Hochschule Kaiserslautern
University of Applied Sciences

Alternative Entwicklungstools zu Qiskit - Cirq

Googles Quantum Computing Bibliothek für Python

Referent: Eric Brunk, 867437

Hochschule Kaiserslautern
University of Applied Sciences

Inhalt

- Was ist Cirq?
- Installation und Import
- NOT, CNOT und Hadamard
- Deutsch-Algorithmus
- Deutsch-Jozsa-Algorithmus

Hochschule Kaiserslautern
University of Applied Sciences

Was ist Cirq?

- Python Bibliothek von Google
- Aufbau und Optimierung von Quantenschaltkreisen
- Schaltkreise können auf Quantencomputern oder im Simulator ausgeführt werden

Quelle: [1]

Cirq ist eine Python Bibliothek von Google die es uns erlaubt Schaltkreise für den Quantencomputer zu erstellen, zu manipulieren und zu optimieren.
Die Schaltkreise können auf einem Quantencomputer oder im Simulator ausgeführt werden.

Hochschule Kaiserslautern
University of Applied Sciences

Installation und Import

```
pip install cirq
```

```
import cirq
```

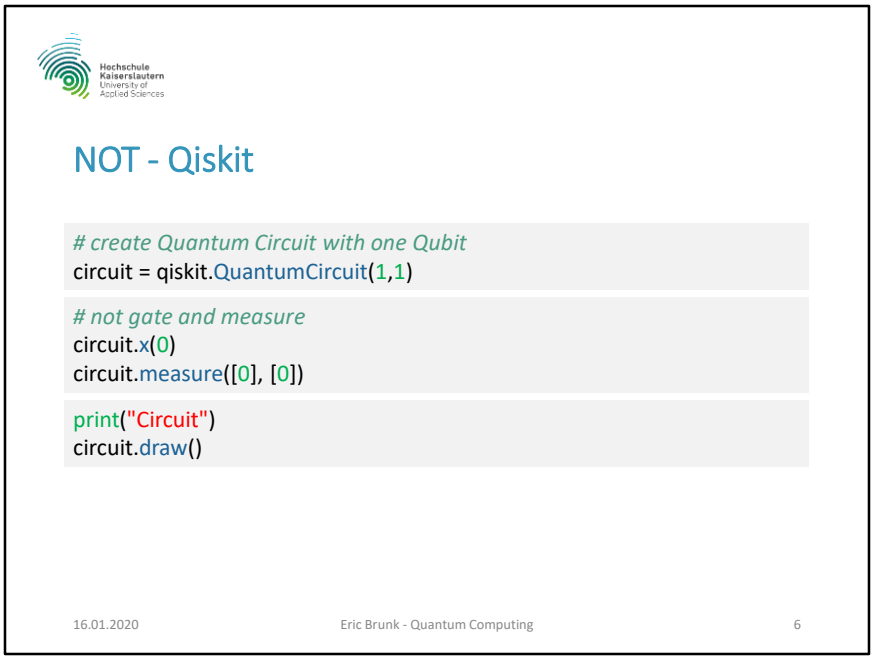

Die Installation erfolgt ganz einfach über pip.
Der Import gestaltet sich in Python auch sehr unkompliziert.

Hochschule Kaiserslautern
University of Applied Sciences

NOT

Quelle: [2]

Als ersten Schritt möchten wir versuchen einen Schaltkreis mit dem aus der Vorlesung bekannten NOT-Gatter und einer anschließenden Messung zu bauen.



Da Cirq sehr ähnlich zu IBM‘s Qiskit ist, wird der der Aufbau des Schaltkreises zuerst mit Qiskit gezeigt.

Es wird ein neuer QuantumCircuit erstellt. Der erste Parameter gibt die Anzahl der Qubits an. Der zweite Parameter die Anzahl der klassischen Bits, auf die die Messung später dargestellt wird.

Für das NOT-Gatter genügt der Einsatz der Methode x(). Als Parameter wird der Index des gewünschten Qubits angegeben.

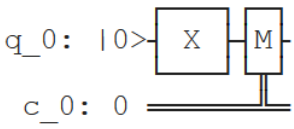
Für das Measure-Gatter wird die Methode measure() aufgerufen. Als ersten Parameter wird ein Array mit den Indizes der gewünschten Qubits angegeben, auf die das Gatter angewandt werden soll. Der zweite Parameter ist ein Array mit den Indizes der kalssischen Bits, auf denen die Messung ausgegeben werden soll.

Im Anschluss wird über die Methode draw() unser Schaltkreis gezeichnet.

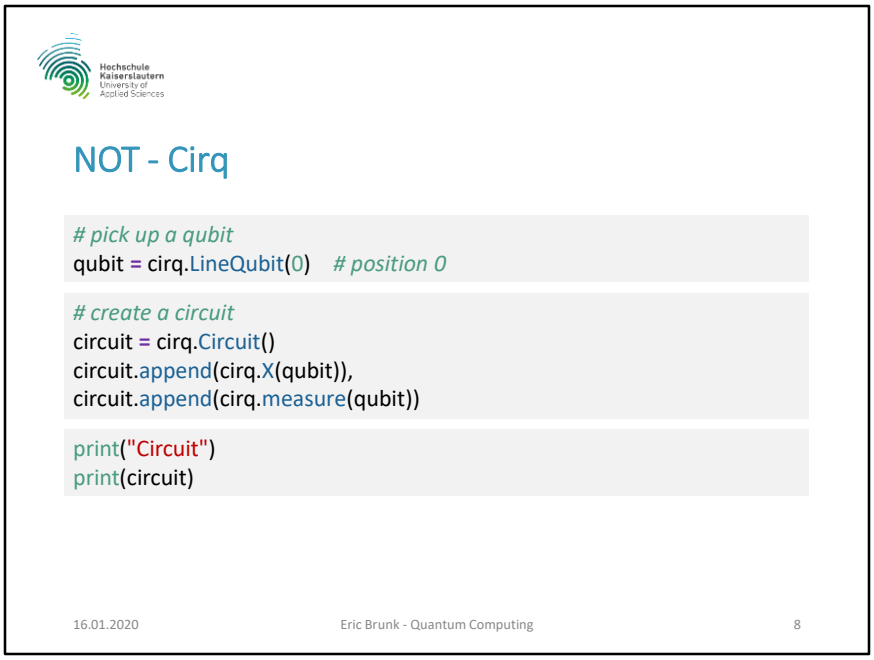
Hochschule Kaiserslautern
University of Applied Sciences

NOT - Qiskit

Circuit



Diese Folie stellt die Zeichnung des Schaltkreises in Qiskit dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.



Der gleiche Schaltkreis wird nun mit Cirq gebaut.

In Cirq können die Qubits über LineQubit() einzeln erstellt werden. Der Parameter gibt die Position im Schaltkreis an.

Im Anschluss wird zuerst der eigentliche Schaltkreis erstellt.

Jetzt können wir mit der Methode append() unsere gewünschten Gatter hinzufügen.

Bei Cirq ist das NOT-Gatter ein großes X(). Als Parameter wird das gewünschte Qubit angegeben. Für unsere Messung das Gleiche.

Unseren Schaltkreis können wir jetzt über print() ausgeben.

Hochschule Kaiserslautern
University of Applied Sciences

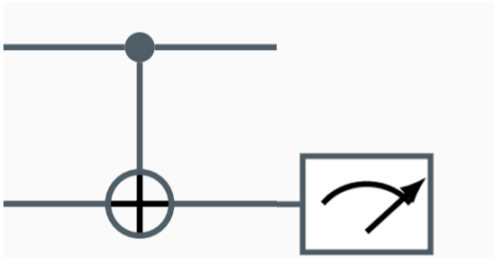
NOT - Cirq

```
Circuit
0: ──X──M──
```

Diese Folie stellt die Zeichnung des Schaltkreises in Cirq dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.

Hochschule Kaiserslautern
University of Applied Sciences

CNOT



Quelle: [2]

Der nächste Schaltkreis den wir uns anschauen möchten ist CNOT mit anschließender Messung.

Die Vorgehensweise in Qiskit ist dabei fast identisch zu der für den vorherigen Schaltkreis.

Allerdings benötigen wir nun zwei Qubits und nur ein klassisches Bit, da auch nur ein Qubit gemessen wird.

Das CNOT-Gatter sprechen wir über cx() an. Der erste Parameter gibt das sogenannte controll-bit an, der zweite das sogenannte target-bit.

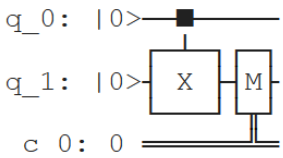
Anschließend wird nur das target-bit gemessen.

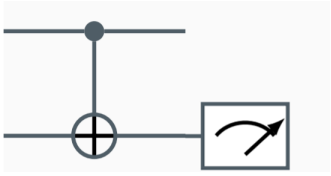
Wir zeichnen wieder unseren Schaltkreis.

Hochschule Kaiserslautern
University of Applied Sciences

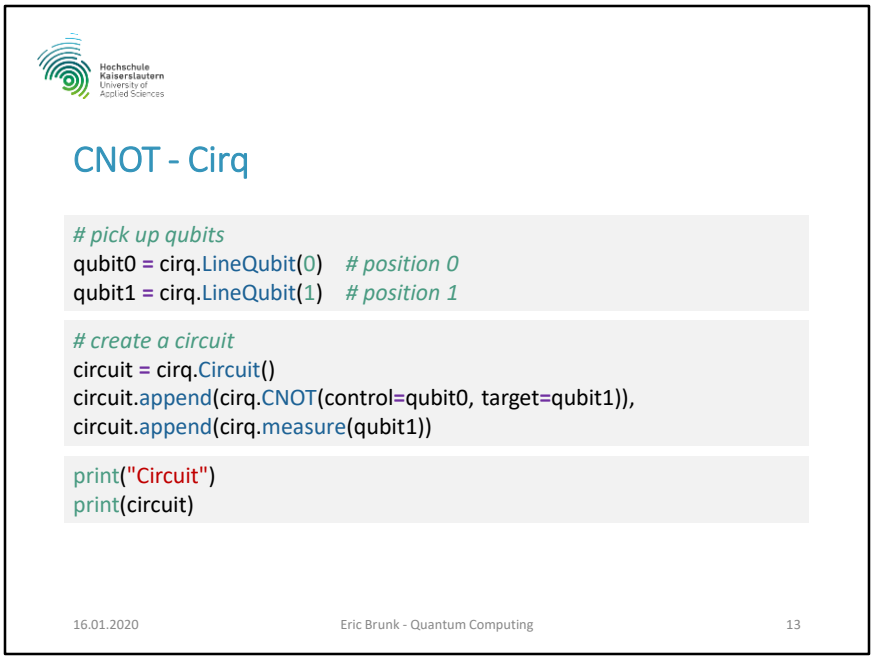
CNOT - Qiskit

Circuit





Diese Folie stellt die Zeichnung des Schaltkreises in Qiskit dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.



Den gleichen Schaltkreis möchten wir wieder in Cirq nachbauen.

Hierfür erstellen wir uns zwei Qubits.

Auch hier ist der Vorgang fast identisch zum Beispiel mit dem NOT-Gatter.

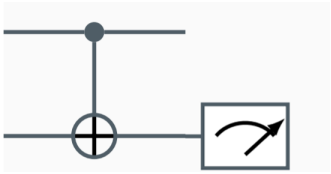
Für Controlled Not wird jetzt die Methode CNOT() augerufen. Die Parameter sind hier dementsprechend benannt.

Auch diesen Schaltkreis geben wir wieder aus.

Hochschule Kaiserslautern
University of Applied Sciences

CNOT - Cirq

```
Circuit
0: ──@──────
     │
1: ──X──M──
```

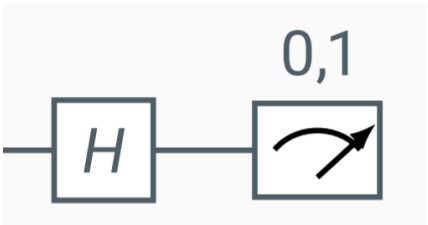


16.01.2020 Eric Brunk - Quantum Computing 14

Diese Folie stellt die Zeichnung des Schaltkreises in Cirq dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.

Hochschule Kaiserslautern
University of Applied Sciences

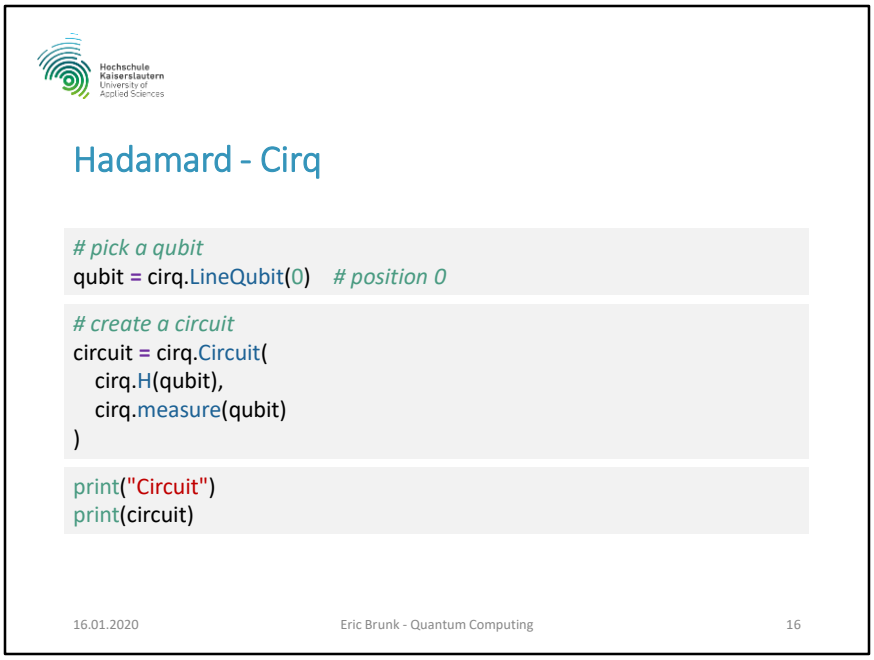
Hadamard



Quelle: [2]

Jetzt möchten wir uns einen Schaltkreis mit dem bekannten Hadamard-Gatter mit anschließender Messung anschauen.

Da mittlerweile klar sein sollte dass Qiskit sehr ähnlich zu Cirq ist, wird darauf verzichtet die Qiskit Beispiele auch zu implementieren.



Wir erzeugen wieder ein Qubit und unseren Circuit.

Hier sehen wir, dass wir die gewünschten Gatter auch direkt in die Klammern des Circuits schreiben können um sie hinzuzufügen.

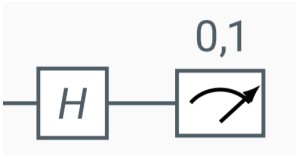
Wir fügen Hadamard mit H() und eine anschließende Messung hinzu.

Danach geben wir den Schaltkreis wieder aus.

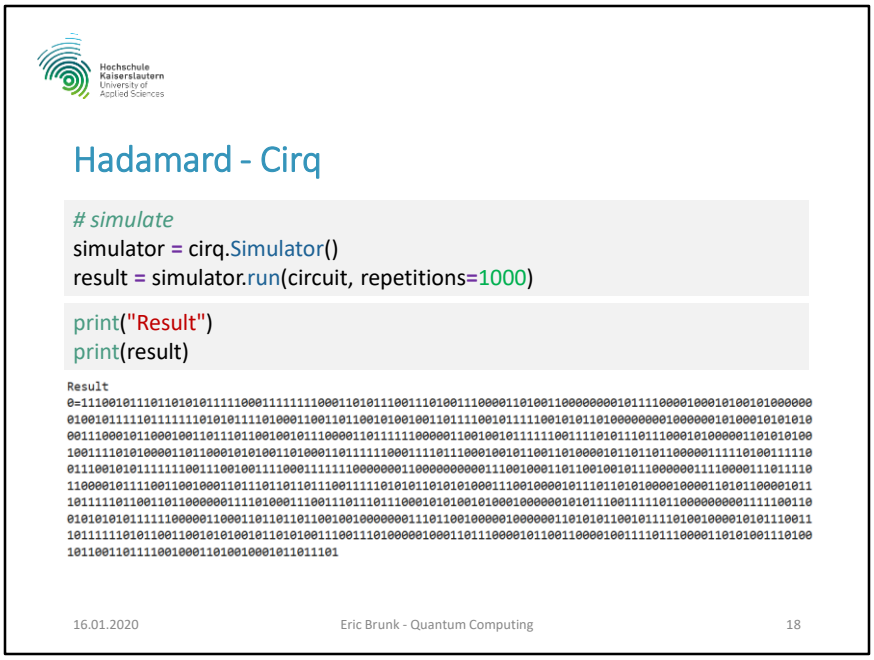
Hochschule Kaiserslautern
University of Applied Sciences

Hadamard - Cirq

```
Circuit
0: ──H──M──
```



Diese Folie stellt die Zeichnung des Schaltkreises in Cirq dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.



Da das Hadamard-Gatter den Zustand unseres Qubits in eine Superposition überführt interessiert uns natürlich unsere Messung.

Unseren Schaltkreis können wir in einem Simulator laufen lassen.

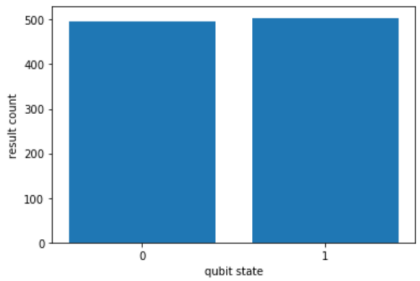
Als Anzahl an Wiederholungen wurde hier 1000 gewählt.

Unser Ergebnis zeigen 1000 Werte mit einer Verteilung von 0 und 1.

Hochschule Kaiserslautern
University of Applied Sciences

Hadamard - Cirq

```
# histogram
cirq.plot_state_histogram(result)
```




```
array([496., 504.])
```

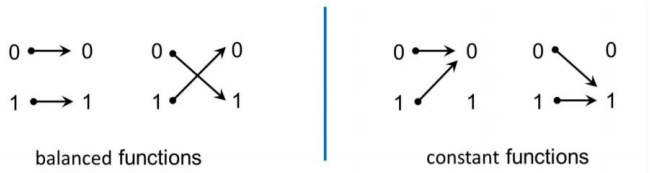
16.01.2020 Eric Brunk - Quantum Computing 19

Um unter Ergebnis auch plotten zu können verwenden wir die von cirq mitgelieferte Methode plot_state_histogram().

Wir sehen in unserem Diagramm, dass wir eine ungefähre Verteilung von 50/50 haben.

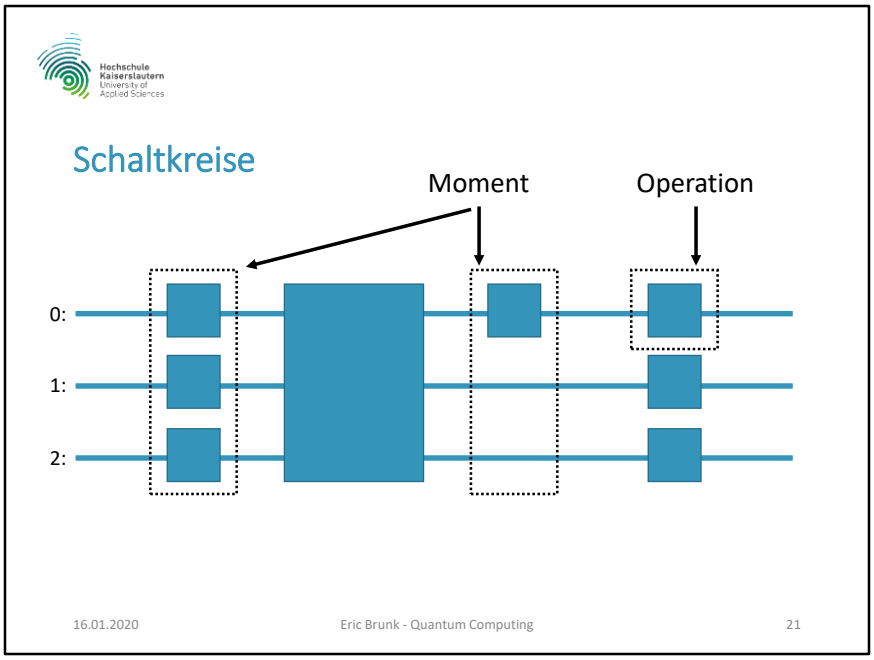
Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus



Quelle: [2]

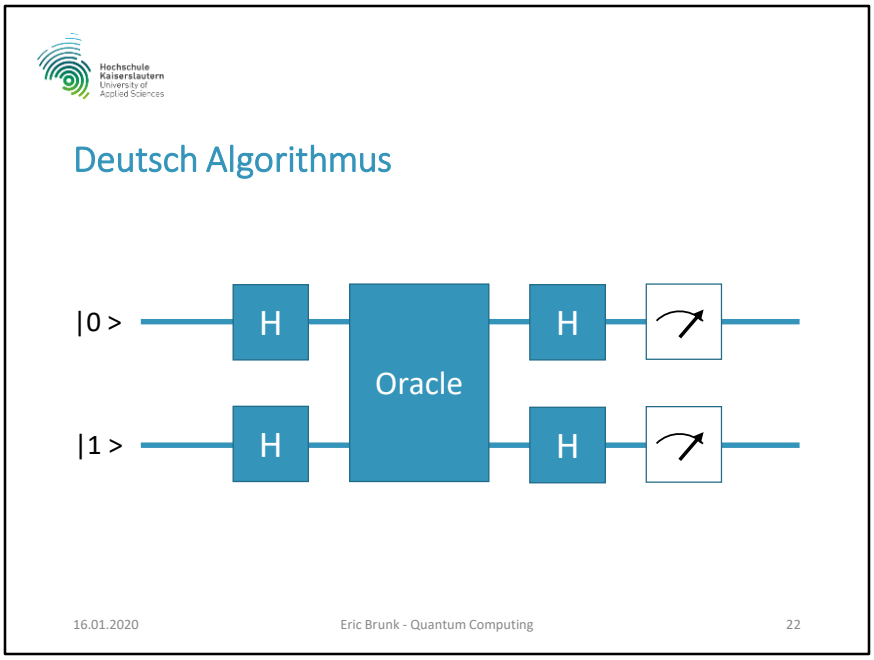
Der Deutsch-Algorithmus kann mit nur einem Durchlauf erkennen, ob eine Funktion balanciert oder konstant ist.
Klassische Rechner benötigen dafür mindestens zwei oder mehr Durchläufe.



Ein Schalkreis in Cirq kann in sogeannnte Moments und Operations unterteilt werden.

Ein Moment spezifiziert eine Sammlung von Gattern zu einem gewissen Moment.

Operations sind die eigentlichen Gatter.



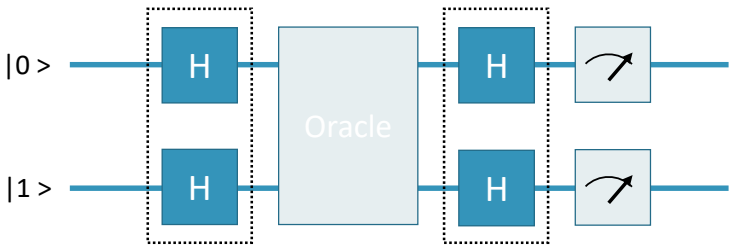
Diese Folie zeigt, wie der Schaltkreis für den Deutsch-Algorithmus aufgebaut ist.

Wir haben zwei Qubits, eines im Zustand 0 und eines im Zustand 1.
Beide Qubits werden durch ein Hadamard-Gatter in die Superposition überführt.
Danach folgt die eigentliche Funktion, Oracle-Funktion genannt, die entweder balanciert oder eben konstant ist.
Anschließend werden beide Qubits wieder jeweils durch ein Hadamard-Gatter geschickt.
Am Ende werden beide Qubits gemessen.

Das zweite Hadamard-Gatter und die Messung sind beim zweiten Qubit eigentlich überflüssig.
Der Vollständigkeit halber wurden sie aber hinzugefügt.

Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus

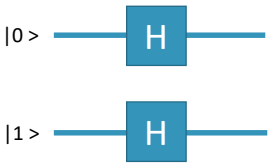


16.01.2020 Eric Brunk - Quantum Computing 23

Der Schaltkreis kann unterteilt werden in Hadamard-Moment, Oracle-Funktion und Mess-Moment.

Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus




```
# create hadamard moment
moment_h_all = cirq.Moment( [cirq.H(qubit0), cirq.H(qubit1)] )
```

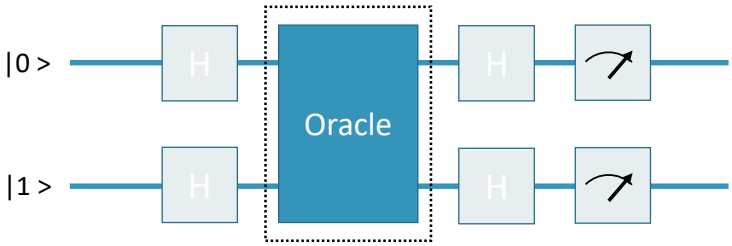

Diese Folie zeigt die Implementierung des Hadamard-Moments.

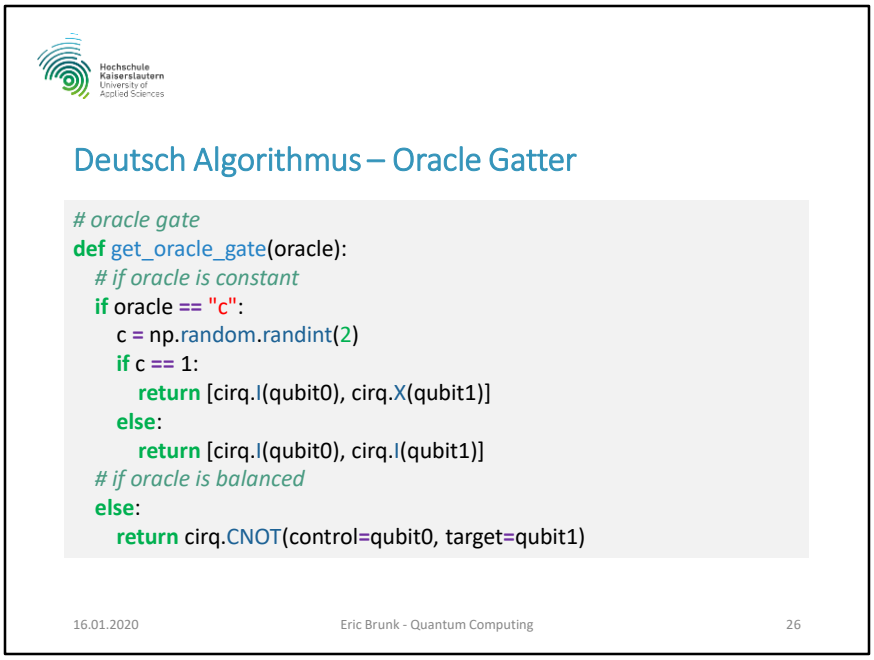
Es wird ein neuer Moment erstellt, dem ein Array an Hadamard-Gattern zugewiesen wird.

Dieser Moment kann beliebig oft wiederverwendet werden.

Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus





Diese Folie zeigt die Implementierung der Oracle-Funktion für den Deutsch-Algorithmus.

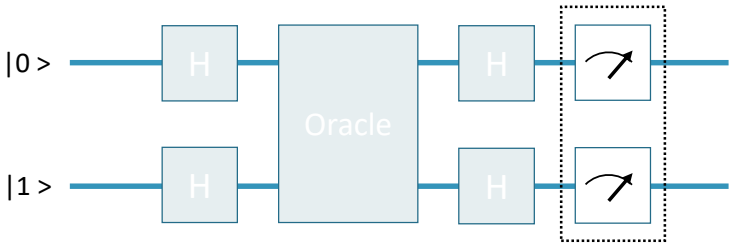
Es wird die Methode get_oracle_gate(oracle) erstellt die einen Parameter („c“ für konstant oder „b“ für balanced) entgegen nimmt.

Wenn ein „c“ übergeben wird, gibt die Methode abhängig von einer generierten Zufallszahl entweder das Gatter [Identity, NOT] oder [Identity, Identity] zurück.

Wird ein „b“ übergeben gibt die Methode eine Möglichkeit einer balancierten Funktion in Form von CNOT zurück.

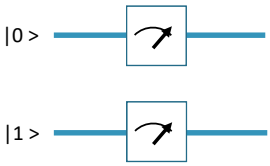
Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus



Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus




```
# create measure moment
moment_m_all = cirq.Moment( [cirq.measure(qubit0), cirq.measure(qubit1)] )
```


Diese Folie zeigt die Implementierung des Mess-Moments.

Es wird ein neuer Moment erstellt, dem ein Array an Messungen zugewiesen wird.

Auch dieser Moment kann beliebig oft wiederverwendet werden.

Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus – Schaltkreis

```
# circuit
circuit = cirq.Circuit()
circuit.append(cirq.X(qubit1))
circuit.append(moment_h_all)
# insert oracle gate
circuit.append(get_oracle_gate("b"))

circuit.append(moment_h_all)
circuit.append(moment_m_all)
```

```
print("Result")
print(result)
```


Diese Folie zeigt, wie die einzelnen Moments und die Oracle-Funktion zu einem Schaltkreis zusammengebaut werden.

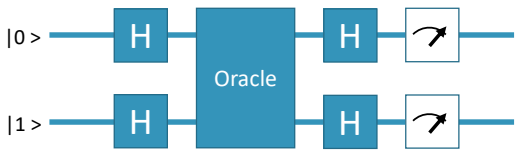
Zuerst muss das letzte Qubit mit dem NOT-Gatter in den Zustand 1 gebracht werden.

Danach folgen die eigentlichen Moments bzw. die Oracle-Funktion.

Am Ende geben wir unseren Schaltkreis wieder aus.

Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus – Schaltkreis




```
Circuit
0: ———H———@———H———M———
          |
1: ———X———H———X———H———M———
```

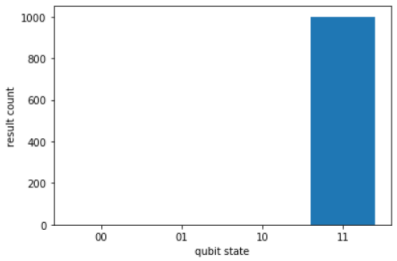
16.01.2020 Eric Brunk - Quantum Computing 30

Diese Folie zeigt zum Vergleich:

Oben -> Schaltkreis den wir erreichen wollten
Unten -> Schaltkreis den uns Cirq ausgibt.

Deutsch Algorithmus – Histogram

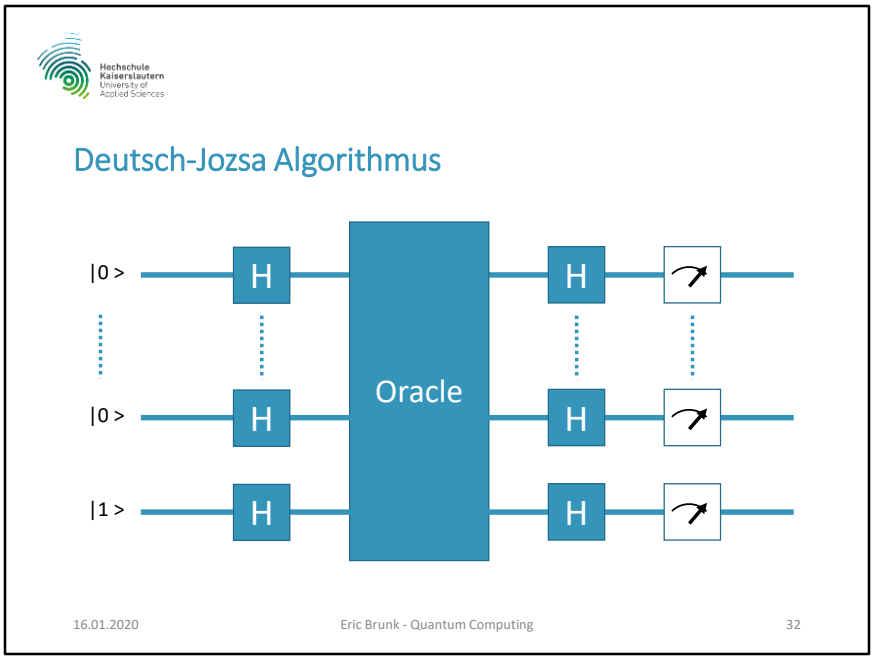
```
# simulate and plot
simulator = cirq.Simulator()
result = simulator.run(circuit, repetitions=1000)
cirq.plot_state_histogram(result)
```




```
array([   0.,    0.,    0., 1000.])
```


Unseren Schaltkreis wollen wir natürlich auch simulieren.

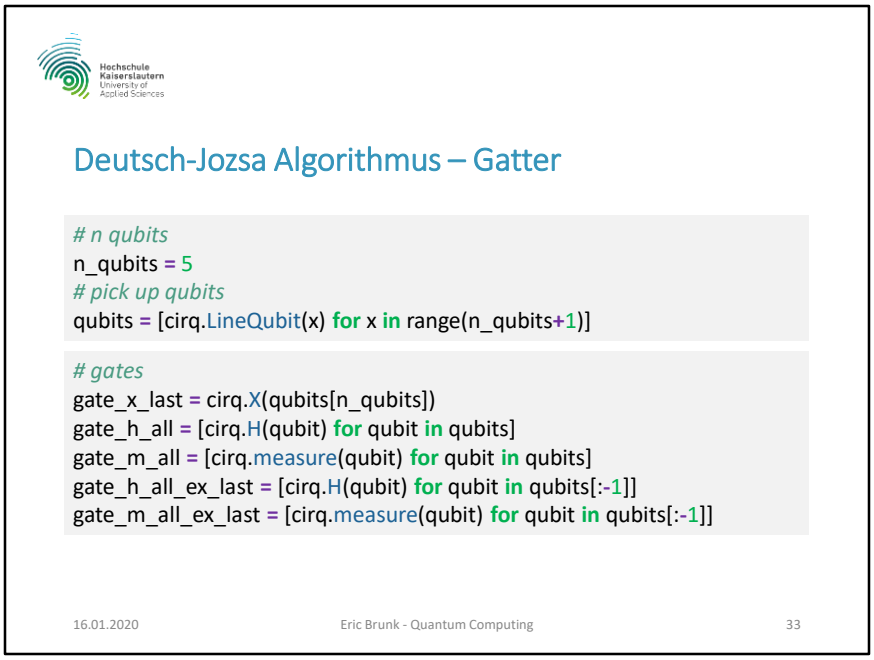
Da wir eine balancierte Funktion in den Schaltkreis eingefügt haben, gibt uns der Schaltkreis eine 100% Verteilung von 11 zurück.



Der Deutsch-Jozsa-Algorithmus erweitert den Deutsch-Algortihmus dadurch dass wir das gleiche Vorgehen nun mit n-Qubits erreichen wollen.

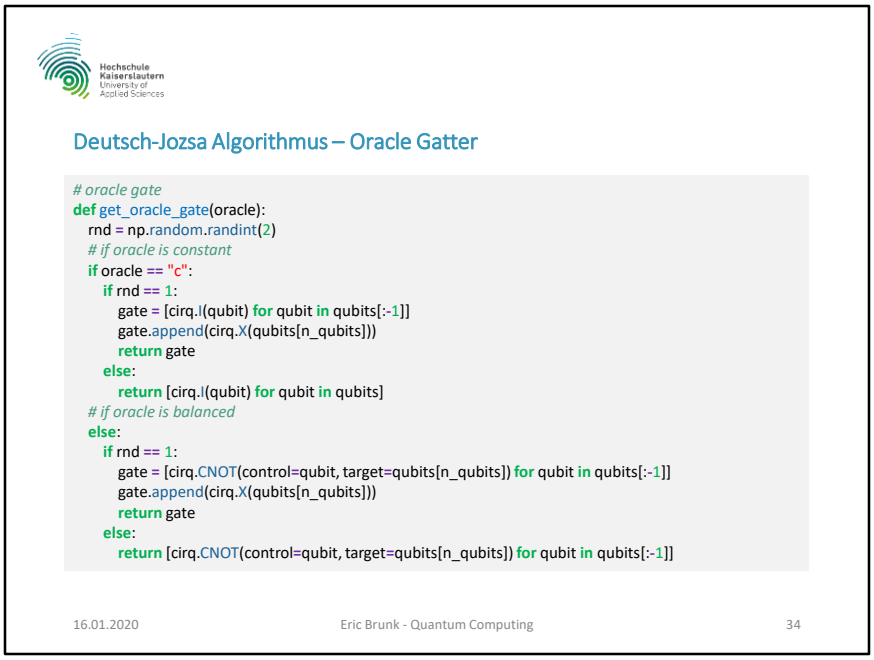
Diese Folie zeigt ungefähr, wie solch ein Schaltkreis aussehen sollte.

Auch hier kann das Hadamard-Gatter und die Messung des letzten Qubits weggelassen werden.



Die vorherige Implementierung wird dadurch erweitert, dass die Anzahl an Qubits über ein Array, abhängig der Variable n_qubits erfasst wird.

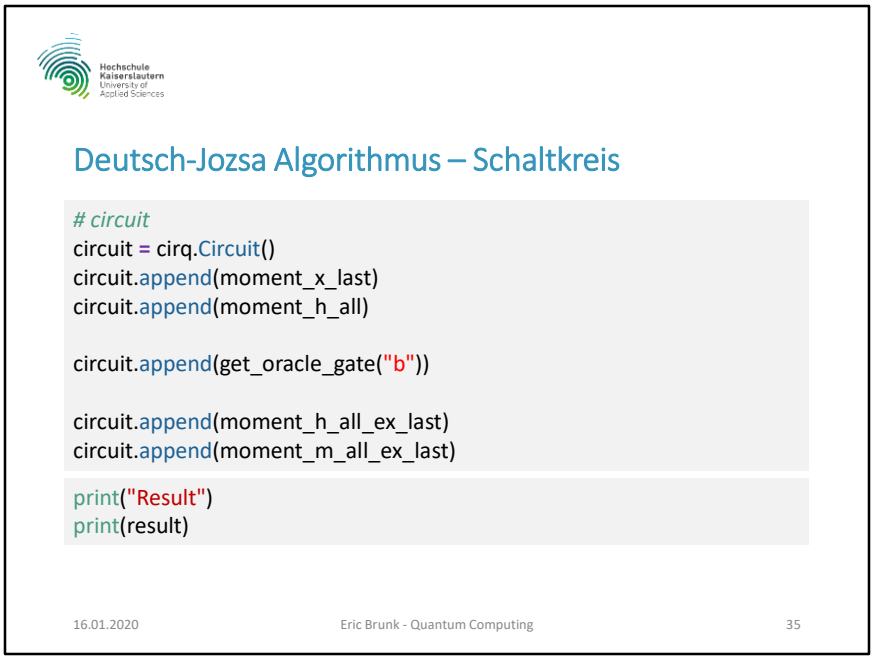
Die eigentlichen Gatter können dann auf die gewünschten Einträge des Arrays angewandt und in eine Variable gespeichert werden.



Diese Folie zeigt die Implementierung der Oracle-Funktion für den Deutsch-Jozsa-Algorithmus.

Der hauptsächliche Unterschied hierbei besteht einfach darin, dass die Gatter für die konstante und die balancierte Funktionen eben auf das Array von Qubits und nicht wie vorher auf zwei spezifische Qubits angewandt werden.

Bei den balancierten Funktionen sind in diesem Beispiel nur zwei Möglichkeiten implementiert.

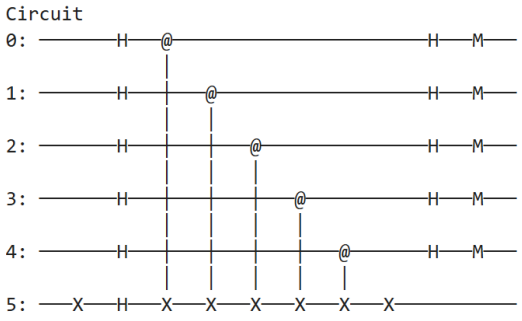


Auch hier können die verschiedenen Moments einfach im Circuit zusammengeführt werden.

Danach erfolgt wieder die Ausgabe des Schaltkreises.

Hochschule Kaiserslautern
University of Applied Sciences

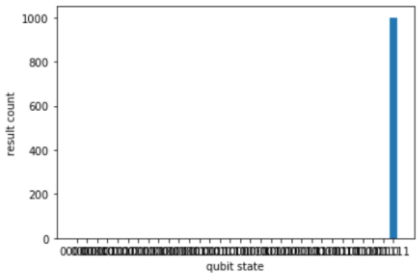
Deutsch-Jozsa Algorithmus – Schaltkreis



Diese Folie zeigt die Ausgabe des Schaltkreises für die DJ-Algorithmus am Beispiel von einer balancierten Funktion in Form von CNOT.

Hochschule Kaiserslautern
University of Applied Sciences

Deutsch Algorithmus – Histogram




```
array([   0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0., 1000.])
```

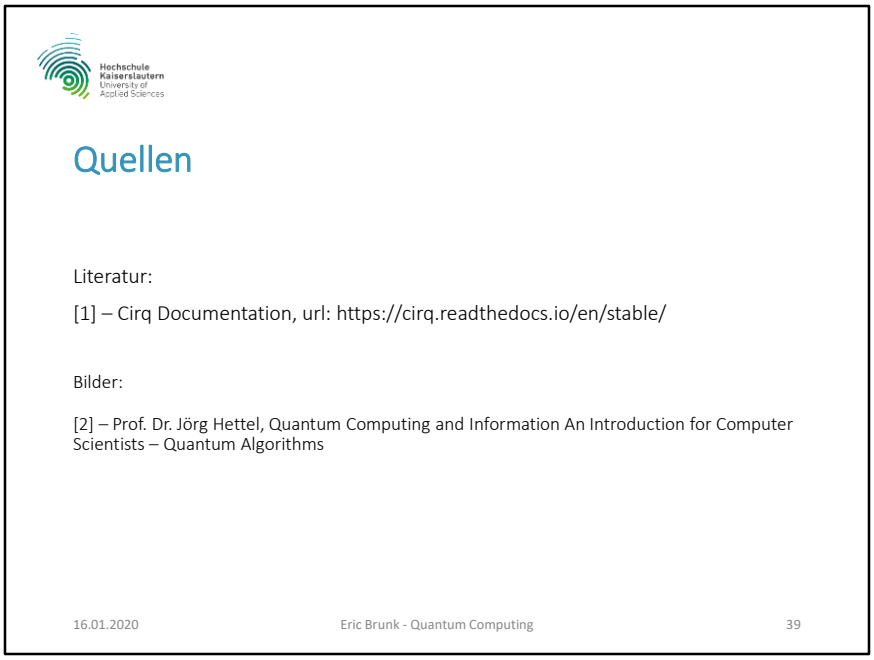

Das dazu gehörige Histogram zeigt auch hier eine 100%-Verteilung des erwarteten Wertes bei 5 Qubits.

Hochschule Kaiserslautern
University of Applied Sciences

Git-Repository

Die hier gezeigte Implementierung und die gesamte Präsentation befindet sich in einem Git-Repository unter dem folgenden Link:

https://github.com/erxn91/quantum_computing



Alle Informationen über Cirq stammen aus der Dokumentation der Bibliothek.

Die mit [2] gekennzeichneten Bilder stammen aus dem Skript von Prof. Dr. Jörg Hettel – Quantum Computing and Information, Kapitel Quantum Algorithms.

Hochschule Kaiserslautern
University of Applied Sciences

Fragen?