Modeling a Quantum Processor using the QRAM Model

Conference Paper · August 2011

DOI: 10.1109/PACRIM.2011.6032928

CITATIONS
5

READS
529

3 authors:

Mostafa Elhoushi
Huawei Technologies
21 PUBLICATIONS **158** CITATIONS
SEE PROFILE

M. W. El-Kharashi
University of Victoria
168 PUBLICATIONS **935** CITATIONS
SEE PROFILE

Hatem Elrefaei
Ain Shams University
7 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Modeling a Quantum Processor using the QRAM Model

Mostafa Elhoushi
Mentor Graphics Egypt, inc.
mostafa_elhoushi@mentor.com

M. Watheq El-Kharashi
Computer & Systems Engineering Department
Ain Shams University
Cairo, Egypt
watheq@engr.uvic.ca

Hatem Elrefaei
Engineering Physics & Mathematics Department
Ain Shams University
Cairo, Egypt
hatem@mcit.gov.eg

Abstract

A quantum processor controlled by classical instructions is modeled. The processor follows the quantum random access machine (QRAM) model: a quantum memory and a quantum ALU (qALU) containing a group of quantum gates, which are controlled by classical signals. The processor's instructions are written using a quantum assembly (QASM) language. As test cases, several well-known quantum circuits are described using the QASM language and executed by our model. This model may later be integrated with classical components to form a hybrid quantum computer.

1. Introduction

The objective of designing a quantum processor is to have a general-purpose quantum module capable of executing arbitrary quantum algorithms. Using the resulting processor, a quantum algorithm is to be described using instructions of the QASM language [1]. The instructions are then interpreted into classical instructions to be loaded into a classical instruction memory within the processor. The instructions are executed sequentially by decoding each instruction and applying the required quantum operation on the required qubits. This concept of manipulating quantum operations using classical signals is known as the QRAM model [2]. An example of a physical implementation of quantum computation which employs this concept is an on-chip integrated waveguide circuit where some quantum optical devices can be controlled by a varying voltage to manipulate the quantum state of photons [3].

Previously, several quantum programming languages have been designed to describe quantum algorithms. Such languages include Quantum Computing Language (QCL) [4], Quantum Programming Language (QPL) and communication capable QPL (cQPL) [5], and LanQ [6]. An extension to the classical C++ language named Q Language [7] was designed to be executed on a classical processor with a quantum module. Although the designers of some of these languages intended to use theirs as descriptions of application-specific quantum circuits to be used by future software tools to generate quantum circuit designs, the programs of all of these "high-level" languages maybe compiled to low-level QASM programs to be executed by quantum processors.

As a first step towards a physical implementation of a quantum processor, a two-qubit quantum processor was implemented using ion traps, which can execute arbitrary quantum gate operations in arbitrary order [8]. However, rather than a physical implementation, this paper is concerned with providing a simulation model of a quantum processor which can simulate executing arbitrary programs written using the QASM language. M. Oskin et al. proposed a physical architecture of a quantum processor, which consists of a quantum memory and a qALU [9]. We have chosen to implement this model using VHDL so that it may later be interfaced with VHDL models of classical processors to control it.

HDL simulation of quantum circuits was first proposed by M. Udrescu et al. [10] and later extended to be synthesizable on an FPGA [11] [12]. However, such work only simulated application-specific quantum circuits each executing a single algorithm, rather than being general-purpose quantum circuits. In this paper, we shall model our quantum processor using an HDL language and the resulting module can be considered as a general-purpose quantum circuit.

This paper is organized as follows. In Section 2, a brief background on quantum computation is introduced. In Section 3, the proposed model is explained. Simulation case studies are shown in Section 4. Challenges of physically implementing the model are discussed in Section 5. Finally, conclusion and future work are mentioned in Section 6.

978-1-4577-0253-2/11/$26.00 ©2011 IEEE

2. Background

In classical computation, either a bit has a value of 0 or 1. In quantum computation [13], a quantum bit – or a qubit – can be considered to have both values of 0 and 1 at the same time. This concept known as superposition is the main advantage of quantum computation since it allows gate operations or algorithms to deal with several values in one step. However, when a qubit is measured, its value collapses to either 0 or 1 and the superposition property no longer applies.

This superposition advantage emerges only in some physical systems and simulating it using classical mechanics is usually inefficient. Feynmann pointed out that it is impossible to simulate quantum mechanics efficiently using classical computation [14], and mentioned that a computer which employs quantum mechanics may efficiently simulate quantum mechanics computation.

A qubit can be represented in vector form:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$$

where $|\alpha|^2$ is the probability that when the qubit is measured it collapses to 0 and $|\beta|^2$ is the probability that when it is measured it collapses to 1. According to probability theory, the condition $|\alpha|^2 + |\beta|^2 = 1$ must always hold true.

A quantum state, in a general form, consists of several qubits. A multiple qubit state maybe represented by the tensor product of each qubit:

$$\begin{aligned}|\psi\rangle &= |\psi_0\rangle\otimes|\psi_1\rangle\\ &= (\alpha_0|0\rangle + \beta_0|1\rangle)\otimes(\alpha_1|0\rangle + \beta_1|1\rangle)\\ &= \alpha_0\alpha_1|00\rangle + \alpha_0\beta_1|01\rangle + \beta_0\alpha_1|10\rangle + \beta_0\beta_1|11\rangle\end{aligned}$$

For example, this means that the probability of measuring both qubits to be $|00\rangle$ is $|\alpha_0\alpha_1|^2$. Multi-qubits may also be represented in vector form as:

$$|\psi\rangle = \begin{bmatrix}\alpha_0\alpha_1\\\alpha_0\beta_1\\\beta_0\alpha_1\\\beta_0\beta_1\end{bmatrix}$$

Generally, an n-qubit system requires a vector representation of 2^n elements.

It is important to note that however a group of independent qubits maybe represented by tensoring each of their qubit representations, operations may occur on the qubits to set them into an entangled state. Entanglement, which is another feature which distinguishes quantum computation from classical computation, occurs when a state of qubits cannot be expressed as a tensor product of individual qubits. In this case, an operation acting on one qubit cannot be simulated by considering the vector representation of that qubit but by considering the vector representation of all of the qubits [15]. This is another reason why quantum computation is more efficient than classical computation: in an algorithm which employs entanglement, one physical quantum operation step maybe simulated by a large number of classical computation steps.

Operations are applied to qubits by quantum gates. A quantum gate manipulates the qubit vector representation. An example of a quantum gate, considering photon polarization implementation of qubits, is an optical polarizer, which changes the polarization of photons [16]. Mathematically, a quantum gate maybe simulated using matrix representation. For example, the quantum NOT gate inverses the value of a qubit:

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

Note that the inverse of any quantum gate, U, i.e., the gate which returns $U|\psi\rangle$ back to $|\psi\rangle$ is $U^{-1} = U^{\dagger}$, where $U^{\dagger}$ is the complex conjugate of U.

An example of 2-qubit gate is the controlled-NOT gate. The first qubit it acts on remains unchanged while the second qubit is XORed with the first qubit:

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

Some quantum gates may have a parameter to determine its operation. For example, considering photon polarization implementation of qubits [16], a wave plate with the angle of its optical axis equivalent to π/k keeps the $|0\rangle$ coefficient of a photon as it is and multiplies the coefficient of $|1\rangle$ by $e^{2\pi i/2^k}$. The matrix representation of such a gate is:

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

Figure 1 shows a summary of a group of quantum gates.

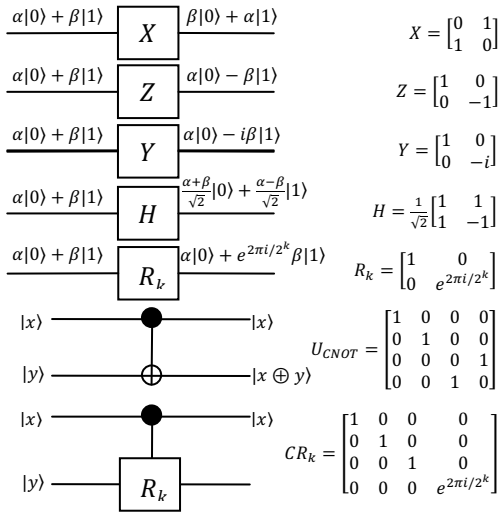

Figure 1. Summary of main quantum gates.

A quantum algorithm is a description of a sequence of quantum operations applied upon quantum bits. There are 3 models to model quantum algorithms. The first model is the quantum circuit model [17]. In this model, quantum gates, each performing a certain operation on one or more qubits, are connected in a similar manner as classical logic gates to form a logical circuit. A second model which is an extension to the quantum circuit model, which shall be used in our paper, is the QRAM model [2]. In this model, a group of qubits or quregisters receives instructions from a classical computer in the form: "apply gate U to qubits q0 and q1", or "measure qubit q0". The third model is the quantum Turing machine model [13]. This model is rather theoretical and can be proved to be equivalent to any of the previous 2 models [13].

3. Proposed Model of Quantum Processor

The quantum processor, as shown in Figure 2, constitutes of a classical instruction memory, a qALU, a quantum register, Q, containing N qubits and a classical register, C, containing N classical bits each representing the measurement result of each quantum bit. A classical instruction, structured in the QASM format, as it will be explained in Section 3.1, is read from the classical instruction memory and loaded into the instruction register of the qALU.

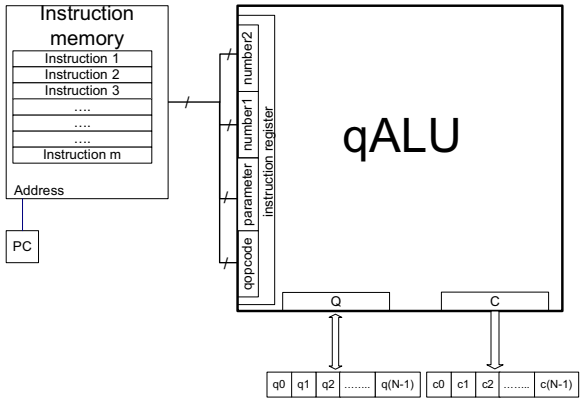

Figure 2. Model of the quantum processor containing: classical instruction memory, quantum ALU, quantum register and classical register.

The qALU, shown in Figure 3, constitutes of an arbitrary number of quantum gates, two decoders, and classical control signals derived from the instruction register. The classical control signals of each quantum gate are simply `enable_input` and `enable_output` signals which are derived from the quantum opcode of the instruction via each decoder. `number1` and `number2` signals are also derived from their respective fields within the instruction and determine which qubits are the gates required to operate upon. A quantum gate may have a `parameter` signal to modify the value of a matrix coefficient of its matrix representation.

In addition to holding quantum gates, qALU has a component to prepare a qubit to either $|0\rangle$ or $|1\rangle$, and a component to measure the value of a qubit and sets the corresponding bit of the classical register, C, to the measured value.

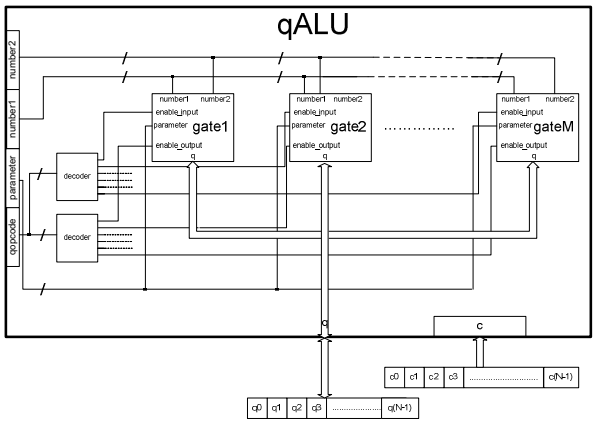

Figure 3. Internal structure of qALU.

3.1. QASM Instruction Set

QASM has been previously mentioned as a programming language [1] and a text-format language [18] but a detailed description of its instruction set has not been developed yet. Here, we propose an instruction set and a format for each instruction.

As shown in Figure 4, each QASM instruction constitutes of:

- *qopcode*: defining the required quantum operation,
- *parameter*: used by some quantum gates to set a value of a matrix coefficient of the matrix representation of the gate,
- *number1*: to determine which qubit to apply the gate upon, and
- *number2*: used by some quantum gates to determine the second qubit to apply the gate upon.

Table 1 shows the QASM instruction set which the proposed processor model supports. Note that we have also implemented the inverse of the R_k and C_{R_k} gates, i.e., the ${R_k}^{\dagger}$ and ${C_{R_k}}^{\dagger}$ gates respectively, using the same instruction but with a negative parameter.

qopcode	parameter	number1	number2

Figure 4. Format of QASM instruction.

Theoretically, there is an infinite number of gates since any gate may have any matrix representation with any coefficients as long as the matrix representation is unitary [13]. However, a set of elementary gates $\{\boldsymbol{H}, \boldsymbol{R_k}, \boldsymbol{C_{R_k}}\}$, is sufficient to represent an arbitrary quantum gate up to a certain precision [19]. Therefore, our instruction set contains more than enough gates to represent any circuit containing arbitrary gates.

Table 1. QASM instruction set.

Instruction	Meaning
`q param, num`	If *param*=0 then set all qubits to $\|0\rangle$. If *param*=1 set qubit *num* to $\|1\rangle$.
`X num`	Apply X gate to qubit *num*.
`Z num`	Apply Z gate to qubit *num*.
`Y num`	Apply Y gate to qubit *num*.
`H num`	Apply H gate to qubit *num*.
`Rk param, num`	If *param*>=0 then apply R_k gate with k=*param* to qubit *num*. If *param*<0 then apply $R_k{}^{\dagger}$ gate with k=|*param*| to qubit *num*.
`CNOT num1, num2`	Apply $CNOT$ gate to qubit *num1* controlled by qubit *num2*.
`CRk param, num1, num2`	If *param*>=0 then apply CR_k gate with k=*param* to qubit *num1* controlled by qubit *num2*. If *param*<0 then apply CR_k gate with k=|*param*| to qubit *num1* controlled by qubit *num2*.
`SWAP num1, num2`	Swap qubits *num1* and *num2*.
`MEASURE num`	Measure qubit *num1* and set *num1*th bit of the classical register, C, to the measured value.

3.2. VHDL Model of Quantum Gates

To simulate the N-qubit register in VHDL, a vector of 2^N complex numbers was used. This was done to handle the general case in which all qubits are entangled and cannot be represented as N individual qubits.

Since a quantum gate theoretically and practically represents an operation over time upon a qubit rather than having a qubit input and qubit output, quantum gates access the quantum register via an `inout` port as shown in Figure 5. A `clock` signal is only used to advance "simulation time" while `reset` is used to reset internal variables and mathematical structures, and both signals have no correspondence in a physical implementation of a quantum gate.

Interfacing a quantum register via an `inout` port requires synchronization and a resolution function since in VHDL terms, all of the gates may drive a value to the same quantum register. Therefore a complex `NULL_COMPLEX` value was defined so that any idle gate drives a `NULL_COMPLEX` value to the complex vector representing the quantum register. The VHDL resolution function was developed to consider the first non-`NULL_COMPLEX` value when more than one driver drives the same qubit vector.

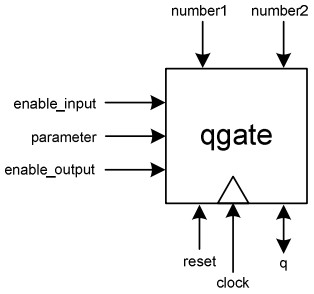

Figure 5. VHDL model of a quantum gate.

In addition to that, a process was developed within the architecture of each quantum gate to do the following:

- `if (enable_input = '1')`, gate reads in the value of the quantum register and multiplies it with its matrix representation. The result is stored in an internal temporary quantum register within the gate, which is only used for the simulation purpose.
- `elsif (enable_output = '1')`, gate drives the system quantum register value to equal that of the gate's temporary quantum register.
- `else` gate drives `NULL_COMPLEX` qubit vector to the quantum register.

The timing diagram in Figure 6 shows the sequence used by the qALU to apply each quantum gate operation on the quantum register:

1. set the `number` bus to the qubit(s) number(s) required to apply the gate operation on.
2. set `enable_input` signal for one clock cycle and then reset it.
3. set `enable_output` signal for two clock cycles and then reset it.

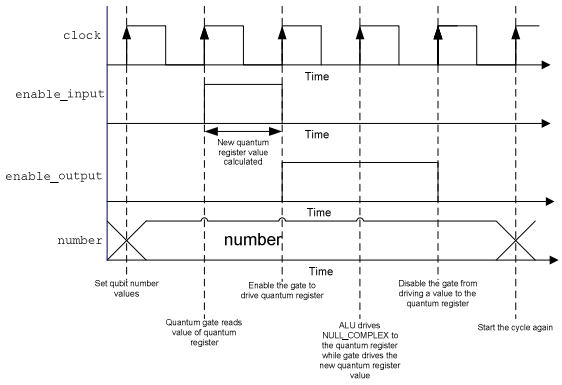

Figure 6. Timing diagram of quantum gate signals.

4. Simulation Case Studies

In this section we show 3 case studies to demonstrate the flexibility of the quantum processor to execute many quantum programs. We have instantiated a 4-qubit quantum processor to run the example programs. All of the gates' matrix representations in the following case studies have been stated in Figure 1 while their corresponding QASM instructions have been stated in Table 1.

A Perl script was developed to translate a QASM program into a hex file representing the machine code. Our quantum processor VHDL design loads the machine code from the hex file into the classical instruction memory (referring to Figure 2) and the processor loops to execute each instruction.

4.1. EPR Creation Circuit

We start with a case study illustrating a simple circuit and measure its output. The EPR creation circuit, shown in Figure 7, is used to set qubits into an entangled state. It has been named after the founders of the quantum entangled state principle: Einstein, Podolsky and Rosen [13].

If $|q_0\rangle$ and $|q_1\rangle$ are both set to $|0\rangle$, then the state of the qubit system after applying the H gate and the $CNOT$ gate is $1/\sqrt{2}\,(|00\rangle + |11\rangle)$. Therefore, the result of measuring both qubits is either $|00\rangle$ or $|11\rangle$. Note that we have only used 2 of the 4-qubits of the quantum processor.

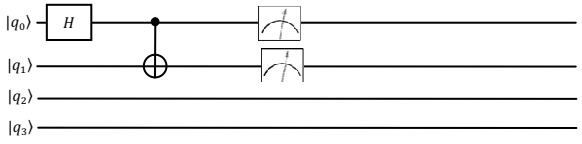

Figure 7. EPR creation circuit.

In Figure 7, the horizontal lines represent time flow. Therefore the gates are implemented in sequential order from left to right. So the circuit diagram states that we first apply the H gate on $|q_0\rangle$, then we apply the $CNOT$ gate, controlled by $|q_0\rangle$, on $|q_1\rangle$, then we measure the qubits. The QASM program which initializes all qubits to the $|0\rangle$ state and executes the circuit is shown in Figure 8.

```
q       0,0
H       0
CNOT    1,0
MEASURE 0
MEASURE 1
```

Figure 8. QASM program to implement EPR creation circuit.

Simulation results showing the Q register before measurements and the C register after measurements are shown in Figure 9.

Q	[(0.707+0i), -- 0000	(0+0i), -- 1000
	(0+0i), -- 0001	(0+0i), -- 1001
	(0+0i), -- 0010	(0+0i), -- 1010
	(0.707+0i), -- 0011	(0+0i), -- 1011
	(0+0i), -- 0100	(0+0i), -- 1100
	(0+0i), -- 0101	(0+0i), -- 1101
	(0+0i), -- 0110	(0+0i), -- 1110
	(0+0i), -- 0111	(0+0i)] -- 1111
C	[0,0,0,0]	

Figure 9. Simulation results of EPR creation circuit.

4.2. Toffoli Gate Circuit

This case study illustrates the ability of our processor to simulate gates which are not within its instruction set. The Toffoli gate is a 3-qubit gate, which is considered the quantum controlled-NAND gate. As shown in Figure 10, the third qubit is inverted if both the first and second qubits are set to $|1\rangle$. According to [19], the implementation of the Toffoli gate using our set of gates is shown in the lower half of Figure 10. Note that we have only used 3 of the 4-qubits of the quantum processor.

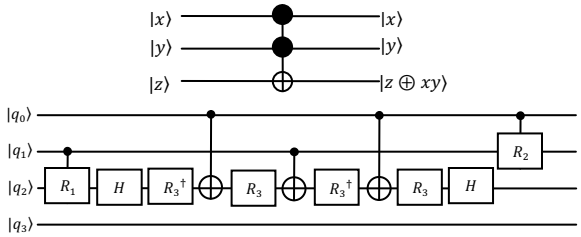

Figure 10. Toffoli gate (above) and its circuit implementation using our gate set (below).

The QASM program which initializes qubits $|q_0\rangle$ and $|q_1\rangle$ to $|1\rangle$ and qubit $|q_2\rangle$ to $|0\rangle$ and executes the circuit is shown in Figure 11.

```
q       1,0
q       1,1
CRk     1,2,1
H       2
Rk      -3,2
CNOT    2,0
Rk      3,2
CNOT    2,1
Rk      -3,2
CNOT    2,0
Rk      3,2
H       2
CRk     2,1,0
```

Figure 11. QASM program to implement Toffoli gate.

The simulation results, shown in Figure 12, show that, as expected, qubits $|q_0\rangle$ and $|q_1\rangle$ remain at $|1\rangle$ and qubit $|q_2\rangle$ changes to $|1\rangle$.

```
Q   [(0+0i),      -- 0000   (0+0i),      -- 1000
    (0+0i),      -- 0001   (0+0i),      -- 1001
    (0+0i),      -- 0010   (0+0i),      -- 1010
    (0+0i),      -- 0011   (0+0i),      -- 1011
    (0+0i),      -- 0100   (0+0i),      -- 1100
    (0+0i),      -- 0101   (0+0i),      -- 1101
    (0+0i),      -- 0110   (0+0i),      -- 1110
    (1+0i),      -- 0111   (0+0i)]      -- 1111
```

Figure 12. Simulation results of Toffoli gate circuit.

4.3. Quantum Fourier Transform Circuit

This case study considers the quantum Fourier transform circuit [20], which is the quantum counterpart of the classical Fourier transform. As an example, we shall demonstrate a 4-bit quantum Fourier transform whose circuit diagram is shown in Figure 13.

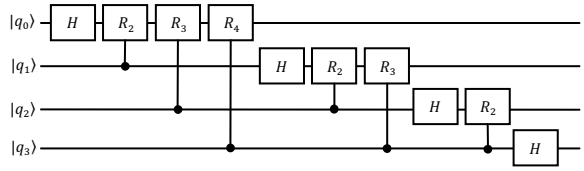

Figure 13. 4-bit quantum Fourier transform circuit.

The circuit acts upon 4 qubits: $|q_0\rangle$, $|q_1\rangle$, $|q_2\rangle$ and $|q_3\rangle$. The QASM program, initializing all qubits to $|0\rangle$ and implementing the required circuit is shown in Figure 14.

```
q       0,0
H       0
Rk      2,0,1
Rk      3,0,2
Rk      4,0,3
H       1
Rk      2,1,2
Rk      3,1,3
H       2
Rk      2,2,3
H       3
```

Figure 14. QASM program to implement 4-qubit quantum Fourier transform.

Our simulation results, shown in Figure 15, show that, as expected, the result of applying quantum Fourier transform to $|0000\rangle$ is that the 4 qubits have equal probabilities of having any value ranging from $|0000\rangle$ to $|1111\rangle$.

```
Q   [(0.2498+0i),-- 0000   (0.2498+0i), -- 1000
    (0.2498+0i), -- 0001   (0.2498+0i), -- 1001
    (0.2498+0i), -- 0010   (0.2498+0i), -- 1010
    (0.2498+0i), -- 0011   (0.2498+0i), -- 1011
    (0.2498+0i), -- 0100   (0.2498+0i), -- 1100
    (0.2498+0i), -- 0101   (0.2498+0i), -- 1101
    (0.2498+0i), -- 0110   (0.2498+0i), -- 1110
    (0.2498+0i), -- 0111   (0.2498+0i)] -- 1111
```

Figure 15. Simulation results of 4-qubit quantum Fourier transform circuit.

Within our quantum processor, we only had to instantiate one instance of each type of gate found in the previous 3 circuits and executed each of the gates in sequential order with different qubit inputs as stated in each QASM program. The same quantum processor maybe used to execute any other program provided its machine code representation is loaded into the classical instruction memory. This is one of the advantages over VHDL simulation models, [10], [11] and [12], which would have instantiated each gate more than once and could only execute the circuit at hand.

5. Physical Implementation Challenges

Although there exists some physical implementations of quantum computers operating on a small number of qubits, implementing our model physically faces many issues [21]. Implementing qubit registers is not yet feasible since there are difficulties in storing a qubit or keeping it in a superposition state without decoherence for a long time. Implementing quantum "wires" to transport qubits reliably from a quantum register to a quantum gate and back again to the quantum register also has its own problems. In addition to that, physical operations of quantum gates introduce their own errors.

Nevertheless, we have provided here a framework for describing a quantum circuit via an assembly language and a processor model to simulate. The interface to our model is identical to that of a physical quantum processor despite that its internal structure is more complex.

6. Conclusion & Future Work

The outcome of this work is a simulation model whose interface is identical to that of a real physical quantum processor, as described in [9]: control signals and qubit number signals, so that a classical computer can control it. The VHDL model is unique from previous VHDL work in that the resulting module may receive commands to implement an arbitrary operation on an arbitrary qubit at runtime, while other work involved a "hard-coded" circuit pre-defined at compilation time.

We have simulated quantum gate operations using mere matrix operations, which maybe inefficient. The model maybe enhanced in the future by incorporating more efficient simulation techniques, such as probabilistic methods [22]. The QASM language, which we have defined, is considered a low-level language. In the future, various compilers maybe developed to translate many of the high-level languages discussed in literature to QASM programs in

order to be executed on our quantum processing module.

Our module can be further integrated, as a quantum module, within a classical processor. Therefore, a program containing both classical and quantum parts maybe executed on this hybrid processor. As a result, a program written in a language such as the Q Language [7], which contains both classical and quantum parts, may have a compiler to generate the classical and quantum assembly instructions to be executed on a single hybrid quantum processor.

6. References

[1] K. M. Svore, A. V. Aho, A. W. Cross, I. Chuang, and I. L. Markov, "A Layered Software Architecture for Quantum Computing Design Tools", *IEEE Computer*, vol. 39, no. 1, January 2006, pp. 74-83.

[2] E. H. Knill, "Conventions for quantum pseudocode", *LANL report LAUR-96-2724*, Los Alamos, NM, USA, 1996, pp. 1-10.

[3] J. Matthews, A. Politi, A. Stefanov, and J. O'Brien, "Manipulating multi-photon entanglement in waveguide quantum circuits", *Nature Photonics*, vol. 3, no. 6, 2009, pp. 346-350.

[4] B. Ömer, "A Procedural Formalism for Quantum Computing", *Master Thesis*, Technical University of Vienna, 1998.

[5] P. Selinger, "Towards a quantum programming language", *Mathematical Structures in Computer Science*, vol. 14, no. 4, 2004, pp. 527-586.

[6] H. Mlnařík, "Quantum programming language LanQ", *PhD Thesis*, Masaryk University, Faculty of Informatics, Brno, Moravia, Czech Republic, 2007.

[7] S. Bettelli, T. Calarco, and L. Serafini, "Toward an architecture for quantum programming", *The European Physical Journal D*, vol. 25, no. 2, 2003, pp. 181-200.

[8] D. Hanneke, J. P. Home, J. D. Jost, J. M. Amini, D. Leibfried, and D. J. Wineland, "Realisation of a programmable two-qubit quantum processor", *Nature Physics*, vol. 6, 2009, pp. 13-16.

[9] M. Oskin, F. T. Chong, and I. L. Chuang, "A Practical Architecture for Reliable Quantum Computers", *IEEE Computer*, vol. 35, No. 1, January 2002, pp. 79-87.

[10] M. Udrescu, L. Prodan, and M. Vlădutiu, "Using HDLs for describing quantum circuits: A framework for efficient quantum algorithm simulation", *Conference on Computing Frontiers*, Ischia, Italy, April 2004, pp. 96-110.

[11] A.U. Khalid, Z. Zilic, and K. Radecka, "FPGA emulation of quantum circuits", *IEEE International Conference on Computer Design*, San Jose, California, USA, November 2004, pp. 310-315.

[12] M. Aminian, M. Saeedi, M. Zamani, and M. Sedighi, "FPGA-Based Circuit Model Emulation of Quantum Algorithms", *Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI*, Montpellier, France, April 2008.

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

[14] R. P. Feynman, "Simulating physics with computers", *International Journal of Theoretical Physics*, Springer, vol. 21, 1982, pp. 467-488.

[15] G. F. Viamontes, I. L. Markov, and J. P. Hayes, *Quantum Circuit Simulation*, Springer, Heidelberg, Germany, 2009.

[16] J. L. O'Brien, "Optical Quantum Computing", *Science*, vol. 318, no. 5856, 2007, pp. 1567-1570.

[17] R. Cleve, "An introduction to quantum complexity theory", in *Collected Papers on Quantum Computation and Quantum Information Theory*, C. Macchiavello, G. M. Palma, and A. Zeilinger, World Scientific, 2000, pp. 103-127.

[18] Quanta Group, *Quantum circuit viewer: qasm2circ*, Massachusetts Institute of Technology, March 2005, Available: http://www.media.mit.edu/quanta/qasm2circ, [Accessed: 8 June 2011].

[19] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, "Elementary gates for quantum computation", *Physical Review A*, vol. 52, no. 5, 1995, pp. 3457-3467.

[20] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, Santa Fe, NM, Nov. 20–22, IEEE Computer Society Press, 1994, pp. 124–134.

[21] T. S. Metodi, and F. T. Chong, *Quantum Computing for Computer Architects*, Morgan & Claypool Publishers, San Rafael, CA, USA, 2006.

[22] M. Van den Nest, "Simulating quantum computers with probabilistic methods", *Workshop on Quantum Algorithms, Computational Models, and Foundations of Quantum Mechanics*, Vancouver, BC, Canada, February 2010.