QX 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 basic QASM language, but it 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 understand it.

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 to 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
```

E. Naming Bits

Similarly to the Qubits, the Bits can be renamed too to make the code more readable:

```
map b0,mybit0
```

After this line, "**mybit**" can be used instead of "**b0**".

In the next example, we use the qubit "**q0**" as an ancilla qubit, we name it "**ancilla**". When "**q0**" is measured, the result of measurement is stored by default in "**b0**", so we can rename it "**ancilla_measurement**" to make the code more readable.

```
map q0,ancilla
map b0,ancilla_measurent
...
measure ancilla
cx ancilla_measurement, q1
```

2.3. Quantum Gates

The Quantum Code syntax support a quantum gate set which includes single and multiple (2,3) qubit(s) gates. It provide support to common controlled gates such as CNOT and Toffoli gates. In addition QC allows the circuit writer to use binary-controlled gates which use the outcome of qubit measurements to control several quantum gates.

The 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	
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** / **S**	**ph** q0 **s** q0	• Apply a phase gate (S). • Ph and S are equivalent.
T gate	**T**	**t** q0	• Apply a T gate.
T dagger (conj-transpose)	**Tdag**	**tdag** q0	• Apply a T dagger gate.
CNOT	**“CNOT” or “CX”**	**cnot** q1, q3 **cx** q3, q1	• 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	
CPHASE / CZ	**CPHASE** / **CZ**	**cphase** q0,q2 **cz** q0,q2	• ‘cphase’ and ‘cz’ are equivalent. • ‘cz’ can be used also as binary controlled pauli-z gate.
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.
Binary-Controlled Pauli-Z	**CZ**	**cz** b1,q1	• b1 is the control bit.
Binary-Controlled Pauli-X	**CX**	**cx** b0,q0	• b0 is the control bit.
Prepare in I 0 > state	**PREPZ**	**prepz** q0	• Initialize the target qubit in a ground state.

Note : as stated before, Quantum Code is not case-sensitive, for instance, “**CNOT**” and “**cnot**" are the same.

2.4. Measurements

A. Partial Measurement (Single Qubit)

Qubits can be measured individually using the keyword "**measure**" followed by the target qubit as in the following example:

```
measure q0
```

A. Register Measurement (All Qubits)

The entire quantum register can be measured at once using same keyword without specifying any target qubit as in the following example:

```
measure
```

2.5. 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 non-null amplitudes of the different states composing the quantum state.

The binary register shows either the outcome of measurement (if measurement has been performed) or a prediction of the measurement 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 the previous lines shows the following output which contains the result of the three "**display**" directives:

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

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

--------------[quantum state]--------------
   (0.707107,0.000000) |00> +
   (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 |
----------------------------------------------------
```

2.6. 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. The names of the circuits being executed are then displayed one by one. For instance the quantum or the binary register can be printed at the end of each sub-circuit execution to visualize the intermediate states allowing the programmer to monitor the execution of his circuit step by step and debug it.

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 the physical data **q0** to the |**1>** state, then the encoding sub-circuit which is named "***encoding***" encodes the logical state **|1>**. The "***error_injection***" sub-circuit inject a bit-flip on qubit 1. For the sake of brevity, the "**parity_check**" , "***error_correction***" and "***decoding***" sub-circuits are not omitted in the example.

```

# quantum error correction circuit
# 3 qubits bit-flip code

# qubit definition
qubits 5

# init logical gate
.init
   x q0
   display_binary

# encoding
.encoding
   cnot q0,q1
   cnot q0,q2
   display_binary

# error injection
.error_injection
   x q1
   display_binary
   …
```

Executing the previous code using the *QX Simulator* will display the following output:

```
[+] loading circuit from 'qec_3q_bitflip_code.qc' ...
[-] loading quantum_code file 'qec_3q_bitflip_code.qc'...
[+] code loaded successfully.
[+] initializing xpu...
[+] initialized.
[+] creating quantum register of 3 qubits...
[+] executing circuit 'init' ...
[>>] binary register:  | 0 | 0 | 1 |
-------------------------------------------
[+] circuit execution time: 0.000147 sec.
[+] executing circuit 'encoding' ...
[>>] binary register:  | 1 | 1 | 1 |
-------------------------------------------
[+] circuit execution time: 0.003223 sec.
[+] executing circuit 'error_injection' ...
[>>] binary register:  | 1 | 0 | 1 |
-------------------------------------------
[+] circuit execution time: 9.4e-05 sec.
```

2.7. Error Model : Noise Simulation

By default the *QX Simulator* executes the circuits using perfect qubits and perfect gates, i.e. without any noise or decoherence. However real-word qubit implementation suffers from decoherence and circuits are realised using imperfect gates introducing “noisy” operations. Finally the qubits are not perfectly isolated and the surrounding environment is and additional noise source which contribute to the introductions of errors into the circuits.

The *QX Simulator* implements currently two error models: the symmetric depolarising channel and the operational noise or operational errors (*work in progress*). When specified by the user, the *QX Simulator* execute the circuit “under noise” using the specified error model with the user-defined configuration such the probability of errors.

A. The Depolarizing Channel Simulation

One of these error model is the “Depolarizing Channel”. Given a probability of single qubit error per “***step***” (we consider each gate of the circuit as a “***step***”), this error model can inject errors into the circuit. These errors are injected in a form of bit-flips (***x error***), phase-flip (***z error***) or both in the same time (**y error**). The *QX Simulator* implements the so called “Symmetric Depolarizing Channel” which use equal probabilities for x,y and z errors.

In order to execute the circuit using the depolarizing channel the user has to add a single line in his circuit description:

```
error_model depolarizing_channel, 0.001
```

This line tells the *QX Simulator* to use the depolarising channel to simulate the circuit execution under noise. The “**0.001**” is the probability of a single physical qubit error, the higher this probability the more errors are injected into the circuit.

When running the circuit under noise, an error report is printed before the execution. The report indicate the number of errors injected in the circuit, their location and their type.
For instance, if we add the later code at the end of the quantum code of the error correction circuit, and we run the simulator, we obtain the following output:

```
[+] loading circuit from 'qec_3q_bitflip_code.qc' ...
[-] loading quantum_code file 'qec_3q_bitflip_code.qc'...
   * using the error model "depolarizing_channel" with error_probability=0.1
[+] code loaded successfully.
[+] initializing xpu...
[+] initialized.
[+] creating quantum register of 3 qubits...
[+] generating noisy circuits...
[>] processing circuit 'init'...
   [e] depolarizing_channel : injecting errors in circuit 'init'...
   [+] circuit steps : 2
   [>>>] error injection step 0 : number of affected qubits: 1
     l--> error on qubit  1 (x error)
   [+] total injected errors in circuit 'init': 1
[>] processing circuit 'encoding'...
   [e] depolarizing_channel : injecting errors in circuit 'encoding'...
   [+] circuit steps : 3
   [>>>] error injection step 0 : number of affected qubits: 1
     l--> error on qubit  1 (x error)
   [>>>] error injection step 2 : number of affected qubits: 1
     l--> error on qubit  2 (y error)
   [+] total injected errors in circuit 'encoding': 2
[>] processing circuit 'error_injection'...
   [e] depolarizing_channel : injecting errors in circuit 'error_injection'...
   [+] circuit steps : 2
   [+] total injected errors in circuit 'error_injection': 0
[+] total errors injected in all circuits : 3
[+] executing circuit 'init(noisy)' ...
[>>] binary register:  l 0 l 1 l 1 l
------------------------------------------
[+] circuit execution time: 0.000147 sec.
[+] executing circuit 'encoding(noisy)' ...
[>>] binary register:  l 1 l 1 l 1 l
------------------------------------------
[+] circuit execution time: 0.003223 sec.
[+] executing circuit 'error_injection' ...
[>>] binary register:  l 1 l 0 l 1 l
------------------------------------------
[+] circuit execution time: 9.4e-05 sec.
```

B. The Operational Errors

…TO BE CONTINUED…

C. Quantum Decoherence

…TO BE CONTINUED…

NOTES