Il Protocollo di Shor

Relazione sull'Home Assignment N° 8
Corso di Fondamenti di Computazione Quantistica

Francesco Matano - S5253162
Edoardo Vassallo - S4965918

1 Giugno 2024

Contents

1 Consegna

Si implementi il protocollo di correzione degli errori di Shor presentato in figura 1. La sezione E denota l'applicazione dell'errore. É possibile scegliere uno o più tipi di errori fra quelli presentati a lezione e sulle note (bit flip, small error, phase flip). Questo dovrà essere modellizzato come un'operatore Hermittiano che agisce sui qubit (o su un qubit) del sistema.

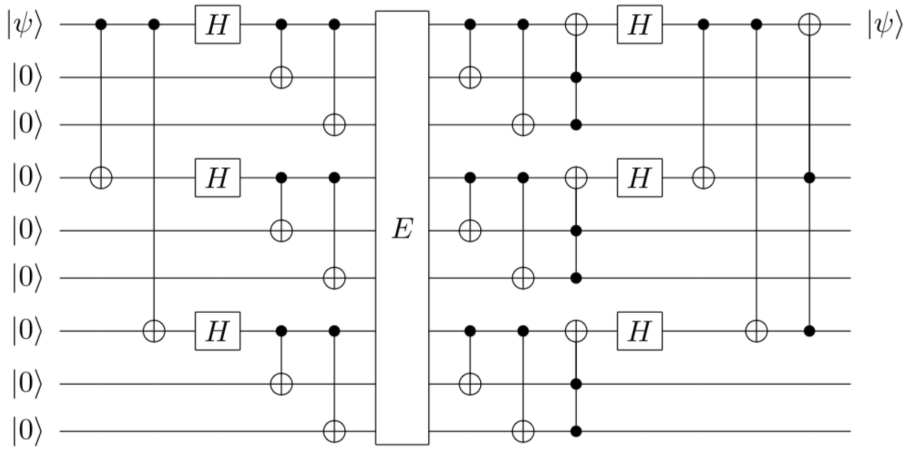


Figure 1: Protocollo di Correzione degli Errori di Shor

2 Introduzione

In questa relazione analizzeremo il funzionamento del protocollo di correzione degli errori di Shor a 9 qubit, e la sua implementazione in Qiskit.

3 Implementazione degli Errori

Iniziamo la nostra analisi con la modellazione dei possibili errori sui nostri qubit.

3.1 Bit-Flip

Il *bit-flip* viene descritto come un inversione dello stato del qubit coinvolto.

$$|0\rangle \xrightarrow{\text{bit-flip}} |1\rangle \tag{1}$$

$$|1\rangle \xrightarrow{\text{bit-flip}} |0\rangle \tag{2}$$

$$\alpha|0\rangle + \beta|1\rangle \xrightarrow{\text{bit-flip}} \beta|0\rangle + \alpha|1\rangle \tag{3}$$

Esso può essere facilmente simulato tramite il "NOT quantistico", la X di Pauli. L'operatore viene definito dalla seguente matrice Hermittiana:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{4}$$

In Qiskit, quest'operatore è implementato dal metodo `x()` della classe `qiskit.QuantumCircuit`.

3.2 Phase-Flip

Il *phase-flip* viene descritto come un inversione del segno della fase relativa del qubit coinvolto.

$$|0\rangle \xrightarrow{\text{phase-flip}} |0\rangle \tag{5}$$

$$|1\rangle \xrightarrow{\text{phase-flip}} -|1\rangle \tag{6}$$

$$\alpha|0\rangle + \beta|1\rangle \xrightarrow{\text{phase-flip}} \alpha|0\rangle - \beta|1\rangle \tag{7}$$

È interessante notare come questo possa essere visto come un *bit-flip* nella base $\pm$

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \xrightarrow{\text{phase-flip}} \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle \tag{8}$$

Analogamente all'errore precedente, anch'esso può essere simulato usando un operatore di Pauli, la Z. L'operatore viene definito dalla seguente matrice Hermittiana:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{9}$$

In Qiskit, quest'operatore è implementato dal metodo `z()` della classe `qiskit.QuantumCircuit`.

4 Correzione degli Errori

L'idea alla base della correzione degli errori è simile a quella dell'informatica classica: la ridondanza. Troveremo un modo di moltiplicare l'informazione logica, e sfruttare questa ripetizione allo scopo di poterla riparare, se necessario.

4.1 Bit-Flip

Consideriamo il nostro stato iniziale $|\phi\rangle$, definito come segue:

$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{10}$$

Questo è lo stato che intendiamo correggere. Ad esso associamo due stati *ancilla*, inizializzati a $|0\rangle$. Lo stato iniziale del sistema $|\psi\rangle$ è quindi:

$$|\psi\rangle = |\phi\rangle|0_1\rangle|0_2\rangle \tag{11}$$

Ora, applichiamo due porte CNOT, $C_\phi NOT_1$ e $C_\phi NOT_2$. Otteniamo quindi:

$$|\psi\rangle \xrightarrow{C_\phi NOT_1} (\alpha|0_\phi 0_1\rangle + \beta|1_\phi 1_1\rangle)|0_2\rangle \tag{12}$$

$$\xrightarrow{C_\phi NOT_2} \alpha|0_\phi 0_1 0_2\rangle + \beta|1_\phi 1_1 1_2\rangle \tag{13}$$

Questa serie di operazioni ha avuto l'effetto di codificare lo stato del singolo qubit $|\phi\rangle$ in un blocco di 3 qubit. Il circuito viene mostrato nella figura 2

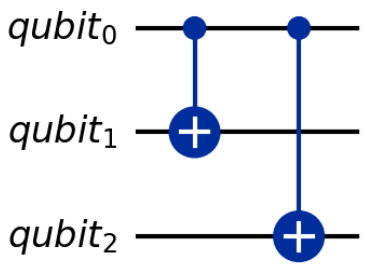


Figure 2: Pre-Protocollo di Codifica per il *Bit-Flip*

Supponiamo ora di ricevere i qubit in seguito ad una perturbazione. Effettueremo nuovamente le stesse operazioni $C_\phi NOT_2$ e $C_\phi NOT_1$. Concentriamoci ora sul valore dei due bit ancilla, che chiameremo **sindrome**, $|s\rangle$

Supponiamo che sia avvenuto un singolo *bit-flip* su uno dei tre qubit. Esistono quindi tre possibili casi:

1. L'errore avviene su $|0_1\rangle$

$$|\psi\rangle \xrightarrow{\text{bit-flip}_1} \alpha|0_\phi 1_1 0_2\rangle + \beta|1_\phi 0_1 1_2\rangle \tag{14}$$
$$\xrightarrow{C_\phi NOT_2} \alpha|0_\phi 1_1 0_2\rangle + \beta|1_\phi 0_1 0_2\rangle \tag{15}$$
$$\xrightarrow{C_\phi NOT_1} \alpha|0_\phi 1_1 0_2\rangle + \beta|1_\phi 1_1 0_2\rangle \tag{16}$$
$$|s\rangle = |1_1 0_2\rangle \tag{17}$$

2. L'errore avviene su $|0_2\rangle$

$$|\psi\rangle \xrightarrow{\text{bit-flip}_2} \alpha|0_\phi 0_1 1_2\rangle + \beta|1_\phi 1_1 0_2\rangle \tag{18}$$
$$\xrightarrow{C_\phi NOT_2} \alpha|0_\phi 0_1 1_2\rangle + \beta|1_\phi 1_1 1_2\rangle \tag{19}$$
$$\xrightarrow{C_\phi NOT_1} \alpha|0_\phi 0_1 1_2\rangle + \beta|1_\phi 0_1 1_2\rangle \tag{20}$$
$$|s\rangle = |0_1 1_2\rangle \tag{21}$$

3. L'errore avviene su $|\phi\rangle$

$$|\psi\rangle \xrightarrow{\text{bit-flip}_\phi} \alpha|1_\phi 0_1 0_2\rangle + \beta|0_\phi 1_1 1_2\rangle \tag{22}$$
$$\xrightarrow{C_\phi NOT_2} \alpha|1_\phi 0_1 1_2\rangle + \beta|0_\phi 1_1 1_2\rangle \tag{23}$$
$$\xrightarrow{C_\phi NOT_1} \alpha|1_\phi 1_1 1_2\rangle + \beta|0_\phi 1_1 1_2\rangle \tag{24}$$
$$|s\rangle = |1_1 1_2\rangle \tag{25}$$

Notiamo quindi che possiamo dedurre l'errore avvenuto sul circuito sulla base del valore di $|s\rangle$. In particolare, il caso che vogliamo correggere è l'errore sul primo qubit, in cui $|s\rangle = |1_1 1_2\rangle$. Possiamo usare questo valore come input per una Porta di Toffoli $C_1C_2NOT_\phi$, la quale invertirà il valore di $|\phi\rangle$ solo nel caso in cui $|s\rangle = |1_1 1_2\rangle$

$$\alpha|1_\phi 1_1 1_2\rangle + \beta|0_\phi 0_1 0_2\rangle \xrightarrow{C_1C_2NOT_\phi} \alpha|0_\phi 1_1 1_2\rangle + \beta|1_\phi 0_1 0_2\rangle \tag{26}$$

Il *bit-flip* viene quindi corretto, a scapito del valore dei bit ancilla. Essi dovranno quindi essere rinizializzati o lasciati decadere naturalmente prima di un nuovo riutilizzo.

Mostriamo quindi in figura 3 e 4 il post-protocollo ed il protocollo completo rispettivamente. Di seguito viene riportato il codice Qiskit per realizzare il circuito

```
# BIT-FLIP CORRECTION

from qiskit import(QuantumCircuit, QuantumRegister, ClassicalRegister)
```

```
qubit = QuantumRegister( 3, "qubit")
bit = ClassicalRegister( 1, "bit")
bit_flip_circ = QuantumCircuit(qubit, bit)

# CNOT
bit_flip_circ.cx(qubit[0], qubit[1])
bit_flip_circ.cx(qubit[0], qubit[2])

bit_flip_circ.barrier() ############################

# Mettere qui la perturbazione

bit_flip_circ.barrier() ############################

# CNOT
bit_flip_circ.cx(qubit[0], qubit[2])
bit_flip_circ.cx(qubit[0], qubit[1])

# TOFFOLI
bit_flip_circ.ccx(qubit[2], qubit[1], qubit[0])

bit_flip_circ.barrier() ############################

# MISURA
bit_flip_circ.measure(qubit[0], bit[0])

# DISEGNO CIRCUITO
bit_flip_circ.draw("mpl", filename="bit_flip_complete.png")
```

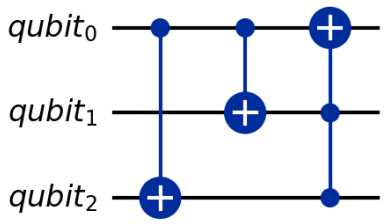


Figure 3: Post-Protocollo di Decodifica per il *Bit-Flip*

È importante notare che il circuito è in grado di riconoscere e correggere il solo un *bit-flip* per volta. Esso inoltre non è in grado di riconoscere eventuali errori di fase

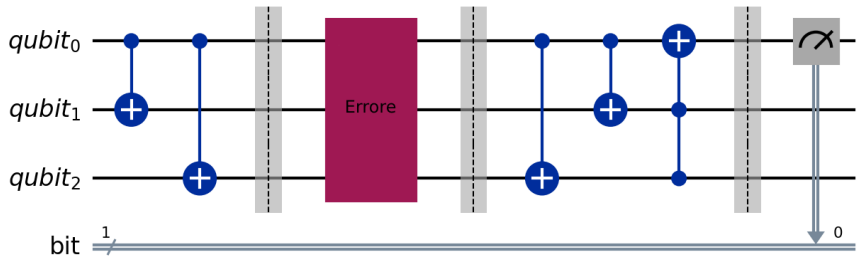


Figure 4: Protocollo di Correzione del *Bit-Flip*

4.2 Phase-Flip

Il protocollo di correzione del *phase-flip* deriva dal precedente, sulla base dell'osservazione effettuata tramite l'espressione 8, ovvero che il *phase-flip* può essere visto come un *bit-flip* nella base $\pm$. Dunque, noi useremo gli stessi metodi del punto precedente, convertendo i nostri qubit nella base $\pm$ usando tre Porte di Hadamard H

$$|\psi\rangle \xrightarrow{C_\phi NOT_1\ C_\phi NOT_2} \alpha|0_\phi 0_1 0_2\rangle + \beta|1_\phi 1_1 1_2\rangle \tag{27}$$

$$\xrightarrow{H^{\otimes 3}} \alpha|+_\phi +_1 +_2\rangle + \beta|-_\phi -_1 -_2\rangle \tag{28}$$

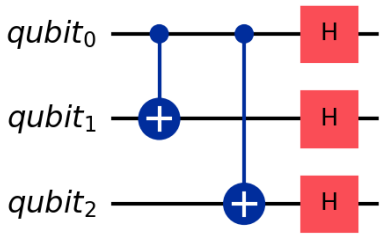


Figure 5: Pre-Protocollo di Codifica per il *Phase-Flip*

In seguito alla perturbazione, riapplicheremo le Porte di Hadamard, riconducendoci ai casi precedenti:

1. L'errore avviene su $|0_1\rangle$

$$|\psi\rangle \xrightarrow{\text{phase-flip}_1} \alpha|+_\phi -_1 +_2\rangle + \beta|-_\phi +_1 -_2\rangle \tag{29}$$

$$\xrightarrow{H^{\otimes 3}} \alpha|0_\phi 1_1 0_2\rangle + \beta|1_\phi 0_1 1_2\rangle \tag{30}$$

2. L'errore avviene su $|0_2\rangle$

$$|\psi\rangle \xrightarrow{\text{phase-flip}_2} \alpha|+_\phi +_1 -_2\rangle + \beta|-_\phi -_1 +_2\rangle \tag{31}$$

$$\xrightarrow{H^{\otimes 3}} \alpha|0_\phi 0_1 1_2\rangle + \beta|1_\phi 1_1 0_2\rangle \tag{32}$$

3. L'errore avviene su $|\phi\rangle$

$$|\psi\rangle \xrightarrow{\text{phase-flip}_\phi} \alpha|-_\phi +_1 +_2\rangle + \beta|+_\phi -_1 -_2\rangle \tag{33}$$

$$\xrightarrow{H^{\otimes 3}} \alpha|1_\phi 0_1 0_2\rangle + \beta|0_\phi 1_1 1_2\rangle \tag{34}$$

La correzione avverrà in maniera analoga alla precedente, tramite una Porta di Toffoli. Il post-protocollo ed il circuito completo sono rispettivamente in figura 6 e 7. Di seguito viene riportato il codice Qiskit per realizzare il circuito

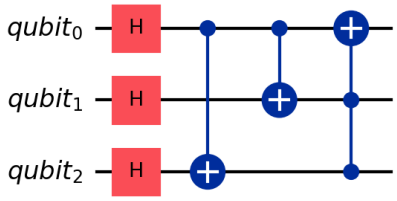


Figure 6: Post-Protocollo di Decodifica per il *Phase-Flip*

```
# PHASE-FLIP CORRECTION

from qiskit import(QuantumCircuit, QuantumRegister, ClassicalRegister)

qubit = QuantumRegister( 3, "qubit")
bit = ClassicalRegister( 1, "bit")
phase_flip_circ = QuantumCircuit(qubit, bit)

# CNOT
phase_flip_circ.cx(qubit[0], qubit[1])
phase_flip_circ.cx(qubit[0], qubit[2])

# HADAMARD
phase_flip_circ.h(qubit[0])
phase_flip_circ.h(qubit[1])
phase_flip_circ.h(qubit[2])

phase_flip_circ.barrier() ############################

# Mettere qui la perturbazione
```

```
phase_flip_circ.barrier() ############################

# HADAMARD
phase_flip_circ.h(qubit[0])
phase_flip_circ.h(qubit[1])
phase_flip_circ.h(qubit[2])

# CNOT
phase_flip_circ.cx(qubit[0], qubit[2])
phase_flip_circ.cx(qubit[0], qubit[1])

# TOFFOLI
phase_flip_circ.ccx(qubit[2], qubit[1], qubit[0])

phase_flip_circ.barrier() ############################

# MISURA
phase_flip_circ.measure(qubit[0], bit[0])

# DISEGNO CIRCUITO
phase_flip_circ.draw("mpl", filename="phase_flip_complete.png")
```

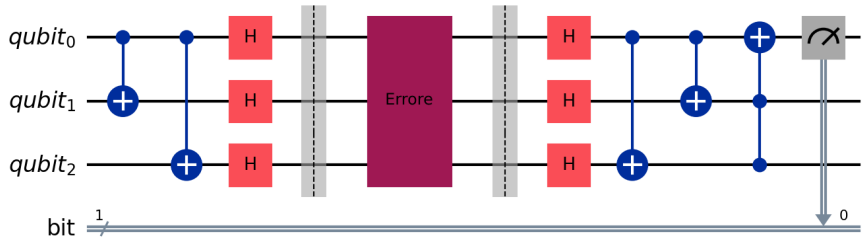


Figure 7: Protocollo di Correzione del *Phase-Flip*

Possiamo fare considerazioni simili al precedente protocollo. Esso infatti è in grado di rilevare un singolo errore per volta, ed incapace di riconoscere errori di *bit-flip*

4.3 Protocollo di Shor

L'idea del protocollo di Shor è quella di utilizzare i precedenti protocolli insieme, in modo da poter correggere sia *bit-flip* che *phase-flip* su di un qubit.

Cominciamo con il nostro qubit logico $|\phi\rangle$ e due qubit ancilla $|00\rangle$, su cui applicheremo il pre-protocollo di correzione di *phase-flip*. In seguito, associamo due ulteriori qubit ancilla $|00\rangle$ ad ognuno dei tre qubit iniziali. Applichiamo

quindi il pre-protocollo di correzione di *bit-flip* per ognuna di queste triplette. Per realizzarlo avremo bisogno di un qubit logico e otto qubit ancilla. Abbiamo quindi il pre-protocollo in figura 8

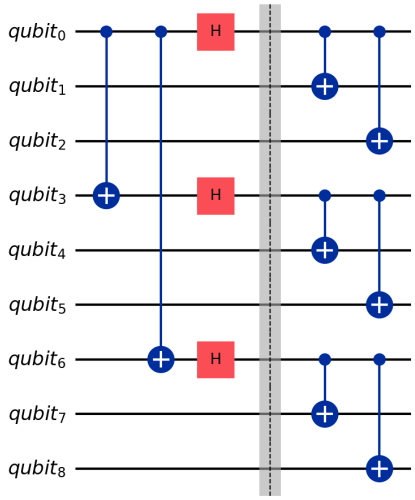


Figure 8: Pre-Protocollo di Shor

In seguito alla perturbazione effettueremo prima il post-protocollo del *bit-flip* per ognuna delle rispettive triplette, e poi il post-protocollo del *phase-flip* sui tre qubit originali. L'insieme delle operazioni viene rappresentato dal circuito in figura 9. Il circuito completo viene mostrato in figura 10. Di seguito viene riportato il codice Qiskit per realizzare il circuito

```
# SHOR PROTOCOL

from qiskit import(QuantumCircuit, QuantumRegister, ClassicalRegister)

qubit = QuantumRegister( 9, "qubit")
bit = ClassicalRegister( 1, "bit")
shor_circ = QuantumCircuit(qubit, bit)

# CNOT
shor_circ.cx(qubit[0], qubit[3])
shor_circ.cx(qubit[0], qubit[6])

# HADAMARD
shor_circ.h(qubit[0])
shor_circ.h(qubit[3])
shor_circ.h(qubit[6])
```

```
shor_circ.barrier() ##############################

# CNOT
shor_circ.cx(qubit[0], qubit[1])
shor_circ.cx(qubit[3], qubit[4])
shor_circ.cx(qubit[6], qubit[7])
shor_circ.cx(qubit[0], qubit[2])
shor_circ.cx(qubit[3], qubit[5])
shor_circ.cx(qubit[6], qubit[8])

shor_circ.barrier() ##############################

# Inserire qui la perturbazione

shor_circ.barrier() ##############################

# CNOT
shor_circ.cx(qubit[0], qubit[2])
shor_circ.cx(qubit[3], qubit[5])
shor_circ.cx(qubit[6], qubit[8])
shor_circ.cx(qubit[0], qubit[1])
shor_circ.cx(qubit[3], qubit[4])
shor_circ.cx(qubit[6], qubit[7])

# TOFFOLI
shor_circ.ccx(qubit[2], qubit[1], qubit[0])
shor_circ.ccx(qubit[5], qubit[4], qubit[3])
shor_circ.ccx(qubit[8], qubit[7], qubit[6])

shor_circ.barrier() ##############################

# HADAMARD
shor_circ.h(qubit[0])
shor_circ.h(qubit[3])
shor_circ.h(qubit[6])

# CNOT
shor_circ.cx(qubit[0], qubit[6])
shor_circ.cx(qubit[0], qubit[3])

# TOFFOLI
shor_circ.ccx(qubit[6], qubit[3], qubit[0])

shor_circ.barrier() ##############################

shor_circ.measure(qubit[0], bit)

shor_circ.draw("mpl", filename="shor_complete.png")
```

L'idea è quindi che il qubit $|\phi\rangle$ è protetto da eventuali *phase-flip*. A sua volta, sia lui che i suoi primi due bit ancilla sono protetti da *bit-flip*

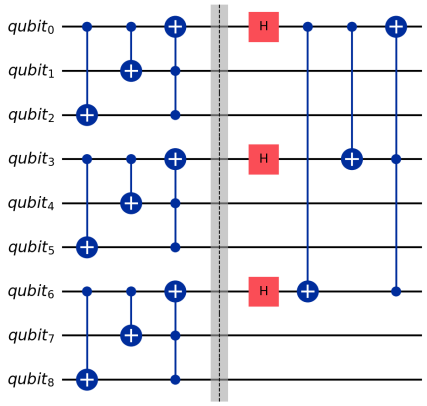


Figure 9: Post-Protocollo di Shor

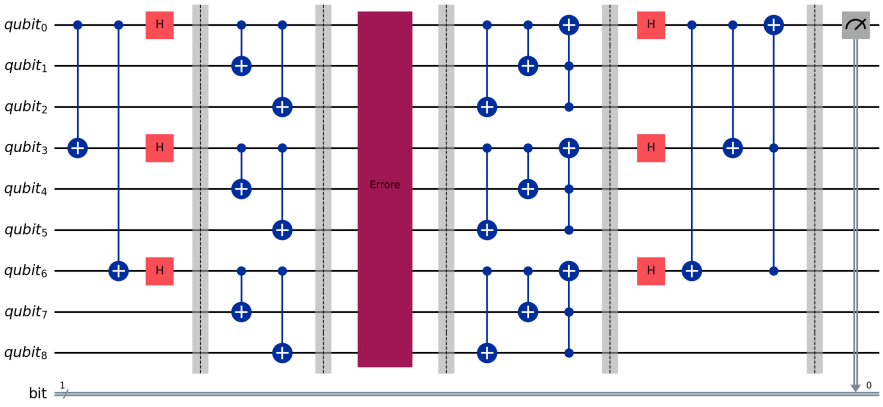


Figure 10: Protocollo di Shor

5 Esperimenti

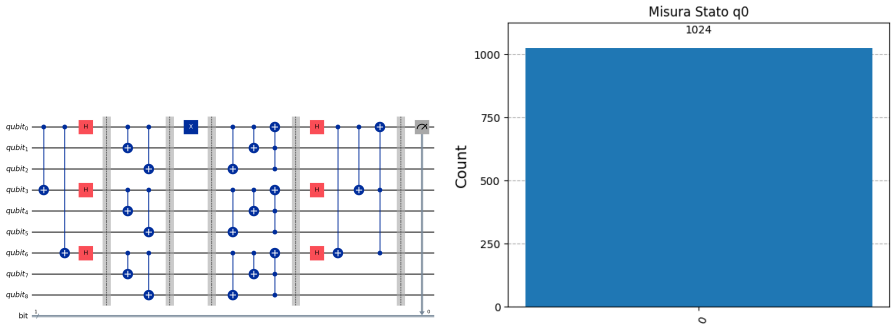
Il circuito realizzato nella precedente sezione è stato testato, usando il simulatore `qiskit_aer.AerSimulator`, come di seguito.

```
# SIMULAZIONE
from qiskit import transpile
from qiskit.visualization import plot_histogram
from qiskit_aer import AerSimulator

simulator = AerSimulator()
shor_circ = transpile(shor_circ, simulator)

result = simulator.run(shor_circ, shots="1024").result()
counts = result.get_counts(shor_circ)
plot_histogram(counts, title='Misura Stato q0')
```

Si è simulata la correzione di un *bit-flip*, *phase-flip* ed entrambi sul primo qubit. I risultati sono mostrati rispettivamente in figura 11, 12 e 13. In tutti i casi, il circuito è stato in grado di riparare lo stato iniziale del qubit, ovvero $|0\rangle$.



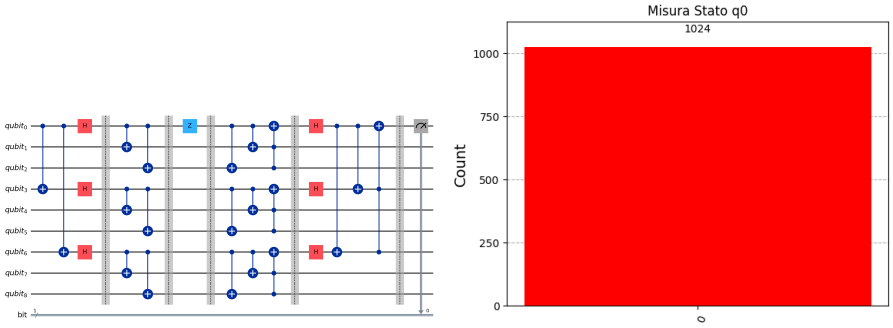
(a) Circuito Eseguito (b) Risultato

Figure 11: Correzione *Bit-Flip*

In seguito a queste verifiche, abbiamo sperimentato con il circuito, testando diversi tipi di input. Riportiamo i due casi che riteniamo più interessanti.

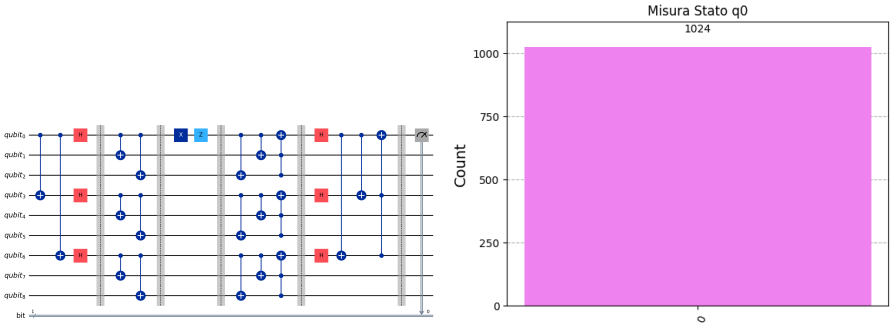
1. Dato il funzionamento della correzione, basato sulla fiducia nelle sindromi calcolate sui qubit ancilla, abbiamo provato a falsificare i loro valori, e causare una correzione non necessaria. Abbiamo quindi causato dei *bit-flip* solo su i qubit ancilla di q_3 e q_6, tramite l'operatore $X_4 \otimes X_5 \otimes X_7 \otimes X_8$ allo scopo di causare una correzione non necessaria su di essi, ed a loro volta una correzione non necessaria su q_1.

   Al contrario delle nostre aspettative, il valore finale non sembra essere stato influenzato dalla perturbazione. I risultati sono mostrati in figura 14



(a) Circuito Eseguito (b) Risultato

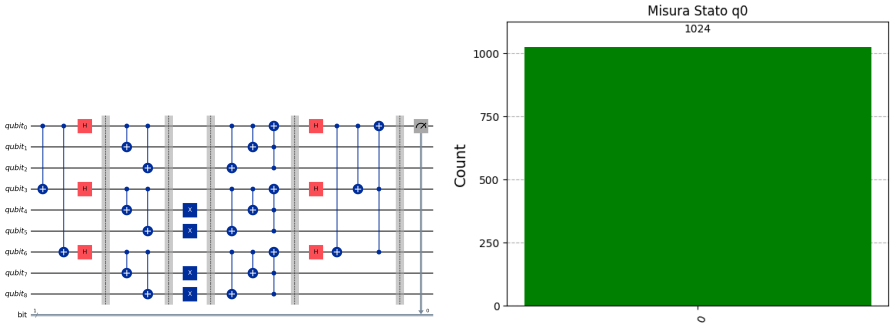
Figure 12: Correzione *Phase-Flip*



(a) Circuito Eseguito (b) Risultato

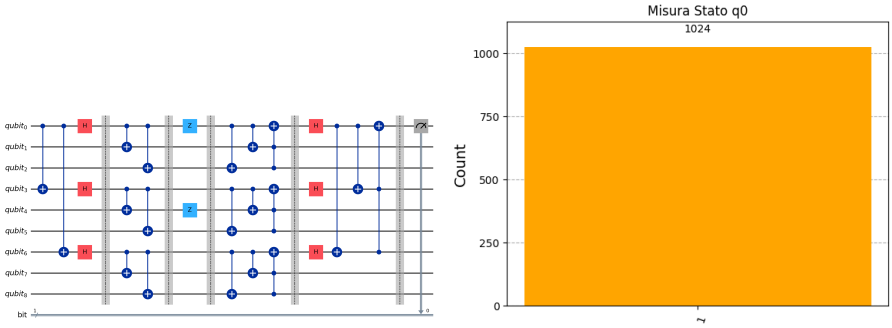
Figure 13: Correzione contemporanea di *Bit-Flip* e *Phase-Flip*

2. Nel corso delle varie esecuzioni, abbiamo ritrovato il seguente caso in cui il protocollo non riesce a correggere il valore. In esso, vengono applicati due *phase-flip*, su q_0 e q_4, tramite l'operatore $Z_0 \otimes Z_4$. Il risultato è in figura 15



(a) Circuito Eseguito (b) Risultato

Figure 14: Particolare circuito 1



(a) Circuito Eseguito (b) Risultato

Figure 15: Particolare circuito 2

6 Conclusioni

Alla fine di questa analisi, possiamo dire di riconoscere l'intuitività e l'efficacia del metodo proposto da Shor. Allo stesso tempo riconosciamo che non sia perfetto, e sia principalmente pensato per casi in cui sia avvenuto un singolo errore. Comunque, questo non discredita la funzionalità del protocollo, specie nei casi in cui le probabilità di multipli errori siano inferiori rispetto al singolo errore.

7 Bibliografia

1. *Note del corso di Fondamenti di Computazione Quantistica, Capitolo 7: Introduzione ai Codici di Correzione degli Errori per Computer Quantistici*, Paolo Solinas, UniGe

2. *Quantum Circuits for Stabilizer Error Correcting Codes: A Tutorial*, Arijit Mondal, Keshab K. Parhi

3. *Digital System Design for Quantum Error Correction Codes*, Othman O. Khalifa, Nur Amirah bt Sharif, Rashid A Saeed, S. Abdel-Khalek, Abdulaziz N. Alharbi, Ali A. Alkathiri