Quantum Random Number Generation Hackathon Challenge

Dylan Brehm and Joshua Deaton

October 29, 2024

Abstract

This document presents our approach to quantum random number generation (QRNG) using quantum computing principles. It covers probability theory relevant to random number generation, quantum computing concepts, implementation of QRNG, and analysis of the results.

Contents

1 Probability Theory

Before any quantum computing can be done to generate true random numbers, the probability theory required to create a uniform distribution of decimal floating-point numbers must first be derived. It must be found how to create a random number in the decimal system from a sequence of binary Bernoulli random variables.

1.1 Definitions

In order to analyze the method of creating a decimal random variable from binary ones, a few definitions must first be declared.

First, a random variable must be defined. A random variable is a number which cannot be predetermined that is the result of an action. In this case, the random variable is the number that results from measuring the outcome of a given qubit in a quantum circuit. The symbol X_i will be used to denote the random variable relating to measuring the same qubit the ith time in a quantum circuit.

The sample space of a random variable is the possible values that can result from the action taking place. The sample space for all X_i is as follows.

$$S_{X_i} = \{0, 1\} \tag{1}$$

This means that the possible values that can be measured from a given qubit are a zero or a one. Due to quantum effects, there are physical laws that assign probabilities for the resulting event occurring from this sample space. These laws drive the trueness of this true random number generator.

In the end, the same quantum circuit will be run multiple times, and the result of the quantum circuit will be recorded each time it is run. Each result will be organized into an ordered list, which, as a whole, creates the binary numeric result. This ordered list is denoted as following N-tuple where N is the number of bits in the binary number.

$$\vec{X} = (X_0, X_1, X_2, ..., X_N) \tag{2}$$

This N-tuple represents the random number generated from the random number generator in binary format. Often, people do mathematics in the decimal system. That means that this binary number must be converted into the base ten format. A transformation from the binary number $\vec{X}$ to a new random variable denoted by Y must be done where Y is the decimal random number. The sample space for Y is shown below.

$$Y = \{0, 1, 2, ..., 2^N - 1\} \tag{3}$$

Where $2^N - 1$ is the maximum decimal number that can be represented using N bits. These are all of the basic definitions necessary to derive the method of creating a truly random uniform distribution of decimal numbers using a quantum computer.

1.2 Making the Uniform Decimal Distribution

The conversion from binary to decimal is a relationship that is both beautiful and must be handled with care. There is a one-to-one mapping from the set of all possible binary representations $S_{\vec{X}}$ to the set of all possible decimal values S_Y. This means for each decimal value y in S_Y there is some unique corresponding $\vec{x}$ from $S_{\vec{X}}$. Additionally, all decimal values in the sample space of Y are mapped to from some binary representation in the sample space of $\vec{X}$. Both these properties together mean that this mapping is bijective. This specific mapping can be found from the formula which converts binary values to decimal values. This is done as follows.

$$Y = g(\vec{X}) = X_0(2^0) + X_1(2^1) + X_2(2^2) + ... + X_N(2^{N-1}) \tag{4}$$

This can be condensed to

$$Y = g(\vec{X}) = \sum_{n=0}^{N-1} X_n(2^n) \tag{5}$$

Because each binary number maps to only one unique decimal number, the probability of getting any given decimal number from the given binary sequence is equivalent to the probability of measuring the given binary sequence itself. This can be written as so.

$$P_Y(Y = g(\vec{x})) = P_{\vec{X}}(\vec{X} = \vec{x}) \tag{6}$$

This utilizes the probabilistic property of equivalent sets. Because these events are equivalent, they have the same probability. The next assumption that will be made is that all measurements are independent and identically distributed. This allows the probability of the tuple of sequential qubit measurements to be expressed as the product of probabilities of each measurement of the qubit. This is shown below.

$$P_{\vec{X}}(\vec{X} = \vec{x}) = \prod_{n=0}^{N-1} P_{X_i}(X_i = x_n) \tag{7}$$

Where each x_n is each element of the N-tuple outcome $\vec{x}$. A a quantum circuit will be chosen such that $P_{X_i}(X_i = 1)$ is fifty percent. Additionally, the total probability of X_i must be equal to one, so $P_{X_i}(X_i = 0)$ is also implicitly fifty percent. This means that no matter the result, equation (7) will always be the same probability. This is shown symbolically below.

$$P_{X_i}(X_i = 1) = P_{X_i}(X_i = 0) = 0.5 \tag{8}$$

$$P_{\vec{X}}(\vec{X} = \vec{x}) = \prod_{n=0}^{N-1} 0.5 \tag{9}$$

$$P_{\vec{X}}(\vec{X} = \vec{x}) = (0.5)^N \tag{10}$$

Since the sample space of X_i allows it only to be able to take on the values 0 or 1, then the only options for the argument of the product in equation (7) are either $P_{X_i}(X_i = 1)$ or $P_{X_i}(X_i = 0)$. It has been shown that these are both 0.5. This means that this product will always evaluate to $(0.5)^N$ no matter what $\vec{x}$ is. Because each $\vec{x}$ uniquely maps to a decimal value, the probability of all decimal values will also be $(0.5)^N$. Because the probability of all events is the same, the distribution of integers is uniform when using the quantum computer to generate qubits of fifty percent probability.

2 Quantum Computing Theory

2.1 Quantum Theory Overview

Quantum physics has allowed computing to be done in a new and exciting way. Rather than operating on bits that take on values of zero and one, the quantum computer operates on the probabilities of a qubit taking on a value of a zero or a one when measured. The qubits get initialized and are sent through a quantum circuit. In this quantum circuit, there are quantum gates that act on the qubits and change their probabilities. The qubits can then be measured many times to find the probability of the qubit and, thus, its state.

2.2 Quantum Bits and Their Probabilities

Quantum computing steals the bra-ket notation of vectors from quantum mechanics. The bra denoted as a $\langle X|$ is equivalent to a row vector $\vec{X}^T$ in linear algebra. Additionally, the ket denoted as a $|X\rangle$ is equivalent to a column vector $\vec{X}$ in linear algebra. The "Bracket" can be done to find the inner product of two vectors denoted as $\langle X\rangle$.

State vectors $|\psi\rangle$ can be expressed as a linear combination of the zero ket $|0\rangle$ and the one ket $|1\rangle$. This is shown symbolically below.

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \quad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix} \tag{11}$$

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle = a\begin{bmatrix}1\\0\end{bmatrix} + b\begin{bmatrix}0\\1\end{bmatrix} \tag{12}$$

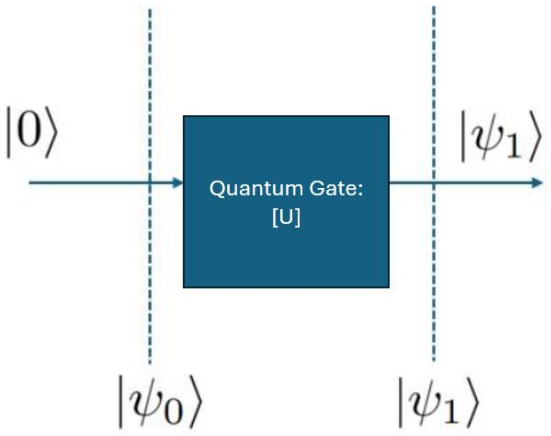

Figure 1: Simple Quantum Circuit

The meaning of the coefficients to the $|0\rangle$ and $|1\rangle$ vectors are very important. The square of each coefficient tells the probability of the measurement of $|\psi\rangle$ being $|0\rangle$ or $|1\rangle$. More explicitly, $|a|^2$ is the probability of measuring a $|0\rangle$ from the qubit and $|b|^2$ is the probability of measuring a $|1\rangle$ from the qubit.

2.3 Quantum Gates

The quantum circuit is initialized with $|\psi_0\rangle$ being $|0\rangle$. This means that if the qubit was measured, it would be certain that the $|0\rangle$ is measured. The programming that can be done to this qubit in order to do useful computations is through quantum gates. Quantum gates are able to modify the coefficients of the quantum state $|\psi_0\rangle$. This effectively modifies the probabilities associated with measuring the qubit resulting in a new quantum state $|\psi_1\rangle$. This process is visualized in Figure 1 where the dotted lines shows the progression of the quantum state $|\psi\rangle$.

This quantum circuit is a way of transforming the quantum state from one state to another. Mathematically, this transformation is linear and can be represented as a matrix multiply. Each quantum gate then simply takes the input column vector and does some matrix multiply to it, resulting in a new column vector. As a reminder, the probability of measuring a zero or a one is $|a|^2$ and $|b|^2$, respectively. Because the sample space is just zero and one, then $P(|0\rangle) + P(|1\rangle) = |a|^2 + |b|^2 = 1$. This is the same as saying that $\langle\psi\rangle = 1$, implying that matrix transformations done on $|\psi\rangle$ must preserve magnitudes. The class of matrices that do this are, in fact, rotation matrices! This can be taken a step further to account for potentially complex values of a and b. The class of matrix for the complex case is called unitary. If all values of the matrix are real, then this simplifies to the transpose being the inverse, which is true for all rotation matrices!

2.4 The Quantum Circuit of Interest

The question of interest now is what rotation matrix [U] will cause a $|0\rangle$ to become the following?

$$[U]\,|0\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle \tag{13}$$

This new state has the property that $|a|^2 = 1/2$ and $|b|^2 = 1/2$, which means that there is a fifty percent chance to measure a $|0\rangle$ state and a fifty percent chance to measure a $|1\rangle$ state. Some reverse engineering can find what matrix is required to do this. The matrix which accomplishes this is shown below.

$$[U] = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{14}$$

That is it! The only thing required to make a one-qubit quantum circuit that has a fifty percent of measuring a $|0\rangle$ and a fifty percent chance of measuring a $|1\rangle$ is just the gate given in equation 14.

This gate has a special name, and it is called the Hadamard gate. It is very useful in doing things like entangling qubits, but here, its only purpose is creating uncertainty from certainty. Figure 2 shows the complete quantum circuit required to make the random number generator.

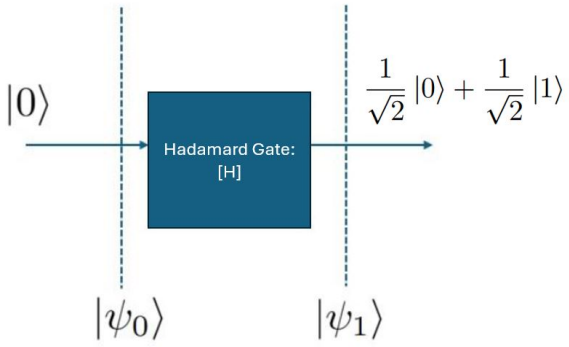

Figure 2: Quantum Circuit to be Implimented

This circuit will be implemented and analyzed in the next section.

3 Implimentation and Analysis

3.1 Code

The code for generating the quantum circuit and utilizing it is described by the block diagram in Figure 3.

Figure 3: Code Information Flow

There is a lot going on in Figure 3, so it will be described here.

1. A request is made to measure the quantum circuit with the Hadamard gate 64 times. Each measurement is stored sequentially in an array in the order they are measured.
2. The array of bits is sent to randint() to be converted to the integer format as described by equation (4).
3. the random integers are sent to rand. In rand, they are normalized by the number of possible integer values (2^{64}), and the result is output. The resulting sample space will be approximately continuous between zero and one.

This is the general information flow of the code. Next, the functions will be shown, and descriptions will be given regarding how each function works.

3.2 Function: randbits

```
def randbits(num_sample_bits : int = _num_sample_bits)-> List[int]:
    """
    Generates a list of random bits

    Returns:
        List[int]: List of random bits
    """

```

```
    circuit = QuantumCircuit(1)

    circuit.h(0)

    circuit.measure_all()

    pub = [(circuit)]
    sampler = BackendSamplerV2(backend=_backend)

    job = sampler.run(pub,shots=num_sample_bits)
    result = job.result()[0]
    rand_bits = result.data.meas.bitcount()

    return rand_bits
```

This function initializes a one-qubit quantum circuit using the QuantumCircuit(1) command. The hadamard gate is then applied to the first and only qubit with the circuit.h(0) command. The sampler then samples this circuit with the sampler.run(pub,shots=num_sample_bits) command. Shots is how many times this circuit is sampled. Here, shots is num_sample_bits which is 64 so this circuit is sampled 64 times.

3.3 Function: randint

```
     """
    Generates a random integer between low(inclusive) and high (exclusive)

    Args:
        low (int): lowest possible value(inclusive)
        high (int): highest possible value(exclusive)

    Returns:
        int: Random integer
    """

    rand_bits = randbits()

    rand_int = 0
    i = 0
    for bit in rand_bits:

        rand_int |= bit << i
        i += 1

    rand_int = int(_map_value(low, high, rand_int))

    return rand_int
```

This function converts the 64-bit binary value to an integer. The resulting value of the integer from the original binary representation can be found by utilizing equation (4). The output of this function is the value resulting from converting the 64-bit value obtained from repeatedly measuring the probabilistic qubit multiple times to an integer.

3.4 Function: rand()

```
def rand() -> float:
    """
    Creates a random float between 0 and 1

    Returns:
        float: Random float between 0 and 1
    """

```

```
    divider = 2**_num_sample_bits

    rand_float = float(randint(0, 2**_num_sample_bits) / divider)

    return rand_float
```

This is the final pitstop. The integer obtained from randint() is divided by the value 2^{64}. This is done to normalize the integers to be in the range of values [0,1).The reason for this divisor value is because the largest integer which can be represented with 64 bits is $(2^{64} - 1)$, so in order to map the integers to a range of values between zero and one while excluding one, all integers are divided by 2^{64}. This function then results in a truly random output with a uniform distribution between zero and one(excluding 1).

3.5 Results

Several tests were done to verify the probabilistic nature of this code along the way. The plots and diagrams are shown and analyzed in this section to verify the probabilistic claims made about this code. First, the probabilities resulting from many measurements of the Qubit will be verified. Then the resulting probabilities after transforming the 64 measurement bits to an integer format will be verified. Lastly, the rand uniform distribution will be shown.

The first test done was running the quantum circuit shown in Figure 2 many times to verify the claim that the Hadamard gate will result in a fifty fifty probability qubit state. This test was done on the actual IBM Sherbrooke quantum computer. The histogram in Figure 4 shows that after 100000 runs, the number of times a $|0\rangle$ was measured and the number of times a $|1\rangle$ was measured are very, very similar. When normalized, these would show an estimate of the probability to be very close to fifty percent. There is more that could be done to analyze the error of this estimate, but this loosely shows that the probabilities associated with this quantum circuit are as expected.

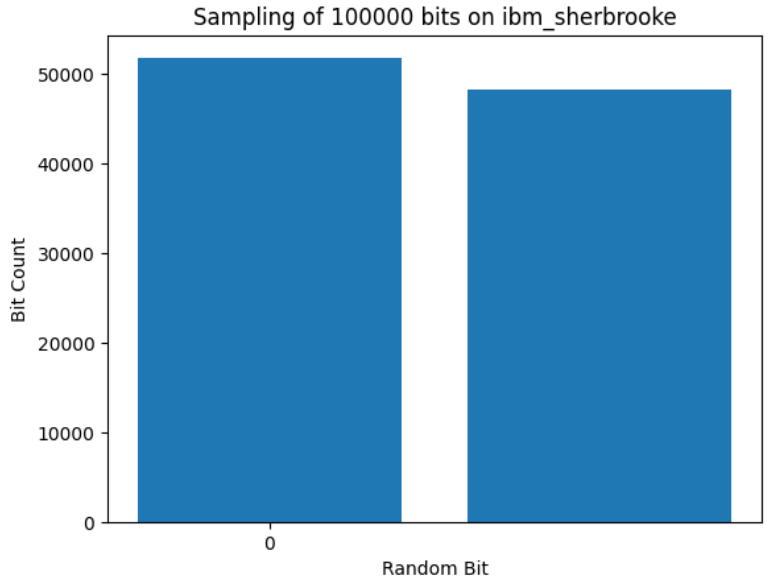

Figure 4: Many Measurment Qubit Histogram from IBM Quantum Computer

The next test to do is to show the distribution of the integers resulting from the randint function. This was done in simulation since it was very expensive to run on the IBM computer. It is expected that the probability of each integer be 6.25 percent since the probability is $(0.5)^4$ as derived before. Another way to rationalize this is that the probabilities must be equal so the total probability of one will be split evenly 16 ways. Equivalently, one could derive the probability to be $\frac{1}{16}$ for this reason. It can approximately be seen that the probability estimates from this test in Figure 5 are all approximately equal to this expected probability of 6.25 percent. Again, more analysis could be done on the expected variance of the probability estimates, but that is for another day.

Lastly, the distribution of the resulting rand() function will be shown. This was also done in simulation because it was very costly to run on the IBM computer. For this test, many random values were generated

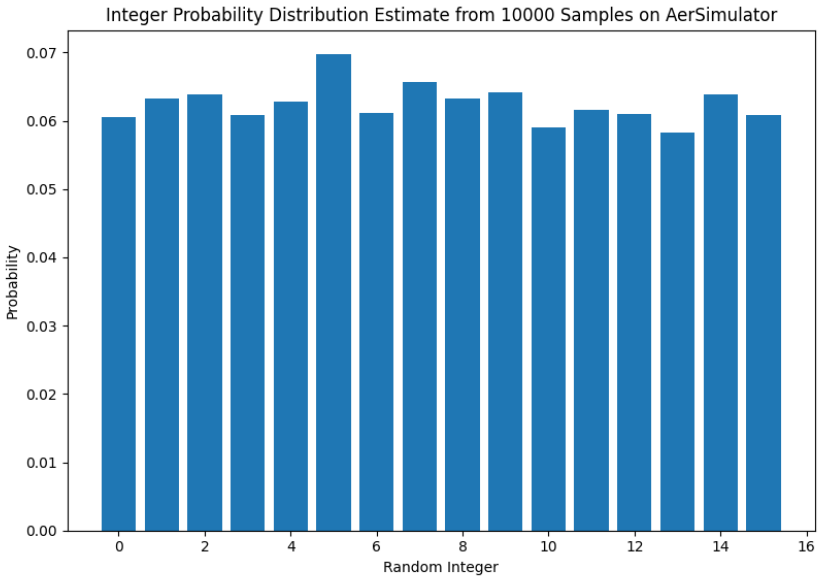

Figure 5: Integer Probability Distribution

from our quantum rand() function. These values were then plotted on a histogram in Figure 6. If the rand() function is truly uniform then it would be expected that the counts in each bucket be similar. This is what is seen in Figure 6. Once again, there is a lot more work that could be done to verify the certainty of this distribution, but the deadline is quickly approaching.

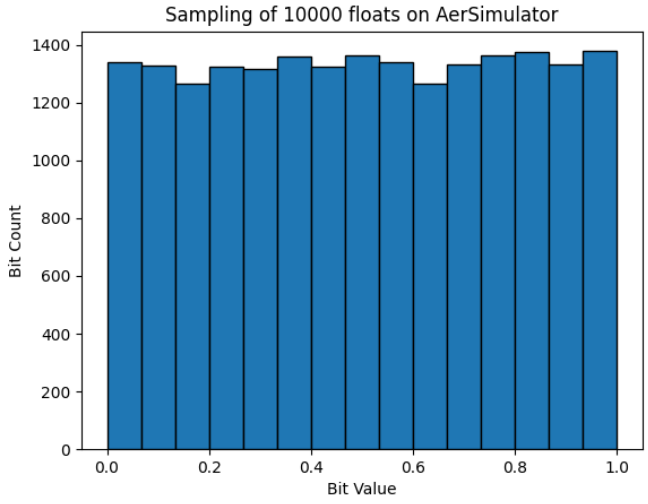

Figure 6: Histogram of Many Repetitions of the Quantum rand() Function

3.6 Future Improvements

Some further areas to look into with this challenge is looking into seeing how accurate the Hadamard gates are at transforming the quantum state as expected. It could be good to look into the effect of deviating rotations from the Hadamard on the resulting probability distribution by rand(). Additionally, there could be a lot more statistical rigor put into studying the certainty of the distributions found in this paper, given the limited sample size.

4 Conclusion

Overall, this hackathon challenge was enjoyable. It allowed us to be able to think critically about the basic principles that govern quantum computing. Theoretical results were derived to predict the expected results of the system. The system was then coded and histograms of results were plotted. The plots sufficiently matched the theory for heuristic verification by simulation. The bit probabilities were also verified not by simulation but by actually running them on the IBM quantum computer. This was very exciting!