

Ausarbeitung

**Werkzeuge für die Algorithmenimplementierung auf
Quantencomputern**

**im Rahmen der
Projektarbeit-II**

Timo Grautstück

Betreuer: *Prof. Dr.-Ing. Ulf Niemeyer*

Fachbereich: *Informations- und Kommunikationstechnik*

Abstract

Um sichere Datenübertragung zu gewährleisten werden häufig asymmetrische Verschlüsselungsverfahren eingesetzt. Aktuell weit verbreitete sowie praktisch eingesetzte Verfahren basieren auf harten mathematischen Problemen wie der Faktorisierung ganzer Zahlen oder dem Berechnen diskreter Logarithmen. Diese Probleme lassen sich nicht effizient durch konventionelle Algorithmen auf Digitalrechnern lösen. Dies gilt jedoch nicht für Quantencomputer, durch vielversprechende Quantenalgorithmen wie dem von Shors erhofft man sich diese harten Probleme in Polynomialzeit zu lösen. Dies ist einer der schwerwiegendsten Gründe, warum Wirtschaft und Wissenschaftler quantensichere Kryptographie sowie Quantencomputer vorantreiben wollen. In dieser Arbeit sollen Grundlagen, sowie die Funktionsweise von Quantencomputern dargestellt werden. Darunter fällt auch die Simulation und Programmierung von Quantenschaltungen, sowie eine Darstellung von Quantenalgorithmen die einen Bezug zur modernen Kryptographie aufweisen.

Inhaltsverzeichnis

1. Einleitung	1
2. Grundlagen Quantencomputer	2
2.1. Quantenbits	2
2.2. Multiple Quantenbits	4
2.3. Quantengatter	5
2.4. Quantenschaltungen	9
3. Programmierung von Quantencomputer	11
3.1. Software und Programmiersprachen	11
3.2. Simulation	12
3.3. Quantum Hardware	13
4. Quantenalgorithmen	14
4.1. Algorithmenüberblick	14
4.2. Quanten Fourier-Transformation	15
4.3. Quanten-Phasenschätzung	16
4.4. Shors Algorithmus	18
5. Zusammenfassung	20
A. Anhang	21
B. Literaturverzeichnis	24

1. Einleitung

Durch die Akzeptanz von Sichtweisen und Phänomenen, die sich von den klassischen und bisher bekannten unterscheiden, wurde uns die Möglichkeit geschaffen Computer und Algorithmen zu entwickeln die Leistungsstärker als klassische sein können. Diese Sichtweise ermöglicht Informationsverarbeitung wie sie klassisch nie möglich war. Quantenmechanik, der Kern der Quanteninformationsverarbeitung, Quantencomputern und der Quantenkryptographie. Der Grund zum effizienten Finden von Perioden in Funktionen, wodurch die Faktorisierung von ganzen Zahlen in $\mathcal{O}(n^3)$ gelöst werden kann. Dank dieser Akzeptanz konnte die Quantenmechanik immer weiter entwickelt und ausgereift werden, sodass heutzutage Quantenprozessoren mit 127 Qubits existieren und Prozessoren mit über 1.000 Qubits entwickelt werden.

2. Grundlagen Quantencomputer

2.1. Quantenbits

Viele Jahre wissenschaftlicher und technologischer Fortschritt haben zur Entwicklung des Quantenbits, kurz *Qubit* beigetragen. Genau wie das klassische Bit, die kleinste Maßeinheit zur Darstellung von Informationsgehalt, kann auch das Quantenbit die Zustände 1 und 0 annehmen. Wesentlicher Unterschied zu einem klassischen Bit ist, dass es sich bei einem Qubit um ein Zweizustandssystem handelt, d.h. es kann sich zu einer gewissen Wahrscheinlichkeit in einem dieser beiden Zustände 1 oder 0 befinden. Hierbei ist wichtig, dass der Zustand dieses Quantenbits nur dann in Erfahrung gebracht werden kann, indem es gemessen wird.

Um den Zustand eines Qubits darzustellen, werden Zustandsvektoren genutzt. Um diese Zustände mathematisch zu veranschaulichen wird die sogenannte Dirac Notation $|\rangle$ genutzt, diese ist die standard Notation, um in der Quantenmechanik Zustände darzustellen.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (2.1)$$

Somit zeigt 2.1 die Basiszustände (*Computational basis state*) $|0\rangle$ und $|1\rangle$, welche eine orthogonale Basis bilden [1]. Es ist jedoch auch möglich, dass sich ein Quantenbit in einem Zustand befindet, der sich von diesen beiden unterscheidet.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \alpha, \beta \in \mathbb{C} \quad (2.2)$$

Das heißt auch Linearkombinationen können aus diesen Zuständen gebildet werden 2.2. Dies wird als Superposition oder auch Überlagerung bezeichnet, wobei α und β die Amplituden des Zustands $|\psi\rangle$ darstellen [1].

Um zu erfahren, zu welcher Wahrscheinlichkeit sich ein Quantenbit in einem bestimmten Zustand befindet, muss das Qubit gemessen werden. Für den Zustandsvektor $|\psi\rangle$ erhält man somit nach der Messung, mit der Wahrscheinlichkeit $|\alpha|^2$ das Ergebnis 0, und mit der Wahrscheinlichkeit $|\beta|^2$ das Ergebnis 1 [1].

$$\begin{aligned} p(|0\rangle) &= |\langle 0|\psi\rangle|^2 \\ \Rightarrow |\alpha\langle 0|0\rangle|^2 + |\beta\langle 0|1\rangle|^2 & \\ &= |\alpha|^2 \end{aligned} \quad (2.3)$$

Die Messung wird durchgeführt, indem das innere Produkt über den Zustandsvektor und den zugehörigen Basiszustand gebildet und quadriert wird 2.3. Um eine Wahrscheinlichkeit von 1 zu gewährleisten, muss der Zustandsvektor normiert sein. Somit muss 2.4 für den Zustandsvektor gelten.

$$\begin{aligned} \langle\psi|\psi\rangle &= 1 \\ \Rightarrow |\alpha|^2 + |\beta|^2 &= 1 \end{aligned} \quad (2.4)$$

Vor der Messung eines Qubits kann es sich in einem Kontinuum an Zuständen zwischen $|0\rangle$ und $|1\rangle$ befinden [1]. Das ermöglicht einem Quantenbit, sich in den Zuständen $|0\rangle$ und $|1\rangle$ gleichzeitig zu befinden. Auf diesen Zustandsvektor wird oft zurückgegriffen, 2.5 soll diesen verdeutlichen.

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \quad (2.5)$$

Da der Zustandsvektor normalisiert sein muss 2.4, ist es möglich die allgemeine Darstellung eines Quantenbits 2.2 mit Hilfe des Additionstheorems $\sqrt{\sin(x)^2 + \cos(x)^2} = 1$ zu vereinfachen.

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \quad \theta, \phi \in \mathbf{R} \quad (2.6)$$

Zustandsvektor $|+\rangle$ lässt sich somit durch $\phi = 0$ und $\theta = \frac{\pi}{2}$ darstellen. 2.6 wird als Blochvektor bezeichnet, jeder Blochvektor lässt sich als Punkt auf einer dreidimensionalen Kugel (Bloch-Kugel) durch die Kugelkoordinaten ϕ und θ mit einem Radius von $r = 1$ darstellen. Die Bloch-Kugel ist ein Unterraum des Hilbertraums.

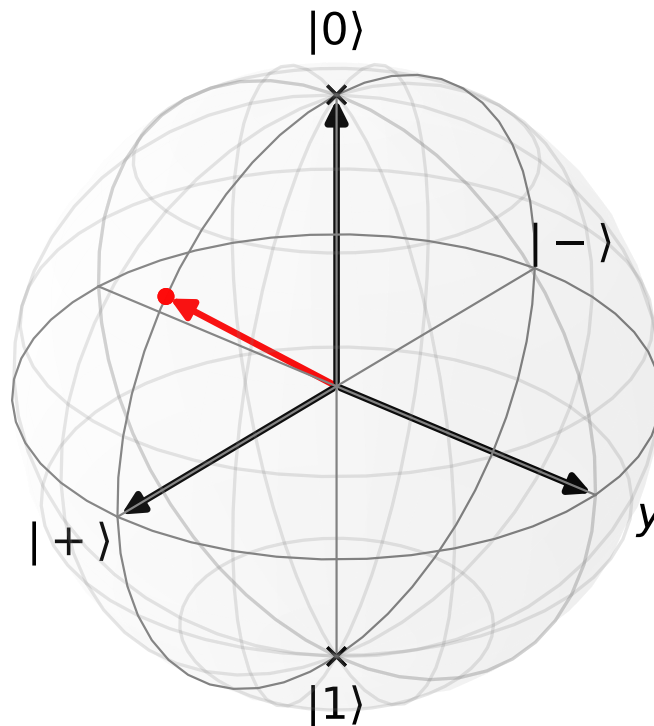


Abbildung 2.1.: Darstellung von Quantenbits innerhalb der Bloch-Kugel

Jede Linearkombination zulässiger Vektoren innerhalb dieses Unterraums bilden wieder einen zulässigen Vektor, daher gibt es unendlich viele Punkte auf der Bloch-Kugel.

2.2. Multiple Qubits

Um Algorithmen oder aufwendige Berechnungen auf Quantencomputern auszuführen wird mehr als nur ein Qubit bzw. ein Bit an Information benötigt. Daher ist es wichtig zu verstehen, wie einzelne Qubits miteinander interagieren, sich zusammenfügen lassen und durch Vektoren beschrieben werden.

Das Kronecker-Produkt wird genutzt um Qubits zusammenzuführen, bzw. deren kollektiven Zustand zu bilden. 2.7 zeigt zwei mögliche kollektive Zustände der Basiszustände aus 2.1.

$$\begin{aligned}
 |0\rangle \otimes |1\rangle &= |01\rangle = \begin{bmatrix} 1 \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \\
 |0\rangle \otimes |0\rangle &= |00\rangle = \begin{bmatrix} 1 \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}
 \end{aligned} \tag{2.7}$$

Somit lassen sich alle vier Basiszustände von zwei Qubits, aus den zwei Basiszuständen von einem Qubit durch das Kronecker-Produkt bilden. Man gehe davon aus, dass Zustände von mehreren Qubits sich also genau wie Zustände von einzelnen Qubits beschreiben lassen. n Qubits besitzen 2^n Amplituden, d.h. diese wachsen exponentiell mit der genutzten Anzahl an Qubits.

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix} \tag{2.8}$$

Eine allgemeine Darstellung eines Zustands von zwei Qubits zeigt 2.8, ein 4-Dimensionaler Vektor mit den jeweiligen Amplituden. Auch das quadrieren dieser Amplituden zeigt, mit welcher Wahrscheinlichkeit eines der 4 Ergebnisse 00, 01, 10, 11 nach der Messung der Qubits erhalten wird. Das heißt auch dieser Zustand muss durch seine Amplituden normalisiert sein, also gilt auch für den Zustandsvektor aus 2.8

$$|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1. \tag{2.9}$$

Ebenso werden die Qubits wie in 2.3 dargestellt gemessen. Um z.B. die Wahrscheinlichkeit zu erfahren, dass sich der Zustand $|\psi\rangle$ in Zustand $|11\rangle$ befindet wird folgende Messung durchgeführt

$$p(|11\rangle) = |\langle 11|\psi\rangle|^2 = |\alpha_{11}|^2. \tag{2.10}$$

Ebenso können auch Messungen, nach den Basisvektoren $|0\rangle$ und $|1\rangle$ durchgeführt werden.

Oft werden Zusammensetzungen aus einzelnen Qubits auch Quantenregister genannt, eine allgemeine und kompakte Form dieser Quantenregister aus [2], sieht folgendermaßen aus

$$R = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \quad (2.11)$$

Somit entspricht $|0\rangle, |1\rangle, \dots, |2^n - 1\rangle$ den Zuständen $|0 \dots 0\rangle, |0 \dots 1\rangle, \dots, |1 \dots 1\rangle$.

2.3. Quantengatter

Um Informationen bzw. Bits zu manipulieren, werden Schaltungen benötigt welche Gatter beinhalten. Dies gilt für klassische Rechner und Quantencomputer. Um in der Digitaltechnik die Funktionsweise von Gattern darzustellen werden gerne Wahrheitstabellen genutzt (rechte Spalte Tabelle 2.1). Bekannte Gatter-Typen aus der Digitaltechnik, wie der Negation oder dem Exklusiv-ODER können auch in der Welt der Quantencomputer abgebildet werden. Jedoch ist der Aufbau dieser Gatter etwas gewöhnungsbedürftig, denn die Realisierung eines auf n Qubits operierenden Gatters erfolgt durch eine unitäre $2^n \times 2^n$ -Matrix. Die Anwendung eines Gatters auf einen Zustandvektor entspricht mathematisch also einer unitären Transformation und führt eine Rotation auf der Bloch-Kugel aus. Dies geschieht durch die Bildung des Skalarprodukts über der unitären Matrix und dem Zustandvektor.

Eine Matrix wird als unitär bezeichnet, wenn das Produkt aus dieser Matrix und dessen adjungierte Matrix eine Einheitsmatrix bildet.

$$I = A^\dagger A \quad (2.12)$$

Die Adjungierte Matrix wird gebildet, indem alle Einträge in dieser Matrix komplex konjugiert und transponiert werden.

$$A^\dagger = A^{*T} \quad (2.13)$$

Somit sind alle Quantengatter unitäre Matrizen, genau wie die drei aus der Quantenmechanik bekannten Paulimatrizen X, Y und Z Tabelle 2.1. Diese führen eine Rotation von π um die x, y und z -Achse auf der Bloch-Kugel durch. Das Hadamard-Gatter H ermöglicht eine Abweichung der Basiszustände $|0\rangle$ und $|1\rangle$ und erzeugt eine Superposition dieser Zustände [3]. Dies wären die Zustandvektoren $|+\rangle$ und $|-\rangle$, welche auch auf der Bloch-Kugel in Abbildung 2.1 zu erkennen sind.

Zwei parametrisierte Gatter, das P -Gatter (Phasen-Gatter) und U -Gatter die nicht

in Tabelle 2.1 aufgeführt sind, erlauben die Spezifizierung von sämtlichen Gattern bzw. unendlich vielen Gattern die auf einem Qubit operieren.

$$P(\phi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \quad \phi \in \mathbf{R}$$

$$U(\theta, \phi, \lambda) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\phi+\lambda)}\sin\left(\frac{\theta}{2}\right) \end{bmatrix} \quad \theta, \phi, \lambda \in \mathbf{R} \quad (2.14)$$

Matrix	Schaltungssymbol	Wahrheitstabelle									
$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$ q\rangle \text{ --- } \boxed{X} \text{ ---}$	<table> <tr> <th>Fall</th><th>$q\rangle$</th><th>$X q\rangle$</th></tr> <tr> <td>1</td><td>$0\rangle$</td><td>$1\rangle$</td></tr> <tr> <td>2</td><td>$1\rangle$</td><td>$0\rangle$</td></tr> </table>	Fall	$ q\rangle$	$X q\rangle$	1	$ 0\rangle$	$ 1\rangle$	2	$ 1\rangle$	$ 0\rangle$
Fall	$ q\rangle$	$X q\rangle$									
1	$ 0\rangle$	$ 1\rangle$									
2	$ 1\rangle$	$ 0\rangle$									
$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$ q\rangle \text{ --- } \boxed{Y} \text{ ---}$	<table> <tr> <th>Fall</th><th>$q\rangle$</th><th>$Y q\rangle$</th></tr> <tr> <td>1</td><td>$0\rangle$</td><td>$i 1\rangle$</td></tr> <tr> <td>2</td><td>$1\rangle$</td><td>$-i 0\rangle$</td></tr> </table>	Fall	$ q\rangle$	$Y q\rangle$	1	$ 0\rangle$	$i 1\rangle$	2	$ 1\rangle$	$-i 0\rangle$
Fall	$ q\rangle$	$Y q\rangle$									
1	$ 0\rangle$	$i 1\rangle$									
2	$ 1\rangle$	$-i 0\rangle$									
$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$ q\rangle \text{ --- } \boxed{Z} \text{ ---}$	<table> <tr> <th>Fall</th><th>$q\rangle$</th><th>$Z q\rangle$</th></tr> <tr> <td>1</td><td>$0\rangle$</td><td>$0\rangle$</td></tr> <tr> <td>2</td><td>$1\rangle$</td><td>$- 1\rangle$</td></tr> </table>	Fall	$ q\rangle$	$Z q\rangle$	1	$ 0\rangle$	$ 0\rangle$	2	$ 1\rangle$	$- 1\rangle$
Fall	$ q\rangle$	$Z q\rangle$									
1	$ 0\rangle$	$ 0\rangle$									
2	$ 1\rangle$	$- 1\rangle$									
$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$ q\rangle \text{ --- } \boxed{H} \text{ ---}$	<table> <tr> <th>Fall</th><th>$q\rangle$</th><th>$H q\rangle$</th></tr> <tr> <td>1</td><td>$0\rangle$</td><td>$\frac{ 0\rangle+ 0\rangle}{\sqrt{2}}$</td></tr> <tr> <td>2</td><td>$1\rangle$</td><td>$\frac{ 0\rangle- 1\rangle}{\sqrt{2}}$</td></tr> </table>	Fall	$ q\rangle$	$H q\rangle$	1	$ 0\rangle$	$\frac{ 0\rangle+ 0\rangle}{\sqrt{2}}$	2	$ 1\rangle$	$\frac{ 0\rangle- 1\rangle}{\sqrt{2}}$
Fall	$ q\rangle$	$H q\rangle$									
1	$ 0\rangle$	$\frac{ 0\rangle+ 0\rangle}{\sqrt{2}}$									
2	$ 1\rangle$	$\frac{ 0\rangle- 1\rangle}{\sqrt{2}}$									
$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	$ q\rangle \text{ --- } \boxed{S} \text{ ---}$	<table> <tr> <th>Fall</th><th>$q\rangle$</th><th>$S q\rangle$</th></tr> <tr> <td>1</td><td>$0\rangle$</td><td>$0\rangle$</td></tr> <tr> <td>2</td><td>$1\rangle$</td><td>$i 1\rangle$</td></tr> </table>	Fall	$ q\rangle$	$S q\rangle$	1	$ 0\rangle$	$ 0\rangle$	2	$ 1\rangle$	$i 1\rangle$
Fall	$ q\rangle$	$S q\rangle$									
1	$ 0\rangle$	$ 0\rangle$									
2	$ 1\rangle$	$i 1\rangle$									
$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$	$ q\rangle \text{ --- } \boxed{T} \text{ ---}$	<table> <tr> <th>Fall</th><th>$q\rangle$</th><th>$T q\rangle$</th></tr> <tr> <td>1</td><td>$0\rangle$</td><td>$0\rangle$</td></tr> <tr> <td>2</td><td>$1\rangle$</td><td>$e^{i\frac{\pi}{4}} 1\rangle$</td></tr> </table>	Fall	$ q\rangle$	$T q\rangle$	1	$ 0\rangle$	$ 0\rangle$	2	$ 1\rangle$	$e^{i\frac{\pi}{4}} 1\rangle$
Fall	$ q\rangle$	$T q\rangle$									
1	$ 0\rangle$	$ 0\rangle$									
2	$ 1\rangle$	$e^{i\frac{\pi}{4}} 1\rangle$									

Tabelle 2.1.: Grundlegende 1-Qubit Gatter

Aus dem P -Gatter lassen sich die Gatter Z , S und T konstruieren. 2.15 zeigt die

Spezifizierung des S - und T -Gatters durch das Phasengatter.

$$P\left(\phi = \frac{\pi}{2}\right) = S \quad P(\phi = \pi) = Z. \quad (2.15)$$

Das U -Gatter ermöglicht die Spezifizierung jeglicher Gatter, z.B. kann das Hadamard-Gatter folgendermaßen durch das U -Gatter spezifiziert werden

$$U\left(\theta = \frac{\pi}{2}, \phi = 0, \lambda = \pi\right) = H. \quad (2.16)$$

Somit kann es eine große Menge an nützlichen Gattern geben, die auf einem Qubit operieren. Es ist möglich diese Gatter auch auf mehreren Qubits operieren zu lassen. Diese Gatter werden dann kontrollierte Gatter genannt, da diese ein oder mehrere kontrollierende Qubits (*controlled qubits*) und ein Zielqubit (*target qubit*) beinhalten. Tabelle 2.2 zeigt oft genutzte Gatter die auf mehr als einem Qubit operieren.

Ein kontrolliertes Gatter funktioniert folgendermaßen: Immer dann wenn sich die kontrollierenden Bits im Zustand 1 befinden wird eine Transformation auf das Zielbit ausgeführt. Die meisten dieser kontrollierten Gatter können durch 1-Qubit Gatter und dem kontrollierten X -Gatter ($CNOT$ o. CX) rekonstruiert werden [4]. Das heißt auch, dass alle in Tabelle 2.2 dargestellten Gatter, die selbe unitäre Transformation wie die Gatter aus Tabelle 2.1 ausführen. Diesmal jedoch nur auf das Zielbit, immer genau dann wenn das kontrollierende Bit 1 ist.

Bei dem Aufbau einer Quantenschaltung ist es somit wichtig zu wissen, welches Qubit als kontrolliertes Bit und welches als Zielbit für ein Gatter dient. Aus diesem Grund erhalten in dieser Ausarbeitung die kontrollierten Gatter einen Index. In diesem stellen die ersten Zahlen die kontrollierenden Bits und die letzte Zahl das Zielbit dar. 2.17 zeigt die Zusammensetzung der beiden möglichen Matrizen für das CX -Gatter.

$$\begin{aligned} CX_{01} &= |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X \\ CX_{10} &= I \otimes |0\rangle\langle 0| + X \otimes |1\rangle\langle 1| \end{aligned} \quad (2.17)$$

Dabei ist $|x\rangle\langle x|$ das äußere Produkt. $\langle x|$ ist der komplex konjugiert und transponiert Vektor von $|x\rangle$ und I die Einheitsmatrix.

Ein wichtiges Gatter in Tabelle 2.2 ist das $SWAP$ -Gatter, welches die Funktion erfüllt die Zustände der beiden Qubits zu tauschen. Auch dieses Gatter kann z.B. nur durch CX -Gatter aufgebaut werden 2.2 (vgl. [3]). Jedoch ist dies nicht der

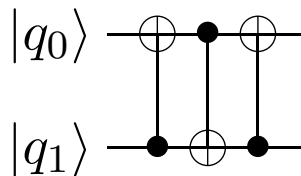


Abbildung 2.2.: $SWAP$ -Gatter realisiert aus 3 kontrollierten CX -Gattern

einzigste Weg ein $SWAP$ -Gatter aus anderen Gattern zu realisieren.

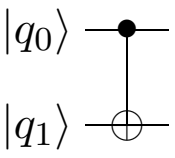
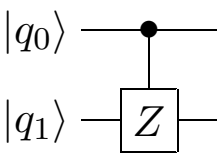
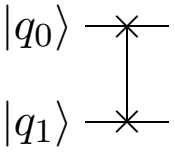
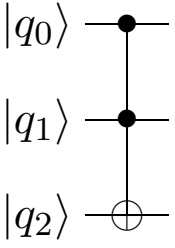
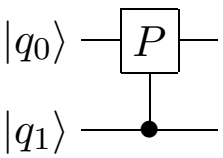
Matrix	Schaltungssymbol	Wahrheitstabelle																											
$CX_{01} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$		<table> <tr> <th>Fall</th><th>$q_0q_1\rangle$</th><th>$CX_{01} q_0q_1\rangle$</th></tr> <tr><td>1</td><td>$00\rangle$</td><td>$00\rangle$</td></tr> <tr><td>2</td><td>$01\rangle$</td><td>$01\rangle$</td></tr> <tr><td>3</td><td>$10\rangle$</td><td>$11\rangle$</td></tr> <tr><td>4</td><td>$11\rangle$</td><td>$10\rangle$</td></tr> </table>	Fall	$ q_0q_1\rangle$	$CX_{01} q_0q_1\rangle$	1	$ 00\rangle$	$ 00\rangle$	2	$ 01\rangle$	$ 01\rangle$	3	$ 10\rangle$	$ 11\rangle$	4	$ 11\rangle$	$ 10\rangle$												
Fall	$ q_0q_1\rangle$	$CX_{01} q_0q_1\rangle$																											
1	$ 00\rangle$	$ 00\rangle$																											
2	$ 01\rangle$	$ 01\rangle$																											
3	$ 10\rangle$	$ 11\rangle$																											
4	$ 11\rangle$	$ 10\rangle$																											
$CZ_{01} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$		<table> <tr> <th>Fall</th><th>$q_0q_1\rangle$</th><th>$CZ_{01} q_0q_1\rangle$</th></tr> <tr><td>1</td><td>$00\rangle$</td><td>$00\rangle$</td></tr> <tr><td>2</td><td>$01\rangle$</td><td>$01\rangle$</td></tr> <tr><td>3</td><td>$10\rangle$</td><td>$10\rangle$</td></tr> <tr><td>4</td><td>$11\rangle$</td><td>$- 11\rangle$</td></tr> </table>	Fall	$ q_0q_1\rangle$	$CZ_{01} q_0q_1\rangle$	1	$ 00\rangle$	$ 00\rangle$	2	$ 01\rangle$	$ 01\rangle$	3	$ 10\rangle$	$ 10\rangle$	4	$ 11\rangle$	$- 11\rangle$												
Fall	$ q_0q_1\rangle$	$CZ_{01} q_0q_1\rangle$																											
1	$ 00\rangle$	$ 00\rangle$																											
2	$ 01\rangle$	$ 01\rangle$																											
3	$ 10\rangle$	$ 10\rangle$																											
4	$ 11\rangle$	$- 11\rangle$																											
$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$		<table> <tr> <th>Fall</th><th>$q_0q_1\rangle$</th><th>$SWAP q_0q_1\rangle$</th></tr> <tr><td>1</td><td>$00\rangle$</td><td>$00\rangle$</td></tr> <tr><td>2</td><td>$01\rangle$</td><td>$10\rangle$</td></tr> <tr><td>3</td><td>$10\rangle$</td><td>$01\rangle$</td></tr> <tr><td>4</td><td>$11\rangle$</td><td>$11\rangle$</td></tr> </table>	Fall	$ q_0q_1\rangle$	$SWAP q_0q_1\rangle$	1	$ 00\rangle$	$ 00\rangle$	2	$ 01\rangle$	$ 10\rangle$	3	$ 10\rangle$	$ 01\rangle$	4	$ 11\rangle$	$ 11\rangle$												
Fall	$ q_0q_1\rangle$	$SWAP q_0q_1\rangle$																											
1	$ 00\rangle$	$ 00\rangle$																											
2	$ 01\rangle$	$ 10\rangle$																											
3	$ 10\rangle$	$ 01\rangle$																											
4	$ 11\rangle$	$ 11\rangle$																											
$CCX_{012} = \begin{bmatrix} I_2 & 0_2 & 0_2 & 0_2 \\ 0_2 & I_2 & 0_2 & 0_2 \\ 0_2 & 0_2 & I_2 & 0_2 \\ 0_2 & 0_2 & 0_2 & X \end{bmatrix}$		<table> <tr> <th>Fall</th><th>$q_0q_1q_2\rangle$</th><th>$CCX_{012} q_0q_1q_2\rangle$</th></tr> <tr><td>1</td><td>$000\rangle$</td><td>$000\rangle$</td></tr> <tr><td>2</td><td>$001\rangle$</td><td>$001\rangle$</td></tr> <tr><td>3</td><td>$010\rangle$</td><td>$010\rangle$</td></tr> <tr><td>4</td><td>$011\rangle$</td><td>$011\rangle$</td></tr> <tr><td>5</td><td>$100\rangle$</td><td>$100\rangle$</td></tr> <tr><td>6</td><td>$101\rangle$</td><td>$101\rangle$</td></tr> <tr><td>7</td><td>$110\rangle$</td><td>$111\rangle$</td></tr> <tr><td>8</td><td>$111\rangle$</td><td>$110\rangle$</td></tr> </table>	Fall	$ q_0q_1q_2\rangle$	$CCX_{012} q_0q_1q_2\rangle$	1	$ 000\rangle$	$ 000\rangle$	2	$ 001\rangle$	$ 001\rangle$	3	$ 010\rangle$	$ 010\rangle$	4	$ 011\rangle$	$ 011\rangle$	5	$ 100\rangle$	$ 100\rangle$	6	$ 101\rangle$	$ 101\rangle$	7	$ 110\rangle$	$ 111\rangle$	8	$ 111\rangle$	$ 110\rangle$
Fall	$ q_0q_1q_2\rangle$	$CCX_{012} q_0q_1q_2\rangle$																											
1	$ 000\rangle$	$ 000\rangle$																											
2	$ 001\rangle$	$ 001\rangle$																											
3	$ 010\rangle$	$ 010\rangle$																											
4	$ 011\rangle$	$ 011\rangle$																											
5	$ 100\rangle$	$ 100\rangle$																											
6	$ 101\rangle$	$ 101\rangle$																											
7	$ 110\rangle$	$ 111\rangle$																											
8	$ 111\rangle$	$ 110\rangle$																											
$CP_{10} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix}$		<table> <tr> <th>Fall</th><th>$q_0q_1\rangle$</th><th>$CP_{10} q_0q_1\rangle$</th></tr> <tr><td>1</td><td>$00\rangle$</td><td>$00\rangle$</td></tr> <tr><td>2</td><td>$01\rangle$</td><td>$01\rangle$</td></tr> <tr><td>3</td><td>$10\rangle$</td><td>$10\rangle$</td></tr> <tr><td>4</td><td>$11\rangle$</td><td>$e^{i\phi} 11\rangle$</td></tr> </table>	Fall	$ q_0q_1\rangle$	$CP_{10} q_0q_1\rangle$	1	$ 00\rangle$	$ 00\rangle$	2	$ 01\rangle$	$ 01\rangle$	3	$ 10\rangle$	$ 10\rangle$	4	$ 11\rangle$	$e^{i\phi} 11\rangle$												
Fall	$ q_0q_1\rangle$	$CP_{10} q_0q_1\rangle$																											
1	$ 00\rangle$	$ 00\rangle$																											
2	$ 01\rangle$	$ 01\rangle$																											
3	$ 10\rangle$	$ 10\rangle$																											
4	$ 11\rangle$	$e^{i\phi} 11\rangle$																											

Tabelle 2.2.: Ausgewählte 2-, 3-Quanten Gatter

2.4. Quantenschaltungen

Eines der wichtigsten Elemente in Quantenschaltungen, wurde in der vorherigen Sektion 2.3 besprochen, die Quantengatter. Denn die Quantenschaltung ist eine geordnete Sequenz von Quantengattern, Messungen und Initialisierungen [3]. Sie beschreiben ein Modell das genutzt wird um Berechnungen auf Quantencomputern durchzuführen. D.h. immer dann wenn eine Berechnung auf einem Quantencomputer durchgeführt wird kann diese Berechnung auch in Form einer Quantenschaltung dargestellt werden. Die meisten Berechnungen basieren jedoch schon vor ihrer Ausführung auf vorher erstellte Quantenschaltungen. Was genau eine Quantenschaltung ist und wie diese Funktionieren soll in dieser Sektion durch ein Beispiel verdeutlicht werden.

Eine ziemlich einfache Schaltung, mit einem jedoch ziemlich effektiven Ergebnis, zeigt Abbildung 2.3. Diese Schaltung kann genutzt werden um einen maxi-

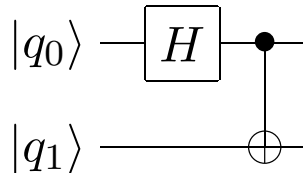


Abbildung 2.3.: Quantenschaltung zur Erzeugung von Bell-Zuständen

mal verschränkten Zustand zu erzeugen. Diese Zustände werden Bell-Zustände (*Bell-State*) genannt 2.18. Diese spielen eine zentrale Rolle in der Quanteninformation und sind unter anderem Grundbestandteil der Quantenteleportation und Quantenkryptographie.

Ein Zustand wird als verschränkt bezeichnet, wenn er nicht in ein Produkt von Zuständen der einzelnen Bits zerlegt werden kann [2]. Als maximal verschränkt bezeichnet man den Zustand dann, wenn deren Bits maximal stark gekoppelt sind. D.h. jedes an diesem Zustand beteiligte Qubit, wird stets das selbe Ergebnis liefern. Ebenso muss der Einfluss der Messung dieser einzelnen Qubits auf das endgültig gemessene Ergebnis gleich sein.

$$\begin{aligned}
 |\beta_{00}\rangle &= \frac{|00\rangle + |11\rangle}{\sqrt{2}} \\
 |\beta_{01}\rangle &= \frac{|01\rangle + |10\rangle}{\sqrt{2}} \\
 |\beta_{10}\rangle &= \frac{|00\rangle - |11\rangle}{\sqrt{2}} \\
 |\beta_{11}\rangle &= \frac{|01\rangle - |10\rangle}{\sqrt{2}}
 \end{aligned} \tag{2.18}$$

Die Schaltung 2.3 ist von links nach rechts zu lesen und enthält bisher nur zwei Qubits, ein Hadamard-Gatter und ein CX-Gatter. Die geraden Linien sollen den

Zeitablauf der Qubits darstellen. Um nun das Ergebnis dieser einzelnen Qubits zu erhalten müssen diese gemessen und das Ergebnis auf klassische Bits übertragen werden. Dazu muss die Schaltung folgendermaßen erweitert werden 2.4. Das neu hinzugekommene Symbol steht nun für eine Messung. Das Ergebnis die-

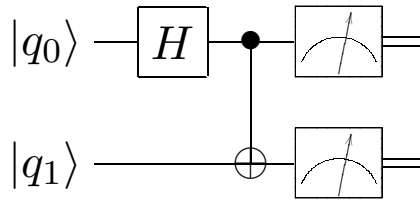


Abbildung 2.4.: Messen von Qubits in der Schaltung zur Ereugung von Bell-Zuständen

ser Messung wird in einem klassischen Bit gespeichert, doppelte Linien stehen in Quantenschaltungen also für klassische Bits.

Es gibt eine Menge an Schaltungen die interessante Probleme lösen oder Algorithmen implementieren, z.B. die Implementierung des Bernstein-Vazirani Problems [5]. Oder die Implementierung des bekannten Quantensuchalgorithmus von Grover [6]. All diese Schaltungen sind unterschiedlich aufgebaut haben aber einige Gemeinsamkeiten, sie sind azyklisch und erlauben somit keine Rückführung. Für die Schaltungen gilt auch, dass das Zusammenführen der Qubits ohne Gatter oder das Kopieren der Qubits nicht erlaubt ist [1].

3. Programmierung von Quantencomputer

In dem vorherigen Kapitel wurde die Funktionsweise von Quantenbits, Quantengattern und Quantenschaltungen erläutert. Dies sind wesentliche Grundlagen von Quantencomputern, welche zum Verständnis seiner Funktionsweise benötigt werden. In den folgenden Kapitel steigt der praktische Anteil, denn in diesem Kapitel sollen Werkzeuge vorgestellt werden um Quantencomputer zu programmieren oder Quantenschaltungen zu simulieren.

3.1. Software und Programmiersprachen

IBM ermöglicht durch ihre Onlineplattform IBM Quantum [7] den Zugang zu realen Quantencomputern oder zu simulierten Quantencomputern auf leistungsfähiger Hardware. IBM Quantum bietet durch das Werkzeug Quantum Composer einen einfachen Einstieg in den Bereich der Quantencomputer für Neueinsteiger. Quantum Composer ermöglicht durch Ziehen und Ablegen (*Drag and Drop*), Quantenschaltungen aufzubauen. Auch vorgefertigte Schaltungen/Algorithmen werden angeboten und können vom Benutzer ausgeführt, verändert und angepasst werden. Auf Grundlage dieser Schaltungen wird Code generiert, dieser kann falls nötig vom Benutzer beliebig verändert und angepasst werden. Abbil-

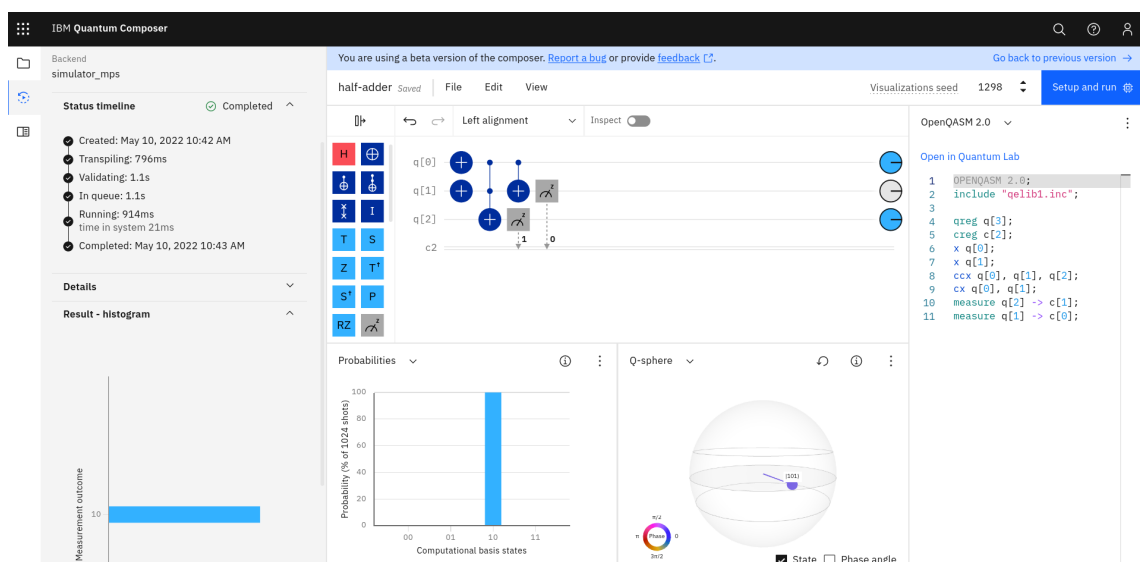


Abbildung 3.1.: Quantum-Halbaddierer simuliert auf den *simulator_mps*

Abbildung 3.1 zeigt die Oberfläche des Quantum Composers. Nördlich kann der Benutzer die erstellte Schaltung sehen und bearbeiten. Südlich des Bildes kann die Wahrscheinlichkeit der Ausgabe sowie dessen Zustand innerhalb der Q-Kugel

betrachtet werden.

Die Ausführung dieser Schaltung bzw. dieses Programms auf einem realen Quantencomputer, oder die Simulation auf einem HPC (*High Performance Computer*) in der Cloud, wird durch den IBM Quantum Service ausgeführt. Der IBM Quantum Service bezeichnet diese Schaltungen als Jobs und verwaltet diese durch einen kubernetes Cluster. Dieses Cluster beinhaltet einen Runtime Manager, Runtime Job und einen Execution Kernel.

Fortgeschrittene Benutzer haben durch das Quantum Lab und Qiskit die Möglichkeit in der Cloud Jupyter-Notebooks zu verfassen. Durch die quelloffene Programmierschnittstelle (*API*) Qiskit, wird es ermöglicht Quantencomputer auch lokal auf Ebene von Schaltungen und Algorithmen zu programmieren. Dabei möchte Qiskit eine standardisierte API entwickeln, die für verschiedenste Benutzergruppen geeignet ist [8]. Um Schaltungen auf unterschiedlicher Hardware interpretieren lassen zu können, werden die in Qiskit erzeugten Objekte von JSON zu OpenQASM umgewandelt. OpenQASM ist dabei jedoch keine Ausgangssprache oder die Instruktionen einer Zielmaschine, sondern eine Zwischendarstellung (*intermediate representation (IR)*) [9]. In Anhang A Listing 1 und 2 wird der Code eines Quanten-Halbaddierers in Qiskit und OpenQASM dargestellt.

3.2. Simulation

In Sektion 2.2 wurde der exponentielle Anstieg der Amplituden für die Nutzung von Qubits schon angesprochen. Aus diesem Grund ist die Simulation von Quantencomputern nur effizient mit einer Anzahl von ≈ 20 Qubits möglich. Selbst die Simulation von Quantencomputern mit 100 Qubits ist auf sehr leistungsfähigen Supercomputer momentan nicht möglich [3].

Trotzdem ermöglicht IBM Quantum mit dem Simulator *simulator_stabilizer* und *simulator_mps* den Zugriff auf eine Simulationsmaschine mit 5000 und 100 Qubits. Das liegt daran, dass es unterschiedliche Simulationstypen gibt. Die Simulation eines Systems mit bis zu 5000 Qubits kann durch die Verbesserung des Gottesman-Knill Theorem [10] ermöglicht werden. Dieses Theorem erlaubt es Stabilisatorschaltungen (*stabilizer circuits*) effizient d.h. in polynomieller Zeit zu simulieren. Dabei versteht man unter einer Stabilisatorschaltung, eine Quantenschaltung die nur aus Clifford-Gattern besteht. Darunter fallen CNOT-, Hadamard- und Phasengatter. Somit können auf diesem Simulationstyp auch nur Schaltungen simuliert werden die aus Clifford-Gattern bestehen.

Die auf der Onlineplattform IBM Quantum genutzten Simulationstypen, sowie verfügbaren Simulatoren sind auf der Abbildung 3.2 veranschaulicht.

simulator_stabilizer Simulator status ● Online Simulator type Clifford simulator Qubits 5000	simulator_mps Simulator status ● Online Simulator type Matrix Product State Qubits 100
simulator_extended_stabilizer Simulator status ● Online Simulator type Extended Clifford (e.g. Clifford+T) Qubits 63	ibmq_qasm_simulator Simulator status ● Online Simulator type General, context-aware Qubits 32
simulator_statevector Simulator status ● Online Simulator type Schrödinger wavefunction Qubits 32	

Abbildung 3.2.: Verfügbare Simulatoren bei IBM Quantum

3.3. Quantum Hardware

Die verschiedenen Simulationstypen von Quantencomputern klingen viel versprechend, jedoch ist auch die reale Quantenhardware heutzutage ziemlich fortgeschritten. Denn IBM Quantum hat es geschafft einen Chip zu entwickeln, der die 100-Qubit Grenze überschreitet und somit auch die Möglichkeiten der Simulation.

Dieser Prozessor heißt IBM-Eagle und ist ein 127-Qubit Quantenprozessor, bestehend aus einem Interposer, einer Verkabelungsebene, einer Resonatorebene und Qubitebene [11]. Der IBM Eagle ist eine Kombination von Techniken aus vorherigen Quantenprozessoren, diese 3D-Packaging-Technik soll auch für den noch erscheinenden Condor-Prozessor mit über 1.000 Qubits geeignet sein. Durch die Nutzung des heavy-hexagon Qubit-Layouts [12] aus dem vorherigen Falcon Prozessor wird das Potential für Fehlerraten reduziert. Um die Menge an Elektronik und Verkablung im Inneren des Chips zu reduzieren, wird das aus dem vorherigen Hummingbird R2 Chip bekannte Auslesemultiplexing genutzt.

IBM Quantum verfügt über 22 unterschiedliche Quantensysteme mit maximal 127 Qubits. Jedoch hat der standartmäßige Benutzer nur Zugriff auf sieben dieser Systeme mit maximal 5 Qubits.

4. Quantenalgorithmen

Quantencomputer sollen effizient Probleme lösen die durch klassische Algorithmen nicht effizient gelöst werden können. Der Grundbaustein von Quantenalgorithmen sind die Quantenschaltungen 2.4, denn sie dienen zur Entwicklung dieser. Jedoch ist die Entwicklung von Algorithmen nicht trivial und erfordert eine Menge Aufwand und Einfallsreichtum. Quantenalgorithmen sollen effizienter sein als klassische, dies ist ebenso ein weiterer Faktor wodurch die Entwicklung von Quantenalgorithmen erschwert wird.

4.1. Algorithmenüberblick

Die bisher bekanntesten Quantenalgorithmen bieten eine quadratische und exponentielle Beschleunigung gegenüber klassischen. Dies ist der Shors Algorithmus bzw. (*Shors Quantum Fourier Transformation*) und der Grovers Algorithmus.

Beide Algorithmen lösen unterschiedliche Probleme, z.B. ist der Grover Algorithmus ein Suchalgorithmus, der zur ungeordneten Suche dient. Dieser kann genutzt werden um in großen Datenbeständen z.B. ein Minimum oder Maximum zu finden. Aber auch für die Suche nach Schlüsseln in Kryptosystemen wie dem Datenverschlüsselungsstandard (*DES*) bietet der Grover Algorithmus eine schnellere Laufzeit als klassische Algorithmen.

Der Shors Algorithmus kann zur Berechnung diskreter Logarithmen oder der Zerlegung ganzer Zahlen genutzt werden. Da der Shors Algorithmus dies in Polynomialzeit tut, sind asymmetrische Verschlüsselungsverfahren wie z.B. RSA und Diffie-Hellman Schlüsselaustausch von Quantencomputern bedroht. Aus diesem Grund versucht man Quantensichere Verfahren zu Standardisieren. Somit sollen ausgewählte asymmetrische Verfahren wie z.B. McEliece, Crystal-Kyber, NTRU und Saber die man als Quantensicher bezeichnet, in naher Zukunft bedrohte Verfahren ersetzen. Gegen die Verdopplung der Schlüsselgrößen in symmetrischen Verschlüsselungsverfahren ist auch der Grovers Algorithmus machtlos, bzw. benötigt auch er zu viel Zeit um valide Schlüssel zu finden. Aus diesem Grund, ist der Shors Algorithmus für die Kryptographie und Post-Quanten-Kryptographie von größerer Bedeutung.

Der Shors-Algorithmus kann Probleme wie z.B. die Faktorisierung von Fastprimzahlen in polynomieller Zeit berechnen, da dieser Algorithmus einen Quantenanteil besitzt, der zum Finden von Perioden (*periodic finding*) dient. Somit ist der Shors Algorithmus in der Lage auch Probleme zu Lösen, die in das Finden von Perioden gewandelt werden können. Um diesen Teil des Algorithmus zu verstehen, müssen zwei weitere Quantenalgorithmen verstanden werden, die

Quanten-Fourier-Transformation und die Quanten-Phasenschätzung.

4.2. Quanten Fourier-Transformation

Die Quanten Fourier-Transformation (QFT), transformiert einen Basiszustand $|x\rangle$ zu einem Fourierzustand $|\tilde{x}\rangle$ [3]. Die QFT ist die Anwendung der Diskreten Fourier-Transformation (DFT) auf Quantenzustände, somit unterscheidet diese sich minimal zur Berechnung von Fourierkoeffizienten mittels DFT.

$$DFT \Rightarrow X[k] = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{i\frac{2\pi}{N}kj} \cdot x[j] \quad (4.1)$$

$$QFT \Rightarrow |\tilde{x}\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{i\frac{2\pi}{N}xj} \cdot |j\rangle$$

Die Berechnung der QFT kann somit auf ein Quantenregister 2.2 ausgeführt werden. Bei der Quantum Fourier-Transformation entspricht $N = 2^n$, dabei ist n die Anzahl der genutzten Qubits. In [3] wird gezeigt, wie es möglich ist die QFT als Produkt auszuschreiben.

$$|\tilde{x}\rangle = \frac{1}{\sqrt{N}} \left(|0\rangle + e^{i\frac{2\pi x}{2^1}} |1\rangle \right) \otimes \left(|0\rangle + e^{i\frac{2\pi x}{2^2}} |1\rangle \right) \otimes \dots \otimes \left(|0\rangle + e^{i\frac{2\pi x}{2^n}} |1\rangle \right) \quad (4.2)$$

Aus diesem Produkt lässt sich die Schaltung der QFT etwas verständlicher nachvollziehen, denn es werden zur Erstellung der Schaltung pro Qubit ein Hadamard-Gatter und mehrere kontrollierte R_k -Gatter benötigt (Abbildung 4.1). Dabei sind R_k -Gatter spezifizierte Phasen-Gatter 4.3. Die aufgetragene QFT-Schaltung kann

$$P\left(\phi = \frac{2\pi}{2^k}\right) = R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{2\pi}{2^k}} \end{bmatrix} \quad (4.3)$$

auch invertiert realisiert werden (erstes H-Gatter auf Qubit $|q_n\rangle$). Bei dieser Struk-

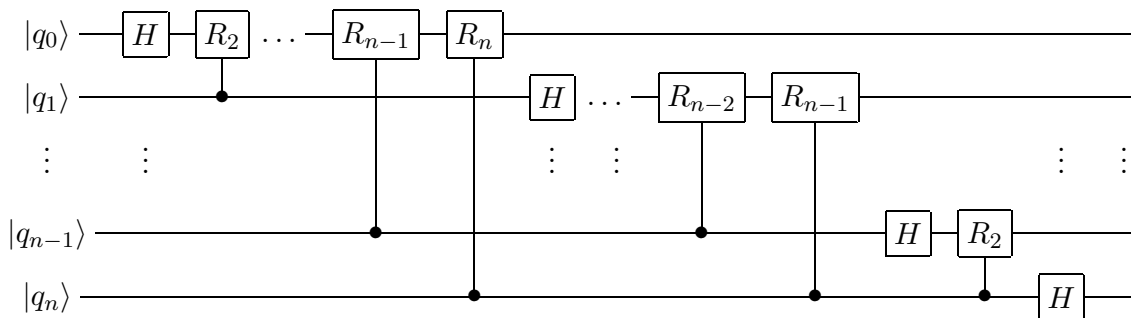


Abbildung 4.1.: Schaltung der Quantum Fourier-Transformation (vgl. [1])

tur ist jedoch zu beachten, dass die Ausgabe der Qubits ebenso durch SWAP-Gattern umgekehrt werden muss. In Anhang A Listing 3 ist die Implementierung

der QFT-Schaltung in Qiskit für $n = 3$ Qubits dargestellt. Abbildung 4.2 zeigt die Rotation der jeweiligen Zustandvektoren auf der Bloch-Kugel für das Quantenregister $|5\rangle$, vor und nach Anwendung der Quantum Fourier-Transformation.

$$|101\rangle \xrightarrow{QFT} |\tilde{1}01\rangle \quad (4.4)$$

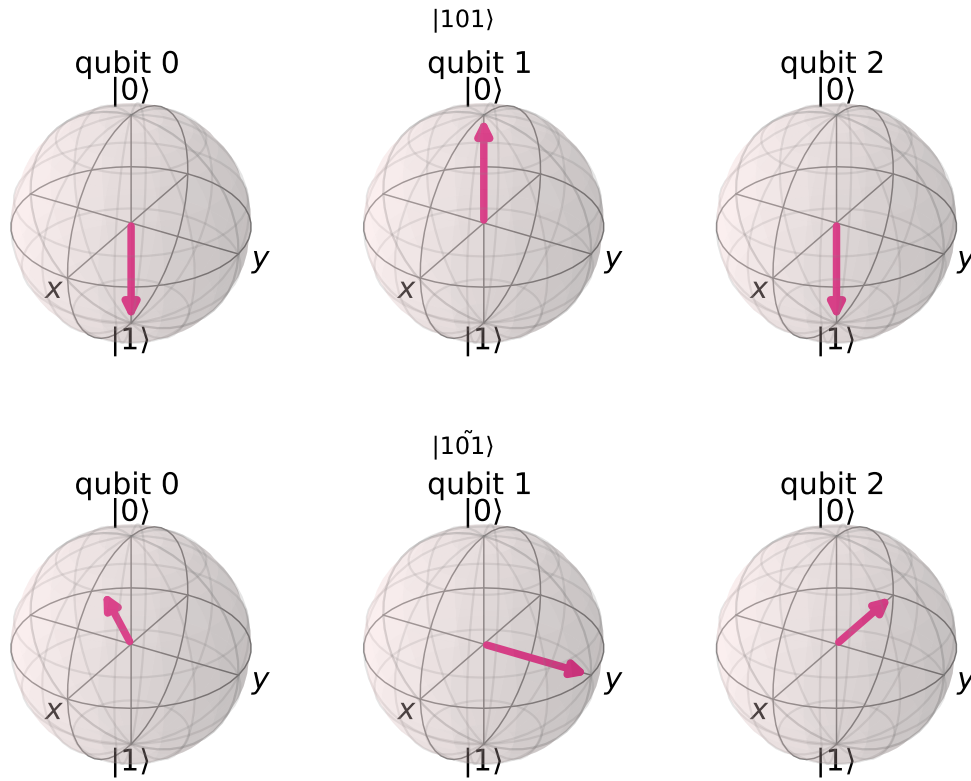


Abbildung 4.2.: Rotation auf der Bloch-Kugel durch Anwenden der QFT (vgl. [3])

4.3. Quanten-Phasenschätzung

Eine unitäre Matrix besitzt einen oder mehrere Eigenvektoren $|\psi\rangle$ mit zugehörigen Eigenwerten $e^{2\pi i\theta}$.

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle \quad (4.5)$$

Die Quantenphasenschätzung (*quantum phase estimation*) kann die Phase θ aus Gleichung 4.5 bestimmen. Dies gilt unter der Bedingung, dass diese Phase die eines Eigenwertes ist. D.h. der Zustand $|\psi\rangle$ muss so angepasst sein, dass er ein Eigenzustand von U ist. Es kann somit Phasenrückschlag (*Phasekickback*) und das wiederholte Ausführen von U genutzt werden, um die Phase von U in θ zu schreiben. Diese Phase kann mittels inverser QFT aus seinem Fourierzustand zu einem Basiszustand gewandelt werden. Die Schaltung der Quantenphasenschätzung (Abbildung 4.3) wird auf zwei Quantenregistern angewandt und wird aus

zwei Teilen zusammengestellt. Das erste Quantenregister besteht aus t Qubits die alle mit dem Basiszustand $|0\rangle$ initialisiert werden. Die Anzahl an Qubits für dieses Register wird nach der gewünschten Genauigkeit für die Schätzung der Phase abhängig gemacht. Je größer t gewählt wird, desto größer ist die Anzahl an Ziffern für die Schätzung und auch die Wahrscheinlichkeit, dass die Phasenschätzung erfolgreich ist. Das zweite Quantenregister ist der Eigenvektor $|\psi\rangle$, für

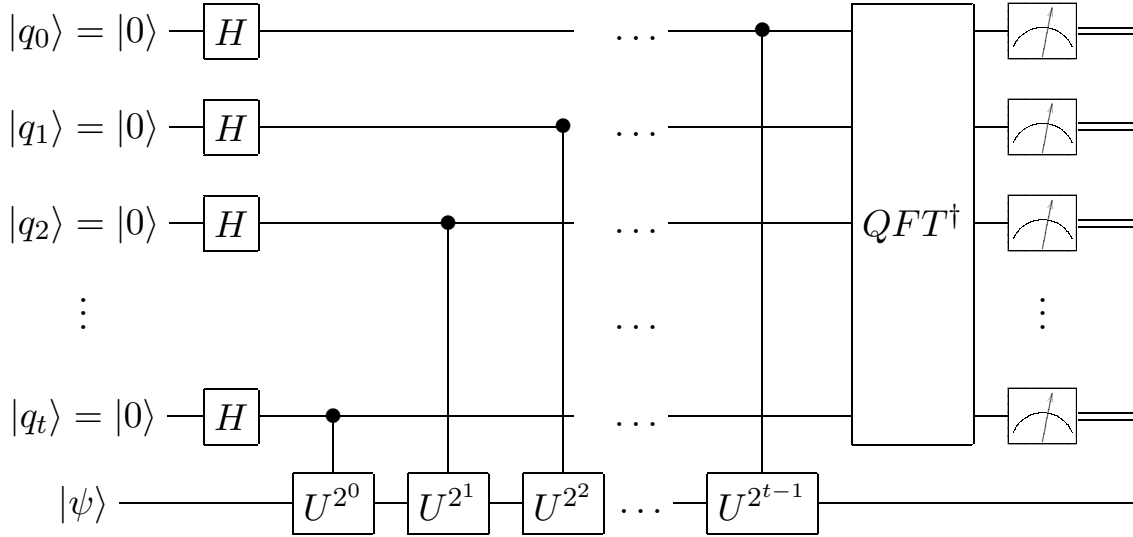


Abbildung 4.3.: Allgemeine Schaltung der Quantenphasenschätzung (vgl. [1])

dieses Register werden so viele Qubits genutzt wie für die Darstellung des Zustands benötigt werden.

Im ersten Teil der Schaltung wird auf allen Qubits aus dem ersten Register eine Hadamard-Transformation ausgeführt. Daraufhin wird eine kontrollierte U -Transformation auf das zweite Register ausgeführt, bei diesem U -Gatter erhöht sich wie ersichtlich stetig die 2er Potenz. Im zweiten Teil der Schaltung wird auf das erste Register eine invertierte Quantum Fourier-Transformation angewandt. [3] zeigt detaillierter wie das erste Register durch die Schaltung in folgenden Zustand gerät 4.6.

$$\frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} e^{2\pi i \theta k} |k\rangle = \frac{1}{\sqrt{2^t}} \left(|0\rangle + e^{2\pi i 2^{t-1} \theta} |1\rangle \right) \otimes \left(|0\rangle + e^{2\pi i 2^{t-2} \theta} |1\rangle \right) \otimes \dots \otimes \left(|0\rangle + e^{2\pi i 2^0 \theta} |1\rangle \right). \quad (4.6)$$

In Anhang A Listing 4 wird die Implementierung der Quantenphasenschätzung für die unitäre Operation $P(\phi = \frac{\pi}{2})$ mit einem Quantenregister von $t = 3$ gezeigt. Das aus dieser Messung erhaltene Ergebnis muss durch 2^t geteilt werden um die Phase θ zu erhalten.

4.4. Shors Algorithmus

Der Shors Algorithmus ist ein randomisierter Algorithmus, dieser besteht aus einem klassischen Teil und einem Quantenteil [13]. Der klassische Teil dieses Algorithmus nutzt modulare Arithmetik und dient zur Vorbereitung der Ordnungsfindung für den Quantenteil.

Um eine Fastprimzahl $N = p \cdot q$ mit dem Shors Algorithmus zu faktorisieren werden folgende Schritte durchgeführt.

1. Finde eine zufällige Zahl (*seed number*) $a < N$ die Teilerfremd zu N ist.
2. Finde die Periode/Ordnung $r \neq 0$ der Funktion

$$f_{a,N}(x) = a^x \mod N$$

3. Faktorisiere N

Die Überprüfung im ersten Schritt kann klassisch durch den euklidischen Algorithmus gelöst werden. Hier muss gelten $\text{ggT}(a, N) = 1$. Teilt a schon N , dann wurde ein Faktor von N gefunden.

Dies ist für Zahlen mit einer Länge von 4096 Bits jedoch ziemlich unwahrscheinlich. Aus diesem Grund muss in Schritt zwei die Periode der Funktion berechnet werden, sodass gilt $f_{a,N}(s+r) = f_{a,N}(s)$. Eine Möglichkeit dieses Problem auf einem klassischen Computer zu lösen, wäre x solange zu erhöhen bis $a^x \equiv 1 \mod N$ ist. Somit ist $x = r$, denn $a^0 \equiv 1 \mod N$ d.h. es wurde eine Periode durchlaufen.

Da dies für große Zahlen jedoch ziemlich uneffizient ist, lag die Idee bei Shor darin die Quanten-Phasenschätzung auf die folgende unitäre Transformation anzuwenden.

$$U|y\rangle = |xy \mod N\rangle \quad (4.7)$$

In [14, p. 196 ff.] wird anschaulich dargestellt, warum der Eigenwert von U die r -ten Wurzel der Einheit (*roots of unity*) ist und somit die Ordnung r aus der Phase des Eigenvektors mittels inverser Quanten Fourier-Transformation extrahiert werden kann.

Wurde diese Ordnung r gefunden kann der 3. Schritt erfolgen. Ist r ungerade muss der Algorithmus erneut von Schritt eins mit einem anderen a durchgeführt werden. Ist r jedoch gerade können die Faktoren von N durch 4.8 bestimmt werden [14].

$$\begin{aligned} q &= \text{ggT}(N, a^{r/2} + 1) \\ p &= \text{ggT}(N, a^{r/2} - 1) \end{aligned} \quad (4.8)$$

Diese Berechnung der Faktoren entsteht durch die Annahme, dass es eine Zahl $x \neq \pm 1$ und $x < N$ gibt, die kongruent 1 Modulo N ist.

$$x^2 - 1 \equiv 0 \mod N \iff (x+1)(x-1) \equiv 0 \mod N \quad (4.9)$$

Die selbe Vorgehensweise kann auch auf die gefundene gerade Ordnung r angewendet werden.

$$\left(a^{r/2}\right)^2 \equiv 1 \pmod{N} \quad (4.10)$$

5. Zusammenfassung

In dieser Arbeit wurden Werkzeuge für die Implementierung von Quantenschaltungen vorgestellt. Diese Ausarbeitung sollte unter anderem zeigen, dass die Bedienung von Qiskit oder dem Quantum Composer allein nicht ausreicht, um Quantenschaltungen bzw. Quantenalgorithmen zu implementieren. Genau wie die Kenntnisse einer Hochsprache wie C, nicht direkt gewährleisten einen Algorithmus implementieren zu können. Denn die Implementierung eines Algorithmus geht damit einher, den Algorithmus sowie dessen Aufgabe zu verstehen. Dies erfordert für klassische Algorithmen oftmals ein fundiertes Wissen in der Mathematik, sowie der Informatik.

Somit ist eines der wichtigsten Werkzeuge zur Implementierung von Quantenalgorithmen, die grundlegende Funktionsweisen von Quantencomputern sowie der Quantenmechanik. Einige dieser Funktionsweisen wie z.B. Quantenbits, Quantengatter, Quantenzustände, Verschränkungen oder unitäre Transformationen wurden in dieser Arbeit aufgegriffen. Eine Möglichkeit effizientere Laufzeiten für klassische Algorithmen zu erhalten, könnte die Überführung des Algorithmus zu einem Quantenalgorithmus sein. Diese Überführung ist nicht trivial und erfordert eine Menge Einfallsreichtum, sowie Verständnis der Quantenmechanik. Daher ist es wichtig, Quantenmechanik und Quanteninformatik Studenten im Grundstudium nahezubringen [15]. Mit quelloffenen und frei verfügbaren Werkzeugen wie Qiskit und Quantum Composer wird dies ermöglicht und die Forschung weiter vorangetrieben.

A.Anhang

```
from qiskit import QuantumRegister,\
    ClassicalRegister, QuantumCircuit

qreg_q = QuantumRegister(3, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
circuit.x(qreg_q[1])
circuit.ccx(qreg_q[0], qreg_q[1], qreg_q[2])
circuit.cx(qreg_q[0], qreg_q[1])

circuit.measure(qreg_q[2], creg_c[1])
circuit.measure(qreg_q[1], creg_c[0])
```

Listing 1: Halbaddierer in Qiskit

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[2];

x q[0];
x q[1];

ccx q[0], q[1], q[2];
cx q[0], q[1];

measure q[2] -> c[1];
measure q[1] -> c[0];
```

Listing 2: Halbaddierer in OpenQASM


```
import numpy as np
from qiskit import QuantumCircuit

# QFT for 3 qubits on the number 3 [101]
qc = QuantumCircuit(3)
qc.x(0)
qc.x(2)

qc.h(0)
qc.cp(np.pi/2, 0, 1)
qc.cp(np.pi/4, 0, 2)

qc.h(1)
qc.cp(np.pi/2, 1, 2)

qc.h(2)
qc.draw()
```

Listing 3: Quantum Fourier-Transformation für 3-Qubits

```

from math import pi
from qiskit import QuantumCircuit

n_qubits = 3
qpe = QuantumCircuit(4, n_qubits)
qpe.x(n_qubits)

for qubit in range(n_qubits):
    qpe.h(qubit)

# controlled unitary P(pi/2)
qpe.cp(pi/2, 2, 3)

qpe.cp(pi/2, 1, 3)
qpe.cp(pi/2, 1, 3)

for i in range(4):
    qpe.cp(pi/2, 0, 3)

# inverse QFT
qpe.h(0)
qpe.cp(-pi/2, 0, 1)
qpe.cp(-pi/4, 0, 2)

qpe.h(1)
qpe.cp(-pi/2, 1, 2)

qpe.h(2)

# measure
qpe.measure(0, 0)
qpe.measure(1, 1)
qpe.measure(2, 2)

```

Listing 4: Quanten Phasenschätzung für $U = P(\phi = \frac{\pi}{2})$

B.Literaturverzeichnis

- [1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.
- [2] M. Homeister, *Quantum Computing verstehen*. Computational Intelligence, Springer Fachmedien Wiesbaden, 2022.
- [3] A. Abbas, S. Andersson, A. Asfaw, A. Corcoles, L. Bello, Y. Ben-Haim, M. Bozzo-Rey, S. Bravyi, N. Bronn, L. Capelluto, A. C. Vazquez, J. Ceroni, R. Chen, A. Frisch, J. Gambetta, S. Garion, L. Gil, S. D. L. P. Gonzalez, F. Harkins, T. Imamichi, P. Jayasinha, H. Kang, A. h. Karamlou, R. Lored, D. McKay, A. Maldonado, A. Macaluso, A. Mezzacapo, Z. Minev, R. Movassagh, G. Nannicini, P. Nation, A. Phan, M. Pistoia, A. Rattew, J. Schaefer, J. Shabani, J. Smolin, J. Stenger, K. Temme, M. Tod, E. Wanzambi, S. Wood, and J. Wootton., *Learn Quantum Computation Using Qiskit*. IBM, 2020.
- [4] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical Review A*, vol. 52, pp. 3457–3467, nov 1995.
- [5] J. Du, M. Shi, X. Zhou, Y. Fan, B. Ye, R. Han, and J. Wu, "Implementation of a quantum algorithm to solve the bernstein-vazirani parity problem without entanglement on an ensemble quantum computer," *Physical Review A*, vol. 64, sep 2001.
- [6] C. Figgatt, D. Maslov, K. A. Landsman, N. M. Linke, S. Debnath, and C. Monroe, "Complete 3-qubit grover search on a programmable quantum computer," *Nature Communications*, vol. 8, dec 2017.
- [7] IBM, "Ibm quantum," 2021. <https://quantum-computing.ibm.com/> Visited: 2022-05-10.
- [8] D. C. McKay, T. Alexander, L. Bello, M. J. Biercuk, L. Bishop, J. Chen, J. M. Chow, A. D. Córcoles, D. Egger, S. Filipp, J. Gomez, M. Hush, A. Javadi-Abhari, D. Moreda, P. Nation, B. Paulovicks, E. Winston, C. J. Wood, J. Wootton, and J. M. Gambetta, "Qiskit backend specifications for openqasm and openpulse experiments," 2018.
- [9] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," 2017.
- [10] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Physical Review A*, vol. 70, nov 2004.

-
- [11] J. Chow, O. Dial, and J. Gambetta, "Ibm quantum breaks the 100 qubit processor barrier," 2021. <https://research.ibm.com/blog/127-qubit-quantum-processor-eagle> Visited: 2022-05-14.
- [12] P. Nation, H. Paik, A. Cross, and Z. Nazario, "The ibm quantum heavy hex lattice," 2021. <https://research.ibm.com/blog/heavy-hex-lattice> Visited: 2022-05-14.
- [13] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, pp. 1484–1509, oct 1997.
- [14] R. Hundt, *Complex Algorithms*, pp. 160-277. Cambridge University Press, 2022.
- [15] S. Fedortchenko, "A quantum teleportation experiment for undergraduate students," 2016.