Builtins and Primitives

Gate	Q#	ProjectQ	Cirq	Qiskit	PyQuil
I	I			iden	I
H	H	H	H	h	H
S	S	S	S	s	S
T	T	T	T	t	T
X, NOT	X	X	X	x	X
Y	Y	Y	Y	y	Y
Z	Z	Z	Z	z	Z
R_x	Rx	Rx	RotXGate	rx	Rx
R_y	Ry	Ry	RotYGate	ry	Ry
R_z	Rz	Rz	RotZGate	rz	Rz
R_ϕ	R1	R		u_1	PHASE
Measure	M	Measure	measure	measure	MEASURE
Barrier		Barrier		barrier	
CX, CNOT	CNOT	CNOT	CNOT	cx	CNOT
CCX, CCNOT, Toffoli	CCNOT	Toffoli	CCX, TOFFOLI	ccx	CCNOT
SWAP	SWAP	Swap	SwapGate	swap	SWAP
CZ	(Controlled Z)	CZ	CZ	cz	CZ
CSWAP, Fredkin	(Controlled SWAP)	C(Swap)	CSWAP, FREDKIN	cswap	CSWAP
CR_z	(Controlled Rz)	CRz		crz	CPHASE
ISWAP			ISWAP		ISWAP
QFT	QFT	QFT			
Other	HY, RAll0, RAll1	Sdag, Tdag, SqrtX, SqrtSwap, Entangle, TimeEvolution, QubitOperator, PhaseOracle, PermutationOracle, AddConstantModN	Rot11Gate, CCZ	cy, ch, rzz	PSWAP, CPHASE00, CPHASE01, CPHASE10

[1] Ryan LaRose. "Overview and Comparison of Gate Level Quantum Software Platforms". In: (July 6, 2018). arXiv: 1807.02500 [quant-ph]. URL: http://arxiv.org/abs/1807.02500 (visited on 09/12/2018).

Features

Operation	Q#	ProjectQ	Cirq	Qiskit	PyQuil
Gate from matrix		`G = BasicGate()` `G.matrix = numpy.matrix(`	`SingleQubitMatrixGate,` `TwoQubitMatrixGate`		`defgate`

Controlled	`(Controlled G)(c, q)`	`C(G) | q # or`
		`with Control(eng, c):`
		`G | q`
			`ControlledGate(G)`	`G.q_if(q)`	

Inverse	`(Adjoint G)(q)`	`with Dagger(eng):` `G | q`	`G.inverse()`	`G.inverse()`	`Program(G(q)).dagger()`
Apply to many qubits	`ApplyToEach(G, qs)`	`All(G) | qs # or` `Tensor(G) | qs`	`G.on_each(qs)`	`G(qs)`	
Simlators	local	local & cloud	local	local & cloud	cloud only
Execute on real quantum computer	no	IBM	Google	IBM	Rigetti
Rotation units	radians	radians	half-turns, radians, degrees	radians	radians
Integrations		Fermilib, OpenFermion	OpenFermion, QAsm	Qiskit-Aqua, QAsm	OpenFermion

Code examples

```
// Q##
// Circuit.qs
namespace MyProgram {
  open Microsoft.Quantum.Primitive;
  open Microsoft.Quantum.Canon;

  operation Circuit(q : Qubit) : (Result) {
    body {
      H(q);
      let result = M(q);
      return result;
    }
    // adjoint            auto;
    // controlled         auto;
    // controlled adjoint auto;
  }
}
```

```
// Q##
// Driver.cs
using Microsoft.Quantum.Simulation.Core;
using Microsoft.Quantum.Simulation.Simulators;

namespace MyProgram {
  class Driver {
    static void Main(string[] args) {
      using (var sim = new QuantumSimulator()) {
        var result = Circuit.run(sim, 1, Result.Zero).Result;
        System.Console.WriteLine($"Measured: {result}");
      }
    }
  }
}
```

```
## ProjectQ
from projectq      import MainEngine
from projectq.ops import *


eng = MainEngine()
q   = eng.allocate_qubit()


H       | q
Measure | q

eng.flush()


print("Measured: {}".format(
    int(q)
))
```

```
## Cirq
from cirq import *



q       = GridQubit(0, 0)
circuit = Circuit()


circuit.append([H(q)])
circuit.append([measure(q, key = "c")])

sim    = google.XmonSimulator()
result = sim.run(circuit)

print("Measured: {}".format(
    int(result.measurements["c"][0,0])
))
```

```
## Qiskit
from qiskit import *



q       = QuantumRegister(1)
c       = ClassicalRegister(1)
circuit = QuantumCircuit(q, c)

circuit.h(q)
circuit.measure(q, c)

sim    = execute(circuit, "local_qasm_simulator", shots = 1)
result = sim.result()

print("Measured: {}".format(
    list(result.get_counts())[0]
))
```

```
## PyQuil
from pyquil.quil  import Program
from pyquil.gates import *
from pyquil.api   import QVMConnection

program = Program()



program.inst(H(0))
program.inst(MEASURE(0, 0))

qvm    = QVMConnection()
result = qvm.run(program, [0])

print(result)
```