FPGA Emulation of Quantum Circuits

Ahmed Usman Khalid

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

October 2005

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Engineering.

2005/10/31

Abstract

In recent years, new and novel forms of computation employing different natural phenomena such as the spin of atoms or the orientation of protein molecules have been proposed and are in the very initial stages of development. One of the most promising of these new computation techniques is quantum computing that employs various physical effects observed at the quantum level to provide significant improvement in certain computation tasks such as data search and factorization. An assortment of software-based simulators of quantum computers have been developed recently to assist in the development of this new computation process. However, efficiently simulating quantum algorithms at the software level is quite challenging since the algorithms have exponential run-times and memory requirements. Furthermore, the sequential nature of software-based computation makes simulating the parallel nature of quantum computation exceedingly difficult. In this thesis, the first hardware-based quantum algorithm emulation technique is presented. The emulator uses FPGA technology to model quantum circuits. Parallel computation available at the hardware level allows considerable speed-up as compared to the state-of-the-art software simulators as well as provides a greater insight into precision requirements for simulating quantum circuits.

Résumé

Ces dernières annes, de nouvelles techniques innovatrices de computation utilisant divers types de phénomènes naturels tels que la rotation des atomes ou l'orientation des molécules de protéine dans l'espace ont été proposées et sont présentement au stade initial de leur développement. Une technique de computation des plus prometteuses est la computation quantique employant différents effets physiques observés au niveau quantique. Cette technique permet une amélioration significative dans certaines tâches de computation telles que la recherche d'information et la factorisation. Un assortiment de logiciels de simulation d'ordinateur quantique ont récemment été développés pour aider au développement de ce nouveau processus de computation. Cependant, simuler efficacement des algorithmes quantiques au niveau logiciel est une tâche complexe, car ces algorithmes ncessitent une durée d'exécution et des ressources de mémoire exponentielles. De plus, la nature séquentielle de computation sur logiciel rend la simulation de la nature parallèle de la computation quantique extrêmement difficile. Dans cette thèse, le premier émulateur d'algorithme quantique sur hardware est présenté. L'émulateur emploie la technologie FPGA pour modeler des circuits quantiques. La computation parallèle réalisable au niveau hardware permet une accélération considérable du temps d'exécution par rapport aux simulateurs sur logiciels actuels les plus puissants. Également, la technique de simulation sur hardware présente procure plus d'information concernant la précision requise pour simuler des circuits quantiques.

Acknowledgments

First and foremost, I would like to thank my supervisors Prof. Zeljko Zilic and Prof. Katarzyna Radecka for giving me this opportunity to work with them. This thesis would not have happened if not for their guidance and patience. I would also like to thank the Microelectronics Strategic Alliance of Quebec (ReSMIQ) for their financial support for my research work.

I am also indebted to Jean-Sebastien Chenard for his friendship and guidance throughout my research. He provided invaluable counsel with the synthesis issues of the emulator as well as in the design of the serial communication interface to the PC. I would like to thank Dr. Dmitri Maslov for reviewing the thesis and his comments provided considerable polish to the final draft. I would also like to thank my friends and colleagues in the "Circle of Truth": Sadok Aouini, Simon Hong, Carmen Au, Karthik Sundaresan, Cristian Radita, Dani Tannir and Tarek AlHajj. Their friendship and company made the entire graduate school experience memorable. I would also like to thank Sadok and Cristian for the French translation of the abstract.

Finally, I would like to thank my family. My parents and my brother have been my support throughout my life. However, never have I felt and appreciated their presence more than during my time as a graduate student. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The new century has ushered in a spattering of new and innovative computational platforms that are envisaged to one day replace the currently prevalent semiconductor-based computers. Computation technology has spilled out of its traditional domains of electrical engineering and computer science into new realms such as quantum physics, biology and biomedical engineering. From storing information using the spin of photons to synthesizing transistors using organic molecules to performing large scale computation using DNA strands, researchers are moving towards the future of computing on many different roads. Research and development of these technologies is no longer limited to academic circles. Companies such as IBM, HP and D-Wave Sys are working towards bringing about the next revolution in computing in the not so distant future.

The focus of this thesis is on emulation of quantum computing. Quantum computing uses various quantum mechanical effects such as entanglement and superposition to provide massive performance speedup in certain types of computation problems such as data searching, factorization and encryption. The notion of using quantum mechanical phenomena for computational purposes was first explored in the the 1970's and early 1980's. Quantum computing came in to being when Richard Feynman proposed an abstract computational model for simulating quantum physics in 1982 [1]. This was followed by proposals by David Deutsch in 1985 of a general purpose quantum computer [2] and by Peter Shor in 1994 [3] where he proposed a factorization algorithm for quantum computers. It was Shor's publication that sparked widespread interest in quantum computing as it was the first large scale problem where quantum computers would outperform their classical counterparts

significantly.

Since then, the development of quantum technologies has been underway and a variety of different techniques are being considered to solve various hurdles facing quantum computing [4]. Some of the approaches being considered are:

- Collection of ions trapped and manipulated by lasers [5],
- NMR (Nuclear Magnetic Resonance) based information processing [6],
- Semi-conductor designs such as those based on quantum dots [7],
- Superconducting electronics [8].

Quantum computers however, are still in their infancy. The general unavailability of quantum computers has garnered interest in developing classical simulators of quantum algorithms. However, simulation of quantum algorithms presents its own set of challenges since simulating such phenomena in classical computers consumes an excessive amount of resources. This thesis proposes an FPGA-based solution to this problem, whereby parallel computation can be performed at a large scale in hardware resulting in significant performance gains as compared to software-based simulators.

1.1 Quantum Computation vs. Classical Computation

Quantum computation is an evolution of a quantum system that starts with a certain initial state. While the evolution of a quantum system can be described by a series of Schrodinger's equations [9], a more intuitive and equivalent abstraction of the quantum evolution is the quantum circuit model. The model breaks down the quantum system into two components: quantum bits which are the particles that make up the quantum system and quantum gates that are transforms that are applied on the collection of quantum bits. Therefore, analogous to classical computing quantum bits are the units of information and quantum gates are logical operations that can manipulate this information. However, unlike mainstream classical computation, quantum computation is probabilistic, that is the result of the computation cannot be achieved correctly in a finite number of steps. Furthermore, obtaining the state of the quantum system causes the system to collapse. At same time there is a significant probability of error in the obtained state information.

While classical units of information can exist in a finite number of states (predominantly binary logic is used where only the states 0 or 1 are present), quantum information can exist in an infinite number of states. Furthermore, quantum bits can undergo a quantum mechanical phenomenon known as *entanglement* whereby they can store an exponentially large amount of information. These effects can be employed to provide significant speed-up in computation in certain applications (such as data search and factorization).

1.2 Quantum Circuit Simulation vs. Emulation

Quantum circuit simulation involves numerical representation of the information stored in a quantum system and applying the necessary transforms on this information, as dictated by the quantum algorithm being simulated. The software-based simulators do not exploit the parallelism and the effect of quantum noise that are present in real-life quantum computers. Furthermore, since the state of the quantum circuits expands exponentially with linear increases in number of qubits, even a modest sized quantum circuit can take hours, even days to simulate.

The emulation of the quantum circuits also comprises of manipulating the mathematical representation of quantum information. However the goal is to replicate the behavior of the quantum algorithms when executed on a real quantum computer. This includes the effect quantum noise at the gate-level. While classical emulation of quantum circuits faces the same problems as in the case of simulation (i.e. significant growth in resource consumption) the added challenges are to perform computation in parallel and also recreate extrinsic factors such as quantum noise. Simulators have been developed that employ a programming language to describe quantum algorithms [10],[11] emulation requires that the quantum algorithm is described in terms of gates and quantum bits. Other simulators such as [12], [13] allow construction of quantum circuits using just gate interconnections, however they do not provide the level of performance as the description language-based simulators such as [10].

1.2.1 Motivation for Emulation of Quantum Circuits

Quantum circuit emulation allows development of quantum algorithms in a more comprehensive manner as compared to simulation. Emulation goes beyond just mathematically

replicating a quantum algorithm but permits insight into more complex issues facing quantum computing such as quantum noise and quantum gate error. Furthermore, hardware-based emulation allows more control over the parameters of emulation such as word length of data-primitives as well as allows computational optimizations at the gate-level that are difficult to achieve in software. This in turn leads to a deeper understanding of issues facing classical modeling of quantum circuits. Finally, hardware-based emulation provides a significant performance improvement compared to software simulators.

1.3 Thesis Contribution

This thesis proposes a gate-level FPGA-based solution to quantum circuit simulation. Using the analogues between the quantum and classical circuits, an emulation environment has been created where various issues such as quantum noise and parallel computation are taken into account. The emulator outperforms the premiere software simulators when considering algorithm runtime. Also combining classical error analysis techniques and relating them with quantum noise suffered by actual quantum computers, a bound on word length of data-primitives has been derived such that the emulated circuit produces results similar to those of a real quantum computer.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 provides a detailed background of the quantum circuit model including the different mathematical notations used to describe quantum computation. This is followed by an overview of different simulation techniques that have been developed for quantum circuit simulation.

Chapter 3 presents an in-depth look at the emulator architecture. Various challenges and hurdles were encountered in creating the emulation environment and details on how these challenges were over come are also presented. Finally, a thorough derivation of the word length bound is also presented.

Chapter 4 evaluates the performance of the performance of the FPGA emulator and compares it with other software simulators. A practical vindication to the theoretical work presented in Chapter 3 is also provided.

Lastly, Chapter 5 summarizes the work and presents the conclusion to the thesis. Directions for future work are also suggested.

Chapter 2

Background

The quantum circuit model provides an algorithmic abstraction for the quantum computation process. The model transforms the quantum physical phenomena that occur during the entire computation or "evolution" of the quantum system into lumped discrete events. A quantum system's time evolution that is normally represented using Schrodinger's equations (a collection of partial differential equations) is no longer required when using quantum circuits [9],[14]. Instead, the quantum information is represented as a state space (also referred to as a state vector) that depicts the state of the entire system at any given time. Evolution of states is then modeled using a collection of linear transforms that appropriately manipulate the state vector.

Information stored in a quantum system grows exponentially with the size of the system (number of quantum particles) [9],[14]. Manipulating the large state space with PDEs can be very resource-intensive for a simulator. Devolving the quantum computation into linear transforms does provide some reprieve for simulators, but for large state spaces they still have to perform a significant amount of computation. This will be clear after reading the following sections.

2.1 Quantum Bits

Quantum bits or *qubits* are the fundamental units of information in quantum computing. Analogous to classical computing, quantum computing is the manipulation of information stored in qubits. Physically, qubits have been realized using electrons, trapped ions and molecules. The information is stored using the spin or polarization of these particles. For

simulation purposes though, the state of the qubit is represented mathematically by a vector in a finite-dimensional complex *Hilbert* space [9]. The major property of the Hilbert space is that all vectors belonging to that space have a well-defined inner-product. For a vector f belonging to the Hilbert space, the norm of f is defined by

$$|f| = \sqrt{\langle f, f \rangle} \tag{2.1}$$

The *Dirac* bra-ket notation is a commonly used notation for denoting the state of the qubit. The notation is quite commonly used in quantum mechanics since it can conveniently represent the state of the system as well as describe operations on the system effectively. In the bra-ket notation, elements of H are "ket" vectors given by $|x\rangle \in H$. A corresponding "bra" vector $\langle x|$ is an element of the dual space H^*[1]. For a quantum bit though the state $|\psi\rangle$ of the bit is represented using the following equation

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.2}$$

where α and β are complex coefficients related by the following expression

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.3}$$

The $|0\rangle$ and $|1\rangle$ states of a qubit can be thought of as analogues to the classical bit. Physically, $|0\rangle$ and $|1\rangle$ states refer to a particular spin/polarity orientation of the qubit. However, as can be observed from Equation 2.2 and Equation 2.3, the qubit can simultaneously be in the both $|0\rangle$ and $|1\rangle$ states (an arbitrary spin/polarity). This phenomenon is known as *superposition* and is one of the fundamental sources of speed-up in quantum algorithms [9],[14]. Geometrically, a qubit can be thought of as a unit vector and can have any value on a unit sphere. Thus, a qubit can be in an infinite number of possible states. A common pictorial notation for qubits is known as the *Bloch sphere* and an example of that is depicted in Figure 2.1. Mathematically, Equation 2.2 can be re-written as Equation 2.4

$$|\psi\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle \tag{2.4}$$

[1]The $\langle x|y\rangle$ notation is the inner product between the conjugate vector of x denoted as x^* and the vector y

Angles θ and ϕ are related to α and β through a transformation from Cartesian to polar coordinates over complex number fields.

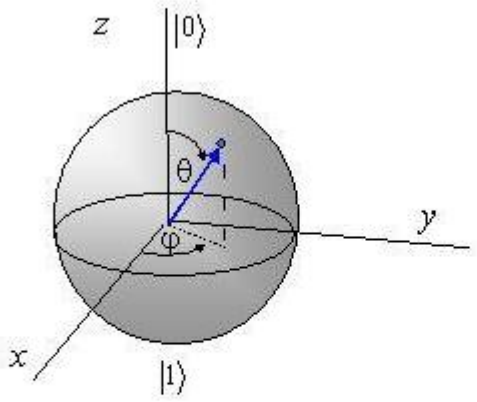

Fig. 2.1 Bloch Sphere

From a simulation point of view, only the values of α and β need to be stored and manipulated as they represent the information stored by the qubit. Thus, a single qubit can be represented as a 2×1 vector $[\alpha \ \ \beta]^T$. While the notation introduced so far is sufficient to describe the state of a single qubit, a more useful notation for representing the state of a $n-$qubit quantum system is as follows

$$|\Psi\rangle = |\psi_1..\psi_n\rangle = |\psi_1\rangle \otimes ... \otimes |\psi_n\rangle \tag{2.5}$$

where $\otimes$ represents tensor product operation. The resulting state space Ψ is a $2^n \times 1$ vector. This notation is useful as it collectively represents the state of the entire quantum system and assists in dealing with the computational interactions between the qubits themselves. The advantages of this representation will be encountered in the subsequent sections. However, an immediate disadvantage to this representation, from a simulation point of view, is that instead of dealing with $2n$ complex numbers for n qubits, the simulator now has to work with a state space of 2^n complex numbers. This leads to an exponential increase in computation resource consumption.

2.2 Entanglement

Quantum entanglement is perhaps the most exotic quantum phenomenon used in quantum computing. Entanglement represents a situation when two or more distinct quantum particles behave in such a way that when one of the particles is subjected to change, a similar change is instantaneously applied to all the other entangled particles. Einstein, Podolsky and Rosen were the first to notice this effect in a pair of photons whereby the change in the spin of one photon resulted in the change of spin in the other simultaneously.

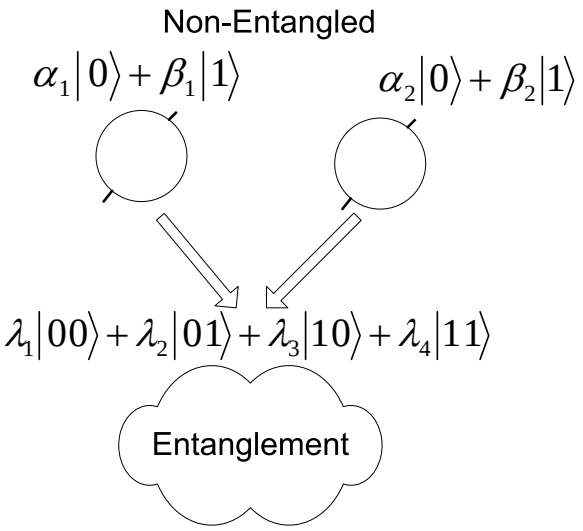

Fig. 2.2 Entanglement of two quantum particles

Mathematically, when a collection of qubits enters a state of entanglement, the notion of individual qubits disappears and the system can no longer be represented as a tensor product of individual qubit state vectors. In the case of n entangled qubits, the state of the system can only be represented using a $2^n \times 1$ vector. Thus, it becomes necessary to represent the quantum system in the expanded notation described in the previous section. For an entangled system, the computational basis states also grow exponentially. While for single qubits the computational basis states were just $|0\rangle$ and $|1\rangle$, for a two qubit entangled system the number of basis states expand to four: $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. However, for entanglement to occur the system can only be in a superposition of the $|00\rangle$ and $|11\rangle$ or $|01\rangle$ and $|10\rangle$ states. In other superposition scenarios, the expanded state can be resolved into the tensor products of individual qubit states and hence the system can no longer said

to be entangled.

2.3 Quantum Gates

The quantum evolution of the system is modeled by applying a sequence of transformations on the quantum state space. The transformation or operations on the quantum system are called *quantum gates*. Classically, gates such as AND, OR and NOT are well known and any classical computation can be broken down into the operations these gates provide. In the quantum computation domain, all quantum algorithms can be broken down into constituent quantum gates that manipulate information and can deal with one or more units of information at one time (similar to multiple-input classical gates).

Mathematically, quantum gates can be represented as unitary matrices (operators) that can be applied to the state space or qubit vectors.

2.3.1 Single Input Quantum Gates

Single input quantum gates are transformations that can operate on a single qubit in the system. Physically, these operations translate to altering the spin/polarity of quantum particle, thus changing the information stored by the particle. For instance, in the case of the trapped-ion implementation, the spin of the ion is changed using laser-ion interactions [5] and this operation can be modeled by a 2×2 matrix.

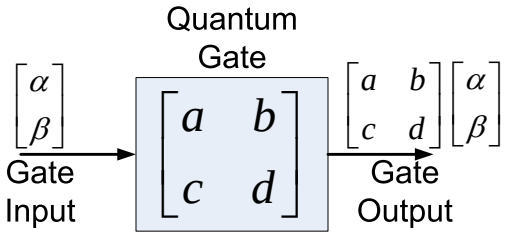

Fig. 2.3 Single-Input Quantum Gate

One of the most important single qubit gates is known as the Hadamard gate or the *H-gate*. The H-gate transform is given as follows

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

The H-gate's importance is paramount in that this operation can be used to place qubits in a state of superposition from a $|0\rangle$ or $|1\rangle$ states. As described in the subsequent sections, many quantum algorithms achieve their speed-up due to the fact that they can operate on multiple quantum states (achieved through superposition) in parallel. Physically, a collection of quantum particles can be prepared in a known state of spins/polarities and the particles are then subjected to the Hadamard transform in order to place them in superposition.

Other common single input gates are the phase rotation gates that can alter the rotation of the qubit by an angle of θ, the quantum NOT or *X-gate* that swaps the value of α and β of a qubit and the *Z-gate* that flips the sign of the β value of the qubit.

$$Rot(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

However, single qubit operations are represented differently when using the expanded state space notation to represent the quantum system. For a n qubit quantum system, each gate operation has to be represented by a $2^n \times 2^n$ sized matrix. This matrix is unique based on the basic operation that the gate performs and the position of the qubit. To elaborate, for a 3 qubit quantum system, if the H-gate is to be applied to the second qubit at any time the matrix representation would be calculated by performing $I \otimes H \otimes I$ operation. If the same operation had to be applied on the third qubit instead, the resulting matrix representation would be performed using $I \otimes I \otimes H$ and these two operations result in two different matrices.

From a simulation point of view, the process of applying these operations on a state space becomes exceedingly resource intensive as the size of these matrices grows exponentially. While most of the quantum gates result in sparse matrices, the sheer size of the matrices itself causes significant problems in simulating large quantum systems.

2.3.2 Multiple Input Quantum Gates

Multiple input quantum gates perform an operation on an input qubit based on the value of the other inputs to the gate. Thus, multiple input quantum gates are controlled operation gates, where the value of the control qubits determines the gates' operation on the data qubit. A special type of multiple input quantum gate is the "swap" gate that swaps the information stored in two qubits.

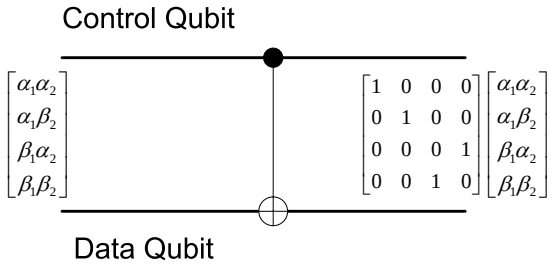

Fig. 2.4 2-input CNOT gate

The input to a multiple input quantum gate is mathematically the expanded representation of all the individual quantum states. Hence, a n input quantum gate's transform is represented by $2^n \times 2^n$ matrix. One of the most important quantum gates is the *controlled-NOT gate* (CNOT). In the simplest case, a two-input CNOT gate is represented by the following matrix

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Basically, if the control qubit is in the $|1\rangle$ state, the gate swaps the α and β values of the data qubit. However, if the control qubit is in the state $|0\rangle$ the inputs are left unchanged. While this description of the gate's behavior works for the case where the control qubit is not in superposition, it fails to describe the behavior of the gate in the situation when the control qubit is in superposition. The behavior of the gate is expressed fully however, by the above matrix that operates on the expanded state space. No decision making is involved and the operation of that gate in this case is to swap the third and fourth entries

of the input state vector. The CNOT gate is however special, because in the case where the control qubit is in superposition and the data qubit is not, the resulting output is entangled. This can be understood by carefully examining the CNOT transform and a similar scenario can be created for generalization of CNOT gates (Toffoli gates). Due to the superposition states of the inputs, the quantum CNOT gate is thus more complex to its classical counterpart as its behavior needs to be expressed using the expanded state space.

Multiple input gate transforms also have to be adjusted to deal with the expanded state space of the entire quantum system. Each gate transform depends on the position of the inputs and overall size of the quantum system. For controlled input gates, it is computed using the *base transform* which is, for example in the case of CNOT is the X-gate transform. Essentially, the control gate transform is constructed by performing the Kronecker product between the base transform and the identity matrix. The resulting control gate transform is then further expanded using the Kronecker product with the identity matrix to adjust its size such that it can be applied to the entire state space.

2.4 Quantum Measurement

Quantum computation is probabilistic. Classical probabilistic circuits output a probability distribution based on the inputs of the circuit, the network topology and the induced probability distribution of the gates in the network. Therefore, in probabilistic computing, the results of computation cannot be accurately determined every time the outputs are subjected to measurement. This leads to probability of error in the measurement. Conversely, deterministic circuits are such that the results of the computation can be measured without error. In classical computation, deterministic circuits are mainly used and measurement error is not considered.

The probabilistic nature of quantum computation, thus, differentiates it from classical computing significantly. Quantum algorithms have to be designed such that the results of the computation can be achieved with low measurement errors and often the computation has to be repeated a considerable number of times before the results can be measured with high enough probability. This advertently affects the performance of the algorithm. Furthermore, quantum measurement or the act of extracting information from a quantum system is an irreversible operation as it destroys the quantum system being measured.

An intuitive way of understanding quantum measurements is to think of them as a

collection of operators on the quantum system (irreversible operators) such that the state of each quantum can be distinguished with some probability. These operators are projection matrices that map information from the quantum to the classical domain with some measurement error. After measurement, the qubit loses superposition and is measured in either of the computational basis states. The choice of these operators and the complexity of the quantum system determine the resulting measurement errors.

In the simplest case of a single qubit system, the $|\alpha|$ and $|\beta|$ coefficients from Equation 2.3 represent the probability that the qubit will be measured as $|0\rangle$ or $|1\rangle$ state respectively. For larger systems, the problem of quantum measurement becomes more complex and have a fundamental impact in quantum algorithm performance. In Appendix A, more details about simulation and optimization of quantum measurements are provided.

2.5 Quantum Algorithms as Quantum Circuits

Quantum circuits are a collection of wires (qubits) and gates that depict the time evolution of a quantum algorithm. The qubits are prepared in a known state and introduced as inputs to the system. The qubits then undergo evolution depicted by the gate operations on them. The evolution ends when the system is subjected to a quantum measurement. A number of quantum algorithms have been developed in the recent few years and they have sparked great interest in the field. In this section, an overview of two of the most famous quantum algorithms is provided.

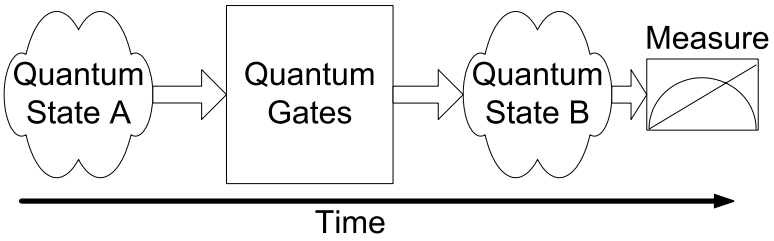

Fig. 2.5 Quantum computation as modeled by quantum circuits

2.5.1 Quantum Fourier Transform

The quantum Fourier transform (QFT) plays an important role in the phase-estimation algorithms and the Shor's factorization algorithm [15]. While the QFT itself does not provide

a speedup in performing the Fourier transform on quantum information, its importance in other quantum algorithms makes it an interesting case-study.

For a n qubit system in state $|j\rangle$ is subjected to the QFT, the output is given by

$$|j\rangle \rightarrow \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi ijk/2^n} |k\rangle \tag{2.6}$$

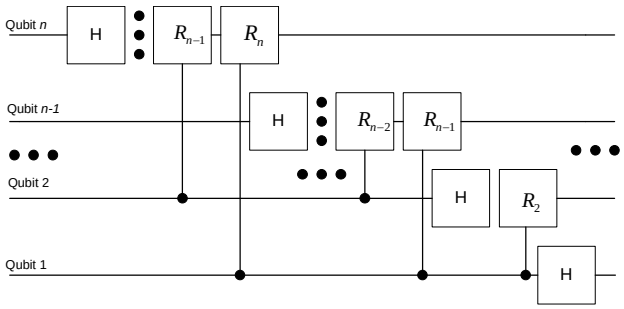

Fig. 2.6 The quantum Fourier transform circuit

Figure 2.6, depicts the QFT circuit for a n sized quantum system. The R_k gates can be represented as follows

$$R_k \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$$

2.5.2 Grover's Search Algorithm

The Grover's search algorithm is the fastest search algorithm at the time of writing. It performs the search on a n sized database at the worst-case complexity of $O(\sqrt{n})$ [16],[9] and can generally speed-up many classical algorithms that use searching or route-finding techniques. Figure 2.7, illustrates a 3 qubit Grover's search algorithm.

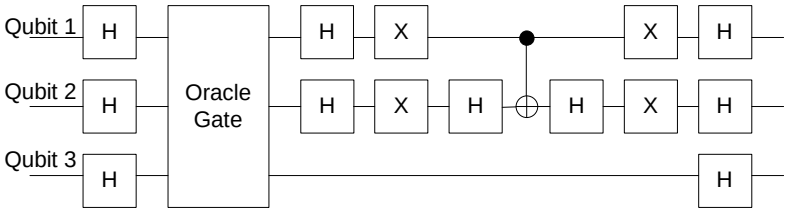

Fig. 2.7 Three qubit Grover's search algorithm

The oracle shown in Figure 2.7 is responsible for providing the answer to the search query. If the query is present in the first n qubits, then the Oracle sets the oracle qubit to $|1\rangle$ state, otherwise the qubit is set to the $|0\rangle$ state. The oracle itself can be constructed in this case using a CNOT gate with inputs according to the search query. Figure 2.8, shows the four different oracles for the three qubit Grover's circuit shown in Figure 2.7

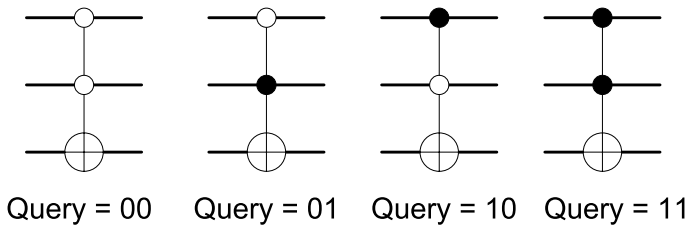

Fig. 2.8 Oracles for the three qubit search algorithm

2.6 Software-based Simulation Techniques

In this section, a brief summary of three software-based simulators is given. The choice of the simulators is based on the fact that all three use very different simulation approaches. At the same time, they also represent the evolution of simulation techniques that have led to the development of the FPGA-based emulation technique. Quantum circuit simulators perform the mathematical operations of quantum gates on a given set of qubit states. The simulators are exclusively software-based.

2.6.1 QuCalc: Mathematica Simulation Library

QuCalc is a simulation library written in Mathematica [13]. It makes use of the numerical and symbolic power of Mathematica to describe quantum circuits and simulate them. QuCalc uses the standard algebraic approach to simulate the quantum circuits. The gates making up the circuit are described symbolically as a multi-dimensional Mathematica array. The transforms of those gates are then concatenated together by perform matrix-matrix multiplication and tensor products to create an overall transformation of the circuit. This becomes the more computationally expensive part of the simulation. The circuit transform is a matrix that can then be applied to an input state vector. In Figure 2.9 a description of how QuCalc can be used to construct a circuit transform is provided.

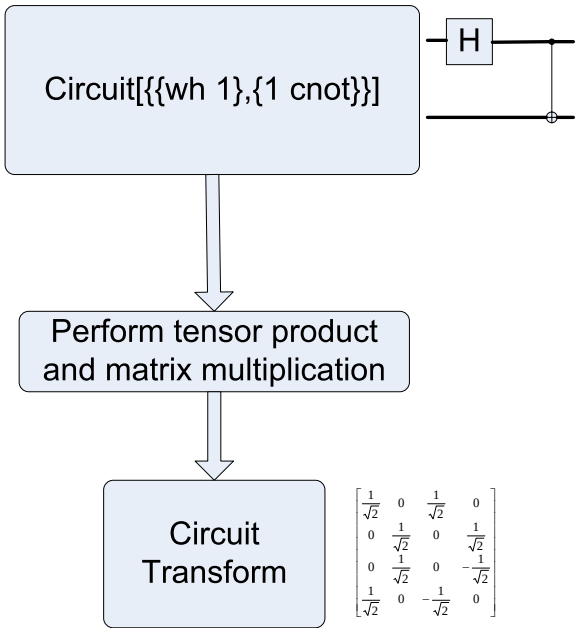

Fig. 2.9 Circuit simulation using QuCalc

QuCalc is extremely straightforward to use, and construction of the circuits is very intuitive. The gate positions in the circuit can have a one-to-one mapping to their position in the array description in QuCalc. However, building very large circuits this way is very cumbersome and no mechanism is provided to facilitate that function.

In terms of performance, the simulator is limited by Mathematica as the simulation environment. As Mathematica is a symbolic computing package, a significant overhead is in the computation. While no specific benchmarks are provided for QuCalc, it is understood that its purpose is not to simulate large quantum circuits efficiently. QuCalc is a simple tool that can allow the construction of small to medium sized quantum circuits and provides a strong environment to algebraically and symbolically deal with issues pertinent to quantum circuits.

2.6.2 QuIDDPro: An ADD based Simulation System

QuiDDPro is one of the fastest software simulators (at the time of writing) that is based on describing quantum transforms in terms of decision diagrams. The circuit is described using a Matlab-like programming language. QuiDDPro's novelty is that it represents the matrices and vectors that depict quantum transforms and quantum information as *Quantum*

Information Decision Diagrams or QuIDDs. A QuIDD is essentially an algebraic decision diagram (ADD), that have commonly been used to logically describe classical circuits [10], [17].

Since ADDs are designed to optimize binary logic circuits, the QuiDDs require additional properties to accommodate representing quantum information. First, unlike ADDs, a QuIDD's terminal nodes can have complex values (ADDs used for binary circuits can only have terminal values of 0 or 1). Secondly, in order to optimize operations on QuIDDs, the structure does not explicitly contain the complex values in the terminal nodes, but actually stores indices to an array that contains the actual terminal node values. This reduces computational overhead when performing operations on QuIDD-based vectors and matrices. Finally, the ordering of the nodes in QuIDD is such that it favors compression of block patterns in matrices. This optimization arises from the fact that the tensor product of quantum transforms produces highly regular block patterns. By ordering the nodes properly, the redundancy in the block patterns can be overcome and a smaller decision diagram can be used to describe the matrix.

An illustration of how a QuIDD can store a vector representation of a quantum state is shown in Figure 2.10. Here it can be observed that a 4×1 vector is represented using 3 nodes. A slightly involved QuIDD is used to describe the application of the Hadamard transform to a two-qubit vector as shown Figure 2.11. Here the left most QuIDD represents the Hadamard transform on a 2-qubit system. The next QuIDD represents the 4×1 vector representing the 2-qubit system. The result of the multiplication can then be stored in just one terminal node.

While the information of the quantum circuit is stored using decision diagrams instead of regular matrices/vectors in QuIDDPro, the actual construction of the circuit transform and obtaining outputs is performed using tensor products and matrix multiplication as in the case of QuCalc. However, the advantage using QuIDD [10] is that the tensor product and matrix multiplication operations are quite efficient and having a reduced representation such as QuIDD can lead to significant performance gains. For matrices represented by nodes a and b, the tensor product comes out to be $O(ab)$ while matrix multiplication is $O((ab)^2)$.

QuIDDPro has a Matlab like feel and can execute scripts written in a syntax similar to Matlab. It supports all necessary quantum transforms, however the construction of a quantum circuit is not immediately intuitive. Quantum algorithms are constructed using the functions provided in QuiDDPro in the form a programming language instead of a

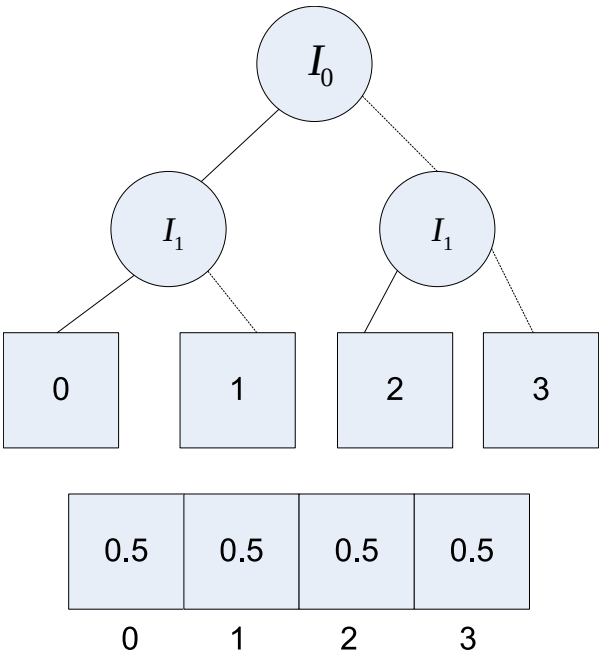

Fig. 2.10 Vector description in QuIDDPro

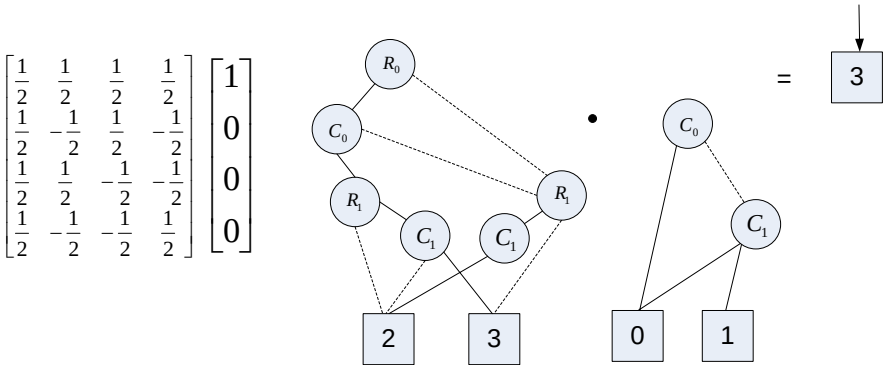

Fig. 2.11 Matrix multiplication example using QuIDDPro

netlist. However, once the language issues are overcome, very large quantum circuits can be constructed quite easily. QuiDDPro's creators have provided detailed runtime performance of the system when simulating the Grover's search algorithm. From [10], QuIDDPro can simulate the Grover's search algorithm with a memory complexity of $O(n)$ and time complexity of $O(R|A|^{16}n^{14})$ where R is the number iterations of the algorithm, A is the number of nodes in the oracle representation and n is the number of qubits. Running times provided in [17] for the Grover's search algorithm are less than 6 minutes for simulating a 20 qubit circuit.

2.6.3 HDL Based Simulation of Quantum Circuits

Simulation of quantum circuits using a hardware description language such as VHDL [18] has been proposed in [19]. This technique uses analogies between the quantum circuit model and classical circuits to construct and simulate the quantum circuits.

HDL simulation of quantum circuits diverges from the strongly algebraic approaches in the previous two simulators. As the quantum circuit model breaks down the quantum transform into quantum gates and quantum bits, the HDL approach can incorporate these architectures directly as they are congruent analogues to classical gates and bits respectively.

Qubits are described as two complex numbers (using the *real* keyword in VHDL). The gates are constructed with qubits as inputs and outputs and the transform of the gate is described using behavioral VHDL. The quantum circuit can be constructed now by combining the gates using structural VHDL.

However, the key issue here is entanglement. The previous two simulators dealt with the quantum circuit by expanding it to the full bases states and applying tensor products to achieve the final circuit transform. The expansion operation explicitly deals with entangled situations since the computational state space is already large enough to hold the entangled information. In the case of HDL simulation however, the goal is to describe the circuit in the simplest gates possible, and thus the presence or absence entanglement has to be explicitly detected and simulated. In [19], detailed *entanglement extraction* algorithms have been derived. They are able to detect when entanglement occurs or disappears. These are simulated using functional VHDL and have exponential complexity. This amounts to similar resource usage as in the case of expanded state space simulators.

Since the simulator uses VHDL to simulate quantum circuits, the CAD simulation tools available for the development of classical circuits using VHDL can now be employed to construct quantum circuits. This involves either using structural VHDL or a graphical netlist creation tool. In both situations, it is straightforward how to create the circuit and simulate the results.

According to [19], the runtime in the non-entangled case of simulation is $O(n^2)$. Some simulation runtimes are also quoted in [19] for small quantum circuits. However, entanglement extraction is still complex. Overall, the simulation would also be limited to the speed of the VHDL simulation tool.

Chapter 3

FPGA Emulator Design

This chapter provides details of the emulator design and design choices. The challenges involved in performing hardware emulation of quantum circuits are laid out. This is followed by a description of how these challenges were overcome.[1]

3.1 Challenges in Emulating Quantum Circuits

While the underlying principles of classical circuits are well understood, and have been widely applied, the same is not true for quantum circuits. So far, quantum circuits have been rarely demonstrated and in rather small sizes. Although existing quantum computing machines employ at most 7 qubits, even such a small number of qubits are notoriously hard to analyze and employ in concerted quantum machines.

Simulating quantum circuits in software is even more cumbersome than in the classical case. Currently available simulators like [19],[10] while providing an environment for the development of quantum algorithms, still greatly abstract the quantum evolution of the qubits. The speed of running a single simulation pass is even more critical than for classical circuits. Due to little understanding of hardware modeling of quantum processes, it is very likely that that many changes will need to be introduced to the model itself. This, in consequence, would lead to several re-runs of simulations of a given quantum design. Therefore emulators which perform fast simulations, and can be easily re-programmed to account for small changes in the model of quantum processes as well as the design itself are

[1] Part of the work presented here has already been published in [20]

essential. Such capabilities can be provided by FPGA-based hardware emulators. Finally, using the quantum circuit model is a very intuitive way of constructing quantum algorithms and would greatly assist in the development of new algorithms. Thus, the emulator combines important quantum computing concepts and a solid development environment to serve as a development and test-bed for quantum algorithms.

The first and foremost problem with quantum circuit emulation, is the mapping of quantum computation concepts to the digital domain. While the quantum circuit model creates a classical analog for quantum computing, it still possesses appreciable differences from classical circuits. The goal is to efficiently describe quantum circuits using a classical hardware description language, such that the final circuit can be synthesized on a FPGA.

The second major challenge is to emulate the quantum circuit such that the resources available on the FPGA are used as efficiently as possible. Resource requirement becomes a serious issue with entangled systems as performing quantum evolution in a highly entangled system requires an exponential amount of computation. Emulating quantum circuits with FPGAs has an advantage that FPGAs have a large amount of logic cells (and multiple FPGAs can be combined for even bigger circuits) and therefore large quantum circuits with entangled states can be emulated easily and efficiently in terms of computation time.

Finally, the overall architecture should still emulate the parallel evolution of the quantum system as closely as possible since that is one of the main motivations for performing quantum circuit emulation in hardware. Thus, balancing the resource usage and parallelism in the emulator is an important design consideration.

3.2 Emulation Overview

The overall design process is illustrated in Figure 3.1. Quantum circuits are constructed from the quantum gate descriptions that are part of the emulator. The correctness of the circuit can be verified either by software simulation or by FPGA emulation. Thus, a technique has been developed for modeling quantum circuits using VHDL and then synthesizing the circuit in hardware to achieve performance needed to make the whole process more practical.

The emulator comprises of two major components. The first is a C++ based command-line interface that has been created and is used to generate VHDL description of the gates required in the quantum circuit. The interface allows a fast way of calling the C++ based

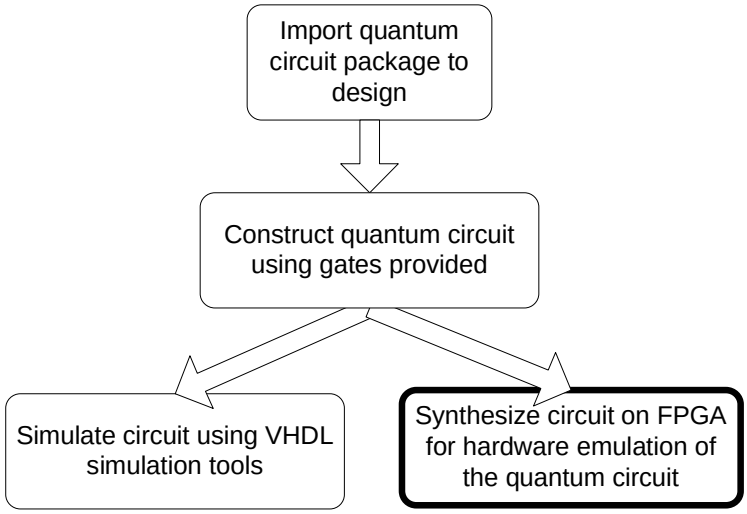

Fig. 3.1 Modeling quantum circuits using the VHDL quantum gate library

functions that generate gate descriptions for gates such as the X-gate, Z-gate, Hadamard Gate, Rotation gates and common multiple input gates such as the controlled version of the single input gates and the swap gate. The interface requires the user to enter the gate name, system size and number of inputs and it then generates the appropriate VHDL description of the gate. All the gates created from the interface use the expanded state space notation for inputs and outputs. More details about the developed software are provided in the subsequent sections.

The second part of the emulation environment are the data primitives defined in VHDL and the final circuit structural description. The data primitives are used to define complex number-based entities such as the qubit state vectors and the expanded state space notation. While all the gate descriptions can be generated via the software component of the emulator, the final quantum circuit model has to be created by either using structural VHDL or a schematic environment such as the one present in Altera's Quartus II software.

The emulator's architecture overview is illustrated in Figure 3.2. Emulator circuitry on the FPGA closely follows the quantum circuit topology as a series of gates are applied to the qubits on the FPGA in the same order as the quantum circuit itself. However, at certain places in the quantum circuit a register is inserted and this sequence of registers allows the pipelining [21] of the entire computation. All these registers are controlled by a single global clock and the speed of the clock depends on the slowest stage [21] of the

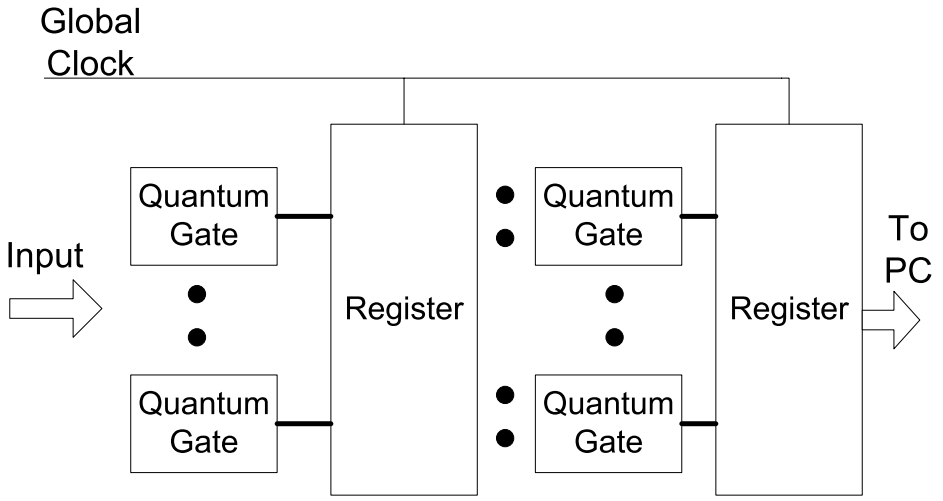

Fig. 3.2 Emulation architecture overview

quantum circuit pipeline. The registers store the state of the quantum system at a given point and time of the quantum evolution. The clock serves to synchronize the computation and can be used to determine the total computation time.

Furthermore, the pipelined architecture allows the quantum circuit to be quickly simulated for multiple inputs thus providing a significant advantage over software simulators where the circuit has to be re-simulated for each set of inputs. Finally, at the end of the computation the final state of the quantum system is transmitted from the FPGA to a PC. Details about each component of the architecture are provided in the following sections.

3.3 Data primitives

The first step in creating the emulator is to develop a mechanism for representing quantum information digitally using VHDL. From Equation 2.2, the information of the qubit is stored in two complex numbers α and β. In VHDL, there are no native data primitives to deal with real numbers (floating or fixed point). Thus, the choice that had to be made was to represent the complex numbers using floating point or fixed point schemes. Initially, a floating point representation was considered but it was discarded because the emulator would have to perform considerable amount of floating point operations and it would be very resource consuming to have many floating point adders or multipliers in hardware. An architecture based on hardware reuse was also considered but that reduced the parallelism

in the emulator significantly.

The other choice was to develop a fixed point scheme for representing the real and imaginary parts of the complex numbers. The fixed point scheme yielded less arithmetic overhead at the cost of lower precision. Figure 3.3 depicts the fixed-point representation that was chosen for the emulator. There are two salient features of this representation

1. According to Equation 2.3, $|\alpha|$ and $|\beta|$ can have a maximum value of 1. Therefore, an extra bit is used in the representation scheme (the second most significant bit) to represent the case where the α or β have a value of 1 or i unambiguously and without discretization errors.

2. The length of the mantissa bit can be varied. The size of the data primitives has a direct relationship with the amount of resources (logic cells or LCs) consumed on the FPGA; the designer has the ability to trade-off resource usage against precision.

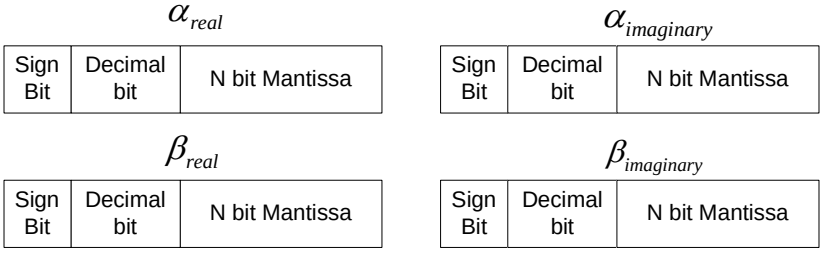

Fig. 3.3 Fixed-point quantum bit representation

Having a variable sized mantissa allows a certain amount of flexibility to the emulator. Depending on circuit size, precision requirements and available resources, the synthesized quantum circuits can be "tuned" to fit these parameters. It also opens the avenue for experimenting with circuit precision and modeling quantum noise that are difficult to perform with software simulators. In [22],[23] quantum noise simulation has been suggested and later in this chapter more details about error analysis and quantum noise simulation are provided.

The following code enlists all the data primitives in VHDL. Notice that in order to vary the system precision, only the value of the constant N is changed and the circuit is recompiled.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE quRecords IS

    CONSTANT N          : INTEGER := 10;

    TYPE complexNum IS RECORD
        real        : STD_LOGIC_VECTOR(N-1 downto 0);
        imaginary       : STD_LOGIC_VECTOR(N-1 downto 0);
    END RECORD;

    TYPE quBit IS RECORD
        alpha       : complexNum;
        beta        : complexNum;
    END RECORD;

    TYPE complexArray IS ARRAY (integer range <>) OF complexNum;

    TYPE entangledQubit IS ARRAY (integer range <>) OF complexNum;

    TYPE quArray IS ARRAY (integer range <>) OF quBit;

    TYPE result IS array (integer range <>) of std_logic_vector(twoN-1 downto 0);
END quRecords;
```

3.3.1 Emulation of the expanded state space

As described in Chapter 2, due to entanglement, single-qubit evolution is not sufficient to describe quantum computation. The same fixed point scheme is used to represent the complex numbers in the expanded state space representation. However, expanding the state space causes an exponential growth in resources. Thus, the quantum circuit has to be synthesized such that the state space is expanded only when absolutely necessary. Originally, quantum emulation begins with the the inputs represented as initial values of the qubits. The expanded state space notation is introduced when a multiple-input quantum gate is encountered. If a qubit, already in an expanded state space, is an input to a multiple input gate, the other input qubits are included in the state space of the originally expanded state. In the case where multiple expanded state spaces are inputs to a quantum gate, the inputs are combined into a larger state space just as the in the second situation for state space expansion.

3.4 Expander Circuits

As shown in Chapter 2, state space expansion is a necessary operation during the emulation of quantum evolution. The two primary reasons for the inevitability of the use of this notation are:

- Entanglement situations can only be represented using a single vector using the state space expansion technique,
- The mathematical operation of multiple-input quantum gates is directly applicable to the expanded state vector.

The state space operation is essentially the implementation of the operation defined in Equation 3.1.

$$\bigotimes_{i=1}^{n} |\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle .. \otimes |\psi_n\rangle \tag{3.1}$$

where $|\psi_i\rangle$ can either be a qubit state vector or an expanded state vector. In terms of a general classical simulator, the state expansion operation presents two challenges:

- Implementation of the Kronecker product operations Equation 3.1 as a large amount of complex multiplications have to be performed,
- Emulation of expanded gate transforms that have to operate on the expanded state.

There are two scenarios of state space expansion: one from a set of qubits, and the second from a combination of state spaces. In the first case, the state space is expanded directly to its maximum size (2^n for an n qubit system). This operation requires $(n-1) \times 2^n$ complex multiplications to implement. Figure 3.4(a) describes the situation for state expansion from a set of qubits. Each λ_i is computed by cycling through all the combination of α and β values of each of the n qubits. Therefore, $n-1$ complex multiplications are performed for each of the 2^n entries of the expanded vector.

The second case allows the state space to be gradually expanded in consequence, a larger state space can be constructed by combining several smaller states. When combining a vector of size 2^m with another vector of size 2^n, the total number of complex multiplications required is 2^{m+n}. Figure 3.4(b) depicts the situation where two state vectors γ and ϕ are

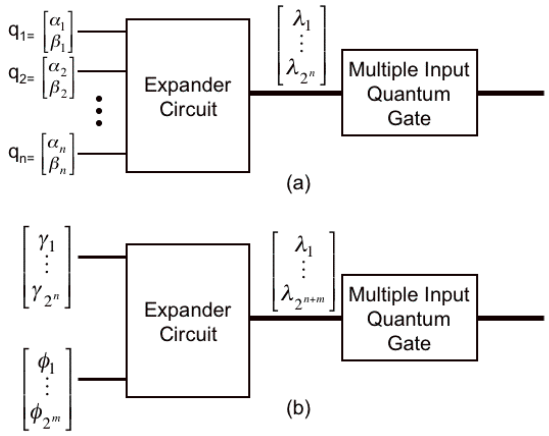

Fig. 3.4 Two cases of state expansion: (a) Set of n qubits (b) Two state spaces to be expanded into a larger one

expanded into a larger vector λ. Each of the 2^m entries of the λ vector are computed by multiplying each element of the γ vector by the entire ϕ vector.

The computational complexity is further exacerbated when even single qubit quantum gates have to be expanded so that they can operate on the expanded state space. Software simulators such as [13],[10] approach the problem by expanding the state-space at the beginning of the computation. In consequence, the rest of the computation involves operations on large matrices. Techniques involving the use of decision diagrams to reduce the complexity of the computation have been employed [10]; however operations on large circuits still require considerable time and memory usage.

State space expansion for purposes of FPGA emulation presents bigger challenges than those faced by any general classical simulator of quantum circuits. In essence, the FPGA emulator performs computation in parallel (by executing concurrent parts of the algorithm at the same time), thus mimicking quantum parallelism and gaining a significant speed-up over software simulators. Quantum gate transform matrices are normally sparse (in most cases diagonal or quasi-diagonal where a significant number of non-zero entries are equal to one and hence require no actual complex multiplication). Therefore, they are mapped to hardware directly, and operate on the input state space in a non-sequential way. However, this design strategy becomes imprudent and impractical when performing the state space expansion operation for even a small number of qubits. The complex multipliers needed to implement directly 3.1 consume an enormous amount of resources on the FPGA (in terms

of logic cells), and become impossible to synthesize beyond a modest number of input qubits.

Additionally, the expanded state space emulation also causes the quantum gate implementation to consume more resources. It is possible to re-factorize an expanded state space into smaller state vectors in the case where no entanglement is present. However it has been shown in [19] that the detection of the non-entanglement condition and factorization of the expanded state space require an exponential amount of computations themselves, and is non-synthesizable in hardware.

A further constraint on the state space expansion operation is imposed by the fact that it has to be completed within one pipeline stage (or split over several pipeline stages). In consequence, a sequential operation within a pipelined architecture has to be performed. In order to keep the circuitry straight-forward, the clock used in the sequential operation has to be in phase with the global clock of the pipeline registers. Therefore, the global clock cycle has to be long enough for the operation to complete, possibly augmenting with increasingly sequential implementations of the state space expansion operations. This in turn extends the overall computation time. The following inequality must hold true for the proposed architecture to operate correctly:

$$clock_{global} \geq M \times clock_{exp} \tag{3.2}$$

where M is the number of sequential operations needed to be performed for state space expansion, and $clock_{exp}$ is the period of the clock driving the expander circuitry.

There are multitudes of ways in which the expansion operation can be realized in hardware. Using more complex multipliers in the expander circuit reduces the number of sequential operations (M), hence a faster global clock can be used. The disadvantage of this approach is the increase in the resource usage of the expander circuit. The benefit of having a smaller $clock_{global}$ to $clock_{exp}$ ratio is that built-in clock multipliers on the FPGA can be used to obtain $clock_{exp}$ without actually slowing down the global clock. Depending on the size of M in Equation 3.2 and the clocking capacity of the FPGA, the expander clock is obtained by multiplying the global clock by M. Therefore, for a fraction of the resources that would be consumed if the state space expansion was to be performed non-sequentially, the sequential approach incurs no penalty in terms of computation time.

Finally, for an arbitrary sized quantum circuit, the architecture of the expander needs to be scalable and parametrizable, since its description has to be generated in software, just as in the case of quantum gates. The design philosophy is thus to design an architecture for the expander circuits such that they perform their tasks in a sequential manner, while balancing the resource usage on the FPGA and the overall performance of the quantum circuit. Furthermore, the state space is also expanded only when necessary (in the case where multiple-input quantum gates are encountered). Therefore, unlike software simulators, the state space is expanded gradually whenever possible and smaller representations for quantum gates are used when possible.

3.4.1 Sequential State Space Expansion

The expansion operation can be implemented sequentially in several ways. First consider an element that can be used to obtain the expanded state space from a set of qubits. Such a circuit would be useful when the qubits are subjected to multiple input quantum gates, Figure 3.4(a). The range of solutions is a function of the number of multipliers used to realize the operation in Figure 3.4(a). In the simplest case a multiply-accumulate (MAC) loop is suitable to handle this task, requiring a clock ratio $clock_{global}/clock_{exp} = (n-1) \times 2^n$ to expand n qubits. The resulting slow down of the global clock is significant even for modest n.

A more favorable solution from the perspective of the FPGA emulator is the one implementing the sequential operation using a ripple multiplier architecture. It was noted that the expansion operation from 3.1, can be broken down into 2^n sequential steps using $n-1$ multipliers in a ripple configuration instead of just one in the case of the MAC. Note that $n-1$ multipliers are needed since each entry of the expanded state vector is computed by performing $n-1$ complex multiplications between n complex numbers, which are the α or β values of the n qubits. The output state vector is formed from the 2^n possible combination of these values.

The algorithm depicted in Figure 3.5 depicts the operation of the ripple multiplier expander circuit. A *count* variable is used to cycle through all the possible combinations of α and β of the input qubits stored in a single dimension array *qubitarray*. Each combination of the complex values is then sent to the ripple multiplier architecture (lines 12 to 14) and the resulting output is stored in the expanded output vector *outstate* indexed by the *count*

variable. With this architecture, the $clock_{global}$ to $clock_{exp}$ ratio is 2^n, which is a significant improvement from the MAC architecture. Furthermore, it is straightforward to design a scalable version of this architecture (to be generated via software) that is parametric on the number n of input qubits. The major issue with the ripple multiplier architecture is a difficulty with its synthesis due to the large logic cone from inputs to outputs. The synthesizer (Leonardo Spectrum running on a PC with 2GB of memory) quickly runs out of memory when synthesizing the expander circuit for more than 9 input qubits. However, at the system level, it is noted that the majority of quantum circuits comprise of single- to three-input quantum gates. Therefore, this architecture can be used for expanding the state space for two to three qubits without any synthesis problem. Since the qubit state vectors have to be expanded when subjected to a multiple-input quantum gate, only two or three qubit expansions need to be performed in most cases.

```
1. expand_direct(start_vec,end_vec,n,qubitarray)
2. {
3. // expansion of vector qubitarray
4. // expanded state stored in outstate
5.       count := 0
6.       while (count not equal 2^n -1) {
7.         for i = 0 to n-1
8.           if (count(i) equals 0)
9.             out_count(i) := qubitarray(i).alpha
10.          else
11.            out_count(i) := qubitarray(i).beta
12.      out_mult := out_cont(0) * out_cont(1)
13.        for i = 1 to n-1
14.        out_mult := out_mult * out_cont(i)
15.      outstate(count) := out_mult
16.      count := count++
17.      }
18. }
```

Fig. 3.5 State space expansion using a ripple multiplier architecture

The proposed architecture, however, is not suitable to cover all these cases where expanded inputs are inputs to a multiple-input quantum gate (Figure 3.4(b)) or when a large quantum gate is encountered (in the case of an oracle circuit [9]). Another expander circuit architecture is therefore needed that can be used to combine expanded state spaces together to form a larger state space. To achieve these goals the gradual state space expansion architecture is proposed that has the ability to combine smaller state vectors into larger ones. Thus, the final state vector is obtained by performing multiple expansions.

3.4.2 Gradual State Space Expansion

The gradual state space expansion architecture faces similar design constraints as the expander architecture described above. The operation of this architecture is also sequential as this expansion requires a significant amount of complex multiplications. Therefore, the goal is still to find an optimal balance between resource usage and overall computation time, while at the same time have a scalable and parametric architecture.

```
1. expand_grad(state1,state2,m,n, outstate)
2. {
3. // state1 of length m and state2 of length n are
4. // combined to form a larger state vector outstate
5.      count := 0
6.      while (count not equal m-1) {
7.        out_cont := state1(count)
8.        count := count++
9.        for k = 0 to n-1 {
10.         out_mult(k) := out_cont* state2(k)
11.         outstate(count + k*m) := out_mult(k)
12.       }
13.     }
14. }
```

Fig. 3.6 Gradual state space expansion

The expander circuit takes two state space vectors as inputs as shown in Figure 3.4(b). One of the state space vectors can be simply a 2×1 qubit state vector. For example, consider a situation where $state_1 = [\lambda_1 ... \lambda_m]$ and $state_2 = [\gamma_1 ... \gamma_n]$ are the input vectors to the expander circuit. The algorithm in Figure 3.6 illustrates the steps for the gradual state space expansion. The controller again comprises of a counter *count* (lines 5 to 10) that is used to cycle through all elements of the state vector with length m. Then n complex multipliers, each with one entry from the $state_2$ vector, compute n entries of the output state vector (lines 12 and 13). The operation of the circuit can be summarized as: $outstate(count + k \times m) = \gamma_k \lambda_{count}$, where $k \in \langle 0, n-1 \rangle$. Figure 3.7 illustrates the architecture of gradual state space expansion example where a two qubit expanded state space is combined with a third qubit ($m = 4$ and $n = 2$).

The choice of the number of multipliers used in this architecture is determined experimentally based on observing various quantum circuit topologies at the system level. In many cases such as the quantum Fourier transform (QFT) [9], adder [24] and Grover's oracle circuits [16], a large state vector combines with a smaller state. The tradeoff be-

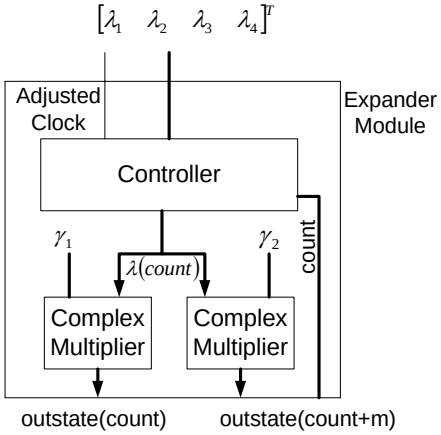

Fig. 3.7 Gradual expansion of 3-qubit state space with two qubits already expanded

tween resource constraints and circuit performance leads to two choices. One represents a fast design, with the number of multipliers equaling the length of larger state vector (γ in Figure 3.4(b)). In the alternative approach, the saving of the resources is obtained through the use of a smaller number of multipliers that equal the length of the smaller state vector (ϕ in Figure 1(b)).

The proposed architectures allow performing the expansion within a pipeline stage efficiently in terms of resources consumed on the FPGA and the penalty in overall computation time. It is scalable and parametrizable, thus satisfying all the requirements of the expansion circuitry. The major advantage of the proposed expansion techniques is seen at the system level. The notion of a gradual state space expansion provides considerable improvement in the performance of the emulator for many topologies such as the QFT and quantum adder circuits.

Figure 3.8 depicts the 3-qubit QFT circuit that uses the proposed state space expansion technique (the expander modules 1 and 2 are implementations of algorithms depicted in Figure 3.5 and 3.6 respectively). In this example, the state space is expanded in two stages permitting Hadamard (H1) and rotation (R2) gates to operate on smaller state spaces, thus saving a considerable amount of resources.

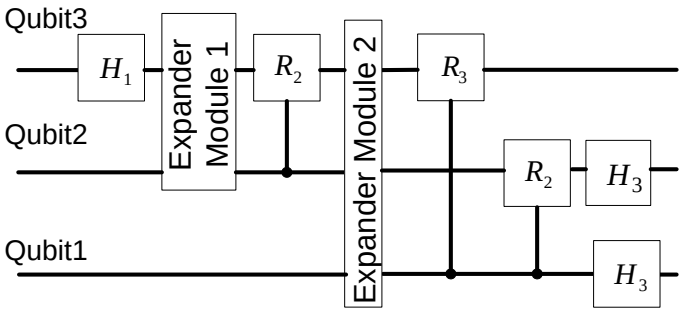

Fig. 3.8 3-qubit QFT Circuit with expander modules

3.5 Quantum Gates

In this section details about the optimization and design decisions with regards to the emulation of quantum gates are provided. An in-depth analysis of the design of three important quantum gates is undertaken in this section.

3.5.1 Optimizing Quantum Gates

The basic single input quantum gate descriptions can be easily created using the data primitives defined previously. Instead of performing matrix multiplication with the input state vector, the gates are designed such that they make use of the matrix structure and only the necessary complex multiplications are synthesized. For gates such as the X-gate or CNOT gate, the matrix elements are only 0 and 1. Furthermore, there is only one non-zero element in each row of the matrix. These gates can then be implemented without any complex multiplications since they simply swap elements in the state vector. The swaps can be performed by using simple structural VHDL commands. On the other hand, the Hadamard gate has a full matrix description and in this case the matrix multiplication is inevitable.

However, when these gates are applied to the expanded state space vector, the matrix descriptions are large ($2^n \times 2^n$ for a n-sized system) and usually sparse. Thus, it becomes necessary to implement the gates' operation on the state vector using the matrix structure and minimizing the number of complex multiplications. The following three case studies look at the implementation of the H-gate, CNOT gate and the controlled rotation gates to further illustrate the implementation of different gate architectures on the FPGA. It is

important to illustrate the implementation of these gates in detail since it would help to justify the performance analysis of the emulator presented in Chapter 4.

Emulation of the CNOT Gate

In the expanded state space notation, the CNOT gate has a similar matrix structure as that of a single input NOT gate albeit at a larger scale. The matrix describing a 3 qubit generalization of the CNOT gate (Toffoli gate) where the NOT operation is applied to the third qubit is as follows:

$$CNOT_{pos3size3}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

As can be observed from the above matrix, the operation of the CNOT gate can be implemented by performing a simple swap of the seventh and eighth entries of the input state vector. The following code illustrates this optimization:

```
USE WORK.QURECORDS.ALL;

ENTITY CNOTGATE3pos0numinput3 IS
PORT(
     input : IN entangledQubit(7 downto 0);
     output : OUT entangledQubit(7 downto 0)
    );
END CNOTGATE3pos0numinput3;

ARCHITECTURE CNOTGATE3_structural OF CNOTGATE3pos0numinput3 IS
BEGIN
     output(0) <= input(0);
     output(1) <= input(1);
     output(2) <= input(2);
     output(3) <= input(3);
     output(4) <= input(4);
     output(5) <= input(5);
     output(7) <= input(6);
```

```
    output(6) <= input(7);
END CNOTGATE3_structural;
```

Thus, in terms of resources used on the FPGA, the CNOT gate can be implemented with no resource consumption (see Chapter 4). This is particularly advantageous since generalizations of CNOT gates form the universal set of quantum boolean gates whereby any other boolean quantum gate can be constructed as a sequence of CNOT gates [25]. Particular circuits such as the quantum full adder comprise of CNOT gates only. Thus, by having a very efficient representation of the CNOT gate, a whole class of quantum circuits can now be efficiently emulated in hardware. While CNOT gates themselves do not incur a resource cost, they do have an initial cost in that they only operate on expanded state spaces. Therefore, the cost of the CNOT operation is in the implementation of expander circuits. However, due to the hardware-reuse approach used in the construction of the expander circuits, the overall operation of CNOT gates is very efficient in terms of resource usage.

Emulation of the Hadamard Gate

As described earlier, the H-gate is an important quantum gate and is commonly present in many quantum circuits. The following matrix describes the H-gate representation for a gate that is to be applied on the second qubit of a 3-qubit system.

$$H_{pos2size3} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

If the same operation is performed on larger state vectors, it is observed that each row still only contains two non-zero elements. Each entry in the output state vector of the gate is always the result of two complex multiplications and one complex addition. Using the matrix structure, the H-gate operation can be translated in a sequence of simple

complex multiplications and additions. Consequently, the Hadamard gate consumes far more resources than the CNOT gate for instance.

Emulation of Controlled Rotation Gates

The controlled rotation gates are used in the QFT circuit. For example in the 3-qubit implementation of this gate, where the first qubit controls the third one, the following matrix describes the gate's operation

$$CROT_{pos3size3}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{bmatrix}$$

In general, the rotation matrix is always a diagonal matrix, thus each matrix-row/state vector multiplication involves a single complex multiplication. The $e^{i\theta}$ term is expressed in its square form $e^{i\theta} = a + ib$ and is represented in hardware using the *complexNum* data primitive. This term is given as an extra input in the gate VHDL entity description. The VHDL description of this gate is significantly less resource intensive compared to the H-gate since far fewer complex number operations have to be performed.

3.6 Error Analysis and Quantum Noise Modeling

As mentioned earlier, the data primitives and gate coefficients are described using a fixed-point scheme. The fixed-point scheme introduces a significantly large data representation or *discretization* error compared to traditional floating-point representation schemes. While the fixed-point scheme has been developed such that it can represent quantum information effectively, the effect of the discretization error on quantum computation requires a closer look. Understanding the discretization error can also lead to the determination of the word length for a particular sized quantum system.

In an actual quantum computer, the quantum computation suffers from two major sources of errors: interaction of the quantum system with the environment that causes a disruption in quantum parallelism (decoherence) and inaccuracies in quantum gates [22]. While these errors are not present in their actual form while emulating the quantum circuit, the discretized error can be modeled such that the output results of the emulator are equivalent or no worse than those from an actual quantum computation.

3.6.1 Single Qubit Error

The two main sources of discretization error are the qubit representation itself and the gate coefficients. The error model of a quantum gate operating on a single qubit is depicted in Figure 3.9. Here, δ is the error in the input that is propagated and augmented with error ϵ, the discretization error of the matrix coefficients representing the given gate.

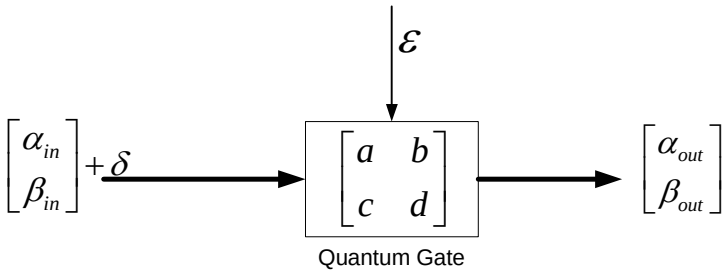

Fig. 3.9 Quantum gate error model

The error model can be expanded as in Figure 3.10. Then, the multiple sources of an error are added linearly. This model is used to evaluate the error at each gate in the network.

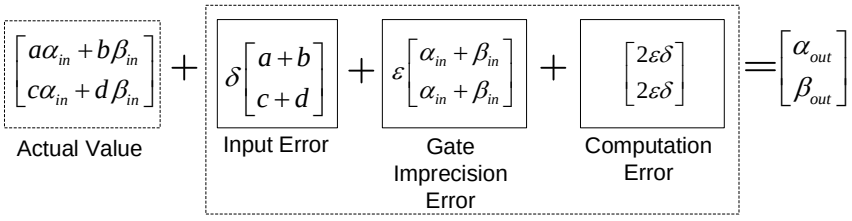

Fig. 3.10 Expanded gate error model

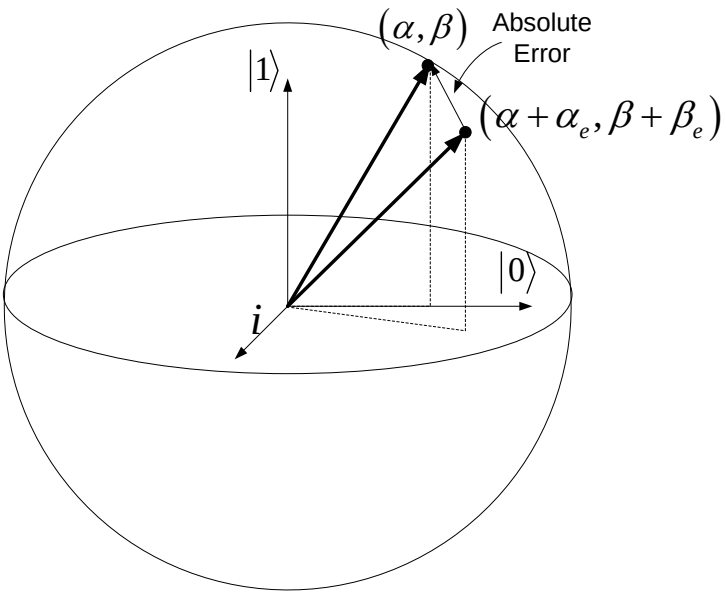

Fig. 3.11 Discretization error in a qubit

The absolute error E is thus,

$$E = \sqrt{\alpha_e^2 + \beta_e^2} \tag{3.3}$$

where α_e and β_e are described in Figure 3.10. These error values affect the probability of the qubit to be in $|0\rangle$ or $|1\rangle$ state when the qubit is subjected to a quantum measurement.

The qubit can be considered as a three dimensional unit vector in Figure 3.11 - while two dimensions are needed for α and β, the third dimension is attributed to the use of complex numbers. The representation error in the qubit is then given as the absolute difference between the true and discretized positions of the vector representing the qubit.

3.6.2 Expanded State Space Error

While the above description of error is sufficient for a single qubit evolution, when dealing with expanded quantum state spaces it is more complex to understand the notion of "closeness" between the actual and discretized state spaces. Furthermore, gate error also becomes more cumbersome as the transformation matrix grows exponentially in size.

The closeness between quantum states is defined as *fidelity* F [22],[23]

$$F = \| \langle\varphi|\psi\rangle \|^2 \tag{3.4}$$

In the case of analyzing error in two quantum states, $|\varphi\rangle$ is the actual quantum state and $|\psi\rangle$ is the quantum state subjected to error (discretization). The concept of fidelity is then involved in the gate evolution of the quantum system by defining an error rate per quantum gate [23]. The error rate is defined as

$$ErrorRate = \frac{1-F}{G} \tag{3.5}$$

where G is the total number of gates that the quantum state is subjected to. The gate error rate has been studied [26],[27] and for decoherence and gate inaccuracy reasons it was shown to be between 10^{-5} and 10^{-3}. The following analysis uses this concept of gate error to provide a relationship between word length, gate error and circuit size (in terms of total number of qubits). The above quantities can be used to deal with quantum noise and gate error rates dealing with quantum noise directly (in actual quantum systems) have been derived in [26],[22]. However, in the case of the emulator the only source of error in the computation is the discretization error. Therefore, a gate error model based on the discretization error introduced per gate is required.

The precision error in each word (real or imaginary part of the complex number) is 2^{-L} where L is the number of bits in the mantissa of the fixed point number (total word length is thus $L+2$). The precision error in the overall quantum number is therefore $\epsilon = 2^{-L}\sqrt{2}$. This can be trivially derived by considering the complex number as a two dimensional vector and calculating the magnitude difference between the actual and discretized complex number vectors. It can also be assumed as worst case, that each operation (multiplication or addition) adds this error to the complex number. From the previous sections, it can be observed that some gates such as the CNOT gate do not introduce any error in the state vector, while on the other extreme, gates such as the H-gate perform multiple operations on each complex number in the state vector. Therefore, a reasonable choice for gate error modeling, is such a gate that linearly adds ϵ to each term in the state vector. This is done by applying a rotation matrix that affects all values of the state vector. This could be a diagonal matrix containing a non-zero real number at each entry. This amounts to one multiplication on the real and imaginary part of each entry in the state vector per gate.

Consider a quantum system comprising of N qubits. The length of the full expanded

state space vector is thus $n = 2^N$. Let E be the final error in each term of the output state $\psi\rangle$ of the quantum system such that

$$E = \sqrt{2}G2^{-L} \tag{3.6}$$

Theorem 1. *The word length L required for simulating the quantum evolution of an N qubit system in the expanded state space is:*

$$(N+1)/2 \leq L \leq (N+35)/2. \tag{3.7}$$

Proof. Assume $|\varphi\rangle = [\lambda_1 ... \lambda_n]^T$ is the actual error-free output of the quantum circuit. Here λ_i are complex numbers such that $\sum_{i=1}^{n} |\lambda_i|^2 = 1$. The fidelity can now be computed between the state vector $|\varphi\rangle$ and $|\psi\rangle$ as follows

$$\begin{aligned} F &= \| \langle\varphi|\psi\rangle \|^2 \\ &= \lambda_1^*(\lambda_1 + E) + ... + \lambda_n^*(\lambda_n + E) \\ &= 1 + E(\lambda_1^* + ... + \lambda_n^*) \end{aligned}$$

The error rate can now be expressed using the above expression for F and Equation 3.6 as

$$\begin{aligned} ErrorRate &= max\left(\frac{1-F}{G}\right) = max\left(\frac{E(\lambda_1^* + ... + \lambda_n^*)}{G}\right) \\ &= max(\lambda_1^* + ... + \lambda_n^*)\frac{E}{G} \end{aligned}$$

The maximum value of the error rate occurs for $\lambda_i^* = |\frac{1}{\sqrt{2^N}}| = |\frac{1}{\sqrt{n}}|$ and can be determined by partial differentiating the equation $\sum_{i=1}^{n} |\lambda_i|^2 = 1$ for each λ_i. The error rate is thus,

$$ErrorRate = E\left(\frac{n|\frac{1}{\sqrt{n}}|}{G}\right) = \sqrt{2n}2^{-L} = 2^{(N-2L+1)/2} \tag{3.8}$$

From [22],[23] the above error rate maybe as high as 10^{-5} to 10^{-3}. Therefore, the following bound can be determined for the word length in terms of the number of qubits

$$\begin{aligned} 10^{-5} &\leq ErrorRate \leq 10^{-3} \\ 10^{-5} &\leq 2^{(N-2L+1)/2} \leq 10^{-3} \\ -17 &\leq \tfrac{N}{2} - L + \tfrac{1}{2} \leq -10 \\ \frac{N+21}{2} &\leq L \leq \frac{N+35}{2} \end{aligned}$$

□

3.6.3 Gradual Word length Expansion

The word length for an arbitrary sized circuit (in terms of number of gates) can be computed using the bound given by Equation 3.7. Note that the above analysis can be applied to other fixed- and floating-point representations by using the appropriate unit round-off expression. The following synthesis rules are derived from Equation 3.7:

- Quantum gates must be synthesized with the expanded precision representation depending on the size of the state space the gate is operating on,
- The output of the expander circuits should also use the expanded precision representation by padding the inputs with the necessary zeros before expansion.

The above rules along with the error bound allow the synthesis of the quantum circuit for a particular error requirement. For example, Figure 3.12 illustrates the word lengths needed to emulate the test circuit from Figure 3.8. The gradual word length expansion allows conserving resources by synthesizing gates operating on a reduced state space with a smaller word length. This enhancement however is modest. For a 3-qubit QFT circuit this optimization yields an improvement of 1.7%. Due to the relatively small change in word length with increases in circuit size, the improvement would grow less gradually with circuit size.

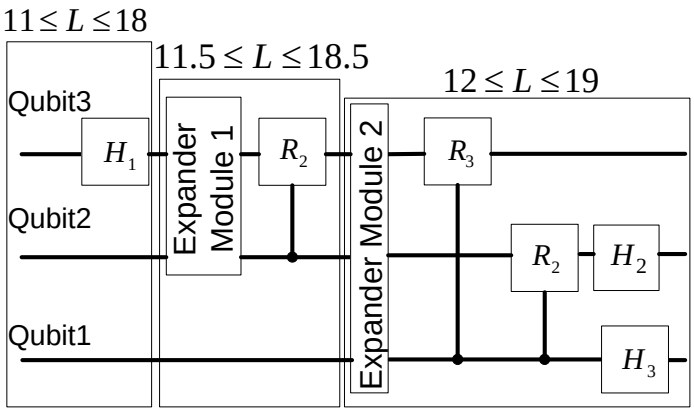

Fig. 3.12 Variation of word length with state expansion

Chapter 4

Implementation Details and Results

This chapter encapsulates several important consequences regarding the operation of the emulator. The particulars of the automated code generation tool to describe quantum gates and extracting emulation results from the emulator to the PC are laid out. More details regarding the choice of fixed-point numbers and their effect on quantum circuit emulations are provided. Furthermore, an ensemble of quantum circuits are created using the emulator to verify its viability as a potent emulation technology. Finally, the performance of emulated circuits is compared to other software-based simulators. All this would reveal that the emulation technique is not only an evolution of quantum circuit simulators but also opens up new avenues and insights into this very complex problem. [1]

4.1 Software-based Gate Generator

The large variety of gate transforms necessitates automation of the process of describing the gate transform in VHDL. To that end, a C++ application has been created that using command-line instructions from the designer, outputs the VHDL implementation of the required gate. Figure 4.1 depicts the UML class diagram [28] of the gate generator software.

C++ was chosen for the object-oriented encapsulation it provides and for the code reuse opportunities. Figure 4.1, depicts only some of the gates that can be created using the gate generator. All the quantum gates that can be generated using the software are child classes

[1] Part of this work has been published in [20].

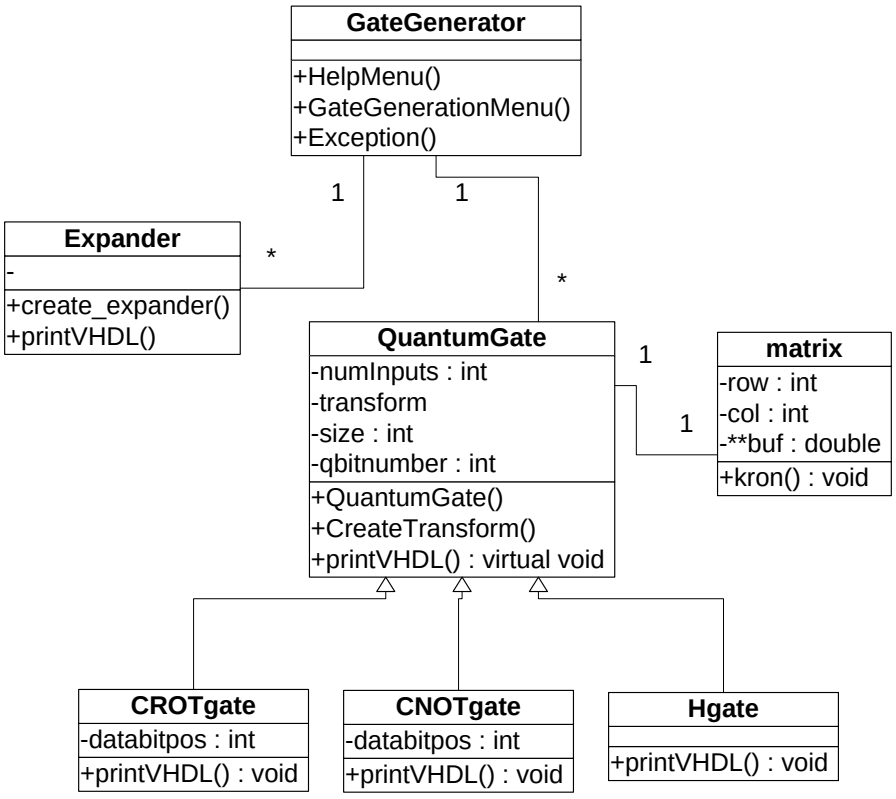

Fig. 4.1 UML class diagram for the gate generator software

of the *QuantumGate* class. This polymorphic structure is particularly useful, since it allows code reuse and also formalizes the addition of new gates to the generator. A new gate class would simply have to implement the *printVHDL()* function that would translate the gate's transform to VHDL.

The transform itself is created using the *CreateTransform()* function in the *QuantumGate* class. The transform is objectified using the *matrix* class. Depending on the *size*, *numInputs* and *databitpos* (for the case of multiple input gates) attributes, the *kron()* function is called by the *CreateTransform()* function to create the final matrix representation of the quantum gate. The *kron()* function in the *matrix* class is an efficient implementation of the Kronecker product [29] and can quickly generate large matrices. While this approach introduces a pre-processing step before the actual emulation of the quantum circuit (unlike other quantum circuit simulators), once the gates descriptions have been generated, they can be reused multiple time and for different quantum algorithms.

The command-line interface makes this technique more accessible. The user simply has to specify the gate code (an acronym based on the gate name that can be referenced easily through the help menu in the interface), the size of the quantum system and the position of the qubit (the top most position is set to 0). Optionally, for multiple input gates, the user also has to specify the number of inputs of the gates, and the position of the data qubit as well). All of these parameters are input as a single command (space delimited) and the generator outputs the VHDL description of the required gate. The output file is a complete description and can be compiled and synthesized using FPGA CAD tools.

4.2 Miscellaneous Architecture Components

The final step in the FPGA emulation of a quantum circuit is to retrieve the final quantum state from the FPGA to the PC. The final quantum state of the qubits can then be subjected to quantum measurements in software (see Appendix A for more details on quantum measurement simulation). Depending on the size of the quantum system and the expanded state space, the data that has to be transferred from the FPGA to the PC can be significantly large. Various PC communication protocols such as USB, parallel port and serial port (RS-232) can be used to send data to the PC. While the USB communication is the fastest way to transfer data, due to the unavailability of USB communication on the FPGA development platform that was available, it was decided to use the RS-232 port instead.

The development of serial communication for the emulator serves as a guideline for other communication techniques that may be use instead of serial RS-232 communication.

The first step to achieve serial communication between the PC and the FPGA was to implement a universal asynchronous receiver/transmitter (UART) controller [30]. The UART controller takes in a byte of data and sends it one bit at a time to the PC. Due to the varying size of the data set and data primitives, the UART is designed to work with various sizes of total input data. The UART then takes 4 bits of data, converts them to an ASCII character and sends it serially to the PC. On the PC side the data is stored on a file and can then be converted to real numbers.

Serial communication can occur at a variety of speeds [30] and for this a customized clock divider circuit using behavioral VHDL has also been developed. The UART clock is much slower than the global system clock, and sending the data to the PC is much slower (except perhaps for very large quantum circuits) than the actual computation of the final quantum state. Since there are a variety of PC communication techniques available, the emulator results presented later in this chapter do not include time required to send the data to the PC. The time for serial communication however, can be determined unequivocally. Based on the total length of the final state vector M, the number of bits per entry of the state vector $2N$ where N is the number of bits assigned for the real or imaginary part of the complex number and the baud rate for the serial communication (bytes per second), the time for sending the data to the PC t_{comm} can be determined from the following equation

$$t_{comm} = \frac{M \times 2N \times 10}{4 \times baudrate} = totalcharacters \times \frac{10}{baudrate} \tag{4.1}$$

For example a 16 qubit circuit, using $N = 18$ (16 bits for the mantissa of the fixed point number), it would take 51.2 seconds to transmit the data from the FPGA to the PC. This number, obviously grows exponentially when simulating a larger number of qubits and therefore, a faster communication protocol (such as USB 2.0 with data rates up to 480 Mbps [31]) would be more practical in that case.

The entire emulation scheme is shown in Figure 4.2. The quantum circuit controller, essentially determines the start and end of a quantum system evolution. For circuits such as the Grover's search algorithm, where the data iterates through the circuit, the controller has to wait for the exact number of clock cycles (of the global clock) before notifying the UART controller to start sending the output of the circuit to the PC. The total clock cycles

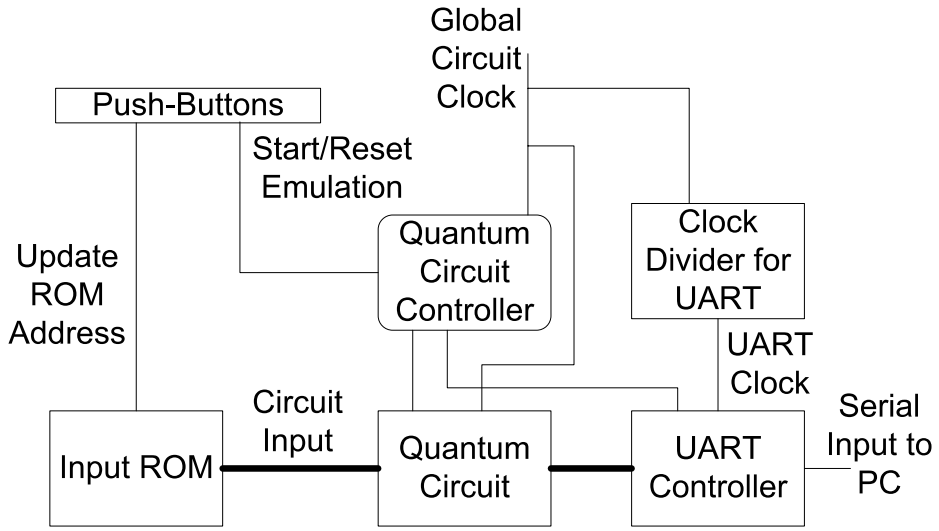

Fig. 4.2 Block diagram of final emulation hardware

are determined by the number of pipeline stages in the circuit (and the iterations of the data in the case of Grover's search algorithm).

A number of push buttons on the FPGA development board were also used. The buttons can be used to start and reset the emulation as well as choose the inputs to the quantum circuit. A set of inputs is loaded into memory (ROM) on the FPGA during synthesis. The push-button updates the address line on the ROM to select which input is sent to the circuit. This mechanism is convenient and necessary since entering the initial values for a large number of qubits is cumbersome.

4.3 Emulator Mapping Results

In this section, details about the synthesis of the various components of emulator are provided along with experimental evidence of the various synthesis techniques discussed in Chapter 3. The techniques presented in this thesis offer the means of undertaking quantum circuit emulations in FPGAs by including the quantum gate library and expander circuits, like most quantum software simulators [10]. No changes to standard FPGA mapping and the overall design flow are required. Figure 4.3 show the FPGA mapping of the 3-qubit QFT circuit. The parallel and pipelined architecture of the circuit is clearly evident.

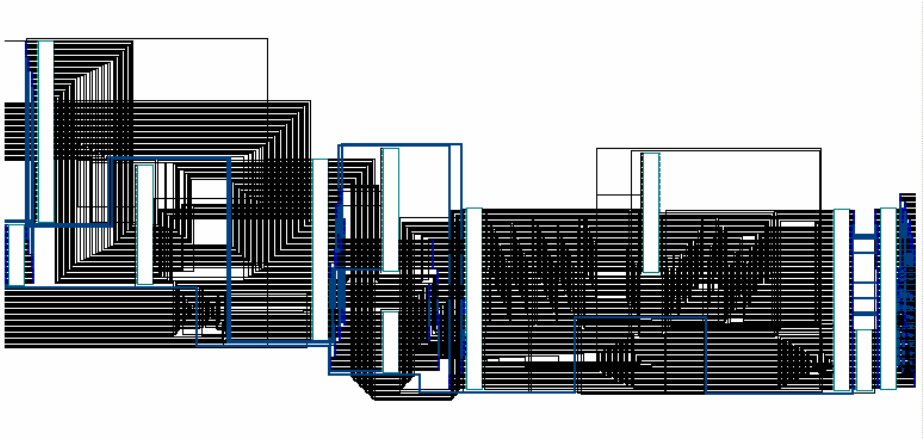

Fig. 4.3 FPGA snapshot of the 3-qubit QFT circuit

4.3.1 Synthesizing Quantum Circuits

The quantum circuit emulator uses a traditional four stage VHDL to FPGA flow illustrated in Figure 4.4. However, the synthesis and optimization stage (stage 2) requires the a considerable amount of processing time as the synthesizer optimizes the quantum circuit's mapping based on area and speed parameters. In order to improve on the time spent during this stage, the optimization (FPGA level block placement and functionality) for individual quantum gates is saved and can be reused for different quantum circuits. This effectively bypasses the stage 2 of the flow and only the routing stage is necessary for circuits using pre-synthesized gates.

4.3.2 Quantum Gate Synthesis Results

Quantum gates, unlike classical gates, have a uniquely different transform matrix depending on the ordering of the inputs. This is especially evident when single input quantum gates operate on an input in the expanded state space. The gate transform depends on the size of the inputs and the position of gate's input in the expanded set. However, as described in detail in Chapter 3 these gate transforms have similar matrix topologies. This similarity in the transforms translates to the fact that resources consumed by a quantum gate depend on the size of the expanded state vector. Furthermore, the resource consumption is oblivious to changes in the position of the actual qubits the gate is operating on. The creation of different quantum gates for different mantissa lengths is conveniently handled via the soft-

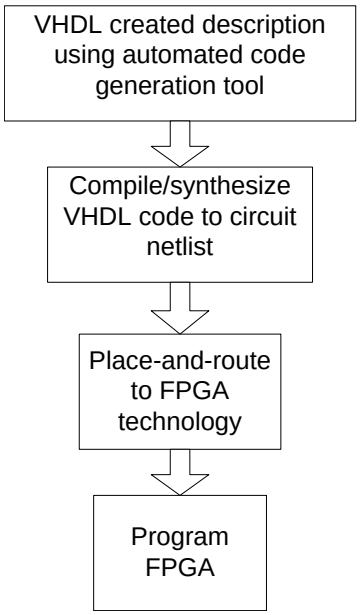

Fig. 4.4 Quantum Circuit Emulator Synthesis Flow

ware scripts. The software scripts automatically generate the correct VHDL code depicting the transform in an efficient number of complex multipliers that can then be mapped to the FPGA directly. Table 4.1 depicts the logic cell usage for the quantum gates in the library for the mantissa size of 12-bits. The device chosen is the Altera Stratix EP1S80F1020C. The simulation tool used is ModelSim and Leonardo Spectrum was employed to obtain synthesis results.

Table 4.1 Logic Cell Usage on Altera Stratix EP1S80F1020C

Circuit/Gate	Single Input	3-qubit Expanded Input	4-qubit Expanded Input	5-qubit Expanded Input
Hadamard Gate	643	1588	3069	4227
Rotation Gate	442	442	1588	2011
CNOT Gate	0	0	0	0
X-Gate	0	0	0	0
Z-Gate	0	0	0	0

As can be observed, the CNOT gates do not occupy any resources themselves as they are implemented as simple bit vector swap operations on the FPGA. However, they do incur a

significant resource cost in the computation because the gates only operate on inputs in the expanded state space form. Therefore, resources are consumed by expander circuits and all the gates subsequent to the CNOT gate in the circuit also have to be expanded to deal with the expanded input. As X-gates and Z-gates also do no involve any adder/multiplier units they do not consume any resources. On the other hand, Hadamard gates and rotation gates do require more resources as when the input size scales up more adder/multiplier units are consumed.

4.3.3 Expander Circuit Synthesis Results

Expander circuits are necessary to convert a set of individual qubit vectors into an expanded state space vector. Two state space expansion circuitry designs were considered. The direct expansion technique basically expands a set of qubits within one pipeline stage using a ripple-multiplier architecture. The major disadvantage of this technique is that it results in a logic-cone that is difficult to synthesize beyond a small number of qubits. The synthesizer software runs out of memory when synthesizing the direct expansion circuit for more than nine qubits (running on a high performance PC with 2.0 GHz AMD Opteron processors and 2GB of RAM). Furthermore, from observing various quantum circuit topologies we realize that direct expansion is only required at most for a set for two or three qubit since most quantum circuits comprise of two/three input quantum gates. Figure 4.5 depicts the direct expansion circuitry synthesis results for varying numbers of qubits. The mantissa length in all cases is 12 bits. Note that the resource usage grow linearly until six qubits, beyond which the synthesizer starts having problems with the large logic cone and the resource usage grows exponentially from then on.

A complimentary state space expansion circuit is then needed to merge smaller state spaces when necessary. This leads to a gradual state space expansion technique whereby the computation state space is gradually expanded and only when necessary. This is unlike the general software simulation approach where all the qubits are represented in the expanded state space at the beginning of the computation. The gradual state space expansion technique greatly improves performance. Table 4.2 illustrate the difference between these two design philosophies by using a 3-qubit QFT circuit as a test-case. As can be observed with the gradual state space expansion technique leads to an 18.5% improvement in resource usage and 11.3% improvement in computation time.

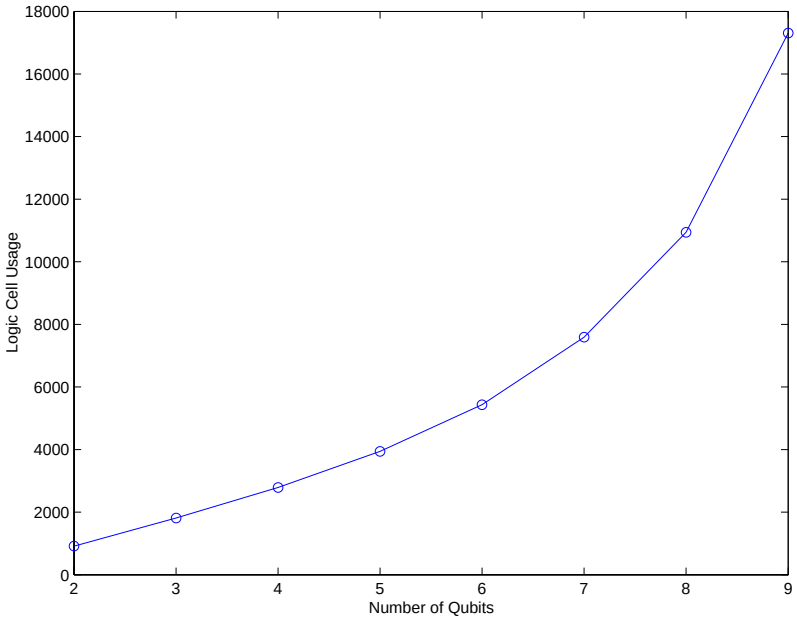

Fig. 4.5 Direct state space expansion circuit synthesis

Table 4.2 Comparison of the direct and gradual expansion techniques

	Logic Cell Usage	Computation Time [ns]
Direct Expansion	13829	110.8
Gradual Expansion	11271	97.65

4.4 Computation Error

In Chapter 3 a detailed error analysis has been provided that relates classical computation error when simulating quantum circuits with the error suffered by real quantum circuits in the form of quantum noise. It has been determined in [26],[27] that if the error introduced per gate is in the range 10^{-5} and 10^{-3} the computation can take place successfully. Using the limits on gate error and classical error analysis techniques, a bound on the size of the classical word length needed to mimic real-life quantum computation successfully has been derived. In this section, the actual results of the application of the error bounds is produced. This provides an experimental validation to the theoretical results presented in Chapter 3.

The error introduced per gate can be experimentally determined by comparing the computed output of a synthesized gate and using the fidelity expression in Equation 3.4.

Table 4.3 presents the error introduced by each quantum gate for various mantissa lengths with the error bound found expressed in Equation 3.7 (using $N = 3$).

Table 4.3 Error introduced by different gates for various mantissa lengths

Gate Name	11 bits	12 bits	17 bits	18 bits
Hadamard gate	6.48×10^{-4}	1.57×10^{-4}	1.68×10^{-5}	0.90×10^{-5}
Controlled Rotation gate	6.53×10^{-4}	1.63×10^{-4}	1.71×10^{-4}	0.95×10^{-5}
CNOT gate	0	0	0	0
Z-gate	0	0	0	0

As can be observed that the actual error per gate for the various gates is well within the tolerance for successful quantum computation. The CNOT gates themselves do not incur error directly (the bit vector swap operation does not incur any precision error), but the error is implicitly present in the form of the expansion operation itself. At the system level we see that the emulated output has high fidelity. For instance in the case of a 3-qubit QFT circuit using a 12-bit mantissa the resulting output has an absolute error of 2.06×10^{-4} which is comparable to the actual gate error of the Hadamard and controlled rotation gates that form the circuit.

The above results also vindicate the choice of fixed-point numbers for depicting quantum information within the emulator. The main concern with this choice was that the error incurred in computation due to the discretization of the gate coefficients and qubit values would make emulation impractical with fixed point numbers. From the above results, it is evident that this is not the case.

4.5 Quantum Circuit Benchmarks

In this section a comparison of the emulator's performance with the eminent software simulator QuIDD [10] is provided. The QFT circuit and Grover's search algorithm are used as test-cases for performance comparison. Apart from the fact that these circuits are two of the most important one's that have been developed so far, the topologies of the circuits require more resources than most of the other algorithms developed so far and therefore they also serve as good stress tests. Table 4.4 compares the emulator with QuIDD when simulating the QFT circuit. Table 4.5 provides another benchmark using the Grover's search algorithm. The timing results of the emulator were obtained from the

place-and-route synthesis process in Leonardo Spectrum. QuiDD-based simulations were executed on a 2 GHz 64-bit Opteron unit with 2 GB of RAM and using the Redhat Linux operating system. The timing results of the algorithms were clocked using native QuIDD profiling functions.

Table 4.4 QFT Benchmark

Number of Qubits	LC Usage	FPGA Emulator [ns]	QuIDD[ns]
3	11271	97.65	2.13×10^7
4	16687	127.80	6.06×10^7
5	21898	147.80	1.2×10^8

Table 4.5 Grover's Search Algorithm Benchmark

Number of Qubits	LC Usage	FPGA Emulator [ns]	QuIDD[ns]
3	14284	97.65	3.40×10^7
4	23525	255.10	6.00×10^7
5	30121	286.80	1.59×10^8

As can be observed the FPGA emulator outperforms QuIDD by several orders of magnitude. It is important that the computation times mentioned here are ones that involve just the actual computation of the output of the circuit and do not take into account time for quantum measurement. The quantum measurement time would be more significant for the emulator as data has to be transferred from the FPGA back to the PC. That time is simply a function of the data transfer protocol being used.

Another interesting observation is the way computation time is scaling up with circuit size. The increase in the case of the QFT is essentially due to the fact that additional pipeline stages have been introduced for larger circuits. In the case of the Grover's search algorithm, the increase of a 3-qubit circuit to a 4-qubit one is greater not only because of additional pipeline stages but also because the 4-qubit circuit requires two iterations to successfully complete the search as opposed to one iteration in the 3-qubit case. The difference between the 4-qubit and 5-qubit circuits is less significant since the number of iterations required are the same for both circuits. Conversely, QuIDD's computation time also scales up even more significantly with increases in circuit size. Therefore, as long as a circuit can be synthesized to the FPGA it should outperform QuIDD because of the emulator's parallel architecture and hardware-level arithmetic computation.

4.6 Scaling Quantum Circuit Emulation

The results presented in the previous section are a proof of concept that FPGA-based emulation of quantum circuits is viable and has a significantly lower computation runtime when compared to the leading software simulator. Ultimately though, the emulation environment's purpose is to emulate large scale quantum circuits and in this section a qualitative analysis at the different aspects of the emulator is undertaken to determine how well the overall architecture scales up.

Using a top-down approach, consider the overall architecture of the quantum circuit. Essentially, it is a pipelined architecture and as quantum circuits increase in size, more pipeline stages are added to the circuit. The only critical issue with a pipelined architecture is timing and with the robust clock distribution system of today's leading FPGA technologies, this issue should not be any hinderance to constructing larger quantum circuits. Furthermore, while the pipelined architecture enforces stringent timing requirements, it also decouples the computation into constituent parts. This fact is critical because larger quantum circuits can be spread over multiple FPGAs by splitting up the pipeline stages over different FPGA units. A global clock synchronizing the operations on all the FPGAs can then be used to retain the cohesive nature of the overall pipeline.

At the next level consider the components that make up the quantum circuit, that is quantum gates and expander circuits. Using the gradual state space expansion technique, the state space expansion operation should not be a constriction on the construction of larger quantum circuits. Since the operation is divided over multiple pipeline stages (at the discretion of the designer) it is architecturally quite benign. As far as quantum gates are concerned, the most commonly used one is the CNOT gate and that itself does not consume any resources on the FPGA. On the other extreme, Hadamard gates can consume a significant amount of resources when operating on large state vectors, however for most quantum circuits the Hadamard gates are applied at the beginning of the computation to individual qubits and therefore they no longer consume an exorbitant amount of resources. However, as illustrated by the expander circuit architectures, a hardware reuse design approach is possible within the parallel/pipelined architecture of the emulator and it can be applied to Hadamard gates as well.

At the lowest level, the emulator comprises of fixed-point adders and multipliers. Larger quantum circuits require more adders/multipliers. FPGA technology is scaling up rapidly

and potentially custom platforms comprising of multiple FPGAs can be constructed to provide the necessary resources for larger quantum circuits. The word length expansion techniques proposed in Chapter 3 can be used to optimize the resource consumption of the adders/multipliers as the state space expands within a large quantum circuit.

Finally, it is important to note that the design of the emulator allows flexibility in the way quantum circuits can be constructed. From quickly generating various architecture descriptions through software to flexible state space expansion techniques to having the ability to vary the word length of data primitives, the emulator has been designed such that it can adapt to the architectural challenges of large scale quantum circuits.

Chapter 5

Conclusion and Future Work

This thesis focuses on the design, implementation and evaluation of a FPGA-based quantum circuit emulator. The emulator uses a pipelined architecture, to emulate the parallelism in quantum computation as well as the time evolution of a quantum system. Optimizations based on computer architecture, sparse-matrix computation and the natural properties of quantum circuits were employed to produce a scalable platform for quantum computation that outperforms software-based simulators by several orders of magnitude. Furthermore, the emulator also takes into account the effect of quantum noise and gate error on the computation, both of which are difficult to reproduce in software. At the same time, the implementation of the emulator provides a deeper understanding of the issues involved in quantum circuit modeling such as word-length of the data-primitives and gradual state space expansion based on quantum circuit topology.

5.1 Thesis Summary

In Chapter 1 a brief history of quantum computing is provided. Quantum computing is one of the most promising new forms of computing that is being heralded as the future of computation. Interest in this form of computing is multidisciplinary and research in the notion of using the spin/polarization of particles exhibiting quantum mechanical properties to store information has been ongoing for almost two decades now. However, the unavailability of quantum computers has led to the development of software based quantum circuit simulators. Simulating quantum circuits using classical computers on the other hand, is computationally expensive as quantum algorithms require exponential resources on classi-

cal computers as opposed to polynomial resources on real quantum computers. Emulation of quantum circuits is next described as a more detailed modeling approach compared to simulation that can be accelerated by performing computation in parallel at the hardware level.

Details of the quantum circuit model are described in Chapter 2. The analogous nature of quantum circuits (in terms of bits and gates) with classical circuits is used to provide a convenient understanding of this new computational model. Quantum mechanical principles of superposition and entanglement as used in quantum computing are also described. The Dirac bra-ket notation as well as the expanded state space notation used to depict information stored by a quantum system are then introduced. Two famous quantum algorithms: the quantum Fourier transform (QFT) and Grover's search algorithm are introduced and they serve as examples and benchmarks in the rest of the thesis. Finally, a brief survey of three software simulators of quantum circuits and the various techniques they employ to optimize quantum circuit simulation is undertaken.

Chapter 3 describes in detail the architecture of the emulator and the various optimizations used to efficiently synthesize quantum circuits on the FPGA. The concept of gradual state space expansion is introduced whereby the state space is expanded gradually when necessary as opposed to software simulators that execute quantum algorithms on a fully expanded state space. The novelty of this technique is that it significantly reduces the amount of computation involved and details of two circuits that perform state space expansion are provided.

Next, sparse-matrix and hardware-reuse based optimization techniques are described that allow efficient implementation of different quantum gates on the FPGA. This is followed by a detailed error analysis of the computation being performed based on the fixed-point data representation scheme used by the emulator. This classical error analysis is then combined with quantum gate error and quantum noise analysis to produce a bound on the word-length of data primitives. As the emulator architecture allows flexibility in word-length, a word-length expansion paradigm is proposed that allows modeling the effect of quantum noise and gate error on quantum computation.

Lastly, in Chapter 4 actual synthesis results of various quantum circuits are provided along with experimental results that vindicate the choice of the optimizations and theoretical analysis laid out in Chapter 3. The chapter begins with the description of a software gate generation tool that outputs VHDL descriptions of various quantum gates based on

the parameters provided by the user. The performance of the emulator is compared with an eminent software simulator and it is shown that the emulator outshines the software simulator in terms of computation runtime by several orders of magnitude. It is empirically proven that the FPGA-based emulator is a viable and effective platform for quantum circuit modeling.

5.2 Future Research Work

The FPGA-based emulator as it stands now is a complete test-bed for quantum circuit emulation. However, more avenues of enhancing the emulator's performance still remain open. An important enhancement to the emulator would be more CAD level support for synthesizing quantum circuits. Currently, regular CAD tools are used to synthesize quantum circuits. However, research and development of CAD tools specifically designed to synthesize massively parallel architectures, fixed-point multipliers/adders and swap operations could lead to significantly faster quantum circuits that consume less resources on the FPGA. Another possibility is synthesis of quantum circuits over multiple FPGA's and the development of a multiple FPGA test-bed to synthesize large-scale quantum circuits.

On the other end of the spectrum, the effect of quantum error-correction algorithms within the emulation environment can be investigated. The FPGA emulator can also be potentially modified to be used in practical applications of quantum computing such as quantum cryptography.

Appendix A

Quantum Measurement Simulation using Quantum Frames

Quantum measurement is a complex process that has serious implications on quantum algorithm performance. Therefore, in developing a simulation environment for quantum computers it is important to take in account the effect of measurements on the system. To this end, a software based quantum measurement simulator has been developed that applies the frame-based measurement technique to the emulated output of quantum circuits.[1]

A.1 Quantum Measurement

Quantum evolution follows from the second postulate of quantum mechanics which states that an evolution of a closed quantum system over time is described by a unitary operator. Thus a state $|\psi'\rangle$ at time t_2 is related to an earlier state $|\psi\rangle$ by a unitary matrix U.

$$|\psi'\rangle = U|\psi\rangle \tag{A.1}$$

The postulate assumes that the evolution of closed quantum system is being considered. Quantum circuits are considered to be closed systems in which quantum gates apply unitary transformations to the quantum state as it progresses through time. However, at the end of a quantum algorithm the system is no longer closed as it is subjected to a measurement so that its state can be determined (the result of the computation). At this point the system

[1] Part of the work presented here has already been published in [32]

is no longer closed and the evolution is no longer guaranteed to be unitary.

At this point a third postulate of quantum mechanics is introduced which defines quantum measurements in a formal manner. The postulate states that quantum measurements are a collection of measurement operators M_m where m is the measurement outcome that may occur in our system. The probability that result m occurs upon measuring a quantum system $|\psi\rangle$ is given by the equation

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle \tag{A.2}$$

It can be observed from the above postulate that measuring quantum systems is a probabilistic process. The measurement operators firstly must meet the *completeness equation*:

$$\sum_m M_m^\dagger M_m = I \tag{A.3}$$

For example, if a single qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is measured the probability of measuring 0 and 1 is $|\alpha|^2$ and $|\beta|^2$ respectively. In order to measure these states the measurement operators are constructed as $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle|\langle 1|$. It is easily checked that these operators satisfy the completeness equation. Thus, when these operators are applied on the system, the resulting state depends on the probability defined by the α and β coefficients.

$$M_0|\psi\rangle = |0\rangle, M_1|\psi\rangle = |1\rangle$$

Suppose a system having $|\psi_i\rangle$ $(1 \leq i \leq n)$ *orthonormal* states is prepared in the ith state. If a measurement operator M_i defined such that $M_i = |\psi_i\rangle\langle\psi_i|$ and is applied to the prepared states then $p(i) = \langle\psi_i|M_i|\psi_i\rangle = 1$. Thus, this state can be measured with certainty.

However, the restriction imposed by quantum measurements is that all the states $|\psi_i\rangle|$ must be orthogonal in order to distinguish them successfully. The probability of error in distinguishing non-orthogonal states is not zero. This causes significant measurement errors when results of the quantum computation are in non-orthogonal quantum states (as in the case of the QFT algorithm).

A.1.1 Projective Measurements

Projective measurements are described by an observable M, which is an observable Hermitan operator on the state space of the system being observed. The spectral decomposition of M is given by

$$M = \sum_m m P_m \tag{A.4}$$

where P_m is a projector on to the eigenspace of M with eigenvalue m. The probability of measuring outcome m is now

$$p(m) = \langle \psi | P_m | \psi \rangle \tag{A.5}$$

Thus, projective measurements are a special case of postulate 3 where the corresponding measurement vector M_m is forced to be Hermitan and $M_m M_{m'} = \delta_m \delta_{m'} M_m$.

A.1.2 POVM: Positive Operator-Valued Measure

POVM are the complete set E_m where E_m is a positive operator such that

$$E_m = M_m^\dagger M_m \tag{A.6}$$

Algebraically POVM completely satisfy postulate 3. The key aspects of POVM is that all their elements are positive and that $\sum_m E_m = I$ which is the completeness equation. POVM are a powerful formalization of the quantum measurement operator. They are less restrictive than the general measurement vectors and their construction is more intuitive.

A.2 Frame-Based Measurements

In this section a brief discussion on the problem of distinguishing non-orthogonal states and a possible solution based on tight frames is given.

A.2.1 Distinguishing non-orthogonal quantum states

As described in the previous section, if a quantum system has non-orthogonal states then distinguishing between these states has a probabilistic error. For a quantum system con-

taining ψ_i quantum states ($1 \leq i \leq n$) and POVM elements are ϕ_i, then the probabilistic error can be described as

$$P_e = 1 - \frac{1}{n} \sum_{i=1}^{n} |\langle \mu_i, \psi_i \rangle|^2 \tag{A.7}$$

If all the states ψ_i are orthonormal then choosing $\mu_i = \psi_i$ results in $P_e = 0$. However, P_e will be non-zero for non-orthogonal pure quantum states. Thus, a fundamental problem in quantum mechanics is to construct measurements optimized to distinguish between a set of non-orthogonal pure quantum states.

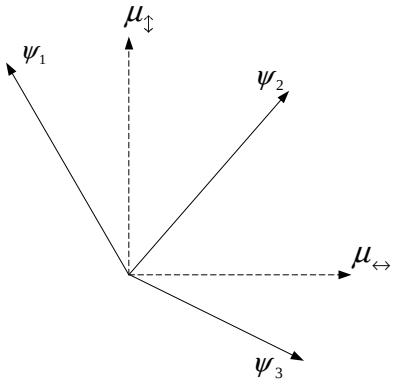

Fig. A.1 Projective Measurements

Figure 2 depicts one the major problem in projective measurement. For simplicity the complex dimension (which leads to a three dimensional figure like that of a Bloch Sphere) is ignored. $\mu_{\leftrightarrow}$ and $\mu_{\updownarrow}$ are the measurement vectors while ψ_1, ψ_2 and ψ_3 are the states that are to be measured. Due to the orthogonality restriction, the quantum states can only be projected on to the orthogonal measurement vectors. A measurement for instance could project ψ_1 or ψ_2 to $\mu_{\updownarrow}$ and it would not be possible to distinguish between the two quantum states.

Figure A.2 shows how POVM measurement vectors can be devised so they allow for less probability error than standard measurements. POVM vectors are not restricted to be orthogonal. Here, the quantum states ψ_1, ψ_2 and ψ_3 can be projected on to the non-orthogonal measurement vectors μ_1, μ_2 and μ_3. The choice of measurement vectors determines the probability error of detection. One possible solution to determining the measurement vectors is to formulate this problem as a quantum detection problem in which, measurement

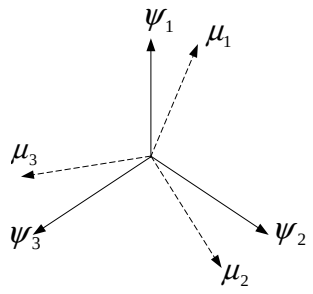

Fig. A.2 POVM Measurements

vectors are constructed to minimize the probability of detection error or more generally the Bayes' cost (the amount of times an algorithm will be repeated in order to reduce the detection error to zero). Attempts to achieve a solution based on direct optimizations on these conditions is difficult and as yet an unsolved problem.

A.2.2 Least Squares Measurements and Tight Frames

An alternative optimization criterion has been proposed known as the squared error criterion. Measurement vectors are chosen to minimize the sum of the squared norms of the error vectors, where the ith error vector is defined as the difference between the ith state vector and the ith measurement vector. This optimized measurement is knows as the *least-squares measurement* (LSM).

Tight frames are generalized bases comprising of μ_i vectors $1 \leq i \leq n$ in the Hilbert space such that

$$\sum_{i=1}^{n} |\langle x, \mu_i \rangle|^2 = \beta^2 ||x||^2 \tag{A.8}$$

and $\beta > 0$.

Tight frames can contain redundant elements (like POVM) which can be modeled to optimize for LSM. Thus, it is desirable to construct tight frames which have vectors μ_i designed to minimize the error

$$E = \sum_{i=1}^{n} \langle \varrho_i, \varrho_i \rangle \tag{A.9}$$

where $\varrho_i = \psi_i - \mu_i$.

The quantum frame being constructed is the *constrained least-squares frame* where β and the frame vectors are chosen to satisfy the squared error criterion. The construction of the frame begins with a set of quantum states grouped in a matrix Ψ. The first step is to compute the value of β used to constrain the frame. This is obtained as follows

$$\beta = \frac{1}{r} Tr((\Psi^* \Psi)^{1/2}) \tag{A.10}$$

where r is the rank of the state matrix and Tr is trace of the resulting matrix. Next the frame itself can be expressed in terms of the state matrix and β.

$$F = \beta \Psi((\Psi \Psi^*)^{1/2})^{\dagger} \Psi \tag{A.11}$$

where the $()^{\dagger}$ operation is Moore-Penrose psuedo-inverse [29] and x^* is the conjugate transpose of x.

The squared error is then given by

$$E = \sum_{i=1}^{r} (\beta - \sigma_i)^2 \tag{A.12}$$

where σ_i are the non-zero positive eigen-values of the state matrix, obtained using a SVD operation on Ψ.

Thus, given a matrix whose columns represent possible states of a quantum system a frame matrix can be constructed whose columns represent measurement vectors constructed to reduce the squared error given above.

A.3 Simulation of Measurement

The two major differences between POVM and standard quantum measurements are that the vector μ_i of the frame (considering its duality with POVM as described in [33]) do not have to be normalized or orthogonal. A POVM is a set of operators. These operators can be constructed from the columns of the frame by performing an outer product on the columns. Thus, operator A_i constructed from the ith column of the frame can be constructed as

$$A_i = \mu_i \mu_i^* \tag{A.13}$$

The probability of observing the ith outcome for a given state ψ is given by

$$p(i) = \langle\psi|A_i|\psi\rangle = |\langle\mu_i|\psi\rangle|^2 = |\mu_i^*\psi|^2 \tag{A.14}$$

The sum of these probabilities for all the observables should be equal to one. One check to see if the generated frame is correct is that

$$FF^* = \beta^2 P_U \tag{A.15}$$

where P_U is the projector on to the quantum space.

One of the motivations for working with frame-based measurement techniques is that it is a closed-form optimization of the state detection problem and hence can be simulated along with quantum algorithms on a classical computing system. The closed form nature of this optimization is an important advantage because constructing optimal standard measurement vectors is an NP-complete problem.

Quantum measurements are a key part of a quantum algorithm. In classical algorithms obtaining the data from the algorithm is normally a trivial exercise and has no bearing on the usefulness and performance of the algorithm. However, in quantum computing the measurement is the key to algorithm performance and usability. For example, in the Grover's search algorithm, the performance of the algorithm is computed to be $O(\sqrt{n})$ where n is the size of the database being searched, only because at least $\sqrt{n}$ measurements have to be performed to obtain the result of the search with high probability.

A.4 Case Study

In this section an example is provided to illustrate the working of frame-based quantum measurement. To illustrate the simplicity of the process, the technique is applied it to a q-ary logic system. The example provided in this section subjects a quantum ternary bit to frame-based measurement. For this purpose, Equations 2.2 and 2.3 can be extended to ternary logic:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle \tag{A.16}$$

where α, β and γ are complex coefficients related by the equation.

$$|\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1 \tag{A.17}$$

A quantum system based on a single quantum-ternary bit is depicted by the following state matrix Ψ

$$\Psi = \begin{pmatrix} 1/\sqrt{3} & 1/\sqrt{3} & 0 \\ 1/\sqrt{3} & 1/\sqrt{3}i & \sqrt{\frac{2}{3}} \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{pmatrix}$$

.

This matrix is now subjected to the frame measurement procedure. Using the state matrix the resulting frame is

$$F = \begin{pmatrix} 0.67 + 0.01i & 0.32 + 0.011i & -0.39 - 0.06i \\ 0.40 - 0.21i & -0.27 + 0.45i & 0.48 - 0.04 \\ 0.09 + 0.22i & 0.50 - 0.28i & 0.56 + 0.06 \end{pmatrix}$$

.

The columns of the frame matrix F are the measurement vectors optimally close to the quantum states in a least squares sense. The squared error calculated is 0.877. The sum of all the probabilities adds up to one using the following expression

$$\sum_{i=1}^{3} |\langle \mu_i | \psi \rangle|^2 = 1$$

where ψ can be any one of the quantum states and μ_i are the frame vectors. Table A.1 details the probabilities of measurement for each quantum state using each of the measurement operators.

Table A.1 Measurement Probabilities for Ψ

	μ_1	μ_2	μ_3
ψ_1	0.634	0.163	0.203
ψ_2	0.163	0.766	0.071
ψ_3	0.203	0.071	0.726

As can be observed from Table A.1, frame-based measurements do not allow error-free state detection. For instance, if the system is in state ψ_1 using measurement vector μ_1

gives approximately 63% probability that ψ_1 will be correctly identified while the other two states are have a much lower probability of detection by μ_1. A similar situation exists when the system is in states ψ_2 or ψ_3 and measurement vectors μ_2 or μ_3 respectively are used to perform the measurement. Thus, frame-based measurement vectors would not allow the deterministic detection of quantum states in just one measurement as there are non-zero probabilities of detection for every state when subjected to each of the measurement vectors. Therefore, a number of measurements have to be performed [34], before a high enough probability of detection can be be observed.

An extension to the square-error optimization has been developed in [35]. This extension is known as the weighed-squared error. Instead of optimizing for the squared error for all the quantum states, certain states that have been prepared with a higher probability of detection could be given more weight in the optimizing procedure. It has also been shown that for systems which exhibit strong symmetry, the measurement vectors generated are optimal and have very low error. To illustrate this, a set of symmetric states is chosen (similar to Fig. 3).

$$\Psi_2 = \begin{pmatrix} -\frac{1}{\sqrt{3}} & 1 & -\frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} & 0 & -\frac{1}{\sqrt{3}} \end{pmatrix}$$

The frame generated in this case is

$$F = \begin{pmatrix} -0.547 & 0.947 & -0.547 \\ -0.611 & 0 & 0.611 \\ 0.611 & 0 & -0.611 \end{pmatrix}$$

and the squared error is 0.009 which is far less than in the previous case. Table A.2 depicts the probabilities of measurement in the new situation. The effect of symmetry can be observed from the probability distribution of the measurements. In this situation measurement vectors μ_1 and μ_3 are optimal to detect states ψ_1 and ψ_3 respectively. Measurement vector μ_2 is optimal for observing the basis state ψ_2 and by symmetry has an equal probability of observing either ψ_1 or ψ_3.

Table A.2 Measurement Probabilities for Ψ_2

	μ_1	μ_2	μ_3
ψ_1	0.7	0.2	0.1
ψ_2	0.2	0.6	0.2
ψ_3	0.1	0.2	0.7

References

[1] R. Feynman, "Simulating Physics with Computers," *International Journal of Thoertical Physics*, p. 467, 1982.

[2] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," *Proceedings of the Royal Society of London Series A*, p. 97, 1985.

[3] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," *Symposium on Fundamentals of Computer Science*, pp. 124–134, 1994.

[4] "A Quantum Information Science and Technology Roadmap," *Report of the Quantum Information Science and Technology Experts Panel*, 2005.

[5] R. Hughes and et al., "The Los Alamos Trapped Ion Quantum Computer Experiment," *Fortschritte der Physik*, pp. 329–362, 1997.

[6] D. Cory, R. Laflamme, and et al., "NMR Based Quantum Information Processing: Achievements and Prospects," *Fortschritte der Physik Special Issue*, p. 298, 2000.

[7] A. Imamoglu, D. Awschalom, and et al., "Quantum information processing using quantum dot spins and cavity QED," *Physics Review Letters*, p. 4204, 1999.

[8] M. Bocko, A. M. Herr, and M. Feldman, "Prospects for quantum coherent computation using superconducting electronics," *IEEE Transactions on Applied Superconductivity*, pp. 3638–3641, 1997.

[9] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[10] G. Viamontes, I. L. Markov, and J. Hayes, "Improving Gate-Level Simulation of Quantum Circuits," *Quantum Information Processing*, vol. 2, no. 5, pp. 347–380, 2003.

[11] B. Oemer, "Classical Concepts in Quantum Programming," *quant-ph/021110*, 2003.

[12] I. G. Karafyllidis, "Quantum Computer Simulator based on the Circuit Model of Quantum Computation," *IEEE Transaction on Circuits and Systems I*, 2005.

[13] F. Phillips, "Quantum Computation," *The Mathematica Journal*, 2001.

[14] J. Gruska, *Quantum Computing*. Osborne McGraw-Hill, 1999.

[15] P. Shor *SIAM Journal of Computing*, p. 1484, 1997.

[16] L. Grover *Physics. Rev. Letter*, pp. 325–328, 1997.

[17] G. Viamontes, I. L. Markov, and J. Hayes, "High Performance Simulation of Quantum Computation using QuIDDs," *Proceedings of Quantum Communication, Measurement and Computation (QCMC)*, pp. 311–314, 2002.

[18] P. J. Ashenden, *The Designer's Guide to VHDL, 2nd Edition*. Sams; 1st ed, 1987.

[19] M. Uderescu, L. Prodan, and M. Vladutiu, "Using HDLs for describing quantum circuits: a framework for efficient quantum algorithm simulation," in *Conference on Computing Frontiers*, pp. 96–110, 2004.

[20] A. U. Khalid, Z. Zilic, and K. Radecka, "FPGA Emulation of Quantum Circuits," in *IEEE International Conference on Computer Design*, pp. 310–315, 2004.

[21] J. L. Hennessy, D. A. Patterson, and D. Goldberg, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2002.

[22] K. M. Obenland and A. Despain, "Simulating the Effect of Decoherence and Inaccuracies on a Quantum Computer," *Proceedings of the 1st NASA Conference on Quantum Computation and Quantum Communication*, pp. 447–459, 1998.

[23] M. A. Nielsen, "The entanglement fidelity and quantum error correction," *quant-ph/9606012*, 2004.

[24] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *quant-ph/9511018*, 1995.

[25] J. J. Vartianinen, M. Mottonen, and M. Salomaa, "Efficient decomposition of quantum gates," *quant-ph/0312218*, Apr 2004.

[26] J. Preskill, "Reliable Quantum Computers," *Proceedings of the Royal Society London Series A*, pp. 385–410, 1998.

[27] E.Knill, R. Laflamme, and W. Zurek, "Threshold Accuracy for Quantum Computation," *quant-ph/9610011*, 1996.

[28] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering - Conquering Complex and Changing Systems*. Prentice Hall, 2000.

[29] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Johns Hopkins Press, 1996.

[30] J. Campbell, *C Programmer's Guide to Serial Communications*. Morgan Kaufmann, 2002.

[31] J. Campbell, *USB Complete: Everything You Need to Develop Custom USB Peripherals*. Lakeview Research, 2nd edition, 2001.

[32] A. U. Khalid, Z. Zilic, and K. Radecka, "Quantum state detection and quantum frames," in *13th International Workshop on Post-Binary ULSI Systems*, pp. 66–70, May 2004.

[33] Y. Eldar and G. Forney, "Optimal Tight Frames and Quantum Measurement," *IEEE Transactions on Information Theory*, pp. 599–610, 2002.

[34] C. Helstrom, *Quantum Detection and Estimation Theory*. Academic Press, 1976.

[35] Y. Eldar and G. Forney, "On Quantum Detection and the Square-Root Measurement," *IEEE Transactions on Information Theory*, pp. 858–872, 2001.