UNIVERSITY ”POLITEHNICA” TIMIŞOARA
FACULTY OF AUTOMATION AND COMPUTERS

PhD Report No. 3

Coordinator: Prof. univ. dr. ing. Mircea Vlăduţiu

Author: Mihai Udrescu-Milosav

2004

A Hardware Engineering View on Dependable Quantum Computation: Reliability Improvement and Simulated Fault Injection

Contents

1 Introduction **11**

1.1 Report goals 12

1.2 Report outline 13

2 Fault Tolerance in Quantum Computation **15**

2.1 Introduction 15

2.2 Specific problems 16

2.2.1 Error model 16

2.2.2 Quantum constraints and problems 17

2.3 Strategies for quantum fault tolerance 19

2.4 Quantum error correction 22

2.4.1 Steane coding 22

2.4.2 Ancilla coding 26

2.4.3 Putting it all together 28

2.5 Stabilizer codes 29

2.6 Gates for fault-tolerant codes 40

2.6.1 Gottesman's fault tolerant gates 43

2.7 Accuracy threshold 44

2.7.1 Technological requirements 44

2.7.2 Concatenated coding 46

2.7.3 Analysis 47

3 Improving Fault Tolerance in Quantum Circuits **49**

3.1 The big picture 49

3.2 Issues to be settled 51

3.3 Sketching the solution 55

3.3.1 Limitations for rQHW 56

3.3.2 Reconfigurable quantum gate array (rQGA) structure 57

3.3.3 Apropriate gates 58

3.3.4 Basic reconfigurable cell 59

3.3.5 Quantum configuration 61

3.4 Conclusions 64

3.4.1 Shor algorithm requirements 65

3.4.2 *rQHW* summary 65

4 Evolvable Quantum Hardware 67
4.1 Introduction . . . 67
4.1.1 Motivation . . . 68
4.2 Quantum Genetic Algorithms . . . 68
4.2.1 Running GAs in a Quantum Computational Environment . . . 68
4.2.2 Mathematical Formalism . . . 70
4.3 A New Approach . . . 71
4.3.1 Computing the Maximum . . . 72
4.3.2 The Oracle . . . 73
4.3.3 Reduced Quantum Genetic Algorithm . . . 75
4.4 Summary . . . 76

5 Fault Injection in Quantum Computation 77
5.1 Fault Injection in Classical Hardware . . . 78
5.1.1 Motivation . . . 78
5.1.2 Simulated Fault Injection Analysis . . . 79
5.2 Sketching the guidlines for the *QUERIST* project . . . 84
5.3 Specific problems in the quantum environment . . . 84
5.3.1 Setup phase . . . 86
5.3.2 Simulation phase . . . 87
5.3.3 Data processing phase . . . 88
5.4 Summary . . . 89

6 Conclusion and Future Work 91
6.1 Report contributions . . . 91
6.2 Future work . . . 92

A Useful configurations 93
A.1 Encoder with classical configuration . . . 93
A.2 Stabilizer code with Steane ancilla . . . 93
A.3 Quantum configuration for the encoding circuit . . . 96

B HDL quantum circuit simulation methodology 99
B.1 The HDL approach . . . 99
B.2 Methodology effectiveness . . . 103
B.2.1 Automated procedure . . . 103
B.2.2 Entanglement analysis . . . 105
B.3 The bubble bit technique . . . 107
B.4 Experimental results . . . 108
B.4.1 Quantum arithmetic . . . 109
B.4.2 Simulation of Grover's algorithm . . . 112

List of Figures

2.1 Example of transforming a bit flip error into a phase error (A, and vice versa (B. 17

2.2 (A The backward propagation of a phase error for the XOR gate. (B Gate network equivalence for the XOR gate resulting in a source-target interchange, where the Hadamard (H) gates are also changing the qubit basis from $\{|0\rangle, |1\rangle\}$ into $\{|+\rangle, |-\rangle\}$. 18

2.3 Majority voting circuit when: A) no error is present; B) q_2 is affected by a flip-error; C) q_3 is affected by a small error. 19

2.4 Ancilla usage for copying relevant information from data qubits. 20

2.5 Fault-tolerant procedure involving the ancilla qubits: coding the data and ancilla so that the data errors are copied onto the coded ancilla. The syndrome is obtained by some quantum gates followed by ancilla measurement. 20

2.6 Ancilla qubit usage: A) design that spreads the phase errors; B) fault-tolerant design. 21

2.7 The circuit that computes the syndrome for Steane's 7-qubit code. 24

2.8 The circuit that returns the Steane encoding of an arbitrary state. 25

2.9 Ancilla coding: A) Shor's tecnique with the two rightmost XOR gates performing ancilla verification; B) Steane's technique; C) Verfication for Steane's ancilla, where the 'State ancilla coding' blocks contain the circuit from B) except the rightmost level of Hadamard gates. 27

2.10 Bit-flip syndrome computation block, for Shor ancilla coding. 30

2.11 Phase-flip syndrome computation block, for Shor ancilla coding. 31

2.12 Single quantum error correcting circuit with Shor ancilla coding. 32

2.13 Single quantum bit-flip error correcting circuit with Steane ancilla coding. . . 33

2.14 Single quantum phase-flip error correcting circuit with Steane ancilla coding. . 34

2.15 Stabilizer generator G_3 is not changing the overall Steane coded superposition. 35

2.16 Syndrome computation circuit for a stabilizer code. 36

2.17 Fault tolerant circuit for stabilizer coding, using one qubit for each syndrome bit. 37

2.18 Stabilizer generator measurement with Steane ancilla coding. 39

2.19 Error-correction with stabilizer generator measurement, Steane ancilla, and syndrome computation according to the check matix. 40

2.20 Straightforward implementation of some quantum gates operating on fault-tolerant coded qubits: a) *Hadamard*; b) *NOT*; c) *Phase Shift*; d) *XOR*. 41

2.21 The principle used for measuring the ancilla qubits in the $\{|\psi\rangle, |\phi\rangle\}$ basis. . . . 42

2.22 The implementation of the fault-tolerant Toffoli gate. 43

2.23 Graphical representation of accuracy degree required for the corresponding N, for different p's: 3 for xi_1, 4 for xi_2, 5 for xi_3. xi_4 corresponds to the no-coding situation, while ref is the reference accuracy (i.e. the accuracy allowed by today's state of the art technology). 45
2.24 Concatenated coding: each qubit can be encoded by a block of sub-qubits. . . 46
2.25 Transversal implementation of the XOR gate for concatenated coding. 46

3.1 Quantum fault tolerant achievements on different abstraction levels. 52
3.2 Classical fault tolerant approach for a safe recovery assumption. Here, if the probability of faulty recovery is of the same order as the probability of faulty data, then coding is useless because the probability of erroneous corrected data remains ξ. 53
3.3 Using structural redundancy and voting of a trustworthy ancilla provided for a lower error probility ($\xi^{\frac{n}{2}}$). 53
3.4 Steane safe recovery procedure. Corrected data has an error probability of order ξ^2 by using structural redundancy when testing reveals bad ancilla. . . . 54
3.5 Concatenated coding affected by correlated errors. 54
3.6 Steane error recovery affected by correlated errors. 55
3.7 Reconfigurable quantum gate array: the involved registers. 56
3.8 The desired quantum switch. 57
3.9 Linear connection of basic reconfigurable quantum gate arrays, allowed by the second limitation. 57
3.10 Special conditioned gates: a) Hadamard, and b) qubit measurement. Qubit c is used for control, t is target for Hadamard, and m measured if $c = |1\rangle$. 59
3.11 The basic reconfigurable cell for stabilizer encoding solutions. 60
3.12 When the configuration register has a quantum nature, the same reconfigurable quantum gate array acts a a superposition of k simultaneous distinct circuits. These circuits share the same input state and the same output qubits. The output qubits encode a superposition of the superposed circuits distinct outputs. 62
3.13 Evolution of accuracy threshold value for $rQHW$ stabilizer codes ('xir' function) with the number of computational steps (N). The technological accuracy limit ('lim') is also provided for a relevant comparison. 64

4.1 Basic idea for fitness function construction: when is applied to valid individuals it produces a value in the valid area (upper half $|10\ldots00\rangle \div |11\ldots11\rangle$) of the fitness register, whereas when applied to invalid individuals the corresponding values in the fitness register will always be in the invalid area (lower half $|00\ldots00\rangle \div |01\ldots11\rangle$). 74
4.2 The format of the fitness register, for the two's complement approach of oracle implementation. 74
4.3 Oracle implementation for a fitness register having the structure from Figure 4.2. 75

5.1 Serial/parallel insertion on simple/complex saboteurs. $D_1 \ldots D_N$ are the drivers, $R_1 \ldots R_N$ the signal receivers, while f_S stands for the function required by complex saboteur computation. 80

5.2 A) Illustration of configuration's role in mutant-based fault injection: the faulty architecture is selected when the fault must be activated. B) The most commonly used mutant construction: a saboteur is added to the normal (no faulty) architecture. 80
5.3 Generation of fault injection executable model within the HDL framework [43]. 83
5.4 Generation of control list, for the experimental fault injection within the HDL framework [43]. 83
5.5 An overview of the QUERIST project. 85
5.6 The bubble bit HDL simulation model, when fault injection is applied according to the error and fault occurence models presented in [39]. 86
5.7 The effect of faulty gate operation on the processed qubits: a) gate bit-flip fault, b) gate phase-shift fault. 87
5.8 Circuit for singular bit-flip error correction. 88
5.9 Bubble records produced by simulating error correction with the circuit from Figure 5.8. 89

A.1 The partition of the single bit-flip error correcting circuit (stabilizer code and Steane ancilla) uses 2 basic cells of 14 input qubits. 94
A.2 Circuit for setting the 6-qubit configuration state, which is used in setting a quantum configuration for the stabilizer encoder. 98

B.1 Example of approaching the HDL simulation of a quantum circuit. 100
B.2 Situations where the same circuit (2 gates: Hadamard and XOR) is involved in an entanglement case (A), and in a non-entanglement case (B). 100
B.3 Data set example for the HDL approach. 101
B.4 Some relevant pieces of VHDL code, describing the circuit from Figure B.2. . . 102
B.5 A) Reduction of any n-qubit quantum circuit when entanglement is not present. B) reduction of the circuit from Figure B.2 situation (B), with $q_{\mathrm{i}} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ the input source, the target input $q_{\mathrm{Ti}} = |0\rangle$, the target output $q_{\mathrm{To}} = |1\rangle$, and the output source $q_{\mathrm{S}} = |1\rangle$. 103
B.6 Qubit group extraction algorithm (with depth d), for states dictated by Shor algorithm arithmetic circuits. 104
B.7 A) Complete entanglement example: for qubit groups $\{q_1, q_2\}$ and $\{q_0, q_3\}$ there is total entanglement inside the group, but there is no entanglement between the two groups; B) unitary transformation expression. 105
B.8 The circuit model of quantum computation. 105
B.9 The entanglement-aware quantum circuit simulation model. 106
B.10 Discrepancy between simulation times with behavioral and structural architectures, for Deutsch-Jozsa algorithm. 106
B.11 Quantum circuit simulation model, when the bubble bit technique is employed. 107
B.12 Bubble bit insertion technique. 108
B.13 A) The 1-qubit full adder; B) obtaining a 2-qubit adder from 1-qubit Σ cells. . 109
B.14 Bubble bit procedure results. 110
B.15 Bubble bit procedure example. 111
B.16 Extra memory requirements. 112

B.17 Grover algorithm circuit for a 2-qubit search register. The oracle U_O can be any of the a)-d) gates; also an entanglement analysis is provided by showing where it appears and where it is absent. 113
B.18 Bubble bit insertion rules for Grover algorithm states. 114
B.19 Relevant states for Grover algorithm simulation. 114
B.20 Bubble bit insertion results for 2-qubit Grover search simulation. 115
B.21 Bubble bit insertion procedure for $|\psi_3\rangle$. 115
B.22 2-qubit search Grover equivalent circuit, obtained with the bubble-bit technique in order to allow structural (i.e. polynomial) simulation. 116
B.23 Data types required by bubble record representation. 116
B.24 VHDL gate level implementation (entity-architecture pair) for bubble bit state transformation. 117
B.25 Time diagram resulted from VHDL simulation of Grover's algorithm, without the bubble bit technique. 118
B.26 Time diagram resulted from VHDL simulation of Grover's algorithm, with the bubble bit technique. 119
B.27 HDL bubble bit runtime results for Grover algorithm simulation, compared with the reference complexity. 120
B.28 Memory overhead dictated by bubble records for Grover algorithm simulation. A trendline is added to the sample data, showing polynomial growth. 120

List of Tables

2.1 All the possible H_A Hamming codes, with the rightmost column indicating whether the code contains an even ('e')or an odd ('o') number of 1s. 23
2.2 Steane code's syndrome interpretation, with the rightmost row indicating the position of the bit-flip error. 25
2.3 Stabilizer code syndrome interpretation; the rightmost row is indicating the bit-flip error position. 37

3.1 Hierarchical fault tolerance approach in classical computer hardware. 50
3.2 Hierarchical fault tolerance interpretation for quantum computational devices. The 'level' column also presents the correspondence with classical hardware levels. 50

5.1 An analysis of HDL-based simulated fault injection techniques, after Rimen *et al.* [42][43]. 81

A.1 The rows present the basis configurations which can be superposed with minimum gate usage (qubits $q_0, q_1 \dots q_{11}$ are the ordered positions of the filled dots from Equation A.13). By permutation of columns C_1, C_2, C_3 all the other distinct superposed configurations can be found. 97

B.1 Quantum full adder simulation results. 111
B.2 Experimental results for modulo adder and multiplier (simulation time). 112

Chapter 1

Introduction

Quantum computation has created a lot of research opportunities in the fields of computer science and engineering. Besides these advantages, serious problems and constraints emerge as physicists and engineers try to build such computational devices.

It is acknowledged that there are two main sources of concern in quantum computation. First, the problem of creating effective (i.e. better than classical) algorithms is linked to the intrinsic hard nature of the problems that are to be solved – on one hand – and, on the other hand, it is thwarted by programmers difficulty of thinking in quantum mechanical terms [30]. The second concern is related to the difficulty of building quantum mechanical devices, which perform computation by a ”coherent manipulation of atomic-scale dynamics” [51]. These atomic-scale particles will exhibit the desired behavior only if they are isolated from the macroscopic environment. Due to todays available technology, this is not possible to an extent that allows us to consider having intrinsic reliable quantum components.

This report is focused on quantum computation dependability issues, by taking into account some very important aspects: arbitrary long fault tolerant quantum computation measured by the accuracy threshold, the reconfigurable quantum hardware (circuits) solutions, and the experimental, simulated assessment of quantum circuits reliability.

In the theory of fault tolerant quantum computation, the most important metric for measuring the reliability attribute [6] is the accuracy threshold [38][39][63]. But having a fault tolerant quantum computation, even when there is compliance with the accuracy threshold, is not likely when the computational process requires a large number of steps. Therefore, a supplementary technique is employed – the concatenated coding – which is designed to maintain arbitrary long, fault tolerant computation. Instead, this report proposes a solution based on what is generically called ”reconfigurable Quantum Hardware” (or rQHW), in order to attain fault tolerant quantum computation on an arbitrary large number of steps. Then, our rQHW investigation goes further by sketching the way we may implement Evolvable Quantum Hardware. This means that the configuration register of our reconfigurable Quantum Gate Array must be set by a quantum genetic algorithm. It turned out that any genetic algorithm can be run in $\mathcal{O}\left(\sqrt{n}\right)$ time in a quantum computational environment.

The simulated quantum circuit reliability assessment is approached as an extension of our already presented quantum circuit simulation framework [54][55][56]. Thus, necessary theoretical and implementation background is provided, so that we have means for applying fault injection and applying correct processing of experimental data results. The inspiration of a multilevel approach, motivated by the HDL features, is drawn from the classical hardware simulated fault injection techniques [11][42][43], which are both very effective and intensively used [61]. The proposed quantum fault injection strategy maintains the flexibility of the classical approach, and its multilevel HDL description manner. Also, the entanglement-related issues are dealt with, with a resulting bubble-bit-based [56] technique.

All these aspects are just part (the theoretical foundation) of a bigger project called QERIST, which stands for QUantum ERror Injection Simulation Tool, designed for reliability measures assessment. QUERIST is however an ongoing project, and its description is beyond the scope of this report, which is concerned only with proving the validity of the basic concepts.

1.1 Report goals

By approaching the field of quantum computation from the above mentioned viewpoints, namely improving the reliability metrics, and the simulated assessment of appropriate fault tolerance parameters, some objectives arise as essential for this report, and others – even though are marginal with respect to our stated theme – have an major impact on the field of quantum computation in general.

The primary objectives of this report are:

- a comprehensive analysis of the available quantum FTAMs (Fault Tolerance Algorithms and Methodologies) [3][4], that allows a necessary critique from an engineering standpoint;
- the design a classical-hardware-inspired methodology which will further improve the measures of quantum dependabilitys attributes;
- the extension of our quantum circuit simulation framework, so that it can perform fault injection in order to experimentally measure the reliability parameters of the simulated circuits.

When considering the domanin that was approached in this report, one may consider the following objective as less important:

- implementing a quantum version of Evolvable Hardware (EHW), threfore designing the reconfigurable Quantum Gate Arrays and quantum circuits for running Quantum Genetic Algorithms [51].

However, this contribution may prove valuable in the quantum computation research community.

All the stated objectives are complying with the general idea of this reports approach: finding common ground for quantum computing and computer hardware engineering description and design methodologies. Some aspects can be mapped with little intervention from one field to the other. But for some problems (i.e. implementing genetic algorithms) quantum computation provides an efficiency framework.

1.2 Report outline

This report is structured as follows: Chapter 2 introduces the concepts and techniques that are used for designing fault tolerant quantum circuits, and for measuring the parameters of the relevant dependability attributes. Then, Chapter 3 is providing the critique of what is presented in the previous chapter, and introduces a radical new technique for preserving arbitrary long reliable quantum compuating by employing the so-called reconfigurable Quantum Gate Arrays (rQGA). This chapter also provides practical examples of how the technique actually works, along with a fault tolerance assessment based on the accuracy theshold. Some of eamples are presented inside Appendix A

If the configuration register of a rQGA contains a state dictated by a genetic algorithm, then we have a quantum version of the evolvable hardware. Chapter 4 presents how is it possible to implement a Genetic Algoriithm in a quantum computational environment.

In Chapter 5.1.1 we have analyzed the way to extend our quantum circuit simulation framework (which is updated in Appendix B with the latest achievements) to fault injection abilities. The analysis is based on a comparison with the classical hardware available techniques and the corresponding study that outlines the aspects that are appropriate for quantum implemetation, and emphasizes the distinctive characteristics of quantum fault injection according to the quantum error and fault occurence models.

Chapter 2

Fault Tolerance in Quantum Computation

In quantum computation dependability is not just a quality indicator. Due to the omnipresent nature of the error sources, the need for error detection and correction is vital here. Without such mechanisms there could be no realistic prospect of an operational quantum computational device. Therefore, the fact that error detecting and correcting techniques have been developed has enhanced the feasibility of a potential quantum computer.

2.1 Introduction

The main idea, just like for classical computation, is to employ special coding in order to protect useful data from the destructive effect of the environment. There are two main sources of errors: the first one is due to the erroneous behavior of the quantum gate and is producing the so-called processing errors, while the second is generated by the macroscopic environment interacting with the quantum state, and measuring it in an unfortunate manner. The later source is producing the storing and transmitting errors.

Within the quantum computational framework, the developed techniques for error detection and correction have a sound error recovery process, and the error propagation is kept under control. However, any reliability technique (and the quantum ones make no exception) is based on making a reliable system out of unreliable components. But how much unreliable could they be? In classical computation this is not a problem, because digital storing, transmitting and processing are very reliably implemented jobs. By contrast, quantum computation could be ruined by innacuracies and errors if the error probability in the basic components (qubits, quantum gates) excedes an *accuracy threshold.* This is a critical aspect; usually the microscopic quantum states are prone frequent errors. Hence, in our context, the *safe recovery* issue becomes extremely important.

The main error source is the *decoherence* effect [30]. The environment is constantly trying to measure the sensible quantum superposition state, and technologically it is not possible to perfectly isolate the microscopic state from the environment. A measurement means that the superposition decays, becoming a projection of the state vector onto the basis vector (the eigenstate). But the most insidious kind of errors appears when decoherence is affecting the quantum amplitudes without destroying them; these errors are very similar to small analog errors.

The solution to the problems stated above is represented, on one hand, by intrinsic fault tolerance by technological implementation (topological interactions Ahranov-Bohm [1]) and, on the other hand, by error correcting techniques at the unitary (gate network) level. We will focus on the error detecting and correcting techniques, which are not easy to approach due to the quantum computational constraints: the useful state could not be observed (otherwise it will decohere), nor could it be cloned.

2.2 Specific problems

Although the fault tolerance techniques in quantum computation are inspired from the classical ones, specific problems arise, increasing the number of important issues to be addressed (safe recovery, for instance), and thus growing the complexity of the corresponding circuitry [62]. This subsection is focusing on presenting problems, without sketching any solution.

2.2.1 Error model

As expressed in bra-ket notation [30], the qubit is a normalized vector in some Hilbert space $\mathcal{H}^2$, with $\{|0\rangle, |1\rangle\}$ being the orthonormal basis: $|\psi\rangle = a_0|0\rangle + a_1|1\rangle$ ($a_0, a_1 \in \mathbb{C}$ are the so-called quantum amplitudes, which represent the square root of the associated measurement probabilities for the eigenstates $|0\rangle$ and $|1\rangle$ respectively, with $|a_0|^2 + |a_1|^2 = 1$). Therefore, the qubit can be affected by 3 types of errors:

- **E 1: Bit flip errors.** This kind of errors is somehow similar to a classical bit blip error. For a single qubit things are exactly the same as in classical computation

$$\begin{aligned} |0\rangle &\mapsto |1\rangle, \\ |1\rangle &\mapsto |0\rangle. \end{aligned} \tag{2.1}$$

For 2 or more qubits, flip errors affecting the state could modify it, or could leave it unchanged. For instance, if we take the so-called *cat state* [39]

$$|\psi\rangle_{\text{Cat}} = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right) \tag{2.2}$$

and the first qubit is affected by the bit flip error the resulted state will be

$$|\psi\rangle_{\text{Cat}} \mapsto \frac{1}{\sqrt{2}}\left(|10\rangle + |01\rangle\right). \tag{2.3}$$

But, if both qubits are affected by bit flips, we will have no change in the state:

$$|\psi\rangle_{\text{Cat}} \mapsto \frac{1}{\sqrt{2}}\left(|11\rangle + |00\rangle\right) = |\psi\rangle_{\text{Cat}}. \tag{2.4}$$

- **E 2: Phase errors.** In quantum computation there is a new kind of error, affecting the phase of one of the qubit's amplitudes. This error is very dangerous, due to its propagation behavior. For a qubit's eigenstates, similar to Equation 2.1, this error is expressed:

$$\begin{array}{rcl} |0\rangle & \mapsto & |0\rangle, \\ |1\rangle & \mapsto & -|1\rangle. \end{array} \tag{2.5}$$

  The phase error makes sense only when dealing with superposition states. If we take the equally weighted qubit superposition state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and affect it with a phase error, the result is presented in the following mapping:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.6}$$

  There is a strict correspondence between the bit flip and the phase error types. This correspondence is due to the way we can map to each other the Hilbert spaces with the same dimension but different basis. The bit flip is an error from the $\mathcal{H}^2$ space with basis $\{|0\rangle, |1\rangle\}$, whereas the phase error appears in the same space with basis $\left\{\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right\}$, or $\{|+\rangle, |-\rangle\}$. The space basis conversion, in our case, is made by applying the Hadamard transform as shown in Figure 2.1.

- **E 3: Small amplitude errors.** The amplitudes a_0 and a_1 of the quantum bit could be affected by small errors, similar to analog errors. Even though we might think that this kind of errors is not too dangerous – after all, it does not destroy the superposition and conserves the value of the superposed states – small amplitude errors could accumulate over time, eventually ruining the computation.

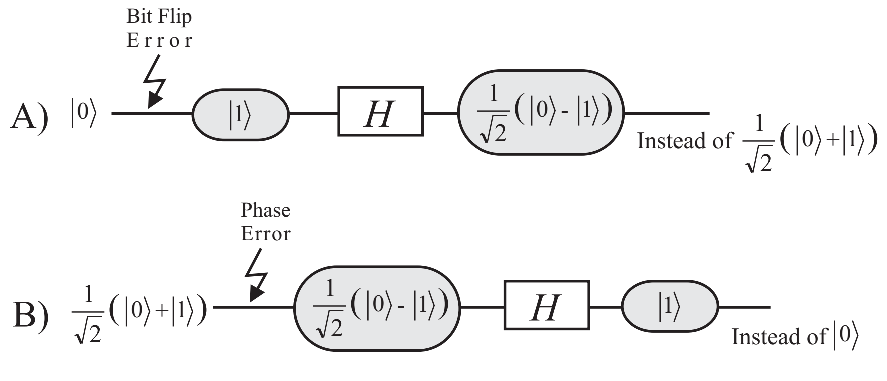

Bit Flip Error
A) $|0\rangle$
$|1\rangle$
H
$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
Instead of $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
Phase Error
B) $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
H
$|1\rangle$
Instead of $|0\rangle$

Figure 2.1: Example of transforming a bit flip error into a phase error (A, and vice versa (B.

2.2.2 Quantum constraints and problems

Quantum computation is not only introducing new types of errors, it is putting some computational constraints while generating some new problems. Thus, even if our fault tolerance techniques are inspired fom classical computation, we will have to adapt them to the quantum physics requirements.

Due to the quantum physics laws, we will have the following computational constraints:

- **C 1: The observation destroys the state.** In classical computation, the encoded information is observed – therefore, measured – in order to detect and correct the potential error. This is not possible in quantum computation, because the observation of a quantum state means that it is measured, destroying the useful state superposition.

- **C 2: Information copying is impossible.** One way to protect information, in classical computation, is to copy it, and use the copies in order to compare them with the original (the majority voting principle). But an important quantum physics law says that there is no possible cloning for the quantum state, meaning that we cannot correctly copy a quantum state.

Because of the error types, the constraints, the high error rate, and the quantum circuit structure quantum error correction encounters some serious problems:

- **P 1: Non-destructive measurement.** In spite of constraint **C 1** we need to measure the encoded information is some way, without destroying it. As we cannot directly measure the encoded state, we need to properly prepare some scratch (ancilla) qubits. We will have to obtain the desired information about the useful state only by measuring the ancilla qubits.

- **P 2: Fault-tolerant recovery.** Due to the high error rate in quantum computational devices, it is very likely that, after detecting the error, the correcting process itself is affected by errors. This is a very important aspect because, if the recovery is not fault-tolerant, then the entire effort of coding and error detecting becomes useless.

- **P 3: Phase error backward propagation.** The phase errors have a awkward propagation behavior. If we take into consideration the *XOR* gate from Figure 2.2 (A, a flip error affecting the target qubit (b) will propagate backward onto the source qubit. This happens due to the gate network equivalence from Figure 2.2 (B and the basis transformation described by Figure 2.1.

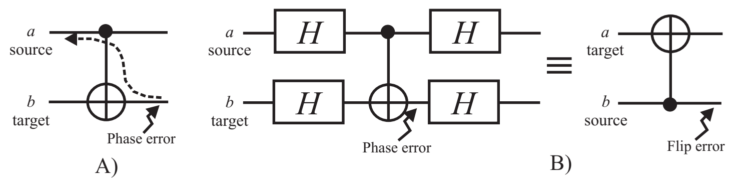
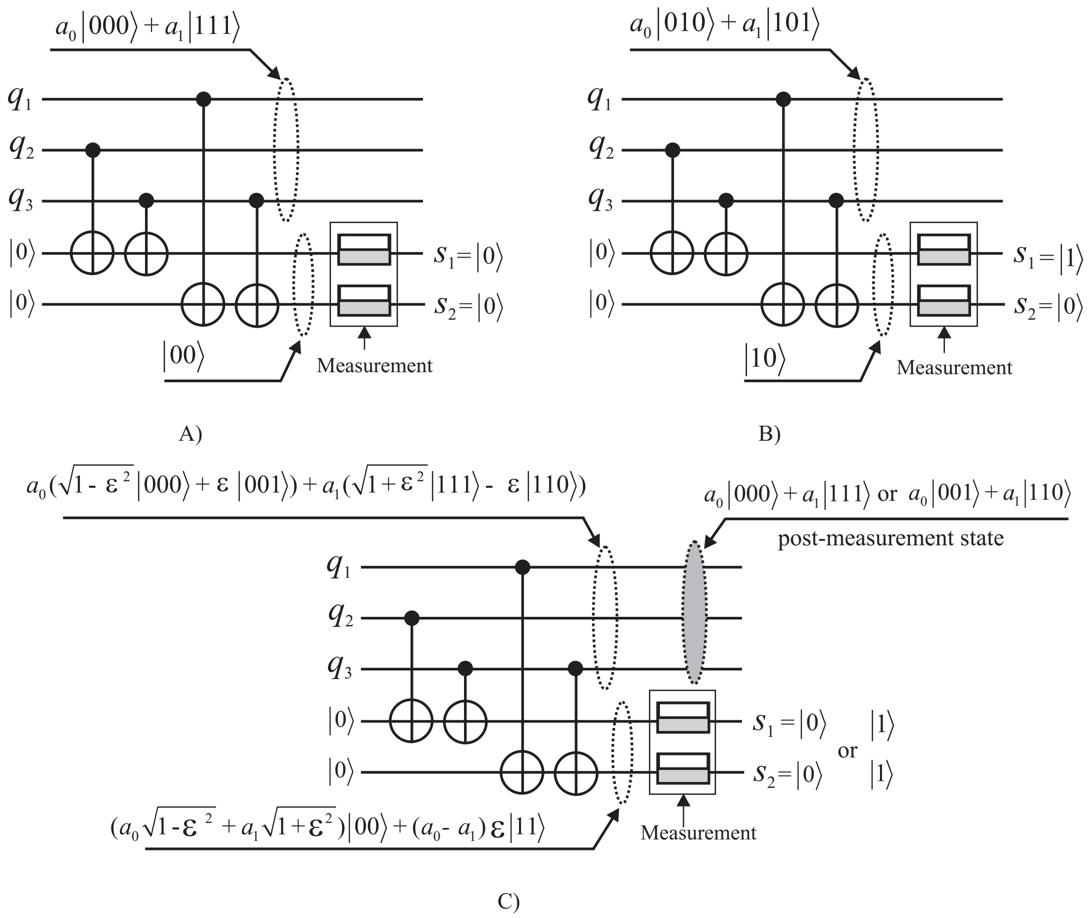


Figure 2.2: (A The backward propagation of a phase error for the XOR gate. (B Gate network equivalence for the XOR gate resulting in a source-target interchange, where the Hadamard (H) gates are also changing the qubit basis from $\{|0\rangle, |1\rangle\}$ into $\{|+\rangle, |-\rangle\}$.

2.3 Strategies for quantum fault tolerance

In order to deal with the problems described in the previous section, despite the serious constraints that quantum physics dictate, we will have to follow the following strategies:

- **S 1: Digitising small errors.** The presence of small errors is not a major concern, as we can digitise the analog-like kind of errors with the following technique.



$a_0|000\rangle + a_1|111\rangle$
q_1
q_2
q_3
$|0\rangle$
$|0\rangle$
$s_1 = |0\rangle$
$s_2 = |0\rangle$
$|00\rangle$
Measurement
A)
$a_0|010\rangle + a_1|101\rangle$
$s_1 = |1\rangle$
$s_2 = |0\rangle$
$|10\rangle$
Measurement
B)
$a_0(\sqrt{1-\varepsilon^2}|000\rangle + \varepsilon|001\rangle) + a_1(\sqrt{1+\varepsilon^2}|111\rangle - \varepsilon|110\rangle)$
$a_0|000\rangle + a_1|111\rangle$ or $a_0|001\rangle + a_1|110\rangle$
post-measurement state
$s_1 = |0\rangle$ or $|1\rangle$
$s_2 = |0\rangle$ $|1\rangle$
$(a_0\sqrt{1-\varepsilon^2} + a_1\sqrt{1+\varepsilon^2})|00\rangle + (a_0 - a_1)\varepsilon|11\rangle$
Measurement
C)

Figure 2.3: Majority voting circuit when: A) no error is present; B) q_2 is affected by a flip-error; C) q_3 is affected by a small error.

Suppose that we use a simple coding technique that is suitable for majority voting, using 3 qubits ($|q_1, q_2, q_3\rangle$) for representing 1 useful qubit ($|q\rangle = a_0|0\rangle + a_1|1\rangle$) the way that the following 2 mappings prescribe: $|0\rangle \mapsto |000\rangle$ and $|1\rangle \mapsto |111\rangle$. Thus, state $|q\rangle$ will be coded the way it is shown in the following equation:

$$|q\rangle = |q_1, q_2, q_3\rangle = a_0|000\rangle + a_1|111\rangle \tag{2.7}$$

The circuit from Figure 2.3 A) will produce the syndrome $(s_1, s_2) = (q_2 \oplus q_3, q_1 \oplus q_3)$ which will have the value $(0, 0)$ when there is no error, and a non-zero value when an error occurs. Moreover, this syndrome is able to indicate the qubit that is affected by a single flip error (10 for q_1, 01 for q_2, and 11 for q_3). Figure 2.3 B) presents the majority voting circuit signaling a flip error affecting qubit q_2. This single flip error is described by:

$$|q\rangle = a_0|010\rangle + a_1|101\rangle \tag{2.8}$$

When we have a small error affecting, for instance, the amplitudes of qubit q_3, then the circuit from Figure 2.3 then the expression for $|q_1, q_2, q_3\rangle$ will be:

$$|q\rangle = a_0\left(\sqrt{1-\epsilon^2}|000\rangle + \epsilon|001\rangle\right) + a_1\left(\sqrt{1+\epsilon^2}|111\rangle - \epsilon|110\rangle\right) \tag{2.9}$$

As Figure 2.3 C) presents, measuring the syndrome qubits will yield either $(0, 0)$ or $(1, 1)$. The first syndrome indicates no error, which is right because the post-measurement state of $|q_1, q_2, q_3\rangle$ will be $a_0|000\rangle + a_1|111\rangle$. For the second possible syndrome $(1, 1)$, the post measurement state of $|q_1, q_2, q_3\rangle$ will completely flip the third qubit: $a_0|001\rangle + a_1|110\rangle$. However, the corresponding syndrome indicates the flip in the third qubit, therefore the small error is digitized and can be corrected as a flip error.

- **S 2: Ancilla usage.** We know that qubit cloning is impossible, and therefore a majority voting strategy is hard to implement. However, some kind of an information copy can be achieved: by using ancilla qubits we can duplicate the eigenstate information inside the existing superposition (Figure 2.4).

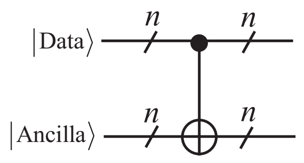


Figure 2.4: Ancilla usage for copying relevant information from data qubits.

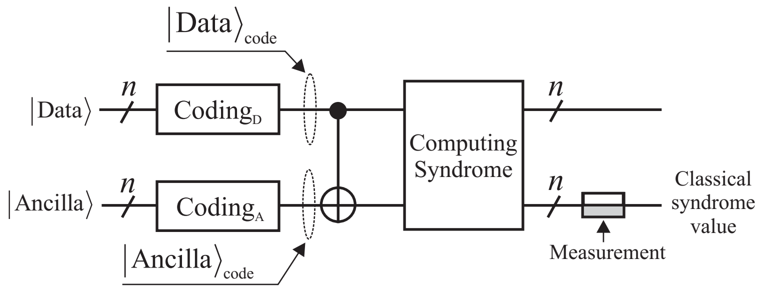


Figure 2.5: Fault-tolerant procedure involving the ancilla qubits: coding the data and ancilla so that the data errors are copied onto the coded ancilla. The syndrome is obtained by some quantum gates followed by ancilla measurement.

As a consequence, the ancilla qubits will be entangled with the useful data and any measurement performed on the ancilla could have a repercussion onto the state of the useful qubits. The appropriate strategy will employ special coding for both data qubits and ancilla, followed by the computation of an error syndrome. The syndrome must be yielded by measuring the ancilla, provided that the encoding enables us to copy only the data errors onto the ancilla, but does not copy the useful data (see Figure 2.5).

- **S 3: Avoiding massive spreading of phase errors.** The way that phase errors are propagating through the quantum gate is from the source to the target. However, as shown in **E2** phase errors are propagated backwards from the target to the source. Therefore, the frequently encountered designs like those in Figure 2.6 are prone to spread phase errors; a phase error on the target qubit will propagate on all the source qubits. The solution is to use more ancilla qubits as targets, so that no ancilla qubit is used more than once (see Figure 2.6 B).

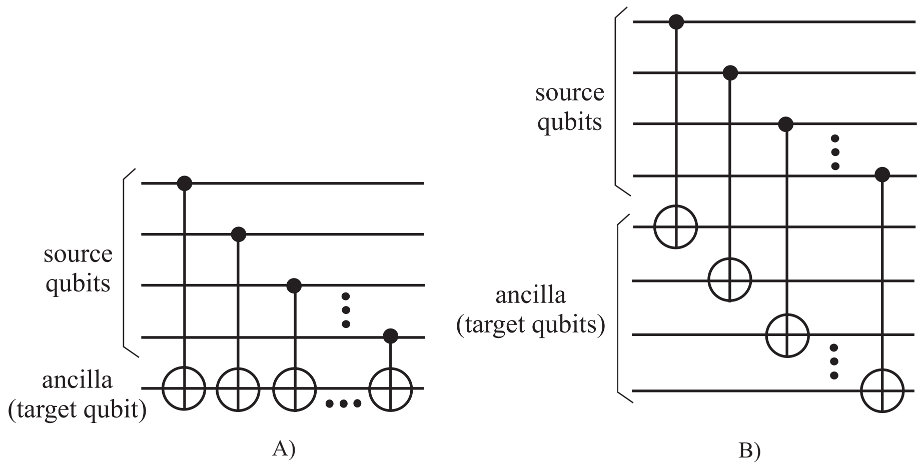


Figure 2.6: Ancilla qubit usage: A) design that spreads the phase errors; B) fault-tolerant design.

- **S 4: Ancilla and syndrome accuracy.** The ancilla code is some known quantum state. However, setting this state could be an erroneous process. Also, computing the syndrome is prone to errors, and if an erroneous syndrome is taken into consideration for the error recovery, then not only that the error is not corrected, but other errors are dictated in the useful state. Hence, on one hand, we have to make sure that the ancilla is in the right state by verifying these qubits and recovering the ancilla if such is the case; on the other hand, in order to have a reliable syndrome, we must compute it several times.

- **S 5: Error recovery.** As the small errors could be digitised as shown in **S1** (therefore, they are either corrected or transformed into flip errors) the recovery must deal only with flip and phase errors. A state that needs to be recovered, because of an error is described by the following equation:

$$a_0|0\rangle + a_1|1\rangle \xrightarrow{\text{error}} \begin{cases} a_0|0\rangle + a_1|1\rangle & \text{if no error is present} \\ a_0|1\rangle + a_1|0\rangle & \text{for a flip error} \\ a_0|0\rangle - a_1|1\rangle & \text{for a phase error} \\ a_0|1\rangle - a_1|0\rangle & \text{for both flip and phase errors} \end{cases} \tag{2.10}$$

Correcting a flip error means negating the affected qubit, thus applying the transformation characterized by:

$$U_{\text{Not}} = \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.11}$$

Changing the basis from $\{|0\rangle, |1\rangle\}$ into $\{|+\rangle, |-\rangle\}$, will transform the phase error into a flip error that can be corrected by σ_x; after the correction, the initial basis must be restored. These actions are summarized in Equation (2.12), and they must be taken in order to correct the phase error.

$$U_Z = H \cdot \sigma_x \cdot H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.12}$$

Of course, the correction of the third kind of errors (with both bit and phase flip) is achieved by applying a composed transformation upon the affected qubit:

$$U_Y = U_{\text{Not}} \cdot U_Z = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.13}$$

2.4 Quantum error correction

Quantum error detection and correction is performed with special coding techniques, which are inspired from the classic Hamming codes. The classical error detecting and correcting coding is adapted so that it becomes suitable for the quantum strategy that allows only the ancilla qubits to be measured. Even so, the syndrome obtained by measuring the proper ancilla qubits reveals the nature of the error. Thus, the error-correcting strategy described in the previous section may be employed in order to recover the detected fault.

2.4.1 Steane coding

Steane's 7-qubit code is inspired from classical Hamming coding, and it is designed to make sure that the fault tolerant strategies from the previous section could be employed. It is used for useful data coding, and could be adapted for ancilla coding as well. Because this code is derived from a classical single error correcting error, it can detect and correct only single qubit faults in the code block. This means that we cannot recover two qubit faults if the same type of error affects them. However, the situation where one of the two erroneous qubits is affected by a bit flip and the other one by a phase flip can be recovered.

Let H_A be a Hamming matrix describing a code with 4 useful bits ($n = 4$), and 3 redundant bits ($k = 3$).

u_0	u_1	u_2	u_3	c_0	c_1	c_2	o\e
0	0	0	0	0	0	0	e
0	0	0	1	1	0	1	o
0	0	1	0	1	1	1	e
0	0	1	1	0	1	0	o
0	1	0	0	0	1	1	o
0	1	0	1	1	1	0	e
0	1	1	0	1	0	0	o
0	1	1	1	0	0	1	e
1	0	0	0	1	1	0	o
1	0	0	1	0	1	1	e
1	0	1	0	0	0	1	o
1	0	1	1	1	0	0	e
1	1	0	0	1	0	1	e
1	1	0	1	0	0	0	o
1	1	1	0	0	1	0	e
1	1	1	1	1	1	1	o

Table 2.1: All the possible H_A Hamming codes, with the rightmost column indicating whether the code contains an even ('e')or an odd ('o') number of 1s.

$$H_A = \begin{array}{c} \begin{array}{ccccccc} c_0 & c_1 & c_2 & u_0 & u_1 & u_2 & u_3 \end{array} \\ \left(\begin{array}{ccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array} \right) \end{array} \tag{2.14}$$

Because $k \geq \log_2\lceil n+k+1 \rceil$ this code is SEC-DED (Single Error Correction - Double Error Detection). The above Hamming matrix can also be expressed as

$$\left\{ \begin{array}{lcl} c_0 & = & u_0 \oplus u_2 \oplus u_3 \\ c_1 & = & u_0 \oplus u_1 \oplus u_2 \\ c_2 & = & u_1 \oplus u_2 \oplus u_3 \end{array} \right. \tag{2.15}$$

This means that Table 2.1 contains all the possible valid codes generated by H_A.

The Steane 7-qubit coding of $|0\rangle$ consists of a equally weighted superposition of all the valid Hamming 7-bit words with an even number of 1s:

$$\begin{array}{rcl} |0\rangle_S & = & \frac{1}{2^{\frac{3}{2}}} \sum \text{even}_{(u_0u_1,u_2u_3c_0c_1c_2)} |u_0u_1u_2u_3c_0c_1c_2\rangle \\ & = & \frac{1}{2^{\frac{3}{2}}} (|0000000\rangle + |0010111\rangle + |0101110\rangle + |0111001\rangle + |1001011\rangle + |1011100\rangle \\ & & +|1100101\rangle + |1110010\rangle) \end{array} \tag{2.16}$$

The similar superposition of the odd 1s Hamming code words is used for $|1\rangle$ coding:

$$\begin{array}{rcl} |1\rangle_S & = & \frac{1}{2^{\frac{3}{2}}} \sum \mathrm{odd}_{(u_0u_1,u_2u_3c_0c_1c_2)} |u_0u_1u_2u_3c_0c_1c_2\rangle \\ & = & \frac{1}{2^{\frac{3}{2}}} (|1111111\rangle + |1101000\rangle + |1010001\rangle + |1000110\rangle + |0110100\rangle + |0100011\rangle \\ & & +|0011010\rangle + |0001101\rangle) \end{array} \tag{2.17}$$

With this code, any singular qubit flip error is detected and can be corrected by computing the following syndrome:

$$\begin{cases} m_0 & = & c_0 \oplus u_0 \oplus u_2 \oplus u_3 \\ m_1 & = & c_1 \oplus u_0 \oplus u_1 \oplus u_2 \\ m_2 & = & c_2 \oplus u_1 \oplus u_2 \oplus u_3 \end{cases} \tag{2.18}$$

Figure 2.7 presents how to compute the syndrome for this code. The ancilla is not prepared according to previous sections strategies. This aspect will be further elaborated in the next subsection.

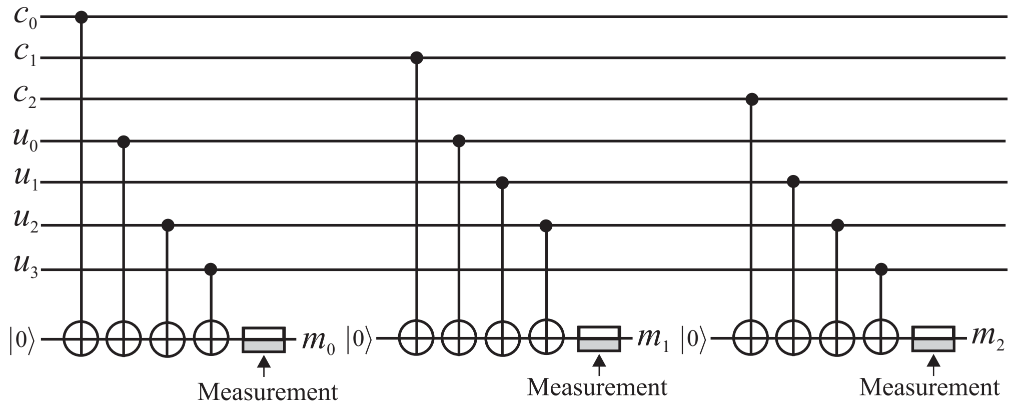


Figure 2.7: The circuit that computes the syndrome for Steane's 7-qubit code.

When we have single errors, the measurement of the syndrome qubits will have certain results, due to the fact that every superposed state will have the same affected position, and therefore will dictate the same syndrome eigenvalue. Table 2.2 indicates the link between the syndrome value and the position of the affected qubit. If, for instance, the flip error affects qubit u_1, then all the eigenstates contained in $|0\rangle_S$ and $|1\rangle_S$ will produce the same syndrome: $(m_0, m_1, m_2) = (0, 1, 1)$, so the syndrome quantum state will be $|m_0, m_1, m_2\rangle = |011\rangle$.

This code it is designed to correct bit-flip errors, but with the basis change (obtained with a Hadamard transform) the phase error is transformed into bit flip error, which can be corrected. Thus, the phase error correcting code will be described by:

$$\begin{array}{rcccl} |\bar{0}\rangle_S & = & H \cdot |0\rangle_S & = & \frac{1}{\sqrt{2}} (|0\rangle_S + |1\rangle_S) \\ |\bar{1}\rangle_S & = & H \cdot |1\rangle_S & = & \frac{1}{\sqrt{2}} (|0\rangle_S - |1\rangle_S) \end{array} \tag{2.19}$$

Steane's 7-qubit code is extremely useful in maintaining the accuracy of the state and will recover the coded block from any singular error (bit or phase flip). However, one has to be

m_0	m_1	m_2	erroneous qubit
0	0	0	no error
0	0	1	c_2
0	1	0	c_1
0	1	1	u_1
1	0	0	c_0
1	0	1	u_3
1	1	0	u_0
1	1	1	u_2

Table 2.2: Steane code's syndrome interpretation, with the rightmost row indicating the position of the bit-flip error.

aware that this code cannot correct flip situations like:

$$\begin{array}{lclcl} |\bar{0}\rangle_S & \longrightarrow & |\bar{1}\rangle_S \text{ and} |\bar{1}\rangle_S & \longrightarrow & |\bar{0}\rangle_S \\ |\bar{0}\rangle_S & \longrightarrow & |\bar{0}\rangle_S \text{ and} |\bar{1}\rangle_S & \longrightarrow & -|\bar{1}\rangle_S \end{array} \tag{2.20}$$

Applying Steane coding on an arbitrary given quantum state $|\psi\rangle = a_0|0\rangle + a_1|1\rangle$, we must obtain the state from Equation 2.21 in order to use it for potential recoveries.

$$|\psi\rangle_S = a_0|0\rangle_S + a_1|1\rangle_S \tag{2.21}$$

This coding procedure is achieved with the circuit presented in Figure 2.8, where succesive

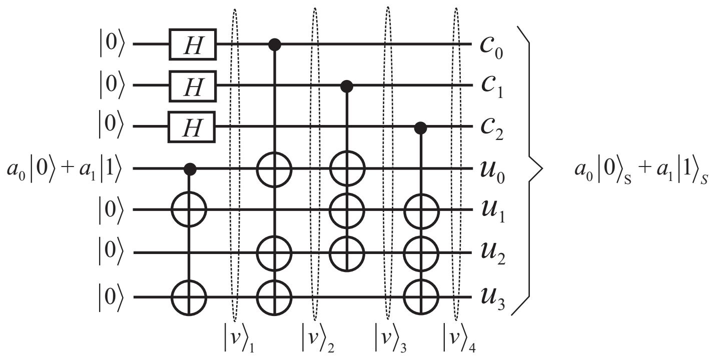


Figure 2.8: The circuit that returns the Steane encoding of an arbitrary state.

states $|v\rangle_1, |v\rangle_2, |v\rangle_3, |v\rangle_4$ are produced:

$$\begin{array}{rcl} |v\rangle_1 & = & a_0 \frac{1}{2^{\frac{3}{2}}} \left(\begin{array}{c} |0000000\rangle + |0010000\rangle + |0100000\rangle + |0110000\rangle \\ +|1000000\rangle + |1010000\rangle + |1100000\rangle + |1110000\rangle \end{array} \right) \\ & + & a_1 \frac{1}{2^{\frac{3}{2}}} \left(\begin{array}{c} |0001101\rangle + |0011101\rangle + |0101101\rangle + |0111101\rangle \\ +|1001101\rangle + |1011101\rangle + |1101101\rangle + |1111101\rangle \end{array} \right) \end{array} \tag{2.22}$$

$$\begin{aligned}|v\rangle_2 &= a_0\frac{1}{2^{\frac{3}{2}}}\begin{pmatrix}|0000000\rangle+|0010000\rangle+|0100000\rangle+|0110000\rangle\\ +|1001011\rangle+|1011011\rangle+|1101011\rangle+|1111011\rangle\end{pmatrix}\\ &+ a_1\frac{1}{2^{\frac{3}{2}}}\begin{pmatrix}|0001101\rangle+|0011101\rangle+|0101101\rangle+|0111101\rangle\\ +|1000110\rangle+|1010110\rangle+|1100110\rangle+|1110110\rangle\end{pmatrix}\end{aligned} \tag{2.23}$$

$$\begin{aligned}|v\rangle_3 &= a_0\frac{1}{2^{\frac{3}{2}}}\begin{pmatrix}|0000000\rangle+|0010000\rangle+|0101110\rangle+|0111110\rangle\\ +|1001011\rangle+|1011011\rangle+|1100101\rangle+|1110101\rangle\end{pmatrix}\\ &+ a_1\frac{1}{2^{\frac{3}{2}}}\begin{pmatrix}|0001101\rangle+|0011101\rangle+|0100011\rangle+|0110011\rangle\\ +|1000110\rangle+|1010110\rangle+|1101000\rangle+|1111000\rangle\end{pmatrix}\end{aligned} \tag{2.24}$$

$$\begin{aligned}|v\rangle_4 &= a_0\frac{1}{2^{\frac{3}{2}}}\begin{pmatrix}|0000000\rangle+|0010111\rangle+|0101110\rangle+|0111001\rangle\\ +|1001011\rangle+|1011100\rangle+|1100101\rangle+|1110010\rangle\end{pmatrix}\\ &+ a_1\frac{1}{2^{\frac{3}{2}}}\begin{pmatrix}|0001101\rangle+|0011010\rangle+|0100011\rangle+|0110100\rangle\\ +|1000110\rangle+|1010001\rangle+|1101000\rangle+|1111111\rangle\end{pmatrix}\end{aligned} \tag{2.25}$$

By inspection, it is obvious that $|v\rangle_4 = |\psi\rangle_S$.

2.4.2 Ancilla coding

In Figure 2.7 the ancilla qubits used for syndrome computing are not complying with strategy **S 3** from section 2.3. We have to correct that, in order to avoid massive phase-flip error spreading. Moreover, the ancilla coding process could be affected by errors, so some form of protection against such errors. Because of the *XOR* gates that connect data qubits to ancilla qubits, errors on the ancilla may affect the data when spreading backwards.

Hence, the design of the circuits that encode the ancilla must take into account two aspects: preventive design according to strategy **S 3**, and ancilla verification in order to make sure that the proper ancilla was set.

There are two coding techniques for the ancilla: the Shor and Steane coding. Shor's ancilla coding is considered as being somehow less prone to further errors in ancilla preparation, because the involved quantum operations are less complex. Figure 2.9 A) and B) presents the Shor and Steane ancilla coding. In A) besides Shor coding, ancilla verification is also included; the verification for Steane ancilla is in Figure 2.9 C).

Shor's code is an equally weighted superposition of all the 4-bit, even parity, binary words:

$$|\mathrm{Anc}\rangle_{\mathrm{Shor}} = \frac{1}{2^{\frac{3}{2}}}\begin{pmatrix}|0000\rangle+|1100\rangle+|1010\rangle+|1001\rangle+\\ |0110\rangle+|0101\rangle+|0011\rangle+|1111\rangle\end{pmatrix} \tag{2.26}$$

In Figure 2.9 A) state $|\mathrm{Anc}\rangle_{\mathrm{Shor}}$ is build by starting with the 4-qubit $|\mathrm{Cat}\rangle$ state, which is obtained with the leftmost Hadamard gate which dicatates the state

$$|w\rangle_1 = \frac{1}{\sqrt{2}}\left(|0000\rangle+|1000\rangle\right) \tag{2.27}$$

and then 3 successive *XOR* gates, so that

$$|w\rangle_2 = |\mathrm{Cat}\rangle = \frac{1}{\sqrt{2}}\left(|0000\rangle+|1111\rangle\right) \tag{2.28}$$

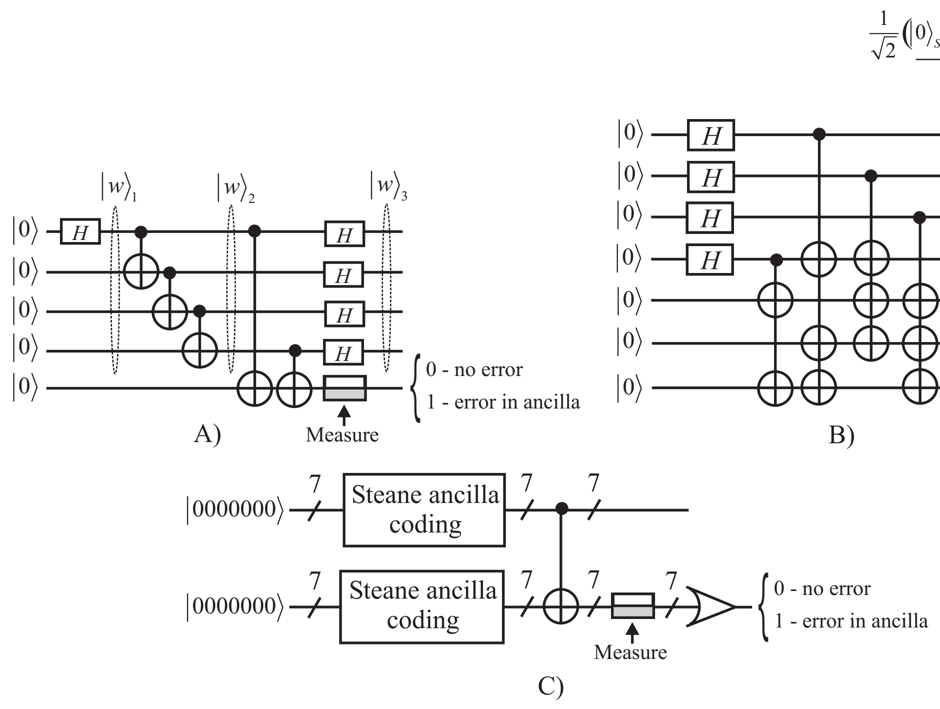


Figure 2.9: Ancilla coding: A) Shor's tecnique with the two rightmost *XOR* gates performing ancilla verification; B) Steane's technique; C) Verfication for Steane's ancilla, where the 'State ancilla coding' blocks contain the circuit from B) except the rightmost level of Hadamard gates.

The Hadamard transform, applied on the Cat state will always map it into an equally weighted superposition of even parity eigenstates:

$$|w\rangle_3 = \frac{1}{2^{\frac{3}{2}}}\left(\begin{array}{c} |0000\rangle + |1100\rangle + |1010\rangle + |1001\rangle + \\ |0110\rangle + |0101\rangle + |0011\rangle + |1111\rangle \end{array} \right) = |\text{Anc}\rangle_{\text{Shor}} \tag{2.29}$$

The idea for Shor's ancilla coding is that, if the bit-flip syndrome qubit is $|0\rangle$, then the parity of all the superposed eigenstates will remain unchanged, i.e. even. If the syndrome qubit is $|1\rangle$, the parity of all superposed eigenstates will be changed, thus becoming odd.

The two *XOR* gates between states $|w\rangle_2$ and $|w\rangle_3$ are verifying the correctness of the $|\text{Cat}\rangle$ generation. If a single error occurs when the Cat state is prepared, then the leftmost and the rightmost eigentate bits will always have complementary values and the measurement of the lower qubit in Figure 2.9 A) will have a certain result, $|1\rangle$. If there is no error in Cat state preparation, then the marginal eigenstate bits will always have the same value, so the measurement will always yield $|0\rangle$.

Figure 2.9 B) is similar to Figure 2.8, but in the earlier the middle qubit is $|0\rangle$ instead of $a_0|0\rangle + a_1|1\rangle$. Therefore, the resulted state will be $|0\rangle_S$, instead of $a_0|0\rangle_S + a_1|1\rangle_S$. Then, by applying $H^{\otimes 7}$, we will obtain

$$|\text{Anc}\rangle_{\text{Steane}} = \frac{1}{\sqrt{2}}\left(|0\rangle_S + |1\rangle_S\right) \tag{2.30}$$

In the Steane ancilla coding case, the bit-flip syndrome is obtained by first applying 7 *XOR* gates having the data qubit as source, and the ancilla qubit with the same position as target. The ancilla is measured, and then we apply the Hadamard matrix check H_A upon the resulted word, in order to get the syndrome.

When the ancilla is prepared with Steane's code, the ancilla verification is performed as presented in Figure 2.9 C). Two $|0\rangle_S$ blocks are prepared, and a bitwise *XOR* is performed on the two qubit blocks. Then, a measurement is performed on the block that contains *XOR*s target qubits. If there is no error, the measurement result will be an eigenstate from $|0\rangle_S$ and the nonmeasured block has successfuly passed the verification. If a bit-flip error occurs, then this fact is revealed by performing an *OR* on the classical bit measurement results. We can apply Hamming correction uppon these measurement results, and at the same time finding if the measured block was $|0\rangle_S$ (the correction is according to the Hamming matrix) or $|1\rangle_S$ (the correction attempt will *qubitwise* negate the data block). However, this procedure is not completely reliable, as we do not know if the bit flip has occured in the verified or in the measured block. One possible solution is to perform this procedure twice.

2.4.3 Putting it all together

Having the results from our previous two subsections, we are able to present the complete circuits for error correction. Figures 2.10 and 2.11 present the syndrome for bit-flip and phase-flip syndromes respectively when using Shor ancilla coding. The later figure is obtained from the previous by transforming the phase error into a flip error (see Figure 2.1) and applying the equivalence from Figure 2.2. The entire error-correction circuit for Steane ancilla preparation is presented in Figures 2.13 (bit-flip), and 2.14 (phase-flip).

Shor's ancilla preparation has the following downsides: it uses a large number of qubits (prone to error), cannot repair the ancilla. Steane ancilla coding uses a smaller qubit number, is able to repair the ancilla, but the ancilla preparation is more complex, and therefore prone to error. The two ancilla methodologies are somehow equivalent, and both of them must use structural redundancy (i.e. ancilla code preparation can be redone until the *ae* qubits will be '0'). However, even if the ancilla was correctly set, errors could still appear in syndrome computation.

When taking into account the model of non-correlated errors, it was shown that the redundant syndrome computation will assure a data fidelity of order $1 - \mathcal{O}(\xi^2)$ when the single qubit error probability is ξ. This means that, the probability of a faulty recovered quantum state is of the same order as the probability of having two simultaneous errors in the initial state; in fact, the double error cannot be recovered by this code. If the safe recovery strategies (ancilla coding, redundant syndrome measurement) were not to be implemented, the fidelity order would have been of the order $1-\mathcal{O}(\xi)$. Such a fidelity factor will not endorse the use of fault tolerant coding and error correction in the first place.

2.5 Stabilizer codes

The stabilizer codes – when compared with Steane coding – are a generalization of syndrome generation. When attempting the detection and correction of multiple errors, this is a more appropriate solution. The *stabilizer code* is a collection of so-called *stabilizer generators*, which are ordered sets of *commuting operators*. The *stabilizer generators* are characterizing the error-correcting code, as the code space is defined by the eigenstates of the *commuting operators*. All the commuting operators must be one of the following 1-qubit unitary transforms: $N = U_{\text{Not}} = U_N, Z = U_Z, Y = U_Y$ (from Equations 2.11, 2.12 and 2.13), and I. Of course, the following properties will also hold: $I^2 = N^2 = Z^2 = Y^2$ and $N \cdot Y = Z, N \cdot Z = Y, Y \cdot Z = N$.

The idea for the generator codes is to transform (with the 1-qubit transformations sets) the Steane-coded state so that each superposed classical state (i.e. eigenstate) is transformed into a different eigenstate, but the overall coded superposition remains unchanged. Therefore, the collection of *stabilizer generators* is the collection of bit-flip and phase-flip generators which assure a complete *non-changing transformation*. If the number of qubits in the encoded block is n and we encode k qubits, then we will need at least $t = n-k$ generators. For our considered example, with just 1 encoded qubit, we need $t = 7 - 1 = 6$ stabilizer generators. One such example of stabilizer generators collection is:

$$\begin{array}{ll} G_0 = (IIZIZZZ) & G_3 = (IININNN) \\ G_1 = (IZIZZZI) & G_4 = (ININNNI) \\ G_2 = (ZZIIZIZ) & G_5 = (NNIININ) \end{array} \tag{2.31}$$

This collection of stabilizer generation, composed out of phase-flip (left column in Equation 2.31) and bit-flip (right column) generators, will not change the overall coded Steane superposition as Figure 2.15 shows.

The error could be easily expressed as a 1-qubit transfromation set, for instance $E = (IINIIIZ)$. In this example, for our qubit block $u_0u_1u_2u_3c_0c_1c_2$ two errors will appear: a bit-flip in u_2 and a phase-flip in c_2. The error vector has an important property of being

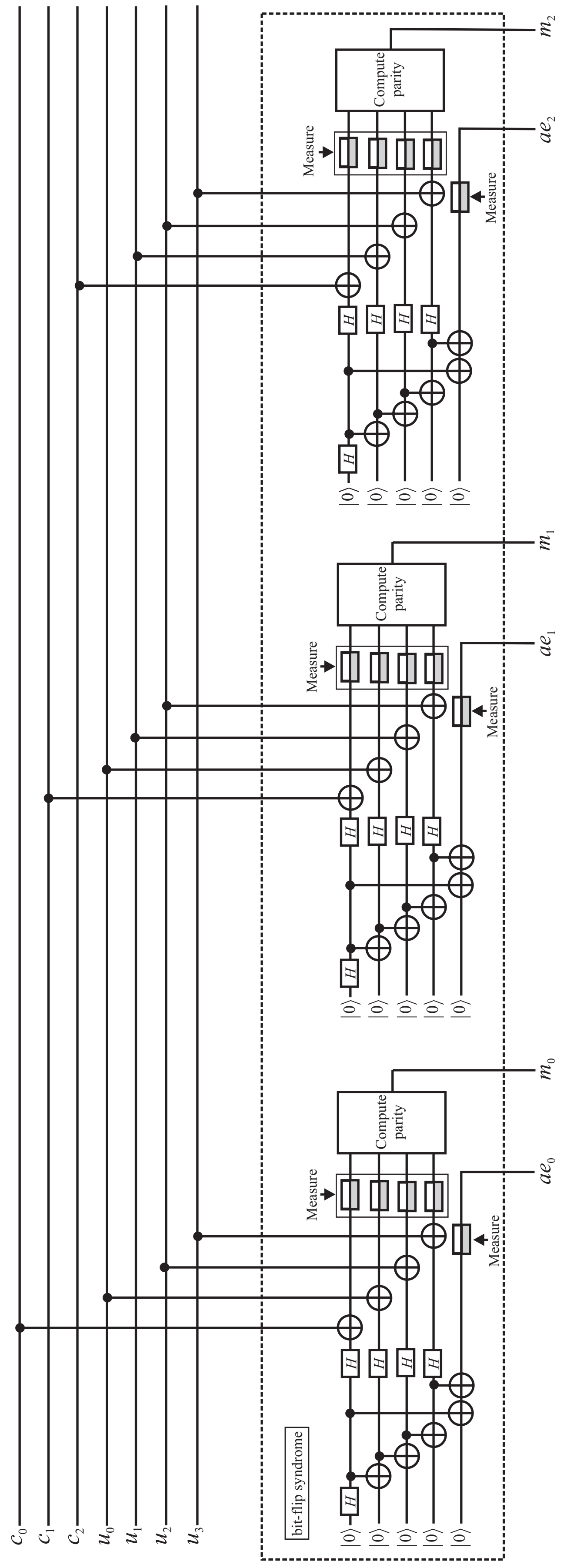


Figure 2.10: Bit-flip syndrome computation block, for Shor ancilla coding.

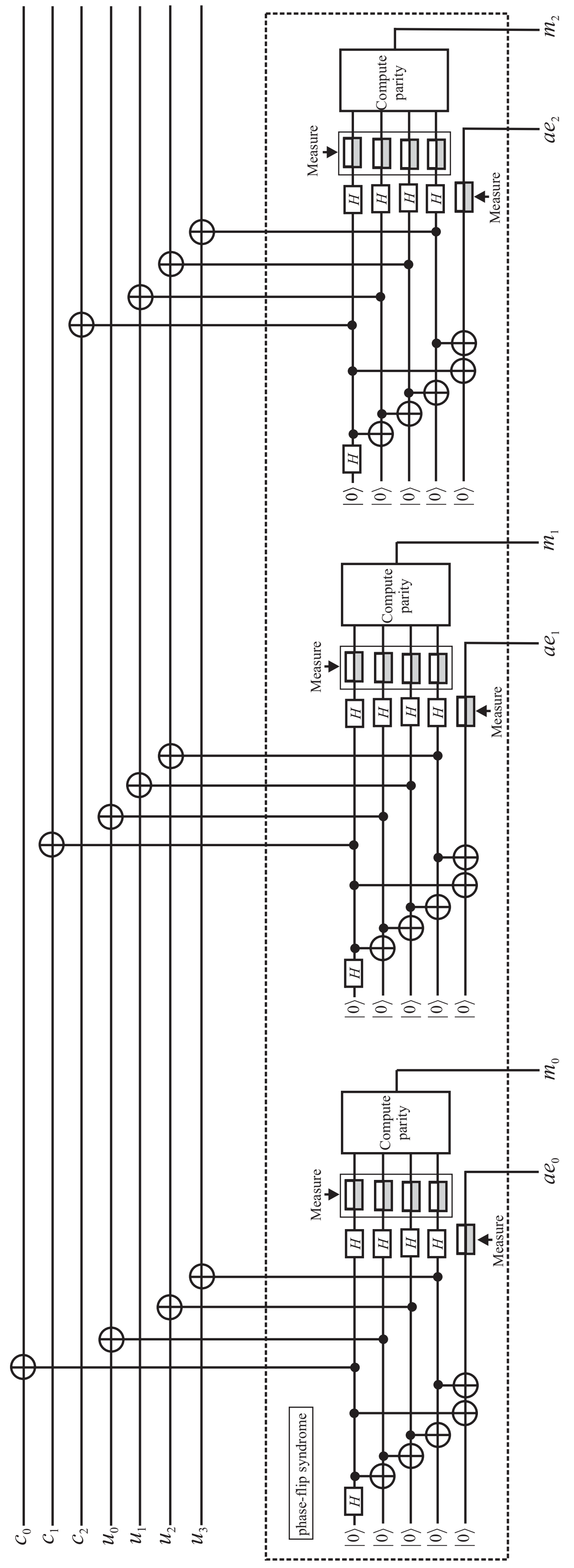


Figure 2.11: Phase-flip syndrome computation block, for Shor ancilla coding.

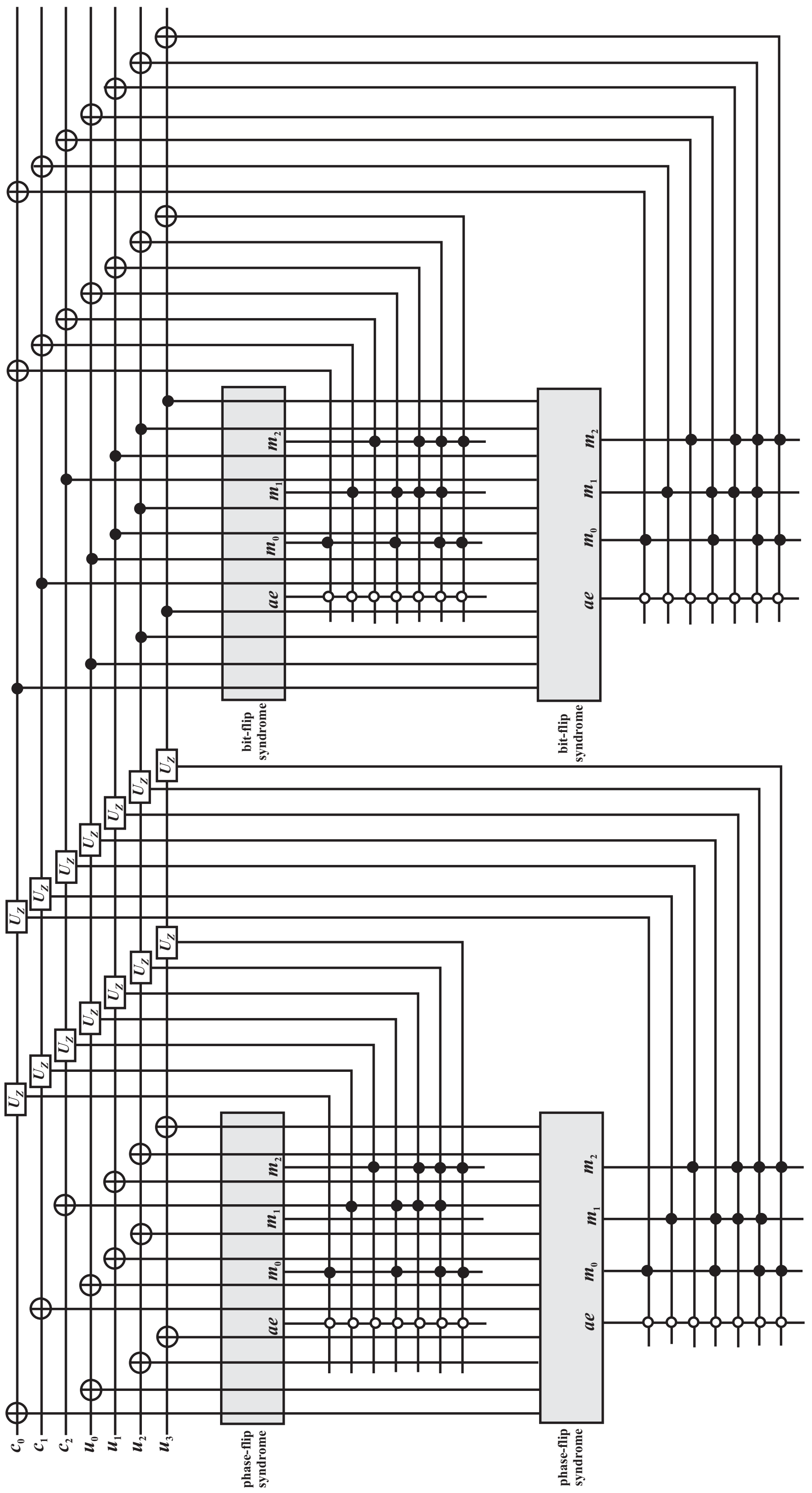


Figure 2.12: Single quantum error correcting circuit with Shor ancilla coding.

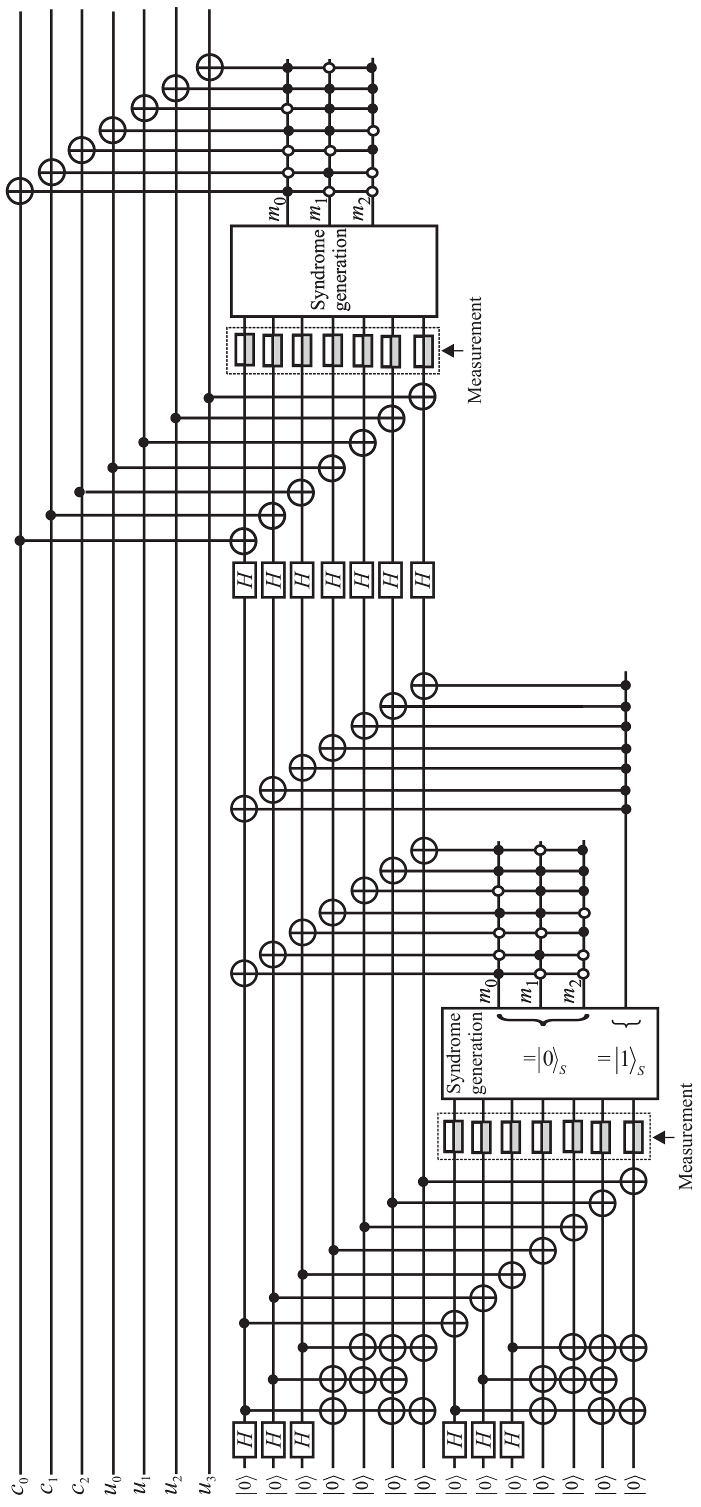


Figure 2.13: Single quantum bit-flip error correcting circuit with Steane ancilla coding.

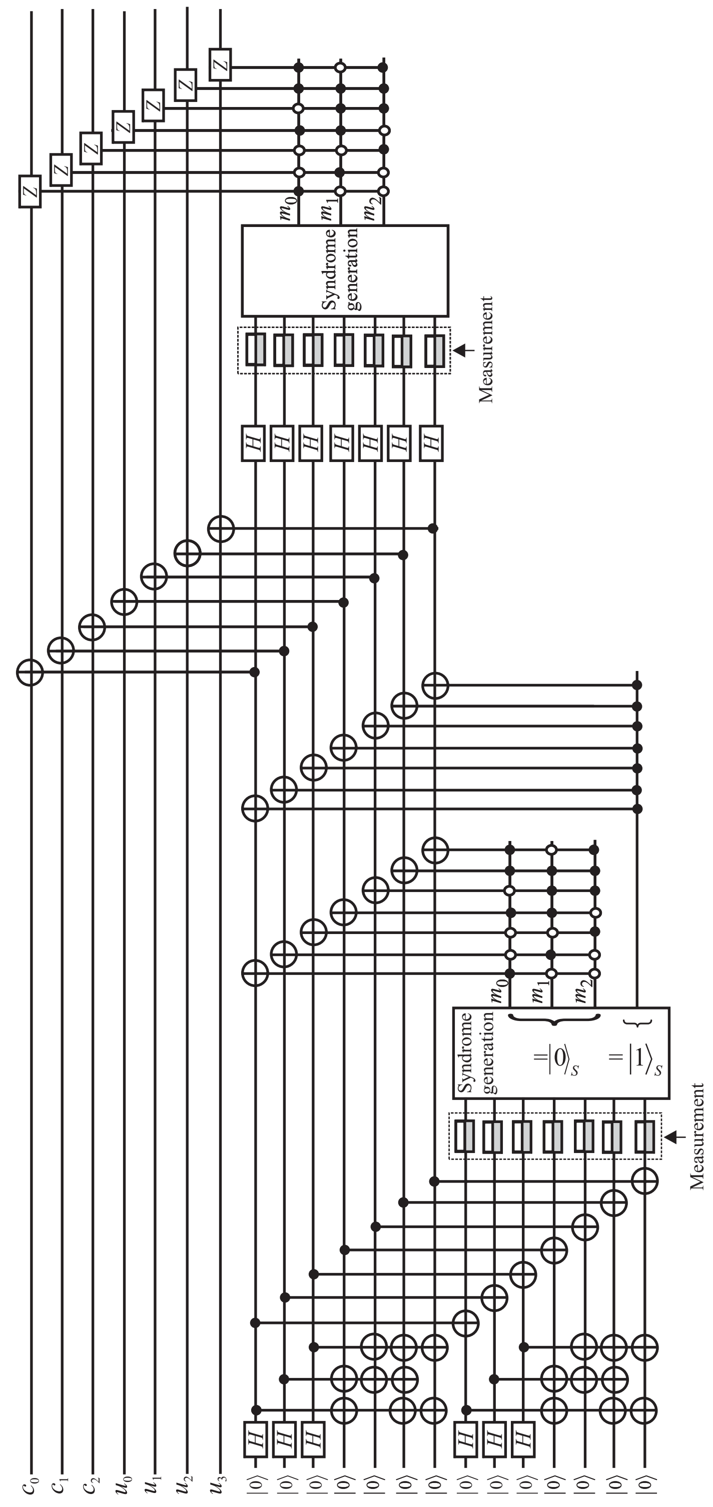
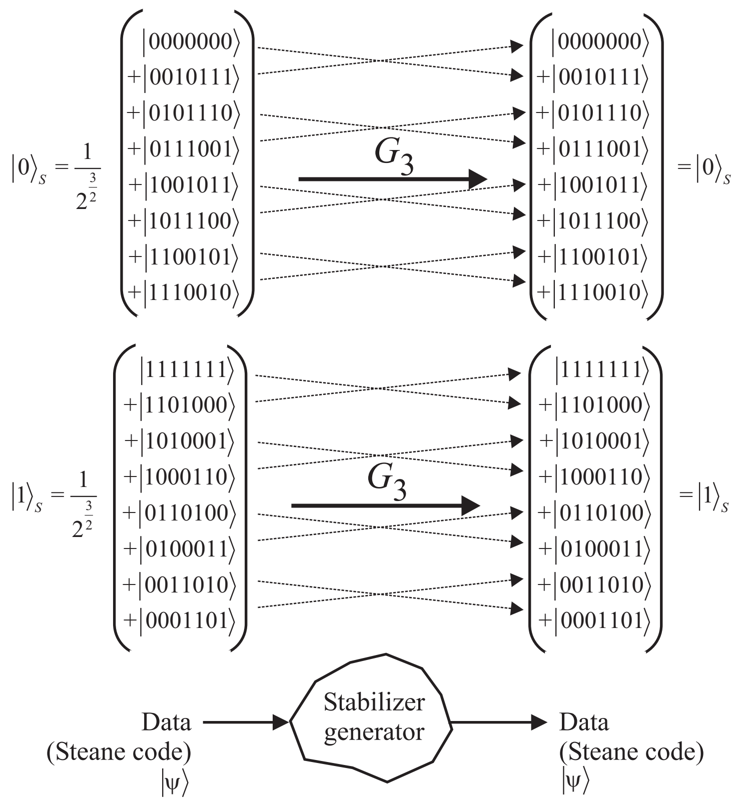


Figure 2.14: Single quantum phase-flip error correcting circuit with Steane ancilla coding.



$|0\rangle_S = \frac{1}{2^{\frac{3}{2}}}$
$|0000000\rangle$ $+|0010111\rangle$ $+|0101110\rangle$ $+|0111001\rangle$ $+|1001011\rangle$ $+|1011100\rangle$ $+|1100101\rangle$ $+|1110010\rangle$
G_3
$= |0\rangle_S$
$|1\rangle_S = \frac{1}{2^{\frac{3}{2}}}$
$|1111111\rangle$ $+|1101000\rangle$ $+|1010001\rangle$ $+|1000110\rangle$ $+|0110100\rangle$ $+|0100011\rangle$ $+|0011010\rangle$ $+|0001101\rangle$
G_3
$= |1\rangle_S$
Data (Steane code) $|\psi\rangle$
Stabilizer generator
Data (Steane code) $|\psi\rangle$

Figure 2.15: Stabilizer generator G_3 is not changing the overall Steane coded superposition.

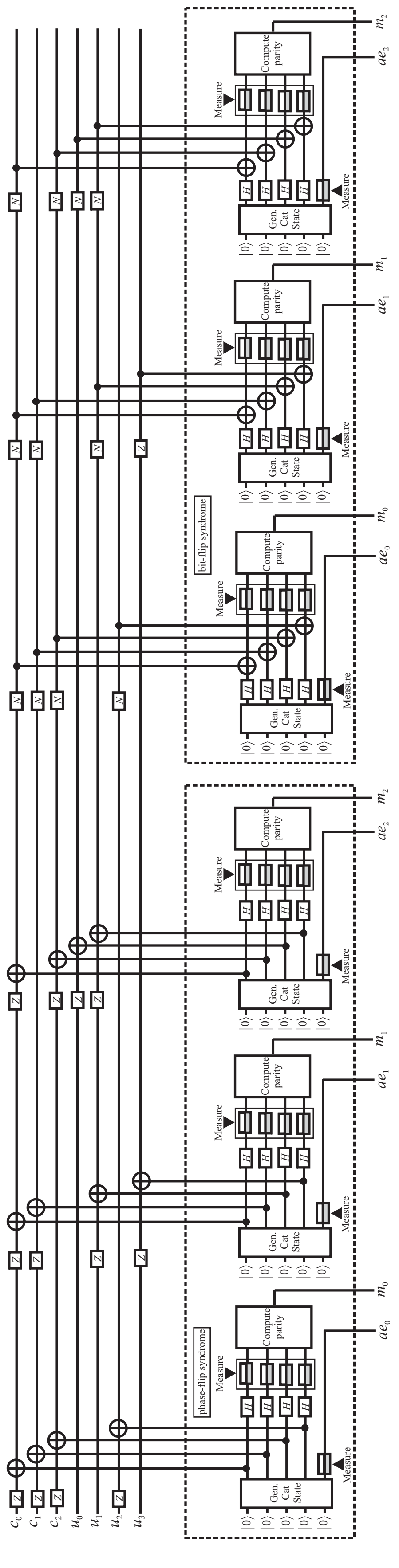


Figure 2.16: Syndrome computation circuit for a stabilizer code.

m_0	m_1	m_2	erroneous qubit
0	0	0	no error
0	0	1	u_0
0	1	0	u_3
0	1	1	u_1
1	0	0	c_2
1	0	1	u_2
1	1	0	c_1
1	1	1	c_0

Table 2.3: Stabilizer code syndrome interpretation; the rightmost row is indicating the bit-flip error position.

revealed by the generators: al least one of them (from the collection) will commute when multiplied with the error vector. This procedure will work for sigle errors and two independent simultaneous errors (i.e. a vector obtained by multiplying two sigle error vectors). In fact, $E = (IINIIIZ) = (IINIIII) \cdot (IIIIIIZ)$.

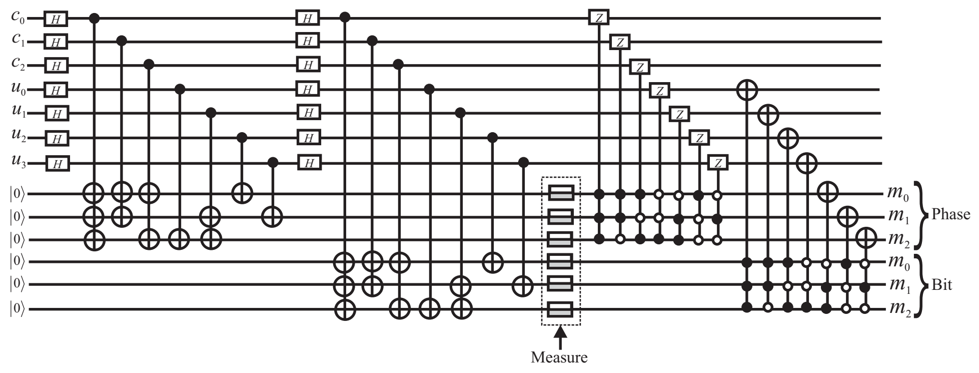


Figure 2.17: Fault tolerant circuit for stabilizer coding, using one qubit for each syndrome bit.

The syndrome computation employs Shor ancilla preparation. The number of qubits of the ancilla equals the number of non-I elements in the generators (4 in our considered example: 1-qubit Steane coding). Each qubit corresponding to such a non-I gate (required by the generator), will be the source of a XOR gate whose target qubit belongs to the ancilla. Then, the ancilla is measured; the syndrome bit is obtained as the parity of the measured ancilla word. Of course, before the phase-flip syndrome is computed the $\{|0\rangle, |1\rangle\}$ is changed with qubitwise Hadamard transforms, and afterwards the original basis is restored by performing the same qubitwise operation. The circuit used for stabilizer code syndrome computation is presented in Figure 2.16. Table 2.3 shows how the error position is found from the syndrome.

We will take into consideration a double error example defined by the set $E_d = (IINIIIZ)$.

Thus, the following equalities will hold:

$$\begin{array}{ll} G_0 \times E_d = 1 & G_3 \times E_d = 1 \\ G_1 \times E_d = 0 & G_4 \times E_d = 0 \\ G_2 \times E_d = 1 & G_5 \times E_d = 0 \end{array} \tag{2.32}$$

Syndromes $(101)_{\text{phase}}$ and $(100)_{\text{bit}}$ indicate that there is a phase-flip error in qubit c_2 and a bit-flip error in u_2.

A better way to measure the generators will take into consideration the possibility of measuring all the generators with just one measurement for bit-flip and phase-flip syndromes. The idea here is to use the proper ancilla, so that only generator eigenvalues are superposed, and therefore only valid generators will be measured without affecting the coherence of our quantum syndromes. For our stabilizer code example, the basic idea is expressed in Figure 2.17. But the presented solution uses ancilla qubits that are targets for multiple XOR gates, so this design is prone to phase (backward propagated) burst errors. The solution would be to use 7-qubit Steane ancilla; a superposition of all the eigenstates dictated by the stabilizer generators. Figure 2.18 presents the design that uses Steane coding for the ancilla, with the Hamming parity being computed according to the generator equivalent matrix.

Although there are many design possibilities, the best fault tolerant approach is due to the generalization of stabilizer coding. The starting point in formalizing stabilizer coding is to associate a Hamming matrix to the generator set. For instance, generators G_3, G_4, G_5 from Equation 2.31 are characterized by the following Hamming matrix:

$$H_{\text{Stab}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix} \tag{2.33}$$

Of course, for the Steane code Hamming matrix (H_A from Equation 2.14) we have a generator collection:

$$\begin{array}{ll} GS_0 = (ZIZZZII) & GS_3 = (NINNNII) \\ GS_1 = (ZZZIIZI) & GS_4 = (NNNIINI) \\ GS_2 = (IZZZIIZ) & GS_5 = (INNNIIN) \end{array} \tag{2.34}$$

In order to generalize these partial correspondences, we will use the so-called *stabilizer code check matrices.* The check matrix is formed by a left half – that is corresponding to the phase-flip control –, and a right half which is corresponding to the bit flip control: $\tilde{H} = (H_Z|H_N)$. The new matrix will have a double number of lines, thus the matrix size is $2n$ columns and $n-k$ lines. For the generator collection in Equation 2.34 the check matrix is:

$$\tilde{H}_A = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \tag{2.35}$$

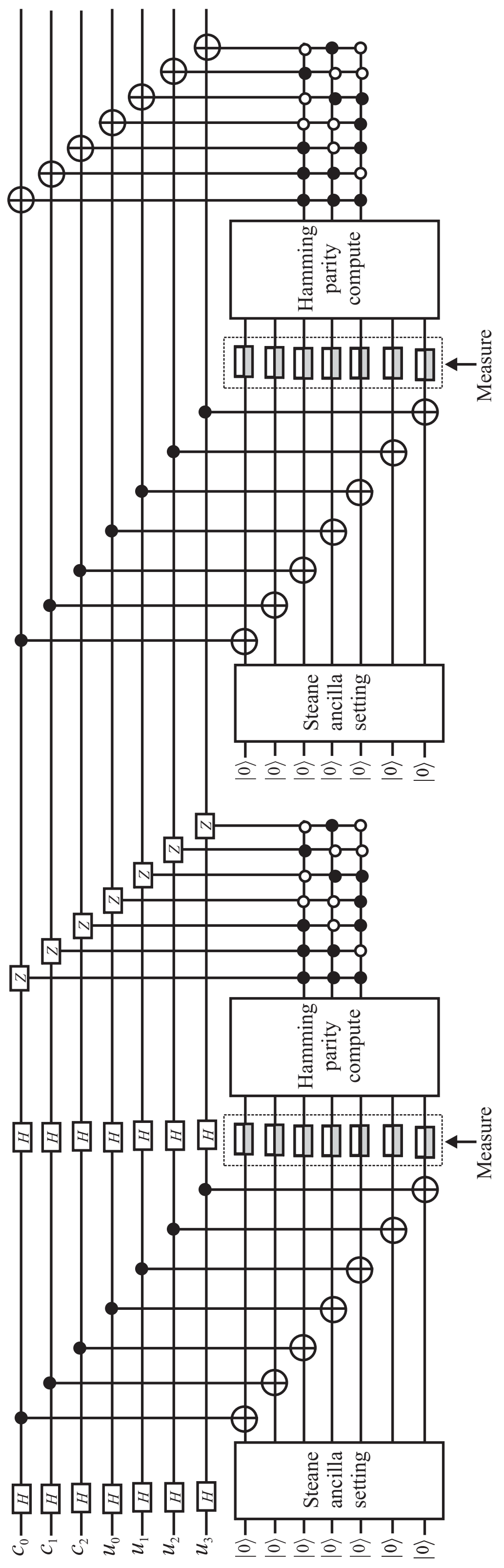


Figure 2.18: Stabilizer generator measurement with Steane ancilla coding.

and for the generators in Equation 2.31 we will have

$$\tilde{H}_{\text{Stab}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix} \tag{2.36}$$

The check matrices are built as follows:

- for each qubit which has a Z in the generator there will be a corresponding '1', the same thing is true for the generators with N elements;
- if an qubit is operated by an Y element, then a '1' will appear on the same position in H_Z and H_N;
- the rows correspond to the ordered list of generators ($G_0, G_1, \dots G_5$ in our generator code example).

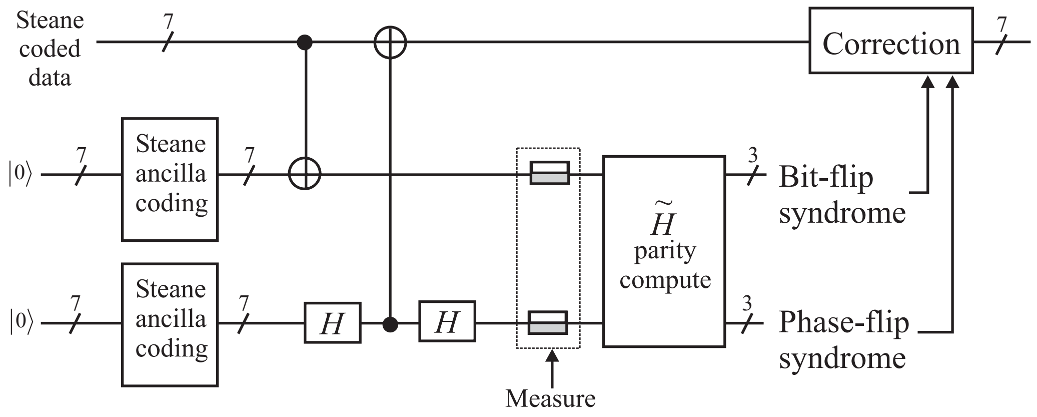


Figure 2.19: Error-correction with stabilizer generator measurement, Steane ancilla, and syndrome computation according to the check matix.

Having the $\tilde{H}$ check matrix generalization theory, better generator measuring devices can be developed, as Figure 2.19 shows. The preparation of the ancilla assures the fact that, excepting the syndrome, no other eigenstate is superposed in the quantum measured state. Thus, operating with only the classical values of the measured ancilla, the affected qubit is identified by computing the parity according to $\tilde{H}$.

2.6 Gates for fault-tolerant codes

We have described the methodologies that are employed in order to detect and correct errors affecting the quantum state. However, we must be able to process the encoded states in order to build fully functional quantum circuits. Therefore special (i.e. fault-tolerant) gates must be designed; these gates will perform the same tasks as the ordinary gates, but the processed states will be the fault-tolerant coded.

There are two ways of approaching the implementation of such quantum gates:

- by decoding the state, applying the ordinary gate, and re-encoding the processed state;
- by processing the encoded state with special designed transformations.

The first approach is not feasible because, although simplifies the operation, it is introducing the decoding and encoding operations which are prone to errors. Hence, we adopt the second approach by designing a universal set of quantum gates that are processing the encoded states (Steane code or any stabilizer code). The gates from the universal set are chosen so that as many as possible are straightforwardly implemented. It is easy to implement *XOR*, *NOT*, *Hadamard*, and *Phase Shift* ($P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$) gates by performing qubitwise ordinary transformations: XOR, N, H, and $P^{\dagger} = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$ respectively (see Figure 2.20).

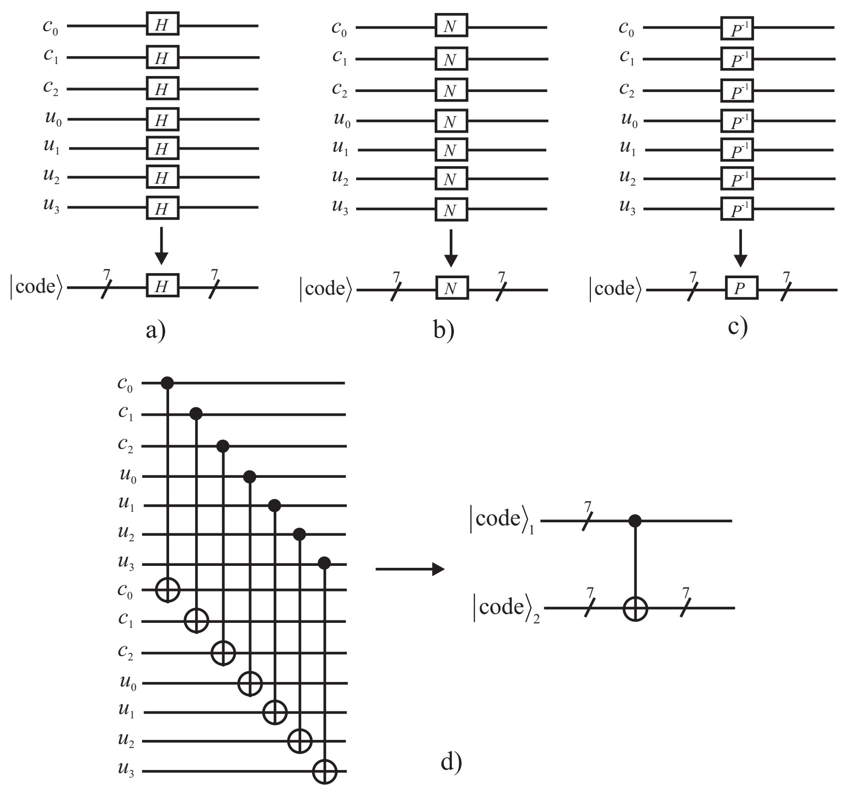


Figure 2.20: Straightforward implementation of some quantum gates operating on fault-tolerant coded qubits: a) *Hadamard*; b) *NOT*; c) *Phase Shift*; d) *XOR*.

In order to have an universal set of fault-tolerant codes operating gates, we must define the Toffoli gate in these 7-qubit code terms. This objective is not achieved in a straightforward manner, it needs to define special ancilla qubits preparation , followed by some operations applied on the processed coded qubits.

The ancilla qubits state will have to be of the form given in the following equation:

$$|\psi\rangle_{\text{code}} \begin{array}{l} = \frac{1}{2}\sum_{x,y\in\{0,1\}} |x,y,x\wedge y\rangle_{\text{code}} \\ = \frac{1}{2}\left(|000\rangle_{\text{code}} + |010\rangle_{\text{code}} + |100\rangle_{\text{code}} + |111\rangle_{\text{code}}\right). \end{array} \tag{2.37}$$

This state is obtained by setting the innitial state:

$$H^{\otimes 3}|000\rangle = \frac{1}{\sqrt{2}}\left(|\psi\rangle_{\text{code}} + |\phi\rangle_{\text{code}}\right). \tag{2.38}$$

where the relationship between $|\psi\rangle_{\text{code}}$ and $|\phi\rangle_{\text{code}}$ is

$$I \otimes I \otimes N|\phi\rangle_{\text{code}} = |\psi\rangle_{\text{code}} \tag{2.39}$$

Therefore, the state superposition from Equation 2.38 could be seen as a vector in a space with basis $\left\{|\psi\rangle_{\text{code}}, |\phi\rangle_{\text{code}}\right\}$ and, obtaining the desired $|\psi\rangle_{\text{code}}$ state could be reduced to measuring the superposition state in this new basis. If the result of the measurement is $|\phi\rangle_{\text{code}}$, then the $I \otimes I \otimes N$ operator is applied on the measurement outcome.

The procedure for measuring in the $\left\{|\psi\rangle_{\text{code}}, |\phi\rangle_{\text{code}}\right\}$ basis is described in Figure 2.21, where the $Z_{\psi\phi}$ gate is a conditional phase shift in the new space basis. The actual implementation in the fault-tolerant Toffoli gate construction, which uses $|\text{cat}\rangle$ states for measuring the ancilla in the corresponding basis is presented in the left side of Figure 2.22.

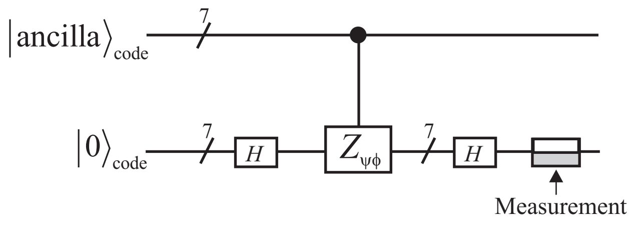


Figure 2.21: The principle used for measuring the ancilla qubits in the $\{|\psi\rangle, |\phi\rangle\}$ basis.

If measuring the 'cat' states will give $|m_1\rangle_{\text{code}} = |0000000\rangle^{\otimes 3}$ then the ancilla is rightly set, otherwise we will negate the least significative ancilla 7-qubit code.

Having the ancilla set, 4 gates are applied on the processed qubits $|x_1\rangle_{\text{code}}$,$|x_2\rangle_{\text{code}}$, $|y\rangle_{\text{code}}$, 3 XORs and 1 Hadamard. The first 2 XORs have ancilla sources and x_1, x_2 qubits as targets, then the last XOR has ancilla code as target with the y qubit as source. Then, the Hadamard gate is applied on the y qubit, as presented in the center of Figure 2.22. The effect of applying these 4 gates, on the processed qubits and the ancilla, is presented in Equation 2.40.

$$\begin{array}{l} |x_1, x_2, y\rangle_{\text{code}} \sum_{a_1,a_2\in\{0,1\}} |a_1, a_2, a_1 \wedge a_2\rangle_{\text{code}} \longmapsto \\ \sum_{a_1,a_2,a_3\in\{0,1\}} (-1)^{a_3 \cdot y} |x_1 \oplus a_1, x_2 \oplus a_2, a_3\rangle_{\text{code}} |a_1, a_2, (a_1 \wedge a_2) \oplus y\rangle_{\text{code}}. \end{array} \tag{2.40}$$

The result of the computation is obtained on the ancilla qubits by measuring the processed qubits, corresponding to x_1, x_2, and y. If the result is $|m_2\rangle = |000\rangle_{\text{code}}$, then $x_1 = a_1$ and

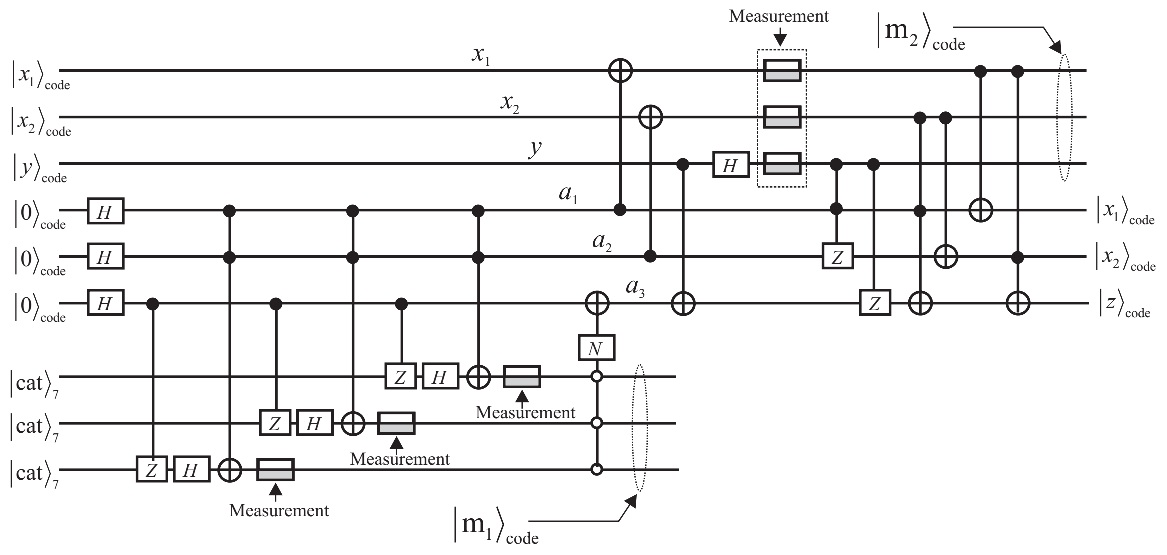


Figure 2.22: The implementation of the fault-tolerant Toffoli gate.

$x_2 = a_2$, therefore the ancila qubits will contain the result. Otherwise, as shown in Figure 2.22, some corresponding gates will have to be applied on the ancilla, for each non $|0\rangle_{\text{code}}$, in order to correct the result.

2.6.1 Gottesman's fault tolerant gates

Although the above-described set of universal set of fault tolerant gates suffices from a theoretical point of view, a much more general approach would be useful for the future engineering issues. Shor [47] has marked the way to the fault tolerant gate generalization, but Gottesman [18][19] gave its exact definition, which can also be found in [39].

We have already presented that for any stabilizer code there is a fault tolerant implementation of the 1-qubit gates N and Z (i.e. the gates used for processing each qubit in the stabilizer code). Also, any stabilizer operator can be expressed with a $\tilde{H} = (H_Z|H_N)$ check matrix having $2n$ columns and $n-k$ lines, with n being the code block size, k the number of encoded useful (i.e. non-redundant) qubits, and $n-k$ the number of operators that are required to generate the stabilizer code.

The error operators that will commute with the stabilizer code generators are forming a group denoted G_E. This group, as a whole, has its own generator set. The original generators set are represented as $2n$-length binary strings, which will have to satisfy $n-k$ independent binary logic equations. Therefore, the error operator group G_E will have a number of $n+k$ independent generators, including the $n-k$ stabilizer code generators. Thus we have an extra-set of $2k$ operators, able to process the coded state (i.e. altering it in a non-trivial fashion) without deteriorating the consistency of the code subspace. All the operators from the extra-set are independent from the stabilizer code generators [10], hence could be used as ordinary unitary operators on the encoded qubits.

Gottesman [18] has shown that we can choose the $2k$ operators so that they are only

1-qubit acting $\left(\hat{Z}_i, \hat{N}_i : i = \overline{1,k}\right)$, and that all of these are fault-tolerant. Because it was demonstrated that a fault tolerant Toffoli gate can be constructed in order to act on qubits encoded by any stabilizer code, we have a comprehensive (and universal) set of fault tolerant gates: $\left(\text{Toffoli}_{\text{FT}}, \hat{Z}_i, \hat{N}_i\right)$. The gates corresponding to $\hat{Z}_i$ will flip the phase of qubit i, while $\hat{Z}_i$ is flipping qubit i. All these operators are complying with the properties described in Equation 2.41.

$$\begin{aligned}
&\left[\hat{Z}_i, \hat{Z}_j\right] = \left[\hat{Z}_i, \hat{Z}_j\right] = 0 \\
&\left[\hat{Z}_i, \hat{N}_j\right] = 0 \text{ if } i \neq j \\
&\hat{Z}_i\hat{N}_i + \hat{N}_i\hat{Z}_i = 0
\end{aligned} \tag{2.41}$$

As incentive for a favourable implementation of the previously defined gates, the 7-qubit Steane code (a particular case of stabilizer coding) is considered as the most convenient choice [39]. Other stabilizer codes may require smaller blocks but the computation will become more complex.

2.7 Accuracy threshold

2.7.1 Technological requirements

Quantum error-correcting codes exist for r errors, $r \in \mathbb{N}, r \geq 1$. When using such a code, a *non-correctable error* occur if a number $\geq r+1$ errors appear in the coded block *at the same time* (i.e. before the recovery process).

If the probability of an *quantum gate error* or *storage error* in the time unit is of order ξ, then the probability of an error affecting the processed data will be of order ξ^{r+1} (a very small quantity if r is sufficiently big). It seems that a coding procedure corresponding to a big r will solve our quantum reliability problem. But this issue is in fact more complex than that. Increasing r means that the safe recovery will become more complex, hence prone to error: with a high probability $r+1$ errors will accumulate in the block buffer before recovery is performed.

Suppose the relationship between r and the number of computational steps required for obtaining the syndrome is polynomial of the order r^p. Then, the block error probability (i.e. the probability of accumulating more than r errors in the coded block before performing recovery) is given by Equation 2.42 [39].

$$BlockErrorProbability \sim \left(\xi r^p\right)^{r+1} \tag{2.42}$$

It was proven [18][39] that, in order to reduce as much as possible the error probability, r must be chosen so that

$$r \sim \mathrm{e}^{-1}\xi^{-\frac{1}{p}} \tag{2.43}$$

As a consequence, we are able to evaluate the degree of accuracy (as error probability in the time unit), by minimizing the *BlockErrorProbability*. In other words, if attempting to execute N cycles of error correction without any $r+1$ error accumulating before the recovery steps, then we must have

$$N \sim \exp\left(\xi^{-\frac{1}{p}}\right) \tag{2.44}$$

Thus, the accuracy degree will have to be

$$\xi \sim (\log N)^{-p} \tag{2.45}$$

Nevertheless Equation 2.45 looks better than the accuracy degree corresponding to the no-coding case:

$$\xi \sim N^{-1} \tag{2.46}$$

Still, we will have an $N_{\max}$ so that if $N > N_{\max}$ then the $> r$ error (non-correctable) is likely. It is obvious that, in these conditions, we have a limit on how long the fault-tolerant computation is. Although the encoding techniques already discussed seem to suffice because $N_{\max}$ is big enough, one must also notice the extremely high number of gates employed by quantum algorithm implementation; therefore $N_{\max}$ must higher than $3 \cdot 10^9$ for Shor's algorithm.

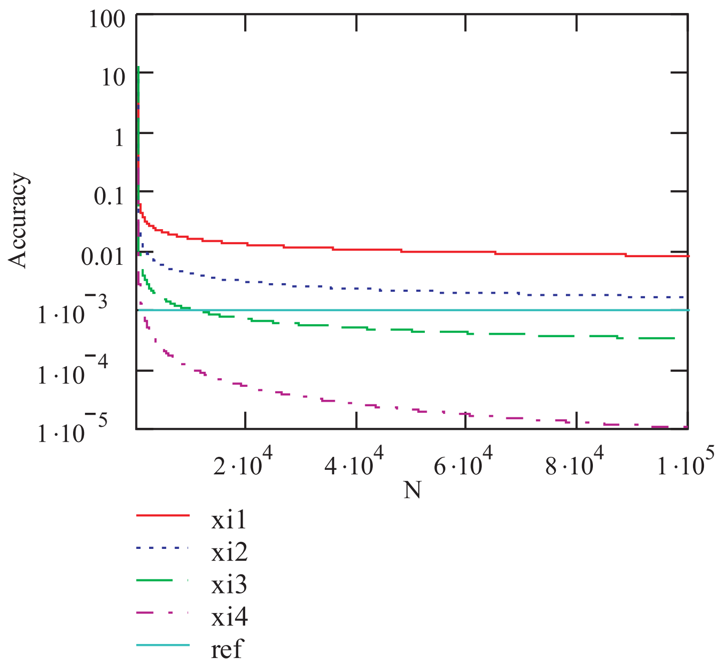


Figure 2.23: Graphical representation of accuracy degree required for the corresponding N, for different p's: 3 for xi_1, 4 for xi_2, 5 for xi_3. xi_4 corresponds to the no-coding situation, while ref is the reference accuracy (i.e. the accuracy allowed by today's state of the art technology).

As Figure 2.23 presents, the required accuracy degree approaches today's technological limit (tipically 10^{-3} for $p = 4$) after $N = 10^5$. For a fault tolerant encoding solution for Shor algorithm implementation this should have happened after $N = 10^9$.

2.7.2 Concatenated coding

A solution for fault tolerant quantum computation with arbitrary length is concatenated coding. It uses a divide-and-conquer strategy where each qubit is encoded by a block of sub-qubits, each sub-qubit being encoded by other blocks of qubits, and so on. Figure 2.24 explains the principle of concatenated coding. If we are using – for instance – Steane codes, then each qubit is encoded by 7 sub-qubits, and therefore a m-depth (or level) concatenated Steane code will require a 7^m-qubit block.

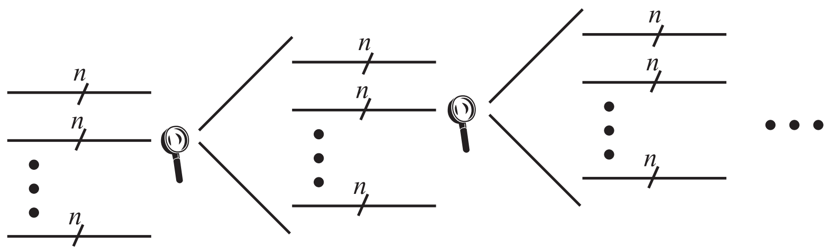


Figure 2.24: Concatenated coding: each qubit can be encoded by a block of sub-qubits.

But will concatenated coding necessarily bring more fault tolerance? This solution seems to be two-way because, on one hand it can make things better if the initial error rate is sufficiently small, however – on the other hand – if the error rates are high, then things will become worse (fault tolerance degradation).

In this chapter we considered an error model that excludes correlated errors (in time or space). This model is very good for strategic considerations in fault tolerance, but correlated errors must not be ignored because they are very likely as long as we can have 2 or 3-qubit faulty gates, inflicting errors on more than one qubit.

For Steanes code the recovery is intended for singular error and, given a fault tolerant recovery with uncorrelated errors appearing with probability ε [errors/qubit], concatenated coding with m levels brings an ε^{2^m} overall error probability at the expense of using a block of size 7^m-qubit. An interesting analysis would be the solution to the following question: how big the block size must be in order to assure fault-tolerant quantum computation of N steps, when the accuracy degree is fixed?

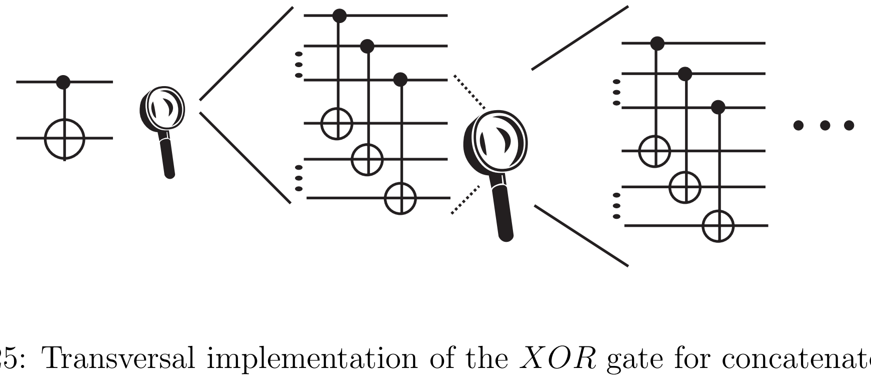


Figure 2.25: Transversal implementation of the XOR gate for concatenated coding.

2.7.3 Analysis

In order to start the analysis required by the above considerations, we have to adopt the starting point premises from the below list, which assure both a simpler analysis and a worst-case analysis type.

- The errors are probabilistic uncorrelated, with all the error models (bit-flip, phase-flip, bit-and-phase-flip) having the same likelihood;
- Total probability in each time slot, between two consecutive fault recoveries, is ξ_{store},with the probability of a gate operation failure being ξ_{gate};
- The analysis is performed by building a "set of concatenation flow equations" [38];
- Steane encoding is used, with Steane syndrome measurement, because is appropriate for a concatenated coding approach: contains Hadamard and XOR gates, which are easy to implement transversally (see Figure 2.25 for a XOR gate example);
- In order to compute the syndrome, distructive measurement can be executed at all the levels in the concatenated code and at the same time [39][63]; this way the recovery will be performed to all the elementary (lower level) qubits and at the same time recovery is completed at the highest level in one step. In other words, with this procedure all the levels of concatenation will execute recovery at the same time.

The idea for *accuracy threshold estimation* is that at each level of concatenation a block of 7's fails if there are at least 2 errors in its subblocks. If p_m is the probability of block error at the m level, the probability of error at the $m+1$ (higher) level will be p_{m+1}, given by

$$p_{m+1} = \sum_{i=2}^{7} \mathrm{C}_7^i p_m^i \sim 21 p_m^2 \tag{2.47}$$

and therefore we consider that, because $p_{m+1} < p_m$ if $p_m < \frac{1}{21}$, at the lower level (0) we must have $p_0 < \frac{1}{21}$ so that level 1 will have a smaller error probability. We have found the threshold value: $\frac{1}{21}$.

As reference for our further analysis we are taking into considerations Figures 2.13 and 2.14, as Steane code with Steane ancilla implementation. By using the threshold value for the circuit described in these two figures, Preskill [39] has calculated the elementary qubit probabilities that will preserve the fault tolerant computation:

$$\xi_{store,0} \sim 6 \cdot 10^{-4} \tag{2.48}$$

$$\xi_{gate,0} \sim 6 \cdot 10^{-4} \tag{2.49}$$

Preskill got these values by assuming that we have only either gate or store errors. These results are much debated and there is a lot of controversy regarding the procedures and premises that are to be taken into consideration in order to perform the analysis. However, the bottom line is that all the computed thresholds are somewhere around 10^{-4}.

Another problem was also addressed by Preskill [39]: how large must be the block size in order to ensure the desired accuracy degree ($\xi_{gate,0}$)? For a computation involving N gates, and an actual gate error rate of ξ the block size must comply to:

$$blocksize \sim \left[\frac{\log \xi_{gate,0} N}{\log \frac{\xi_{gate,0}}{\xi}}\right]^{\log_2 7} \tag{2.50}$$

Chapter 3

Improving Fault Tolerance in Quantum Circuits

This chapter presents a radically new solution for attaining fault tolerance in quantum circuits by employing reconfiguration techniques. It begins with an analysis that provides the necessary critique of the actual quantum fault tolerance techniques, therefore identifying the weaknesses of those methods, and then sketches the solution based on the so-called *reconfigurable Quantum Hardware* or *rQHW*.

The proposed technique, applied on a specific case shows drastic improvement of the accuracy threshold, as an availability (as defined by Avižienis *et. al.* [6]) measure.

3.1 The big picture

In a classical computer system, there are several fault tolerant techniques that are employed on different abstraction levels. This section starts by presenting tables with the techniques that are used on each abstraction level (inspired by classical hardware synthesis [5]), for both classical and quantum computing systems (see tables 3.1 and 3.1).

In quantum computation we have just 4 abstraction levels; in table 3.1 on the "level" column we show the analogy correspondence between the classical hardware and the actual quantum levels. On the *architectural* level we deal with quantum computing systems: quantum computers or coprocessors [33][55] where some architectural strategies can be used [35]. At the *circuit* level the testing techniques are applied; for example, Shors algorithm arithmetic circuits are implemented with strategies like *carry-dependent sum*, which provides for parity fault detection [36]. The *unitary* level is concerned with the special coding that is thoroughly discussed throughout this chapter, and the intrinsic reliability at the *particle* level is provided by sophisticated technology like *topological quantum computation* [1].

The so-called "big picture" in fault tolerant quantum computation is provided by Figure 3.1, which shows the different abstraction levels from the top (architectural – quantum computer) to the bottom level, which has the highest detail degree (technology – particle). Also, in Figure 3.1 the associated faut-tolerant strategies are presented.

This is our engineering general interpretation of fault tolerant strategy implementations on different quantum hardware abstraction levels. The abstraction levels were also inspired from classical hardware design (Gajsky and Kuhns Y diagram [5]). It is also based on what

level	behavioral domain	fault tolerance	example components
system	performance attributes	fault-tolerant architecture	computers computer networks
chip	algorithms	testing (BIST) FT reconfiguration	μP, FPGA, DSP, UART
register	data flow	Embryonics BILBO testable design	register ALU MUX
gate	boolean equations	EDC-ECC testing	gates flip-flops
circuit	differential equations	manufacturing testing post-fabrication reconf.	MOS tranzistors bipolar technology
silicon	–	manufacturing verif. intrinsic reliability	geometrical forms

Table 3.1: Hierarchical fault tolerance approach in classical computer hardware.

level	behavioral domain	fault tolerance	example components
architectural { system, chip	complexity measures	reliable architecture [35]	quantum oracle [55]
circuit { register	qustate flows, bra-ket equations	reliable quantum arithmetic devices	quantum ALU, Grover iteration circuit
unitary { gate	matrix operations	Steane coding, ancilla preparation	qubits, quantum gates
particle { circuit, silicon	Schrödinger equations	intrinsic fault tolerance	topological quantum computation Aharonov Böhm [1]

Table 3.2: Hierarchical fault tolerance interpretation for quantum computational devices. The 'level' column also presents the correspondence with classical hardware levels.

is already achieved in this field. The researchers have approached all the invoked strategies, including (rather recently) the testable circuits [40].

This section will also explore some of the limitations of the principles that are applied in quantum error detection and correction. As already presented in this chapter's first section, classical computation is intrinsically fault tolerant, and the encoding techniques are effective due to the fact that an error is unlikely to occur, thus facilitating a safe recovery. But in quantum computation the circuits are prone to fail, and safe recovery is improbable. Therefore, a simple classically inspired solution is not feasible because ancillary qubits preparation will give the overall fail rate, regardless of the number of ancillary levels. Figure 3.2 shows that data and ancilla qubits having a error rate of the order ξ will always give an overall ξ fail rate when a classical approach is used and no matter how many ancilla qubit level are used for a safe recovery.

The only conceivable solution to the safe recovery problem is to use structural redundancy. If we prepare n_a ancilla qubit sets, each with the corresponding syndrome, and the trustworthy syndrome is selected with a voter, then the probability of a faulty syndrome becomes $\xi^{\frac{n_a}{2}}$ (see Figure 3.3). This is a good result for a sufficiently small ξ and a big n_a. However, a big n_a means that a lot of ancillary qubits are used and, although fault tolerance is a very important issue, we want to spare as much as possible qubits.

As already stated in this chapter, our objective is to assure an ξ^2 error probability for the corrected encoded data. The method that is actually used is based on the voter principle presented in Figure 3.3, but uses a limited structural redundancy. The Steane safe recovery procedure is based on Steane ancilla preparation, which provides for effective syndrome testing. If one syndrome is not good, then another is available. The probability of both failing is ξ^2 and therefore this procedure, schematically presented in Figure 3.4, is sufficient for achieving the stated goal. The procedure must be quite simple in order to maintain a limited number of involved qubits, thus limiting the area of computation that is prone to error.

3.2 Issues to be settled

The theory of fault tolerant quantum computation has evolved and makes the quantum processors feasible [31][32]. But still there are some potential problems of theoretical nature that could affect the future quantum hardware engineering issues. Here we will debate two of these problems:

(α the fact that the *error occurrence model* that was taken into consideration is of uncorrelated errors;

(β the inflexibility of quantum circuits for ancilla preparation which requires at least two ancilla sets to be used even if the syndrome computed on the first set is correct.

Both problems would indicate reconfigurable quantum hardware as solution, even though such a solution does not operate as we would expect from a classical hardware point of view [30]. Nevertheless, we will thoroughly analyze now the above listed problems, which could have reconfigurable solutions.

When discussing problem (α, we observe that, for a theoretical approach, it is convenient to consider an error occurrence model that excludes correlated errors (in time or space). But

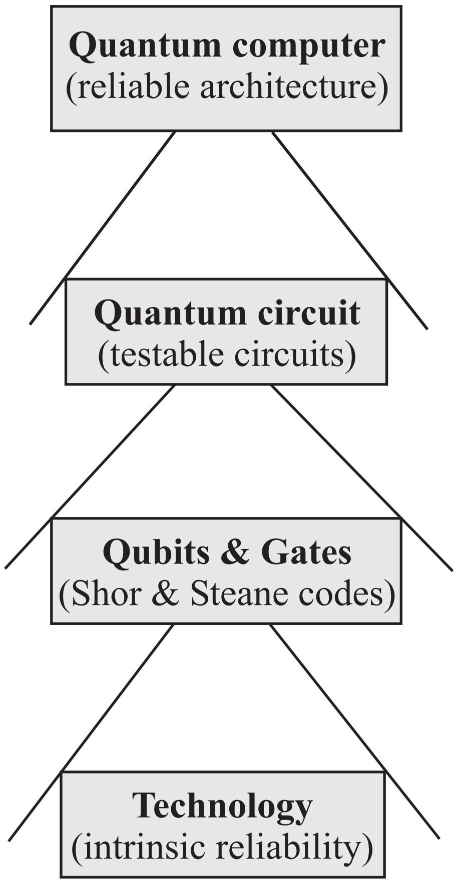


Figure 3.1: Quantum fault tolerant achievements on different abstraction levels.

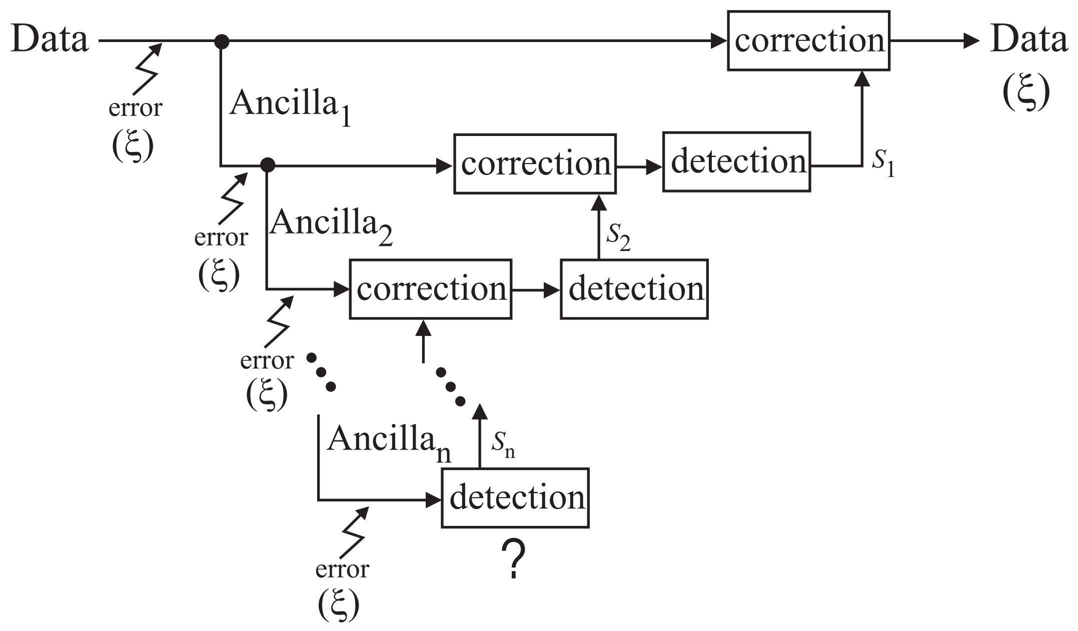


Figure 3.2: Classical fault tolerant approach for a safe recovery assumption. Here, if the probability of faulty recovery is of the same order as the probability of faulty data, then coding is useless because the probability of erroneous corrected data remains ξ.

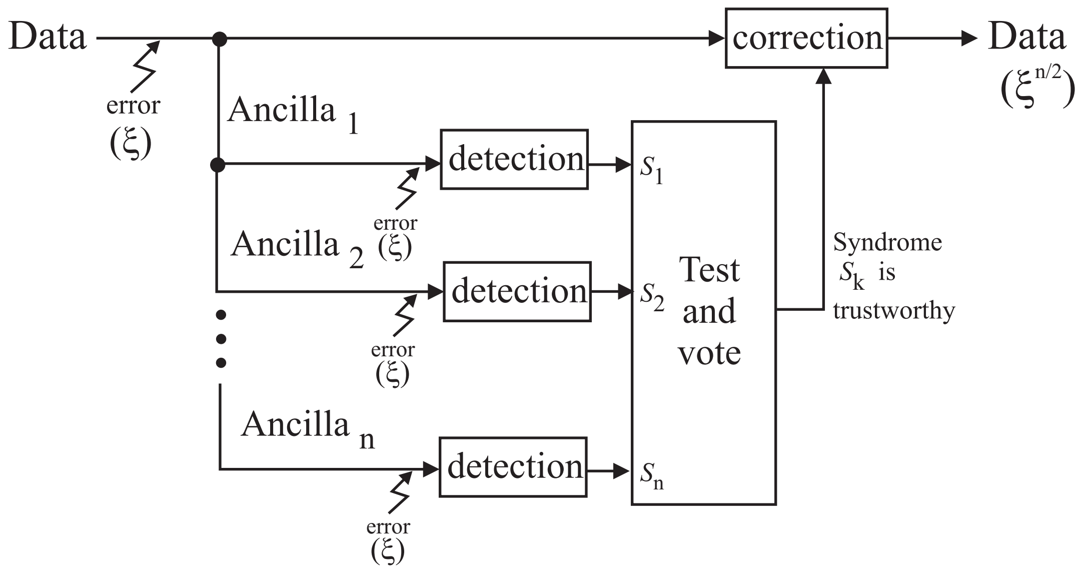


Figure 3.3: Using structural redundancy and voting of a trustworthy ancilla provided for a lower error probility ($\xi^{\frac{n}{2}}$).

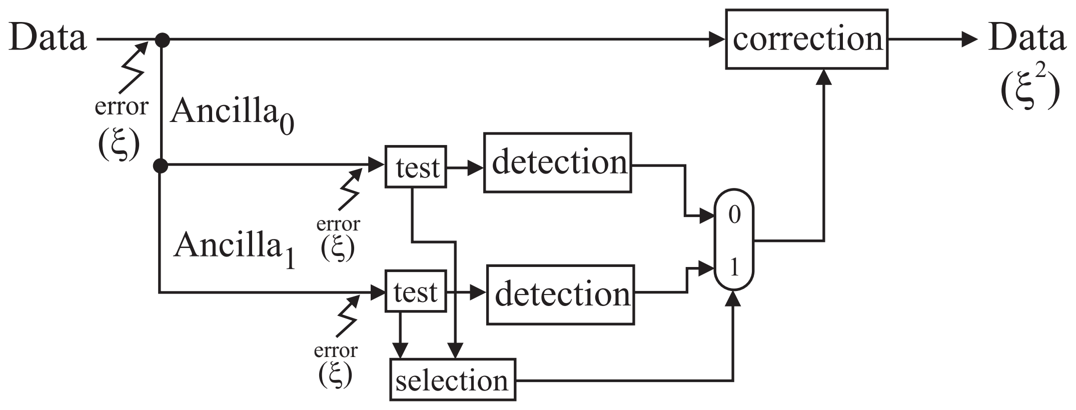


Figure 3.4: Steane safe recovery procedure. Corrected data has an error probability of order ξ^2 by using structural redundancy when testing reveals bad ancilla.

it would be unrealistic to consider such a model in practice. On the other hand, dealing with this kind of errors is rather difficult. Nevertheless, the biggest problem that comes from correlated errors is that concatenated code blocks are jeopardised. For example, if we use a concatenated code of size 7 on 3 levels, we have a total of $7^3 = 343$ qubits. We also consider that only one error/block is tolerable. Thus, when 5 low-level qubits (from 343) are erroneous, the probability of not being able to correct the entire code is very low. But when these errors are correlated in space, there is very likely to have a code that cannot be corrected (a block with at least 2 erroneous sub-blocks). Figure 3.5 presents such an example situation, where the integrity of the entire code is affected due to the correlated nature of errors in the low-level qubits.

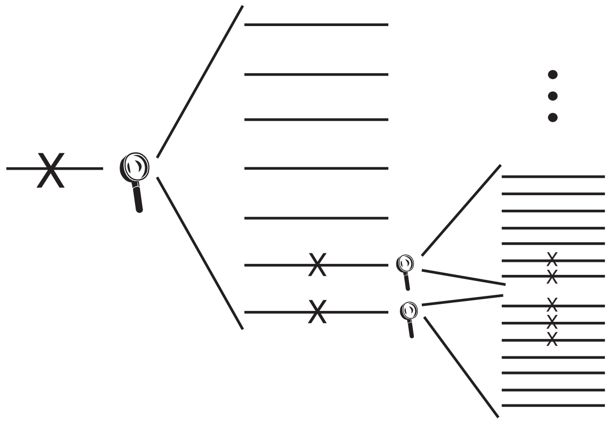


Figure 3.5: Concatenated coding affected by correlated errors.

Also, when we prepare the ancilla qubits – used in syndrome computation – the way Steane and Preskill prescribe [39][52][53], we have limited structural redundancy because at lest two ancilla sets are employed for the corresponding syndromes. If we place the ancilla sets in neighbouring positions, correlated errors will also affect the syndrome computation (see Figure 3.6 for such an example).

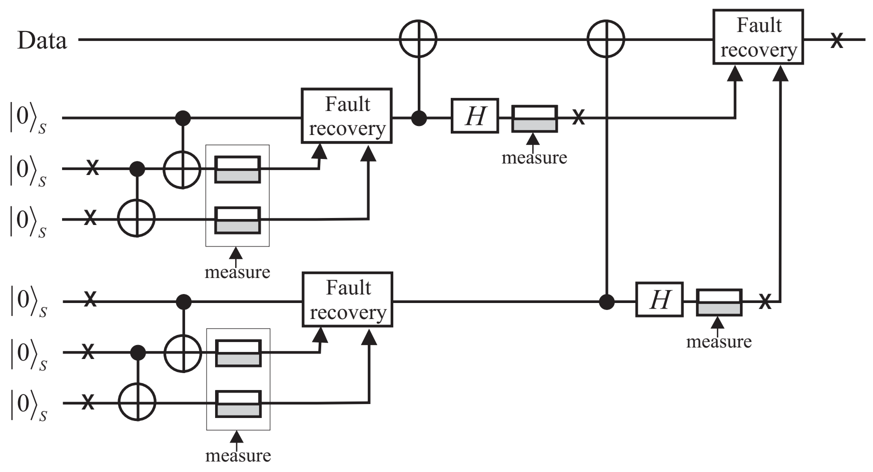


Figure 3.6: Steane error recovery affected by correlated errors.

As for the problem (β, this is an obvious resource-related issue. When using concatenated coding, a lot of qubits are used in order to assure the reliability of just one qubit. With all this waste of encoding required qubits, we still have to use structural redundancy, because otherwise we will not have a safe recovery and all the encoding effort is useless. Structural redundancy means at least doubling the ancillary qubit consumption (ancillary syndrome qubits also use to concatenated coding!), even if the first ancilla set was correctly prepared.

All the critique contained in the considerations regarding problems (α and (β will not question the validity of the quantum fault-tolerant techniques including concatenated coding, ancilla preparation, syndrome measurement and the use of stabilizer codes. These are the best options from a theoretical point of view. The observations were made from an engineering point of view, which require that some practical aspects be taken into consideration. Among these aspects we will find the huge ancillary qubit waste, and the inevitable problems that the practical technology will face when maintaining and handling qubits (i.e. correlated errors).

3.3 Sketching the solution

The fact that assuming any uncorrelated probabilistic errors is not realistic, from an engineering point of view, is recognized by Preskill [39][38]. Reconfigurable quantum hardware *rQHW* could be a solution for problems (α and (β, which are related to fault tolerance issues induced by the presence of correlated errors. Also, when choosing dynamically the depth of concatenated coding, the quantum hardware is reconfigurated accordingly. Another motivation comes from the fact that reconfigurable solutions are already used successfully in classical computing fault tolerance [41].

When fighting the *transient correlates errors*, *rQHW* brings flexibility that allows for dynamic ancilla qubit preparation. The actions to be taken against correlated errors are:

- choosing (i.e. setting a corresponding configuration) the ancilla qubits from a distinct ancillary set so that they will not be neighbors to each other;

- just one set of ancillary qubits is set for one data block; if testing reveals that is faulty, then another ancilla set is configurated in a different area of the reconfigurable quantum circuit.

From a fundamental point of view, all the coding techniques used in fault-tolerant quantum computation are designed to move the "prone to fail area" from the data block to ancilla used in syndrome computation, where structural redundancy is used. As we will see, by using a superposition of basis states in the configuration register, we create a fundamentally lower error probability if the configuration register state is trustworthy. Therefore, the "prone to fail area" is moved onto the configuration register.

In principle, a reconfigurable quantum circuit is a quantum gate array (QGA), acting on an *input register* the way it is prescribed in a *configuration register*. The processed input is stored in an *output register*. Normally, we should not care about the outcome corresponding to the configuration register (see Figure 3.7). In a formalized expression, we have:

$$U_{QGA} : |input\rangle \otimes |config\rangle \longmapsto |output\rangle \otimes |don't\ care\rangle \tag{3.1}$$

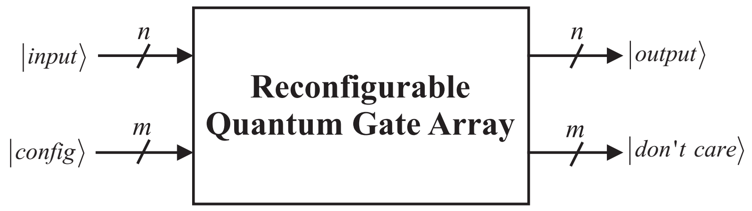


Figure 3.7: Reconfigurable quantum gate array: the involved registers.

3.3.1 Limitations for rQHW

There are two main limitations that do not allow us to deal with the quantum programmable gate arrays the way it is done in classical hardware.

1. As shown by Nielsen and Chuang we cannot have a programmable gate array that can be configured so that it performs any unitary operations, unless the gate array "operates in a probabilistic fashion" [29].
2. It is impossible to build a *switch-based* quantum gate array, as shown in the following proposition.

Proposition 2.1 (No switches) Due to the qubit cloning impossibility, we cannot have a switch-based programmable quantum gate array.

Proof: In order to have a switch-based QGA we must be able to implement a basic switch in quantum terms. Figure 3.8 presents such a device.

Building the switch from Figure 3.8 requires that the black box marked as "quantum switch ?" will act like this:

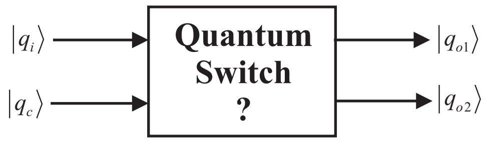


Figure 3.8: The desired quantum switch.

$$U_{switch} : |q_i\rangle_{i1} \otimes |q_c\rangle_{i2} \otimes |0\rangle_{o1} \otimes |0\rangle_{o2} \mapsto \begin{cases} |q_i\rangle_{i1} \otimes |q_c\rangle_{i2} \otimes |q_i\rangle_{o1} \otimes |0\rangle_{o2} & \text{for some} |q_c\rangle; \\ |q_i\rangle_{i1} \otimes |q_c\rangle_{i2} \otimes |0\rangle_{o1} \otimes |q_i\rangle_{o2} & \text{otherwise.} \end{cases} \tag{3.2}$$

Suppose that we are in the first instance of Equation 3.2 (for some $|q_c\rangle$). Then, the switch is reduced (because $|q_c\rangle$ is fixed) to

$$U_{sw1} : |q_i\rangle_{i1} \otimes |0\rangle_{o1} \longmapsto |q_i\rangle_{i1} \otimes |q_i\rangle_{o1} \tag{3.3}$$

But Equation 3.3 is impossible because the no-cloning law of quantum mechanics says that there is no U_{clone} so that for any $|\psi\rangle$:

$$U_{clone} : (|\psi\rangle \otimes |0\rangle) \longmapsto (|\psi\rangle \otimes |\psi\rangle) . \tag{3.4}$$

3.3.2 Reconfigurable quantum gate array (rQGA) structure

Due to the second limitation of the *rQHW*, any programmable quantum gate array, which consists of a set of basic reconfigurable cells, will have to compose the cells in a linear fashion. Figure 3.9 presents the consequence of limitation 2. Here, we have highlighted the number of qubits for all inputs and outputs involved.

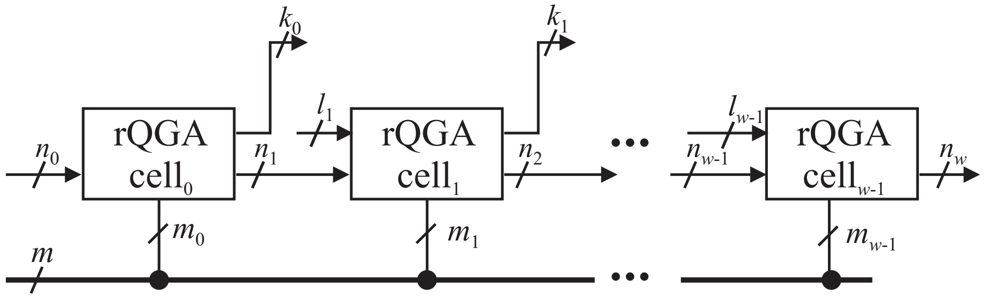


Figure 3.9: Linear connection of basic reconfigurable quantum gate arrays, allowed by the second limitation.

There are w linear connected cells, with two kinds of inputs – outputted by a previous cell (n_j with $j = \overline{0, w-1}$, n_0 from hole circuit's input) and coming from the input (l_j with $j = \overline{1, w-1}$ in number). Also, there are two kinds of outputs: going to be inputs for the next cell ($n_{j+1}, j = \overline{1, w}$ in number, with n_w being part of hole circuit's output), and going to the general output ($k_j, j = \overline{0, w-2}$ qubits for each cell). Also, there is $m = m_0 + m_1 + m_{w-1}$ qubits control register; cell j having m_j corresponding control qubits.

The only limitation is that, for each $j = \overline{0, w-1}$, the following equation has to hold:

$$l_j + n_j = k_j + n_{j+1} \tag{3.5}$$

when considering that we have a $l_0 = 0$ and $k_{w-1} = 0$.

As for the circuit as a hole, we have a total of $n_0 + \sum_{i=1}^{w-1} l_i$ input qubits, which is equal to the total output qubits number $n_{w-1} + \sum_{i=0}^{w-2} k_i$.

3.3.3 Apropriate gates

In order to have a programmable array of gates, the usage of each gate has to be conditioned by some dedicated qubits (which form the *configuration register*). Therefore, the most convenient set of gates to be used in a basic cell of *rQGA* is inspired by gate family $\wedge_n(U)$ that was introduced by Barenco *et al* [7]. With this formalism $\wedge_n(U)$ is a $(n+1)$-qubit unitary operator, described by [7]:

$$\wedge_n(U)(|a_0, \dots a_{n-1}, b\rangle) \mapsto \begin{cases} u_{b0}|a_0, \dots a_{n-1}, 0\rangle + u_{b1}|a_0, \dots a_{n-1}, b\rangle & \text{if } \wedge_{i=0}^{n-1} a_i = 1 \\ |a_0, a_1, \dots a_{n-1}, b\rangle & \text{if } \wedge_{i=0}^{n-1} a_i = 0, \end{cases} \tag{3.6}$$

where U is any unitary transformation [30][12][13] $U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$ with $u_{00}, u_{01}, u_{10}, u_{11} \in \mathbb{C}$, $n \in \mathbb{N}$, and $a_0, a_1, \dots a_{n-1}, b \in \mathbb{B} = \{0, 1\}$.

However, for any gate that we want to conditionally introduce in our rQGA, we need a particular case of this formalism, where $n = 1$ and U is a $m-qubit$ unitary transformation, representing the gate which is conditioned by one qubit from the configuration register. This means that the general form of our 1-qubit conditioned gate $\wedge_1\left(U_{m\text{-qubit}}\right)$ is the $2^{m+1} \times 2^{m+1}$ matrix:

$$\wedge_n(U) = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & & u_{0,0} & \dots & u_{0,2^m-1} \\ & & & & & \ddots & \\ & & & & u_{2^m-1,0} & \dots & u_{2^m-1,2^m-1} \end{pmatrix}. \tag{3.7}$$

In the basic quantum reconfigurable cell architecture, we will use the following elementary gates: $\wedge_1(U_{CNOT})$, $\wedge_1(H)$. U_{CNOT} stands for any matrix describing a $CNOT$ transform, while H is the Hadamard matrix. While the relationship between U_{CNOT} and $\wedge_1(U_{CNOT})$ has been extensively described in [7][62], $\wedge_1(H)$ is a special 2-qubit gate described by matrix and depicted in Figure 3.10.a:

$$\wedge_1(H) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{3.8}$$

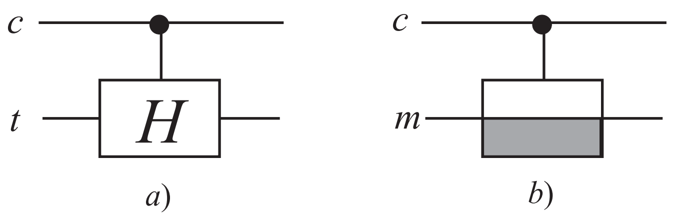


Figure 3.10: Special conditioned gates: *a*) Hadamard, and *b*) qubit measurement. Qubit c is used for control, t is target for Hadamard, and m measured if $c = |1\rangle$.

Also, a conditioned measurement is needed in order to build a general reconfigurable cell for the studied quantum circuits. The control for conditioned measurement gates (see Figure 3.10.b) can be only a basis state (i.e. its c is a bit, not a qubit).

Another observation is that we can use either a "hardwired" implementation for the Toffoli gate the way it is described in 2.22, or a reconfigured version by using 2 basic reconfigurable quantum gate array ($rQGA$) cells.

3.3.4 Basic reconfigurable cell

The basic reconfigurable quantum gate array ($rQGA$) cell is designed so that rippling several such basic cells allows for configuration of any concatenated code based on stabilizer coding and Steane-like ancilla qubit preparation. The idea of this cell architecture is based on the fact that, when looking at Figures 2.8,2.9.B,2.13 and 2.14, the circuits are built by a regular array of gates, with a common pattern for both data stabilizer encoding, ancilla qubit preparation, and syndrome computation.

This general architecture of the basic cell is presented in Figure 3.11. There are n input and output qubits, and m control qubits. The grey lines represent control qubits, while the black lines correspond to the processed qubits. Also, a dashed line stands for a control that can only be classical (no basis state superpositions); full grey lines means that the control can also be of quantum nature (i.e. superposition of basis states).

In Figure 3.11, the first level of gates from left to right are represented by two Hadamard gate sub-levels (each having n gates): the first one is common for all the involved circuits (code generation, ancilla preparation and syndrome computation), the second is used due to the fact that in some circuits we need a space basis transformation (like in Figure 2.1).

The second level of gates is formed out of Toffoli gates, i.e. XOR gates with 1 qubit for control. The controlled XORs are placed so that there are gates with each qubit as source for all the other qubits being targets. There are two sets of such gates, therefore having a total of $n^2 - n$ gates in this level (n is the number of qubits in the input register).

The third level of gates is again formed out of two Hadamard gate sub-levels. The first sub-level is responsable with basis transformation, while the second corresponds to the second Hadamard gate level from the ancilla preparation circuit (fig 2.9). Thus, we add another $2n$ gates to the basic cell architecture.

The fourth gate level consists of conditional measurement gates: they are required by the construction of syndrome computation circuits that are processing ancilla qubits (Figures 2.132.14), and by the fault tolerant Toffoli gate architecture from Figure 2.22.

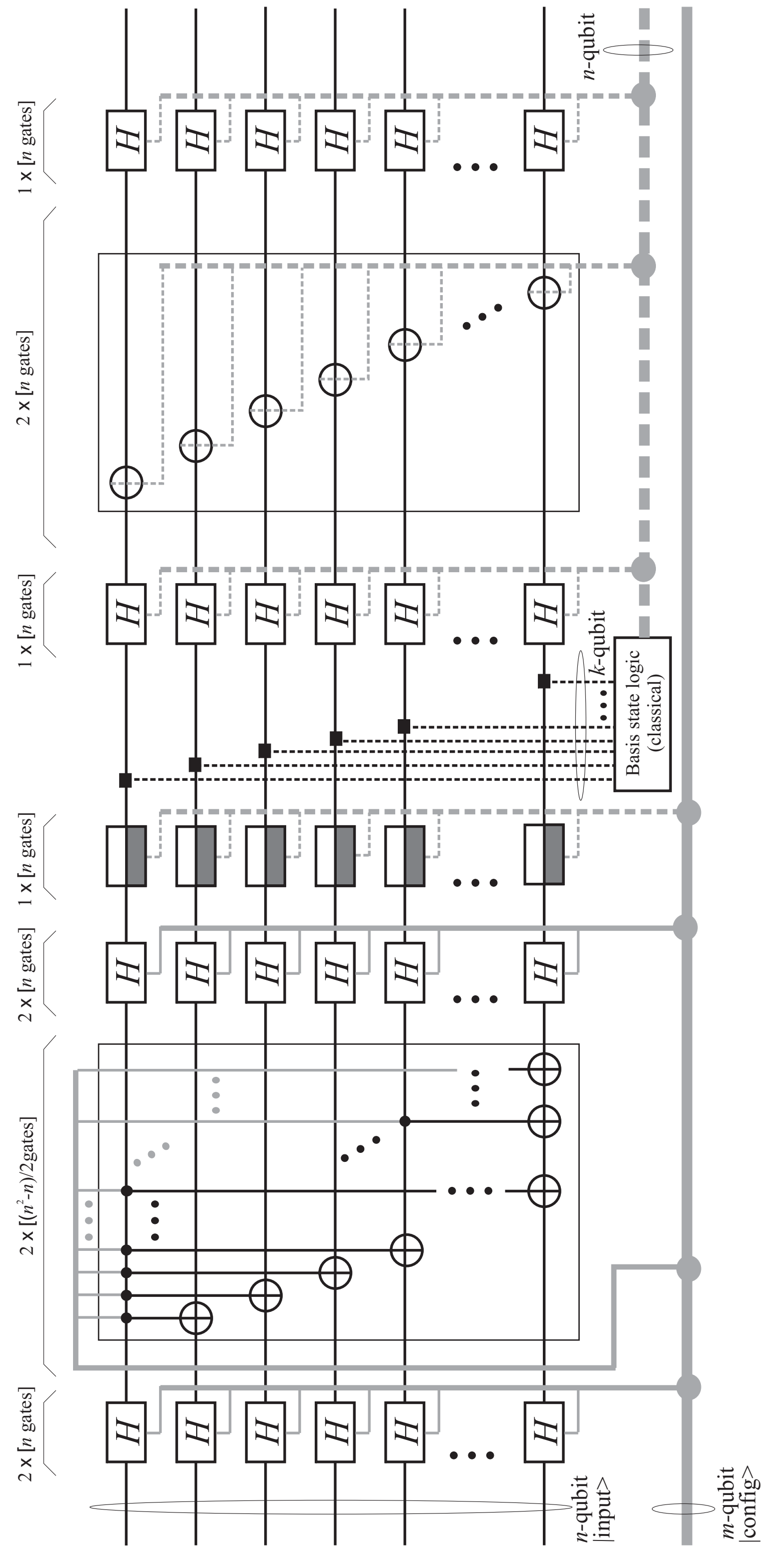


Figure 3.11: The basic reconfigurable cell for stabilizer encoding solutions.

Then, Figure 3.11 shows that we need to be able to process the qubit measurement classical outcomes with classical circuits (i.e. processing with basis states). The basis state logic takes k classical inputs and produces n classical outputs, which are very important because these contol signals will configure the rest of the basic cell architecture. Such a solution is motivated by the fact that the syndrome for a stabilizer code is computed with classical means after the measurement step, and the correction is also driven (controlled) by the outcome of that classical computation.

The remained gate levels are controlled with basis states (i.e. clasically) and are responsible with the correction step.We have two sets of XOR gates, having each of the n qubits as targets, and being controlled classically. The XOR gate level is guarded by two Hadamard gates levels, which are responsible for changing the state basis.

3.3.5 Quantum configuration

For the basic cell in Figure 3.11 we are able to give a structure of the configuration register. First of all, the configuration information for the left half of the cell is of quantum nature, while the right half has classical configuration, hence we have:

$$|config\rangle_{basic\,cell} = |\psi\rangle_{conf} \otimes |bit\ string\rangle \tag{3.9}$$

The *bit string* classical configuration register has the following structure:

$$|bit\ string\rangle = |\underbrace{mm\ldots m}_{n\text{ bits}}\underbrace{hh\ldots h}_{n\text{ bits}}\underbrace{xx\ldots x}_{2n\text{ bits}}\underbrace{hh\ldots h}_{n\text{ bits}}\rangle \tag{3.10}$$

In Equation 3.10 we have denoted with m a bit that controls a measurement gate, h a Hadamard gate, and x a XOR gate.

As for the $|\psi\rangle_{conf}$ part of the configuration register, as we will see, it can be a superposition of basis state configurations with the same structure given in Equation 3.12. The quantum configuration

$$|\psi\rangle_{conf} = \sum_{i=0}^{k} a_i |N_i\rangle \tag{3.11}$$

with $\sum_{i=0}^{k} \|a_i\|^2 = 1$, $a_i \in \mathbb{C}$, has $N_i \in \mathbb{N}$ as basis states, which is equivalent to the structured binary string in Equation 3.12.

$$N_i = |\underbrace{hh\ldots h}_{2n\text{ bits}}\underbrace{t^0_{0,1}\ldots t^0_{0,n-1}t^0_{1,2}\ldots t^0_{1,n-1}\ldots t^0_{n-2,n-1}}_{\frac{n(n-1)}{2}\text{ bits}}\underbrace{t^1_{0,1}\ldots t^1_{n-2,n-1}}_{\frac{n(n-1)}{2}\text{ bits}}\underbrace{hh\ldots h}_{2n\text{ bits}}\rangle \tag{3.12}$$

All h's and t's are binary digits ($\in \{0,1\}$) with the following meaning: h controlls a Hadamard gate, and $t^l_{s,g}$ is a Toffoli gate from layer l (could be only 0 or 1) with the first control qubit in the configuration register, the second being input qubit s, and the g input qubit as target.

Appendix A presents (in a practical manner) the way configurations may be build in order to obtain some useful fault tolerant circuits. The described reconfiguration methods can be applied for any qubit or qubit group from the input register, so that the qubit vicinity is manipulated as a weapon in fighting correlated errors. When errors appear, the qubits from the code block are selected so that they are physically distanced from each other.

However, this is just one aspect of improving the fault tolerance methodologies of quantum computation with *rQHW*. An interesting approach will employ the quantum configuration opportunity.

If the configuration register is a superposition of classical configurations (i.e. basis states), then the reconfigurable quantum gate array (*rQGA*) will have all the configurations from the superposition at the same time. Therefore, we will have a superposition of k distinct circuits at the same time. Figure 3.12 presents this feature of quantum reconfigurable circuits.

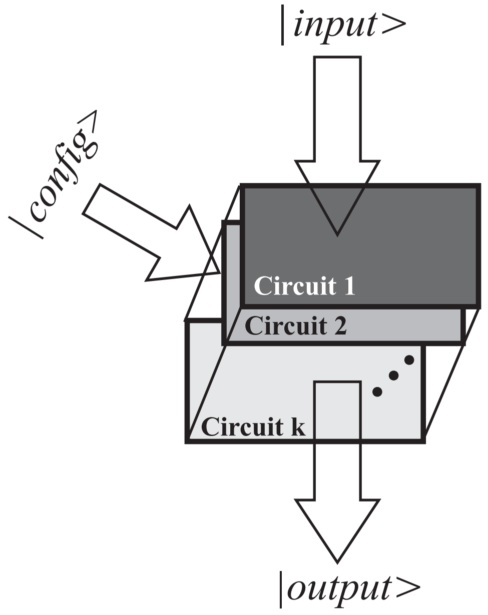


Figure 3.12: When the configuration register has a quantum nature, the same reconfigurable quantum gate array acts a a superposition of k simultaneous distinct circuits. These circuits share the same input state and the same output qubits. The output qubits encode a superposition of the superposed circuits distinct outputs.

For our previously discussed fault tolerant circuits, we can use a superposition of classical configurations so that the circuit produces a superposition of all the possible stabilizer codes. Moreover, we will be able to configure at the same time all the possible versions (corresponding to all possible stabilizer codes) of the circuit from – for instance – 2.13. Of course, when measuring the configuration register, just one such fault tolerant configuration will remain, corresponding to a particular stabilizer code. The idea is that, with such a procedure, the probability of a gate error occurring in the fault tolerant circuit becomes lower, given a reliable configuration state.

Suppose all the superposed classical configurations have the same quantum amplitude, then they will have the same probability of being measured. Also, if we take the circuit from Figure 2.13 (stabilizer code with Steane ancilla preparation), we have $4! = 24$ possible configurations each corresponding to a distinct stabilizer code (all the possible u_0, u_1, u_2, u_3 column arangements from the Hamming matrix in Equation 2.14), and a $\frac{1}{4}$ probability of having the presence of the same gate in two distinct classical configurations. Then, after the measurement of the configuration register then – with a ξ_{gate} gate error probability given by the available technology – the reconfiguration solution will have an overall gate error probability of $\xi_{\text{gate}}^{\frac{24}{4}} = \xi^6$.

In a general form, the gate error rate for the overal reconfigurable gate array ($rQGA$) is:

$$\xi_{\text{rQGA}} = \left(\xi_{\text{gate}}\right)^{k\times f_r} \tag{3.13}$$

where ξ_{gate} is the technology gate error rate, k the number of superposed circuits in $rQGA$ (i.e. the number of superposed basis states in the configuration register), and f_r is the so-called *freedom rate* or the frequency of a gate not being used in one particular configuration, but used in the other superposed configurations. In our example from above, $f_r = \frac{1}{4}$ and $k = 24$.

This is nevertheless a very good result, which capitalizes on the parallelism of quantum computation, but it is obtained under the assumption that the configuration register is reliable. However, the quantum configuration technique is useful because reduces the problem of gate fault tolerance in quantum circuits to a quantum state storage problem. In other words, with the reconfiguration technique we achieve fault-tolerant operations, by assuring the storing fault tolerance of a particular quantum state in the configuration register. That is, naturally, an easier job.

As it is explained in appendix A, there are some practical reasons which make the theoretical ξ^6 hard to achieve. Mainly, there are two reasons: it is hard to set a quantum superposition containing a number of basis states that is not equal to a power of two, and the physical limits of the basic cells may force us to a low f_r (i.e. the same gate is used by many superposed configurations, so if an error occurs on that gate, it will spread on all the configurations that are using it). Our heuristic engineering approach to these problems provides for a solution that guarantees an overall $\xi^{\frac{8}{3}}$ error rate. This result (obtained for configuring the stabilizer encoder in a quantum fashion) is even better than ξ^2, which is the target for single error correction coding.

In oreder to assess, in fundamental terms, the consequences of reconfigurable quantum hardware approach, we will get back at Equation 2.45 from section 2.7.1. As we are using the same type of circuits, which are superposed with a quantum configuration, and the circuit for setting the quantum configuration will not increase p (appendix A shows the way to obtain such configuration states, with a fixed number of extra gates), we can write the accuracy threshold:

$$\xi_{threshold}^{rQHW} \sim (\log N)^{-p\cdot(1-f_r)\cdot\frac{1}{S}}. \tag{3.14}$$

All the superposed configurations can be grouped, so that the members of one configuration group will not use any gate which is used by the configurations from any other group. The number of such distinct groups is S. In the above equation, f_r is the freedom rate dictated by the members of one particular configuration group.

For a circuit using stabilizer encoding generated with *rQHW* in a quantum configuration, and for the heuristic implementation provided in appendix A, suppose we have a high $p = 6$. Then, our accuracy threshold $\xi_{threshold}^{rQHW}$ is of the order given by $\text{xir}\,(N) = \log N^{-\frac{18}{8}}$ because $S = 2$ and $f_r = \frac{1}{4}$. The comparison between function $\text{xir}\,(N)$ and the technological assuring threshold ($\lim = 10^{-3}$) is given in Figure 3.13. This figure shows in a very comprehensive way the fact that the *rQHW* technique provides means for arbitrary long fault-tolerant quantum computation, because the xir (N) function is way over the technological limit, even for very high N.

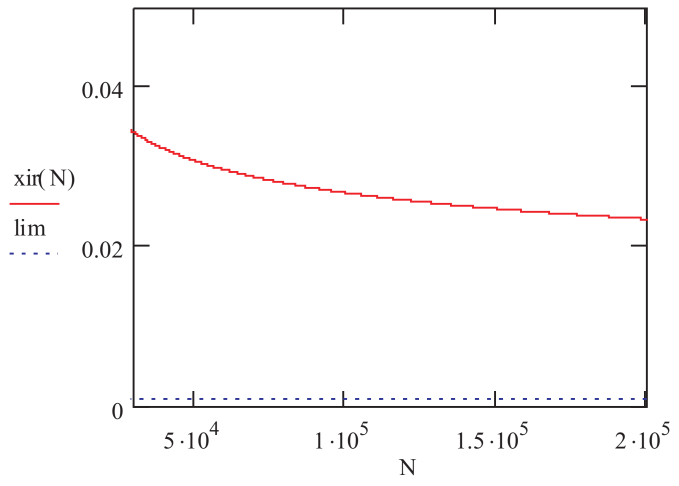


Figure 3.13: Evolution of accuracy threshold value for *rQHW* stabilizer codes ('xir' function) with the number of computational steps (N). The technological accuracy limit ('lim') is also provided for a relevant comparison.

3.4 Conclusions

The qubits and gates involved in quantum computation implementations are prone to frequent errors, due to the delicate nature of quantum basis state superposition. Decoherence and leakage errors must be fight against, along with those having a classical correspondent. While there is an ongoing and outstanding effort in developing a quantum technology with inherent fault tolerance [1], special encoding techniques and quantum circuits have been created in order to reduce the probability of an error affecting the functional circuit. The specially designed techniques rely on classically inspired codes (i.e. *Hamming*, *stabilizer codes*). Also, due to the fact that there is a serious difficulty in achieving safe recovery, we are forced to employ *structural redundancy*. With these techniques we would expect a ξ^2 error probability for the entire circuit, when the qubit and gate error probability are of the order ξ. Moreover, the *concatenated coding* provides the possibility of performing arbitrary long quantum computations in fault tolerant conditions.

Nevertheless, if the error rates (ξ's) are exceeding the calculated threshold values, instead of reducing the overall error probability, concatenated coding can make things even worse because it generates more qubits and gates which are, of course, prone to more errors. Therefore, some of the most relevant papers and books of this field are stressing the need for better, more reliable, quantum hardware [1][16].

3.4.1 Shor algorithm requirements

The effort that has to be employed for achieving fault tolerance in quantum algorithm implementations is better emphasized by the following facts. When applying Steane ancilla preparation and stabilizer coding in Shor algorithm implementation, we will use:

- $3 \cdot 10^9$ Toffoli gates;
- 2160 qubits [39].

By introducing concatenated coding in Shor algorithm circuit, with the technological values for storage and gate error probability being $\xi_{\text{store}} = \xi_{\text{gate}} = 10^{-6}$, we will have to use:

- 3 levels of concatenation, therefore a block size of $7^3 = 343$ qubits;
- a total number of 10^6 qubits.

3.4.2 *rQHW* summary

As already pointed out by John Preskill [39], due to the fact that quantum reliability theory is based on some simplifying assumptions (no correlated errors, no dependency between the error rate and the number of qubits), it is possible that additional techniques will be needed for the future quantum circuit engineering issues. The presented theory and circuit designs are ignoring

- gate placement;
- ancilla availability;
- leakage errors (there is a circuit which detects this kind of errors [39]).

The leakage errors were consistently addressed [39] but the engineering problems linked to the correlated errors and the high ancillary qubit usage remain. We have introduced the so-called *reconfigurable Quantum Hardware*(rQHW) and the *reconfigurable Quantum Gate Array* (rQGA) Cells, as incentive in avoiding the destructive effect of the correlated errors, and for reducing the number of required ancilla qubits.

Summarizing, the advantages drawn from *rQHW* usage are:

- structural redundancy (used for syndrome computation) with reconfigurable properties provides for ancillary qubit saving;
- qubits used in fault tolerant techniques for one data block can be choose so that they are not neighbors to each other and the possibility of space correlated errors is limited;
- the fault tolerance in syndrome computation can be reduced to preserving a reliable configuration register, hence we will concentrate mainly on storage errors, as gate errors will become less likely;
- quantum configurations for *rQHW* provides for a "superposition of circuits at the same time", which means that structural redundancy in syndrome computation is achieved with less ancillary qubits.

Chapter 4

Evolvable Quantum Hardware

The reconfigurable quantum gate array from Figure 3.7 will translate to *evolvable quantum hardware (qEHW)* if its configuration register, the m-qubit register $|config\rangle$, represents a state outputted by the means of a genetic algorithm. Therefore, if a qEHW is to be synthesised, one has to specify how to run genetic algorithms in quantum computing. This chapter proves that there is a methodology of running any genetic algorithm on a quantum computer in $\mathcal{O}\left(\sqrt{n}\right)$ time. Then, we also provide the guidlines for implementing the corresponding quantum circuit.

4.1 Introduction

By clearly identifying its most major problems and limitations, computer science has become mature [30]. The research community has put a lot of effort in the attempt to solve these problems and further pushing the computing frontiers; however, by using the means of what is now called *classical computation*, it seems that one can hardly expect more than marginal improvements, even with sophisticated approaches.

In this context, inspiration was mainly found in biology and physics: bio-inspired computing and quantum computing are considered as possible solutions. The opti-mism is fed by theoretical and practical achievements. Genetic algorithms and evolv-able hardware are already successfully used in a wide range of applications, spanning from image compression, robotics and other artificial intelligence related issues, to engineering problems like fault tolerance and reliability in critical environments. Moreover, quantum computing seems to draw even more power from its exponential parallelism: Peter Shor has proven that a classical exponential problem (integer fac-torization) can be solved in polynomial time [46][48].

The above considerations indicate that the merge between the two novel computing promises, namely genetic algorithms (GAs) and quantum computing (QC) would be natural and benefic [51][44][45]. Researchers already follow the path of so-called Quantum Evolutionary Programming (QEP) [17] with outstanding results [49][50]. For instance, the best approach for automated synthesis of quantum circuits uses genetic programming [26]. Also, quantum algorithm design can be approached by evolutionary means [50]. In fact, the majority of such applications are addressing quantum computation design issues regarding quantum algorithms and implementations [49]; they are all part of QEPs sub-area called Quantum Inspired Genetic Algorithms (QIGAs) [17][28]. The other sub-area, called Quantum Genetic

Algorithms (QGAs), tries to implement ge-netic algorithms in a quantum computation environment [17][44][45]. This paper pro-poses a new perspective on QGAs, by showing that a different strategy is necessary in quantum computation, because the genetic search can be reduced to Grovers algorithm [20].

4.1.1 Motivation

The QGAs rely on qubit representations for the chromosomes and the use of quantum operators in order to process them during the quest for the optimal solution of the search problem. In principle, this approach redefines the GA operators in quantum terms; these new operators will perform better due to the exploit of the quantum parallelism [45]. Nevertheless, approaching specific applications this way will result in a significant performance enhancement [21][22].

Because the chromosome represented by qubits, just one quantum chromosome register would be able to store the entire population as a superposition of all the possi-ble classical states. The function that evaluates the fitness of the initial population (which could also be the entire population) would take the *chromosome register* as input and the output would be stored in a *fitness register*. This would store a superposition of all the fitness values, corresponding to the superposition of the indi-viduals from the chromosome register.

The key observation that led us to this new perspective is the fact that if the best fitness value can be *marked* (i.e. by changing the phase of the corresponding eigenstate) without destroying the superposition of the registers, then Grovers algo-rithm will find the solution in $\mathcal{O}\left(\sqrt{n}\right)$. Therefore, all the quantum versions of GA operators, like crossover or mutation, would become useless if we figured out a way to mark the best fitness, inside the fitness superposition state.

4.2 Quantum Genetic Algorithms

As part of Quantum Evolutionary Programming, QGAs have the ingredients of a sub-stantial algorithmic speedup, due to the inherited properties from both QC and GA. However, there still are questions as to how would it be possible to implement a ge-netic algorithm on a quantum computer. The attempts made in this particular direction suggest there is room left for taking advantage from the massive quantum computation parallelism [45]. Moreover, some questions were left open, as pointed out in [17].

4.2.1 Running GAs in a Quantum Computational Environment

For the first time, the possibility (and the advantages) of the QGAs were indicated in [45]. The approach described here contains hard evidence for QGA speedup, but there still are some unanswered questions [17]. The proposed algorithm uses a number of m register pairs:

$$|\psi\rangle_i = |\phi\rangle_i^{individual} \otimes |\rho\rangle_i^{fitness} \tag{4.1}$$

where $i = \overline{0, m-1}$. The first (left) register contains the individual, while the second contains its corresponding fitness. Because we are dealing with quantum regis-ters, both $|\phi\rangle$ and $|\rho\rangle$ can encode a superposition of exponentially many individuals and their corresponding superposed fitness values. Each time a new population (set of individuals) is generated in the *individual* register, the corresponding fitness is computed and stored in the *fitness* register. Of course, if the fitness register is measured then, due to entanglement [30], the result is only one of the superposed values; in the individual register will remain superposed the individuals that give the measured fitness. Fitness register measurement is a crucial element in developing QGAs [45]. For the general expression of the pair register (N-qubit for the individual register and M-qubit for the fitness register) given in Equation 4.2 the measurement of the second register ($|y\rangle$) will have r as result with probability from Equation 4.3.

$$|\psi\rangle_i = \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^M-1} c_{x,y}|x,y\rangle, \text{ with } \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^M-1} |c_{x,y}|^2 = 1 \tag{4.2}$$

$$P(r) = \sum_{x=0}^{2^N-1} |c_{x,r}|^2 = 1 \tag{4.3}$$

The post-measurement state of the pair register will be:

$$|\psi_r\rangle_i = \frac{1}{\sqrt{P(r)}} \sum_{x=0}^{2^N-1} c_{x,r}|x,r\rangle \tag{4.4}$$

Due to the fact that an individual cannot have more than one fitness, it is obvious that, if individual u has a fitness value v, then $c_{u,y} = 0$ for all $y \neq v$.

The QGA, as described in [45], is presented in the following pseudo code:

Genetic Algorithm Running on a Quantum Computer (QGA)

```
1. For $i := 1$ to $m$ prepare $|\phi\rangle_i^{individual}$ as superposi-tions of individuals and compute the corre-
   sponding fitness pair register $|\rho\rangle_i^{fitness}$ (the outcome will be a set of $m$ fitness values superpo-
   sition).
2. Measure all fitness registers.
3. Repeat
   (a) Selection according to the $m$ measured fitness values.
   (b) Crossover and mutation are employed in order to prepare a new population (setting the
       $m$ individual registers).
   (c) For the new population, the corresponding fitness values will be computed and then
       stored in the fitness registers.
   (d) Measure all fitness registers.
   Until the condition for termination is satisfied.
```

Reference [17] provides analysis and critique for the above presented algorithm. The identified advantages of using QGAs over the classical GAs, which are drawn from the quantum computational features, are :

- Due to the superposition of individuals (i.e. basis states) that is stored in the individual register, the building block [17] could be crossed not by just one individual, but by a superposition of exponentially many individuals. Thus, the selection of a new population is made with the contribution of many attraction pools.

- In quantum computation true random numbers can be generated. It was proven that a GA with a true random number generator will outperform a pseudo-random solution, which is the only possibility in classical computation [44].

The questions that remain open are:

- How is it possible to build the crossover operator in quantum computation?

- How is it possible to implement the fitness function on a quantum computer?

- How can the correlation be maintained – by using the entanglement – between the individual register (superposed) basis states and the (superposed) fitness values from the fitness register?

Although the advantages appear to be substantial, one can easily argue that the power of quantum computation is not sufficiently used by this approach. However, some of the opened questions have been addressed in reference [17]. Giraldi *et al.* developed a mathematical formalism in order to avoid misinterpretations regarding the last question. The second question is also addressed by defining quantum genetic operators. The proposed formalism establishes the necessary correlation between the fitness and the individual registers, which cannot be accomplished with the QGA construction provided in [45].

4.2.2 Mathematical Formalism

The QGA formalism uses m quantum register pairs (N-qubit individual register and M-qubit fitness register,) as presented in Section 4.2.1. Also, in order to achieve proper correlation between the individual and its fitness value, the fitness function must be chosen so that it is a "quantum function" as defined by [33], hence a pseudo-classical operator with a corresponding Boolean function: $f : \{0,1\}^N \rightarrow \{0,1\}^M$, $U_f : |x\rangle \otimes |0\rangle \rightarrow |x\rangle \otimes |f(x)\rangle$ if $|x\rangle$ is a basis state. When acting on a superposition, the unitary operator corresponding to function f will dictate the following mapping:

$$U_f : \sum_{x=0}^{2^N-1} a_x |x\rangle \otimes |0\rangle \rightarrow \sum_{x=0}^{2^N-1} a_x |x\rangle \otimes |f(x)\rangle = \sum_{x=0}^{2^N-1} a_x |x, f(x)\rangle \tag{4.5}$$

An important aspect regarding the pseudo-classical Boolean functions is that they are universal (i.e. any computational function can be represented in such a form), and easy to be implemented as gate networks. In fact, due to their universality, Boolean functions form the backbone of the classical computation's circuit model.

The QGA algorithm, after adopting Giraldi's formalism can be rewritten as in the below pseudo-code.

Genetic Algorithm Running on a Quantum Computer (QGA) with proper formalism

1. For $i := 1$ to m set the individual-fitness pair registers as $|\psi\rangle_i^1 = \frac{1}{\sqrt{n}} \sum_{u=0}^{n-1} |u\rangle_i^{ind} \otimes |0\rangle_i^{fit}$ (a superposition of n individuals with $0 \leq n \leq 2^N$).

2. Compute the fitness values corresponding to the individual superposition, by applying a unitary transformation $U_{f_{fit}}$ (corresponding to pseudo-classical Boolean operator $f_{fit} : \{0,1\}^N \rightarrow \{0,1\}^M$). For $i := 1$ to m do $|\psi\rangle_i^2 = U_{f_{fit}} |\psi\rangle_i^1 = \frac{1}{\sqrt{n}} \sum_{u=0}^{n-1} |u\rangle_i^{ind} \otimes |f_{fit}(u)\rangle_i^{fit}$.

3. For $i := 1$ to m measure the fitness registers, obtaining the post-measurement states (we suppose that $|y\rangle_i$ is measured): $|\psi\rangle_i^3 = \frac{1}{\sqrt{k_i}} \sum_{v \in \{0,1,\ldots,n-1\}} |v\rangle_i^{ind} \otimes |y\rangle_i^{fit}$ with k_i values in $\{0, \ldots, n-1\}$ to satisfy $f_{fit}(v) = y$.

4. Repeat

   - Selection according to the m measured fit-ness values $|y\rangle_i$.
   - Crossover and mutation are employed in order to prepare a new population (setting the m individual registers $|u\rangle_i^{ind}$).
   - For the new population, the corresponding fit-ness values will be computed and then stored in the fitness registers($|f_{fit}(u)\rangle_i^{fit}$).
   - Measure all fitness registers

   Until the condition for termination is satisfied.

Besides the necessary formalism, reference [17] also provides some insight regarding the implementation of the genetic operators in the quantum computational environment. These considerations lead towards two main implementation problems:

α) the number of all valid individuals is not always a power of 2, which is the total number of basis states;

β) crossover implementation is difficult and requires a much thoroughly investigation, including quantum computation architectural aspects [35].

4.3 A New Approach

An observation concerning the individual-fitness quantum register pair is that all the possible valid individuals (n) can be encoded in the same quantum state superposition, which has a total of 2^N possible basis states ($n \leq 2^N$). If we can figure out a method of measuring the highest fitness value from the fitness register, then by measuring the individual register we will get that corresponding individual (or one of them, if several have the same highest fitness value).

Approaching the QGAs in this manner renders genetic operators as no longer necessary, as long as finding the maximum has an efficient solution. This effectively leads to solving problem β.

Because the individual is encoded on N qubits, we have a total of 2^N basis states which can participate in the superposition. It is possible that not all of these basis states will encode valid individuals (problem α); the proposed method relies on defining some constrains regarding the fitness function and the fitness value format, without losing the generality of the solution. We will consider the fitness function as a Boolean pseudo-classical unitary operator U_f (characterized by $f : \{0,1\}^N \to \{0,1\}^M$) which can be also applied to non-valid individuals. The fitness value space $\{0,1\}^M$ can be split, so that a distinct subspace is allocated to the fitness values corresponding to valid individuals and another distinct subspace corresponds only to non-valid individuals. This enables us to concentrate only on processing states that correspond to valid individuals (Section 4.3.2 further elaborates on this particular aspect).

The method of finding the highest fitness value is inspired from efficient quantum algorithms for finding the maximum [2][14]. Finding the best fitness value is equivalent to marking the highest classical state that is superposed in the fitness register state or, in other words, the highest basis state with non-zero amplitude. Basically, the pro-posed methodology relies on reducing the highest fitness value problem to Grover's algorithm. In order to do so, special oracle and fitness value format are defined. Section 4.3.1 presents the quantum algorithm for finding the maximum [2], Section 4.3.2 presents details for oracle implementation and fitness register structure, while Sec-tion 4.3.3 provides our adaptation of the algorithm in order to find the best value in the fitness register.

4.3.1 Computing the Maximum

The minimum/maximum finding quantum algorithms [2][14] are inspired from the classical "bubble sort" algorithm, but their complexity [8] in quantum version is $\mathcal{O}(\sqrt{n})$.

The quantum algorithm for finding the maximum takes an unsorted table of m ele-ments as input, in order to return the index of the maximum value element. By adopting the formalism from [2], we have a pool $P[i]$ of m elements ($i = \overline{0, m-1}$) which will be processed in order to obtain the index k of the maxi-mum element ($P[k]$). The Grovers algorithm can be used with a special oracle that marks all the basis states greater than some given value j:

$$O_j(i) = \begin{cases} 1 & \text{if} P[i] > P[j] \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

Therefore, the resulted algorithm will have the form of the following pseudo code: **Quantum**

Algorithm for finding the maximum from an unsorted table of m elements

```
1. Initialize $k := P[r]$; $0 \le r \le m-1$ and is randomly chosen;
2. Repeat $\mathcal{O}(\sqrt{m})$ times
    (a) Set two quantum registers as $|\psi\rangle = \frac{1}{\sqrt{m}} \sum_{i=0}^{m-1} |i\rangle|k\rangle$;
    (b) Use Grovers algorithm for finding marked states from the first register (i.e. those which
        make $O_k(i) = 1$);
    (c) Measure the first register. The outcome will be one of the basis states that are $> k$. Let
        the measurement result be $x$. Make $k := x$;
```

3. Return k as result. It is the index of the maximum.

The complexity analysis performed in [2] reveals the fact that this algorithm will find the index of the maximum in $13.6\sqrt{m}$ steps, with an error rate smaller than $\frac{1}{2}$.

4.3.2 The Oracle

In order to deal with problem α) (see Section 4.2.2), we have to adopt a constraint, which does not restrict the generality of the fitness functions. If we consider the ordinary fitness function f_{fit} (which applies only on the valid individuals) $f_{\text{fit}} : \{0,1\}^N \to \{0,1\}^M$, it is Boolean (and therefore universal), with a straightforward correspondence to the unitary representation $U_{f_{\text{fit}}}$ [33]. The modified fitness function will accept invalid individuals as argument, and the returned values will belong to distinct areas, corresponding to valid or invalid individuals. This can be achieved by defining $f_{\text{fit}}^{\text{mod}} : \{0,1\}^N \to \{0,1\}^{M+1}$ as:

$$f_{\text{fit}}^{\text{mod}}(x) \in \begin{cases} 0 \times \{0,1\}^M & \text{if} \quad x \text{ is a non-valid individual} \\ 1 \times \{0,1\}^M & \text{if} \quad x \text{ is a valid individual} \end{cases} \tag{4.7}$$

The fitness values are encoded by the qubits in a modified fitness register, which has a $(M+1)$-qubit size. The valid individuals always produce fitness values with the most significant qubit being '1'; a '0' value for the most significant qubit in the fitness register indicates the correspondence to a non-valid individual, as presented in Figure 4.1 (quantum state matrix representation is used).

Another implementation-related problem concerns the oracle described by Equation 4.6. We propose a solution that uses two's complement number representation [36] for marking the states that have a value greater than a given $j \in \mathbb{N}, j > 0$. As a consequence, the fitness register will have the form from Figure 4.2. The oracle processes all the fitness register qubits except the most significant (v), which indicates if the value represented by the other qubits belongs to a valid individual or not. All value qubits ($f_M \dots f_0$) in the fitness register encode two's complement positive integers as fitness values. The oracle adds $-(j+1)$ to the fitness register, therefore the basis states (from the state output by the quantum adder [58]) greater than j will always have $f_M{}' = 0$ (see the oracle implementation from Figure 4.3.) For the solution in Figure 4.3 we used 2 negation gates (denoted with 'x') and one XOR gate [7][30]. The architectures for the quantum arithmetic circuits, including the adder/subtractor, are presented in [58]. Only the qubits containing the result of the arithmetic function ($f_0{}' \dots f_M{}'$) are used by the Grover iteration circuit [20][30] in order to find one of the marked basis states.

Although the oracle uses two's complement addition (which means that we will have to change the fitness values in the superposition), the correlation between the individual and the fitness registers is not destroyed, because the addition is a pseudo-classical permutation function [33][58]. However, the Grover iteration will find as a marked basis state $|p\rangle = |f'_0 \dots f'_M\rangle$, with $p \in \mathbb{N}, f'_M, \dots f'_0 \in \{0,1\}$, which is given by $|p\rangle = |q-(j+1)\rangle$, for $q \in \mathbb{N}$ and $|q\rangle = |f_M \dots f_0\rangle$, with $f_M, \dots, f_0 \in \{0,1\}$. This means that, after measuring $|p\rangle$, we have to add $j+1$ to this value in order to have the correct desired basis state ($> j$).

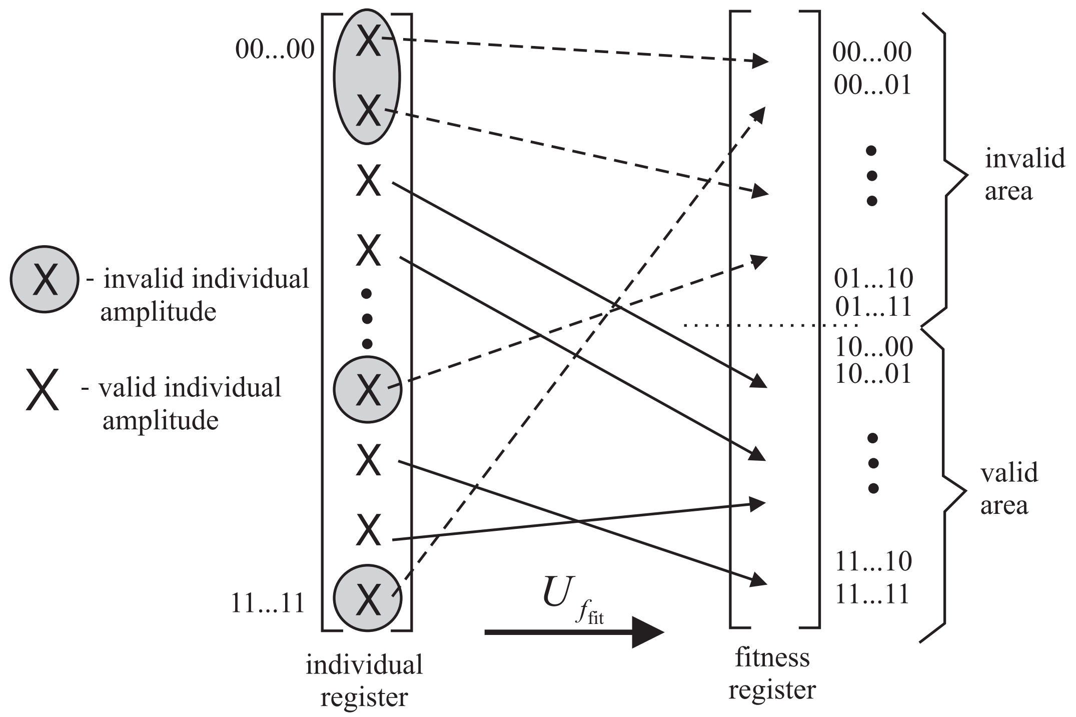


Figure 4.1: Basic idea for fitness function construction: when is applied to valid individuals it produces a value in the valid area (upper half $|10\ldots00\rangle \div |11\ldots11\rangle$) of the fitness register, whereas when applied to invalid individuals the corresponding values in the fitness register will always be in the invalid area (lower half $|00\ldots00\rangle \div |01\ldots11\rangle$).

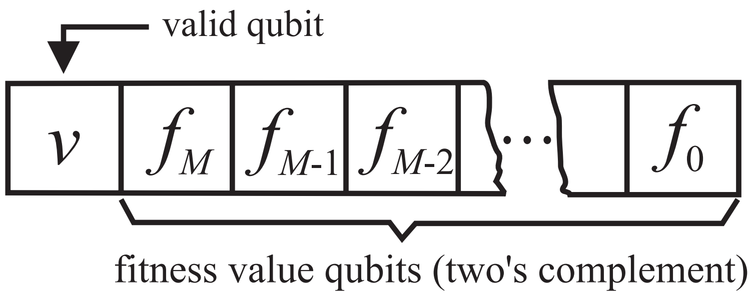


Figure 4.2: The format of the fitness register, for the two's complement approach of oracle implementation.

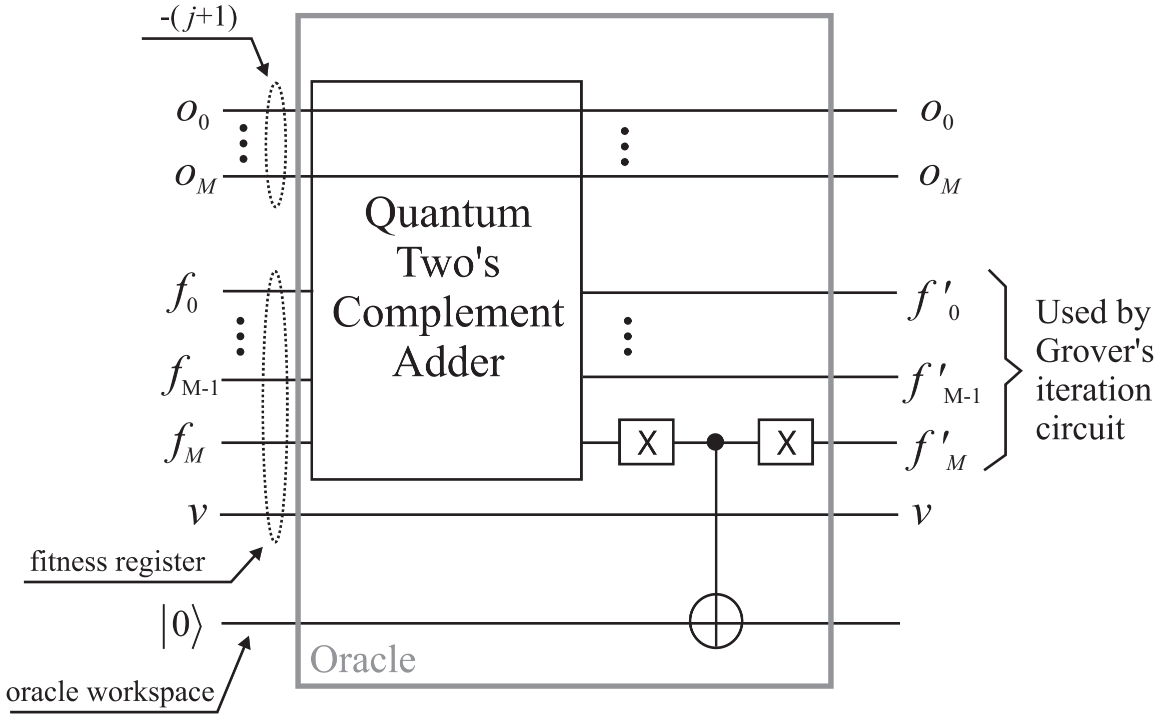


Figure 4.3: Oracle implementation for a fitness register having the structure from Figure 4.2.

4.3.3 Reduced Quantum Genetic Algorithm

Having a fitness register as defined in the previous subsection, the corresponding fitness function, and the specially defined oracle, we are able to provide the pseudo- code that corresponds to running a Genetic Algorithm in the quantum computational environment. It is called *reduced Quantum Genetic Algorithm* (rQGA) because it uses only one population (encoded in just one quantum state), consisting of all possi-ble individual binary representations (that correspond to valid and invalid individuals). Crossover and mutation operators are not used for finding the highest fitness value (they are not required in a quantum context), which is obtained by employing Grovers algorithm.

The algorithm listed below is inspired from the quantum maximum algorithm from Section 4.3.1. The initial max value must obey the $2^{M+1} \leq max \leq 2^{M+2} - 1$ relation, so that the search for the highest fitness value will take place only in the valid fitness area. We have a number of $m \in \mathcal{O}\left(\sqrt{N}\right)$ pair registers (individual-fitness), where the individual register is on N qubits, and the fitness register on $M + 2$ qubits.

Reduced Quantum Genetic Algorithm

1. For $i := 0$ to $m - 1$ set the pair registers as $|\psi\rangle_i^1 = \frac{1}{\sqrt{2^N}} \sum_{u=0}^{2^N-1} |u\rangle_i^{ind} \otimes |0\rangle_i^{fit}$;

2. For $i := 0$ to $m - 1$ compute the unitary operation corresponding to fitness calculation $|\psi\rangle_i^2 = U_{f_{\text{fit}}} |\psi\rangle_i^1 = \frac{1}{\sqrt{2^N}} \sum_{u=0}^{2^N-1} |u\rangle_i^{ind} \otimes |f_{\text{fit}}(u)\rangle_i^{fit}$;

3. $max := randominteger$, so that $2^{M+1} \leq max \leq 2^{M+2} - 1$;
4. For $i := 0$ to $m - 1$ loop
   (a) Apply the oracle. Therefore, if $|f_{\text{fit}}(u)\rangle_i^{fit} > max$ then the corresponding $|f_{\text{fit}}(u) - max\rangle_i^{fit}$ basis states are marked;
   (b) Use Grover's algorithm for finding marked states in the fitness register after applying the oracle. We find one of the marked basis states $|p\rangle = |f_{\text{fit}}(u) - max\rangle_i^{fit}$, with $f_{\text{fit}}(u)\, max \geq 0$;
   (c) $max := p + max + 1$;
5. Having the highest fitness value in the $|\bullet\rangle_{m-1}^{fit}$ register, we measure the $|\bullet\rangle_{m-1}^{ind}$ register in order to obtain the corresponding individual (or one of the corresponding individuals).

4.4 Summary

This paper described a methodology for running Genetic Algorithms on a Quantum Computer. By taking advantage of the quantum computation features, all the possible chromosome binary representations can be encoded in just one individual quantum register. This register is correlated with its pair (fitness) register, which contains a superposition of all corresponding fitness values. Due to quantum mechanical proper-ties, measuring the highest fitness value in the fitness register, leads to a post-measurement state of the corresponding individual register that contains superposed basis state(s) encoding the individual(s) with the highest fitness.

Therefore, the initial problem is reduced to finding the best fitness value without destroying the individual-fitness register correlation. This objective is achieved by adapting an existing quantum algorithm for finding the maximum. Without loosing the generality of the solution, the adaptation requires that a specific structure be adopted for the fitness register, and a special oracle be defined by employing two's complement integer representation. As a result, the problem of finding the highest fitness value can be solved by Grover's algorithm without employing any genetic operators such as crossover and mutation.

Because the complexity of our algorithm adaptation is identical with its original form, and based on the analysis provided by [2], we reached the conclusion that any GA can be performed on a Quantum Computer in $\mathcal{O}\left(\sqrt{N}\right)$ steps (Grover iterations in our case). This consequence broadens the area of computational problems where the quantum solutions outperform the classical ones.

Chapter 5

Fault Injection in Quantum Computation

In classical hardware, fault injection techniques are used for validation of *Fault Tolerance Algorithms and Mechanisms* (*FTAMs*). This dependability verification ability is used for the ultimate goal of incorporating the assessment of used fault tolerance techniques within the design process , which may use an integrated environment [3][4][42][43].

As it is the case of classical circuits, the *Hardware Description Languages* (*HDLs*) are able to support dual behavioral - structural descriptions on different abstraction levels, and are suitable for implementing various experimental and formal testing techniques. These features make the HDLs the most appropriate tools for integrating description, simulation, synthesis, testing, and FTAM testing in the same environment [11][42][23][43].

This chapter will focus on extending our already defined HDL-based quantum circuit simulation framework [51][55] (latest developments of this methodology is presented in Appendix B), so that it can support fault injection that helps evaluating the dependability attributes [6] with relevance for quantum circuits. The basic classical fault injection techniques are the starting point of our quantum methodology; therefore, we will emphasize only the differences dictated by the quantum nature of the processed information.

Quantum entanglement is the most important quantum feature that influences fault injection. The reason is clear, it is impossible to have a real structural description for the circuit in the presence of entanglement. Because of the fact that entanglement influences the way that fault injection is performed (by structural or behavioral architectures), a natural question is how can we involve the bubble bit technique, so that structural fault injection is still possible in the presence of entanglement. One robust answer comes from the stabilizer formalism [30][18], but the recently developed bubble bit technique [56] can also be adapted to error injection, so that the consequences of such this methodology are extended in order to approach fault tolerance assessment.

This chapter presents the achievements in classical hardware FTAM assessment, identifies the features that can be adapted to our quantum computational needs, and then sketches the basic guidlines for the *QUantum ERror Injection Simulation Tool* (or QUERIST) which is an extension of our Bubble Bit HDL-based Quantum Circuit Simulation Tool (i.e. features error injection). The implications of QUERIST development – for the Bubble Bit approach – are indicated in the last part of this chapter, along with the chapter summary.

5.1 Fault Injection in Classical Hardware

5.1.1 Motivation

The usage of simulated fault injection was proven to be very effective in classical hardware [61]. It provides means for *dependability validation and assessment* with the possibility of taking into account all the dependability attributes as defined by [6]. As shown by Rimen *et al.*, the simulated fault injection has several advantages over hardware fault injection:

- it can be used within the design process, as indicator to be taken into consideration for making decisions;
- it can also be used in validating existing circuit implementations;
- it can be employed on different design abstraction levels, by making use of any of the available Hardware Description Languages (HDLs);
- it provides means for assessing fault and error latencies and cover;
- it provides tools for approaching the study of the so-called "chain of threats" (i.e. the path for error propagation through the hierarchical abstraction levels);
- it is a less expensive technique.

Summarizing, according to the most relevant publications [3][4][61][42][11] the Software Fault Injection-Based tools are used for the following purposes:

- validation of FTAMS;
- validation of error models that were assumed in the design process of the implemented fault tolerant system;
- estimation of fault coverage;
- investigation of error propagation paths and mechanisms, along with relating the results to the corresponding design abstraction levels;
- incorporation of fault injection-based FTAM assessment in an unified design environment (this "ultimate goal" according to [43]).

The last item from the above purpose listing indicates HDLs (and VHDL in particular) as the best possible simulation framework for the following reasons:

- the HDLs are consecrated tools in classical digital design;
- the abstraction hierarchy is covered by their description capabilities;
- the description can be made from both behavioral and structural views;
- the HDLs are already used for testing and validation purposes in classical hardware design
- VHDL is already used as an integrated framework in classical hardware, which brings together description, simulation, synthesis and optimization, fault injection, and simulated testing (for example, see the MEFISTO tool [23]).

The most important motivation for VHDL-based simulated fault injection is the fact that the integrated framework creates incentive for applying the so-called "white box testing" [42][43] of the developed system. This means that the controllability and observability of the system and its components (as defined by Avižienis *et al.*) is substantially increased [42][43]. As a consequence, the chain of threats becomes easier to identify, hence FTAMS responsible for fault removal and forecasting are better assessed and optimized.

5.1.2 Simulated Fault Injection Analysis

The relevant literature [11][42] defines two main methodologies for attaining simulated fault injection: those which use only available simulator commands, and those which require intervention in the *ordinary HDL code*. The term ordinary HDL code refers to the code used for system description when correct functioning (i.e. no faults present) is assumed.

As the first methodology is inflexible, and it can hardly be used for the purposes listed in Section 5.1.1, we will concentrate only on the second one (code intervention) by identifying two specific techniques:

- Insertion of *saboteur components*: these extra-components (or fault-injection components) are added with the only purpose of generating faults;
- Definition of *mutant descriptions*: in VHDL terms a mutant architecture is defined besides the normal functioning architecture; when the fault is injected, the mutant architecture is activated by means of a corresponding configuration.

The *Saboteur* Technique

This technique consists of simply adding new components to the functional system model, dedicated only to fault injection. Of course, the relevance of these components (i.e. activation occurrence) does not regard systems non-faulty functioning regime. The saboteurs can be inserted manually or automatically into the HDL source code (for VHDLs case, as an additional driver for a resolved signal), and this insertion can be performed in serial or parallel fashion. Figure 5.1 presents the way saboteurs are inserted between the driver (HDL signal source) and the receiver signal. Function $f_S(D_1 \ldots D_N)$ is used for computing the new value for the receiver signals in the case of complex saboteurs, based on input values provided by the relevant drivers.

The *Mutant* technique

The mutant technique is a component description (architecture in VHDL) that is written in order to replace the ordinary architecture. The mutant description must behave identically to the replaced one in the absence of faults, but when faults are activated the component will act according to the assumed faulty behavior. The mutant and the normal architectures are describing the same entity, therefore "share the same interface" [43]. The switching from normal to faulty component functioning is easy to implement in VHDL due to the configuration mechanism, which indicates the used architecture at the given time (see Figure 5.2.A).

The mutated description (architecture in VHDL) can be written according to the following strategies:

- insertion of saboteurs within the normal descriptions (structural or behavioral) – the most frequently used (see Figure 5.2.B);

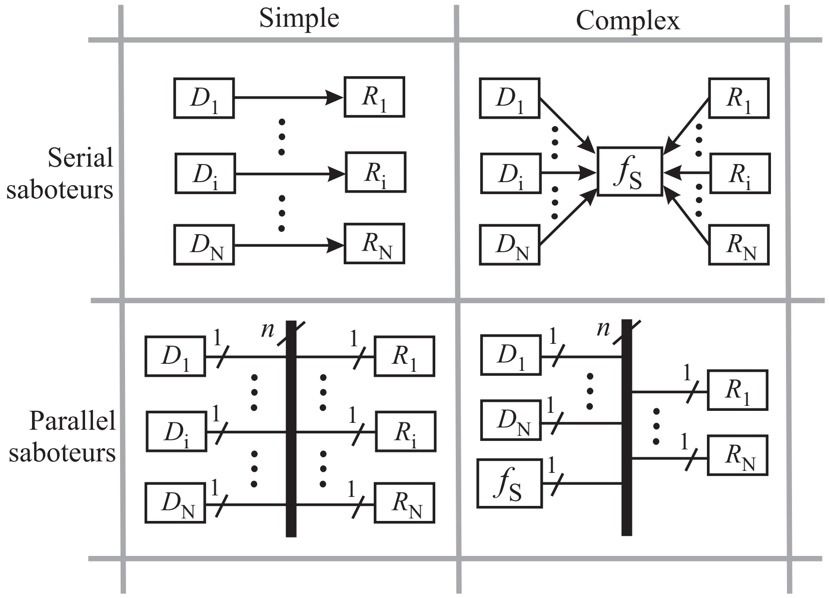


Figure 5.1: Serial/parallel insertion on simple/complex saboteurs. $D_1 \dots D_N$ are the drivers, $R_1 \dots R_N$ the signal receivers, while f_S stands for the function required by complex saboteur computation.

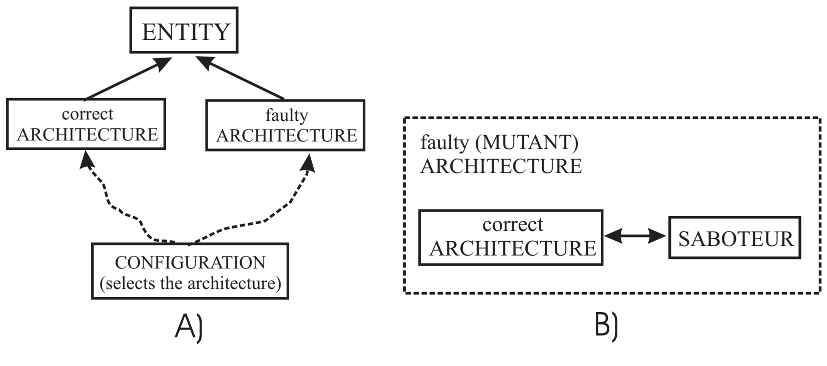


Figure 5.2: A) Illustration of configuration's role in mutant-based fault injection: the faulty architecture is selected when the fault must be activated. B) The most commonly used mutant construction: a saboteur is added to the normal (no faulty) architecture.

	mutants	saboteurs	signal manipulation	variable manipulation
a)	highest	medium	low	very low
b)	big	very big	small	small
c)	due to: amount of occuring events code size\event fault injection control complexity		due to: fault injection control complexity	
d)	Tradeoff between: modelling capability $\leftrightarrow$ time overhead simulation $\leftrightarrow$ compilation time overheads			

Table 5.1: An analysis of HDL-based simulated fault injection techniques, after Rimen *et al.* [42][43].

- mutation of subcomponents from a structural description;
- statement mutation in the behavioral descriptions;
- manual mutation: the normal description is manually altered in order to induce faulty behavior.

The first 3 strategies can be performed automatically, whereas the last one is due to user intervention.

Built-In Simulator Commands

This technique relies on deliberate altering signal and variable values, by making use of the simulator commands. When the fault must be injected according to the experimental scenario, a faulty value is forced for the signal or variable. Such a fault injection strategy is rather inflexible and simulation environment dependent. Therefore, it could be effective only if the simulator commands are designed so that they offer functionality and flexibility.

Technique Comparison

When adopting one of the available Simulated Fault Injection techniques, one has to be able to properly evaluate how appropriate is the technique to the actual needs. References [42][43][61] provides the means for such an evaluation in classical hardware, by selecting the most relevant criteria:

a) the capacity of appropriate fault modeling;

b) the effort required for setting up an experiment;

c) the simulation time overhead;

d) the capacity of setting up series of experiments (experiment campaigns).

A straightforward analysis leads to the classification from Table 5.1, according to the above listed criteria.

Context

The above described and analysed techniques are part of a more complex process of fault tolerance assessment. The entire fault tolerance estimation process consists of 3 phases [43]:

1. the simulation setup phase;
2. the experiment simulation phase;
3. the experiment data assessment phase.

The fault insertion techniques are actually applied in phase number 1, but they influence the way that next phases are performed. Summarizing, the first (setup) phase has 2 main objectives: to produce a final model (compiled and executable) of the system including all the extras required by fault injection, and to generate a control description (i.e. observed signals and states list) for each experiment and experiment campaign.

The second (simulation) phase takes as inputs the *executable model* which cannot be changed during execution, and the *experiment control list* that describes how to perform the experiments. Both inputs are produced by the previous (setup) phase. The way setup phase manage to attain the generation of both the executable model and experiment control list is presented in Figures 5.3 and 5.4. In these figures, the HDL code descriptions are presented in the darker boxes, the abstract non-code (as the Abstract Fault Model or AFM [43]) in non-shaded boxes, while the actions to be taken are written in the non-shaded ovals.

The simulations dictated by the experiment control list are controlled by a *scheduler*. The scheduler algorithm takes the *experiment control list* and the *system start state* as inputs, in order to generate the outputs which consist of the *final simulation state* and the saved *signal traces*.

The signal traces, representing the evolution of the observed signals during the simulation experiment, are feeding the third (data processing) phase. This final phase has two tasks to attain: extraction of relevant experimental data, and the actual data processing. The first goal is achieved by the so-called *data extraction rules*, while pursuing the second one requires some assumed fault tolerance models, attributes and measures [3][4].

The executable model according to AFMs

There are two methodologies for generating the abstract model according to an AFM:

a) Using a distinctive *abstract fault injector*, which controls all fault activation process.

b) Using a *classification structure*, in order to build group fault targets in classes; therefore the activation works the same for all the targets in the given class by means of a "be faulty call" [43]. The switching from the non-faulty to the faulty status of the target object (i.e. signal, variable, component) is an intrinsic object feature.

The experiment control list

The experiment control list consists of a set of batch mode simulator commands. These commands may differ due to various commercial simulator implementations. However, the command set is built so that it corresponds to a *campaign description* and a *signal observation list* [3][23] (see Figure 5.4). The campaign description consists of: the used fault set, the starting state, the termination state, and the activation conditions.

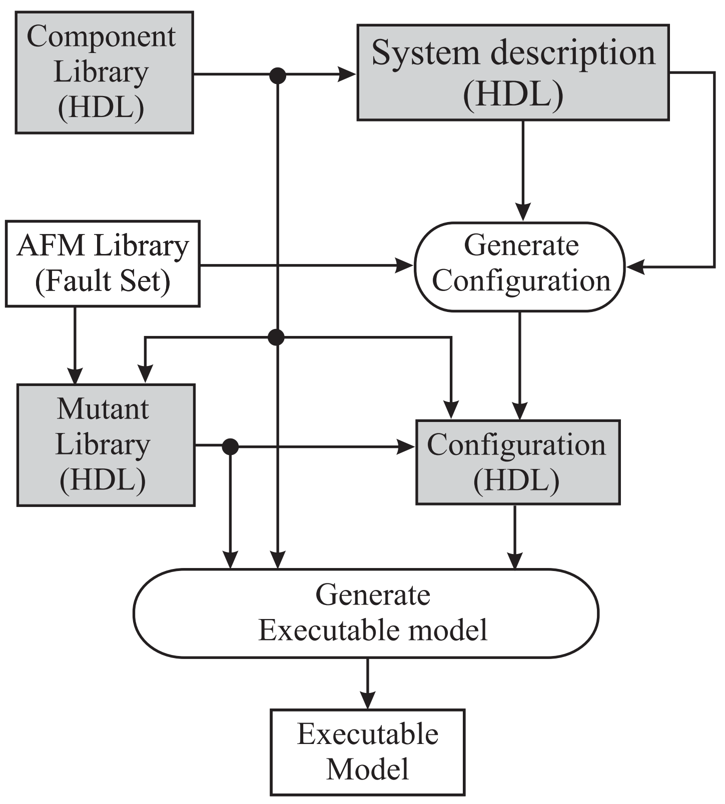


Figure 5.3: Generation of fault injection executable model within the HDL framework [43].

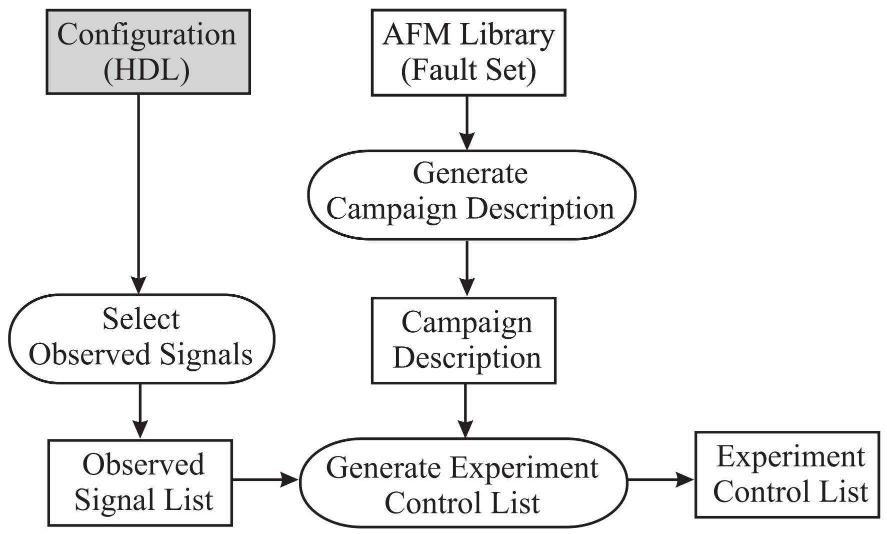


Figure 5.4: Generation of control list, for the experimental fault injection within the HDL framework [43].

5.2 Sketching the guidlines for the *QUERIST* project

With the inspiration drawn from the classical hardware HDL-based fault injection techniques, we extend our quantum circuit simulation framework. The classical fault injection methodologies can be mapped without intervention, so that the HDL framework supports fault injection into quantum circuit simulations. Of course, we cannot expect any efficiency from such an approach. Therefore, the right solution would be to adapt those methodologies to one of the available efficient simulation frameworks [55][56][59][60].

This report will describe the guidelines of a bigger software project that fosters simulated fault injection techniques in quantum circuits; the project is called *QUantum ERror Injection Simulation Tool* or *QUERIST*. In this description, only the adaptation issues will be emphasized, so we present the decisions that were made so that quantum computational constraints and specific problems are solved.

The overview of the QUERIST project is presented in Figure 5.5. In the classical approach there are 3 cycles; likewise the quantum version has the initialization, simulation, and data computation cycles. The first cycle takes the quantum circuit HDL description as an input. Also, there are 2 abstract (i.e. theoretical assumption) inputs: the HDL model and the assumed error model. The first one influences how the HDL description is presented, while the second one dictates the test scenario. In Chapter 2 we have presented the theory of fault tolerant quantum computation, along with the most commonly assumed error model: random faults, no time or space-correlated errors. QERIST endorses this error occurrence model, which means that the test scenario has to deal with defining the start and the stop simulation states because all the signals must be observed (all qubits are equally prone to error). References [54][55][56] are documenting the HDL modeling of quantum circuits in order to attain efficient simulation.

The outputs of the first cycle, which are also inputs for the simulation cycle consist of a test scenario (basically a description of when simulation starts and when it ends), and an executable HDL model with the corresponding entanglement analysis (i.e. the HDL description according to the structural architecture dictated by the bubble-bit encoded quantum states, see Appendix B [56]). The output for the second cycle is the time diagrams of all qubits, from the start to the stop state.

Special designed rules will extract the useful information from the raw, bubble-bit-represented, qubit traces. The entanglement analysis and the quantum computation reliability theory are used in order to compare the correct qubit values with the extracted values. The result of that comparison results in computing the probabilistic accuracy threshold value, in the third cycle.

5.3 Specific problems in the quantum environment

This section explains how to perform simulated quantum fault injection, within the HDL bubble bit [55][56] framework, by following the rules given by quantum error model theory [39].

Fault injection according to the above stated constraints means that the bubble bit simulation model as presented in Figure B.11 from Appendix B, has to be modified as Figure 5.6 shows.

Figure 5.6 shows that fault injection is performed just before the bubble bit technique is applied. Also, fault injection is performed only if the ”random number generator” dictates so. The way fault injection is triggered, its nature, and the way it is implemented is part of the so-called ”Setup phase”. This phase is similar to the setup phase from classical hardware fault injection. We also have a simulation phase which corresponds to running the experiment according to the scenario that was set in the setup phase. In the end, the data processing phase uses the simulation signal trace results, in order to compute the appropriate reliability measure.

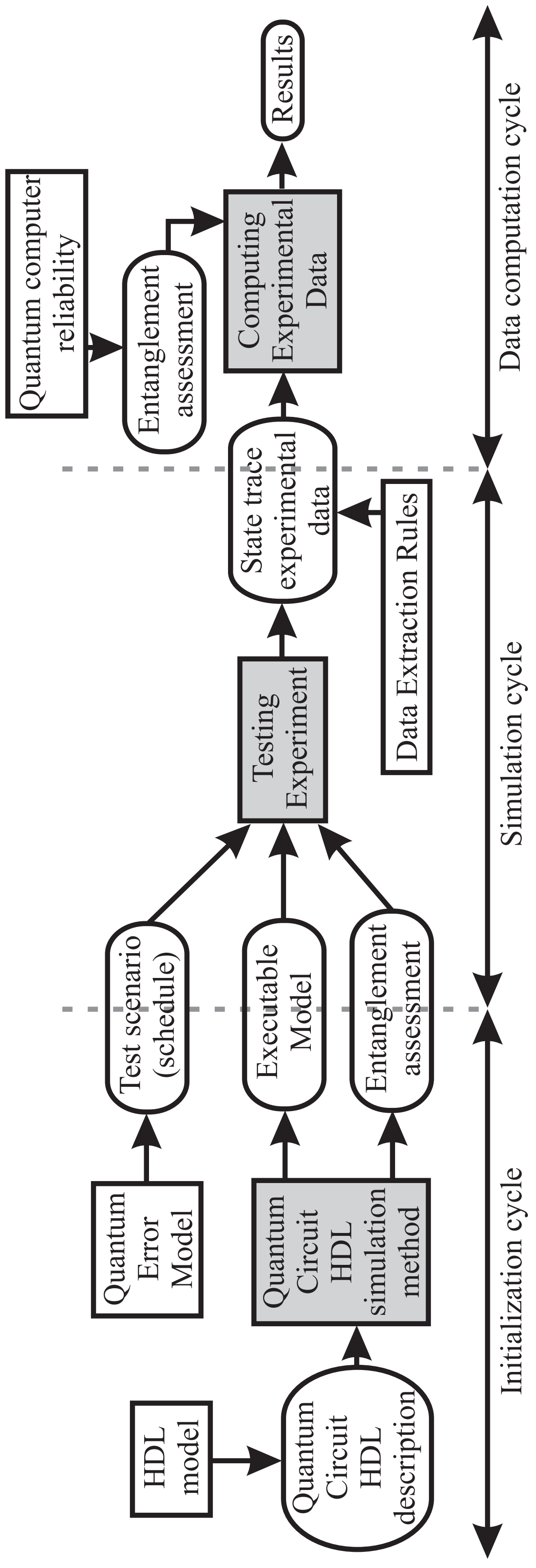


Figure 5.5: An overview of the QUERIST project.

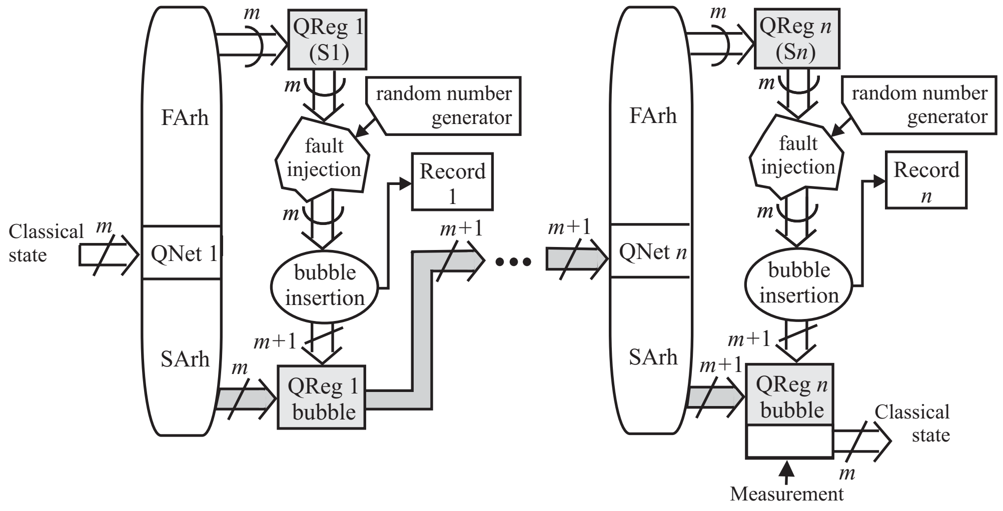


Figure 5.6: The bubble bit HDL simulation model, when fault injection is applied according to the error and fault occurence models presented in [39].

5.3.1 Setup phase

Injecting a fault, in our simulation framework [54][55][56], consists of accordingly modifying the quantum state matrix:

$$|\psi_S\rangle = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{2^n-1} \end{bmatrix}. \tag{5.1}$$

When the fault is a bit-flip, then the fault injection means that we rearrange the matrix elements, whereas for the phase shift some matrix elements will be multiplied with -1. In the bit flip case the elementary operation is exchanging values between two matrix positions: $a_i \leftrightarrow a_j$ for $i \neq j$. This allows building an exchange function that operates on blocks of matrix elements:

$$Exchange\left[(u_0, \dots u_{w-1}), (v_0, \dots v_{w-1})\right] \Leftrightarrow u_i \leftrightarrow v_i \text{ for every } i = \overline{0, w-1}. \tag{5.2}$$

Suppose we have a quantum state on n qubits, $|q_{n-1} \dots q_2 q_1\rangle$, then if a fault occurs on qubit k, and that fault is a bit-flip, then we will execute the following algorithm:

Bit-flip fault injection

```
For $i := 0$ to $2^{n-k}$
    $Exchange\left[\left(a_{i\cdot 2^k}, \dots a_{(i+1)\cdot 2^k-1}\right), \left(a_{(i+1)\cdot 2^k}, \dots a_{(i+2)\cdot 2^k-1}\right)\right]$
End For
```

When the nature of the error is phase shift, the corresponding algorithm is

Phase-shift fault injection

```
For $i := 0$ to $2^n - 1$
    If $i$ mod $2^k \geq 2^k - 1$ Then $a_i := (-1) \times a_i$
End For
```

We have settled the way the error injection is performed, but how is it going to be triggered? According to the fault occurrence model [38][39] it has to be a random triggering. Therefore, we have to use a random number generator.

For a quantum state, we use the generator for the first time in order to find out if an error occurs. Then, we use the random number generator for selecting one of the following fault types: bit-flip, phase-shift, both faults.

When first used, the generator returns the number r_1. If $r_1 < n_\xi$ (for a n_ξ given by a fixed error rate), then we have a fault. We start the number generator again – yielding r_2 – and the selected fault nature is set by the following equation:

$$r_2 = \begin{cases} 0 \le r_2 \le \frac{1}{3} & \text{we have a bit-flip} \\ \frac{1}{3} < r_2 \le \frac{2}{3} & \text{we have a phase-shift} \\ \frac{2}{3} < r_2 \le 1 & \text{we have both bit-flip and phase-shift} \end{cases} \tag{5.3}$$

For each simulated gate, when the fault is triggered the actual injection is performed on the processed state. The gate fault is triggered the same way the state fault is triggered: teo random numbers are deciding if we have a fault, and the nature of the fault. The bit-flip fault for a gate will have the effect of inducing a bit-flip fault on the target qubit. Instead, the gate phase-shift not only induces the phase-shift fault on the target qubit, but also spreads the error on all the source qubits. Figure 5.7 presents these cases (a) and b) respectively), which are considered by taking into consideration the quantum fault tolerance problems described by Preskill [38][39].

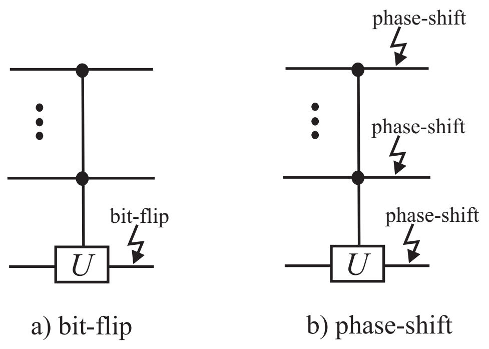


Figure 5.7: The effect of faulty gate operation on the processed qubits: a) gate bit-flip fault, b) gate phase-shift fault.

5.3.2 Simulation phase

This subsection will take an example of an error correcting quantum device, and show how fault injection simulation actually works on this circuit. We use a coding technique that replaces 1 qubit with a cluster of 3 qubits. The qubit basis state $|0\rangle$ is encoded as $|000\rangle$, and $|1\rangle$ as $|111\rangle$. For example, state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ will become $|xyz\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

If a bit-flip error occurs, then the error is indicated by the syndrome $|s_1 s_2\rangle$, where $s_1 = x \oplus z$ and $s_2 = y \oplus z$. The syndrome value indicates the fault: $|10\rangle$ means bit-flip on x, $|01\rangle$ on y, $|11\rangle$ on z, and $|00\rangle$ indicates that there is no error. The entire circuit is presented in Figure 5.8, and we start with state $|p0\rangle_{correct} = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ which is affected by a fault on qubit y: $|p0\rangle = \frac{1}{\sqrt{2}}(|010\rangle + |101\rangle)$. The evolution of the bubble-bit quantum state representation throughout the circuit simulation is described in the following equations:

$$|p0\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|0\rangle_{s_2}+\boxed{rec_0} \tag{5.4}$$

$$|p1\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|0\rangle_{s_2}+\boxed{rec_1} \tag{5.5}$$

$$|p2\rangle \equiv |p1\rangle \tag{5.6}$$

$$|p3\rangle \equiv |p0\rangle \tag{5.7}$$

$$|p4\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|0\rangle_{s_2}+\boxed{rec_2} \tag{5.8}$$

$$|p5\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|1\rangle_{s_2}+\boxed{rec_2} \tag{5.9}$$

$$|p6\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|1\rangle_{s_2}+\boxed{rec_0} \tag{5.10}$$

$$|p7\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|1\rangle_{s_2}+\boxed{rec_3} \tag{5.11}$$

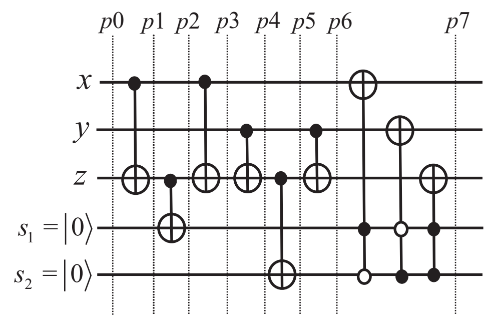


Figure 5.8: Circuit for singular bit-flip error correction.

The corresponding records are presented in Figure 5.9.

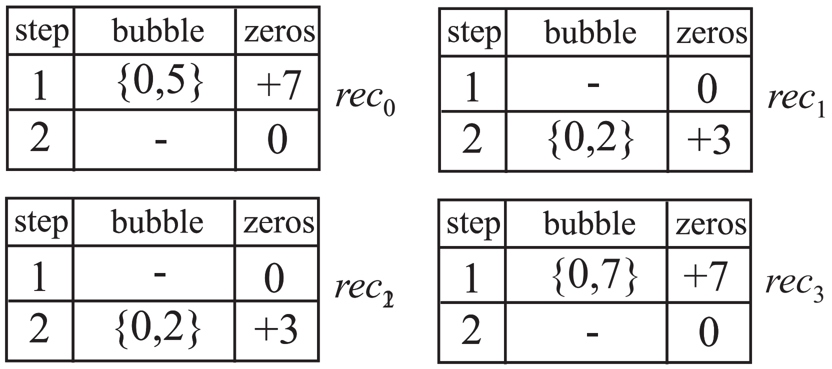
This indicates that state $|p7\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$, therefore the inflicted error has been corrected.

5.3.3 Data processing phase

Suppose that, at simulation time t we observe signals $\{s_0, s_1, \ldots s_{n-1}\}$. Each such state has a bubble bit description. If s_i is on k_i qubits, the bubble-bit representation is given by the following equation:

$$s_i = |qb_0\rangle \otimes |qb_1\rangle \otimes \ldots |qb_{k_i}\rangle + \boxed{rec_i}. \tag{5.12}$$

In our analysis, s_i is the state observed during non-faulty simulation, so for the same state in a faulty environment we will have the bubble expression given by:



step	bubble	zeros
1	{0,5}	+7
2	-	0

rec_0

step	bubble	zeros
1	-	0
2	{0,2}	+3

rec_1

step	bubble	zeros
1	-	0
2	{0,2}	+3

rec_2

step	bubble	zeros
1	{0,7}	+7
2	-	0

rec_3

Figure 5.9: Bubble records produced by simulating error correction with the circuit from Figure 5.8.

$$s_i^* = |qb_0^*\rangle \otimes |qb_1^*\rangle \otimes \ldots |qb_{k_i}^*\rangle + \boxed{rec_i^*}. \tag{5.13}$$

For validation of the quantum FTAMs, we need to compare s_i with s_i^*. This can be done with the operator presented in the following equation:

$$\text{dif}\,(s_i, s_i^*) = \begin{cases} 1 & \text{if } |qb_i\rangle \neq |qb_i^*\rangle; \forall i = \overline{0, k_i} \text{ or } rec_i \neq rec_i^* \\ 0 & otherwise \end{cases} \tag{5.14}$$

This means that the total number of overall state errors at simulation time t is

$$e_t = \sum_{i=0}^{n-1} \text{dif}\,(s_i, s_i^*)\,. \tag{5.15}$$

The error rate on the overall observed states at moments $t_0, t_1, \ldots t_{m-1}$ will be given by:

$$\xi_{sim} = \frac{1}{m} \sum_{j=0}^{m-1} e_{t_j} \tag{5.16}$$

As pointed out in references [38][39], the used FTAMs are valid if the relationship between the experimental ξ_{sim} and the assumed singular error rate ξ is of the order:

$$\xi_{sim} \sim \xi^2. \tag{5.17}$$

5.4 Summary

This chapters objective was to produce a comprehensive overview on the simulated fault injection in classical digital circuits, so that to identify the opportunities of extending the available quantum circuit simulation framework [56] to fault injection.

Therefore, we propose the theoretical guidelines of a bigger project called QUERIST, designed to attain quantum FTAMs validation by simulated fault injection. The proposed guidelines guarantee the success of such an enterprise if the error and fault occurrence models are those considered in the available literature concerning quantum fault tolerance theory [24][25][18][19][38][39]. The problem of setting a framework that assures the flexibility of implementing whatever fault model remains opened.

Also, an opened question is what other reliability and availability measures to take into consideration, besides the accuracy threshold, and how to compute these parameters within the simulation framework.

Another problem is finding out if there is a better way to inject gate errors. The proposed solution is an approximation of what is actually happens, but is the only acceptable way if the simulation framework uses the bubble-bit technique. The advantage of our methodology is that assures simulation time efficiency, as proven in [56].

Chapter 6

Conclusion and Future Work

In this report, there are a lot of aspects for which the stake is very high. First of all, we have the improvement of the reliability of quantum computation by increasing the accuracy threshold. Dependability is a critical issue in quantum computation [38], and the prospect of a feasible quantum computer relies mostly on the advances made in this particular field. If the error rate exceeds the accuracy threshold, then reliable quantum computation is unconceivable, because after a number of steps the errors will ruin the computation. We propose a technique, proving to be effective on the considered examples. Thus, we employ the reconfigurable Quantum Gate Arrays in order to raise the accuracy threshold to such an extent that arbitrary long reliable quantum computation is possible. The qualitative assessment provided in this report proves that the rQGA technique is successful in achieving its stated goal.

We have investigated the way to implement a quantum version of evolvable hardware. It was proven by years of experience that this computational paradigm (reconfigurable hardware + genetic algorithms) is appropriate for designing robust adaptive digital systems [27][34]. The quantum implementation required two aspects be addressed: the actual structure of the reconfigurable Quantum Hardware (rQHW) and the circuit implementation of the Genetic Algorithm that dictates the configuration state. The first aspect was defined by describing the rQGA in the fault-tolerant context, along with the constraints imposed by the inner nature of quantum mechanics [9][16]. But in defining the second one, it turned out that any Genetic Algorithm can be implemented, so that it is runned in just $\mathcal{O}\left(\sqrt{n}\right)$ steps.

Chapter 5.1.1 is dealing with continuing our simulation framework description. The continuation refers to adapting our methodologies to the fault injection goal. Simulated fault injection is extensively used in classical digital circuit design as an indicator to whether the design solutions meet the reliability requirements (i.e. validation of fault tolerance algorithms and methodologies – FTAMs). We capitalize on the available classical achievements, by taking into account – also – the specific quantum computation error and fault occurrence model, in order to produce a quantum interpretation of the VHDL-based MEFISTO tool [42][43]. The examples provided with the experimental results clearly indicate the fact that fault injection could be implemented straightforwardly within the bubble-bit framework (i.e. avoidance of entangled representations).

6.1 Report contributions

The original contribution provided in this report is basically centered on the following issues:

1. An engineering standpoint critique over the field of fault tolerant quantum computation, which

clearly points to the rQHW solution.

2. The rQGA structure, designed for optimal implementation of the fault tolerant quantum circuits;
3. The quantum configuration of the rQGA, so that the accuracy threshold is improved to a point where arbitrary long reliable quantum computation is possible (the effects are comparable with those obtained with the concatenated coding technique).
4. The implementation of Evolvable Quantum Hardware.
5. The demonstration that any Genetic Algorithm can be run in $\mathcal{O}\left(\sqrt{n}\right)$ steps. This result is very important for the field of quantum computing, as it indicates that there are other aspect of classical computation that could be more efficiently approached in the quantum computational framework.
6. The extension of our original quantum circuit simulation framework, in order to attain simulated quantum fault injection that is used for assessing the relevant measures of quantum reliability.

6.2 Future work

This report is presenting achievements that are only sketching further research directions. All of them concern the field of fault tolerant quantum computation:

- building the QUERIST project, for the automatic (simulated) assessment of fault tolerance measures, within the HDL, bubble bit, quantum circuit simulation framework;
- a quantitative evaluation of the rQGA fault tolerant solution, with the QUERIST project;
- integrating QUERIST and the bubble-bit framework with the available automated quantum circuit synthesis tools (the most effective ones are employing Genetic Algorithms).

Appendix A

Useful configurations

A.1 Encoder with classical configuration

The qubit Steane encoder circuit from Figure 2.8 (a particular case of stabilizer encoding) is configured from one basic *rQGA* cell (Figure 3.11) with a configuration given the way it was specified in Equations 3.9, 3.10, 3.11, and 3.12. The classical configuration is not necessary, and the input-output size is $n = 7$ qubits.

$$|config\rangle^{0}_{basic\,cell} = |\psi\rangle^{Steane}_{conf} \otimes |bit\ string\rangle^{Steane} \tag{A.1}$$

$$|bit\ string\rangle^{Steane} = |0\rangle^{\otimes 35} \tag{A.2}$$

$$\begin{aligned} |\psi\rangle^{Steane}_{conf} = N^{Steane}_{0} = \; & \underbrace{|1110000\rangle \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}} \otimes \\ & \underbrace{|0\rangle^{\otimes 15} \otimes |101000\rangle}_{t^{0}_{i,j}} \otimes \\ & \underbrace{|001011011100111\rangle \otimes |0\rangle^{\otimes 6}}_{t^{1}_{i,j}} \otimes \\ & \underbrace{|1\rangle^{\otimes 7} \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}} \end{aligned} \tag{A.3}$$

A.2 Stabilizer code with Steane ancilla

In this section we will provide a configuration for building the single quantum bit-flip error correcting circuit with Steane ancilla coding from Figure 2.13. The basic cell partition of this circuit is presented in Figure A.1.

The configuration states for the two 14-qubit basic cells is given in the following equations. For the first cell (”Cell 0” in Figure A.1) we have:

$$|config\rangle^{0}_{basic\,cell} = |\psi\rangle^{0}_{bft} \otimes |bit\ string\rangle^{0}_{bft} \tag{A.4}$$

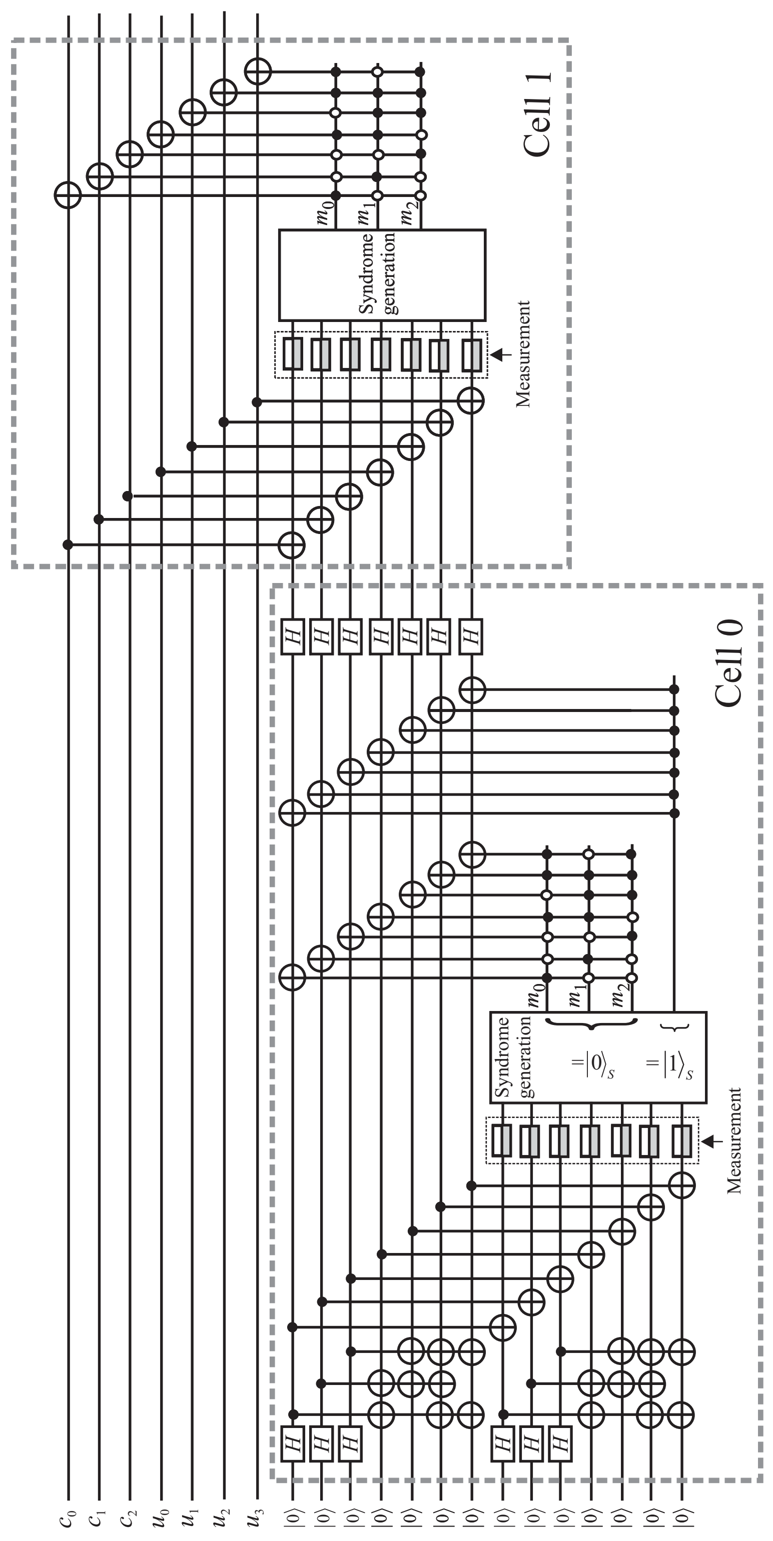


Figure A.1: The partition of the single bit-flip error correcting circuit (stabilizer code and Steane ancilla) uses 2 basic cells of 14 input qubits.

$$
\begin{aligned}
|bit\ string\rangle^0_{bft} = \ & \underbrace{|0\rangle^{\otimes 7} \otimes |1\rangle^{\otimes 7}}_{\text{measurement}} \otimes \underbrace{|0\rangle^{\otimes 14}}_{\text{Hadamard}} \otimes \\
& \underbrace{|\overbrace{classical\ circuit\ outcome}^{\text{7 qubits}}\rangle \otimes |0\rangle^{\otimes 7}}_{\text{XORs}} \otimes \\
& \underbrace{|1\rangle^{\otimes 7} \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}}
\end{aligned} \tag{A.5}
$$

$$
\begin{aligned}
|\psi\rangle^0_{bft} = \ & \underbrace{|1110000\rangle^{\otimes 2} \otimes |0\rangle^{\otimes 14}}_{\text{Hadamard}} \otimes \\
& \underbrace{|001011\rangle \otimes |0\rangle^{\otimes 7} \otimes |01110\rangle \otimes |0\rangle^{\otimes 7} \otimes |0111\rangle \otimes |0\rangle^{\otimes 7}}_{t^0_{i,j}} \otimes \\
& \underbrace{|0\rangle^{\otimes 34} \otimes |001011011100111\rangle}_{t^0_{i,j}} \otimes \\
& \underbrace{|0\rangle^{\otimes 6} \otimes |1\rangle \otimes |0\rangle^{\otimes 12} \otimes |1\rangle \otimes |0\rangle^{\otimes 11} \otimes |1\rangle}_{t^1_{i,j}} \otimes \\
& \underbrace{|0\rangle^{\otimes 10} \otimes |1\rangle \otimes |0\rangle^{\otimes 9} \otimes |1\rangle \otimes |0\rangle^{\otimes 8} \otimes |1\rangle \otimes |0\rangle^{\otimes 7} \otimes |1\rangle}_{t^1_{i,j}} \otimes \\
& \underbrace{|0\rangle^{\otimes 21}}_{t^1_{i,j}} \otimes \underbrace{|0\rangle^{\otimes 28}}_{\text{Hadamard}} .
\end{aligned} \tag{A.6}
$$

As for "Cell 1" from Figure A.1, the corresponding configuration is given in the following 3 equations:

$$
|config\rangle^1_{basic\,cell} = |\psi\rangle^1_{bft} \otimes |bit\ string\rangle^1_{bft} \tag{A.7}
$$

$$
\begin{aligned}
|bit\ string\rangle^1_{bft} = \ & \underbrace{|0\rangle^{\otimes 7} \otimes |1\rangle^{\otimes 7}}_{\text{measurement}} \otimes \underbrace{|0\rangle^{\otimes 14}}_{\text{Hadamard}} \otimes \\
& \underbrace{|\overbrace{classical\ circuit\ outcome}^{\text{7 qubits}}\rangle \otimes |0\rangle^{\otimes 7}}_{\text{XORs}} \otimes \\
& \underbrace{|0\rangle^{\otimes 14}}_{\text{Hadamard}}
\end{aligned} \tag{A.8}
$$

$$
\begin{aligned}
|\psi\rangle^1_{bft} = \ & \underbrace{|0\rangle^{\otimes 28}}_{\text{Hadamard}} \otimes \\
& \underbrace{|0\rangle^{\otimes 6} \otimes |1\rangle \otimes |0\rangle^{\otimes 12} \otimes |1\rangle \otimes |0\rangle^{\otimes 11} \otimes |1\rangle \otimes |0\rangle^{\otimes 10} \otimes |1\rangle}_{t^0_{i,j}} \otimes \\
& \underbrace{|0\rangle^{\otimes 9} \otimes |1\rangle \otimes |0\rangle^{\otimes 8} \otimes |1\rangle \otimes |0\rangle^{\otimes 7} \otimes |1\rangle \otimes |0\rangle^{\otimes 21}}_{t^0_{i,j}} \otimes \\
& \underbrace{|0\rangle^{\otimes 91}}_{t^1_{i,j}} .
\end{aligned} \tag{A.9}
$$

A.3 Quantum configuration for the encoding circuit

We provide some practical means to implement a quantum configuration (superposition state) for the encoding circuit from Figure 2.8. This circuit corresponds to one particular stabilizer code, described in Equation 2.34. Of course, we can construct many such codes, but for practical reasons (easy implementation, less gates controlled by an error-prone quantum state) we consider only those obtained by permuting the u_0, u_1, u_2, u_3 columns in the Hamming matrix H_A from Equation 2.14. Therefore, we have $4! = 24$ such distinct codes. As we will see, due to engineering problems, only some of these codes will be used, a number that is a power of 2, which makes things easier in preparing the configuration register.

Normally, for obtaining the theoretical probability of ξ^6 (see Quantum configuration from section 3.3) we have to prepare a 12 qubit quantum state given by:

$$|\psi\rangle_{12} = \frac{1}{2\sqrt{6}} \begin{pmatrix} |011110111101\rangle + |011110111110\rangle + |011111011011\rangle + \\ |011111011110\rangle + |011111101011\rangle + |011111101101\rangle + \\ |101101111101\rangle + |101101111110\rangle + |101111010111\rangle + \\ |101111011110\rangle + |101111100111\rangle + |101111101101\rangle + \\ |110101111011\rangle + |110101111110\rangle + |110110110111\rangle + \\ |110110111110\rangle + |110111100111\rangle + |110111101011\rangle + \\ |111001111011\rangle + |111001111101\rangle + |111010110111\rangle + \\ |111010111101\rangle + |111011010111\rangle + |111011011011\rangle + \end{pmatrix} \tag{A.10}$$

This 12-qubit state must be fitted in the 12 positions, marked with filled dots, of the configuration state, as shown below. The configuration state will be

$$|\psi\rangle_{conf}^{stabil} = \left(\frac{1}{2\sqrt{6}} \sum_{i=0}^{23} |N_i\rangle \right) \oplus |bit\ string\rangle^{stabil} \tag{A.11}$$

with the 'bit string' being only 0's ($|bit\ string\rangle^{stabil} = |0\rangle^{\otimes 35}$), and the structure of the superposed configure states:

$$\begin{aligned} |N_i\rangle = \ & \underbrace{|1110000\rangle \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}} \otimes \underbrace{|0\rangle^{\otimes 15} \otimes |101000\rangle}_{\text{fixed } t^0_{i,j}} \otimes \\ & |t^1_{i,j}\rangle \otimes \underbrace{|1\rangle^{\otimes 7} \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}}. \end{aligned} \tag{A.12}$$

The $|t^1_{i,j}\rangle$ contains fixed qubits and the 12 qubits that participate to the superposition state from Equation A.10, having the following basis state structure:

$$|t^1_{i,j_{basis}}\rangle = |00 \bullet\bullet\bullet\bullet 0 \bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet 000000\rangle. \tag{A.13}$$

Here, the 0's correspond to the fixed values, whereas the filled dots mark the qubits from the superposition.

If we are to prepare state $|\psi\rangle_{12}$ from Equation A.10 on the qubits highlited in Equation A.13, we have to acknowledge the fact that – on one hand – the state itself is hard to obtain due to the fact that it is a superposition of 24 (not a power of 2) basis states, and – on the other hand – there are many gates that are used at the same time by the superposed configurations, hence one gate error can affect several superposed encoding circuits. We have to came out with some engineering decision in order to deal with these problems.

Column C_1				Column C_2				Column C_3			
q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	$q_1$0	$q_1$1
0	1	1	1	1	0	1	1	1	1	0	1
1	0	1	1	1	1	1	0	0	1	1	1
1	1	0	1	0	1	1	1	1	1	1	0
1	1	1	0	1	1	0	1	1	0	1	1

Table A.1: The rows present the basis configurations which can be superposed with minimum gate usage (qubits $q_0, q_1 \dots q_{11}$ are the ordered positions of the filled dots from Equation A.13). By permutation of columns C_1, C_2, C_3 all the other distinct superposed configurations can be found.

First, we will arrange the basis state codes from Equation A.10 so that the same gate will not be used in all the superposed configurations. Four such configurations are given in table A.1. If we permute the highlighted columns C_1, C_2, C_3 we will get all the configurations from Equation A.10. However, in our heuristic engineering approach, we will use only the codes from table A.1. The problem is that when we superpose these configurations, the same gate will be used in 3 out of 4 superpositions, hence producing only an overall $\xi^{\frac{4}{3}}$ error probability order.

One solution would be to add one more level of controlled Toffoli gates $(t^2_{i,j})$ in the basic cell. This will change Equation 3.12 so that we will have:

$$N_i^{new} = |\underbrace{hh\dots h}_{2n \text{ bits}}\underbrace{t^0_{0,1}\dots t^0_{n-2,n-1}}_{\frac{n(n-1)}{2}\text{ bits}}\underbrace{t^1_{0,1}\dots t^1_{n-2,n-1}}_{\frac{n(n-1)}{2}\text{ bits}}\underbrace{t^2_{0,1}\dots t^2_{n-2,n-1}}_{\frac{n(n-1)}{2}\text{ bits}}\underbrace{hh\dots h}_{2n \text{ bits}}\rangle \tag{A.14}$$

Now, we will present $|t^1_{i,j}\rangle$ and $|t^2_{i,j}\rangle$ in detail, the way Equation A.13 prescribes.

$$|t^1_{i,j\,basis}\rangle^{new} = |00\bullet\bullet\bullet\bullet0\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet000000\rangle \tag{A.15}$$

$$|t^2_{i,j\,base}\rangle^{new} = |00\square\square\square\square0\square\square\square\square\square\square\square\square000000\rangle. \tag{A.16}$$

The orthonormal basis structure of the qubit group states marked with filled dots and boxes are presented, respectively, in the following two equations:

$$|\rho\rangle^{\bullet}_{basis} = |b_0b_1111b_21b_3b_41b_51\rangle \tag{A.17}$$

$$|\rho\rangle^{\square}_{basis} = |11b_6b_7b_81b_911b_{10}1b_{11}\rangle. \tag{A.18}$$

The positions not marked with b_i's are fixed with the corresponding binary value. Therefore, it is only necessary to present the states corresponding to the b_i qubit groups. Fortunately, it is the same state:

$$|b_0b_1b_2b_3b_4b_5\rangle = |b_6b_7b_8b_9b_{10}b_{11}\rangle = \frac{1}{\sqrt{2}}\left(\begin{array}{c}|010110\rangle+\\|101001\rangle\end{array}\right). \tag{A.19}$$

The circuit for obtaining the state in Equation A.19 is presented in Figure A.2.

The result of this configuration, for a described modification of the reconfigurable basic cell and a gate error rate of order ξ, is that the overall error probability will be $\xi^{\frac{4}{3}\cdot 2} = \xi^{\frac{8}{3}}$.

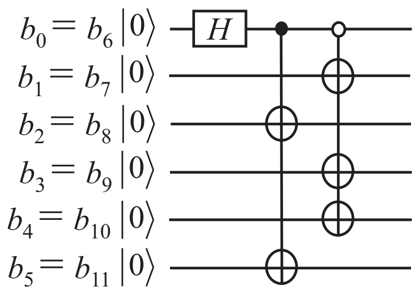


Figure A.2: Circuit for setting the 6-qubit configuration state, which is used in setting a quantum configuration for the stabilizer encoder.

Appendix B

HDL quantum circuit simulation methodology

The HDL-based original approach for simulation of quantum circuits [55] is an entanglement-aware simulation methodology, thus considering the entanglement as the main source of simulation complexity. The usage of Hardware Description Languages (HDLs) is motivated by the fact that they are able to describe in a compact manner the circuit with both structural and behavioral (functional) architectures [5], thus isolating the entanglement situations. The non-entangled information is separated inside a simulation engine with specialized algorithms, which are efficient at least for some specific states appearing in Grover and Shor algorithms [37][55].

This methodology is an enhancement only if there are *non-totally entangled states* [55]. A case study for the HDL-based framework, involving states appearing in Shors and Grovers algorithms, showed that the probability of total entanglement is rising exponentially with the number of qubits [55]. However, total entanglement could be avoided at least in some specific algorithm states by "bubble bit" insertion [55]. This technique is favorable for the structural architectures, with the expense of building some records of size $\mathcal{O}\left(n^2\right)$. Here, we present the bubble bit technique working within the HDL-based simulation framework, for some arithmetic circuits, used for Shors algorithm implementation, and also for Grover's algorithm circuits.

B.1 The HDL approach

When entanglement is not present in the processed quantum state, it is possible to describe the circuit and the states in a structural manner, employing only polynomial resources for simulation. By contrast, when entanglement is detected in the processed state, the circuit has to be described with a behavioral architecture, and exponential resources must be used in this case. That happens because, when entanglement occurs between two quantum subsystems, their overall state cannot be represented correctly as a reunion (assuming implicit tensor product state composition) of the two individual subsystem states.

In Figure B.1 an HDL simulation approach is presented. Two quantum circuits, functionally described, guard the entangled quantum state (S_3). The first quantum circuit (gate network) is having a structural description because is guarded by 2 non-entangled states (S_1 and S_2). $Cp_1 \ldots Cp_n$ are the smallest components of the quantum circuit, and $A_1 \ldots A_n$ are their corresponding architectures. A quantum register corresponding to a non-entangled state is using a '/' notation, while for the entangled case the used sign is ')'. Of course, if we are to perform a *gate-level* simulation [32][59] of the quantum algorithm implementation, then the circuit becomes a single quantum gate.

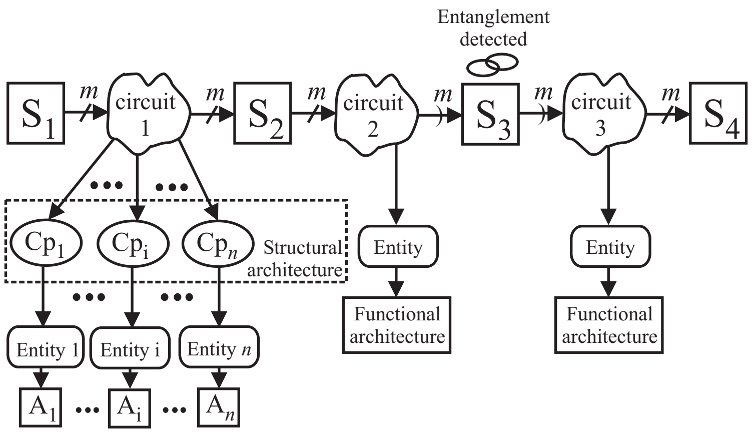


Figure B.1: Example of approaching the HDL simulation of a quantum circuit.

In order to implement this methodology, each circuit must be described both by structural and functional architectures. For a gate network, if entanglement is detected in the previous or next quantum state, then the functional architecture has to be selected to describe it; otherwise the structural architecture is chosen. Figure 2.29 presents a circuit, which can be simulated with a structural architecture (case B), but for some input states is producing entanglement, and therefore can only be simulated by functional (behavioral) architectures (case A). In Figure B.2 the bra-ket notation is used, in order to indicate the individual qubit and overall states.

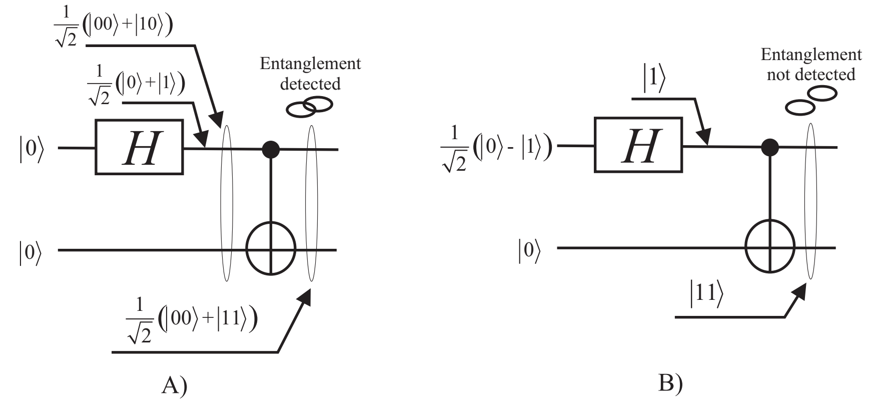

$\frac{1}{\sqrt{2}}(|00\rangle+|10\rangle)$
$\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$
Entanglement detected
$|0\rangle$
H
$|0\rangle$
$\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle)$
A)
$|1\rangle$
Entanglement not detected
$\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$
H
$|0\rangle$
$|11\rangle$
B)

Figure B.2: Situations where the same circuit (2 gates: Hadamard and XOR) is involved in an entanglement case (A), and in a non-entanglement case (B).

The matrix representation of quantum states and unitary operators is adopted; therefore the quantum states are type array (of complex) signals [55]. The data structure for HDL-simulation is designed so that the circuit is capable of processing both array of qubit states (the structural case) and overall states (the behavioral case), depending on entanglement detection [15]. For instance, if the circuit is processing a 16-qubit state, then the structural architecture will handle 16 $[2 \times 1]$ matrixes (2^5 complex numbers), whereas the behavioral architecture will have to handle 1 $\left[2^{16} \times 1\right]$

matrix (2^{16} complex numbers). Figure B.3 gives an appropriate data structure example described in VHDL.

```
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
package qupack is
 -- the qubit state representation
 type qubit is array(0 to 1)of complex;
 -- array of qubits representation
 type qubit_vector is array(natural range<>)of qubit;
 -- quantum register overall state representation
 type quregister is array(natural range<>)of complex;
 -- data type for simulation of 2-qubit circuits
 -- when ent=true we have entanglement and 'qr' field
 -- will be taken into consideration
 type qudata is record
   qr:quregister(0 to 3);
   qa:qubit_vector(0 to 1);
   ent:boolean;
 end record;
end qupack;
```

Figure B.3: Data set example for the HDL approach.

With the data structure from Figure B.3 and the above considerations, we are able to describe the circuit from Figure B.2 with both structural and behavioral architectures. The behavioral architecture ('functional' in Figure B.4) has a group of 4 variable assignments, motivated by the fact that the overall transformation produced by the circuit is characterized with the resulted matrix from Equation B.2. The effect of the Hadamard gate over the overall input state is given by

$$H \otimes I = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}. \tag{B.1}$$

Applying the XOR gate over the 2 qubits with the source qubit outputted by the Hadamard gate and the target qubit not being transformed will have the following effect:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}. \tag{B.2}$$

When structural description is possible, the circuit can be reduced to the form given by Figure B.5(A), with $U_0, U_1, U_2 \ldots U_{n-1}$ being 1-qubit unitary transformations, and $q_0, q_1, q_2 \ldots q_{n-1}$ individual qubits. For the case in Figure B.2(B), the structural description is possible, because the circuit can be reduced as Figure B.5(B) shows. This is, in fact, the motivation for the architecture 'structural' of entity 'circ-ex' (see Figure B.4).

```
entity circ_ex is
 port(q1:in qudata;q2:inout qudata);
end circ_ex;
architecture functional of circ_ex is
begin
process(q1)
variable t:quregister(0 to 3);
variable r:qudata;
begin
if (q1.ent) then
 l1:for i in 0 to 3 loop t(i):=q1.qr(i);
 end loop l1;
else t:=tensor_product_1(q1.qa(0),q1.qa(1));
end if;
r.qr(0):=(1.00/sqrt(2.00))*(t(0)+t(2));
r.qr(1):=(1.00/sqrt(2.00))*(t(1)+t(3));
r.qr(2):=(1.00/sqrt(2.00))*(t(1)-t(3));
r.qr(3):=(1.00/sqrt(2.00))*(t(0)-t(2));
r.ent:=true;
q2<=r after 20 ns;
end process; end functional;
architecture structural of circ_ex is
component Hadamard_gate
 port(qi:in qubit;qo:out qubit);
end component;
component qxor
 port(qs:inout qubit;qti:in qubit;qto:out qubit);
end component;
begin
c1:Hadamard_gate port map(q1.qa(0),q2.qa(0));
c2:qxor port map(q2.qa(0),q1.qa(1),q2.qa(1));
end structural;
_________________________________________________

entity Hadamard_gate is
 port(q1:in qubit;qo:out qubit);
end Hadamard_gate;
architecture hga of Hadamard_gate is
begin
qo(0)<=(1.00/sqrt(2.00))*(qi(0)+qi(1))after 10ns;
qo(1)<=(1.00/sqrt(2.00))*(qi(0)-qi(1))after 10ns;
end hga;
_________________________________________________

entity qxor is
 port(qs:inout;qti:in qubit;qto:out qubit);
end qxor;
architecture qxa of qxor is
begin
process(qs,qti)
begin
assert (qs(0).im=0.00 and qs(0).re=0.00) or
       (qs(1).re=0.00 and qs(1).im=0.00)
report "XOR's output will be entangled"
severity failure;
if qs(0).im=0.00 and qs(0).re=0.00 then
 qto(0)<=qti(1) after 10 ns;
 qto(1)<=qti(0) after 10 ns;
elsif qs(1).im=0.00 and qs(1).re=0.00 then qto<=qti;
end if; end process; end qxa;
```

Figure B.4: Some relevant pieces of VHDL code, describing the circuit from Figure B.2.

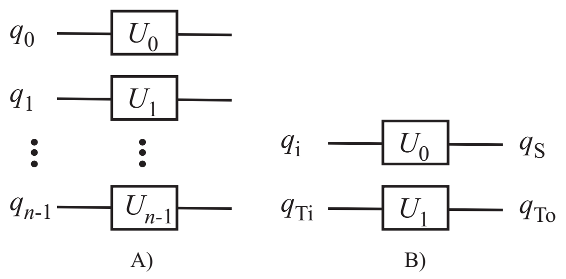


Figure B.5: A) Reduction of any n-qubit quantum circuit when entanglement is not present. B) reduction of the circuit from Figure B.2 situation (B), with $q_\text{i} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ the input source, the target input $q_\text{Ti} = |0\rangle$, the target output $q_\text{To} = |1\rangle$, and the output source $q_\text{S} = |1\rangle$.

B.2 Methodology effectiveness

B.2.1 Automated procedure

The simulation methodology is automated by using qubit group extraction algorithms. These algorithms are responsible for detecting *total entanglement* (this notion will be explained in the next subsection), and for extracting automatically the non-entangled qubit groups, if such is the case, thus setting the entanglement flag (the 'ent' field in the Figure B.3 example) accordingly.

Efficient automated extraction of non-entangled qubit group states is not conceivable unless we have some *a priori* information about the overall state [51]. When dealing with states from certain points in the circuits implementing specific algorithms, we have that knowledge because of the characteristic form these states exhibit. For example, the states dictated by Shor's algorithm [46] arithmetic circuits, (i.e. all the states except those dictated by the last algorithm step – the Quantum Fourier Transform) are described by the following equation:

$$|\psi_{arithmetic}\rangle = \xi \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{2^n-1} \end{bmatrix} \tag{B.3}$$

where n is the number of qubits, $\xi \in \mathbb{C}$, and $b_i \in \mathbb{B} = \{0, 1\}$.

Only if the extraction algorithms are not successful, the employment of our simulation methodology will not be an improvement. Such algorithms do exist, for states appearing in Grover and Shor algorithms, and for an arbitrary state [55][37]. As for the states involved in Shor's algorithm arithmetic operations, they are not totally entangled iff, in the set $\{b_0, b_1, \ldots b_{2^n-1}\}$ of Equation B.3 elements, there is a $d \in \mathbb{N}$ so that all the $\left(b_{k\cdot 2^d}, b_{k\cdot 2^d+1, \ldots b_{(k+1)\cdot 2^d-1}}\right)$ subsets (with $k = 0..2^{n-d} - 1$) are in the same set of two $2^d \times 1$ matrixes with binary elements: one matrix with all elements being zeros, and the other with at least one non-zero element [55]. With theoretical result, an extraction algorithm can be developed. It detects and extracts d-qubit ($1 \leq d < n$) groups that are not entangled with the rest in the register (see Figure B.6). Here, function *Compute-set* returns the decimal correspondent of the binary information encoded by the 2^d-dimensional subset. The algorithm will return q (the state of the d-depth qubit group) and Q (overall state of the remained $n - d$ qubits).

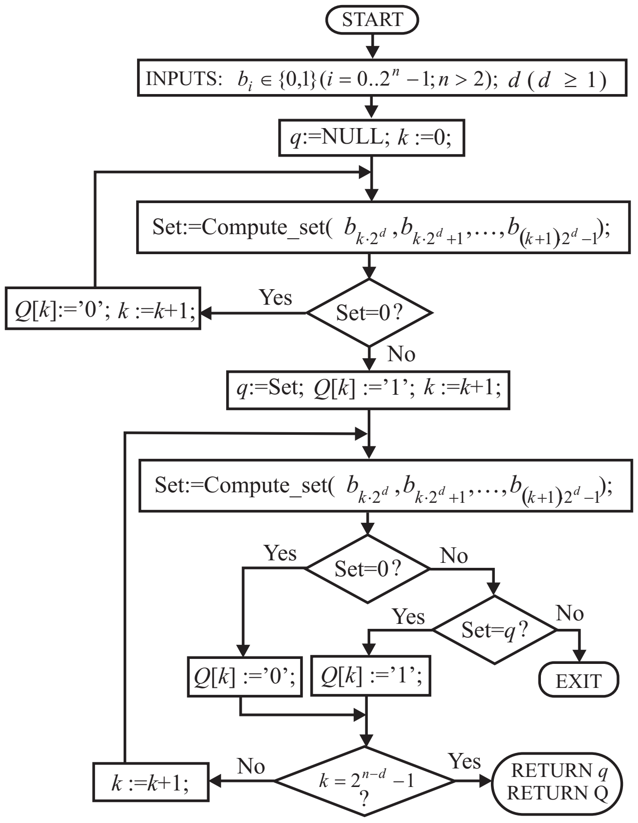

START
INPUTS: $b_i \in \{0,1\} (i = 0..2^n - 1; n > 2)$; d ($d \geq 1$)
q:=NULL; k :=0;
Set:=Compute_set($b_{k \cdot 2^d}, b_{k \cdot 2^d + 1}, \ldots, b_{(k+1)2^d - 1}$);
Set=0 ?
Yes
$Q[k]$:='0'; k :=k+1;
No
q:=Set; $Q[k]$:='1'; k :=k+1;
Set:=Compute_set($b_{k \cdot 2^d}, b_{k \cdot 2^d + 1}, \ldots, b_{(k+1)2^d - 1}$);
Yes
Set=0 ?
No
Yes
Set=q ?
No
$Q[k]$:='0';
$Q[k]$:='1';
EXIT
k :=k+1;
No
$k = 2^{n-d} - 1$?
Yes
RETURN q
RETURN Q

Figure B.6: Qubit group extraction algorithm (with depth d), for states dictated by Shor algorithm arithmetic circuits.

B.2.2 Entanglement analysis

Even though the entanglement affects all the state qubits (*complete entanglement*), there could be a qubit group state that is not entangled with the rest. A state is having total entanglement if there is no qubit or group of qubits that could be extracted as non-entangled. Figure B.7(A) presents an example of complete, but non-total, entanglement. Any unitary transformation which outputs the state presented in Figure B.7(A) can be reduced to the expression from Figure B.7(B).

In Figure B.6, when the extraction algorithm is successful for $d = 1$, then the entanglement is not complete, and if there is a $d < n$ so that the algorithm avoids 'EXIT' in Figure B.6, then we have a non-total entanglement situation. We use the circuit model as basic point in our *gate-level* [51] simulation methodology. Figure B.8 presents the circuit model of quantum computation, which is a rippling of consecutive quantum networks of gates and registers, with the last register being measured in order to get the result. The HDL-based, entanglement aware simulation methodology is presented in Figure B.9. The entanglement analysis units are using algorithms like the one in Figure B.6 in order to extract the non-entangled qubit groups and passes this qubit groups information to the corresponding quantum network and register. The network selects the structural architecture with the appropriate unitary transformations (as suggested by Figure B.7), while the quantum register will be represented now as a collection of independent qubit groups.

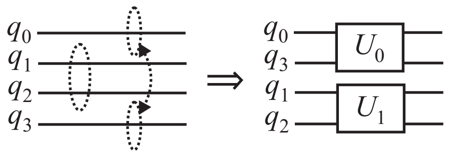


Figure B.7: A) Complete entanglement example: for qubit groups $\{q_1, q_2\}$ and $\{q_0, q_3\}$ there is total entanglement inside the group, but there is no entanglement between the two groups; B) unitary transformation expression.

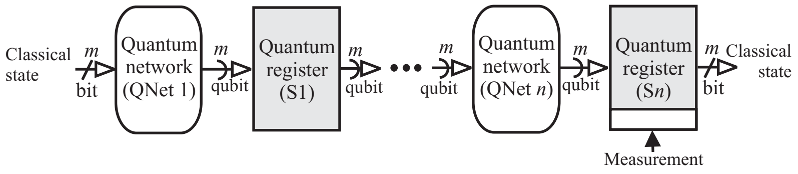


Figure B.8: The circuit model of quantum computation.

The total entanglement is the only situation in which our HDL simulation approach brings no improvement. For Shor, Deutsch-Jozsa and Grover algorithms the probability of finding non-total entanglement decreases exponentially with the number of qubits in the processed state [37][55], therefore further investigations are necessary. However, there are many algorithm situations where entanglement is not present. Such is the case for Deutsch-Jozsa algorithm with the oracle being a circuit that computes the parity of the quantum state encoded in the query register [30]. For a query register of 4-16 qubits and the oracle being the circuit computing query's parity, the simulation time evolution with the number of qubits (for both structural and behavioral architectures) is presented in Figure B.10.

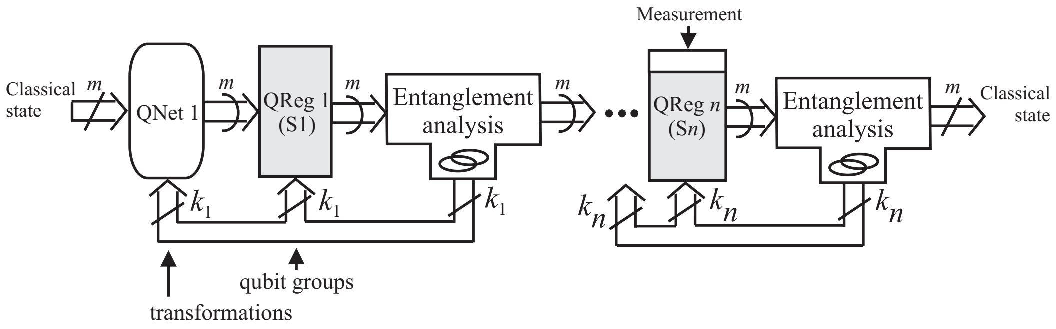


Figure B.9: The entanglement-aware quantum circuit simulation model.

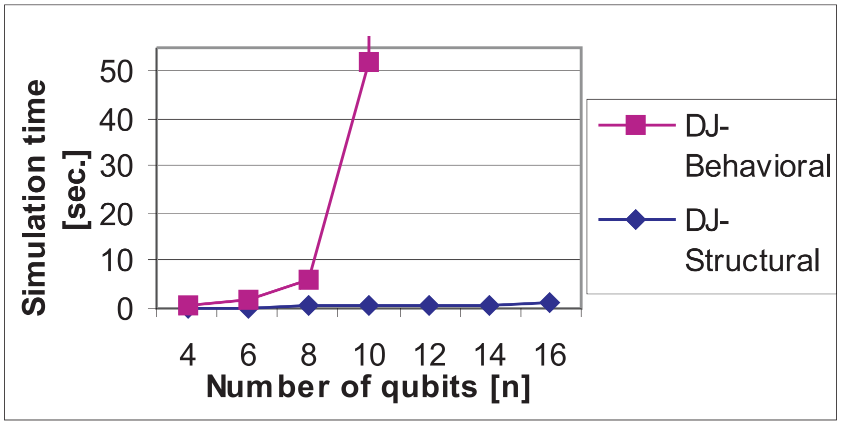


Figure B.10: Discrepancy between simulation times with behavioral and structural architectures, for Deutsch-Jozsa algorithm.

B.3 The bubble bit technique

Considering the arithmetic circuits involved in Shor's algorithm (with Grover's algorithm experiencing a similar situation [55][56]), the difference between a non-entangled and a totally entangled state could be a simple binary couple flip. Therefore we developed an algorithm that creates a new entanglement-free-represented state, in order to alter the entangled state representation by inserting appropriate values called "bubble bits" and storing their positions in the state vector. Our technique is similar to the stabilizer codes, which offer the opportunity for efficient simulation (as proven in Gottesman-Knill theorem [30]), but instead finding transformations that leaves the n-qubit state unchanged or stabilized, we produce a corresponding $n+1$-qubit state which is not entangled (it is used for simulation), and a set of memorized inserted matrix elements (the bubble records).

The purpose is to avoid the $2^n \times 2^n$ matrix expression of the n-qubit register unitary operator. After performing the bubble bit insertion procedure, the equivalent quantum network will have only 1-qubit gates, and after applying the unitary operator in this manner, the original state can be restored. Because the unitary transform is obtained with at most n [$2\times$]-size matrixes, incentive for structural (i.e. polynomial) simulation is provided.

The bubble bit insertion technique generates a new simulation model, under the form shown in Figure B.11. First, the *FArh* architectures are used for the quantum networks. These architectures are used by the quantum networks ($QNet_1 \ldots QNet_n$) from Figure B.9 simulation model, with a high probability of being functional architectures. The state output by $QNet_i$, having *FArh* as architecture, will be processed with the bubble bit procedure, and the result stored in $QReg_i$ (bubble). At this point, $QNet_i$ will have non-entangled input and output states, hence it will be described by an entirely structural architecture (computation flowing along the darker arrow in Figure B.11).

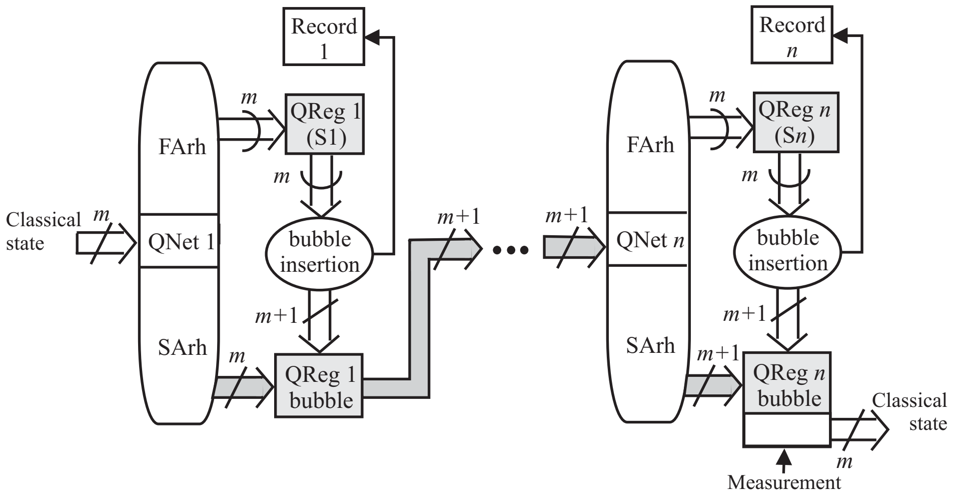


Figure B.11: Quantum circuit simulation model, when the bubble bit technique is employed.

The procedure for bubble bit insertion works as follows: every couple (b_{2k}, b_{2k+1}) from the state vector (as considered in Equation B.3) is scanned. From this equation, ξ will be ignored because all non-zero amplitudes are equal. We denote couple matrixes as: $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \hat{0}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \hat{1}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \hat{2}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \hat{3}$. When a non-$\hat{0}$ value is encountered all the other couples to be processed will have to be of either this particular value or $\hat{0}$.

The bubble insertion described in Figure B.12 must be performed until all the elements from the state vector are scanned; here the oval is the first relevant couple detected and the rectangle represents the current processed couple. The bubble bit is inserted between the bits shown in rectangles in Figure B.12. After the bubble insertion, a current processed couple (c) results along with a next couple (n) that could be already processed when no '?' sign appears. There are 2 cases where a bubble bit could also be inserted in the next couple; that happens when becomes obvious that it would be the only choice (see Figure B.12 for details). When the entire state vector is scanned and processed in this way, the extraction of one qubit (characterized by the first encountered non-$\hat{0}$) becomes straightforward, and it can be said that one bubble step is completed. Several bubble steps must be performed until all qubits are extracted.

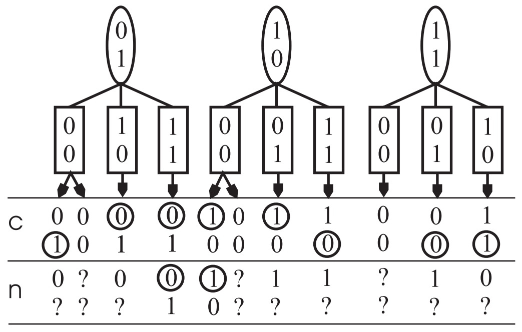


Figure B.12: Bubble bit insertion technique.

Any bubble-bit insertion will also increase the number of state matrix elements (b_i). The solution for maintaining a coherent matrix-form quantum state is to add an extra-qubit to the state representation. Thus, the number of b_i elements will be increased from 2^n to 2^{n+1} – at the first bubble step – by inserting extra 0's. The next bubble steps will require erasure of 0's, so that the matrix-form representation further complies with the quantum state coherence requirement (a k-qubit state implies 2^k vector elements in the state matrix representation).

For every bubble-bit insertion, its position inside the vector is recorded. Each bubble $\{b, pos\}$ is described by its nature ($b = 0/1$) and its position in the resulted state (pos). Performing all the necessary bubble steps requires a total of $\mathcal{O}\left(n^2\right)$ records be produced.

B.4 Experimental results

Efficient quantum gate-level simulation may be achieved by using the HDL simulation framework, at least for some particular circuit cases (Grover iteration, arithmetic circuits) [51][55]. The ability of HDLs to describe a circuit with both structural and behavioral architectures allows isolating entangled qubit cases, which are the sources of simulation complexity. Besides special algorithms for non-entangled qubit group extraction [55], the simulation methodology we developed relies on the bubble bit technique, introduced as a method of avoiding entangled representations. This method substantially (i.e. exponentially) improves simulation times with the expense of buiding some records of size $\mathbb{O}\left(n^2\right)$, as experimented for Shor's algorithm arithmetic circuits and Grover algorithm circuit.

B.4.1 Quantum arithmetic

In order to illustrate how the bubble bit technique works, we take as example the backbone of quantum arithmetic circuits: the 1-qubit full adder from Figure B.13(A). The way this add-cell could be rippled in order to build n-qubit adders is suggested in Figure B.13(B). The simulation of the 1-qubit full adder will have to take into consideration the successive states from part (A) of Figure B.13. The input state ($|\psi_1\rangle$) is not entangled, as shown in the following equation:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle. \tag{B.4}$$

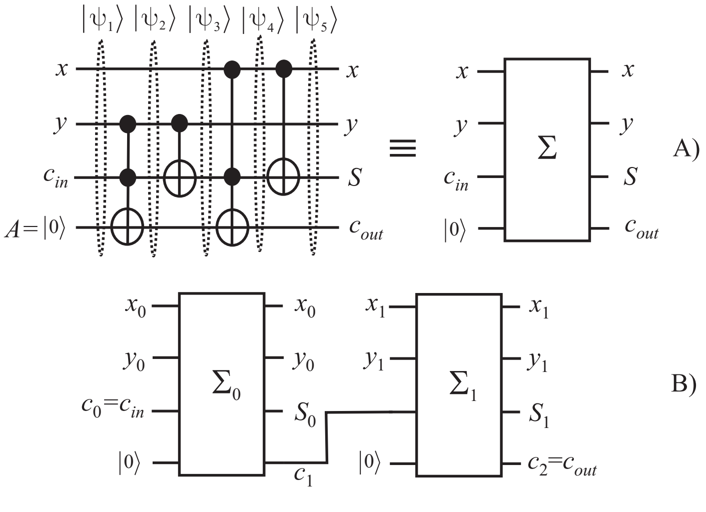


Figure B.13: A) The 1-qubit full adder; B) obtaining a 2-qubit adder from 1-qubit Σ cells.

The other states are entangled, with the last one ($|\psi_5\rangle$) being totally entangled and therefore the bubble bit technique has to be applied. As presented by Equations B.5 to B.8, the resulted state representations are identical, with only the corresponding records being different.

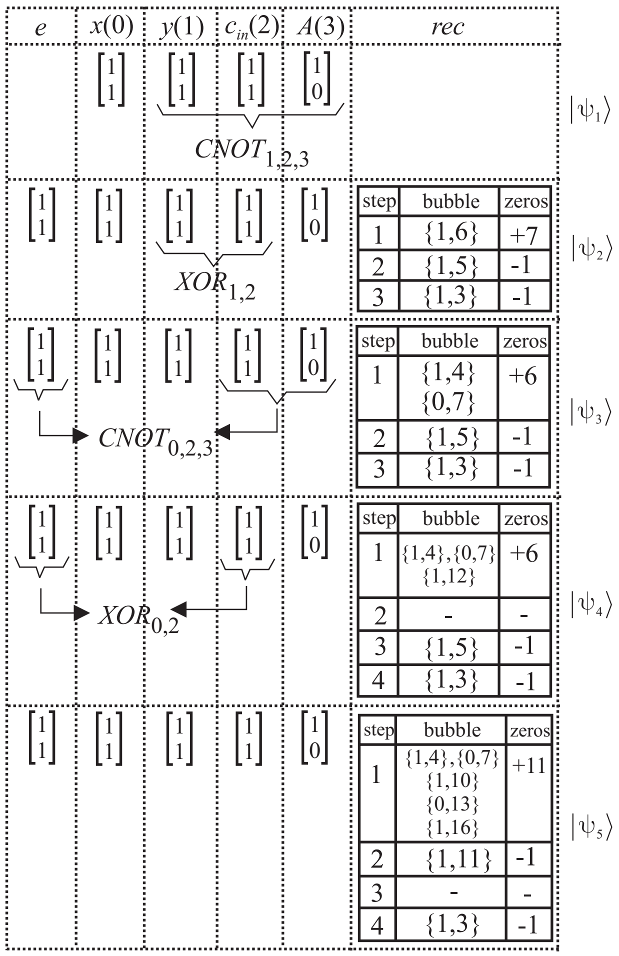
$$|\psi_2\rangle = \frac{1}{2\sqrt{2}}\begin{pmatrix} |0000\rangle + |0010\rangle + |0100\rangle + |0111\rangle \\ |1000\rangle + |1010\rangle + |1100\rangle + |1111\rangle \end{pmatrix} \overset{\text{bubble}}{\longrightarrow} \frac{1}{4}\hat{3}^{\otimes 4} \otimes \hat{1} + \boxed{rec_2} \tag{B.5}$$

$$|\psi_3\rangle = \frac{1}{2\sqrt{2}}\begin{pmatrix} |0000\rangle + |0010\rangle + |0110\rangle + |0101\rangle \\ |1000\rangle + |1010\rangle + |1110\rangle + |1101\rangle \end{pmatrix} \overset{\text{bubble}}{\longrightarrow} \frac{1}{4}\hat{3}^{\otimes 4} \otimes \hat{1} + \boxed{rec_3} \tag{B.6}$$

$$|\psi_4\rangle = \frac{1}{2\sqrt{2}}\begin{pmatrix} |0000\rangle + |0010\rangle + |0110\rangle + |0101\rangle \\ |1000\rangle + |1011\rangle + |1111\rangle + |1101\rangle \end{pmatrix} \overset{\text{bubble}}{\longrightarrow} \frac{1}{4}\hat{3}^{\otimes 4} \otimes \hat{1} + \boxed{rec_4} \tag{B.7}$$

$$|\psi_5\rangle = \frac{1}{2\sqrt{2}}\begin{pmatrix} |0000\rangle + |0010\rangle + |0110\rangle + |0101\rangle \\ |1010\rangle + |1001\rangle + |1101\rangle + |1111\rangle \end{pmatrix} \overset{\text{bubble}}{\longrightarrow} \frac{1}{4}\hat{3}^{\otimes 4} \otimes \hat{1} + \boxed{rec_5} \tag{B.8}$$

Figure B.14 presents the step-by-step results of the procedure applied on the 1-qubit full adder, while Figure B.15 contains the details regarding all the bubble steps performed for $|\psi_2\rangle$. Figure B.14 has 6 columns and 5 rows; the columns correspond to the following: 1 record (*rec*), 4 qubits for the circuit's inputs (x, y, c_{in}, A also labeled as 0, 1, 2, 3), and the extra qubit required by bubble bit insertions (e). All the involved successive states $|\psi_{1..5}\rangle$ have a distinct allocated row in this procedure illustration.



step	bubble	zeros
1	{1,6}	+7
2	{1,5}	-1
3	{1,3}	-1

step	bubble	zeros
1	{1,4} {0,7}	+6
2	{1,5}	-1
3	{1,3}	-1

step	bubble	zeros
1	{1,4},{0,7} {1,12}	+6
2	-	-
3	{1,5}	-1
4	{1,3}	-1

step	bubble	zeros
1	{1,4},{0,7} {1,10} {0,13} {1,16}	+11
2	{1,11}	-1
3	-	-
4	{1,3}	-1

Figure B.14: Bubble bit procedure results.

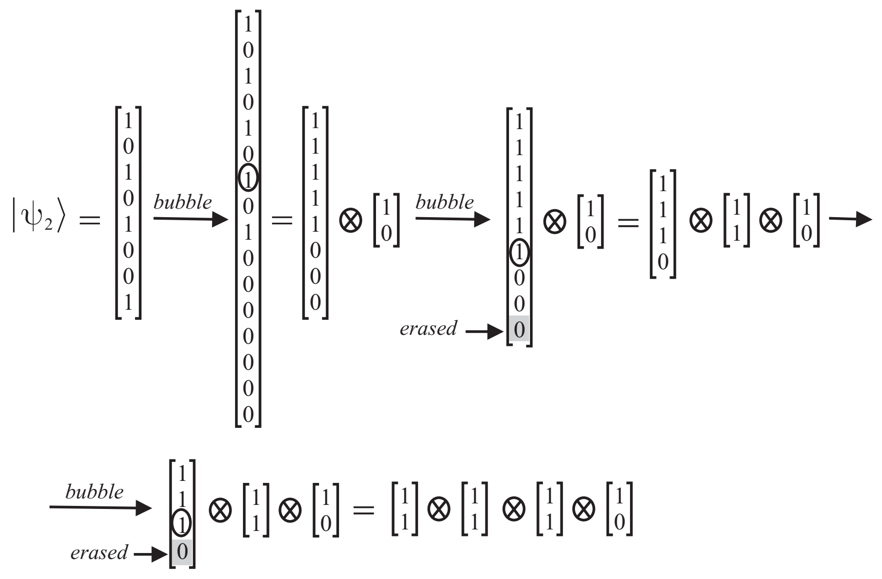


Figure B.15: Bubble bit procedure example.

size	*n*	*ent*	*gates*	*bub*	t_{WB}	t_B
1	4	total	4	4	<0.5 sec	<0.5 sec
4	13	total	16	12	13.5 sec	2 sec
8	25	total	32	24	4 hr, 12 sec	13 sec
16	49	total	64	48	timed out	41 sec
32	97	total	128	94	timed out	3 min, 48 sec
64	193	total	256	192	timed out	16 min, 7 sec

Table B.1: Quantum full adder simulation results.

The results from Figure B.14, as well as Equations from B.5 to B.8, indicate a structural network ('SArh' from Figure B.11) of only identity qubit gates (characterized by I matrix). The new equivalent network was obtained the way section B.1 and Figure B.5 explain. This is important, because the structural (i.e. polynomial) simulation is now possible, with the original quantum states that can be restored because of the information stored in the appropriate records.

The presented results are due to VHDL simulations, carried on a Windows XP$^{\text{TM}}$, PENTIUM$^{\text{TM}}$ IV CPU 1,6GHz, 192MB RAM machine. We have performed the gate-level simulation of quantum arithmetic [58] of the full adder (see Figure B.13). The experiment was pursued in the presence of total entanglement (therefore not a trivial simulation, as it is defined by [57]), thus requiring the bubble bit technique. The results are presented in table B.1, where *size* is the size of the adder in qubits, n is the size of the corresponding overall state (in qubits), *ent* is the type of the entanglement after the rightmost gate of the circuit, *gates* is the number of gates involved, *bub* is the maximum number of bubbles inserted for one record, and t_{WB}, t_B are the simulation times obtained without and with the bubble technique respectively.

Table B.2 presents simulation times for the modulo-k (we considered $k = 2^{size-1}$) quantum adders and multipliers, as essential circuits used for Shor's algorithm implementations [46][48][58].

size	Modulo adder		Modulo multiplier	
	t_{WB}	t_B	t_{WB}	t_B
4	33 min, 4 sec	3.5 sec	6 hr, 12 min	9 sec
8	8hr, 53 min	17 sec	timed out	44.5 sec
16	timed out	58.5 sec	timed out	2 min, 16 sec
32	timed out	5 min, 42 sec	timed out	16 min, 23 sec
64	timed out	21 min, 4 sec	timed out	53 min, 18 sec

Table B.2: Experimental results for modulo adder and multiplier (simulation time).

Because additional memory is required in order to store the records dictated by the bubble bit technique, Figure B.16 presents the polynomial memory overhead for the simple quantum ripple adder, modulo adder, and modulo multiplier.

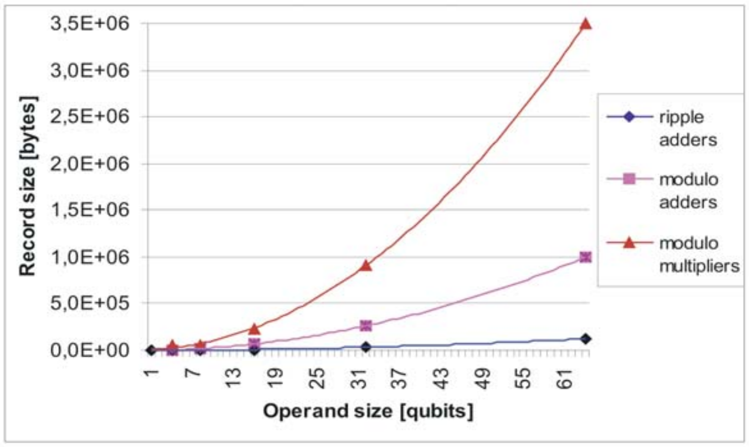


Figure B.16: Extra memory requirements.

B.4.2 Simulation of Grover's algorithm

When considering the states involved in Grover's algorithm, we will have a more general approach to avoiding entangled representations in the quantum states. The general form of states dictated by circuits from Grover's algorithm implementations is:

$$|\psi_{Grover}\rangle = \xi \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{2^n-1} \end{bmatrix} \tag{B.9}$$

where $a_i \in \{-1, 0, 1\}$ for $i = 0..2^n - 1$.

For explaining how our experiment works, we take as example the Grover algorithm circuit from [30] (see Figure B.17), which performs quantum search on a 2-qubit register.

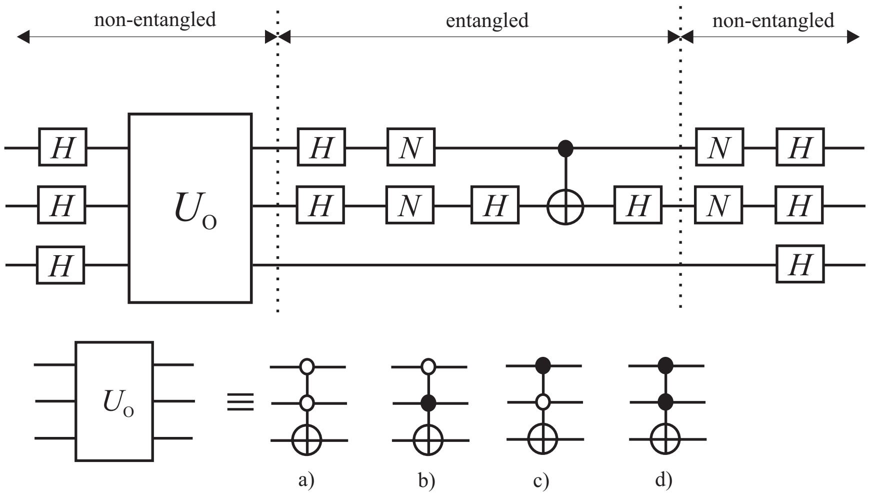


Figure B.17: Grover algorithm circuit for a 2-qubit search register. The oracle U_O can be any of the a)-d) gates; also an entanglement analysis is provided by showing where it appears and where it is absent.

The algorithm that transformes the state representation into a non-entangled one consists of repreating the bubble insertion algorithm until $(n + 1)$ [2×]-size matrices are extracted. The insertion algorithm is described by the following pseudocode:

Bubble insertion algorithm

```
1. scan all the couples $(a_k, a_{k+1})$ from Equation B.9;
    (a) memorize the first non-$\hat{0}$ couple;
    (b) insert bubbles according to rules in Figure B.18 and memorize their nature
        and position;
2. if the number of $a_i$ elements is a power of 2 $(= 2^m)$ then go to step 4;
3. if the previous adjustement consisted of a 0's padding then erase zeros so that
   the number of $a_i$ (matrix) elements will be the closest power of 2;
4. extract the first detected non-$\hat{0}$ couple as a non-entangled qubit representation;
```

The rules for bubble insertion are presented in Figure B.18, where 'x' stands for either '-1' or '1'. Of course, -x = 1 when x=-1.

In order to keep track of the operations involved by the bubble bit technique, we will watch the highlighted states $(|\psi_1\rangle \dots |\psi_5\rangle)$ from Figure B.19. In this figure, the lower qubit value is known throughout the computation (it is shown in Figure B.19) and it is not entangled with the rest. The search register is made out of qubits A and B, while qubit e is the extra qubit which is used only because it is required by the bubble bit non-entangled representation. Initially, $e = |0\rangle$.

The result of applying the bubble bit technique on the $|\psi_1\rangle \dots |\psi_1\rangle$ states is presented in Figure B.20. In fact, as forecasted in the entanglement analysis from Figure B.17, the bubble bit technique is only necessary for states $|\psi_2\rangle$ and $|\psi_3\rangle$.

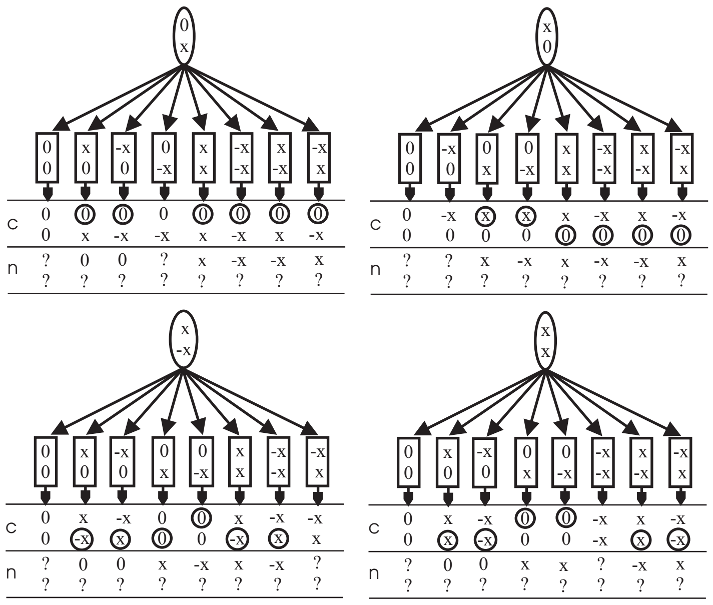


Figure B.18: Bubble bit insertion rules for Grover algorithm states.

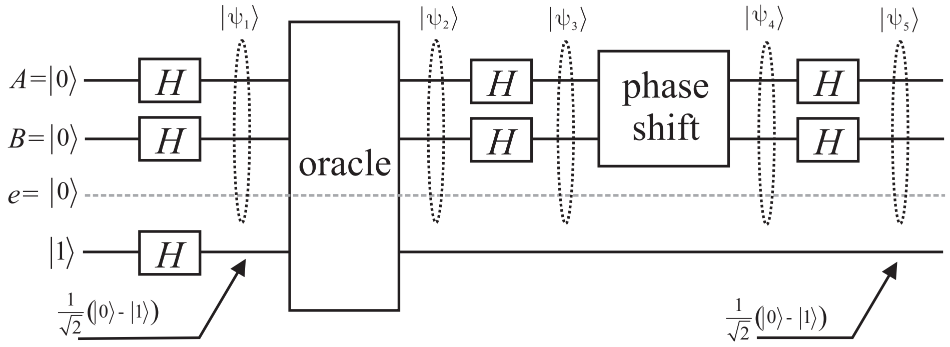


Figure B.19: Relevant states for Grover algorithm simulation.

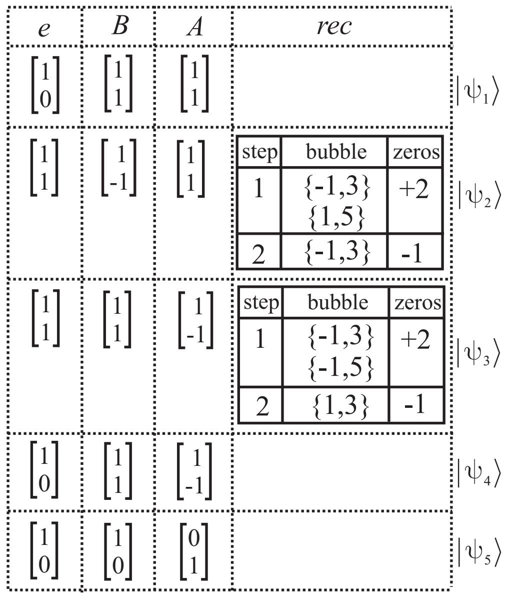


Figure B.20: Bubble bit insertion results for 2-qubit Grover search simulation.

These results can also be expressed as equations, where $\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \hat{4}$ and $\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \hat{5}$:

$$|\psi_1\rangle = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) = \hat{0} \otimes \hat{3}^{\otimes 2} \tag{B.10}$$

$$|\psi_2\rangle = \frac{1}{2}\left(|00\rangle + |01\rangle - |10\rangle + |11\rangle\right) = \hat{3} \otimes \hat{5} \otimes \hat{3} + \boxed{rec_1} \tag{B.11}$$

$$|\psi_3\rangle = \frac{1}{2}\left(|00\rangle - |01\rangle + |10\rangle + |11\rangle\right) = \hat{3}^{\otimes 2} \otimes \hat{5} + \boxed{rec_2} \tag{B.12}$$

$$|\psi_4\rangle = \frac{1}{2}\left(|00\rangle + |01\rangle - |10\rangle - |11\rangle\right) = \hat{1} \otimes \hat{5} \otimes \hat{3} \tag{B.13}$$

$$|\psi_5\rangle = |10\rangle = \hat{1} \otimes \hat{2} \otimes \hat{1} \tag{B.14}$$

The way that the bubble insertion procedure works is presented in Figure B.21.

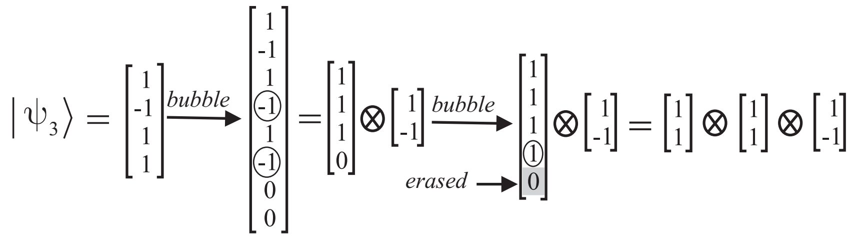


Figure B.21: Bubble bit insertion procedure for $|\psi_3\rangle$.

The result is the possibility of performing HDL structural simulation of the circuit, and therefore obtaining polynomial simulation times. The equivalent gate network, that can be simulated structurally, is presented in Figure B.22.

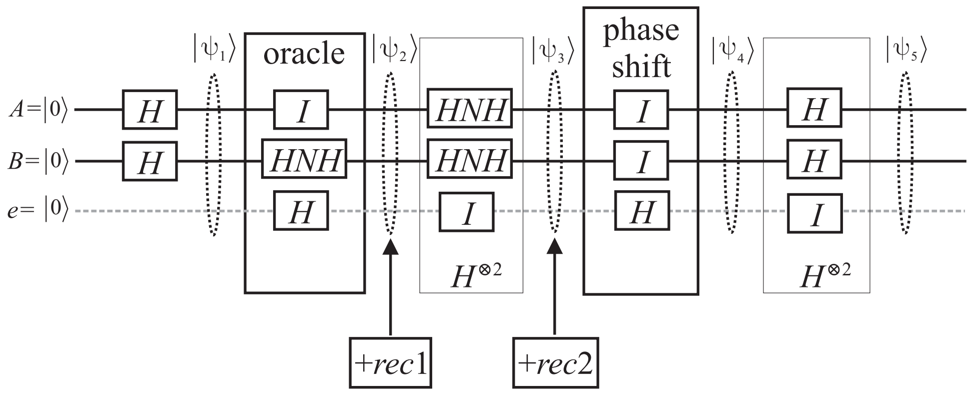


Figure B.22: 2-qubit search Grover equivalent circuit, obtained with the bubble-bit technique in order to allow structural (i.e. polynomial) simulation.

In Figure B.22 we use a HNH gate. It is a gate that performs negation in a changed basis space. Its equivalent network is $H \cdot N \cdot H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

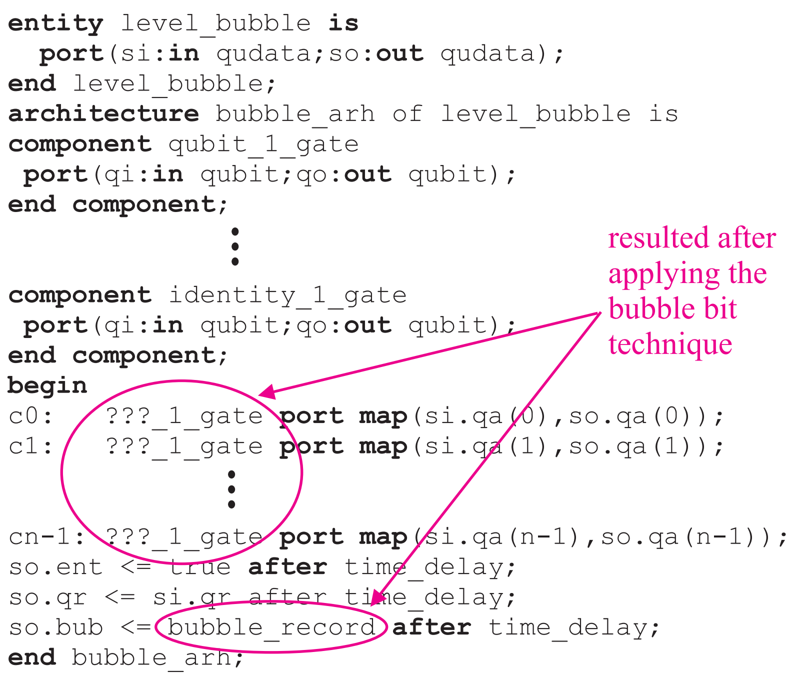
In order to represent the bubble records, we will add the data structure from Figure B.23 to the VHDL package from Figure B.3. In Figure B.24 we present the methodology that was used when building corresponding VHDL entity-architecture pairs, for the 1-qubit gate levels that were dictated by the bubble quantum state representation (as, for instance, "phase shift" in Figure B.22). These entity-architecture pairs have a fixed form, with only the marked components and signals being dictated by bubble bit technique's outcome (see Figure B.24).

```
-- the type describing bubble structure
type bubb is record
 nature:integer;
 position:integer;
end record;
-- the bubble type
type bubble_type is array(natural range<>) of bubb;
-- structure of bubble records
type rec_rec is record
 bubble:bubble_type(0 to 1);
 zeros:integer;
end record;
-- data type for bubble records
type bubble_record is array(natural range<>) of rec_rec;
```

Figure B.23: Data types required by bubble record representation.

Grover's algorithm was simulated for an Oracle that marks just one basis state, like [60]. Figures B.25 and B.26 present the time diagrams resulted from simulation of Grover's algorithm with a 2-qubit data register and $|10\rangle$ the "marked" basis state. In these figures the relevant datapath is highlighted by arrows, which point the fields that are actually used by the corresponding structural or behavioral architectures. Of course, for the bubble bit simulation only structural architectures are required.

The runtime evolution with the number of qubits in the data register is presented in Figure B.27. Also, the measured simulation times are compared here with the runtime complexity reported in [60],



```
entity level_bubble is
  port(si:in qudata;so:out qudata);
end level_bubble;
architecture bubble_arh of level_bubble is
component qubit_1_gate
 port(qi:in qubit;qo:out qubit);
end component;
                 ⋮
component identity_1_gate
 port(qi:in qubit;qo:out qubit);
end component;
begin
c0:    ???_1_gate port map(si.qa(0),so.qa(0));
c1:    ???_1_gate port map(si.qa(1),so.qa(1));
                 ⋮
cn-1: ???_1_gate port map(si.qa(n-1),so.qa(n-1));
so.ent <= true after time_delay;
so.qr <= si.qr after time_delay;
so.bub <= bubble_record after time_delay;
end bubble_arh;
```

Figure B.24: VHDL gate level implementation (entity-architecture pair) for bubble bit state transformation.

which is 0.22×1.44^n. The graphical representation shows substantial runtime improvement. Also, Figure B.28 presents the memory overhead of bubble bit simulation, dictated by the bubble records. The added trendline indicates that the supplementary memory overhead grows polynomially with the number of qubits in the data register.

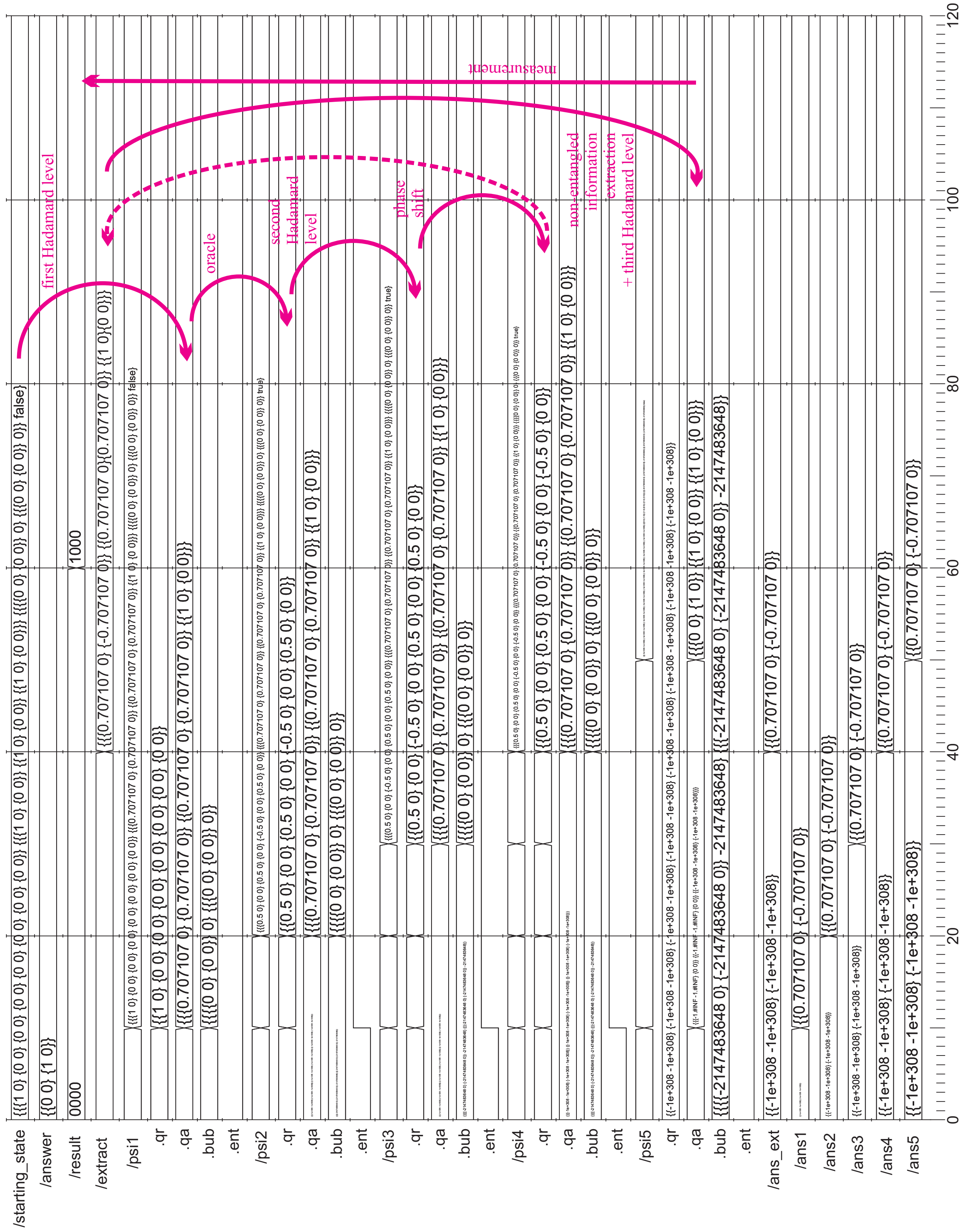


Figure B.25: Time diagram resulted from VHDL simulation of Grover's algorithm, without the bubble bit technique.

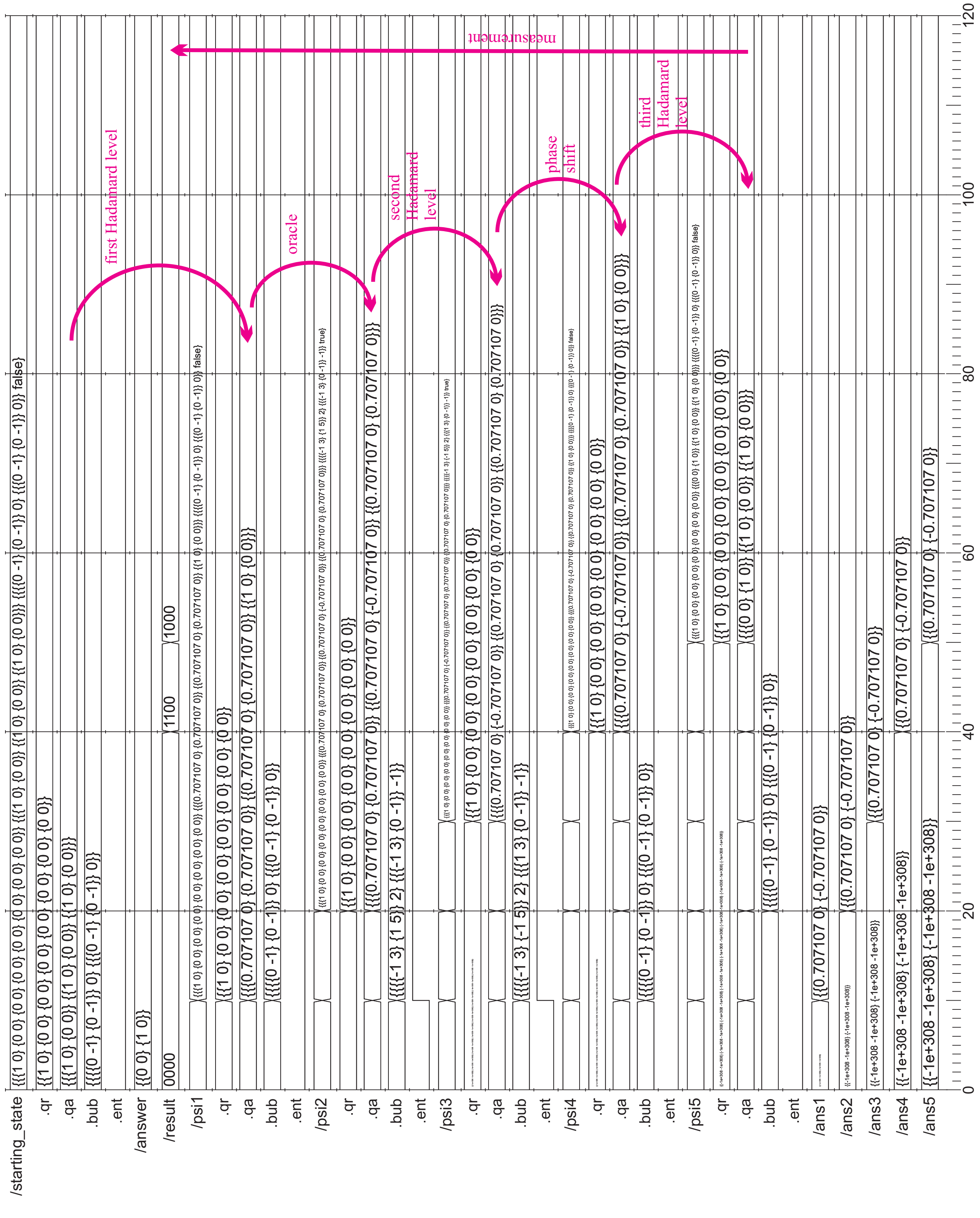


Figure B.26: Time diagram resulted from VHDL simulation of Grover's algorithm, with the bubble bit technique.

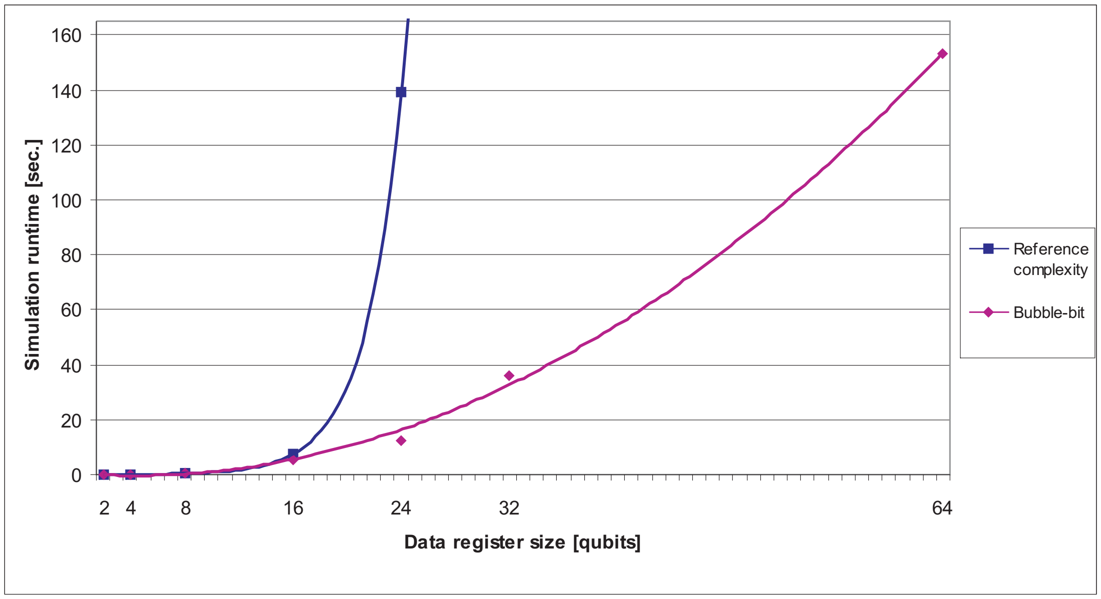


Figure B.27: HDL bubble bit runtime results for Grover algorithm simulation, compared with the reference complexity.

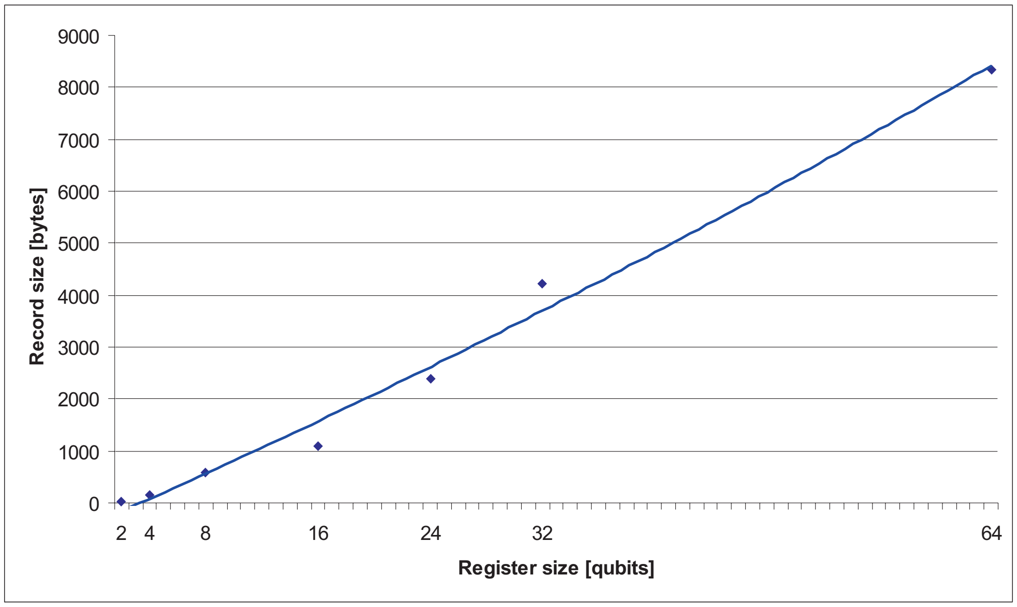


Figure B.28: Memory overhead dictated by bubble records for Grover algorithm simulation. A trendline is added to the sample data, showing polynomial growth.

Bibliography

[1] D. Aharonov, M. Ben-Or, "Fault Tolerant Quantum Computation with Constant Error", In Proc. 29th Ann. ACM Symposium on Theory of Computing, pp. 176-188, El Paso, Texas, (May 1997).

[2] A. Ahuja, S. Kapoor, "A Quantum Algorithm for Finding the Maximum", ArXiv:quant-ph/9911082 (November 1999).

[3] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications", IEEE Trans. on Soft. Eng. Vol.16, Iss.2 pp.166 - 182 (1990).

[4] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems", LAAS-CNRS Research Report 91260 (1992).

[5] P. J. Ashenden, "The Designers guide to VHDL (second edition)", Morgan Kaufmann Publishers (2001).

[6] A. Avižienis, J-C. Laprie, B. Randell, C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Transactions on Dependable and Secure Computing, Volume 1, Number 1, pp. 11-33, (January-March 2004).

[7] A. Barenco, C.H. Bennett, R. Cleve, D.P. Vincenzo, N. Margolus, P. Shor, T.Sleator, J. Smolin and H. Weinfurter, "Elementary gates for quantum computation", Phys. Rev. A 52, 3457-3467 (1995), quant-ph/9503016.

[8] E. Bernstein, U. Vazirani, "Quantum Complexity Theory", SIAM J. Computing 26, 1411-73 (1997).

[9] B.H. Bransden, C.J. Joachain, "Introduction to Quantum Mechanics", Addison-Wesley Longman Ltd. (1989).

[10] R. Cleve, D. Gottesman, "Efficient computations of encoding for quantum error correction," Phys. Rev. A 56, 76-82 (1997).

[11] T.A. Delong, B.W.Johnson, J.A. Profeta III, "A Fault Injection Technique for VHDL Behavioral-Level Models", IEEE Design and Test of Computers Volume 13, Issue 4, pp. 24-33, (1996).

[12] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer", Proc. R. Soc. Lond. A 400, 97 (1985).

[13] D. Deutsch, "Quantum computational networks", Proceedings Royal Society London A 425, 73 (1989).

[14] C. Durr, P. Hoyer, "A Quantum Algorithm for Finding the Minimum". ArXiv:quant-ph/9607014, (July 1996).

[15] A. Ekert, R. Jozsa, "Quantum Algorithms: Entanglement Enhanced Information Processing," Phil. Trans. Roy. Soc. Lond. A, pp. 1779-1782, (1998).

[16] R.P. Feynman, "Quantum mechanical computers", Optics News, 11, p.11 (1985).

[17] G.A. Giraldi, R. Portugal, R.N. Thess, "Genetic Algorithms and Quantum Computation", ArXiv:cs.NE/0403003 (March 2004).

[18] D. Gottesman, "Class of quantum error-correcting codes saturating the quantum Hamming bound," Phys. Rev. A 54, 1862-1868 (1996).

[19] D. Gottesman, "A theory of fault-tolerant quantum computation," Phys. Rev. A 57, 127-137 (1998).

[20] L. Grover, "Quantum mechanics helps in searching for a needle in a haystack", Phys. Rev. Lett. (79), pp. 325-328, (1997).

[21] K-H. Han, J-H. Kim, "Genetic Quantum Algorithm and its Application to Combinatorial Optimization Problem", In Proc. of the 2000 Congress on Evolutionary Computation, citeseer.nj.nec.com/han00genetic.html, (2000).

[22] K-H. Han, J-H. Kim, "Quantum-Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization", IEEE Transactions on Evolutionary Computation, vol. 6, No. 6, pp.580-593 (2002).

[23] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson "Fault injection into VHDL models: the MEFISTO tool", 24th Annual International Symposium on Fault-Tolerant Computing (FTCS-24), Austin (USA), pp.66-75 (1994).

[24] E. Knill, "Fault-Tolerant Postselected Quantum Computation: Schemes.", quant-ph/0402171, (2004).

[25] E. Knill, "Fault-Tolerant Postselected Quantum Computation: Threshold Analysis.", quant-ph/0404104, (2004).

[26] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, H-Y. Chung, K. Chung, H. Jeech, K. Byung-Guk, K. Yong-Duk, "Evolutionary Approach to Quantum and Reversible Circuits Synthesis", Artificial Intelligence Review, Vol. 20 , Issue 3-4, pp. 361-417 (2003).

[27] D. Mange, M. Sipper,A. Stauffer, G. Tempesti, "Toward Robust Integrated Circuits: The Embryonics Approach," Proc. IEEE, 88, 4, pp.516-541, (2000).

[28] A. Narayanan, M. Moore, "Quantum-Inspired Genetic Algorithms". IEEE International Conference on Evolutionary Computation (ICEC-96), Nagoya, Japan pp. 61-66 (May 1996).

[29] M.A. Nielsen, I.L. Chuang, "Programmable Quantum Gate Arrays", Phys. Rev. Lett. 79, Issue 2-14 321324 (1997).

[30] M.A. Nielsen, I.L. Chuang "Quantum Computation and Quantum Information",Cambridge University Press (2000).

[31] K.M. Obenland, "Using simulation to assess the feasibility of quantum computing", Ph.D. Thesis, University of Southern California, Electrical Engineering - Systems (1998).

[32] K.M. Obenland, A. Despain, "Simulating the Effect of Decoherence and Inaccuracies on a Quantum Computer", Proc. 1st NASA Conference on Quantum Computation and Quantum Communication (1998).

[33] B. Omer, "Quantum programming in QCL", Technical Report, Institute of Information Systems, Technical University of Vienna (2000).

[34] C. Ortega, A. Tyrrell, "Reliability Analysis in Self-Repairing Embryonic Systems," Proc. 1st NASA/DoD Workshop Evolvable Hardware, pp. 120-128 (1999).

[35] M. Oskin, F. Chong, I. Chuang, "A Practical Architecture for Reliable Quantum Computers". IEEE Computer, 35(1): 79-87 (2002).

[36] B. Parhami, "Computer Arithmetic. Algorithms and Hardware Designs", Oxford University Press (2000).

[37] S. Parker, M. Plenio, "Entanglement Simulations of Shor's Algorithm," J. Mod. Opt., Vol. 49, Nr. 8 pp. 1325-1353, (2002).

[38] J. Preskill, "Reliable Quantum Computers", Proc. Roy. Soc. London A for the Proc. of Santa Barbara Conference on Quantum Coherence and Decoherence (1996).

[39] J. Preskill, "Fault-Tolerant Quantum Computation", Online preprint quant-ph/9712048 (1997).

[40] K. N. Patel, J. P. Hayes, I. L. Markov, "Fault Testing for Reversible Circuits," IEEE Trans. on CAD, 23(8), pp. 1220-1230, (August 2004), quant-ph/0404003

[41] L. Prodan, M. Udrescu, M. Vlăduţiu, "Self-Repairing Embryonic Memory Arrays", IEEE NASA/DoD Conference on Evolvable Hardware, Seattle WA, USA, June 24 - 26, pp. 130-137, (2004).

[42] M. Rimen, J. Ohlsson, J. Karlsson, E. Jenn, J. Arlat, "Validation of fault tolerance by fault injection in VHDL simulation models", Rapport LAAS No.92469, (December 1992).

[43] M. Rimen, J. Ohlsson, J. Karlsson, E. Jenn, J. Arlat, "Design guidelines of a VHDL-based simulation tool for the validation of fault tolerance", Rapport LAAS No93170 Contrat ESPRIT III Esprit Basic Research Action No.6362, (May 1993).

[44] B. Rylander, T. Soule, J. Foster, "Computational complexity, genetic programming, and implications", Proc. 4th EuroGP, pp. 348-360 (2001).

[45] B. Rylander, T. Soule, J. Foster, J. Alves-Foss, "Quantum Evolutionary Programming", In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pp. 1005-1011 Morgan Kaufmann (2001).

[46] P.W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," Proc. 35th Symp. on Foundations of Computer Science, pp.124-134, (1994).

[47] P.W. Shor, "Fault-tolerant quantum computation", Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (1996).

[48] P. W. Shor "Quantum Computing", Documenta Mathematica, Extra Volume ICM 1998, 1-1000 (1998).

[49] L. Spector, H. Barnum, K.J. Bernstein, N. Swamy, "Quantum Computing Applications of Genetic Programming", In Advances in Genetic Programming, volume 3. (1999)

[50] L. Spector, H. Barnum, K.J. Bernstein, N. Swamy, "Finding a Better-than-Classical Quantum AND/OR Algorithm Using Genetic Programming", In Proceedings of 1999 Congress of Evolutionary Computation, Piscataway, NJ, IEEE Press vol 3 pp.2239-2246 (1999)

[51] L. Spector, "Automatic Quantum Computer Programming: A Genetic Programming Approach", Kluwer Academic Publishers, (2004).

[52] A.M. Steane, "Error Correcting Codes in Quantum Theory", Phys. Rev. Lett. 77, 793 (1996).

[53] A.M. Steane, "Multiple Particle Interference and Quantum Error Correction", Proc. Roy. Soc. Lond. A 452, 2551 (1996).

[54] M. Udrescu, L. Prodan, M. Vlăduţiu, "A new perspective in simulating quantum circuits", Proc. LBP AAAI GECCO pp.283-290, Chicago IL (2003).

[55] M. Udrescu, L. Prodan, M. Vlăduţiu, "Using HDLs for Describing Quantum Circuits: A Framework for Efficient Quantum Algorithm Simulation", Proc. 1st ACM Conf. On Computing Frontiers pp.96-110 (Ischia, April 2004).

[56] M. Udrescu, L. Prodan, M. Vlăduţiu, "The Bubble Bit Technique as Improvement of HDL-Based Quantum Circuits Simulation." IEEE 38th Annual Simulation Symposium, San Diego CA, USA, April 2 - 8, 2005. Accepted.

[57] L.G. Valiant, "Quantum Circuits That Can Be Simulated Classically in Polynomial Time," SIAM J.Comp. 31:4, pp. 1229-1254, (2002).

[58] V. Vedral, A. Barenco, A. Ekert, "Quantum Networks for Elementary Arithmetic Operations", Online preprint quant-ph/9511018, (1996).

[59] G. Viamontes, M. Rajagopalan, I. Markov, J.P. Hayes, "Gate-Level Simulation of Quantum Circuits", Proc.of the Asia South Pacific Design Automation Conference, pp.295-301, (2003).

[60] G. Viamontes, I. Markov, J.P. Hayes, "High-performance QuIDD-based Simulation of Quantum Circuits," Proc. Design Autom. and Test in Europe (DATE), Paris, France, pp. 1354-1359, (February 2004).

[61] Y. Yangyang, Barry W. Johnson, "A Perspective on the State of Research on Fault Injection Techniques", Technical Report UVA-CSCS-FIT-001 University of Virginia (May 2002).

[62] A.C.-C. Yao, "Quantum Circuit Complexity", Proc. 34th Annual Symposium on the Foundations of Computer Science (IEEE Computer Society Press, Los Alamitos CA) p.352 (1993).

[63] C. Zalka, "Threshold Estimate for Fault Tolerant Quantum Computation", quant-ph/9612028 (1996).