

# Quantum Circuit Project

Due Nov. 18, 2019

## Introduction

In this project you will use a quantum computer simulator to explore several simple quantum circuits. The tasks are simple and shouldn't take you much time, but be sure to start on the project early in case you have trouble running the simulator or some other technical problem. (You are welcome to use another quantum simulator if you prefer, such as `Quantum-Computer-Simulator` on the course website.)

## About the jQuantum Simulator

jQuantum (<http://jqquantum.sourceforge.net/>) is a quantum computer simulator using the *circuit architecture* (as opposed to another type of quantum computer architecture, such as annealing). It seems to have been developed by a German named Andreas de Vries in 2004 and may have been updated last in 2010. According to the documentation, it is still a beta product and is not likely to be intended for any market other than inquisitive quantum computing students. However, it will prepare you for using quantum computers now becoming available in the cloud.

jQuantum is a Java-based application that can run either natively on one's machine or directly from the web-site. Initial testing showed few bugs, but there may be infrequent crashes that require killing the program.

While the bugs mentioned above were not encountered during heavy qubit operations, it is important to remember that this is only a quantum computer simulator, not a quantum computer. Thus, while a quantum computer would be able to handle  $2^n$  states easily, the jQuantum simulator must simulate  $2^n$  states, which can quickly overwhelm a modern-day laptop, even at a seemingly low  $n$  (approximately 10).

## Using the jQuantum Simulator

The jQuantum simulator is a simple point-and-click application. Most of the information you need to install (or not) and run the application can be found in the first three pages of the documentation file (<http://jqquantum.sourceforge.net/jQuantum.pdf>) or the *Help with jQuantum* item in the *Help* menu of the application.

There are three rather non-intuitive features of the application. First, is the display of the quantum registers. jQuantum allows for two quantum registers, named  $x$  and  $y$  respectively (the exercises below only make use of one register). The values are displayed as blocks or a grid. An example of a two-qubit register (with four basis states,  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ ) is displayed below on the next page.



Hovering the cursor over each block displays a pop-up menu of the value that the respective block represents:



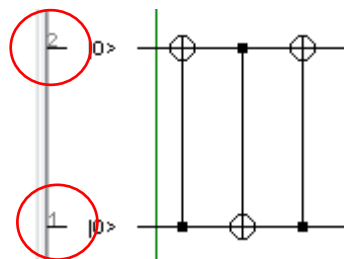
Each block is colored corresponding to its current value (the color wheel is available via the help menu, but in general black represents no value, red represents a positive real value, and light blue represents a negative real value; other colors represent complex amplitudes).

In a classical computer, only one state could be represented at a given point in time, but in a quantum computer, a register can have multiple states simultaneously:



In the above picture, the register is holding the states  $|00\rangle$  and  $|10\rangle$ .

The second non-intuitive aspect of using `jQuantum` is the qubit numbering. Notice that in a register, each qubit has a numerical designator (this is an underlined number at the far left of the qubit).



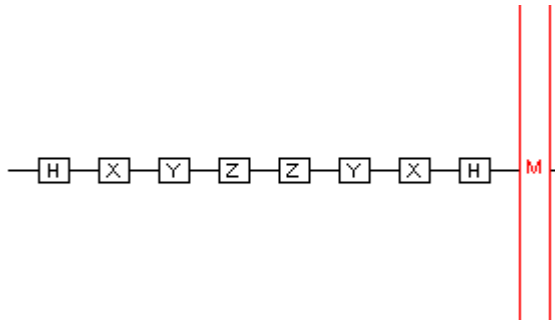
This qubit number is used when creating gates that use more than one qubit and when assigning initial qubit values. The qubits are displayed in the register in ascending order (based on their qubit number) from left to right.

Lastly, one must remember that this is not a drag-and-drop application. You cannot insert a gate into a circuit; you can only add additional gates to the end of the circuit. Thus, a mistake early on in a circuit

will require that one delete the circuit right-to-left to return to the mistake and then rebuild the circuit left-to-right.

## Questions

1. The four different Bell states can be constructed with only one circuit by changing the initial qubit values. Build this circuit and run simulations for each initial qubit value combination and verify that the corresponding Bell state is created (take screen captures to show each simulation result). (Note that you can use the leftmost button in the pane above your circuit to change the initial qubit values of your circuit.)
2. What is the effect of the following single qubit circuit? Include screen shots and explain your results.



3. Build a 3 “bit” random number generator. Include a screen shot of your circuit and several outputs.
4. Demonstrate superdense coding. Set up a quantum circuit with four qubits. The top two qubits will be initialized to  $|z\rangle$  and  $|x\rangle$  representing the bits  $z$  and  $x$  to be transmitted. The bottom two qubits will be initialized to  $|0\rangle$  and used to generate the entangled pair. At the end of your circuit, the bottom two qubits should encode  $z$  and  $x$ . Show and explain your results for each  $zx$  pair. Take your screen shots before any final measurements so that we can see superpositions. (Hint: you will have to use another conditional operator to program the conditional **Z** operator.)
5. For this question you will demonstrate quantum error correction. Implement a triple redundancy encoder and the corresponding syndrome extraction circuits (section III.D.5.d, pp. 164–6). You will need six qubits. Qubits 1 to 3 will be used for the syndrome and should be initialized to  $|000\rangle$ . Qubits 4 to 6 will be used for the encoded qubit, so qubits 4 and 5 should be initialized to  $|00\rangle$ . Qubit 6 will be used for the quantum state that we want to encode. For this question, initialize it to  $|1\rangle$  and apply the Hadamard gate so that we are encoding  $(|0\rangle - |1\rangle)/\sqrt{2}$ . Prefix this to your circuit to do triple-redundancy encoding. Next, simulate a bit-flip error by applying an X gate to one of qubits 4 to 6. Finally, apply your circuit for syndrome extraction. Run your quantum program and include a screen shot showing your circuit and the final register state. Explain why the final state is correct (assuming that it is!). (Note: you cannot measure the syndrome and use that to complete the error correction, since there is a bug in the `jQuantum` simulator’s implementation of single-qubit measurement.)