UNIVERSITATEA "POLITEHNICA" DIN TIMIŞOARA

Quantum Circuits Engineering: Efficient Simulation and Reconfigurable Quantum Hardware

Ph.D. Thesis

Mihai UDRESCU-MILOSAV
Computer Engineering Department
Politehnica University of Timişoara
Timişoara, Romania

Advisor: Prof. Mircea Vlăduţiu, UPT

November 25, 2005

Abstract

In the quantum computational framework, there are polynomial time solving algorithms for problems having exponential classical solutions. The quest is - on one hand - to search if there are other possible effective quantum algorithms and - on the other hand - to be able to produce efficient implementations for the already known algorithms. The most feasible implementation of quantum algorithms is based on the quantum circuit (gate network) model.

Our work aims at bridging the gap between classical hardware CAD with design automation techniques, and quantum circuit design rules. This attempt would be extremely difficult without the possibility of the efficient quantum circuit simulation. Thus, our first direction was to try using Hardware Description Languages (HDLs) for simulating quantum circuits, because their property of being able to describe - in a compact manner - the circuit with both structural and behavioral (functional) architectures isolates the inner source of simulation complexity: the entanglement. Our analysis showed that the probability of simulation improvement just by using the HDL procedure is small. Therefore, we developed a special algorithm for avoiding entangled state representations, the *bubble bit technique*, which is effective at least when dealing with specific algorithm states. Our simulation framework has the ability of fault injection, in order to create incentive for validation of quantum circuit fault tolerance strategies and algorithms. The other direction of this Ph.D. work is to find common ground for reliability techniques and assessment methodologies from the Embryonics project and fault tolerant quantum computation. Embryonics is a biologically inspired reconfigurable hardware project, which is suitable for attaining reliability in aggressive, critical environments, similar to quantum computation in terms of fault model and fault occurrence frequency. Adopting the accuracy threshold as reliability measure in Embryonic memories is benefic. Also, when considering a reconfigurable strategy (reconfigurable quantum gate arrays - rQGAs) in quantum computation fault tolerant stabilizer encoding, the appropriate reliability measure is drastically improved.

When entanglement is not present, it is possible to describe the circuit and the processed quantum 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. The practical implementation of the initial simulation methodology requires that each circuit be described both by structural and functional (behavioral) 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. We adopted the matrix representation of quantum states and unitary operators; therefore the quantum states are type array of complex signals. Efficient automated extraction of non-entangled qubit group states is not conceivable unless we have some *a priori* information about the overall state: the so-called *simulation shortcuts*. 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. We have performed an analysis concerning the effectiveness of our methodology, for specific states from Shor and Grover algorithms. Unfortunately, as shown by our case study for Shor, Deutsch-Jozsa and Grover algorithms, the probability of success for the extraction algorithms is decreasing exponentially with the number of qubits in the processed state. Nevertheless, the HDL-based simulation methodology can be further improved. The bubble bit coding technique creates a new entanglement-free-represented state. Therefore, the simulation works with equivalent gate networks operating on corresponding bubble-coded non-entangled states, and after applying the unitary operator the original state can be restored. This way, the unitary transform is obtained with at most n $[2 \times 1]$-size matrixes, with the expense of memorizing $\mathcal{O}(n^2)$ size records. The bubble bit procedure can also be used for simulated fault injection, according to the fault models. We present here experimental and assessment results describing the most important contributions of our Ph.D. work in the simulation part. The simulation runtimes show an important runtime improvement at the expense of a polynomial memory overhead, as compared with our reference simulator (QuIDD Pro, developed by the Quantum Circuits Group at University of Michigan).

The need for fault tolerance is vital in quantum computation, due to the omnipresent nature of quantum decoherence errors. A specific reliability parameter was defined, under the form of the *accuracy threshold*. If the quantum circuit's fault tolerance dictates accuracy greater or equal with the threshold, then it could be used for arbitrary long reliable quantum computation. The quantum circuit fault tolerance techniques - even the most recent ones - use the concatenated coding for both protected data and ancilla qubits. Our reconfigurable quantum hardware strategy employs a quantum nature (i.e. superposition of classical basis states) configuration register in order to have a superposition of error detection and correction circuits at the same time. The starting idea is that if the gate error probability is ξ, and we have k superposed correction circuits then, after the measurement of the configuration register, the overall circuit error probability becomes ξ^k (negligible for a small ξ). We developed a reconfigurable quantum circuit, the so-called reconfigurable Quantum Gate Array (rQGA), which we assessed with the accuracy threshold measure. Our analytical estimate of the accuracy threshold shows that the rQGA solution clearly dominates the actual technologic accuracy limit, thus allowing for arbitrary long fault tolerant quantum computation. This way, the rQGA technique can replace the concatenated coding, a solution that is vulnerable in the presence of correlated faults.

The last part of the thesis is dedicated to the implementation of the Quantum Genetic Algorithms (QGA). Our solution is based on an already known quantum algorithm (the maximum finding algorithm) and on a specially designed oracle, which reduces the entire QGA problem to Grover's search algorithm. The conclusion is that the genetic strategy is not applicable to the quantum computation environment, with the crossover and mutation genetic operators becoming useless. The complexity of the proposed Reduced Quantum Genetic Algorithm is linear, thus proving the superiority of the quantum computing in yet another computation field.

Rezumat

În cadrul computaţiei cuantice există algoritmi care rezolvă în timpi polinomiali probleme care au soluţii clasice exponenţiale. Obiectivul este – pe de o parte – de a proiecta alţi algoritmi cuantici eficienţi şi – pe de altă parte – de a putea produce implementări eficiente pentru algoritmii deja cunoscuţi. Cea mai fezabilă implementare a algoritmilor cuantici este bazată pe modelul circuit (sau reţea de porţi). Lucrarea noastră este concepută pentru a face legătura între proiectarea asistată de calculator (CAD) din hardware-ul clasic, bazată pe proiectarea automatizată, şi regulile de proiectare ale circuitelor cuantice. Această tentativă ar fi extrem de dificilă în absenţa posibilităţii de a simula circuitele cuantice în mod eficient. Astfel, prima direcţie a acestei teze de doctorat constă în încercarea de a folosi limbajele de descriere hardware (HDL) pentru simularea circuitelor cuantice, datorită proprietăţii acestora de a putea descrie – într-o manieră compactă – circuitul cu arhitecturi structurale şi comportamentale (funcţionale) şi care face ca izolarea entanglement-ului ca sursă principală a complexităţii de simulare să fie posibilă. Analiza pe care am efectuat-o arată faptul că probabilitatea de a reduce timpii de simulare doar prin folosirea procedurii de simulare HDL este mică. Prin urmare, am dezvoltat un algoritm special, pentru evitarea reprezentărilor afectate de entanglement ale stărilor cuantice, aşa-numita tehnică *bubble bit*, care este eficientă cel puţin atunci când sunt procesate stări specifice anumitor algoritmi. Metodologia noastră de simulare este înzestrată şi cu abilitatea de a injecta defecte, în ideea de a face posibilă procedura de validare a strategiilor şi algoritmilor de toleranţă la defectare. Cealaltă direcţie a activităţii doctorale reflectate în această teză constă în încercarea de a găsi teren comun pentru tehnicile de fiabilizare şi metodologiile de evaluare aferente proiectului Embryonics pe de o parte, şi calculul cuantic tolerant la defecte pe de altă parte. Embryonics este un proiect hardware inspirat din biologie, care este pretabil obţinerii fiabilităţii în medii critice, agresive, similare calculului cuantic în termini de model al defectului şi al frecvenţei de apariţie. Adoptarea *pragului de acurateţe* ca măsură a fiabilităţii în memoriile Embryonics este benefică. Deasemenea, atunci când luăm în consideraţie o strategie reconfigurabilă (matrici reconfigurabile de porţi cuantice – rQGAs) pentru computaţia cuantică tolerantă la defectare şi coduri stabilizatoare, se imbunătăţeşte drastic gradul de fiabilitate.

Atunci când fenomenul de entanglement nu este present, este posibil să descriem circuitul cuantic şi stările procesate în manieră structurală, revendicând doar resurse polinomiale pentru simulare. În schimb, atunci când entanglement-ul este detectat în starea procesată, circuitul trebuie să fie descris printr-o arhitectură comportamentală, şi simularea va dicta acum utilizarea unor resurse exponenţiale. Acest lucru se întâmplă deoarece, atunci când apare entanglement-ul pentru două subsisteme cuantice, starea lor generală nu poate fi reprezentată correct ca simpla reuniune a stărilor cuantice aferente celor două subsisteme individuale (am presupus că produsul tensorial este unealta implicită de compunere a stărilor individuale). Implementarea practică a metodologiei iniţiale de simulare necesită ca fiecare circuit să fie descris prin ambele arhitecturi: structurală şi comportamentală (funcţională). Dacă pentru o reţea de porţi cuantice entanglement-ul este detectat în starea precedentă sau următoare, atunci arhitectura funcţională este selectată pentru a o descrie; altminteri se selectează arhitectura structurală. În cadrul acestei abordări, am ales reprezentarea matriceală a stărilor cuantice şi a operatorilor unitari; prin urmare, stările cuantice sunt reprezentate ca tip vector de numere complexe. Extragerea automată eficientă, a stărilor cuantice representând

grupuri de qubiţi care nu se află în entanglement, nu este de conceput fără a avea la dispoziţie informaţie *apriorică* referitoare la starea cuantică generală: aşa-numitele *scurtături de simulare*. Atunci când avem de a face cu stări din anumite puncte ale circuitelor ce implementează algoritmi specifici, vom avea la dispoziţie acea informaţie apriorică, datorită aspectului caracteristic pe care îl au aceste stări. Am efectuat o analiză privitoare la eficienţa metodologiei de simulare HDL, pentru stări specifice din algoritmii Shor şi Grover. Din nefericire, aşa cum se arată în studiul nostru de caz pentru algoritmii Shor, Deutsch-Jozsa şi Grover, probabilitatea ca algoritmii de extracţie să fie încununaţi de succes descreşte exponenţial cu numărul de qubiţi ai stării procesate. Fără îndoială, metodologia de simulare bazată pe limbajele de descriere hardware (HDLs) poate fi îmbunătăţită în continuare. Tehnica de codificare bubble bit crează o nouă reprezentare lipsită de entanglement. Prin urmare, simularea se face cu reţele de porţi cuantice echivalente ce operează pe stări codificate prin metoda bubble bit; după aplicarea transformării unitare starea originală poate fi restaurată. În acest fel, transformarea unitară este obţinută cu cel mult n matrici de dimensiune $[2 \times 1]$, cu preţul memorării unor înregistrări de dimensiune $\mathcal{O}(n^2)$. Procedura bubble bit poate deasemenea să fie folosită pentru injecţia simulată de defecte, în concordanţă cu modelul de defectare. În cadrul acestei lucrări se prezintă rezultate experimentale şi analitice ce descriu cele mai importante contribuţii în domeniul simulării HDL-bubble-bit. Timpii de simulare arată o îmbunătăţire semnificativă cu preţul unui consum suplimentar polinomial de memorie, prin comparaţie cu simulatorul referinţă (QuIDD Pro, dezvoltat de către Quantum Circuits Group de la University of Michigan).

Necesitatea toleranţei la defectare este vitală în calculul cuantic, datorită naturii omniprezente a erorilor de decoerenţă. În plus, a fost definit un parametru specific fiabilităţii, sub forma *pragului de acurateţe*. Dacă toleranţa la defectare a circuitului cuantic dictează o acurateţe mai mare sau egală cu pragul, atunci poate fi folosit pentru un calcul cuantic fiabil pe o perioadă arbitrar de lungă. Tehnicile de toleranţă la defectare pentru circuitele cuantice (chiar şi cele mai recente) folosesc codificarea concatenată atât pentru datele codificate cât şi pentru qubiţii auxiliari (ancilla). Strategia noastră pentru hardware-ul cuantic fac uz de un registru de configurare de natură cuantică (superpoziţie de stări clasice ale bazei), pentru a avea o superpoziţie simultană de circuite corectoare de erori. Ideea de plecare este că dacă probabilitatea de defectare a porţilor este ξ, şi avem k circuite corectoare de erori superpuse, atunci, după măsurarea registrului de configurare, probabilitatea de apariţie a erorii în circuitul ca întreg devine ξ^k (neglijabilă pentru un ξ sufficient de mic). Am proiectat un circuit reconfigurabil cuantic (rQGA), pe care l-am evaluat cu ajutorul pragului de acurateţe. Estimarea noastră analitică pentru pragul de acurateţe demonstrează că soluţia rQGA este mult deasupra limitei tehnologice de acurateţe, permiţând calculul cuantic tolerant la defectare arbitrar de lung. Astfel, tehnica rQGA poate înlocui codificarea concatenată, o soluţie vulnerabilă la acţiunea defectelor corelate.

Ultima parte a tezei este dedicată implementării Algoritmilor Genetici Cuantici (QGA). Soluţia propusă este bazată pe un algoritm cuantic deja cunoscut (algoritmul găsirii maximului) şi pe un oracol proiectat în mod special, care reduce întreaga problematică rQGA la algoritmul de căutare al lui Grover. Concluzia este că strategia genetică nu poate fi aplicată în mediul computaţional cuantic, deoarece operatorii genetici de crossover şi mutaţie sunt inutili. Complexitatea algoritmului propus (Reduced Quantum Genetic Algorithm) este liniară, probând astfel superioritatea calculului cuantic într-un nou domeniu computaţional.

Published papers and impact

This thesis is supported by the following published papers as first author:

- M. Udrescu, L. Prodan, M. Vlăduţiu, "A new perspective in simulating quantum circuits", Proc. LBP AAAI GECCO pp.283-290, Chicago IL (2003).
- 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).
- 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, IEEE Press, pp. 217-224 (April 2 - 8, 2005).
- M. Udrescu, L. Prodan, M. Vlăduţiu, "Simulated Fault Injection in Quantum Circuits with the Bubble Bit Technique." 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA), Coimbra, Portugal, Springer WienNewYork pp. 276-279 (March 21 - 23, 2005).
- M. Udrescu, L. Prodan, M. Vlăduţiu, "Improving Quantum Circuit Dependability with Reconfigurable Quantum Gate Arrays", 2nd ACM Conference On Computing Frontiers, ACM Press, pp. 133-144 (Ischia, Italy, 2005).

Other papers with relevance to particular aspects of this thesis (published as second author):

- 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).
- L. Prodan, M. Udrescu, M. Vlăduţiu, "Reliability Assessment in Embryonics Inspired by Fault-Tolerant Quantum Computation", Proc. 2nd ACM Conference On Computing Frontiers, ACM Press, pp. 323-333 (Ischia, Italy, 2005).
- L. Prodan, M. Udrescu, M. Vlăduţiu, "Multiple-Level Concatenated Coding in Embryonics: A Dependability Analysis", GECCO (ACM-SIGEVO), pp. 941-948, Washigton, DC, USA, (June 25-29 2005).
- L. Prodan, M. Udrescu, M. Vlăduţiu, "Survivability of Embryonic Memories: Analysis and Design Principles", IEEE NASA/DoD Conference on Evolvable Hardware (EH'05), pp. 280-289, Washington, DC, USA, (June 29 - July 1, 2005).

This work was prepared by 3 Ph.D. reports presented in the Computer Engineering Departmant of University "Politehnica" of Timişoara, with the second one, entitled "Quantum Algorithms Implementation: Circuit Design Principles and Entanglement Analysis", being cited in the following book:

- Lee Spector, "Automatic Quantum Computer Programming: A Genetic Programming Approach", Kluwer Academic Publishers, (2004).

The presentation of this Ph.D. thesis, as a whole, was accepted at the ACM SIGDA Ph.D. forum, within the ACM/IEEE Design Automation Conference:

- M. Udrescu, "Using Hardware Engineering in Quantum Computation: Efficient Circuit Simulation and Reliability Improvement", SIGDA Ph.D. Forum at Design Automation Conference, (Anaheim, CA, 2005).

Acknowledgements

This thesis reflects the author's original perspective on reliable quantum circuit simulation and design. Its contributions arise after years of personal strive but – in a multidisciplinary field – the ideas exchanged with others often make the difference. Therefore, in a way, this work belongs also to the people whose ideas found fertile ground in my efforts. It would be impossible to thank, individually, all the people that helped me throughout this process; therefore I will kindly ask for indulgence from all the ones that I may have missed.

Research is not the easiest of jobs – besides sagacity, it takes dedication and confidence, and for acquiring each of these perks one has to be constantly encouraged and helped. I was privileged during this journey, as being lured into the world of research by my first advisor, Professor Radu-Ioan Stoinescu, and then by my mentor, Professor Mircea Vlăduţiu. However, along this difficult path, is due to my parents, my wife and my son, that I augmented my capacity to withstand hitting snags. My work would have never soared without their love and care.

I thank professor Mircea Vlăduţiu for his constant support, for sharing his outstanding knowledge of reliable computer architectures, and for driving me into this computer-engineering-oriented view of quantum computation. A great influence for this thesis came from my colleague and friend Lucian Prodan; together with professor Vlăduţiu we have founded the Advanced Computing Systems and Architectures (ACSA) laboratory, a project which is fed by our common enthusiasm. Our students from ACSA were also very important; Nicola Velciov and Cristian Ruican have constantly helped me with formatting papers and preparing conference presentations.

Two dear friends, Claire Russell and George Heath, provided most of the important papers and books that have shaped the way I approached research. I take this opportunity to thank them again.

I am also grateful to professors Radu Mărculescu and Lee Spector as their advices and encouragements have oriented this thesis on the right track, and to Mr. Florin Talpeş the general manager of SoftWin, who has supported me in order to attend a very important conference.

This thesis has also been supported by the Romanian government grant CNCSIS nr. 643/2005, ACM SIGMicro (2004) and ACM SIGDA (2005) traveling grants.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer science and engineering have reached the degree of maturity where the difficulties and limitations are identified, and the frontiers of computation are tackled in an endeavoring paradigm-shifting effort. At first glance, this burden is mainly supported by computer science; however, the complexity of today's computational problems has made this quest ubiquitous.

Researchers from all the computing directions – computer science, computer engineering, software engineering, and information systems – are striving towards grand objectives, as the classical computing establishment does not even provide incentive for salvation from what it seems a dreadful curse: even the most marginal improvements are yielded with exhausting efforts. The promise is that new technologies, accompanied by new computing paradigms will save the day.

There are two main reasons to be lured by this new trend; one has a qualitative nature, whereas the other deals with the quantitative aspects of computation. This thesis deals with two emerging technologies: quantum computing and adaptive, reconfigurable computation. They both have dual qualitative-quantitative motivations, although one may note the potential epistemological arrogance of quantum computation. What Richard Feynman has foreseen [31] – today's quantum computing frenzy – also opened a topical interest discussion on the inner nature of computation [18]. Defined as "computation that uses atomic scale dynamics" [96], offering a framework for powerful algorithms, quantum computation may also be required as Moore's law dictates 1 atom/bit in 2010 $\sim$ 2020.

The qualitative approach is related to the very demanding attempt, in terms of resources, to simulate quantum processes. The majority of these quantum processes, in order to be simulated on a classical computer, require exponential algorithms. Richard Feynman [30] has sugested that all these problems could be overcomed by building a quantum computer. This potential quantum computer "will have no problem in simulating itself" [30]. Given this fact, a new question was rised: if the hypotetical quantum computer is able to simulate quantum processes in polinomial time, then it can solve in the same manner other *hard problems* [70] (i.e. problems that, given a classical computational context, could be approached in the best case with exponential algorithms). In 1985, David Deutsch theoretically built a universal quantum Turing machine [24], a priceless tool for defining new algorithmic complexity classes: `(EQP, BQP, BQTime(T(n)))`. Also, it was demonstrated that, at least theoretically, $\mathtt{P} \subseteq \mathtt{EQP}$ and $\mathtt{BPP} \subseteq \mathtt{BQP} \subseteq \mathtt{P}^{\#\mathtt{P}} \subseteq \mathtt{PSPACE}$ [11]. Almost a decade after Deutsch's breakthrough article, Peter Shor has published the first quantum algorithms able to solve, polynomially, integer factoring and discrete logarithms. These two are *hard problems* in the context of classical computation [89].

The second motivation, the quantitative approach, is a discussion on the limits of integrated circuits manufacturing technology. Today, there is a lot of discussion whether Moore's law is obsolete or not. But a lot of the arguments related with this discussion were pointed out a long time ago, starting with the 1960s: the works of Rolf Landauer, and later Charles Bennett, constitute the basis of reversible computing theory [8][9][53]. The universal gate for the reversible computing the Toffoli gate is also extremely important for quantum circuits [10]. The practical implementations of the quantum gates, networks and circuits, including error detection and correction, are based on several technologies, independently developed: Ion Trap at the National Institute for Standards and Technology (NIST), Cavity QED at Caltech, and NMR at Stanford, Berkeley and MIT [15][32].

With all this spectacular evolution of the quantum computing theoretical aspects and the unquestionable technological progress, it's still not clear whether the prospect of building a scalable quantum computer is feasible. The answer to the problem of quantum computer feasibility must come not only from the physicists making their esoteric research in very expensive laboratories. Computer engineering has also its part in this huge effort. The simulation of quantum circuitry, although employs exponential algorithms, has given eloquent results about the impact of errors and the opportunity of building fault tollerant quantum circuits [63]. Simulation of quantum circuits itself has capitalized on computer engineering research efforts, while vital aspects as fault tolerance would not be possible to approach without taking into account the engineering issues.

1.1 Motivation

Although quantum computing is, undisputedly, a paradigm that was created and advocated mainly by physicists and mathematicians, today it is widely considered that engineering is also vital [62][77]. As a fact, the most prestigious computer engineering scientific conferences and journals have adopted emerging technology tracks where quantum and reconfigurable computing are highly placed.

This thesis was motivated by the attempt to bring together classical computer hardware design and test and the novel, emerging technologies. This also was a source of inspiration for establishing a new computing laboratory at "Politehnica" University of Timişoara: the Advanced Computing Systems and Architectures (ACSA) Laboratory. Its fundamental, innate principles, which draw their essence from the above mentioned thesis motivation, are presented in Figure 1.1; it is a set of interfering computing fields spanned by the classical and novel computation axes.

In the quantum computational framework there are polynomial time solving algorithms, for problems having exponential classical solutions. The quest is – on one hand – to search if there are other possible effective quantum algorithms and – on the other hand – to be able to produce efficient implementations for the already known algorithms. The most feasible implementation of quantum algorithms is based on the quantum circuit (gate network) model [25] [27]. Our work aims at bridging the gap between classical hardware CAD with design automation techniques and quantum circuit design rules. This attempt would be extremely difficult without the possibility of efficient quantum circuit simulation. Thus, our first direction was to try using Hardware Description Languages (HDLs) for simulating quantum circuits, because their property of being able to describe – in a compact manner – the circuit with both structural and behavioral (functional) architectures isolates the inner source of simulation complexity: the entanglement [29][77].

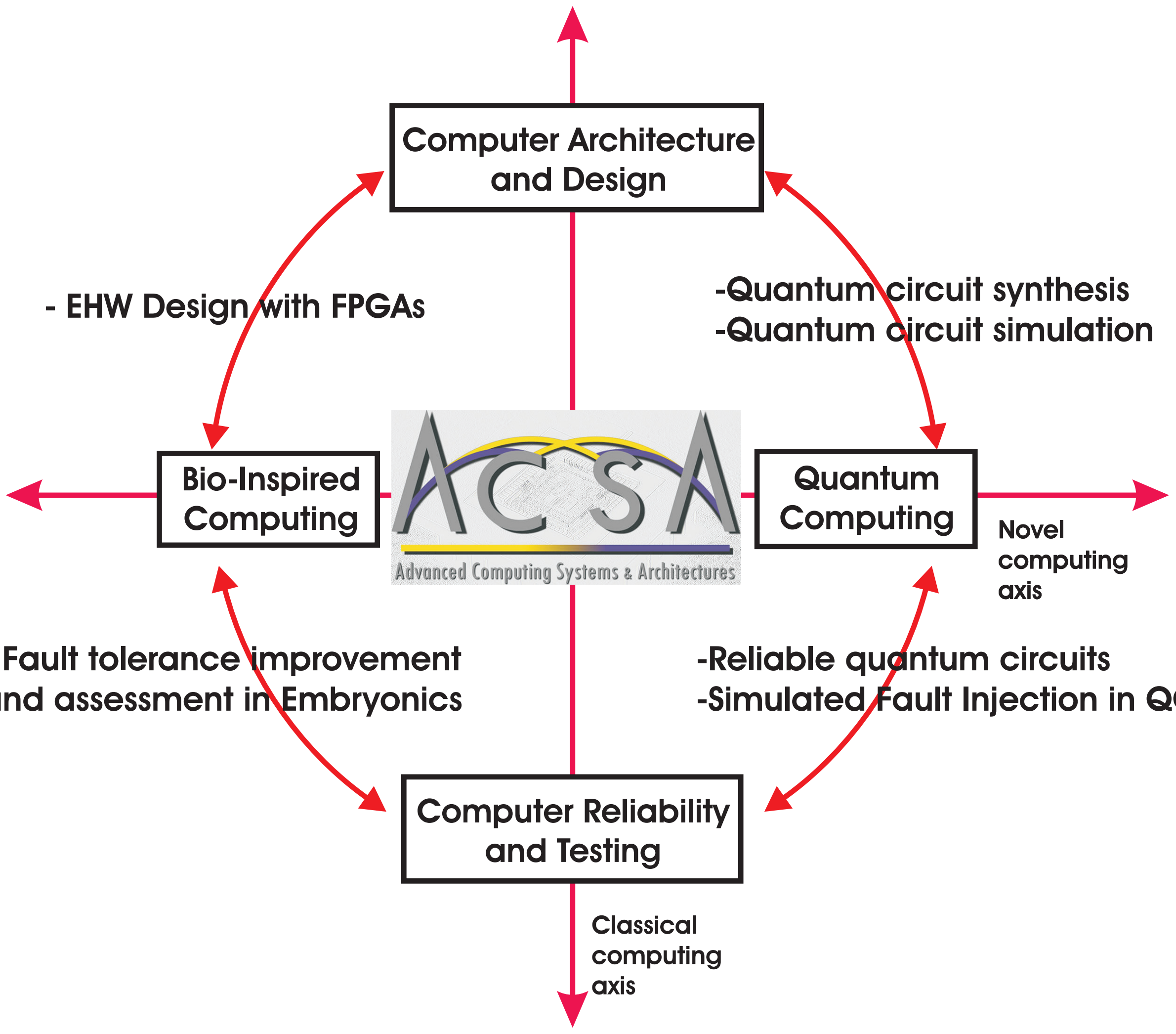

Computer Architecture and Design
- EHW Design with FPGAs
-Quantum circuit synthesis
-Quantum circuit simulation
Bio-Inspired Computing
ACSA
Advanced Computing Systems & Architectures
Quantum Computing
Novel computing axis
- Fault tolerance improvement and assessment in Embryonics
-Reliable quantum circuits
-Simulated Fault Injection in QC
Computer Reliability and Testing
Classical computing axis

Figure 1.1: ACSA laboratory overview.

The other direction of this PhD work is finding common ground for reliability techniques and assessment methodologies from the Embryonics project and fault tolerant quantum computation. Embryonics is a biologically inspired reconfigurable hardware project [57], which is suitable for attaining reliability in aggressive, critical environments [79], similar to quantum computation in terms of fault model and fault occurrence frequency. Adopting the accuracy threshold as reliability measure in Embryonic memories is benefic [80]. Also, when considering a reconfigurable strategy (reconfigurable quantum gate arrays – rQGAs) in quantum computation fault tolerant stabilizer encoding, the appropriate reliability measure may be drastically improved.

This second direction opens up a new discussion, which could be of great importance for computer science in general. Initially, the quest is to design a quantum circuit that is suitable for supporting evolvable hardware applications. As the reconfigurable (or programmable) quantum gate arrays are not new – being theoretically underpinned [61], the entire problems relies on finding a quantum computation implementation for genetic algorithms. As defined in the fundamental literature of this field, the evolvable hardware is a reconfigurable device (circuit) which is configured by evolutionary means, usually a genetic algorithm (EHW = RHW + GA, or evolvable hardware = reconfigurable hardware + genetic algorithms) [102]. The Quantum Genetic Algorithms or QGAs (i.e. genetic algorithms running on a quantum computer) are controversial as far as their implementation is concerned; in this thesis a new perspective is presented: by making use of the exponential quantum computer parallelism, the maximum finding algorithm [2] and a specially designed oracle circuit, the genetic algorithm is reduced to Grover search [37] which solves the problem in $\mathcal{O}\left(\sqrt{n}\right)$ time.

1.1.1 Industry requirements

The motivation presented in this section may seem theoretical and pretty much detached from the actual industry problems. But the fact is that the industry is seriously taking into consideration the aspects related to the emerging technologies, and quantum circuits in particular.

The new challenges facing supercomputing applications will put a strain on the supporting technology. It is clear that we will need to build at least zeta-flops computers in order to deal with some very complex unsolved problems like: long-duration climate modeling, controlled-fusion reactor simulation, network security simulation, molecular modeling, and so forth [20]. In this context, the software and architecture requirements must be met by the underlying technology, and it seems that the classical solutions are not good enough [19].

The industry representatives have quickly reacted to these emerging problems, and founded a global organization called ITRS (International Technology Roadmap for Semiconductors), which is jointly sponsored by European Semiconductor Industry Association, Japan Electronics and Informational Technology Industries Association, Korea Semiconductor Industry Association, Taiwan Semiconductor Industry Association, and Semiconductor Industry Association from U.S.A. As this organization defines its documents, they are about a continuous evaluation of the semiconductor technology requirements, aimed at increasing the performance of the integrated circuits. This effort is supported by industry, suppliers, academia, research groups, and governments [126].

The results of the ITRS assessments are published as ITRS reports, which are annually updated. The 2004 update contains a report on "Emerging Research Devices" [126]. Within this document, the "Emerging Research Architectures" chapter contains a section called "Coherent Quantum Computing" ([126], pages 37–40), in which the quantum computation fault tolerance requirements are evaluated. Table 64, "Emerging Research Architecture Implementations" [126] is listing the following defect tolerance imperative for coherent quantum computing devices: "error correcting algorithms needed". This industry conclusion is further stressing the importance of quantum FTAMs (Fault Tolerance Algorithms and Methodologies), that are making an important direction of this thesis (Chapter 4).

The "Emerging Research Materials" chapter from the same document was added in the

2004 update for the first time, emphasizing in the ”Modeling and Simulation” section ([126], page 55) the importance of reducing simulation complexity. As already mentioned, an important part of the present thesis deals with this extremely important research problem (Chapters 2 and 3).

1.2 Thesis goals

As this work is structured on two main directions, one of which having an important extension, there are 3 main objectives: efficient simulation of quantum circuits, improving the dependability of the quantum circuits with rQGA, and implementing QGAs (Quantum Genetic Algorithms).

Attaining these objectives means that some very important aspects of quantum computation are approached. First of all, simulating quantum computation processes in general – and quantum circuits in particular – is usually exponential. The source of this exponential simulation complexity depends on the level of abstraction that is used. From our perspective, which is on the unitary (or gate) level, the main source of simulation complexity is the entanglement [62] [77]. On the other hand, the entanglement is essential for making quantum algorithms more efficient than their classical counterparts [29] and it cannot be removed from the quantum states involved in simulations. However, we can use some *a priori* knowledge about the particular pattern of the states processed by specific algorithms – the so-called *simulation shortcuts* [96] – along with clever state coding techniques [107], in order to reduce the computational burden dictated by simulation of quantum algorithms on classical computers.

The second objective deals with a extremely important issue, because in quantum computation dependability is not just a quality indicator, it is vital [75][76]. The state-of-the-art here is intended to prove the feasibility of quantum computation, by improving the *accuracy threshold* as main reliability attribute. The already developed techniques are using special state encoding (similar to classical ECCs), concatenated coding and structural redundancy, so that for an component fault rate of the order of ξ, the overall circuit error rate would be of the order of ξ^2. For a sufficiently small ξ, given by intrinsic component fault tolerance (like, for instance, the ones described by [1]) the overall circuit reliability can be sufficiently improved.

However, the assumed fault model [76] is not taking into account the correlated errors, even if these errors are unavoidable from an engineering point of view, at the same time making the concatenated coding effort useless [109]. The solution can arise from a much flexible implementation platform, under the form of pQGAs (programmable Quantum Gate Arrays) or rQGAs (reconfigurable Quantum Gate Arrays) [61][109].

It was already mentioned that, generally, the artificial intelligence approach considers almost all computations as searches, and therefore quantum computation can be involved as Grover’s algorithm provides important search speedups [96]. Also, important work [43][44] proves that Grover’s algorithm can be used for significant speedups of specific purpose searches.

After defining rQGA structures, the initial intention was to define a framework for implementing Evolvable Quantum Hardware (EQHW) as rQGA + QGA (Quantum Genetic Algorithms). The QGAs, as defined by Giraldi *et al.* [33] are the quantum algorithms that perform genetic-based searches. Our approach reduces any GA in quantum computation to

Grover's search, by defining a special purpose oracle quantum circuit and performing the quantum maximum finding algorithm [2].

1.2.1 Simulation problems

In order to approach the aspects concerning the simulation of quantum computation, the research must rely on a quantum computer model, based on formal mathematics. In turn, simulation is used to reveal important issues like entanglement effect, error impact and quantum error correction [90], or techniques for building quantum hardware [121].

Quantum computing itself emerged from the attempt to simulate quantum systems [30]. But, although the simulation is often considered to be a tool used in theoretical approaches, it could also be employed for quantum hardware design, i.e. adapted design automation and computer-aided design techniques. In doing that, one must select an appropriate model. In his surveys [65][66], Omer summarizes the models used by quantum computing researchers, for theoretical and practical purposes: *mathematical*, *machine*, *circuit*, and *algorithmic*.

These models are valid for any computing device, classic or quantum. Therefore, the general models from above have classical expressions with quantum counterparts. From a mathematical point of view, a computer is modeled by *partial recursive functions* having as quantum counterpart *unitary operators*. The machine model in classical computing is given by the *Universal Turing Machine* (TM). For quantum computation we have the *Universal Quantum Turing Machine* (QTM) [11][24]. The circuit model is the *logical gates circuit model* for the classical digital computer and *quantum gates circuit (or network) model* for the quantum computer. Also, from the algorithmic perspective, the model for classical computer is *the universal programming language* with the *quantum programming languages* (QPLs) as quantum counterpart. Of course, there are other important quantum computing models like the *Quantum Cellular Automata* [112], but they are out of the scope of this thesis.

The circuit model

If the goal is to build a quantum computing device, then we must employ some specific design techniques (inspired from classical CAD). The substantiation of these techniques must be sustained by an appropriate model. The best conceivable model is the *quantum gates model*. Simulation according to this model [25] means that we perform *gate-level* or *unitary-level simulation* [96].

Because of the extreme hardness in designing efficient quantum algorithms [62][77], there are just few such examples. An efficient quantum algorithm is an algorithm running on a quantum computer, able to solve a problem dramatically better (i.e. polynomial time) than a classical algorithm. For example, in computational complexity terms, an efficient quantum algorithm solving a problem could be in `BQP` with the best classical algorithm solving the same problem in `EXP`. The point here is obvious: given the quantum algorithm design limitations it would be extremely unprofitable to build an expensive universal machine, which would be able to outperform a classical computer only when solving a few specific problems. Hence the best prospect for involving quantum computing in computer manufacturing is an *universal classical computer with a quantum oracle* [66]. This quantum oracle could be seen as a co-processor – the processor is a classical computer passing specific hard tasks to the quantum co-processor, which has several hardwired *hard* algorithms (see Figure 1.2).

From our view, we need a tool to simulate quantum algorithms and, at the same time, to help designing quantum gate networks according to the circuit model. In classical computing, for hardware design we have such simulation tools: the Hardware Description Languages (HDLs) [5][6][22].

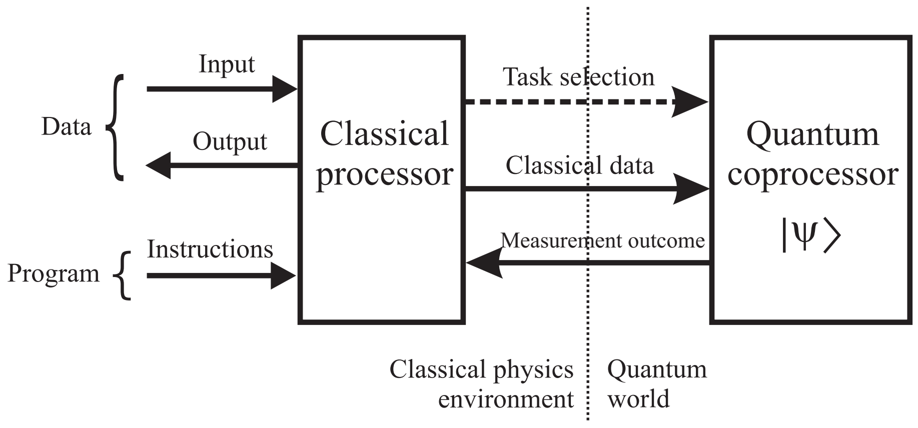


Figure 1.2: The Universal Classical Computer with a Quantum Oracle, after Omer [66]. The Quantum Coprocessor must include a task selection logic and a classical to quantum translator. The other way conversion is made by employing measurement.

Entanglement

The information storage unit in quantum computing is the quantum bit or qubit, which is presented here in bra-ket notation [62]. Any qubit $|\psi\rangle$ is a normalized vector in a $\mathcal{H}^2$ Hilbert space, with $\{|0\rangle, |1\rangle\}$ as the orthonormal basis: $|\psi\rangle = a_0|0\rangle + a_1|1\rangle$. Parameters $a_0, a_1 \in \mathbb{C}$ are called quantum amplitudes, and represent the square root of the associated measurement probabilities of the basis states $|0\rangle$ and $|1\rangle$ respectively, with $|a_0|^2 + |a_1|^2 = 1$.

The qubits can be organized in linear structures called quantum registers, encoding a superposition of all possible states of the corresponding classical register. For a n-qubit quantum register, its corresponding state is a normalized vector in a $\mathcal{H}^{2^n}$ space, $|\psi_r\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle$, where $\sum_{i=0}^{2^n-1} |a_i|^2 = 1$, $i \in \mathbb{N}$. When the individual qubit states are known (for example $|\psi_A\rangle = a_0|0\rangle + a_1|1\rangle$ and $|\psi_B\rangle = a_2|0\rangle + a_3|1\rangle$) the tensor product gives the overall state $|\psi_A\rangle \otimes |\psi_B\rangle = a_0a_2|00\rangle + a_0a_3|01\rangle + a_1a_2|10\rangle + a_1a_3|11\rangle$. The matrix representation provides for a straightforward form of the quantum state; the above 2-qubit tensor product is $\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \otimes \begin{bmatrix} a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_0a_2 & a_0a_3 & a_1a_2 & a_1a_3 \end{bmatrix}^\dagger$.

For a quantum register state, we have entanglement iff it cannot be represented as a tensor product of its parts [62]. Let us consider the following 2-qubit example, where $|\psi_1\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$ and $|\psi_2\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$; we say that state $|\psi_1\rangle$ is not entangled while $|\psi_2\rangle$ is entangled, because $|\psi_1\rangle = |0\rangle \otimes \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right]$, but there are no $|\phi_1\rangle$ and $|\phi_2\rangle$ qubits so that the relation $|\psi_2\rangle = |\phi_1\rangle \otimes |\phi_2\rangle$ is satisfied.

The quantum circuits are constrained networks of gates, with no cloning and no feedback allowed [10][62]. The quantum gate is the physical device implementing an unitary operator, which represents the quantum state transform. Due to the unitary property, all quantum operators are reversible.

In this thesis approach, **the entanglement is considered as the main source of simulation complexity**. The explanation, assuming the matrix representation of quantum states, resides in the following example. If a quantum circuit is processing a 16-qubit state, then for non-entanglement the circuit handles 16 $[2 \times 1]$ matrixes, whereas for the entanglement situation it will have to handle 1 $[2^{16} \times 1]$ matrix. Summarizing, in the absence of entanglement processing a 16-qubit state 32 matrix elements (i.e. complex numbers) are handled; but when entanglement occurs the number of matrix elements to be handled becomes exponential: 2^{16}.

Related work

Because the stakes are high when it comes to quantum computer simulation, and its complexity reduction techniques, there are many attempts to build efficient quantum computer simulators. These attempts are aimed at different level of abstraction.

For instance, Obenland and Despain [63][64] have designed a simulator at physical level – corresponding to the trapped ions technology [15]. This simulator was used for assessing the feasibility of the trapped ion technology by modeling the quantum errors as laser device angle errors [96].

At the higher level – the algorithmic level – a fine example is provided by Omer's *Quantum Programming Language* [65][66]. This metalanguage is very good for synthetically describing the quantum algorithms but, because it does not deal with the actual implementations of the algorithms, it cannot approach any simulation shortcut speculation [96].

This thesis' concern is related to the unitary-transformation-level or gate-level simulation. From our computer hardware, engineering view, the most representative simulator at this level is QuIDD Pro – developed by the Quantum Circuits Group from University of Michigan [73][114][115][116][117]. This simulator is based on a special quantum state encoding, inspired by the Binary Decision Diagram theory. Due to the fact that the QuIDD encoding provides compression, the first advantage of this simulator is the reduction of the simulation memory overhead. Also, the simulation runtime is improved, as reported [114][115], although the complexity problem is not fundamentally solved.

The QuIDD encoding process is far for being efficient, as this condensed representation is not a straightforward one. Also, processing the encoded state attains efficiency only in some particular cases. Fortunately, when simulating specific useful quantum algorithms, advantageous state patterns are encountered – the so-called *simulation shortcuts* [96] – therefore this simulation framework is more efficient than the previous ones, while remaining robust and gate-oriented. However, this simulator cannot provide means for performing any kind of tradeoff between time and space complexity.

Still, the simulation theoretical complexity [111] [118] is far from being attained. The conditions for polynomial quantum computer simulation was also defined in theory, under the form of Gottesman-Knill theorem [62]. The QuIDD simulation procedure uses an engineering approach, which is not trying to solve the complexity problem the way it is prescribed by the quantum computer science theory. Summarizing, the most robust and representative gate-level quantum computer simulator has the following advantages and drawbacks:

The advantages:

A1) it significantly reduces the simulation memory overhead;

A2) the simulation runtime is improved in comparison with previous gate-level simulators;

A3) the special encoding technique uses a compressed symbolic representation, which is similar to binary decision diagrams, that are familiar to the computer engineers;

A4) the encoded quantum states are appropriate for capitalizing on the *simulation shortcuts*.

With all these very important achievements, there are still some **problems to be solved:**

P1) the simulation runtime improvement does not even make the simulation sub-exponential;

P2) the memory-time tradeoff is impossible within this simulation framework;

P3) the solution does not tackle the fundamental causes of the simulation complexity;

P4) the experimental results are provided only for one quantum algorithm: Grover's algorithm.

1.2.2 Status-quo in reliable quantum computation

Unlike classical computation [7], where we have intrinsic fault-tolerance of the components and therefore dependability is just a quality indicator, in quantum computation it is **vital**. The quantum world has an erroneous nature, because the macroscopic environment is constantly trying to measure the very fragile superposition of basis states [62][75][76][77]. In these conditions, the destructive effect of the decoherence [62][77] phenomenon can be considered as ubiquitous.

The assumed fault occurrence model [75][76] is influenced by the need to assess the feasibility of implementing quantum hardware [63]. Therefore we deal with the following assumptions:

- the faults are single and their occurrence is governed by probabilistic rules;
- the fault are not correlated, neither in time or space;
- in some evaluations, we are dividing faults in two categories: *store faults* and *processing* or *gate faults*.

In quantum computation we have 3 types of qubit faults:

- *bit-flip*, where the effect is described by the basis state mappings $|0\rangle \mapsto |1\rangle$, $|1\rangle \mapsto |0\rangle$, and the following equation $|\psi\rangle = a_0|0\rangle + a_1|1\rangle \xrightarrow{\text{fault}} a_0|1\rangle + a_1|0\rangle$;
- *phase-shift*, described by $|0\rangle \mapsto |0\rangle$, $|1\rangle \mapsto -|1\rangle$, and $|\psi\rangle = a_0|0\rangle + a_1|1\rangle \xrightarrow{\text{fault}} a_0|0\rangle - a_1|1\rangle$;

- *small amplitude* which are similar to analog errors, and affect the qubit amplitudes.

Here, the bottom line is that all these fault types can be reduced, by appropriate techniques [76][75][77], to bit-flip faults.

The first drive, when trying to implement fault tolerant techniques in quantum computation, is to map the already known techniques from classical hardware. This job is not straightforward even when the reference comes from reversible classical circuits [88][73], as the quantum computation generates some constraints and additional problems to be solved.

Quantum computation constraints:

- the observation destroys the state;
- information copying is impossible.

The inner principle of using error-correcting codes (ECCs) is to use observation. Moreover, all the structures build upon the ECC principles are applying structural redundancy, which requires information copying.

Quantum computation additional problems:

- we need to be able to get state information without destroying it, therefore we are forced to use ancilla qubits;
- we need a fault tolerant recovery process, otherwise the coding fault tolerant techniques become useless
- the phase-shift fault propagates backward, so we have to apply special techniques designed for thwarting the massive spread of these errors.

In order to deal with the encountered problems, some very important quantum fault-tolerance strategies have been developed [62][75][76][77].

Strategies for attaining fault tolerance:

- digitizing small errors [76];
- using ancilla qubits in order to measure the information without destroying it;
- assuring ancilla and syndrome accuracy for a fault tolerant recovery process;
- appropriate quantum ECCs for detection and correction.

With the ECCs, a syndrome is computed, thus revealing the nature of each qubit, which corresponds to one of the following situations (assuming that the correct qubit value is $a_0|0\rangle + a_1|1\rangle$):

- no fault, the actual qubit expression is $a_0|0\rangle + a_1|1\rangle$;
- bit-flip, the actual qubit $a_0|1\rangle + a_1|0\rangle$;
- phase-shift, the actual qubit $a_0|0\rangle - a_1|1\rangle$;

- both bit-flip and phase-shift, the actual qubit having the following expression $a_0|1\rangle - a_1|0\rangle$.

Correcting a qubit fault, means applying one of the following 1-qubit unitary transformations: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ for the bit-flip, $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ for the phase-shift, and $\begin{bmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{bmatrix}$ for both faults.

The state-of-the-art in quantum ECCs is represented by Steane encoding [97][98] with its generalization provided by stabilizer codes [16][35][36]; and the assessment of quantum fault tolerance algorithms and methodologies (qFTAMs) is made by using a measure called *accuracy threshold* [76][124]. The accuracy threshold is the component fault rate that still allows the overall correct computation when employing qFTAMs.

The accuracy threshold estimates indicate that arbitrary long fault tolerant computation is possible only if another techniques are applied (i.e. concatenated coding [76][77]).

Our approach of qFTAMs starts with a critical view of the error model and the concatenated coding technique, and prescribes the replacement by defining a technique based on the so-called reconfigurable quantum gate arrays (rQGA). The accuracy threshold assessment proves that this technique brings a significant improvement [109].

1.2.3 Genetic algorithms and quantum computation

Although this thesis direction emerged from an engineering effort – implementing evolvable quantum hardware (EQHW) – its contributions may have significant impact in the *computer science* area.

By clearly identifying its most major problems and limitations, computer science has become aware of the so-called *computing frontiers* [62][70]. 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 for the most sophisticated approaches.

In this context, inspiration was mainly found in biology and physics: bio-inspired computing [57] and quantum computing [62] are considered as possible solutions. The optimism is fed by theoretical and practical achievements. Genetic algorithms and evolvable 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 as fault tolerance and reliability in critical environments [79][80][81][82]. Moreover, quantum computing seems to draw even more power from its exponential parallelism: Peter Shor has proven that a classical exponential problem (integer factorization) can be solved in polynomial time [89][91].

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 [96]. Researchers already follow the path of so-called Quantum Evolutionary Programming (QEP) [33] with outstanding results [94]. For instance, the best approach for automated synthesis of quantum circuits [121] uses genetic programming [55][56][74][50][51]. Also, quantum algorithm design can be approached by evolutionary means [95]. In fact, the majority of such applications address quantum computation design issues regarding quantum algorithms and implementations [94]; they are all part of QEP's sub-area called Quantum Inspired Genetic Algorithms (QIGAs) [33][60]. The other sub-area, called Quantum Genetic Algorithms (QGAs), tries to implement genetic algorithms in a quantum computation environment [33][86] [87][93] in order to capitalize on the quantum computation exponential parallelism.

This thesis proposes a new perspective on QGAs, by showing that the genetic algorithm strategy is essentially different in quantum computation: crossover and mutation are not required, because finding the best fitness can be reduced to Grover's algorithm [37][38]. The search space is entirely covered by the QGA because all individuals are encoded in a superposition state (at the same time), also fitness values generated for all individuals are encoded as a superposition of basis states (at the same time) in a quantum register. As opposed to classical GAs where the best individual-fitness pair may not be available because the population is limited, in Quantum Computation the best individual is available.

1.3 Objectives summary

From our proposed engineering view, there are 5 main objectives when approaching the simulation and design of reliable quantum circuits, which are presented here with the means to achieve them:

$\Omega 1$) Efficient simulation of quantum algorithms, at gate-level, by employing:

- a Hardware Description Language (HDL) framework;
- the bubble-bit encoding technique for capitalizing on the "simulation shortcuts" [96];

$\Omega 2$) Defining a simulated fault injection framework in quantum circuits (the QUERIST project) by using:

- our HDL-based quantum circuit simulator;
- the adapted version of the fault injection (and assessment) techniques from classical hardware design [84][85][45];

$\Omega 3$) Exploring the allowed design techniques, for building reconfigurable Quantum Gate Arrays (rQGAs) by:

- studying the quantum limitations of this concept;
- adapting the solutions from classical reconfigurable computing;

$\Omega 4$) Employing rQGAs, in order to improve the dependability of quantum circuits, by

- identifying the drawbacks of the state-of-the-art in reliable quantum computation theory;
- using the quantum configurations for rQGAs, as source of exponential structural redundancy;

Ω5) Designing adaptive, evolvable quantum circuits, by means of:

  - already designed rQGAs;
  - building the circuits for Quantum Genetic Algorithms (QGAs) implementation.

1.4 Thesis outline

The thesis structure is related to the objective list from Section 1.3: Chapters 2 and 3 correspond to objective Ω1, a part of Chapter 3 corresponds to Ω2, Chapter 4 deals with Ω3 and Ω4, while Chapter 5 has to do entirely with objective Ω5. Finally, the conclusions are presented in Chapter 6: a summary of the Ph.D. work, the main contributions, and prospective thoughts.

Chapter 2

The HDL-Based Simulation Framework

This chapter tries to find common ground between classical circuit design techniques and quantum computation, by identifying quantum circuit specification and simulation tools under the form of Hardware Description Languages (HDLs). The HDL-based simulation approach could reduce the complexity of quantum circuit simulation, by considering entanglement as the main source of gate-level simulation complexity and isolating it in an automated manner. This is possible by taking advantage of the HDL feature of describing a circuit with both structural and functional architectures. We also performed an analysis of our methodology effectiveness, for the arithmetic circuits involved in Shor's algorithm and the circuits implementing Grover's algorithm.

2.1 Preliminaries

Today, the circuit model of quantum computation [25] is considered as the most feasible, and we use it in order to describe the coprocessor. It is seen as a succession of consecutive quantum networks of gates ($QNet_1 \dots QNet_n$ in Figure 2.1) and quantum registers ($QReg$) storing quantum states ($S_1 \dots S_n$) in Figure 2.1), which takes a classical state as input. Also, the final (rightmost, storing state S_n in Figure 2.1) register will be measured in order to obtain the outcome of the quantum computational process. The quantum circuit (or gate network) design techniques are inspired by the classical approaches [22] including classical reversible circuit design [103], always taking into account the quantum mechanical features [10]. On the other hand, automated design techniques and Computer Aided Design (CAD) [6][22] enhance classical hardware design; in this context, our main effort is the attempt to adapt these techniques to quantum circuits.

2.2 Quantum computational background

This section deals with the formal representations of quantum computation. Although there are various such formalisms [13], we will present the most used (Dirac's bra-ket) and matrix [62] – the notation that is taken into consideration by our approach [104][105][106][107][108].

At the same time, in this section, we present the quantum algorithms that are simulated in the thesis: Deutsch-Jozsa [26], Grover [37][38] and Shor [89][91].

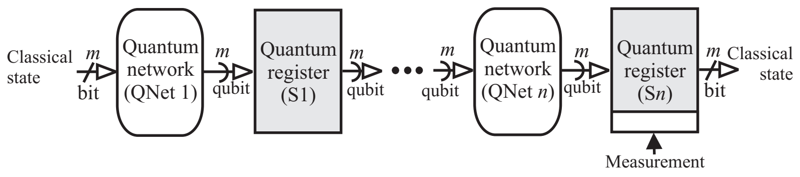


Figure 2.1: The circuit model of quantum computation.

2.2.1 Basic operations and notation

In quantum computation the information storage unit is the quantum bit, or *the qubit*. According to the nature of quantum mechanics theory, the qubit could be seen as a classical bit extension. In bra-ket notation [13], we have: $|\psi\rangle = a_0|0\rangle + a_1|1\rangle$. Using mathematical terms, the qubit is a normalised vector in some Hilbert space $\mathcal{H}^2$, having $\{|0\rangle, |1\rangle\}$ as the orthonormal basis, with $a_0, a_1 \in \mathbb{C}$ being quantum amplitudes. Therefore, the qubit is a superposition of the classical bit states, the eigen-vectors $|0\rangle$ and $|1\rangle$. A measurement of a qubit $|\psi\rangle$ will yield either $|0\rangle$ or $|1\rangle$, (i.e. a classical orthogonal state) with probabilities $|a_0|^2$ or $|a_1|^2$ respectively. Of course, we have $|a_0|^2 + |a_1|^2 = 1$ [62]. Qubits are organized in linear structures: the so-called *quantum registers*. In its quantum version, a register state could be a superposition of all its possible classical states. For a n qubit quantum register, its state is a normalised vector in a 2^n-dimensional Hilbert space:

$$|\psi_r\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle, \text{with} \sum_{i=0}^{2^n-1} |a_i|^2 = 1 \tag{2.1}$$

In Equation 2.1, $a_i \in \mathbb{C}$ and i could be written in a binary system: $i = c_0 \cdot 2^0 + c_1 \cdot 2^1 + \ldots + c_{n-1} \cdot 2^{n-1}$ where $c_j \in \mathbb{B} = \{0, 1\}$. Hence, the register state can be rewritten as:

$$|\psi_r\rangle = \sum_{c_0c_1\ldots c_{n-1}\in\mathbb{B}^n} |c_0c_1\ldots c_{n-1}\rangle \tag{2.2}$$

with

$$\sum_{c_0c_1\ldots c_{n-1}\in\mathbb{B}^n} |c_0c_1\ldots c_{n-1}|^2 = 1. \tag{2.3}$$

When we know the individual state of each qubit from the register, the tensor product is used for obtaining register's overall state:

$$\otimes_{i=0}^{2^n-1}|\psi_i\rangle = \sum_{c_0c_1\ldots c_{n-1}\in\mathbb{B}^n} a_{i0}a_{i1}\ldots a_{in-1}|c_0c_1\ldots c_{n-1}\rangle \tag{2.4}$$

where

$$|\psi_i\rangle = a_{i0}|0\rangle + a_{i1}|1\rangle \tag{2.5}$$

is the state of qubit i. The power of quantum computation is due to the register exponential parallelism: any transformation on that register willl be applied on each superposed eigenstate [62][77].

Matrix representation

The bra-ket notation that we used in the previous section for representing quantum states may not always be the most convenient. A better handling of quantum states having binary–labeled eigenvectors, and their transformations could therefore result by using the matrix representation [62]. Thus, Equation 2.4 becomes:

$$\otimes_{i=0}^{2^n-1}|\psi_i\rangle = \begin{bmatrix} a_{00} \\ a_{01} \end{bmatrix} \otimes \begin{bmatrix} a_{10} \\ a_{11} \end{bmatrix} \ldots \otimes \begin{bmatrix} a_{2^n-1,0} \\ a_{2^n-1,1} \end{bmatrix}. \tag{2.6}$$

where $a_{ij} \in \mathbb{C}$ and the state of the individual qubit i is

$$|\psi_i\rangle = \begin{bmatrix} a_{i0} \\ a_{i1} \end{bmatrix}. \tag{2.7}$$

Measurement

Measurement is the only way to extract information out of a quantum state. It is a truly random quantum operation, and unfortunately destroys the useful exponential parallelism of quantum computation. The measurement outcome is one of the eigenvectors, with an associated probability. For instance, if we consider the quantum register described by Equation 2.1 , then a measurement result can be any of the $|i\rangle$ eigenvectors with a probability of $|a_i|^2$.

Entanglement

For a quantum register state, entanglement occurs iff it cannot be represented as a tensor product of its parts (individual qubits). In a 2-qubit example we could say that state $|\psi_1\rangle$ is not entangled while $|\psi_2\rangle$ is entangled, because:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.8}$$

but there are no qubits $|\phi_1\rangle$ and $|\phi_2\rangle$ to satisfy

$$|\psi_2\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |\phi_1\rangle \otimes |\phi_2\rangle. \tag{2.9}$$

Quantum circuits

In order to perform the steps required by a quantum computational process (i.e. quantum algorithm), networks of gates (circuits) are placed between succesive registers which encode succesive quantum states (see Figure 2.1). Thus, the gate network ($QNet_i$ in Figure 2.1 terms) is processing the information from the register on the left $QReg_{i-1}$), in order to obtain the state encoded in its right neighboring register ($QReg_i$). The devices involved in these circuits are the quantum gates. A quantum gate implements a unitary operator over a 2^n-dimensional Hilbert space (n being the number of qubits processed by the quantum gate) which performs a unitary transformation over a quantum state [62][65]. The description of the unitary transform in both bra-ket and matrix forms is given in Equations 2.10 and 2.11.

$$U = \sum_{x=0}^{2^n-1} \sum_{y=0}^{2^n-1} |x\rangle \cdot u_{xy} \cdot \langle y|, \text{where} \sum_{i=0}^{2^n-1} u_{ix}^* u_{iy} = \delta_{xy} \tag{2.10}$$

$$U = \begin{bmatrix} u_{0,0} & \cdots & u_{0,2^n-1} \\ \vdots & \ddots & \vdots \\ u_{2^n-1,0} & \cdots & u_{2^n-1,2^n-1} \end{bmatrix} \tag{2.11}$$

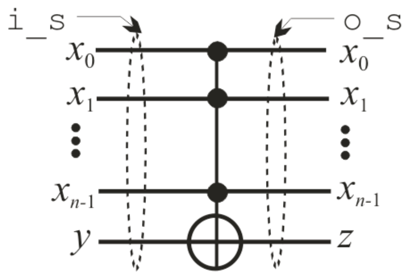
In Equation 2.11, matrix U is unitary, and therefore characterizes a reversible transform. The quantum circuits are networks of gates, built with the following restrictions: no cloning is possible and no feedback is allowed. An example of a 1-qubit gate (Hadamard), with its VHDL description, is shown in Figure 2.2.

$$H : \begin{cases} |0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{cases}$$

```
entity walsh_hadamard_gate is
  port(intrare:in qubit;iesire:out qubit);
end walsh_hadamard_gate;
architecture whg_a of walsh_hadamard_gate is
begin
  iesire(0)<=(1.00/sqrt(2.00))*(intrare(0)+intrare(1))
            after 10 ns;
  iesire(1)<=(1.00/sqrt(2.00))*(intrare(0)-intrare(1))
            after 10 ns;
end whg_a;
```

Figure 2.2: The Hadamard gate: symbol used in diagrams and VHDL description.

Barenco *et al.* [10] proved that $\{XOR, \wedge_0(U)\}$ is a universal set of gates in quantum computation. The $n+1$ qubit transform $\wedge_n(U)$ is a conditional operator, applying 1-qubit unitary operator U on the target qubit iff the other n input qubits are '1'. If $U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \sigma_x$, then the conditional transform is a $CNOT$ operator, which negates the target qubit only for the other input bits being '1'. Thus, the XOR gate is $\wedge_1(\sigma_x)$. In Figure 2.3 we have represented the general $CNOT$ gate on $n+1$ qubits along with its functional description in VHDL (one of the most used HDLs for design and test of classical circuits [6]). Qubits $x_0 \ldots x_{n-1}$ are input qubits, y is the target qubit with its value replaced by z after applying the gate.



$$z = (\wedge_{i=0}^{n-1} x_i) \oplus y$$

```
-- type quregister  is array(natural range <>)of complex;
entity c_not is
 generic (delay:time);
 port(i_s:in quregister; o_s:out quregister);
end c_not;
architecture cnot_a of c_not is
 begin
  process(i_s)
   variable lg:integer;
   variable temp:quregister;
   begin
    lg:= i_s'length;
    assert lg-1 > 1 report "not a valid CNOT gate"
    severity error;
    l1:for i in 0 to lg-1 loop
     if i < lg-2 then temp(i):=i_s(i);
     elsif i=lg-2 then temp(i):=i_s(i+1);
     elsif i=lg-1 then temp(i):=i_s(i-1);
     end if;
    end loop l1;
    o_s<=temp after delay;
 end process;
end cnot_a;
```

Figure 2.3: General form of the *CNOT* gate: corresponding logic diagram and VHDL description.

2.2.2 Quantum algorithms

From a computational complexity point of view [11][101], assuming that `P` $\neq$ `NP`, then there is a class, `NPI` (`NP Intermediate`) of problems that are not solvable by employing polynomial resources, but are not `NP-complete`. It seems that this intermediate class contains the problems with efficient solutions in quantum computation, but has no known efficient classical computation solving. Up to date, we know just a few such quantum algorithms.

As presented by Nielsen and Chuang [62], there are 3 kinds of quantum algorithms that are fundamentally more efficient than their classical counterparts:

- simulation algorithms;
- search algorithms;
- algorithms based on quantum discrete Fourier transforms.

The first kind of algorithms is used for simulating quantum systems on quantum computers. Although this is an extremely important aspect, it is not of our concern in this thesis. The second is represented by Grover [37][38] and Deutsch-Jozsa algorithms [25], while the third kind is best represented by Shor's algorithm [89][91].

Deutsch-Jozsa algorithm

The Deutsch-Jozsa algorithm is an example of quantum computing power. It is solving the so-called Deutsch problem, which is about determining the nature of a unknown decision function g (operating on n bits) in one computational step [25]. The nature of g could be either constant (all $g(x)$ are equal) or balanced ($g(x) =$'0' for exactly one half of the inputs, and ='1' for the other half). The circuit implementing this algorithm is presented in Figure 2.4, where 2 registers are used (n-qubit query register, and 1-qubit answer register) and the relevant states are given by the following equations:

$$|\psi_A\rangle = \sum_{x=0}^{2^n-1} \frac{|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], \tag{2.12}$$

$$|\psi_B\rangle = \sum_{x=0}^{2^n-1} \frac{(-1)^{g(x)}|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], \tag{2.13}$$

$$|\psi_{AA}\rangle = \sum_{k=0}^{2^n-1} \sum_{x=0}^{2^n-1} \frac{(-1)^{x\cdot k+g(x)}|k\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{2.14}$$

After obtaining $|\psi_{AA}\rangle$ the query register is measured and , if g is constant then superposed classical state $|00\ldots0\rangle$ from $|\psi_{AA}\rangle$ will have an amplitude of ±1 with all the other superposed classical states having zero amplitudes; on the other hand if g is balanced then the state $|00\ldots0\rangle$ will have a zero amplitude with at least one other eigenstates having a non-zero amplitude.

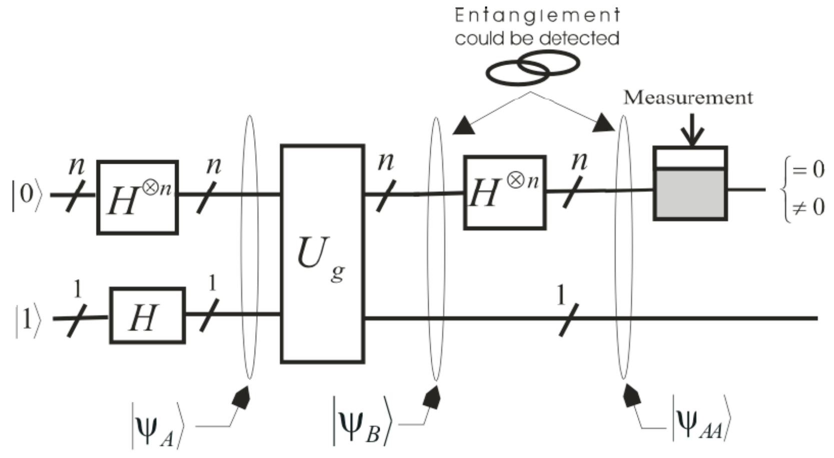


Figure 2.4: Circuit implementing the Deutsch-Jozsa algorithm [62].

Grover's algorithm

The quantum search algorithm, also known as Grover's algorithm [38], is a solution designed to substantially reduce the complexity of search algorithms [14], from $O(n)$ to $O(\sqrt{n})$. It could be considered as a generalization of Deutsch-Jozsa algorithm.

Suppose we have an n-element search space, with each element being labeled by an index $x \in \{0, 1, \dots n-1\}$, $n \in \mathbb{N}$. If the label is represented on m qubits, because of the superposition of all indexes we have $n = 2^m$. Then, we consider that our search problem has k solutions, with $1 \leq k \leq n$. We could reduce this computational task to a decision problem [62] (i.e. defining a function $f_d(x)$, which is '1' only when x is the solution, otherwise is '0').

The circuit implementing Grover's algorithm is presented in Figure 2.5 [37][62]. It has 3 quantum registers: a n-qubit index register, a m-qubit scratch register, and a 1-qubit oracle register. The circuits from point (1) to the measured register are forming the so–called "Grover iteration circuit" which is rippled $O(\sqrt{n})$ times in order to get the result [12] . The U_O circuit is described by the following mapping:

$$U_O : |x\rangle \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \mapsto (-1)^{f_d(x)} |x\rangle \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \tag{2.15}$$

x is one superposed state in the index register in point (1) of Figure 2.5. Because of the Hadamard gate level, the index register contains at this point a superposition of all the possible indexes. Thus, we can say that the Oracle U_O is marking the solution. The Grover iteration circuit as a whole performs the action described in the following equation:

$$U_{Gro} : |\psi_i\rangle \mapsto (2|\psi_i\rangle\langle\psi_i| - I) \cdot U_O \tag{2.16}$$

In order to get the desired state in the measured register, we have to apply the Grover iteration circuit q times, where $q \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{n}{k}} \right\rceil$ [12][62].

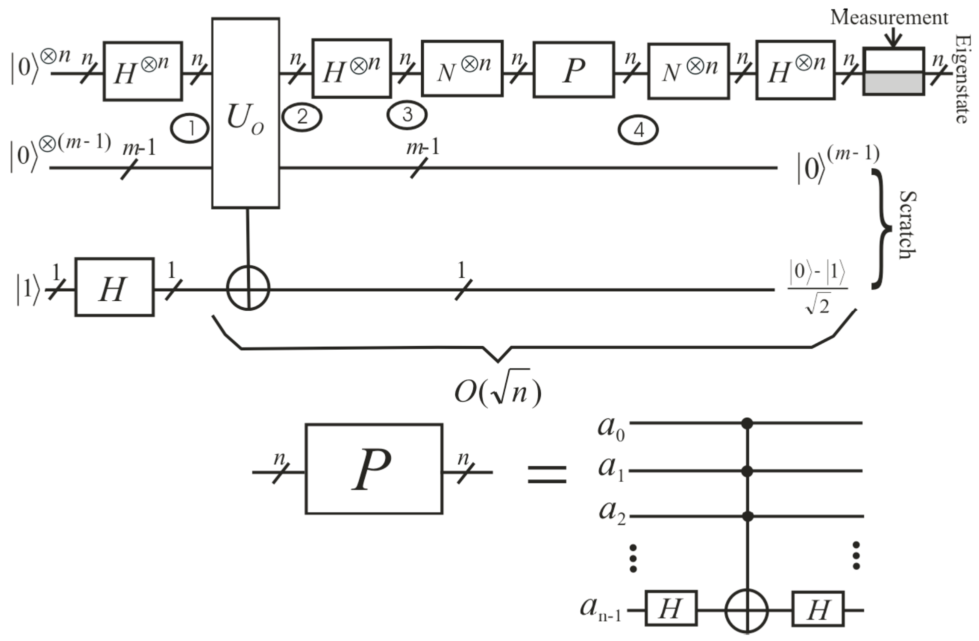


Figure 2.5: Circuit implementing Grover's algorithm.

Shor's algorithm

This algorithm [89] solves the factoring problem in polynomial time, while the best-known classical solution is exponential [54][49]. Here, we have 2 quantum registers: the input register is the left register in the following equations, and the output register is on the right. The goal is to find the factors of a large integer (N) represented on L qubits, by using function $f_{a,N}(x) = a^x \text{mod} N$ where a and N are co-prime integers. The size of the registers is $2L$, due to algorithm requirements [89][91]. Essentially, Shor's algorithm consists of 4 main steps, dictating 5 corresponding main states:

- **State 1:** $|\psi_1\rangle = |\psi_i\rangle|\psi_o\rangle = |0\rangle^{\otimes 2L}|0\rangle^{\otimes 2L}$
- **State 2:** $|\psi_2\rangle = \left(\frac{1}{2^L}\sum_{i=0}^{2^{2L}-1}|i\rangle\right)|0\rangle^{\otimes 2L}$
- **State 3:** $|\psi_3\rangle = \frac{1}{2^L}\sum_{i=0}^{2^{2L}-1}|i\rangle|f(i)\rangle$
- **State 4:** $|\psi_4\rangle = \sqrt{\left\lceil\frac{r}{2^{2L}}\right\rceil}\sum_{i=0}^{\left\lceil\frac{2^{2L}}{r}\right\rceil}|r\cdot i+b\rangle|m\rangle$
- **State 5:** $|\psi_5\rangle = U_{DFT} : |\psi_4\rangle$

State $|\psi_2\rangle$ is obtained by applying a layer of Hadamard gates on the input register. $|\psi_3\rangle$ is yielded by putting in the output register the result of special function f (periodic with period r) over the input register. $|\psi_4\rangle$ is obtained by measuring the output register (the result is m, b is the bias [89]), and $|\psi_5\rangle$ by applying the Quantum Discrete Fourier Transform (QDFT) over the input register [17][39].

2.3 A HDL-based perspective

Building quantum CAD techniques for the design and test of quantum circuits is possible only if their simulation is efficient. Our simulation methodology consists of describing each quantum gate network from Figure 2.1 by functional and structural architectures. When entanglement is detected in a neighboring register, then a functional (behavioral) architecture is selected for the circuit as the only possibility.

The entanglement detecting procedure is automated, by designing specialized non-entangled qubit extraction algorithms. If the extraction algorithm is successful, then in that register there is non-entangled information (groups of qubits), and a structural architecture is possible. As presented in this thesis chapter and references [105][106], the functional architecture will employ exponential resources at simulation, whereas the qubit-level structural architecture means only a polynomial overhead.

This methodology is an enhancement only if there are non-"totally entangled" states (see section 2.4 for definitions). We have performed case studies for our HDL-based framework, involving states appearing in Shor's and Grover's algorithms. The conclusion is that the probability of total entanglement is rising exponentially with the number of qubits. However, when running algorithms for practical purposes, the information encoded in registers tends not to be totally entangled [114].

Nevertheless, total entanglement omnipresence when taking into account "blind" probabilities is a downside of this approach. But at least in some cases this framework offers the solution: state's entangled representation is avoided by *bubble bit* insertion (Chapter 3). This technique is favorable for the polynomial structural architectures with the expense of building some records of size $\mathcal{O}(n^2)$.

2.3.1 Circuit model interpretation

When approaching the design of quantum circuits, this thesis will relate to our quantum hardware interpretation of Gajsky and Kuhn's Y-diagram [23] (see Figure 2.6). We have modified the classical diagram [105] in quantum terms, therefore having three abstracting levels: *architectural*, *unitary* and *particle*. The architectural level corresponds to the algorithm data flow encoded in the overall quantum states. The unitary level is concerned with quantum gate networks (quantum circuits at the gate level), basic quantum unitary transforms, and unitary operators. Finally, the particle level is a technological interpretation related to the physical implementations of the quantum gates, networks and circuits [62].

These abstracting levels could be seen from three different perspectives, or views. The views in the Y-diagram are *behavioral*, *structural* and *physical*. From a behavioral (or functional) view, the quantum circuit is a functional description, without taking into account the implementation issues. In the structural view, the circuit is just the sum of interconnections between basic components, while the physical view is concerned with the physical aspects of circuit implementation.

At the architectural and unitary (logical for classical computation) levels, HDLs (Hardware Description Languages) are able to describe classical and quantum circuits from both behavioral and structural views. Moreover, existing software tools could assist both architectural and unitary synthesis. The physical design and the transformation from unitary to

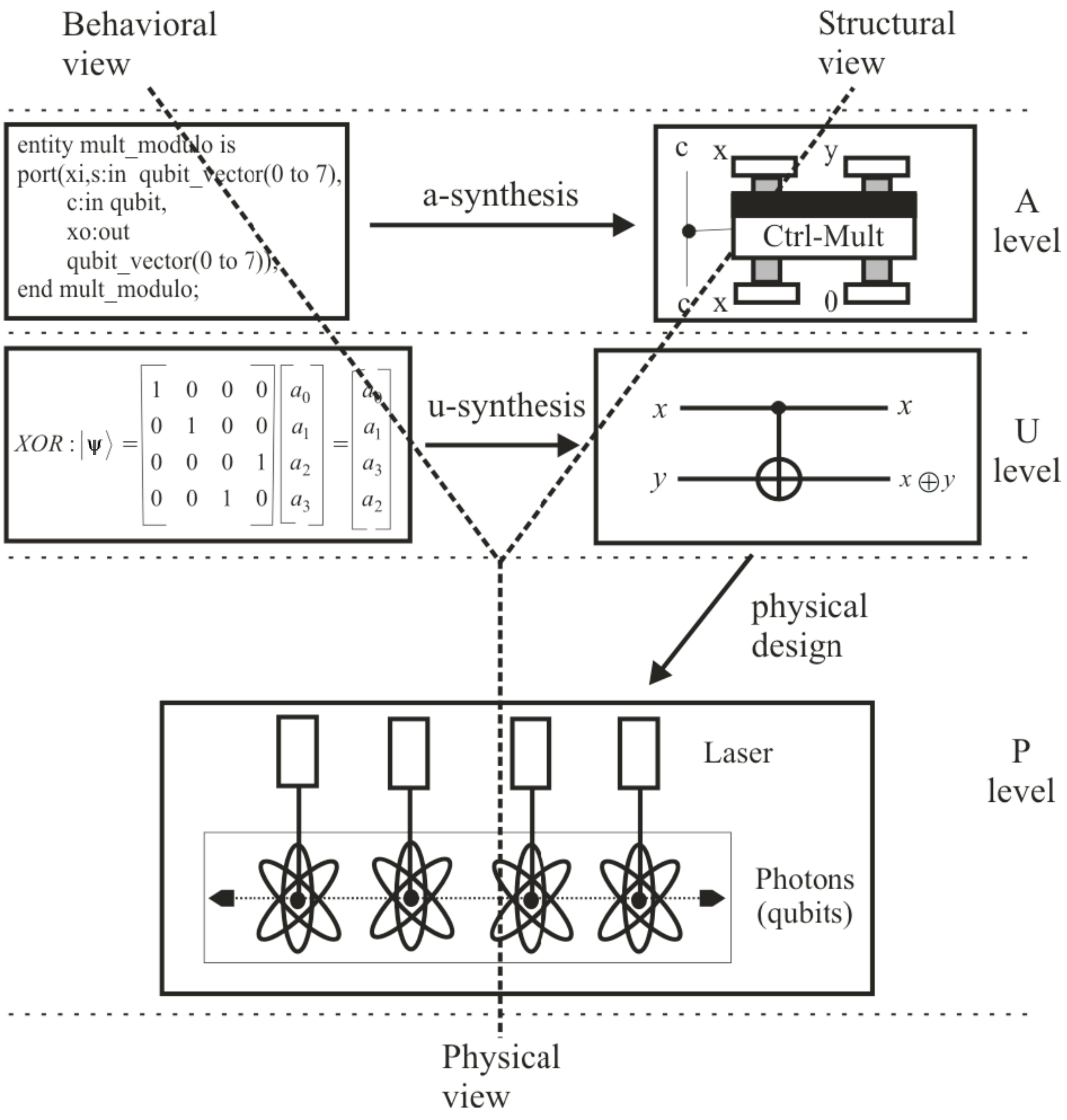


Figure 2.6: Quantum interpretation of the Y-diagram.

particle level, while both important, are not targeted here. Also, the physical implementation (from the particle level in physical view) is not of this thesis' concern.

Barenco *et al.* [10] addressed unitary level issues; other aspects, such as coding and circuit complexity were also consistently addressed [59][58], along with some classicaly-inspired implementations like Programmable Gate Arrays [61]. At the architectural level, advanced classical arithmetic designs [71] could be adapted to quantum circuit architectural needs.

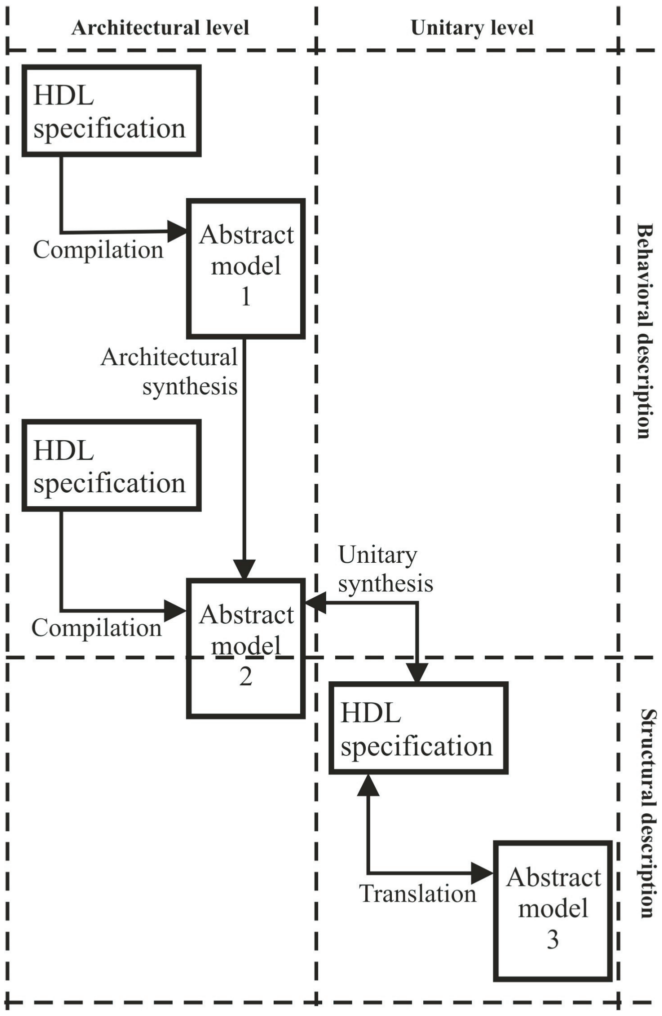


Figure 2.7: The quantum hardware interpretation of HDLs involvement in circuit synthesis.

2.3.2 HDL involvement

Our simulation methodology is based on the circuit model of quantum computation (see Figure 2.1) which was taken into account for other simulation approaches [46][100]. The circuits (gate networks) and the registers are simulated by HDLs (VHDL in particular), which are synthesis tools for classical circuits. In Figure 2.7, we present the possible involvement of HDLs in a quantum interpretation [105][106] of classical circuits' synthesis [22][23].

But, if simulating quantum processes is already an exponential job, is any HDL necessarily a better tool for quantum circuit simulation, as opposed to an ordinary programming language? The best enhancement that could be achieved by employing another simulation tool – for a functional description – is linear, with time and space overhead being exponential. However, the positive motivation is presented in Figure 2.8, based on the HDLs feature of being able to describe the same circuit with both behavioral (functional) and structural architectures [6][23]. This allows for avoiding unnecessary exponential state representations [105][114].

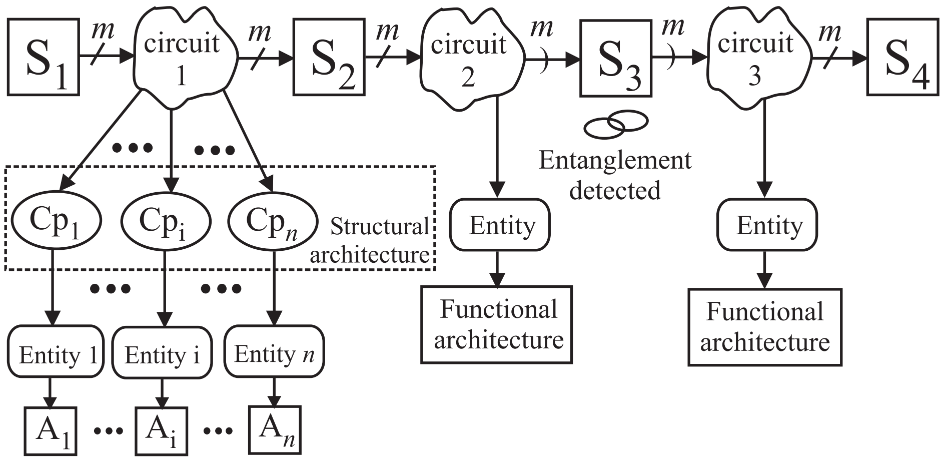


Figure 2.8: Example of approaching the HDL simulation of a quantum circuit.

Whenever entanglement appears, the quantum system cannot be correctly represented as a tensor product [13] of its components' individual states. A correct representation of the overall quantum state, employing linear algebra, means an exponential overhead with respect to the number of qubits. Therefore, when entanglement occurs between two quantum subsystems, their overall state cannot be represented correctly as a reunion (assuming an implicit state composition with the tensor product) of the two individual subsystem states.

Even though researchers are investigating better representations in order to replace linear algebra [114][115][117], handling overall states is a computationally hard (exponential) job. Moreover, when dealing with overall states, there is no truly gate-level simulation of quantum networks (circuits).

We will use the HDLs for describing a circuit in structural fashion, so that a non-entangled state will not be represented as an overall state (with exponential overhead), but as a reunion of individual qubit states. If the previous or the next quantum state is an entangled state, then the quantum circuit must be represented in a functional (or behavioral) manner.

In Figure 2.8 a HDL simulation approach is presented. Two quantum circuits, functionally described, guard the entangled quantum state (S_3). The first quantum circuit (network) is having a structural description because is guarded by 2 non-entangled states (S_1 and S_2). $Cp_1 \dots Cp_n$ are the smallest components of the quantum circuit, and $A_1 \dots A_n$ 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 [96][114] of the quantum algorithm implementation, then the circuit becomes a single quantum gate.

For a practical implementation of this methodology, each circuit must be described both by structural and functional (behavioral) architectures. The structural description is at the unitary level (quantum gate) in modified Y-diagram terms (see Figure 2.6) [105]. 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. The structural case is the desired one because the simulation will require only polynomial resources.

This simulation methodology could be automated by being able to extract the non-entangled qubits from the register, if such is the case. This is also an exponential job if we deal with arbitrary quantum multi-qubit states; however, when dealing with specific algorithm states, it becomes much simpler [105][106].

2.3.3 Methodology implementation

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 2.9 presents a circuit that can be simulated with a structural architecture (case B), but for some input states it produces entanglement, and therefore can only be simulated by functional (behavioral) architectures (case A).

The matrix representation of quantum states and unitary operators is adopted; therefore the quantum states are type array (of complex) signals [6]. 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 (see Figure 2.10 for the appropriate data structure example - described in VHDL.)

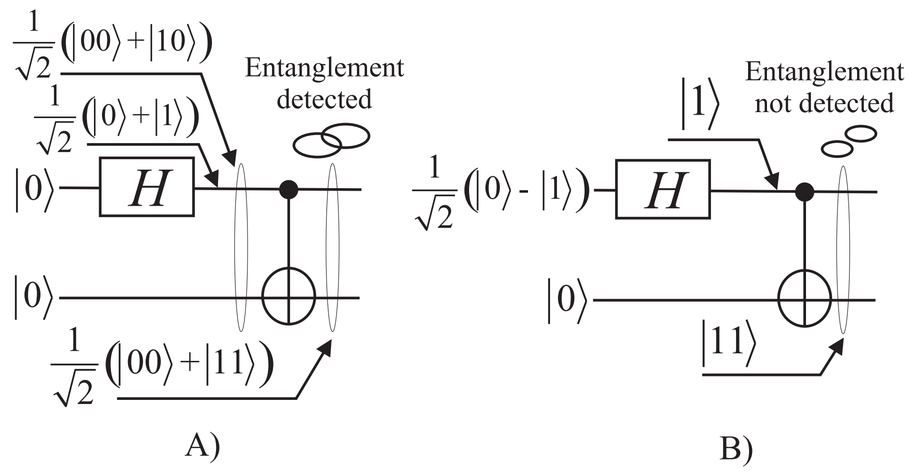

$\frac{1}{\sqrt{2}}(|00\rangle+|10\rangle)$
$\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$
Entanglement detected
$|0\rangle$
H
$|0\rangle$
$\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle)$
A)
$|1\rangle$
Entanglement not detected
$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
H
$|0\rangle$
$|11\rangle$
B)

Figure 2.9: Entanglement example.

With the data structure from Figure 2.10 and the above considerations, we are able to describe the circuit from Figure 2.9 with both structural and behavioral architectures. The behavioral architecture (see Figure 2.11, for architecture "functional") has a group of 4 variable

```
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 2.10: VHDL data set example.

assignments, motivated by the fact that the overall transformation produced by the circuit is characterized with the resulted matrix from Equation 2.18. The effect of the Hadamard gate over the overall input state is given by:

$$H \otimes I = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}. \tag{2.17}$$

Applying the XOR gate over the 2 qubits will have the following effect:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} (H \otimes I) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}. \tag{2.18}$$

When structural description is possible, the circuit can be reduced to the form given by Figure 2.12(A), with $U_0, \ldots U_{n-1}$ being 1-qubit unitary transformations, and $q_0, \ldots q_{n-1}$ individual qubits. For the Figure 2.9(B) case, the structural description is possible, because the circuit can be reduced as shown by Figure 2.12(B) (with $q_i = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, the target input $q_{T_i} = |0\rangle$, the target output and source qubits $q_{T_o} = q_S = |1\rangle$). This is, in fact, the motivation for the architecture "structural" of entity "circ-ex" (see Figure 2.11).

Summarizing, the simulation methodology (as developed until this point) is presented in Figure 2.13. After each quantum register i, an entanglement analysis is produced, generating k_i non-entangled qubit groups. This information is passed to $QReg_i$ and the quantum network that has outputted this state ($Qnet_i$) in order to select the appropriate structural architecture (if such would be the case).

```
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 2.11: Relevant pieces of VHDL code.

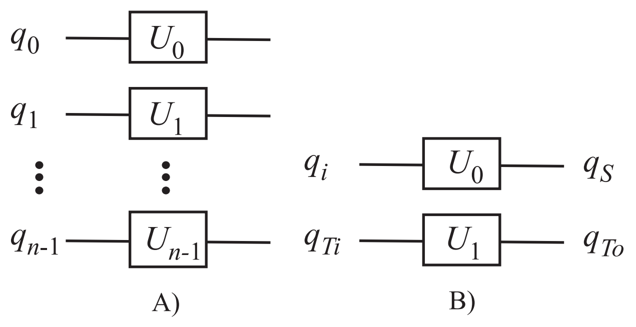


Figure 2.12: Non-entanglement circuit reduction.

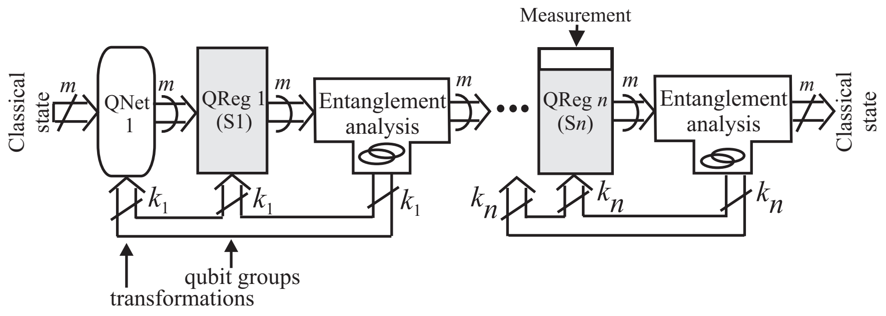


Figure 2.13: HDL-based, entanglement-aware, quantum circuit simulation model.

2.4 Methodology effectiveness

Our simulation methodology's effectiveness is affected by entanglement situations. This kind of study was already performed numerically for Shor's algorithm [72], but we are considering it for our HDL framework. For assessing the entanglement role in our HDL-based simulation method, we consider two definitions.

Definition 1 (Complete entanglement): A n-qubit quantum state is completely entangled (i.e. an entanglement that includes all the qubits) iff it cannot be expressed as a tensor product of a 1-qubit state and a $(n-1)$-qubit state.

Definition 2 (Total entanglement): A n-qubit state is totally entangled iff there is no tensor product of two k and l-qubit states (with arbitrary $2 \leq (k+l) = n$) to express it.

Definition 2 is particularly useful, because there are cases of complete entanglement where qubits are totally entangled inside well defined groups, but the groups as wholes are not completely entangled between themselves. Figure 2.14 presents such an example, where the overall state is not completely entangled, and the overall state of all qubits except q_3 is not totally entangled. A situation where the entanglement is complete but not total is still advantageous for our simulation approach and a structural description is possible (though not at qubit level).

2.4.1 Automated extraction of non-entangled information

Grover algorithm case study

In order to draw any conclusion on the opportunity of applying our simulation methodology to Grover's algorithm, we must analyze the entanglement in Figure 2.5, at the indicated points: (1), (2), (3), and (4). It is obvious that 1-qubit gates will not produce entanglement. Therefore, we have 4 possible distinct situations while simulating Grover's algorithm:

A) No entanglement is encountered;

B) Entanglement appears at point (2) and it is cancelled at point (4);

C) Entanglement appears at point (4);

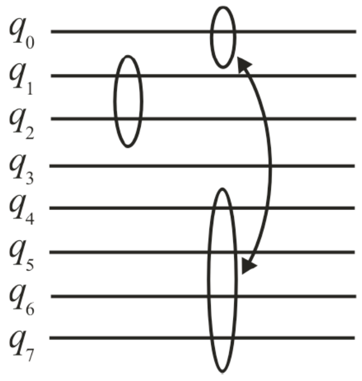


Figure 2.14: Example of groups of entangled qubits. Qubits q_0, q_4, q_5, q_6, q_7 are in the first group, q_1, q_2 in the second, while q_3 is single.

D) Entanglement appears at point (2) and it would not be cancelled.

From our simulation methodology perspective, these situations are ordered from A) - the best - to D) - the worst. Due to the fact that the state first entering U_O is of the form

$$\psi_{(1)} = \frac{1}{2^{\frac{n}{2}}} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \tag{2.19}$$

and the state exiting U_O is

$$\psi_{(2)} = \frac{1}{2^{\frac{n}{2}}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} \tag{2.20}$$

with $a_i = \pm 1$ and $i = 0..2^n - 1$, we could establish if $\psi_{(2)}$ is entangled or not by trying to write $\begin{bmatrix} a_0 & a_1 & \ldots & a_{2^n-1} \end{bmatrix}^\dagger$ as a tensor product of individual qubit states (i.e. $[2 \times 1]$ individual qubit matrixes.)

Lemma 1 (U_0 non-complete entanglement): The oracle is not dictating *complete entanglement* iff in the set $\{a_0, a_1, \ldots a_{2^n-1}\}$ of Equation 2.20 elements, all the couples (a_{2k}, a_{2k+1}) (with $k = 0..2^{n-1} - 1$) are either *constant* or *balanced*.

Proof: First we are considering all the possible 2×1 matrices containing ± 1 elements: $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\begin{bmatrix} -1 \\ -1 \end{bmatrix}$, $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$, and $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$. The first two are *constant* and the following ones are *balanced*. The matrix from Equation 2.20 represents an n-qubit state. Therefore, if we have an $(n-1)$-qubit state, adding one qubit is described by the following tensor product:

$$\frac{1}{2^{\frac{n-1}{2}}}\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{2^{n-1}-1} \end{bmatrix} \otimes \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \\ \vdots \\ a_{2^{n-1}-1} b_0 \\ a_{2^{n-1}-1} b_1 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{2^n-2} \\ c_{2^n-1} \end{bmatrix} \tag{2.21}$$

In Equation 2.21, the rightmost form of the product matrix will give the following set of couples:

$$S_c = \{\ (c_0, c_1), (c_2, c_3) \dots (c_{2^n-2}, c_{2^n-1})\} = \{\ a_0 (b_0, b_1), a_1 (b_0, b_1) \dots a_{2^{n-1}-1} (b_0, b_1)\} \tag{2.22}$$

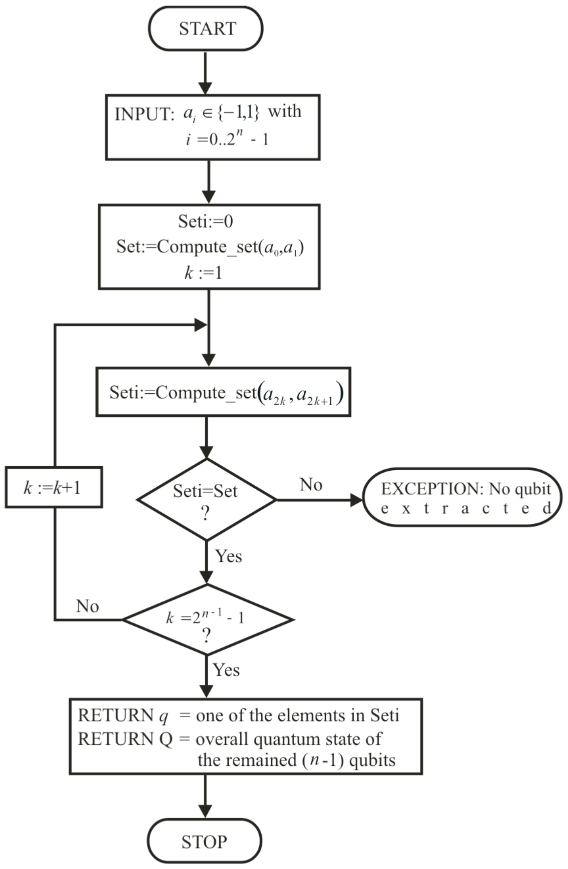

START
INPUT: $a_i \in \{-1,1\}$ with $i = 0..2^n - 1$
Seti:=0
Set:=Compute_set(a_0,a_1)
k :=1
Seti:=Compute_set(a_{2k}, a_{2k+1})
Seti=Set ?
No
EXCEPTION: No qubit extracted
Yes
k :=k+1
No
$k = 2^{n-1} - 1$?
Yes
RETURN q = one of the elements in Seti
RETURN Q = overall quantum state of the remained (n-1) qubits
STOP

Figure 2.15: Algorithm 1 described with a flowchart, where n is the number of qubits. Set and Seti are two variables indicating the set of allowed 1-qubit matrixes: '=0' for constant, and '=1' for balanced.

Thus, the tensor product from Equation 2.21 is possible iff the resulted elements could be coupled the way Equation 2.22 shows. Of course $b_j, c_l = \pm 1$ for $j = 0..1$ and $l = 0..2^n - 1$.This means that all the couples will have the form of 1-qubit state $\begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$: balanced or constant.

Due to Lemma 1, we have elaborated a simple algorithm that extracts one non-entangled qubit state, from a n-qubit overall state (given in the matrix form) dictated by Grover's algorithm oracle. From now on, we will call this algorithm "Algorithm 1" (see Figure 2.15). Using Algorithm 1, we are able to extract all the individual non-entangled qubits from the state dictated by the oracle. This new algorithm (Algorithm 2 from Figure 2.16) also returns the overall state of the qubits that cannot be extracted (Q).

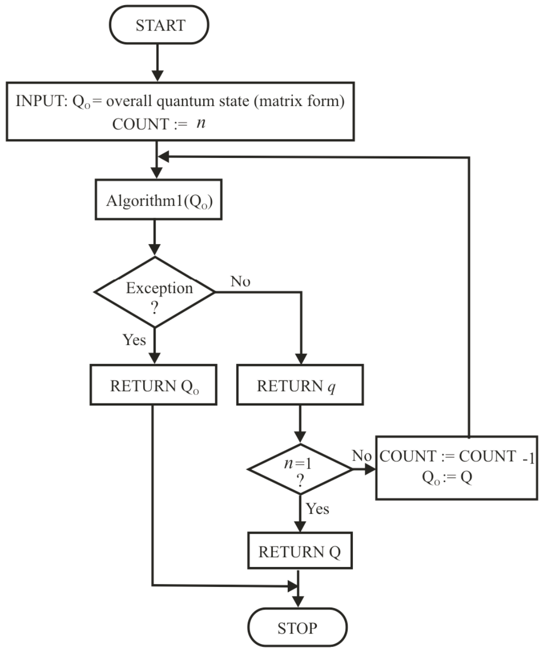


Figure 2.16: Algorithm 2 described with a flowchart, where n is the number of qubits. Here, q is the individual qubit - matrix form - state, and Q,Q_o the entangled overall states of the qubits that cannot be extracted.

For finding entangled qubit groups of arbitrary depth d, we need to modify Algorithm 1, which finds only 1-depth groups. This is obtained by redefining the sets of Algorithm 1 $S_{10} = \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\}$ and $S_{11} = \left\{ \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\}$ as all the possible couples of complementary matrixes. As an example for the new Algorithm 3, Equation 2.23 shows the possible couples that represent entangled 2-qubit states.

$$\begin{aligned}
S_{20} &= \left\{ \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \right\} \\
S_{21} &= \left\{ \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \right\} \\
S_{22} &= \left\{ \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \right\} \\
S_{23} &= \left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \right\}
\end{aligned} \qquad (2.23)$$

Lemma 2 (U_O non-total entanglement): The oracle is not dictating total entanglement iff in the set $\{a_0, a_1, \ldots a_{2^n-1}\}$ of Equation 2.20 elements there is a $2 \leq d < n$, $d \in \mathbb{N}$ so that all the 2^d-uples $\left(a_{k \cdot 2^d}, a_{k \cdot 2^d+1}, \ldots a_{(k+1)\cdot 2^d-1}\right)$ are in the same set of complementary matrixes S_{dx} $(x = 0..2^{2^d} - 1)$.

Proof: The demonstration of this lemma becomes obvious if we reconsider the demonstration of Lemma 1, by replacing the "couple" with the 2^d-uple, and using the above-described notion of complementary matrixes set.

The flowchart explaining Algorithm 3, for extracting d-depth entangled qubit groups, is presented in Figure 2.17.

Shor algorithm case study

In our entanglement-related case study, states $|\psi_1\rangle$ and $|\psi_2\rangle$ are not entangled: the first is a basis state (eigenvector), the later is obtained from the first by using only 1-qubit gates. Also, the registers involved in $|\psi_3\rangle$ and $|\psi_4\rangle$ (dictated by arithmetic circuits) is given in Equation 2.24, where n is the number of qubits, $\xi \in \mathbb{C}$, and $b_i \in \mathbb{B} = \{0, 1\}$.

$$|\psi_b\rangle = \xi \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{2^n-1} \end{bmatrix} \qquad (2.24)$$

Due to Equation 2.24, we have developed Lemma 3.

Lemma 3 (Entanglement in Shor's arithmetic circuits): The registers involved in states $|\psi_3\rangle$ and $|\psi_4\rangle$ are not totally entangled iff, in the set $\{b_0, b_1, \ldots b_{2^n-1}\}$ of Equation 2.24 elements, there is a $d \in \mathbb{N}$ so that all the 2^d-dimensional subsets $\left(b_{k \cdot 2^d}, b_{k \cdot 2^d+1}, \ldots, b_{(k+1)\cdot 2^d-1}\right)$

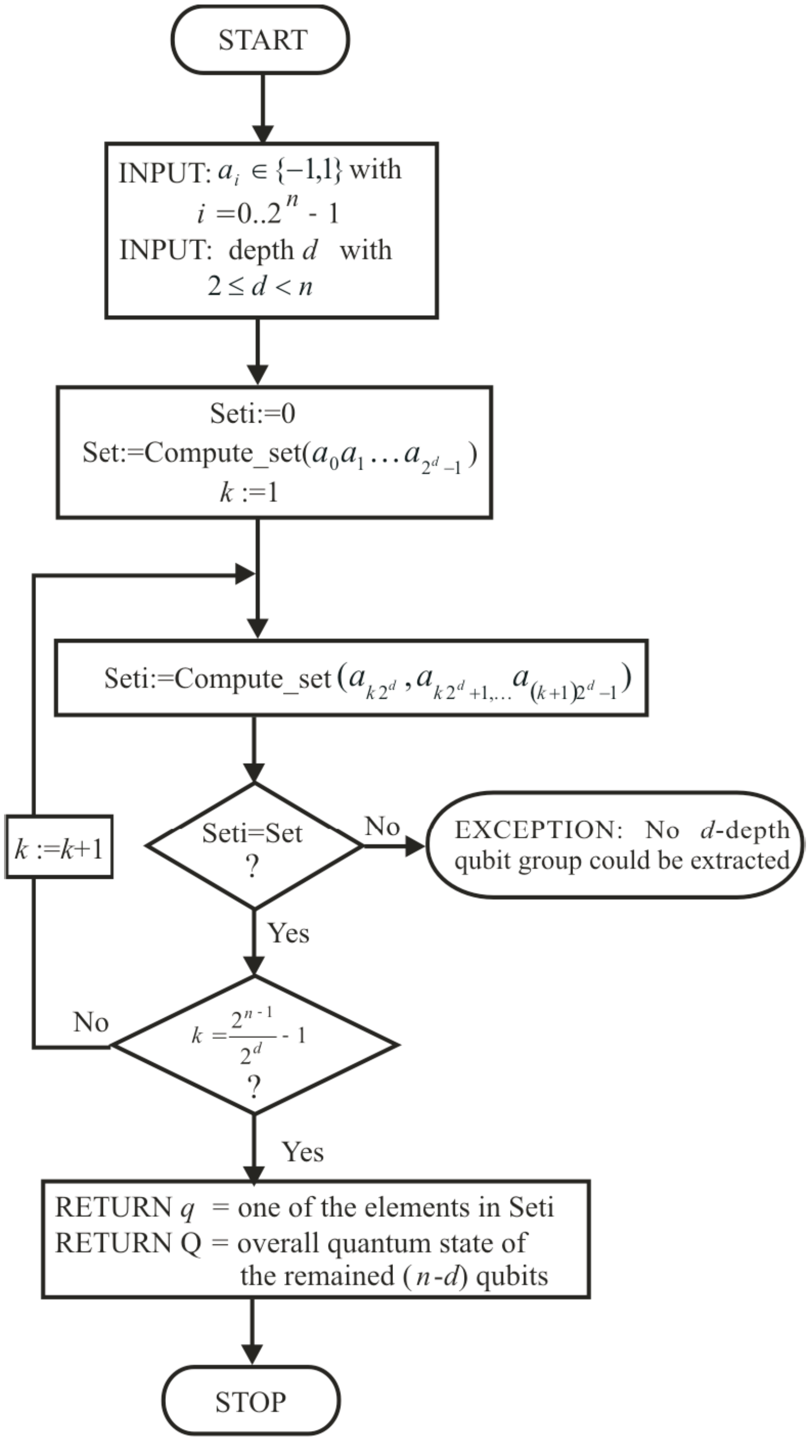

START
INPUT: $a_i \in \{-1,1\}$ with $i = 0..2^n - 1$
INPUT: depth d with $2 \le d < n$
Seti:=0
Set:=Compute_set($a_0 a_1 \ldots a_{2^d-1}$)
k :=1
Seti:=Compute_set ($a_{k2^d}, a_{k2^d+1,\ldots} a_{(k+1)2^d-1}$)
Seti=Set ?
No
EXCEPTION: No d-depth qubit group could be extracted
Yes
k :=k+1
No
$k = \frac{2^{n-1}}{2^d} - 1$?
Yes
RETURN q = one of the elements in Seti
RETURN Q = overall quantum state of the remained (n-d) qubits
STOP

Figure 2.17: Flowchart describing Algorithm 3 for extracting a d-depth entangled qubit group from the given state.

(with $k = 0..\left(2^{n-d} - 1\right)$) 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.

Proof: Non-total entanglement occurs when there is a subset of d qubits, which is not entangled with the rest. This is possible iff there are 2 corresponding matrixes so that:

$$|\psi_b\rangle = |\psi_e\rangle \otimes |\psi_e\rangle = \xi \begin{bmatrix} be_0 \\ be_1 \\ \vdots \\ be_{2^{n-d}-1} \end{bmatrix} \otimes \begin{bmatrix} bn_0 \\ bn_1 \\ \vdots \\ bn_{2^d-1} \end{bmatrix} \tag{2.25}$$

Given the fact that $be_i, bn_j \in \mathbb{B}$, be_i is either copying $|\psi_n\rangle$ elements or making all of them '0' in the tensor product, thus confirming this lemma's assertion. We developed an algorithm (Algorithm 4) for extracting d-qubit ($1 \leq d < n$) groups that are not entangled with the rest in the register (see Figure 2.18). 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). 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 2.18 then we have a non-total entanglement situation.

2.4.2 Non-entanglement probabilities

Finding and extracting forms of non-entangled information is an advantage for our HDL-based simulation method. But its effectiveness is given by the frequency of non-entanglement situations. Our simulation approach gives no advantage if the circuit could be described only by an entirely functional architecture (i.e. no structure could be specified). This means that we cannot draw any advantage if total entanglement is encountered.

We could also say that total entanglement appears only if Algorithm 3 and Algorithm 4 will give no expected answer ($EXCEPTION$ in Figure 2.17, or $EXIT$ in Figure 2.18).

Grover's algorithm probabilities

Unfortunately, the improvement obtained by our simulation methodology [105][106] is not always present. Furthermore, the probability of encountering an advantageous situation decreases exponentially with the number m of qubits in the register (see Figure 2.19). This is first expressed in Equation 2.26:

$$p1\left(m\right) = 2^{-2^m+1} \tag{2.26}$$

where the probability of being able to extract at least one entangled qubit ($p1\left(m\right)$) is defined by the number of matrix sets (2, balanced and constant), the number of matrixes/set (2), the number of couples (2^{m-1}), and the number of the initial matrix elements (2^m).

The probability of decomposing all the given quantum state in individual (non-entangled) qubits is even lower for a big m. This probability is expressed in Equation 2.27:

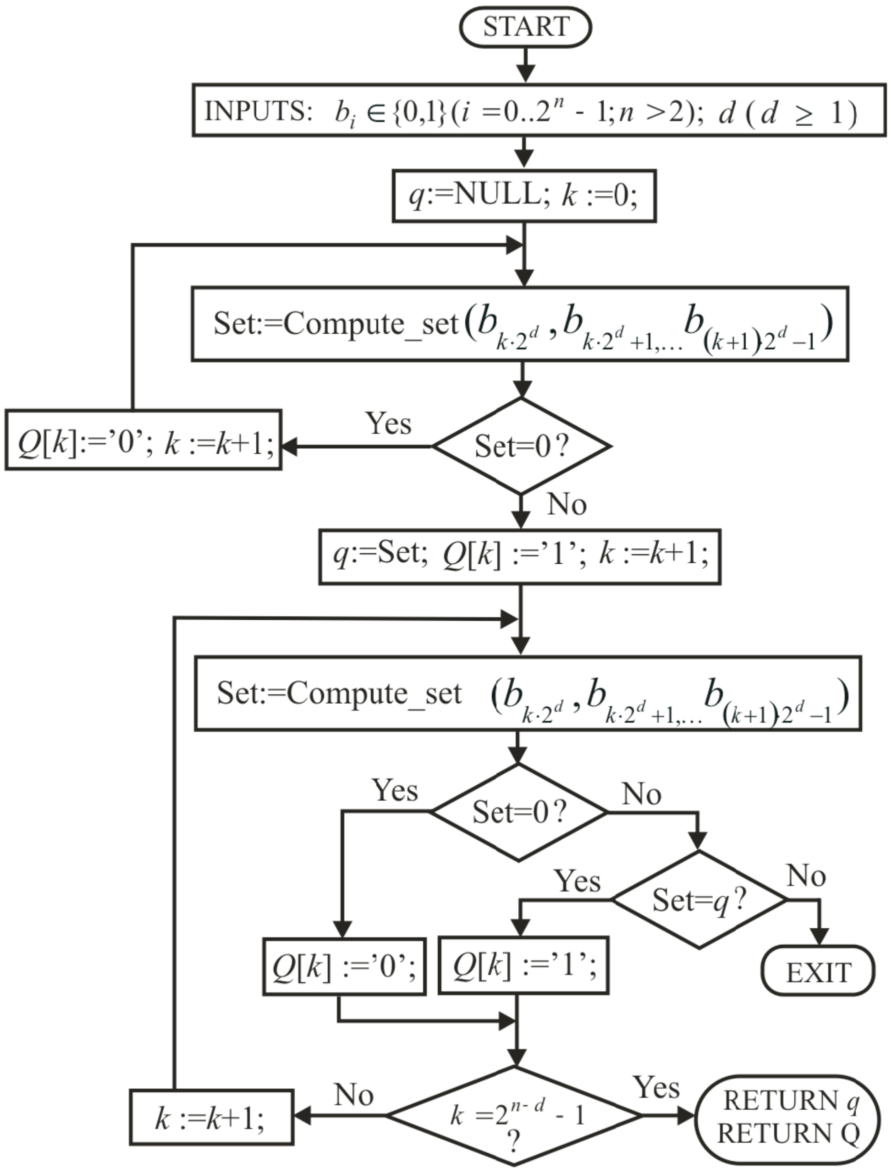

START
INPUTS: $b_i \in \{0,1\}(i = 0..2^n - 1; n > 2)$; d $(d \geq 1)$
q:=NULL; k :=0;
Set:=Compute_set $(b_{k \cdot 2^d}, b_{k \cdot 2^d+1,\ldots} b_{(k+1)2^d-1})$
Set=0 ?
Yes
$Q[k]$:='0'; k :=k+1;
No
q:=Set; $Q[k]$:='1'; k :=k+1;
Set:=Compute_set $(b_{k \cdot 2^d}, b_{k \cdot 2^d+1,\ldots} b_{(k+1)2^d-1})$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

Figure 2.18: Qubit group extraction Algorithm 4.

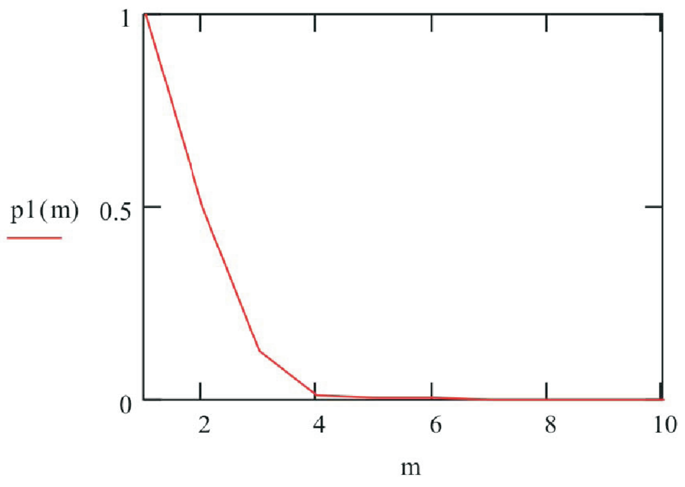


Figure 2.19: Probability $p1$ of extracting one non-entangled qubit; evolution with the number of qubits (m).

$$p2\,(m) = \prod_{i=1}^{m} \left(2^{-2^{m-1}+1}\right) \tag{2.27}$$

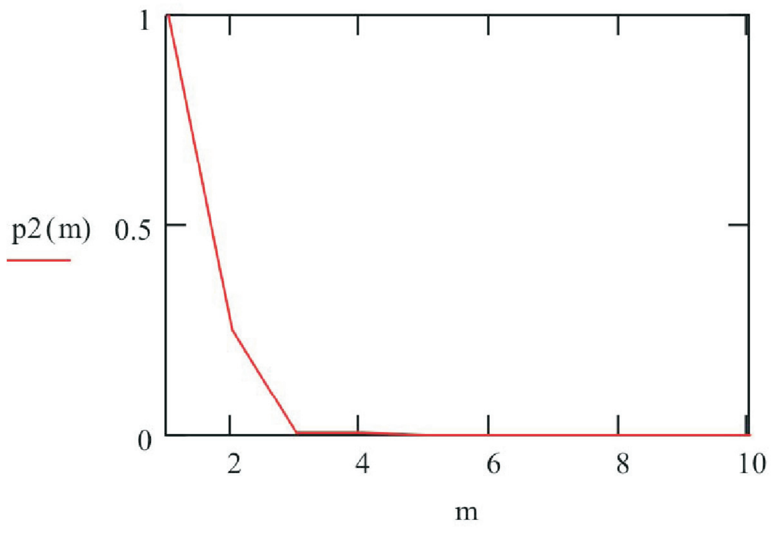


Figure 2.20: Probability $p2$ of extracting all the qubits as non-entangled; evolution with the number of qubits (m).

Figures 2.19 and 2.20 show little room for simulation improvement with our approach. But extracting individual non-entangled qubits is not the only possibility of separating non-entangled information from the given quantum state.

If searching for a d-depth non-entangled group in the given state, then the total number of such possibilities is $2^{2^d+2^{m-d}-1}$. Thus, when searching d-depth groups for all d $(1 \leq d < m)$, the probability of finding one is:

$$p3\left(m\right) = \frac{\sum_{d=1}^{m-1} 2^{2^d + 2^{m-d} - 1}}{2^{2^m}} \tag{2.28}$$

The graphic representation of $p3\left(m\right)$ is shown in Figure 2.21; the interpretation is that our simulation methodology will present some improvement, although it will decrease exponentially with m. Nevertheless, these probabilities are "blind" measures for effectiveness. Researchers have shown that, when running algorithms in practical cases, *quantum information is organizing itself so that total entanglement is not always present.* Moreover, special coding could be employed so that we avoid entangled representations [114].

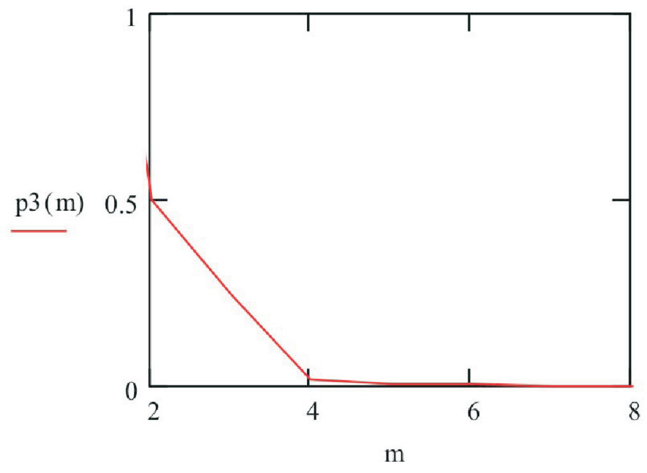


Figure 2.21: Probability $p3$ of extracting one of the possible d-depth entangled qubit groups, for $2 \leq d < m$.

Shor's algorithm probabilities

Algorithm 4 is extracting non-entangled information from register states involved in Shor's algorithm arithmetic operations. Related with the results from the previous section, we are interested in finding the probabilities of non-complete (Pnc) and non-total entangled (Pnt) states. Pnc is the probability of finding at least one non-entangled qubit, whereas Pnt is the probability of finding at least one non-entangled qubit group (depth d, $2 \leq d < m$). The result is given in Equations 2.29, 2.30, and Figure 2.22:

$$Pnc\left(m\right) = \frac{3 \cdot 2^{2^m - 1} - 2}{2^{2^m}} \tag{2.29}$$

$$pnt\left(m\right) = \frac{\sum_{i=1}^{m-1} \left[\left(2^{2^i} - 1\right) \cdot 2^{2^{m-i}} - \left(2^{2^i} - 2\right)\right]}{2^{2^m}} \tag{2.30}$$

with $m \in \mathbb{N}$, $m > 2$. Unfortunately, these exponentially decreasing probabilities do measure the effectiveness of our approach in this particular case study.

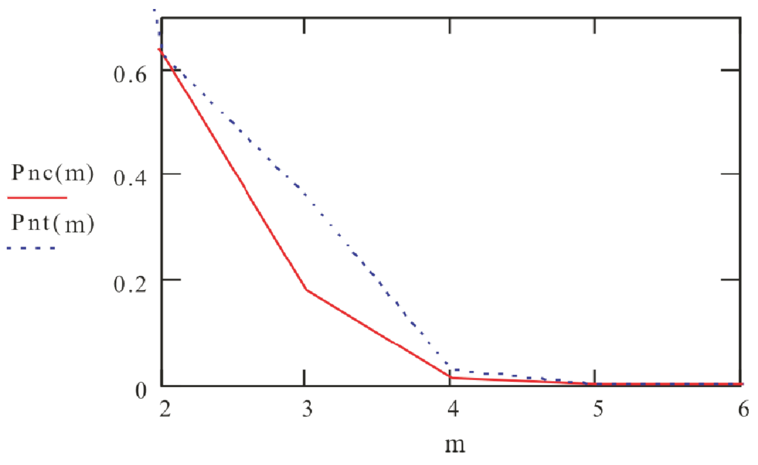


Figure 2.22: Probabilities of non-complete (Pnc) and non-total entangled (Pnt) states, with the number of qubits (m).

Arbitrary state qubit extraction

Using the results from Section 2.4.1 we will develop an algorithm for non-entangled qubit groups, applicable not just for a specific algorithm state, but for an arbitrary one. This algorithm is still exponential when extracting the non-entangled qubits, but is efficient when discarding a state as totally entangled.

The straightforward algorithm for non-entangled qubit group extraction is contained in Figure 2.23 from point marked as (*) downwards. Our approach is to extract relevant information so that some totally entangled states could be easily detected (from $START$ to point (*)).

When attempting to extract a d-depth qubit group from an n-qubit state, we split the 2^n state matrix into 2^{n-d} matrixes of the form $\left[a_{k2^d}, a_{k2^d+1}, \ldots a_{(k+1)2^d-1}\right]^\dagger$ with $k = 0..2^{n-d}-1$. We denote $\left[s\left(a_{k2^d}\right) s\left(a_{k2^d+1}\right) \ldots s\left(a_{(k+1)2^d-1}\right)\right]^\dagger$ as $M_{k,d}$, where

$$s(w) = \begin{cases} 0 & \text{if} \quad w = 0 \\ -1 & \text{if} \quad \operatorname{sign}(w) = -1 \quad \forall w \in \mathbb{C} \\ 1 & \text{if} \quad \operatorname{sign}(w) = +1 \end{cases} \tag{2.31}$$

In Figure 2.23, the function $Compute\_set$ is associating the integer index value of the set matrix $M_{k,d}$ is belonging to. The set, for matrix $M_{k,d}$ is $\{M_{k,d}, (-1) \times M_{k,d}, 0_d\}$ with 0_d being a $\left(2^d \times 1\right)$-size matrix with only 0 elements. Sets are labelled with indexes from 0 to $\frac{3^{2^d}-1}{2} - 1$. For example, when $d = 1$ we have the sets from Equation 2.32:

$$\begin{array}{ll} \text{Set}_0 = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} & \text{Set}_1 = \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} \\ \text{Set}_2 = \left\{ \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} & \text{Set}_3 = \left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} \end{array} \tag{2.32}$$

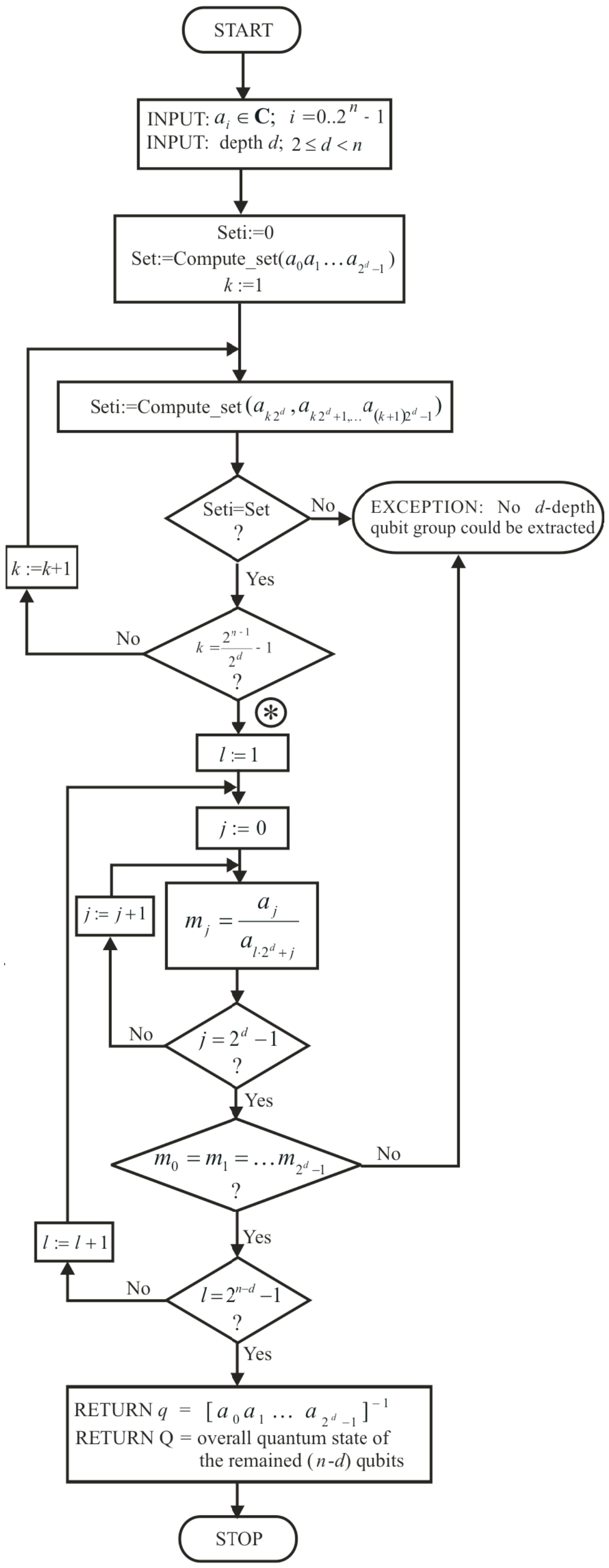

START
INPUT: $a_i \in \mathbf{C}$; $i = 0..2^n - 1$
INPUT: depth d; $2 \le d < n$
Seti:=0
Set:=Compute_set($a_0 a_1 \ldots a_{2^d-1}$)
$k := 1$
Seti:=Compute_set ($a_{k2^d}, a_{k2^d+1}, \ldots a_{(k+1)2^d-1}$)
Seti=Set ?
No
EXCEPTION: No d-depth qubit group could be extracted
Yes
$k = \frac{2^{n-1}}{2^d} - 1$?
No
$k := k+1$
$l := 1$
$j := 0$
$m_j = \frac{a_j}{a_{l \cdot 2^d + j}}$
$j := j+1$
$j = 2^d - 1$?
No
Yes
$m_0 = m_1 = \ldots m_{2^d-1}$?
No
Yes
$l = 2^{n-d} - 1$?
No
$l := l+1$
Yes
RETURN $q = [a_0 a_1 \ldots a_{2^d-1}]^{-1}$
RETURN Q = overall quantum state of the remained (n-d) qubits
STOP

Figure 2.23: Algorithm 5: non-entangled qubit group extraction from an arbitrary state.

2.5 Experimental results

All simulations from this thesis were pursued on a Windows XP^{TM}, PENTIUM^{TM} IV CPU 1.6GHz, 192MB RAM machine. For experimenting our HDL-based simulation approach, we have performed an experiment concerning the Deutsch-Jozsa algorithm [25]. Entanglement problems here are similar to Grovers algorithm case study, due to similar circuit structures. We have simulated the algorithm with 2 distinct circuits implementing a balanced function g: the *odd-even* (Figure 2.24) and the *parity circuits* (Figure 2.25). Simulation has taken into account several instances of the circuits: with a 4, 6, 8, 10, 12, 14, and 16-qubit query register. It was performed for all instances with both behavioral and structural architectures. That was possible because entanglement was not detected here.

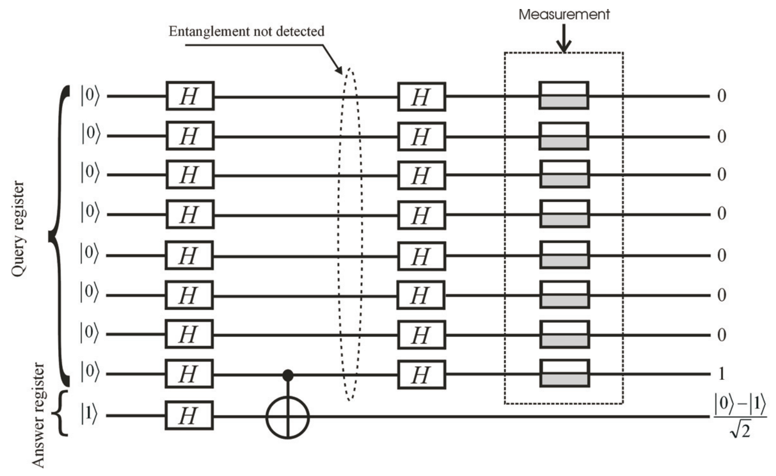


Figure 2.24: Deutsch-Jozsa with the *Odd-Even* circuit.

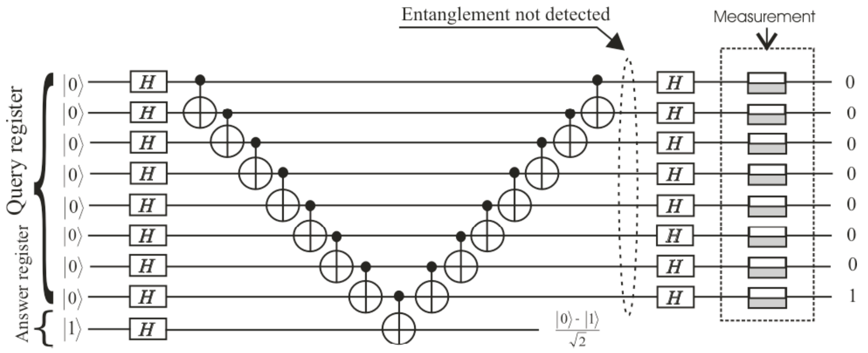


Figure 2.25: Deutsch-Jozsa implementation with the *Parity* circuit.

The time diagram for such a simulation is presented in Figure 2.26. The circuits for function g are presented in Figures 2.24 and 2.25. In Figure 2.26 signal *xin* encodes the input state of the left level of Hadamard gates (operating on the query register) and *ah* the corresponding output state (see Figure 2.25); signal *aht* encodes the state exiting the level of XOR gates; *xout* is the signal exiting the right level of Hadamard gates. Signal result is created by applying an OR gate on the outputs of the measuring devices from the rightmost part of Figure 2.25.

Simulation times are presented in Table 2.1 (n is the number of qubits, *Struct.* and *Behav.* stand for structural and behavioral), with the discrepancy between the structural and behavioral simulations being obvious because behavioral simulation times are rising exponentially with the number of qubits [105][106]. The discrepancy between behavioral and structural runtimes for the Deutsch-Jozsa algorithm is also presented in Figure 2.27.

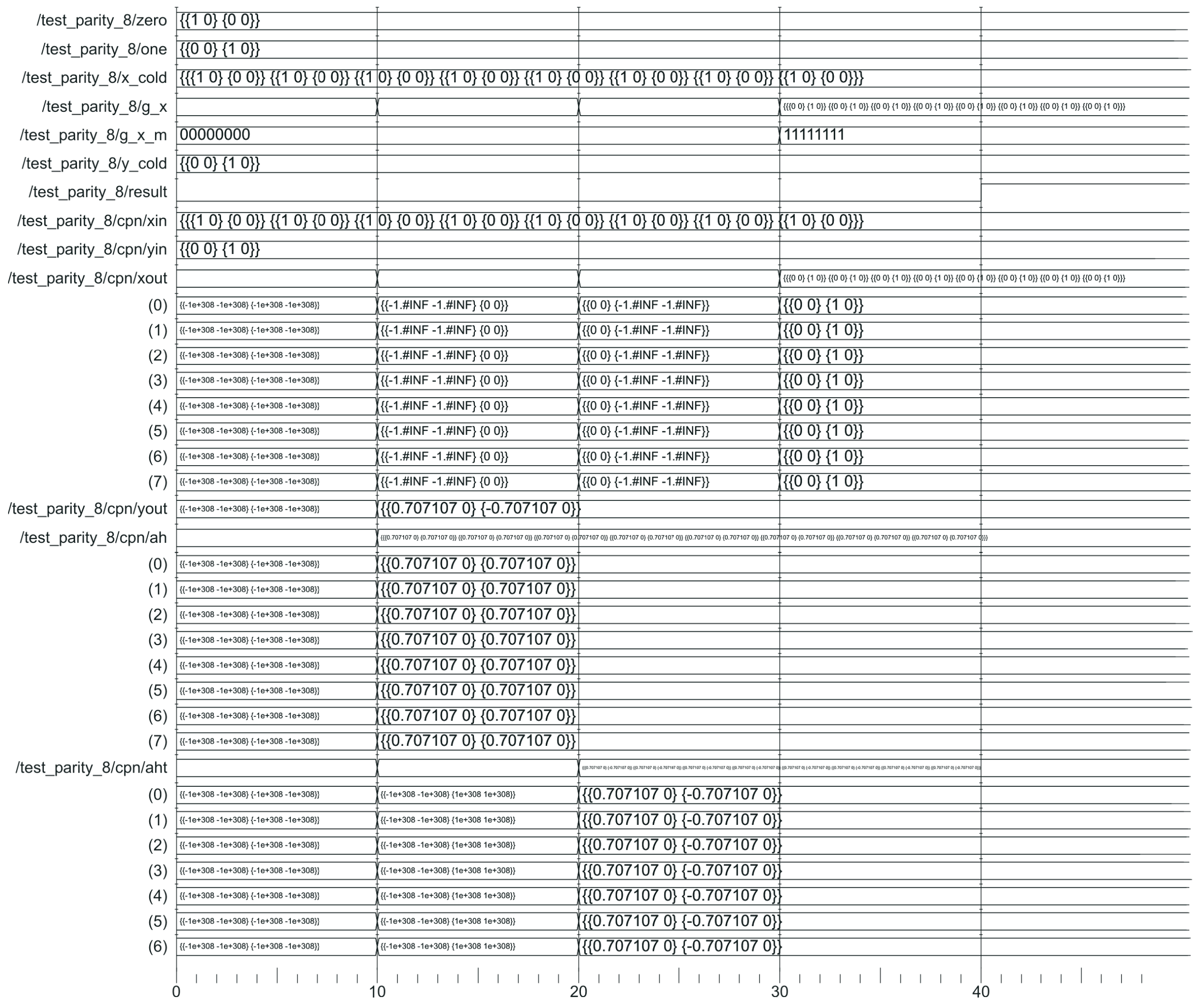


Figure 2.26: Time diagram with relevant signals of the Deutsch-Jozsa circuit simulation the *Parity* circuit, with an 8-qubit query register. Each gate is considered as operating with a 10 ns delay.

	odd-even		parity	
n	str.	beh.	str.	beh.
4	< 0.5 sec	0.5 sec	< 0.5 sec	0.5 sec
6	<0.5 sec	1 sec	< 0.5 sec	1.5 sec
8	< 0.5 sec	3 sec	< 0.5 sec	5.5 sec
10	< 0.5 sec	21.5 sec	0.5 sec	51.5 sec
12	< 0.5 sec	45min,8 sec	0.5 sec	1h,3min,7sec
14	< 0.5 sec	timed out	0.5 sec	timed out
16	0.5 sec	timed out	1 sec	timed out

Table 2.1: Deutsch-Jozsa algorithm simulation results.

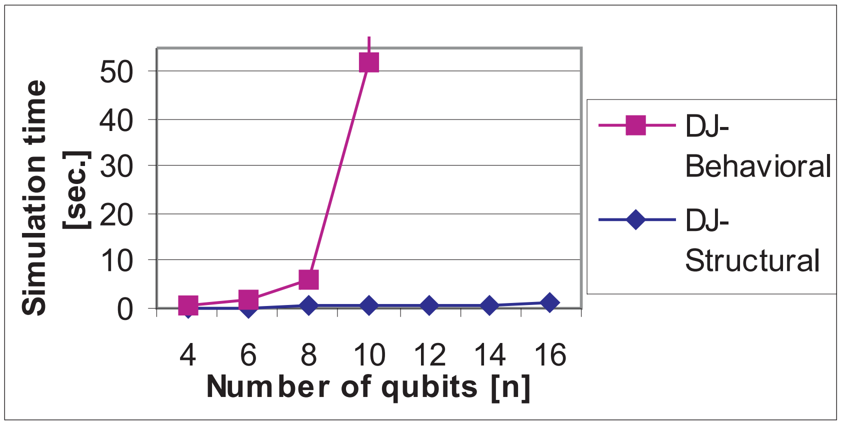


Figure 2.27: Deutsch-Jozsa simulation runtimes: structural Vs. behavioral.

Chapter 3

The Bubble Bit Technique

Although the conclusion of the previous chapter may not seem promising for our approach, we could still improve it; as shown in this thesis and in [106][107][108], at least for states appearing in Shor's arithmetical circuits and Grover's algorithm we have an encoding technique that creates the possibility of structural (i.e. polynomial) simulation.

Considering the arithmetic circuits involved in Shor's algorithm (with Grover's algorithm experiencing a similar situation [106][107]), 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 [62]), but instead finding transformations that leave 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 $[2 \times 1]$-size matrixes, incentive for structural (i.e. polynomial) simulation is provided.

3.1 Preliminaries

The bubble bit insertion technique is a quantum state and circuit encoding, which generates a new simulation model, under the form shown in Figure 3.1. 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 2.13 simulation model, with a high probability of being functional. The state outputted 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 3.1).

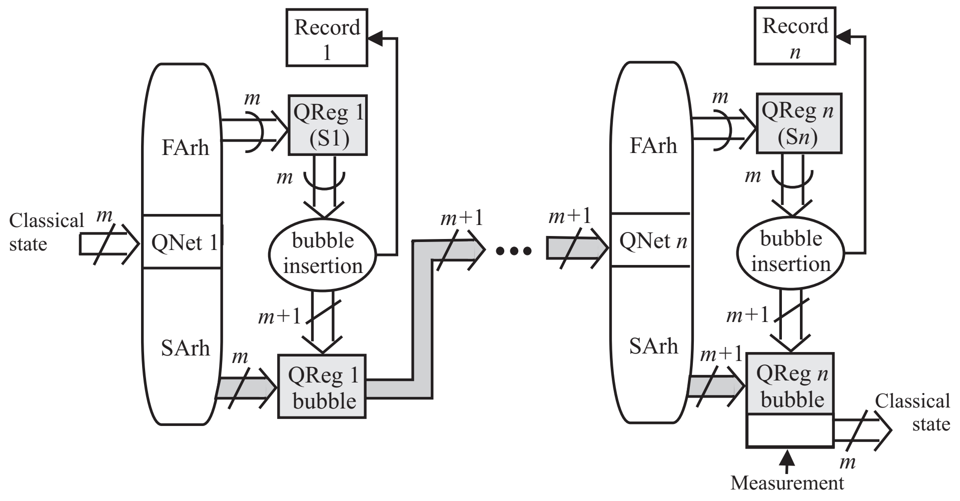


Figure 3.1: Quantum circuit simulation model, when the bubble bit technique is employed.

3.2 Shor's algorithm simulation

3.2.1 Bubble insertion algorithm

The procedure for bubble bit insertion works as follows: every couple (b_{2k}, b_{2k+1}) from the state vector (as considered in Equation 2.24) 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 3.2 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 3.2. 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 3.2 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.

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 0s. The next bubble steps will require erasure of 0s, 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).

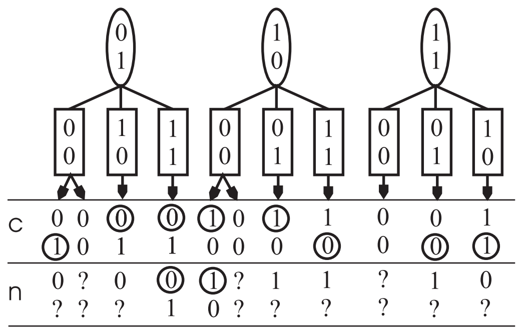


Figure 3.2: Bubble bit insertion technique.

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 $\mathcal{O}(n^2)$ records be produced.

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) [96][106]. 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 [106], 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 $\mathcal{O}(n^2)$, as experimented for Shor's algorithm arithmetic circuits and Grover algorithm circuit.

3.2.2 Example and experimental results

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 3.3(A). The way this add-cell could be rippled in order to build n-qubit adders is suggested in Figure 3.3(B). The simulation of the 1-qubit full adder will have to take into consideration the successive states from part (A) of Figure 3.3. The input state ($|\psi_1\rangle$) is not entangled, as shown in the following equation:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle. \tag{3.1}$$

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 3.2 to 3.5, the resulted state representations are identical, with only the corresponding records being different.

$$|\psi_2\rangle = \frac{1}{2\sqrt{2}} \begin{pmatrix} |0000\rangle + |0010\rangle + |0100\rangle + |0111\rangle \\ |1000\rangle + |1010\rangle + |1100\rangle + |1111\rangle \end{pmatrix} \overset{\text{bubble}}{\longrightarrow} \frac{1}{4}\hat{3}^{\otimes 4} \otimes \hat{1} + \boxed{rec_2} \tag{3.2}$$

$$|\psi_3\rangle = \frac{1}{2\sqrt{2}} \begin{pmatrix} |0000\rangle + |0010\rangle + |0110\rangle + |0101\rangle \\ |1000\rangle + |1010\rangle + |1110\rangle + |1101\rangle \end{pmatrix} \overset{\text{bubble}}{\longrightarrow} \frac{1}{4}\hat{3}^{\otimes 4} \otimes \hat{1} + \boxed{rec_3} \tag{3.3}$$

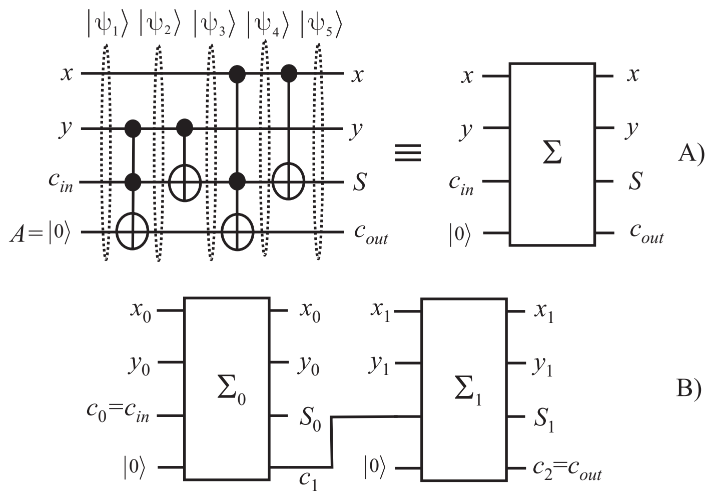


Figure 3.3: A) The 1-qubit full adder; B) obtaining a 2-qubit adder from 1-qubit Σ cells.

$$|\psi_4\rangle = \frac{1}{2\sqrt{2}} \left(\begin{array}{c} |0000\rangle + |0010\rangle + |0110\rangle + |0101\rangle \\ |1000\rangle + |1011\rangle + |1111\rangle + |1101\rangle \end{array} \right) \xrightarrow{\text{bubble}} \frac{1}{4}\hat{3}^{\otimes 4} \otimes \hat{1} + \boxed{rec_4} \tag{3.4}$$

$$|\psi_5\rangle = \frac{1}{2\sqrt{2}} \left(\begin{array}{c} |0000\rangle + |0010\rangle + |0110\rangle + |0101\rangle \\ |1010\rangle + |1001\rangle + |1101\rangle + |1111\rangle \end{array} \right) \xrightarrow{\text{bubble}} \frac{1}{4}\hat{3}^{\otimes 4} \otimes \hat{1} + \boxed{rec_5} \tag{3.5}$$

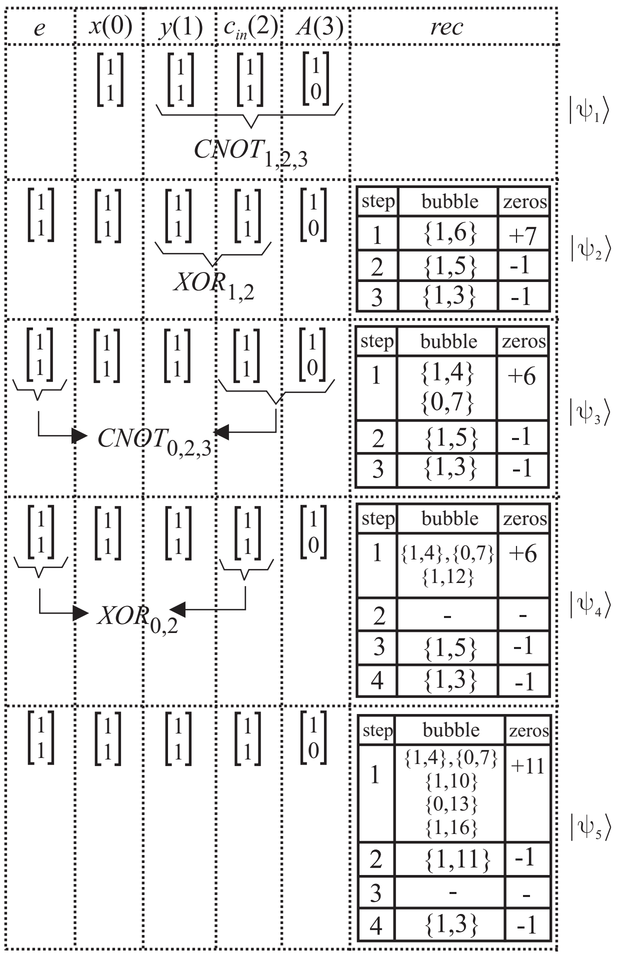
Figure 3.4 presents the step-by-step results of the procedure applied on the 1-qubit full adder, while Figure 3.5 contains the details regarding all the bubble steps performed for $|\psi_2\rangle$. Figure 3.4 has 6 columns and 5 rows; the columns correspond to the following: 1 record (*rec*), 4 qubits 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.

The results from Figure 3.4, as well as Equations from 3.2 to 3.5, indicate a structural network ('SArh' from Figure 3.1) of only identity qubit gates (characterized by I matrix). The new equivalent network was obtained the way section 2.3.3 and Figure 2.12 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 [113] of the full adder (see Figure 3.3). The experiment was pursued in the presence of total entanglement (therefore not a trivial simulation, as it is defined by [111]), thus requiring the bubble bit technique.

The results are presented in Table 3.1, 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 3.2 presents simulation times for the modulo-k (we considered $k = 2^{size-1}$) quan-



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 3.4: Bubble bit procedure results.

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 3.1: Quantum full adder simulation results.

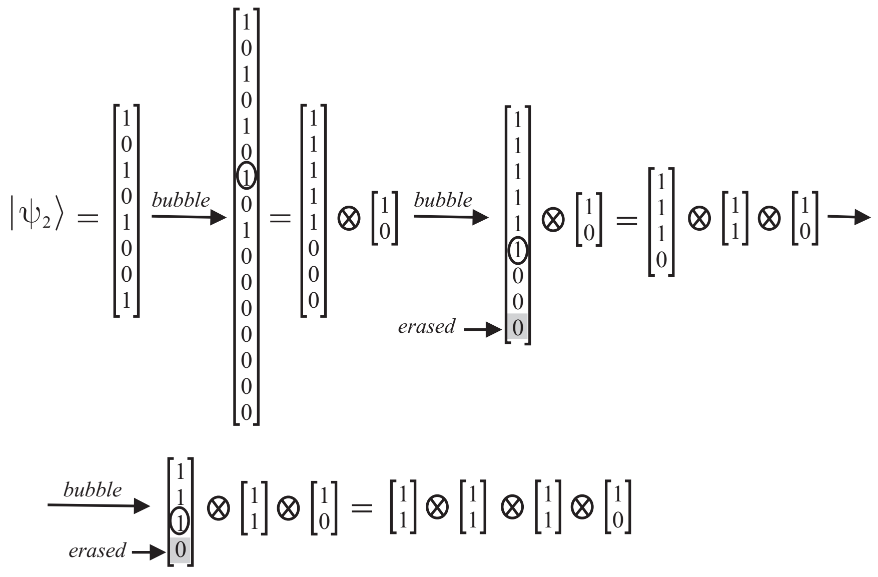


Figure 3.5: Bubble bit procedure example.

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 3.2: Experimental results for modulo adder and multiplier (simulation time).

tum adders and multipliers, as essential circuits used for Shor's algorithm implementations [89][91][113]. Because additional memory is required in order to store the records dictated by the bubble bit technique, Figure 3.6 presents the polynomial memory overhead for the simple quantum ripple adder, modulo adder, and modulo multiplier.

3.3 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 the states dictated by circuits from Grover's algorithm implementations is:

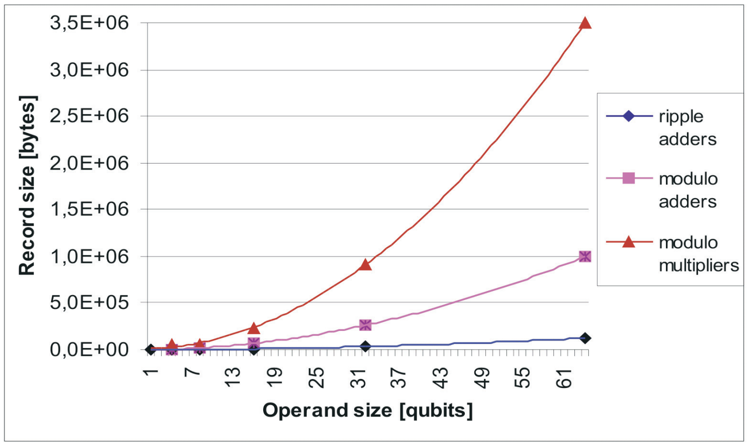


Figure 3.6: Extra memory requirements.

$$|\psi_{Grover}\rangle = \xi \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{2^n-1} \end{bmatrix} \tag{3.6}$$

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 [62] (see Figure 3.7), which performs quantum search on a 2-qubit register.

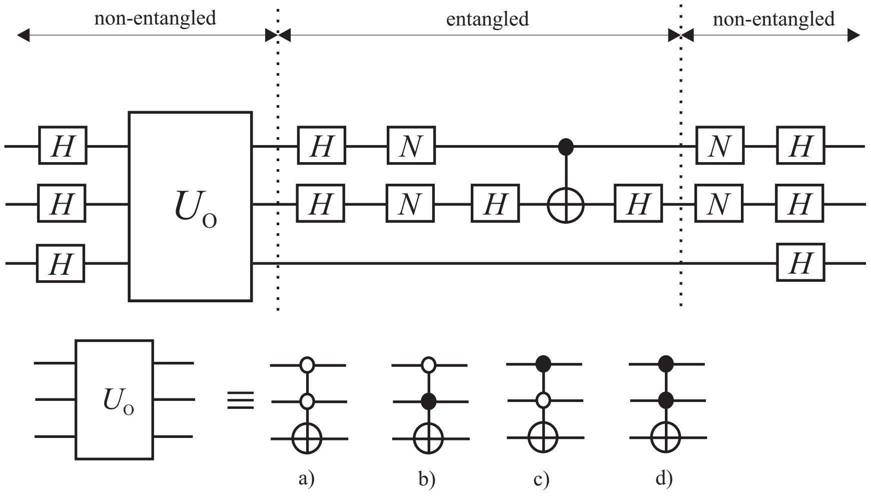


Figure 3.7: Grover algorithm circuit for a 2-qubit search register. The oracle U_O can be any of the a)-d) gates; also an entanglement analysis is provided by showing where it appears and where it is absent.

3.3.1 Bubble-bit insertion

The algorithm that transformes the state representation into a non-entangled one consists of repeating the bubble insertion algorithm until $(n+1)$ $[2\times 1]$-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 3.6;
   (a) memorize the first non-$\hat{0}$ couple;
   (b) insert bubbles according to rules in Figure 3.8 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 3.8, where 'x' stands for either '-1' or '1'.

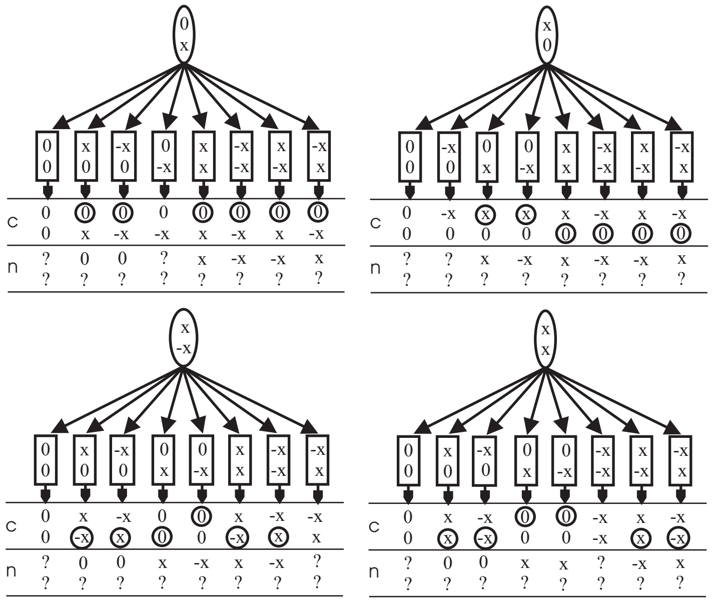


Figure 3.8: Bubble bit insertion rules for Grover algorithm states.

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 3.9. In this figure, the lower qubit value is known throughout the computation (it is shown in Figure 3.9) 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, $e = |0\rangle$.

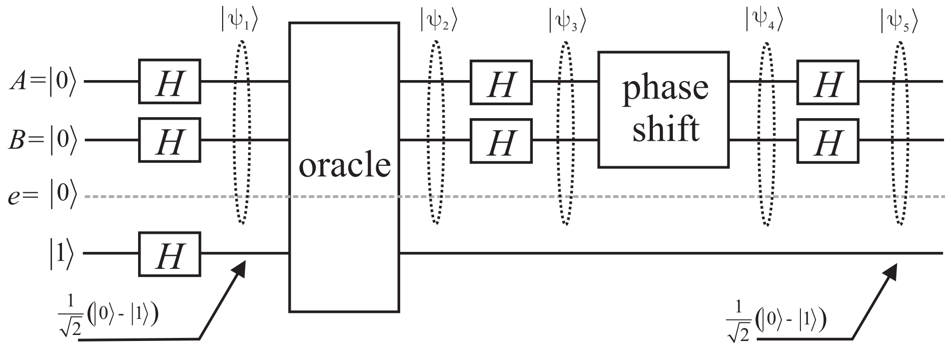
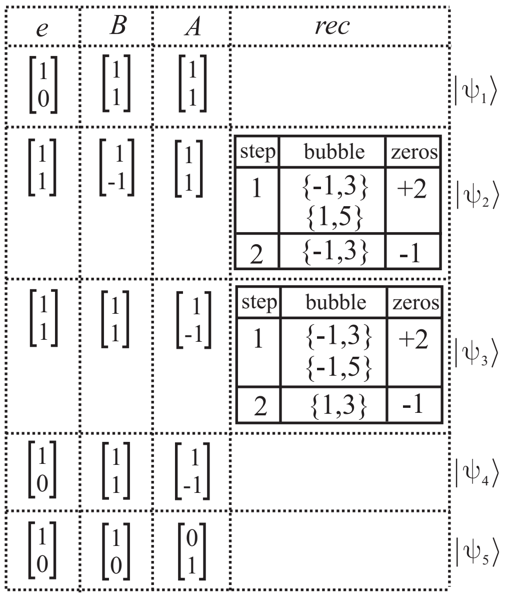


Figure 3.9: Relevant states for Grover algorithm simulation.

The result of applying the bubble bit technique on the $|\psi_1\rangle \ldots |\psi_5\rangle$ states is presented in Figure 3.10. In fact, as forecasted in the entanglement analysis from Figure 3.7, the bubble bit technique is only necessary for states $|\psi_2\rangle$ and $|\psi_3\rangle$.



e	B	A	rec: step	rec: bubble	rec: zeros	
$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$				$|\psi_1\rangle$
$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1\\-1\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	1	{-1,3} {1,5}	+2	$|\psi_2\rangle$
			2	{-1,3}	-1	
$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1\\-1\end{bmatrix}$	1	{-1,3} {-1,5}	+2	$|\psi_3\rangle$
			2	{1,3}	-1	
$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1\\-1\end{bmatrix}$				$|\psi_4\rangle$
$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\end{bmatrix}$				$|\psi_5\rangle$

Figure 3.10: 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 $\begin{bmatrix}1\\-1\end{bmatrix} = \hat{5}$:

$$|\psi_1\rangle = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) = \hat{0} \otimes \hat{3}^{\otimes 2} \tag{3.7}$$

$$|\psi_2\rangle = \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle + |11\rangle) = \hat{3} \otimes \hat{5} \otimes \hat{3} + \boxed{rec_1} \tag{3.8}$$

$$|\psi_3\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle + |11\rangle) = \hat{3}^{\otimes 2} \otimes \hat{5} + \boxed{rec_2} \tag{3.9}$$

$$|\psi_4\rangle = \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle - |11\rangle) = \hat{1} \otimes \hat{5} \otimes \hat{3} \tag{3.10}$$

$$|\psi_5\rangle = |10\rangle = \hat{1} \otimes \hat{2} \otimes \hat{1} \tag{3.11}$$

The way that the bubble insertion procedure works is presented in Figure 3.11.

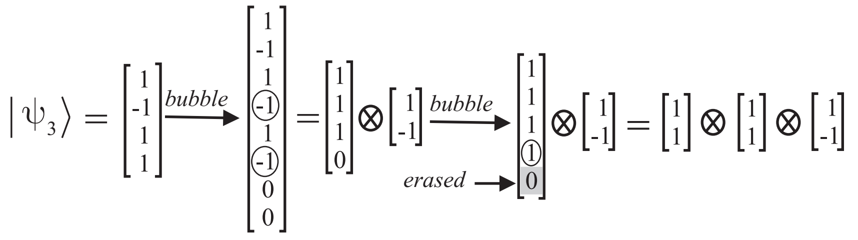


Figure 3.11: Bubble bit insertion procedure for $|\psi_3\rangle$.

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 3.12.

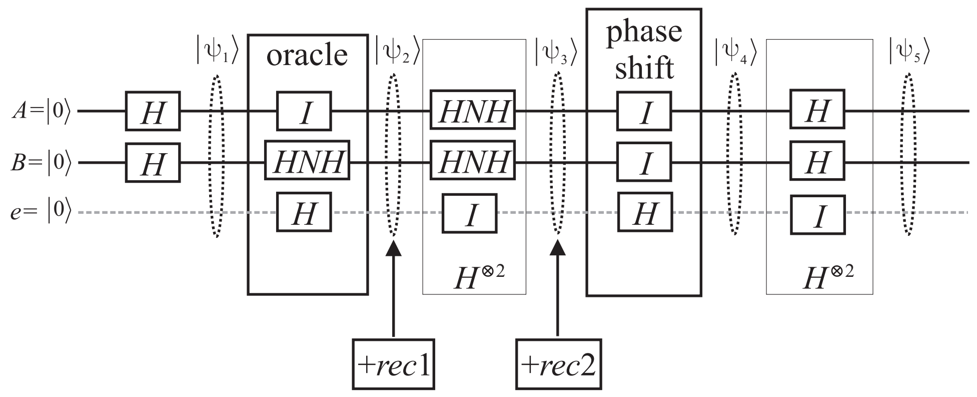


Figure 3.12: 2-qubit search Grover equivalent circuit, obtained with the bubble-bit technique in order to allow structural (i.e. polynomial) simulation.

In Figure 3.12 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 3.13 to the VHDL package from Figure 2.10. In Figure 3.14 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 3.12). 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 3.14).

```
-- 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 3.13: Data types required by bubble record representation.

Grover's algorithm was simulated for an Oracle that marks just one basis state, like [115]. Figures 3.15 and 3.16 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.

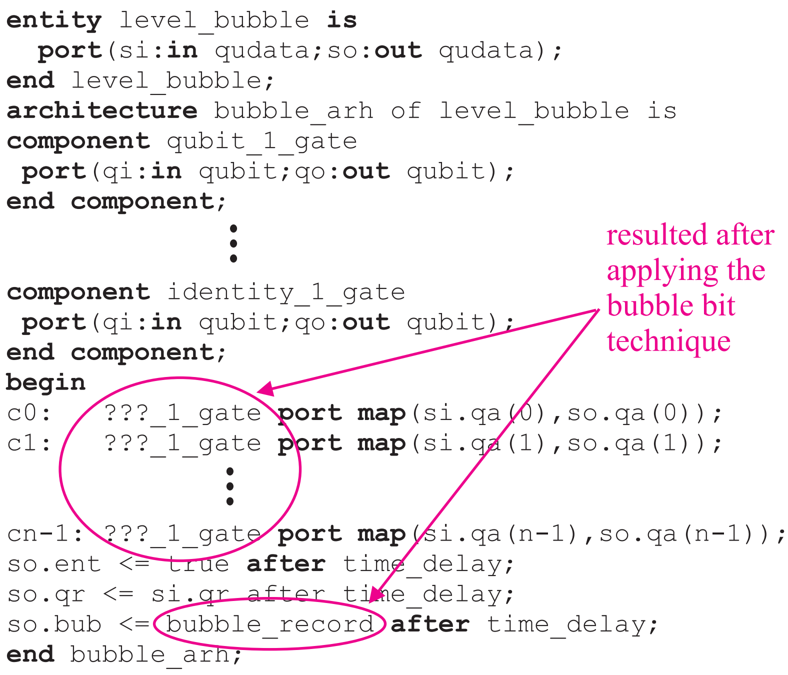
The runtime evolution with the number of qubits in the data register is presented in Figure 3.17. Also, the measured simulation times are compared here with the runtime complexity reported in [115], which is 0.22×1.44^n. The graphical representation shows substantial runtime improvement. Also, Figure 3.18 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.

3.4 Simulated fault injection

3.4.1 Preliminaries

In classical hardware, fault injection techniques are used for validation of *Fault Tolerance Algorithms and Mechanisms* (*FTAMs*) [92]. 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] [84][85].

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 [83], testing, and FTAM testing in the same environment [21][45][84] [85].



```
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 3.14: VHDL gate level implementation (entity-architecture pair) for bubble bit state transformation.

This section will focus on extending our already defined HDL-based quantum circuit simulation framework [96] [106], so that it can support fault injection that helps evaluating the dependability attributes [7] 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 [62][35], but the recently developed bubble bit technique [107] can also be adapted to error injection, so that the consequences of such this methodology are extended in order to approach fault tolerance assessment [108].

This section 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.

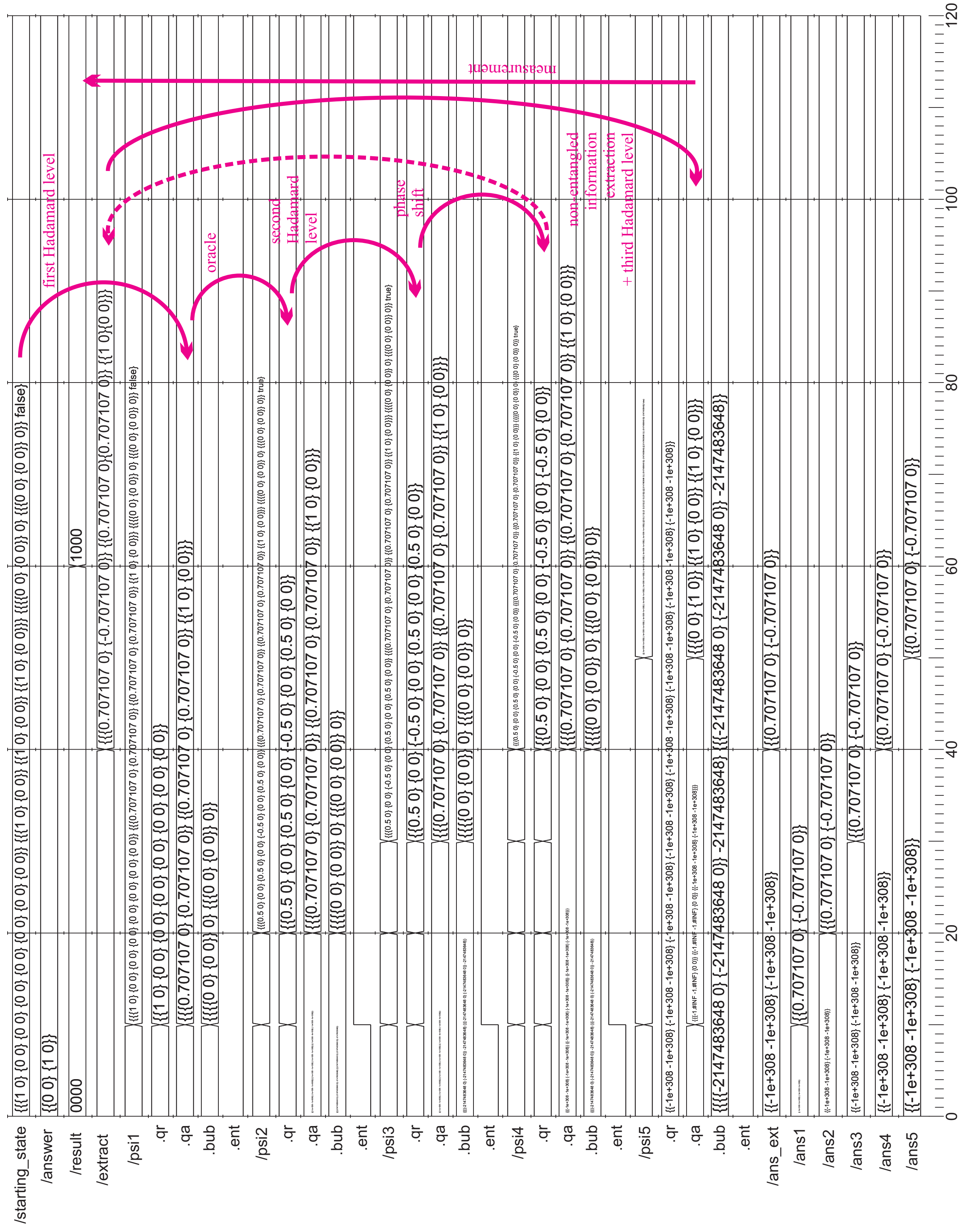


Figure 3.15: Time diagram resulted from VHDL simulation of Grover's algorithm, without the bubble bit technique.

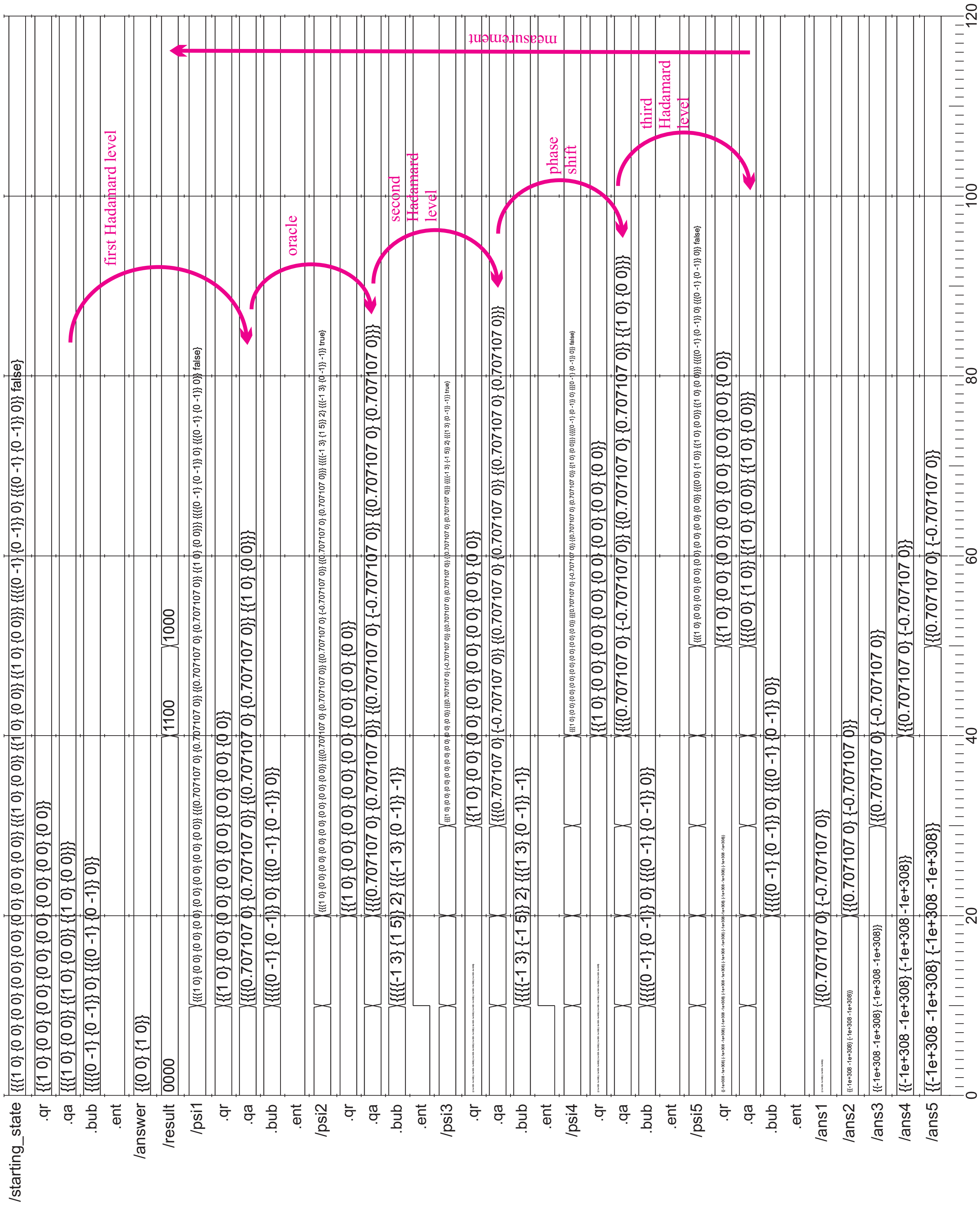


Figure 3.16: Time diagram resulted from VHDL simulation of Grover's algorithm, with the bubble bit technique.

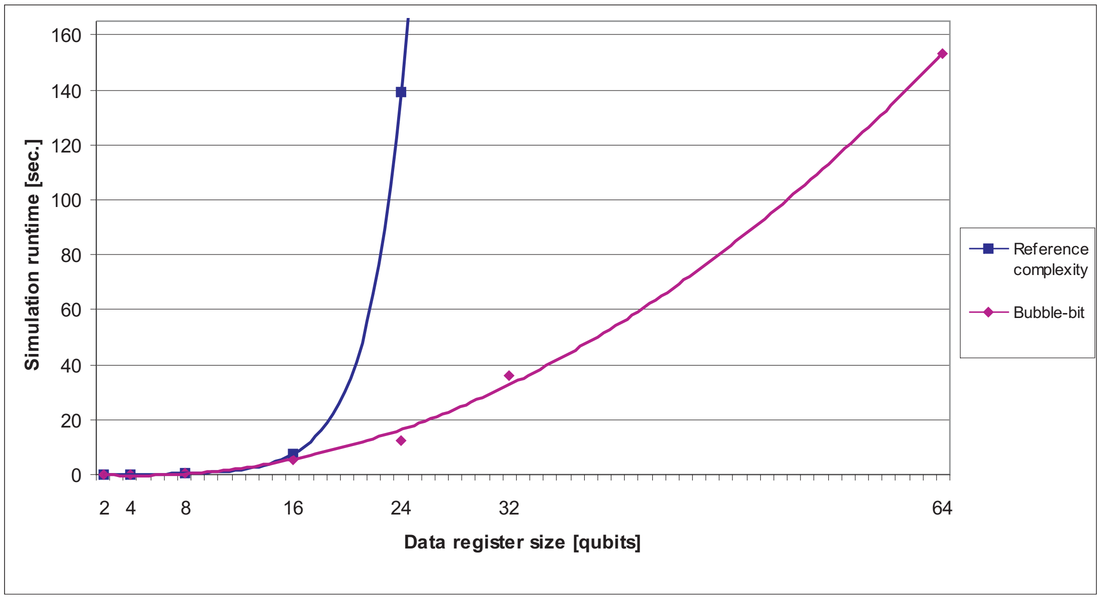


Figure 3.17: HDL bubble bit runtime results for Grover algorithm simulation, compared with the reference complexity.

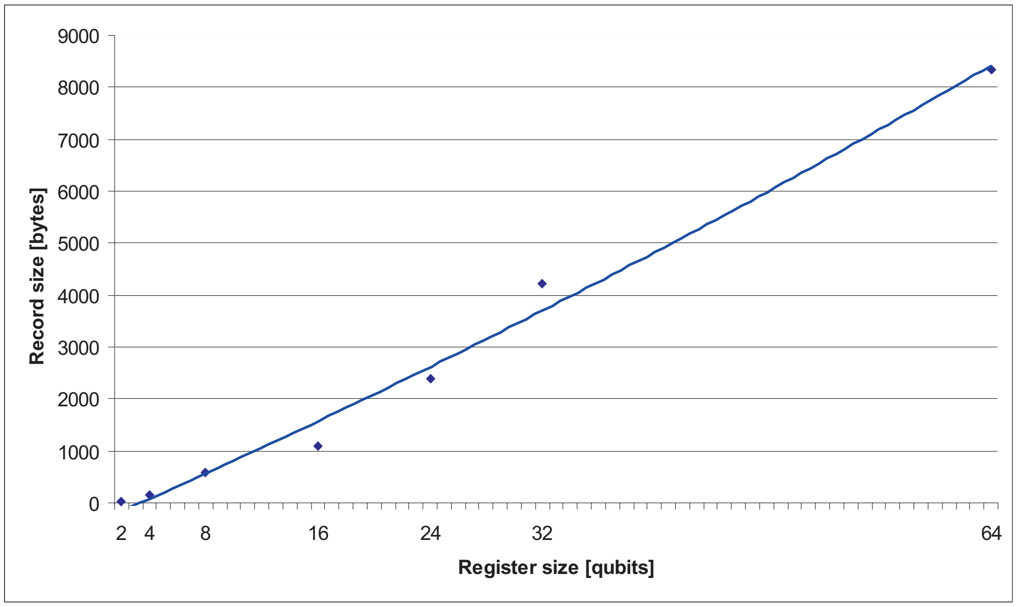


Figure 3.18: Memory overhead dictated by bubble records for Grover algorithm simulation. A trendline is added to the sample data, showing polynomial growth.

3.4.2 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 [106] [107][114][115].

This report will describe the guidelines of a bigger software project that fosters simulated fault injection techniques in quantum circuits; the project is called *QUERIST*. In this description, only the adaptation issues will be stressed, 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 3.19. 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 the theory of fault tolerant quantum computation [76], 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 errors). References [105][106][107] 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. 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.

3.5 Specific problems in the quantum environment

This section explains how to perform simulated quantum fault injection, within the HDL bubble bit [106] [107] framework, by following the rules given by quantum error model theory [76].

Implementing fault injection according to the constraints means that the bubble bit simulation model from Figure 3.1, has to be modified as Figure 3.20 shows.

Figure 3.20 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

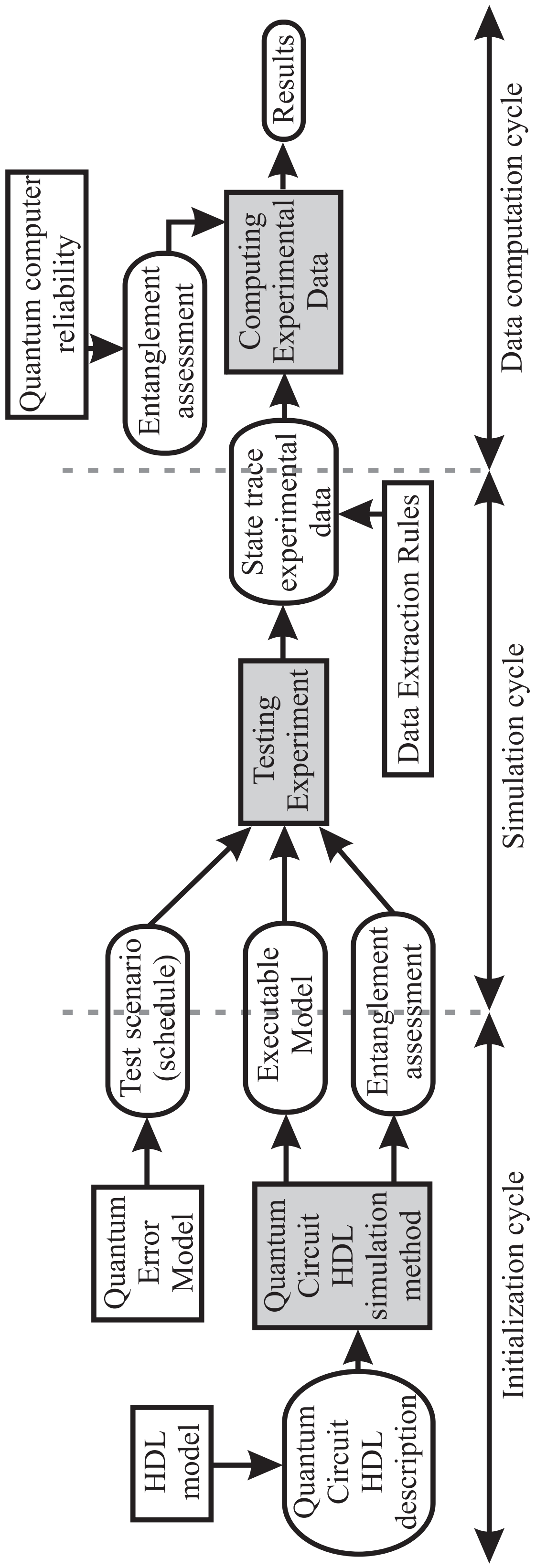


Figure 3.19: An overview of the QUERIST project.

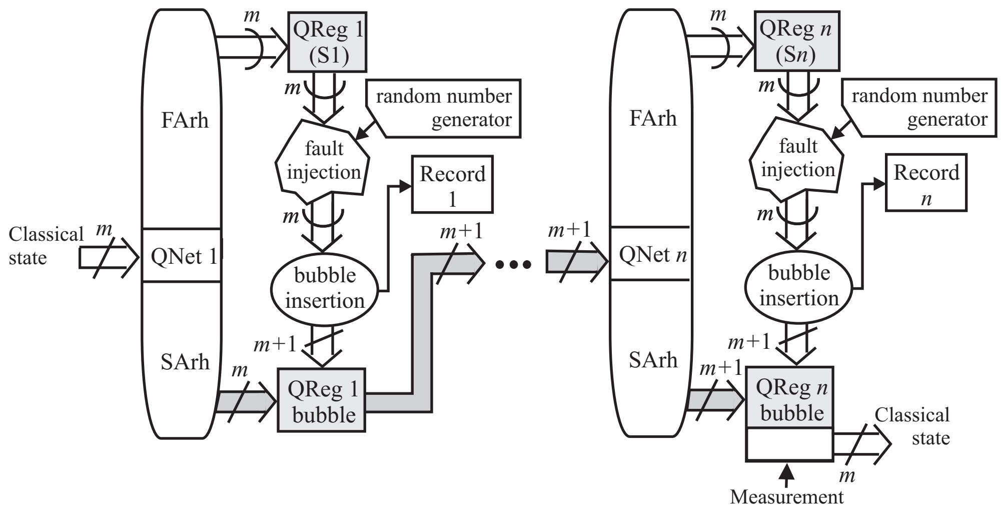


Figure 3.20: The bubble bit HDL simulation model, when fault injection is applied according to the error and fault occurence models presented in [76].

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.

3.5.1 Setup phase

Injecting a fault, in our simulation framework [105][106][107][108], consists of accordingly modifying the quantum state matrix:

$$|\psi_S\rangle = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{2^n-1} \end{bmatrix}. \tag{3.12}$$

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 the building of an exchange function that operates on blocks of matrix elements:

$$Exchange\left[(u_0, \ldots u_{w-1}), (v_0, \ldots v_{w-1})\right] \Leftrightarrow u_i \leftrightarrow v_i \text{ for every } i = \overline{0, w-1}. \tag{3.13}$$

Suppose we have a quantum state on n qubits, $|q_{n-1} \ldots q_2 q_1\rangle$, 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 $2^{n-k}$
    $Exchange\left[\left(a_{i\cdot 2^k}, \ldots a_{(i+1)\cdot 2^k-1}\right), \left(a_{(i+1)\cdot 2^k}, \ldots a_{(i+2)\cdot 2^k-1}\right)\right]$
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 [75][76] 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 [76].

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:

$$r_2 = \begin{cases} 0 \leq r_2 \leq \frac{1}{3} & \text{we have a bit-flip} \\ \frac{1}{3} < r_2 \leq \frac{2}{3} & \text{we have a phase-shift} \\ \frac{2}{3} < r_2 \leq 1 & \text{we have both bit-flip and phase-shift} \end{cases} \tag{3.14}$$

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: the 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 3.21 presents these cases (a) and b) respectively), which are considered by taking into consideration the quantum fault tolerance problems described by Preskill [75][76].

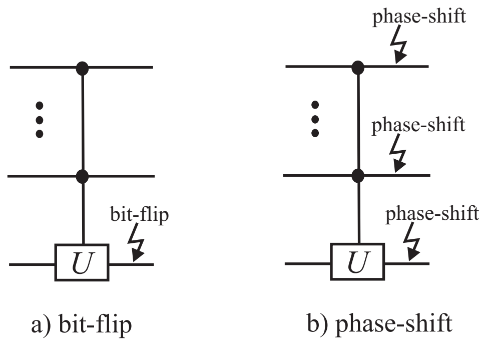


Figure 3.21: The effect of faulty gate operation on the processed qubits: a) gate bit-flip fault, b) gate phase-shift fault.

3.5.2 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 $|xyz\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$.

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 3.22, 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:

$$|p0\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|0\rangle_{s_2}+\boxed{rec_0} \tag{3.15}$$

$$|p1\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|0\rangle_{s_2}+\boxed{rec_1} \tag{3.16}$$

$$|p2\rangle \equiv |p1\rangle \tag{3.17}$$

$$|p3\rangle \equiv |p0\rangle \tag{3.18}$$

$$|p4\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|0\rangle_{s_2}+\boxed{rec_2} \tag{3.19}$$

$$|p5\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|1\rangle_{s_2}+\boxed{rec_2} \tag{3.20}$$

$$|p6\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|1\rangle_{s_2}+\boxed{rec_0} \tag{3.21}$$

$$|p7\rangle = \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}\right)\otimes|0\rangle_{s_1}\otimes|1\rangle_{s_2}+\boxed{rec_3} \tag{3.22}$$

The corresponding records are presented in Figure 3.23.

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

3.5.3 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:

$$s_i = |qb_0\rangle \otimes |qb_1\rangle \otimes \ldots |qb_{k_i}\rangle + \boxed{rec_i}. \tag{3.23}$$

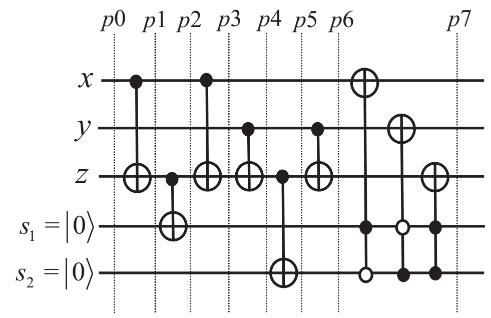


Figure 3.22: Circuit for singular bit-flip error correction.

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 3.23: Bubble records produced by simulating error correction with the circuit from Figure 3.22.

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:

$$s_i^* = |qb_0^*\rangle \otimes |qb_1^*\rangle \otimes \dots |qb_{k_i}^*\rangle + \boxed{rec_i^*}. \tag{3.24}$$

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:

$$\text{dif}\left(s_i, s_i^*\right) = \begin{cases} 1 & \text{if } |qb_i\rangle \neq |qb_i^*\rangle; \forall i = \overline{0, k_i} \text{ or } rec_i \neq rec_i^* \\ 0 & otherwise \end{cases} \tag{3.25}$$

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

$$e_t = \sum_{i=0}^{n-1} \text{dif}\left(s_i, s_i^*\right). \tag{3.26}$$

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

$$\xi_{sim} = \frac{1}{m} \sum_{j=0}^{m-1} e_{t_j} \tag{3.27}$$

As pointed out in references [75][76][78], the used FTAMs are valid if the relationship between the experimental ξ_{sim} and the assumed singular error rate ξ is of the order:

$$\xi_{sim} \sim \xi^2. \tag{3.28}$$

Chapter 4

Reliability with Reconfigurable Quantum Hardware

The need for error detection and correction techniques is vital in quantum computation, due to the omnipresent nature of quantum errors. No realistic prospect of an operational quantum computational device may be warranted without such mechanisms. Therefore, the fact that error detecting and correcting techniques have been developed has enhanced the feasibility of a potential quantum computer [76] [97]. The ITRS [126] is also listing the need for effective quantum fault-tolerant algorithms as an imperative for the quest of tomorrow's technology.

This chapter presents a methodology for improving the fault tolerance of quantum circuits by using the so-called *reconfigurable Quantum Gate Arrays* (rQGAs). Our solution reduces the problem of stabilizer coding safe recovery to preserving a given quantum configuration state. As shown in this chapter's practical example, the configuration register to be protected has a reduced number of qubits, and the overall dependability attribute [7] – reliability measured by the accuracy threshold [76] – is drastically improved.

4.1 Preliminaries

The theory of fault tolerant quantum computation employs special coding in order to protect useful data from the destructive effect of the environment. There are two main error sources: the first is due to the faulty behavior of the quantum gate that produces the so-called processing errors, while the second is generated by the macroscopic environment interacting with the quantum state (storing errors).

Within the quantum computational framework, the developed techniques for error detection and correction have the potential of a sound error recovery process, and error propagation is thwarted. However, quantum computation could be ruined if the error probability in the basic components (qubits, quantum gates) exceeds a certain *accuracy threshold*. Usually the microscopic quantum states are prone to frequent errors, thus *safe recovery* becomes extremely important [76].

The main error source is the *decoherence* effect [62]. The environment is constantly trying to measure the sensitive, microscopic quantum superposition state, while technologically it is not possible to perform a perfect isolation between the two of them. But the most insidious error appears when decoherence affects the quantum amplitudes without destroying them; these are very similar to small analogical errors. The solution to these problems is represented, on one hand, by intrinsic fault tolerance due to technological implementation (topological interactions Aharanov-Bohm [1]) and, on the other hand, by error correcting techniques [35][16][36][98].

The error detecting and correcting techniques 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. However, the necessary theoretical background is available for designing error detection and correction quantum circuits [69][76], which are effective if the error rate does not exceed the accuracy threshold.

4.1.1 Contributions

This chapter presents a fault tolerance improvement technique, based on reconfigurable quantum gate arrays (rQGA), inspired by the classical reconfigurable solutions to dependability problems. A relevant example is given by the Embryonics project [57], which improves the dependability attributes [68] of classical digital circuits by reconfigurable means. Moreover, recent developments have proven the effectiveness of Embryonics in attaining feasible computation in critical environments [79][80][81][82], very similar to quantum computation due to the similarities of fault models and error rates.

When using a quantum state (i.e. a superposition of classical configuration states) in order to configure the rQGA, the resulting circuit is equivalent to a superposition of distinctive circuits, each of which corresponding to a classical superposed configuration. Our method uses this feature in order to configure a superposition of error correcting circuits, based on distinctive encodings. Eventually, by applying measurement on the configuration register, just one classical configuration will remain. If the probability of one of the superposed circuits developing faulty behavior is ξ, then the probability of getting a faulty circuit by measuring the quantum configuration register (having m superposed classical configuration states) becomes ξ^m. For a sufficiently small ξ, the overall error probability becomes even smaller. As a result, the circuit reliability [7], (as a dependability attribute) measured by the accuracy threshold, is drasticaly improved [109].

The reconfigurable quantum hardware (rQHW) concept – under the form of rQGAs – can also be used in order to fight against correlated errors. Usually, it is considered that the quantum error has a single, random nature, but this theoretically convenient error model may prove as not completely accurate when dealing with future quantum hardware engineering problems.

4.2 Quantum fault tolerance

The available quantum error detection and correction techniques use stabilizer coding and special methods for ancilla qubit preparation [76]. Recent developments in this field [47][48] are also based on this approach.

Quantum computation not only introduces new types of errors, it also puts some computational constraints while generating some new problems. All these were already addressed [76][98], and the circuits presented in this chapter are built according to the available solutions.

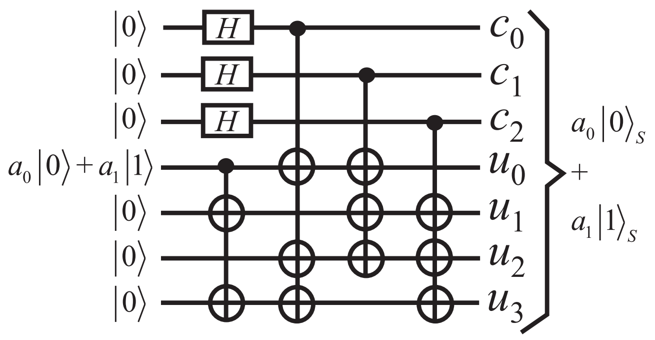


Figure 4.1: The circuit that returns the Steane encoding of an arbitrary state.

4.2.1 Quantum faults

The qubit can be affected by 3 types of errors: bit-flip, phase, both bit-flip and phase (see Equation 4.1). Besides these errors, small errors could affect the quantum amplitudes. However, there are methods of reducing any error to a bit-flip error [76].

$$a_0|0\rangle + a_1|1\rangle \overset{\text{error}}{\longrightarrow} \begin{cases} a_0|0\rangle + a_1|1\rangle & \text{no fault} \\ a_0|1\rangle + a_1|0\rangle & \text{bit-flip} \\ a_0|0\rangle - a_1|1\rangle & \text{phase-shift} \\ a_0|1\rangle - a_1|0\rangle & \text{both faults} \end{cases} \tag{4.1}$$

4.2.2 Quantum error detection and correction

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

$$N = \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{4.2}$$

As presented in Equation 4.3, in order to correct the phase error we apply the Z operator.

$$Z = H \cdot \sigma_x \cdot H = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{4.3}$$

The correction of the third error type (i.e. the situation when both bit-flip and phase shift are activated, see Equation 4.1) is achieved by applying a composed transformation upon the affected qubit:

$$Y = U \cdot Z = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{4.4}$$

Quantum error detection and correction is performed with special coding techniques, which are inspired from the classic Hamming codes. The syndrome obtained by measuring the proper ancilla qubits reveals the nature of the error.

Steane encoding

Steane's 7-qubit code [97][98][99] is derived from a classical single error correcting error, so it can detect and correct only single qubit faults in the code block [52].

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

$$H_A = \begin{array}{c} \begin{array}{ccccccc} c_0 & c_1 & c_2 & u_0 & u_1 & u_2 & u_3 \end{array} \\ \left(\begin{array}{ccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right) \end{array} \tag{4.5}$$

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

$$\begin{aligned} |0\rangle_S &= \frac{1}{2^{\frac{3}{2}}} \sum\text{even}_{(u_0u_1,u_2u_3c_0c_1c_2)} |u_0u_1u_2u_3c_0c_1c_2\rangle = \\ &= \frac{1}{2^{\frac{3}{2}}} \left(\begin{array}{l} |0000000\rangle + |0010111\rangle + |0101110\rangle + \\ |0111001\rangle + |1001011\rangle + |1011100\rangle + \\ |1100101\rangle + |1110010\rangle \end{array} \right) \end{aligned} \tag{4.6}$$

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

$$\begin{aligned} |1\rangle_S &= \frac{1}{2^{\frac{3}{2}}} \sum\text{odd}_{(u_0u_1,u_2u_3c_0c_1c_2)} |u_0u_1u_2u_3c_0c_1c_2\rangle = \\ &= \frac{1}{2^{\frac{3}{2}}} \left(\begin{array}{l} |1111111\rangle + |1101000\rangle + |1010001\rangle + \\ |1000110\rangle + |0110100\rangle + |0100011\rangle + \\ |0011010\rangle + |0001101\rangle \end{array} \right) \end{aligned} \tag{4.7}$$

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

$$\left\{ \begin{array}{lcl} m_0 & = & c_0 \oplus u_0 \oplus u_2 \oplus u_3 \\ m_1 & = & c_1 \oplus u_0 \oplus u_1 \oplus u_2 \\ m_2 & = & c_2 \oplus u_1 \oplus u_2 \oplus u_3 \end{array} \right. \tag{4.8}$$

The interpretation given to the syndrome from Equation 4.8 is presented in Table 4.1.

Applying Steane coding over an arbitrary given quantum state $|\psi\rangle = a_0|0\rangle + a_1|1\rangle$ (see Figure 4.1), generates the state from Equation 4.9 in order to use it for potential recoveries.

$$|\psi\rangle_S = a_0|0\rangle_S + a_1|1\rangle_S \tag{4.9}$$

Steane's ancilla coding

In order to compute the syndrome, ancillary qubits are necessary because we cannot measure encoded qubits. The design of circuits for encoding the ancilla must take into account two aspects: a preventing strategy against backward error propagation and ancilla accuracy verification [76].

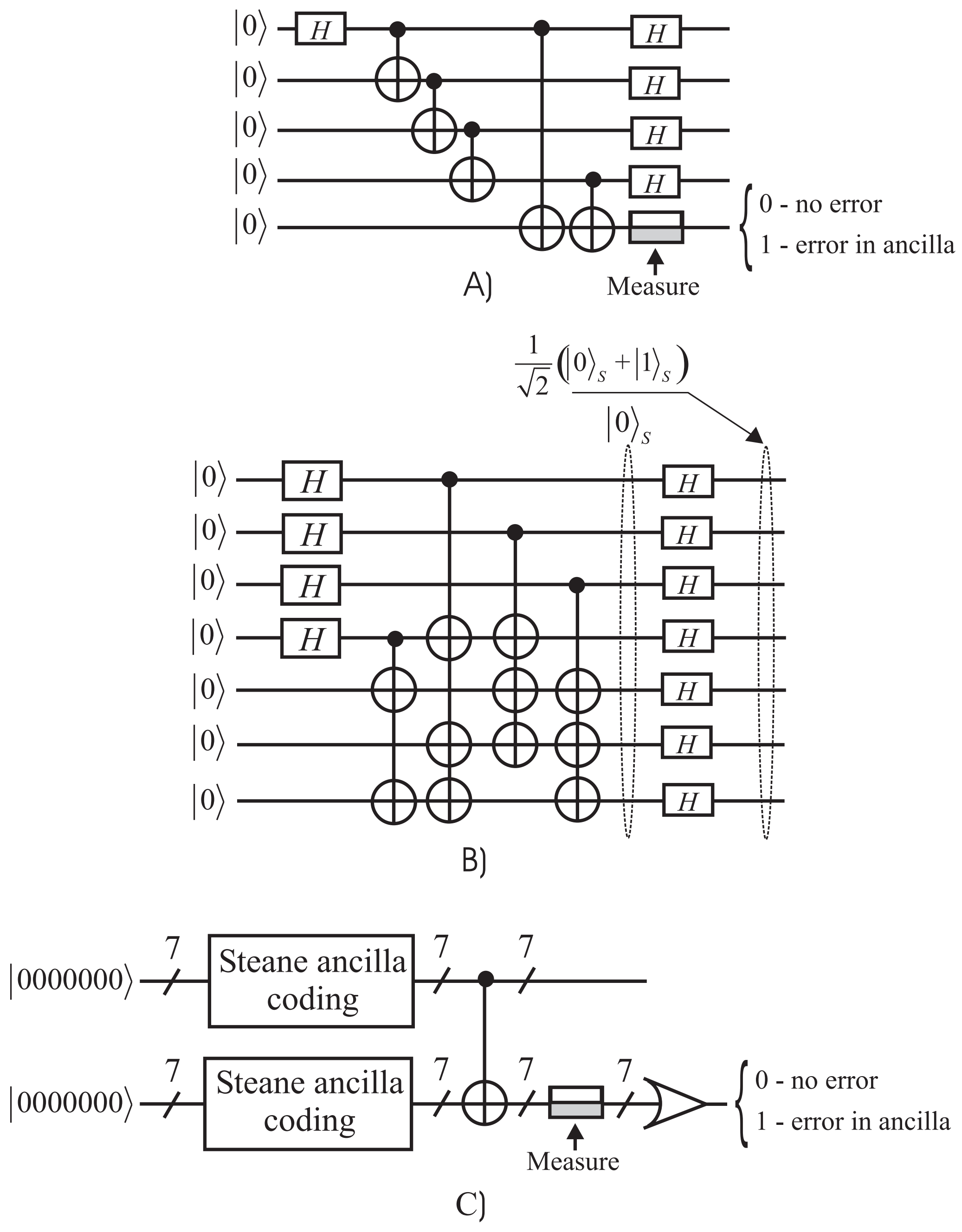


Figure 4.2: Ancilla coding: A) Shor's tecnique; B) Steane's technique; C) Verfication for Steane's ancilla, where the "State ancilla coding" blocks contain the circuit from B) except the rightmost level of Hadamard gates.

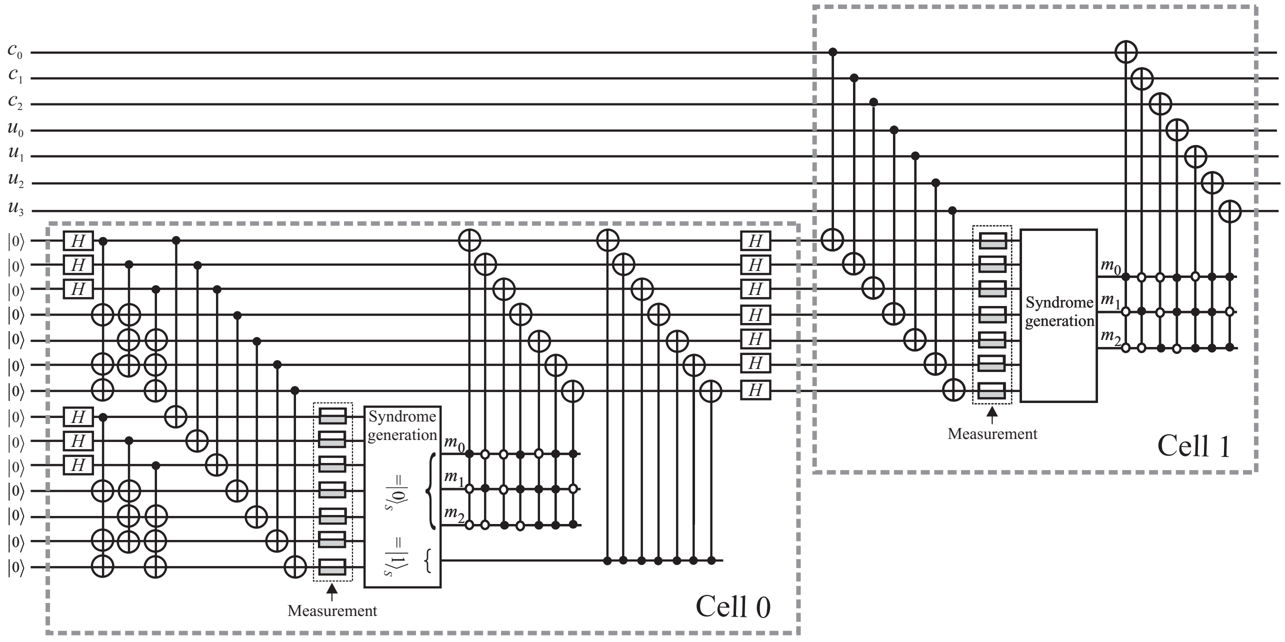


Figure 4.3: Single quantum bit-flip error correcting circuit with Steane ancilla coding.

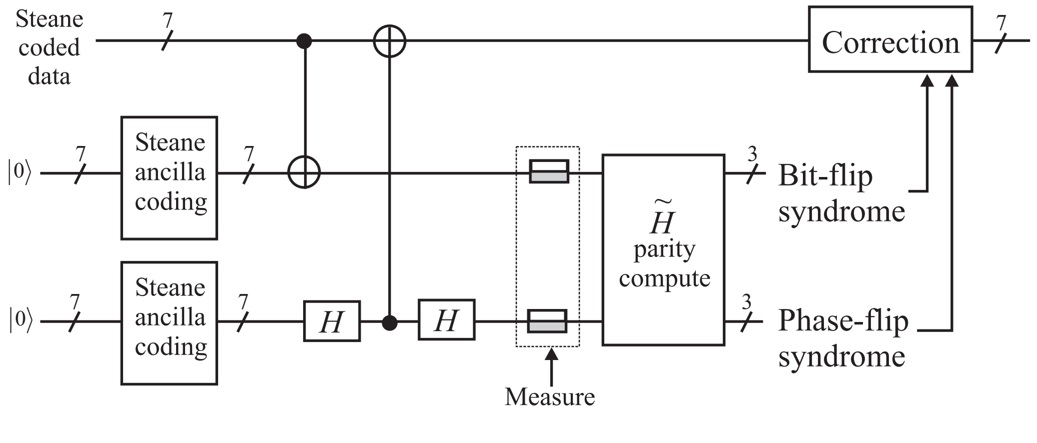


Figure 4.4: Error-correction with stabilizer generator measurement, Steane ancilla, and syndrome computation according to the check matix.

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

There are two coding techniques for the ancilla: the Shor and Steane coding (see Figure 4.2). We will focus on the most effective one, Steane ancilla preparation [76] (Figure 4.2 B) which generates:

$$|\mathrm{Anc}\rangle_{\mathrm{Steane}} = \frac{1}{\sqrt{2}} \left(|0\rangle_S + |1\rangle_S \right) \tag{4.10}$$

The bit-flip syndrome is obtained by first applying 7 *XOR* gates, having the data qubit as source, and the same position verified ancilla qubit (see Figure 4.2 C) as target. The ancilla is measured, and then the Hadamard matrix check H_A is applied in order to get the syndrome.

4.2.3 Putting it all together

With the results from the previous two subsections, we are able to present (in Figure 4.3) the complete circuits for bit-flip error correction (for Steane code, Steane ancilla preparation).

Considering the model of non-correlated errors, it was shown that the redundant syndrome computation with the circuits from Figure 4.3 will assure a data fidelity of order $1 - \mathcal{O}\left(\xi^2\right)$ when the single qubit error probability is ξ [76].

Stabilizer codes

Steane's code is a particular case of a stabilizer code [35]. In the stabilizer formalism, the code given in Equation 4.9 is characterized by the *check matrix* [62]:

$$\tilde{H}_A = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{4.11}$$

Having the check matrix generalization theory, more general generator measuring devices can be developed, as Figure 4.4 shows.

4.2.4 Accuracy threshold

Technological requirements

For a quantum code that corrects r errors ($r \in \mathbb{N}, r \geq 1$), Preskill [76] has shown that the accuracy threshold is:

$$\xi \sim (\log N)^{-p} \tag{4.12}$$

where N is the number of error correction cycles, and r^p is the number of computational steps required for syndrome computation.

Still, there will be an $N_{\max}$ so that, if $N > N_{\max}$, then the non-correctable error situation ($> r$ simultaneous errors) becomes likely. Under these conditions, there is a limit on how long the fault-tolerant computation can be.

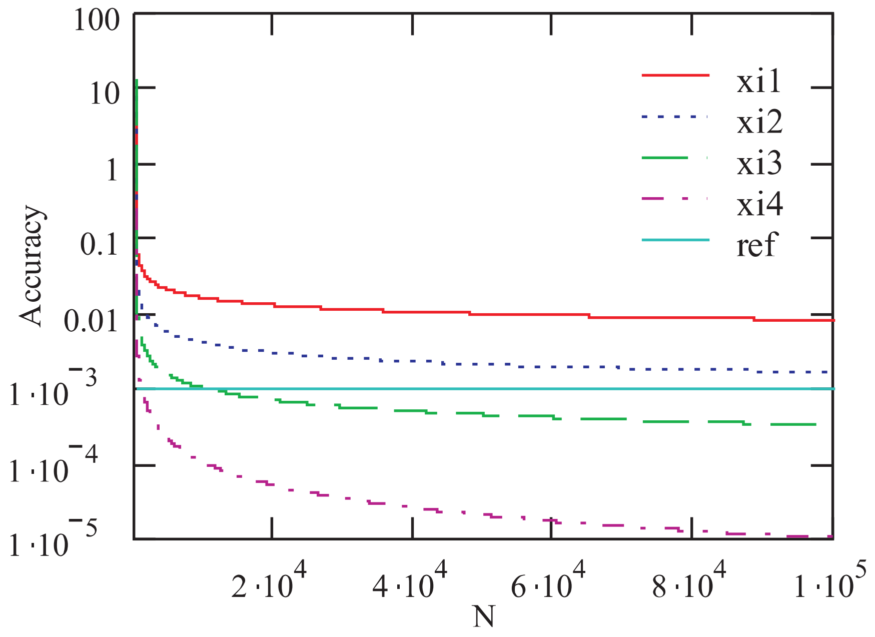


Figure 4.5: Graphical representation of accuracy degree required for the corresponding N, for different p's: 3 for xi_1, 4 for xi_2, 5 for xi_3. xi_4 corresponds to the no-coding situation, while ref is the reference accuracy (i.e. the accuracy allowed by today's state of the art technology).

Figure 4.5 presents the required accuracy degree getting closer to the present day's technological limit (tipically 10^{-3} for $p = 4$) after $N = 10^5 \approx N_{\max}$. For a fault tolerant Shor algorithm [89] encoding solution, this should have happened after $N = 10^9$ steps [76]. Therefore, $N_{\max}$ may not be large enough.

Concatenated coding

A solution for fault tolerant quantum computation with arbitrary length is *concatenated coding*, where each qubit is encoded by a block of n sub-qubits, each sub-qubit being encoded by other n blocks of qubits, and so on (Figure 4.6). For a m-depth (concatenation levels), a number of n^m qubits is required.

Preskill [76] provided an analysis that proves the effectiveness of 3-level concatenated coding, stabilizer coding, Steane ancilla circuit, by estimating its accuracy threshold:

$$\xi_{store,0} \sim 6 \cdot 10^{-4} \tag{4.13}$$

$$\xi_{gate,0} \sim 6 \cdot 10^{-4} \tag{4.14}$$

The analysis assumes two types of errors: gate errors and store errors. If the error rate is lower than values given by Equations 4.13 and 4.14, arbitrary long fault tolerant quantum computation is preserved.

4.3 A bird's eye critical view

This analysis provides the necessary critique of the actual quantum fault tolerance techniques, by identifying their weaknesses, and at the same time pointing a potential solution.

4.3.1 The big picture

In quantum computation the circuits are prone to fail, and safe recovery is difficult. Therefore, a straightforward classically-inspired solution is not feasible because the overall fail rate will be given by ancillary qubits preparation. Figure 4.7 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, regardless of the number of ancillary levels.

The only conceivable solution to the safe recovery problem is to use structural redundancy. Our objective, for the random single error model, is to assure an ξ^2 error probability for the corrected data. The method that is actually applied 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 (Figure 4.8), is sufficient for achieving the stated goal.

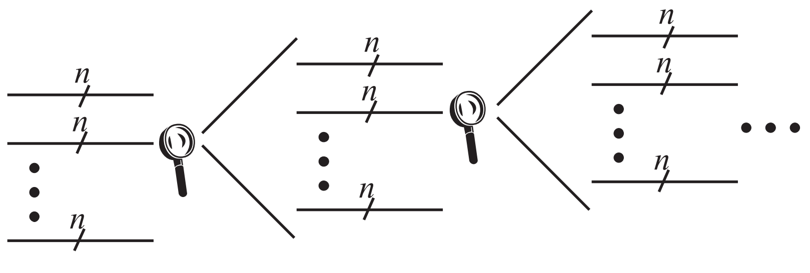


Figure 4.6: Concatenated coding: each qubit can be encoded by a block of sub-qubits.

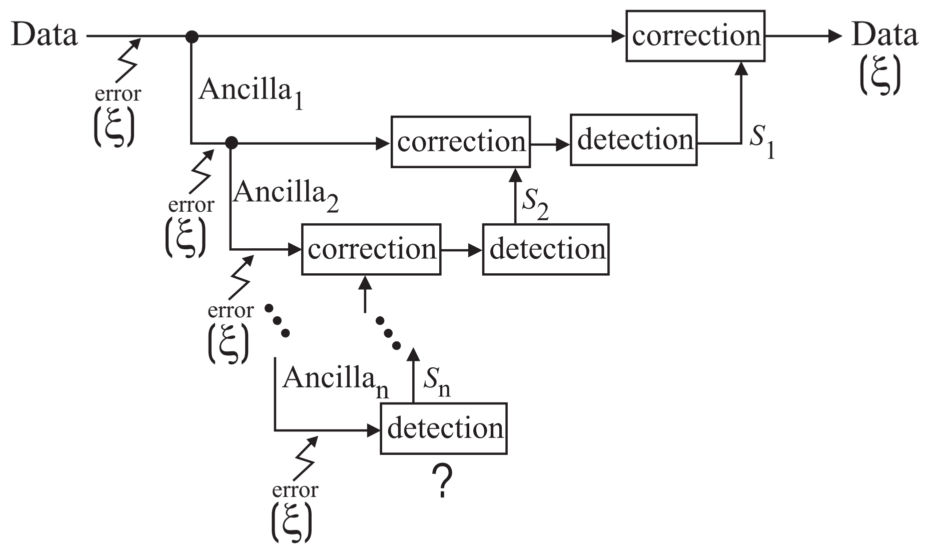


Figure 4.7: Classical fault tolerance approach for safe recovery.

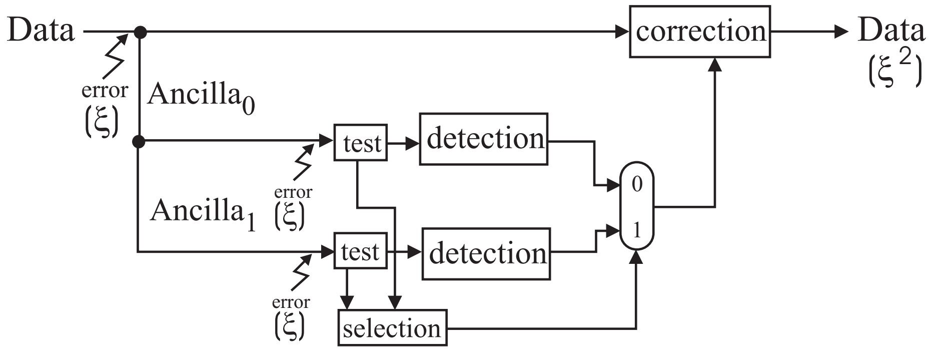


Figure 4.8: Steane safe recovery procedure.

4.3.2 Issues to be settled

There are some potential problems – of theoretical nature – that could affect the future quantum hardware engineering. We will debate over 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.

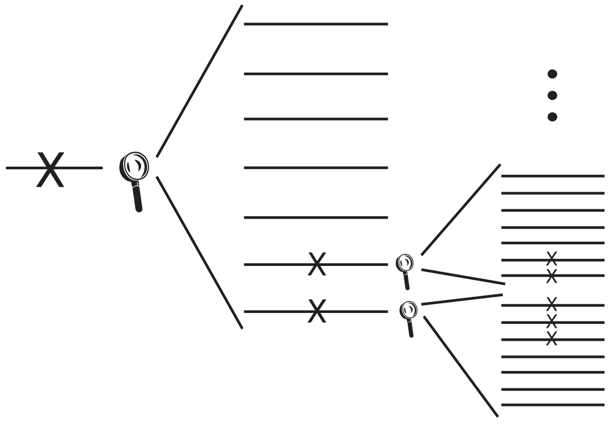


Figure 4.9: Concatenated coding affected by correlated errors.

When discussing problem (α from an engineering standpoint, one cannot exclude correlated errors (in time or space). Nevertheless, the biggest problem that comes from correlated errors is that concatenated code blocks are jeopardized. For example, if we use a concatenated code of size 7 on 3 levels, we have a total of $7^3 = 343$ qubits. Let us 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, it is very likely that the original code cannot be recovered (Figure 4.9).

As for the problem (β, when concatenated coding is employed, a lot of qubits are used in order to assure the reliability of just one qubit, and one still has to use structural redundancy for the safe recovery. Structural redundancy means at least doubling the ancillary qubit consumption (ancillary syndrome qubits also use concatenated coding!), even if the first ancilla set was correctly prepared.

Another theoretical aspect worth being mentioned, is that the described Quantum FTAMs (Fault Tolerance Algorithms and Methodologies) employ mostly classical algorithms, thus not making use of the full quantum computational power. One can assume that, by capitalizing on the exponential parallelism of quantum computation, better FTAMs can be developed.

4.4 The rQHW-based solution

4.4.1 Motivation

Assuming only uncorrelated probabilistic errors is not realistic from an engineering point of view [76]. Reconfigurable quantum hardware (rQHW) could be a solution for problems (α and (β, which are related to fault tolerance issues, induced by correlated errors. Also, it is possible to choose dynamically the depth of concatenated coding, because the quantum hardware can be reconfigurated accordingly.

When fighting the transient, correlated errors, the rQHW solution brings flexibility that allows for dynamic ancilla qubit preparation:

- choosing the ancilla qubits so that they will not be neighbors to each other;
- just one set of ancillary qubits is used for one data block; if testing [73] [42] reveals that it is faulty, then another ancilla set is configured in a different area of the reconfigurable quantum circuit.

In principle, a reconfigurable quantum circuit is a quantum gate array (QGA), acting on an *input register* in a way that it is prescribed by a *configuration register*. The processed input is stored in an *output register* (see Figure 4.10). In a formalized expression, we have:

$$U_{QGA} : |input\rangle \otimes |config\rangle \longmapsto |output\rangle \otimes |don't\ care\rangle \tag{4.15}$$

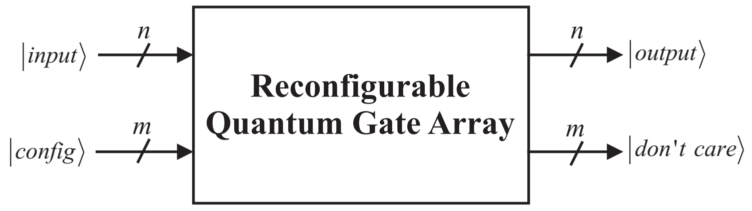


Figure 4.10: Reconfigurable quantum gate array: the involved registers.

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” [61].
2. It is impossible to build a *switch-based* quantum gate array, as shown in the following proposition.

Proposition (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.

Building the switch requires:

$$U_{switch} : |q_i\rangle_{i1} \otimes |q_c\rangle_{i2} \otimes |0\rangle_{o1} \otimes |0\rangle_{o2} \mapsto \left\{ \begin{array}{ll} |q_i\rangle_{i1} \otimes |q_c\rangle_{i2} \otimes |q_i\rangle_{o1} \otimes |0\rangle_{o2} & \text{for some } |q_c\rangle; \\ |q_i\rangle_{i1} \otimes |q_c\rangle_{i2} \otimes |0\rangle_{o1} \otimes |q_i\rangle_{o2} & \text{otherwise.} \end{array} \right. \tag{4.16}$$

Suppose that we are in the first instance of Equation 4.16 (for some $|q_c\rangle$). Then, the switch is reduced (because $|q_c\rangle$ is fixed) to

$$U_{sw1} : |q_i\rangle_{i1} \otimes |0\rangle_{o1} \longmapsto |q_i\rangle_{i1} \otimes |q_i\rangle_{o1} \tag{4.17}$$

But Equation 4.17 is impossible because the no-cloning law of quantum mechanics says that there is no U_{clone} so that for any $|\psi\rangle$:

$$U_{clone} : (|\psi\rangle \otimes |0\rangle) \longmapsto (|\psi\rangle \otimes |\psi\rangle) \,. \tag{4.18}$$

4.4.2 rQGA structure

Due to the second limitation of the rQHW, any programmable quantum gate array – consisting of a set of basic reconfigurable cells – will have to ripple the cells in a linear fashion. Figure 4.11 presents the consequence of Limitation 2.

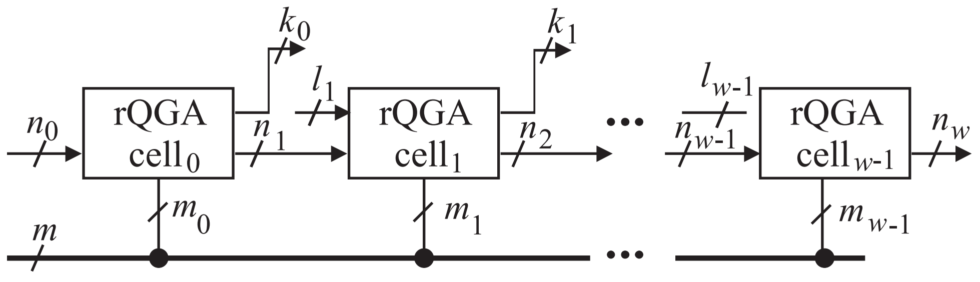


Figure 4.11: Linear connection of basic reconfigurable quantum gate arrays, allowed by the second limitation.

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 whole circuit's input) and coming from the input (l_j with $j = \overline{1, w-1}$). 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 whole circuit's output), and going to the general output ($k_j, j = \overline{0, w-2}$ qubits for each cell).Also, we have a $m = m_0 + m_1 + m_{w-1}$ qubit 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 ($l_0 = 0$ and $k_{w-1} = 0$):

$$l_j + n_j = k_j + n_{j+1} \tag{4.19}$$

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

Apropriate gates

For the rQGA, the usage of each gate has to be conditioned by dedicated qubits that 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 $\wedge_n(U)$ [10]. With this formalism $\wedge_n(U)$ is a $(n+1)$-qubit unitary operator, described by:

$$\begin{aligned} &\wedge_n(U)\left(|a_0, \ldots a_{n-1}, b\rangle\right) \mapsto \\ &\begin{cases} u_{b0}|a_0, \ldots a_{n-1}, 0\rangle + u_{b1}|a_0, \ldots a_{n-1}, b\rangle & \text{if } \wedge_{i=0}^{n-1} a_i = 1 \\ |a_0, a_1, \ldots a_{n-1}, b\rangle & \text{if } \wedge_{i=0}^{n-1} a_i = 0, \end{cases} \end{aligned} \tag{4.20}$$

where U is any unitary transformation $U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$ with $u_{00}, u_{01}, u_{10}, u_{11} \in \mathbb{C}$, $n \in \mathbb{N}$, and $a_0, a_1, \ldots a_{n-1}, b \in \mathbb{B} = \{0, 1\}$ [62][10].

For our rQGA purposes, 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:

$$\wedge_n(U) = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & & u_{0,0} & \ldots & u_{0,2^m-1} \\ & & & & & \ddots & \\ & & & & u_{2^m-1,0} & \ldots & u_{2^m-1,2^m-1} \end{pmatrix}. \tag{4.21}$$

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 [10], $\wedge_1(H)$ is a special 2-qubit gate described by the matrix from Equation 4.22 and depicted in Figure 4.12.a:

$$\wedge_1(H) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{4.22}$$

A conditioned measurement is also required in order to build a general reconfigurable cell for the studied quantum circuits. The control for conditioned measurement gates (see Figure 4.12.b) can be only a basis state (i.e. line c represents a bit, not a qubit).

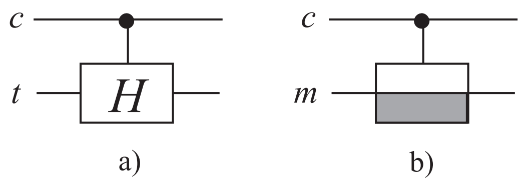


Figure 4.12: Special conditioned gates: *a*) Hadamard, and *b*) qubit measurement.

Basic reconfigurable cell

The basic cell of a reconfigurable quantum gate array (rQGA) 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.

Its general architecture, presented in Figure 4.13, has 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; full grey line means that the control can also be of quantum nature.

In Figure 4.13, the first level of gates from left to right are represented by two Hadamard gate sub-levels: the first one is common for all circuits involved (code generation, ancilla preparation and syndrome computation), the second is used for space basis transformation.The second level is formed of Toffoli gates. The controlled *XOR*s are placed so that there are gates with each qubit as source (and all the other qubits being targets). There are two sets of such gates, with a total of $n^2 - n$ gates in this level (n is the input register size). The third level of gates has two Hadamard gate sub-levels. The first sub-level is responsable with basis transformation, while the second corresponds to the second Hadamard level from the ancilla preparation circuit (Figure 4.1). The fourth gate level consists of conditional measurement gates.

Figure 4.13 shows that we need to be able to process the classical outcome of the qubit measurement with classical circuits. The "basis state logic" takes k classical inputs and produces n classical outputs. These control signals will configure the rest of the basic cell architecture, in order to correct both bit flip and phase shift errors.

The remaining gate levels are controlled with basis states (i.e. clasically), implementing the correction step.We have two sets of XOR gates, with each of the n qubits as targets, being classically controlled. The XOR gate level is guarded by two Hadamard (base changing) gate levels.

4.4.3 Quantum configuration

For the basic cell in Figure 4.13 we can present a structure of the configuration register. The configuration information for the left half of the cell is of quantum nature, while the right half is classical:

$$|config\rangle_{basic\,cell} = |\psi\rangle_{conf} \otimes |bit\ string\rangle \tag{4.23}$$

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

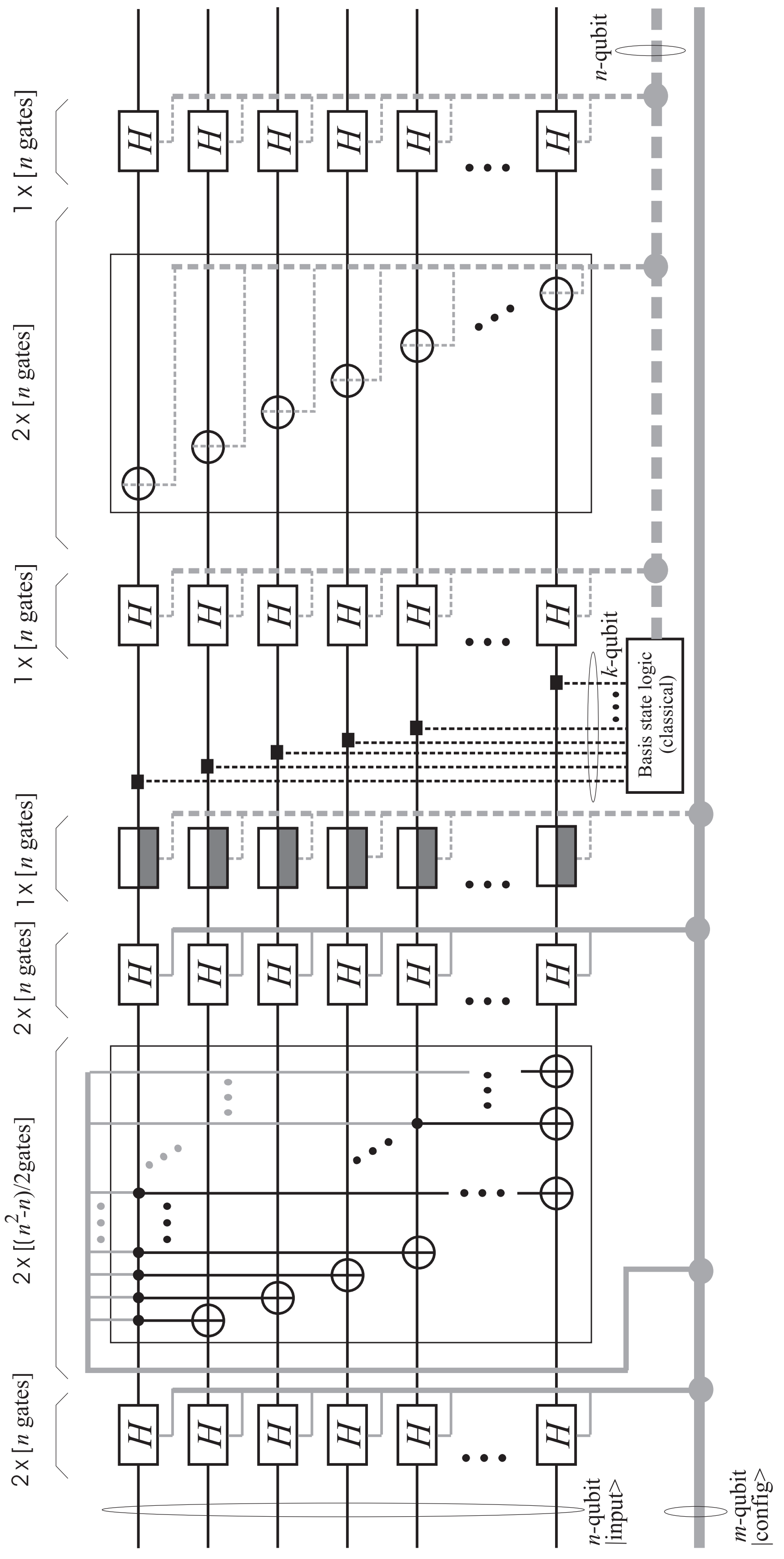


Figure 4.13: The basic reconfigurable cell for stabilizer encoding solutions.

$$|bit\ string\rangle = |\underbrace{mm\ldots m}_{n \text{ bits}}\underbrace{hh\ldots h}_{n \text{ bits}}\underbrace{xx\ldots x}_{2n \text{ bits}}\underbrace{hh\ldots h}_{n \text{ bits}}\rangle \tag{4.24}$$

In Equation 4.24, m denotes 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, it can be a superposition of basis state configurations with the structure given in Equation 4.26. The quantum configuration

$$|\psi\rangle_{conf} = \sum_{i=0}^{k} a_i|N_i\rangle \tag{4.25}$$

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 following structured binary string.

$$\begin{aligned} N_i = \ & |\underbrace{hh\ldots h}_{2n \text{ bits}}\underbrace{t^0_{0,1}\cdots t^0_{0,n-1}t^0_{1,2}\cdots t^0_{1,n-1}\cdots t^0_{n-2,n-1}}_{\frac{n(n-1)}{2} \text{ bits}}\rangle \\ & \otimes|\underbrace{t^1_{0,1}\cdots t^1_{n-2,n-1}}_{\frac{n(n-1)}{2} \text{ bits}}\underbrace{hh\ldots h}_{2n \text{ bits}}\rangle \end{aligned} \tag{4.26}$$

All hs and ts are binary digits 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 g input qubit as target.

The described reconfiguration methods can be applied for any qubit or qubit group from the input register, so that the qubit vicinity may fight correlated errors. When errors occur, the qubits from the code block are selected in such a way that they are physically separated from each other.

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 4.14 presents this feature of quantum reconfigurable circuits.

For our previously discussed fault tolerant circuits, we can use a superposition of classical configurations so that the circuit produces a superposition of all possible stabilizer codes. Moreover, we will be able to configure the circuit from Figure 4.3 with all possible stabilizer code versions, *at the same time*.

Of course, when measuring the configuration register, just one such fault tolerant configuration will remain, corresponding to a particular stabilizer code. With such a procedure, the probability of a gate error occurring in the fault tolerant circuit is substantially lowered; if the probability of one gate failing its service is ξ, then the probability of one gate failing *in the measured circuit* becomes ξ^k.

4.5 Code genaration with rQHW

This section provides examples of useful rQGA configurations, which implement the presented fault tolerant circuits.

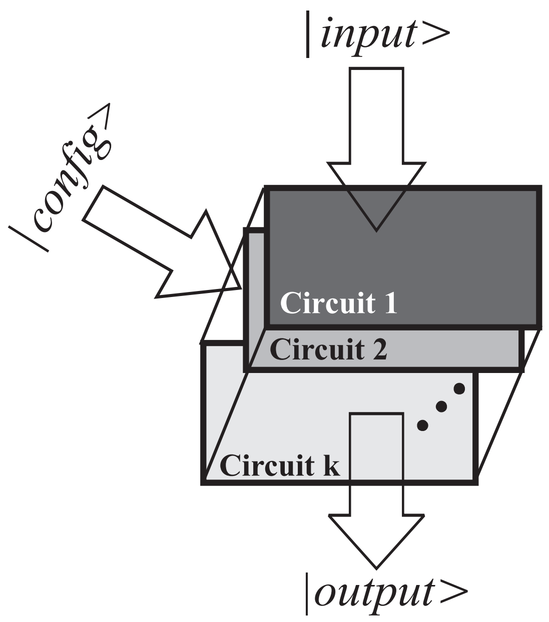


Figure 4.14: When the configuration register has a quantum nature, the same reconfigurable quantum gate array acts as a superposition of k simultaneous distinct circuits. These circuits share the same input state and the same output qubits. The output qubits encode a superposition of the superposed circuits distinct outputs.

4.5.1 Encoder with classical configuration

The qubit Steane encoder circuit from Figure 4.1 is configured from one basic rQGA cell as specified by Equations 4.23, 4.24, 4.25, and 4.26. The classical part of the configuration (see section 4.4.3) is not necessary, and the input-output size is $n = 7$ qubits.

$$|config\rangle^{0}_{basic\,cell} = |\psi\rangle^{Steane}_{conf} \otimes |bit\ string\rangle^{Steane} \tag{4.27}$$

$$|bit\ string\rangle^{Steane} = |0\rangle^{\otimes 35} \tag{4.28}$$

$$\begin{aligned} |\psi\rangle^{Steane}_{conf} = N^{Steane}_{0} = \ & \underbrace{|1110000\rangle \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}} \otimes \\ & \underbrace{|0\rangle^{\otimes 15} \otimes |101000\rangle}_{t^{0}_{i,j}} \otimes \\ & \underbrace{|001011011100111\rangle \otimes |0\rangle^{\otimes 6}}_{t^{1}_{i,j}} \otimes \\ & \underbrace{|1\rangle^{\otimes 7} \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}} \end{aligned} \tag{4.29}$$

4.5.2 Stabilizer code with steane ancilla

We provide a configuration for the circuit with Steane ancilla coding from Figure 4.3 (the partition is highlighted). The configuration states for the two 14-qubit basic cells are given in the following equations. For the first cell ("Cell 0") we have:

$$|config\rangle^{0}_{basic\,cell} = |\psi\rangle^{0}_{bft} \otimes |bit\ string\rangle^{0}_{bft} \tag{4.30}$$

$$\begin{aligned} |bit\ string\rangle^{0}_{bft} = & \underbrace{|0\rangle^{\otimes 7} \otimes |1\rangle^{\otimes 7}}_{\text{measurement}} \otimes \underbrace{|0\rangle^{\otimes 14}}_{\text{Hadamard}} \otimes \\ & \underbrace{|\overbrace{classical\ circuit\ outcome}^{7\text{ qubits}}\rangle \otimes |0\rangle^{\otimes 7}}_{\text{XORs}} \otimes \\ & \underbrace{|1\rangle^{\otimes 7} \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}} \end{aligned} \tag{4.31}$$

$$\begin{aligned} |\psi\rangle^{0}_{bft} = & \underbrace{|1110000\rangle^{\otimes 2} \otimes |0\rangle^{\otimes 14}}_{\text{Hadamard}} \otimes \\ & \underbrace{\begin{matrix} |001011\rangle \otimes |0\rangle^{\otimes 7} \otimes |01110\rangle \otimes |0\rangle^{\otimes 7} \\ \otimes |0111\rangle \otimes |0\rangle^{\otimes 7} \end{matrix}}_{t^{0}_{i,j}} \otimes \\ & \underbrace{|0\rangle^{\otimes 34} \otimes |001011011100111\rangle}_{t^{0}_{i,j}} \otimes \\ & \underbrace{|0\rangle^{\otimes 6} \otimes |1\rangle \otimes |0\rangle^{\otimes 12} \otimes |1\rangle \otimes |0\rangle^{\otimes 11} \otimes |1\rangle}_{t^{1}_{i,j}} \otimes \\ & \underbrace{\begin{matrix} |0\rangle^{\otimes 10} \otimes |1\rangle \otimes |0\rangle^{\otimes 9} \otimes |1\rangle \otimes |0\rangle^{\otimes 8} \otimes |1\rangle \\ \otimes |0\rangle^{\otimes 7} \otimes |1\rangle \end{matrix}}_{t^{1}_{i,j}} \otimes \\ & \underbrace{|0\rangle^{\otimes 21}}_{t^{1}_{i,j}} \otimes \underbrace{|0\rangle^{\otimes 28}}_{\text{Hadamard}} . \end{aligned} \tag{4.32}$$

and for "Cell 1":

$$|config\rangle^{1}_{basic\,cell} = |\psi\rangle^{1}_{bft} \otimes |bit\ string\rangle^{1}_{bft} \tag{4.33}$$

$$\begin{aligned} |bit\ string\rangle^{1}_{bft} = & \underbrace{|0\rangle^{\otimes 7} \otimes |1\rangle^{\otimes 7}}_{\text{measurement}} \otimes \underbrace{|0\rangle^{\otimes 14}}_{\text{Hadamard}} \otimes \\ & \underbrace{|\overbrace{classical\ circuit\ outcome}^{7\text{ qubits}}\rangle \otimes |0\rangle^{\otimes 7}}_{\text{XORs}} \otimes \\ & \underbrace{|0\rangle^{\otimes 14}}_{\text{Hadamard}} \end{aligned} \tag{4.34}$$

$$
\begin{aligned}
|\psi\rangle^1_{bft} = \quad & \underbrace{|0\rangle^{\otimes 28}}_{\text{Hadamard}} \otimes \\
& \underbrace{\begin{array}{l}|0\rangle^{\otimes 6} \otimes |1\rangle \otimes |0\rangle^{\otimes 12} \otimes |1\rangle \otimes |0\rangle^{\otimes 11} \otimes |1\rangle \\ \otimes |0\rangle^{\otimes 10} \otimes |1\rangle\end{array}}_{t^0_{i,j}} \otimes \\
& \underbrace{\begin{array}{l}|0\rangle^{\otimes 9} \otimes |1\rangle \otimes |0\rangle^{\otimes 8} \otimes |1\rangle \otimes |0\rangle^{\otimes 7} \otimes |1\rangle \\ \otimes |0\rangle^{\otimes 21}\end{array}}_{t^0_{i,j}} \otimes \\
& \underbrace{|0\rangle^{\otimes 91}}_{t^1_{i,j}}.
\end{aligned}
\tag{4.35}
$$

Quantum configuration for the encoding circuit

In this subsection, we present practical means for implementing a quantum configuration for the encoding circuit like the one from Figure 4.3, corresponding to a superposition of stabilizer encodings. For practical reasons, we consider as basis states in the superposition only the codes obtained by permuting the u_0, u_1, u_2, u_3 columns in the Hamming matrix H_A from Equation 4.5 ($4! = 24$ distinct codes).

For the best theoretical probability, we have to prepare a 12 qubit quantum state given by:

$$
|\psi\rangle_{12} = \frac{1}{2\sqrt{6}} \left(\begin{array}{c}
|011110111101\rangle + |011110111110\rangle + \\
|011111011011\rangle + |011111011110\rangle + \\
|011111101011\rangle + |011111101101\rangle + \\
|101101111101\rangle + |101101111110\rangle + \\
|101111010111\rangle + |101111011110\rangle + \\
|101111100111\rangle + |101111101101\rangle + \\
|110101111011\rangle + |110101111110\rangle + \\
|110110110111\rangle + |110110111110\rangle + \\
|110111100111\rangle + |110111101011\rangle + \\
|111001111011\rangle + |111001111101\rangle + \\
|111010110111\rangle + |111010111101\rangle + \\
|111011010111\rangle + |111011011011\rangle
\end{array} \right)
\tag{4.36}
$$

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

$$
|\psi\rangle^{stabil}_{conf} = \left(\frac{1}{2\sqrt{6}} \sum_{i=0}^{23} |N_i\rangle \right) \oplus |bit\ string\rangle^{stabil}
\tag{4.37}
$$

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

$$|N_i\rangle = \underbrace{|1110000\rangle \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}} \otimes \underbrace{|0\rangle^{\otimes 15} \otimes |101000\rangle}_{\text{fixed } t^0_{i,j}} \otimes |t^1_{i,j}\rangle \otimes \underbrace{|1\rangle^{\otimes 7} \otimes |0\rangle^{\otimes 7}}_{\text{Hadamard}}. \tag{4.38}$$

The $|t^1_{i,j}\rangle$ contains fixed qubits and the 12 qubits that participate to the superposition state from Equation 4.36, with the following basis state structure:

$$|t^1_{i,j\,basis}\rangle = |00\bullet\bullet\bullet\bullet 0\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet 000000\rangle. \tag{4.39}$$

The 0s correspond to the fixed values, whereas the filled dots mark the qubits that contribute to the superposition.

If we are to prepare state $|\psi\rangle_{12}$ from Equation 4.36 on the qubits highlited in Equation 4.39, we have to acknowledge the fact that the state itself is hard to obtain, because it is a superposition of 24 (not a power of 2) basis states.

We arrange the basis state codes from Equation 4.36 so that the same gate will not be used in all the superposed configurations. Four such configurations, in our heuristic approach, are given in Table 4.2.

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_{10}	q_{11}
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 4.2: The rows present the basis configurations which can be superposed with minimum gate usage (qubits $q_0, q_1 \ldots q_{11}$ are the ordered positions of the filled dots from Equation 4.39).

When these configurations are superposed, the same gate will be used in 3 out of 4 superpositions, hence producing 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 4.26:

$$N_i^{new} = |\underbrace{hh\ldots h}_{2n \text{ bits}} \underbrace{t^0_{0,1}\ldots t^0_{n-2,n-1}}_{\frac{n(n-1)}{2}\text{ bits}} \underbrace{t^1_{0,1}\ldots t^1_{n-2,n-1}}_{\frac{n(n-1)}{2}\text{ bits}}\rangle \otimes |\underbrace{t^2_{0,1}\ldots t^2_{n-2,n-1}}_{\frac{n(n-1)}{2}\text{ bits}} \underbrace{hh\ldots h}_{2n \text{ bits}}\rangle \tag{4.40}$$

Next, we will present $|t^1_{i,j}\rangle$ and $|t^2_{i,j}\rangle$ in detail, the way Equation 4.39 prescribes.

$$|t^1_{i,j\,basis}\rangle^{new} = |00\bullet\bullet\bullet\bullet 0\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet 000000\rangle \tag{4.41}$$

$$|t^2_{i,j\,base}\rangle^{new} = |00\square\square\square\square 0\square\square\square\square\square\square\square\square 000000\rangle. \tag{4.42}$$

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

$$|\rho\rangle^{\bullet}_{basis} = |b_0b_1111b_21b_3b_41b_51\rangle \tag{4.43}$$

$$|\rho\rangle^{\square}_{basis} = |11b_6b_7b_81b_911b_{10}1b_{11}\rangle. \tag{4.44}$$

The positions not marked with b_is are fixed with the corresponding binary value; it is only necessary to present the states corresponding to the b_i qubit groups:

$$|b_0b_1b_2b_3b_4b_5\rangle = |b_6b_7b_8b_9b_{10}b_{11}\rangle = \frac{1}{\sqrt{2}}\left(\begin{array}{c}|010110\rangle+\\|101001\rangle\end{array}\right). \tag{4.45}$$

The circuit for obtaining the state from Equation 4.45 is presented in Figure 4.15.

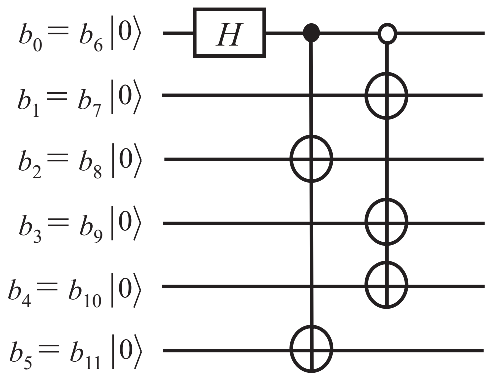


Figure 4.15: Circuit for setting the 6-qubit configuration state, from Equation 4.45.

4.5.3 Accuracy threshold analysis

If all the superposed classical configurations have the same quantum amplitude, then they will have the same probability of being measured. For the circuit from Figure 4.3, we have $4! = 24$ possible configurations, each corresponding to a distinct stabilizer code, and a $\frac{1}{4}$ probability of having the same gate in two distinct classical configurations. After the measurement of the configuration register – with a ξ_{gate} gate error probability given by the available technology – the reconfiguration solution will have an overall gate error probability of $\xi^{\frac{24}{4}}_{\text{gate}} = \xi^6$.

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

$$\xi_{\text{rQGA}} = \left(\xi_{\text{gate}}\right)^{k\times f_r} \tag{4.46}$$

where ξ_{gate} is the 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, and was obtained under the assumption that the configuration register is reliable. Due to practical reasons, the theoretical ξ^6 is hard to achieve. Basically, it is hard to set a quantum superposition containing a number of basis states that is not equal to a power of 2, and the physical limits of the basic cells may force us to a low f_r. Our engineering approach to these problems (Section 4.5.2) provides for a solution that guarantees an overall $\xi^{\frac{4}{3}\cdot 2} = \xi^{\frac{8}{3}}$ error rate (configuration is split in two groups).

In order to assess the consequences of reconfigurable quantum hardware approach, we will get back at Equation 4.12. As we use the same type of circuits, which are superposed with a quantum configuration, and the circuit for setting the quantum configuration will not increase p, we can express the accuracy threshold:

$$\xi_{threshold}^{rQHW} \sim (\log N)^{-p\cdot(1-f_r)\cdot\frac{1}{S}}. \tag{4.47}$$

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.

For a circuit using superposed stabilizer encodings, generated with rQHW in a quantum configuration, and for the heuristic implementation provided in 4.5.2, suppose we have a high $p = 6$. Then, our accuracy threshold $\xi_{threshold}^{rQHW}$ is of the order given by $\mathrm{xir}\,(N) = \log N^{-\frac{18}{8}}$ because $S = 2$ and $f_r = \frac{1}{4}$. The comparison between function $\mathrm{xir}\,(N)$ and the technologically assured threshold ($\mathrm{lim} = 10^{-3}$) is given in Figure 4.16. This comprehensively shows that the rQHW technique provides means for arbitrary long fault-tolerant quantum computation, because the $\mathrm{xir}\,(N)$ function clearly dominates the technological limit, even for very high N.

4.6 Summary

The qubits and gates involved in quantum computation implementations are prone to frequent errors, due to the delicate nature of quantum basis state superposition. While there is an ongoing effort in developing a quantum technology with inherent fault tolerance [1], special encoding techniques and quantum circuits have been created in order to attain fault tolerant quantum computation.

The specially designed techniques rely on classically inspired codes (i.e. *Hamming*, *stabilizer codes* [35] [97]). Due to the fact that there is a serious difficulty in achieving safe recovery, we have 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. But if the error rates (ξ's) exceed the estimated threshold values, the error-correcting techniques become useless.

4.6.1 Achievements

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.

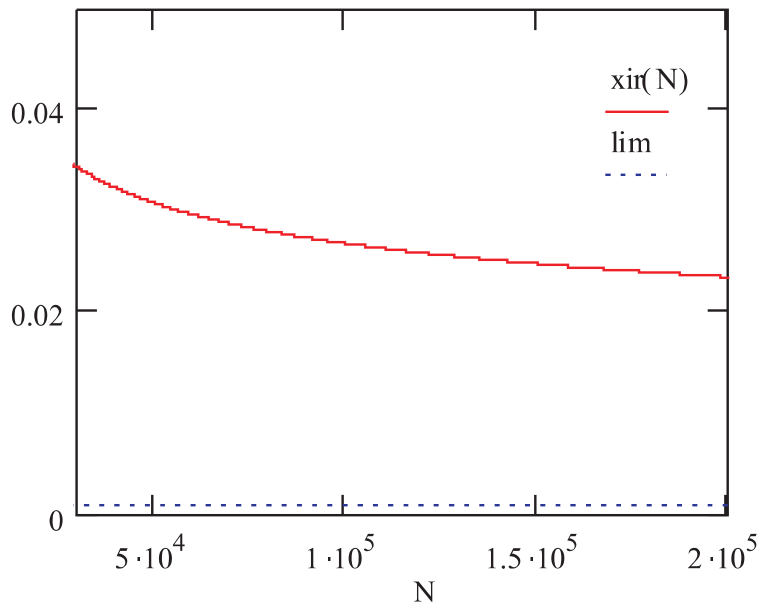


Figure 4.16: Evolution of accuracy threshold value for rQHW stabilizer codes ('xir' function) with the number of computational steps (N). The technological accuracy limit ('lim') is also provided for comparison.

As it turned out, if the rQGA uses a quantum configuration, then the overall reliability (measured by the accuracy threshold) is improved by reducing the problem of controlling gate errors to preserving a given (reduced) quantum configuration register.

The effectiveness of our proposed solution was proved by our particular case study, which used the rQGA basic cell construction, in order to generate a superposition of 4 distinct stabilizer code circuits. The considered error correction framework was stabilizer data 7-qubit encoding with Steane ancilla preparation. Given the case study conditions, the accuracy threshold estimation clearly dominates the technologically assured failure rate, thus creating incentive for fault tolerant quantum computation in this context.

4.6.2 Issues to be settled

Although the qualitative evaluation of our approach is promising, a quantitative assessment [64] of this paper's theoretical analysisis needed, by employing simulated fault injection. This approach is intensively used in classical digital circuit design [122]. In quantum computation it can be implemented by adding fault injection features to one of the available simulation frameworks [106][107][115] (See Section 3.4).

Chapter 5

Evolvable Quantum Hardware

Building the core of this chapter has started as an attempt to design a quantum counterpart for the classical evolvable hardware [102]. The benefits of implementing this concept were also mentioned in the previous chapter: a very versatile and adaptive structure that has various applications.

Starting from the definition of evolvable hardware (EHW): EHW = RHW (reconfigurable hardware) + GAs (genetic algorithms), our quest in the quantum computation field is presented in Figure 5.1. In the figure, the configuration of the programmable quantum circuit structure, which was discussed in the previous chapter, is generated by a quantum version of the genetic algorithms (i.e. the so-called QGAs, or Quantum Genetic Algorithms [33]).

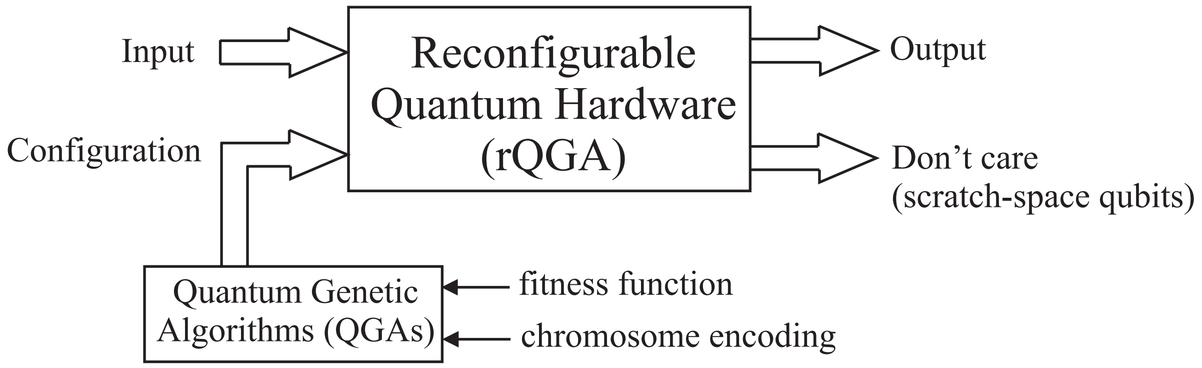

Input
Reconfigurable Quantum Hardware (rQGA)
Output
Configuration
Don't care (scratch-space qubits)
Quantum Genetic Algorithms (QGAs)
fitness function
chromosome encoding

Figure 5.1: Evolvable quantum hardware.

As a consequence, the problem of designing evolvable quantum hardware is reduced to designing circuits for implementing Quantum Genetic Algorithms. Here, there are a lot of open problems, as it is not clear yet how to design and build QGAs [33][86][87][93].

5.1 Preliminaries

5.1.1 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 [87]. Nevertheless, approaching specific applications this way will result in a significant performance enhancement [40][41].

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 possible 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 individuals 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 Grover's algorithm 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 can figure out a way to mark the best fitness, inside the fitness superposition state.

5.1.2 Objective

The reconfigurable quantum gate array from Figure 4.10 will translate to *evolvable quantum hardware (QEHW)* if its configuration register, the m-qubit register $|config\rangle$, represents a state outputted by means of a genetic algorithm. Therefore, if 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.

5.2 Quantum genetic algorithms

As part of Quantum Evolutionary Programming, QGAs have the ingredients of a substantial algorithmic speedup, due to the inherited properties from both QC and GA. However, there are still questions as to how would it be possible to implement a genetic algorithm on a quantum computer. The attempts made in this particular direction suggest there is room left for taking advantage of the massive quantum computation parallelism [87]. Moreover, some questions were left open, as pointed out in [33].

5.2.1 Running GAs in a quantum computational environment

For the first time, the possibility (and the advantages) of the QGAs were indicated in [87]. The approach described here contains hard evidence for QGA speedup, but there still are some unanswered questions [33]. The proposed algorithm uses a number of m register pairs:

$$|\psi\rangle_i = |\phi\rangle_i^{individual} \otimes |\rho\rangle_i^{fitness} \tag{5.1}$$

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 registers, 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 [62], the result is only one of the superposed values; in the individual register will remain as superposed the individuals that give the measured fitness. Fitness register measurement is a crucial element in developing QGAs [87]. For the general expression of the pair register (N-qubit for the individual register and M-qubit for the fitness register) given in Equation 5.2, the measurement of the second register ($|y\rangle$) will have r as result with the probability from Equation 5.3.

$$|\psi\rangle_i = \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^M-1} c_{x,y}|x, y\rangle, \text{ with} \sum_{x=0}^{2^N-1} \sum_{y=0}^{2^M-1} |c_{x,y}|^2 = 1 \tag{5.2}$$

$$P(r) = \sum_{x=0}^{2^N-1} |c_{x,r}|^2 = 1 \tag{5.3}$$

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

$$|\psi_r\rangle_i = \frac{1}{\sqrt{P(r)}} \sum_{x=0}^{2^N-1} c_{x,r}|x, r\rangle \tag{5.4}$$

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 [87], 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 superpositions of individuals and compute the corresponding fitness pair register $|\rho\rangle_i^{fitness}$ (the outcome will be a superposition of m fitness values).

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 [33] 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 [33] 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 [86].

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 [33]. 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 [87].

5.2.2 Mathematical formalism

The QGA formalism uses m quantum register pairs (N-qubit individual register and M-qubit fitness register,) as presented in Section 5.2.1. 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 [65][66], hence a pseudo-classical operator with a corresponding Boolean function: $f : \{0,1\}^N \to \{0,1\}^M$, $U_f : |x\rangle \otimes |0\rangle \to |x\rangle \otimes |f(x)\rangle$ if $|x\rangle$ is a basis state.

When acting on a superposition, the unitary operator corresponding to function f will dictate the following mapping:

$$U_f : \sum_{x=0}^{2^N-1} a_x|x\rangle \otimes |0\rangle \rightarrow \sum_{x=0}^{2^N-1} a_x|x\rangle \otimes |f(x)\rangle = \sum_{x=0}^{2^N-1} a_x|x, f(x)\rangle \tag{5.5}$$

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 obtained by measurement): $|\psi\rangle_i^3 = \frac{1}{\sqrt{k_i}}\sum_{v\in\{0,1,\ldots,n-1\}}|v\rangle_i^{ind} \otimes |y\rangle_i^{fit}$ with k_i values in $\{0,\ldots,n-1\}$ to satisfy $f_{fit}(v) = y$.
4. Repeat

   a. Selection according to the m measured fitness values $|y\rangle_i$.

   b. Crossover and mutation are employed in order to prepare a new population (setting the m individual registers $|u\rangle_i^{ind}$).

   c. For the new population, the corresponding fitness values will be computed and then stored in the fitness registers($|f_{fit}(u)\rangle_i^{fit}$).

   d. Measure all fitness registers

   Until the condition for termination is satisfied.

Besides the necessary formalism, reference [33] 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 [69].

5.3 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 some 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 $f : \{0,1\}^N \rightarrow \{0,1\}^M$) 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 5.4 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][28]. 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 proposed 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 5.3.1 presents the quantum algorithm for finding the maximum [2], Section 5.4 presents details for oracle implementation and fitness register structure, while Section 5.5 provides our adaptation of the algorithm in order to find the best value in the fitness register.

5.3.1 Computing the maximum

This subsection analyzes the available quantum methodologies for finding the maximum, and provides a modified version of the original algorithm [2][28], in order to meet our QGA demands.

The initial algorithm

The quantum algorithms for minimum/maximum finding [2][28] are inspired from the classical "bubble sort" algorithm, but their complexity in quantum version is $\mathcal{O}(\sqrt{n})$.

Such an algorithm takes an unsorted table of m elements 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 maximum element ($P[k]$).

In order to meet our demands, Grover's algorithm uses a specially designed oracle that "marks" all the basis states greater than some given value j (within the pool):

$$O_j(i) = \begin{cases} 1 & \text{if } P[i] > P[j] \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

We notice that the oracle works with the pool indexes as parameters. Also, there is no indication as to how is the pool represented. The oracle just "knows" the answer to the following question: "is $P[i]$ bigger than $P[j]$?" Therefore, the resulting 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 $k := random\ number$; $0 \le k \le m-1$ as the starting index of this search;
2. Repeat $\mathcal{O}(\sqrt{m})$ times
    a. Set two quantum registers as $|\psi\rangle = \frac{1}{\sqrt{m}} \sum_{i=0}^{m-1} |i\rangle|k\rangle$; the first register is a superposition of all indexes;
    b. Use Grover's 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 which are indexes for values $> P[k]$. 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 $\frac{1}{2}$.

The modified algorithm

In the initial algorithm, a quantum form for the pool of elements is not necessary. However, for our QGA related purposes, we need to maintain the correlation between the individual register (corresponding to the indexes) and the fitness register (corresponding to the fitness values). Therefore, a maximum finding algorithm – that is usable in the desired, genetic algorithm context – must have a quantum (i.e. basis state superposition) state for representing the values, which in turn has to be correlated appropriately with the quantum register representing the indexes.

This means that the oracle will operate on the values register, where its input data is available. Hence, the oracle expression from Equation 5.6 will be modified accordingly:

$$\tilde{O}_y(x) = \begin{cases} 1 & \text{if } x > y \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

In Equation 5.7, $x, y \in \mathbb{N}$ and are encoded by the superposed basis states from the values register. Also, because we will have to run a number of $s \in \mathcal{O}\left(\sqrt{m}\right)$ steps to complete the algorithm, so it is necessary that a number of s indexes-values quantum register pairs be prepared. Each register pair, except the last one used, generates a partial maximum search solution. The modified quantum maximum finding algorithm is presented in the following pseudocode:

Quantum Algorithm for finding the maximum from an unsorted table of m elements, which is represented as a quantum state

```
1. Initialize $k := random\ integer$ with $0 \leq k \leq m-1$; $max := P[k]$ ;
2. For $j := 0$ to $s-1$ set the pair registers as $|\psi\rangle_j^1 = \frac{1}{\sqrt{m}} \sum_{i=0}^{m-1} |i\rangle_j^{index} \otimes |0\rangle_j^{value}$;
3. For $j := 0$ to $s-1$ set the value corresponding to the index $|\psi\rangle_j^2 = P|\psi\rangle_j^1 = \frac{1}{\sqrt{m}} \sum_{i=0}^{m-1} |i\rangle_j^{index} \otimes |P[i]\rangle_j^{value}$;
4. For $j := 0$ to $s-1$ loop
    (a) Apply the oracle on the value register $\tilde{O}_{max}(P[i])$. Therefore, if $|P[i]\rangle_j^{value} > max$ then the corresponding basis states are marked;
    (b) Use Grover's algorithm for finding marked states in the value register after applying the oracle. As pointed out in reference [62], we find one of the marked basis states $|p\rangle = |P[i]\rangle_j^{value}$, with $P[i]\, max > 0$;
    (c) $max := p$;
5. Having the highest value in the $|\bullet\rangle_{s-1}^{value}$ register, we measure the $|\bullet\rangle_{s-1}^{index}$ register in order to obtain the corresponding individual (or one of the corresponding individuals).
```

5.4 The oracle

The oracle implementation must be made so that the problems mentioned in reference [33], namely α) and β) from Subsection 5.2.2, are dealt with. This subsection presents the envisaged solutions.

5.4.1 Solving problem α)

In order to deal with problem α), we have to adopt a constraint, which does not restrict the generality of the fitness functions. We consider the ordinary fitness function f_{fit} (which applies only on the valid individuals) $f_{fit} : \{0,1\}^N \rightarrow \{0,1\}^M$, which is Boolean (and therefore universal), with a straightforward correspondence to the unitary representation $U_{f_{fit}}$ [62][65]. 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 $f_{fit}^{mod} : \{0,1\}^N \rightarrow \{0,1\}^{M+1}$ as:

$$f_{fit}^{mod}(x) \in \begin{cases} 0 \times \{0,1\}^M & \text{if} \quad x \text{ is a non-valid individual} \\ 1 \times \{0,1\}^M & \text{if} \quad x \text{ is a valid individual} \end{cases} \tag{5.8}$$

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 5.2 (where the quantum state matrix representation is used).

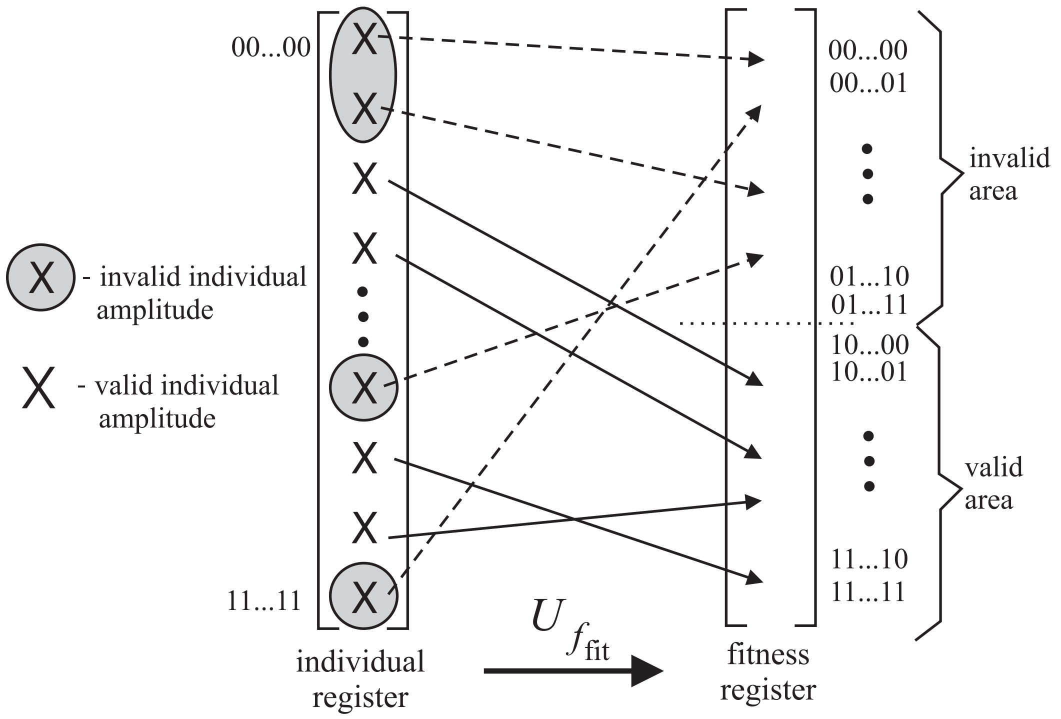


Figure 5.2: The basics of fitness function construction: when is applied to valid individuals it produces a value in the valid area (upper half: $|10\dots00\rangle \dots |11\dots11\rangle$) of the fitness register, whereas when applied to invalid individuals, the corresponding values in the fitness register will always be in the invalid area (lower half: $|00\dots00\rangle \dots |01\dots11\rangle$).

5.4.2 Building the oracle

As our approach avoids solving problem β directly, the remaining task concerns the definition of an appropriate (i.e. application specific) oracle, starting from Equation 5.7 and the algorithm from Subsection 5.3.1.

We propose a solution that uses two's complement number representation [71] for marking the states that have a value greater than a given $l \in \mathbb{N}, l > 0$. As a consequence, the fitness register will have the form from Figure 5.3.

The oracle processes all the fitness register qubits except the most significant one (v), which indicates if the value represented by the other qubits belongs to a valid individual or not. All the value qubits ($f_M \dots f_0$) from the fitness register encode two's complement positive integers as fitness values. The oracle adds $-(l+1)$ to the fitness register, therefore the basis states (from the state output by the quantum adder [113]) greater than l will always have $f_M{}' = 0$ (see the oracle implementation from Figure 5.4.)

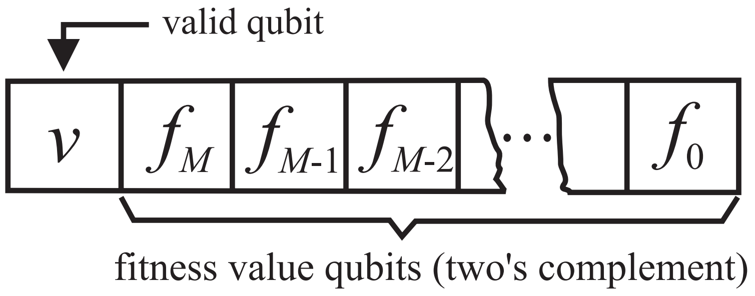


Figure 5.3: The format of the fitness register, for the oracle implementation that is based on a two's complement approach.

For the solution given in Figure 5.4 we have used 2 negation gates (denoted with 'x') and one XOR gate [10] [62], in order to change the phase of the corresponding superposed basis states. The architectures for the quantum arithmetic circuits, the adder/subtractor for our particular case, are presented in references [34] and [113]. After marking the corresponding basis states (by shifting their amplitudes), their value is restored by adding $l+1$. Only the qubits containing the result of the arithmetic function $(f''_0 \dots f''_M)$ are used by the Grover iteration circuit [38][62] in order to find one of the marked basis states.

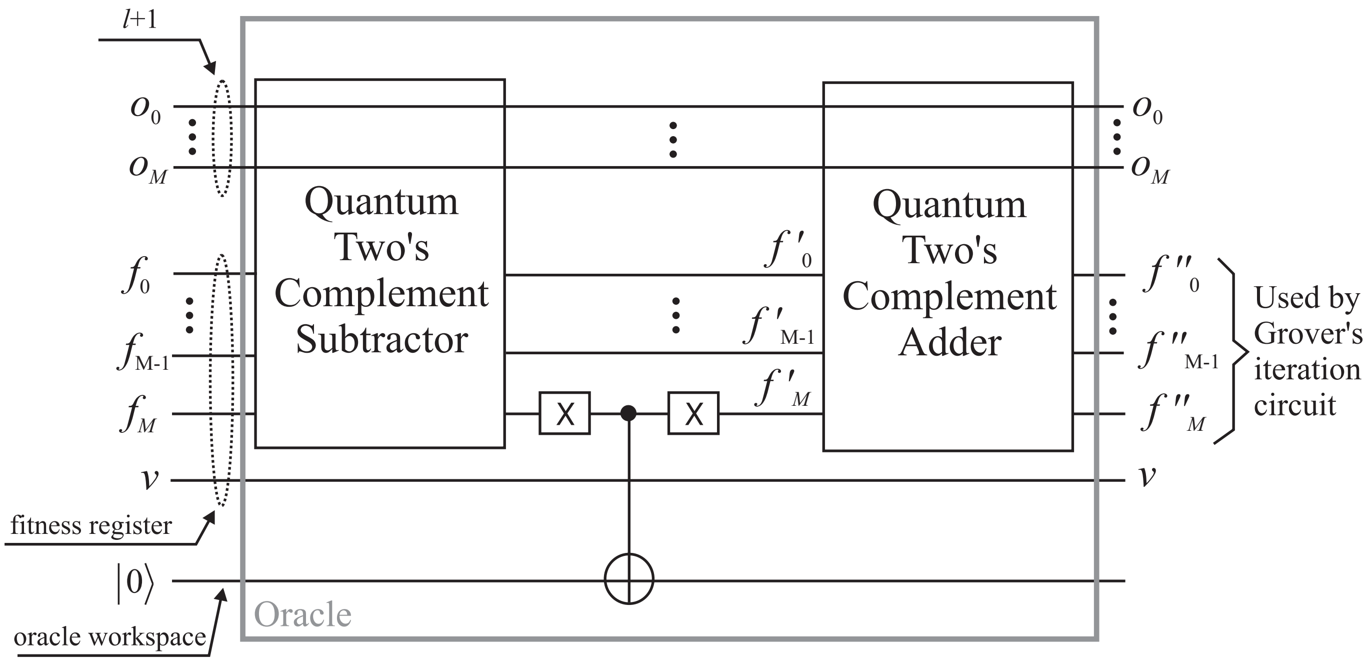


Figure 5.4: Oracle implementation for a fitness register having the structure from Figure 5.3.

The Grover algorithm that is actually applied to the $f''_M \dots f''_0$ register complies with the Grover algorithm version defined by reference [12], in order to find one of the marked solutions, without any a priori knowledge about the number of solutions.

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 [65][66][113]. The Grover iteration will find as a marked basis state $|p\rangle = |f''_M \dots f''_0\rangle$, with $p \in \mathbb{N}$, $f''_M, \dots, f''_0 \in \{0,1\}$ which is given by $|p\rangle = -|q\rangle$ for $|q\rangle = |f_M \dots f_0\rangle$, with $f_M, \dots, f_0 \in \{0,1\} = \mathbb{B}$.

The algorithm listed below is inspired from the quantum maximum algorithm from Section 5.3.1. The initial *max* value must obey the $2^{M+1} \leq max \leq 2^{M+2} - 1$ condition, so that the search for the highest fitness value will take place only in the valid fitness area. We have a number of $m \in \mathcal{O}\left(\sqrt{N}\right)$ (due to the complexity analysis provided in [2]) pair registers (individual-fitness), where the individual register is on N qubits, and the fitness register on $M+2$ qubits. Also, it can be said that m "quantum selection steps" are required by this algorithm.

5.5 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 possible 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 Grover's algorithm. Although these operators are not required in the given context, we may say that this new approach also induces some form of a "quantum evolution". The best fitness value emerges step by step, with each register pair being employed. The selection is implemented by the "marking the basis states" process. Grover's algorithm is used in order to obtain one of these basis states (i.e. selected individuals) out of the quantum superposition. The selection process of the next step uses the value of Grover's algorithm output, and so on.

The algorithm listed below is inspired from the modified quantum maximum algorithm from Section 5.3.1.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 $m \in \mathcal{O}\left(\sqrt{N}\right)$ (due to the complexity analysis provided in reference [2])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 computation
   $|\psi\rangle_i^2 = U_{f_{fit}} |\psi\rangle_i^1 = \frac{1}{\sqrt{2^N}} \sum_{u=0}^{2^N-1} |u\rangle_i^{ind} \otimes |f_{fit}(u)\rangle_i^{fit}$;

3. $max := random\ integer$, so that $2^{M+1} \leq max \leq 2^{M+2} - 1$;

4. For $i := 0$ to $m-1$ loop

    (a) Apply the oracle $\tilde{O}_{max}(f_{fit}(u))$. Therefore, if $|f_{fit}(u)\rangle_i^{fit} > max$ then the corresponding
        $|f_{fit}(u)\rangle_i^{fit}$ basis states are marked;
```

(b) Use Grover's iterations for finding marked states in the fitness register after applying the oracle. We find one of the marked basis states $|p\rangle = |f_{fit}(u)\rangle_i^{fit}$, with $f_{fit}(u)\,max \geq 0$;

(c) $max := p + 1$;

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, if there are more than one solution).

Measuring the individual corresponding to the best fitness value is possible due to the fact that only pseudo-classical operators [65] (i.e. Controlled NOT and Hadamard gates [62]) were applied on the second (fitness) register, so that the correlation with the first (individual) register was not destroyed.

Grover's algorithm, interpreted as prescribed by reference [12], is a method of augmenting the amplitude of oracle marked basis states, and it works even when the number of such states is not known in advance. Therefore, Grover's search will work even if the superposed basis states in the search quantum register have uneven amplitudes. However, this means that the complexity assessment of our proposed algorithm cannot be performed straightforwardly by referring to the complexity of the initial algorithm (maximum finding quantum algorithm) which uses even amplitudes for the basis states within the quantum superposition. Even if a much thorough investigation is required, as far as the exact complexity analysis is concerned, we can assess the complexity with the following proof. If the quantum amplitudes are not equal for all the superposed individual-fitness pair basis states, an extra number of at most $\mathcal{O}(\sqrt{n})$ steps are required to get to this point. Therefore, we will have at most $\mathcal{O}(\sqrt{n} \cdot \sqrt{n}) = \mathcal{O}(n)$ steps to complete the execution of the Reduced Quantum Genetic Algorithms. A practical example of how this algorithm works is presented in Appendix D.

5.6 Quantum evolutionary strategy

In this section we analyze how is affected the evolutionary strategy by the quantum computation features, according to our perspective over the QGAs. Our main reference would be the general, classical evolutionary computation systems, as described in reference [96] (pages 37-39). Figure 5.5 (a) is presenting the typical classical evolutionary system: *generation* of some individuals (usually in a random manner), followed by the *assessment-selection-variation* loop (which will eventually generate the solution).

The assessment process is based on computing fitness values, corresponding to the individuals in the successive populations. Because we cannot be sure that the solution is an individual from the current population, other individuals may be generated by sexual variation (crossover) or non-sexual variation (mutation).

In our quantum computation approach, as presented in 5.5, the *variation* stage of the evolutionary strategy is no longer necessary. The problem solving strategy for evolutionary quantum computation is presented in Figure 5.5 (b). The generation stage means that all the possible individuals (valid or non-valid) are generated as a basis-state superposition due to the fact that the qubits are used to represent the chromosome (i.e. individual encoding). Then, the assessment is applied on all the superposed individuals, therefore generating a fitness register, which consists of a superposition of all the fitness values. The selection is applied on all the superposed individuals, which are available within the quantum chromosome, by making use of Grover's algorithm. The fact that all the individuals are already available, as superposed, means that the techniques used for generating new individuals are useless. Figure 5.5 (b) is reflecting this situation, because it is simpler than the model

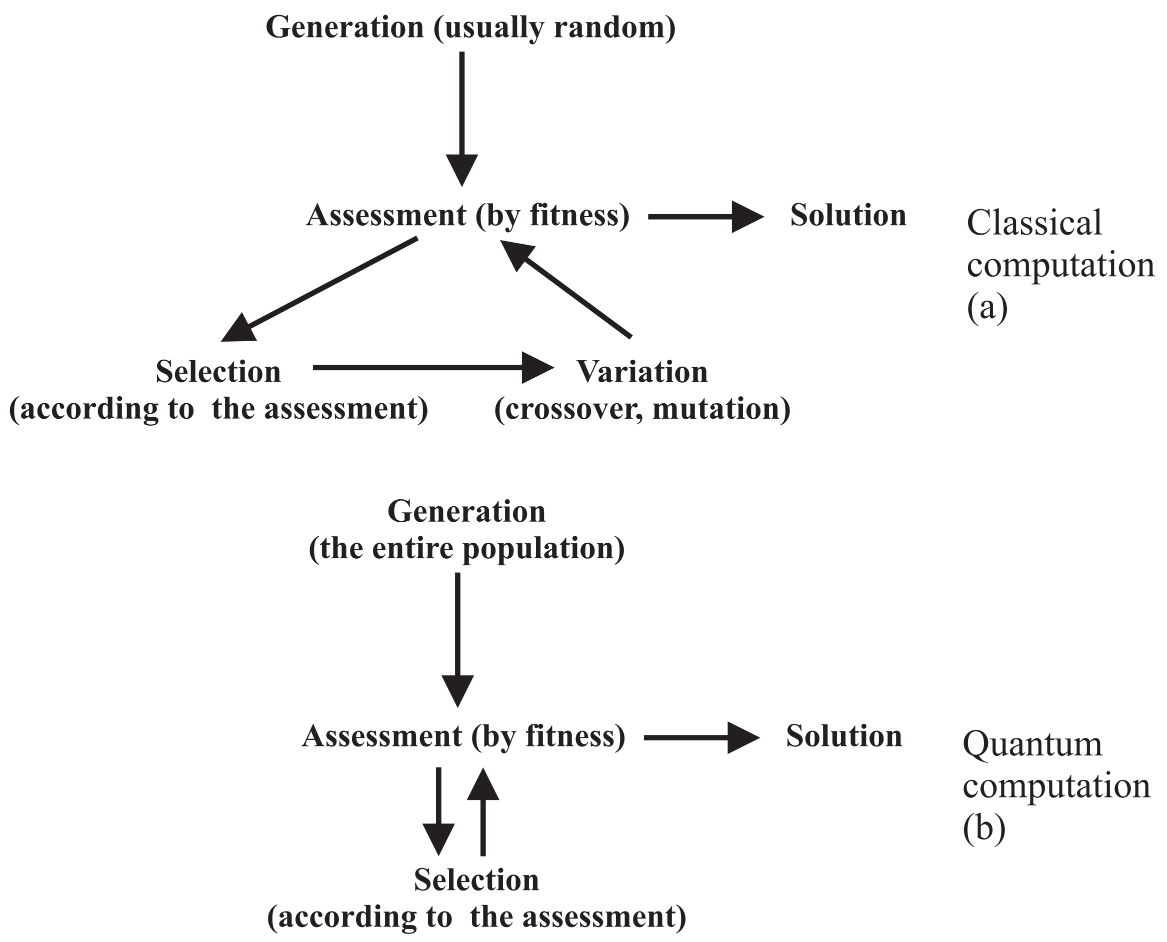

Generation (usually random)
Assessment (by fitness)
Solution
Classical computation (a)
Selection (according to the assessment)
Variation (crossover, mutation)
Generation (the entire population)
Assessment (by fitness)
Solution
Quantum computation (b)
Selection (according to the assessment)

Figure 5.5: A comparison between the classical and quantum evolutionary (and genetic) algorithm strategy, inspired by [96].

from Figure 5.5 (a); this is the reason why we called the algorithm created according to this model "Reduced Quantum Genetic Algorithm".

5.7 Summary

This chapter 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 properties, 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 [2][28]. 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 genetic operators such as crossover and mutation.

We can conclude that the search strategy itself is different for QGAs in comparison with the GAs; in fact, it isnt really genetic anymore. The special Oracle and Grover iterations are performing m successive selection steps.

The fact that the complexity of the original quantum maximum finding algorithm is $\mathcal{O}(N)$ [2] and our proposed algorithm is an adaptation of that does not increase the number of steps, indicates that any GA may be performed on a Quantum Computer with polynomial efficiency. This consequence would broaden the area of computational problems where the quantum solutions outperform the classical ones, and can also be counterbalance to the rising skepticism that regards the effectiveness of Grover's search [116]. However, a thorough complexity analysis is required because, unlike the initial algorithm, our proposal works with uneven quantum amplitudes in the search register.

Chapter 6

Conclusions

The new computational paradigms are subject for discussion within the computer science community for a long time. Nevertheless, computer engineering has approached this subject for some years, and a quite interesting debate has started, questioning whether Moore's law is obsolete or not. Regardless of the individual position and commitments involved in this debate, the general conclusion is that the quest for new technologies is necessary in order to achieve significant progress in building computational devices.

Because the new technology research efforts have soared, there is also commitment towards concentrating these efforts on the most important issues. As already mentioned in the first chapter of this thesis, the International Technology Roadmap for Semiconductors [126] is the worldwide recognized document that defines the guidelines in order to achieve these goals.

This thesis acknowledges the objectives defined by the ITRS, and aims at contributing to developing CAD tools for designing quantum circuits – on one hand – and to designing fault tolerance quantum algorithms and methodologies – on the other hand. Figures 6.1 and 6.2 present the ITRS perspective on emerging technologies, including quantum computation, along with a illustrative comparison with the classical CMOS technology.

As a reminder, we will present the main directions of this thesis:

- $\Delta 1$) Runtime-efficient quantum circuits simulation;
- $\Delta 2$) Design of fault-tolerant quantum circuits;
- $\Delta 3$) Implementing evolvable quantum hardware;

The main motivating idea that underpins the whole thesis structure is to approach the quantum computation field in a computer engineering fashion, by taking advantage of the already known methodologies and tools developed in classical computer hardware design automation, CAD, reconfigurable computing, and evolvable hardware.

This does dot mean that the proposed techniques are just a quantum counterpart for the classical ones; where the simple quantum adaptation is not appropriate, the necessary explanations are provided along with emphasizing the distinctive characteristics of quantum computation.

This chapter will summarize the contributions of this thesis, by presenting the way goals stated in Section 1.2 were approached from 3 points of view: the relevance for the ITRS document, the original contributions, and what is to be done in the future with the proposed solutions.

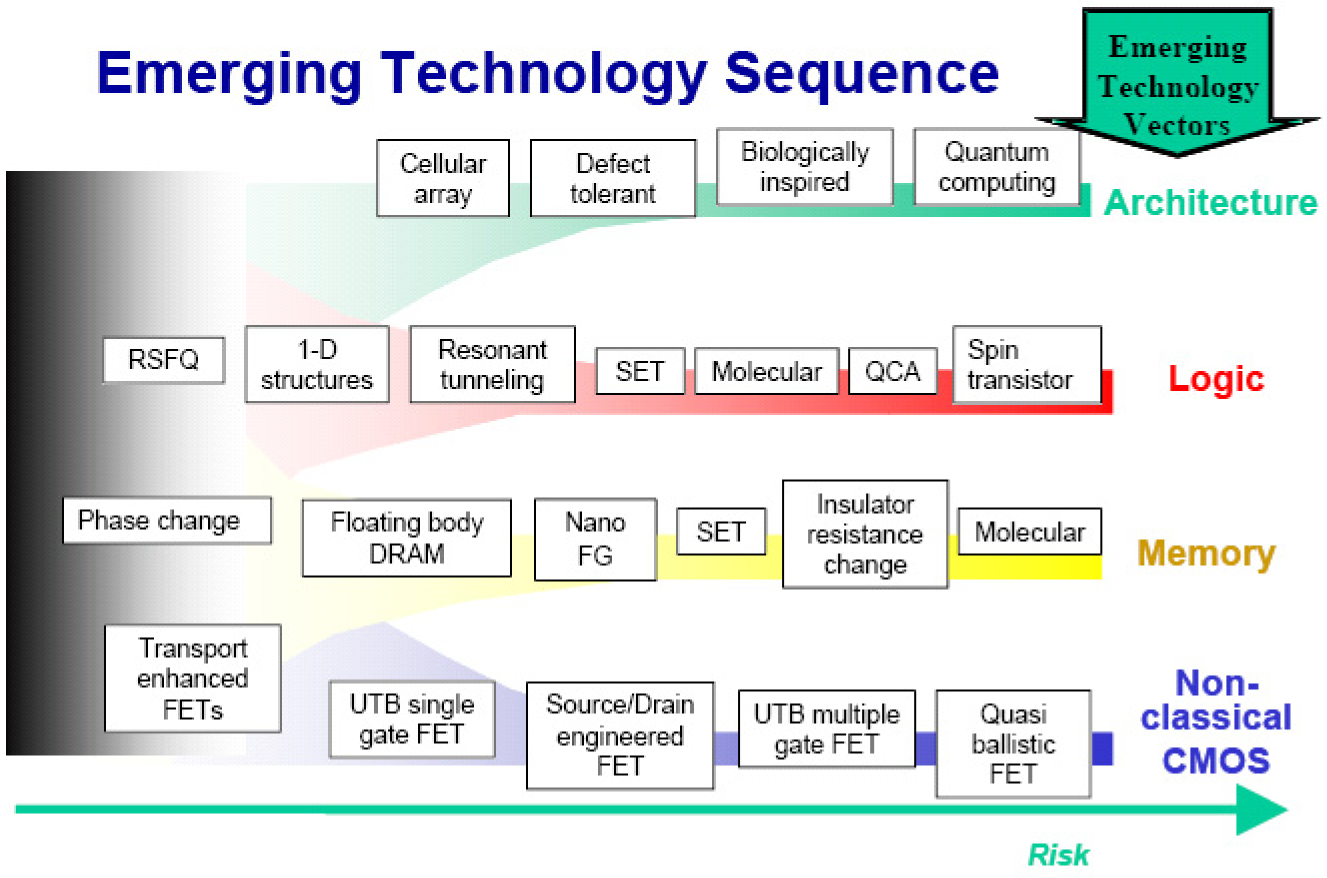


Figure 6.1: Emergent technology sequence, according to the Emerging Research Devices, within the ITRS [126].

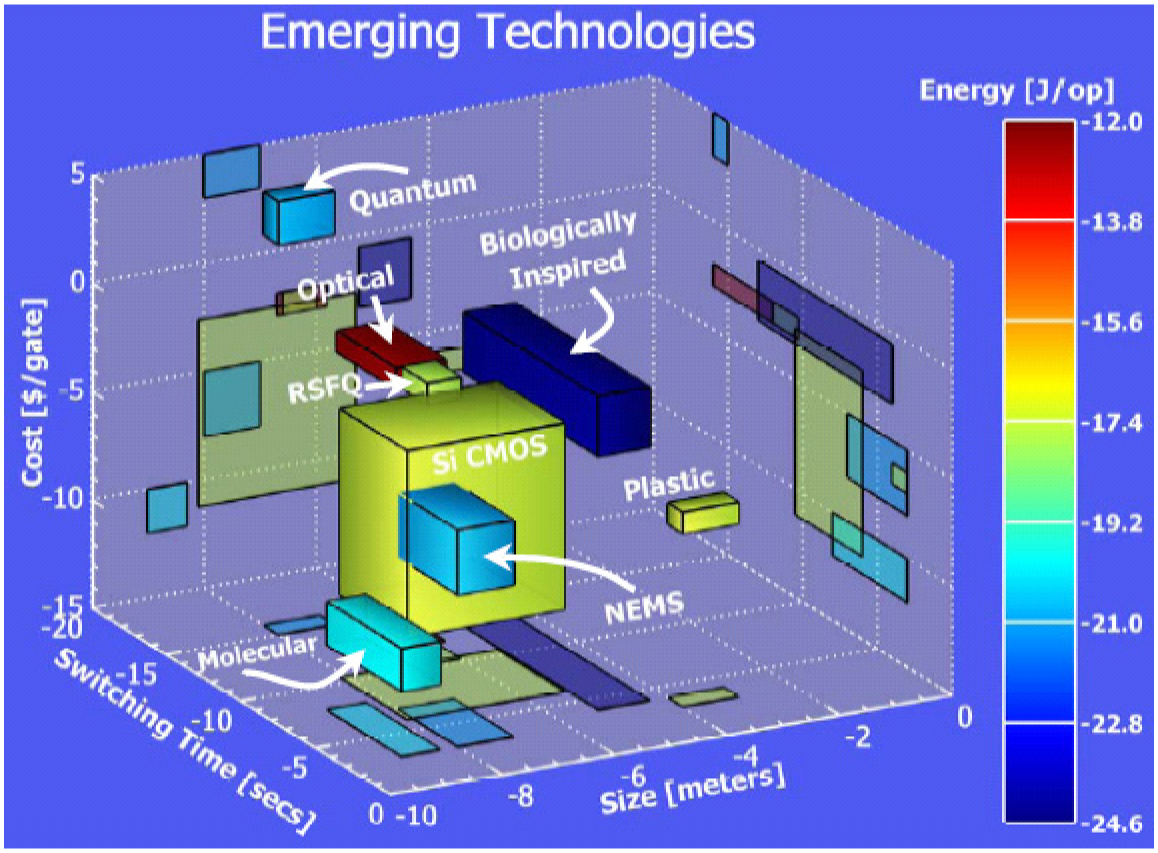


Figure 6.2: The parametric comparison between new technologies and CMOS – with respect to speed, size, cost, and energy consumption – according to ITRS [126].

6.1 Thesis relevance

6.1.1 $\Delta 1$ – Simulation

In the Emerging Research Devices (ERD) ITRS document [126], simulation is considered as playing a "key role (...) in characterizing the role of ERM (Emerging Research materials)". Because we are in the early stages of developing these new technologies, simulation cannot be very accurate, but it can provide some valuable quantitative assessment means [126].

Some empirical and systematic modeling and simulation methodologies are required at the physical level, in order to deal with metrology needed for the assessments [126]. Although this thesis is not concerned with this type of simulations, the results from the physical level are of great importance at the unitary level in quantum computation, especially when it's about fault injection with the fault abstract models, and error models taken into consideration for the QUERIST project.

The next subsection deals with the quantum fault tolerance imperatives, as described by ITRS documents referring to ERD and ERM; but in order to assess the effectiveness of the quantum FTAMs there has to be some simulation tool dedicated to that purpose. The guidelines for the QUERIST project are intended to satisfy these needs.

6.1.2 $\Delta 2$ – Fault tolerance

In reference [126] (pages 37 - 41) the need for fault tolerant ("coherent") quantum computation is defined and characterized. The main enemy for all the technologies used (QED, trapped ions, and solid state semiconductors and superconductors) is the decoherence phenomenon. ITRS states, as we have already presented in Chapter 4, that the existing error correcting strategies are extremely costly.

The fundamental issue is that classical approaches are used in order to fix massively parallel and complex quantum state decoherence problems. Our rQGA and rQHW approaches use a quantum-nature phenomenon (the superposed ECC circuits) in order to deal with the source of quantum ECC inefficiency (unsafe recovery).

The fact that fault tolerance is a vital aspect for quantum computation is emphasized by the fact that Table 64 from [126] urges the quest for efficient and lower-cost fault tolerance algorithms and methodologies (FTAMs). The same table also points to a very important aspect which is still opened to research solutions: quantum circuit testing is not possible directly.

6.1.3 $\Delta 3$ – Evolvable quantum hardware

This thesis direction has a special significance with respect to ITRS specifications, because it merges two ERD architecture implementations (as presented in Table 64 from the [126] document): "Biologically Inspired Implementations" and "Coherent Quantum Computation".

Although this very idea was already brought up by others [33][96], this thesis proposes an original view of this problem and provides the circuit implementation details.

6.2 Contributions

This section will enumerate the original contributions brought by this thesis, linked to the 3 main directions as pointed in the prologue of the Chapter 6. The introductory first chapter provided the motivation for the 3 directions ($\Delta 1$, $\Delta 2$, and $\Delta 3$) by presenting the thesis goals.

6.2.1 $\Delta 1$ – Simulation

Simulation of quantum computational systems is usually exponential. The quantum circuits make no difference to this rule. Reducing the inner complexity of quantum circuit simulation may be a prerequisite in approaching CAD and EDA (Electronic Design Automation) techniques to quantum circuit design but, on the other hand, the polynomial simulation of quantum systems is not a realistic goal. If a classical computer is able to simulate a quantum computer in polynomial time, then there is no point in building a quantum computer in the first place. However, it was demonstrated that quantum computation is much powerful than classical computation from the complexity point of view [24].

Nevertheless, our quest is facilitated by the fact that there are few useful quantum algorithms at this time, and therefore the involved quantum states will exhibit some *a priori* known patterns. These features that can be capitalized on are also referred as *simulation shortcuts* [96].

This thesis proposes a HDL-based approach of quantum circuit simulation, which is appropriate in the sense that it can be used for isolating the very source of simulation complexity – the entanglement phenomenon [105][106]. However, it was shown that this approach would not be effective unless some sort of quantum state encoding is provided [106][107].

This encoding – used to facilitate effective quantum circuit simulation by providing incentive for structural quantum circuit description even in the presence of entanglement – comes under the form of the so-called bubble bit technique [107]. The experimental results show a better runtime performance of our technique in comparison with the state-of-the-art. The drawback consists of the memory overhead (polynomial with the number of qubits in the state) dictated by the bubble records produced by the bubble bit insertion algorithm. Another advantage is that the bubble bit technique can be easily adapted in order to allow fault injection [108].

Summarizing, the original contributions of this thesis in simulating quantum circuits are:

- a new, HDL-based, and entanglement-aware perspective on simulation of quantum circuits;
- a case study (concerning the well-known quantum algorithms: Shor, Grover, Deutsch-Jozsa) aimed at defining the entanglemet role in quantum circuit simulation complexity [123];
- the bubble-bit encoding technique, which facilitates structural (therefore, polynomial) simulation of quantum circuits;
- the guidlines for the QUERIST project, a tool that is designed to allow quantum fault injections in order to assess the effectiveness of the quantum FTAMs.

These achievements come with the following advantages:

- the simulation runtimes are significantly improved for Deutsch-Jozsa, Grover, and arithmetic circuits involved in Shor's algorithm;
- using the HDLs for quantum circuits description and simulation brings quantum computation closer to design automation and future computer-aided design and test techniques;
- the bubble-bit technique is easily adaptable for simulation fault injection purposes.

The drawbacks, as compared with other gate-level simulation techniques, are:

- the memory overhead dictated by the bubble-bit encoding technique;
- the fact that the unitary transformations (which define the quantum gate actions) are also encoded.

6.2.2 $\Delta 2$ – Fault tolerance

Fault tolerance techniques are vital in quantum computation; therefore any progress in this area has an important significance. Due to the nature of quantum computation, and the nature of the errors that are affecting quantum circuits, detecting and correcting errors is difficult.

Due to the specific nature of quantum mechanics, implementing error correcting methods in quantum circuits is confronted with a series of limitations and problems, which were surpassed by ingenuous quantum circuit design techniques [76]. However, because of the high fault rate in quantum computation, the problem can also be represented by the *safe recovery*.

The result is that arbitrary long fault tolerant quantum computation is not possible with only the error correcting circuits. Therefore, besides appropriate coding (Steane-based codes [76]) the state-of-the-art also relies on concatenated coding. However, as shown in this thesis, the concatenated coding can be useless if the errors affecting the circuit are correlated.

Our solution [109] is based on designing the so-called reconfigurable (or programmable) Quantum Gate Arrays (rQGA), as an application of the reconfigurable Quantum Hardware (rQHW) concept. The reconfigurable quantum circuit has a quantum-nature configuration register (i.e. superposition of states) which configures the rQGA as a superposition of distinct error-correcting circuits. Only one of these circuits remains after measuring the configuration register, in order to be used for the actual correction. However, the probability of an error occurring in the measured circuit decreases exponentially with the number of actual superposed circuits.

Practical circuit implementation brings other problems to be solved, by further complicating the issues. However, as shown in our analytical study, the proposed solution drastically improves the accuracy threshold. Moreover, the circuit used for generating the configuration state is simplified, thus dealing with the problem of preserving an accurate configuration.

Summarizing, the advantages brought by our solution are:

- it solves the problem of safe recovery in quantum error-correcting circuits;
- offers a solution of replacing the concatenated coding technique (prone to fail in the presence of correlated errors), so that arbitrary long faul tolerant quantum computation is preserved;
- in fundamental terms, it brings a solution that exploits the exponential parallelism of quantum computation in order to achieve dependability.

The limitations consists of the following issues:

- some gates are used by more than one superposed correcting circuit, thus an error affecting that gate will affect more than one correcting circuit;
- the configuration state must be maintained accurately.

Even with the above-mentioned limitations, the main quantum circuit reliability measure (i.e. the accuracy threshold) is significantly improved, with respect to the state-of-the-art [47][48].

6.2.3 $\Delta 3$ – Evolvable quantum hardware

This direction provides a solution within the computer-engineering context (i.e. is presenting a circuit implementation of our approach) but its possible consequences transcend this field – it concerns the computer science area.

The initial goal was to develop a quantum counterpart for the versatile and robust evolvable hardware concept. In classical computation EHW = RHW + GA (evolvable hardware = reconfigurable hardware + genetic algorithms), hence our objective can be reduced to designing a quantum computation architecture for implementing genetic algorithms.

Again, the attempt to run GAs on quantum computers is not new [33], but the previous approaches were blocked when trying to build quantum counterparts for genetic operators [119] such as crossover. Instead, our proposed view does not require crossover and mutation. As a matter of fact, the selection mechanism consist of marking the "better than current" partial solution individuals (with a specially designed quantum oracle [62]) and then finding one of them with Grover's algorithm used as means for augmenting the amplitude of the marked states.

Another engineering *ad-hoc* decision was to define a special kind of fitness function, which will return values even for non-valid individuals. The fitness values assigned to non-valid individuals will clearly identify them. Therefore, one quantum register can be used to encode all the possible individuals as superposed basis states. The individual with the best fitness will emerge by using and adaptation of the already published maximum finding algorithm [2].

Even if the initial maximum finding algorithm has a $\mathcal{O}(\sqrt{n})$ complexity, we cannot conclude that this also the complexity of running GAs on the quantum computer, because the algorithm from [2] deals with basis states that have all the same amplitude. Nevertheless, even if further investigation is necessary in order to establish the exact complexity of the proposed Reduced Quantum Genetic Algorithm (RQGA), the extra Grover steps used for extra-amplitude-augmentation will be no more than $\mathcal{O}(\sqrt{n})$. This means that the overall complexity cannot be more than $\mathcal{O}(n)$.

As a summary, we present the main achievement of this thesis' third direction:

- for the first time, a comprehensive QGA (Quantum Genetic Algorithm) implementation is presented;
- it is shown that the crossover and mutation operators are not necessary;
- the selection is made by means of specific quantum computation features, reducing the selection process to Grover's algorithm working with a specially-designed quantum oracle.

The most important consequence of the proposed QGA implemetation is that QGAs will outperform the classical implementations, with the observation that the exact complexity of our algorithm (RQGA) is still to be assessed.

6.3 Future work

This thesis has addressed an emerging subject, the design of reliable quantum circuits. We are still quite far from actually developing such devices on a commercial scale, but the academia and the industry are preoccupied with the possibility of encountering such problems.

However, even if we will got to a point where the manufacturability of such devices becomes possible, design automation and CAD techniques will encounter the fundamental problem of exponential-time simulation for quantum circuits (and quantum systems in general). Of course this drawback must be dealt with in advance, in order not to affect the scalability of the automated designs.

Another aspect consists of dealing with the reliability issues in quantum circuitry. Without specialized algorithms and methodologies used for mitigating the destructive effect of decoherence in quantum states, there could be no realistic prospect of quantum computation.

This thesis tries to deal with all these problems, by proposing a computer engineering view on the field of quantum computation. It is inevitable, however, that this thesis covers a lot of other issues from physics, computer science, placing itself at the frontiers of computation research.

It comes as obvious that this thesis is one of the many efforts to open a new highway on computer engineering research. A lot is still to be done on all of the approached issues, but this thesis underpins the idea that HDLs can be used successfully in simulating quantum circuits and in quantum simulated fault injection. These elements must be subsequently used by automated quantum circuit design techniques.

In the field of quantum reliability, this thesis shows the fact that reconfigurable (or programmable) arrays of gates could offer the ideal platform for building robust error-correcting circuits. Also, genetic algorithms can program the reconfigurable quantum hardware, thus incentives for developing evolvable quantum hardware are created. As a consequence, we bring together two very hot directions in emerging technology research. Still, a lot is to be done, including the development of intrinsic fault tolerant quantum devices, innovative error correcting encoding, and finding new applications for the QEHW concept.

We will now summarize what we think that is to be pursued in this area, by having the present thesis as one of the emergence points.

6.3.1 $\Delta 1$ – Simulation

- Exploring the possibility of improving the bubble-bit encoding method, in order to reduce the bubble record burden.
- Unitary-level automated synthesis [120] [121] [125] for our developed HDL-based simulation framework.
- Integrating simulated fault injection technique, which has been developed in this thesis, in the design process.
- Approaching system-level design in quantum computation.
- Using simulated fault injection for developing fault-tolerant architectures at the system level.

6.3.2 $\Delta 2$ – Fault tolerance

- Extending the rQGA technique for safe recovery, so that it will become suitable for more quantum error-correcting codes.
- Developing reconfiguration techniques in order to avoid fighting with correlated errors.
- Settling the issue of concatenated coding necessity, when the rQGA technique is applied.
- Finding the most suitable quantum technology for the configuration register and for the rQGA structure itself.

6.3.3 $\Delta 3$ – Evolvable quantum hardware

- Finding the exact complexity for the Quantum Genetic Algorithms.
- Finding applications for the Evolvable Quantum Hardware Concept.

- Exploring the possibility of developing Quantum Cellular Automata (QCA) on EQHW platforms.

Appendix A

VHDL Description of Elementary Quantum Gates

In order to describe the simple gates, a data structure must be built so that we can represent qubits and quantum registers according to the matrix model. We have reduced the data package to the necessary. Also, the example VHDL code fragments provided in this appendix are stripped from any irrelevant statements or instructions.

The data structure

```
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
package quantum is
type qubit is array(0 to 1) of complex;
type quregister is array(natural range<>)of complex;
type qubit_vector is array(natural range<>)of qubit;
end quantum;
```

Another useful feature would be a function for transforming a register quantum state into a new state, including an extra-qubit:

```
function tensor_product_1(reg:quregister;qb:qubit)return quregister is
variable newreg:quregister;
variable half,odd:integer;
begin
assert power_2(reg'length)
report "not a valid quregister state"
severity error;
l1:for i in 0 to (2* reg'length)-1 loop
half:=i/2;
odd:=i-(2*half);
newreg(i):=reg(half)*qubit(odd);
end loop l1;
return newreg;
end tensor_product_1;
```

This function can be included in the above package. Here, the function `power_2` returns true if its integer argument is a power of 2, and false if it isn't.

A.1 Hadamard gate

The representation and the characterizing matrix for this gate are presented in Figure A.1. The describing VHDL code follows:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Figure A.1: Hadamard gate representation and corresponding unitary matrix.

```
entity walsh_hadamard_gate is
generic(delay:time);
port(intrare:in qubit; iesire:out qubit);
end walsh_hadamard_gate;
architecture whg_a of walsh_hadamard_gate is
begin
iesire(0)<= (1.00/sqrt(2.00))*(intrare(0)+intrare(1)) after delay;
iesire(1)<= (1.00/sqrt(2.00))*(intrare(0)-intrare(1)) after delay;
end whg_a;
```

A.2 Negation gate

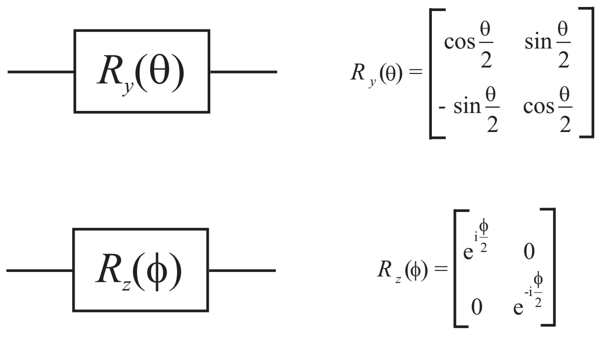
$$N = \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Figure A.2: Negation gate representation and corresponding unitary matrix.

Figure A.2 presents the symbol used for this gate along with the corresponding gate. The describing VHDL code is:

```
entity not_gate is
generic(delay:time);
port(intrare:in qubit; iesire:out qubit);
end not_gate;
architecture ng_a of not_gate is
begin
iesire(0)<= intrare(1) after delay;
iesire(1)<= intrare(0) after delay;
end ng_a;
```

A.3 Rotation gate



$R_y(\theta)$
$R_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & \sin\frac{\theta}{2} \\ -\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$
$R_z(\phi)$
$R_z(\phi) = \begin{bmatrix} e^{i\frac{\phi}{2}} & 0 \\ 0 & e^{-i\frac{\phi}{2}} \end{bmatrix}$

Figure A.3: Rotation gates representation with corresponding unitary matrixes.

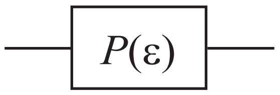
Figure A.3 presents the symbol used for this gates along with the corresponding gate. The rotation gate is a generalization of Haddamard gate, operating only on a single qubit. $R_y(\theta)$ is the rotation with θ around axis $\hat{y}$, while $R_z(\phi)$ is the rotation with ϕ around axis $\hat{z}$ The appropriate VHDL code is:

```
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity y_rotation_gate is
generic(theta:real;delay:time);
port(intrare:in qubit; iesire:out qubit);
end y_rotation_gate;
architecture yrg_a of y_rotation_gate is
begin
process(intrare)
variable sinus,cosinus:real;
begin
sinus:=sin(theta/2);
cosinus:=cos(theta/2);
iesire(0)<=cosinus*intrare(0)+sinus*intrare(1) after delay;
iesire(1)<=cosinus*intrare(1)-sinus*intrare(0) after delay;
end process;
end yrg_a;
entity z_rotation_gate is
generic(phi:principal_value;delay:time);
port(intrare:in qubit; iesire:out qubit);
end z_rotation_gate;
architecture zrg_a of z_rotation_gate is
begin
process(intrare)
variable element0,element1:complex_polar;
begin
```

```
element0.arg:=phi/2;element0.mag:=1;
element1.arg:=-(phi/2);element1.mag:=1;
iesire(0)<=element0*intrare(0) after delay;
iesire(1)<=element1*intrare(1) after delay;
end process;
end zrg_a;
```

A.4 Conditional phase-shift gate

The conditional phase shift gate is a gate that could operate on more than one qubit as presented Figure A.4. Besides the matrix expression, in this figure will appear the corresponding symbol. The VHDL description of this n-qubit gate must take into account the fact that the state of the n qubits is not always describable as the tensor product of the individual qubit states. Therefore, the simplest way to simulate this gate is to have as input and output the overall input and output states (type `quregister`):



$$P(\varepsilon) = \begin{bmatrix} 1 & 0 & 0 & \cdots & & 0 \\ 0 & 1 & 0 & & & \\ 0 & 0 & 1 & & & \vdots \\ \vdots & & & \ddots & & \\ & & & & 1 & 0 \\ 0 & & \cdots & & 0 & e^{i\varepsilon} \end{bmatrix}$$

Figure A.4: A n-qubit conditional phase-shift gate representation, with the $2^n \times 2^n$-size corresponding unitary matrix.

```
entity phase_shift is
generic(epsilon:real;delay:time);
port(intrare:in quregister; iesire:out quregister);
end phase_shift;
architecture ps_a of phase_shift is
begin
process(intrare)
variable temp:quregister;
variable tmp,phase:complex_polar;
variable lg:integer;
begin
lg:=intrare'length;
l1:for i in 0 to lg-2 loop
temp(i):=intrare(i);
end loop l1;
phase.mag:=1;phase.arg:=epsilon;
tmp:=complex_to_polar(intrare(lg-1))*phase;
temp(lg-1):=polar_to_complex(tmp);
iesire<=temp after delay;
```

```
end process;
end ps_a;
```

A.5 XOR and Toffoli gates

The XOR and Toffoli gates are in the same unitary gate class $CNOT_{n-qubit}$, where $XOR = CNOT_{2-qubit}$ and $TOFF = CNOT_{3-qubit}$. Figure A.5 a,b, and c, present the XOR, $TOFF$ and the general $CNOT_{n-qubit}$ gates.

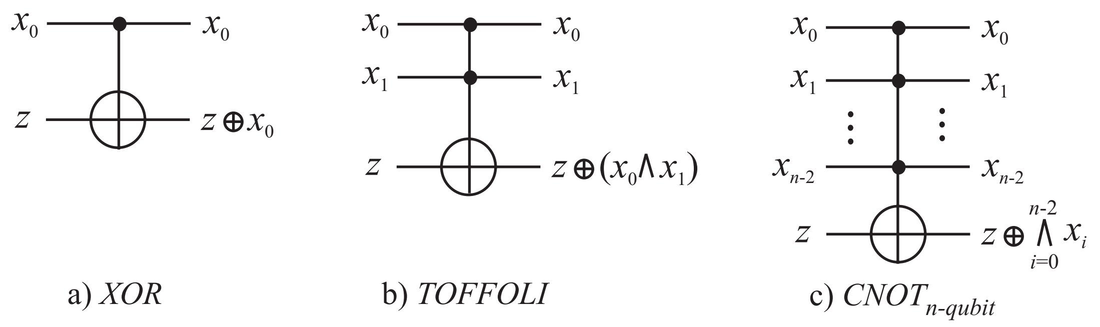


Figure A.5: CNOT gates: a) XOR; b) $TOFFOLI$; c) the general $CNOT$ operating on n qubits.

Then, the description for the general n-qubit $CNOT$ gate is:

```
entity c_not is
generic(delay:time);
port(intrare:in quregister; iesire:out quregister);
end c_not;
architecture cnot_a of c_not is
begin
process(intrare)
variable lg:integer;
variable temp:quregister;
begin
lg:= intrare'length;
assert lg-1 > 1
report "not a valid gate"
severity error;
l1:for i in 0 to (intrare'length)-1 loop
if i < lg-2
temp(i):=intrare(i);
elsif i=lg-2 then
temp(i):=intrare(i+1);
elsif i=lg-1 then
temp(i):=intrare(i-1);
end if;
end loop l1;
```

```
iesire<=temp after delay;
end process;
end cnot_a;
```

A.6 Swap gate

The *SWAP* gate is operating over 2 qubits. Its matrix expression along with the symbol are given in Figure A.6. The VHDL code follows:

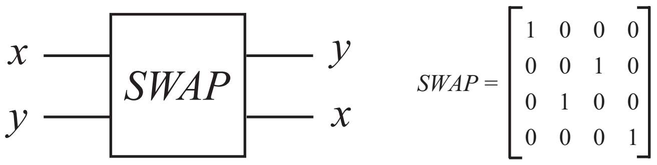


Figure A.6: The symbol and corresponding unitary matrix for the *SWAP* gate.

```
entity swap_gate is
generic(delay:time);
port(intrare:in quregister; iesire:out quregister);
end swap_gate;
architecture arh_sw of swap_gate is
begin
process(intrare)
variable temp:quregister;
variable lg:integer;
begin
lg:= intrare'length;
assert lg /= 4
report "not a valid swap gate"
severity error;
temp(0):=intrare(0);temp(3):=intrare(3);
temp(1):=intrare(2);temp(2):=intrare(1);
iesire<=temp after delay;
end process;
end arh_sw;
```

A.7 Quantum adders

The quantum adder, operating over a quantum register, is a composition of pseudo-classical operators. A n-qubit quantum adder or qadder could be decomposed in n 1-qubit full qadders. The unitary expression of the 1-qubit qadder is

$$SUM_{1-qubit} : |x, y, c_{in}, 0\rangle \longrightarrow |x, y, x \oplus y \oplus c_{in}, x \cdot y + x \cdot c_{in} + y \cdot c_{in}\rangle \tag{A.1}$$

based on the equations of the classical 1-bit full adder [67][71].

Since we are forced to use only valid quantum pseudo-classical operators, the unitary diagram of the 1-qubit quadder cell is the quantum network from Figure A.7.

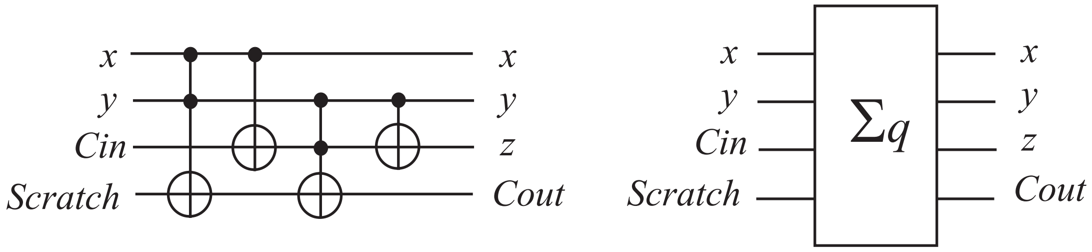


Figure A.7: 1-Qubit full quadder, implemented with unitary gates XOR and $TOFF$ [63][113].

The way to get z is straightforward, $z = x \oplus y \oplus c_{in}$, but getting c_{out} is much tricky. Therefore, we must rewrite the classical c_{out} expression so that it could be implemented with $TOFF$ and XOR gates:

$$\begin{aligned} c_{out} &= xy + xc_{in} + yc_{in} \\ &= xyc_{in} + xy\overline{c_{in}} + xc_{in} + xyc_{in} + \overline{x}yc_{in} \\ &= y\left(x \oplus c_{in}\right) + xc_{in} + xyc_{in} \\ &= y\left(x \oplus c_{in}\right) + xc_{in} \\ &= \left[y\left(x \oplus c_{in}\right)\right] \oplus c_{in} \end{aligned} \tag{A.2}$$

because

$$\begin{aligned} ab + b = a \\ a + b = (a \oplus b) + ab \end{aligned} \tag{A.3}$$

and

$$\overline{xc_{in}y\left(x \oplus c_{in}\right)} = \overline{xc_{in}\overline{x}c_{in} + xc_{in}yx\overline{c_{in}}} = \overline{0+0} = 1. \tag{A.4}$$

Thus, the VHDL description is (entanglement is considered absent):

```
entity add_cell is
port(x,y,cin,scratch:in qubit;xo,yo,z,cout:out qubit);
end add_cell;
architecture structural of add_cell is
component xor_gate
generic(delay:time);
port(a,b:in qubit;a,rez:out qubit);
end component;
component toffoli_gate
generic(delay:time);
port(a,b,c:in qubit;a,b,rez:out qubit);
end component;
signal t1,t2:qubit;
begin
c1:toffoli_gate
generic map(delay=>10 ns);
port map(a=>x,b=>y,c=>scratch,rez=>t2);
c2:xor_gate
```

```
generic map(delay=>10 ns);
port map(a=>x,b=>cin,rez=>t1);
c3:toffoli_gate
generic map(delay=>10 ns);
port map(a=>y,b=>t1,c=>t2,rez=>cout);
c4:xor_gate
generic map(delay=>10 ns);
port map(a=>y,b=>t1,rez=>z);
xo<=x;yo<=y;
end structural;
```

The typical way to operate for the quantum adder is making use of its ability to sum all the possible superposed input states. In our case, for the 1-qubit adder from Figure A.7, qubits x and y could be prepared to represent the superposition state of the 4 classical distinct states (eigenvectors in a $\mathcal{H}^4$ Hilbert space): $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. In this situation (Figure A.8), carry in is $|1\rangle$ and each superposed classical state contains the input qubit values and the corresponding output bit values. When measuring the output state, the outcome will be one of the superposed input classical states, with the corresponding output values (see Figure A.8 for details).

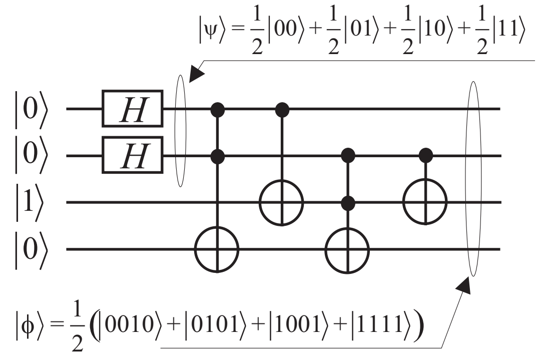

$|\psi\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$
$|0\rangle$
$|0\rangle$
$|1\rangle$
$|0\rangle$
H
H
$|\phi\rangle = \frac{1}{2}(|0010\rangle + |0101\rangle + |1001\rangle + |1111\rangle)$

Figure A.8: A 1-qubit full quadder operating over a superposition of possible classical states for inputs x and y. Carry in is $|1\rangle$ in this example, $|\psi\rangle$ is the state of inputs x and y ($|x, y\rangle$) while $|\phi\rangle$ is the state of the 4-qubit output.

The way that 1-qubit full quadder cells could be combined so that an n-qubit qadder is obtained is presented in Figures A.9 and A.10.

The corresponding code, which is valid in the absence of entanglement, is:

```
entity adder_4 is
port(x,y,scratch:in qubit_vector(0 to 3);cin:in qubit;
xx,yy,z:out qubit_vector(0 to 3);cout:out qubit);
end adder_4;
architecture struct of adder_4 is
component add_cell
port(xi,yi,ci,scr:in qubit;xo,yo,zo,co:out qubit);
end add_cell;
constant zero:qubit:=((1,0),(0,0));
signal carry:qubit_vector(0 to 3);
```

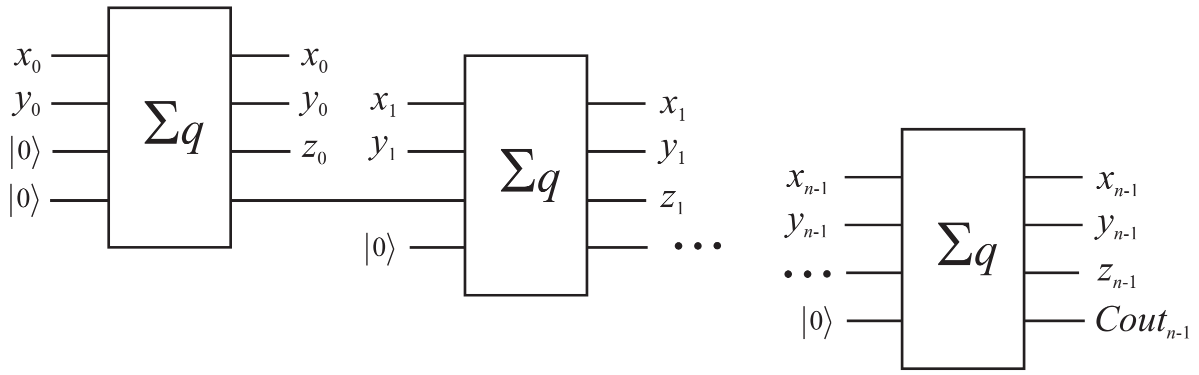


Figure A.9: An n-qubit adder obtained by rippling the carry. Note that we need n scratch qubits for completing this operation.

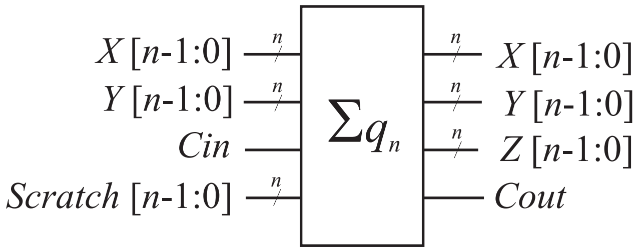


Figure A.10: The symbol for the n-qubit, used in quantum networks, with its details contained in Figure A.9.

```
begin
u1:add_cell
port map(xi=>x(0),yi=>y(0),ci=>zero,scr=>zero,
xo=>xx(0),yo<=yy(0),zo<=z(0),
co=>carry(0));
u2:add_cell
port map(xi=>x(1),yi=>y(1),ci=>carry(0),scr=>zero,
xo=>xx(1),yo<=yy(1),zo<=z(1),
co=>carry(1));
u3:add_cell
port map(xi=>x(2),yi=>y(2),ci=>carry(1),scr=>zero,
xo=>xx(2),yo<=yy(2),zo<=z(2),
co=>carry(2));
u4:add_cell
port map(xi=>x(3),yi=>y(3),ci=>carry(2),scr=>zero,
xo=>xx(3),yo<=yy(3),zo<=z(3),
co=>carry(3));
cout<=carry(3);
end struct;
```

Appendix B

Deutsch-Jozsa Algorithm Simulation

The VHDL description of a circuit implementing Deutsch's algorithm is contained in "deutsch.vhd":

```
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
package quantum is
type qubit is array(0 to 1) of complex;
type quregister is array(natural range<>)of complex;
end quantum;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
use work.quantum.all;
entity deutsch is
port (xi,yi:in qubit;f0,f1:in bit; rez:out bit);
end deutsch;
architecture structural of deutsch is
component walsh_hadamard_gate
port(intrare:in qubit;iesire:out qubit);
end component;
component walsh_hadamard_phase
port(intrare:in qubit;iesire:out qubit);
end component;
component ufd_gate
port(xii,yii:in qubit;fd0,fd1:in bit;xoo,yoo:out qubit);
end component;
component measure_phase
port(intrare:in qubit;vector_baza:out bit);
end component;
signal x_in,y_in,qx1,qy1,qx2,qy2:qubit;
signal fct0,fct1:bit;
begin
c1:  walsh_hadamard_gate
port map(x_in,qx1);
c2:  walsh_hadamard_gate
```

```
port map(y_in,qy1);
c3:  ufd_gate
port map(qx1,qy1,fct0,fct1,qx2,qy2);
c4:  measure_phase
port map(qx2,rez);
x_in(0).im <= 0.00;
x_in(1).im <= 0.00;
x_in(0).re <= 1.00;
x_in(1).re <= 0.00;
y_in(0).im <= 0.00;
y_in(1).im <= 0.00;
y_in(0).re <= 0.00;
y_in(1).re <= 1.00;
fct0 <= '1';
fct1 <= '0';
end structural;
configuration cf_deutsch of deutsch is
for structural
end for;
end cf_deutsch;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
use work.quantum.all;
entity ufd_gate is
port(xii,yii:in qubit;fd0,fd1:in bit;xoo,yoo:out qubit);
end ufd_gate;
architecture b of ufd_gate is
begin
process (xii,yii)
variable a0,a1:real;
begin
if fd0='0' then a0:=1.00;
else a0:=-1.00;
end if;
if fd1='0' then a1:=1.00;
else a1:=-1.00;
end if;
xoo(0)<= (a0*xii(0)) after 10 ns;
xoo(1)<= (a1*xii(1)) after 10 ns;
yoo<=yii after 10 ns;
end process;
end b;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
use work.quantum.all;
entity measure_phase is
```

```
port(intrare:in qubit;vector_baza:out bit);
end measure_phase;
architecture m of measure_phase is
begin
process (intrare)
variable bal_eq:bit;
begin
if intrare(0).im=0.00 and intrare(1).im=0.00 then
if ((intrare(0).re)*(intrare(1).re)) >= 0.00 then
bal_eq:= '0';
else bal_eq:='1';
end if;
end if;
vector_baza <= bal_eq after 5 ns;
end process;
end m;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
use work.quantum.all;
entity walsh_hadamard_gate is
port(intrare:in qubit;iesire:out qubit);
end walsh_hadamard_gate;
architecture whg_a of walsh_hadamard_gate is
begin
iesire(0)<=(1.00/sqrt(2.00))*(intrare(0)+intrare(1))
after 10 ns;
iesire(1)<=(1.00/sqrt(2.00))*(intrare(0)-intrare(1))
after 10 ns;
end whg_a;
```

B.1 Time diagrams for VHDL simulation of Deutsch's algorithm

B.1.1 For a balanced oracle

The time diagram resulted from simulating the algorithm, for a balanced oracle, is presented in Figure B.1.

B.1.2 For a constant oracle

The time diagram resulted from simulating the algorithm, for a constant oracle, is presented in Figure B.2.

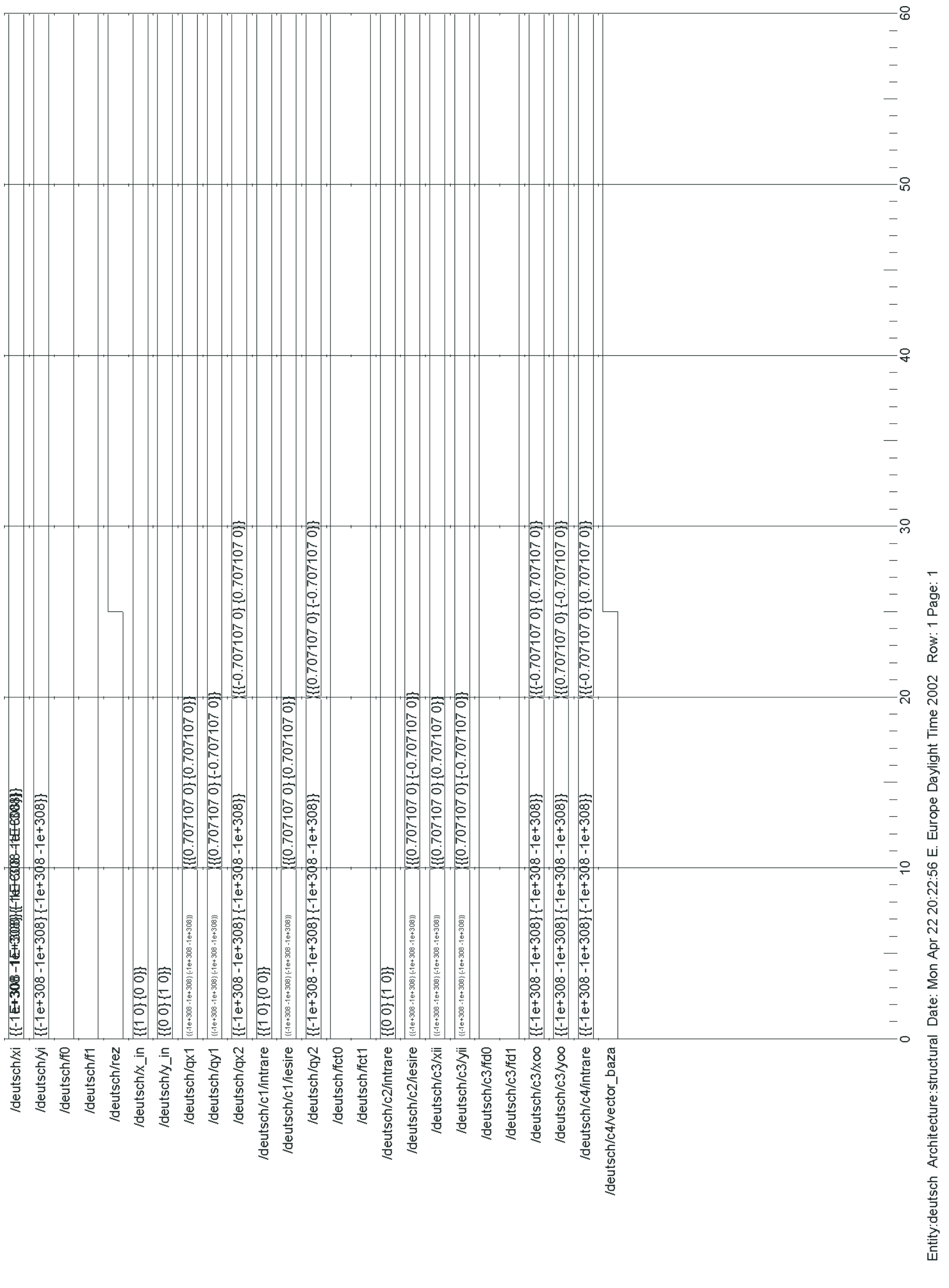


Figure B.1: Simulation results for Deutsch-Jozsa algorithm, when the oracle is a balanced function.

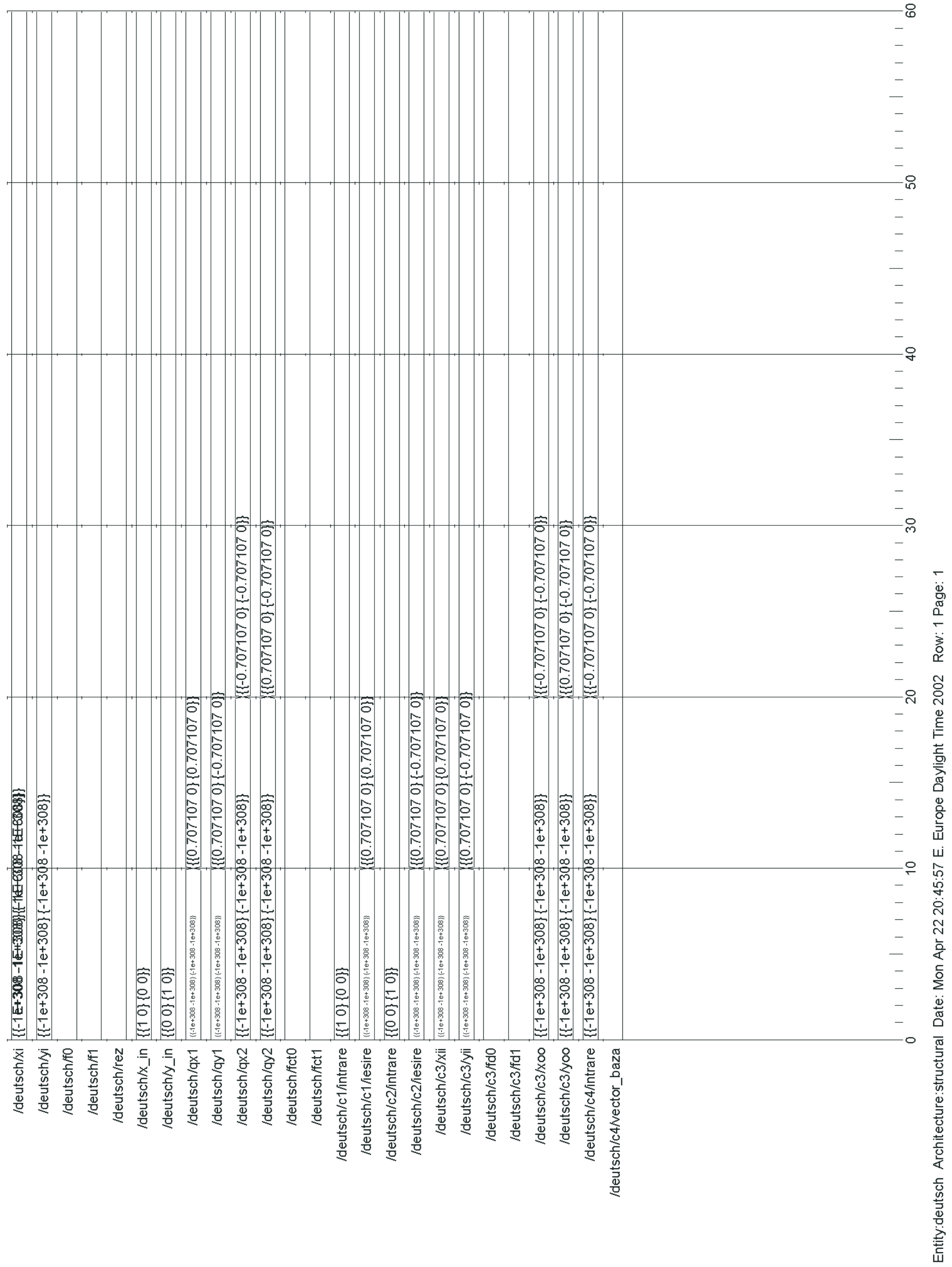


Figure B.2: Simulation results for Deutsch-Jozsa algorithm, when the oracle is a constant function.

Appendix C

Grover Algorithm Simulation (2-qubit querry)

```
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;
   -- 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;
   -- data type for simulation of 2-qubit circuits
   -- when ent=true we have entanglement and 'qr' field
   -- will be taken into consideration
   type qudata_2q is record
   qr:quregister(0 to 3);
   qa:qubit_vector(0 to 1);
   ent:boolean;
   end record;
   type qudata_3q is record
   qr:quregister(0 to 7);
   qa:qubit_vector(0 to 2);
   bub:bubble_record (0 to 1);
```

```
ent:boolean;
end record;
end qupack;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity grover_circ is
port(search_regi:in qudata_3q;ansi:in qubit;rez:out bit_vector(0 to 3));
end grover_circ;
architecture struct1 of grover_circ is
component level1
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component oracle
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component level2
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component phase_shift
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component level3
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component measurement
port(si:in qudata_3q;ai:in qubit; r:out bit_vector(0 to 3));
end component;
component ent_anal
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
signal ta,search_rego_t,search_rego_tt,search_rego_ttt,search_rego_tttt,
search_rego_ttttt:qudata_3q;
signal tata,anso_t,anso_tt,anso_ttt,anso_tttt,anso_ttttt:qubit;
begin
c1:level1 port map(search_regi,ansi,search_rego_t,anso_t);
c2:oracle port map(search_rego_t,anso_t,search_rego_tt,anso_tt);
c3:level2 port map(search_rego_tt,anso_tt,search_rego_ttt,anso_ttt);
c4:phase_shift port map(search_rego_ttt,anso_ttt,search_rego_tttt,anso_tttt);
c41:ent_anal port map(search_rego_tttt,anso_tttt,ta,tata);
c5:level3 port map(ta,tata,search_rego_ttttt,anso_ttttt);
c6:measurement port map(search_rego_ttttt,anso_ttttt,rez);
end struct1;
architecture struct_bub of grover_circ is
component level1
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component oracle_bub
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component level2_bub
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
```

```
component phase_shift_bub
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component level3_bub
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end component;
component measurement_bub
port(si:in qudata_3q;ai:in qubit; r:out bit_vector(0 to 3));
end component;
signal search_rego_t,search_rego_tt,search_rego_ttt,search_rego_tttt,
search_rego_ttttt:qudata_3q;
signal anso_t,anso_tt,anso_ttt,anso_tttt,anso_ttttt:qubit;
begin
c1:level1 port map(search_regi,ansi,search_rego_t,anso_t);
c2:oracle_bub port map(search_rego_t,anso_t,search_rego_tt,anso_tt);
c3:level2_bub port map(search_rego_tt,anso_tt,search_rego_ttt,anso_ttt);
c4:phase_shift_bub port map(search_rego_ttt,anso_ttt,search_rego_tttt,anso_tttt);
c5:level3_bub port map(search_rego_tttt,anso_tttt,search_rego_ttttt,anso_ttttt);
c6:measurement_bub port map(search_rego_ttttt,anso_ttttt,rez);
end struct_bub;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity level1 is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end level1;
architecture l1a_b of level1 is
begin
process (si,ai)
variable r:qudata_3q;
variable ti :  quregister(0 to 7);
variable ra:qubit;
variable ati :  quregister(0 to 1);
begin
l1:  for i in 0 to 7 loop ti(i):=si.qr(i);
end loop l1;
ati(0):=ai(0);ati(1):=ai(1);
r.qr(0):=(1.00/2.00)*(ti(0)+ti(2)+ti(4)+ti(6));
r.qr(1):=(1.00/2.00)*(ti(1)+ti(3)+ti(5)+ti(7));
r.qr(2):=(1.00/2.00)*(ti(0)-ti(2)+ti(4)-ti(6));
r.qr(3):=(1.00/2.00)*(ti(1)-ti(3)+ti(5)-ti(7));
r.qr(4):=(1.00/2.00)*(ti(0)+ti(2)-ti(4)-ti(6));
r.qr(5):=(1.00/2.00)*(ti(1)+ti(3)-ti(5)-ti(7));
r.qr(6):=(1.00/2.00)*(ti(0)-ti(2)-ti(4)+ti(6));
r.qr(7):=(1.00/2.00)*(ti(1)-ti(3)-ti(5)+ti(7));
r.qa:=si.qa;
ra(0):=(1.00/sqrt(2.00))*(ati(0)+ati(1));
ra(1):=(1.00/sqrt(2.00))*(ati(0)-ati(1));
r.ent:=true; r.bub:=si.bub;
so <= r after 10 ns; ao <= ra after 10 ns;
end process;
end l1a_b;
architecture l1a_s of level1 is
```

```
component Hadamard_gate
port(qi:in qubit;qo:out qubit);
end component;
component Identity_gate
port(qi:in qubit;qo:out qubit);
end component;
begin
c1:  Hadamard_gate port map(si.qa(0),so.qa(0));
c2:  Hadamard_gate port map(si.qa(1),so.qa(1));
c3:  Hadamard_gate port map(ai,ao);
c4:  Identity_gate port map(si.qa(2),so.qa(2));
so.ent<=false after 10 ns;
so.qr<=si.qr after 10 ns;
so.bub<=si.bub after 10 ns;
end l1a_s;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity Hadamard_gate is
port(qi: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 10 ns;
qo(1)<=(1.00/sqrt(2.00))*(qi(0)-qi(1))after 10 ns;
end hga;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity Identity_gate is
port(qi:in qubit;qo:out qubit);
end Identity_gate;
architecture iga of Identity_gate is
begin
qo(0)<= qi(0) after 10 ns;
qo(1)<= qi(1) after 10 ns;
end iga;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity hnh_gate is
port(qi:in qubit;qo:out qubit);
end hnh_gate;
architecture hnha of hnh_gate is
begin
qo(0)<= qi(0) after 10 ns;
qo(1)<= (-1.00)*qi(1) after 10 ns;
end hnha;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity test_grover is
end test_grover;
architecture tga of test_grover is
component grover_circ
port(search_regi:in qudata_3q;ansi:in qubit;rez:out bit_vector(0 to 3));
end component;
signal ts:qudata_3q;
signal ta:qubit;
signal ro:bit_vector(0 to 3);
begin
t1:grover_circ port map(ts,ta,ro);
process
variable tsv:qudata_3q;
variable tav:qubit;
begin
tsv.ent:=false;
tsv.qr(0).re:=1.00;tsv.qr(0).im:=0.00;
tsv.qr(1).re:=0.00;tsv.qr(1).im:=0.00;
tsv.qr(2).re:=0.00;tsv.qr(2).im:=0.00;
tsv.qr(3).re:=0.00;tsv.qr(3).im:=0.00;
tsv.qr(4).re:=0.00;tsv.qr(4).im:=0.00;
tsv.qr(5).re:=0.00;tsv.qr(5).im:=0.00;
tsv.qr(6).re:=0.00;tsv.qr(6).im:=0.00;
tsv.qr(7).re:=0.00;tsv.qr(7).im:=0.00;
tsv.qa(0)(0).re:=1.00;tsv.qa(0)(0).im:=0.00;
tsv.qa(0)(1).re:=0.00;tsv.qa(0)(1).im:=0.00;
tsv.qa(1)(0).re:=1.00;tsv.qa(1)(0).im:=0.00;
tsv.qa(1)(1).re:=0.00;tsv.qa(1)(1).im:=0.00;
tsv.qa(2)(0).re:=1.00;tsv.qa(2)(0).im:=0.00;
tsv.qa(2)(1).re:=0.00;tsv.qa(2)(1).im:=0.00;
tsv.bub(0).zeros:=0;
tsv.bub(0).bubble(0).nature:=0;
tsv.bub(0).bubble(0).position:=-1;
tsv.bub(0).bubble(1).nature:=0;
tsv.bub(0).bubble(1).position:=-1;
tsv.bub(1).zeros:=0;
tsv.bub(1).bubble(0).nature:=0;
tsv.bub(1).bubble(0).position:=-1;
tsv.bub(1).bubble(1).nature:=0;
tsv.bub(1).bubble(1).position:=-1;
tav(0).re:=0.00;tav(0).im:=0.00;
tav(1).re:=1.00;tav(1).im:=0.00;
ts<=tsv;
ta<=tav;
wait;
end process;
end tga;
configuration beh_sim of grover_circ is
for struct1
for c1:level1 use entity work.level1(l1a_b);
end for;
end for;
end beh_sim;
```

```
configuration bub_sim of test_grover is
for tga
for t1:grover_circ use entity work.grover_circ(struct_bub);
end for;
end for;
end bub_sim;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity oracle is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end oracle;
architecture oa_b of oracle is
begin
process (si,ai)
variable r:qudata_3q;
variable ti :  quregister(0 to 7);
begin
if si.ent then
l1:  for i in 0 to 7 loop ti(i):=si.qr(i);
end loop l1;
else
ti(0):=si.qa(0)(0)*si.qa(1)(0)*si.qa(2)(0);
ti(1):=si.qa(0)(0)*si.qa(1)(0)*si.qa(2)(1);
ti(2):=si.qa(0)(0)*si.qa(1)(1)*si.qa(2)(0);
ti(3):=si.qa(0)(0)*si.qa(1)(1)*si.qa(2)(1);
ti(4):=si.qa(0)(1)*si.qa(1)(0)*si.qa(2)(0);
ti(5):=si.qa(0)(1)*si.qa(1)(0)*si.qa(2)(1);
ti(6):=si.qa(0)(1)*si.qa(1)(1)*si.qa(2)(0);
ti(7):=si.qa(0)(1)*si.qa(1)(1)*si.qa(2)(1);
end if;
r.qr(0):=ti(0);
r.qr(1):=ti(1);
r.qr(2):=ti(2);
r.qr(3):=ti(3);
r.qr(4):=-1.00 *ti(4);
r.qr(5):=ti(5);
r.qr(6):=ti(6);
r.qr(7):=ti(7);
r.qa:=si.qa; r.bub:=si.bub;
r.ent:=true;
so <= r after 10 ns; ao <= ai after 10 ns;
end process;
end oa_b;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity level2 is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end level2;
architecture l2a_b of level2 is
begin
```

```
process (si,ai)
variable r:qudata_3q;
variable ti :  quregister(0 to 7);
begin
l1:  for i in 0 to 7 loop ti(i):=si.qr(i);
end loop l1;
r.qr(0):=(1.00/2.00)*(ti(0)+ti(2)+ti(4)+ti(6));
r.qr(1):=(1.00/2.00)*(ti(1)+ti(3)+ti(5)+ti(7));
r.qr(2):=(1.00/2.00)*(ti(0)-ti(2)+ti(4)-ti(6));
r.qr(3):=(1.00/2.00)*(ti(1)-ti(3)+ti(5)-ti(7));
r.qr(4):=(1.00/2.00)*(ti(0)+ti(2)-ti(4)-ti(6));
r.qr(5):=(1.00/2.00)*(ti(1)+ti(3)-ti(5)-ti(7));
r.qr(6):=(1.00/2.00)*(ti(0)-ti(2)-ti(4)+ti(6));
r.qr(7):=(1.00/2.00)*(ti(1)-ti(3)-ti(5)+ti(7));
r.qa:=si.qa;
r.ent:=true; r.bub:=si.bub;
so <= r after 10 ns; ao <= ai after 10 ns;
end process;
end l2a_b;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity phase_shift is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end phase_shift;
architecture pha_b of phase_shift is
begin
process (si,ai)
variable r:qudata_3q;
variable ti :  quregister(0 to 7);
begin
l1:  for i in 0 to 7 loop ti(i):=si.qr(i);
end loop l1;
r.qr(0):=ti(0);
r.qr(1):=ti(1);
r.qr(2):=-1.00 * ti(2);
r.qr(3):=ti(3);
r.qr(4):=-1.00 * ti(4);
r.qr(5):=ti(5);
r.qr(6):=-1.00 * ti(6);
r.qr(7):=ti(7);
r.qa:=si.qa; r.bub:=si.bub;
r.ent:=true;
so <= r after 10 ns; ao <= ai after 10 ns;
end process;
end pha_b;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity ent_anal is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end ent_anal;
```

```
architecture eaa of ent_anal is
begin
process
variable r:qudata_3q;
variable ti :  quregister(0 to 7);
begin
l1:  for i in 0 to 7 loop ti(i):=si.qr(i);
end loop l1;
r.qr(0):=ti(0);
r.qr(1):=ti(1);
r.qr(2):=ti(2);
r.qr(3):=ti(3);
r.qr(4):=ti(4);
r.qr(5):=ti(5);
r.qr(6):=ti(6);
r.qr(7):=ti(7);
r.qa(0)(0).re:=(1.00/sqrt(2.00)); r.qa(0)(0).im:=0.00;r.qa(0)(1).re:=(-1.00/sqrt(2.00));
r.qa(0)(1).im:=0.00;
r.qa(1)(0).re:=(1.00/sqrt(2.00)); r.qa(1)(0).im:=0.00;r.qa(1)(1).re:=(1.00/sqrt(2.00));
r.qa(1)(1).im:=0.00;
r.qa(2)(0).re:=1.00; r.qa(2)(0).im:=0.00; r.qa(2)(1).re:=0.00;r.qa(2)(1).im:=0.00;
r.bub:=si.bub;
r.ent:=false;
wait on si.qa;
so <= r ; ao <= ai;
end process;
end eaa;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity level3 is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end level3;
architecture l3a_s of level3 is
component Hadamard_gate
port(qi:in qubit;qo:out qubit);
end component;
component Identity_gate
port(qi:in qubit;qo:out qubit);
end component;
begin
c1:  Hadamard_gate port map(si.qa(0),so.qa(0));
c2:  Hadamard_gate port map(si.qa(1),so.qa(1));
c3:  Identity_gate port map(ai,ao);
c4:  Identity_gate port map(si.qa(2),so.qa(2));
so.ent<=false after 10 ns;
so.qr<=si.qr after 10 ns;
so.bub<=si.bub after 10 ns;
end l3a_s;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
```

```
entity measurement is
port(si:in qudata_3q;ai:in qubit; r:out bit_vector(0 to 3));
end measurement;
architecture masa of measurement is
begin
process(si,ai)
variable rv:bit_vector(0 to 3);
variable t:integer;
begin
if si.ent then
l0:  for i in 0 to 7 loop if (si.qr(i).re /= 0.00 and si.qr(i).im /= 0.00) then
t:=i;
end if;
end loop l0;
case t is
when 0 => rv(0):='0';rv(1):='0';rv(2):='0';
when 1 => rv(0):='0';rv(1):='0';rv(2):='1';
when 2 => rv(0):='0';rv(1):='1';rv(2):='0';
when 3 => rv(0):='0';rv(1):='1';rv(2):='1';
when 4 => rv(0):='1';rv(1):='0';rv(2):='0';
when 5 => rv(0):='1';rv(1):='0';rv(2):='1';
when 6 => rv(0):='1';rv(1):='1';rv(2):='0';
when others => rv(0):='1';rv(1):='1';rv(2):='1';
end case;
else
l1:for i in 0 to 2 loop if si.qa(i)(0).re = 0.00 and si.qa(i)(0).im = 0.00 then rv(i):='1';
elsif si.qa(i)(1).re = 0.00 and si.qa(i)(1).im = 0.00 then rv(i):='0';
else rv(i):='0';
end if;
end loop l1;
end if;
if (ai(0).re = 0.00 and ai(0).im = 0.00) then rv(3):='1';
elsif ai(1).re = 0.00 and ai(1).im = 0.00 then rv(3):='0';
else rv(3):='0';
end if;
r <= rv after 10 ns;
end process;
end masa;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity oracle_bub is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end oracle_bub;
architecture oba of oracle_bub is
component hnh_gate
port(qi:in qubit;qo:out qubit);
end component;
component Hadamard_gate
port(qi:in qubit;qo:out qubit);
end component;
component Identity_gate
port(qi:in qubit;qo:out qubit);
```

```
end component;
begin
c1:  Identity_gate port map(si.qa(0),so.qa(0));
c2:  hnh_gate port map(si.qa(1),so.qa(1));
c3:  Identity_gate port map(ai,ao);
c4:  Hadamard_gate port map(si.qa(2),so.qa(2));
so.ent<=true after 10 ns;
so.qr<=si.qr after 10 ns;
so.bub(0).bubble(0).nature <= -1 after 10 ns;
so.bub(0).bubble(0).position <= 3 after 10 ns;
so.bub(0).bubble(1).nature <= 1 after 10 ns;
so.bub(0).bubble(1).position <= 5 after 10 ns;
so.bub(0).zeros <= 2 after 10 ns;
so.bub(1).bubble(0).nature <= -1 after 10 ns;
so.bub(1).bubble(0).position <= 3 after 10 ns;
so.bub(1).bubble(1).nature <= 0 after 10 ns;
so.bub(1).bubble(1).position <= -1 after 10 ns;
so.bub(1).zeros <= -1 after 10 ns;
end oba;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity level2_bub is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end level2_bub;
architecture l2ba of level2_bub is
component hnh_gate
port(qi:in qubit;qo:out qubit);
end component;
component Identity_gate
port(qi:in qubit;qo:out qubit);
end component;
begin
c1:  hnh_gate port map(si.qa(0),so.qa(0));
c2:  hnh_gate port map(si.qa(1),so.qa(1));
c3:  Identity_gate port map(ai,ao);
c4:  Identity_gate port map(si.qa(2),so.qa(2));
so.ent<=true after 10 ns;
so.qr<=si.qr after 10 ns;
so.bub(0).bubble(0).nature <= -1 after 10 ns;
so.bub(0).bubble(0).position <= 3 after 10 ns;
so.bub(0).bubble(1).nature <= -1 after 10 ns;
so.bub(0).bubble(1).position <= 5 after 10 ns;
so.bub(0).zeros <= 2 after 10 ns;
so.bub(1).bubble(0).nature <= 1 after 10 ns;
so.bub(1).bubble(0).position <= 3 after 10 ns;
so.bub(1).bubble(1).nature <= 0 after 10 ns;
so.bub(1).bubble(1).position <= -1 after 10 ns;
so.bub(1).zeros <= -1 after 10 ns;
end l2ba;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity phase_shift_bub is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end phase_shift_bub;
architecture psba of phase_shift_bub is
component Hadamard_gate
port(qi:in qubit;qo:out qubit);
end component;
component Identity_gate
port(qi:in qubit;qo:out qubit);
end component;
begin
c1:  Identity_gate port map(si.qa(0),so.qa(0));
c2:  Identity_gate port map(si.qa(1),so.qa(1));
c3:  Identity_gate port map(ai,ao);
c4:  Hadamard_gate port map(si.qa(2),so.qa(2));
so.ent<=false after 10 ns;
so.qr<=si.qr after 10 ns;
so.bub(0).bubble(0).nature <= 0 after 10 ns;
so.bub(0).bubble(0).position <= -1 after 10 ns;
so.bub(0).bubble(1).nature <= 0 after 10 ns;
so.bub(0).bubble(1).position <= -1 after 10 ns;
so.bub(0).zeros <= 0 after 10 ns;
so.bub(1).bubble(0).nature <= 0 after 10 ns;
so.bub(1).bubble(0).position <= -1 after 10 ns;
so.bub(1).bubble(1).nature <= 0 after 10 ns;
so.bub(1).bubble(1).position <= -1 after 10 ns;
so.bub(1).zeros <= 0 after 10 ns;
end psba;
use work.qupack.all;
library ieee;
use ieee.math_real.all;
use ieee.math_complex.all;
entity level3_bub is
port(si:in qudata_3q;ai:in qubit;so:out qudata_3q;ao:out qubit);
end level3_bub;
architecture l3ba of level3_bub is
component Hadamard_gate
port(qi:in qubit;qo:out qubit);
end component;
component Identity_gate
port(qi:in qubit;qo:out qubit);
end component;
begin
c1:  Hadamard_gate port map(si.qa(0),so.qa(0));
c2:  Hadamard_gate port map(si.qa(1),so.qa(1));
c3:  Identity_gate port map(ai,ao);
c4:  Identity_gate port map(si.qa(2),so.qa(2));
so.ent<=false after 10 ns;
so.qr<=si.qr after 10 ns;
so.bub <= si.bub after 10 ns;
end l3ba;
use work.qupack.all;
library ieee;
```

```
use ieee.math_real.all;
use ieee.math_complex.all;
entity measurement_bub is
port(si:in qudata_3q;ai:in qubit; r:out bit_vector(0 to 3));
end measurement_bub;
architecture masab of measurement_bub is
begin
process(si,ai)
variable rv:bit_vector(0 to 3);
variable t:integer;
begin
l1:for i in 0 to 2 loop
if si.qa(i)(0).re = 0.00 and si.qa(i)(0).im = 0.00 then rv(i):='1';
elsif si.qa(i)(1).re = 0.00 and si.qa(i)(1).im = 0.00 then rv(i):='0';
else rv(i):='0';
end if;
end loop l1;
if (ai(0).re = 0.00 and ai(0).im = 0.00) then rv(3):='1';
elsif ai(1).re = 0.00 and ai(1).im = 0.00 then rv(3):='0';
else rv(3):='0';
end if;
r <= rv after 10 ns;
end process;
end masab;
```

Appendix D

Running a genetic application on a quantum computer

In this appendix we will examine an example of how the genetic algorithms will run according to the algorithm described in Chapter 5 (Reduced Quantum Genetic Algorithm). Therefore we will take into consideration a typical problem for the genetic approach.

Problem: A person has a backpack with a maximum capacity of 10 kilograms. He also has 4 items with the following characteristics: item I_1 having 7 kg and a value of 40 \$, item I_2 (4 kg, 100 \$), item I_3 (2 kg, 50 \$), and item I_4 (3 kg, 30 \$). Which items would this person carry in his backpack so that it would have the maximum value ?

The classical genetic approach will have as chromosome a 4-bit code, where each '1' bit means that the corresponding item is present in the backpack. For instance, a 1001 code indicates that the backpack contains items I_1 and I_4. In the quantum version, the chromosome encoding will have all the 4-qubit classical values (see Table D.1) as superposed basis states, which represent valid and invalid (i.e. the items will add up to more than 10 kg) individuals.

$$|\psi\rangle_i^1 = \frac{1}{4}\sum_{u=0}^{15} |u\rangle_i^{ind} \otimes |0\rangle_i^{fit} = \frac{1}{4}\left(\begin{array}{c} |0000\rangle + |0001\rangle + |0010\rangle + |0011\rangle \\ +|0100\rangle + |0101\rangle + |0110\rangle + |0111\rangle \\ +|1000\rangle + |1001\rangle + |1010\rangle + |1011\rangle \\ +|1100\rangle + |1101\rangle + |1110\rangle + |1111\rangle \end{array}\right) \otimes |0000000000\rangle \tag{D.1}$$

The fitness function that we will apply, would be an arithmetic function according to the requirements formulated in Section 5.2.2. The fitness formula, given in Equation D.2, contains two variables, the chromosome value val and the chromosome mass m; also two constants are used, the total added value of all items ($val_t = 220$) and the maximum allowed package mass ($m_{max} = 10$). For any chromosome x, the fitness function is given by:

$$f_{fit}(x) = val(x) - (val_t + 1) \times (m(x) \operatorname{div} m_{max}) = val(x) - 221 \times (m(x) \operatorname{div} 10) \tag{D.2}$$

Applying the fitness function over the individual registers $|\rho\rangle_i^{ind}$ means that we use the $U_{f_{fit}}$ basis state permutation, obtaining the state presented in Equation D.3.

I_1	I_2	I_3	I_4	Value [\$]	mass [kg]	Validity
0	0	0	0	0	0	valid
0	0	0	1	30	3	valid
0	0	1	0	50	2	valid
0	0	1	1	80	5	valid
0	1	0	0	100	4	valid
0	1	0	1	130	7	valid
0	1	1	0	150	6	valid
0	1	1	1	180	9	valid
1	0	0	0	40	7	valid
1	0	0	1	70	10	valid
1	0	1	0	90	9	valid
1	0	1	1	120	12	invalid
1	1	0	0	140	11	invalid
1	1	0	1	170	14	invalid
1	1	1	0	190	13	invalid
1	1	1	1	220	16	invalid

Table D.1: All the chromosome binary combinations, valid and invalid, with the corresponding fitness values.

$$|\psi\rangle_i^2 = |\rho\rangle_i^{ind} \otimes |\rho\rangle_i^{fit} = \frac{1}{4}\begin{pmatrix} |0000\rangle \otimes |1000000000\rangle \\ +|0001\rangle \otimes |1000011110\rangle \\ +|0010\rangle \otimes |1000110010\rangle \\ +|0011\rangle \otimes |1001010000\rangle \\ +|0100\rangle \otimes |1001100100\rangle \\ +|0101\rangle \otimes |1010000010\rangle \\ +|0110\rangle \otimes |1010010110\rangle \\ +|0111\rangle \otimes |1010110100\rangle \\ +|1000\rangle \otimes |1000101000\rangle \\ +|1001\rangle \otimes |1001000110\rangle \\ +|1010\rangle \otimes |1001011010\rangle \\ +|1011\rangle \otimes |0110011011\rangle \\ +|1100\rangle \otimes |0110101111\rangle \\ +|1101\rangle \otimes |0111001101\rangle \\ +|1110\rangle \otimes |0111100001\rangle \\ +|1111\rangle \otimes |0111111111\rangle \end{pmatrix} \tag{D.3}$$

The fitness values of the invalid individuals are negative numbers (all least significant 9 bits of the fitness values represent 2's complement numbers), with the most significant bit being dedicated to indicating the validity of the corresponding chromosome (0 means invalid, 1 indicates a valid individual), see Section 5.4.2 for details.

The next step of the Reduced Quantum Genetic Algorithm is to apply the oracle over the first pair of individual-fitness registers. According to the algorithm from Section 5.5, we have to get a random value for variable *max*. Suppose that the yielded value for *max* is 85, then the state of the $|\psi\rangle_0$ register at this point is given in Equation D.4 and D.5: $|\psi\rangle_0^3$ is the pair registers state after applying the subtractor and phase-shift part of the oracle, while $|\psi\rangle_0^4$ is the state obtained after applying the entire oracle, including the adder part (see Figure 5.4 from Section 5.4.2). One observation linked to the details presented in Section 5.4.2 is that the phase shift (i.e. amplitude a_i becomes $-a_i$) is triggered by a '0' value of the 2nd bit from the left in the fitness register.

$$|\psi\rangle_0^3 = |\rho\rangle_0^{ind} \otimes |\phi\rangle_0^{fit} = \frac{1}{4} \begin{pmatrix} |0000\rangle \otimes |1110101011\rangle \\ +|0001\rangle \otimes |1111001001\rangle \\ +|0010\rangle \otimes |1111011101\rangle \\ +|0011\rangle \otimes |1111111011\rangle \\ -|0100\rangle \otimes |1000001111\rangle \\ -|0101\rangle \otimes |1000101101\rangle \\ -|0110\rangle \otimes |1001000001\rangle \\ -|0111\rangle \otimes |1001011111\rangle \\ +|1000\rangle \otimes |1111010011\rangle \\ +|1001\rangle \otimes |1111110001\rangle \\ -|1010\rangle \otimes |1000000101\rangle \\ +|1011\rangle \otimes |1101000110\rangle \\ +|1100\rangle \otimes |1101011010\rangle \\ +|1101\rangle \otimes |1101111000\rangle \\ +|1110\rangle \otimes |1110001100\rangle \\ +|1111\rangle \otimes |1110101010\rangle \end{pmatrix} \tag{D.4}$$

$$|\psi\rangle_0^4 = |\rho\rangle_0^{ind} \otimes |\phi\rangle_0^{fit} = \frac{1}{4} \begin{pmatrix} |0000\rangle \otimes |1000000000\rangle \\ +|0001\rangle \otimes |1000011110\rangle \\ +|0010\rangle \otimes |1000110010\rangle \\ +|0011\rangle \otimes |1001010000\rangle \\ -|0100\rangle \otimes |1001100100\rangle \\ -|0101\rangle \otimes |1010000010\rangle \\ -|0110\rangle \otimes |1010010110\rangle \\ -|0111\rangle \otimes |1010110100\rangle \\ +|1000\rangle \otimes |1000101000\rangle \\ +|1001\rangle \otimes |1001000110\rangle \\ -|1010\rangle \otimes |1001011010\rangle \\ +|1011\rangle \otimes |0110011011\rangle \\ +|1100\rangle \otimes |0110101111\rangle \\ +|1101\rangle \otimes |0111001101\rangle \\ +|1110\rangle \otimes |0111100001\rangle \\ +|1111\rangle \otimes |0111111111\rangle \end{pmatrix} \tag{D.5}$$

After applying the Grover algorithm over the fitness register (rightmost 10 qubits) of $|\psi\rangle_0$, we will get state $|\psi\rangle_0^5$, as presented in Equation D.6, where the amplitudes $a_0, a_1, a_2, a_3, a_8, a_9, a_{11}, a_{12}, \ldots a_{15}$ ≈ 0 and $|a_4|^2 + |a_5|^2 + |a_6|^2 + |a_7|^2 + |a_{10}|^2 \approx 1$.

$$|\psi\rangle_0^5 = |\rho\rangle_0^{ind} \otimes |\phi\rangle_0^{fit} = \left(\begin{array}{r} a_0|0000\rangle \otimes |1000000000\rangle \\ +a_1|0001\rangle \otimes |1000011110\rangle \\ +a_2|0010\rangle \otimes |1000110010\rangle \\ +a_3|0011\rangle \otimes |1001010000\rangle \\ +a_4|0100\rangle \otimes |1001100100\rangle \\ +a_5|0101\rangle \otimes |1010000010\rangle \\ +a_6|0110\rangle \otimes |1010010110\rangle \\ +a_7|0111\rangle \otimes |1010110100\rangle \\ +a_8|1000\rangle \otimes |1000101000\rangle \\ +a_9|1001\rangle \otimes |1001000110\rangle \\ +a_{10}|1010\rangle \otimes |1001011010\rangle \\ +a_{11}|1011\rangle \otimes |0110011011\rangle \\ +a_{12}|1100\rangle \otimes |0110101111\rangle \\ +a_{13}|1101\rangle \otimes |0111001101\rangle \\ +a_{14}|1110\rangle \otimes |0111100001\rangle \\ +a_{15}|1111\rangle \otimes |0111111111\rangle \end{array} \right) \tag{D.6}$$

Therefore, if we measure the fitness register of $|\psi\rangle_0$ after applying Grover iterations, then we will measure (with a high probability) one of the following basis states (of the rightmost 10 qubits of $|\psi\rangle_0^5$): $|1001100100\rangle, |1010000010\rangle, |1010010110\rangle, |1010110100\rangle$. Suppose that we measure $|1010000010\rangle$ (+ 130 if we convert this value in decimal). In the individual register we will have $|0101\rangle$; also the new $max := 130 + 1$.

The next algorithm iteration will involve the next individual-fitness pair registers ($|\psi\rangle_1 = |\rho\rangle_1^{ind} \otimes |\phi\rangle_1^{fit}$), by subsequently setting states $|\psi\rangle_1^3$ (oracle − subtractor and phase-shift), $|\psi\rangle_1^4$ (oracle − adder), and $|\psi\rangle_1^5$ (Grover iterations):

$$|\psi\rangle_1^3 = |\rho\rangle_1^{ind} \otimes |\phi\rangle_1^{fit} = \frac{1}{4} \left(\begin{array}{r} |0000\rangle \otimes |1101111101\rangle \\ +|0001\rangle \otimes |1110011011\rangle \\ +|0010\rangle \otimes |1110101111\rangle \\ +|0011\rangle \otimes |1111001101\rangle \\ +|0100\rangle \otimes |1111100001\rangle \\ +|0101\rangle \otimes |1111111111\rangle \\ -|0110\rangle \otimes |1000010011\rangle \\ -|0111\rangle \otimes |1000110001\rangle \\ +|1000\rangle \otimes |1110100101\rangle \\ +|1001\rangle \otimes |1111000011\rangle \\ +|1010\rangle \otimes |1111010111\rangle \\ +|1011\rangle \otimes |0100011000\rangle \\ +|1100\rangle \otimes |0100101100\rangle \\ +|1101\rangle \otimes |0101001010\rangle \\ +|1110\rangle \otimes |0101011110\rangle \\ +|1111\rangle \otimes |0101111100\rangle \end{array} \right) \tag{D.7}$$

$$|\psi\rangle_1^4 = |\rho\rangle_1^{ind} \otimes |\phi\rangle_1^{fit} = \frac{1}{4}\begin{pmatrix} |0000\rangle \otimes |1000000000\rangle \\ +|0001\rangle \otimes |1000011110\rangle \\ +|0010\rangle \otimes |1000110010\rangle \\ +|0011\rangle \otimes |1001010000\rangle \\ +|0100\rangle \otimes |1001100100\rangle \\ +|0101\rangle \otimes |1010000010\rangle \\ -|0110\rangle \otimes |1010010110\rangle \\ -|0111\rangle \otimes |1010110100\rangle \\ +|1000\rangle \otimes |1000101000\rangle \\ +|1001\rangle \otimes |1001000110\rangle \\ +|1010\rangle \otimes |1001011010\rangle \\ +|1011\rangle \otimes |0110011011\rangle \\ +|1100\rangle \otimes |0110101111\rangle \\ +|1101\rangle \otimes |0111001101\rangle \\ +|1110\rangle \otimes |0111100001\rangle \\ +|1111\rangle \otimes |0111111111\rangle \end{pmatrix} \tag{D.8}$$

$$|\psi\rangle_1^5 = |\rho\rangle_1^{ind} \otimes |\phi\rangle_1^{fit} = \begin{pmatrix} a_0|0000\rangle \otimes |1000000000\rangle \\ +a_1|0001\rangle \otimes |1000011110\rangle \\ +a_2|0010\rangle \otimes |1000110010\rangle \\ +a_3|0011\rangle \otimes |1001010000\rangle \\ +a_4|0100\rangle \otimes |1001100100\rangle \\ +a_5|0101\rangle \otimes |1010000010\rangle \\ +a_6|0110\rangle \otimes |1010010110\rangle \\ +a_7|0111\rangle \otimes |1010110100\rangle \\ +a_8|1000\rangle \otimes |1000101000\rangle \\ +a_9|1001\rangle \otimes |1001000110\rangle \\ +a_{10}|1010\rangle \otimes |1001011010\rangle \\ +a_{11}|1011\rangle \otimes |0110011011\rangle \\ +a_{12}|1100\rangle \otimes |0110101111\rangle \\ +a_{13}|1101\rangle \otimes |0111001101\rangle \\ +a_{14}|1110\rangle \otimes |0111100001\rangle \\ +a_{15}|1111\rangle \otimes |0111111111\rangle \end{pmatrix} \tag{D.9}$$

After applying the Grover algorithm over the fitness register (rightmost 10 qubits) of $|\phi\rangle_1^{fit}$ from $|\psi\rangle_1^4$, we will get state $|\psi\rangle_1^5$, as presented in Equation D.9, where the amplitudes $a_0, a_1, a_2, a_3, a_4, a_5, a_8, a_9, \ldots a_{15} \approx 0$ and $|a_6|^2 + |a_7|^2 \approx 1$.

Therefore, if we measure the fitness register of $|\psi\rangle_1^5$ after applying Grover iterations, then we will measure (with a high probability) one of the following basis states (of the rightmost 10 qubits of $|\psi\rangle_1^5$): $|1010010110\rangle, |1010110100\rangle$. Suppose that we measure $|1010110100\rangle$ (+ 180 in decimal). In the individual register we will have $|0111\rangle$, which is also the solution for our problem.

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] J. R. Armstrong, F. Gray, F. Gail, "VHDL Design: Representation and Synthesis", Prentice Hall (2000).

[6] P. J. Ashenden, "The Designer's guide to VHDL (second edition)", Morgan Kaufmann Publishers (2001).

[7] 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).

[8] C.H. Bennett, "Logical reversibility of computation", IBM Journal of Research an Development, 17, pp. 525 (1973).

[9] C. H. Bennett, "The Thermodynamics of Computation – a Review", International Journal of Theoretical Physics (1981)

[10] 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.

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

[12] M. Boyer, G. Brassard, P. Hoyer, A.Tapp, "Tight bounds on quantum searching", Fortsch. Phys. – Prog.Phys., 46(4-5):493-505 (1998).

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

[14] G. Brassard, P.Hoyer, "Quantum counting", A. Tapp, arXive e-print quant-ph/9805082 (1998).

[15] J. Cirac, P. Zoller, "Quantum computation with cold trapped ions", Phys. Rev. Lett., Vol. 74, pp. 4091-4094 (1995).

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

[17] D. Coppersmith, "An approximate Fourier transform useful in quantum factoring", IBM Research Report RC 19642 (1994).

[18] J. Crowcroft, "On the Nature of Computing", Communications of the ACM, Vol. 48, Nr. 2, pp. 19-20, (Feb. 2005).

[19] E.P. DeBenedictis, "Will Moore's Law Be Sufficient?", Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, pp.45, (2004).

[20] E.P. DeBenedictis, "Reversible logic for supercomputing", Proceedings of the 2nd ACM Conference on Computing Frontiers, pp. 391 - 402, (2005).

[21] 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).

[22] G. De Micheli, M. Sami, "Hardware/Software CO-Design", Kluwer Academic Publishers (1996).

[23] G. De Micheli, "Design and Optimization of Digital Circuits", McGraw-Hill, (1996).

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

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

[26] D. Deutsch and R. Jozsa, "Rapid Solution of Problems by Quantum Computation", Proc. R. Soc. London A, 439:553 (1992).

[27] D. DiVincenzo, "Two-bit gates are universal for quantum computation", Phys. Rev. A (AT5101) (1995).

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

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

[30] R.P. Feynman, "Simulating Physics with Computers", International Journal of Theoretical Physics 21, p.467 (1982).

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

[32] N. A. Gershenfeld, I. Chuang, S. Lloyd, "Bulk quantum computation", Proceedings of the 4th Workshop on Physics of Computation, PhysComp96, p.134 (1996).

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

[34] P. Gossett,"Quantum Carry-Save Arithmetic",Online preprint quant-ph/9808061 (1998).

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

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

[37] L.K. Grover, In Proc. 28th Annual ACM Symposium on the Theory of Computation, pp.212-219, ACM Press, (1996).

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

[39] L. Hales, S. Hallgren, "An improved quantum Fourier transform algorithm and applications", 41st Annual Symposium on Foundations of Computer Science (FOCS) Redondo Beach CA (2000).

[40] 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).

[41] 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).

[42] J. P. Hayes, I. Polian, B. Becker, "Testing for Missing-Gate Faults in Reversible Circuits", Proc. 13th Asian Test Symposium, pp. 100-105, (2004).

[43] T. Hogg, "Highly structured searches with quantum computers", Physical Review Letters, 80:2473-2476, (1998).

[44] T. Hogg, "Quantum search heuristics", Physical Review A, 61:052311, (2000).

[45] 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).

[46] A. U. Khalid, Z. Zilic, K. Radecka, "FPGA Emulation of Quantum Circuits", In Proc. ICCD 2004: pp. 310-315 (2004).

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

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

[49] D.E. Knuth, "The art of computer programming. Volume 2: Seminumerical algorithms", Addison-Wesley, Massachusetts, 3rd edition, (1997).

[50] J. Koza, "Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems", Technical Report STAN-CS-90-1314, Department of Computer Science, Stanford University (1990).

[51] J. Koza, "Genetic programming: On the programming of computers by means of natural selection", MIT Press, Cambridge, MA (1992).

[52] P. K. Lala, "Self-Checking and Fault-Tolerant Digital Design", Morgan Kaufmann Publishers, San Francisco, CA, (2000).

[53] R. Landauer, "Irreversibility and heat generation in the computing process", IBM Journal of Research an Development, 5, pp.183 (1961).

[54] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, "The number field sieve", In Proceedings of the 22nd Annual ACM Symposium on the Theory of Computation, pages 564–572, (1990).

[55] 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).

[56] M. Lukac, M. Perkowski, "Evolving Quantum Circuits Using Genetic Algorithm", In Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02), IEEE Computer Society, Washington, DC, 177, (July 15 - 18, 2002).

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

[58] C. Moore and M. Nilsson, "Parallel quantum computation and quantum codes", Los Alamos Preprint Archives (1998), quant-ph/9808027.

[59] C.Moore, "Quantum circuits: fanout, parity, and counting.", Los Alamos Preprint Archives (1999), quant-ph/9903046.

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

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

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

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

[64] 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).

[65] B. Omer, "A procedural formalism for quantum computing", Technical Report, Department of Theoretical Physics, Technical University of Vienna (1998).

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

[67] A. Omondi, ”Computer Arithmetic Systems. Algorithms, Architecture and Implementations”, Prentice Hall (C.A.R. Hoare series) (1994).

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

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

[70] C.M. Papadimitriou, ”Computational Complexity”, Addison-Wesley, Reading MA, (1994).

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

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

[73] 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

[74] M. Perkowski, M. Lukac, M. Pivtoraiko, P. Kerntopf, M. Folgheraiter, D. Lee, H. Kim, H. Kim, W. Hwangboo, J.-W. Kim, and Y. Choi, ”A hierarchical approach to computer aided design of quantum circuits”, In Proceedings of 6th International Symposium on Representations and Methodology of Future Computing Technology, RM pp.201-209, (2003).

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

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

[77] J. Preskill, ”Quantum Computation”, Course Handouts, CALTECH (2001).

[78] J. Preskill, ”The cost of quantum fault tolerance”, 8th Workshop on Quantum Information Processing (Invited Talk), MIT Cambridge, MA, USA (January 2005).

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

[80] L. Prodan, M. Udrescu, M. Vlăduţiu, ”Reliability Assessment in Embryonics Inspired by Fault-Tolerant Quantum Computation”, Proc. 2nd ACM Conference On Computing Frontiers, ACM Press, pp. 323-333 (Ischia, Italy, 2005).

[81] L. Prodan, M. Udrescu, M. Vlăduţiu, ”Multiple-Level Concatenated Coding in Embryonics: A Dependability Analysis”, GECCO (ACM-SIGEVO), ACM Press, pp. 941-948, Washigton, DC, USA, (June 25-29 2005).

[82] L. Prodan, M. Udrescu, M. Vlăduţiu, ”Survivability of Embryonic Memories: Analysis and Design Principles”, IEEE NASA/DoD Conference on Evolvable Hardware (EH’05), IEEE Press, pp. 280-289, Washington, DC, USA, (June 29 - July 1, 2005).

[83] M. Reck, A. Zeilinger, H.J. Bernstein, and P. Bertani, "Experimental realization of any discrete unitary operator", Phys. Rev. Lett. 73, 58 (1994).

[84] 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).

[85] 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).

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

[87] 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).

[88] V. V. Shende, S. S. Bullock, I. L. Markov, "A Practical Top-down Approach to Quantum Circuit Synthesis," Proc. Asia and South Pacific Design Automation Conference, pp. 272-275, Shanghai, China, (2005) quant-ph/0406176.

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

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

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

[92] R. Some, W. Kim, G. Khanoyan, L. Callum, A. Agrawal, J. Beahan, "A Software-Implemented Fault Injection Methodology for Design and Validation of System Fault Tolerance", International Conference on Dependable Systems and Networks (DSN'01), pp. 501-506 (2001).

[93] L. Spector, H. Barnum, K.J. Bernstein, N. Swamy, "Genetic Programming For Quantum Computers", Proceedings of the 3rd Annual Conference on Genetic Programming, Madison, Wisconsin, USA, Morgan Kaufmann Publishers, pp. 365–373, (1998).

[94] L. Spector, H. Barnum, K.J. Bernstein, N. Swamy, "Quantum Computing Applications of Genetic Programming", In Advances in Genetic Programming, volume 3, chapter 7, pp. 135–160, MIT Press (1999).

[95] 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).

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

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

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

[99] A.M. Steane, "Overhead and noise threshold of fault-tolerant quantum error correction", Los Alamos e-printarchive, quant-ph/0207119, (2002).

[100] K. Svore, A. Cross, A. Aho, I. Chuang, I. Markov, "Towards a software architecture for quantum computing design tools", Proc. 2nd International Workshop on Quantum Programming Languages, pp. 145-162, Turku Finland, (July 2004).

[101] K. Svozil, "Quantum computation and complexity theory", Course given at the Institut für Informationssysteme, Abteilung für Datenbanken und Expertensysteme, University of Technology Vienna (1994) hep-th/9412047

[102] A. Thompson, "Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution", Springer-Verlag distinguished dissertation series, (1998).

[103] T. Toffoli, "Reversible Computing", Automata, Languages and Programming, eds. J.W. de Bakker and J. van Leeuwen, Springer New York, (1980).

[104] M. Udrescu, "The basics of quantum computing", Transactions on Automatic Control and Computer Science, University "Politehnica" Timişoara, Vol 44 (58) No. 1,2, pp. 17-24 (1999).

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

[106] 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).

[107] 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, IEEE Press, pp. 217-224 (April 2 - 8, 2005).

[108] M. Udrescu, L. Prodan, M. Vlăduţiu, "Simulated Fault Injection in Quantum Circuits with the Bubble Bit Technique." 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA), Coimbra, Portugal, Springer WienNewYork pp. 276-279 (March 21 - 23, 2005).

[109] M. Udrescu, L. Prodan, M. Vlăduţiu, "Improving Quantum Circuit Dependability with Reconfigurable Quantum Gate Arrays", 2nd ACM Conference On Computing Frontiers, ACM Press, pp. 133-144 (Ischia, Italy, 2005).

[110] M. Udrescu, "Using Hardware Engineering in Quantum Computation: Efficient Circuit Simulation and Reliability Improvement", SIGDA Ph.D. Forum at Design Automation Conference, (Anaheim, CA, 2005), http://www.sigda.org/daforum/abs/38.html.

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

[112] W. van Dam, "Quantum Cellular Automata", Master's Thesis, Computer Science Institute, University of Nijmegen, The Netherlands (1996).

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

[114] 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).

[115] 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).

[116] G. F. Viamontes, I. L. Markov and J. P. Hayes, "Is Quantum Search Practical?," Intl. Workshop on Logic and Synthesis, pp. 478-485, (2004).

[117] G. F. Viamontes, I. L. Markov and J. P. Hayes, "Graph-based Simulation of Quantum Computation in the Density Matrix Representation," Quantum Information and Computation, vol.5, no.2 pp. 113-130, (February 2005) quant-ph/0403114.

[118] G. Vidal, "Efficient Classical Simulation of Slightly Entangled Quantum Computations", Phys. Rev. Lett. 91, 147902 (2003)

[119] D. Whitley, "A Genetic Algorithm Tutorial", Statistics and Computing Journal, Chapman and Hall, (1994).

[120] C.P. Williams, A.G. Gray, "Automated design of quantum circuits", In Proc. of 1st NASA International Conference on Quantum Computing and Quantum Communications QCQC '98, LNCS No. 1509 Springer-Verlag, pp. 113-125, Palm Springs, CA (1998).

[121] G. Yang, W.N.N. Hung, X. Song, M. Perkowski, "Exact Synthesis of 3-Qubit Quantum Circuits from Non-Binary Quantum Gates Using Multiple-Valued Logic and Group Theory", Proceedings Design Automation and Test in Europe (DATE) Conference, Munich, Germany, Vol.1, pp. 434-435 (2005).

[122] 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).

[123] 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).

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

[125] J. Zhang, J. Vala, S. Sastry, and K.B. Whaley, "Optimal Quantum Circuit Synthesis from Controlled-Unitary Gates", Phys. Rev. A 69, 042309 (2004).

[126] ****, "Emerging Research Devices with a New Section on Emerging Research Materials", International Technology Roadmap for Semiconductors (ITRS) Report (2004 Update), http://www.itrs.net/Common/2004Update/, (2004).