FPGA Emulation of Quantum Circuits

Ahmed Usman Khalid
Microelectronics and Computer Systems Laboratory
McGill University
Montreal, Quebec
Email: akhalid@macs.ece.mcgill.ca

Zeljko Zilic
Microelectronics and Computer Systems Laboratory
McGill University
Montreal, Quebec
Email: zeljko@macs.ece.mcgill.ca

Katarzyna Radecka
Department of Electrical and Computer Engineering
Concordia University
Montreal, Quebec
Email: kasiar@ece.concordia.ca

***Abstract*— Quantum computing offers immense speedup in performing tasks such as data encryption and searching. The quantum algorithms can be modeled using classical computing devices, however classical computer simulations cannot deal efficiently with the parallelism present in quantum algorithms. The quantum circuit model for quantum algorithms is sufficient to describe the known quantum algorithms. Using analogies between quantum and digital circuits, we design the emulator of quantum algorithms in FPGAs that allows efficient experimentation with new quantum algorithms. This paper concentrates on new techniques for modeling quantum circuits, including the entanglement and probabilistic computing realization, as well as the critical issues in the required precision of computing.**

I. Introduction

There is an increased interest in quantum computing and algorithms [5]. Many quantum algorithms outperform their classical counterparts through parallelism that is impossible in classical computing. Such algorithms use physical effects like entanglement and super-position to achieve the speedup. These effects are hard to replicate at large scale and lead to reliability and precision issues, as well as to the need to employ suitable quantum measurement procedures [1]. Nevertheless, some of the quantum effects have been successfully used in practical applications such as data encryption and communication [5]. Further, several quantum-computing systems are being developed [7]. IBM has developed a small-scale quantum machine that is able to execute the celebrated Shor's algorithms for factoring numbers. However, creating larger and more practical quantum computers is still not possible, as knowledge about building quantum systems is still in its infancy. To develop quantum algorithms, simulation models nevertheless suffice.

Feynman noted that a quantum computer can be modeled efficiently only by another quantum machine. In absence of large-scale quantum machines, quantum algorithms are currently being simulated by classical computers. Modeling of quantum processes in software is the arduous task that is currently facilitated mostly by quantum computing libraries [3], [11], [12]. The challenge here comes from the need for using approximations of quantum processes, as their exact representation in classical computing is not possible. Even by using approximations, it is estimated that a single simulation run over a 20-bit quantum system requires a day of computing time on modern computers [8]. For developing quantum systems, it is advantageous to have a hardware emulator which approximates quantum effects, but mimics the parallel nature of quantum computation more closely than software-based simulators.

Quantum circuits are one convenient way of describing quantum algorithms. Such circuits comprise of analogues to digital bits and gates. These components can be emulated in existing FPGAs, which can map inherently parallel computational tasks more efficiently than software simulations. For this reason, we investigate the design of quantum circuit emulators by classical circuits, and devise an FPGA-based quantum circuit emulator. Using quantum circuit primitives, the construction of new quantum algorithms becomes intuitive and similar to the common software library approaches.

The paper is organized as follows. In Section II, we provide the background on quantum computation. In Section III, we give details of our quantum circuit emulation system, followed by several case studies and performance analysis in Section IV.

II. Background

In this section, we provide a brief review of the concepts in quantum computing that have implications to the design of a quantum circuit emulator.

A. Probabilistic vs. Deterministic Computing

One of the major distinctions of quantum computing is that it is probabilistic and that quantum algorithms have to deal with the reality of measurement errors. Surprisingly, this difference often gets overlooked when modeling quantum circuits and most modeling approaches still try to make the simulations fit within the deterministic mode of computation.

Deterministic circuits present a computation model where results of the computation can be obtained without any measurement error. A classical (non-quantum) probabilistic circuit runs a series of inputs through the network of gates and outputs the bits according to the probability distribution induced by the given network. Hence, in probabilistic computing the result of computation cannot be determined correctly every time a measurement of the result is made. Consequently, there is a probability of an error in measuring probabilistic circuit outputs, and the computation has to be performed a sufficient number of times to make the expected error acceptable. In this paper, we address both modes of modeling by FPGA emulators.

B. Quantum Information Representation

The second major difference to classical computing arises from the types of signal values required to perform computing. While the basic information units for classical circuits are 0 and 1, quantum computing uses complex numbers as bearers of information. Hence, representing a single quantum information unit might require a large number of classical bits, depending on the precision required.

More formally, the states of the quantum system belong to a vector space over complex numbers in which there exists an inner product of vectors. Such a vector space is usually referred to as the *Hilbert Space H*. For our purposes, it suffices to say that the quantum states are depicted as vectors of complex numbers.

In denoting these vectors, commonly used is the Dirac 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^* of all *operators* on the vector space that act on vectors and produce scalar values.

C. Quantum Bits

The basic units of quantum information can be viewed as simple two-state systems, such as magnetic spin of plus/minus one half. The state of a spin is given as a continuous quantity represented by two real numbers. It is exactly this continuity in spin representation that contributes to the ability of storing the infinite classical information by a single quantum system. Quantum bits defined this way are commonly referred to as *qubits*. Qubits can be realized by means such as NMR and trapped ion interactions.

Binary qubits have two *computational base states* denoted as $|0\rangle$ and $|1\rangle$. Unlike classical bits, quantum bits are in a linear superposition of the basis states $|0\rangle$ and $|1\rangle$.

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

where α and β are complex coefficients related as

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

The superposition phenomena, by which the qubits simultaneously exist in states $|0\rangle$ and $|1\rangle$ is explained by considering $|\alpha|^2$ and $|\beta|^2$ as probabilities of being in $|0\rangle$ and $|1\rangle$, respectively. However, when a measurement is performed on a qubit, it collapses to either of the two basis states.

III. FPGA Quantum Circuit Emulator

A. Challenges in Emulating Quantum Circuits

Emulation of quantum circuits requires mapping concepts from quantum physics to classical technologies. The main goal is to simulate quantum computation in a way that is more flexible and efficient than software simulators. As most quantum algorithms require an exponential amount of resources when simulated by classical technology, resource management is a key design issue. The second goal is to emulate the parallelism in quantum computing using FPGAs. Finally, it is desired that the modeling tool be simple to use and that the construction of the model does not require significant effort from the developer. The overall design process is illustrated in

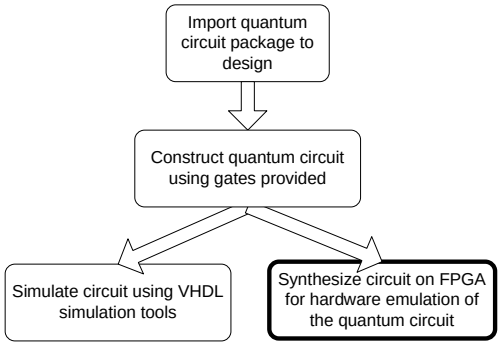

Fig. 1. Modeling quantum circuits using the VHDL quantum gate library

Figure 1. Quantum circuits are constructed from the quantum gate components provided in the library that we have created. The correctness of the circuit can be verified either by software simulation or by FPGA emulation. We thus, have a technique for modeling quantum ciruits using VHDL and then synthesizing the circuit in hardware to achieve the performance needed to make the whole process more practical.

We next show how the fundamental constructs of quantum circuits and the rules governing quantum computation are simulated by classical technology.

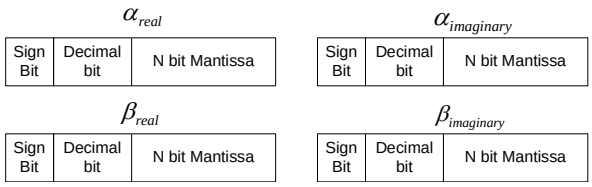

Fig. 2. Fixed-point quantum bit representation

B. Emulation of Pure Quantum Bits

The quantum bit is implemented using Equation 1. Thus, we need to store the values of α and β to describe a qubit. The finite precision description of α and β introduces imprecision errors, as quantum gates involve operations like add and multiply on α and β. To keep the size of the quantum circuit to a manageable proportion, we implement the α and β using the fixed point scheme described in Figure 2.

Each qubit is represented by four fixed point numbers. The fixed point scheme was chosen over the floating point representations because α and β can have a decimal part of 0 or 1 only. Having the exponent field in the number representation (as in floating point arithmetic) does not bring benefits in this case. Regarding the precision, the emulator has been designed in a modular way - changing the size of the fractional part is achieved without any modifications to the other components of the system. This is an advantage for experiments dealing with precision and fault-tolerance of

quantum algorithms and it incorporates the ideas of quantum error correction to the emulator.

Quantum gates, described by matrices of complex numbers, bring additional imprecision to the system. The error model of a quantum gate is depicted in Figure 3. 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.

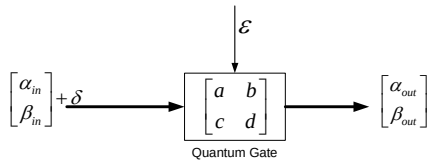

Fig. 3. Quantum gate error model

The error model can be expanded as in Figure 4. Then, the multiple sources of an error are added linearly. This model is used to evaluate the error at each gate in the network.

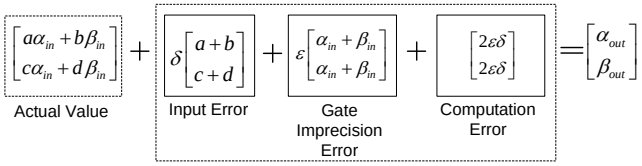

Fig. 4. Expanded gate error model

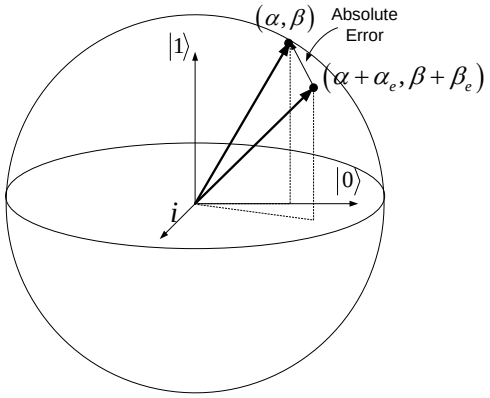

Fig. 5. Discretization error in a qubit

The absolute error *E* is thus,

$$E = \sqrt{\alpha_e^2 + \beta_e^2} \tag{3}$$

where α_e and β_e are described in Figure 4. These error values effect 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 5 - 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.

C. Quantum Gates

A quantum gate is the analogue of a logic gate in a classical circuit model. Few gates that are useful in developing quantum algorithms are given next, together with the error magnitudes obtained in their modeling by classical circuits.

Quantum systems are reversible by nature. Information can travel freely in both directions: from inputs to outputs and vice versa. Thus, each gate must have the same number of inputs and the outputs. The single input gates are defined by a 2×2 matrix with complex entries.

1) Walsh-Hadamard Gate: The Walsh-Hadamard gate (*H* gate) facilitates the superposition of pure quantum states.

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

If $|\psi\rangle = \alpha|0\rangle + \beta 1\rangle$ is the input qubit to the Hadamard gate, then the transformed state is:

$$H|\psi\rangle = \frac{1}{\sqrt{2}}((\alpha + \beta)|0\rangle + (\alpha - \beta)|1\rangle)$$

The new state $H|\psi\rangle$ is a superposition of the computational basis states. This gate is also referred to as the square root of the identity, since $H^2 = I$, and therefore $H|H|\phi\rangle = |\phi\rangle$.

The implementation of the H gate requires four multiplications and four additions, as both α and β are complex. Since the coefficients of the Hadamard gate cannot be represented without imprecision, the gate incurs a discretization error on both the α and β values of the output qubit.

2) Phase Shift Gate: The operation of the phase shift gate ϕ on the single qubit is defined in the following way:

$$|0\rangle \rightarrow |0\rangle \quad and \quad |1\rangle \rightarrow e^{i\phi}|1\rangle$$

The definition of a phase shift gate is given by matrix N_S as:

$$N_S = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

The phase-shift gate requires multiplication of two complex entities: $e^{i\phi}$ and β. This means that 4 multiplications and two additions have to be performed in this gate. Due to the finite representation of $e^{i\phi}$, a discretization error is incurred on the β value of the qubit at the output.

3) X-Gate: The *X*-gate is a single-qubit gate that performs the quantum equivalent of a NOT operation on the qubit. The quantum NOT operation simply swaps the α and β values of a qubit:

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

For the vector $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, this transformation amounts to producing

$$X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$$

The gate does not incur any error to the algorithm as the emulation of the gate involves swapping the bits of α and β

4) Controlled NOT Gate: Controlled NOT, or CNOT gate accepts two quantum bits: a control qubit, $|\eta\rangle$ and a target qubit, $|\psi\rangle$, and produces the outputs: $|\eta\rangle$ and $|(\eta \oplus \psi)\rangle$ where $\oplus$ is the XOR operation.

Its transform matrix in input order $|\eta\rangle$ and $|\psi\rangle$ is:

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

In simple terms, if $|\eta\rangle = |0\rangle$, then $|\psi\rangle$ is inverted, hence the name CNOT. This operation is well-defined for any linear superposition of states, unlike binary NOT gates which only convert 0 to 1, and vice-versa.

5) Z-Gate: The *Z*-gate is a single-qubit gate that inverts the phase of the qubit in $1\rangle$ basis.

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

That is,

$$Z|\psi\rangle = \alpha|0\rangle - \beta|1\rangle$$

The *Z*-gate does not introduce any error as it simply flips the sign bits for the complex beta value.

The data in Table I was obtained by computing the absolute error in the output of each gate using 16-bit mantissa length for gate coefficients and the inputs. The input to each gate was chosen to be $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. From experiment it was found that the error is maximum when the qubit is in a superposition state. For multi-qubit gates like the CNOT gate, the error varies depending on the entangled or unentangled nature of the output of the gates.

TABLE I
ABSOLUTE ERROR ON QUBIT IN STATE $|\psi\rangle = 1/\sqrt{2}|0\rangle + 1/\sqrt{2}|1\rangle$

Gate	Absolute Error (16-bit mantissa)
Hadamard Gate	3.05×10^{-5}
Phase-Shift Gate	3.08×10^{-5}
X-Gate	0
Z-Gate	0

D. Emulation of Quantum Gates

A library of common quantum gates has been developed comprising of most of the simple quantum gates (Hadamard gate, CNOT gate, X-gate, Z-gate, phase-shift gate) that are commonly used. The gates are realized by mapping their transformation to VHDL code. Therefore, gates with simple transformations like the X-gate and Z-gate require less resources than the other gates.

We then decided to use the code-generating capability of the VHDL language to automatically produce descriptions of multiple input quantum gates from single-input gates. In general, a n-input gate is represented by a 2^n by 2^n matrix. For controlled gates, the number of control variables is passed as a parameter to the code generating script. For efficiency reasons, the script produces the VHDL description of the resulting transformation, rather than a large matrix form. The outlined procedure hence automates the construction of arbitrary size quantum gates. To construct network of quantum gates, we

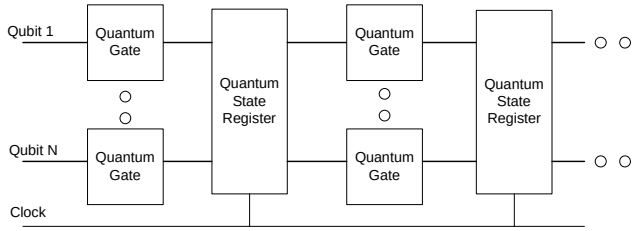

Fig. 6. Emulated quantum circuit overview

insert intermediate registers to hold the qubit values after each gate. These quantum state registers (QSRs) essentially represent the state of the entire quantum system at any given stage of evolution. Since the QSRs are clocked they can synchronize the data flow in the system, which is important for large circuits.

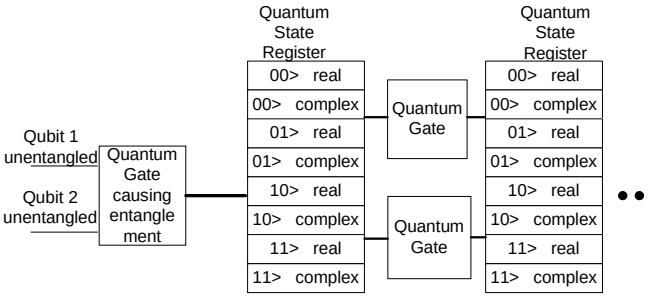

Fig. 7. Emulation of quantum evolution of an entangled system

E. Emulation of Entanglement

A gate resulting in *entanglement* requires considerably more resources than a gate where no entanglement occurs. This is the reason why efficient simulation of quantum computation is difficult on classical computing devices. Consider a two-input C-NOT gate and a situation where the controlled qubit is in superposition. In such a case the gate operation is described as follows

$$|\psi_{control}\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$$

$$|\psi_{target}\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$$

$$|\psi_{out}\rangle = \alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \beta_1\beta_2|10\rangle + \beta_1\alpha_2|11\rangle$$

In an entangled state, the qubits cannot be represented individually. For the case of two qubits $|\psi_1\rangle$ and $|\psi_2\rangle$, if they are unentangeled we can represent them as two distince qubits $|\psi_1\rangle|\psi_2\rangle$. However, once entangled the qubits can only be represented in $|\psi_1\psi_2\rangle$ form.

The C-NOT gate requires 4 complex multiplications in this case. For an n input C-NOT gate, the number of complex

multiplications is 2^n. This exponential increase becomes a serious issue with entangled systems. 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. While efforts are made to overcome this overhead in the resource usage when simulating the evolution of an entangled system [10], we note that the entanglement poses a fundamental bottleneck in modeling quantum systems by classical means.

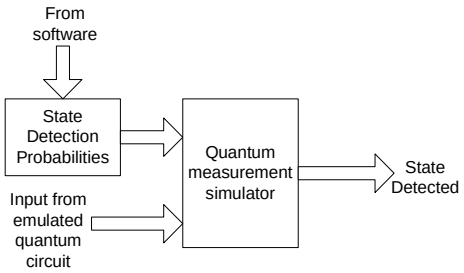

Fig. 8. Emulation of probabilistic quantum computing

F. Emulation of Quantum Measurements and Probabilistic Computing

Quantum measurements and the probabilistic nature of quantum algorithms are currently supported directly by a combination of hardware and software means. Due to the complexity of quantum measurement algorithms (which are being investigated and developed in parallel), software simulation of quantum measurements is currently employed [1]. We however foresee the possibility of emulating quantum measurements in hardware once the measurement algorithms become developed and stable. To perform the measurement in hardware, it suffices that the probabilities for detecting each state are pre-computed in software and stored in hardware. The probabilities can then be used as weights to emulate the random state detection in hardware.

IV. Quantum Circuit Case Studies

A. Emulator Mapping Results

The techniques presented in this paper offer the means for quantum circuit emulations in FPGAs by including quantum gate library and entanglement components, like most quantum software simulators [11]. No changes to standard FPGA mapping and the overall design flow are required.

Table II depicts the logic cell usage for the quantum gates in the library and that of the quantum circuits described above. The device chosen were Altera Stratix EP1S80B956C6. The simulation tool used is ModelSim and Leonardo Spectrum was used to obtain synthesis results. Since *X* and *Z* gates perform a swap and a bit flip respectively, they consume negligible resources.

As the number of qubits increases, the circuit size grows exponentially due to entanglement effects. However, from the emulation results we can observe that fairly large circuits can be constructed. The potential of this emulation technique for large quantum circuits is also evident as by combining multiple FPGAs we have a possibility of emulating the functionality and other pertinent quantum effects (like quantum error-correction) of more complex quantum circuits.

TABLE II
Gate Logic Cell Usage on Altera Stratix EP1S80B956C6

Gate	LC Usage (8-bit mantissa)	LC Usage (16-bit mantissa)
Hadamard Gate	704	1284
Phase-Shift Gate	386	708
C-Not Gate	40	231
X-Gate	0	0
Z-Gate	0	0

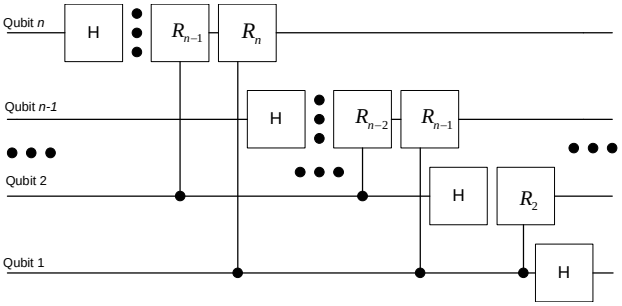

Fig. 9. A N-qubit QFT Circuit

B. Quantum Fourier Transform

The quantum Fourier transform (QFT) [2], [5], [6] is an important quantum algorithm as it plays a key role in phase estimation, order-finding and factoring algorithms. A N qubit QFT circuit is depicted in Figure 9. The QFT algorithm achieves exponential speedup compared even to FFT, which by itself is one of the most important nontrivial algorithms. The entire QFT circuit is constructed from one or two input quantum gates. The QFT comprises of Hadamard and the controlled-R_j gates. The R_j transform is defined as follows

$$R_j = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^j} \end{bmatrix}$$

The R_j gate is similar to the phase-shift gate so the controlled-R_j is implemented using controlled phase-shift gates. We implemented a 3-qubit QFT circuit using our library.

C. Grover's Search Algorithm

The Grover's search algorithm [5], [6], [9] is yet another illustration of quantum algorithms significantly outperforming classical algorithms. The algorithm performs searches in is $O(\sqrt{N})$ time, rather than classically possible $O(N)$, for a database with N entries. The Grover's search algorithm circuit for 4 element data base is depicted in Figure 10. The oracle gate is itself a "black box" quantum circuit that queries the database for the search key. The oracle gate can be constructed using the quantum gates provided in the library. In this circuit,

entanglement can occur depending on the query result from the oracle gate. The circuit in Figure 10 was implemented using our library.

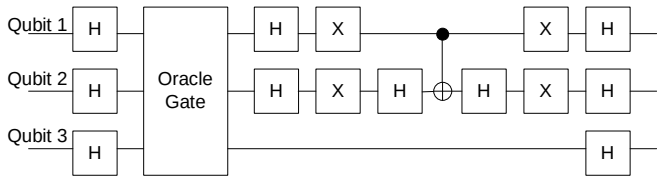

Fig. 10. Grover's search algorithm for a 4 element database

Emulation data for the algorithms is provided in Table III. Both of the algorithms were synthesized on the Altera Stratix EP1S80B956C6 chip using 16-bit mantissas.

TABLE III
QUANTUM CIRCUIT PERFORMANCE

Circuit	LC Usage (16-bit mantissa)	Clock Speed
3-qubit Quantum Fourier Transform	5076	82.1 MHz
Grover's Search	12636	82.1 MHz

D. Comparison to software simulation

The hardware emulation of quantum circuits has significant advantages over software simulators. First, the evolution of each quantum bit can occur in parallel. Software simulators have to sequentially simulate the evolution of each quantum bit. Efforts are being made to create parallel software simulators [4] but they cannot achieve the parallelism available through emulating quantum circuits using FPGA.

Secondly, by using QSRs, we can pipeline the quantum circuit. This introduces latency to the design but provides the ability to emulate quantum circuits for multiple inputs very efficiently. This is not possible in software and the algorithm has to be executed in its entirety for each new input.

In order to compare the performance of the FPGA emulator to software, we found the average time required to execute the above algorithms in software using [3]. The test was performed on a 2 GHz Pentium IV machine running Redhat Linux. The FPGA emulation of the same algorithms was performed using 16-bit mantissas to represent gate coefficients and inputs. The benchmark results are presented in Table IV. The FPGA emulator performs significantly better than the software simulator. For larger circuits, the disparity in performance is even greater because the sequential software simulation of the qubits' evolution becomes a more constricting bottleneck.

TABLE IV
BENCHMARKS

Circuit	Libquantum (seconds)	FPGA Emulator (seconds)
Quantum Fourier Transform	40×10^{-6}	61×10^{-9}
Grover's Search	90×10^{-6}	84×10^{-9}

V. CONCLUSIONS AND FUTURE WORK

We presented the issues in the design and the operation of a quantum circuit emulator based on FPGAs and developed a platform for the development of quantum circuits. The emulator allows the construction of fairly complex quantum circuits from the component library in a simple way. At the same time, it emulates the parallelism present in quantum computers by constructing parallel evolution paths for each quantum bit on the FPGA. We show that FPGA emulations are advantageous, as it is difficult to efficiently emulate the parallel evolution of the quantum system in software. The emulator is also scalable and has the potential of emulating complex quantum circuits. This emulator can also incorporate further quantum computing concepts like quantum error-correction, fault-tolerant quantum computing and quantum measurement techniques. These would be especially helpful in developing practical systems for quantum computers.

Other uses of the quantum emulator can be for the analysis, optimization and approximation of quantum Fourier transforms, which are critical for most spectacular quantum algorithms. We plan to further explore the development of quantum/reversible gate libraries and specialized architectures for emulation of quantum algorithms [8] and [15].

Finally, we plan to undertake a study and optimization of quantum measurement algorithms using this emulator, which is currently not practical by software simulations.

REFERENCES

[1] A.U. Khalid, Z. Zilic, K. Radecka; *Quantum State Detection using Frames*, Int. workshop on Post-Binary ULSI Systems, May 2004

[2] Z. Zilic and K. Radecka, *The Role of Super-fast Transforms in Speeding up Quantum Computations*, Proceeding of the 32nd IEEE International Symposium on Multiple-Valued Logic (ISMVL02), 2002

[3] "Libquantum", Online Quantum Library Documentation, http://www.enyo.de/libquantum/

[4] Kevin M. Obenland and Alvin M. Despain, *Parallel Quantum Computer Simulation*, UGC 1998

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

[6] Gruska, Jozef; *Quantum Computing*, Cambridge Univ. Press, 1999

[7] M. Steffen, L. M. K. Vandersypen and I.L. Chuang, *Toward Quantum Computation: A Five-Qubit Quantum Processor*, IEEE Micro, Vol. 27, No. 1, pp. 24-34, 2001

[8] J.P. Hayes and I.L. Markov, *Simulation, Synthesis and Testing of Quantum Circuits*, DARPA QuIST annual research review, Beverly Hills, CA, June 2003

[9] V.V. Shende, A.K. Prasad, I.L. Markov and J.P. Hayes, *Synthesis of Reversible Logic Circuits*, IEEE Transactions. on CAD, vol 22(6), June 2003

[10] G.F. Viamontes, M. Rajagopalan, I. L. Markov and J.P. Hayes, *Gate-Level Simulation of Quantum Circuits*, ASPDAC, January 2003

[11] QCL - A Programming Language for Quantum Computers, Quantum Computing Library Online, http://tph.tuwien.ac.at/ oemer/qcl.html

[12] D. Greves, "QDD : A Quantum Computer Emulation Library", http://thegreves.com/david/QDD/qdd.html

[13] P. W. Shor, "Polynomial Time Algorithms for Prime Factorization and Discrete Logarithm", *SIAM Journal of Computing*, 26(5), 1484-1509, 1997.

[14] H. M. Wiseman, *Quantum Trajectories and Quantum Measurement Theory*, Quantum Semiclass 1996

[15] M. Fujishima, K. Saito, M. Onouchi, H. Hoh; *High-speed processor for quantum-computing emulation and its applications*, IEEE International Symposium on Circuits and Systems, Volume: 4, 25-28 May 2003