Quantum Random Generation

Report

INSTRUCTOR: Dr. V Narayanan

Name: **SUMIT KUMAR**

Roll Number: **(M23IQT007)**

Program: **MTECH(QUANTUM TECHNOLOGIES)**

Chapter 1

Introduction

1.1 Objective

The objective of this project is to generate quantum random numbers (QRNG) using two different methods—SPAD with a Swabian Time Tagger, and IBM's Qiskit platform (simulator and hardware). The generated random numbers were analyzed using statistical tests (NIST and Dieharder) to assess their quality and randomness, a critical requirement for cryptographic and other high-security applications.

1.2 Generating QRNG Data Using SPAD and Time Tagger

Quantum random data was generated using a Single-Photon Avalanche Diode (SPAD) combined with a Swabian Time Tagger. This setup captured photon arrival times, which were then converted into binary sequences. Two datasets were created from this setup:

- **1 lakh data points**: Captured and saved as a binary sequence for analysis.
- **2 lakh data points**: Collected to provide a larger dataset for improved statistical accuracy.

These binary sequences were saved in respective files for further statistical analysis.

1.3 Additional QRNG Data Using IBM Qiskit Simulator and Hardware

In addition to the SPAD-generated data, quantum random numbers were generated using IBM's Qiskit platform:

- **IBM Simulator**: A quantum circuit was implemented on Qiskit's quantum simulator to generate a sequence of random binary data.
- **IBM Hardware**: IBM's actual quantum hardware was also used to produce another sequence of quantum random numbers, offering a comparison with the simulator.

These additional datasets were saved in files and evaluated using the Dieharder test suite for randomness quality.

1.4 Statistical Testing Using NIST and Diehard Tests

The generated QRNG datasets were subjected to two main statistical test suites:

1.4.1 NIST Statistical Tests on SPAD Data

NIST Statistical Tests were applied to the binary data generated by the SPAD setup:

1.4.2 Diehard Tests on IBM Qiskit Data

Diehard tests were also run on the datasets generated by IBM's Qiskit platform (simulator and hardware) to evaluate their randomness.

- The IBM-generated sequences passed most tests, although some tests, such as the OPSO, OQSO, and GCD tests, displayed weaker results or lower p-values.

1.5 Testing Data Summary

The following summarizes the results of the statistical testing for each dataset:

- **SPAD-Generated Data**: NIST tests indicated primarily non-random behavior, especially for the 1 lakh dataset. The 2 lakh dataset showed improved results in the Diehard tests, with several tests passing and indicating better randomness.
- **IBM Qiskit Data (Simulator and Hardware)**: Both datasets performed reasonably well in the Diehard tests, with most tests passed. However, certain tests indicated areas for improvement in randomness, particularly with some lower p-values and weaker results in specific tests.

1.6 Significance of NIST and Diehard Tests

The NIST and Diehard tests are industry-standard tools for evaluating randomness in binary sequences:

- **NIST Tests**: Widely used in cryptography and security applications, NIST tests assess various aspects of randomness, including frequency, complexity, and entropy.
- **Diehard Tests**: Dieharder provides a complementary assessment of randomness quality, covering a different set of tests that evaluate sequences for patterns and statistical anomalies.

The combination of these test suites provides a robust evaluation of the QRNG data's quality and ensures that the generated sequences meet high standards for randomness.

Chapter 2

Using Qiskit Simulator

2.1 Objective

The objective of this project is to generate quantum random numbers (QRNG) using IBM's Qiskit Simulator and evaluate the randomness of the generated sequences. Standard randomness tests from the NIST and Dieharder test suites were applied to assess the quality of the generated data.

2.2 Qiskit Simulator

2.2.1 Part 1: Simulation

In this part, random binary sequences were generated using the Qiskit Aer simulator, which uses quantum circuits to produce quantum-random sequences. The implementation for generating these sequences is available in the Jupyter Notebook.

```
# %%
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

# %%
from qiskit_aer import AerSimulator
import numpy as np

# %%
# Quantum Random Bit Generator
def generate_random_bits(qubits=3, shots=1000, simulator=None, circ=
    ↪ None):
    if simulator is None:
        simulator = AerSimulator()

    if circ is None:
        # Create quantum and classical registers
        q = QuantumRegister(qubits, 'q')
        c = ClassicalRegister(qubits, 'c')
        circ = QuantumCircuit(q, c)

        # Apply Hadamard gate to each qubit to create superposition
        for i in range(qubits):
```

```
            circ.h(q[i])

        # Measure the qubits
        circ.measure(q, c)

    # Use AerSimulator to run the circuit
    job = simulator.run(circ, shots=shots)
    result = job.result()
    counts = result.get_counts(circ)

    # Convert the counts into a list of bits
    bits = []
    for bitstring, count in counts.items():
        bits.extend([bit for _ in range(count) for bit in bitstring])

    return bits

# %%
# Generate random bits using 3 qubits and 100,000 shots
simulator = AerSimulator()
q = QuantumRegister(3, 'q')
c = ClassicalRegister(3, 'c')
circ = QuantumCircuit(q, c)
for i in range(3):
    circ.h(q[i])
circ.measure(q, c)

# Collect 100,000 binary bits
total_bits = []
while len(total_bits) < 100000:
    total_bits.extend(generate_random_bits(3, shots=1000, simulator=
        ↪ simulator, circ=circ))

# %%
# Truncate to exactly 100,000 bits
total_bits = total_bits[:100000]

# Print the first 100 bits for validation
print(f"First 1000000 bits: {''.join(total_bits[:100000])}")

# Save the bits to a text file
with open('quantum_random_bits.txt', 'w') as f:
    f.write(''.join(total_bits))

print(f"Generated 100,000 random bits and saved to '
    ↪ quantum_random_bits.txt'.")

# %%
```

2.3 Quantum Random Sequence

The binary sequence generated by the Qiskit simulator is saved in a text file for further analysis. This file, `quantum_random_bits.txt`, contains a sequence of 0s and 1s representing the quantum random numbers.

(a) Overview of Quantum Random Sequence

This section discusses the properties of the quantum random sequence generated by the simulator. The sequence was extracted and stored in for subsequent testing.

(b) Statistical Tests Setup

Two test suites were applied to the quantum random sequence:

- NIST Statistical Test Suite: Used for testing the randomness of the sequence.
- Diehard Test Suite: An additional set of tests that complements the NIST suite, providing a comprehensive evaluation.

2.4 Results

The results of the randomness tests provide insights into the quality of the quantum random sequence. These results are presented below.

NIST Test Results

The NIST test suite was used to assess the randomness of the generated sequence. The results are saved in the file `Result_NIST.txt` and are summarized here.

```
Test Data File:C:/Users/sumit/Downloads/IIT J Sem Notes/Sem 3/Intro
   ↪ to Quantum tech/Project QRNG/Using_qiskit/Qiskit_Binary_Sim.txt


Type of Test                                              P-Value
   ↪                  Conclusion
01. Frequency (Monobit) Test
   ↪ 0.5692136494737111      Random
02. Frequency Test within a Block                         0.0
   ↪                      Non-Random
03. Runs Test
   ↪ 0.24495230982170202     Random
04. Test for the Longest Run of Ones in a Block           0.0
   ↪                      Non-Random
05. Binary Matrix Rank Test
   ↪ 2.7933048058255856e-137 Non-Random
06. Discrete Fourier Transform (Spectral) Test            0.0
   ↪                      Non-Random
07. Non-overlapping Template Matching Test                7.260532764801739
   ↪ e-39    Non-Random
08. Overlapping Template Matching Test
   ↪ 1.6675149109769573e-29  Non-Random
```

```
09. Maurer's "Universal Statistical" Test            -1.0                 Non-Random
10. Linear Complexity Test                                                0.0                 Non-Random
11. Serial Test:
                                                                          0.0                 Non-Random
                                                                          0.0                 Non-Random
12. Approximate Entropy Test                                              0.0                 Non-Random
13. Cumulative Sums Test (Forward)                                        3.305007351469549
    e-73    Non-Random
14. Cumulative Sums Test (Backward)                                       8.454714082700325
    e-69    Non-Random
15. Random Excursions Test:
                State       Chi Squared             P-Value             Conclusion
                -4          1.8366385800788132      0.8712557845329562  Random
                -3          1.9974153846153846      0.849502551492505   Random
                -2          2.3770180436847106      0.79489159786277    Random
                -1          1.6153846153846154      0.899382654403362   Random
                +1          3.3076923076923075      0.652664201251164   Random
                +2          3.1519468186134856      0.676573272311288   Random
                +3          2.449353846153846       0.7841024129538753  Random
                +4          4.313555249415308       0.5052095486858288  Random
16. Random Excursions Variant Test:
                State       COUNTS                  P-Value             Conclusion
                -9.0        13                      1.0                 Random
                -8.0        14                      0.9596148041121082  Random
                -7.0        13                      1.0                 Random
                -6.0        13                      1.0                 Random
                -5.0        14                      0.9478777812782534  Random
                -4.0        13                      1.0                 Random
                -3.0        13                      1.0                 Random
                -2.0        14                      0.9098500327472846  Random
                -1.0        13                      1.0
```

```
    ↪ ␣␣␣␣␣␣␣Random
42 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+1.0␣␣␣␣␣␣␣␣␣11␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.6948866023724733␣␣␣␣␣␣␣Random
43 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+2.0␣␣␣␣␣␣␣␣␣8␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.5712996214994486␣␣␣␣␣␣␣Random
44 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+3.0␣␣␣␣␣␣␣␣␣8␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.6610028456239788␣␣␣␣␣␣␣Random
45 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+4.0␣␣␣␣␣␣␣␣␣9␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.7668484756777447␣␣␣␣␣␣␣Random
46 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+5.0␣␣␣␣␣␣␣␣␣8␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.7437736057682318␣␣␣␣␣␣␣Random
47 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+6.0␣␣␣␣␣␣␣␣␣8␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.7674926513236701␣␣␣␣␣␣␣Random
48 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+7.0␣␣␣␣␣␣␣␣␣9␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.8277631346049141␣␣␣␣␣␣␣Random
49 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+8.0␣␣␣␣␣␣␣␣␣8␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.8001253795101059␣␣␣␣␣␣␣Random
50 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+9.0␣␣␣␣␣␣␣␣␣8␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ 0.8120162482472952␣␣␣␣␣␣␣Random
```

Diehard Test Results

The Diehard test suite was also used to evaluate the quantum random sequence. The results of these tests are stored in `Result_Diehard.txt`.

```
1  inux_prash@prashant:~$ cd download
2  bash: cd: download: No such file or directory
3  linux_prash@prashant:~$ cd Downloads
4  linux_prash@prashant:~/Downloads$ ls
5  binary_output2.txt  output.txt              Sumit_dieHard.txt
6  binary_output.txt   Qiskit_Binary_Sim.txt
7  linux_prash@prashant:~/Downloads$ dieharder -a -f Qiskit_Binary_Sim.
     ↪ txt
8  #
     ↪ ==========================================================================
     ↪
9  #            dieharder version 3.31.1 Copyright 2003 Robert G. Brown
     ↪           #
10 #
     ↪ ==========================================================================
     ↪
11    rng_name    |           filename             |rands/second|
12         mt19937|           Qiskit_Binary_Sim.txt|  7.71e+06  |
13 #
     ↪ ==========================================================================
     ↪
14         test_name   |ntup| tsamples |psamples|  p-value |Assessment
15 #
     ↪ ==========================================================================
     ↪
16    diehard_birthdays|   0|       100|     100|0.10484325|  PASSED
17       diehard_operm5|   0|   1000000|     100|0.22876681|  PASSED
18   diehard_rank_32x32|   0|     40000|     100|0.18953203|  PASSED
19     diehard_rank_6x8|   0|    100000|     100|0.92912225|  PASSED
```

```
   diehard_bitstream|   0|   2097152|      100|0.54970658|  PASSED
         diehard_opso|   0|   2097152|      100|0.86698710|  PASSED
         diehard_oqso|   0|   2097152|      100|0.04256341|  PASSED
          diehard_dna|   0|   2097152|      100|0.11220360|  PASSED
diehard_count_1s_str|   0|    256000|      100|0.00146117|   WEAK
diehard_count_1s_byt|   0|    256000|      100|0.21898741|  PASSED
 diehard_parking_lot|   0|     12000|      100|0.59549008|  PASSED
    diehard_2dsphere|   2|      8000|      100|0.53931710|  PASSED
    diehard_3dsphere|   3|      4000|      100|0.67642627|  PASSED
     diehard_squeeze|   0|    100000|      100|0.02890493|  PASSED
        diehard_sums|   0|       100|      100|0.16308531|  PASSED
        diehard_runs|   0|    100000|      100|0.52380508|  PASSED
        diehard_runs|   0|    100000|      100|0.81199862|  PASSED
       diehard_craps|   0|    200000|      100|0.41388261|  PASSED
       diehard_craps|   0|    200000|      100|0.06010301|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|      100|0.29305999|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|      100|0.86418745|  PASSED
         sts_monobit|   1|    100000|      100|0.28578925|  PASSED
            sts_runs|   2|    100000|      100|0.37914167|  PASSED
          sts_serial|   1|    100000|      100|0.52771601|  PASSED
          sts_serial|   2|    100000|      100|0.02172778|  PASSED
          sts_serial|   3|    100000|      100|0.12969765|  PASSED
          sts_serial|   3|    100000|      100|0.63440071|  PASSED
          sts_serial|   4|    100000|      100|0.52668330|  PASSED
          sts_serial|   4|    100000|      100|0.69780652|  PASSED
          sts_serial|   5|    100000|      100|0.73861234|  PASSED
          sts_serial|   5|    100000|      100|0.70698165|  PASSED
          sts_serial|   6|    100000|      100|0.67032709|  PASSED
          sts_serial|   6|    100000|      100|0.41239986|  PASSED
          sts_serial|   7|    100000|      100|0.55316572|  PASSED
          sts_serial|   7|    100000|      100|0.04138010|  PASSED
          sts_serial|   8|    100000|      100|0.18030199|  PASSED
          sts_serial|   8|    100000|      100|0.53006027|  PASSED
          sts_serial|   9|    100000|      100|0.35960165|  PASSED
          sts_serial|   9|    100000|      100|0.40334384|  PASSED
          sts_serial|  10|    100000|      100|0.46268798|  PASSED
          sts_serial|  10|    100000|      100|0.14892231|  PASSED
          sts_serial|  11|    100000|      100|0.33449684|  PASSED
          sts_serial|  11|    100000|      100|0.38345428|  PASSED
          sts_serial|  12|    100000|      100|0.98804991|  PASSED
          sts_serial|  12|    100000|      100|0.12276102|  PASSED
          sts_serial|  13|    100000|      100|0.62083918|  PASSED
          sts_serial|  13|    100000|      100|0.25303871|  PASSED
          sts_serial|  14|    100000|      100|0.58032479|  PASSED
          sts_serial|  14|    100000|      100|0.89595276|  PASSED
          sts_serial|  15|    100000|      100|0.55882203|  PASSED
          sts_serial|  15|    100000|      100|0.98144066|  PASSED
          sts_serial|  16|    100000|      100|0.01967165|  PASSED
          sts_serial|  16|    100000|      100|0.10795622|  PASSED
         rgb_bitdist|   1|    100000|      100|0.21152201|  PASSED
         rgb_bitdist|   2|    100000|      100|0.42391326|  PASSED
         rgb_bitdist|   3|    100000|      100|0.32925949|  PASSED
         rgb_bitdist|   4|    100000|      100|0.70380427|  PASSED
         rgb_bitdist|   5|    100000|      100|0.11472007|  PASSED
```

```
          rgb_bitdist|   6|    100000|     100|0.89563731|  PASSED
          rgb_bitdist|   7|    100000|     100|0.15670703|  PASSED
          rgb_bitdist|   8|    100000|     100|0.68674883|  PASSED
          rgb_bitdist|   9|    100000|     100|0.66855814|  PASSED
          rgb_bitdist|  10|    100000|     100|0.91528520|  PASSED
          rgb_bitdist|  11|    100000|     100|0.00232367|   WEAK
          rgb_bitdist|  12|    100000|     100|0.77493468|  PASSED
rgb_minimum_distance|   2|     10000|    1000|0.50229520|  PASSED
rgb_minimum_distance|   3|     10000|    1000|0.12168602|  PASSED
rgb_minimum_distance|   4|     10000|    1000|0.62467177|  PASSED
rgb_minimum_distance|   5|     10000|    1000|0.01219930|  PASSED
    rgb_permutations|   2|    100000|     100|0.05636426|  PASSED
    rgb_permutations|   3|    100000|     100|0.97573587|  PASSED
    rgb_permutations|   4|    100000|     100|0.41677926|  PASSED
    rgb_permutations|   5|    100000|     100|0.85717028|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.99842781|   WEAK
      rgb_lagged_sum|   1|   1000000|     100|0.67510975|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.10355520|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.98222217|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.50553427|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.90169202|  PASSED
      rgb_lagged_sum|   6|   1000000|     100|0.32960778|  PASSED
      rgb_lagged_sum|   7|   1000000|     100|0.31451770|  PASSED
      rgb_lagged_sum|   8|   1000000|     100|0.39476211|  PASSED
      rgb_lagged_sum|   9|   1000000|     100|0.50640471|  PASSED
      rgb_lagged_sum|  10|   1000000|     100|0.25685843|  PASSED
      rgb_lagged_sum|  11|   1000000|     100|0.66008380|  PASSED
      rgb_lagged_sum|  12|   1000000|     100|0.48410093|  PASSED
      rgb_lagged_sum|  13|   1000000|     100|0.32772958|  PASSED
      rgb_lagged_sum|  14|   1000000|     100|0.70264346|  PASSED
      rgb_lagged_sum|  15|   1000000|     100|0.68734209|  PASSED

    rgb_permutations|   3|    100000|     100|0.09980266|  PASSED
    rgb_permutations|   4|    100000|     100|0.96659490|  PASSED
    rgb_permutations|   5|    100000|     100|0.78098543|  PASSED
      rgb_lagged_sum|   0|   1000000|     100|0.57730198|  PASSED
      rgb_lagged_sum|   1|   1000000|     100|0.85180213|  PASSED
      rgb_lagged_sum|   2|   1000000|     100|0.43380480|  PASSED
      rgb_lagged_sum|   3|   1000000|     100|0.19218276|  PASSED
      rgb_lagged_sum|   4|   1000000|     100|0.53990973|  PASSED
      rgb_lagged_sum|   5|   1000000|     100|0.87843486|  PASSED
      rgb_lagged_sum|   6|   1000000|     100|0.21913205|  PASSED
      rgb_lagged_sum|   7|   1000000|     100|0.06027132|  PASSED
      rgb_lagged_sum|   8|   1000000|     100|0.71840015|  PASSED
      rgb_lagged_sum|   9|   1000000|     100|0.96914031|  PASSED
      rgb_lagged_sum|  10|   1000000|     100|0.67929659|  PASSED
      rgb_lagged_sum|  11|   1000000|     100|0.67788082|  PASSED
      rgb_lagged_sum|  12|   1000000|     100|0.59333396|  PASSED
      rgb_lagged_sum|  13|   1000000|     100|0.95379794|  PASSED
      rgb_lagged_sum|  14|   1000000|     100|0.08420541|  PASSED
      rgb_lagged_sum|  15|   1000000|     100|0.97806831|  PASSED
      rgb_lagged_sum|  16|   1000000|     100|0.42151882|  PASSED
      rgb_lagged_sum|  17|   1000000|     100|0.65285144|  PASSED
      rgb_lagged_sum|  18|   1000000|     100|0.56537774|  PASSED
```

```
      rgb_lagged_sum|  19|   1000000|     100|0.76658545|  PASSED
      rgb_lagged_sum|  20|   1000000|     100|0.91831718|  PASSED
      rgb_lagged_sum|  21|   1000000|     100|0.96754603|  PASSED
      rgb_lagged_sum|  22|   1000000|     100|0.99947593|   WEAK
      rgb_lagged_sum|  23|   1000000|     100|0.99946583|   WEAK
      rgb_lagged_sum|  24|   1000000|     100|0.54156150|  PASSED
      rgb_lagged_sum|  25|   1000000|     100|0.54959154|  PASSED
      rgb_lagged_sum|  26|   1000000|     100|0.13494025|  PASSED
      rgb_lagged_sum|  27|   1000000|     100|0.95548421|  PASSED
      rgb_lagged_sum|  28|   1000000|     100|0.80959370|  PASSED
      rgb_lagged_sum|  29|   1000000|     100|0.09321913|  PASSED
      rgb_lagged_sum|  30|   1000000|     100|0.84569819|  PASSED
      rgb_lagged_sum|  31|   1000000|     100|0.56779178|  PASSED
      rgb_lagged_sum|  32|   1000000|     100|0.03361580|  PASSED
     rgb_kstest_test|   0|     10000|    1000|0.22746354|  PASSED
     dab_bytedistrib|   0|  51200000|       1|0.95774898|  PASSED
             dab_dct| 256|     50000|       1|0.52843799|  PASSED
Preparing to run test 207.  ntuple = 0
        dab_filltree|  32|  15000000|       1|0.24305124|  PASSED
        dab_filltree|  32|  15000000|       1|0.71077585|  PASSED
Preparing to run test 208.  ntuple = 0
       dab_filltree2|   0|   5000000|       1|0.39573147|  PASSED
       dab_filltree2|   1|   5000000|       1|0.22975306|  PASSED
Preparing to run test 209.  ntuple = 0
        dab_monobit2|  12|  65000000|       1|0.53374292|  PASSED
```

Chapter 3

Conclusion

The Qiskit simulator provides an accessible way to generate quantum random numbers using simulated quantum circuits. The randomness of these sequences was tested using the NIST and Diehard suites. While the generated data met some randomness criteria, specific results suggest areas for improvement, particularly if these sequences are to be used in high-stakes applications such as cryptography.

Chapter 4

Using Qiskit Hardware

4.1 Objective

The objective of this section is to generate quantum random numbers (QRNG) using IBM's actual quantum hardware and to evaluate the quality of randomness in the generated sequences. The Diehard test suite was used to assess the randomness of the data generated.

4.2 Qiskit Hardware

4.2.1 Hardware-Based Quantum Random Sequence Generation

For this experiment, IBM's quantum hardware was used to generate binary random sequences. A quantum circuit was constructed and executed on real quantum hardware to produce true quantum random bits.

4.2.2 Implementation

The implementation for generating the QRNG data on IBM hardware was carried out in Python using Qiskit.

```
# %%
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
    ↪ , transpile
from qiskit_ibm_runtime import QiskitRuntimeService, Session, Sampler
import numpy as np
from qiskit.transpiler.preset_passmanagers import
    ↪ generate_preset_pass_manager


# %%
def initialize_service(api_token):
    QiskitRuntimeService.save_account(channel="ibm_quantum", token="
        ↪ deed0480b9f0aa55dcba0e6b87e880978deecf31964257d1b986d4968f4e4b980ef0b38
        ↪ ", overwrite=True)
    service = QiskitRuntimeService()
    return service

# %%
```

```
# Quantum Random Bit Generator using IBM Hardware
def generate_random_bits_hardware(api_token, qubits=100, shots=1000):
    # Initialize service
    service = initialize_service(api_token)

    # Select real quantum backend (for example, 'ibmq_montreal' or '
       ↪ ibmq_brisbane')
    backend = service.backend("ibm_brisbane")


    # Create quantum and classical registers
    q = QuantumRegister(qubits, 'q')
    c = ClassicalRegister(qubits, 'c')
    circ = QuantumCircuit(qubits)

    # Apply Hadamard gate to all qubits to create superposition
    for i in range(qubits):
        circ.h(q[i])

    # Measure all qubits
    circ.measure_all()

    # Transpile the circuit for the backend
    compiled_circuit = transpile(circ, backend)

    # Initialize sampler and run the job on the selected backend
    sampler = Sampler(backend)

    pm = generate_preset_pass_manager(optimization_level=3, backend=
       ↪ backend)
    isa = pm.run(circ)

    pub = (isa)
    job = sampler.run([pub], shots=shots)

    # Get the result
    result = job.result()

    # Extract counts and convert to binary digits
    counts = result[0].data.meas.get_counts()


    # Generate random bits from the counts
    random_bits = []
    for key in counts:
        random_bits.extend([key] * int(counts[key] * shots))

    return ''.join(random_bits)

# %%
# Main function to generate 100,000 binary digits
def main():
    api_token = "
       ↪ deed0480b9f0aa55dcba0e6b87e880978deecf31964257d1b986d4968f4e4b980ef0b38
```

```
        ↪ "  # Replace with your IBM Quantum API token
    required_bits = 100000
    total_bits = []

    # Generate random bits in batches until we reach 100,000 bits
    while len(total_bits) < required_bits:
        new_bits = generate_random_bits_hardware(api_token, qubits
            ↪ =100, shots=1000)
        total_bits.extend(new_bits)

    # Truncate to exactly 100,000 bits
    total_bits = total_bits[:required_bits]

    # Save the random bits to a file
    with open('quantum_random_bits_100000.txt', 'w') as f:
        f.write(''.join(total_bits))

    print(f"Generated␣100,000␣random␣bits␣and␣saved␣to␣'
        ↪ quantum_random_bits_100000.txt'.")

# Execute the main function
if __name__ == "__main__":
    main()

# %%
```

4.3 Quantum Random Sequence

The binary sequence generated by IBM's quantum hardware is stored in the file.

(a) Overview of Quantum Random Sequence

This section provides an overview of the 100,000-bit quantum random sequence generated by the hardware. The sequence was saved in `quantum_random_bits_100000.txt` for randomness testing.

(b) Statistical Tests Setup

To evaluate the randomness of the sequence, we applied the Diehard test suite. This set of tests is designed to rigorously evaluate the statistical properties of the random data.

4.4 Results

The Diehard test results provide an assessment of the randomness in the quantum sequence generated by the Qiskit hardware.

Diehard Test Results

The results of the Diehard test suite, stored in `Result_diehard.txt`, indicate the degree of randomness in the hardware-generated sequence.

```
linux_prash@prashant:~/Downloads$ ls
binary_output2.txt  binary_output.txt  output.txt  Qiskit_Binary_Sim.txt  quantum_random_bits_100000.txt  Sumit_dieHard.txt
linux_prash@prashant:~/Downloads$ dieharder -a -f quantum_random_bits_100000.txt
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
        mt19937|  quantum_random_bits_100000.txt|  3.35e+07  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.06018839|  PASSED
      diehard_operm5|   0|   1000000|     100|0.13806673|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.17395287|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.17454821|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.69676465|  PASSED
        diehard_opso|   0|   2097152|     100|0.24132572|  PASSED
        diehard_oqso|   0|   2097152|     100|0.71654661|  PASSED
         diehard_dna|   0|   2097152|     100|0.97822671|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.39531238|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.32284515|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.47166550|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.09262770|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.66975408|  PASSED
     diehard_squeeze|   0|    100000|     100|0.90097014|  PASSED
        diehard_sums|   0|       100|     100|0.30508659|  PASSED
        diehard_runs|   0|    100000|     100|0.91554858|  PASSED
        diehard_runs|   0|    100000|     100|0.58355106|  PASSED
       diehard_craps|   0|    200000|     100|0.07781339|  PASSED
       diehard_craps|   0|    200000|     100|0.95626396|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.39584935|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.79540298|  PASSED
         sts_monobit|   1|    100000|     100|0.09412547|  PASSED
            sts_runs|   2|    100000|     100|0.74247395|  PASSED
          sts_serial|   1|    100000|     100|0.52952451|  PASSED
          sts_serial|   2|    100000|     100|0.51671623|  PASSED
          sts_serial|   3|    100000|     100|0.91485962|  PASSED
          sts_serial|   3|    100000|     100|0.38671305|  PASSED
          sts_serial|   4|    100000|     100|0.82758756|  PASSED
```

```
           sts_serial|   4|    100000|      100|0.78721787|  PASSED
           sts_serial|   5|    100000|      100|0.62178642|  PASSED
           sts_serial|   5|    100000|      100|0.76983891|  PASSED
           sts_serial|   6|    100000|      100|0.31161970|  PASSED
           sts_serial|   6|    100000|      100|0.11043302|  PASSED
           sts_serial|   7|    100000|      100|0.73411246|  PASSED
           sts_serial|   7|    100000|      100|0.30487648|  PASSED
           sts_serial|   8|    100000|      100|0.86798655|  PASSED
           sts_serial|   8|    100000|      100|0.22974106|  PASSED
           sts_serial|   9|    100000|      100|0.78280011|  PASSED
           sts_serial|   9|    100000|      100|0.75389381|  PASSED
           sts_serial|  10|    100000|      100|0.75119268|  PASSED
           sts_serial|  10|    100000|      100|0.28786462|  PASSED
           sts_serial|  11|    100000|      100|0.91548026|  PASSED
           sts_serial|  11|    100000|      100|0.42832579|  PASSED
           sts_serial|  12|    100000|      100|0.54793890|  PASSED
           sts_serial|  12|    100000|      100|0.71661571|  PASSED
           sts_serial|  13|    100000|      100|0.19653101|  PASSED
           sts_serial|  13|    100000|      100|0.17959329|  PASSED
           sts_serial|  14|    100000|      100|0.29998442|  PASSED
           sts_serial|  14|    100000|      100|0.31887814|  PASSED
           sts_serial|  15|    100000|      100|0.05481799|  PASSED
           sts_serial|  15|    100000|      100|0.52889838|  PASSED
           sts_serial|  16|    100000|      100|0.39918874|  PASSED
           sts_serial|  16|    100000|      100|0.85708988|  PASSED
          rgb_bitdist|   1|    100000|      100|0.87563362|  PASSED
          rgb_bitdist|   2|    100000|      100|0.98299922|  PASSED
          rgb_bitdist|   3|    100000|      100|0.68435478|  PASSED
          rgb_bitdist|   4|    100000|      100|0.24506140|  PASSED
          rgb_bitdist|   5|    100000|      100|0.96784096|  PASSED
          rgb_bitdist|   6|    100000|      100|0.73136745|  PASSED
          rgb_bitdist|   7|    100000|      100|0.27095485|  PASSED
          rgb_bitdist|   8|    100000|      100|0.05460533|  PASSED
          rgb_bitdist|   9|    100000|      100|0.35233405|  PASSED
          rgb_bitdist|  10|    100000|      100|0.39318346|  PASSED
          rgb_bitdist|  11|    100000|      100|0.25216216|  PASSED
          rgb_bitdist|  12|    100000|      100|0.85523761|  PASSED
rgb_minimum_distance|   2|     10000|     1000|0.68026678|  PASSED
rgb_minimum_distance|   3|     10000|     1000|0.82200208|  PASSED
rgb_minimum_distance|   4|     10000|     1000|0.44067562|  PASSED
rgb_minimum_distance|   5|     10000|     1000|0.02443506|  PASSED
    rgb_permutations|   2|    100000|      100|0.39374773|  PASSED
    rgb_permutations|   3|    100000|      100|0.97508522|  PASSED
    rgb_permutations|   4|    100000|      100|0.72961050|  PASSED
    rgb_permutations|   5|    100000|      100|0.97819857|  PASSED
      rgb_lagged_sum|   0|   1000000|      100|0.00475002|    WEAK
      rgb_lagged_sum|   1|   1000000|      100|0.51066630|  PASSED
      rgb_lagged_sum|   2|   1000000|      100|0.24421013|  PASSED
      rgb_lagged_sum|   3|   1000000|      100|0.79797066|  PASSED
      rgb_lagged_sum|   4|   1000000|      100|0.95676011|  PASSED
      rgb_lagged_sum|   5|   1000000|      100|0.57667841|  PASSED
      rgb_lagged_sum|   6|   1000000|      100|0.57124813|  PASSED
```

Chapter 5

Conclusion

The results demonstrate that IBM's quantum hardware can generate high-quality random numbers that pass most of the Diehard tests. However, certain test results indicate areas where the randomness could be further improved. These findings are significant for applications requiring high-security random numbers, such as cryptography, where genuine randomness is essential.

Chapter 6

Using SPAD for Quantum Random Number Generation

6.1 Objective

The purpose of this section is to generate quantum random numbers using a Single-Photon Avalanche Diode (SPAD) and analyze their randomness. A dataset containing 2 lakh data points was collected and evaluated using NIST and Diehard statistical tests to assess the quality of randomness.

Chapter 7

SPAD Dataset Analysis

7.1 2 Lakh Data Points

The 2 lakh dataset consists of binary values generated by the SPAD, saved and photon arrival times recorded. This dataset was analyzed using both NIST and Diehard statistical tests to assess randomness quality.

7.1.1 Data Collection and Format

The binary sequence generated by the SPAD is stored.

7.1.2 Statistical Testing of Randomness

NIST Test Results

The NIST test suite was applied to assess the randomness of the 2 lakh dataset. The test results are stored.

```
Test Data File:C:/Users/sumit/Downloads/IIT J Sem Notes/Sem 3/Intro to Quantum tech/Project QRNG/binary_output2.txt


Type of Test                                                P-Value                 Conclusion
01. Frequency (Monobit) Test                                0.0                     Non-Random
02. Frequency Test within a Block                           0.0                     Non-Random
03. Runs Test                                               0.0                     Non-Random
04. Test for the Longest Run of Ones in a Block             0.0                     Non-Random
05. Binary Matrix Rank Test                                 3.051157884563011 e-275  Non-Random
06. Discrete Fourier Transform (Spectral) Test              0.0                     Non-Random
07. Non-overlapping Template Matching Test                  0.0                     Non-Random
```

```
08. Overlapping Template Matching Test
    ↪ 1.5758998532716707e-69  Non-Random
09. Maurer's "Universal Statistical" Test           -1.0                
    ↪         Non-Random
10. Linear Complexity Test                          0.0                 
    ↪         Non-Random
11. Serial Test:
                                                    0.0                 
    ↪         Non-Random
                                                    0.0                 
    ↪         Non-Random
12. Approximate Entropy Test                        0.0                 
    ↪         Non-Random
13. Cumulative Sums Test (Forward)                  0.0                 
    ↪         Non-Random
14. Cumulative Sums Test (Backward)                 0.0                 
    ↪         Non-Random
15. Random Excursions Test:
                State        Chi Squared            P-Value             
    ↪         Conclusion
                -4           63.0                   
    ↪ 2.9111549198896303e-12    Non-Random
                -3           35.0                   
    ↪ 1.5046506621757205e-06    Non-Random
                -2           15.0                   
    ↪ 0.010362337915786429      Random
                -1           3.0                    
    ↪ 0.6999858358786276        Random
                +1           1.0                    
    ↪ 0.9625657732472964        Random
                +2           0.3333333333333333     
    ↪ 0.9969687632568645        Random
                +3           0.2                    
    ↪ 0.9991138612111875        Random
                +4           0.14285714285714285    
    ↪ 0.9996100613790039        Random
16. Random Excursions Variant Test:
                State        COUNTS                 P-Value             
    ↪         Conclusion
                -9.0         1                      1.0                 
    ↪         Random
                -8.0         1                      1.0                 
    ↪         Random
                -7.0         1                      1.0                 
    ↪         Random
                -6.0         1                      1.0                 
    ↪         Random
                -5.0         1                      1.0                 
    ↪         Random
                -4.0         1                      1.0                 
    ↪         Random
                -3.0         1                      1.0                 
    ↪         Random
                -2.0         1                      1.0                 
```

```
    ↪ ␣␣␣␣␣␣␣Random
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-1.0␣␣␣␣␣␣␣␣␣1␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣1.0␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ ␣␣␣␣␣␣␣␣Random
```

Diehard Test Results

Additionally, the Diehard test suite was used for a more comprehensive evaluation of randomness for the 2 lakh dataset. The test results are stored

```
sudo apt-get install dieharder
sudo apt-get install libdieharder-dev
dpkg -l | grep dieharder
dpkg -l | grep gsl
man dieharder
dieharder -v
cd Downloads
binary_output.txt
dieharder -a -f binary_output2.txt

#
    ↪ ==========================================================================
    ↪
#             dieharder version 3.31.1 Copyright 2003 Robert G. Brown
    ↪           #
#
    ↪ ==========================================================================
    ↪
    rng_name    |           filename             |rands/second|
         mt19937|              binary_output2.txt|  1.70e+07  |
#
    ↪ ==========================================================================
    ↪
         test_name   |ntup| tsamples |psamples|  p-value |Assessment
#
    ↪ ==========================================================================
    ↪
    diehard_birthdays|   0|       100|     100|0.91088416|  PASSED
       diehard_operm5|   0|   1000000|     100|0.34971292|  PASSED
   diehard_rank_32x32|   0|     40000|     100|0.67227970|  PASSED
     diehard_rank_6x8|   0|    100000|     100|0.76219697|  PASSED
    diehard_bitstream|   0|   2097152|     100|0.27336921|  PASSED
         diehard_opso|   0|   2097152|     100|0.48417279|  PASSED
         diehard_oqso|   0|   2097152|     100|0.72537769|  PASSED
          diehard_dna|   0|   2097152|     100|0.28605357|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.72009025|  PASSED
diehard_count_1s_byt|   0|    256000|     100|0.11602562|  PASSED
 diehard_parking_lot|   0|     12000|     100|0.97946883|  PASSED
    diehard_2dsphere|   2|      8000|     100|0.56820811|  PASSED
    diehard_3dsphere|   3|      4000|     100|0.30134087|  PASSED
     diehard_squeeze|   0|    100000|     100|0.11489486|  PASSED
        diehard_sums|   0|       100|     100|0.22905076|  PASSED
        diehard_runs|   0|    100000|     100|0.87066632|  PASSED
        diehard_runs|   0|    100000|     100|0.01880172|  PASSED
       diehard_craps|   0|    200000|     100|0.35548734|  PASSED
```

```
       diehard_craps|   0|    200000|      100|0.23137819|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|      100|0.98732141|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|      100|0.98971297|  PASSED
         sts_monobit|   1|    100000|      100|0.02686061|  PASSED
            sts_runs|   2|    100000|      100|0.33630387|  PASSED
          sts_serial|   1|    100000|      100|0.85246403|  PASSED
          sts_serial|   2|    100000|      100|0.27591560|  PASSED
          sts_serial|   3|    100000|      100|0.02448923|  PASSED
          sts_serial|   3|    100000|      100|0.04161327|  PASSED
          sts_serial|   4|    100000|      100|0.10261999|  PASSED
          sts_serial|   4|    100000|      100|0.55113060|  PASSED
          sts_serial|   5|    100000|      100|0.46836317|  PASSED
          sts_serial|   5|    100000|      100|0.61736399|  PASSED
          sts_serial|   6|    100000|      100|0.11718685|  PASSED
          sts_serial|   6|    100000|      100|0.33910613|  PASSED
          sts_serial|   7|    100000|      100|0.26591895|  PASSED
          sts_serial|   7|    100000|      100|0.69935910|  PASSED
          sts_serial|   8|    100000|      100|0.97529162|  PASSED
          sts_serial|   8|    100000|      100|0.04777839|  PASSED
          sts_serial|   9|    100000|      100|0.38104705|  PASSED
          sts_serial|   9|    100000|      100|0.04457882|  PASSED
          sts_serial|  10|    100000|      100|0.98566771|  PASSED
          sts_serial|  10|    100000|      100|0.98882750|  PASSED
          sts_serial|  11|    100000|      100|0.12885066|  PASSED
          sts_serial|  11|    100000|      100|0.04245456|  PASSED
          sts_serial|  12|    100000|      100|0.00495070|   WEAK
          sts_serial|  12|    100000|      100|0.02399071|  PASSED
          sts_serial|  13|    100000|      100|0.08055456|  PASSED
          sts_serial|  13|    100000|      100|0.80370804|  PASSED
          sts_serial|  14|    100000|      100|0.76197005|  PASSED
          sts_serial|  14|    100000|      100|0.91972517|  PASSED
          sts_serial|  15|    100000|      100|0.05400461|  PASSED
          sts_serial|  15|    100000|      100|0.27123778|  PASSED
          sts_serial|  16|    100000|      100|0.56574484|  PASSED
          sts_serial|  16|    100000|      100|0.71146249|  PASSED
         rgb_bitdist|   1|    100000|      100|0.37541986|  PASSED
         rgb_bitdist|   2|    100000|      100|0.96796363|  PASSED
         rgb_bitdist|   3|    100000|      100|0.23805959|  PASSED
         rgb_bitdist|   4|    100000|      100|0.87484411|  PASSED
         rgb_bitdist|   5|    100000|      100|0.99435408|  PASSED
         rgb_bitdist|   6|    100000|      100|0.48276982|  PASSED
         rgb_bitdist|   7|    100000|      100|0.92917224|  PASSED
         rgb_bitdist|   8|    100000|      100|0.89168527|  PASSED
         rgb_bitdist|   9|    100000|      100|0.96904599|  PASSED
         rgb_bitdist|  10|    100000|      100|0.11358462|  PASSED
         rgb_bitdist|  11|    100000|      100|0.20348258|  PASSED
         rgb_bitdist|  12|    100000|      100|0.57409853|  PASSED
rgb_minimum_distance|   2|     10000|     1000|0.36250261|  PASSED
rgb_minimum_distance|   3|     10000|     1000|0.53233276|  PASSED
rgb_minimum_distance|   4|     10000|     1000|0.10784369|  PASSED
rgb_minimum_distance|   5|     10000|     1000|0.16848636|  PASSED
    rgb_permutations|   2|    100000|      100|0.93885971|  PASSED
    rgb_permutations|   3|    100000|      100|0.09980266|  PASSED
    rgb_permutations|   4|    100000|      100|0.96659490|  PASSED
```

```
    rgb_permutations|   5|    100000|      100|0.78098543|  PASSED
      rgb_lagged_sum|   0|   1000000|      100|0.57730198|  PASSED
      rgb_lagged_sum|   1|   1000000|      100|0.85180213|  PASSED
      rgb_lagged_sum|   2|   1000000|      100|0.43380480|  PASSED
      rgb_lagged_sum|   3|   1000000|      100|0.19218276|  PASSED
      rgb_lagged_sum|   4|   1000000|      100|0.53990973|  PASSED
      rgb_lagged_sum|   5|   1000000|      100|0.87843486|  PASSED
      rgb_lagged_sum|   6|   1000000|      100|0.21913205|  PASSED
      rgb_lagged_sum|   7|   1000000|      100|0.06027132|  PASSED
      rgb_lagged_sum|   8|   1000000|      100|0.71840015|  PASSED
      rgb_lagged_sum|   9|   1000000|      100|0.96914031|  PASSED
      rgb_lagged_sum|  10|   1000000|      100|0.67929659|  PASSED
      rgb_lagged_sum|  11|   1000000|      100|0.67788082|  PASSED
      rgb_lagged_sum|  12|   1000000|      100|0.59333396|  PASSED
      rgb_lagged_sum|  13|   1000000|      100|0.95379794|  PASSED
      rgb_lagged_sum|  14|   1000000|      100|0.08420541|  PASSED
      rgb_lagged_sum|  15|   1000000|      100|0.97806831|  PASSED
      rgb_lagged_sum|  16|   1000000|      100|0.42151882|  PASSED
      rgb_lagged_sum|  17|   1000000|      100|0.65285144|  PASSED
      rgb_lagged_sum|  18|   1000000|      100|0.56537774|  PASSED
      rgb_lagged_sum|  19|   1000000|      100|0.76658545|  PASSED
      rgb_lagged_sum|  20|   1000000|      100|0.91831718|  PASSED
      rgb_lagged_sum|  21|   1000000|      100|0.96754603|  PASSED
      rgb_lagged_sum|  22|   1000000|      100|0.99947593|   WEAK
      rgb_lagged_sum|  23|   1000000|      100|0.99946583|   WEAK
      rgb_lagged_sum|  24|   1000000|      100|0.54156150|  PASSED
      rgb_lagged_sum|  25|   1000000|      100|0.54959154|  PASSED
      rgb_lagged_sum|  26|   1000000|      100|0.13494025|  PASSED
      rgb_lagged_sum|  27|   1000000|      100|0.95548421|  PASSED
      rgb_lagged_sum|  28|   1000000|      100|0.80959370|  PASSED
      rgb_lagged_sum|  29|   1000000|      100|0.09321913|  PASSED
      rgb_lagged_sum|  30|   1000000|      100|0.84569819|  PASSED
      rgb_lagged_sum|  31|   1000000|      100|0.56779178|  PASSED
      rgb_lagged_sum|  32|   1000000|      100|0.03361580|  PASSED
     rgb_kstest_test|   0|     10000|     1000|0.22746354|  PASSED
     dab_bytedistrib|   0|  51200000|        1|0.95774898|  PASSED
             dab_dct| 256|     50000|        1|0.52843799|  PASSED
Preparing to run test 207.  ntuple = 0
        dab_filltree|  32|  15000000|        1|0.24305124|  PASSED
        dab_filltree|  32|  15000000|        1|0.71077585|  PASSED
Preparing to run test 208.  ntuple = 0
       dab_filltree2|   0|   5000000|        1|0.39573147|  PASSED
       dab_filltree2|   1|   5000000|        1|0.22975306|  PASSED
Preparing to run test 209.  ntuple = 0
        dab_monobit2|  12|  65000000|        1|0.53374292|  PASSED
```

Chapter 8

Conclusion

The SPAD-generated 2 lakh dataset underwent randomness testing through NIST and Diehard suites. The NIST tests indicated some non-random behavior, while the Diehard results showed better randomness, with several tests indicating sufficient randomness. These results suggest that increasing the dataset size may improve randomness quality, though further refinements may be necessary for applications requiring high-security randomness.