Titel der Bachelorarbeit

Grundlagen der Quantenmechanik und Qiskit für Quantencomputing

vorgelegt von

Ole Grimsel

zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

Bearbeitungszeit:	11. Juli 2023 bis 11. September 2023
Studiengang:	Fächerübergreifender Bachelor Physik und Informatik
Matrikelnummer:	123456789
Erstgutachter:	Prof. Dr. Tobias J. Osborne
Zweitgutachterin:	Andreea-Iulia Lefterovici

Zusammenfassung

Quantencomputer werden zukünftig in vielen unterschiedlichen Bereichen Anwendung finden, da sie bestimmte Probleme schneller und effizienter als klassische Computer lösen können. Darüber hinaus können mit Quantencomputern Probleme erfolgreich bearbeitet werden, die für klassische Computer unlösbar sind. Dementsprechend müssen künftig viele Menschen in der Lage sein, mit Quantencomputern und Quantentechnologien im Allgemeinen zu arbeiten, wofür ihnen jedoch überwiegend die Grundkenntnisse fehlen. Die Vermittlung der nötigen Grundprinzipien stellt eine Herausforderung dar, da sich quantenmechanische Phänomene grundlegend von Phänomenen der klassischen Physik sowie Phänomenen aus dem Alltag unterscheiden. Verantwortlich dafür ist hauptsächlich das Prinzip der Superposition und die Tatsache, dass die Quantenmechanik eine probabilistische Theorie ist und es sich bei mit ihr getroffenen Vorhersagen um Wahrscheinlichkeiten handelt.

Diese Bachelorarbeit behandelt die quantenmechanischen Grundlagen sowie die Grundlagen von Qiskit für das Quantencomputing, um eine Vermittlung dieser Inhalte zu ermöglichen. Anhand eines vom Verfasser erstellten Jupyter Notebooks wird aufgezeigt, wie diese Grundlagen unter der Verwendung von Qiskit vermittelt werden können. Das Jupyter Notebook kam im Rahmen eines Projektes des Theoretischen Instituts für Physik der Leibniz Universität Hannover an verschiedenen Schulen zum Einsatz und lässt sich ebenso auf andere Kontexte zur Vermittlung der dargestellten Inhalte übertragen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ASCII ..... American Standard Code für Information Interchange

Bit ..... binary digit

Qubit ..... Quantenbit

SDK ..... Software Development Kit

1 Einleitung

Am 14. Dezember des Jahres 1900 wurde ein Vortrag gehalten, dessen Inhalt die Entwicklung der Menschheit bis heute fundamental beeinflusst [1]. Der deutsche Physiker Max Planck referierte „Zur Theorie des Gesetzes der Energieverteilung im Normalspektrum“ in dem Kolloquium der Physikalischen Gesellschaft [1, 2]. Der Vortrag legte mit dem Konzept der Energiequantelung den Grundstein für die Entwicklungen der Quantentheorie und der Quantenmechanik [1]. Daher wurde der 14. Dezember 1900 von Max von Laue retrospektiv als „Geburtstag der Quantentheorie“ (s. [3], S. 4) bezeichnet. Dieser Tag ist Ausgangspunkt für viele technische Revolutionen, die heute allgegenwärtig sind, wie beispielsweise den Laser, Halbleiter und somit auch den klassische modernen Computer [4].

Die Weiterentwicklung des Computers folgt bislang dem Mooreschen Gesetz [5]. Das Gesetz wurde 1965 von Gordon Moore aufgestellt und besagt, dass sich die Anzahl von Transistoren auf einem Chip ungefähr alle zwei Jahre verdoppelt, wodurch es einen exponentiellen Leistungszuwachs gibt [6]. Jedoch ist die Realisierung der Transistoren auf der atomaren Größenordnung angelangt, weshalb diese den Gesetzen der Quantenmechanik folgen [4]. Dies führt zu unerwünschten Quanteneffekten, die die wirkliche Umsetzung funktionierender Transistoren erschweren bis unmöglich machen und somit die bestehende Gültigkeit des Mooreschen Gesetzes gefährden [5].

Eine Lösung für das mögliche Ende des Mooreschen Gesetzes könnten Quantencomputer darstellen [5]. Quantencomputer basieren vollständig auf der Quantenmechanik, das heißt, die Durchführung von Berechnungen basiert auf der Quantenmechanik [5]. Die Hauptaspekte sind dabei Qubits, welche die kleinste Informationseinheit von Quantencomputern darstellen, und die Überlagerung von Zuständen, das sogenannte Superpositionsprinzip, sowie Messungen, die den finalen Zustand von Qubits determinieren [5]. Mit Quantencomputern lassen sich bestimmte Problemstellungen schneller lösen als mit klassischen Computern und es gibt Probleme, die ausschließlich mit Quantencomputern gelöst werden können [5]. Dementsprechend wird die Bedeutung der Technologie immer weiter zunehmen. Dies wird zudem daran deutlich, dass die EU von 2020 bis 2030 Vorhaben im Bereich der Quantentechnologie durch das Projekt „European Quantum Flagship“ mit 1 Milliarde Euro fördert [7].

Wenn Quantencomputer kommerziell Einzug in die Wirtschaft halten, müssen die betreffenden Personen über die nötigen Kenntnisse verfügen, um mit der Technologie umgehen können. Dafür reichen im Kern einfache Prinzipien [4]. Jedoch sind die grundlegenden physikalischen Prinzipien von Quantentechnologien nicht intuitiv verständlich [4]. Sie stehen konträr zu Alltagserfahrungen und erfordern in der Tiefe viel Grundlagenwissen, was einen Einstieg in die Thematik erschwert [4]. Daher stellt insbesondere die Vermittlung der Grundlagen zum Verständnis von Quantencomputern eine zentrale Herausforderung dar.

Die Vermittlung der Inhalte sollte nicht erst in den Firmen beginnen, die aktuell oder potenziell mit Quantencomputern arbeiten, sondern bereits in Schulen, um zukünftige Fachkräfte frühzeitig grundlegend mit der Thematik vertraut zu machen. Die Quantenphysik ist zwar Teil des Kerncurriculums für die gymnasiale Oberstufe des Unterrichtsfachs Physik, jedoch erfolgt die Auseinandersetzung mit dem Themenkomplex in Hinblick auf die für das Quantencomputing erforderlichen Inhalte oberflächlich und Quantencomputer werden nicht behandelt [8]. Aus diesem Grund wurde am Institut für Theoretische Physik der Leibniz Universität ein Projekt initiiert. Das Projekt bestand darin, Schulen in der Region Hannover zu besuchen und eine Unterrichtsstunde durchzuführen, in der die Grundlagen der Quantenmechanik für das Quantencomputing vermittelt werden. Zu diesem Zweck wurde von dem Verfasser dieser Bachelorarbeit ein Jupyter Notebook zur Vermittlung der Grundlagen erstellt, welches durch die Verwendung von Qiskit eine einfache Implementierung von quantenmechanischen Experimenten ermöglicht. Herkömmliche quantenmechanische Experimente werden häufig lediglich als Gedankenexperimente durchgeführt oder sind sehr aufwändig zu realisieren [9].

Der grundsätzliche Verlauf der Unterrichtsstunde wird in der Bachelorarbeit „Didaktische Analyse einer Unterrichtsstunde zur Vermittlung quantenmechanischer Grundlagen“ von Lara Niemann ausgeführt und didaktisch begründet. In der Bachelorarbeit „Statistische Analyse der konzeptionellen Durchführung einer Unterrichtsstunde zur Vermittlung quantenmechanischer Grundlagen“ von Maren Lankhorst wird untersucht, inwieweit die Unterrichtsstunde ein effektives Mittel zur Steigerung der quantenmechanischen Kenntnisse bietet. Dabei wird der Wissenszuwachs analysiert.

Die vorliegende Bachelorarbeit zielt darauf ab, die quantenmechanischen Grundlagen sowie die Grundlagen von Qiskit objektiv und verständlich darzustellen, sodass eine Vermittlung dieser mit dem Einsatz von Qiskit möglich ist. Eine exemplarische Vermittlung wird anhand des für das Schulprojekt erstellten Jupyter Notebooks ausgeführt und kann dementsprechend damit erfolgen. Im ersten Kapitel der Arbeit werden die Grundlagen der Quantenmechanik für das Quantencomputing dargestellt (2). Daran anschließend wird in Kapitel 3 Qiskit vorgestellt und die Implementierung der zuvor erläuterten Grundlagen beispielhaft durchgeführt. Schließlich mündet die Arbeit in einer begründeten Beschreibung der Struktur des Jupyter Notebooks (4). Dieses repräsentiert ein Beispiel zur Vermittlung der zuvor dargestellten Grundlagen.

2 Grundlagen der Quantenmechanik für das Quantencomputing

In diesem Kapitel werden die grundlegenden Prinzipien der Quantenmechanik erläutert, die für das Verständnis von Quantencomputern unabdingbar sind. Daher sollten sie primär bei der Vermittlung der quantenmechanischen Grundlagen für das Quantencomputing berücksichtigt werden. Quantencomputer basieren auf den Grundsätzen der Quantenmechanik und arbeiten mit Quantenbits, die kurz als Qubits bezeichnet und gemeinsam mit den klassischen Bits in 2.1 ausgeführt werden [4]. Abschnitt 2.2 geht anschaulich auf das Konzept der Zustände ein, wofür die Bloch-Kugel eingeführt wird. Mit Qubits werden analog zu Bits in klassischen Computern Rechnungen durchgeführt. Um die Rechnungen zu verwirklichen, werden Qubits zu Quantenschaltkreisen kombiniert, wofür Quantengatter verwendet werden (2.3) [5]. Der Frage, was mit einem Qubit passiert, wenn eine Messung daran durchgeführt wird, widmet sich 2.4. Abschließend werden aktuelle und potenzielle zukünftige Anwendungsmöglichkeiten von Quantencomputern in 2.5 dargestellt.

2.1 Bits und Quantum Bits

Bits und Qubits sind jeweils die kleinste Speichereinheit auf einem klassischen bzw. quantenmechanischen Computer [5]. Im Gegensatz zu Bits können Qubits zum einen grundsätzlich mehr als zwei verschiedene Zustände und zum anderen mehrere Zustände zugleich annehmen, wodurch bestimmte Problemstellungen schneller und effizienter gelöst werden können als mit einem klassischen Computer [5].

2.1.1 Eigenschaften von Bits

Ein klassischer Computer codiert Informationen in Form von Bits [10], wobei sich der Name *Bit* aus dem englischen „binary digit“ ergibt [6]. Es wird somit das binäre Zahlensystem genutzt, um Daten darzustellen [10]. Ein Bit kann einen von zwei möglichen Zuständen annehmen, die für gewöhnlich mit „0“ und „1“ bezeichnet werden [10]. In einem klassischen Computer werden die beiden potenziellen Zustände mit „Strom aus“ und „Strom an“ realisiert [10]. Das Zahlensystem basiert dementsprechend auf zwei Zuständen pro Bit, mit deren Kombinationen Buchstaben oder auch Dezimalzahlen codiert und gespeichert werden können, wodurch ein Bit jeweils eine Stelle repräsentiert [10]. Dies wird als Binärcode bezeichnet [10]. Beispielsweise entspricht die Binärzahl „1101_2“ der Dezimalzahl „13_{10}“, wobei der Index jeweils das zugrundeliegende Zahlensystem angibt. Die Umrechnung lautet wie folgt [11]:

$$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = 13_{10}.$$

Im Alltag ist eher der Begriff Byte präsent, der vor allem im Zusammenhang mit Speicherkapazitäten von Handys oder Computern genannt wird. Dabei besteht 1 Byte aus 8 Bit [6]. In den Kontexten Handy und Computer werden primär die Größenordnungen Gigabyte ($1\ GB = 10^9\ Byte$) und Megabyte ($1\ MB = 10^6\ Byte$) verwendet [6]. Mit 1 Byte, das heißt 8 Bit und somit 8 Stellen lassen sich insgesamt $258 = 2^8$ verschiedene Zeichen codieren. Allgemein gilt, mit n Bits lassen sich 2^n Zeichen darstellen [6].

Um Buchstaben, Ziffern und Sonderzeichen eindeutig zu codieren, wird der ASCII-Code genutzt [10]. Dieser wurde zunächst mit 7 Bits definiert und auf insgesamt 8 Bits erweitert, um sprachenspezifische Zeichen wie Umlaute darstellen zu können [10]. Der Buchstabe „A“ entspricht dem ASCII-Code folgernd „$0100\ 0001_2$“ [6]. Klein- und Großbuchstaben unterscheiden sich im ASCII-Code nur in der sechsten Stelle, daher steht „$0110\ 0001_2$“ für „a“ [6]. Dabei gilt es zu beachten, dass Binärzahlen von rechts nach links mit Null beginnend gezählt und für gewöhnlich in Vierergruppen dargestellt werden [6]. Dementsprechend ändert sich bei dem Vergleich von „A“ und „a“ das sechste Bit von rechts von einer „0“ zu einer „1“.

Wie bereits ausgeführt lassen sich mit dem 8 Bit ASCII-Code 256 Zeichen codieren. Um alle 256 Zeichen zeitgleich speichern zu können, sind $8 \cdot 256 = 2048\ Bits$ nötig, da jedes Zeichen 8 Bit als Speicherplatz belegt [6]. Wenn die doppelte Menge an Zeichen, die doppelte Menge an Informationen codiert werden soll, ist die doppelte Anzahl an Bits erforderlich. Dies ist darauf zurückzuführen, dass sich ein Bit *entweder* im Zustand „0“ *oder* im Zustand „1“ befindet. Daher lässt sich ein Bit mit einer Münze assoziieren [5]. Bei einem Münzwurf ist entweder Kopf oder Zahl das Ergebnis, wobei die sehr unwahrscheinliche Möglichkeit, dass die Münze auf dem Rand landet, nicht mit betrachtet wird. Ebenso kann ein Lichtschalter zur Veranschaulichung herangezogen werden [12]. Entweder leuchtet die Lampe nicht („0“) oder sie leuchtet („1“) [12].

2.1.2 | Eigenschaften von Qubits

Das quantenmechanische Analogon zu klassischen Bits sind Qubits [5]. Qubits sind Zwei-Zustands-Quantensysteme [13], die als mathematische Objekte definiert werden können [5]. Die zwei grundlegenden Zustände von Qubits werden mit $|0\rangle$ und $|1\rangle$ bezeichnet, die analog zu den zwei möglichen Zuständen klassischer Bits verstanden werden können [5]. Zustände werden in der Quantenmechanik in der Dirac-Notation dargestellt, die nach Paul Dirac (1902-1984) benannt ist [12]. $|0\rangle$ wird als „Ket-Vektor-Null“ oder kurz „Ket-Null“ gelesen und steht für einen zweidimensionalen Zustandsvektor [14]. Für $|0\rangle$ und $|1\rangle$ gilt [15]:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ und } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Die beiden möglichen Zustände eines Qubits bilden dabei die Basis eines zweidimensionalen Hilbertraums $\mathcal{H}$, der mit $\mathbb{C}^2$ bezeichnet wird und der Zustandsraum eines Qubits ist [15]. Ausgehend von den Basiszuständen eines Hilbertraums $\mathcal{H}$ können alle Zustände eines Quantensystems gebildet werden [7]. Ein Hilbertraum $\mathcal{H}$ ist ein komplexer Vektorraum mit einem Skalarprodukt [7]. Das Skalarprodukt für den Vektorraum $\mathbb{C}^n$ ist wie folgt

definiert [5]:

$$\langle\psi|\varphi\rangle = \begin{pmatrix}\psi_1^* & \psi_2^* & \dots & \psi_n^*\end{pmatrix} \cdot \begin{pmatrix}\varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_n\end{pmatrix} = \sum_{n=1}^{N} \psi_n^* \varphi_n$$

Ergibt die Berechnung des Skalarprodukts zwischen zwei Vektoren Null, so sind diese orthogonal zueinander [5]. Die Basisvektoren des Zustandsraums des Qubits sind orthogonal zueinander und haben darüber hinaus die Länge 1, was in den folgenden Rechnungen gezeigt wird. Die Berechnung der Länge wird in 2.3 und 2.4 durchgeführt und erfolgt mittels der Norm [5].

$$\langle 0|1\rangle = \begin{pmatrix}1 & 0\end{pmatrix} \cdot \begin{pmatrix}0 \\ 1\end{pmatrix} = 1 \cdot 0 + 0 \cdot 1 = 0 \tag{2.1}$$

$$\langle 1|0\rangle = \begin{pmatrix}0 & 1\end{pmatrix} \cdot \begin{pmatrix}1 \\ 0\end{pmatrix} = 0 \cdot 1 + 1 \cdot 0 = 0 \tag{2.2}$$

$$\||0\rangle| = \sqrt{\langle 0|0\rangle} = \sqrt{\begin{pmatrix}1 & 0\end{pmatrix} \cdot \begin{pmatrix}1 \\ 0\end{pmatrix}} = \sqrt{1 \cdot 1 + 0 \cdot 0} = 1 \tag{2.3}$$

$$\||1\rangle| = \sqrt{\langle 1|1\rangle} = \sqrt{\begin{pmatrix}0 & 1\end{pmatrix} \cdot \begin{pmatrix}0 \\ 1\end{pmatrix}} = \sqrt{0 \cdot 0 + 1 \cdot 1} = 1 \tag{2.4}$$

Demnach bilden $|0\rangle$ und $|1\rangle$ eine orthonormale Basis des Hilbertraums, da sie orthogonal zueinander und jeweils normiert sind [5]. Die Kombination der beiden Basiszustände lässt sich mathematisch wie folgt formulieren [5]:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle, \text{ mit } \alpha, \beta \in \mathbb{C} \tag{2.5}$$

Der Zustand eines Qubits ist somit mathematisch ausgedrückt eine Linearkombination der möglichen Basiszustände [5]. Dabei sind α und β komplexe Zahlen, auf die in Abschnitt 2.4 genauer eingegangen wird [5]. Anhand der Gleichung lässt sich eine zentrale Eigenschaft von Qubits ablesen: Qubits können sich nicht nur in Zustand $|0\rangle$ oder Zustand $|1\rangle$ befinden, sondern auch in jedem Zustand zwischen den beiden Basiszuständen, der sich durch die Linearkombination dieser ergibt [5]. Diese Eigenschaft wird als Superposition bezeichnet [5]. Das Konzept der Superposition beschreibt die Überlagerung der Basiszustände eines Quantenteilchens zu einem Gesamtzustand, einem Überlagerungszustand [12]. Ein Qubit lässt sich dementsprechend nicht mit einer Münze oder einem Lichtschalter assoziieren. Unter Berücksichtigung der Grenzen kann ein Dimmer als anschauliches Modell für ein Qubit herangezogen werden. Jedoch werden nicht alle Komponenten von α und β beachtet, weshalb sich eine Kugel, insbesondere als Äquivalenz zu der Münze für ein Bit, unter physikalischen Gesichtspunkten besser eignet [5]. Abschnitt 2.2 geht näher auf diesen Aspekt ein. Doch auch der Dimmer zeigt bereits anschaulich, dass sich ein Qubit nicht nur in den Zuständen $|0\rangle$ („Licht aus“) oder $|1\rangle$ („Licht an “) befinden, sondern auch alle Zustände dazwischen annehmen kann. Wenn mehrere Qubits betrachtet werden, ist eine Superposition ebenfalls möglich [5]. Beispielhaft wird dies mit zwei Qubits, einem

Qubitpaar dargestellt. Für zwei Bits sind die Zustände „00“, „01“, „10“ und „11“ möglich. Analog dazu bilden die Zustände $|00\rangle$, $|01\rangle$, $|10\rangle$ und $|11\rangle$ die Basis des Hilbertraums für ein Qubitpaar [5]. Für das Qubitpaar ergibt sich daher [5]:

$$|\Psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle, \text{ mit } \alpha_{nm} \in \mathbb{C} \text{ und } n, m \in \{0, 1\}.$$

Neben der Superposition ist ein weiteres zentrales Phänomen der Quantenmechanik, das für das Quantencomputing eine entscheidende Rolle spielt, die Verschränkung [5]. Diese steht im intuitiven Sinne so konträr zu den Regeln der klassischen Physik, insbesondere der Relativitätstheorie, dass Einstein sie als „spukhafte Fernwirkung“ bezeichnet hat [9]. Im Kern bedeutet Verschränkung, dass zwei oder mehr quantenmechanische Teilchen, beispielsweise Qubits, über beliebige Distanzen miteinander verbunden sind [12]. Das heißt, wenn etwas an einem der quantenmechanischen Teilchen verändert wird, verändern sich die verschränkten Teilchen ebenfalls [12]. Dies schließt ein, dass die Zustände der einzelnen quantenmechanischen Teilchen nicht isoliert beschrieben werden können, sondern nur der Gesamtzustand der verschränkten quantenmechanischen Teilchen [12]. Inzwischen wurden bereits Photonen über eine Distanz von 1203 km miteinander verschränkt, wobei die Photonen mit Satelliten voneinander distanziert wurden [16]. Durch die Verschränkung ist die Rechenleistung von Quantencomputern bei bestimmten mathematischen Problemen sehr groß, insbesondere verglichen mit klassischen Computern [12]. Im Gegensatz zu einem klassischen Computer müssen die Daten für die Berechnung nicht von einem Bit zum anderen weitergegeben werden, weil ein Rechenschritt mit allen beteiligten verschränkten Qubits zeitgleich vollzogen wird [12]. Die Menge an Information, die ein Netzwerk von verschränkten Qubits enthält, ist somit größer als die Summe der Informationen, die jedes Qubit für sich betrachtet enthält [12]. Demnach steigt die Rechenleistung eines Quantencomputers exponentiell mit der Anzahl der verwendeten Qubits an, während die Rechenleistung eines klassischen Computers linear mit der Anzahl der Bits steigt [12].

Die Realisierung von Qubits kann auf verschiedene Art und Weise erfolgen, darunter die Verwendung von Atomen, Elektronen oder Photonen [12]. Ein wesentliches Kriterium ist, dass Qubits wieder in ihren Grundzustand zurückkehren können [12]. Darüber hinaus müssen Messungen an dem Qubit durchgeführt und Quantengatter darauf angewendet werden können [12]. Ein konkretes Beispiel für die Realisierung eines Qubits ist die Betrachtung des Zustands eines Elektrons, der durch zwei verschiedene Positionen repräsentiert werden kann [5].

Insbesondere anhand der zuletzt genannten Realisierungsmöglichkeit von Qubits lässt sich erkennen, dass es in der Praxis zu Herausforderungen bei der Umsetzung von Qubits und Quantencomputern im Allgemeinen kommt [5]. Wenn ein Qubit wie beschrieben durch zwei unterschiedliche Positionen eines Elektrons realisiert wird, interagiert ebendieses Elektron mit seiner Umgebung [5]. Solche unerwünschten Interaktionen mit der Außenwelt weisen ideale geschlossene Quantensysteme nicht auf [5]. Jedoch sind reale Quantensysteme nicht ideal und daher niemals vollständig geschlossen [5]. Dies führt zu dem sogenannten Quantenrauschen, das im Englischen als quantum noise bezeichnet wird [5]. Offene und geschlossene Quantensysteme unterscheiden sich somit dahingehend, dass offene Quantensysteme mit externen Systemen wechselwirken. Eine mögliche Folge von Quantenrauschen ist Dekohärenz, wobei die Begriffe teilweise auch synonym verwendet werden [5]. Dekohärenz beschreibt die Abnahme der Stabilität eines quantenmechanischen

Zustands unter anderem durch die Wechselwirkung mit externen Systemen, demnach durch Quantenrauschen [15]. Damit einher geht ein möglicher Informationsverlust, beispielsweise durch einen Qubit-Flip, das heißt, der Zustand eines Qubits ändert sich, z. B. $|0\rangle \xrightarrow{Qubit-Flip} |1\rangle$, oder einen Phasen-Flip, der durch das Vorzeichen eines komplexen Vorfaktors mathematisch dargestellt wird: $\alpha |0\rangle + \beta |1\rangle \xrightarrow{Phasen-Flip} \alpha |0\rangle - \beta |1\rangle$ [12]. Daraus lässt sich ableiten, dass Quantenrauschen auch den Messvorgang beeinflusst, was in dem Abschnitt 2.4 deutlicher wird [12]. Die Effekte von Quantenrauschen werden durch die Verschränkung und die damit verbundene Beeinflussung der verschränkten Qubits untereinander von noch größerer Bedeutung [12].

Um trotz der Herausforderungen durch Quantenrauschen einen Quantencomputer realisieren zu können, müssen die auftretenden Fehler korrigiert werden [5]. Dafür werden oberflächlich beschrieben beispielsweise Messungen durchgeführt, die prüfen, ob sich das quantenmechanische System wie erwartet verhält [5]. Darüber hinaus werden sogenannte Quantum Error Correction Algorithmen entwickelt, die *quantum error correcting codes* [5]. Diese basieren darauf, die Quantenzustände so zu codieren, dass das Quantenrauschen keinen Einfluss auf sie hat bzw. der Einfluss korrigiert werden kann [5]. Die Zustände werden erst wieder decodiert, wenn sie benötigt werden [5]. Dabei werden analog zu den *error correcting codes* (ECC) auf klassischen Computern [17] zusätzliche Qubits hinzugefügt, anhand derer eine Paritätsprüfung durchgeführt wird [5, 17]. In dem Kontext Bits steht Parität für die Anzahl der Bits, die sich in Zustand 1 befinden [17]. Es liegt ein Übertragungsfehler vor, wenn die Parität gerade ist [17]. Um mögliche Fehler zu erkennen, wird somit künstlich Redundanz erzeugt [5, 17]. Wird ein Fehler identifiziert, so wird er durch den Algorithmus unter Anwendung eines entsprechenden Gatters (s. Abschnitt 2.3.2) korrigiert [5].

Qubits unterscheiden sich somit grundlegend von klassischen Bits. Die Divergenz zeigt sich vor allem in den quantenmechanischen Konzepten der Superposition, wodurch sich ein Qubit nicht nur in den Zuständen $|0\rangle$ oder $|1\rangle$, sondern auch in jedem beliebigen Zustand dazwischen befinden kann, und in der Verschränkung. Darüber hinaus muss bei der Realisierung von Qubits und Quantencomputern insgesamt quantum noise und die dadurch notwendige Quantum Error Correction berücksichtigt werden.

2.2 Die Bloch-Kugel

In Abschnitt 2.1.2 wurde bereits angedeutet, dass ein Dimmer als Modell für ein Qubit zwar sehr anschaulich ist, jedoch eindeutige Grenzen hat und sich eine Kugel unter physikalischen Gesichtspunkten besser eignet. Das für Qubits verwendete Modell ist die Bloch-Kugel, die auf den Schweizer Physiker Felix Bloch (1905-1983) [18] zurückgeht [5]. Die Bloch-Kugel stellt allgemein den Zustandsraum eines Quantensystems geometrisch dar und wird speziell zur Visualisierung von Qubits verwendet (s. Abb. 2.1) [5]. Der Zustandsraum wird durch die Oberfläche der Bloch-Kugel, welche einen Radius von 1 hat, repräsentiert, wodurch jeder Punkt auf der Oberfläche einem möglichen Zustand des Qubits entspricht [5]. Der „Nordpol“ verkörpert dabei den Basiszustand $|0\rangle$ und der „Südpol“ den Basiszustand $|1\rangle$ [12]. Zwischen den Polen liegen demnach genau die Zustände, die sich durch Superposition ergeben [7]. Die Eigenschaft, die von dem Dimmer nicht veranschaulicht werden kann, ist die Phase des Zustands eines Qubits, welche

mithilfe der Rotation um die Achse der Bloch-Kugel repräsentiert wird [5]. Deutlicher wird dies, wenn in der Gleichung 2.5 α und β umgeschrieben werden, wobei $|\alpha|^2 + |\beta|^2 = 1$ gilt [5]. Warum dieser Zusammenhang nicht nur gilt, sondern gelten muss, wird genauer in Abschnitt 2.4 beleuchtet. Es ergibt sich [5]:

$$|\Psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right) \tag{2.6}$$

Anhand von 2.6 lässt sich direkt ablesen, dass sich abhängig von den Faktoren vor $|0\rangle$ und $|1\rangle$ der Winkel zwischen dem Pfeil, der das Qubit repräsentiert und der Z-Achse verändert. Der Faktor $e^{i\gamma}$ wirkt sich nicht sichtbar aus und wird daher zum Teil nicht notiert [5].

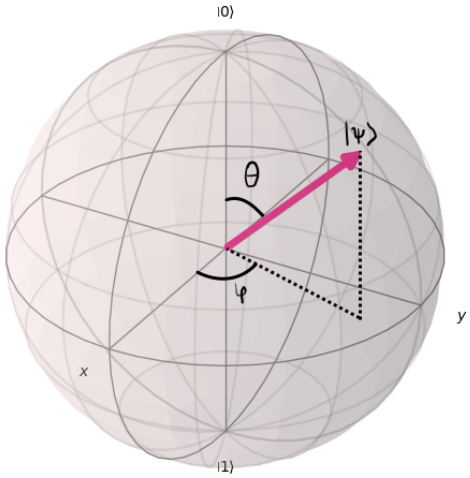


Abbildung 2.1.: Die Bloch-Kugel, die den Zustandsraum eines Quantensystems darstellt. Die Winkel sind entsprechend eingezeichnet. Abbildung eigens erstellt nach [5] S. 15.

Die folgende Abbildung 2.2 zeigt zwei Zustände, die mit der Bloch-Kugel visualisiert werden. Links ist der Zustand $|+\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$ dargestellt [12]. Die rechte Bloch-Kugel zeigt den Zustand $|-\rangle = \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle$ [12]. Diese beiden Zustände werden häufig verwendet und haben daher eine eigene Bezeichnung. Beide Zustände entsprechen nicht genau einem der Basiszustände, sondern ergeben sich aus einer Überlagerung dieser. Die Bloch-Kugel ermöglicht somit eine intuitive Darstellung von Qubits, insbesondere für das Konzept der Superposition.

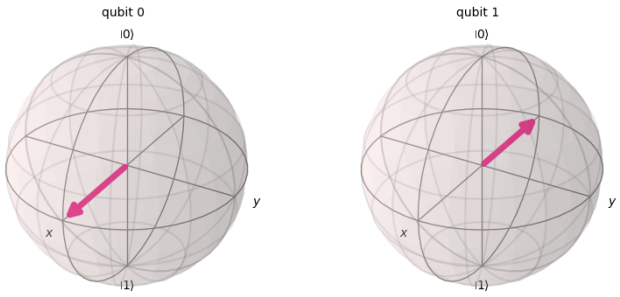


Abbildung 2.2.: Die linke Bloch-Kugel zeigt das Qubit 0, welches sich in dem Zustand $|+\rangle$ befindet. Qubit 1 in Zustand $|-\rangle$ wird anhand der rechten Bloch-Kugel dargestellt.

2.3 | Gatter und Schaltkreise

Um mit Bits und Qubits Rechnungen durchführen zu können, sind sowohl im klassischen Computer als auch im Quantencomputer Schaltkreise nötig [5]. Für einen klassischen Schaltkreis werden klassische Gatter verschaltet [6]. Analog dazu wird ein Quantenschaltkreis aus Quantengattern gebildet [5]. Die relevantesten klassischen und quantenmechanischen Gatter werden im Folgenden vorgestellt. Die Schaltsymbole wurden eigens erstellt. Dafür dienten die Darstellungen von Nielsen und Chuang in [5] als Grundlage und wurden mit den Versionen von Ernst aus [6] ergänzt, die nach der DIN 40700 und der IEC 60617-12 Norm erstellt wurden [6]. Die Darstellungen der Quantengatter wurden eigens mithilfe von Qiskit erstellt [19]. Dabei wurde `mpl` als Darstellungsform gewählt [19].

2.3.1 | Logische Gatter

Logische Gatter, die auch als Logikgatter oder klassische Gatter bezeichnet werden, führen logische Operationen durch [6]. Für ihre Realisierung werden Transistoren und Widerstände zu einer elektrischen Schaltung verknüpft [6]. Diese elektrische Schaltung verfügt über einen oder mehrere Ausgänge und einen oder mehrere Eingänge, an denen jeweils keine bzw. eine niedrige Spannung für den Zustand 0 und eine im Verhältnis dazu hohe Spannung für den Zustand 1 anliegt [6]. Der Zustand an den Ausgängen ergibt sich durch die logische Operation auf die Eingangszustände [6]. Mathematisch lässt sich die Funktion eines Logikgatters wie folgt beschreiben: $f : \{0,1\}^n \rightarrow \{0,1\}^m$[5]. Wenn ein logisches Gatter nur einen Ausgang aufweist, realisiert es eine boolesche Funktion, wodurch sich der obige mathematische Ausdruck zu $f : \{0,1\}^n \rightarrow \{0,1\}$ reduziert [5]. Die konkrete Funktion der Logikgatter lässt sich in Wahrheitstabellen ausdrücken [10].

Das in seiner Funktionsweise simpelste logische Gatter ist das NOT-Gatter, welches jeweils nur einen Eingang und einen Ausgang hat (s. Abb. 2.3) [6]. Mit dem NOT-Gatter wird eine Invertierung des Eingangssignals verwirklicht und an den Ausgang angelegt [6]. Das bedeutet, wenn sich das Bit am Eingang im Zustand 0 befindet, liegt am Ausgang eine 1 an und umgekehrt. Es ergibt sich die folgende Wahrheitstabelle 2.1.

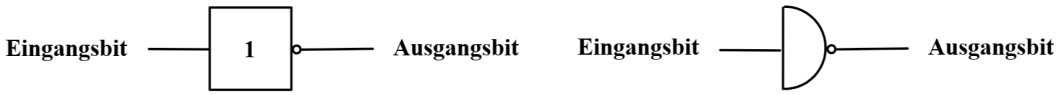


Abbildung 2.3.: Zwei mögliche Schaltsymbole für das NOT-Gatter. Charakteristisch ist die „1“ innerhalb und der kleine Kreis, der die Invertierung darstellt, am Ausgang des Gattersymbols.

Tabelle 2.1.: Die Wahrheitstabelle des NOT-Gatters

Eingangsbit	Ausgangsbit
0	1
1	0

Neben dem NOT-Gatter, dem einzigen Ein-Bit-Gatter, das nicht trivial ist, gibt es verschiedene klassische Gatter mit zwei Eingängen und einem Ausgang [5]. Zu dieser Kategorie zählt das AND-Gatter (s. Abb. 2.4), mithilfe dessen die UND-Verknüpfung realisiert wird [6]. Demnach ist es im Prinzip ein Vergleich der beiden Eingänge, durch den sich für den Ausgang nur 1 ergibt, wenn auch an beiden Eingängen eine 1 anliegt 2.2. Die Eingänge des Gatters werden, wie in der Informatik üblich, von 0 beginnend und von rechts nach links bzw. oben nach unten aufsteigend benannt.

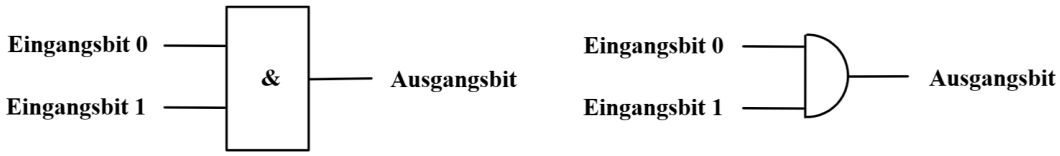


Abbildung 2.4.: Zwei mögliche Schaltsymbole für das AND-Gatter. Charakteristisch ist das &-Zeichen innerhalb des Gattersymbols.

Tabelle 2.2.: Die Wahrheitstabelle des AND-Gatters

Eingangsbit 1	Eingangsbit 0	Ausgangsbit
0	0	0
0	1	0
1	0	0
1	1	1

Als drittes grundlegendes Gatter ist das OR-Gatter (s. Abb. 2.5) zu nennen [6]. Die Funktion ist in der zugehörigen Wahrheitstabelle 2.3 ersichtlich. Befindet sich mindestens eines der beiden Eingangsbits in Zustand 1, so wird das Ausgangsbit in den Zustand 1 gesetzt [6]. Wenn beide Eingangsbits in Zustand 0 sind, wird das Ausgangsbit in Zustand 0 ausgegeben. Es fungiert somit als ODER/UND-Verknüpfung der Eingangsbits. Das heißt, das Ausgangsbit ist ebenfalls in Zustand 1, wenn sich beide Eingangsbits in Zustand 1 befinden [6].

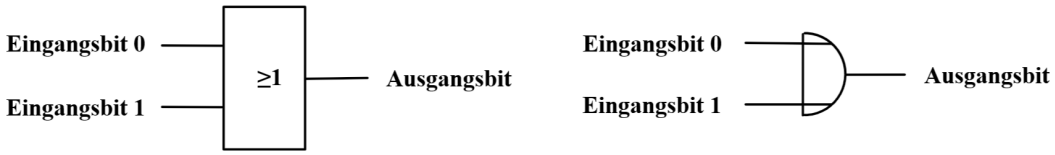


Abbildung 2.5.: Zwei mögliche Schaltsymbole für das OR-Gatter. Charakteristisch ist das „≥ 1“ innerhalb des Gattersymbols.

Tabelle 2.3.: Die Wahrheitstabelle des OR-Gatters

Eingangsbit 1	Eingangsbit 0	Ausgangsbit
0	0	0
0	1	1
1	0	1
1	1	1

Die Schaltskizze des Logikgatters, das die Funktion entweder... oder... realisiert, ist in Abbildung 2.6 zu sehen und wird als XOR-Gatter bezeichnet [15]. XOR steht dabei für exklusiv-OR [5]. Demnach wird das Ausgangsbit nur auf 1 gesetzt, wenn sich das Eingangsbit 0 *oder* das Eingangsbit 1 in Zustand 1 befinden [15]. Das Ausgangsbit ist somit nicht in Zustand 1, wenn beide Eingangsbits in Zustand 1 sind (s. Tab. 2.4).

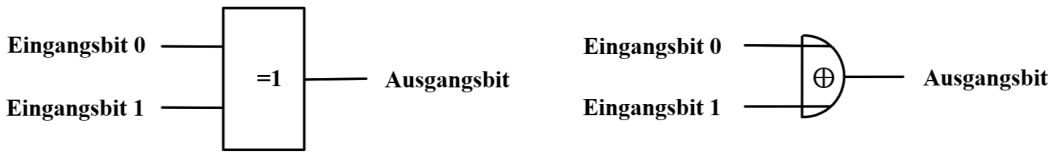


Abbildung 2.6.: Zwei mögliche Schaltsymbole für das XOR-Gatter. Charakteristisch ist das „$= 1$“ oder das „$\oplus$“ innerhalb des Gattersymbols.

Tabelle 2.4.: Die Wahrheitstabelle des XOR-Gatters

Eingangsbit 1	Eingangsbit 0	Ausgangsbit
0	0	0
0	1	1
1	0	1
1	1	0

Aus den bereits bekannten logischen Gattern lassen sich alle möglichen Schaltfunktionen konstruieren [17]. Mit Kombinationen der Logikgatter ergeben sich zudem weitere gängige Gatter. Wird nach einer AND-Verknüpfung das Ausgangsbit invertiert, erfüllt das gesamte Gatter die Funktion eines NOT-AND-Gatters und wird als NAND-Gatter bezeichnet [6]. In Abbildung 2.7 ist zu erkennen, dass die Invertierung durch einen kleinen Kreis am Ausgang des AND-Gatters gekennzeichnet wird [6]. Die Wahrheitstabelle entspricht

daher der Wahrheitstabelle des AND-Gatters mit invertiertem Ausgangsbit (s. Tab. 2.5). Das NAND-Gatter ist das universelle logische Gatter, das heißt, jede mögliche Funktion auf die Eingangsbits kann mit einer Kombination von NAND-Gattern realisiert werden [5].

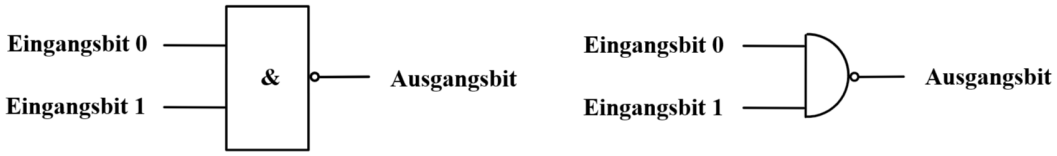


Abbildung 2.7.: Zwei mögliche Schaltsymbole für das NAND-Gatter. Charakteristisch ist das &-Zeichen innerhalb und der kleine Kreis, der die Invertierung darstellt, am Ausgang des Gattersymbols.

Tabelle 2.5.: Die Wahrheitstabelle des NAND-Gatters

Eingangsbit 1	Eingangsbit 0	Ausgangsbit
0	0	1
0	1	1
1	0	1
1	1	0

Wird das OR-Gatter mit einem NOT-Gatter verschaltet, ergibt sich dementsprechend ein NOR-Gatter (s. Abb. 2.8) [17]. Die Wahrheitstabelle ist folglich die Wahrheitstabelle des OR-Gatters mit negiertem Ausgangsbit (s. Tab. 2.6) [17].

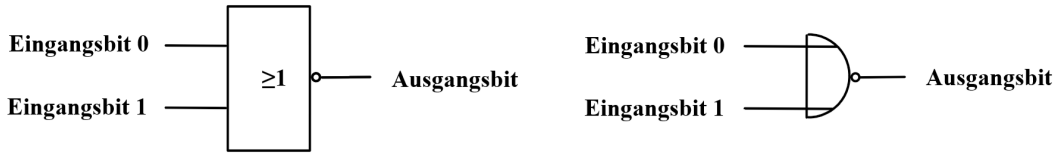


Abbildung 2.8.: Zwei mögliche Schaltsymbole für das NOR-Gatter. Charakteristisch ist das „≥ 1“ innerhalb und der kleine Kreis, der die Invertierung darstellt, am Ausgang des Gattersymbols.

Tabelle 2.6.: Die Wahrheitstabelle des NOR-Gatters

Eingangsbit 1	Eingangsbit 0	Ausgangsbit
0	0	1
0	1	0
1	0	0
1	1	0

Auch das verbliebene XOR-Gatter lässt sich mit dem NOT-Gatter kombinieren, wodurch sich das XNOR-Gatter (s. Abb. 2.9) mit folgender Wahrheitstabelle ergibt [17].

Tabelle 2.7.: Die Wahrheitstabelle des XNOR-Gatters

Eingangsbit 1	Eingangsbit 0	Ausgangsbit
0	0	1
0	1	0
1	0	0
1	1	1

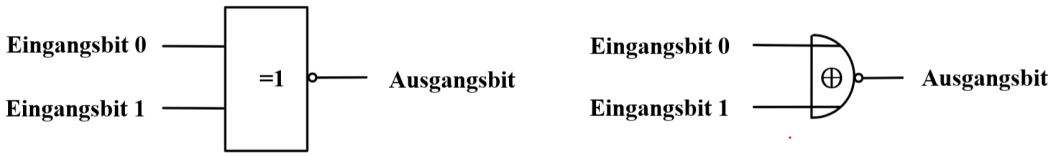


Abbildung 2.9.: Zwei mögliche Schaltsymbole für das XNOR-Gatter. Charakteristisch ist das „= 1“ innerhalb und der kleine Kreis, der die Invertierung darstellt, am Ausgang des Gattersymbols.

Grundsätzlich können die mit zwei Eingängen vorgestellten Logikgatter mit mehr Eingänge konstruiert werden [17]. Die zugehörige Wahrheitstabelle wird entsprechend erweitert [17]. Die Regeln für das Ausgangsbit sind dabei analog zu den Regeln bei der Verwendung von zwei Eingangsbits [17]. Lediglich das XOR-Gatter stellt eine Abweichung dar [17]. Es gilt zu beachten, dass das Ausgangsbit 1 ist, wenn eine ungeraden Anzahl an Eingangsbits 1 ist (s. Tab. 2.8) [17]. Dementsprechend gleichen sich zwei Eingangsbits, die sich in Zustand 1 befinden, stets aus. Das Ausgangsbit des XNOR-Gatters ergibt sich analog mit den bestehenden Regeln.

Tabelle 2.8.: Die Wahrheitstabelle eines XOR-Gatters mit drei Eingangsbits

Eingangsbit 2	Eingangsbit 1	Eingangsbit 0	Ausgangsbit
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

2.3.2 | Quantengatter

Das quantenmechanische Analogon zu den klassischen Gattern sind die Quantengatter [5]. Im Gegensatz zu klassischen Gattern, die auf Bits operieren, arbeiten Quantengatter mit den in Abschnitt 2.1.2 eingeführten Qubits [5]. Quantengatter sind dabei keine elektronischen Bausteine wie die Logikgatter, sondern stehen für eine zeitlich kontrollierbare Interaktion von Qubits mit der Umgebung oder zwischen den Qubits [12]. Diese Interaktion kann beispielsweise durch Messungen erzeugt werden [12]. Mathematisch können Quantengatter mit einem unitären Operator U beschrieben werden [5], der auf

ein Qubit wirkt, folglich mathematisch auf einen Vektor eines Hilbertraums $\mathcal{H}$ [15]. Es gelten die folgenden Eigenschaften für U [5, 15]:

$$U^\dagger U = UU^\dagger = \mathbb{I} \tag{2.7}$$

$$U : \mathcal{H}^{\otimes n} \to \mathcal{H}^{\otimes n} \tag{2.8}$$

Die erste Eigenschaft 2.7 drückt die Unitarität des Operators mathematisch aus. $U^\dagger$ ist dabei der transponierte und komplex konjugierte Operator U, der auch als adjungierter Operator bezeichnet wird [5]. Aus der zweiten Eigenschaft 2.8 lässt sich folgern, dass ein Quantengatter mit n Eingängen auch n Ausgänge aufweist. Dementsprechend lassen sich nicht alle der in Abschnitt 2.3.1 vorgestellten Logikgatter als Quantengatter realisieren oder zumindest nicht direkt. Anstelle von Wahrheitstabellen, die aufgrund der möglichen Superposition von Qubits nicht allumfassend aufgestellt werden können, werden für Quantengatter mit n Qubits unitäre $2^n \times 2^n$ Matrizen formuliert [5]. Allgemein lassen sich unitäre Matrizen wie folgt formulieren, für die natürlicherweise ebenso die Eigenschaft 2.7 gilt [5]:

$$U^\dagger U = \begin{pmatrix} u_{11}^* & u_{21}^* \\ u_{12}^* & u_{22}^* \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbb{I}$$

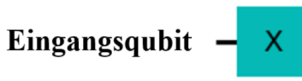
Dabei weist jedes Quantengatter die Eigenschaft 2.7 auf und ist somit unitär [5]. Umgekehrt gilt, dass jede unitäre Matrix ein Quantengatter repräsentiert [5]. Im weiteren Verlauf widmet sich dieser Abschnitt zunächst Ein-Qubit-Quantengattern mit $n = 1$, die ebenso als unäre Quantengatter bezeichnet werden, und nachfolgend Mehr-Qubit-Gattern.

Ein-Qubit-Gatter

Es gibt nur ein nicht triviales Logikgatter, das auf einem Bit operiert: das NOT-Gatter (s. Abb. 2.3). Im Gegensatz dazu können mehrere Quantengatter konstruiert werden, die nur mit einem Qubit arbeiten [5]. Ein wichtiges Ein-Qubit-Gatter ist das X-Gatter (s. Abb. 2.10) [12]. Das X-Gatter wird auch als Pauli-X-Gatter oder Q-NOT-Gatter bezeichnet [15]. Der letztgenannte Name deutet schon auf die Wirkung des Gatters hin. Es ist die quantenmechanische Realisierung des klassischen NOT-Gatters [15]. Dementsprechend invertiert das X-Gatter das Eingangsqubit. So wird aus einem Qubit in Zustand $|0\rangle$ ein Qubit in Zustand $|1\rangle$ und umgekehrt [15]. Die Anwendung des Quantengatters lässt sich als Rechnung darstellen. Dabei wird das Pauli-X-Gatter mathematisch mit der Pauli-Matrix σ_x bzw. σ_1 beschrieben, die nach Wolfgang Pauli benannt ist [9]. Um den Zustand nach der Anwendung des Gatters zu erhalten, ist somit die Anwendung der Matrix auf den Zustandsvektor zu berechnen [15]. Für X-Gatter wird in Rechnungen ein X als Formelsymbol verwendet [5]. Die Rechnung für die Basiszustände ist im Folgenden ausgeführt:

$$X|0\rangle = \sigma_1|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle, \tag{2.9}$$

$$X|1\rangle = \sigma_1|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \tag{2.10}$$



$$X = \sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Abbildung 2.10.: Ein verwendetes Schaltsymbol für das X-Gatter und die dazugehörige Matrix. Charakteristisch ist das „X“ innerhalb des Gattersymbols.

Befindet sich das Eingangsqubit nicht in einem der beiden Basiszustände, sondern in einer Superposition, so werden im Prinzip die Faktoren vor den Basiszuständen vertauscht, bzw. die Basiszustände werden getauscht (s. Gl. 2.11).

$$X|\Psi\rangle = \alpha\, X|0\rangle + \beta\, X|1\rangle = \alpha\,|1\rangle + \beta\,|0\rangle \tag{2.11}$$

Bei genauerer Überlegung werden die Vorfaktoren auch getauscht, wenn sich das Eingangsqubit in einem der beiden Basiszustände befindet. Verdeutlicht wird dies anhand von Gl. 2.12.

$$X|0\rangle = 1\, X|0\rangle + 0\, X|1\rangle = 1\,|1\rangle + 0\,|0\rangle = |1\rangle \tag{2.12}$$

Bildlich gesprochen bewirkt das X-Gatter eine 180°-Drehung um die X-Achse [7]. Darüber hinaus gilt, dass das X-Gatter linear wirkt, wobei dies eine grundlegende Eigenschaft quantenmechanischer Prozesse ist [5]. Das X-Gatter ist ein zentrales Quantengatter und wird in vielen Quantenschaltkreisen verwendet, darunter Schaltkreise, die Teleportation ermöglichen [5].

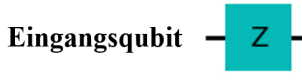
Ein weiteres wichtiges Ein-Qubit-Gatter, welches ebenfalls Teil der Pauli-Gatter ist, ist das Z-Gatter (s. Abb. 2.11) [5]. Demnach wird das Pauli-Z-Gatter mathematisch von der Pauli-Z-Matrix $\sigma_z = \sigma_3$ dargestellt [15]. Das Z-Gatter wird auch als Phasengatter bezeichnet, weil es eine Phasenverschiebung des Qubits um π bewirkt, indem es das Vorzeichen vor dem Basiszustand $|1\rangle$ ändert [5]. Die folgende Rechnung zeigt dies.

$$Z|\Psi\rangle = \sigma_3|\Psi\rangle = \alpha\sigma_3|0\rangle + \beta\sigma_3|1\rangle \tag{2.13}$$

$$= \alpha \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.14}$$

$$= \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ -1 \end{pmatrix} \tag{2.15}$$

$$= \alpha|0\rangle - \beta|1\rangle \tag{2.16}$$



$$Z = \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Abbildung 2.11.: Ein verwendetes Schaltsymbol für das Z-Gatter und die dazugehörige Matrix. Charakteristisch ist das „Z“ innerhalb des Gattersymbols.

Anschaulich bewirkt das Z-Gatter eine Drehung von 180° um die Z-Achse, daher ist bei der Betrachtung der Bloch-Kugel keine Wirkung zu sehen, wenn sich das Qubit in einem der beiden Basiszustände $|0\rangle$ oder $|1\rangle$ befindet und das Z-Gatter angewendet wird [7].

Die Pauli-Gatter werden mit dem Y-Gatter komplettiert, welches eine weniger zentrale Rolle im Vergleich zu den anderen beiden Pauli-Gattern inne hat und daher als letztes vorgestellt wird [5]. Analog zu dem X- und dem Z-Gatter wird das Pauli-Y-Gatter mathematisch von der Pauli-Matrix $\sigma_y = \sigma_2$ ausgedrückt (s. Abb. 2.12) [15]. Wenn die Bloch-Kugel betrachtet wird, bewirkt das Y-Gatter eine 180°-Drehung um die Y-Achse, demnach führt es simultan zu einer Phasenverschiebung um π und invertiert die Amplitude, was mit der folgenden Rechnung ausgedrückt wird [15].

$$Y|\Psi\rangle = \sigma_2|\Psi\rangle = \alpha\sigma_2|0\rangle + \beta\sigma_2|1\rangle \tag{2.17}$$

$$= \alpha \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.18}$$

$$= \alpha \begin{pmatrix} 0 \\ i \end{pmatrix} + \beta \begin{pmatrix} -i \\ 0 \end{pmatrix} \tag{2.19}$$

$$= i\alpha|1\rangle - i\beta|0\rangle \tag{2.20}$$

Abbildung 2.12.: Ein verwendetes Schaltsymbol für das Y-Gatter und die dazugehörige Matrix. Charakteristisch ist das „Y“ innerhalb des Gattersymbols.

Das hier final vorgestellte Ein-Qubit-Gatter ist das Hadamard-Gatter (s. Abb. 2.13) [5]. Das Gatter heißt wegen der Hadamard-Matrix (s. Abb. 2.13), durch die es mathematisch ausgedrückt wird, Hadamard-Gatter [4]. Die Hadamard-Matrix ist nach dem französischen Mathematiker Jacques Hadamard (1865-1963) benannt [4]. Grundlage der Hadamard-Matrix sind die beiden Pauli-Matrizen σ_1 und σ_3, weshalb die Funktion des Hadamard-Gatters auch auf der Grundlage einer Kombination des X- und des Z-Gatters realisiert werden kann [15]. Mithilfe des Quantengatters kann ein Qubit in einen Superpositionszustand versetzt werden [5]. Die nachfolgende Rechnung veranschaulicht dies [7]:

$$H|0\rangle = \frac{\sigma_1 + \sigma_3}{\sqrt{2}}|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} \tag{2.21}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\end{pmatrix} + \frac{1}{\sqrt{2}}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{\sqrt{2}}\Big(|0\rangle + |1\rangle\Big) |+\rangle \tag{2.22}$$

$$H|1\rangle = \frac{\sigma_x + \sigma_z}{\sqrt{2}}|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} \tag{2.23}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\end{pmatrix} + \frac{1}{\sqrt{2}}\begin{pmatrix}0\\-1\end{pmatrix} = \frac{1}{\sqrt{2}}\Big(|0\rangle - |1\rangle\Big) = |-\rangle \tag{2.24}$$

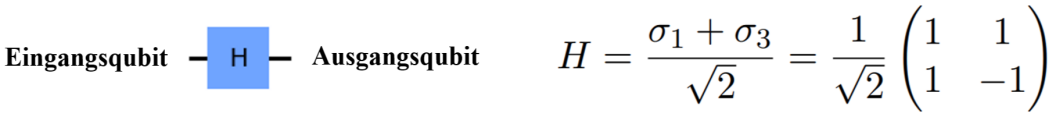


Abbildung 2.13.: Ein verwendetes Schaltsymbol für das Hadamard-Gatter und die dazugehörige Matrix. Charakteristisch ist das „H“ innerhalb des Gattersymbols.

Von einem der Basiszustände ausgehend befindet sich das Qubit nach Anwendung des Hadamard-Gatters bei Betrachtung der Bloch-Kugel somit genau zwischen den Basiszuständen, zwischen den Polen (s. Abb 2.14). Der Zustand ergibt sich aus einer Linearkombination der Basiszustände.

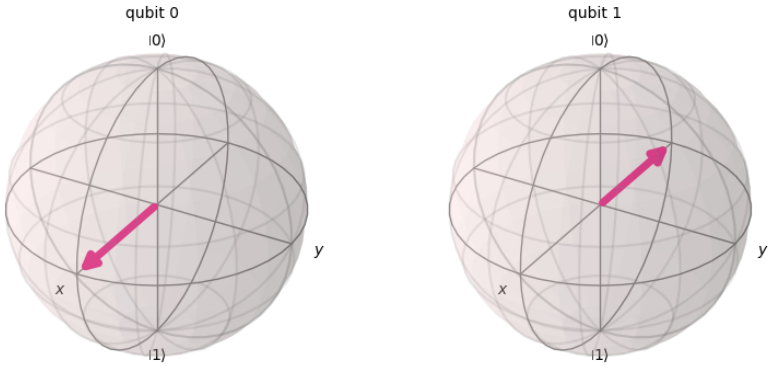


Abbildung 2.14.: Die Bloch-Kugel auf der linken Seite zeigt die Wirkung des Hadamard-Gatters auf ein Qubit in Zustand $|0\rangle$. Auf der rechten Seite befindet sich das Qubit zu Beginn in Zustand $|1\rangle$. Dies entspricht genau den Zuständen $|+\rangle$ und $|-\rangle$, die in Abschnitt 2.2 zur Einführung der Bloch-Kugel verwendet wurden (s. Abb. 2.2).

Es liegt nun die Vermutung nahe, dass das Qubit von einem Basiszustand in den anderen übergeht, wenn das Hadamard-Gatter zweimal direkt hintereinander angewendet wird [5]. Dies ist jedoch nicht der Fall, da eine zweifache Anwendung des Hadamard-Gatters den Zustand des Qubits nicht verändert, weil für das Hadamard-Gatter gilt: $H^2 = I$, wobei I die Identitätsmatrix ist [5]. Die Identitätsmatrix hat keinen Effekt auf

den Zustand eines Qubits [5]. Das heißt, dass das Hadamard-Gatter reversibel ist, was eine grundlegende Eigenschaft von Quantengattern ist.

$$H^2 = H \cdot H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I \tag{2.25}$$

Anwenden der Identitätsmatrix auf $|0\rangle$:

$$I|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \tag{2.26}$$

Anwenden der Identitätsmatrix auf ein Qubit in einem Superpositionszustand:

$$I|\Psi\rangle = \alpha I|0\rangle + \beta I|1\rangle = \alpha \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.27}$$

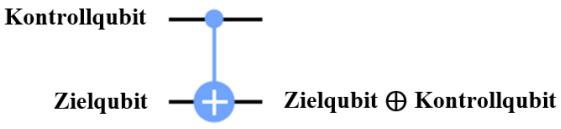
$$= \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \alpha|0\rangle + \beta|1\rangle \tag{2.28}$$

Zusammenfassend lässt sich sagen, dass das Hadamard-Gatter ein zentrales Quantengatter ist, weil es ein Qubit in einen Superpositionszustand versetzen kann. Daher wird es häufig verwendet, um Qubits für weitere Berechnungen vorzubereiten [5].

Mehr-Qubit-Gatter

Neben den Ein-Qubit-Gattern werden zudem Mehr-Qubit-Gatter realisiert, mit denen Funktionen von Logikgattern mit mehr als einem Eingangsbit quantenmechanisch umgesetzt werden [5]. Ein fundamentales Quantengatter, welches mit zwei Eingangsqubits arbeitet, ist das CNOT-Gatter (s. Abb. 2.15) [5]. CNOT steht für „controlled NOT-Gatter“, was die Funktionsweise des Quantengatters treffend beschreibt [9]. Ein Eingangsqubit fungiert als Kontrollqubit, das andere als Zielqubit, das heißt, darin soll das Ergebnis der Operation gespeichert werden [9]. Das Zielqubit wird invertiert, wenn sich das Kontrollqubit in Zustand $|1\rangle$ befindet [9]. Ist das Kontrollqubit in Zustand $|0\rangle$, so bleibt der Eingangszustand des Zielqubits unverändert [5]. Das Kontrollqubit determiniert somit die Operation auf das Zielqubit. Diese Funktion kann als Verallgemeinerung des klassischen XOR-Gatters aufgefasst werden, weil die XOR-Operation zwischen dem Kontrollqubit und dem Zielqubit vollzogen und das Ergebnis an das Zielqubit übergeben wird [5]. Dies lässt sich mathematisch wie folgt ausdrücken, wobei $\oplus$ für die Addition modulo 2 steht: $|Kontrollqubit, Zielqubit\rangle \rightarrow |Kontrollqubit, Zielqubit \oplus Kontrollqubit\rangle$ [5]. Aufgrund der Funktion des CNOT-Gatters lässt sich mit diesem Quantengatter und den Ein-Qubit-Gattern jedes beliebige Mehr-Qubit-Gatter realisieren [5]. Die Kombination dieser Quantengatter ist folglich universell [5]. Dementsprechend wird das CNOT-Gatter in vielen Quantenalgorithmen verwendet [9] und dient als wichtiges Werkzeug, um zwei Qubits zu verschränken [7].

Das CNOT-Gatter kann demnach als quantenmechanisches Analogon zu dem verallgemeinerten XOR-Gatter aufgefasst werden. Die anderen vorgestellten Logikgatter mit mehr als einem Eingangsbit lassen sich nicht direkt übertragen, weil sie nicht unitär sind [5]. Das bedeutet, sie sind nicht reversibel, nicht invertierbar und somit nicht umkehrbar [5]. Dies lässt sich anhand eines Beispiels veranschaulichen. Wenn sich das Ausgangsbit des NAND-Gatters in Zustand 1 befindet, sind drei Kombinationen der Eingangsbits möglich (s. Tab. 2.5). Von dem Ausgangsbit lässt sich nicht eindeutig auf die Eingangsbits schließen, wodurch eine Umkehrung der Operation nicht grundsätzlich möglich ist [5]. Jedoch ist diese Umkehrbarkeit eine zentrale Eigenschaft unitärer Quantengatter, was sich in der Verwendung unitärer Matrizen als mathematische Formulierung der Quantengatter widerspiegelt [5]. Allerdings gibt es trotzdem die Möglichkeit, die Funktionen des AND- und OR-Gatters quantenmechanisch umzusetzen [4]. Dafür wird das Toffoli-Gatter verwendet [4].



$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Abbildung 2.15.: Ein verwendetes Schaltsymbol für das CNOT-Gatter und die dazugehörige Matrix.

Ursprünglich wurde das Toffoli-Gatter bereits 1967 von Carl Adam Petri auf Deutsch in einem Tagungsband publiziert, welcher jedoch wenig Aufmerksamkeit erhalten hat [4]. Daher wurde das Konzept des Toffoli-Gatters 1981 erneut von Tommaso Toffoli publiziert, weshalb der Name des Quantengatters auf ihn zurückgeht [4]. Das Toffoli-Gatter ist ein Quantengatter mit drei Eingangsqubits, wobei es sich bei zwei Qubits um Kontrollqubits handelt (s. Abb. 2.16) [5]. Demnach wird nur der Zustand des dritten Qubits, des Zielqubits, von dem Quantengatter verändert [5]. Für diesen Abschnitt wird das Toffoli-Gatter explizit dahingehend betrachtet, klassische Schaltkreise als Quantenschaltkreise zu realisieren, weshalb Eingangsqubits, die sich in Superposition befinden, nicht beleuchtet werden. Die Matrix des Toffoli-Gatters lautet wie folgt [5]:

$$Toffoli = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

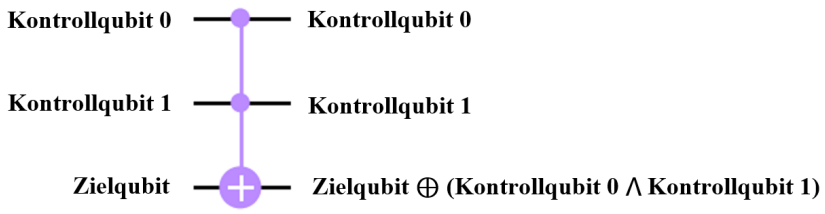


Abbildung 2.16.: Allgemeine Darstellung des Toffoli-Gatters.

Mit dem Toffoli-Gatter lässt sich unter anderem das klassische NOT-Gatter realisieren [5]. Dabei wird das Zielqubit invertiert, wenn sich beide Kontrollqubits im Zustand $|1\rangle$ befinden [5]. Mathematisch lässt sich das wie folgt formulieren [4]:

$$Zielqubit \oplus (Kontrollqubit\,0 \wedge Kontrollqubit\,1) = Zielqubit \oplus (1 \wedge 1) = \overline{Zielqubit}.$$

Die Funktion der Invertierung lässt sich mit dem Toffoli-Gatter immer herbeiführen, wenn die beiden Kontrollqubits künstlich in den Zustand $|1\rangle$ versetzt werden [4]. Wenn das zum Zielqubit gehörige Eingangsqubit wiederum konstant in den Zustand $|0\rangle$ versetzt wird, kann mit dem Toffoli-Gatter die Funktion eines AND-Gatters umgesetzt werden [4]. Dies ergibt sich aus dem logischen Ausdruck [4]:

$$\begin{aligned} &Zielqubit \oplus (Kontrollqubit\,0 \wedge Kontrollqubit\,1) \\ &= 0 \oplus (Kontrollqubit\,0 \wedge Kontrollqubit\,1) \\ &= (Kontrollqubit\,0 \wedge Kontrollqubit\,1). \end{aligned}$$

Beide Konfigurationen des Toffoli-Gatters sind in Abbildung 2.17 dargestellt.

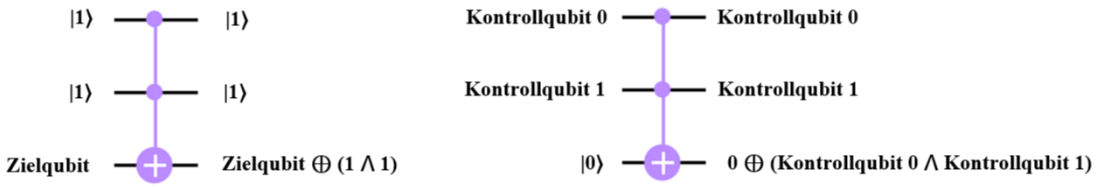


Abbildung 2.17.: Auf der linken Seite ist die Konfiguration zur Realisierung der NOT-Funktion zu sehen. Die rechte Seite zeigt die AND-Funktion.

Wird dahingegen das Zielqubit zu Beginn in den Zustand $|1\rangle$ versetzt, erfüllt das Toffoli-Gatter die Funktion eines NAND-Gatters (s. Abb. 2.19) [5]:

$$\begin{aligned} &Zielqubit \oplus (Kontrollqubit\,0 \wedge Kontrollqubit\,1) \\ &= 1 \oplus (Kontrollqubit\,0 \wedge Kontrollqubit\,1) \\ &= \overline{(Kontrollqubit\,0 \wedge Kontrollqubit\,1)}. \end{aligned}$$

Auch das klassische OR-Gatter lässt sich mit dem Toffoli-Gatter realisieren [4]. Dafür ist die Regel von De Morgan nötig, die auf diesen Fall angewendet besagt, dass ein OR-Gatter äquivalent zu einem NAND-Gatter mit invertierten Eingangsqubits ist [17]. Demzufolge gilt [4]: $Eingangsqubit\,0 \vee Eingangsqubit\,1 = \overline{(\overline{Eingangsqubit\,0} \wedge \overline{Eingangsqubit\,1})}$. Die Realisierung erfolgt mit insgesamt drei Toffoli-Gattern, von denen eines die Funktion des NAND-Gatters und zwei die Funktion der NOT-Gatter erfüllen (s. Abb. 2.18) [4].

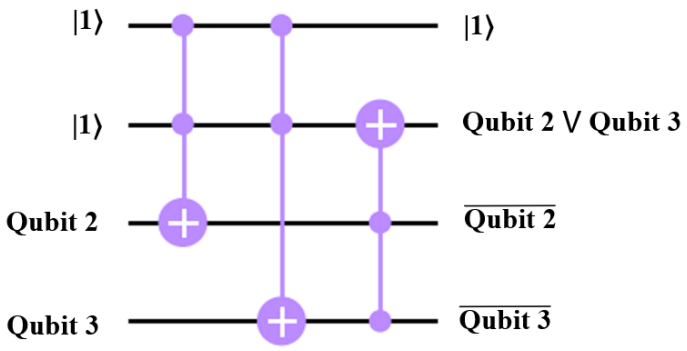


Abbildung 2.18.: Umsetzung des OR-Gatters mit drei Toffoli-Gattern.

Eine weitere Operation, die sich mit dem Toffoli-Gatter realisieren lässt ist die FANOUT-Operation [5]. Dabei wird das zweite Kontrollqubit als relevantes Eingangsqubit aufgefasst und unverändert an das Zielqubit übergeben [5]. Der gewünschte Output liegt somit bei dem Kontrollqubit 1 und dem Zielqubit an [5]. Das Kontrollqubit 0 und der Eingang des Zielqubits wird jeweils in einem der Basiszustände präpariert [5]. Basierend auf der NAND- und der FANOUT-Funktion kann jeder klassische Schaltkreis mit Toffoli-Gattern quantenmechanisch simuliert werden [5]. So können beispielsweise klassische Algorithmen simuliert werden, die zufällige Eingangswerte aufweisen [5]. Diese zufälligen Eingangswerte lassen sich mit Hadamard-Gattern präparieren, womit erneut die Möglichkeit der Superposition des Zustands eines Qubits eine zentrale Rolle spielt [5].

Abbildung 2.19.: Auf der linken Seite ist die Konfiguration zur Realisierung der NAND-Funktion zu sehen. Die rechte Seite zeigt die FANOUT-Funktion.

Quantengatter sind grundlegende Komponenten von Quantencomputern. Sie ermöglichen die Durchführung von Operationen auf Qubits, um den Zustand dieser zu verändern. Eine Hauptcharakteristik von Quantengattern ist, dass sie umkehrbar sind, was sich in ihrer mathematischen Definition durch unitäre Matrizen äußert. Der folgende Abschnitt 2.4 widmet sich dem Aspekt, wie der Zustand eines Qubits während oder am Ende einer Berechnung bestimmt bzw. gemessen werden kann.

2.4 | Messungen in der Quantenmechanik

Messungen sind ein wesentlicher Aspekt von Quantencomputern, weil sie die Ergebnisse von Berechnungen eines Quantenschaltkreises liefern [13]. Jedoch unterscheiden sich Messungen in der Quantenmechanik grundlegend von Messungen in der klassischen Physik, was vor allem auf unterschiedliche Eigenschaften von Bits und Qubits zurückzuführen ist [9]. Bei einer Berechnung mit einem klassischen Computer steht der finale Zustand der Bits unmittelbar vor der Messung bereits fest [9]. Der Zustand ist zu jedem Zeitpunkt der Berechnung eindeutig definiert, was als Realismus bezeichnet wird [9]. Zudem ändert sich gemäß des Prinzips der Lokalität bei der Anwendung eines Logikgatters lediglich der Zustand der Bits, auf die es direkt angewendet wird [9]. Für Qubits gelten die beiden Prinzipien des Realismus und der Lokalität nicht, das heißt, der Zustand eines Qubits steht nicht bereits vor dem Messvorgang fest und die Anwendung eines Quantengatters kann dazu führen, dass nicht nur der Zustand des Qubits, auf das das Quantengatter direkt angewendet wird, verändert wird, sondern auch andere Qubits beeinflusst werden [9]. Zurückzuführen ist dies auf die quantenmechanischen Eigenschaften der Superposition und der Verschränkung [9]. Die möglichen Schaltsymbole für einen Messvorgang in einem Quantenschaltkreis sind in Abbildung 2.20 dargestellt [4, 5].

Abbildung 2.20.: Verwendete Schaltsymbole für Messungen in Quantenschaltkreisen.

Messungen werden in der Quantenmechanik mit Messoperatoren der Form $\{M_m\}$ formuliert, wobei der Index für den Zustand steht, der gemessen wird [5]. Die Messoperatoren sind lineare Abbildungen $\mathcal{H} \to \mathcal{H}$ [15], die auch Observablen genannt werden und in der Quantenmechanik immer hermitesch sind, das heißt, es gilt $M^\dagger = M$ [5]. Um ein Qubit auf den Zustand $|0\rangle$ hin zu untersuchen, lässt sich demnach der Messoperator

$$M_{|0\rangle} = |0\rangle\langle 0| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

formulieren, wobei $M^2_{|0\rangle} = M_{|0\rangle}$ gilt [5]. Mit diesem Messoperator erfolgt die Messung somit in der Basis $\{|0\rangle, |1\rangle\}$ [5]. Wird beispielsweise ein Qubit, das sich in dem Zustand $|0\rangle$ befindet, mit dem Messoperator $M_{|0\rangle}$ gemessen, ergibt sich das Ergebnis 1 [5]. Für ein Qubit in Zustand $|1\rangle$ ergibt sich 0 [5]. Die Messergebnisse 1 und 0 stehen dabei

für die Wahrscheinlichkeit, mit der der untersuchte Zustand in dem gesuchten Zustand gemessen wird [5]. Wenn sich das Qubit eindeutig in Zustand $|1\rangle$ befindet, kann es mit einer Wahrscheinlichkeit von 0 % in Zustand $|0\rangle$ gemessen werden. Grundsätzlich ist die Quantenmechanik probabilistisch, die von ihr gelieferten Vorhersagen sind dementsprechend Wahrscheinlichkeiten [5]. Wenn ein Qubit in Superposition betrachtet wird ergibt sich mit $|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ allgemein [5]:

$$p(|0\rangle) = \langle\psi|M^{\dagger}_{|0\rangle}M_{|0\rangle}|\psi\rangle = \langle\psi|M^{2}_{|0\rangle}|\psi\rangle = \langle\psi|M_{|0\rangle}|\psi\rangle = |\alpha|^2.$$

Ein Qubit, welches sich in Superposition befindet, wird somit mit einer Wahrscheinlichkeit von $|\alpha|^2$ in Zustand $|0\rangle$ gemessen. α und β können folglich als Gewichtungen für die entsprechenden Basiszustände aufgefasst werden [5]. Aus denen geht hervor, mit welcher Wahrscheinlichkeit das Qubit in dem nachfolgenden Basiszustand gemessen wird. Die Ergebnisse sind daher wie auch bei anderen Zufallsexperimenten (Münzwurf, Würfelwurf, ...) statistisch verteilt [5]. Nach dem Messvorgang befindet sich das Qubit in dem Zustand [5]:

$$\frac{M_{|0\rangle}|\psi\rangle}{|\alpha|} = \frac{a}{|a|}|0\rangle,$$

wobei der Faktor $\frac{a}{|a|}$ nicht betrachtet wird und sich somit der Zustand $|0\rangle$ ergibt [5]. Die Messung sorgt demnach dafür, dass sich das Qubit nicht mehr in einem Superpositionszustand, sondern eindeutig in einem der Basiszustände befindet [5]. Welcher der Basiszustände gemessen wird, kann nur in Form von Wahrscheinlichkeiten vorhergesagt werden und ist folglich nicht im Voraus determiniert [5]. Messvorgänge können daher den Zustand von Qubits verändern [9]. Daher ist ein Rückschluss auf den Zustand des Qubits vor der Messung nicht möglich. Es können - falls umsetzbar - lediglich viele Messungen durchgeführt werden, um Wahrscheinlichkeiten für die Basiszustände des Qubits anzugeben. Dadurch lässt sich ein unbekanntes Qubit nicht vollständig kopieren, was ausführlicher von dem „no-cloning Theorem“ erfasst wird [5].

Konkret ergibt sich für den Superpositionszustand eines Qubits, der mithilfe des Hadamard-Gatters herbeigeführt werden kann (s. ab Gl. 2.21):

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$
$$|\alpha|^2 = |\beta|^2 = \frac{1}{2} = 50\%.$$

Das Qubit wird mit einer 50-prozentigen Wahrscheinlichkeit entweder in Zustand $|0\rangle$ oder $|1\rangle$ gemessen. Dieses mit dem Hadamard-Gatter präparierte Qubit gleicht somit einem Münzwurf, bei dem etwa die Hälfte der Würfe Kopf und die andere Hälfte Zahl zeigt. Anhand dieses Beispiels lässt sich eine wichtige Bedingung für die berechneten Wahrscheinlichkeiten für das Auftreten der einzelnen Basiszustände ableiten: Sie müssen aufsummiert immer 1 bzw. 100 % ergeben [5]:

$$\sum_m p(m) = \sum_m \langle\psi|M^{\dagger}_m M_m|\psi\rangle = 1. \tag{2.29}$$

Daher gilt:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.30}$$

Die Vorhersagen der Quantenmechanik sind insbesondere aufgrund der Superposition Wahrscheinlichkeiten. Daher sind die Ergebnisse wie auch bei anderen Zufallsexperimenten (Münzwurf, Würfelwurf, ...) statistisch verteilt.

2.5 | Anwendungen von Quantencomputern und Quantenalgorithmen

Wie bereits in der Einleitung angedeutet, werden Quantencomputer vor allem in der Zukunft in vielen verschiedenen Bereichen eine zentrale Rolle spielen [20]. Insbesondere die Eigenschaft der Superposition ermöglicht es, eine größere Menge an Information zu verarbeiten und parallele Berechnungen durchzuführen [5]. Dadurch lassen sich beispielsweise Optimierungsprobleme in vielen verschiedenen Bereichen lösen oder vielfältige Simulationen durchführen [5, 20]. In diesem Abschnitt werden aktuelle und mögliche zukünftige Anwendungsgebiete von Quantencomputern kurz umrissen.

Die Kryptographie und Datensicherheit sind ein Gebiet, welches von Quantencomputern und Quantenalgorithmen voraussichtlich stark beeinflusst werden wird [5]. Darunter fällt vor allem der Schlüsselaustausch, der für Kryptographie erforderlich ist [4, 7]. Mithilfe der Quantenmechanik ist es möglich, diesen Austauschprozess abhörsicher zu gestalten [9]. Ein Beispiel dafür ist das BB84-Protokoll, welches 1984 von Charles Bennet und Gilles Brassard veröffentlicht wurde [4, 21]. Kurz skizziert, basiert das Verfahren auf der Verwendung von einer Reihe polarisierter Photonen, die jeweils von der sendenden und der empfangenden Person gemessen werden [7]. Die Messung der einzelnen Photonen erfolgt in einer zufälligen Basis, wobei zwei Basen möglich sind [7]. Die Messergebnisse müssen bei gleicher Basis identisch sein, somit können diese als Schlüssel verwendet und bei Abweichungen auf einen Angriff von außen geschlossen werden [7]. Eine weitere Methode beruht auf der Eigenschaft der Verschränkung und wurde 1991 von Artur Ekert publiziert [22]. Im Gegensatz zu der Entwicklung neuer Verschlüsselungsmethoden sollen Quantencomputer in Zukunft in der Lage sein, herkömmliche Verschlüsselungsverfahren zu entschlüsseln [12]. Dabei ist besonders der Shor-Algorithmus von Bedeutung, der 1994 von Peter Shor veröffentlicht wurde [23]. Ein herkömmliches Verschlüsselungsverfahren ist das RSA-Verfahren, welches auf der Schwierigkeit der Faktorisierung großer Zahlen beruht [4]. Mit dem Shor-Algorithmus, der Eigenschaften von Quantencomputern nutzt, ist es möglich, diese Faktorisierung auf effektive Weise in polynomialer Zeit durchzuführen [4]. Jedoch wurde bislang noch kein Quantencomputer realisiert, der den Shor-Algorithmus für praktisch relevante Größenordnungen ausführen konnte [4]. Somit bietet die Quantenmechanik Chancen und Herausforderungen für die Kryptographie.

Auch im Bereich der Simulation haben Quantencomputer vielversprechende Anwendungen [24–26]. Zum einen können quantenmechanische Phänomene simuliert werden [24], was grundsätzlich eine Herausforderung darstellt [27]. Zum anderen können durch chemische Modellierungen neue Medikamente und Materialien designt werden [25, 26].

Das maschinelle Lernen ist ein weiteres Gebiet, auf welchem Quantencomputer eine zentrale Rolle spielen und spielen können [28]. Auf Basis ihrer Fähigkeit große Datenmengen parallel verarbeiten zu können, lassen sich Quantencomputer zur Entwicklung neuer Algorithmen für Datenauswertung und Mustererkennung einsetzen [28]. Hinzu kommt, dass Quantencomputer bestehende Algorithmen oder Prozesse optimieren können [28, 29], darunter insbesondere kombinatorische Problemstellungen [29].

3 Qiskit

Sowohl klassische Computer als auch Quantencomputer bilden mit Programmen, die sich auf dem jeweiligen Gerät befinden, eine untrennbare Einheit [5, 30]. Die physischen Komponenten eines Computers stellen die Hardware dieser Verbindung dar [30]. Programme sind Codierungen von Algorithmen, die innerhalb einer finiten Anzahl von Schritten Eingangsgrößen eindeutig in Ausgangsgrößen überführen [30]. Sie zählen daher zu der Seite der Software und steuern die Hardware an [30]. Die Schritte der Algorithmen werden auf der Hardware ausgeführt, die unter anderem die Daten speichert und mit der Gatter realisiert werden [4, 30]. Algorithmen sind somit ein zentraler Bestandteil der Informatik und der Quanteninformatik, der eingesetzt wird, um Problemstellungen zu lösen [4]. Beispiele für bekannte Quantenalgorithmen sind der bereits angesprochene Shor-Algorithmus [23], der Deutsch-Jozsa Algorithmus [31] und der Grover Algorithmus, der für das Durchsuchen von Datenbanken verwendet wird [32, 33].

Algorithmen werden, wie bereits erwähnt, als Programme codiert, das heißt in einer Programmiersprache formuliert [30]. Quantenalgorithmen können unter anderem in der Programmiersprache Q#, gesprochen „Q-Sharp“, oder in Python programmiert werden [34]. Q# wurde eigens für Quantencomputer entwickelt und ist Bestandteil des Quantum Development Kits von Microsoft [34]. Python hingegen ist eine weit verbreitete und vielseitige Programmiersprache, die im Kontext der Physik besonders auf den Gebieten der Simulation und Visualisierung Anwendung findet [35]. Auch für Python wurden Bibliotheken erstellt, um Quantencomputer zu programmieren [36]. Zum einen ist Cirq zu nennen, welches ein open-scource Framework von Google ist, mit dem Quantenschaltkreise konstruiert, optimiert und verändert werden können [36]. Insbesondere „noisy intermediate-scale“ Quantencomputer stehen bei Cirq im Vordergrund [36]. Zum anderen wurde mit Python ein Software Development Kit, kurz SDK, für Quantenalgorithmen realisiert, was eine Sammlung von Programmen und Bibliotheken darstellt [36]. Dieses SDK heißt Qiskit und stammt von IBM [36]. Das in dieser Bachelorarbeit behandelte Jupyter Notebook wurde mit ebenjenem Qiskit erstellt, weshalb sich dieses Kapitel zu Beginn Qiskit allgemein widmet (3.1). Daran anschließend wird die Umsetzung und Implementierung der relevanten quantenmechanischen Konzepte aus dem vorangegangenen Kapitel 2 thematisiert (3.2).

3.1 Einführung Qiskit

Qiskit ist ein open-source SDK, welches von IBM entwickelt wurde [37]. Es ist zentraler Bestandteil der im März 2017 veröffentlichten Plattform IBM Quantum Experience [36]. Die Plattform wurde geschaffen, um Quantencomputer einer breiten Öffentlichkeit zugänglich zu machen und diese kommerziell einsetzen zu können [37]. Mithilfe von Qiskit kann mittels Cloud-Zugriff mit realen Quantencomputern von IBM interagiert werden

[37]. Neben der Konstruktion von realen Quantenschaltkreisen ist auch die Simulation von Quantenschaltkreisen auf klassischen Computern möglich [36]. Die konkrete Programmierung kann entweder über ein graphisches, ein command-line oder ein application programming interface erfolgen [36]. Letzteres wird verbreitet verwendet und stellt eine Sammlung von Klassen mit Funktionen dar, um Software zu programmieren und mit der Software zur Simulation und der Hardware von IBM zu interagieren [37]. Das SDK setzt sich hauptsächlich aus vier Bibliotheken zusammen, die jeweils nach einem der vier Elemente in lateinischer Sprache benannt sind: *Terra, Aqua, Aer und Ignis* [37, 38].

Die Bibliothek *Terra* (lat. für Erde [38]) bildet im Prinzip die Basis für Qiskit und erlaubt die Konstruktion, Manipulation und Optimierung von Quantenschaltkreisen [37]. Vor allem die Funktion *QuantumCircuit* ist dafür relevant [37]. Mithilfe von *Aqua* (lat. für Wasser [38]) können Quantenschaltkreise erstellt werden, ohne auf ein großes Vorwissen zurückgreifen zu müssen [37]. Sie liefert eine Oberfläche auf der die Parameter des gewünschten Quantenschaltkreises eingegeben werden und verwendet *Terra*, um diesen Quantenschaltkreis zu realisieren [37]. Für die Simulation von Quantenschaltkreisen auf klassischen Computern, wird die *Aer* Bibliothek (lat. für Luft [38]) verwendet [37]. Komplettiert werden die vier Elemente durch das Feuer, lateinisch *Ignis* [38]. Die entsprechende Bibliothek umfasst alle nötigen Funktionen, um quantenmechanische Experimente durchzuführen, darunter die Hardware zu manipulieren und zu korrigieren und Rauschen auszugleichen [36, 37].

Bis zu dem Jahr 2019 wurden auf den Quantencomputern von IBM bereits mehr als 6.5 Millionen Experimente von mehr als 100 000 Personen durchgeführt [37]. Diese umfangreiche Nutzung hat zu über 100 wissenschaftlichen Arbeiten geführt, welche wichtige Fortschritte und Erkenntnisse auf dem Gebiet der Quantencomputer ermöglicht haben [37].

Qiskit bietet folglich die Möglichkeit, quantenmechanische Experimente durchzuführen und die Grundlagen des Quantumcomputings darzustellen. Dies kann einerseits für kommerzielle Zwecke und andererseits für die Vermittlung dieser Inhalte genutzt werden.

3.2 | Umsetzung der einzelnen Bestandteile

In diesem Abschnitt wird anhand von Beispielen erläutert, wie die Grundsätze des Quantencomputings in Qiskit umgesetzt werden können. Dabei wird zunächst darauf eingegangen, was zu Beginn nötig ist, um einen Quantenschaltkreis zu initialisieren. Darauf aufbauend werden Qubits und Gatter in Qiskit vorgestellt, was in beispielhaften Anwendungen der Messung und einem Experiment mündet. Die Codebeispiele wurden eigenständig in einem Jupyter Notebook erstellt und als Abbildungen eingefügt.

3.2.1 | Initialisierung eines Quantenschaltkreises

Zu Beginn eines jeden Programmcodes müssen die verwendeten Bibliotheken und Module eingebunden und gegebenenfalls installiert werden [39]. Diese Bibliotheken stellen Klassen, Funktionen und Konstanten zur Verfügung, die innerhalb des Programms verwendet werden können [35]. Mit dem Code `from module import *` können alle Klassen, Funktionen und Konstanten eines Moduls auf einmal eingebunden werden [35]. Der Import einer bestimmten Klasse oder Funktion lässt sich mit dem Aufruf des Klassen-

oder Funktionsnamens spezifizieren: `from module import class1, class2` [35]. Die für die nachfolgenden Codebeispiele erforderlichen Bibliotheken können der Abbildung 3.1 entsprechend importiert werden. In Zeile 1 wird Qiskit selbst importiert, die anderen Module werden im Verlauf an gegebener Stelle erklärt.

```
1 from qiskit import *
2 from pylatexenc import *
3 from qiskit.visualization import plot_histogram
```

Abbildung 3.1.: Mit dieser Codezelle werden die für die nachfolgenden Beispiele erforderlichen Bibliotheken, Module und Funktionen eingebunden.

Um einen Quantenschaltkreis zu erstellen, wird die Funktion `QuantumCircuit()` aufgerufen, die als Argument die Anzahl der zum Schaltkreis gehörigen Qubits übernimmt [39]. Ein Quantenschaltkreis mit einem Qubit lässt sich demnach wie folgt erstellen:

```
1 from qiskit import *
2
3 qc = QuantumCircuit(1)
```

Abbildung 3.2.: Definition eines Quantenschaltkreises mit einem Qubit.

Der erstellte Quantenschaltkreis ist in der Variablen `qc` gespeichert und kann mithilfe der Funktion `draw()` ausgegeben werden (s. Abb. 3.3) [36]. Das Argument der Funktion bestimmt die Darstellungsform des ausgegebenen Quantenschaltkreises [19]. In den nachfolgenden Codebeispielen wird *mpl* als als Darstellung verwendet [19].

```
1 from qiskit import *
2 from pylatexenc import *
3
4 qc = QuantumCircuit(1)
5 qc.draw('mpl')
```


Abbildung 3.3.: Definition und Darstellung eines Quantenschaltkreises mit einem Qubit.

Mit dem bisherigen Code wurde ein Quantenschaltkreis erstellt, der auch ausgegeben werden kann. Jedoch handelt es sich dabei lediglich um eine technische Definition und nicht um ein quantenmechanisches System [39]. Um mit dem Quantenschaltkreis Experimente durchführen und quantenmechanische Fragestellungen untersuchen zu können, muss dieser auf einem Backend simuliert werden [39]. Ein beispielhafter Ablauf ist in Abbildung 3.4 programmiert. Zunächst wird in Zeile 4 mit der Funktion `get_backend()` der Klasse `Aer` der `statevektor_simulator` als Backend ausgewählt und der Variable `backend` übergeben [39]. Das ausgewählte Backend wird in Zeile 5 auf den Quantenschaltkreis angewendet und wiederum in einer Variablen gespeichert [39]. Mithilfe von Zeile 6 und 7 wird der Quantenschaltkreis als Wellenfunktion an die Variable `state` übergeben, welche durch den Code in Zeile 9 ausgegeben wird [39].

```
1 from qiskit import *
2
3 qc = QuantumCircuit(1)
4 backend = Aer.get_backend('statevector_simulator')
5 task = execute(qc, backend=backend, shots=1, memory=True)
6 task_result = task.result()
7 state = task_result.get_statevector(qc)
8
9 state.draw('text', prefix= 'Zustand des Qubits: ')
```

```
Zustand des Qubits: [1.+0.j,0.+0.j]
```

Abbildung 3.4.: Definition eines Quantenschaltkreises mit einem Qubit und Ausgabe des Zustands des Qubits.

Mit den ausgeführten Codezeilen kann somit ein Quantenschaltkreis simuliert werden. Neben der Ausgabe des Zustands als Liste, lässt sich ein Qubit auch mithilfe von Qiskit als Bloch-Kugel darstellen, wie es in Abschnitt 2.2 eingeführt wurde [19]. Dafür ist lediglich in Zeile 9 in Abbildung 3.4 bei der Funktion `draw()` das Argument von `text` zu `bloch` zu ändern (s. Abb. 3.5) [19].

```
1 from qiskit import *
2
3 qc = QuantumCircuit(1)
4 backend = Aer.get_backend('statevector_simulator')
5 task = execute(qc, backend=backend, shots=1, memory=True)
6 task_result = task.result()
7 state = task_result.get_statevector(qc)
8
9 state.draw('bloch')
```

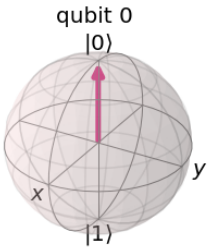


Abbildung 3.5.: Definition eines Quantenschaltkreises mit einem Qubit und Ausgabe des Zustands des Qubits als Bloch-Kugel.

Der Zustandsvektor und die Bloch-Kugel zeigen beide dasselbe Ergebnis: Das Qubit befindet sich zu Beginn und ohne Manipulation des Zustands im Zustand $|0\rangle$ [39]. Es ist gewissermaßen eine Art Ausgangszustand eines Qubits. Ein Quantenschaltkreis mit mehr Qubits lässt sich vollständig analog erstellen [39]. Lediglich das Argument von `QuantumCircuit` muss auf die gewünschte Anzahl an Qubits angepasst werden (s. Abb. 3.6) [39]. Um mehrere Ausgaben aus einer Codezelle in einem Jupyter Notebook simultan auszugeben, kann die Funktion `display` aus dem Modul `IPython.display` verwendet werden [40]. Die gewünschten Ausgaben sind wie in Zeile 10 der Abbildung 3.6 mit einem Komma getrennt als Argument an die Funktion zu übergeben.

```
1  from qiskit import *
2  from pylatexenc import *
3
4  qc = QuantumCircuit(4)
5  backend = Aer.get_backend('statevector_simulator')
6  task = execute(qc, backend=backend, shots=1, memory=True)
7  task_result = task.result()
8  state = task_result.get_statevector(qc)
9
10 display(qc.draw('mpl'), state.draw('bloch'))
```

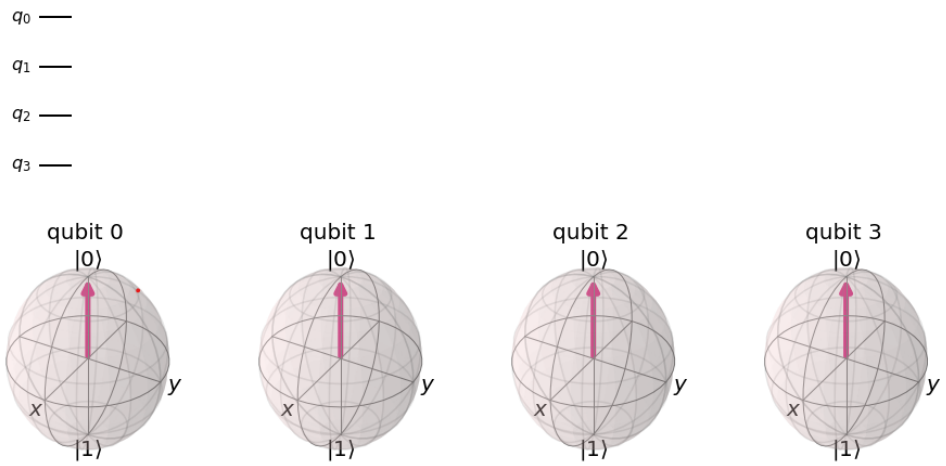


Abbildung 3.6.: Definition und bildliche Darstellung eines Quantenschaltkreises mit vier Qubits und Ausgabe der Zustände der Qubits als Bloch-Kugel.

3.2.2 | Präparation von Qubits

Bisher wurden nur Qubits über die Definition des Quantenschaltkreises erstellt, weshalb sich diese alle in dem Zustand $|0\rangle$ befunden haben. Die Zustände der Qubits können jedoch auch separat definiert werden (s. Abb. 3.7) [41]. Dafür ist die Klasse `initialize` notwendig, die unter anderem den gewünschten Zustand als `String`, `Liste`, `Integer` oder `Statevector` als Parameter übernimmt [41]. Zunächst wird in Zeile 3 aus dem Modul `math` die Funktion `sqrt` importiert, um die Quadratwurzel berechnen zu können. Dies wird in Zeile 5 bei der Definition des Zustands durchgeführt, womit der Zustand $|+\rangle$ definiert und an die Variable `state_plus` übergeben wird. Der Zustand $|+\rangle$ ist bereits aus der Abbildung 2.14 bekannt. Nach der Erstellung eines Quantenschaltkreises mit zwei Qubits in Zeile 7, wird in der darauf folgenden Zeile 8 das erste Qubit, Qubit 0 in den vordefinierten Zustand $|+\rangle$ versetzt. Dazu wird die Funktion `initialize` auf den Quantenschaltkreis angewendet und als Argument der vordefinierte Zustand und das Zielqubit des Zustands übergeben. Die Darstellung des Quantenschaltkreises zeigt die Definition des Qubits in einem lila farbenen Rechteck mit Angabe der Vorfaktoren α und β aus der allgemeinen Definition eines Qubits (s. Gl. 2.5). Anhand der Bloch-Kugeln wird deutlich, dass nur der Zustand des Qubits 0 in den gewünschten Zustand versetzt wird. Qubit 1 befindet sich weiterhin in den Ausgangszustand $|0\rangle$.

```
1  from qiskit import *
2  from pylatexenc import *
3  from math import sqrt
4
5  state_plus = [1/sqrt(2), 1/sqrt(2)]
6
7  qc = QuantumCircuit(2)
8  qc.initialize(state_plus, 0)
9  backend = Aer.get_backend('statevector_simulator')
10 task = execute(qc, backend=backend, shots=1, memory=True)
11 task_result = task.result()
12 state = task_result.get_statevector(qc)
13
14 display(qc.draw('mpl'), state.draw('bloch'))
```

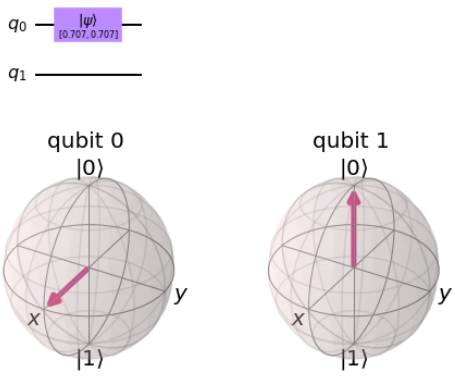


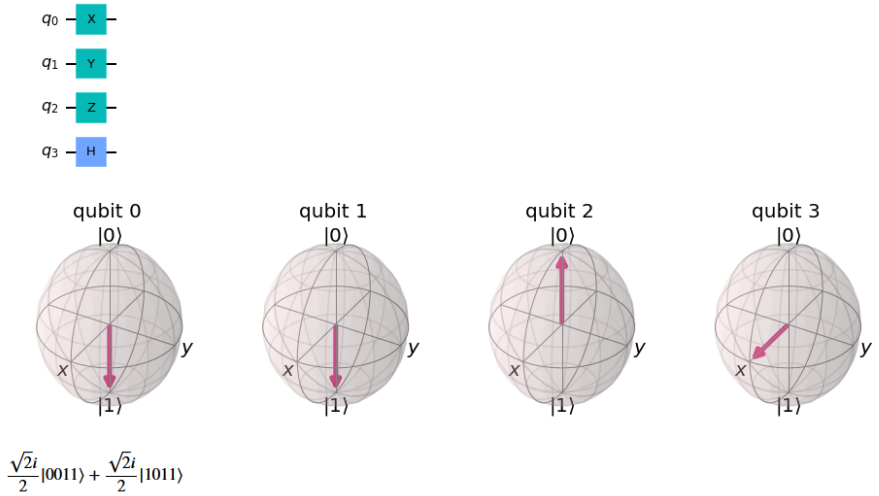
Abbildung 3.7.: Definition des Zustands eines Qubits und Darstellung des Zustands als Bloch-Kugel und des Quantenschaltkreises.

Bei der Definition eines Zustands ist zu beachten, dass das Betragsquadrat der Vorfaktoren in Summe 1 ergibt (s. Gl. 2.30). Dies wird implizit von der Klasse `initialize` überprüft und gegebenenfalls als Fehlermeldung ausgegeben [41].

3.2.3 | Gatter in Qiskit

Wie bereits in dem Theorieteil 2.3.2 ausgeführt, können Quantengatter genutzt werden, um den Zustand eines Qubits zu manipulieren. In Qiskit werden Quantengatter unter Angabe der beteiligten Qubits als Funktionen auf den Quantenschaltkreis aufgerufen [39]. Die zentralen Ein-Qubit-Gatter werden in Abbildung 3.8 in den Zeilen 6-9 auf einzelne Qubits angewendet. Der Zustand wird jeweils als Bloch-Kugel und für alle vier Qubits gesamt betrachtet als Zustandsvektor ausgegeben. Qubit 0 wird dabei von dem ganz rechten Eintrag des Ket-Vektors repräsentiert, da die Einträge erneut von rechts nach links beginnend mit Null durchnummeriert werden.

```
1  from qiskit import *
2  from pylatexenc import *
3
4  qc = QuantumCircuit(4)
5
6  qc.x(0)
7  qc.y(1)
8  qc.z(2)
9  qc.h(3)
10
11 backend = Aer.get_backend('statevector_simulator')
12 task = execute(qc, backend=backend, shots=1, memory=True)
13 task_result = task.result()
14 state = task_result.get_statevector(qc)
15
16 display(qc.draw('mpl'), state.draw('bloch'), state.draw('latex'))
```



$$\frac{\sqrt{2}i}{2}|0011\rangle + \frac{\sqrt{2}i}{2}|1011\rangle$$

Abbildung 3.8.: Definition des Zustands eines Qubits und Darstellung des Zustands als Bloch-Kugel und des Quantenschaltkreises.

Anhand der Visualisierung mit der Bloch-Kugel wird die in dem Theorieteil 2.3.2 diskutierte Wirkung der einzelnen Gatter deutlich. Das X-Gatter (s. Zeile 6) invertiert den Zustand des Qubits 0. Die anderen beiden Pauli-Gatter (s. Zeile 7-8) verursachen eine Rotation und die Y- bzw. Z-Achse. Die Superposition des Zustands nach der Anwendung des Hadamard-Gatters in Zeile 9 wird anhand der Bloch-Kugel deutlich und ist ebenfalls an dem Zustandsvektor ablesbar, der sich lediglich in der Stelle der Ket-Vektoren der einzelnen Summanden unterscheidet, die für Qubit 3 steht.

Die Wirkung der Ein-Qubit-Gatter lässt sich darüber hinaus mit dem Widget `gate_demo` veranschaulichen [19, 42]. Das Widget gibt eine Bloch-Kugel aus und die als Argument übergebenen Ein-Qubit-Gatter, die verwendet werden sollen, in Form von Button [19, 42]. Zudem wird ein Reset-Button ausgegeben, um das Qubit wieder in den Ausgangszustand $|0\rangle$ zu versetzen. In Zeile 1 der Abbildung 3.9 wird zunächst das Widget importiert und anschließend in Zeile 2 ausgeführt, wobei die gewünschten Gatter im Argument übergeben werden.

```
1 from qiskit_textbook.widgets import gate_demo
2 gate_demo(gates= 'pauli+h')
```

X Y Z H Reset

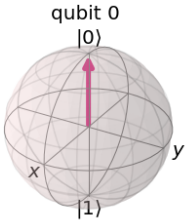


Abbildung 3.9.: Das Widget veranschaulicht die Wirkung der Ein-Qubit-Gatter anhand der Bloch-Kugel.

Das Widget eignet sich besonders, um die Wirkung von verschiedenen Ein-Qubit-Gattern hintereinander zu veranschaulichen und zur Identifikation äquivalenter Realisierungsmöglichkeiten von Quantengattern und Quantengatterfolgen. Neben den Ein-Qubit-Gattern sind ebenfalls die in Abschnitt 2.3.2 dargestellten Mehr-Qubit-Gatter in Qiskit implementiert und werden analog aufgerufen [36]. Für das CNOT-Gatter muss als erstes das Kontrollqubit und als zweites das Zielqubit angegeben werden. Das Toffoli-Gatter erfordert zwei Kontrollqubits, die zuerst definiert werden, worauf die Angabe des Zielqubits folgt [36]. Die Anwendung des Quantengatters erfolgt jeweils in Zeile 6 von Abbildung 3.10 für das CNOT-Gatter und 3.11 für das Toffoli-Gatter. Mit der in Abbildung 3.8 vorgestellten Möglichkeit Qubits in einem bestimmten Zustand zu initialisieren, können die Eingangsqubits des Toffoli-Gatters entsprechend konfiguriert werden, um ein NAND-Gatter oder ein OR-Gatter zu realisieren, wie es in dem Theorieteil ausgeführt ist.

```
1  from qiskit import *
2  from pylatexenc import *
3
4  qc = QuantumCircuit(2)
5
6  qc.cx(0, 1)
7
8  backend = Aer.get_backend('statevector_simulator')
9  task = execute(qc, backend=backend, shots=1, memory=True)
10 task_result = task.result()
11 state = task_result.get_statevector(qc)
12
13 qc.draw('mpl')
```

Abbildung 3.10.: In Zeile 6 wird das CNOT-Gatter angewendet, wobei Qubit 0 als Kontrollqubit und Qubit 1 als Zielqubit definiert sind. Anschließend wird der Quantenschaltkreis mit Zeile 13 ausgegeben.

```
1  from qiskit import *
2  from pylatexenc import *
3
4  qc = QuantumCircuit(3)
5
6  qc.ccx(0, 1, 2)
7
8  backend = Aer.get_backend('statevector_simulator')
9  task = execute(qc, backend=backend, shots=1, memory=True)
10 task_result = task.result()
11 state = task_result.get_statevector(qc)
12
13 qc.draw('mpl')
```

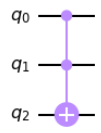


Abbildung 3.11.: Das Toffoli-Gatter wird mit Qubit 0 und Qubit 1 als Kontrollqubits und Qubit 2 als Zielqubit in Zeile 6 angewendet und mithilfe von Zeile 13 ausgegeben.

Die Tabelle A.1 in dem Anhang A.1 stellt eine Auflistung einiger ausgewählter Quantengatter und der dazugehörigen Funktion in Qiskit dar.

3.2.4 | Messungen und Experimente in Qiskit

Messungen können in Qiskit mit der Funktion `measure` definiert werden [39]. Als Argumente übernimmt die Funktion zum einen das Qubit, welches gemessen werden soll, und zum anderen ein klassisches Bit, an das das Messergebnis übergeben wird [39]. Dementsprechend muss bei der Initialisierung des Quantenschaltkreises mindestens ein klassisches Bit definiert werden, um eine Messung durchführen zu können [39]. Die Anzahl der Bits wird als zweites Argument an die Funktion `QuantumCircuit` übergeben [39]. In Zeile 5 des Codebeispiels in Abbildung 3.12 erfolgt genau dies. Es wird ein Quantenschaltkreis mit vier Qubits und vier Bits konstruiert. Die Anzahl der Bits ist in der bildlichen Ausgabe des Quantenschalkreises anhand der „4“ ablesbar, die über dem diagonalen Strich notiert ist, der durch die „Leitung“ für die klassischen Bits verläuft. Die „Leitung“ ist mit einem „c“ beschriftet, was für „classical“steht. Die Messung von Qubit 1 wird in Zeile 6 definiert, wobei das Ergebnis in Bit 1 gespeichert wird. Analog wird das Ergebnis der Messung von Qubit 2 in Zeile 7 in Bit 2 gespeichert. In welchem Bit ein Messergebnis gespeichert wird, lässt sich anhand der Zahl rechts neben dem Pfeil ablesen, der von dem Schaltsymbol für die Messung zu der Leitung der klassischen Bits verläuft.

```
1 from qiskit import *
2 from pylatexenc import *
3
4
5 qc = QuantumCircuit(4,4)
6 qc.measure(1,1)
7 qc.measure(2,2)
8 qc.draw("mpl")
```

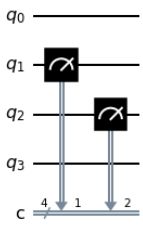


Abbildung 3.12.: In Zeile 6 und 7 wird die Messung auf Qubit 1 bzw. Qubit 2 durchgeführt und das Ergebnis an ein klassisches Bit übergeben.

Mit dem vorangegangenen Code wurde der Quantenschaltkreis mit den Messungen theoretisch definiert. Es wurde noch keine Messung durchgeführt. Dafür folgt erneut die Simulation auf einem entsprechenden Backend. Für diesen Fall wird `qasm_simulator` als Backend verwendet [39]. Dieser Simulator ermöglicht die Simulation von Messungen innerhalb des Quantenschalkreises auf einem klassischen Gerät, ohne dass ein Quantencomputer benötigt wird [39]. In Abbildung 3.13 wird der obig definierte Quantenschaltkreis simuliert und das Ergebnis der Messungen als Histogramm dargestellt. Für die Visualisierung der Ausgabe in Form eines Histogramms wird das Modul `qiskit.visualization` benötigt [43]. In Zeile 3 des Codebeispiels wird die Funktion `plot_histogram` des Moduls importiert, welche als Argumente das Messergebnis, die Farbe der Säulen und einen Titel übernehmen kann [43]. In dem Codebeispiel wird der Quantenschaltkreis 1000-mal simuliert, was in Zeile 10 als Parameter an `shots` übergeben wird. Die Anzahl der Simulationen kann zudem über der Säule des Histogramms abgelesen werden bzw. ergibt sich als Summe der einzelnen Vorkommen der der gemessenen Zustände. Unter der Säule wird darüber hinaus der gemessene Zustand vermerkt, der in diesem Fall eindeutig $|0000\rangle$ ist, was sich direkt ergibt, weil kein Quantengatter angewendet wurde und sich somit alle Qubits noch in dem Ausgangszustand $|0\rangle$ befinden. Der ausgegebene Zustand beschreibt demzufolge vier Objekte, jedoch werden nur zwei Qubits gemessen. Dies liegt darin begründet, dass der Quantenschaltkreis simuliert und anschließend der Zustand aller zu Beginn definierten klassischen Bits betrachtet wird und als Ergebnis ausgegeben wird. Es werden folglich auch die Bits aufgeführt, in die kein Messwert gespeichert wurde. Da der Ausgangszustand von Bits der Zustand 0 ist, ergibt sich für diesen Quantenschaltkreis $|0000\rangle$ als Ergebnis. Die Bits 1 und 2 befinden sich in Zustand 0, weil die Qubits 1 und 2 in Zustand $|0\rangle$ gemessen wurden, und die Bits 0 und 3 befinden sich noch in dem Grundzustand 0.

```
1  from qiskit import *
2  from pylatexenc import *
3  from qiskit.visualization import plot_histogram
4
5  qc = QuantumCircuit(4,4)
6  qc.measure(1,1)
7  qc.measure(2,2)
8
9  sim = Aer.get_backend('qasm_simulator')
10 result = sim.run(qc, shots=1000).result()
11 counts = result.get_counts()
12 plot_histogram(counts, color='royalblue', title='Messergebnisse des Quantenschaltkreises qc')
```

Abbildung 3.13.: Die Simulation wird in beginnend in Zeile 9 definiert und folgt dem Muster der Simulation des Zustands in Abbildung 3.6.

Mit der Funktion `measure_all` können ausschließlich die Ergebnisse der Messvorgänge betrachtet und ausgewertet werden, ohne Bits zu berücksichtigen, in denen kein Messwert gespeichert wurde [44]. In diesem Fall ist die Angabe der Anzahl an Bits bei der Definition des Quantenschaltkreises obsolet, da durch die Funktion die erforderliche Menge an Bits generiert wird [44]. Die entsprechende „Leitung“ wir mit „meas“ beschriftet. Im Folgenden wird diese Methode verwendet.

Gemäß der theoretischen Grundlage aus Kapitel 2 kann sich ein Qubit in einem Superpositionszustand befindet, welcher sich durch eine Messung zu einem der Basiszustände auflöst. Über den gemessenen Zustand des Qubits können nur Vorhersagen in Form von Wahrscheinlichkeiten getroffen werden. Wird ein Qubit mit einem Hadamard-Gatter präpariert, befindet es sich nach einer Messung zu jeweils 50 % entweder in Zustand $|0\rangle$ oder $|1\rangle$. Diese theoretische Erkenntnis lässt sich durch Experimente und Simulationen mit Qiskit überprüfen. Mit dem Codebeispiel in Abbildung 3.14 wird genau ein solches Experiment durchgeführt. Zunächst wird in Zeile 5 ein Quantenschaltkreis mit zwei Qubits erstellt. Auf Qubit 0 wird das X-Gatter angewendet, wodurch es sich in Zustand $|1\rangle$ befindet (s. Zeile 6). In der nachfolgenden Zeile 7 wird Qubit 1 durch ein Hadamard-Gatter in den Superpositionszustand $|+\rangle$ versetzt. Die Simulation der Messung ergibt das aus der Theorie abgeleitete Ergebnis: Die Qubits befinden sich bei ungefähr 50 % der Messungen in Zustand $|01\rangle$ und zu ebenfalls ungefähr 50 % der Messungen in Zustand $|11\rangle$. Wie bei ähnlichen Zufallsexperimenten wie beispielsweise einem Münzwurf oder einem Würfelwurf entspricht die reale Verteilung nicht exakt den angegebenen Wahrscheinlichkeiten, sondern ist statistisch verteilt.

```python
from qiskit import *
from pylatexenc import *
from qiskit.visualization import plot_histogram

qc = QuantumCircuit(2)
qc.x(0)
qc.h(1)
qc.measure_all()

sim = Aer.get_backend('qasm_simulator')
result = sim.run(qc, shots=1000).result()
counts = result.get_counts()

display(qc.draw('mpl'))
plot_histogram(counts, color='royalblue', title = 'Messergebnisse bei Anwendung eines Hadamard-Gatters')
```

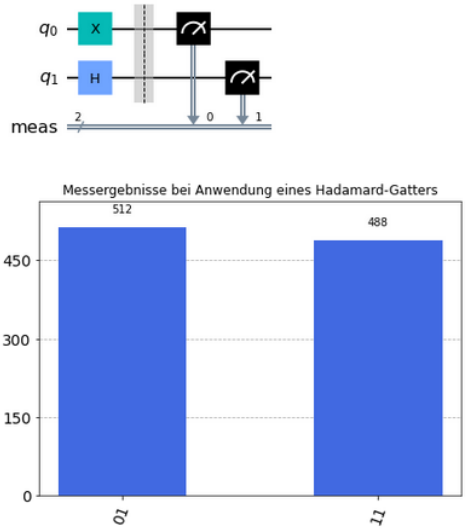


Abbildung 3.14.: Es wird ein Quantenschaltkreis mit zwei Qubits simuliert, wovon sich ein Qubit in einem Superpositionszustand befindet. Die „Leitung“ für die Messung ist mit „meas“ benannt, weil `measure_all` verwendet wurde.

Auf diese Weise können quantenmechanische Fragestellungen mithilfe von Qiskit untersucht und quantenmechanische Experimente durchgeführt werden, wozu lediglich einige wenige Funktionen benötigt werden. Das vorangegangene Kapitel gibt jedoch nur einen kleinen Einblick in die implementierten Module und Funktionen von Qiskit und soll lediglich als Einführung dienen und eine Verständnisgrundlage bilden, um die theoretischen Konzepte aus Kapitel 2 mithilfe von Qiskit vermitteln zu können.

4 Implementierung eines Jupyter Notebooks

Neben dem Einsatz von Qiskit zur Untersuchung quantenmechanischer Probleme kann Qiskit auch zur Vermittlung und zum Verständnis der Grundlagen der Quantenmechanik und der Quantencomputer verwendet werden. Dafür eignen sich besonders Jupyter Notebooks, da in diesen sowohl Codezellen als auch Textzellen erstellt werden können [45]. Dieses Kapitel widmet sich der Vorstellung eines beispielhaften Jupyter Notebooks, welches im Rahmen des Schulprojekts für die Vermittlung quantenmechanischer Grundlagen für das Quantencomputing erstellt wurde. Zum einen besteht das Jupyter Notebook aus Textzellen mit theoretischen Erläuterungen und Abbildungen und zum anderen aus Codezellen, die die in Kapitel 3 vorgestellten Funktionen enthalten, um die Theorie anschaulich darzustellen und zu belegen. Komplettiert wird das Jupyter Notebook von einem eigens erstellten Single Choice Widget. Das vollständige Jupyter Notebook ist im Anhang aufgeführt A.2.

4.1 Jupyter Notebook

Jupyter Notebook ist eine Open-Source Anwendung, die das Erstellen von Dokumenten ermöglicht, die Text und Programmcode enthalten [45]. Die mit der webbasierten Anwendung erstellten Notebooks eignen sich daher zur Lehre [45] und sind insbesondere bei Datenwissenschaftlern der Standard, da sie die Datenanalyse selbst sowie deren Aufzeichnung und Reproduzierbarkeit vereinfachen [46]. Darüber hinaus können Jupyter Notebooks in einem Online-Repository gespeichert und mittels verschiedener Plattformen kollaborativ bearbeitet werden [47]. Eine Option Jupyter Notebooks zu verwenden ist beispielsweise die von Google bereitgestellte Umgebung Google Colaboratory, die kurz als Colab bezeichnet wird [48]. Colab ist kostenfrei und erfordert lediglich einen Google-Account zur Nutzung [48]. Mit Colab kann Python-Code direkt online programmiert und ausgeführt werden, ohne dass eine lokale Installation notwendig ist [48]. Aus den genannten Gründen wurde Google Colab für das Schulprojekt verwendet, wobei Google Accounts zur Verfügung gestellt wurden.

4.2 Single Choice Widget

Die Jupyter Notebook Anwendung bietet außerdem die Möglichkeit, Widgets zu verwenden [49]. Mithilfe der Widgets kann das Notebook interaktiv gestaltet werden [49]. Das bereits angesprochene selbst erstellte Single Choice Widget wurde unter anderem aus den bestehenden Widgets erstellt und wird im Folgenden vorgestellt.

Im Kern ist das Single Choice Widget eine Funktion, die zwei weitere Funktionen enthält (s. Abb. 4.1). Zunächst werden die erforderlichen Module importiert (s. Zeile 2 bis 4). Die Funktion `sc_widget` übernimmt als Argument die Frage, die falschen alternativen Antwortmöglichkeiten und die richtige Antwort (s. Zeile 7). Die Antworten werden zu einer Liste zusammengeführt und mit der Funktion `shuffle` aus dem Modul `random` zufällig angeordnet (s. Zeile 9 und 10) [50]. Um die Antwortmöglichkeiten in Form von auswählbaren Punkten aufzuführen, werden diese in den Zeilen 13 bis 19 sogenannten „RadioButtons“ aus dem `RadioButtons` Widget zugewiesen [51]. Diese können via Mausklick oder Tippen auf den Bildschirm ausgewählt werden [51]. Um die ausgewählte Antwortmöglichkeit zur Überprüfung abzugeben, wird ein Abgabebutton mit dem Widget `Button` erstellt [52]. Dadurch kann registriert werden, ob der Button geklickt wird (s. Zeile 22-25) [52]. Damit die Frage und die Widgets mit den Antwortmöglichkeiten und dem Abgabebutton ausgegeben werden, wird die Variable `rueckmeldung_ausgeben` als Instanz des Widgets `Output` in Zeile 28 erstellt [53].

Anschließend werden zwei Funktionen definiert. Die Funktion `ausgeben` ermöglicht bei Aufruf die Ausgabe der Antwortmöglichkeiten und des Abgabebuttons (s. Zeile 31-35). Darüber hinaus übernimmt die Funktion ein Argument, was als Text ebenfalls ausgegeben wird. Mithilfe der Funktion `ueberpruefen` in den Zeilen 38 bis 42 wird die gegebene Antwort mit der korrekten Antwort verglichen und abhängig von dem Ergebnis die Rückmeldung als Ausgabetext in der Variable `ausgabetext` gespeichert (s. Zeile 41). Nachfolgend wird die vorher definierte Funktion `ausgabe` aufgerufen und der Ausgabetext übergeben, um bei Aufruf der Funktion die Antwortmöglichkeiten, den Abgabebutton und den Ausgabetext auszugeben.

In Zeile 45 wird die Funktion `ausgabe` aufgerufen, um bei der Ausführung des Single Choice Widgets die Antwortmöglichkeiten und den Abgabebutton auszugeben. Wenn der Button geklickt wird, wird mit der darauf folgenden Zeile 46 die Funktion `ueberpruefen` aufgerufen und somit die gegebene Antwort kontrolliert und das entsprechende Feedback ausgegeben. Zunächst löscht die Funktion den aktuellen Output (s. Zeile 40), damit mögliche vorherige Rückmeldungen nicht mehr angezeigt werden. Aus diesem Grund werden mit der Funktion `ausgabe` die Antwortmöglichkeiten und der Abgabebutton erneut ausgegeben.

Die Frage wird als Titel bzw. Label einer vertikalen Box dargestellt, die die weiteren Widgets und Ausgaben enthält. Dies wird damit umgesetzt, dass die Outputinstanz `rueckmeldung_ausgeben` als Argument übergeben wird (s. Zeile 49).

Abschließend ist in den Zeilen 53 bis 57 ein beispielhafter Aufruf des Single Choice Widgets aufgeführt. In diesem Fall werden die falschen Antwortmöglichkeiten separat als Variable `testfrage_options` definiert und der Funktion übergeben. Für die Ausgabe wird erneut die Funktion `display` aus dem Modul `IPython.display` verwendet [40].

```python
# nötige Module importieren
import ipywidgets as widgets
from IPython.display import display
import random

# Funktion erstellen, die Single-Choice Fragen ausgibt
def sc_widget(frage, falschemoeglichkeiten, richtige_antwort):
    # Übergebene Antworten zusammenfügen und durchmischen
    antworten_text = [richtige_antwort] + falschemoeglichkeiten
    random.shuffle(antworten_text)

    # Antwortmöglichkeiten der Buttons für Single-Choice übergeben und Fenstergröße für Widget definieren
    antwortmoeglichkeiten = widgets.RadioButtons(options = antworten_text,
                                        description = '',
                                        disabled = False,
                                        style= {'description_width': 'initial'},
                                        align_items='stretch',
                                        layout = widgets.Layout(width='750px', height='auto')
                                        )

    # Button zur Abgabe der Antwort definieren
    abgabeButton = widgets.Button(description='einloggen',
                                  disabled=False,
                                  button_style=''
                                  )

    # Output Instanz mit Widget erstellen und an Variable übergeben
    rueckmeldung_ausgeben = widgets.Output()

    # Antwortmöglichkeiten als Radiobutton und Abgabebutton bei Display der Funktion sc_widget ausgeben
    def ausgeben(text):
      with rueckmeldung_ausgeben:
        display(antwortmoeglichkeiten)
        display(abgabeButton)
        print(str(text))

    # Funktion definieren, die die Antwort überprüft und eine Rückmeldung ausgibt
    def ueberpruefen(index):
      with rueckmeldung_ausgeben:
          rueckmeldung_ausgeben.clear_output()
          ausgabetext = "\x1b[6;37;42m Korrekt " if antwortmoeglichkeiten.value == richtige_antwort else "\x1b[5;37;41m Fal
          ausgeben(ausgabetext)                # else "\x1b[5;37;41m Falsch "
                                               # Der obige Kommentar ist für die Lesbarkeit beim Export erstellt.
    # Funktionen aufrufen
    ausgeben('')
    abgabeButton.on_click(ueberpruefen)

    # Box erstellen, um Widget im Ganzen auszugeben
    return widgets.VBox([widgets.Label(value=frage), rueckmeldung_ausgeben])


######################################## Single-Choice Fragen erstellen ########################################
testfrage_options = ['TU Braunschweig','Universität Hildesheim']
Testfrage = sc_widget('An welcher Universität wird diese Bachelorarbeit eingereicht?'
                      , testfrage_options,'Leibniz Universität Hannover')
# Frage ausgeben
display(Testfrage)
```

An welcher Universität wird diese Bachelorarbeit eingereicht?

- ○ Universität Hildesheim
- ◉ Leibniz Universität Hannover
- ○ TU Braunschweig

einloggen

Korrekt

Abbildung 4.1.: Implementierung des eigens erstellten Single Choice Widgets mit einer beispielhaften Frage.

4.3 Gestaltung des Jupyter Notebooks

Zu Beginn Unterrichtsstunde wurde mit einer Präsentation grundsätzlich in die Thematik eingeführt und Begriffe wie die Bloch-Kugel, Qubits und Gatter ein erstes Mal beschrieben. Nach der anschließenden Bearbeitung des Jupyter Notebooks wurde eine abschließende Diskussionsrunde abgehalten, an deren Ende die Fragebögen zur Erhebung der Wirksamkeit der Unterrichtsstunde bezüglich der Vermittlung der Inhalte individuell ausgefüllt wurden. Das Jupyter Notebook wurde in Gruppen mit maximal vier Lernenden bearbeitet und folgt grundsätzlich der Struktur, die auch für das theoretische Kapitel 2 gewählt wurde. Dementsprechend werden nach einer praktischen Einführung in die Benutzung von Jupyter Notebooks zunächst die Grundlagen für einen Quantenschaltkreis gelegt, indem Qubits und Quantengatter erläutert werden. Dabei wird ebenfalls der Vergleich mit Bits gezogen und die Modelle des Lichtschalters und des Dimmers zur Veranschaulichung verwendet. Dies erfolgt, um an mögliches Vorwissen der Lernenden anzuschließen und verdeutlicht anhand des Vergleichs, wie sich die Quantenmechanik von der klassischen Physik und somit im Prinzip von Phänomenen unterscheidet, die aus dem Alltag bekannt sind und sich intuitiv erschließen lassen. Die einzelnen inhaltlichen Abschnitte zu Bits und Qubits schließen mit Single Choice Fragen.

Daran anschließend werden Logikgatter und Quantengatter eingeführt. Zur Erklärung der Funktion von Logikgattern wurden hauptsächlich Wahrheitstabellen herangezogen. Die Beschreibung der Auswirkungen der Quantengatter beschränkt sich ebenfalls auf eine tabellarische Darstellung, wenn dies möglich ist. Zudem wird die Wirkung in einfachen Termen und mithilfe von textlichen Beschreibungen ausgeführt. Auch am Ende dieses Abschnitts sind Single Choice Fragen zu beantworten.

Darauf aufbauend wird die Zustandsdarstellung mithilfe der Bloch-Kugel genutzt, um Zustandsveränderungen durch Quantengatter zu veranschaulichen und Quantenschaltkreise einzuführen. Im Mittelpunkt des Abschnitts steht das `gate_demo` Widget, welches ausführlich in dem vorangegangenen Abschnitt 3.2.3 beschrieben wird. Darüber hinaus werden Quantenschaltkreise beispielhaft implementiert und so die Verwendung der vorherig erklärten Quantengatter deutlich gemacht.

Das Jupyter Notebook mündet in der Thematisierung von Wahrscheinlichkeiten in der Quantenmechanik, wodurch es zu der Anwendung des gesamten vorher erlangten Wissens kommt und wofür Experimente mit Qiskit verwendet werden. Zunächst werden Messungen auf Qubits simuliert, die sich in einem der Basiszustände befinden. Dabei wird mit dem Single Choice Widget vor der Durchführung der Messung jeweils das erwartete Ergebnis abgefragt, was zu einer aktiven Auseinandersetzung mit dem Inhalt des Abschnitts führen soll. Im finalen Teil des Abschnitts werden Qubits betrachtet, die sich in einem Superpositionszustand befinden. Dafür werden in fünf Messdurchgängen jeweils 1000 Messungen auf die Qubits durchgeführt. Überdies ist es möglich, ein Qubit in einem gewünschten Zustand zu präparieren, wozu lediglich die Eingabe der Wahrscheinlichkeit mittels eines „Sliders“ erforderlich ist [54]. Der „Slider“ lässt sich mit dem Slider-Widget implementieren [54].

Den Abschluss des Jupyter Notebooks bildet ein Single Choice Quiz mit Fragen, die alle vorangegangenen Themen abdecken.

4.4 Begründung der Gestaltung des Jupyter Notebooks

Bei der Erstellung des Jupyter Notebooks wurden wie bei jeder Planung von Wissensvermittlung zu Beginn die beiden Prozesse der didaktischen Reduktion und der didaktischen Rekonstruktion durchlaufen [55]. Die didaktische Reduktion meint die begründete Auswahl von relevanten Inhalten für eine bestimmte Zielgruppe und Lernsituation aus einer größeren Obermenge von Inhalten [55]. Nicht relevante oder unpassende Inhalte werden ausgelassen, um den Lernprozess effektiver zu gestalten [55]. Beispielsweise wird die Matrixdarstellung der Quantengatter in dem Jupyter Notebook nicht verwendet, sie wurde didaktisch reduziert. Dies lässt sich mit der Zielgruppe begründen. Bei der Erstellung des Jupyter Notebooks waren gymnasiale Physikkurse der zwölften Klasse mit erhöhtem Anforderungsniveau die Adressaten. Das Kerncurriculum sieht für den Physikunterricht in der gymnasialen Oberstufe keine Matrizendarstellung vor [8]. Zwar werden Matrizen im Mathematikunterricht eingeführt [56], dies erfolgt jedoch größtenteils erst ab der Klassenstufe 13, weshalb den Lernenden Matrizen noch nicht bekannt sind. Dementsprechend wurde ebenfalls auf Rechnungen mit Matrizen und Rechnungen im Allgemeinen verzichtet und die Vorgänge in Worten beschrieben. Mit der Didaktischen Rekonstruktion werden die reduzierten Inhalte in eine sinnvolle Reihenfolge gebracht und für die Lernenden aufbereitet [55]. Insbesondere der Punkt der didaktischen Reduktion und der Strukturierung der Inhalte wird vertiefend in der Bachelorarbeit „Didaktische Analyse einer Unterrichtsstunde zur Vermittlung quantenmechanischer Grundlagen“ von Lara Niemann behandelt.

Um das aufbereitete Wissen möglichst lernwirksam vermitteln zu können, werden einige Ziele angestrebt. Es sollte eine hohe kognitive Aktivierung [57], eine hohe Verarbeitungstiefe [58] und die Selbstgesteuertheit [59] der Lernenden gefördert werden, wobei der serielle Positionseffekt [60] berücksichtigt werden sollte.

Kognitive Aktivierung gilt es zu fördern, da nach der Auffassung des Konstruktivismus Lernende nur wirklich lernen, wenn die vermittelten Informationen aktiv verarbeitet werden und die Lernenden aktiv und aufmerksam an dem Lernprozess teilnehmen [61]. Insbesondere durch den Rückgriff auf Vorwissen und die Anregung zum Denken auf einem individuell hohen Niveau lässt sich die kognitive Aktivierung steigern [57]. An das Vorwissen der Lernenden wird durch den Vergleich zwischen der klassischen Physik und der Quantenmechanik angeknüpft. Darüber hinaus werden Modelle aus dem Alltag verwendet, um die physikalischen Konzepte anschaulich darzustellen. Bits werden mit Lichtschaltern assoziiert und Qubits mit Dimmern bzw. Kugeln. Mithilfe von offenen Diskussionsfragen für die Kleingruppen wurde zudem die kognitive Auseinandersetzung mit den Inhalten und ein möglichst hoher individueller Denkprozess angeregt.

Die Diskussionsfragen fördern außerdem eine höhere Verarbeitungstiefe. Die Verarbeitungstiefe spielt nach der „Theorie der Verarbeitungstiefe“ von Craik und Lockhart eine zentrale Rolle bei der Abspeicherung von Informationen im Gedächtnis [58]. Eine tiefere Verarbeitung kann insbesondere dadurch erreicht werden, Informationen aus verschiedenen Perspektiven zu betrachten, auf unterschiedliche Arten zu verarbeiten und mit bestehendem Wissen zu verknüpfen [58]. Bei Diskussionen in Gruppen treffen differierende Ansichten aufeinander, was die Verarbeitungstiefe erhöht. Außerdem verbessert ein mündlicher Austausch in Form einer Diskussion über vorher gelesene Informationen das Verständnis dieser [62]. Darüber hinaus sollten möglichst viele Sinne während der

Verarbeitung der Informationen angesprochen werden, um die Verarbeitungstiefe zu erhöhen [63]. Diese Punkte werden durch die Diskussionsfragen, die Textabschnitte mit den darin enthaltenen veranschaulichenden Abbildungen und die Codezellen mit den dazugehörigen Ausgaben berücksichtigt. Dadurch werden kognitive, visuelle, auditive und kommunikative Reize gesetzt und der Verarbeitungsprozess und die Auseinandersetzung mit den Inhalten vertieft.

Neben den Diskussionsfragen nehmen die Single Choice Fragen eine zentrale Rolle innerhalb des Jupyter Notebooks ein. Diese Methode wurde aus verschiedenen Gründen gewählt. Das Abfragen von Wissen verbessert nachweislich den Lerneffekt von Lehr-Lern-Situationen [64]. Dies spiegelt sich ebenfalls in den Leistungen der Lernenden bezogen auf das gelernte Wissen wider [65]. Insbesondere im Vergleich zu Lernenden, die die Inhalte lediglich erneut lernen und die nicht abgefragt werden, schneiden die zuvor abgefragten Lernenden in Tests besser ab [65]. Neben dem abgefragten Wissen wird auch nicht abgefragtes Wissen besser behalten [65]. Zudem wird mit dem Single Choice Widget in Teilen die lernfördernde Methode „Interleaving“ angewendet [66]. Interleaving meint, dass Inhalte nicht nur in Blöcken nacheinander vermittelt werden, sondern eine Durchmischung erfolgt [66]. Wenn beispielsweise die Inhalte A, B und C vermittelt werden sollen, ist der Ablauf nicht $A \rightarrow B \rightarrow C$, sondern zum Beispiel $A_1 \rightarrow B_1 \rightarrow C_1 \rightarrow B_2 \rightarrow A_2 \rightarrow C_2$. Dies erfolgt vor allem in dem Abschnitt zu Wahrscheinlichkeiten in der Quantenmechanik und dem abschließenden Quiz, in dem die Fragen nicht der Struktur des Jupyter Notebooks entsprechen, was ebenfalls den Lerneffekt fördert [66]. Das Single Choice Widget wurde somit als Übung verwendet, wodurch sich die Lernenden aktiv mit den Inhalten auseinander setzen und das neu erlangte Wissen anwenden müssen, was sich positiv auf den Lerneffekt auswirkt [61, 63]. Speziell wird das Single Choice Widget als sogenannte „apponierte Übung“ eingesetzt, die als solche 2007 von Hubwieser eingeführt wurde [61]. Das bedeutet, dass die Fragen direkt an die theoretischen Ausführungen anschließen [61].

Das Single Choice Widget soll zudem durch seinen Spielecharakter für eine positive Lernatmosphäre sorgen. Diese nimmt in Bezug auf die Speicherung von Wissen eine zentrale Rolle ein, da sie ausschlaggebend dafür ist, in welchem Gehirnareal die neuen Informationen abgespeichert werden [67]. Wenn die emotionale Grundstimmung der Lernenden negativ ist, wird das Gelernte in der Amygdala abgespeichert, wodurch ein kreativer Umgang mit dem neu erworbenem Wissen nicht möglich ist [67]. Dies liegt darin begründet, dass die Amygdala insbesondere in Gefahrensituationen aktiv ist und daher schnell geradlinige Gedanken liefert [67]. Im Gegensatz dazu werden Lerninhalte in einer positiven Grundstimmung in dem Hippocampus abgespeichert, was einen kreativen Umgang mit den neu erworbenen Inhalten ermöglicht [67].

Die Verwendung eines Jupyter Notebooks als Wissenvermittlungs- und zugleich Übungsmedium ermöglicht nicht nur die Verwendung von interaktiven Widgets, sondern unterstützt auch Kernideen des Konstruktivismus [59]. Gemäß dem Konstruktivismus sollte das Lernen selbstgesteuert und problemorientiert erfolgen [59]. Das Jupyter Notebook kann eigenständig und in dem eigenen Lerntempo bearbeitet werden und gewährleistet dauerhaft einen Rücksprung zu vorherigen Abschnitten. Außerdem ist durch die digitale Realisierung ein Zugriff unabhängig von Ort und Zeit gegeben. Die Problemorientierung

ergibt sich vor allem aus den Codezellen, da sie es ermöglichen, konkrete Probleme zum Gegenstand des Lernens zu erheben. In diesem Fall sind speziell gedankliche Probleme und Widersprüche zu den intuitiven Vorstellungen der Lernenden die zentralen Gegenstände des Lernprozesses.

Bei dem Aufbau des Jupyter Notebooks wird ebenfalls der serielle Positionseffekt berücksichtigt [60]. Dieser besagt, dass Informationen, die am Anfang und am Ende eines Informationsblocks präsentiert werden, besser erinnert werden als die Informationen in der Mitte des Blocks [60]. Aus diesem Grund wurden jeweils am Ende jedes Informationsabschnitts kurze zusammenfassende Blöcke verfasst, die die wichtigsten Inhalte kompakt darstellen. Diese Blöcke sind zudem visuell durch zwei vertikale Linien von den anderen Inhalten abgegrenzt. In der Google Colaboratory Umgebung ist es möglich, Überschriften zu definieren, die in einem Verzeichnis aufgelistet werden und deren nachfolgende Inhalte eingeklappt werden können. Dies fördert die Navigation und die Übersichtlichkeit.

Aus den aufgeführten Gründen wurde basierend auf den Grundlagen aus den Kapiteln 2 und 3 das Jupyter Notebook zur Vermittlung von quantenmechanischen Grundlagen für das Quantencomputing erstellt.

5 Fazit und Ausblick

Das Ziel der vorliegenden Bachelorarbeit bestand darin, die Grundlagen der Quantenmechanik und von Qiskit für das Quantencomputing zu erläutern, um eine Vermittlung der quantenmechanischen Grundlagen mit Qiskit zu ermöglichen und eine solche Vermittlung anhand eines Jupyter Notebooks beispielhaft auszuführen.

Zusammenfassend wird aus der Bachelorarbeit unter anderem deutlich, dass Quantencomputer auf Qubits basieren. Zudem können sich Qubits im Gegensatz zu Bits in einem Superpositionszustand, einem Überlagerungszustand befinden und untereinander verschränkt sein. Das heißt, wenn aus einer Menge verschränkter Qubits ein Qubit verändert wird, werden alle verschränkten Qubits beeinflusst. Eine solche Manipulation kann mit Quantengattern erfolgen, die das Äquivalent zu klassischen Logikgattern bilden. Quantengatter werden in Ein- und Mehr-Qubit-Gatter unterschieden, wobei beide Typen grundsätzlich reversibel sind. Die Wirkung eines Quantengatters kann dementsprechend rückgängig gemacht werden. Darüber hinaus lässt sich festhalten, dass mit Quantengattern Superpositionszustände erzeugt und Qubits verschränkt werden können. Durch die Verschaltung von Quantengattern werden Quantenschaltkreise realisiert, mit denen Berechnungen ausgeführt werden können. Um das Ergebnis von Berechnungen zu bestimmen, werden Messungen durchgeführt, die sich fundamental von Messungen in der klassischen Physik unterscheiden. Der Zustand eines Qubits, welches sich in einem Superpositionszustand befindet, steht vor dem Messvorgang nicht fest und wird erst durch diesen determiniert. Dementsprechend handelt es sich bei Vorhersagen in der Quantenmechanik um die Angabe von Wahrscheinlichkeiten.

Diese quantenmechanischen Grundlagen für das Quantencomputing sind in Qiskit implementiert. Mit dem SDK von IBM können Quantenschaltkreise konstruiert und auf klassischen Geräten sowie per Cloud-Zugriff auf realen Quantencomputern von IBM simuliert werden. Für die Implementierung erster Quantenschaltkreise einschließlich Messvorgänge sind lediglich einige wenige Funktionen erforderlich, was anhand der Einführung in Qiskit in Kapitel 3 aufgezeigt wird. Dadurch lassen sich mit geringem Aufwand quantenmechanische Experimente durchführen.

Als Umgebung für Qiskit eignet sich ein Jupyter Notebook, da in diesem sowohl Textzellen als auch Codezellen erstellt werden können. Ein beispielhaftes Jupyter Notebook wurde in dieser Bachelorarbeit begründet vorgestellt. Damit lässt sich abschließend festhalten, dass die quantenmechanischen Grundlagen und die Grundlagen von Qiskit für das Quantencomputing mithilfe von Qiskit und der Verwendung der Jupyter Notebook Umgebung vermittelt werden können.

Es bleibt jedoch offen, wie effektiv die Verwendung des Jupyter Notebooks als Medium zur Vermittlung dieser Inhalte im Vergleich zu anderen Medien und Methoden ist. Eine weiterführende Untersuchung könnte diese Frage überprüfen. Zum Beispiel können klassische Verfahren, wie herkömmliche Arbeitsblätter, Vorlesungen und klassische Übungszettel, und Jupyter Notebooks gegenübergestellt und die Wirksamkeit in Bezug auf die Vermittlung der Grundlagen für das Quantencomputing untersucht werden.

Da in dieser Bachelorarbeit gezeigt wurde, wie quantenmechanische Grundlagen mithilfe von Qiskit vermittelt werden können, lassen sich darauf aufbauend weitere Jupyter Notebooks erstellen, die fortschreitende Aspekte der Thematik vermitteln. Ein anschließendes Jupyter Notebook könnte beispielsweise das Konzept der Verschränkung von Qubits thematisieren, welches grundsätzlich in Qiskit implementiert ist [36]. Außerdem können relevante Quantenalgorithmen in Qiskit programmiert und mit Textzellen in einem Jupyter Notebook ausführlich erklärt werden. Ebenso können Programmierübungen in die Jupyter Notebooks integriert werden, indem leere Codezellen oder Codezellen mit Lücken auf eine Aufgabenstellung folgen.

Literatur

[1] D. Hoffmann. *Max Planck und die moderne Physik*. Berlin, Heidelberg: Springer, 2010. DOI: 10.1007/978-3-540-87845-2.

[2] M. Planck. »Über das Gesetz der Energieverteilung im Normalspektrum«. In: *Von Kirchhoff bis Planck: Theorie der Wärmestrahlung in historisch-kritischer Darstellung*. Hrsg. von H.-G. Schöpf. Wiesbaden: Vieweg + Teubner Verlag, 1978, S. 178–191. ISBN: 978-3-663-13885-3. DOI: 10.1007/978-3-663-13885-3_16.

[3] M. von Laue. »Max Planck«. *Naturwissenschaften* 35 (1948), S. 1–7. DOI: 10.1007/BF00626622.

[4] M. Homeister. *Quantum Computing verstehen: Grundlagen - Anwendungen Perspektiven*. 5. Aufl. Wiesbaden: Springer Vieweg, 2018. DOI: 10.1007/978-3-658-22884-2.

[5] M. A. Nielsen und I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge: Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[6] H. Ernst, J. Schmidt und G. Beneken. *Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis - Eine umfassende, praxisorientierte Einführung*. 7. Aufl. Wiesbaden: Springer Vieweg, 2020. DOI: 10.1007/978-3-658-30331-0.

[7] G. Pospiech. *Quantencomputer & Co: Grundideen und zentrale Begriffe der Quanteninformation verständlich erklärt*. Wiesbaden: Springer Spektrum, 2021. DOI: 10.1007/978-3-658-30445-4.

[8] N. Kultusministerium, Hrsg. *Physik: Kerncurriculum für das Gymnasium – gymnasiale Oberstufe die Gesamtschule – gymnasiale Oberstufe das Berufliche Gymnasium das Abendgymnasium das Kolleg*. 2017. URL: https://cuvo.nibis.de/cuvo.php?p=detail_view&docid=1517 (besucht am 06. 09. 2023).

[9] B. Just. *Quantencomputing kompakt: Spukhafte Fernwirkung und Teleportation verständlich erklärt*. Berlin, Heidelberg: Springer Vieweg, 2020. DOI: 10.1007/978-3-662-61889-9.

[10] R. Drechsel, A. Fink und J. Stoppe. *Computer: Wie funktionieren Smartphone, Tablet & Co.?* Berlin, Heidelberg: Springer, 2017. DOI: 10.1007/978-3-662-53060-3.

[11] B. Tolg. *Informatik auf den Punkt gebracht: Informatik für Life Science Studierende und ander Nicht-Informatiker*. Wiesbaden: Springer Vieweg, 2019. DOI: 10.1007/978-3-658-24131-5.

[12] B. M. Ellerhoff. *Mit Quanten rechnen: Quantencomputer für Neugierige*. 1. Aufl. Wiesbaden: Springer Spektrum, 2020. DOI: 10.1007/978-3-658-31222-0.

[13] D. P. DiVincenzo. »The Physical Implementation of Quantum Computation«. *Fortschritte der Physik* 48.9-11 (Sep. 2000), S. 771–783. DOI: 10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e.

[14] T. Filk. *Quantenmechanik (nicht nur) für Lehramtsstudierende*. Berlin, Heidelberg: Springer Spektrum, 2019. DOI: 10.1007/978-3-662-59736-1.

[15] W. Scherer. *Mathematik der Quanteninformatik: Eine Einführung*. Berlin, Heidelberg: Springer Spektrum, 2016. DOI: 10.1007/978-3-662-49080-8.

[16] J. Yin et al. »Satellite-based entanglement distribution over 1200 kilometers«. *Science* 356.6343 (2017), S. 1140–1144. DOI: 10.1126/science.aan3211.

[17] W. Gehrke et al. *Digitaltechnik: Grundlagen, VHDL, FPGAs, Mikrocontroller*. 7. Aufl. Berlin, Heidelberg: Springer-Verlag, 2016. DOI: 10.1007/978-3-662-49731-9.

[18] A. Kumar. »Felix Bloch (1905-1983)«. *Resonance* 20.11 (2015), S. 956–958. DOI: 10.1007/s12045-015-0262-8.

[19] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. 2023. DOI: 10.5281/zenodo.2573505.

[20] A. Montanaro. »Quantum algorithms: an overview«. *npj Quantum Information* 2.1 (Jan. 2016). DOI: 10.1038/npjqi.2015.23.

[21] C. H. Bennett und G. Brassard. »Quantum cryptography: Public key distribution and coin tossing«. *Theoretical Computer Science* 560 (Dez. 2014), S. 7–11. DOI: 10.1016/j.tcs.2014.05.025.

[22] A. K. Ekert. »Quantum cryptography based on Bell's theorem«. *Phys. Rev. Lett.* 67 (6 Aug. 1991), S. 661–663. DOI: 10.1103/PhysRevLett.67.661.

[23] P. Shor. »Algorithms for quantum computation: discrete logarithms and factoring«. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, S. 124–134. DOI: 10.1109/SFCS.1994.365700.

[24] I. M. Georgescu, S. Ashhab und F. Nori. »Quantum simulation«. *Reviews of Modern Physics* 86.1 (März 2014), S. 153–185. DOI: 10.1103/revmodphys.86.153.

[25] H. Mustafa et al. *Variational Quantum Algorithms for Chemical Simulation and Drug Discovery*. 2022. arXiv: 2211.07854 [quant-ph].

[26] S. McArdle et al. »Quantum computational chemistry«. *Reviews of Modern Physics* 92.1 (März 2020). DOI: 10.1103/revmodphys.92.015003.

[27] R. P. Feynman. »Simulating physics with computers«. *International Journal of Theoretical Physiscs* 21 (1982), S. 467–488. DOI: 10.1007/BF02650179.

[28] J. Biamonte et al. »Quantum machine learning«. *Nature* 549.7671 (Sep. 2017), S. 195–202. DOI: 10.1038/nature23474.

[29] E. Farhi, J. Goldstone und S. Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. DOI: 10.48550/arXiv.1411.4028.

[30] H. Müller und F. Weichert. *Vorkurs Informatik: Der Einstieg ins Informatikstudium*. 6. Aufl. Wiesbaden: Springer Vieweg, 2023. DOI: 10.1007/978-3-658-36468-7.

[31] D. Deutsch und R. Jozsa. »Rapid solution of problems by quantum computation«. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), S. 553–558. DOI: 10.1098/rspa.1992.0167.

[32] L. K. Grover. »Quantum Mechanics Helps in Searching for a Needle in a Haystack«. *Phys. Rev. Lett.* 79 (2 Juli 1997), S. 325–328. DOI: 10.1103/PhysRevLett.79.325.

[33] L. K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. DOI: 10.48550/arXiv.quant-ph/9605043.

[34] S. Kaiser und C. Granade. *Learn Quantum Computing with Python and Q#: A hands-on approach*. Shelter Island, NY: Manning Publications Co., 2021. ISBN: 978-1-61729-613-0.

[35] O. Natt. *Physik mit Python: Simulationen, Visualisierungen und Animationen von Anfang an*. Berlin, Heidelberg: Springer Spektrum, 2022. DOI: 10.1007/978-3-662-66454-4.

[36] S. Ganguly. *Quantum Machine Learning: An Applied Approach: The Theory and Application of Quantum Machine Learning in Science and Industry*. Berkeley, CA: Apress, 2021. DOI: 10.1007/978-1-4842-7098-1.

[37] R. Wille, R. Van Meter und Y. Naveh. »IBM's Qiskit Tool Chain: Working with and Developing for Real Quantum Computers«. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, S. 1234–1240. DOI: 10.23919/DATE.2019.8715261.

[38] J. M. Stowasser, M. Petschenig und F. Skutsch. *STOWASSER: Lateinisch-deutsches Schulwörterbuch*. Hrsg. von F. Losek. Bearbeiteter und erweiterter Nachdr. München, Düsseldorf, Stuttgart: Oldenbourg Schulbuchverlag GmbH, 2008. ISBN: 978-3-486-13405-6.

[39] D. Koch, L. Wessing und P. M. Alsing. *Introduction to Coding Quantum Algorithms: A Tutorial Series Using Qiskit*. 2019. arXiv: 1903.04359 [quant-ph].

[40] F. Pérez und B. E. Granger. »IPython: a System for Interactive Scientific Computing«. *Computing in Science and Engineering* 9.3 (Mai 2007), S. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53.

[41] Qiskit Development Team. *Qiskit 0.44 documentation - API Reference - Quantum Circuit Extensions - Initialize*. IBM. URL: https://qiskit.org/documentation/stubs/qiskit.extensions.Initialize.html (besucht am 03. 09. 2023). Zuletzt aktualisiert: 25.08.2023. Internet Archiv vom 07.06.2023 (Zuletzt aktualisiert: 05.06.2023): https://web.archive.org/web/20230607114421/https://qiskit.org/documentation/stubs/qiskit.extensions.Initialize.html.

[42] F. Harkins. *qiskit-textbook*. URL: https://github.com/qiskit-community/qiskit-textbook/blob/main/qiskit-textbook-src/qiskit_textbook/widgets/__init__.py (besucht am 03. 09. 2023). Zuletzt aktualisiert: 30.05.2023.

[43] Qiskit Development Team. *Qiskit 0.44 documentation - API Reference - Visualizations*. IBM. URL: https://qiskit.org/documentation/apidoc/visualization.html (besucht am 03. 09. 2023). Zuletzt aktualisiert: 25.08.2023. Internet Archiv vom 07.06.2023 (Zuletzt aktualisiert: 24.03.2023): https://web.archive.org/web/20230324000234/https://qiskit.org/documentation/apidoc/visualization.html.

[44] Qiskit Development Team. *Qiskit 0.44 documentation - API Reference - Quantum Circuits - Quantum Circuit*. IBM. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html#qiskit.circuit.QuantumCircuit.measure_all (besucht am 03. 09. 2023). Zuletzt aktualisiert: 25.08.2023. Internet Archiv vom 07.06.2023 (Zuletzt aktualisiert: 18.03.2023): https://web.archive.org/web/20230318073416/https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.measure_all.html#qiskit.circuit.QuantumCircuit.measure_all.

[45] J. Wang, L. Li und A. Zeller. *Better Code, Better Sharing:On the Need of Analyzing Jupyter Notebooks*. 2019. arXiv: 1906.05234 [cs.SE].

[46] M. Choetkiertikul et al. *Mining the Characteristics of Jupyter Notebooks in Data Science Projects*. 2023. arXiv: 2304.05325 [cs.SE].

[47] B. M. Boscoe et al. *Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study*. 2018. arXiv: 1804.05492 [cs.DL].

[48] E. Bisong. »Google Colaboratory«. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, S. 59–64. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_7.

[49] Project Jupyter. *Jupyter Widgets*. URL: https://ipywidgets.readthedocs.io/en/latest/index.html (besucht am 04. 09. 2023). Internet Archiv vom 27.07.2023: https://web.archive.org/web/20230727174632/https://ipywidgets.readthedocs.io/en/latest/index.html.

[50] Python Software Foundation. *random — Generate pseudo-random numbers*. URL: https://docs.python.org/3/library/random.html (besucht am 04. 09. 2023). Internet Archiv vom 03.09.2023: https://web.archive.org/web/20230903195057/https://docs.python.org/3/library/random.html.

[51] Project Jupyter. *Widget List: Selection widgets: RadioButtons*. URL: https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html#radiobuttons (besucht am 04.09.2023). Internet Archiv vom 27.07.2023 (Zuletzt aktualisiert: 07.08.2023): https://web.archive.org/web/20230807044239/https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html#radiobuttons.

[52] Project Jupyter. *Widget List: Button*. URL: https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html#button (besucht am 04.09.2023). Internet Archiv vom 27.07.2023 (Zuletzt aktualisiert: 07.08.2023): https://web.archive.org/web/20230807044239/https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html#button.

[53] Project Jupyter. *Output widgets: leveraging Jupyter's display system*. URL: https://ipywidgets.readthedocs.io/en/latest/examples/Output%20Widget.html (besucht am 04.09.2023). Internet Archiv vom 27.07.2023 (Zuletzt aktualisiert: 14.08.2023): https://web.archive.org/web/20230814040404/https://ipywidgets.readthedocs.io/en/latest/examples/Output%20Widget.html.

[54] Project Jupyter. *Widget List: Numeric widgets: IntSlider*. URL: https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html#intslider (besucht am 04.09.2023). Internet Archiv vom 27.07.2023 (Zuletzt aktualisiert: 07.08.2023): https://web.archive.org/web/20230807044239/https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html#intslider.

[55] J. Weinberg. »Didaktische Reduktion und Rekonstruktion«. In: *Didaktische Dimensionen der Erwachsenenbildung*. Hrsg. von H. Tietgens. Frankfurt (Main): Pädagogische Arbeitsstelle des DVV, 1992, S. 130–150. ISBN: 9783885135661.

[56] N. Kultusministerium, Hrsg. *Mathematik: Kerncurriculum für das Gymnasium – gymnasiale Oberstufe die Gesamtschule – gymnasiale Oberstufe das Berufliche Gymnasium das Abendgymnasium das Kolleg*. 2022. URL: https://cuvo.nibis.de/cuvo.php?p=detail_view&docid=1647&k0_0=Schulbereich&v0_0=Sek%5C%20II&k0_1=Schulform%5C&v0_1=Gymnasiale%5C%20Oberstufe&k0_2=Fach%5C&v0_2=Mathematik (besucht am 06. 09. 2023).

[57] B. Fauth und T. Leuders. *Kognitive Aktivierung im Unterricht*. Hrsg. von L. f. S. Alexandra Dehmel. Stuttgart: Landesinstitut für Schulentwicklung, 2018. ISBN: 978-3-944346-28-1.

[58] F. I. Craik und R. S. Lockhart. »Levels of processing: A framework for memory research«. *Journal of Verbal Learning and Verbal Behavior* 11.6 (1972), S. 671–684. ISSN: 0022-5371. DOI: https://doi.org/10.1016/S0022-5371(72)80001-X.

[59] K.-U. Götzelt und M. Schertler. »Bedarfsorientierte Wissensvermittlung durch Kontextualisierung von Lernobjekten«. ger. In: *Auf zu neuen Ufern! E-Learning heute und morgen*. 2005, S. 77–86. ISBN: 3-8309-1557-8. DOI: https://doi.org/10.25656/01:11749.

[60] R. J. Gerrig. *Psychologie*. Hrsg. von T. Dörfler und J. Ross. 21. Auflage. Hallbergmoos: Pearson Deutschland GmbH, 2018. ISBN: 978-3-86894-323-8.

[61] P. Hubwieser. *Didaktik der Informatik: Grundlagen, Konzepte, Beispiele*. 3. Aufl. Berlin, Heidelberg: Springer-Verlag, 2007. DOI: 10.1007/978-3-540-72478-0.

[62] P. Murphy et al. »Examining the Effects of Classroom Discussion on Students' Comprehension of Text: A Meta-Analysis«. *Journal of Educational Psychology* 101 (Aug. 2009), S. 740–764. DOI: 10.1037/a0015576.

[63] S. Schneider und P. Hitzig. *Das Business-Gedächtnistraining*. Hannover: Humboldt, eine Marke der Schlüterschen Verlagsgesellschaft, 2011. ISBN: 978-3-86910-767-7.

[64] H. L. Roediger und J. D. Karpicke. »The Power of Testing Memory: Basic Research and Implications for Educational Practice«. *Perspectives on Psychological Science* 1 (2006). PMID: 26151629, S. 181–210. DOI: 10.1111/j.1745-6916.2006.00012.x.

[65] C. Yang et al. »Testing (quizzing) boosts classroom learning: A systematic and meta-analytic review.« *Psychological bulletin* 147.4 (2021), S. 399–435. DOI: 10.1037/bul0000309.

[66] F. Sana und V. X. Yan. »Interleaving Retrieval Practice Promotes Science Learning«. *Psychological Science* 33.5 (2022). PMID: 35436145, S. 782–788. DOI: 10.1177/09567976211057507.

[67] T. Havener und M. Spitzbart. *Denken Sie nicht an einen blauen Elefenaten! Die Macht der Gedanken*. Reinbek: Rowohlt Taschenbuch Verlag, 2010. ISBN: 978-3-499-62609-8.

A Anhang

A.1 Implementierung ausgewählter Quantengatter in Qiskit

Tabelle A.1.: Quantengatter in Qiskit

Quantengatter	Funktion in Qiskit
X-Gatter	`qc.x(Qubit)`
Y-Gatter	`qc.y(Qubit)`
Z-Gatter	`qc.z(Qubit)`
Hadamard-Gatter	`qc.h(Qubit)`
CNOT-Gatter	`qc.cx(Kontrollqubit, Zielqubit)`
Toffoli-Gatter	`qc.ccx(Kontrollqubit 1, Kontrollqubit 2, Zielqubit)`

A.2 Jupyter Notebook

In dem nachfolgenden Anhang ist das in Kapitel 4 diskutierte Jupyter Notebook vollständig angeführt. Das Jupyter Notebook kann darüber hinaus über den folgenden Link heruntergeladen werden und ist, wenn in dem verwendetem Browser ein Google Account angemeldet ist, nach Bestätigung über den Link direkt in Google Colab einsehbar: `https://drive.google.com/file/d/1qLjkngcvp3HmLeS8YAs8CGODs_73yloe/view?usp=sharing`

Quantenmechanische Grundlagen für Quantencomputing

Einführung Jupyter Notebook

Wir führen zunächst gemeinsam die nachfolgenden Code Zellen aus.

In [1]:

```
!pip install qiskit
!pip install git+https://github.com/qiskit-community/qiskit-textbook.git#subdirectory=qiskit-textbook-src
!pip install pylatexenc
!pip install qiskit-aer

# used mathematical functions
import numpy as np
from numpy import pi
from math import sqrt, pi
from pylatexenc import *

# Importing standard Qiskit libraries
from qiskit import *
from qiskit.quantum_info import Statevector
from qiskit.visualization import plot_bloch_multivector, plot_histogram, plot_bloch_vector

# jupyter notbook interactive
import ipywidgets as widgets
from IPython.display import display
import random
```

...

In [2]:

```
# Funktion erstellen, die Single-Choice Fragen ausgibt
def sc_widget(frage, falschemoeglichkeiten, richtige_antwort):
    # Übergebene Antworten zusammenfügen und durchmischen
    antworten_text = [richtige_antwort] + falschemoeglichkeiten
    random.shuffle(antworten_text)

    # Antwortmöglichkeiten der Buttons für Single-Choice übergeben und Fenstergröße für Widget definieren
    antwortmoeglichkeiten = widgets.RadioButtons(options = antworten_text,
                                        description = '',
                                        disabled = False,
                                        style= {'description_width': 'initial'},
                                        align_items='stretch',
                                        layout = widgets.Layout(width='750px', height='auto')
                                        )

    # Button zur Abgabe der Antwort definieren
    abgabeButton = widgets.Button(description='einloggen',
                                  disabled=False,
                                  button_style=''
                                  )

    # Output Instanz mit Widget erstellen und an Variable übergeben
    rueckmeldung_ausgeben = widgets.Output()

    # Antwortmöglichkeiten als Radiobutton und Abgabebutton bei Display der Funktion sc_widget ausgeben
    def ausgeben(text):
      with rueckmeldung_ausgeben:
        display(antwortmoeglichkeiten)
        display(abgabeButton)
        print(str(text))

    # Funktion definieren, die die Antwort überprüft und eine Rückmeldung ausgibt
    def ueberpruefen(index):
      with rueckmeldung_ausgeben:
          rueckmeldung_ausgeben.clear_output()
          ausgabetext = "\x1b[6;37;42m Korrekt " if antwortmoeglichkeiten.value == richtige_antwort else "\x1b[5;
          ausgeben(ausgabetext)                 # else "\x1b[5;37;41m Falsch "
                                                # Der obige Kommentar ist für die Lesbarkeit beim Export erstellt.
    # Funktionen aufrufen
    ausgeben('')
    abgabeButton.on_click(ueberpruefen)

    # Box erstellen, um Widget im Ganzen auszugeben
    return widgets.VBox([widgets.Label(value=frage), rueckmeldung_ausgeben])


###################################### Single-Choice Fragen erstellen ######################################

testfrage_options = ['TU Braunschweig','Universität Hildesheim']
Testfrage = sc_widget('Von welcher Universität kommen wir?', testfrage_options,'Leibniz Universität Hannover')

# Bit vs. Qubit
bit1 = sc_widget('In welchem Zustand kann sich ein Bit nicht befinden? Warum?',['0','1'],'0.5')
bit2 = sc_widget('Wie viele verschiedene Zustände kann ein Computer mit 3 Bits annehmen?',
                 ['2','4','12','16','18','32'],'8')

Q1 = sc_widget('Wie lautet α, wenn |Ψ> = |1>',['1','0.5'],'0')
Q2 = sc_widget('Wie viele verschiedene Zustände kann ein Qubit annehmen?',['2','1','4'],
               'nahezu unendlich')
Q4 = sc_widget('Kann ein Bit oder ein Qubit mehr verschiedene Zustände annehmen und warum?',['Bit'],
               'Qubit')

# Logische Gatter vs. Quantengatter
quantumgates1_options = ['AND', 'OR','XOR','Zu keinem']
quantumgates1 = sc_widget('Zu welchem klassischen Logikgatter ist das X-Gatter analog?',
                          quantumgates1_options,'NOT')

quantumgates2_options = ['AND', 'OR','NOT','Zu keinem']
quantumgates2 = sc_widget('Zu welchen klassischen Logikgatter ist das CNOT-Gatter analog?',
                          quantumgates2_options, 'XOR')

quantumgates3_options = ['AND', 'OR','XOR','NOT']
quantumgates3 = sc_widget('Zu welchen klassischen Logikgatter ist das Hadamard-Gatter analog?',
                          quantumgates3_options, 'Zu keinem')

# Die Blochkugel
bloch1 = sc_widget("""In welchem Zustand befindet sich ein Qubit (vorher präpariert in |1>), wenn kein Gatter
                      angewendet wird?""",['|0>', 'in einer Überlagerung aus |0> und |1>'],'|1>')
bloch2 = sc_widget("""In welchem Zustand befindet sich ein Qubit (vorher präpariert in |1>), wenn das X-Gatter
                      angewendet wird?""",['|1>', 'in einer Überlagerung aus |0> und |1>'],'|0>')
bloch3 = sc_widget("""In welchem Zustand befindet sich ein Default-Qubit, wenn kein Gatter angewendet wird?""",
                      ['|1>', 'in einer Überlagerung aus |0> und |1>'],'|0>')
bloch4 = sc_widget('In welchem Zustand befindet sich ein Default-Qubit, wenn das X-Gatter angewendet wird?',
                      ['|0>', 'in einer Überlagerung aus |0> und |1>'],'|1>')
bloch5 = sc_widget('In welchem Zustand befindet sich ein Default-Qubit, wenn das Hadamard-Gatter angewendet wird?
                    ['|0>','|1>'],'in einer Überlagerung aus |0> und |1>')
bloch6 = sc_widget("""In welchem Zustand befindet sich ein Default-Qubit, wenn zunächst das X-Gatter und dann das
                      Hadamard-Gatter angewendet wird?""",['|0>','|1>'],
                      'in einer Überlagerung aus |0> und |1>')
bloch7 = sc_widget("""In welchem Zustand befindet sich ein Default-Qubit, wenn das Hadamard-Gatter  und dann das
```

```
                    X-Gatter angewendet wird?""",['|0>','|1>'],'in einer Überlagerung aus |0> und |1>')

# Quantenschaltkreise
quantumcircuit1 = sc_widget('Wie viele Qubits lassen sich (theoretisch) zu einem Quantenschaltkreis kombinieren?',
                            ['2','4','8','16','32'], 'so viele, wie benötigt werden')
quantumcircuit2 = sc_widget('Was ist jeweils der Default-Zustand der Eingangsqubits?',
                            ['|1>', 'eine Überlagerung aus |0> und |1>'],'|0>')


# Wahrscheinlichkeiten
probability1 = sc_widget("""Welcher Zustand wird gemessen, wenn vorher ein X-Gatter auf ein Qubit in |0> angewende
                         wird?""", ['|0>', 'eine Überlagerung aus |0> und |1>'],'|1>')
probability2 = sc_widget('Welcher Zustand wird gemessen?',['|000>','|111>', '|100>',
                         '|001>','|110>','|101>', '|011>','jeweils eine Überlagerung aus |0> und |1>'],'|010>')
probability3 = sc_widget('Wofür stehen α und β in |Ψ> =  α |0> + β |1>?',["""Geben die Wahrscheinlichkeit an, mit
                          der sich das Qubit in dem nachfolgenden Zustand befindet."""],
                         """Das Betragsquadrat gibt die Wahrscheinlichkeit an, mit der sich das Qubit in dem
                         nachfolgenden Zustand befindet.""")

probability4 = sc_widget('Was liefert die Quantenmechanik für Informationen über Experimente?',
                         ['Aussagen, was ein Experiment eindeutig ergibt.',
                         'Aussage über Eingangszustand eines \n beliebigen Qubits'],
                         'Vorhersagen in Form von Wahrscheinlichkeiten')

# Diskussionsfragen

answer1 = 'Antwort auf die Diskussionsfrage\nα und β sind jeweils die Gewichtung der einzelnen Zustände. Aus den
answer2 = """Antwort auf die Diskussionsfrage\nFolgende Gatter Kombination ist äquivalent zu dem X-Gatter: H-Y-H.
```

Anschließend wird in den nachfolgenden Codezellen der Umgang mit Jupyter Notebooks kurz exemplarisch dargestellt.

In [3]:

```
# mit einem Hashtag sind Kommentare gekennzeichnet, sie werden nicht ausgeführt


# Der folgende Code gibt die enthaltene Zeichenfolge, einen sogenannten String (engl. Zeichenkette) aus.
# Tippen Sie auf den Button oben links in der Ecke der Zelle mit dem Startsymbol, um die Zelle auszuführen
# Wenn kein Button zu sehen ist, sondern eine Zahl in eckigen Klammer, bspw. [3], dann tippen Sie auf diese.

print("Hallo Welt!")
```

```
Hallo Welt!
```

In dem Notebook werden Ihnen häufig One Choice Fragen begegnen. Diese dienen zum einen als Zusammenfassung der wichtigsten Inhalte eines Abschnitts, zum anderen der persönlichen Überprüfung Ihres Verständnisses.

In [4]:

```
# <- Bitte Zelle links ausführen
display(Testfrage)

```

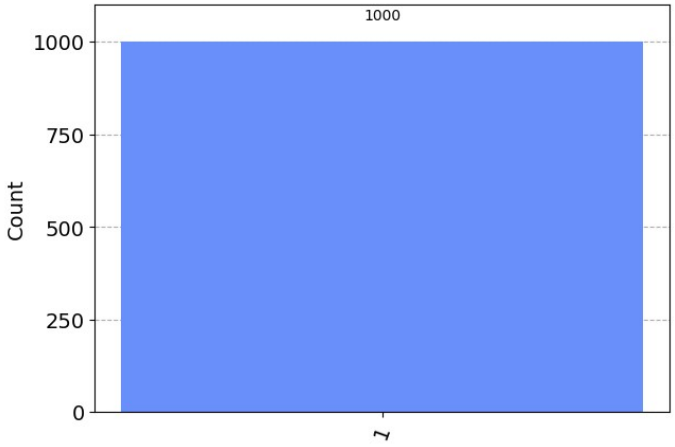
Von welcher Universität kommen wir?

- ○ Universität Hildesheim
- ○ TU Braunschweig
- ◉ Leibniz Universität Hannover

einloggen

```
Korrekt
```

Das Histogramm wird in dem Notebook häufig vorkommen. Es zeigt eine Messergebnis von 1000 Messungen. Das Ergebnis ist $|1\rangle$.



Zum Abschluss betrachten wir noch das Slider-Widget, was gegen Ende des Notebooks genutzt wird.

In [5]:
```
# <- Bitte Zelle links ausführen
slider = widgets.IntSlider(
    min=0,
    max=35,
    step=1,
    description='',
    value=17,
    layout = widgets.Layout(min_width='650px', height='50px')
)

print('Wie warm ist es heute?')
display(slider)
```

```
Wie warm ist es heute?
```

17

In [6]:
```
# <- Bitte Zelle links ausführen
temperature = slider.value
print("Die Temperatur beträgt heute ungefähr " + str(temperature) +"°C.")
```

```
Die Temperatur beträgt heute ungefähr 17°C.
```

Jede Codezelle muss für die volle Funktionalität des Notebooks ausgeführt werden. Kommen Sie bei Errors, Unklarheiten oder sonstigen Fragen jederzeit gerne auf uns zu.

Bit vs. Qubit

Dass *Nullen* und *Einsen* etwas mit klassischen Computern zu tun haben, haben viele Menschen bereits einmal gehört. Teilweise ist sogar das Wort *Bit* schon ein Begriff. In der Einleitung haben Sie bereits erfahren, dass die klassische Welt gegenüber der Quantenwelt grundverschieden ist. Dementsprechend unterscheiden sich Bits und Qubits ebenfalls, was mithilfe eines Lichtschalters und eines Dimmers veranschaulicht werden kann.

Bits oder auch `01000010 01101001 01110100 01110011`

Informationen werden auf einem klassischen Computer in Bits (engl.: "binary digit") gespeichert. Bits befinden sich eindeutig in einem von zwei Zuständen: `0` oder `1`. Zustände wie `0.2`, `0.5` oder `0.7` sind nicht möglich. Es sind also zwei Zustände pro Bit möglich, mit deren Kombination Dezimalzahlen oder auch wie in der Überschrift Buchstaben kodiert und gespeichert werden können. `0` und `1` können als Alphabet eines klassischen Computers assoziiert werden. Der Defaultzustand, also der normale Zustand in dem sich ein unverändertes Bit befindet, ist `0`.

Somit sind Bits mit einem Lichtschalter vergleichbar. Ein Lichtschalter ist eindeutig so geschaltet, dass das Licht entweder an oder aus ist. Es gibt keinen Zwischenzustand, auch wenn viele als Kind versucht haben, einen Lichtschalter auszubalancieren. Selbst in Balance war das Licht entweder an oder aus.

Wenn wir 2 Bits betrachten, ergeben sich folgende Kombinationsmöglichkeiten, also Zustände: `00`, `01`, `10`, `11`, da jedes Bit jeweils entweder `0` oder `1` sein kann. `00`, `01` usw. sind der Gesamtzustand der beiden Bits im Verbund. Dabei sind die einzelnen Gesamtzustände ebenfalls eindeutig. Ein Zwischenzustand zwischen `00` und `10` ist nicht möglich.

In [7]:

```
# <- Bitte Zelle links ausführen
display(bit1)
display(bit2)
```

In welchem Zustand kann sich ein Bit nicht befinden? Warum?

- ○ 1
- ○ 0
- ◉ 0.5

einloggen

Korrekt

Wie viele verschiedene Zustände kann ein Computer mit 3 Bits annehmen?

- ○ 4
- ○ 32
- ○ 2
- ○ 18
- ◉ 8
- ○ 16
- ○ 12

einloggen

Korrekt

> Bits weisen vergleichbar mit einem Lichtschalter einen eindeutigen Zustand auf (`0` oder `1`) und kodieren die Informationen auf einem klassischen Computer. Auch Bit-Kombinationen haben einen eindeutigen Zustand (entweder `00` **oder** `01` **oder** `10` **oder** `11`).

Quantenbits und Zustände

Das quantenmechanische Analogon zu den klassischen Bits sind die Qubits - Quantenbits. Auch ein Qubit hat einen Zustand, wobei ohne Überlagerung erneut zwei Zustände möglich sind, die als Kombinationszustände verstanden werden können. Ein Qubit setzt sich aus einer Kombination dieser Zustände zusammen. Die möglichen Zustände werden in der Quantenmechanik beispielsweise mit $|0\rangle$ und $|1\rangle$ bezeichnet. Die ungewöhnlich aussehende Notierung heißt Dirac-Notation und lässt sich leicht verstehen. $|0\rangle$ ist ein Vektor, der "Ket-Vektor Null" bzw. kurz: "Ket Null" genannt wird. In diesem Fall ist $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ und $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

Wenn die Mathematik für einen Moment außer Acht wird, können $|0\rangle$und $|1\rangle$ schlichtweg als Namen für die Kombinationszustände verstanden werden. Dabei ist in Quantenschaltkreisen in der Regel $|0\rangle$ ("Ket Null") der Defaultzustand eines Qubits.

Ein klassisches Bit befindet sich entweder eindeutig in dem Zustand `0` oder dem zustand `1`. Es verhält sich wie bereits angesprochen wie ein Lichtschalter. Im Gegensatz dazu kann ein Qubit mit einem Dimmer verglichen werden. Dementsprechend ist das Licht nicht immer zu 100 % aus oder 100% an, sondern kann beispielsweise 50 % "aus" und 50 % "an" sein. D. h., ein Qubit ist nicht zwingend in dem Zustand $|0\rangle$ (Licht ist aus) oder in Zustand $|1\rangle$ (Licht aus).

Ein Qubit befindet sich in einer Kombination, einer Überlagerung der beiden Kombinationszustände $|0\rangle$ und $|1\rangle$. So kann das Qubit zum Beispiel lauten:

Qubit = 70 % $|0\rangle$ und 30 % $|1\rangle$. Die Kombination der beiden Zustände für ein Qubit wird als Formel folgendermaßen ausgedrückt (Ψ ist der griechische Buchstabe Psi):

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$|\Psi\rangle$ ("Ket Psi") ist der Zustand Psi, der das Qubit beschreibt.

$|\Psi\rangle$ wird dabei auch als Zustandsvektor bezeichnet, weil es den Zustand des Qubits in Form eines Vektors angibt. Mathematisch ausgedrückt ist ein Qubit somit eine Linearkombination der einzelnen Kombinationszustände. Dies wird auch **Superposition** genannt. Diskutieren Sie kurz (max. 1 Minute) in der Gruppe, wofür α und β stehen könnten, bevor Sie weiterlesen und -arbeiten.

In [8]:
```
# <- Bitte Zelle links ausführen
print(answer1)
print()
display(Q1)
display(Q2)
```

```
Antwort auf die Diskussionsfrage
α und β sind jeweils die Gewichtung der einzelnen Zustände. Aus den Gewichtungen lässt sich bestimmen, zu welchen
Anteilen sich das Qubit in dem nachfolgenden Zustand befindet und mit welcher Wahrscheinlichkeit es in dem Zustand
gemessen wird. In der oben aufgeführten Formel enthält α Informationen zu |0> und β zu |1>. Darauf kommen wir später
noch genauer zurück. Die Superposition ist somit eine gewichtete Addition der Kombinationszustände.
```

Wie lautet α, wenn |Ψ> = |1>

- (•) 0
- ( ) 1
- ( ) 0.5

einloggen

```
Korrekt
```

Wie viele verschiedene Zustände kann ein Qubit annehmen?

- ( ) 2
- ( ) 1
- ( ) 4
- (•) nahezu unendlich

einloggen

```
Korrekt
```

> Ein Qubit hat nicht nur zwei eindeutige Zustände, sondern setzt sich aus einer Kombination bzw. Überlagerung zweier Zustände zusammen ($|0\rangle$, $|1\rangle$). Die Kombination hat eine vergleichbare Wirkung wie ein Dimmer und wird *Superposition* genannt.

Logische Gatter vs. Quantengatter

klassische Logikgatter

In einem klassichen Computer führen klassische Logikgatter logische Operationen auf bereitgestellte Informationen aus. Als einzelne Ein- und Ausgänge sind die Zustände `0` und `1` möglich. Ein Gatter kombiniert oder vergleicht die Eingänge und gibt der Funktion des Gatters entsprechend das Ergebnis an den Ausgang weiter. Gatter werden in der Regel mit Transistoren realisiert. Transistoren sind elektronische Halbleiter-Bauelemente. Dabei stehen für `0` und `1` verschiedene Spannungswerte.

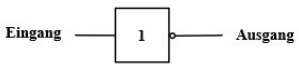
Um beispielsweise auf einem Computer 2 + 2 zu rechnen, werden viele Transistoren angesteuert. Diese führen einzelne Operationen durch, die in Kombination das Ergebnis liefern. Die übergeordnete Rechnung wird von vielen Bauteilen realisiert, die jeweils kleine Einzelrechnungen/ -operationen durchführen. Die klassischen Logikgatter sind vergleichbar mit den Operatoren aus der Mathematik (+, −, ×, ÷). Im Folgenden werden verschiedene klassische Logikgatter vorgestellt.

NOT-Gatter

Das simpelste Gatter ist das NOT-Gatter, da es nur einen Eingang aufweist. Es ist im Prinzip eine logische Negierung bzw. Invertierung, also eine Umkehrung des am Eingang anliegenden Bits:

Eingang	Ausgang
0	1
1	0

Die Schaltskizze in einem Schaltkreis sieht schematisch wie folgt aus:

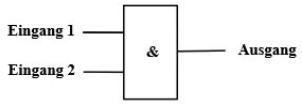


AND-Gatter

Das AND-Gatter hat zwei Eingänge und einen Ausgang. Es ist ein Vergleich der Eingänge. Der Ausgang ist nur `1`, wenn auch an beiden Eingängen `1` anliegt.

Eingang 1	Eingang 2	Ausgang
0	0	0
0	1	0
1	0	0
1	1	1

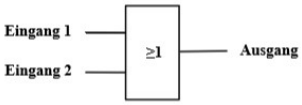
Eingang 1 **AND** Eingang 2 müssen somit 1 sein, damit am Ausgang 1 anliegt. In einem Schaltkreis wird das AND-Gatter folgendermaßen dargestellt:



OR-Gatter

Die Funktion des OR-Gatters ergibt sich wie auch bei dem AND-Gatter aus dem Namen. Am Ausgang liegt eine 1 an, wenn an Eingang 1 oder Eingang 2 eine 1 anliegt. Dabei handelt es sich um ein "Oder" im Sinne von "Eingang 1 oder/ und Eingang 2", nicht um ein "entweder Eingang 1 oder Eingang 2". Der Ausgang ist somit auch 1 , wenn an beiden Eingängen eine 1 anliegt.

Eingang 1	Eingang 2	Ausgang
0	0	0
0	1	1
1	0	1
1	1	1



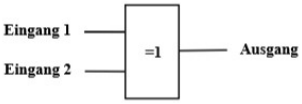
Die Abbildung zeigt die Darstellung des OR-Gatters in einem Schaltkreis

XOR-Gatter

XOR steht für e**X**klusiv OR. Dementsprechend erfüllt das XOR-Gatter die Funktion, dass entweder Eingang 1 **oder** Eingang 2 1 ist, wobei nur einer der Eingänge eXklusiv 1 sein darf, damit der Ausgang 1 ist. Es ergibt sich folgende logische Tabelle:

Eingang 1	Eingang 2	Ausgang
0	0	0
0	1	1
1	0	1
1	1	0

Das XOR-Gatter wird in einer Schaltskizze ähnlich dem OR-Gatter skizziert.



Quantengatter

Quantengatter unterscheiden sich notwendigerweise vor allem in ihrer Realisierung von den klassischen Logikgatter. Klassische Logikgatter werden wie bereits angesprochen elektronisch mit Transistoren umgesetzt. Bei Quantengattern spielen vor allem quantenmechanische Theorien die zentrale Rolle, wobei mit Systemen und Wechselwirkungen und nicht mit physikalischen Bauteilen gearbeitet wird.

Grundlegend entspricht die Funktion der Quantengatter der Funktion der klassischen Logikgatter. Quantengatter führen Operationen auf die eingehenden Qubits durch, um übergeordnete Berechnungen zu realisieren. Es gibt sogar ein paar Quantengatter, die in ihrer Funktion analog zu einem Teil der vorgestellten klassischen Logikgatter sind. Diese und ein weiteres werden im Folgenden dargestellt.

X-Gatter

Das X-Gatter ist vergleichbar mit dem NOT-Gatter. Dementsprechend sorgt es für einen Wechsel des Zustands eines Qubits.

Eingang	Ausgang
$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$

$|0\rangle$ ("Ket Null") wird mithilfe des X-Gatters somit zu $|1\rangle$ und umgekehrt. Wenn wir ein Qubit haben, was sich in einem "Dimmer-Zustand" (nicht eindeutig $|0\rangle$ oder $|1\rangle$) befindet, passiert Folgendes:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \xrightarrow{X\text{-}Gatter\ anwenden} |\tilde{\Psi}\rangle = \alpha|1\rangle + \beta|0\rangle$$

Kurz gesagt werden die Gewichtungen vor $|0\rangle$ und $|1\rangle$ vertauscht. Bei genauer Überlegung geschieht das gleiche, wenn wir ein Qubit in Zustand $|0\rangle$ haben und dieses unter Anwendung des X-Gatters zu $|1\rangle$ wird. Es gilt nämlich:

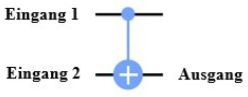
$$|\Psi\rangle = |0\rangle = 1 \cdot |0\rangle + 0 \cdot |1\rangle \xrightarrow{X\text{-}Gatter\ anwenden} |\tilde{\Psi}\rangle = 1 \cdot |1\rangle + 0 \cdot |0\rangle = |1\rangle$$

Das X-Gatter gehört zu den sogenannten Pauli-Gattern, die von dem Y- und Z-Gatter komplettiert werden. Diese haben jedoch kein direktes klassisches Analogon und werden daher an dieser Stelle nicht als Tabelle oder Gleichung dargestellt.

Abb. Darstellung des X-Gatters in einem Quantenschaltkreis

CNOT-Gatter

Das CNOT-Gatter fungiert als quantenmechanisches XOR-Gatter. Dementsprechend hat es zwei Eingänge $|Eingang1\rangle$ und $|Eingang2\rangle$. Diese werden meist in einer "Ket-Klammer" dargestellt: $|Eingang1\ Eingang2\rangle$. Bei dem CNOT-Gatter ist das zweite Eingangsqubit ebenfalls das Ausgangsqubit. Die nachfolgende Abbildung zeigt das CNOT-Gatter.



Eingänge	Ausgang
$\|00\rangle$	$\|0\rangle$
$\|01\rangle$	$\|1\rangle$
$\|10\rangle$	$\|1\rangle$
$\|11\rangle$	$\|0\rangle$

Die beiden Eingänge werden verglichen und der Zustand des Ausgangsqubits gegebenenfalls verändert.

Hadamard-Gatter

Für das Hadamard-Gatter gibt es kein vergleichbares klassisches Logikgatter. Da es jedoch häufig verwendet wird, wird es an dieser Stelle kurz vorgestellt. Das Hadamard-Gatter hat einen Eingang und einen Ausgang. Dabei tauscht es nicht einfach $|0\rangle$ und $|1\rangle$ wie das X-Gatter, sondern verändert den Zustand des Qubits auf den ersten Blick etwas komplizierter (weiterführend wird die Zustandsveränderung im anschließenden Abschnitt thematisiert). Für das Eingangsqubit $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ ergibt sich:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \xrightarrow{Hadamard\ anwenden} |\tilde{\Psi}\rangle = \frac{\alpha+\beta}{\sqrt{2}}|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}|1\rangle.$$

Ein Hadamard-Gatter kann somit aus einem Qubit, das sich in einem eindeutigen Zustand befindet, ein Qubit generieren, das sich in Superposition befindet, weil die einzelnen Gewichtungen kombiniert werden $\left(\frac{\alpha \pm \beta}{\sqrt{2}}\right)$. Dafür betrachten wir ein Qubit in Zustand $|1\rangle$:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle = 0 \cdot |0\rangle + 1 \cdot |1\rangle = |1\rangle$$

Wenden wir nun das Hadamard-Gatter darauf an, ergibt sich:

$$\begin{aligned} |\Psi\rangle = 0 \cdot |0\rangle + 1 \cdot |1\rangle \xrightarrow{Hadamard\ anwenden} |\tilde{\Psi}\rangle &= \frac{0+1}{\sqrt{2}}|0\rangle + \frac{0-1}{\sqrt{2}}|1\rangle \\ &= \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle. \end{aligned}$$

Das Qubit ist nun nicht mehr eindeutig in Zustand $|1\rangle$, sondern befindet sich in einer Überlagerung der Zustände $|0\rangle$ und $|1\rangle$.

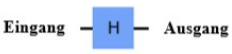


Abb. Darstellung des Hadamard-Gatters in einem Quantenschaltkreis

> Quantengatter sind in ihrer Funktion vergleichbar mit klassischen Logikgattern und führen ebenfalls Operationen auf die eingehenden Qubits aus. Dabei gibt es Ein-Qubit-Gatter ebenso wie Mehr-Qubit-Gatter. Mithilfe von Quantengattern können darüber hinaus Superpositionszustände erzeugt werden.

In [9]:

```
# <- Bitte Zelle links ausführen
display(quantumgates1)
print()
display(quantumgates2)
print()
display(quantumgates3)
```

Zu welchem klassischen Logikgatter ist das X-Gatter analog?

- ○ Zu keinem
- ○ AND
- ○ XOR
- ◉ NOT
- ○ OR

einloggen

Korrekt

Zu welchen klassischen Logikgatter ist das CNOT-Gatter analog?

- ○ Zu keinem
- ○ NOT
- ○ OR
- ○ AND
- ◉ XOR

einloggen

Korrekt

Zu welchen klassischen Logikgatter ist das Hadamard-Gatter analog?

- ○ OR
- ○ NOT
- ○ XOR
- ◉ Zu keinem
- ○ AND

einloggen

Korrekt

Zustände verändern - Quantenschaltkreise

Quantenschaltkreise mit einem Qubit

Die ersten beiden Abschnitte widmeten sich der Aufarbeitung von Qubits und Quantengattern. Für Quantencomputer werden diese beiden Konzepte kombiniert, analog zu der Kombination von Bits und klassischen Logikgatter. Klassische Logikgatter dienen dazu, Berechnungen auszuführen, gespeicherte Daten zu verändern u. Ä., also kurzum: Der Veränderung des Zustands eines Eingangsbits und Ausgabe des Ausgangsbits mit dem modifizierten Zustand (bzw. mehrere Eingangsbits zu einem oder mehrere Ausgangsbits).

Dieselbe Funktion erfüllen auch die Quantengatter für Qubits. Um die Veränderung des Zustands anschaulich darzustellen, wird die bereits eingeführte Darstellung der Bloch-Kugel genutzt.

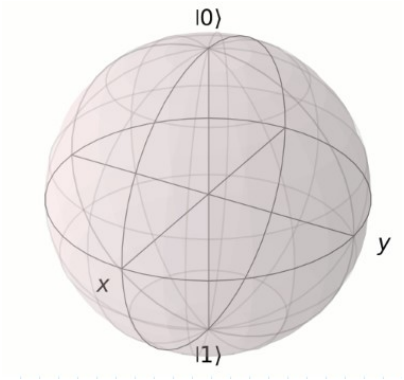
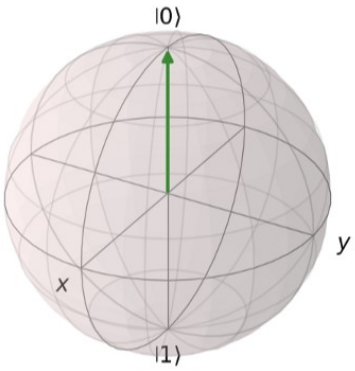


Abb.: Bloch-Kugel ohne Zustand

Wie bereits eingangs dargestellt, kann mithilfe der Bloch-Kugel der Zustand eines Qubits dargestellt werden. Dabei wird der Zustand mit einem Punkt auf der Kugeloberfläche, bzw. einem auf diesem Punkt endenden Pfeil dargestellt. Die Pfeilspitze zeigt an, in welchem Zustand sich das Qubit befindet. Die nachfolgende Abbildung zeigt Zustand $|0\rangle$.



Um nun den Zustand eines Qubits zu verändern, können die eingeführten Gatter verwendet werden. Dafür wird ein Quantenschaltkreis (engl. Quantum Circuit) konstruiert. Wenn wir beispielsweise auf das Qubit im Defaultzustand $|0\rangle$ das X-Gatter anwenden, sieht der Quantenschaltkreis schematisch wie folgt aus:

In [10]:
```
# <- Bitte Zelle links ausführen
qc1 = QuantumCircuit(1)
qc1.x(0)
qc1.draw("mpl")
```

Out[10]:

Auf das Eingangsqubit wird das X-Gatter angewendet. Das X-Gatter vertauscht die Gewichtungen vor $|0\rangle$ und $|1\rangle$ des Qubits. Wie muss also die Bloch-Kugel nach der Anwendung des X-Gatters auf ein Qubit, das sich zu Beginn in Zustand $|0\rangle$ befindet, aussehen? (Überlegen Sie es sich zunächst im Kopf, gleich können Sie es ausprobieren.)

Das analoge klassische NOT-Gatter sieht als Schaltskizze relativ ähnlich aus, wobei der kleine Kreis am Ausgang und die "1" in dem Kasten das NOT-Gatter kennzeichnen. Grundlegend sind die Darstellung von klassischen und quantenmechanischen Schaltkreise ähnlich (s. o.).

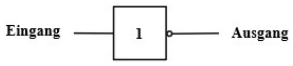


Abb. Schaltkreis NOT-Gatter

Wenn wir ein Gatter anwenden, können wir natürlich auch zwei, drei, vier, ... Gatter anwenden, bis der Quantenschaltkreis die geplante Funktion erfüllt

In [11]:
```
# <- Bitte Zelle links ausführen
print('Hier wird ein weiteres X-Gatter auf unser Qubit angewendet. In welchem Zustand befindet es sich jetzt? (Anf
qc2 = QuantumCircuit(1)
qc2.x(0)
qc2.x(0)
qc2.draw("mpl")
```

```
Hier wird ein weiteres X-Gatter auf unser Qubit angewendet. In welchem Zustand befindet es sich jetzt? (Anfangszustan
d: |0>)
```

Out[11]:

In [12]:
```
# <- Bitte Zelle links ausführen
print('In diesem Fall wenden wir ein Hadamard-Gattern nach den beiden X-Gatter auf das Qubit an. In welchem Zustan
qc3 = QuantumCircuit(1)
qc3.x(0)
qc3.x(0)
qc3.h(0)
qc3.draw("mpl")
```

```
In diesem Fall wenden wir ein Hadamard-Gattern nach den beiden X-Gatter auf das Qubit an. In welchem Zustand befindet
es sich jetzt? (Anfangszustand: |0>)
```

Out[12]:

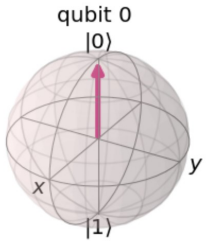
Mit dem Widget in der nächsten Codezelle können Sie sich die Anwendung unterschiedlicher Gatter veranschaulichen. Wenn Sie nacheinander auf die Gatter tippen, werden diese auch nacheinander auf das Qubit angewendet, wie in unserem Beispielschaltkreis. Um das Qubit wieder in den Anfangszustand $|0\rangle$ zu bringen, tippen Sie einfach auf *Reset*. Starten Sie am besten mit den oben dargestellten Schaltkreisen und überprüfen Sie Ihre Vermutungen.

Führen Sie die anschließende Zelle aus, damit das Widget funktioniert.

Hinweis: Wenn sich der Pfeil auf der Z-Achse befindet, also auf der senkrechten Achse, geschieht nichts, wenn das Z-Gatter angewendet wird. Dies lässt sich damit erklären, dass die X-, Y- und Z-Gatter jeweils eine Rotation um die X-, Y- bzw. Z-Achse verursachen. Wenn der Zustand bereits auf der Z-Achse liegt und dann um die Z-Achse rotiert wird, geschieht nichts Sichtbares.

In [13]:
```
# <- Bitte Zelle links ausführen
from qiskit_textbook.widgets import gate_demo
gate_demo(gates= 'pauli+h')
```

X Y Z H Reset



Mit welcher Kombination der Quantengatter kann das X-Gatter ersetzt werden? Also welche Gatter müssen nacheinander angewendet werden, um $|0\rangle$ und $|1\rangle$ zu vertauschen?

In [14]:

```
# <- Bitte Zelle links ausführen
print(answer2)
display(bloch1)
display(bloch2)
display(bloch3)
display(bloch4)
display(bloch5)
display(bloch6)
```

```
Antwort auf die Diskussionsfrage
Folgende Gatter Kombination ist äquivalent zu dem X-Gatter: H-Y-H.
```

In welchem Zustand befindet sich ein Qubit (vorher präpariert in |1>), wenn kein Gatter angewendet wird?

- ○ |0>
- ◉ |1>
- ○ in einer Überlagerung aus |0> und |1>

einloggen

Korrekt

In welchem Zustand befindet sich ein Qubit (vorher präpariert in |1>), wenn das X-Gatter angewendet wird?

- ○ in einer Überlagerung aus |0> und |1>
- ◉ |0>
- ○ |1>

einloggen

Korrekt

In welchem Zustand befindet sich ein Default-Qubit, wenn kein Gatter angewendet wird?

- ○ in einer Überlagerung aus |0> und |1>
- ◉ |0>
- ○ |1>

einloggen

Korrekt

In welchem Zustand befindet sich ein Default-Qubit, wenn das X-Gatter angewendet wird?

- ◉ |1>
- ○ in einer Überlagerung aus |0> und |1>
- ○ |0>

einloggen

Korrekt

In welchem Zustand befindet sich ein Default-Qubit, wenn das Hadamard-Gatter angewendet wird?

- ○ |0>
- ◉ in einer Überlagerung aus |0> und |1>
- ○ |1>

einloggen

Korrekt

In welchem Zustand befindet sich ein Default-Qubit, wenn zunächst das X-Gatter und dann das Hadamard-Gatter angewendet wird?

- ○ |1>
- ◉ in einer Überlagerung aus |0> und |1>
- ○ |0>

einloggen

Korrekt

Quantenschaltkreise mit mehreren Qubits

Wir haben bereits gesehen, dass auf ein Qubit mehrere Gatter angewendet werden können. Mit dem CNOT-Gatter ist ein Gatter bekannt, für das zwei Eingangsqubits benötigt werden. Dementsprechend können Quantenschaltkreise mit mehreren Qubits aufgebaut werden, analog zu einem klassischen Schaltkreis, der aus mehreren Bits besteht. Nur durch die Kombination mehrerer Qubits können vielfältige Funktionen mit einem

In [15]:
```
# <- Bitte Zelle links ausführen
print('Ein Quantenschaltkreis mit zwei Qubits und mit einem X-Gatter angewendet\nauf das erste Qubit kann wie folg
qc4 = QuantumCircuit(2)
qc4.x(0)
qc4.draw("mpl")
```

```
Ein Quantenschaltkreis mit zwei Qubits und mit einem X-Gatter angewendet
auf das erste Qubit kann wie folgt dargestellt werden.
```

Out[15]:

In [16]:
```
# <- Bitte Zelle links ausführen
print('Ein Quantenschaltkreis mit zwei Qubits und mit einem CNOT-Gatter kann wie folgt dargestellt werden.')
qc5 = QuantumCircuit(2)
qc5.cx(0,1)
qc5.draw("mpl")
```

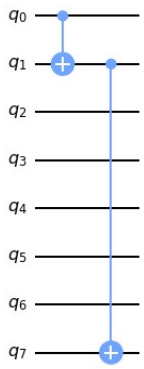
```
Ein Quantenschaltkreis mit zwei Qubits und mit einem CNOT-Gatter kann wie folgt dargestellt werden.
```

Out[16]:

In [17]:
```
# <- Bitte Zelle links ausführen
print('Ein beispielhafter Quantenschaltkreis mit 8 Qubits und 2 CNOT-Gattern.')
qc6 = QuantumCircuit(8)
qc6.cx(0,1)
qc6.cx(1,7)
qc6.draw("mpl")
```

```
Ein beispielhafter Quantenschaltkreis mit 8 Qubits und 2 CNOT-Gattern.
```

Out[17]:

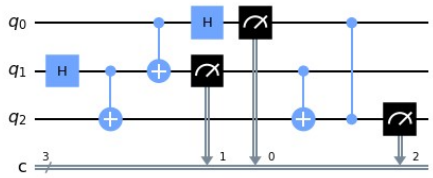


In [18]:

```
# <- Bitte Zelle links ausführen
print('Mit dieser Zelle wird ein funktionierender Quantenschaltkreis für Quantenteleportation programmiert und dar
qc7 = QuantumCircuit(3,3)
qc7.h(1)
qc7.cx(1,2)
qc7.cx(0,1)
qc7.h(0)
qc7.measure(1,1)
qc7.measure(0,0)
qc7.cx(1,2)
qc7.cz(0,2)
qc7.measure(2,2)
qc7.draw("mpl")
```

```
Mit dieser Zelle wird ein funktionierender Quantenschaltkreis für Quantenteleportation programmiert und dargestellt.
Dies ist ein interessantes Verfahren, wofür uns heute leider die Grundlagen und die Zeit fehlen.
(Quelle: Liu, Chang. (2020). Reverse Checking of Quantum Algorithm Execution. IEEE Access. PP. 1-1. 10.1109/ACCESS.20
20.3043187. ⑦)
```

Out[18]:



Der abgebildete Quantenschaltkreis für Quantenteleportation sieht zunächst einmal erschlagend aus. Jedoch besteht er fast ausschließlich aus Komponenten die wir bereits kennen. Lediglich der weiße Bogen mit weißem Strich in dem schwarzen Quadrat ist uns noch nicht bekannt. Diese Komponente wird im kommenden Abschnitt vorgestellt. Was könnte das sein?

In [19]:

```
# <- Bitte Zelle links ausführen
display(quantumcircuit1)
print()
```

Wie viele Qubits lassen sich (theoretisch) zu einem Quantenschaltkreis kombinieren?

- ○ 8
- ○ 4
- ○ 32
- ◉ so viele, wie benötigt werden
- ○ 16
- ○ 2

einloggen

Korrekt

> Qubits lassen sich zu einem Quantenschaltkreis zusammenschließen. Dabei ist der Default-Zustand $|0\rangle$. Mit den Quantengattern lassen sich Operationen auf den Qubits durchführen, also deren Zustände verändern, um Berechnungen durchzuführen. Damit lassen sich wie zu Beginn angesprochen Probleme, die in eine mathematische Form gebracht wurden, lösen, neue Moleküle simulieren oder Prozesse optimieren.

Wahrscheinlichkeiten

Wir wissen nun also, was Qubits sind, welche Quantengatter es gibt und was wir damit machen können. Doch wie gewinnen wir Informationen über ein uns unbekanntes Qubit, wenn wir bspw. nicht nur ein simples X-Gatter anwenden oder sich das Qubit in Superposition befindet ($|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$)?

In einem klassischen Experiment würden wir messen, ob sich das Bit in Zustand `0` oder `1` befindet. Es werden eindeutige Werte gemessen - entweder das Bit befindet sich in Zustand `0` oder `1` (Lichtschalter). Der gemessene Wert entspricht dem Wert, in dem sich das Bit vor der Messung befindet. Der Messwert ist im Vorhinein eindeutig festgelegt. Doch welche Werte liefert uns eine Messung in der Quantenmechanik? Betrachten wir zunächst den simplen Fall für ein Qubit, das sich in Zustand $|0\rangle$ befinden, das in Zustand $|0\rangle$ präpariert und anschließend gemessen wird.

In [20]:
```
# <- Bitte Zelle links ausführen
print('Und hier ist auch gleich die unbekannte Komponente. Sie steht also dafür, dass zu dem Zeitpunkt des Experim
qc8 = QuantumCircuit(1,1)
qc8.measure(0,0)
qc8.draw("mpl")
```

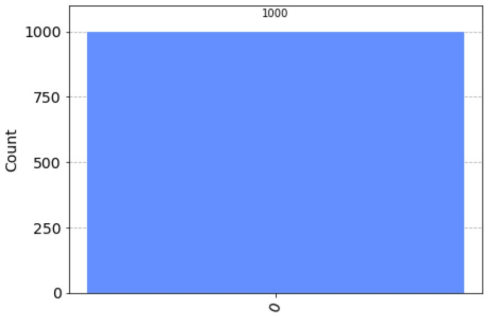
```
Und hier ist auch gleich die unbekannte Komponente. Sie steht also dafür, dass zu dem Zeitpunkt des Experiments
an dem entsprechenden Qubit eine Messung durchgeführt wird.
```

Out[20]:

Der Zustand unseres Qubits wird mit dem obigen Quantenschaltkreis gemessen. Mit der anschließenden Codezelle wird der Quantenschaltkreis 1000-mal simuliert und der Zustand des Qubits gemessen. Die Messwerte werden in einem Histogramm, also einer Art Säulendiagramm visualisiert. Überlegen Sie sich vor dem Ausführen der Zelle, wie das Ergebnis aussehen wird, und diskutieren Sie kurz in der Gruppe.

In [21]:
```
# <- Bitte Zelle links ausführen
sim = Aer.get_backend('aer_simulator')
result = sim.run(qc8, shots=1000).result()
counts = result.get_counts()
plot_histogram(counts)
```

Out[21]:



Das Ergebnis ist keine Überraschung. Wir präparieren das Qubit in Zustand $|0\rangle$ und die Messung ergibt in 100 % der Fälle den Zustand $|0\rangle$.

In [22]:
```
# <- Bitte Zelle links ausführen
display(probability1)
```

Welcher Zustand wird gemessen, wenn vorher ein X-Gatter auf ein Qubit in |0> angewendet wird?

- ○ eine Überlagerung aus |0> und |1>
- ○ |0>
- ◉ |1>

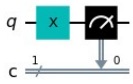
einloggen

Korrekt

In [23]:
```
# <- Bitte Zelle links ausführen
print('Auf das Qubit wird vor der Messung das X-Gatter angewendet.')
qc9 = QuantumCircuit(1,1)
qc9.x(0)
qc9.measure(0,0)
qc9.draw("mpl")
```

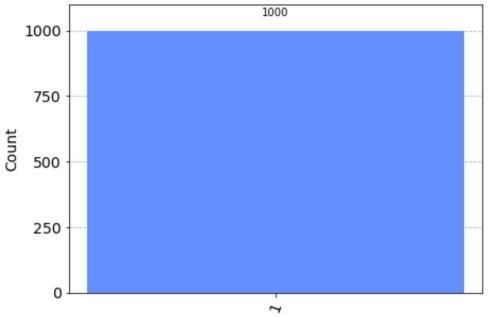
```
Auf das Qubit wird vor der Messung das X-Gatter angewendet.
```

Out[23]:



In [24]:
```
# <- Bitte Zelle links ausführen
sim = Aer.get_backend('aer_simulator')
result = sim.run(qc9, shots= 1000).result()
counts = result.get_counts()
plot_histogram(counts)
```

Out[24]:



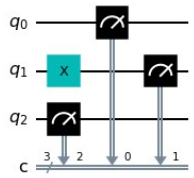
Auch dieses Ergebnis ergibt sich aus den Erkenntnissen, die wir vorher gewonnen haben. Wir haben ein Qubit im Zustand $|0\rangle$. Darauf wird das X-Gatter angewendet. Dies ist vergleichbar mit dem klassischen NOT-Gatter, dementsprechend ist das Qubit nun in Zustand $|1\rangle$:

$$|0\rangle \xrightarrow{X\text{-}Gatter} |1\rangle.$$

Die nachfolgende Zelle gibt einen Quantenschaltkreis mit 3 Qubits aus, wobei auf das zweite Qubit das X-Gatter angewendet wird. Was wird gemessen?

In [25]:
```
# <- Bitte Zelle links ausführen
qc10 = QuantumCircuit(3,3)
qc10.x(1)
qc10.measure(0,0)
qc10.measure(1,1)
qc10.measure(2,2)
qc10.draw("mpl")
```

Out[25]:



In [26]:
```
# <- Bitte Zelle links ausführen
display(probability2)
```

Welcher Zustand wird gemessen?

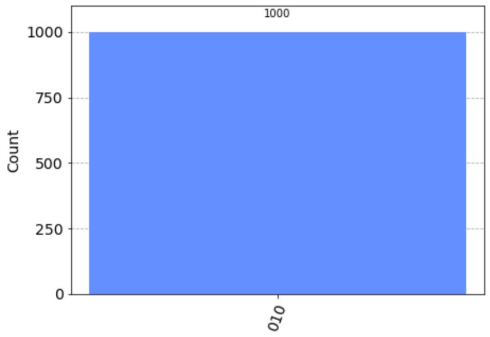
- ○ jeweils eine Überlagerung aus |0> und |1>
- ○ |001>
- ○ |101>
- ○ |011>
- ○ |100>
- ○ |110>
- ○ |000>
- ○ |111>
- ◉ |010>

einloggen

Korrekt

In [27]:
```
# <- Bitte Zelle links ausführen
sim = Aer.get_backend('aer_simulator')
result = sim.run(qc10, shots= 1000).result()
counts = result.get_counts()
plot_histogram(counts)
```

Out[27]:



Die Qubits befinden sich entweder eindeutig in Zustand $|0\rangle$ - dies gilt für das 1. und 3. Qubit, weil sie sich im Defaultzustand befinden - oder eindeutig in Zustand $|1\rangle$ - dies gilt für das 2. Qubit, auf das das X-Gatter angewendet wird.

Bisher haben wir uns lediglich Qubits gewidmet, die eindeutig in einem der Kombinationszustände $|0\rangle$ oder $|1\rangle$ präpariert waren. Die Messung solcher Qubits ergibt eindeutig den entsprechenden Zustand. In diesem Fall kann mit der Messung darauf geschlossen werden, in welchem Zustand sich das Qubit vor der Messung befindet.

> Analog zu den Bits ist das Messergebnis vor dem Messprozess festgelegt, wenn sich das Qubit eineutig in einem der Kombinationszustände befindet. Dies ist ein Spezialfall.

Der nächste Abschnitt widmet sich Qubits, die sich nicht eindeutig in einem der Kombinationszustände befinden, sondern in einer Superposition:

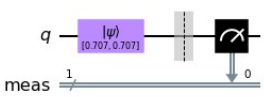
$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

"Dimmer-Zustände"

Als erstes betrachten wir ein Qubit, das sich in einer 50/ 50 Überlagerung von $|0\rangle$ und $|1\rangle$ befindet. In der nächsten Codezelle wird der Quantenschaltkreis dargestellt. Mit der anschließenden Zelle werden 5 Messdurchgänge mit jeweils 1000 Einzelmessungen simuliert und geplottet. Diskutieren Sie ein Minute in der Gruppe, was Sie für ein Histogramm erwarten.

In [28]:
```
# <- Bitte Zelle links ausführen
state5050 = [1/sqrt(2), 1/sqrt(2)]
qc11 = QuantumCircuit(1)
qc11.initialize(state5050, 0)
qc11.measure_all()
qc11.draw("mpl")
```

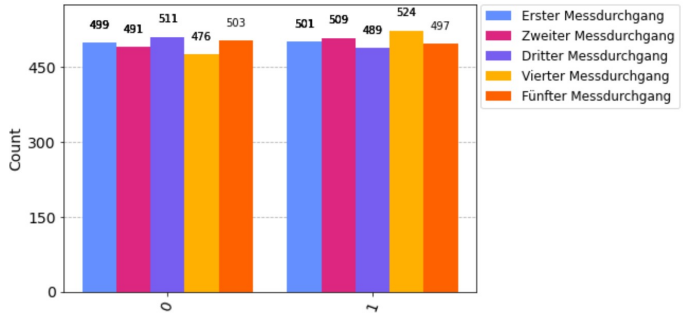
Out[28]:



Diskutieren Sie nach dem Ausführen der Codezelle eine Minute Ihre Vermutungen, was der Grund für das Histogramm sein kann.

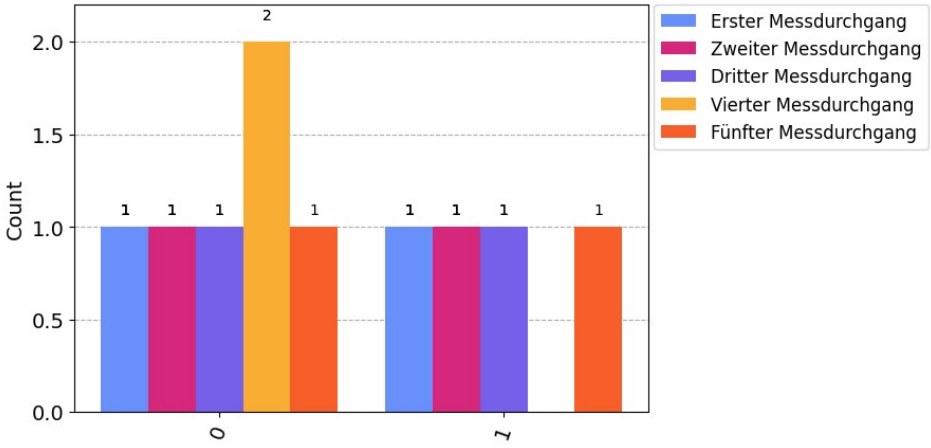
In [29]:
```
# <- Bitte Zelle links ausführen
backend = Aer.get_backend('qasm_simulator')
counts1 = execute(qc11, backend, shots= 1000).result().get_counts()
counts2 = execute(qc11, backend, shots= 1000).result().get_counts()
counts3 = execute(qc11, backend, shots= 1000).result().get_counts()
counts4 = execute(qc11, backend, shots= 1000).result().get_counts()
counts5 = execute(qc11, backend, shots= 1000).result().get_counts()
plot_histogram([counts1, counts2, counts3, counts4, counts5], legend = ['Erster Messdurchgang', 'Zweiter Messdurch
```

Out[29]:



Das Diagramm zeigt etwas Ungewöhnliches. Wir erhalten bei allen fünf Messdurchgängen unterschiedliche Ergebnisse. Es ist also nicht eindeutig vorbestimmt und von dem Qubit ablesbar, ob wir das Qubit in Zustand $|0\rangle$ oder $|1\rangle$ messen. Zudem lässt sich aus dem Diagramm erkennen, dass das Qubit **nach** der Messung eindeutig in Zustand $|0\rangle$ oder $|1\rangle$ befindet, obwohl es sich vorher in einer Superposition befindet. Es wird mit der Messung gewissermaßen "festgelegt", in welchem Zustand sich das Qubit befindet. Bis zu der Messung ist das Qubit somit nicht eindeutig in Zustand $|0\rangle$ oder in Zustand $|1\rangle$ und enthält Informationen zu beiden Zuständen (vielleicht haben sie schon einmal etwas über Schrödingers Katze gehört).

Das Messergebnis ist trotz identischer Präparation des Qubits, also trotz des identischen Eingangszustands bei den insgesamt 5000 Messvorgängen nicht vor der Messung festgelegt. Dies wird noch greifbarer, wenn wir Messdurchgänge mit jeweils 2 Messungen betrachten. Die Messergebnisse sind nicht immer 1x $|0\rangle$ und 1x $|1\rangle$.



Wenn das Eingangsqubit (die Präparation dessen) betrachten wird, lässt sich das verstehen. Das Qubit befindet sich in dem "50/50-Zustand", also in einem "Dimmer-Zustand". Es befindet sich zu 50 % in Zustand $|0\rangle$ und zu 50 % in Zustand $|1\rangle$. Die Formeln geben uns also nur Wahrscheinlichkeiten als Aussagen. Jeder der beiden Zustände ist zu 50 % messbar. Dies ist vergleichbar mit einem Münzwurf. Wir erhalten zu 50 % Kopf und zu 50 % Zahl. Wie bei Zufallsexperimenten üblich, ergibt sich nicht exakt eine 50/50 Verteilung. Ein weiteres Beispiel dafür ist ein Würfel. Wenn wir sechs mal würfeln, erhalten wir nicht jeweils 1x die 1, 1x die 2, usw., sondern zum Beispiel 2x die 1, 3x die 4 und 1x die 6. Trotzdem ist weder unser Würfel, noch die Wahrscheinlichkeitstheorie "kaputt".

Die Quantenmechanik ist somit eine *probabilistische Theorie*, die als Vorhersagen über ein Experiment Wahrscheinlichkeiten liefert. Sie berücksichtigt also die Wahrscheinlichkeit beim Eintreten von Ereignissen.

Es bleibt noch die Frage offen, wo genau die Wahrscheinlichkeiten enthalten sind. Die kurze Antwort ist: In den Gewichtungen der einzelnen Zustände, die (linear) kombiniert werden. Die Gewichtungen gewichten also, mit welcher Wahrscheinlichkeit sich das Qubit nach der Messung in dem Zustand befindet. Mathematisch korrekt betrachtet, wird noch das Betragsquadrat der Gewichtung berechnet, um wirklich die Wahrscheinlichkeit zu bestimmen.

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$\Rightarrow \text{ Wahrscheinlichkeit } |0\rangle \text{ zu messen: } |\alpha|^2$$

$$\Rightarrow \text{ Wahrscheinlichkeit } |1\rangle \text{ zu messen: } |\beta|^2$$

Für den "50/50-Zustand" ergibt sich:

$$|\Psi_{50/50}\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

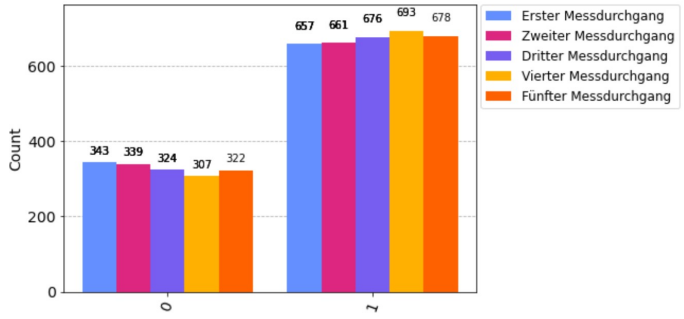
$$\text{D. h. } \alpha = \frac{1}{\sqrt{2}},\ \beta = \frac{1}{\sqrt{2}} \text{ und } |\alpha|^2 = |\beta|^2 = \frac{1}{2}$$

Im Folgenden wird ein Qubit in dem Zustand bestehend aus einem Drittel $|0\rangle$ und zwei Drittel $|1\rangle$ präpariert und ausgewertet. Was erwarten Sie für ein Säulendiagramm bei fünf Messdurchgängen mit jeweils 1000 einzelnen Messungen pro Durchgang?

In [30]:

```
# <- Bitte Zelle links ausführen
state13_23 = [1/sqrt(3), sqrt(2/3)]  # Define state |q_0>
qc12 = QuantumCircuit(1) # Must redefine qc
qc12.initialize(state13_23, 0) # Initialize the 0th qubit in the state `initial_state`
qc12.measure_all()
backend = Aer.get_backend('qasm_simulator')
counts1 = execute(qc12, backend, shots= 1000).result().get_counts()
counts2 = execute(qc12, backend, shots= 1000).result().get_counts()
counts3 = execute(qc12, backend, shots= 1000).result().get_counts()
counts4 = execute(qc12, backend, shots= 1000).result().get_counts()
counts5 = execute(qc12, backend, shots= 1000).result().get_counts()
plot_histogram([counts1, counts2, counts3, counts4, counts5], legend = ['Erster Messdurchgang', 'Zweiter Messdurch
```

Out[30]:



Also auch hier haben wir ein Messergebnis, was einem Zufallsexperiment entspricht. Zu ungefähr 33 % befindet sich das Qubit in Zustand $|0\rangle$, zu ungefähr 66 % in Zustand $|1\rangle$, aber nicht genau zu diesen Anteilen.

Mit dem nachfolgenden Slider können Sie wie in der Einführung gezeigt einen Wert festlegen. In diesem Fall legen sie damit die Wahrscheinlichkeit fest, mit der sich das Qubit in Zustand $|0\rangle$ befindet. Probieren Sie verschiedene Werte aus und plotten Sie das dazugehörige Histogramm, nachdem Sie sich überlegt haben, wie dieses aussehen wird.

In [31]:

```
# <- Bitte Zelle links ausführen
probslider = widgets.IntSlider(
    min=0,
    max=100,
    step=1,
    description='',
    value=50,
    layout = widgets.Layout(min_width='650px', height='50px')
)

print('Wahrscheinlichkeit für |0> (in Prozent):')
display(probslider)
```

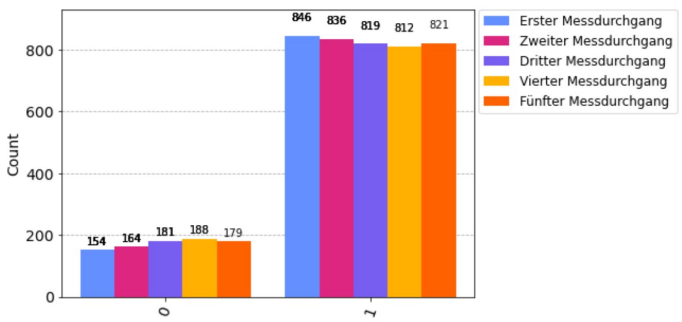
```
Wahrscheinlichkeit für |0> (in Prozent):
```

In [32]:

```
# <- Bitte Zelle links ausführen
prob0 = float(probslider.value) * 0.01
print()
prob1 = abs(1- prob0)
print("Der von Ihnen präparierter Zustand: |Ψ> = " + str(round(sqrt(prob0),2)) + " |0> + " + str(round(sqrt(abs(1-
print()
state_student = [sqrt(prob0), sqrt(abs(1-prob0))]  # Define state |q_0>
qc13 = QuantumCircuit(1) # Must redefine qc
qc13.initialize(state_student, 0) # Initialize the 0th qubit in the state `initial_state`
qc13.measure_all()
backend = Aer.get_backend('qasm_simulator')
counts1 = execute(qc13, backend, shots= 1000).result().get_counts()
counts2 = execute(qc13, backend, shots= 1000).result().get_counts()
counts3 = execute(qc13, backend, shots= 1000).result().get_counts()
counts4 = execute(qc13, backend, shots= 1000).result().get_counts()
counts5 = execute(qc13, backend, shots= 1000).result().get_counts()
plot_histogram([counts1, counts2, counts3, counts4, counts5], legend = ['Erster Messdurchgang', 'Zweiter Messdurch
```

```
Der von Ihnen präparierter Zustand: |Ψ> = 0.41 |0> + 0.91 |1>
```

Out[32]:



> Die Quantenmechanik liefert für Experimente somit Vorhersagen in Form von Wahrscheinlichkeiten. Dies liegt in den Qubits begründet, deren Zustand eine Superposition ist. Die Messergebnisse sind wie bei anderen Zufallsexperimenten (Münzwurf, Würfel, ...) statistisch verteilt.

Abschließendes Quiz

In [33]:

```
# <- Bitte Zelle links ausführen
display(probability3)
print()
display(quantumcircuit2)
print()
display(bloch7)
print()
display(Q4)
print()
display(probability4)
```

Wofür stehen α und β in |Ψ> = α |0> + β |1>?

- (•) Das Betragsquadrat gibt die Wahrscheinlichkeit an, mit der sich das Qubit in dem nachfolgenden Zustand befindet.
- ( ) Geben die Wahrscheinlichkeit an, mit der sich das Qubit in dem nachfolgenden Zustand befindet.

einloggen

Korrekt

Was ist jeweils der Default-Zustand der Eingangsqubits?

- (•) |0>
- ( ) |1>
- ( ) eine Überlagerung aus |0> und |1>

einloggen

Korrekt

In welchem Zustand befindet sich ein Default-Qubit, wenn das Hadamard-Gatter und dann das X-Gatter angewendet wird?

- ( ) |1>
- ( ) |0>
- (•) in einer Überlagerung aus |0> und |1>

einloggen

Korrekt

Kann ein Bit oder ein Qubit mehr verschiedene Zustände annehmen und warum?

- (•) Qubit
- ( ) Bit

einloggen

Korrekt

Was liefert die Quantenmechanik für Informationen über Experimente?

- ( ) Aussagen, was ein Experiment eindeutig ergibt.
- (•) Vorhersagen in Form von Wahrscheinlichkeiten
- ( ) Aussage über Eingangszustand eines beliebigen Qubits

einloggen

Korrekt