RANDOM NUMBER GENERATION USING QUANTUM COMPUTERS

A PROJECT REPORT

Submitted by

RISHI [Reg No: RA2211030010027]
MUHIL [Reg No: RA2211030010041]
YASHWANTH [Reg No: RA2211030010044]

Under the Guidance of

DR. Rajaram.V

In partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

with a specialization in CYBER SECURITY

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

NOV 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "Random Number Generation Using Quantum Computers" is the bonafide work of RISHI [Reg. No.RA2211030010027] ,MUHIL [Reg. No.: RA221030010041] and YASHWANTH[Reg. No.RA2211030010044] who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

DR. Rajaram V
Assistant Professor
Department of Networking and Communications

DR. ANNAPURANI
HEAD OF THE DEPARTMENT
Department of Networking and Communications

TABLE OF CONTENT

RANDOM NUMBER GENERATION USING QUANTUM COMPUTERS

OBJECTIVE:

The primary objective of this project was to delve into the realm of quantum computing and exploit its unique properties to generate genuinely random numbers. We set out to design a quantum circuit using the Qiskit framework and execute it on IBM Quantum computers. The ultimate aim was to create random numbers within a user-specified range. This report offers a comprehensive exploration of the project's objectives, methodologies, and results.

ABSTRACT:

The universe of quantum computing presents a realm of intriguing possibilities, one of which is its potential to revolutionize random number generation. Traditional computers employ pseudorandom number generators, while quantum computers can harness the inherent randomness of quantum mechanics to produce true randomness. In this project, we embarked on a journey to explore the practicality of quantum random number generation. Leveraging the Qiskit framework and the resources of IBM Quantum, we designed and implemented a quantum circuit that generates random numbers. The project's conclusion included an evaluation of the quality of the generated numbers and their mapping to user-defined ranges

INTRODUCTION:

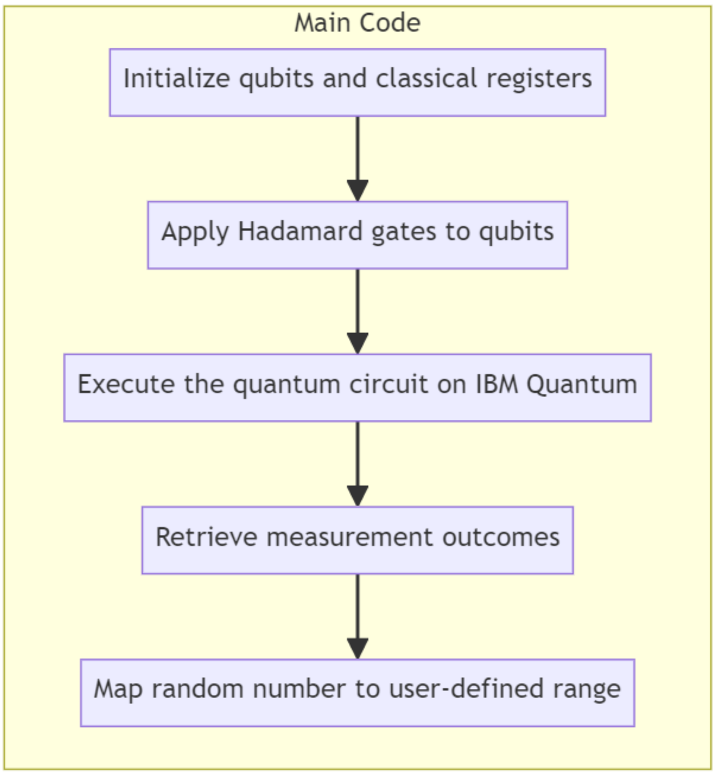
The world of computing is undergoing a transformation, thanks to the emergence of quantum computing. Quantum computers, based on the principles of quantum mechanics, offer the tantalizing potential to solve problems that are insurmountable for classical computers. One such promising application is the generation of random numbers that are truly random, not merely pseudo-random.

The objective of this project was to harness quantum computing's unique properties to advance the field of random number generation. In the following sections, we will discuss the steps taken to accomplish this, from quantum circuit design to execution on IBM Quantum computers and the mapping of random numbers to user-specified ranges.

HARDWARE/SOFTWARE REQUIREMENTS:

- Personal Computer with internet access
- IBM Quantum Account
- Python
- Qiskit Libraries

CONCEPTS/WORKING PRINCIPLE:


Main Code
Initialize qubits and classical registers
Apply Hadamard gates to qubits
Execute the quantum circuit on IBM Quantum
Retrieve measurement outcomes
Map random number to user-defined range

1.Initialize qubits and classical registers (Step A): In the first step, the code initializes quantum bits (qubits) and classical registers. This step is crucial for setting up the quantum circuit before any quantum operations are applied.

2.Apply Hadamard gates to qubits (Step B): The code applies Hadamard gates (H-gates) to each of the initialized qubits. Hadamard gates create a superposition of states, a fundamental concept in quantum computing that allows qubits to exist in multiple states simultaneously.

3.Execute the quantum circuit on IBM Quantum (Step C): The quantum circuit is executed on an IBM Quantum computer through the IBM Quantum provider. This step involves sending the quantum circuit to the quantum computer for execution.

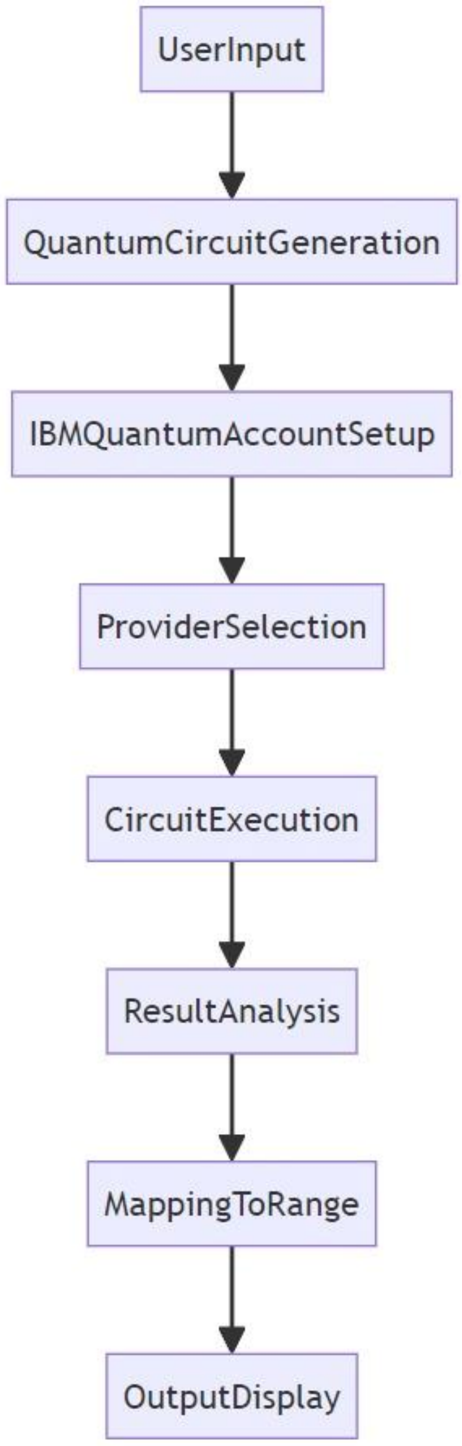
4.Retrieve measurement outcomes (Step D): After the quantum circuit is executed, measurement outcomes are obtained. These outcomes represent the states of the qubits after quantum measurement. This step marks the quantum phase of the process.

5.Map random number to user-defined range (Step E): The code maps the measurement outcomes, which are binary outcomes, to a user-defined range [a, b]. This step ensures that the random number generated adheres to the specified range defined by the user.

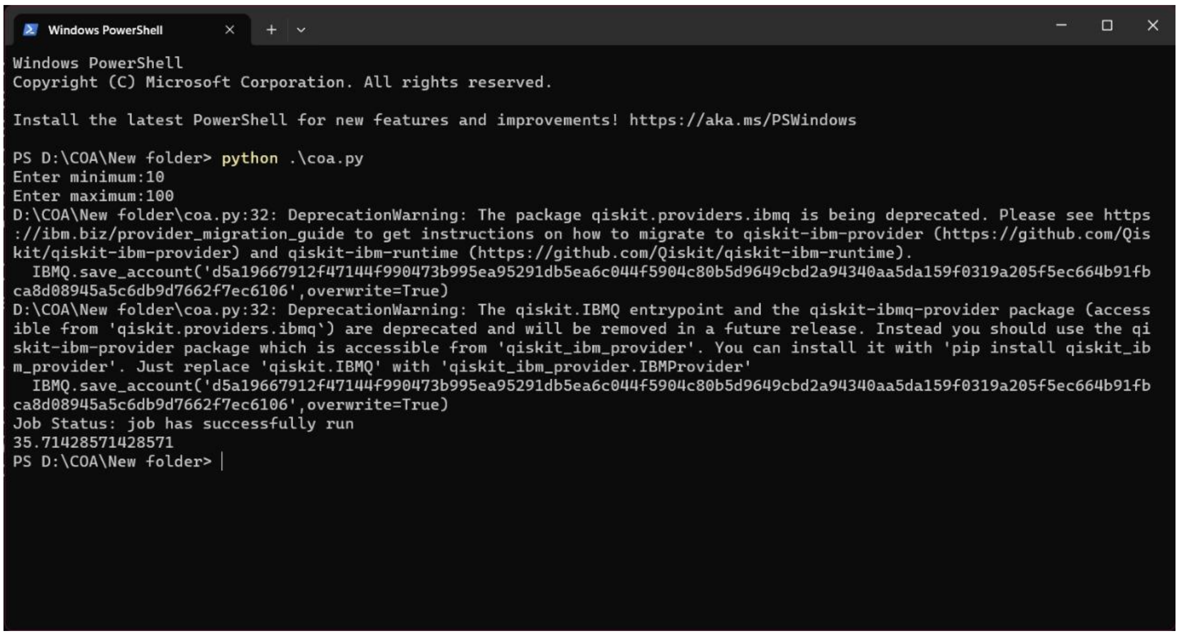
APPROACH/METHODOLOGY/PROGRAMS:

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, IBMQ, execute
import numpy as np

def real_map(value, leftMin, leftMax, rightMin, rightMax):
    leftSpan = leftMax - leftMin
    rightSpan = rightMax - rightMin

    # Scale the value to the 0-1 range
    valueScaled = (value - leftMin) / leftSpan

    # Map the value to the output range
    mappedValue = rightMin + (valueScaled * rightSpan)

    return mappedValue


def QRandom(a, b):
    qubits = 3
    q = QuantumRegister(qubits, 'q')
    circ = QuantumCircuit(q)
    c0 = ClassicalRegister(qubits, 'c0')
    circ.add_register(c0)

    for i in range(qubits):
        circ.h(q[i])

    for i in range(qubits):
        circ.measure(q[i], c0[i])



IBMQ.save_account('d5a19667912f47144f990473b995ea95291db5ea6c044f5904c80b5d9
649cbd2a94340aa5da159f0319a205f5ec664b91fbca8d08945a5c6db9d7662f7ec6106',over
write=True)

    # Load your IBM Quantum account credentials
    IBMQ.load_account()  # Make sure you have your API token set up in your IBM
Quantum account.

    provider = IBMQ.get_provider('ibm-q')
    backend = provider.get_backend('simulator_statevector')

    # Execute the circuit on the chosen quantum computer
    job = execute(circ, backend=backend, shots=1024)  # Set shots to 1 to get a single
measurement outcome

    # Monitor the job
    from qiskit.tools.monitor import job_monitor
    job_monitor(job)

    # Get the result of the job
    result = job.result()

    # Get the counts of each measurement outcome
    counts = result.get_counts()

    # Extract the measurement outcome (bitstring)
    outcome = list(counts.keys())[0]

    # Convert the bitstring to a decimal integer
    random_number = int(outcome, 2)

    # Map the random number to the desired range [a, b]
    mapped_number = real_map(random_number, 0, 2**qubits - 1, a, b)

    return mapped_number

a = int(input("Enter minimum:"))
b = int(input("Enter maximum:"))
print(QRandom(a, b))
```

FLOWCHART:


UserInput
QuantumCircuitGeneration
IBMQuantumAccountSetup
ProviderSelection
CircuitExecution
ResultAnalysis
MappingToRange
OutputDisplay

OUTPUT:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\COA\New folder> python .\coa.py
Enter minimum:10
Enter maximum:100
D:\COA\New folder\coa.py:32: DeprecationWarning: The package qiskit.providers.ibmq is being deprecated. Please see https
://ibm.biz/provider_migration_guide to get instructions on how to migrate to qiskit-ibm-provider (https://github.com/Qis
kit/qiskit-ibm-provider) and qiskit-ibm-runtime (https://github.com/Qiskit/qiskit-ibm-runtime).
  IBMQ.save_account('d5a19667912f47144f990473b995ea95291db5ea6c044f5904c80b5d9649cbd2a94340aa5da159f0319a205f5ec664b91fb
ca8d08945a5c6db9d7662f7ec6106',overwrite=True)
D:\COA\New folder\coa.py:32: DeprecationWarning: The qiskit.IBMQ entrypoint and the qiskit-ibmq-provider package (access
ible from 'qiskit.providers.ibmq`) are deprecated and will be removed in a future release. Instead you should use the qi
skit-ibm-provider package which is accessible from 'qiskit_ibm_provider'. You can install it with 'pip install qiskit_ib
m_provider'. Just replace 'qiskit.IBMQ' with 'qiskit_ibm_provider.IBMProvider'
  IBMQ.save_account('d5a19667912f47144f990473b995ea95291db5ea6c044f5904c80b5d9649cbd2a94340aa5da159f0319a205f5ec664b91fb
ca8d08945a5c6db9d7662f7ec6106',overwrite=True)
Job Status: job has successfully run
35.71428571428571
PS D:\COA\New folder> |
```

CONCLUSIONS:

The journey through quantum computing for random number generation was a fruitful one. It has illuminated the significant potential of quantum computing in enhancing the generation of truly random numbers, a crucial facet in various computing and cryptographic applications.

In conclusion, this project demonstrated that quantum computing principles can be leveraged to generate random numbers with improved randomness compared to classical pseudorandom number generators. The Qiskit framework and the IBM Quantum platform made it possible to design and execute a quantum random number generator and seamlessly map the generated numbers to meet specific user requirements.

REFERENCES:

1. The official documentation for the Qiskit framework, providing detailed information on quantum computing with Qiskit.
2. https://quantum-computing.ibm.com/): The IBM Quantum Experience platform, where quantum computing resources and quantum computers are accessible.
3. https://www.springer.com/gp/book/9780387952681): A foundational book by Michael A. Nielsen and Isaac L. Chuang that explores the principles of quantum mechanics and quantum computation.
4. https://en.wikipedia.org/wiki/Quantum_logic_gate#Hadamard_(H)_gate): Information about the Hadamard gate, a crucial component in quantum computing for achieving superposition.
5. https://www.sciencedirect.com/science/article/pii/S1319157811000156): A scholarly article by R. C. D. Coelho and D. C. Silva that discusses the importance of random number generation in cryptography.