Quantum Code 0.1
User Manual

Doc. T005 - Jan. 2016

Nader Khammassi
n.khammassi@tudelft.nl
QuTech, Computer Engineering Lab
TU Delft, The Netherlands

Summary

Quantum Code is a low-level language for describing quantum circuit. It is very similar to the QASM language, but introduces more features which are mainly related to the execution and the simulation of quantum circuits. This document describes the syntax and the semantic of the Quantum Code. Several circuits are given as examples to help the quantum programmer.

1. Notations

- Code examples are shown within a box.
- QCode pre-defined keywords are noted in mallow color, for example: **“qubits”, “measure”, “cnot”…**

2. Syntax

2.1. Case sensitivity and Comments

The QC language is not case-sensitive, i.e. upper case letters are equivalent to lower case one.

To make the code more readable, the quantum programmer can add comments in his code. Comments starts with “**#**” and can be added either in a separate line or at the end of a line containing code as in the following example:

```
x q0   # inject a bit-flip

# parity check
measure q1   # measure the first ancilla
measure q2   # measure the second ancilla
```

2.2. Qubits Definition

A. Specifying Qubit Number

qubits *number*

Qubits number should be defined before in the beginning of the QC file before any gate definition. Example: Defining a quantum register with 17 qubits. We not that all qubits are initialized to zero at the time of creation of the register.

qubits *17*

B. Default Qubit Identifier

Once the number of qubits defined, the qubits can be addressed individually through its default identifier "**q***n*" where "n" is the identifier of the target qubit (in our example, n is in [0..16], so qubits identifier are "**q0**", "**q1**",… or "**q16"**").

For example, applying a pauli-x gate to the qubit 5 can be specified through the following line:

x q5

C. Naming Qubits

In order to give a meaningful name to each qubit and make the quantum program more readable, it is possible to name qubits using the keyword "**map**". For example, if we want to use the qubit 1 as an axilla and we want to name it "a0" instead of "q1". we can do it as follow:

map q1,a0

The previous line means that "a0" is mapped to qubit "q1". After that line, "q1" is equivalent to "a0". For example the 2 following lines are equivalents:

x q1

x a0

D. Binary Register

By default, a binary register is associated to the quantum register. It is mainly used to store the result of measurements (*or predict the value of non-entangled qubits (experimental)*). Typically after measuring a qubit "**q0**", the result of the measurement is stored into a bit "**b0**". The later ("**b0**") can be used to apply binary-controlled gates to some qubits. The following example shows how we measure a qubit "**q0**" then use the measured bit (stored in b0) to control a pauli-x gate which we apply to a second qubit "**q1**":

```
measure q0
cx b0,q1
```

2.3. Quantum Gates

Available gates are listed in the following table:

Quantum Gate	Keyword	Example	Notes
Hadamard	**H**	h q0	
Pauli-X	**X**	x q3	
Pauli-Y	**Y**	y q0	
Pauli-Z	**Z**	z q5	
CNOT	**“CNOT” or “CX”**	cnot q1, q3 cx q3, q1 cnot q1, q3	• Control qubit is the first argument, the target qubit is the second. • “cx” and “cnot” are equivalent, the only difference is that “cx” can be used to perform a binary controlled gate if a bit is given as a first argument (control bit).
Toffoli	**Toffoli**	toffoli q0,q1,q3	• Control qubits are “q0” and “q1”.
Swap	**SWAP**	swap q1, q2	
Rx	**RX**	rx q0, 1.553	• The angle is given in radian.
Ry	**RY**	ry q3, 0.327	• The angle is given in radian.
Rz	**RZ**	rz q9, 132	• The angle is given in radian.
Phase	**Ph**	ph q0	• Apply a phase gate (S).
Controlled Phase Shift with an angle $\frac{\pi}{2^k}$ where k = control_qubit - target_qubit	**CR**	cr q0,q1	• This gate is designed specifically to ease the specification of the Rk gates used in the QFT.
Controlled Pauli-Z	**CZ**	cz b1,q1	• b1 is the control bit.
Binary-Controlled X	**CX**	cx b0,q0	• b0 is the control bit.

2.4. Debugging and Monitoring Tools:

In order to visualise the evolution of the quantum state and the results of measurement, two monitoring directive can be inserted at any position of the circuit : “**display**” and “**display_binary**”.

A. Displaying the Quantum State

The directive “**display**” can be used to display both the quantum state and the binary register values. The quantum state is shown as a list of the amplitude of the different states composing the quantum state. The binary register shows either the results of measurement (if measurement has been performed) or a prediction of the value. The prediction mechanism keeps track of the binary values starting from their initial values and updating these values each time a gate is applied or a measurement is performed. The shown values can be “**0**”, “**1**” or “**X**”. The value changes to “**X**” (unknown state) when there is a superposition of states, for example when a *Hadamard* gate is applied to a given qubit, it associated “bit” in the binary register turns to “**X**”.

Example: in the following example we display the initial state then we apply a *Pauli-X* gate on **q0**, we display the result of *bit-flip*, then we a *Hadamard* gate on “**q0**”, then a *CNOT* on “**q1**” using “**q0**” as control qubit and finally we display the quantum state:

```
qubits 2
display
x q0
display
h q0
cnot q0,q1
display
```

The result of the execution of these lines shows the following output which contains the result of the three “**display**” directives:

```
-------------[quantum state]-------------
   (1.000000,0.000000) |00> +
   (0.000000,0.000000) |01> +
   (0.000000,0.000000) |10> +
   (0.000000,0.000000) |11> +
[>>] binary register:  | 0 | 0 |
-----------------------------------------------

-------------[quantum state]-------------
   (0.000000,0.000000) |00> +
   (1.000000,0.000000) |01> +
   (0.000000,0.000000) |10> +
   (0.000000,0.000000) |11> +
[>>] binary register:  | 0 | 1 |
---------------------------------------

-------------[quantum state]-------------
   (0.707107,0.000000) |00> +
   (0.000000,0.000000) |01> +
   (0.000000,0.000000) |10> +
   (0.707107,0.000000) |11> +
[>>] binary register:  | X | X |
-----------------------------------------------
```

B. Displaying only the Binary Register

When we use a lot of qubits (too verbose quantum state), or we want to display only the measurement results of some qubits such as ancillas, we can display only the binary register using the "**display_binary**" then only the binary register will be displayed. In the following example, we display the initial state then we flip the qubit 0 and we display only the binary register.

```
qubits 2
display_binary
x q0
display_binary
```

The execution of this circuit gives the following result:

```
-------------[quantum state]-------------
[>>] binary register:  | 0 | 0 |
-----------------------------------------------

-------------[quantum state]-------------
[>>] binary register:  | 0 | 1 |
-----------------------------------------------
```

2.5. Defining Sub-Circuits:

The quantum programmer can split his circuit into several parts which performs different tasks and gives different names to these sub-circuits. To do so, the programmer can use "labels" such as in the following example: the first sub-circuit is called "***init***" and is responsible to initialise the q0 to the 1,then the encoding sub-circuit named "***encoding***" encode the logical state |1>. The "***error injection***" sub-circuit inject a bit-flip on qubit 1, the "parity_check" sub-circuit extract the syndrome… etc

```
 5 # qubit definition
 6 qubits 5
 7
 8 # init logical gate
 9 .init
10     x q0
11     display_binary
12
13 # encoding
14 .encoding
15     cnot q0 q1
16     cnot q0 q2
17     display_binary
18
19 # error injection
20 .error_injection
21     x q1
22     display_binary
23
```

```
25 # parity check
26 .parity_check
27     cnot q0,q3
28     cnot q1,q3
29     cnot q0,q4
30     cnot q2,q4
31     measure q3
32     measure q4
33     # display
34     display_binary
35
36 # (error correction can be done here)
37 #.error_correction
38
39 # decoding
40 .decoding
41     cnot q0,q2
42     cnot q0,q1
```

2.6. Simulator Configuration Directives: (TO DO)

A. Error Model Specification

- *Symmetric/Asymmetric Depolarizing Channel (Error Probability Specification)*
- *Phase damping*
- *…*

B. Error Correction Scheme Selection

For automatic fault-tolerant operation synthesis:

- *Steane 7Q*
- *17Q Ninja-Star*
- *…*

C. Physical Platform Selection

Qubit implementation specification (Spin, Super-conducting …)

- *Coherence time*
- *Measurement error-model.*
- *Gate Operation duration*

… To be completed …