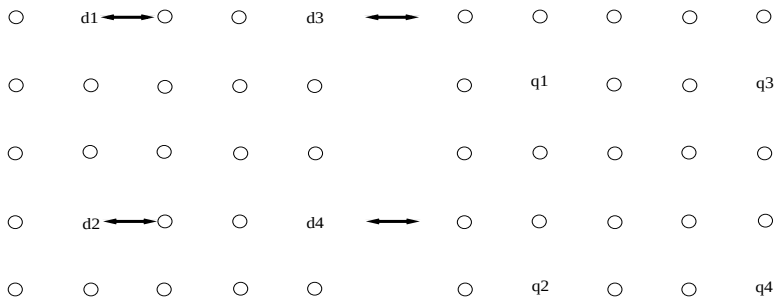
Time step 4:



Time step 5:



Time step 6:



Time step 7:

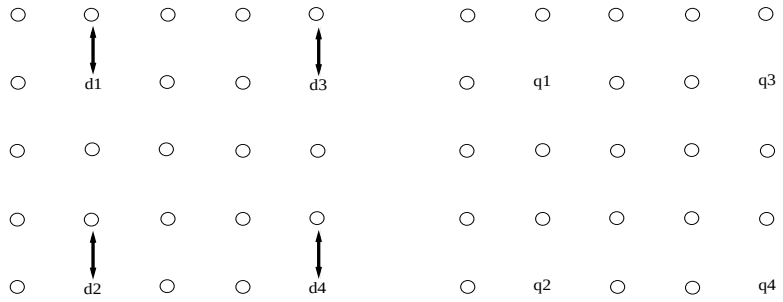
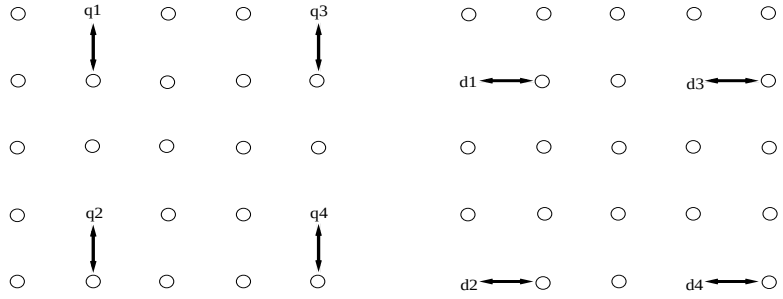


FIGURE C.3: Encoded SWAP gate for the Bacon Shor code

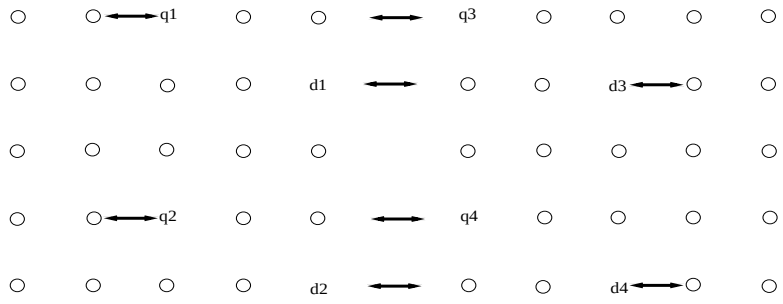
C.4 CNOT

An encoded CNOT operation between $|d_1 d_2 \dots d_9\rangle$ (target) and $|q_1 q_2 \dots q_9\rangle$ (control)

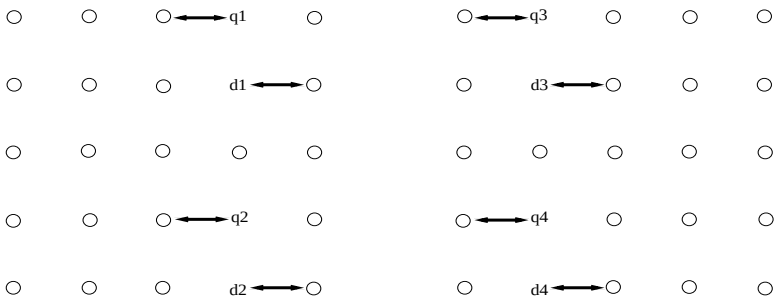
Time step 1:



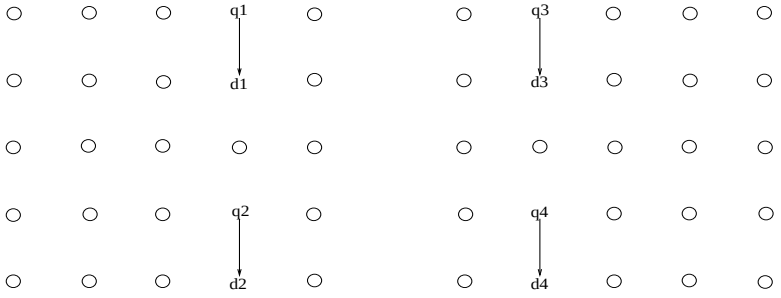
Time step 2:



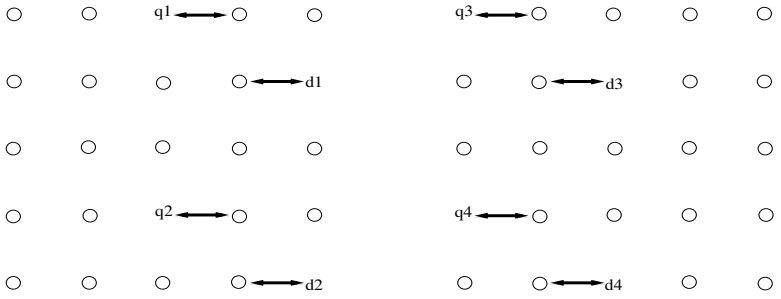
Time step 3:



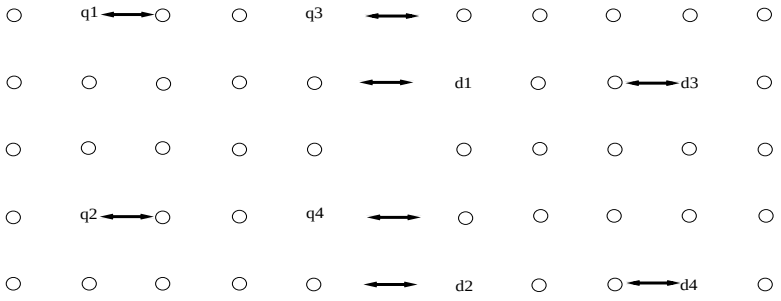
Time step 4:



Time step 5:



Time step 6:



Time step 7:

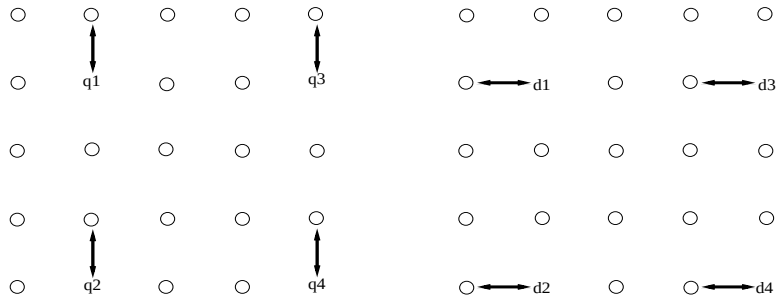
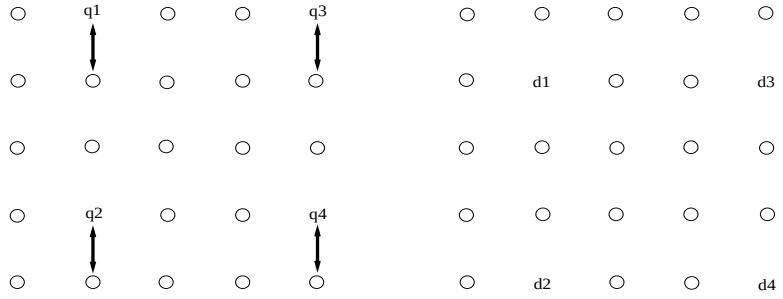


FIGURE C.4: Encoded CNOT gate for the Steane code

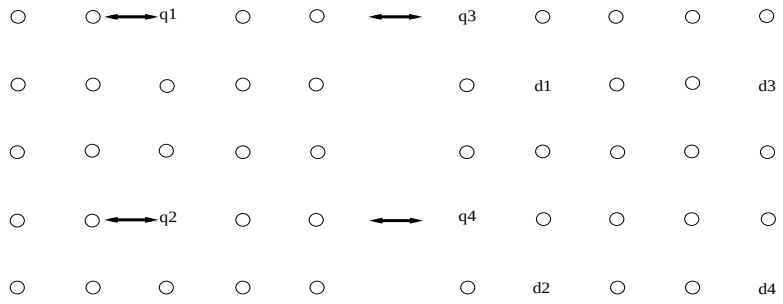
C.5 S

An encoded S operation on $|d_1 d_2 \dots d_9\rangle$. We assume $|q_1 q_2 \dots q_9\rangle$ is in $\left|\overline{+i}\right\rangle$ state.

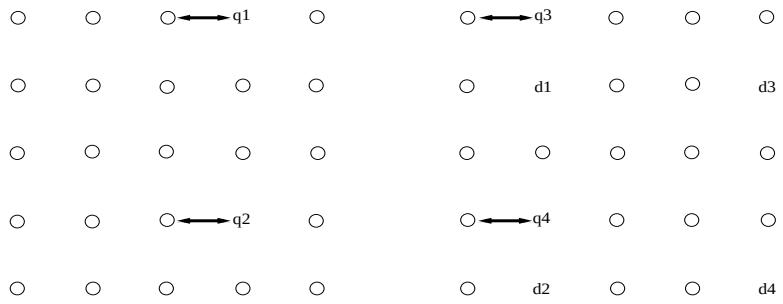
Time step 1:



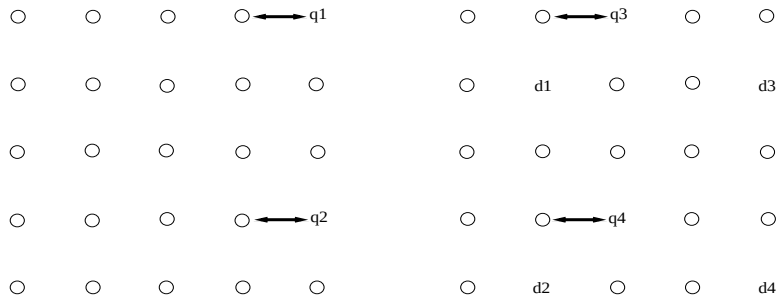
Time step 2:



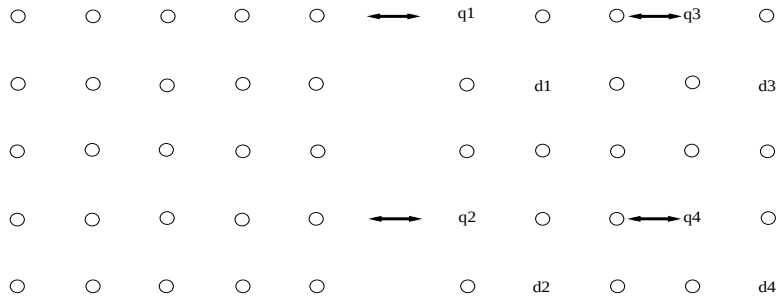
Time step 3:



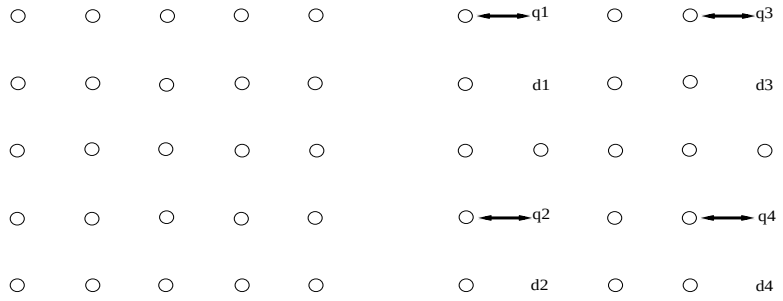
Time step 4:



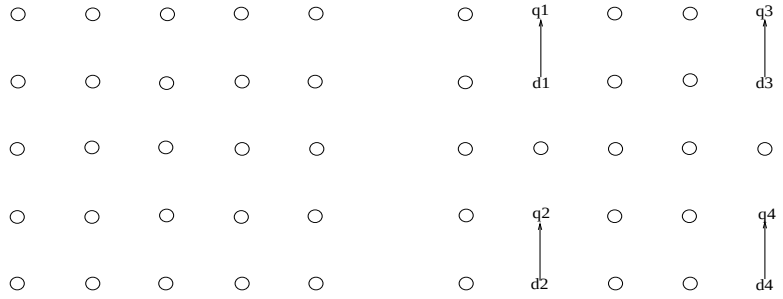
Time step 5:



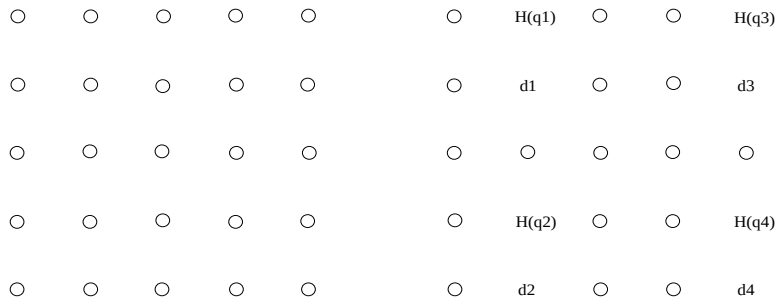
Time step 6:



Time step 7:



Time step 8:



Time step 9:

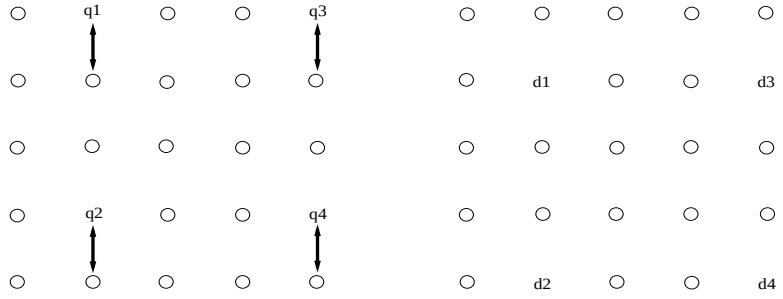
Time step 10:

FIGURE C.5: Encoded S gate for the Steane code

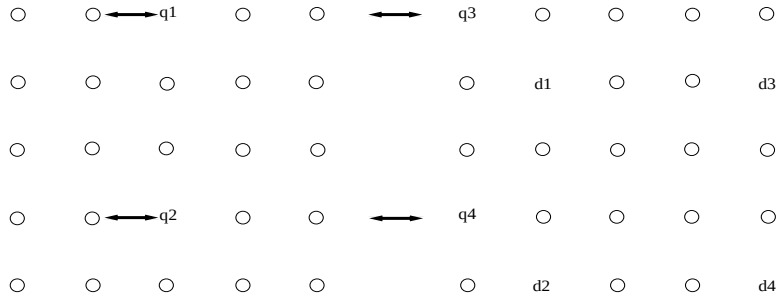
C.6 T

An encoded T operation on $|d_1 d_2 \dots d_9\rangle$. We assume $|q_1 q_2 \dots q_9\rangle$ is in $T\,|\overline{+}\rangle$ state.

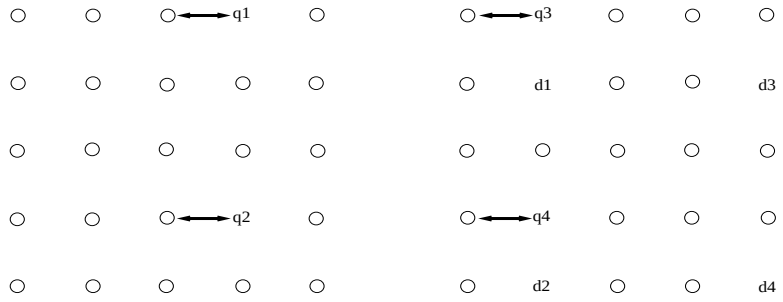
Time step 1:



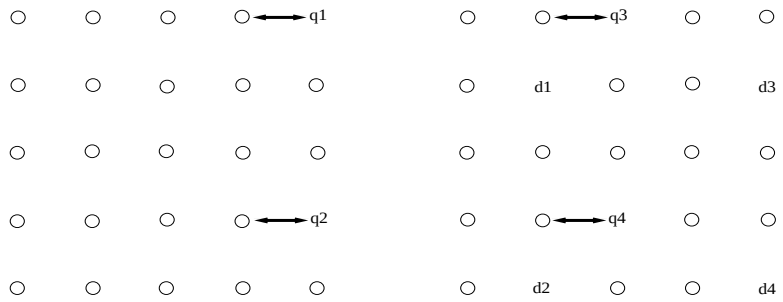
Time step 2:



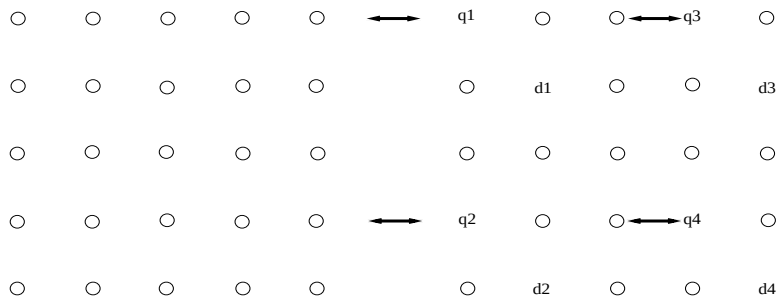
Time step 3:



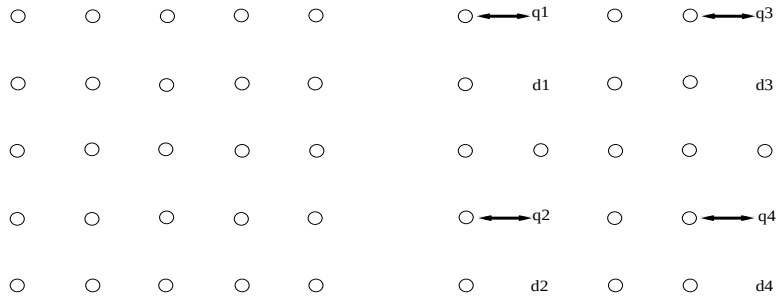
Time step 4:



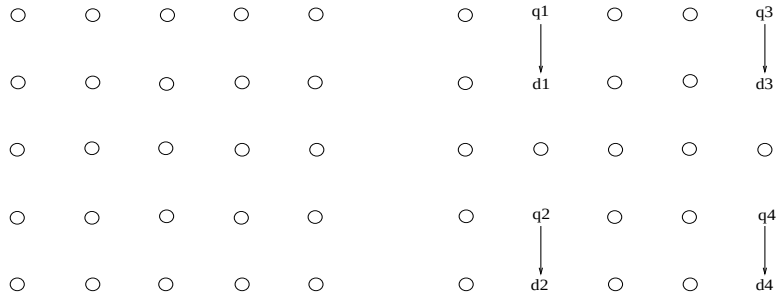
Time step 5:



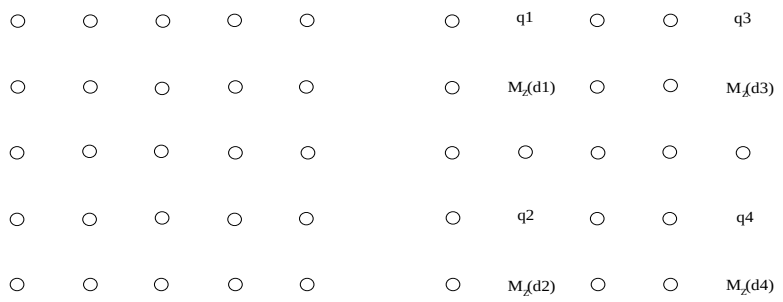
Time step 6:



Time step 7:



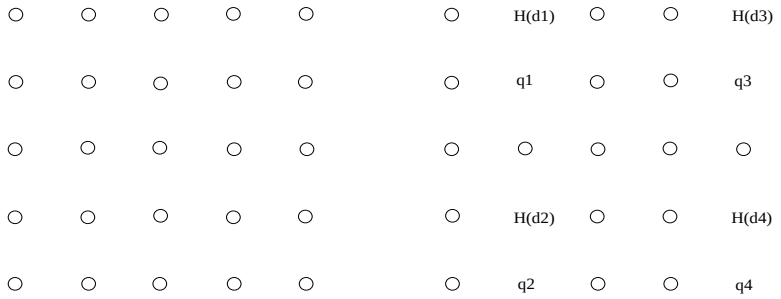
Time step 8:



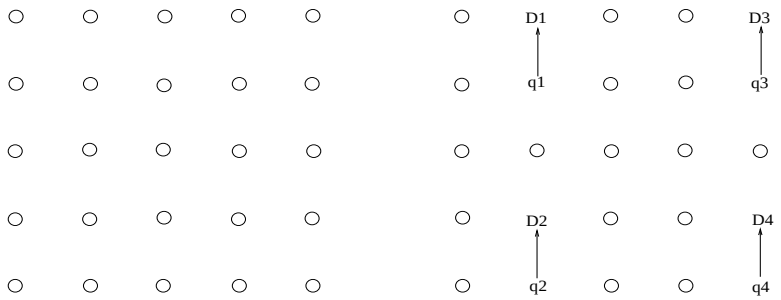
Time step 9:

Time step 10:

Time step 11:



Time step 12:



Time step 13:

FIGURE C.6: Encoded T gate for the Knill code

Bibliography

[1] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem.* Proceedings of the London mathematical society 42, no. 2, pp 230-265, 1936.

[2] J. Von Neumann, *Probabilistic logics and the synthesis of reliable organisms from unreliable components.*, Automata studies 34, pp 43-98, 1956.

[3] R. P. Feynman, *Simulating physics with computers.*, International Journal of Theoretical Physics 21, no. 6, pp 467-488, 1982.

[4] W. K. Wootters, W. H. Zurek, *A single quantum cannot be cloned.*, Nature 299, no. 5886, pp 802-803, 1982.

[5] P.W. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp 124-134, IEEE, 1994.

[6] I. Chuang, R. Laflamme, P. Shor, W. Zurek, *Quantum computers, factoring, and decoherence.*, arXiv preprint quant-ph/9503007, 1995.

[7] R. Landauer, *Is quantum mechanics useful?.*, Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences 353, no. 1703, 367-376, 1995.

[8] P. W. Shor, *Scheme for reducing decoherence in quantum computer memory.*, Physical review A 52, no. 4,pp R2493-R2496, 1995.

[9] W. G. Unruh, *Maintaining coherence in quantum computers.*, Physical Review A 51, no. 2: 992, 1995.

[10] R. A. Calderbank, P. W. Shor, *Good quantum error-correcting codes exist.*, Physical Review A 54, no. 2: 1098, 1996.

[11] L. K. Grover, *A fast quantum mechanical algorithm for database search.*, Proceedings of the 28^{th} annual ACM symposium on Theory of computing, pp. 212-219. ACM, 1996.

[12] E. Knill, R. Laflamme, *Concatenated quantum codes.*, arXiv preprint quant-ph/9608012, 1996.

[13] A. M. Steane, *Error correcting codes in quantum theory.*, Physical Review Letters 77, no. 5, pp 793-797, 1996.

[14] A. Steane, *Multiple-particle interference and quantum error correction.*, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 452, no. 1954, pp 2551-2577, 1996.

[15] P. W. Shor, *Fault-tolerant quantum computation.*, In Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, pp. 56-65. IEEE, 1996.

[16] V. Vedral, A. Barenco, A. Ekert, *Quantum networks for elementary arithmetic operations.*, Physical Review A 54, no. 1: 147, 1996.

[17] D. Aharonov, M. Ben-Or, *Fault-tolerant quantum computation with constant error.*, In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pp. 176-188. ACM, 1997.

[18] C. H. Bennett, E. Bernstein, G. Brassard, U. Vazirani. *Strengths and weaknesses of quantum computing.*, SIAM Journal on Computing 26, no. 5, pp 1510-1523, 1997.

[19] D. Gottesman, *Stabilizator codes and quantum error correction.*, PhD diss., PhD thesis, CIT, quant-ph/9705052, 1997.

[20] A. Y. Kitaev, *Quantum computations: algorithms and error correction.*, Russian Mathematical Surveys 52, no. 6 (1997): 1191-1249.

[21] P.W. Shor, *Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal on Computing, 26(5):1484-1509, 1997.

[22] E. Knill, R. Laflamme, W. H. Zurek, *Resilient quantum computation: error models and thresholds.*, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 454, no. 1969, pp 365-384, 1998.

[23] A. M. Steane, *Efficient fault-tolerant quantum computing.*, Nature 399, no. 6732, pp 124-126, 1999.

[24] E. Knill, R. Laflamme, L. Viola, *Theory of quantum error correction for general noise.*, Physical Review Letters 84, no. 11, pp 2525-2528, 2000.

[25] E. Knill, R. Laflamme, G. J. Milburn, *A scheme for efficient quantum computation with linear optics.*, nature 409, no. 6816, pp 46-52, 2001.

[26] C. Monroe, C. A. Sackett, D. Kielpinski, B. E. King, C. Langer, V. Meyer, C. J. Myatt, M. Rowe, Q. A. Turchette, W. M. Itano, D. J. Wineland, Scalable entanglement of trapped ions, AIP Conference Proceedings, vol 551, 2001.

[27] S. Beauregard, *Circuit for Shor's algorithm using 2n+ 3 qubits.*, arXiv preprint quant-ph/0205095, 2002.

[28] D. Leibfried, B. DeMarco, V. Meyer, D. Lucas, M. Barrett, J. Britton, B. Jelenkovicacute WM Itano, C. Langer, and D. J. T Rosenband, *Experimental demonstration of a robust, high-fidelity geometric two ion-qubit phase gate.*, Nature 422, no. 6930, pp 412-415, 2003.

[29] C. liwa, K. Banaszek, *Conditional preparation of maximal polarization entanglement.*, Physical Review A 67, no. 3 : 030101, 2003.

[30] F. W. Strauch, P. R. Johnson, A. J. Dragt, C. J. Lobb, J. R. Anderson, F. C. Wellstood, *Quantum logic gates for coupled superconducting phase qubits.*, Physical review letters 91, no. 16: 167005, 2003.

[31] S. A. Cuccaro, T. G. Draper, S. A. Kutin, D. P. Moulton, *A new quantum ripple-carry addition circuit.*, arXiv preprint quant-ph/0410184, 2004.

[32] J. D. Franson, B. C. Jacobs, T. B. Pittman, *Quantum computing using single photons and the Zeno effect.*, Physical Review A 70, no. 6: 062302, 2004.

[33] P. Aliferis, D. Gottesman, J. Preskill, *Quantum accuracy threshold for concatenated distance-3 codes.*, arXiv preprint quant-ph/0504218, 2005.

[34] M. D. Barrett, T. Schaetz, J. Chiaverini, D. Leibfried, J. Britton, W. M. Itano, J. D. Jost et al, *Quantum information processing with trapped ions.*, In AIP Conference Proceedings, vol. 770, p. 350. 2005.

[35] R. McDermott, R. W. Simmonds, M. Steffen, K. B. Cooper, K. Cicak, K. D. Osborn, S. Oh, D. P. Pappas, John M. Martinis, *Simultaneous state measurement of coupled Josephson phase qubits.*, Science 307, no. 5713 : 1299-1302, 2005.

[36] C. M. Dawson, M. A. Nielsen, *The Solovay-Kitaev Algorithm*, arXiv preprint quant-ph/0505030, 2005.

[37] E. Knill, *Quantum computing with realistically noisy devices.*, Nature 434, no. 7029, pp 39-44, 2005.

[38] W. J. Munro, K. Nemoto, T. P. Spiller, S. D. Barrett, P. Kok, R. G. Beausoleil, *Efficient optical quantum information processing.*, Journal of Optics B: Quantum and Semiclassical Optics 7, no. 7: S135, 2005.

[39] M. Saffman, T. G. Walker, *Analysis of a quantum logic device based on dipole-dipole interactions of optically trapped Rydberg atoms.*, Physical Review A 72, no. 2: 022347, 2005.

[40] D. Bacon, A. Casaccino, *Quantum error correcting subsystem codes from two classical linear codes.*, arXiv preprint quant-ph/0610088, 2006.

[41] K. M. Svore, D. P. DiVincenzo, B. M. Terhal, *Noise threshold for a fault-tolerant two-dimensional lattice architecture.*, arXiv preprint quant-ph/0604090, 2006.

[42] P. M. Leung, T. C. Ralph, *Optical zeno gate: bounds for fault tolerant operation.*, New Journal of Physics 9, no. 7: 224, 2007.

[43] J. M. Taylor, J. R. Petta, A. C. Johnson, A. Yacoby, C. M. Marcus, M. D. Lukin. *Relaxation, dephasing, and quantum control of electron spins in double quantum dots.* Physical Review B 76, no. 3: 035315, 2007.

[44] X. Zou, S-L Zhang, K. Li, G. Guo, *Linear optical implementation of the two-qubit controlled phase gate with conventional photon detectors.*, Physical Review A 75, no. 3: 034302, 2007.

[45] D. Aharonov, M. Ben-Or, *Fault-tolerant quantum computation with constant error rate.*, SIAM Journal on Computing 38, no. 4, pp 1207-1282, 2008.

[46] F. M. Spedalieri, V. P. Roychowdhury, *Latency in local, two-dimensional, fault-tolerant quantum computing.*, arXiv preprint arXiv:0805.4213, 2008.

[47] L. DiCarlo, J. M. Chow, J. M. Gambetta, L. S. Bishop, B. R. Johnson, D. I. Schuster, J. Majer et al, *Demonstration of two-qubit algorithms with a superconducting quantum processor.*, Nature 460, no. 7252 : 240-244, 2009.

[48] F. Motzoi, J. M. Gambetta, P. Rebentrost, F. K. Wilhelm, *Simple pulses for elimination of leakage in weakly nonlinear qubits.*, Physical review letters 103, no. 11: 110501, 2009.

[49] M. Whitney, *Practical fault tolerance for quantum circuits*, 2009.

[50] R. C. Bialczak, M. Ansmann, M. Hofheinz, E. Lucero, M. Neeley, A. D. OConnell, D. Sank et al, *Quantum process tomography of a universal entangling gate implemented with Josephson phase qubits.*, Nature Physics 6, no. 6, pp 409-413, 2010.

[51] M. A. Nielsen, I. L. Chuang, *Quantum computation and quantum information*, Cambridge university press, 2010.

[52] M. Saffman, T. G. Walker, K. Mlmer, *Quantum information with Rydberg atoms.*, Reviews of modern Physics 82, no. 3: 2313, 2010.

[53] T. Yamamoto, M. Neeley, E. Lucero, R. C. Bialczak, J. Kelly, M. Lenander, M. Mariantoni et al, *Quantum process tomography of two-qubit controlled-Z and controlled-NOT gates using superconducting phase qubits.*, Physical Review B 82, no. 18 : 184515, 2010.

[54] M. W. Johnson,, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris et al. *Quantum annealing with manufactured spins.*, Nature 473, no. 7346, pp 194-198, 2011.

[55] C. Weitenberg, M. Endres, J. F. Sherson, M. Cheneau, P. Schau, T. Fukuhara, I. Bloch, S. Kuhr, *Single-spin addressing in an atomic Mott insulator.*, Nature 471, no. 7338, pp 319-324, 2011.

[56] J. Booth, *Quantum Compiler Optimization*, arXiv preprint arXiv:1206.3348. 2012.

[57] V. Kliuchnikov, D. Maslov, M. Mosca, *Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and T gates.*, arXiv preprint arXiv:1206.5236, 2012.

[58] C-C. Lin, A. Chakrabarti, N. K. Jha. *Optimized Quantum Gate Library for Various Physical Machine Descriptions.*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2013.

[59] C-Y, Lai, G. Paz, M. Suchara, T. A. Brun, *Performance and Error Analysis of Knill's Postselection Scheme in a Two-Dimensional Architecture.*, arXiv preprint arXiv:1305.5657, 2013.

[60] C-C. Lin, A. Chakrabarti, N. K. Jha, *FTQLS: Fault-tolerant Quantum Logic Synthesis.*, manuscript.

[61] M. Suchara, A. Faruque, C-Y. Lai, G. Paz, F. T. Chong, J. Kubiatowicz, *Estimating the resources for quantum computation with the QuRE toolbox*, manuscript.

[62] C-C. Lin, A. Chakrabarti, N. K. Jha, *Qlib: Quantum Module Library.*, manuscript.