Qubit Reset and Refresh: A Gamechanger for Random Number Generation

Julie Germain
Computer Science and Engineering
University of North Texas
Denton, TX, USA
JulieGermain@my.unt.edu

Ram Dantu
Computer Science and Engineering
University of North Texas
Denton, TX, USA
Ram.Dantu@unt.edu

Mark Thompson
Computer Science and Engineering
University of North Texas
Denton, TX, USA
Mark.Thompson2@unt.edu

ABSTRACT

Generation of random binary numbers for cryptographic use is often addressed using pseudorandom number generating functions in compilers and specialized cryptographic packages. Using the IBM's Qiskit reset functionality, we were able to implement a straight-forward in-line Python function that returns a list of quantum-generated random numbers, by creating and executing a circuit on IBM quantum systems.

We successfully created a list of 1000 1024-bit binary random numbers as well as a list of 40,000 25-bit binary random numbers for randomness testing, using the NIST Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. The quantum-generated random data we tested showed very strong randomness, according to the NIST suite.

Previously, IBM's quantum implementation required a single qubit for each bit of data generated in a circuit, making generation of large random numbers impractical. IBM's addition of the reset instruction eliminates this restriction and allows for the creation of functions that can generate a larger quantity of data-bit output, using only a small number of qubits.

CCS CONCEPTS

•Computer systems organization~Architectures~Other architectures~Quantum computing. •Theory of computation~Randomness, geometry and discrete structures

KEYWORDS

Quantum Computing, Random Number Generator, QRNG, Qiskit

ACM Reference Format:

Julie Germain, Ram Dantu, & Mark Thompson. 2022. Qubit Reset and Refresh: A Gamechanger for Random Number Generation. *In Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy (CODASPY '22), April 24–27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3508398.3519364

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author(s).
CODASPY '22, April 24–27, 2022, Baltimore, MD, USA.
© 2022 Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-9220-4/22/04.
https://doi.org/10.1145/3508398.3519364

1 INTRODUCTION AND MOTIVATION

Using the current generation of quantum computing we demonstrate how it can serve as a complement to classical computing. In the same way we might offload certain functions to a graphics accelerator or a tensor processor, we can offload targeted processes to quantum computers where they excel. In this work, we have implemented a quantum random number generator function via Python, highlighting the potential of cloud-based quantum computing for use by programs primarily written for classical computers.

Dedicated Quantum Random Number Generator (QRNG) hardware and cloud-based services exist [2], representing quantum technology's superior randomness, but they require specialized hardware or services, making it impractical for some uses. Using our technique with IBM's reset instruction, cryptographers will be able to call a simple inline Python function to generate random binary data, with access to only general-purpose cloud-based quantum hardware.

Our new qrandom function generates a list of quantum-generated random numbers of the bit length and quantity desired by the user. Using Reuse and Reset [4], a single circuit is created and customized to generate a single random number of the bit length desired, without being limited by the number of qubits available in the quantum computer. We created a multi-phase circuit to generate a single random binary number of the desired length to be run on machines with no more than seven qubits. By executing the resulting circuit the specified number of times as a batch job, a list of random numbers is generated. The function then returns the binary random numbers to the user as a list of ASCII strings.

2 CONTRIBUTIONS

- Innovative use of reset and refresh in the quantum hardware
- Validation of Randomness using NIST criteria
- A novel inline QRNG python function for cryptographers for generation of the keys, nonce, and other applications
- Theoretical proof that random number is not biased either towards 0 or 1

3 METHODOLOGY

Quantum circuits are created via a Python program using the IBM Qiskit SDK implementation. For this project, our function qrandom builds a customized circuit, based on the length and quantity of random numbers desired.

- One Random number of user specified length is created each time the circuit is executed
- The quantity of random numbers requested are generated by executing the circuit multiple times, once per random number
- Reset and Reuse enable multi-phase implementation, allowing a larger number of output random bits than qubits available [4],
- Output of the qrandom function is a list of random numbers returned as ASCII binary strings

A simple circuit with just one phase can be used to generate random numbers up to 7-bits in length. Figure 1 shows such a circuit to generate 4-bit random number. For our 4-qubit circuit represented here, we see each qubit is initialized (i.e., reset) to a value of 0, its ground state. To each qubit we then apply a single Hadamard gate (H) that places the qubit into superposition, a unique state in quantum physics that can maintain several separate quantum states simultaneously. When a Hadamard gate is applied to a qubit with a value of zero, its state is unbiased such that that the probability of measuring a value of zero or one is equal and thus purely random. In the circuit diagram, we see each qubit being measured and stored to a numbered classical bit in the order measured. After execution, the data stored in the classical bits is retrieved and returned to the user.

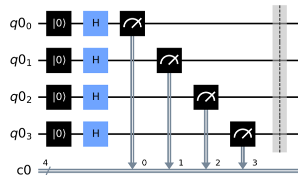

Figure 1: Qiskit circuit to generate a 4-bit random number in a single round (each qubit only measured once)

In comparison, we observe a 9-bit random number circuit implemented in three phases in Figure 2. Like the earlier single-phase circuit, we see the same gates used to generate just three random bits, using three qubits. Those three qubits are then reset to their ground state of zero for use in subsequent phases.

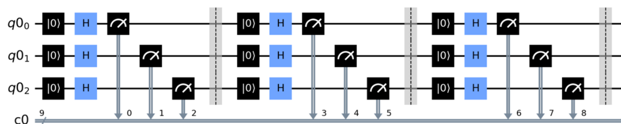

Figure 2: Qiskit circuit to generate a 9-bit random number in three phases (each qubit measured 3 times)

After the completion of all three phases, the classical bits numbered from 0-8 will store the 9-bit random number that results.

Most quantum algorithms examples implemented using Qiskit will output their results as a sorted list (the default) to produce a statistical histogram of results. In contrast, to generate lists of random numbers, sorting the data results in a loss of some aspects of its randomness, resulting in poor NIST test results. To overcome this issue, the data was read from memory prior to sorting.

The qrandom function created for this project has the signature qrandom(num_bits,num_random) where:

- num_bits is the length of random number required
- num_random is the quantity of random numbers the user wants implemented.

The output of the function is a list of ASCII binary number strings.

4 RANDOMNESS OF QUANTUM-GENERATED NUMBERS

The underlying quantum physics provides randomness as one of its most basic functions. Generating a random bit requires placing a qubit into superposition, after which its measured value will result in zero or one with equal probability.

Quantum state and the application of quantum gates in a circuit can be described using linear algebra and Dirac notation [3]. The Hadamard gate is defined by the following matrix:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1}$$

In Dirac notation, applying a Hadamard gate to either a value of 0 or 1 will result in the probability of ½. We also show a bloch sphere visualization of the qubit state of 0 and the qubit state after the Hadamard is applied, lying equally between a 0 and 1 value :

$$\begin{aligned} H: \quad & |0\rangle \rightarrow \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ & |1\rangle \rightarrow \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned} \tag{2}$$

$$\begin{aligned} &= {}^{1}/\sqrt{2}(\,|0\rangle + |1\rangle\,) \\ &= {}^{1}/\sqrt{2}\,|0\rangle + {}^{1}/\sqrt{2}\,|1\rangle \\ &= |\alpha|^2 \text{ or } |\beta|^2 \\ &= ({}^{1}/\sqrt{2})^2 \\ &= ½ \end{aligned} \tag{3}$$

Start | Apply H-gate

Therefore, we can be assured that whether the value starts at zero or one, applying a Hadamard gate puts the qubit into superposition and the probability of measuring a zero or one will be ½.

5 RANDOMNESS TESTING

The NIST Random Number testing suite [1] was used to compare our quantum-generated random numbers to those from a standard python random function, the PyCrytpodome random function, and NIST test data for pi. 40,000 samples of 25-bit length random numbers were generated from each source and saved off as ASCII

text files for use in the NIST suite of tests. The data test size was selected to be at least 1,000,000 bits in length, allowing us to select a stream length of 10,000 with 100-bit streams, greater than the 55 required of some tests. Our data set size of 1,000,000 bits was adequate to perform all tests except Rank, Linear Complexity, Non-Overlapping, Overlapping, Random Excursions and Random Excursion variants tests.

Key results from the NIST test suite are the Proportion (the number of our 100 samples that meet the criteria for randomness) and the P-Value (representing the distribution of statistical outcomes where perfectly evenly distributed would result in a value of 1).

Best Proportion	Best P-Value	Worst P-Value

	PyRandom		CryptoRandom		Data.pi		qRandom	
	P-Value	Proportion	P-Value	Proportion	P-Value	Proportion	P-Value	Proportion
Frequency	0.924076	99	0.595549	99	0.262249	98	0.153763	100
BlockFrequency	0.759756	100	0.224821	99	0.699313	100	0.514124	98
CumulativeSums	0.383827	99	0.883171	99	0.319084	100	0.946308	100
CumulativeSums	0.616305	99	0.102526	98	0.514124	99	0.657933	100
Runs	0.637119	99	0.935716	99	0.657933	100	0.037566	100
LongestRun	0.075719	100	0.678686	97	0.978072	100	0.319084	100
FFT	0.455937	99	0.494392	98	0.162606	100	0.037566	98
Serial	0.798139	99	0.534146	100	0.289667	98	0.637119	99
Serial	0.883171	99	0.699313	98	0.108791	100	0.062821	100

Table 1: NIST Random Number Test results Comparison

The results are shown in Table 1. We have color-coded these results, to assist in visualizing the comparable test success of each data source. The NIST provided pi data shows as random in the tests performed, though pi is not truly random due to its predictable placement of each number. The pi and the quantum random data sets were the most successful, in terms of the proportion of statistically random results (100% in 6 tests each).

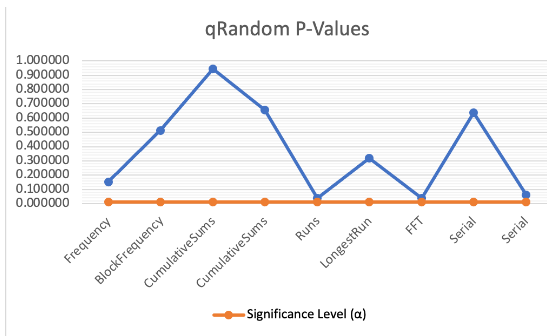

Figure 3: Quantum Random NIST test P-Values

Figure 3 shows a summary of the NIST random number test results, showing the Proportion of 100 samples that were statistically classified as Random and the P-Value (where one represents the optimal statistical distribution of the results).

6 CONCLUSIONS

The utilization of quantum computers for focused functions within a classical program has near-term applicability with the current generation of cloud-based quantum computing technology.

We have shown that in-line quantum functions, are practical to use for random number generation. They produce improved results, per the NIST test suite, over traditional random implementations in Python and PyCryptodome and comparable results to the NIST test data set based on pi.

The IBM Reset functionality, allowed the output of a large number of data bits with only a modest quantity of qubits, opening the door to more expanded and efficient use of the quantum computer resources.

ACKNOWLEDGMENTS

We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team.

In this paper we used ibm_perth, which is one of the IBM Quantum Falcon Processors.

We thank National Security Agency for the partial support through grants: H98230-20-1-0329, H98230-20-1-0403, H98230-20-1-0414, and H98230-21-1-0262

REFERENCES

[1] NIST Special Publication 800-22: A statistical test suite for random and pseudorandom number generators for cryptographic applications. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf

[2] Jacak, M.M., Jóźwiak, P., Niemczuk, J. et al. Quantum generators of random numbers. Sci Rep 11, 16108 (2021). https://doi.org/10.1038/s41598-021-95388-7

[3] Lam, R., The Math Behind Quantum Computing – Qubits and Superposition, (2019) https://medium.datadriveninvestor.com/the-math-behind-quantum-computing-qubits-and-superposition-f7a871668125

[4] Nation, P., Johnson, B., How to Measure and Reset a Qubit in the Middle of a Circuit Execution, 2021 https://www.ibm.com/blogs/research/2021/02/quantum-mid-circuit-measurement/