Sommatore Quantistico

Introduzione classica e trattazione quantistica

Beltramo Gregorio s248417
Cerutti Francesco s246695
Scarciglia Lorenzo s250796

I. Introduzione

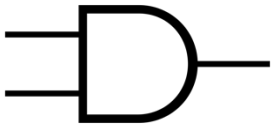
L' idea di questo progetto consiste nella realizzazione di un sommatore a 2 bit a livello quantistico. La prima parte sarà caratterizzata da una trattazione classica con una descrizione delle porte logiche necessarie alla costruzione di un Half Adder e un Full Adder, elementi base di un sommatore. Estenderemo la trattazione a livello quantistico e infine verranno presentati degli esempi funzionanti tramite Quantum Composer e Qiskit forniti da IBM.

II. Trattazione Classica

A. Porte logiche classiche: AND, OR e XOR

Una porta logica AND a 2 ingressi restituisce '1' quando entrambi gli input sono pari a '1' e '0' nei restanti casi. È descritta dalla seguente tabella di verità:

AND		
A	**B**	**A and B**
0	0	0
0	1	0
1	0	0
1	1	1



Invece, una porta OR restituisce '0' quando entrambi gli input sono 0, e '1' negli altri casi. La sua tabella di verità è la seguente:

OR		
A	**B**	**A or B**
0	0	0
0	1	1
1	0	1
1	1	1

Una porta XOR restituisce '1' quando gli input sono complementari tra loro e '0' quando sono uguali. La sua tabella di verità è la seguente:

XOR		
A	**B**	**A xor B**
0	0	0
0	1	1
1	0	1
1	1	0

B. Half Adder

È un componente fondamentale per il sommatore, che si andrà a descrivere in seguito. È costituito da una porta AND e una porta XOR. Serve a calcolare la somma di 2 bit alla volta senza un carry (riporto) in ingresso. In uscita restituisce 2 bit, il MSB (Most Significant bit) contenente il carry di uscita e il LSB (Less Significant bit) la somma binaria. La rappresentazione logica si deriva dalla sua tavola di verità:

HALF ADDER			
A	**B**	**Carry Out**	**Somma**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

In appendice A sono presenti i calcoli di derivazione (SoP).

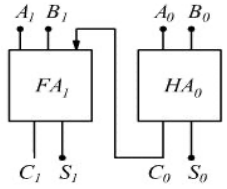
C. *Full Adder*

Il Full Adder si può considerare l'evoluzione dell'Half Adder. Ha come ingresso 3 bit, i due da sommare e un eventuale bit di carry in ingresso. Restituisce le stesse uscite dell'Half Adder. La sua tabella della verità è la seguente:

FULL ADDER				
A	**B**	**Carry In**	**Carry Out**	**Somma**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

D. *Adder di valori a 2 bit*

Un sommatore a 2 bit è formato dalla cascata di un Half Adder e di un Full Adder. In generale, per la somma di 2 valori a n bit sono necessari *n - 1* Full Adder e *1* Half Adder. L'Half Adder è adibito al calcolo del LSB della somma, mentre gli altri Full Adder calcolano le altre cifre. Nel caso in cui la somma sia troppo grande e il numero di bit in entrata sia minore del numero di bit in uscita si ottiene un overflow. Ovvero, il carry out dell'ultimo Full Adder sarà pari a '1'.



In figura si ha un Adder formato dalla cascata di Half e Full Adder

III. Trattazione Quantistica

A. *Porte logiche quantistiche: CNOT e TOFFOLI*

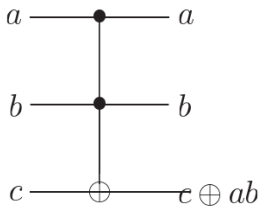
Come ogni porta logica quantistica, anche la CNOT e la Toffoli sono unitarie e reversibili. Unitarie in quanto prodotto di porte logiche quantistiche più semplici anch'esse unitarie. Reversibili, poiché se vengono applicate nuovamente riottengo i qubit con il valore che avevano in ingresso.

La porta CNOT agisce su 2 qubit differenti, uno di controllo e uno di target (in tabella denominato *Data*). Quando il qubit di controllo risulta nello stato $|1\rangle$ il qubit di target viene invertito. Viene rappresentato dalla seguente tabella:

Control - $\|C\rangle$	**Data -** $\|D\rangle$	**CNOT**$\|CD\rangle$
$\|0\rangle$	$\|0\rangle$	$\|00\rangle$
$\|0\rangle$	$\|1\rangle$	$\|01\rangle$
$\|1\rangle$	$\|0\rangle$	$\|11\rangle$
$\|1\rangle$	$\|1\rangle$	$\|10\rangle$

La porta di Toffoli esegue la stessa operazione, ma ha 2 qubit di controllo (*a* e *b*) e $|1\rangle$ di target (*c*). Il risultato sul bit di target è quindi lo XOR dell'AND dei 2 qubit di controllo.

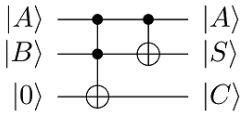
Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0



Le porte logiche classiche come AND e OR non sono applicabili in quanto non reversibili. Per ovviare a tutto ciò si utilizzano la porta di Toffoli e la CNOT per eseguire le stesse logiche.

B. *Half Adder Quantistico*

Un Half Adder quantistico ha lo stesso scopo di quello classico. Siccome i qubit non possono essere distrutti, come succede per i segnali di ingresso di alcune porte nella logica classica, si hanno come minimo 3 ingressi e 3 uscite.



Nella nostra implementazione abbiamo modificato il circuito affinché esso operi a 4 qubit. Così facendo otteniamo in output il carry out in prima posizione, a seguire la somma binaria e nelle ultime 2 posizioni gli addendi *b* e *a*.

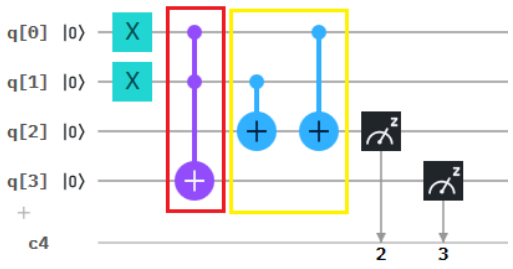


Figura 1

q[0] e q[1] sono rispettivamente gli addendi *a*, *b*; q[2] è inizializzato a $|0\rangle$ e sarà il qubit in cui viene salvata la somma, mentre l'ultimo qubit anch'esso inizializzato a $|0\rangle$ sarà il qubit adibito al carry in output.

In questo esempio abbiamo effettuato la somma di 2 qubit nello stato $|1\rangle$. La prima porta applicata dopo le 2 NOT su q[0], q[1] è la porta di Toffoli. Visto che q[3] è inizializzato a $|0\rangle$, siccome q[0] e q[1] sono $|1\rangle$, il suo valore viene invertito, cioè passerà allo stato $|1\rangle$. Successivamente vengono applicate 2 porte CNOT che servono a settare il q[2]. La prima effettua una negazione e la seconda pure in

quanto entrambi gli addendi sono |1⟩. Da questa configurazione ci aspettiamo come risultato più probabile la configurazione |1011⟩. I risultati verranno discussi in seguito.

Riapplicando le stesse porte nella stessa sequenza (quindi specchiando il circuito) si verifica che è reversibile.

C. *Full Adder Quantistico*

L'implementazione di un Full Adder a livello quantistico richiede l'uso di 4 qubit. Questo è dovuto al fatto che in ingresso si hanno 2 addendi (come nel caso precedente) e un carry in. La quarta posizione (q[3]) è inizializzata a |0⟩. L'output sarà caratterizzato da un carry out in posizione q[3], la somma in q[2] (che riutilizza il qubit di carry in) e infine i 2 addendi.

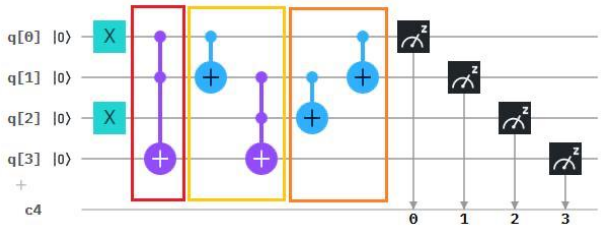


Figura 2

Nel riquadro rosso abbiamo una Toffoli che porta il carry out q[3] a |1⟩ se entrambi gli addendi sono pari a |1⟩. In quello giallo è presente una CNOT e una Toffoli che impostano a |1⟩ il carry out se *a* o *b* sono nello stato |1⟩ e il carry in è a |1⟩. Agisce quindi nel caso in cui non agisca la prima Toffoli. Facendo riferimento alla logica classica questo blocco implementa (q[0] XOR q[1]) AND q[2]. L'ultimo riquadro serve a sovrascrivere la somma sul carry in e riportare l'ingresso q[1] allo stato iniziale. Da questo esempio ci aspettiamo come stato finale |1001⟩ dove l'ordine dei qubit è q[3], q[2], q[1], q[0] e i primi 2 sono rispettivamente il carry out e la somma.

Da notare come applicando una seconda volta le stesse porte si verifica che il circuito è reversibile.

D. *Adder di valori a 2 qubit*

Abbiamo costruito l'Adder partendo dal modello classico, quindi abbiamo utilizzato un Half Adder e un Full Adder (Figura 3). In verde sono evidenziati gli input. Nel blocco rosso si può notare l'Half Adder che risulta immutato. Nel blocco giallo il Full Adder ha subito qualche modifica conseguente alla necessità di utilizzare il carry out dell'Half Adder come carry in del Full Adder. Nell' ultimo riquadro si sono utilizzate 3 porte di swap (porta che scambia l'ordine dei 2 qubits selezionati) per riordinare nel seguente modo lo stato finale:

q[6] = Carry out del Full Adder q[3], q[2] = b_1, b_0

q[5] = Somma Full Adder q[1], q[0] = a_1, a_0

q[4] = Somma Half Adder

Il circuito risultante è il seguente:

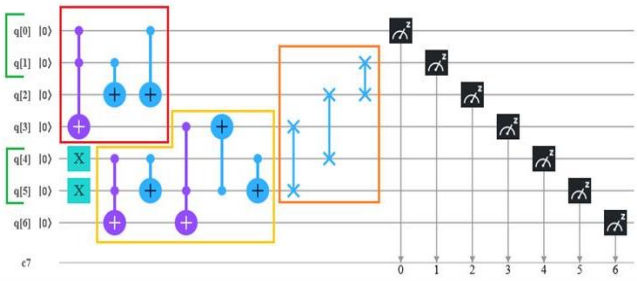


Figura 3

Nel nostro esempio abbiamo deciso di sommare |10⟩ e |10⟩ (in binario) quindi a_0=b_0=|0⟩ e a_1=b_1=|1⟩; da cui ci aspettiamo il risultato pari a |1001010⟩.

Anche in questo ultimo caso si verifica facilmente che il circuito è reversibile in quanto riapplicando le stesse porte si ottiene come stato finale |0000000⟩.

IV. RISULTATI

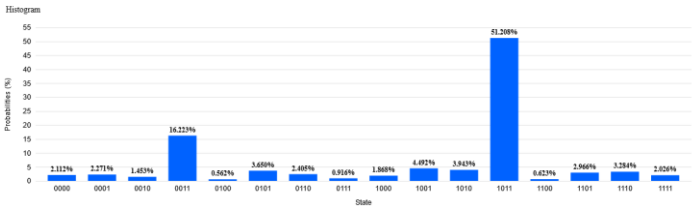
I circuiti discussi in precedenza sono stati eseguiti sia su Quantum Composer che su Qiskit attraverso notebook 'Jupyter'.

Per verificare che effettivamente i circuiti eseguano la logica richiesta, inizialmente, si sono utilizzati in ingresso soltanto dei qubit nello stato |0⟩ oppure |1⟩. Successivamente si è andato a modificare gli stati in ingresso (attraverso porte H, CNOT, Z, S, T) per evidenziare la capacità di questi circuiti quantistici di eseguire correttamente la logica anche in presenza di sovrapposizione degli stati e variazioni di fase.

A. *Risultati senza sovrapposizione di stati in ingresso*

Half Adder:

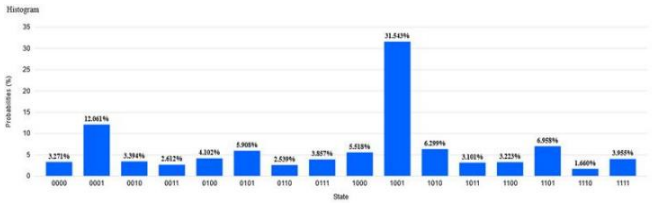
Il primo esempio riportato è quello dell'Half Adder in *Figura 1*. Eseguendolo sul quantum computer "ibmq_london" (con 1024 shots) si può notare che il risultato con maggiore probabilità (del 51.208%) è lo stato |1011⟩ come si era supposto in precedenza.



NB: per visualizzare al meglio il grafico si consiglia di visualizzare l'appendice C.

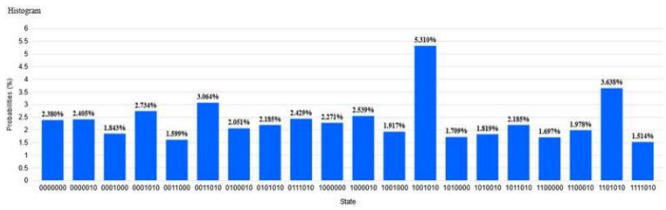
Full Adder:

L'esempio del Full Adder ha come circuito quello in *Figura 2*. Vengono sommati a_0=|1⟩, b_0=|0⟩ e il carry_in=|1⟩. Lo stato iniziale sarà |0101⟩ mentre ci si aspetta che quello finale sia |1001⟩. Questo esempio è stato eseguito sul "ibmq_16_melbourne v2.1.0" (con 4096 shots) e il risultato aspettato ha una probabilità del 31.543%:



Adder:

L'esempio dell'Adder è quello della somma di 2 qubits entrambi di valore 10 (2 in decimale) in *Figura 3*. Pertanto, lo stato iniziale è |0110000⟩, mentre quello finale che ci aspettiamo è |1001010⟩, dopo i dovuti swap. Conseguenza del rumore insito nel computer quantistico i risultati ottenuti contengono tutte le possibili combinazioni di stati; che sono 128 (cioè 2^7 poiché sono stati utilizzati 7 qubits). Quelli sottostanti sono quegli stati che hanno avuto la più alta probabilità di osservazione; si può notare come il risultato ottenuto con probabilità maggiore (5.310%) è effettivamente quello stato che ci si aspettava di osservare.



L'elevato rumore osservato nell'esecuzione di questo circuito, in confronto ai due esempi precedenti, è dovuto ad un più elevato numero di porte utilizzate e alla decoerenza. Quest'ultima è dovuta ad un più elevato tempo di esecuzione del singolo shot rispetto ai circuiti precedenti.

B. *Risultati con sovrapposizione di stati in ingresso*

Sull'Adder abbiamo eseguito un ulteriore esperimento. È stata effettuata una somma di qubits in sovrapposizione applicando Hadamard ad ogni ingresso quindi a_0, b_0 e a_1, b_1 possono essere contemporaneamente |00⟩, |01⟩, |10⟩ o |11⟩, con egual probabilità.

Il circuito sarà:

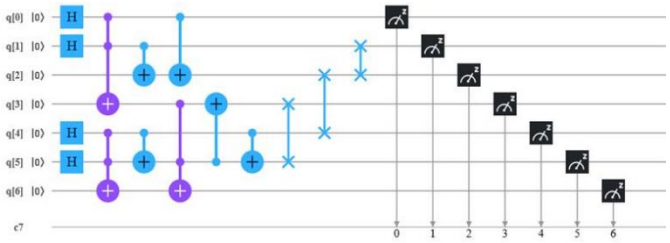
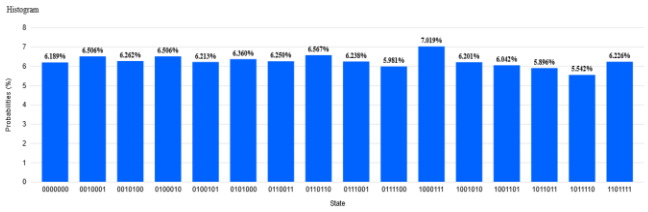


Figura 4

In questo caso non ci si aspetta di osservare un unico stato in uscita poiché ogni qubit in ingresso ha il 50% di possibilità di essere osservato nello stato |0⟩ o nello stato |1⟩. Per questo, ciò che ci si aspetta come risultato finale sono 16 stati (2^4 in quanto abbiamo applicato 4 porte di Hadamard ai 4 ingressi) che rappresentano la somma in binario di tutte le 16 combinazioni di ingressi che posso avere.

Visto l'elevato rumore ottenuto nell'esempio precedente si è ritenuto più consono simulare questo tipo di circuito sul "qasm_simulator"; anche perché dovendo osservare 16 risultati diversi con un rumore così elevato sarebbe stato impossibile vedere una netta differenza tra gli stati finali corretti e quelli affetti da errore. Magari, in futuro, quando si riuscirà a diminuire l'influenza del rumore si potranno effettivamente osservare questi 16 stati finali anche facendo girare il codice su un computer quantistico.

Simulando sul cloud di IBM ci si aspetta di vedere in uscita soltanto gli stati giusti, senza errore, con una probabilità pari a circa $1/2^4 = 6.25\%$. Infatti, ciò che si ottiene simulando con 1024 shots è:



V. CIRCUITI CON VARIAZIONE DI FASE DI UNO O PIÙ QUBITS IN INGRESSO

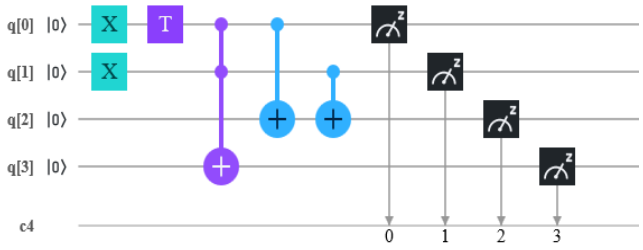
In questa tipologia di circuiti che è stata descritta si nota come la fase abbia un comportamento peculiare. Risulta che, dato anche un solo qubit in ingresso con fase diversa da 0 radianti, lo stato finale risulta essere lo stato che identifica la somma in binario dei qubits in ingresso (come descritto in precedenza) ma con fase pari a quella dell'unico qubit in ingresso che risultava sfasato. Questo comportamento sulla fase lo si è riscontrato utilizzando lo strumento "*Statevector*" presente sul Quantum Composer di IBM. Purtroppo, non è possibile visualizzare la fase utilizzando il nostro circuito dell'Adder a 7 qubits poiché lo strumento "*Statevector*" visualizza l'ampiezza di probabilità e la fase degli stati solo fino ad un massimo di 6 qubits utilizzati. Si può comunque visualizzare questo comportamento dividendo ed analizzando separatamente i

due circuiti costitutivi dell'Adder, cioè Half e Full Adder. Inoltre, vista la possibilità riportata in precedenza di costruire l'Half Adder con tre qubits, si è costruito un Adder a 6 qubits, perdendo però l'informazione di uno dei due qubits in ingresso dell'Half Adder visto che la somma viene sovrascritta su b_0.

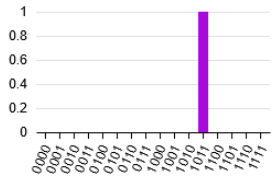
Qui di seguito vengono illustrati questi due procedimenti.

A. *Half Adder con un unico qubit con variazione di fase:*

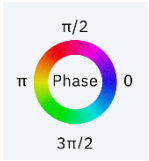
In questo circuito viene applicata una porta X sul primo qubit in ingresso così da portarlo nello stato 1 e successivamente viene applicata una porta T che va ad eseguire una rotazione di π/4 attorno all'asse *z*.



Lo strumento "*Statevector*" restituisce:

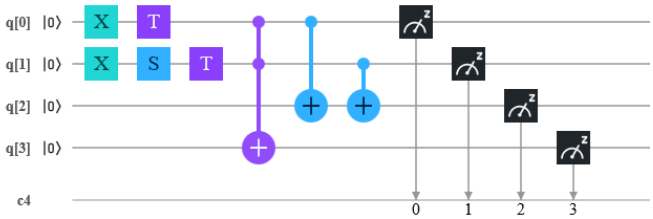


Dove l'altezza dell'istogramma identifica l'ampiezza di probabilità di ogni singolo stato mentre il colore identifica la fase dello stato. Il *Circuit Composer* ha la seguente codifica colori per rappresentare la fase:

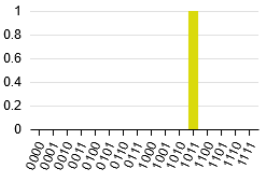


B. *Half Adder con più qubits con variazione di fase:*

In questo circuito viene aggiunto uno sfasamento anche sul secondo qubit applicando, dopo la porta X, una porta S ed una T; cioè viene ruotato di 3π/4. Il primo qubit è nella stessa configurazione del circuito precedente.

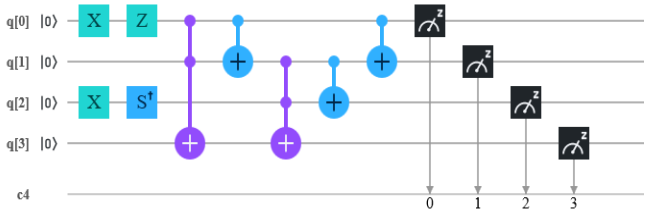


Da questo circuito è interessante notare come lo stato finale avrà una fase che sarà la somma delle fasi dei due qubits in ingresso, quindi sarà di: π/4 + 3π/4 = π. Ciò è confermato dallo "*Statevector*" infatti:

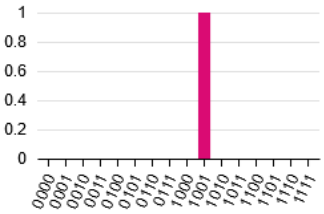


C. *Full Adder con qubit di carry in con variazione di fase:*

Questo circuito mostra come lo sfasamento del carry out dell'Half Adder (che in questo caso viene rappresentato con l'applicazione della porta S† sul q[2]) può andare a sommarsi agli sfasamenti dei qubits in ingresso del Full Adder e dare in uscita uno stato con una fase che sarà la somma di quelle degli ingressi e del carry in.

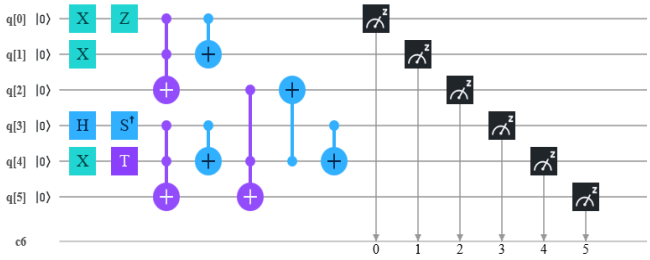


Lo stato finale, con fase π - π/2 = π/2, viene visualizzato da:

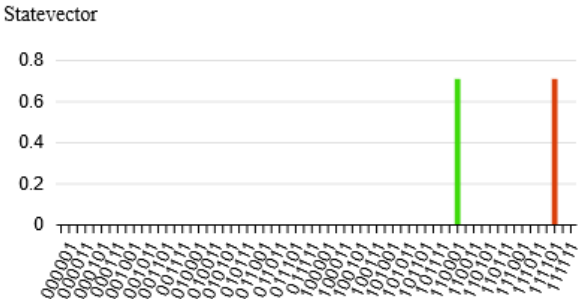


D. *Adder a 6 qubits:*

In questo circuito viene implementato l'Adder a 6 qubits per far notare che la fase dei qubits si somma anche se essi sono in ingresso del Full Adder o dell'Half Adder. Viene fatto notare, attraverso la porta H, come gli operatori di sfasamento attorno all'asse *z* non cambino la fase se il qubit è nello stato |0⟩.



Lo "*Statevector*" restituirà:



VI. CONSIDERAZIONI FINALI

Dall'esempio dell'Adder in cui venivano inseriti in ingresso 4 porte di Hadamard abbiamo notato come i risultati possibili siano 2^4 dove l'esponente è pari alle porte di Hadamard applicate. Estendendo questa considerazione al generale è possibile ottenere in uscita un numero di stati più probabili pari a 2^n, dove n è il numero di porte H applicate. Facendo un esempio sul "ibmq_16_melbourne" (computer quantistico a 15 qubits) è possibile compilare un circuito contenente 1 Half Adder a 4 qubits e 3 Full Adder utilizzando un totale di 13 qubits. In ingresso avremmo 8 qubits a cui sarà possibile applicare una porta di Hadamard perciò avremmo 256 (2^8) risultati della somma binaria (di valori a 4 qubits) egualmente probabili, con probabilità circa pari a $1/2^8$.

VII. REFERENZE

- Cloud IBM per circuiti, grafici ed immagini inerenti al Circuit Composer.
- Qiskit (IBM) e Jupyter Notebook per codici che seguono nell'appendice B.
- Quantum Computation and Quantum Information, M. A. Nielsen, I. L. Chuang, Cambridge University.
- Per Quantum Half Adder a 3 qubits: https://www.researchgate.net/publication/235083817_Quantum_Approaches_to_Logic_Circuit_Synthesis_and_Testing
- Per Quantum Full Adder: https://www.quantum-inspire.com/kbase/Full-Adder/
- Per scrittura codici: https://qiskit.org/documentation/stubs/
- Canale YouTube di Qiskit: https://www.youtube.com/channel/UClBNq7mCMf5xm8baE_VMl3A

Appendice A

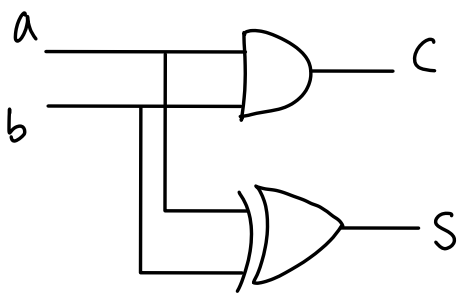
Derivazione HA, FA

Half Adder:

$c = ab$

$s = \bar{a}b + a\bar{b} = a \oplus b$

a	b	c	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Full Adder:

$c_{i+1} = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$

$= c_i(a \oplus b) + ab$

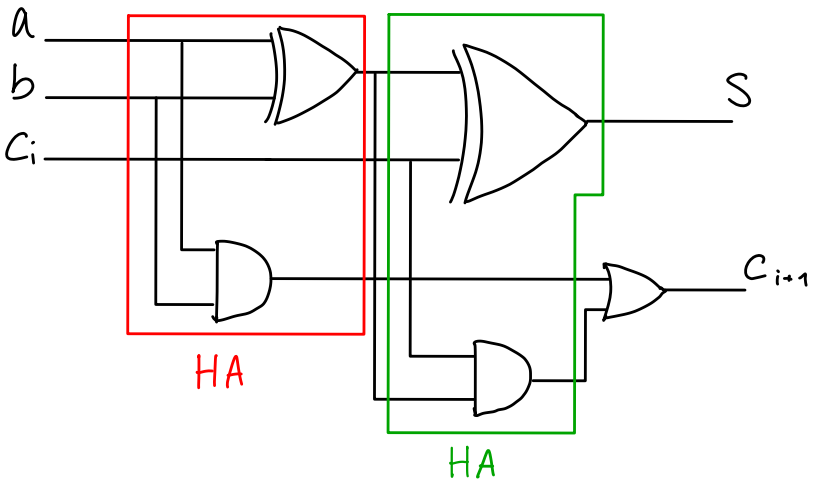
$s = \bar{a}\bar{b}c_i + \bar{a}b\overline{c_i} + a\bar{b}\overline{c_i} + abc_i$

$= \overline{c_i}(\bar{a}b + a\bar{b}) + c_i(\overline{a\bar{b}} + ab)$

$= \overline{c_i}(a \oplus b) + c_i$

$= c_i \oplus (a \oplus b)$

a	b	c_i	c_{i+1}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Appendice B
Codici

È bene segnalare che i codici di seguito rispecchiano i circuiti visti in esempio ma che possono variare in qualche porzione specifica. Ad esempio i backends utilizzati e il numero di shots lanciati. Questo per minimizzare le possibilità di errore quando i codici sono stati scritti.

```
#HALF ADDER
from qiskit import *
from qiskit import IBMQ
from qiskit.visualization import plot_histogram
get_ipython().run_line_magic('config', "InlineBackend.figure_format =
'svg' # Makes the images look nice")

qr = QuantumRegister(4)
cr = ClassicalRegister(4)

qcircuit = QuantumCircuit(qr, cr)

get_ipython().run_line_magic('matplotlib', 'inline')



#half adder circuit to add 1 and 1
qcircuit.x(qr[0])
qcircuit.x(qr[1])
#qcircuit.h(qr[2])
#qcircuit.cx(qr[0], qr[1])
qcircuit.ccx(qr[0], qr[1], qr[3]) #controllo, controllo, target
qcircuit.cx(qr[1], qr[2]) #controllo, target
qcircuit.cx(qr[0], qr[2])
qcircuit.measure(qr, cr)
qcircuit.draw(output = 'mpl')


#simulate it in local
simulator = Aer.get_backend('qasm_simulator')
result = execute(qcircuit, backend = simulator).result()
#from qiskit.tools.visualization import plot_histogram
plot_histogram(result.get_counts(qcircuit))



#load the IBM account to launch the circuit on a quantum computer
IBMQ.load_account() #load the account in order to execute the code on
the quantum computer


#select the backend
```

```
provider = IBMQ.get_provider('ibm-q')
qcomp = provider.get_backend('ibmq_burlington')
job = execute(qcircuit, backend = qcomp, shots = 8192)
from qiskit.tools.monitor import job_monitor
job_monitor(job)



#here we can see how the circuit is transpiled on the quantum computer
from qiskit.visualization import plot_circuit_layout

qc_transpiled = transpile(qcircuit, backend = qcomp,
optimization_level=3)
plot_circuit_layout(qc_transpiled, qcomp, view = 'virtual')




#and the error map, thus we can choose if continue with this backend or
change it
from qiskit.visualization import plot_error_map
plot_error_map(qcomp) #we plot the error map of the backend selected



#to see the results

qresult = job.result()
plot_histogram(qresult.get_counts(qcircuit))



#interactive plot

from qiskit.visualization import iplot_histogram
iplot_histogram(qresult.get_counts(qcircuit))
```

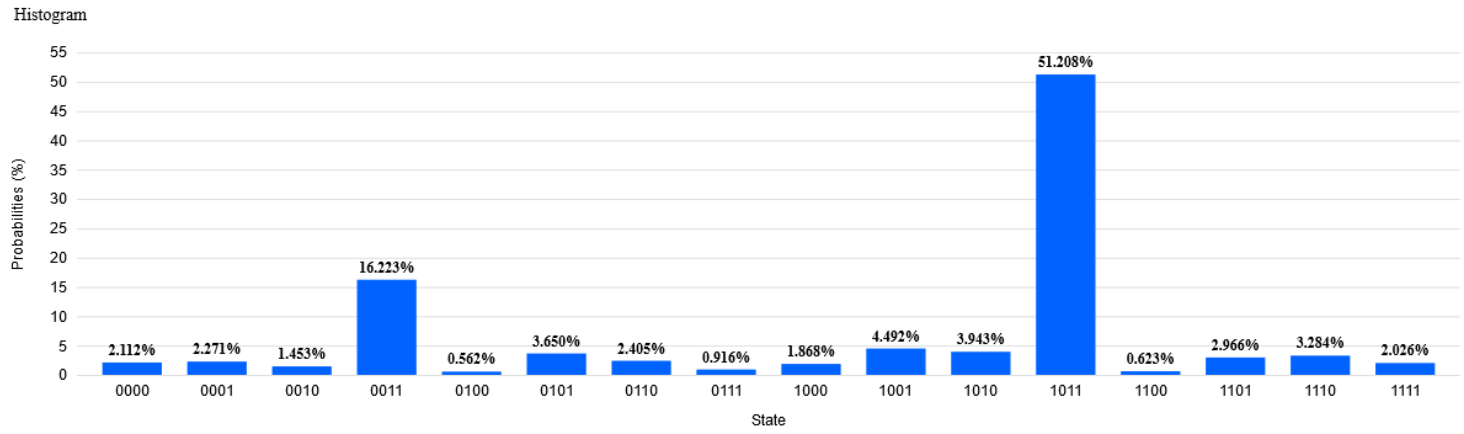
```
#FULL ADDER
from qiskit import *
from qiskit import IBMQ
from qiskit.visualization import plot_histogram, plot_bloch_vector
get_ipython().run_line_magic('config', "InlineBackend.figure_format =
'svg' # Makes the images look nice")

qr = QuantumRegister(4)
cr = ClassicalRegister(4)

qcircuit = QuantumCircuit(qr, cr)

get_ipython().run_line_magic('matplotlib', 'inline')

#our circuit to implement the sum of 1, 0 and a carry in equal to 1


qcircuit.x(qr[0]) #set to 1 the 1st addend
qcircuit.x(qr[2]) #set to 1 the carry in
qcircuit.ccx(qr[0], qr[1], qr[3]) #control, control, target (Toffoli
gate)
qcircuit.cx(qr[0], qr[1]) #control, target (CNOT gate)
qcircuit.ccx(qr[1], qr[2], qr[3])
qcircuit.cx(qr[1], qr[2])
qcircuit.cx(qr[0], qr[1])
qcircuit.measure(qr, cr)
qcircuit.draw(output = 'mpl')


#in local
simulator = Aer.get_backend('qasm_simulator')
result = execute(qcircuit, backend = simulator).result()
print(result.get_counts(qcircuit))
plot_histogram(result.get_counts(qcircuit))



#load the account in order to launch our circuit on a real quantum
computer
IBMQ.load_account() #load the account in order to execute the code on
the quantum computer



#get the provider and use a job monitor
provider = IBMQ.get_provider('ibm-q')
qcomp = provider.get_backend('ibmq_16_melbourne') #had to change from
imbq_london for more accuracy
job = execute(qcircuit, backend = qcomp, shots = 8192)
from qiskit.tools.monitor import job_monitor
job_monitor(job)


#see the results
qresult = job.result()
print(qresult.get_counts(qcircuit))
plot_histogram(qresult.get_counts(qcircuit))
```

```
#ADDER
from qiskit import *
from qiskit import IBMQ
from qiskit.visualization import plot_histogram, plot_bloch_vector
get_ipython().run_line_magic('config', "InlineBackend.figure_format =
'svg' # Makes the images look nice")

qr = QuantumRegister(7)
cr = ClassicalRegister(7)

qcircuit = QuantumCircuit(qr, cr)

get_ipython().run_line_magic('matplotlib', 'inline')


#this 4 statements are applied in order to see the superposition of all
qubits
#uncomment them to see the results
#qcircuit.h(qr[4])
#qcircuit.h(qr[5])
#qcircuit.h(qr[0])
#qcircuit.h(qr[1])

#sum of 10 and 10 in our quantum adder
qcircuit.x(qr[4])
qcircuit.x(qr[5])
qcircuit.ccx(qr[0], qr[1], qr[3]) #control, control, target
qcircuit.cx(qr[1], qr[2]) #control, target
qcircuit.cx(qr[0], qr[2])
qcircuit.ccx(qr[4], qr[5], qr[6])
qcircuit.cx(qr[4], qr[5])
qcircuit.ccx(qr[3], qr[5], qr[6])
qcircuit.cx(qr[5], qr[3])
qcircuit.cx(qr[4], qr[5])
qcircuit.swap(qr[3], qr[5])
qcircuit.swap(qr[2], qr[4])
qcircuit.swap(qr[1], qr[2])

#ENG
#with swaps I change my output in order to have a clear ouput
#starting from the MSQB (most significant Qbit)
#cout, somma full adder, somma half adder, b1, b0, a1, a0
#where a and b are the addends, the output is the match of carry out,
sum of FA and sum of HA

#IT
#con gli swap modifico la visione, avro' (partendo dal MSQb):
#cout, somma full adder, somma half adder, b1, b0, a1, a0
#dove gli a e b sono gli addendi e l'insieme di cout, somma FA e somma
HA sono il risultato finale

qcircuit.measure(qr, cr)
qcircuit.draw(output = 'mpl')


#launch it locally
simulator = Aer.get_backend('qasm_simulator')
```

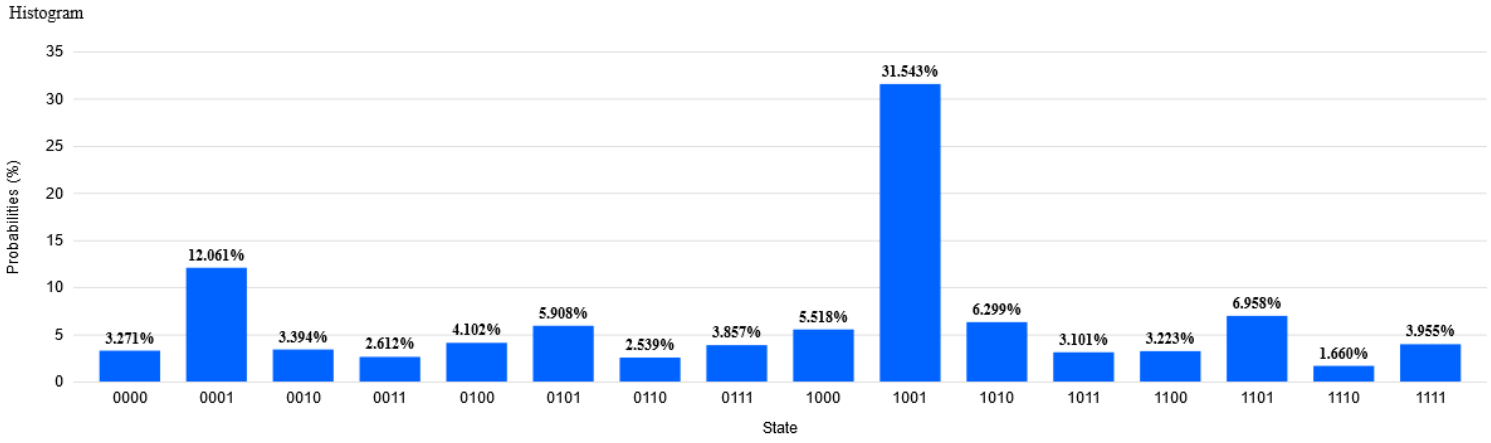
```
result = execute(qcircuit, backend = simulator).result()

plot_histogram(result.get_counts(qcircuit))


#load the IBM account
IBMQ.load_account() #load the account in order to execute the code on
the quantum computer


#choose the quantum computer where you will launch your circuit
provider = IBMQ.get_provider('ibm-q')
qcomp = provider.get_backend('ibmq_16_melbourne') #had to change from
imbq_london for more accuracy
job = execute(qcircuit, backend = qcomp, shots = 8192)
from qiskit.tools.monitor import job_monitor
job_monitor(job)


#to see how it is transpiled
from qiskit.visualization import plot_circuit_layout

qc_transpiled = transpile(qcircuit, backend = qcomp,
optimization_level=3)
plot_circuit_layout(qc_transpiled, qcomp, view = 'virtual')




#plot the error map and see if you have choose a quantum computer that
will satisfy your circuit
from qiskit.visualization import plot_error_map
plot_error_map(qcomp) #we plot the error map of the backend selected


#plot of the probabilities
qresult = job.result()
plot_histogram(qresult.get_counts(qcircuit))


#interactive plot
from qiskit.visualization import iplot_histogram
iplot_histogram(qresult.get_counts(qcircuit))
```

Appendice C
Grafici

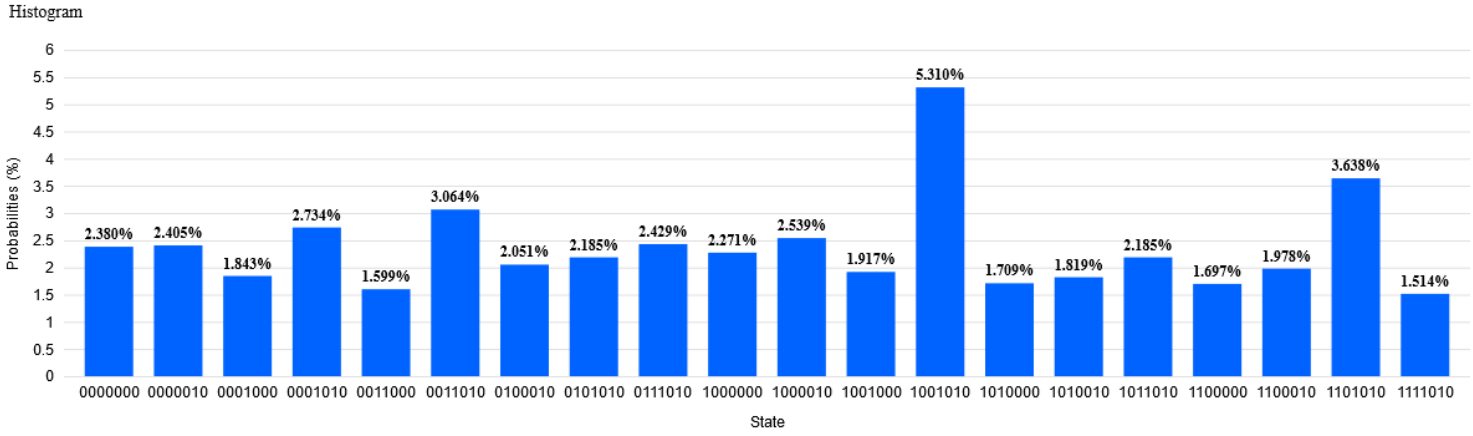
Istogramma dei risultati finali dell'Half Adder a 4 qubits (*Figura 1*):



Istogramma dei risultati finali del Full Adder a 4 qubits (*Figura 2*):



Istogramma dei risultati finali dell'Adder a 7 qubits (*Figura 3*):



Istogramma dei risultati finali dell'Adder a 7 qubits con applicati agli ingressi quattro porte Hadamard (*Figura 4*):