CprE Quantum Computing Hackathon

Quantum Random Number Generation

Group: Dylan Brehm, and Josh Deaton

Challenge Goal and Presentation Timeline

Probability Theory

Quantum Gate Formulation

Code

Results

Probability Theory

- Define the sample space for the measurement of a qubit

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

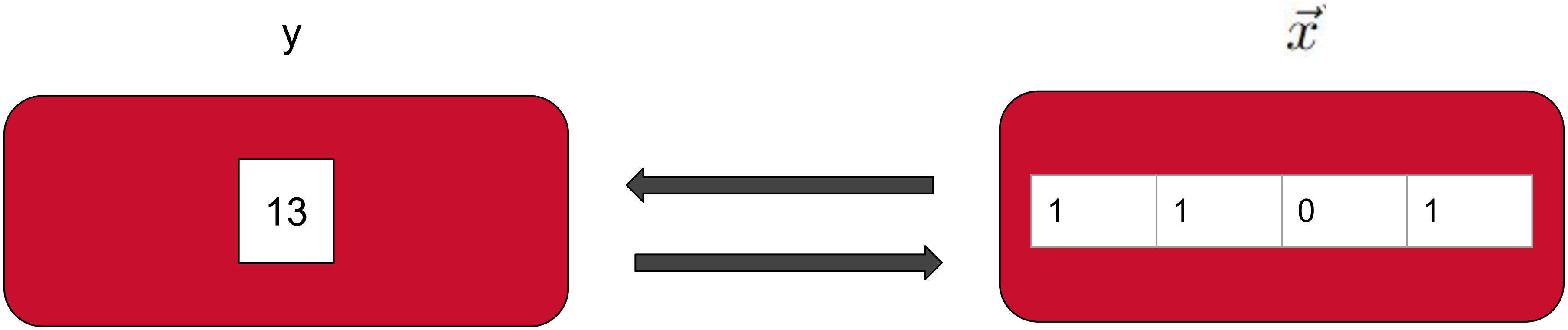
- An ordered list representing a binary number can be made using a series of measurements

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

- Define our decimal number as a number between 0 and 2^N-1

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

Probability Theory



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

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

Probability Theory

Two assumptions are made

- The Qubit measurements are independent
- The probability of measuring a zero and a one are equal at 50%

Independence

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

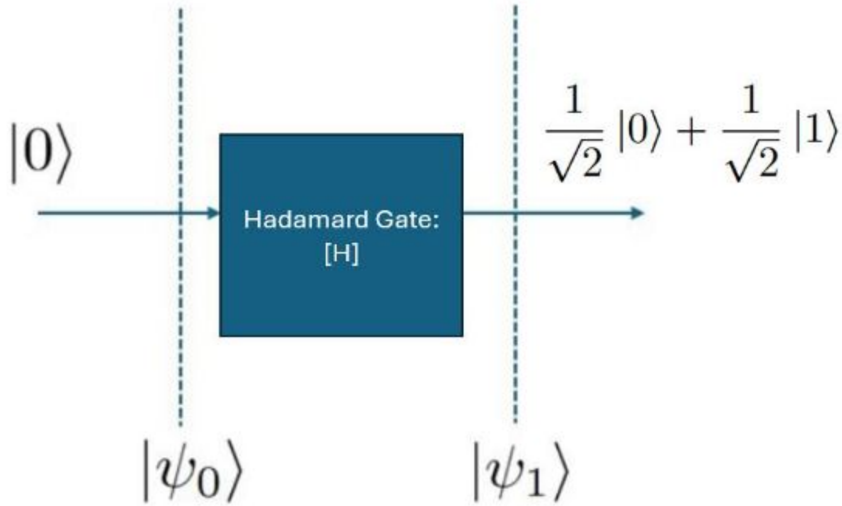
Equal Qubit Probability

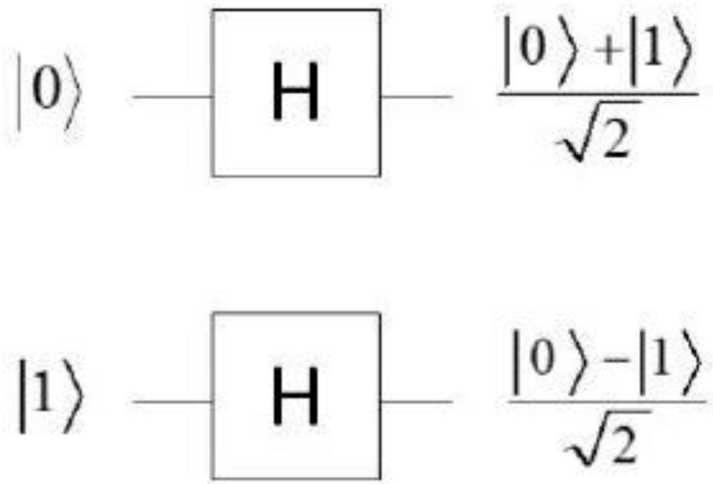
$$P_{X_i}(X_i = 1) = P_{X_i}(X_i = 0) = 0.5$$

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

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

Implementation

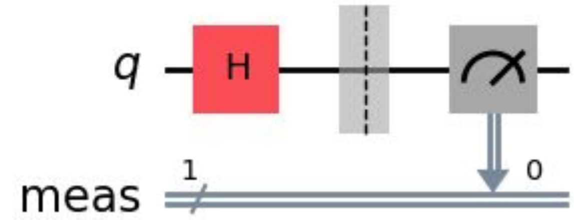




Implementation

- By implementing a 1 qubit hadamard gate and measurement in qiskit the theoretical design has been created

```
def _init_circuit():
    global _circuit

    _circuit = QuantumCircuit(1)

    _circuit.h(0)

    _circuit.measure_all()

    _circuit.draw(output="mpl")

    plt.show()

    pm = generate_preset_pass_manager(optimization_level=3, backend=_backend)

    _circuit = pm.run(_circuit)
```



Implementation

- A series of random bits can be generated by performing the sampler any number of times

- For integer and double generation, a mapping can be performed from 2^64 possible values to the desired number of buckets.

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

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

    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
```

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

    Returns:
        float: Random float between 0(inclusive) and 1(exclusive)
    """

    divider = 2**_num_sample_bits

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

    return rand_float
```

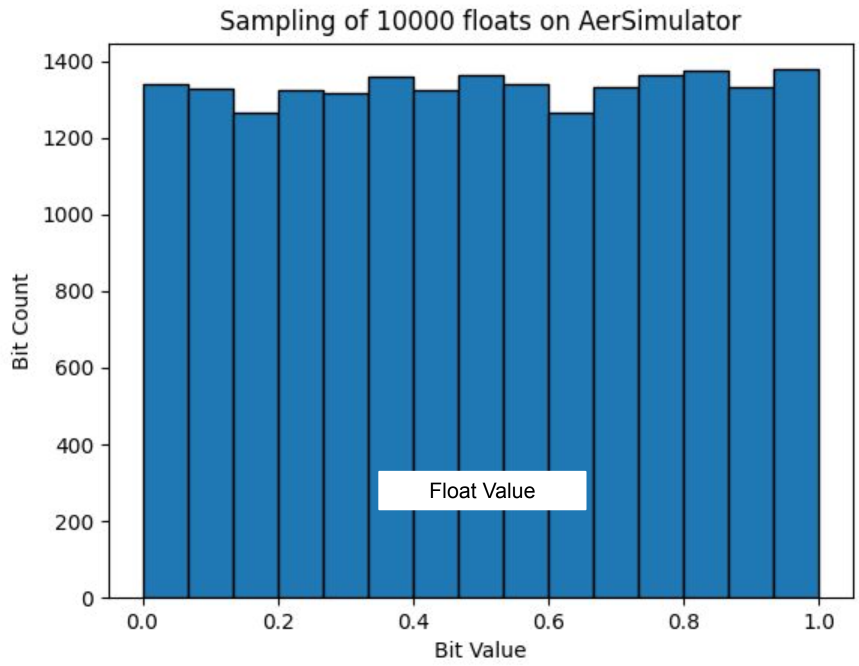
Implementation

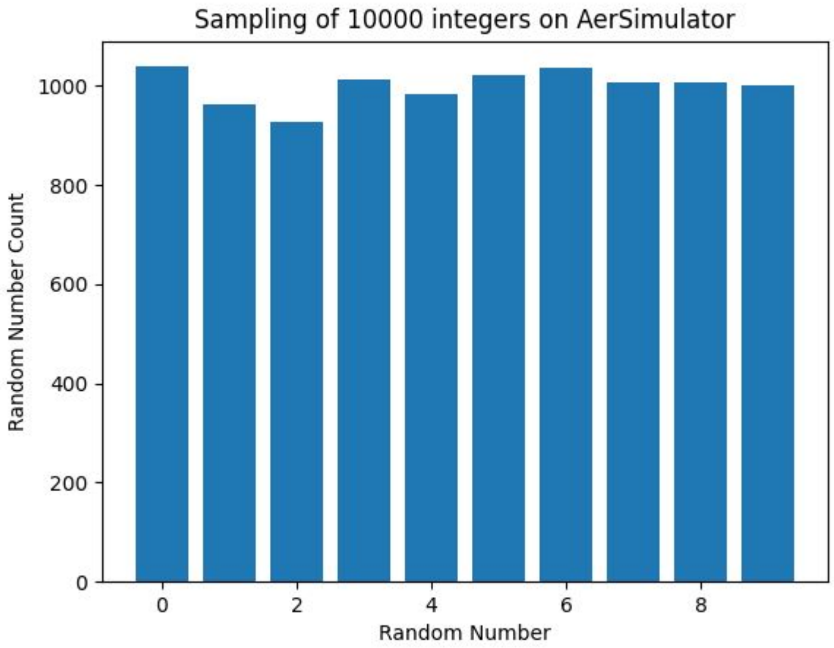
- The mapping can create a small error for non-factors of 2 as the 2^64 values is not evenly divided into the range

- However, using 2^64 bits mitigates this. For example, a user requesting 2000 different values will have a max probability difference of ~5*10^-20. This is much less then the error introduced by the quantum computer itself.

```python
def randint(low : int, high : int) -> int:
    """
    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
```

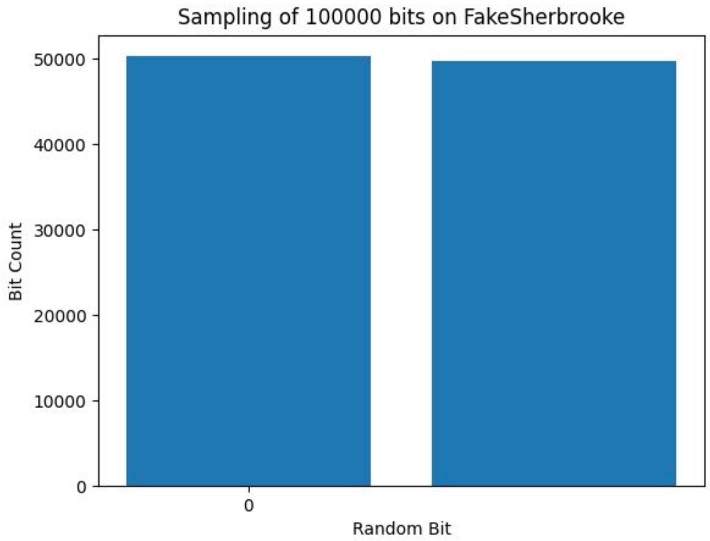
Simulation Results

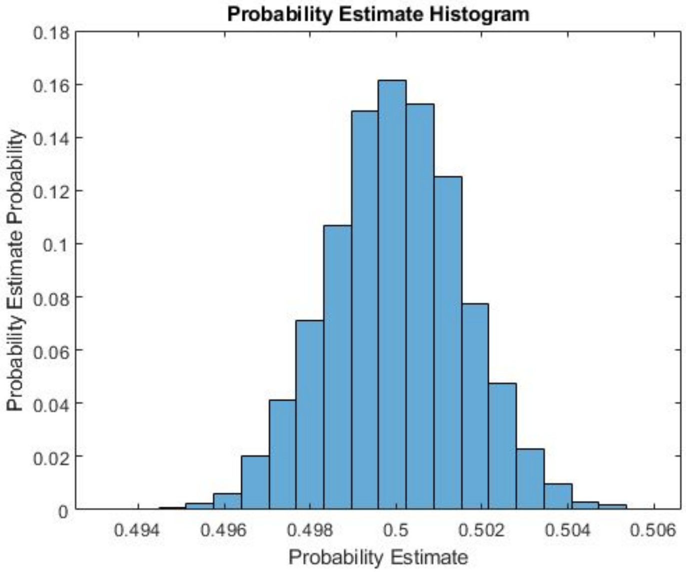


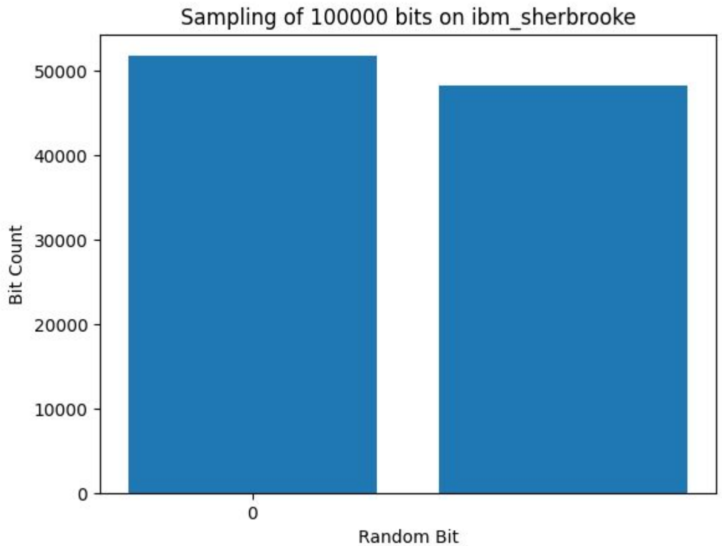


Hardware Results

- Estimated probability of getting a 0 on hardware from 100000 samples is 51.7%
- The histogram of probability estimates for 100000 sample experiments shows this is highly unlikely
- There is error in the hadamard gate







Conclusion

- It was an enjoyable exercise in quantum computing that allowed us to work on our fundamentals in Qiskit and probability
- After seeing our hardware results, it is clear that error mitigation is needed
- It would be cool to test how small deviations from equal probability would affect our results
- Optimization could be done in the qrand integer and float algorithm as only the bit results could be produced by the hardware in our allotted free 10 minutes

Questions?