QUANTUM PROGRAMS

Anton Karazeev

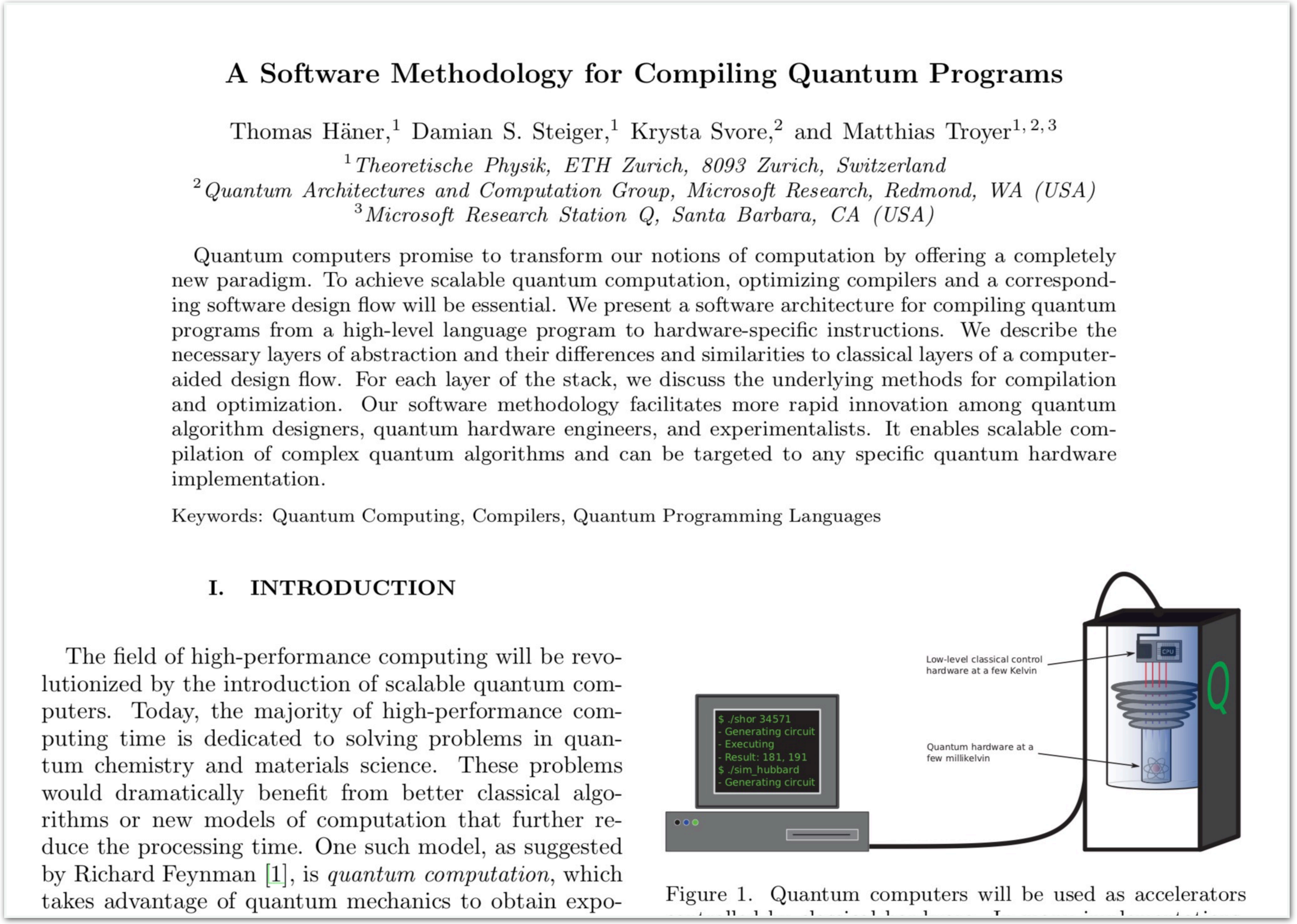
Department of Innovation
and High Technologies

MIPT, 2017

Outline

- Applications of quantum computers
- Building blocks of QC
- IBM Q experience

Paper



A Software Methodology for Compiling Quantum Programs

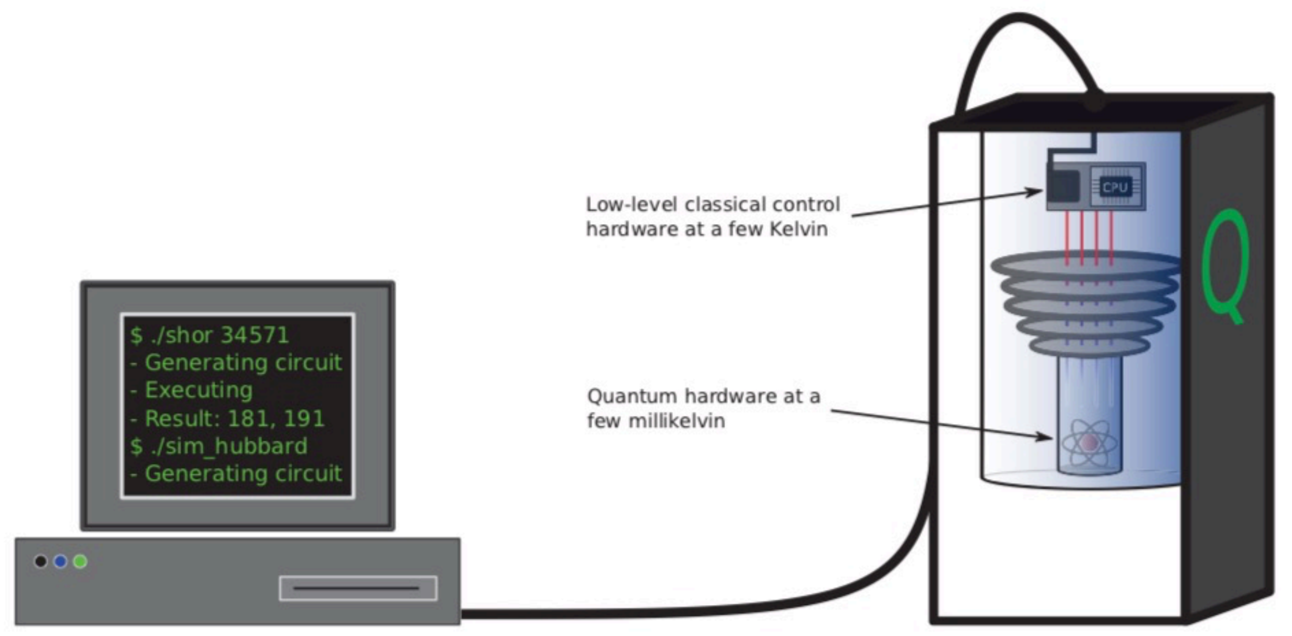
Thomas Häner,[1] Damian S. Steiger,[1] Krysta Svore,[2] and Matthias Troyer[1, 2, 3]
[1] *Theoretische Physik, ETH Zurich, 8093 Zurich, Switzerland*
[2] *Quantum Architectures and Computation Group, Microsoft Research, Redmond, WA (USA)*
[3] *Microsoft Research Station Q, Santa Barbara, CA (USA)*

Quantum computers promise to transform our notions of computation by offering a completely new paradigm. To achieve scalable quantum computation, optimizing compilers and a corresponding software design flow will be essential. We present a software architecture for compiling quantum programs from a high-level language program to hardware-specific instructions. We describe the necessary layers of abstraction and their differences and similarities to classical layers of a computer-aided design flow. For each layer of the stack, we discuss the underlying methods for compilation and optimization. Our software methodology facilitates more rapid innovation among quantum algorithm designers, quantum hardware engineers, and experimentalists. It enables scalable compilation of complex quantum algorithms and can be targeted to any specific quantum hardware implementation.

Keywords: Quantum Computing, Compilers, Quantum Programming Languages

I. INTRODUCTION

The field of high-performance computing will be revolutionized by the introduction of scalable quantum computers. Today, the majority of high-performance computing time is dedicated to solving problems in quantum chemistry and materials science. These problems would dramatically benefit from better classical algorithms or new models of computation that further reduce the processing time. One such model, as suggested by Richard Feynman [1], is *quantum computation*, which takes advantage of quantum mechanics to obtain expo-


Low-level classical control hardware at a few Kelvin
CPU
Q
Quantum hardware at a few millikelvin

```
$ ./shor 34571
- Generating circuit
- Executing
- Result: 181, 191
$ ./sim_hubbard
- Generating circuit
```

Figure 1. Quantum computers will be used as accelerators

https://arxiv.org/abs/1604.01401 [2016]

Applications of quantum computers

- Materials science
- Cryptography
- Machine learning

https://arxiv.org/abs/1604.01401

Building blocks of QC

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

The state of one qubit

https://arxiv.org/abs/1604.01401

Building blocks of QC

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

The state of one qubit

$$\begin{pmatrix}\alpha_0\\\alpha_1\end{pmatrix} \otimes \begin{pmatrix}\beta_0\\\beta_1\end{pmatrix} = \begin{pmatrix}\alpha_0\beta_0\\\alpha_0\beta_1\\\alpha_1\beta_0\\\alpha_1\beta_1\end{pmatrix}$$

The state of the entire system
with two qubits

https://arxiv.org/abs/1604.01401

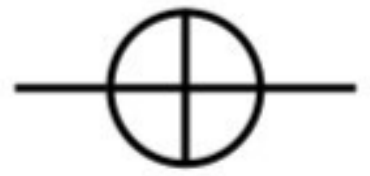
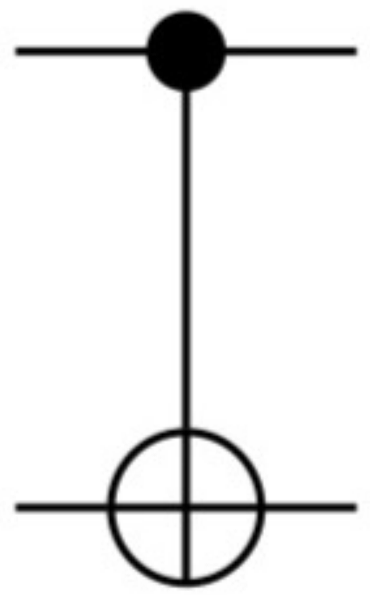
Building blocks of QC

$$\alpha\left|0\right\rangle + \beta\left|1\right\rangle = \alpha\begin{pmatrix}1\\0\end{pmatrix} + \beta\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}\alpha\\\beta\end{pmatrix}$$

The state of one qubit

$$\begin{pmatrix}\alpha_0\\\alpha_1\end{pmatrix} \otimes \begin{pmatrix}\beta_0\\\beta_1\end{pmatrix} = \begin{pmatrix}\alpha_0\beta_0\\\alpha_0\beta_1\\\alpha_1\beta_0\\\alpha_1\beta_1\end{pmatrix}$$

The state of the entire system with two qubits

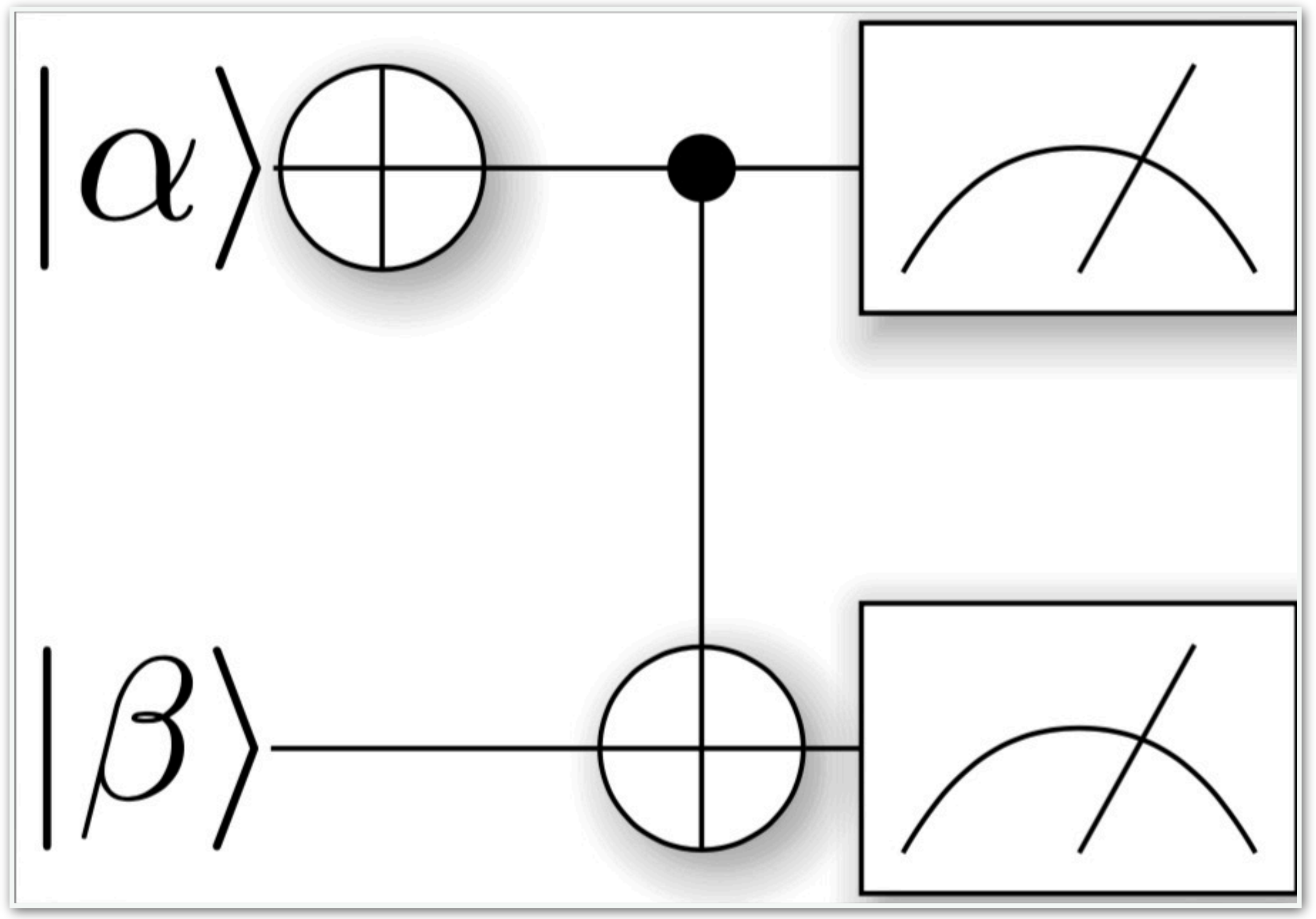
Gate	Matrix	Symbol
NOT or X gate	$\begin{pmatrix}0&1\\1&0\end{pmatrix}$	
Controlled NOT (CNOT)	$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}$	

Gates (to change qubit’s state)

https://arxiv.org/abs/1604.01401

Quantum circuits

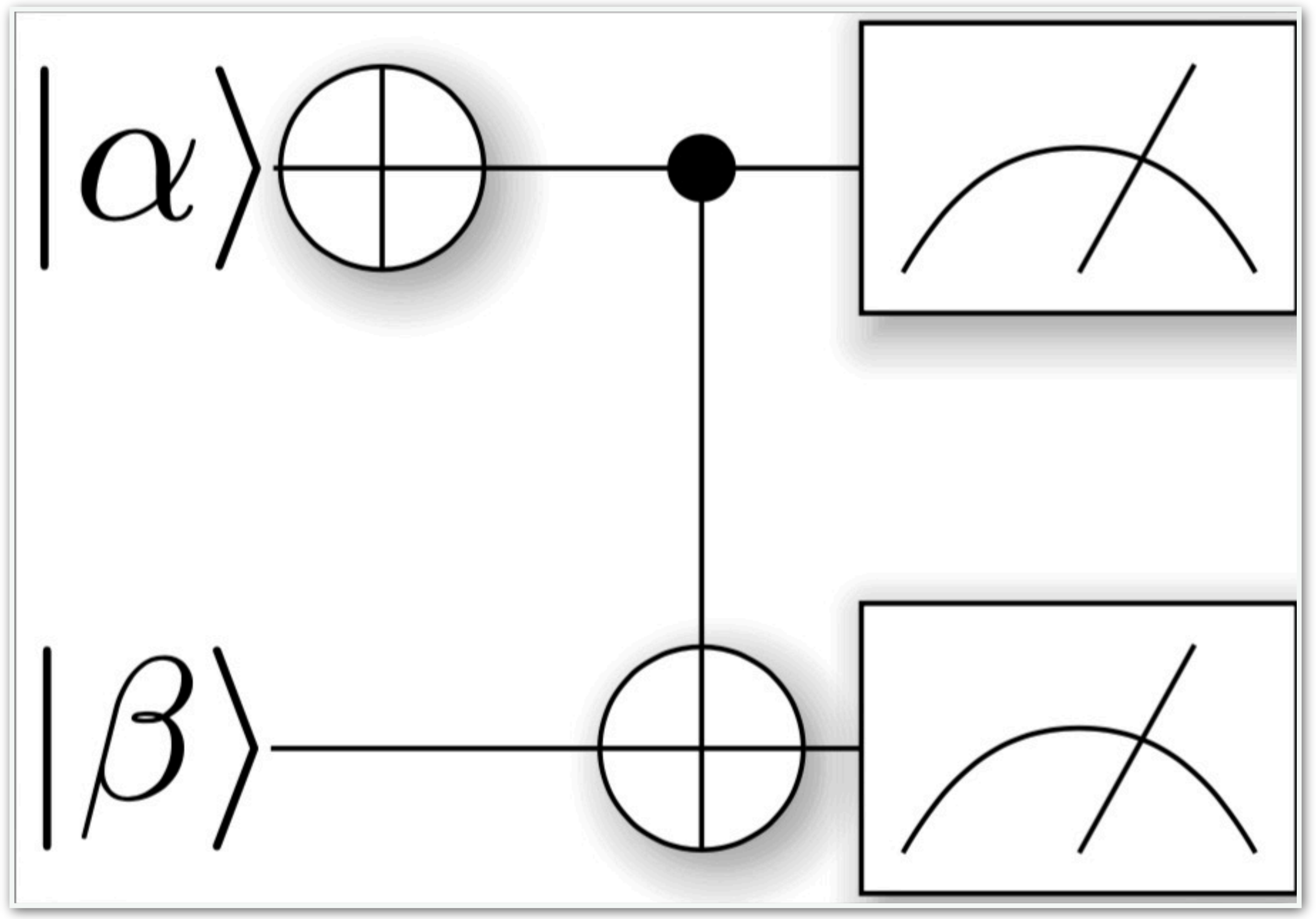
Apply a NOT gate to the first qubit, then a CNOT gate



$$
\begin{aligned}
|00\rangle &\rightarrow |11\rangle \\
|01\rangle &\rightarrow |10\rangle \\
|10\rangle &\rightarrow |00\rangle \\
|11\rangle &\rightarrow |01\rangle
\end{aligned}
$$

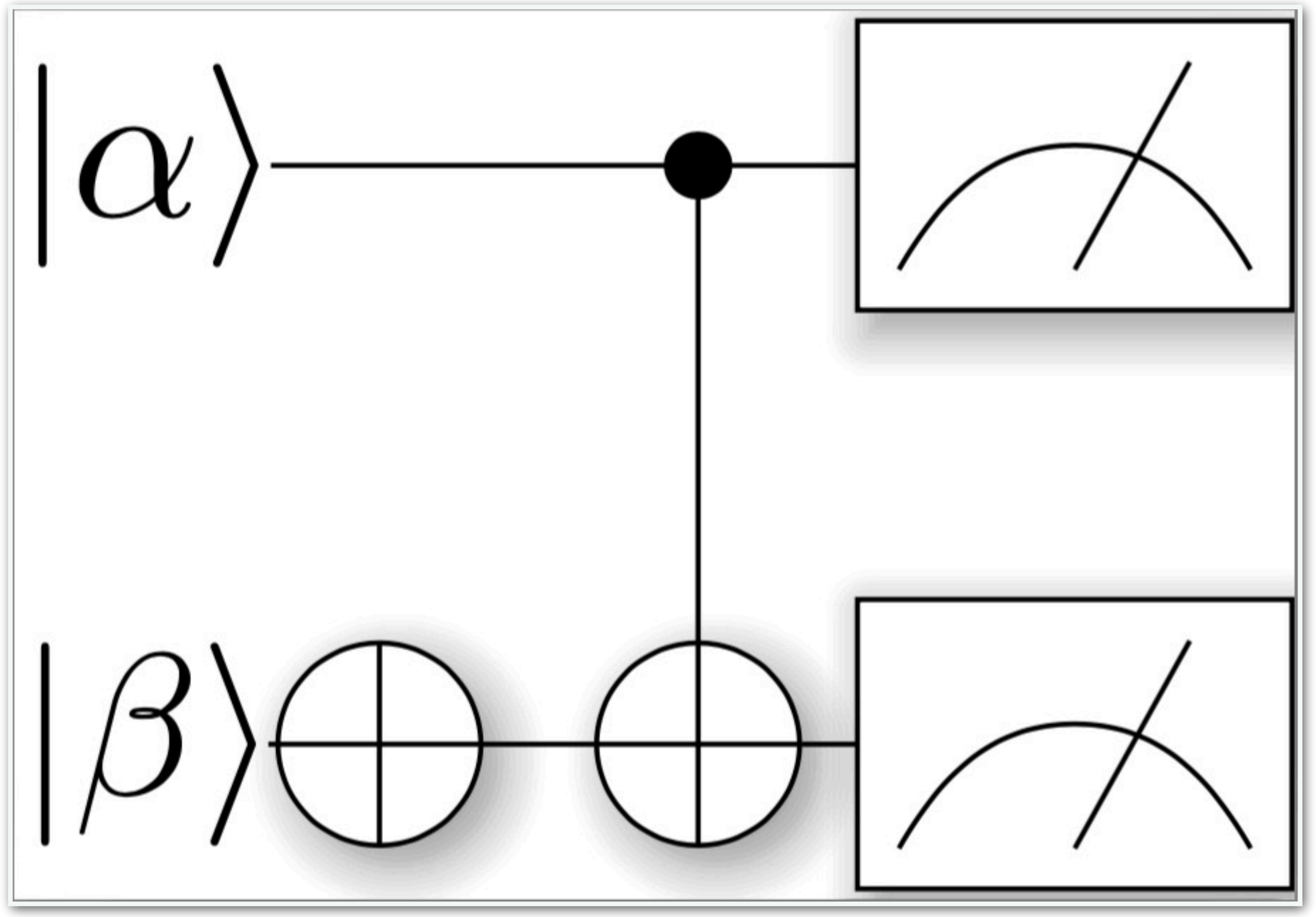
Quantum circuits

Apply a NOT gate to the <u>first</u> qubit, then a CNOT gate



$$\begin{aligned}|00\rangle &\rightarrow |11\rangle\\ |01\rangle &\rightarrow |10\rangle\\ |10\rangle &\rightarrow |00\rangle\\ |11\rangle &\rightarrow |01\rangle\end{aligned}$$

Apply a NOT gate to the <u>second</u> qubit, then a CNOT gate



$$\begin{aligned}|00\rangle &\rightarrow |01\rangle\\ |01\rangle &\rightarrow |00\rangle\\ |10\rangle &\rightarrow |10\rangle\\ |11\rangle &\rightarrow |11\rangle\end{aligned}$$

Quantum circuits

Apply a NOT gate to the <u>first</u> qubit, then a CNOT gate

$$1.\ X \otimes \mathbb{1}_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$2.\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_1\beta_0 \\ \alpha_1\beta_1 \\ \alpha_0\beta_0 \\ \alpha_0\beta_1 \end{pmatrix}$$

$$3.\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1\beta_0 \\ \alpha_1\beta_1 \\ \alpha_0\beta_0 \\ \alpha_0\beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_1\beta_0 \\ \alpha_1\beta_1 \\ \alpha_0\beta_1 \\ \alpha_0\beta_0 \end{pmatrix}$$

$$|00\rangle \rightarrow |11\rangle$$
$$|01\rangle \rightarrow |10\rangle$$
$$|10\rangle \rightarrow |00\rangle$$
$$|11\rangle \rightarrow |01\rangle$$

Apply a NOT gate to the <u>second</u> qubit, then a CNOT gate

$$1.\ \mathbb{1}_2 \otimes X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

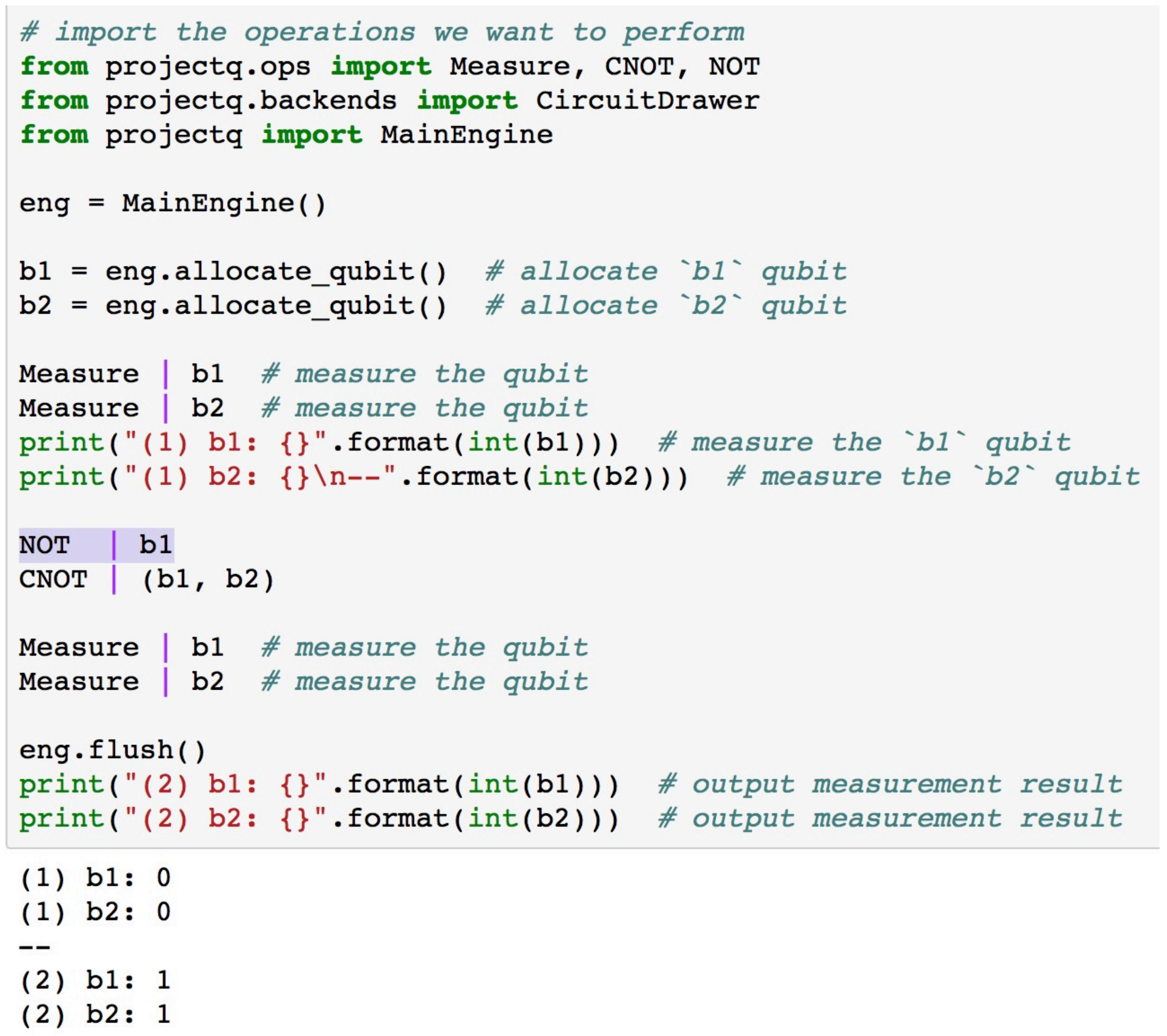
$$2.\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_1 \\ \alpha_0\beta_0 \\ \alpha_1\beta_1 \\ \alpha_1\beta_0 \end{pmatrix}$$

$$3.\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_0\beta_1 \\ \alpha_0\beta_0 \\ \alpha_1\beta_1 \\ \alpha_1\beta_0 \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_1 \\ \alpha_0\beta_0 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix}$$

$$|00\rangle \rightarrow |01\rangle$$
$$|01\rangle \rightarrow |00\rangle$$
$$|10\rangle \rightarrow |10\rangle$$
$$|11\rangle \rightarrow |11\rangle$$

Quantum circuits

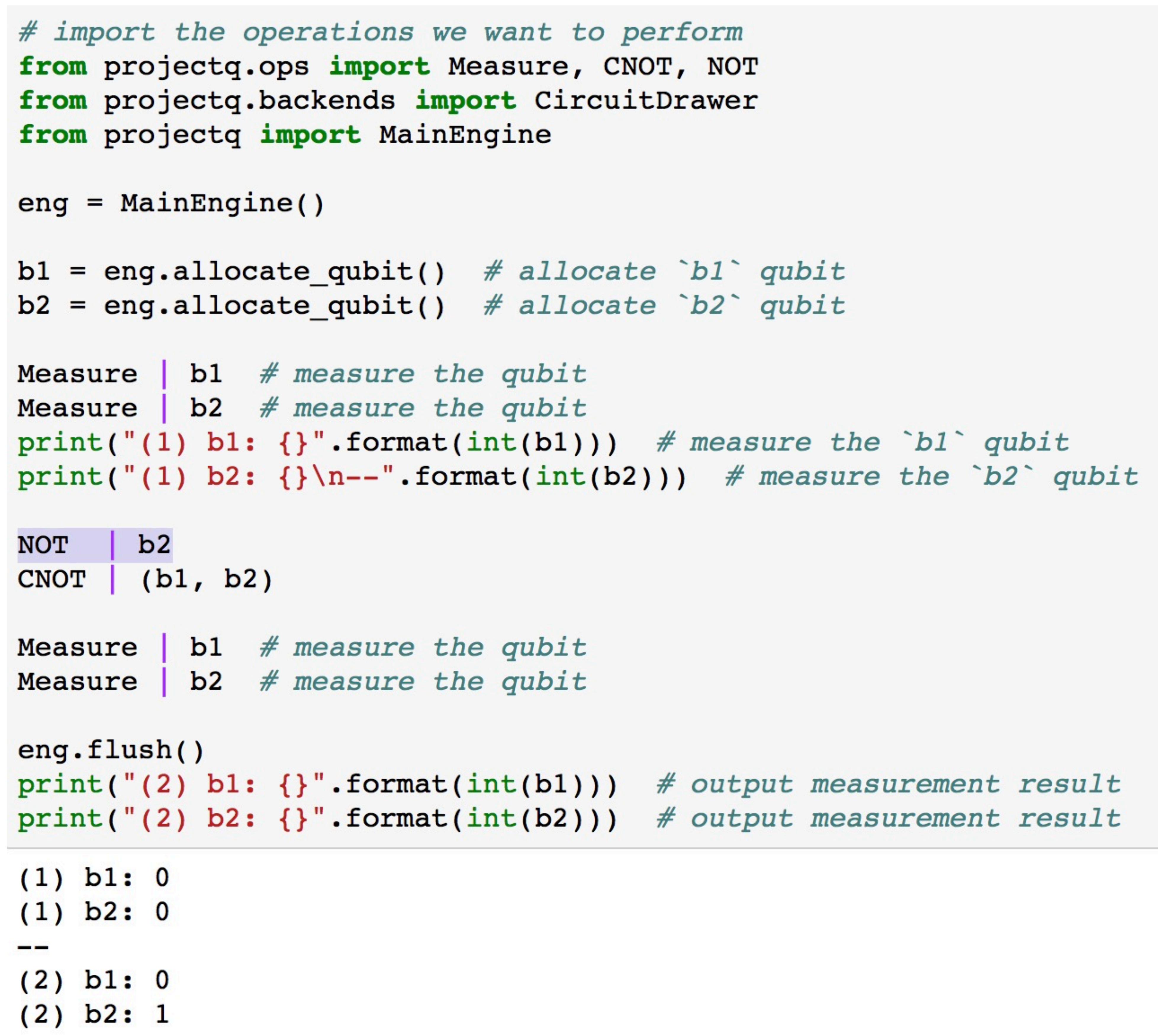
Apply a NOT gate to the <u>first</u> qubit, then a CNOT gate



```python
# import the operations we want to perform
from projectq.ops import Measure, CNOT, NOT
from projectq.backends import CircuitDrawer
from projectq import MainEngine

eng = MainEngine()

b1 = eng.allocate_qubit()  # allocate `b1` qubit
b2 = eng.allocate_qubit()  # allocate `b2` qubit

Measure | b1  # measure the qubit
Measure | b2  # measure the qubit
print("(1) b1: {}".format(int(b1)))  # measure the `b1` qubit
print("(1) b2: {}\n--".format(int(b2)))  # measure the `b2` qubit

NOT   | b1
CNOT | (b1, b2)

Measure | b1  # measure the qubit
Measure | b2  # measure the qubit

eng.flush()
print("(2) b1: {}".format(int(b1)))  # output measurement result
print("(2) b2: {}".format(int(b2)))  # output measurement result
```

```
(1) b1: 0
(1) b2: 0
--
(2) b1: 1
(2) b2: 1
```

Apply a NOT gate to the <u>second</u> qubit, then a CNOT gate

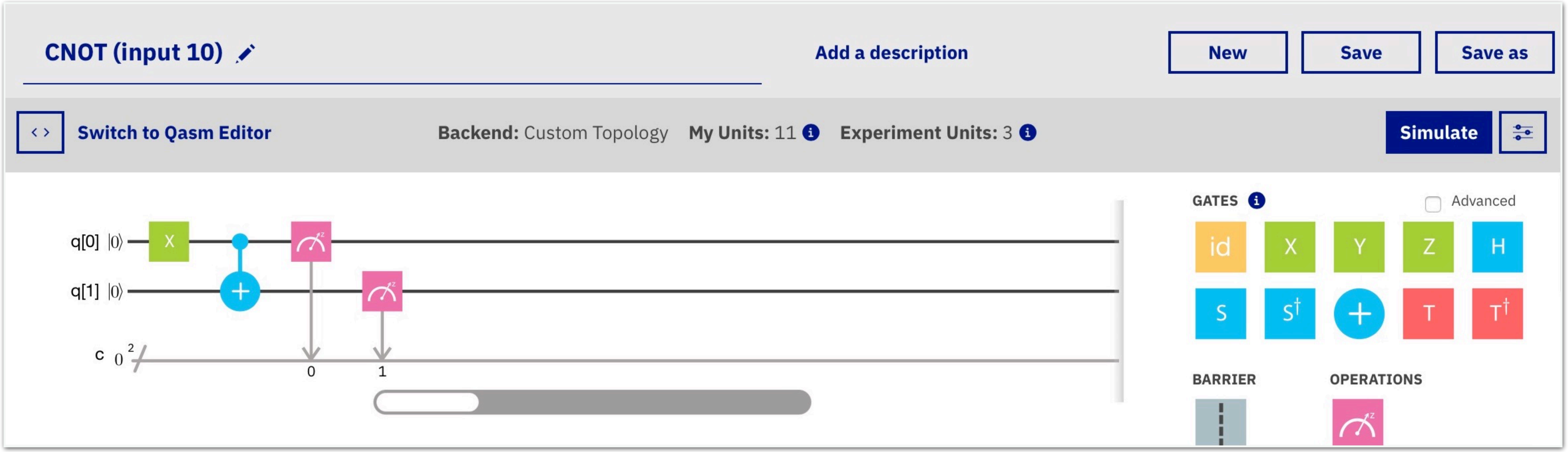


```python
# import the operations we want to perform
from projectq.ops import Measure, CNOT, NOT
from projectq.backends import CircuitDrawer
from projectq import MainEngine

eng = MainEngine()

b1 = eng.allocate_qubit()  # allocate `b1` qubit
b2 = eng.allocate_qubit()  # allocate `b2` qubit

Measure | b1  # measure the qubit
Measure | b2  # measure the qubit
print("(1) b1: {}".format(int(b1)))  # measure the `b1` qubit
print("(1) b2: {}\n--".format(int(b2)))  # measure the `b2` qubit

NOT   | b2
CNOT | (b1, b2)

Measure | b1  # measure the qubit
Measure | b2  # measure the qubit

eng.flush()
print("(2) b1: {}".format(int(b1)))  # output measurement result
print("(2) b2: {}".format(int(b2)))  # output measurement result
```

```
(1) b1: 0
(1) b2: 0
--
(2) b1: 0
(2) b2: 1
```

https://github.com/ProjectQ-Framework/ProjectQ

IBM Q experience

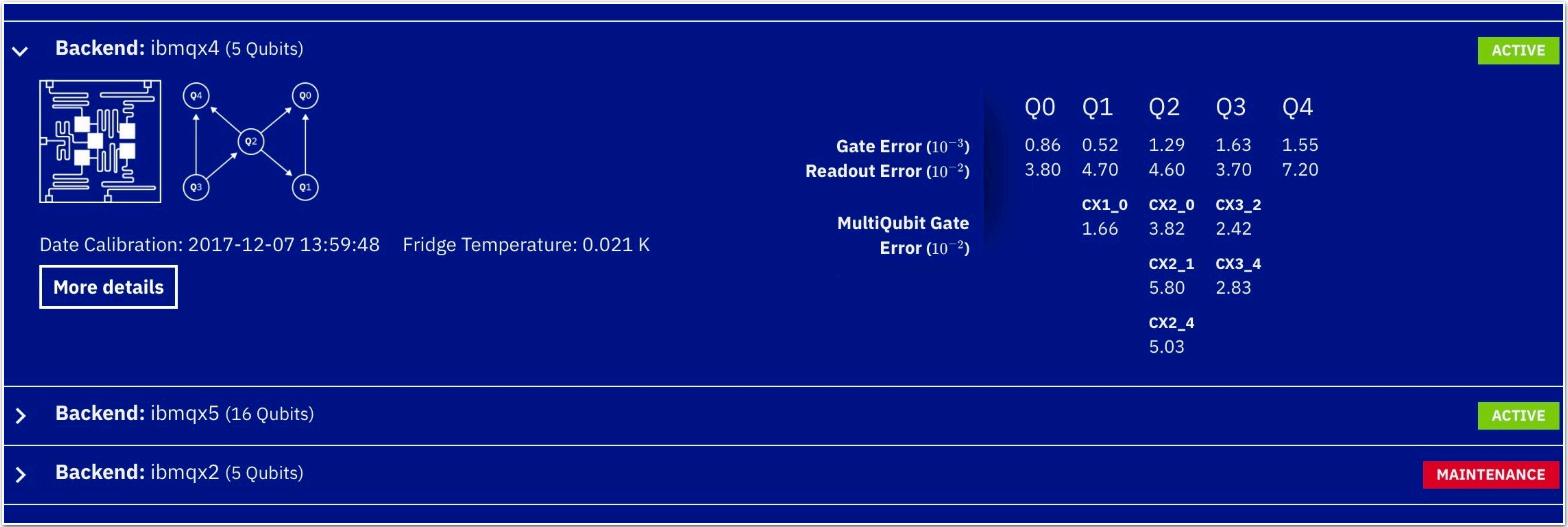

CNOT (input 10)

Add a description

Backend: Custom Topology My Units: 11 Experiment Units: 3

https://quantumexperience.ng.bluemix.net

IBM Q experience



Backend: ibmqx4 (5 Qubits) ACTIVE

Date Calibration: 2017-12-07 13:59:48 Fridge Temperature: 0.021 K

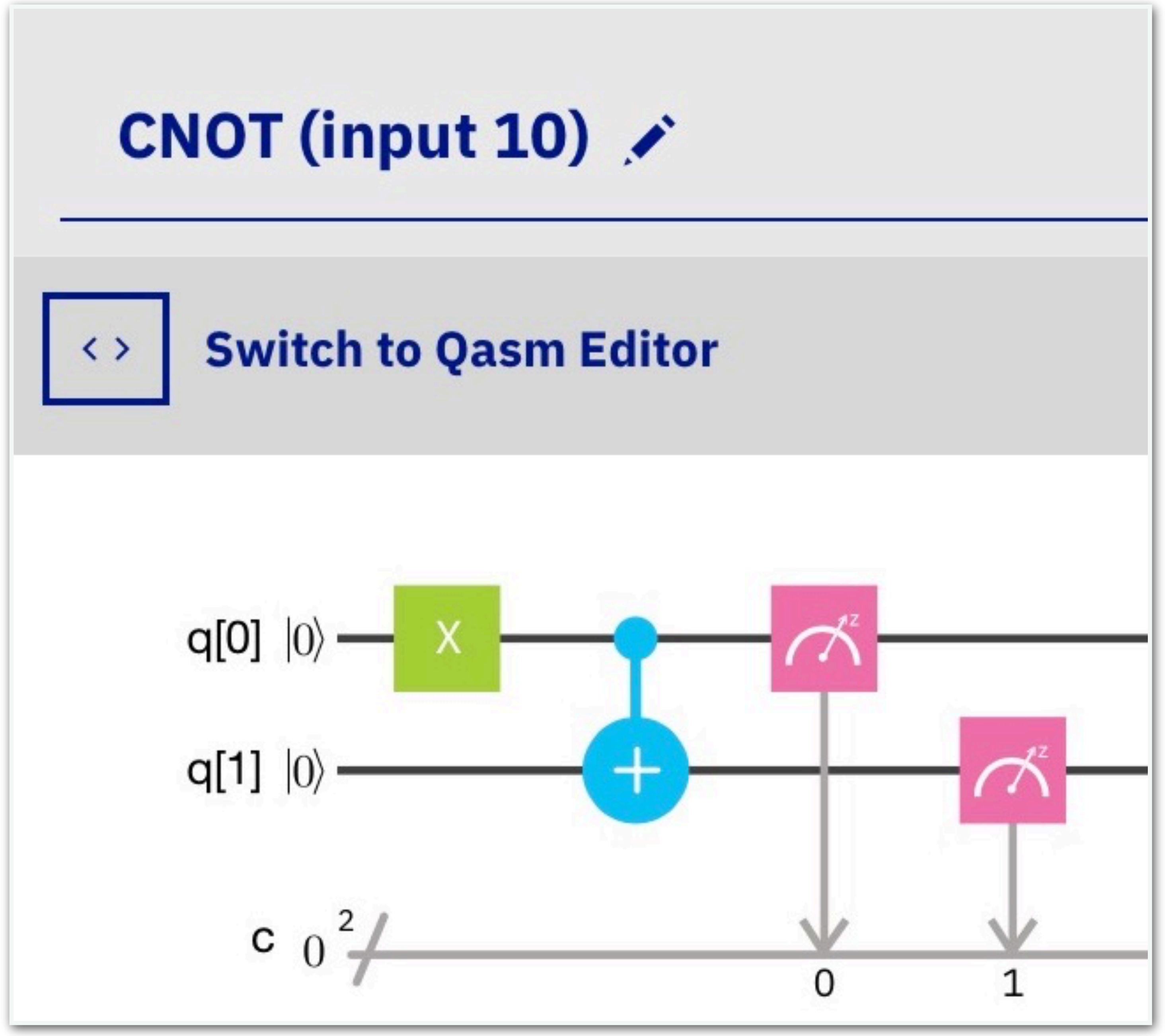
	Q0	Q1	Q2	Q3	Q4
Gate Error (10^{-3})	0.86	0.52	1.29	1.63	1.55
Readout Error (10^{-2})	3.80	4.70	4.60	3.70	7.20
MultiQubit Gate Error (10^{-2})		**CX1_0** 1.66	**CX2_0** 3.82 **CX2_1** 5.80 **CX2_4** 5.03	**CX3_2** 2.42 **CX3_4** 2.83	

Backend: ibmqx5 (16 Qubits) ACTIVE

Backend: ibmqx2 (5 Qubits) MAINTENANCE

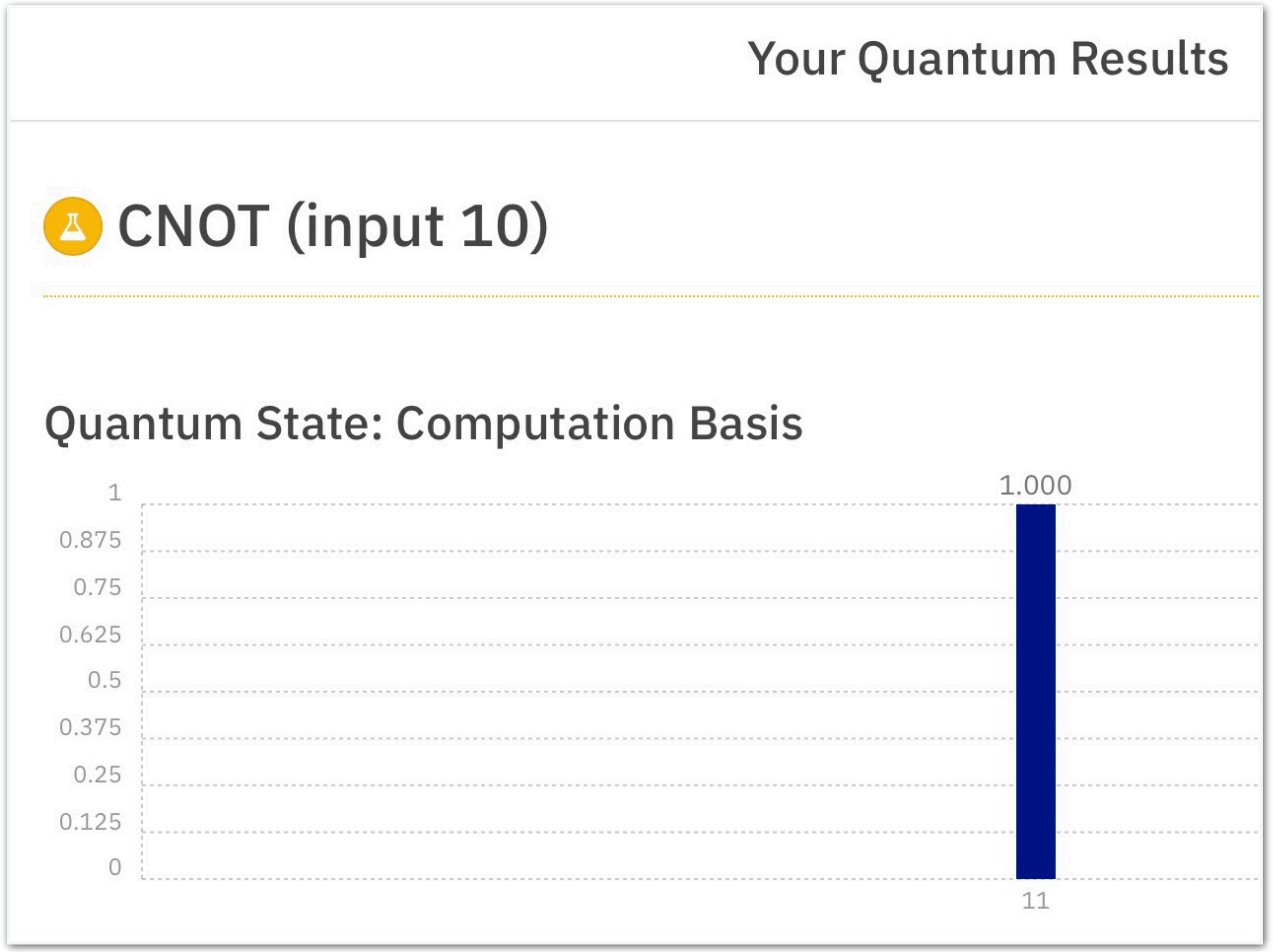
https://quantumexperience.ng.bluemix.net

IBM Q experience


CNOT (input 10)

Switch to Qasm Editor

Simulation


Your Quantum Results

CNOT (input 10)

Quantum State: Computation Basis

https://quantumexperience.ng.bluemix.net

Summary

- Applications of quantum computers
- Qubits and gates
- IBM Q experience

References

- Source for image on title slide: https://www.linkedin.com/pulse/quantum-computing-innovation-disrupt-silicon-valley-cami-rosso
- Source for image on other slides: https://hackernoon.com/google-plans-to-introduce-a-cloud-service-for-quantum-computing-74f47d808490
- Quantum circuits were drawn using ProjectQ framework for Python: https://github.com/ProjectQ-Framework/ProjectQ