Manually testing the properties:

Testing with input qubits in states “0111” (7 denary), 10,000 shots:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Implementation Language | + | i | - | -i | Phase result from R code |
| Qiskit – Qubit 0 | 0 | 2487 | 5000 | 2513 | 180 |
| Qubit 1 | 2530 | 0 | 2470 | 5000 | 270 |
| Qubit 2 | 4236 | 717 | 764 | 4283 | 314.5041 |
| Qubit 3 | 189 | 3413 | 4811 | 1587 | 157.5781 |
| CirQ – Qubit 0 | 0 | 2569 | 5000 | 2431 | 180 |
| Qubit 1 | 2509 | 0 | 2491 | 5000 | 270 |
| Qubit 2 | 4283 | 765 | 717 | 4235 | 315.4959 |
| Qubit 3 | 177 | 3503 | 4823 | 1497 | 158.3105 |
| Q# – Qubit 0 | 0 | 2495 | 5000 | 2505 | 180 |
| Qubit 1 | 2483 | 0 | 2517 | 5000 | 270 |
| Qubit 2 | 4270 | 763 | 730 | 4237 | 315.0724 |
| Qubit 3 | 198 | 3515 | 4802 | 1485 | 157.0433 |

“0111” QFT state vector result visualization from the Qiskit code: Compare it to resulting phase from the table - Assuming an arrow towards “x” in the diagram is 0

Diagram

Description automatically generated with medium confidence

**Property 1: Inverse QFT, will always return the original qubit states**

|  |  |  |
| --- | --- | --- |
| Qiskit | CirQ | Q# |
| def qft\_dagger(qc, n):  for j in range(n):  for m in range(j):  qc.cp(-math.pi/float(2\*\*(j-m)), m, j)  qc.h(j) | def qft\_dagger\_cirq(qc, qubits, n):  for j in range(n):  for m in range(j):  qc.append((cirq.CZ\*\*(-1/2\*\*(j-m)))(qubits[m],qubits[j]))  qc.append(cirq.H(qubits[j])) | operation qft\_dagger\_qsharp(qubits : Qubit[], n: Int) : Unit is Adj + Ctl {  for j in 0..n-1 {  for m in 0..j-1 {  let divisor = PowD(2.0, IntAsDouble(j-m));  (Controlled R1)([qubits[m]], ((-PI()/divisor), qubits[j]));  }  H(qubits[j]);  }  } |

Use this code to define a function that applies the inverse of QFT, apply some X gates at different registers and check that the output qubits, resemble where you placed the X gates.

Shape

Description automatically generated with low confidence

Every outputted result should be the same, no matter how many shots you take.



Try this across the three different languages, with these code snippets, if you use the code provided in the Github it will work.

If you try to implement QFT yourself, keep in mind if you apply SWAP gates at the end of the normal QFT, you will have to apply them again in the QFT inverse circuit.

**Property 2: The equation for the phase is where ‘’ is the real value in qubits, and N is the position of the qubit (starting from 1)**

Using this property, we can calculate the expected phase that should be applied each qubit from the QFT algorithm.

Qubit 0:

Qubit 1:

Qubit 2:

Qubit 3:

|  |  |  |
| --- | --- | --- |
| Implementation Language | Phase result from R code | Multiple amount |
| Qiskit – Qubit 0 | 180 | =180 |
| Qubit 1 | 270 | =270 |
| Qubit 2 | 314.5041 | ≈315 |
| Qubit 3 | 157.5781 | ≈157.5 |
| CirQ – Qubit 0 | 180 | =180 |
| Qubit 1 | 270 | =270 |
| Qubit 2 | 315.4959 | ≈315 |
| Qubit 3 | 158.3105 | ≈157.5 |
| Q# – Qubit 0 | 180 | =180 |
| Qubit 1 | 270 | =270 |
| Qubit 2 | 315.0724 | ≈315 |
| Qubit 3 | 157.0433 | ≈157.5 |

**Property 3: A simpler property that can be used to do with phase, no matter the qubit value, we know that the phase for each qubit must be a multiple of certain amount of c c**

Qubit 0: , needs to be a multiple of 180

Qubit 1:= , needs to be a multiple of 90

Qubit 2:, needs to be a multiple of 45

Qubit 3: *,* needs to be a multiple of 22.5

We can also compare these with the table at the top of the document.

|  |  |  |
| --- | --- | --- |
| Implementation Language | Phase result from R code | Multiple amount |
| Qiskit – Qubit 0 | 180 | ≈1 x 180 |
| Qubit 1 | 270 | ≈3 x 90 |
| Qubit 2 | 314.5041 | ≈7 x 45 |
| Qubit 3 | 157.5781 | ≈7 x 22.5 |
| CirQ – Qubit 0 | 180 | ≈1 x 180 |
| Qubit 1 | 270 | ≈3 x 90 |
| Qubit 2 | 315.4959 | ≈7 x 45 |
| Qubit 3 | 158.3105 | ≈7 x 22.5 |
| Q# – Qubit 0 | 180 | ≈1 x 180 |
| Qubit 1 | 270 | ≈3 x 90 |
| Qubit 2 | 315.0724 | ≈7 x 45 |
| Qubit 3 | 157.0433 | ≈7 x 22.5 |

Now, the calculated phase from the R code will not be exact, so far with 10,000 shots it seems to be accurate up to ±1, though further testing is needed.

**Property 4: If the inputted qubits are all 0, then the phase of all qubits should be 0 c**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Implementation Language | + | i | - | -i | Phase result from R code |
| Qiskit – Qubit 0 | 0 | 90.77925 | 180 | 89.22075 | 0 |
| Qubit 1 | 0 | 89.72498 | 180 | 90.27502 | 0 |
| Qubit 2 | 0 | 89.44995 | 180 | 90.55005 | 0 |
| Qubit 3 | 0 | 90.20627 | 180 | 89.79373 | 0 |
| CirQ – Qubit 0 | 0 | 90.29794 | 180 | 89.70206 | 0 |
| Qubit 1 | 0 | 91.19184 | 180 | 88.80816 | 0 |
| Qubit 2 | 0 | 89.81665 | 180 | 90.18335 | 0 |
| Qubit 3 | 0 | 89.6333 | 180 | 90.3667 | 0 |
| Q# – Qubit 0 | 0 | 88.9457 | 180 | 91.0543 | 0 |
| Qubit 1 | 0 | 89.49579 | 180 | 90.50421 | 0 |
| Qubit 2 | 0 | 89.56455 | 180 | 90.43545 | 0 |
| Qubit 3 | 0 | 90.48129 | 180 | 89.51870 | 0 |

We can see that with these 10,000 shots runs across all three languages, that if we use input the ‘0000’ state, that no rotation will be applied.

**Property 5: The LSB qubit will always be less than**

For our ‘0111’ case in the top of this document, the phase is between 157.04 – 158.31, which is less than 360.

For our ‘0000’ case, the phase for the last qubit is 0, this is due to how the R program calculates the phase.

This property should hold up no matter what qubit binary value is (‘11111…’), or how many qubits are used in the QFT circuit.