Quantum Random Number Generator using Qiskit

Qiskit Fall Fest 2025 - Nepal PNC

Problem Statement:

Quantum Random Number Generator (QRNG) – Leveraging quantum mechanics for truly random outcomes, built with Qiskit.

Understanding Quantum Randomness:

In contrast, quantum mechanics introduces intrinsic unpredictability. When we measure a qubit in a superposition state, the result is truly random - governed by the Born rule, not by math tricks. That's the foundation for our QRNG.

Step-by-step approaches to deal with the QRNG:

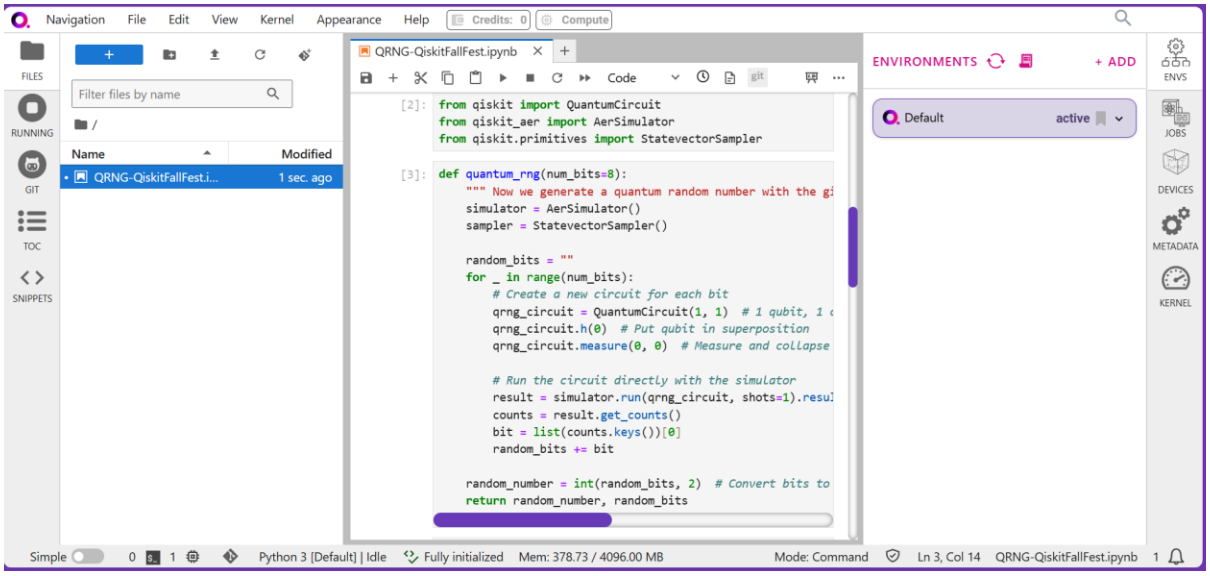
- Start with one qubit
- Apply a Hadamard gate (H) to put it into superposition
- Measure the qubit to collapse it into either 0 or 1, randomly
- Repeat this process for multiple bits

By repeating this simple 1-qubit experiment multiple times, we can build up a fully random binary number.

Sample Code & Key Terminologies:

- **QuantumCircuit:** Holds all our quantum operations and the instructions for the system.
- **Measure ([0,1], [0,1]):** The first argument indexes the qubits, the second argument indexes the classical bits. The *n*th qubit's measurement result will be stored in the *n*th classical bit.

- **Qiskit Aer:** a high-performance simulator that provides several backends to achieve different simulation goals. Here, we will use the qasm_simulator.



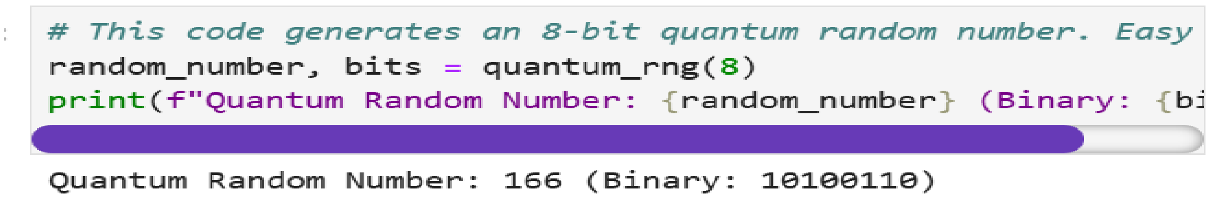
```
[2]: from qiskit import QuantumCircuit
     from qiskit_aer import AerSimulator
     from qiskit.primitives import StatevectorSampler
```

```
[3]: def quantum_rng(num_bits=8):
         """ Now we generate a quantum random number with the gi
         simulator = AerSimulator()
         sampler = StatevectorSampler()

         random_bits = ""
         for _ in range(num_bits):
             # Create a new circuit for each bit
             qrng_circuit = QuantumCircuit(1, 1)  # 1 qubit, 1 c
             qrng_circuit.h(0)  # Put qubit in superposition
             qrng_circuit.measure(0, 0)  # Measure and collapse

             # Run the circuit directly with the simulator
             result = simulator.run(qrng_circuit, shots=1).resul
             counts = result.get_counts()
             bit = list(counts.keys())[0]
             random_bits += bit

         random_number = int(random_bits, 2)  # Convert bits to
         return random_number, random_bits
```



```
# This code generates an 8-bit quantum random number. Easy
random_number, bits = quantum_rng(8)
print(f"Quantum Random Number: {random_number} (Binary: {bi
```

```
Quantum Random Number: 166 (Binary: 10100110)
```

Practical Considerations & Limitations:

Considering some points while implementing QRNG as follows:

- **Access**: True QRNG requires real quantum hardware. Qiskit gives access, but devices are shared and usage is queued.
- **Speed**: Classical PRNGs are much faster. QRNGs aren't ideal for high-frequency randomness needs (yet).
- **Bias & Noise**: Real quantum measurements may have slight biases. Post-processing (e.g., von Neumann debiasing) helps.
- **Integration**: We will still need secure methods to transport or store generated bits in live systems.

We can experiment more broadly, including the error mitigation techniques to enhance the fairness and accuracy of QRNG.