**Superdense Coding**

Superdense coding aims to send an encoded qubit to another party to transmit 2 classical bits of information.

This process is often called the inverse of quantum teleportation, where you send 2 classical bits of information, to transmit a qubit’s quantum state into another qubit.

**Properties:**

Postcondition testing

* Check that the measurement in the Z basis after encoding, returns the output of one of the bell states. We expect to measure the output sets to be either, {11,00} and {01,10}, for each message we encode.
* Check that the decoded message is equal to the message that was encoded

**Side-by-side implementation examples**

|  |  |  |
| --- | --- | --- |
| **Qiskit** | **Cirq** | **Q#** |
|  |  |  |
| Place the qubits in a bell state, this is a well-known method. First use a Hadamard gate, then a controlled not gate with the controlled qubit being the same qubit that had the Hadamard operation on it. | **🡨**  The same, but we pass in the qubit objects into the function. | **🡨**  We apply the ‘Controlled’ functor here to make the CX gate. Though you can also use the CNOT gate to the same effect. |
|  |  |  |
| Define a function to encode a message, into target qubit.  The input quantum circuit should be the bell state circuit.  The message parameter, is a string containing the two bits we intend to encode. | 🡨 | 🡨  Message here is an array of ints instead of a string, to make it easier to work with in Q#.  We only need the qubits to apply the operations to. |
|  |  |  |
| If a message that is not made up of ‘1’ or ‘0’ is passed in, throw an exception | 🡨 | 🡨 |
|  |  |  |
| Apply X and Z to the target qubit according to the message in order to encode the message. | 🡨 | 🡨 |
|  |  |  |
| Decoding the message consists of reversing the bell state.  We apply the bell state circuit in reverse order. | 🡨 | 🡨 |
|  |  |  |
| Create the bell state circuit, to get the two entangled qubits necessary for this protocol. | 🡨  We also create an array of qubit objects, which we need to create the bell pair in cirq. | 🡨  As in cirq, we need to create an array of qubits, but we do not need a quantum circuit.  We are also running in a separate operation, like a main method (which is called by the C# host). |
|  |  |  |
| We create encode the message ’11’ into the circuit.  The message can be changed to whatever 2 bits are needed. | 🡨  We pass in the second qubit object from the qubit array into the encode message method, instead of just an int signifying what regiser in the circuit. | 🡨  Passing the qubit object like cirq, but there is no need for a circuit. |
|  |  |  |
| In the protocol, the qubit that has been encoded would be sent to the other party.  Now assuming the other party has received the qubit, they would decode the qubit and measure it. | 🡨  Need to specify the qubit objects to measure/decode | 🡨  Need to specify the qubit objects to measure/decode |
|  |  |  |
| Set up the backend to run the simulations and record the result (which should be the same as the encoded message) | 🡨 | No need to set up the backend, we only need to reset the qubits to the |0 state after they have been measured. |
|  |  |  |
|  |  | Simple host C# program that runs the main Q# operation. |