VQE_Screening

September 18, 2020

1 VQE Screening

In [1]:
```
scaffold_codeXX = """
const double alpha0 = 3.14159265359;

module initialRotations(qbit reg[2]) {
  Rx(reg[0], alpha0);
  CNOT(reg[0], reg[1]);
  H(reg[0]);

}

module entangler(qbit reg[2]) {
  H(reg[0]);
  CNOT(reg[0], reg[1]);

  H(reg[1]);
  CNOT(reg[1], reg[0]);
}


module prepareAnsatz(qbit reg[2]) {
  initialRotations(reg);
  entangler(reg);
}

module measure(qbit reg[2], cbit result[2]) {
  result[0] = MeasX(reg[0]);
  result[1] = MeasX(reg[1]);
}

int main() {
  qbit reg[2];
  cbit result[2];

  prepareAnsatz(reg);
  measure(reg, result);
```

```
  return 0;
}


"""
```

In [2]:
```
scaffold_codeYY = """
const double alpha0 = 3.14159265359;

module initialRotations(qbit reg[2]) {
  Rx(reg[0], alpha0);
  CNOT(reg[0], reg[1]);
  H(reg[0]);

}

module entangler(qbit reg[2]) {
  H(reg[0]);
  CNOT(reg[0], reg[1]);

  H(reg[1]);
  CNOT(reg[1], reg[0]);
}


module prepareAnsatz(qbit reg[2]) {
  initialRotations(reg);
  entangler(reg);
}

module measure(qbit reg[2], cbit result[2]) {
  Rx(reg[0], 1.57079632679);
  result[0] = MeasZ(reg[0]);
  Rx(reg[1], 1.57079632679);
  result[1] = MeasZ(reg[1]);
}

int main() {
  qbit reg[2];
  cbit result[2];

  prepareAnsatz(reg);
  measure(reg, result);

  return 0;
}
```

```
"""
```

```
In [3]: scaffold_codeZZ = """
        const double alpha0 = 3.14159265359;

        module initialRotations(qbit reg[2]) {
          Rx(reg[0], alpha0);
          CNOT(reg[0], reg[1]);
          H(reg[0]);

        }

        module entangler(qbit reg[2]) {
          H(reg[0]);
          CNOT(reg[0], reg[1]);

          H(reg[1]);
          CNOT(reg[1], reg[0]);
        }


        module prepareAnsatz(qbit reg[2]) {
          initialRotations(reg);
          entangler(reg);
        }

        module measure(qbit reg[2], cbit result[2]) {
          result[0] = MeasZ(reg[0]);
          result[1] = MeasZ(reg[1]);
        }

        int main() {
          qbit reg[2];
          cbit result[2];

          prepareAnsatz(reg);
          measure(reg, result);

          return 0;
        }


        """
```

2 Executing it

```
In [4]: # Compile the Scaffold to OpenQASM
        from scaffcc_interface import ScaffCC
        openqasmXX = ScaffCC(scaffold_codeXX).get_openqasm()
        openqasmYY = ScaffCC(scaffold_codeYY).get_openqasm()
        openqasmZZ = ScaffCC(scaffold_codeZZ).get_openqasm()
        print(openqasmXX)
        print(openqasmYY)
        print(openqasmZZ)
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg reg[2];
creg result[2];
rx(3.141593e+00) reg[0];
cx reg[0],reg[1];
h reg[0];
h reg[0];
cx reg[0],reg[1];
h reg[1];
cx reg[1],reg[0];
h reg[0];
measure reg[0] -> result[0];
h reg[1];
measure reg[1] -> result[1];


OPENQASM 2.0;
include "qelib1.inc";
qreg reg[2];
creg result[2];
rx(3.141593e+00) reg[0];
cx reg[0],reg[1];
h reg[0];
h reg[0];
cx reg[0],reg[1];
h reg[1];
cx reg[1],reg[0];
rx(1.570796e+00) reg[0];
measure reg[0] -> result[0];
rx(1.570796e+00) reg[1];
measure reg[1] -> result[1];


OPENQASM 2.0;
include "qelib1.inc";
qreg reg[2];
```

```
creg result[2];
rx(3.141593e+00) reg[0];
cx reg[0],reg[1];
h reg[0];
h reg[0];
cx reg[0],reg[1];
h reg[1];
cx reg[1],reg[0];
measure reg[0] -> result[0];
measure reg[1] -> result[1];
```

2.0.1 Execute on a Simulator

In [5]:
```python
from qiskit import Aer,QuantumCircuit, execute
Aer.backends()
```

Out[5]:
```
[<QasmSimulator('qasm_simulator') from AerProvider()>,
 <StatevectorSimulator('statevector_simulator') from AerProvider()>,
 <UnitarySimulator('unitary_simulator') from AerProvider()>]
```

In [6]:
```python
simulator = Aer.get_backend('qasm_simulator')
vqe_circXX = QuantumCircuit.from_qasm_str(openqasmXX)
vqe_circYY = QuantumCircuit.from_qasm_str(openqasmYY)
vqe_circZZ = QuantumCircuit.from_qasm_str(openqasmZZ)
num_shots = 100000
sim_resultXX = execute(vqe_circXX, simulator, shots=num_shots).result()
sim_resultYY = execute(vqe_circYY, simulator, shots=num_shots).result()
sim_resultZZ = execute(vqe_circZZ, simulator, shots=num_shots).result()

countsXX = sim_resultXX.get_counts()
countsYY = sim_resultYY.get_counts()
countsZZ = sim_resultZZ.get_counts()

expected_valueXX = (countsXX.get('00', 0) - countsXX.get('01', 0) - countsXX.get('10', 0
expected_valueYY = (countsYY.get('00', 0) - countsYY.get('01', 0) - countsYY.get('10', 0
expected_valueZZ = (countsZZ.get('00', 0) - countsZZ.get('01', 0) - countsZZ.get('10', 0

expected_value = 0.5 - 0.5 * expected_valueXX - 0.5 * expected_valueYY + 0.5 * expected_
print('The lowest eigenvalue is the expected value, which is : %s' % expected_value)

#print(expected_valueXX)
#print(expected_valueYY)
#print(expected_valueZZ)
```

```
The lowest eigenvalue is the expected value, which is : -1.0
```

3 Circuit Visualisation

In [7]:
```
from qiskit.tools.visualization import circuit_drawer
circuit_drawer(vqe_circXX, scale=.4)
```

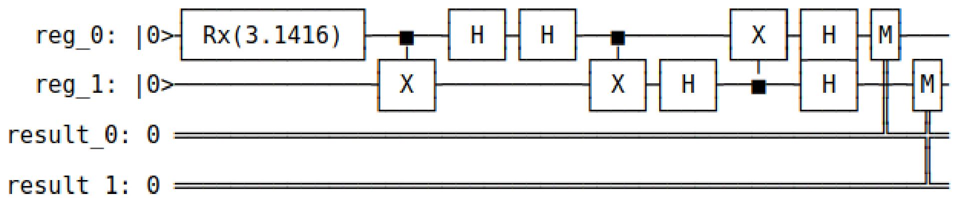
Out[7]: <qiskit.visualization.text.TextDrawing at 0x7f7bdc3a6470>

In [8]:
```
from qiskit.tools.visualization import circuit_drawer
circuit_drawer(vqe_circYY, scale=.4)
```

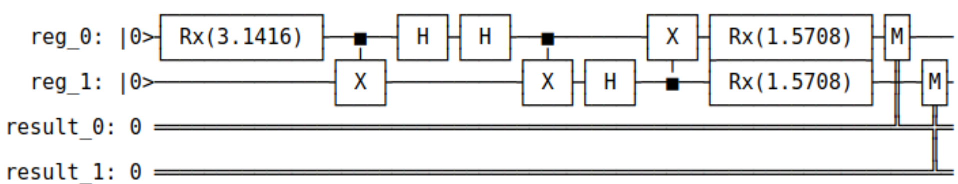
Out[8]: <qiskit.visualization.text.TextDrawing at 0x7f7b853be668>

In [9]:
```
from qiskit.tools.visualization import circuit_drawer
circuit_drawer(vqe_circZZ, scale=.4)
```

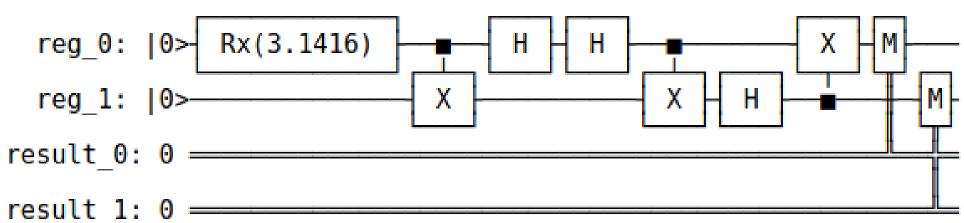
Out[9]: <qiskit.visualization.text.TextDrawing at 0x7f7b853beb00>



vqe_circXX



vqe_circYY



vqe_circZZ