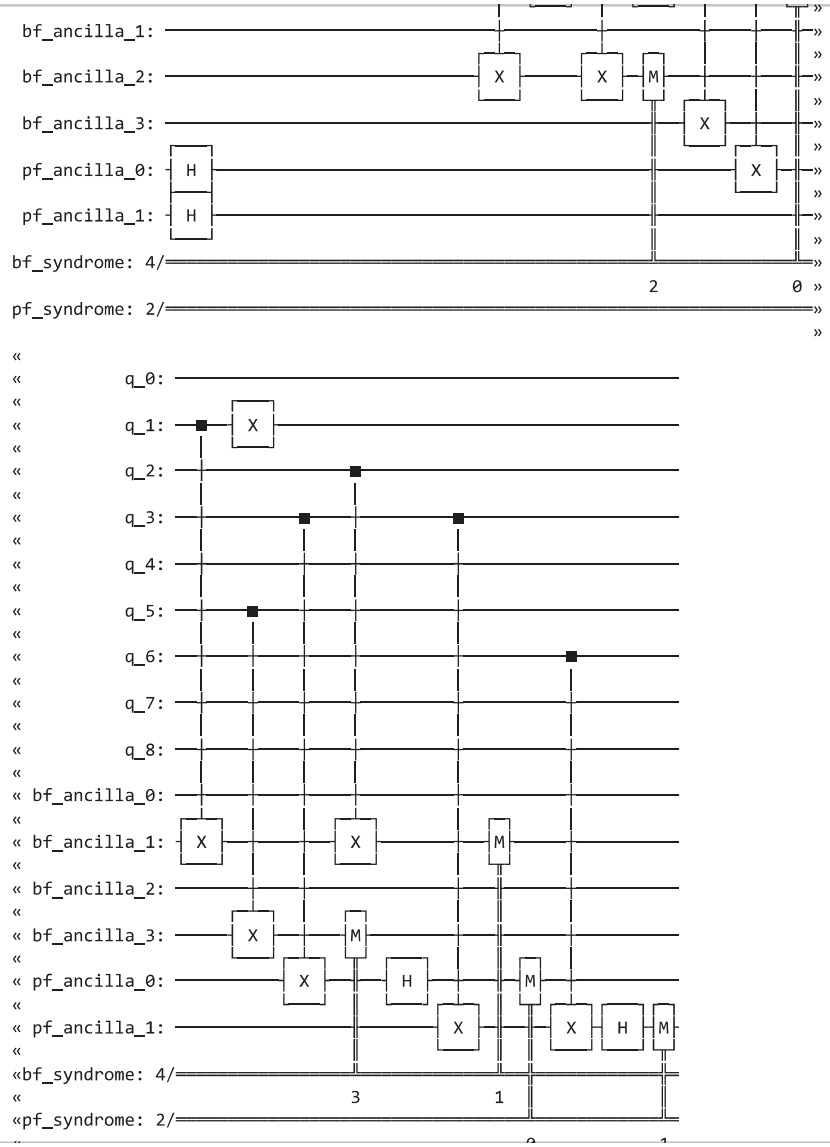
Executive Summary

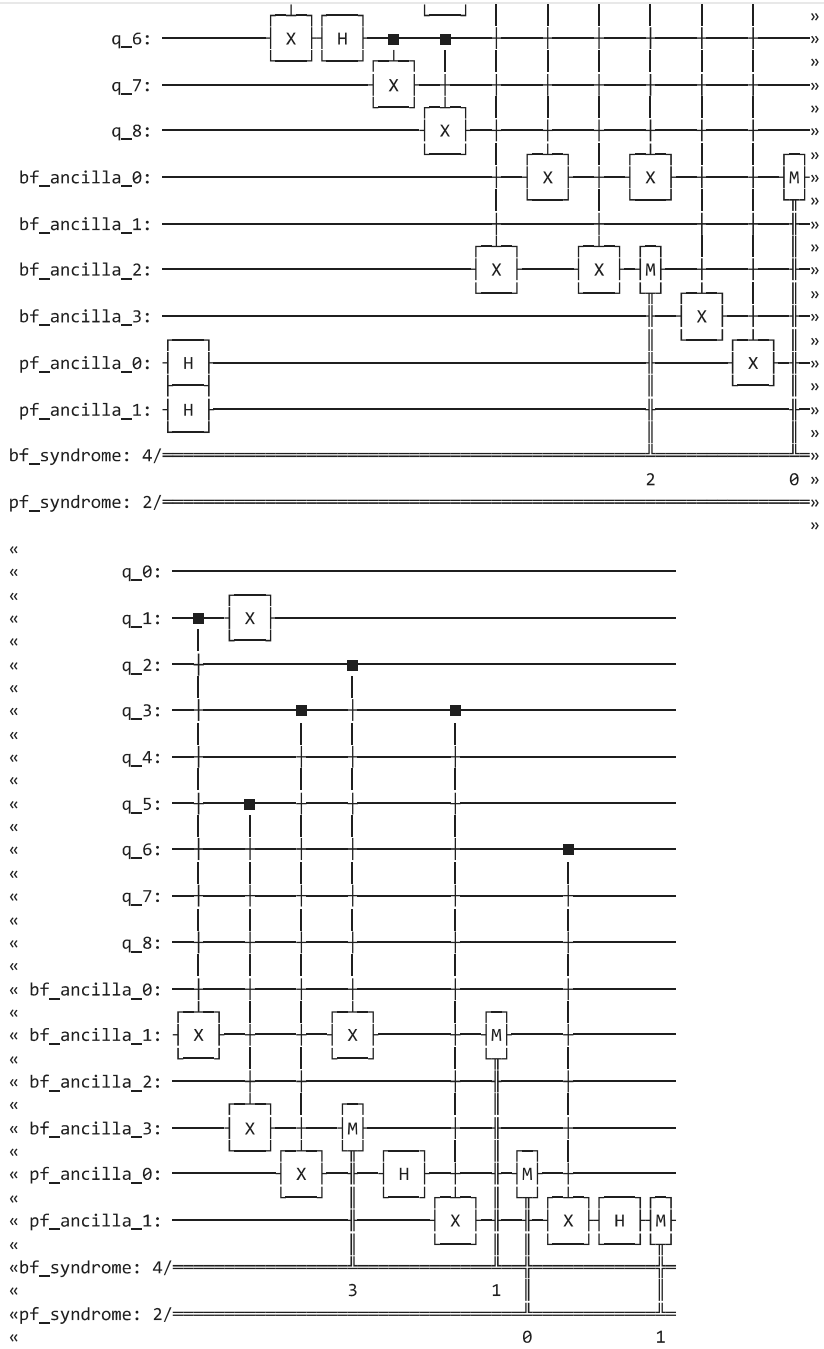
- **Problem Solved**: This project successfully addressed the critical challenge of protecting fragile quantum information from environmental noise by implementing and demonstrating the Shor 9-qubit quantum error correction code.
- **Quantum Advantage**: The Shor code harnesses unique quantum mechanical principles to simultaneously detect and correct both bit-flip and phase-flip errors, a capability essential for building fault-tolerant quantum computers and unattainable by classical error correction methods alone.
- **Key Result**: The developed pipeline accurately identified and corrected a single-qubit bit-flip (X) error introduced into the encoded logical qubit, validating the functionality of the syndrome measurement and correction mechanism within the Shor code framework.
- **Portfolio Impact**: This work showcases strong foundational skills in quantum circuit design and implementation using Qiskit, simulation of quantum error correction protocols, and practical application of quantum computing concepts in a noisy environment, demonstrating problem-solving abilities within a complex quantum system context.

```
from qiskit_aer import AerSimulator

# a. Initialize a new quantum circuit for the error correction process
ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

# b. Compose the shor_code_qc (encoding part) onto the new circuit
ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

# c. Introduce an example error using the introduce_error function
# Let's introduce an X error on qubit 1 as an example
error_qubit_index = 1
error_type_introduced = 'X'
# We apply the error directly to the data_qr in ec_qc
ec_qc.x(data_qr[error_qubit_index])
print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

# d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
ec_qc = add_syndrome_measurement(ec_qc)

# e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
simulator = AerSimulator()
compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
counts = result.get_counts(compiled_circuit)

# For simplicity, we'll take the first (and only) measurement result
measured_syndrome = list(counts.keys())[0]

# The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
pf_syndrome_result = measured_syndrome.split(' ')[0]
bf_syndrome_result = measured_syndrome.split(' ')[1]

print(f"Raw syndrome measurement result: {measured_syndrome}")
print(f"Bit-flip syndrome: {bf_syndrome_result}")
print(f"Phase-flip syndrome: {pf_syndrome_result}")

# g. Use the decode_syndrome function to determine the error location and type.
error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

# h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
# Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
if error_type is not None and qubit_to_correct is not None:
    ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
else:
    print("No correctable error detected or identified by the decoder.")

# i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
print(ec_qc.draw('text', idle_wires=False))
```



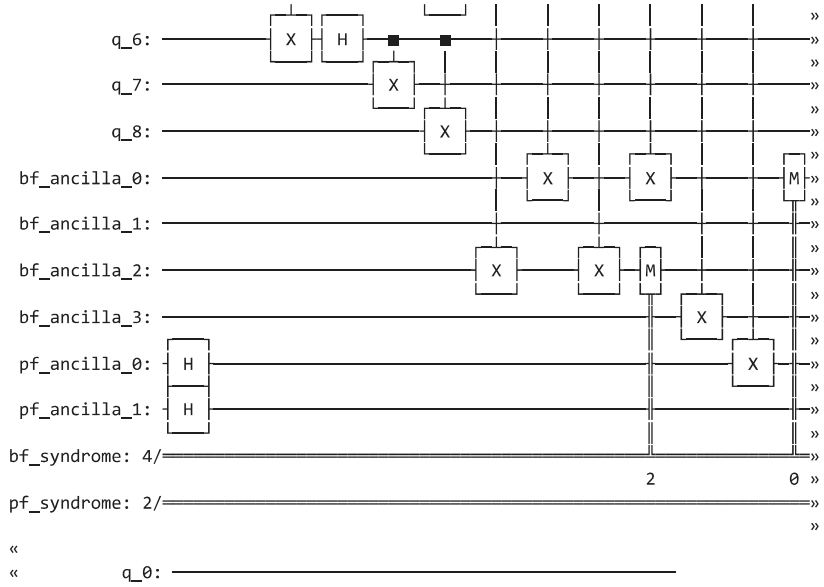

```
from qiskit_aer import AerSimulator

# a. Initialize a new quantum circuit for the error correction process
ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

# b. Compose the shor_code_qc (encoding part) onto the new circuit
ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

# c. Introduce an example error using the introduce_error function
# Let's introduce an X error on qubit 1 as an example
error_qubit_index = 1
error_type_introduced = 'X'
# We apply the error directly to the data_qr in ec_qc
ec_qc.x(data_qr[error_qubit_index])
print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

# d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
ec_qc = add_syndrome_measurement(ec_qc)

# e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
simulator = AerSimulator()
compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
counts = result.get_counts(compiled_circuit)

# For simplicity, we'll take the first (and only) measurement result
measured_syndrome = list(counts.keys())[0]

# The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
pf_syndrome_result = measured_syndrome.split(' ')[0]
bf_syndrome_result = measured_syndrome.split(' ')[1]

print(f"Raw syndrome measurement result: {measured_syndrome}")
print(f"Bit-flip syndrome: {bf_syndrome_result}")
print(f"Phase-flip syndrome: {pf_syndrome_result}")
```

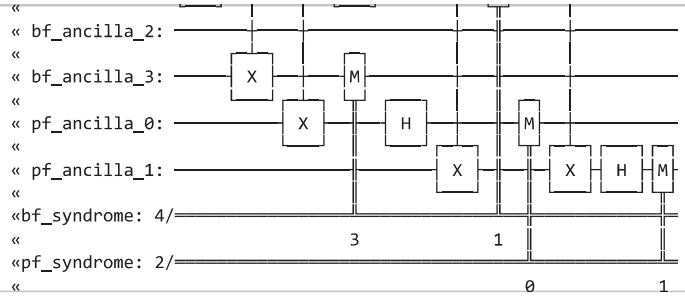
```
# g. Use the decode_syndrome function to determine the error location and type.
error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

# h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
# Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
if error_type is not None and qubit_to_correct is not None:
    ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
else:
    print("No correctable error detected or identified by the decoder.")

# i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
print(ec_qc.draw('text', idle_wires=False))
```




```
from qiskit_aer import AerSimulator

# a. Initialize a new quantum circuit for the error correction process
ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

# b. Compose the shor_code_qc (encoding part) onto the new circuit
ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

# c. Introduce an example error using the introduce_error function
# Let's introduce an X error on qubit 1 as an example
error_qubit_index = 1
error_type_introduced = 'X'
```

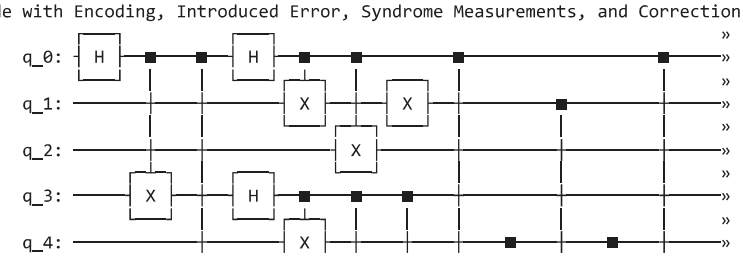
```
# We apply the error directly to the data_qr in ec_qc
ec_qc.x(data_qr[error_qubit_index])
print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

# d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
ec_qc = add_syndrome_measurement(ec_qc)

# e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
simulator = AerSimulator()
compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
counts = result.get_counts(compiled_circuit)

# For simplicity, we'll take the first (and only) measurement result
measured_syndrome = list(counts.keys())[0]

# The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
pf_syndrome_result = measured_syndrome.split(' ')[0]
bf_syndrome_result = measured_syndrome.split(' ')[1]

print(f"Raw syndrome measurement result: {measured_syndrome}")
print(f"Bit-flip syndrome: {bf_syndrome_result}")
print(f"Phase-flip syndrome: {pf_syndrome_result}")

# g. Use the decode_syndrome function to determine the error location and type.
error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

# h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
# Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
if error_type is not None and qubit_to_correct is not None:
    ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
else:
    print("No correctable error detected or identified by the decoder.")

# i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
print(ec_qc.draw('text', idle_wires=False))
```

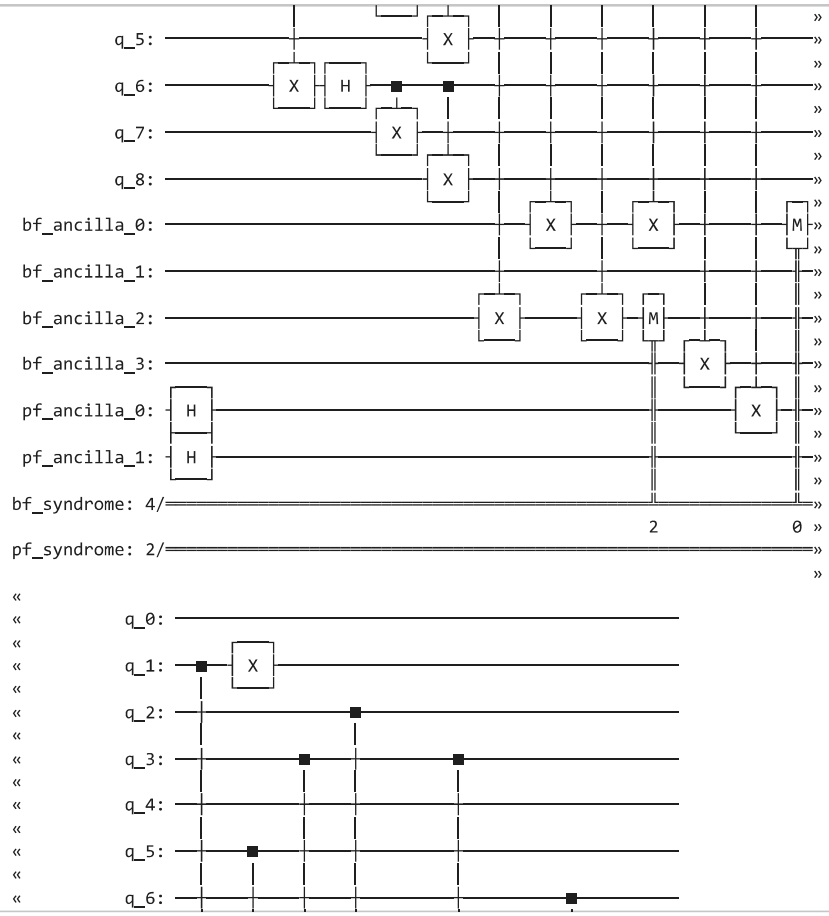





```
from qiskit_aer import AerSimulator

# a. Initialize a new quantum circuit for the error correction process
ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

# b. Compose the shor_code_qc (encoding part) onto the new circuit
ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

# c. Introduce an example error using the introduce_error function
# Let's introduce an X error on qubit 1 as an example
error_qubit_index = 1
error_type_introduced = 'X'
# We apply the error directly to the data_qr in ec_qc
ec_qc.x(data_qr[error_qubit_index])
print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

# d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
ec_qc = add_syndrome_measurement(ec_qc)

# e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
simulator = AerSimulator()
compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
counts = result.get_counts(compiled_circuit)

# For simplicity, we'll take the first (and only) measurement result
measured_syndrome = list(counts.keys())[0]

# The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
pf_syndrome_result = measured_syndrome.split(' ')[0]
bf_syndrome_result = measured_syndrome.split(' ')[1]

print(f"Raw syndrome measurement result: {measured_syndrome}")
print(f"Bit-flip syndrome: {bf_syndrome_result}")
print(f"Phase-flip syndrome: {pf_syndrome_result}")

# g. Use the decode_syndrome function to determine the error location and type.
error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

# h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
# Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
if error_type is not None and qubit_to_correct is not None:
    ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
else:
    print("No correctable error detected or identified by the decoder.")

# i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
print(ec_qc.draw('text', idle_wires=False))
```

```
Introduced X error on data qubit 1
Raw syndrome measurement result: 00 0011
Bit-flip syndrome: 0011
Phase-flip syndrome: 00
Decoded error: Type=X, Qubit=1
Applied bit-flip (X) correction on qubit 1

Shor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:
```

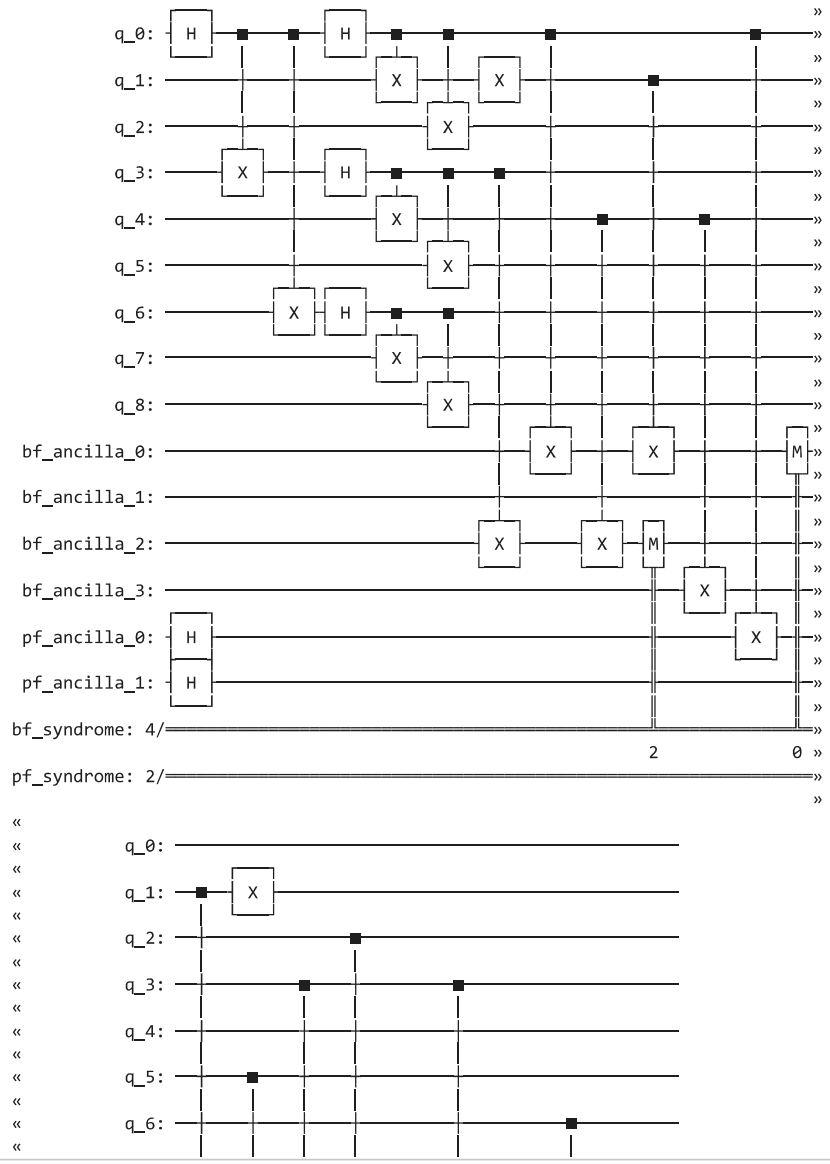





```
from qiskit_aer import AerSimulator

# a. Initialize a new quantum circuit for the error correction process
ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

# b. Compose the shor_code_qc (encoding part) onto the new circuit
ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

# c. Introduce an example error using the introduce_error function
# Let's introduce an X error on qubit 1 as an example
error_qubit_index = 1
error_type_introduced = 'X'
# We apply the error directly to the data_qr in ec_qc
ec_qc.x(data_qr[error_qubit_index])
print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

# d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
ec_qc = add_syndrome_measurement(ec_qc)

# e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
simulator = AerSimulator()
compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
counts = result.get_counts(compiled_circuit)

# For simplicity, we'll take the first (and only) measurement result
measured_syndrome = list(counts.keys())[0]

# The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
pf_syndrome_result = measured_syndrome.split(' ')[0]
bf_syndrome_result = measured_syndrome.split(' ')[1]

print(f"Raw syndrome measurement result: {measured_syndrome}")
print(f"Bit-flip syndrome: {bf_syndrome_result}")
print(f"Phase-flip syndrome: {pf_syndrome_result}")

# g. Use the decode_syndrome function to determine the error location and type.
error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

# h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
# Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
if error_type is not None and qubit_to_correct is not None:
```

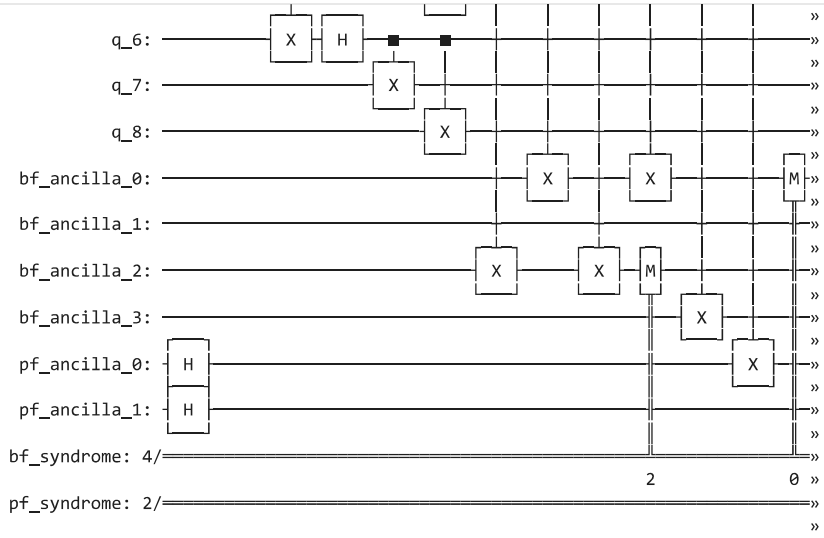
```
    ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
else:
    print("No correctable error detected or identified by the decoder.")

# i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
print(ec_qc.draw('text', idle_wires=False))
```

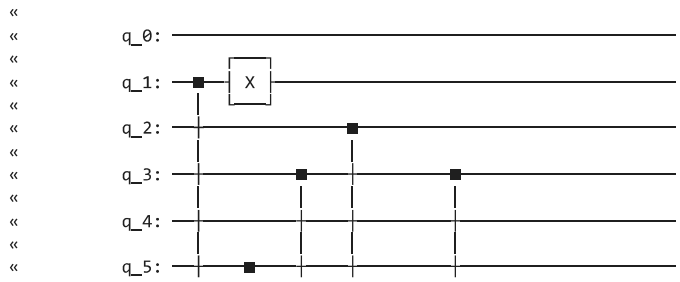
```
Introduced X error on data qubit 1
Raw syndrome measurement result: 00 0011
Bit-flip syndrome: 0011
Phase-flip syndrome: 00
Decoded error: Type=X, Qubit=1
Applied bit-flip (X) correction on qubit 1

Shor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:
```




```
from qiskit_aer import AerSimulator

# a. Initialize a new quantum circuit for the error correction process
ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

# b. Compose the shor_code_qc (encoding part) onto the new circuit
ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

# c. Introduce an example error using the introduce_error function
# Let's introduce an X error on qubit 1 as an example
error_qubit_index = 1
error_type_introduced = 'X'
# We apply the error directly to the data_qr in ec_qc
ec_qc.x(data_qr[error_qubit_index])
print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

# d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
ec_qc = add_syndrome_measurement(ec_qc)

# e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
simulator = AerSimulator()
```

```
simulator = AerSimulator()
compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
counts = result.get_counts(compiled_circuit)

# For simplicity, we'll take the first (and only) measurement result
measured_syndrome = list(counts.keys())[0]

# The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
pf_syndrome_result = measured_syndrome.split(' ')[0]
bf_syndrome_result = measured_syndrome.split(' ')[1]

print(f"Raw syndrome measurement result: {measured_syndrome}")
print(f"Bit-flip syndrome: {bf_syndrome_result}")
print(f"Phase-flip syndrome: {pf_syndrome_result}")

# g. Use the decode_syndrome function to determine the error location and type.
error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

# h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
# Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
if error_type is not None and qubit_to_correct is not None:
    ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
else:
    print("No correctable error detected or identified by the decoder.")

# i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
print(ec_qc.draw('text', idle_wires=False))
```

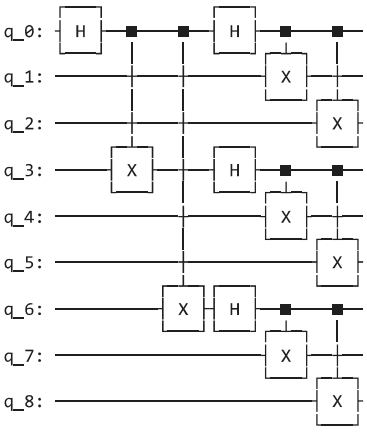




```
«                                3          1 ‖           ‖
«pf_syndrome: 2/═══════════════════════════════╩═══════════╩═
«                                              0           1
```

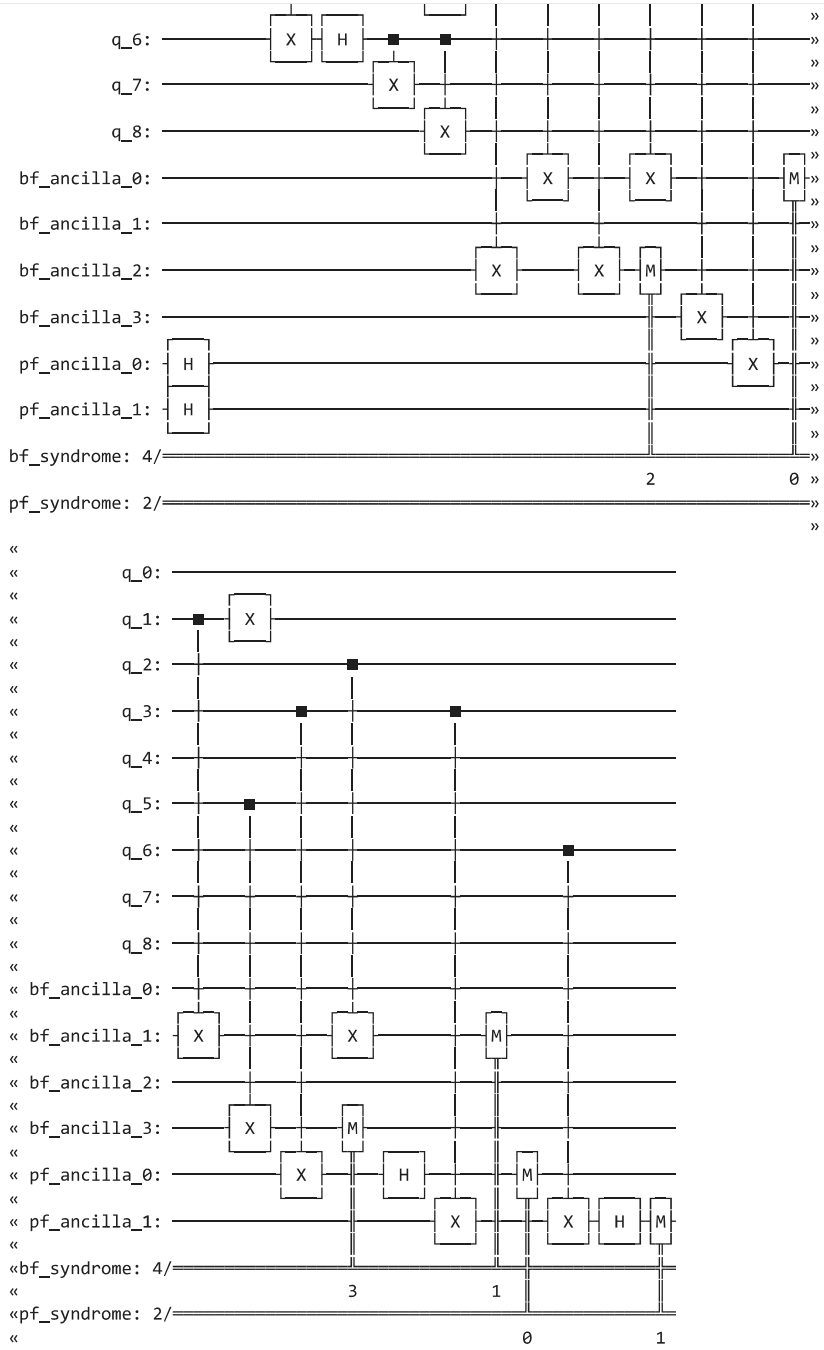
```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister

# Create a QuantumCircuit object with 9 quantum registers
shor_code_qc = QuantumCircuit(9, name='Shor Code Encoding')

# Implement Shor encoding gates
# a. Apply a Hadamard gate (H) to qubit 0.
shor_code_qc.h(0);

# b. Apply CNOT gates from qubit 0 to qubits 3 and 6.
shor_code_qc.cx(0, 3);
shor_code_qc.cx(0, 6);

# c. Apply Hadamard gates (H) to qubits 0, 3, and 6.
shor_code_qc.h(0);
shor_code_qc.h(3);
shor_code_qc.h(6);

# d. Apply CNOT gates from qubit 0 to qubit 1 and qubit 2.
shor_code_qc.cx(0, 1);
shor_code_qc.cx(0, 2);

# e. Apply CNOT gates from qubit 3 to qubit 4 and qubit 5.
shor_code_qc.cx(3, 4);
shor_code_qc.cx(3, 5);

# f. Apply CNOT gates from qubit 6 to qubit 7 and qubit 8.
shor_code_qc.cx(6, 7);
shor_code_qc.cx(6, 8);

# Optionally draw the circuit to visualize
print("Shor Code Encoding Circuit:")
print(shor_code_qc.draw('text'))
```

```
Shor Code Encoding Circuit:
```



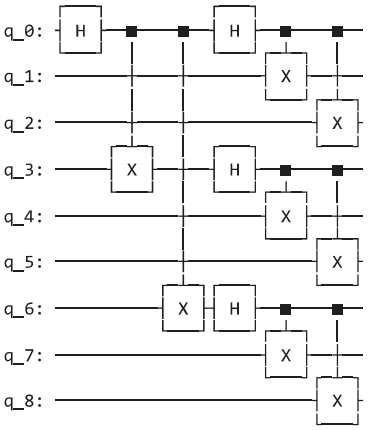

```
from qiskit_aer import AerSimulator

# a. Initialize a new quantum circuit for the error correction process
ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

# b. Compose the shor_code_qc (encoding part) onto the new circuit
ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

# c. Introduce an example error using the introduce_error function
# Let's introduce an X error on qubit 1 as an example
error_qubit_index = 1
error_type_introduced = 'X'
# We apply the error directly to the data_qr in ec_qc
ec_qc.x(data_qr[error_qubit_index])
print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

# d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
ec_qc = add_syndrome_measurement(ec_qc)

# e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
simulator = AerSimulator()
compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
counts = result.get_counts(compiled_circuit)
```

```
# For simplicity, we'll take the first (and only) measurement result
measured_syndrome = list(counts.keys())[0]

# The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
pf_syndrome_result = measured_syndrome.split(' ')[0]
bf_syndrome_result = measured_syndrome.split(' ')[1]

print(f"Raw syndrome measurement result: {measured_syndrome}")
print(f"Bit-flip syndrome: {bf_syndrome_result}")
print(f"Phase-flip syndrome: {pf_syndrome_result}")

# g. Use the decode_syndrome function to determine the error location and type.
error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

# h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
# Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
if error_type is not None and qubit_to_correct is not None:
    ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
else:
    print("No correctable error detected or identified by the decoder.")

# i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
print(ec_qc.draw('text', idle_wires=False))
```



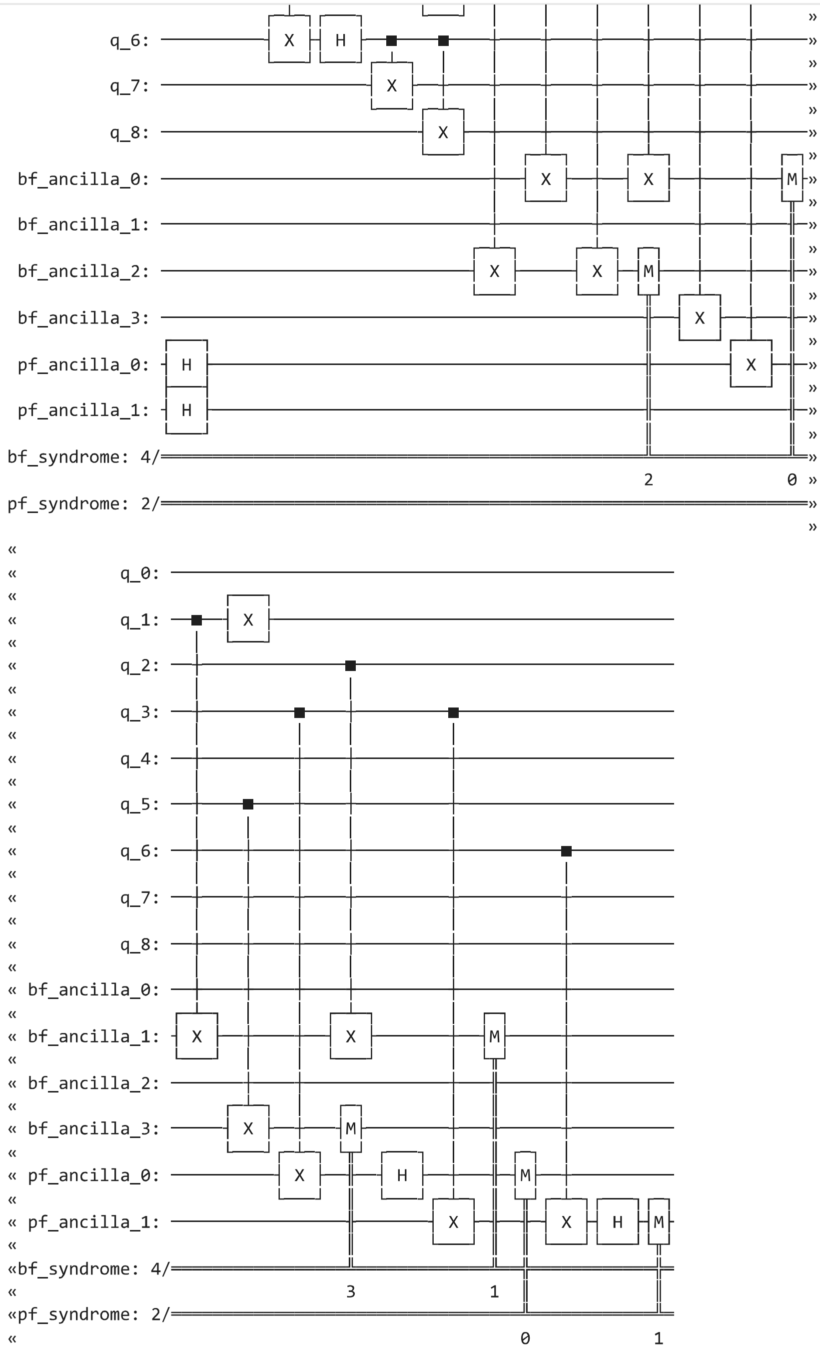
```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister

# Create a QuantumCircuit object with 9 quantum registers
shor_code_qc = QuantumCircuit(9, name='Shor Code Encoding')

# Implement Shor encoding gates
# a. Apply a Hadamard gate (H) to qubit 0.
shor_code_qc.h(0);

# b. Apply CNOT gates from qubit 0 to qubits 3 and 6.
shor_code_qc.cx(0, 3);
shor_code_qc.cx(0, 6);

# c. Apply Hadamard gates (H) to qubits 0, 3, and 6.
shor_code_qc.h(0);
shor_code_qc.h(3);
shor_code_qc.h(6);

# d. Apply CNOT gates from qubit 0 to qubit 1 and qubit 2.
shor_code_qc.cx(0, 1);
shor_code_qc.cx(0, 2);

# e. Apply CNOT gates from qubit 3 to qubit 4 and qubit 5.
shor_code_qc.cx(3, 4);
shor_code_qc.cx(3, 5);

# f. Apply CNOT gates from qubit 6 to qubit 7 and qubit 8.
shor_code_qc.cx(6, 7);
shor_code_qc.cx(6, 8);

# Optionally draw the circuit to visualize
print("Shor Code Encoding Circuit:")
print(shor_code_qc.draw('text'))
```

```
Shor Code Encoding Circuit:
```




```
from qiskit_aer import AerSimulator

# a. Initialize a new quantum circuit for the error correction process
ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

# b. Compose the shor_code_qc (encoding part) onto the new circuit
ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

# c. Introduce an example error using the introduce_error function
# Let's introduce an X error on qubit 1 as an example
error_qubit_index = 1
error_type_introduced = 'X'
# We apply the error directly to the data_qr in ec_qc
ec_qc.x(data_qr[error_qubit_index])
print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

# d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
ec_qc = add_syndrome_measurement(ec_qc)

# e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
simulator = AerSimulator()
compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
counts = result.get_counts(compiled_circuit)

# For simplicity, we'll take the first (and only) measurement result
measured_syndrome = list(counts.keys())[0]
```

```
# The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
pf_syndrome_result = measured_syndrome.split(' ')[0]
bf_syndrome_result = measured_syndrome.split(' ')[1]

print(f"Raw syndrome measurement result: {measured_syndrome}")
print(f"Bit-flip syndrome: {bf_syndrome_result}")
print(f"Phase-flip syndrome: {pf_syndrome_result}")

# g. Use the decode_syndrome function to determine the error location and type.
error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

# h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
# Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
if error_type is not None and qubit_to_correct is not None:
    ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
else:
    print("No correctable error detected or identified by the decoder.")

# i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
print(ec_qc.draw('text', idle_wires=False))
```




```
import sys
!{sys.executable} -m pip install qiskit
!{sys.executable} -m pip install qiskit-aer
```

```
!{sys.executable} -m pip install pylatexenc
print("Required Qiskit packages have been installed.")
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.12/dist-packages (2.3.0)
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.12/dist-packages (from qiskit) (0.17.1)
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/python3.12/dist-packages (from qiskit) (2.0.2)
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.12/dist-packages (from qiskit) (1.16.3)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.12/dist-packages (from qiskit) (0.3.8)
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.12/dist-packages (from qiskit) (5.6.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages (from qiskit) (4.15.0)
Requirement already satisfied: qiskit-aer in /usr/local/lib/python3.12/dist-packages (0.17.2)
Requirement already satisfied: qiskit>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (2.3.0)
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (2.0.2)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (1.16.3)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (5.9.5)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (2.9.0.pc
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.0->qiskit-aer)
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.12/dist-packages (from qiskit>=1.1.0->qiskit-aer)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.12/dist-packages (from qiskit>=1.1.0->qiskit-aer) (0.3.8)
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.12/dist-packages (from qiskit>=1.1.0->qiskit-aer)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages (from qiskit>=1.1.0->qiskit-aer)
Requirement already satisfied: pylatexenc in /usr/local/lib/python3.12/dist-packages (2.10)
Required Qiskit packages have been installed.
```

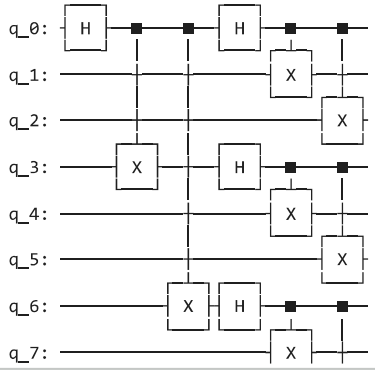
Implement 9-Qubit Quantum Error Correction Code

Subtask:

Define and implement the quantum circuit for encoding 1 logical qubit into 9 physical qubits (Shor code). This includes setting up the initial logical state and the encoding gates.

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister

# Create a QuantumCircuit object with 9 quantum registers
shor_code_qc = QuantumCircuit(9, name='Shor Code Encoding')

# Implement Shor encoding gates
# a. Apply a Hadamard gate (H) to qubit 0.
shor_code_qc.h(0);

# b. Apply CNOT gates from qubit 0 to qubits 3 and 6.
shor_code_qc.cx(0, 3);
shor_code_qc.cx(0, 6);

# c. Apply Hadamard gates (H) to qubits 0, 3, and 6.
shor_code_qc.h(0);
shor_code_qc.h(3);
shor_code_qc.h(6);

# d. Apply CNOT gates from qubit 0 to qubit 1 and qubit 2.
shor_code_qc.cx(0, 1);
shor_code_qc.cx(0, 2);

# e. Apply CNOT gates from qubit 3 to qubit 4 and qubit 5.
shor_code_qc.cx(3, 4);
shor_code_qc.cx(3, 5);

# f. Apply CNOT gates from qubit 6 to qubit 7 and qubit 8.
shor_code_qc.cx(6, 7);
shor_code_qc.cx(6, 8);

# Optionally draw the circuit to visualize
print("Shor Code Encoding Circuit:")
print(shor_code_qc.draw('text'))
```

```
Shor Code Encoding Circuit:
```



```
q_8: ──────────────────────┤ X ├
```

Introduce Single-Qubit Errors

Subtask:

Develop a mechanism to introduce various single-qubit errors (bit-flip, phase-flip, or both) on a selected physical qubit within the encoded state.

```
def introduce_error(circuit, error_type, qubit_index):
    """
    Introduces a single-qubit error (bit-flip, phase-flip, or both) to a specified qubit.

    Args:
        circuit (QuantumCircuit): The input quantum circuit.
        error_type (str): The type of error to introduce ('X' for bit-flip, 'Z' for phase-flip, 'Y' for bit-phase-flip).
        qubit_index (int): The index of the qubit on which to introduce the error.

    Returns:
        QuantumCircuit: The modified quantum circuit with the error introduced.
    """
    modified_circuit = circuit.copy() # Create a copy to avoid modifying the original circuit directly
    if error_type == 'X':
        modified_circuit.x(qubit_index) # Apply bit-flip error
        print(f"Introduced bit-flip (X) error on qubit {qubit_index}")
    elif error_type == 'Z':
        modified_circuit.z(qubit_index) # Apply phase-flip error
        print(f"Introduced phase-flip (Z) error on qubit {qubit_index}")
    elif error_type == 'Y':
        modified_circuit.y(qubit_index) # Apply bit-phase-flip error (Y = iXZ)
        print(f"Introduced bit-phase-flip (Y) error on qubit {qubit_index}")
    else:
        raise ValueError("Invalid error_type. Choose 'X', 'Z', or 'Y'.")
    return modified_circuit

print("The 'introduce_error' function has been defined.")
```

```
The 'introduce_error' function has been defined.
```

Perform Syndrome Measurements

Subtask:

Design and implement the quantum circuits for measuring the syndromes to detect bit-flip and phase-flip errors. This will involve ancilla qubits and CNOT gates, followed by measurement of the ancillas.

```
# Define the number of data qubits, ancilla qubits for bit-flip and phase-flip, and classical bits
NUM_DATA_QUBITS = 9
NUM_BF_ANCILLAS = 4  # For measuring Z_0Z_1, Z_1Z_2, Z_3Z_4, Z_4Z_5
NUM_PF_ANCILLAS = 2  # For measuring X_0X_3, X_3X_6

TOTAL_QUBITS = NUM_DATA_QUBITS + NUM_BF_ANCILLAS + NUM_PF_ANCILLAS
TOTAL_CLASSICAL_BITS = NUM_BF_ANCILLAS + NUM_PF_ANCILLAS

# Create quantum registers for data qubits and ancilla qubits
data_qr = QuantumRegister(NUM_DATA_QUBITS, 'q')
bf_ancilla_qr = QuantumRegister(NUM_BF_ANCILLAS, 'bf_ancilla')
pf_ancilla_qr = QuantumRegister(NUM_PF_ANCILLAS, 'pf_ancilla')

# Create classical registers for syndrome measurements
bf_c_reg = ClassicalRegister(NUM_BF_ANCILLAS, 'bf_syndrome')
pf_c_reg = ClassicalRegister(NUM_PF_ANCILLAS, 'pf_syndrome')


def add_syndrome_measurement(circuit):
    """
    Adds the syndrome measurement circuits for bit-flip and phase-flip errors
    to the given quantum circuit.

    Args:
        circuit (QuantumCircuit): The quantum circuit with encoded state and optional error.

    Returns:
        QuantumCircuit: The circuit with syndrome measurements added.
```

```
    """
    # Bit-flip syndrome measurement (Z_i Z_{i+1} checks)
    # Using ancillas bf_ancilla_qr[0] to bf_ancilla_qr[3] and classical bits bf_c_reg[0] to bf_c_reg[3]

    # Measure Z_0 Z_1 (data qubits 0 and 1) with bf_ancilla_qr[0]
    circuit.cx(data_qr[0], bf_ancilla_qr[0])
    circuit.cx(data_qr[1], bf_ancilla_qr[0])
    circuit.measure(bf_ancilla_qr[0], bf_c_reg[0])

    # Measure Z_1 Z_2 (data qubits 1 and 2) with bf_ancilla_qr[1]
    circuit.cx(data_qr[1], bf_ancilla_qr[1])
    circuit.cx(data_qr[2], bf_ancilla_qr[1])
    circuit.measure(bf_ancilla_qr[1], bf_c_reg[1])

    # Measure Z_3 Z_4 (data qubits 3 and 4) with bf_ancilla_qr[2]
    circuit.cx(data_qr[3], bf_ancilla_qr[2])
    circuit.cx(data_qr[4], bf_ancilla_qr[2])
    circuit.measure(bf_ancilla_qr[2], bf_c_reg[2])

    # Measure Z_4 Z_5 (data qubits 4 and 5) with bf_ancilla_qr[3]
    circuit.cx(data_qr[4], bf_ancilla_qr[3])
    circuit.cx(data_qr[5], bf_ancilla_qr[3])
    circuit.measure(bf_ancilla_qr[3], bf_c_reg[3])

    # Phase-flip syndrome measurement (X_i X_j checks)
    # Using ancillas pf_ancilla_qr[0] to pf_ancilla_qr[1] and classical bits pf_c_reg[0] to pf_c_reg[1]

    # Measure X_0 X_3 (data qubits 0 and 3) with pf_ancilla_qr[0]
    circuit.h(pf_ancilla_qr[0])
    circuit.cx(data_qr[0], pf_ancilla_qr[0])
    circuit.cx(data_qr[3], pf_ancilla_qr[0])
    circuit.h(pf_ancilla_qr[0])
    circuit.measure(pf_ancilla_qr[0], pf_c_reg[0])

    # Measure X_3 X_6 (data qubits 3 and 6) with pf_ancilla_qr[1]
    circuit.h(pf_ancilla_qr[1])
    circuit.cx(data_qr[3], pf_ancilla_qr[1])
    circuit.cx(data_qr[6], pf_ancilla_qr[1])
    circuit.h(pf_ancilla_qr[1])
    circuit.measure(pf_ancilla_qr[1], pf_c_reg[1])

    return circuit

print("Syndrome measurement function 'add_syndrome_measurement' has been defined.")
```

```
Syndrome measurement function 'add_syndrome_measurement' has been defined.
```

Apply Error Correction

Subtask:

Based on the measured syndromes, apply the appropriate correction operators (X, Y, or Z gates) to the physical qubits to rectify the introduced errors.

```
def decode_syndrome(bf_syndrome_result, pf_syndrome_result):
    """
    Decodes bit-flip and phase-flip syndrome results to identify the error type and location.

    Args:
        bf_syndrome_result (str): The bit-flip syndrome result (e.g., '0000', '0010').
        pf_syndrome_result (str): The phase-flip syndrome result (e.g., '00', '10').

    Returns:
        tuple: (error_type, qubit_index) or (None, None) if no error is detected.
               error_type: 'X', 'Z', 'Y' or None
               qubit_index: int (0-8) or None
    """

    bf_syndrome = bf_syndrome_result
    bf_error_qubit = None

    # Decode for first block (q0, q1, q2) using s0 (Z0Z1) and s1 (Z1Z2)
    s0_bf = bf_syndrome[-1] # Z0Z1
    s1_bf = bf_syndrome[-2] # Z1Z2

    if s0_bf == '1' and s1_bf == '0': # Z0Z1=1, Z1Z2=0 -> X on q0
        bf_error_qubit = 0
    elif s0_bf == '1' and s1_bf == '1': # Z0Z1=1, Z1Z2=1 -> X on q1
        bf_error_qubit = 1
```

```
        elif s0_bf == '0' and s1_bf == '1': # Z0Z1=0, Z1Z2=1 -> X on q2
            bf_error_qubit = 2

        # Decode for second block (q3, q4, q5) using s2 (Z3Z4) and s3 (Z4Z5)
        s2_bf = bf_syndrome[-3] # Z3Z4
        s3_bf = bf_syndrome[-4] # Z4Z5

        if bf_error_qubit is None:
            if s2_bf == '1' and s3_bf == '0': # Z3Z4=1, Z4Z5=0 -> X on q3
                bf_error_qubit = 3
            elif s2_bf == '1' and s3_bf == '1': # Z3Z4=1, Z4Z5=1 -> X on q4
                bf_error_qubit = 4
            elif s2_bf == '0' and s3_bf == '1': # Z3Z4=0, Z4Z5=1 -> X on q5
                bf_error_qubit = 5

        # Phase-flip syndrome decoding (based on Shor code structure)
        pf_syndrome = pf_syndrome_result
        pf_error_block = None # Identifies which block (0, 1, or 2) had a phase flip

        s0_pf = pf_syndrome[-1] # X0X3
        s1_pf = pf_syndrome[-2] # X3X6

        if s0_pf == '1' and s1_pf == '0': # X0X3=1, X3X6=0 -> Phase flip on Block 0 (q0,q1,q2)
            pf_error_block = 0
        elif s0_pf == '1' and s1_pf == '1': # X0X3=1, X3X6=1 -> Phase flip on Block 1 (q3,q4,q5)
            pf_error_block = 1
        elif s0_pf == '0' and s1_pf == '1': # X0X3=0, X3X6=1 -> Phase flip on Block 2 (q6,q7,q8)
            pf_error_block = 2

        # Combine bit-flip and phase-flip information
        if bf_error_qubit is not None and pf_error_block is not None:
            if pf_error_block == 0 and bf_error_qubit in [0, 1, 2]:
                return 'Y', bf_error_qubit
            elif pf_error_block == 1 and bf_error_qubit in [3, 4, 5]:
                return 'Y', bf_error_qubit
            elif pf_error_block == 2 and bf_error_qubit in [6, 7, 8]:
                return 'Y', bf_error_qubit
            else:
                print(f"Warning: Unmatched bit-flip {bf_error_qubit} and phase-flip block {pf_error_block}. Assuming independer
                if bf_error_qubit is not None:
                    return 'X', bf_error_qubit
                elif pf_error_block is not None:
                    if pf_error_block == 0: return 'Z', 0
                    if pf_error_block == 1: return 'Z', 3
                    if pf_error_block == 2: return 'Z', 6
                return None, None


        elif bf_error_qubit is not None: # Only bit-flip detected
            return 'X', bf_error_qubit
        elif pf_error_block is not None: # Only phase-flip detected
            if pf_error_block == 0: return 'Z', 0
            if pf_error_block == 1: return 'Z', 3
            if pf_error_block == 2: return 'Z', 6
        return None, None # No error detected

    print("The 'decode_syndrome' function has been defined.")
```

```
The 'decode_syndrome' function has been defined.
```

```
def apply_correction(circuit, error_type, qubit_index):
    """
    Applies the appropriate correction operator to a specified qubit.

    Args:
        circuit (QuantumCircuit): The input quantum circuit.
        error_type (str): The type of error to correct ('X', 'Z', or 'Y').
        qubit_index (int): The index of the qubit on which to apply the correction.

    Returns:
        QuantumCircuit: The modified quantum circuit with the correction applied.
    """
    if error_type == 'X':
        circuit.x(circuit.qregs[0][qubit_index])
        print(f"Applied bit-flip (X) correction on qubit {qubit_index}")
    elif error_type == 'Z':
        circuit.z(circuit.qregs[0][qubit_index])
        print(f"Applied phase-flip (Z) correction on qubit {qubit_index}")
    elif error_type == 'Y':
        circuit.y(circuit.qregs[0][qubit_index])
        print(f"Applied bit-phase-flip (Y) correction on qubit {qubit_index}")
    else:
```

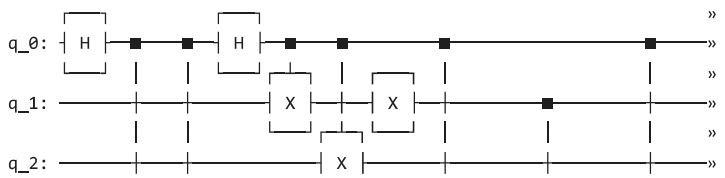
```
            print("No correction needed or invalid error type.")
        return circuit

    print("The 'apply_correction' function has been defined.")
```

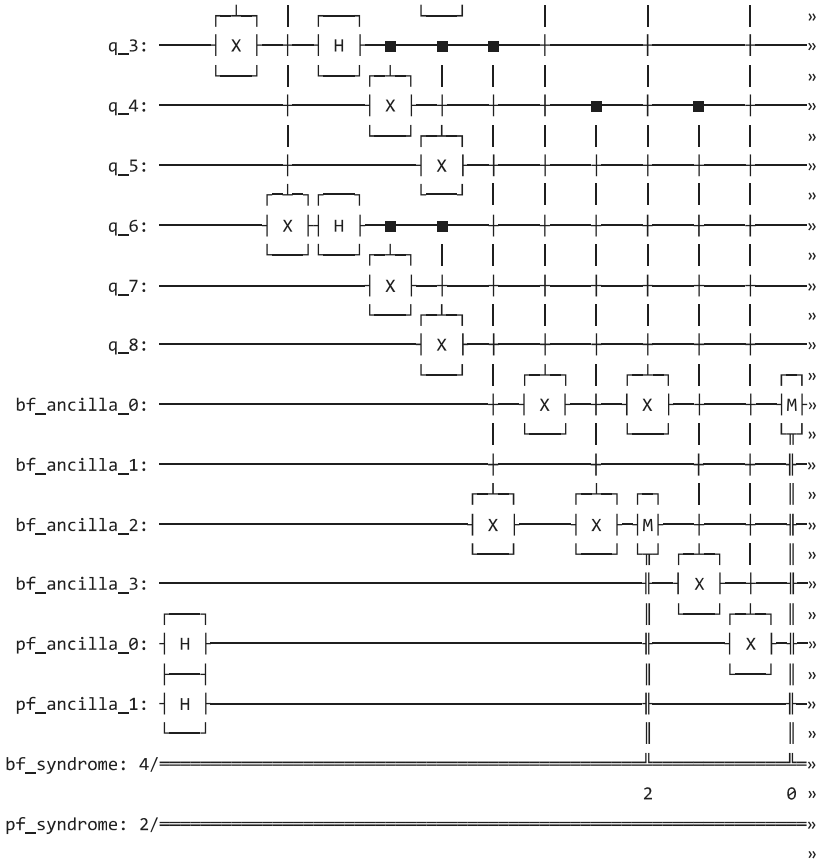
```
    The 'apply_correction' function has been defined.
```

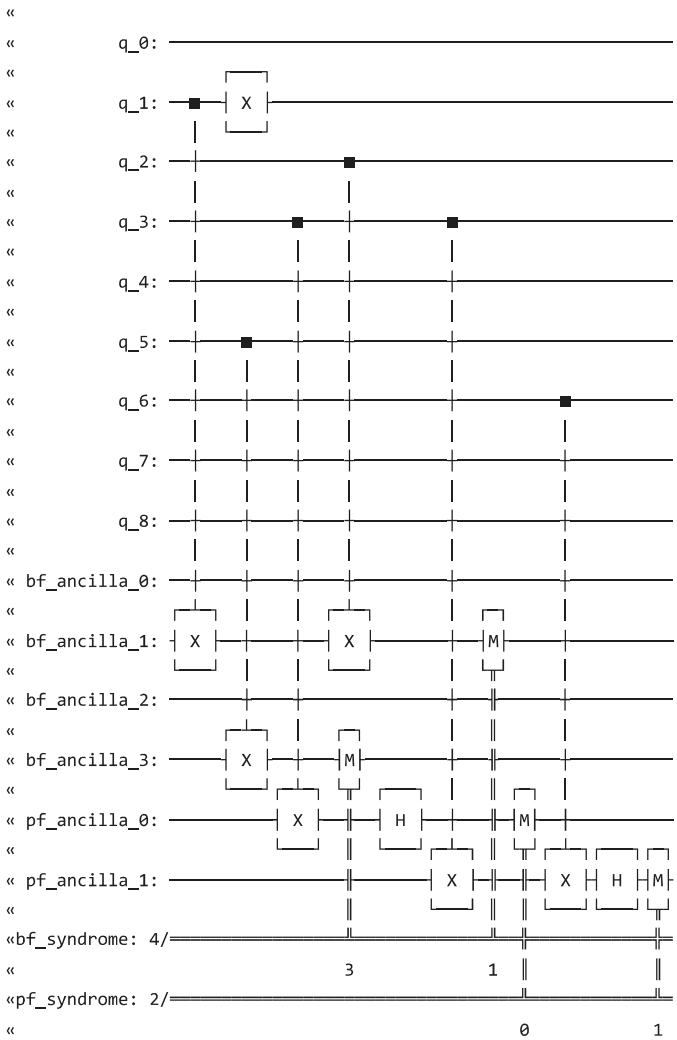
Integrate and Test the Shor Code Error Correction Pipeline

```
    from qiskit.providers.aer import AerSimulator

    # a. Initialize a new quantum circuit for the error correction process
    ec_qc = QuantumCircuit(data_qr, bf_ancilla_qr, pf_ancilla_qr, bf_c_reg, pf_c_reg, name='Shor Code EC')

    # b. Compose the shor_code_qc (encoding part) onto the new circuit
    ec_qc.compose(shor_code_qc, qubits=data_qr, inplace=True);

    # c. Introduce an example error using the introduce_error function
    # Let's introduce an X error on qubit 1 as an example
    error_qubit_index = 1
    error_type_introduced = 'X'
    # We apply the error directly to the data_qr in ec_qc
    ec_qc.x(data_qr[error_qubit_index])
    print(f"Introduced {error_type_introduced} error on data qubit {error_qubit_index}")

    # d. Add the syndrome measurement gates to this circuit using the add_syndrome_measurement function.
    ec_qc = add_syndrome_measurement(ec_qc)

    # e. To simulate the syndrome measurement and apply correction, we need to run the circuit on a simulator.
    simulator = AerSimulator()
    compiled_circuit = ec_qc.copy() # Make a copy for simulation before correction gates are added
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    counts = result.get_counts(compiled_circuit)

    # For simplicity, we'll take the first (and only) measurement result
    measured_syndrome = list(counts.keys())[0]

    # The measured_syndrome will be in the format 'pf_syndrome bf_syndrome' (e.g., '00 0000')
    pf_syndrome_result = measured_syndrome.split(' ')[0]
    bf_syndrome_result = measured_syndrome.split(' ')[1]

    print(f"Raw syndrome measurement result: {measured_syndrome}")
    print(f"Bit-flip syndrome: {bf_syndrome_result}")
    print(f"Phase-flip syndrome: {pf_syndrome_result}")

    # g. Use the decode_syndrome function to determine the error location and type.
    error_type, qubit_to_correct = decode_syndrome(bf_syndrome_result, pf_syndrome_result)

    print(f"Decoded error: Type={error_type}, Qubit={qubit_to_correct}")

    # h. Based on the decoded error, apply the appropriate correction gate using the apply_correction function.
    # Note: Correction gates are appended AFTER the measurements for visualization purposes in the circuit.
    if error_type is not None and qubit_to_correct is not None:
        ec_qc = apply_correction(ec_qc, error_type, qubit_to_correct)
    else:
        print("No correctable error detected or identified by the decoder.")

    # i. Print the final circuit with encoding, error, syndrome measurements, and error correction gates.
    print("\nShor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:")
    print(ec_qc.draw('text', idle_wires=False))
```

```
Introduced X error on data qubit 1
Raw syndrome measurement result: 00 0011
Bit-flip syndrome: 0011
Phase-flip syndrome: 00
Decoded error: Type=X, Qubit=1
Applied bit-flip (X) correction on qubit 1

Shor Code with Encoding, Introduced Error, Syndrome Measurements, and Correction:
```




q_3:
q_4:
q_5:
q_6:
q_7:
q_8:
bf_ancilla_0:
bf_ancilla_1:
bf_ancilla_2:
bf_ancilla_3:
pf_ancilla_0:
pf_ancilla_1:
bf_syndrome: 4/
2 0
pf_syndrome: 2/


q_0:
q_1:
q_2:
q_3:
q_4:
q_5:
q_6:
q_7:
q_8:
bf_ancilla_0:
bf_ancilla_1:
bf_ancilla_2:
bf_ancilla_3:
pf_ancilla_0:
pf_ancilla_1:
bf_syndrome: 4/
3 1
pf_syndrome: 2/
0 1