FH Aachen
Fachbereich Elektrotechnik und Informationstechnik
Prof. Dr. Marko Schuba

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Bachelorarbeit

Implementierung einer Bibliothek von Quantenalgorithmen zur Kryptoanalyse

Eingereicht von:
Simon Maximilian Kalytta
Matrikelnummer: 3190683

Studienrichtung: Informatik

8. September 2023

Betreuerin: B.Sc. Janis König
Prüfer: Prof. Dr. Marko Schuba

In Kooperation mit it.sec GmbH

Kurzfassung

Schlagwörter: placeholder

Abstract

Keywords: placeholder

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

Glossar

1 Einleitung

1.1 Ziele und Vorgehen

Das Ziel dieser Arbeit ist die Entwicklung und Implementierung einer auf Quantenalgorithmen basierenden Bibliothek zur Kryptoanalyse aktueller Verschlüsselungsverfahren. Angesichts der Komplexität und des Fachwissens, das zur Entwicklung von Quantenalgorithmen erforderlich ist, konzentriert sich diese Arbeit auf die Implementierung und Anpassung prominenter Quantenalgorithmen aus der Literatur. Die verwendeten Quantenalgorithmen können bestimmte Problemstellungen, die aufgrund ihrer Komplexität eine zentrale Bedeutung in modernen Verschlüsselungsverfahren haben, deutlich schneller lösen als die effizientesten klassischen Algorithmen.

In den zugrunde liegenden wissenschaftlichen Arbeiten werden diese Quantenalgorithmen als abstrakte oder konzeptuelle Algorithmen vorgestellt, ohne dass auf konkrete Implementierungsdetails eingegangen wird. Diese Arbeit beseitigt die Diskrepanz zwischen theoretischen Konzepten und praktischen Realisierungen, indem auf der Basis der abstrakten Quantenalgorithmen eine konkrete Implementierung entwickelt wird. Anschließend werden die resultierenden Implementierungen der Quantenalgorithmen mit klassischen Algorithmen kombiniert, um die Problemstellungen, die für die Sicherheit der Verschlüsselungsverfahren entscheidend sind, effektiv lösen zu können.

Diese Arbeit implementiert die Bibliothek auf der Abstraktionsebene von Quantenschaltkreisen. Dazu wird das Open-Source-Softwareentwicklungskit Qiskit genutzt das auf der Programmiersprache Python basiert.

1.2 Motivation

In den letzten Jahren haben Fortschritte in der Forschung und Entwicklung von Quantencomputern neue Möglichkeiten für die praktische Untersuchung von Quantenalgorithmen ermöglicht. Gegenwärtig ermöglichen sowohl Simulatoren von Quantencomputer als auch real existierende, wenn auch leistungsbegrenzte, Quantencomputer die Durchführung praktischer Tests. Zur Zeit der ursprünglichen Konzeption der in dieser Arbeit

verwendeten Quantenalgorithmen war eine praktische Erprobung entweder undenkbar oder nur durch umständliche Experimente mit stark vereinfachten Versuchen möglich.

Indem diese technologischen Möglichkeiten zur Ausführung und Erprobung der implementierten Quantenalgorithmen genutzt werden, eröffnet sich ein neuer Standpunkt, der die Betrachtung aus anderen Blickwinkeln erlaubt.

1.3 Gliederung

Hier wird dann beschrieben in welchen Kapiteln der Leser welche Inhalte erwarten kann und wieso die Reihenfolge der Kapitel so gewählt wurden, inklusive der Struktur der Arbeit

2 Schwerpunkt

2.1 Entschlüsselungsfunktion für RSA

Der Fokus dieser Arbeit liegt auf der Implementierung einer Entschlüsselungsfunktion, die in der Lage ist, den privaten Schlüssel des RSA-Verfahrens aus dem zugehörigen öffentlichen Schlüssel abzuleiten.

Das RSA-Verfahren stellt ein sogenanntes asymmetrisches Kryptosystem dar. Bei asymmetrischen Kryptosystemen kommt ein mathematisch verknüpftes Schlüsselpaar zum Einsatz. Dieses Schlüsselpaar besteht aus einem öffentlichen und einem privaten Schlüssel, wobei Letzterer ausschließlich dem Eigentümer des Schlüsselpaares zugänglich sein sollte. Mit dem privaten Schlüssel ist der Eigentümer in der Lage, Nachrichten zu signieren. Ein Nutzer kann unter der Verwendung des öffentlichen Schlüssels die Authentizität der signierten Nachricht überprüfen. Hingegen erlaubt der öffentliche Schlüssel die Verschlüsselung von Nachrichten, deren anschließende Entschlüsselung ausschließlich mit dem privaten Schlüssel durchführbar ist. Eine Bedingung der asymmetrischer Kryptosysteme ist, dass die Ableitung des privaten Schlüssels aus dem öffentlichen Schlüssel eine Herausforderung von erheblicher Komplexität darstellt [DH76]. Um dieser Anforderung gerecht zu werden, basieren asymmetrische Kryptosysteme auf mathematischen Problemstellungen, deren Lösung sogar mit dem Nutzen eines Computers von erheblicher Komplexität ist.

Die Sicherheit des kryptographischen Verfahren RSA beruht auf der Annahme, dass die Faktorisierung eines Produkts bestehend aus zwei großen Primzahlen in keiner vertretbaren Zeit berechenbar ist. Andernfalls wäre es möglich, aus dem öffentlichen Schlüssel die beiden Primfaktoren zu extrahieren, um anschließend damit den privaten Schlüssel zu bestimmen. Daraus ergibt sich die Folgerung, dass, sofern die Faktorisierung großer Zahlen mit geringen Aufwand bewältigt werden kann, das RSA-Verfahren als kompromittiert angesehen werden muss [Cor+09].

Bislang ist kein klassischer Algorithmus bekannt, der die Primfaktorzerlegung effizient berechnen kann [Hoe22]. In diesem Zusammenhang bedeutet "effizient", dass die Laufzeit des Algorithmus in einem höchstens polynomialen Verhältnis zur Größe der Eingabezahl anwächst.

Im Kontext der Entschlüsselungsfunktion für RSA wird eine Funktion implementiert, die in der Lage ist, die Primfaktorzerlegung effizient zu berechnen. Zudem wird diese Funktion spezifisch darauf ausgerichtet, aus einem öffentlichen Schlüssel den zugehörigen privaten Schlüssel des RSA-Verfahrens abzuleiten. Die Entschlüsselungsfunktion besteht aus einem Quantenalgorithmus und einem klassischer Algorithmus. Der eingesetzte Quantenalgorithmus kann effektiv die Ordnung eines Elements in einer Gruppe bestimmen. Unter Einbeziehung der zuvor bestimmten Ordnung werden anschließend die Primfaktoren des öffentlichen Schlüssels mithilfe eines klassischen Algorithmus berechnet [Sho97]. In einem abschließenden Schritt wird der private Schlüssel auf der Grundlage der berechneten Primfaktoren ermittelt.

3 Quantencomputer

3.1 Entwicklung

Die Grundidee der Quantencomputer findet ihren Ursprung in dem Jahre 1980 als Paul Benioff ein theoretisches Modell einer klassischen Turing-Maschine beschrieb, die den Gesetzen der Quantenmechanik unterlag. Benioff demonstrierte, dass die Zustände einer Turing-Maschine in einem Quantensystem darstellbar sind. Er zeigte außerdem, dass klassische Operationen, die in einer Turing-Maschine ausgeführt werden, durch entsprechende Quantengatter auf einem Quantensystem äquivalent durchgeführt werden können [Ben80].

Kurz nach der Veröffentlichung von Benioffs Arbeit beschäftigte sich Richard Feynman im Jahr 1981 mit der Frage, wie effizient ein klassischer Computer bei der Simulation physikalischer Prozesse ist. Feynman befasste sich mit der Schwierigkeit, die Quantenmechanik mithilfe eines klassischen Computers zu simulieren. Er stellte fest, dass die Anforderungen für die Simulation der Quantenmechanik auf einem klassischen Computer mit jedem zusätzlichen Quantenteilchen exponentiell ansteigen. Die Begründung dafür liegt in der exponentiell wachsenden Menge an Zustandsinformationen, welche für die Beschreibung des Quantensystems benötigt werden. Als Lösungsansatz nannte Feynman einen Quantencomputer, der selbst auf den Prinzipien der Quantenmechanik basiert und dadurch eine effiziente Simulation von Quantensystemen ermöglicht [Fey82].

David Deutsch präsentierte im Jahr 1985 den ersten Quantenalgorithmus, der eine spezifische Fragestellung effizienter lösen konnte als jegliche bekannten klassischen Algorithmen. Allerdings bezog sich diese Fragestellung auf ein eher unrealistisches Problem. Ein klassischer Algorithmus würde zur Beantwortung dieser Frage zwei Funktionsauswertungen benötigen. Hingegen benötigt der Quantenalgorithmus, aufgrund der Nutzung

von Quantenparallelismus, für die gleiche Fragestellung nur eine einzige Funktionsauswertung [Deu85].

Im Jahr 1994 stellte Shor einen Quantenalgorithmus vor, der in der Lage ist, die Ordnung beziehungsweise Periode eines Elements in der multiplikativen Gruppe eines Modulus mit nur polynomialem Aufwand zu ermitteln. Da trotz erheblicher Anstrengungen lediglich klassische Algorithmen mit exponentiellem Aufwand für diese Berechnung zur Verfügung stehen, stellte Shors Entdeckung eine Errungenschaft dar, die das allgemeine Interesse für Quantencomputing enorm anregte [Sho97].

Zwei Jahre später stellte Lov K. Grover einen Quantenalgorithmus vor. Genau wie der Quantenalgorithmus von Shor, kann auch Grover's Quantenalgorithmus ein bestimmtes Problem effizienter lösen als jeglicher klassische Algorithmen. Der Quantenalgorithmus ermöglicht die Suche in einer unsortierten Datenbank mit einem Aufwand von $\mathcal{O}(\sqrt{N})$, während der beste klassische Such-Algorithmus diese Aufgabe in $\mathcal{O}(N)$ bewältigt. Darüber hinaus konnte Grover zeigen, dass es auch unter vollständiger Ausnutzung der Prinzipien der Quantenmechanik nicht möglich ist, die unstrukturierte Suche in weniger als $\mathcal{O}(\sqrt{N})$ Schritten durchzuführen [Gro96].

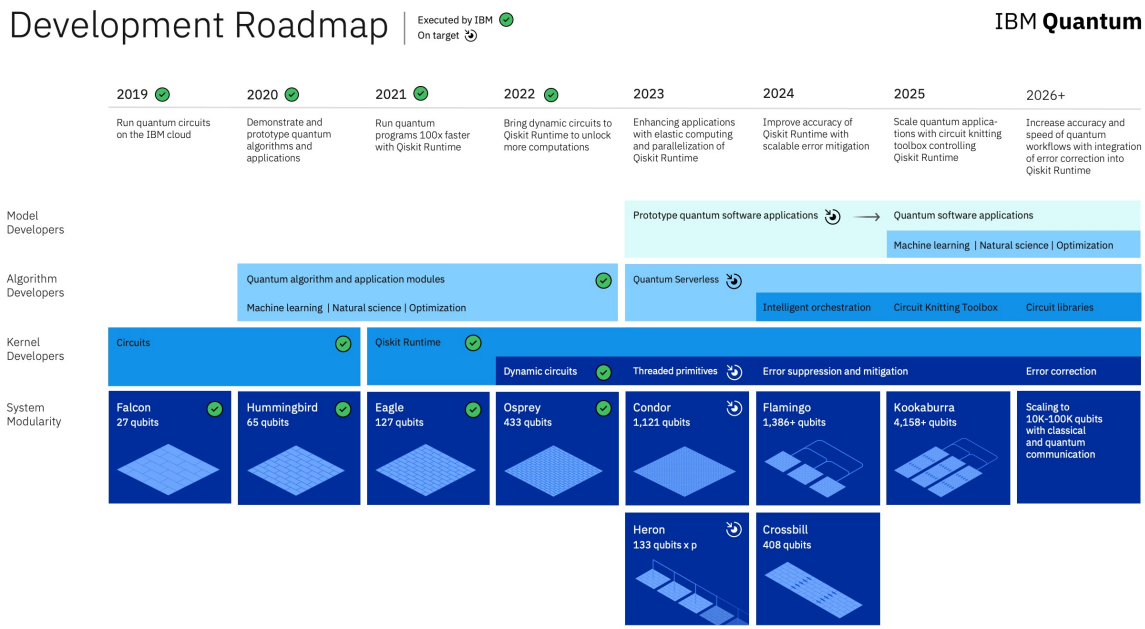
Die erste erfolgreiche physikalische Implementation eines Quantenalgorithmus wurde im Jahre 1998 durchgeführt. In dem Versuch wurde der Quantenalgorithmus von David Deutsch auf einem zwei Qubit Quantencomputer Implementiert. Der eingesetzte Quantencomputer nutzte die Kernspinresonanz-Technologie, wobei die zwei Qubits durch Ausnutzung der Spin-Zustände von Atomkernen in spezifischen Molekülstrukturen realisiert wurden [JM98].

In der fortlaufenden Entwicklung der Quanteninformationstechnologie wurden verschiedene Arten von Quantencomputern konzeptioniert und realisiert. Zu diesen zählen adiabatische Quantencomputer, schaltkreisbasierte und topologische Quantencomputer. Im Kontext des Quantencomputing wird der Begriff „Quantencomputer“ oft auf universell einsetzbare Quantencomputer angewendet. Der Eigenschaft „universell“ wird in dem Zusammenhang gemäß DiVincenzo's Kriterien definiert. Im Wesentlichen besagen DiVincenzo's Kriterien, dass ein universeller Quantencomputer in der Lage sein muss, jede Quantenberechnung auszuführen, vorausgesetzt, er verfügt über ausreichende Ressourcen [DiV00]. Die Schaltkreis-basierende und topologische Quantencomputer erfüllen die DiVincenzo's Kriterium und zählen somit als universelle Quantencomputer. Hingegen sind adiabatische Quantencomputer nicht universell Programmierbar und werden explizit zum Lösen von Optimierungsproblemen eingesetzt.

Das Unternehmen International Business Machines (IBM) präsentierte im Jahr 2019 den ersten kommerziell verfügbaren, schaltkreisbasierten Quantencomputer. Dieses System, bekannt als „IBM Q System One“, verfügt über 20 Qubits. Seit der Vorstellung des System One hat IBM die Qubit-Kapazität kontinuierlich erhöht. Aktuelle Systeme, wie sie in der Abbildung 1 dargestellt sind, verfügen über 433 Qubits.

In Anbetracht zukünftiger Entwicklungen haben Unternehmen wie Google, IBM und Microsoft umfangreiche Pläne zur Erweiterung ihrer Quantencomputing-Kapazitäten. Google plant beispielsweise die Errichtung eines Quanten-Campus, der bis 2030 über

Abbildung 1: IBM-Quantum-Roadmap [23b]



einen Quantencomputer mit einer Million Qubits verfügen soll [23c]. Microsoft verfolgen ähnliche Ziele, wobei der konkrete Umfang ihrer Projekte bislang nicht öffentlich spezifiziert wurde. Des weiteren erwartet das Bundesamt für Sicherheit in der Informationstechnik die Existenz kryptografisch relevanter Quantencomputer zu Beginn der 2030er Jahre [23a].

3.2 Vorteile gegenüber klassischen Rechnern

Quantenparallelismus

4 Fundament

4.1 Literatur

Shor's Algorithmus

Der Algorithmus wurde erstmals in der Publikation *„Algorithms for Quantum Computation: Discrete Logarithms and Factoring“* von Peter W. Shor veröffentlicht.

Die Arbeit von Shor umfasste zwei Algorithmen. Der erste Algorithmus ermöglicht die effiziente Berechnung der Primfaktorzerlegung, während der zweite Algorithmus die effiziente Berechnung des diskreten Logarithmus ermöglicht. Aufgrund ihrer konzeptio-

nellen Ähnlichkeiten werden beide Algorithmen häufig kollektiv als „Shor's Algorithmus“ bezeichnet.

Die Quantenberechnungen beider Algorithmen basieren auf arithmetische Operationen in Restklassen und der Quanten-Fourier-Transformation. Da Shor die Umsetzung von arithmetischen Operationen in Restklassen sowie die Quanten-Fourier-Transformation nicht explizit behandelt, stützt sich die Implementierung auf den Ergebnissen weiterer Arbeiten. Diese untersuchen insbesondere effiziente Methoden zur Durchführung von arithmetischer Operationen in Restklassen innerhalb eines Quantenschaltkreises. Einige dieser Arbeiten untersuchen die Quanten-Fourier-Transformation, da diese ein notwendiges Element für gewisse Berechnungen darstellt.

In der Publikation *„Quantum Networks for Elementary Arithmetic Operations“* erklären Vlatko Vedral, Adriano Barenco und Artur Ekert, wie die modulare Exponentiation in einem Quantenschaltkreis berechnet werden kann.

Stéphane Beauregard baut auf den Erkenntnissen von Vedral, Barenco und Ekert auf und verbessert den Quantenschaltkreis für die arithmetische Operation der modularen Exponentiation. Hierfür ersetzt Beauregard einen Teil des ursprünglichen Quantenschaltkreis, der in der modularen Exponentiation für die Berechnung der Addition genutzt wurde, durch einen effizienteren Quantenschaltkreis, wie ihn Thomas G. Draper in *„Addition on a Quantum Computer“* beschreibt. Des weiteren verwendet Beauregard diese Optimierungen in seiner Arbeit *„Circuit for Shor's algorithm using 2n+3 qubits“*, um eine Realisierung von Shor's Algorithmus zur Faktorisierung zu beschreiben.

Andere Arbeiten, die sich mit der Implementierung von Shors's Algorithmus auseinandersetzen, realisieren die modularen Exponentiation, indem klassische Schaltkreise identisch in den Quantenkontext übersetzt werden. Der Nachbau von klassischen Operationen ist aufgrund der unitären Natur von Quantenoperationen nicht die effizienteste Realisierung. Des Weiteren meiden andere Arbeiten die Realisierung der modularen Exponentiation für allgemeine Eingaben. Ohne modulare Exponentiation sind diese Varianten nicht in der Lage die Berechnung variabler Werte zu verarbeiten. Stattdessen dienen diese ausschließlich der Demonstration der Funktionsweise von Shor's Algorithmus und sind nur in der Lage, ausgewählte Eingaben zu verarbeiten.

Bei Recherchen wurde keine allgemeine Variante gefunden, die weniger Qubits benötigt als die Variante aus der Arbeit *„Circuit for Shor's algorithm using 2n+3 qubits“*, deswegen bildet diese die Grundlage der Implementierung ab.

5 Grundlagen

5.1 Qubits

„Der Anfang enthält also beides, Sein und Nichts; ist die Einheit von Sein und Nichts, – oder ist Nichtsein, das zugleich Sein, und Sein, das zugleich Nichtsein ist.“ - Georg Wilhelm Friedrich Hegel

Die Repräsentation und Speicherung von Information ist ein zentraler Aspekt aller Computer, unabhängig von der verwendeten Technologie oder Architektur. Bei klassischen Computern basiert diese Repräsentation auf dem Binärsystem. In der klassischen Repräsentation ist das Bit die kleinste Informationseinheit, die eine von zwei mögliche Zustände annehmen kann: 0 oder 1. Ein Bit hat zu jedem Zeitpunkt einen klar definierten Zustand. Mehrmaliges Auslesen eines Bits führt zu keiner Zustandsänderung, sofern keine Operationen zwischen den Auslesungen durchgeführt werden.

Quantencomputer hingegen funktionieren grundlegend anders. Sie stützen sich auf die Prinzipien der Quantenmechanik und verwenden anstelle von Bits Quatenbits beziehungsweise Qubits zur Informationsrepräsentation. Ein einzelnes Qubit stellt die kleinstmögliche Informationseinheit dar über die ein Quantencomputer verfügt. Um quantenmechanische Zustände von Qubits zu beschreiben wird die Dirac-Notation als mathematische Schreibweise genutzt [Dir39].

Ein Qubit kann den Zustände 0 oder den Zustand 1 annehmen:

$$|0\rangle \text{ oder entsprechend } |1\rangle$$

Des weiteren kann ein Qubit auch beide der Basiszustände gleichzeitig einnehmen:

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

Dieses Phänomen ist eine der charakteristischen Eigenschaften von Qubits und wird als **Superposition** bezeichnet. Dabei handelt es sich bei den Vorfaktoren α und β um komplexe Zahlen für die gilt:

$$|\alpha|^2 + |\beta|^2 = 1$$

Es gibt also unendlich viele Zustände, die ein Qubit in einer Superposition der beiden Basiszustände annehmen kann.

In der klassischen Welt findet man nur schwer eine Analogie zur Superposition. Man kann sich dies jedoch in etwa an einer Münze vorstellen [Hoe23a]:

Ein klassisches Bit kann dabei entweder auf der Kopf- oder auf der Zahl-Seite liegen. Ein Qubit hingegen ist eine Münze die auf der Kante um die eigene Achse rotiert. Dabei geben die Vorfaktoren α und β an, wie die Münze zu der einen oder zu der anderen Seite tendiert. Ohne die Münze zu beeinflussen ist es nicht möglich die Tendenz, also die Vorfaktoren, zu bestimmen.

Möchte man nun aber doch ein konkretes Ergebnis haben, muss man das Qubit lesen, beziehungsweise genauer gesagt messen. Dabei wird die Superposition zerstört und das Qubit kollabiert in einen der beiden Basiszustände.

In der Analogie zu der Münze wird die Rotation der Münze gezielt gestoppt, sodass diese auf eine der beiden Seiten kippt.

Die Wahrscheinlichkeiten dafür werden durch die Vorfaktoren des Qubits bestimmt:

$$|\alpha|^2 \text{ für } |0\rangle,\ |\beta|^2 \text{ für } |1\rangle$$

Es ist also nicht möglich den Zustand eines Qubit während einer Berechnung nur „anzuschauen“, da ansonsten die Superposition zerstört wird und sich somit der Zustand des Qubits verändert. Aus diesem Grund erfolgt die Messung in der Regel erst am Ende eines Quantenalgorithmus, um das endgültige Ergebnis zu ermitteln.

5.2 Quantengatter

5.3 Qiskit

5.4 RSA

5.5 Quanten-Fourier-Transformation

Die Quanten-Fourier-Transformation bildet einen wesentlichen Bestandteil des implementierten Quantenalgorithmus. Im folgen Abschnitt wird die allgemeine Anwendung der Quanten-Fourier-Transformation erklärt. Darüber hinaus wird die Implementierung des Quantenschaltkreis anhand der Formel der Quanten-Fourier-Transformation hergeleitet.

Im Prinzip handelt es sich bei der Quanten-Fourier-Transformation um eine Transformation, die Qubits von der Standardbasis ($|0\rangle$, $|1\rangle$), in die entsprechende Fourierbasis ($|+\rangle$, $|-\rangle$) überführt [Hom]. Bei dem Basiswechsel werden die Informationen des vorherigen Standardbasiszustandes in die Phase des neuen Zustandes übertragen [RG17]. Anschließend können in der Fourierbasis Rechnungen durchgeführt werden die im Grunde durch Manipulationen der Phase realisiert werden. Mit diesem Ansatz ist es möglich arithmetische Operationen, wie beispielsweise die Addition, effizienter zu berechnen [Dra00][RG17].

Unter Verwendung der inversen Quanten-Fourier-Transformation, die als Rücktransformation dient, ist es möglich wieder zurück in die Standardbasis zu transformieren. Dies bewirkt, die Extraktion der Information aus der Phase in einen messbaren Zustand.

Die Quanten-Fourier-Transformation ist für ein $N = 2^n$ mit n Qubits für die Basisvektoren $|x\rangle, x = 0, ..., N-1$ wie folgt definiert [Hoe23b]:

$$QFT_N |x\rangle_n = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i x y}{N}} |y\rangle_n$$

Anhand dieser Definition kann der Quantenschaltkreis nicht direkt hergeleitet werden. Stattdessen musst die Formel umgeformt werden.

Die folgende Herleitung stammt aus dem Textbuch *Quantum Computation and Quantum Information* Seite 218 [NC10]. Die Herleitung wird der Übersicht halber um Zwischenschritte ergänzt:

Indem y in der ersten Formel in die Binärschreibweise übertragen wird, erhält man:

$$y = \sum_{k=1}^{n} 2^{n-k} y_{n-k+1}$$

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}} \sum_{y_n=0}^{1} ... \sum_{y_1=0}^{1} e^{\frac{2\pi i x \sum_{k=1}^{n} 2^{n-k} y_{n-k+1}}{N}} \left|y_n ... y_2 y_1\right\rangle \tag{1}$$

Mit $N = 2^n$ kann der Bruch im Exponenten gekürzt werden:

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}} \sum_{y_n=0}^{1} ... \sum_{y_1=0}^{1} e^{\frac{2\pi i x \sum_{k=1}^{n} 2^{n-k} y_{n-k+1}}{2^n}} \left|y_n ... y_2 y_1\right\rangle$$

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}} \sum_{y_n=0}^{1} ... \sum_{y_1=0}^{1} e^{2\pi i x \sum_{k=1}^{n} 2^{-k} y_{n-k+1}} \left|y_n ... y_2 y_1\right\rangle$$

Anschließend kann der Ausdruck 2^{-k} zu $\frac{1}{2^k}$ umgeformt werden:

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}} \sum_{y_n=0}^{1} ... \sum_{y_1=0}^{1} e^{\frac{2\pi i x \sum_{k=1}^{n} y_{n-k+1}}{2^k}} \left|y_n ... y_2 y_1\right\rangle$$

Die Summe im Exponenten der Basis e kann als Produkt umgeschrieben werden. Anstatt dem Produktzeichen $\prod$ wird das Tensorprodukt $\bigotimes$ verwendet, da es sich um Qubits handelt:

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}} \sum_{y_n=0}^{1} ... \sum_{y_1=0}^{1} \bigotimes_{k=1}^{n} e^{\frac{2\pi i x y_k}{2^k}} \left|y_k\right\rangle$$

Der Ausdruck kann weiter vereinfacht werden indem das Tensorprodukt vorgezogen wird:

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} [\sum_{y_k=0}^{1} e^{\frac{2\pi i x y_k}{2^k}} \left|y_k\right\rangle]$$

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} [\left|0\right\rangle + e^{\frac{2\pi i x}{2^k}} \left|1\right\rangle]$$

Schreibt man das Tensorprodukt voll aus und notiert x in Binärschreibweise, erhält man:

$$\frac{1}{\sqrt{N}} (\left|0\right\rangle + e^{\frac{2\pi i (2^{n-1} x_n + ... + 2^1 x_2 + 2^0 x_1)}{2^1}} \left|1\right\rangle) \bigotimes \left|0\right\rangle + e^{\frac{2\pi i (2^{n-1} x_n + ... + 2^1 x_2 + 2^0 x_1)}{2^2}} \left|1\right\rangle ... \left|0\right\rangle + e^{\frac{2\pi i (2^{n-1} x_n + ... + 2^1 x_2 + 2^0 x_1)}{2^n}} \left|1\right\rangle$$

Die komplexe Exponentialfunktion ergibt für eine natürliche Zahl k: $e^{2\pi ik} = 1$. Mit dieser Eigenschaft kann man beispielsweise die Phasenverschiebung des ersten Tensors vereinfachen:

$$e^{\frac{2\pi i(2^{n-1}x_n+\ \dots\ +2^1x_2+2^0x_1)}{2^1}} \equiv e^{\frac{2\pi i(2^{n-1}x_n)}{2^1}}\ \dots\ e^{\frac{2\pi i(2^1x_2)}{2^1}} e^{\frac{2\pi i(2^0x_1)}{2^1}} \equiv e^{2\pi i(2^{n-2}x_n)}\ \dots\ e^{2\pi i(2^0x_2)} e^{2\pi i(2^{-1}x_1)}$$

Dabei ergibt nur der Term $e^{2\pi i(2^{-1}x_1)} \neq 1$ und verursacht somit eine relevante Phasenverschiebung.

Abschließend lässt sich das gesamte Tensorprodukt vereinfachen:

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}}(\left|0\right\rangle + e^{\frac{2\pi i(2^0x_1)}{2^1}} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{\frac{2\pi i(2^1x_2+2^0x_1)}{2^2}} \left|1\right\rangle) \dots (\left|0\right\rangle + e^{\frac{2\pi i(2^{n-1}x_n+\dots+2^1x_2+2^0x_1)}{2^n}} \left|1\right\rangle)$$

Ein einzelner Tensor repräsentiert die Wirkung der Schaltung auf ein einzelnes Qubit. Somit sind die Phasenverschiebungen erkenntlich die auf ein Qubit wirken. Ausserdem verdeutlicht die Binärschreibweise dass die angewendete Phasenverschiebung vom Zustand anderer Qubits abhängt.

Für die Implementierung der Quanten-Fourier-Transformation sind nur die Terme relevant welche eine Phasenverschiebung von $\neq 1$ bewirken. Dies kann man mit folgender Formel beschreiben:

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} [\left|0\right\rangle + e^{\frac{2\pi i \sum_{b=0}^{k-1} 2^b x_{b+1}}{2^k}} \left|1\right\rangle]$$

Im weiteren wird die Formel verwendet um ein Quantenschaltkreis der Quanten-Fourier-Transformation für drei Qubits zu implementieren:

$$QFT_8 \left|x\right\rangle_3 = \frac{1}{\sqrt{8}}[(\left|0\right\rangle + e^{\frac{2\pi i(2^0x_1)}{2^1}} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{\frac{2\pi i(2^1x_2+2^0x_1)}{2^2}} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{\frac{2\pi i(2^2x_3+2^1x_2+2^0x_1)}{2^3}} \left|1\right\rangle)]$$

Man kann die Phasenverschiebung die durch den Zustand eines einzelnen Qubits erzeugt wird verdeutlichen, indem man die Addition im Exponenten zu einer Multiplikation der gleichen Basen umformt:

$$= \frac{1}{\sqrt{8}}[(\left|0\right\rangle + e^{\frac{2\pi i(2^0x_1)}{2^1}} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{\frac{2\pi i(2^1x_2)}{2^2}} e^{\frac{2\pi i(2^0x_1)}{2^2}} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{\frac{2\pi i(2^2x_3)}{2^3}} e^{\frac{2\pi i(2^1x_2)}{2^3}} e^{\frac{2\pi i(2^0x_1)}{2^3}} \left|1\right\rangle)]$$

Die Phasenverschiebung wird eindeutiger indem man die Brüche kürzt:

$$= \frac{1}{\sqrt{8}}[(\left|0\right\rangle + e^{\pi i x_1} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{\pi i x_2} e^{\frac{\pi i x_1}{2}} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{\pi i x_3} e^{\frac{\pi i x_2}{2}} e^{\frac{\pi i x_1}{4}} \left|1\right\rangle)]$$

In einer abschließenden Umformung lässt sich die $\frac{1}{\sqrt{8}}$ aufteilen. Dadurch erinnern die einzelnen Tensoren, beziehungsweise Qubits an die Form die bei eine Hadamard-Transformation entsteht:

$$= \frac{1}{\sqrt{2}}(\left|0\right\rangle + e^{\pi i x_1} \left|1\right\rangle) \bigotimes \frac{1}{\sqrt{2}}(\left|0\right\rangle + e^{\pi i x_2} e^{\frac{\pi i x_1}{2}} \left|1\right\rangle) \bigotimes \frac{1}{\sqrt{2}}(\left|0\right\rangle + e^{\pi i x_3} e^{\frac{\pi i x_2}{2}} e^{\frac{\pi i x_1}{4}} \left|1\right\rangle)$$

Abbildung 2: 3-Qubit QFT ohne Swaps

Abbildung 3: 3-Qubit inverse QFT ohne Swaps

Der Ausdruck $e^{\pi i x_k}$ ist in jedem der einzelnen Qubits vorhanden. Des weiteren ist an dem x_k erkennbar das die angewendete Phasenverschiebung von dem Zustand des Qubits abhängig ist, auf welches die Verschiebung auch angewendet wird. Konkret bedeutet dass, das auf jedes Qubit mit dem Zustand $|x_k\rangle = |1\rangle$ eine Phasenverschiebung von $e^{\pi i}$ wirkt. Anhand des ersten Qubits, beziehungsweise Tensor, würde das bedeuten, dass der Zustand bei $x_1 = 0$ zu $|0\rangle + e^{\pi i 0} |1\rangle \equiv |0\rangle + |1\rangle$ wird. Bei $x_1 = 1$ würde man $|0\rangle + e^{\pi i 1} |1\rangle$ erhalten, was $|0\rangle - |1\rangle$ entspricht. Aufgrund des Vorfaktors von $\frac{1}{\sqrt{2}}$ entsprächen beide Fälle also der Hadamard-Transformation. Da auch alle anderen Tensoren den Ausdruck $e^{\pi i x_k}$ und den Vorfaktor $\frac{1}{\sqrt{2}}$ beinhalten, wirkt auf jedes Qubit ein Hadamard-Gatter.

Die weiteren Tensoren des Tensorproduktes beinhalten zunehmend mehr Ausdrücke die für unterschiedliche Phasenverschiebungen sorgen. Die Phasenverschiebung von einem einzelnen Ausdruck kann mit einem Phasen-Gatter realisiert werden. Das Phasengatter entspricht einer Rotation mit $P(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$.

Beispielsweise wirkt auf dem zweiten Qubit noch ein Phase-Gatter mit $P(\frac{\pi i}{2})$. Diese Phasenverschiebung soll aber nur angewendet werden wenn $x_1 = 1$ ist. Deswegen wird das Phase-Gatter durch den Zustand des ersten Qubits kontrolliert und mit einem kontrollierten-Phase-Gatter realisiert. Das gleiche Prinzip gilt für alle weiteren Qubits. Anschließend erhält man einen Quantenschaltkreis wie in Abbildung 2.

Wie im Quantenschaltkreis in Abbildung 2 erkennbar, spiegelt die Quanten-Fourier-Transformation die Reihenfolge der Qubits [Hoe23b]. Um die Reihenfolge wiederherzustellen werden am Ende der Quanten-Fourier-Transformation Swap-Operationen verwendet.

Anhand der Implementierung wird ersichtlich, dass die Quanten-Fourier-Transformation unitär ist. Diese Eigenschaft ergibt sich aus der Tatsache, dass der zugehörige Quantenschaltkreis ausschließlich unter Verwendung von unitären Gattern realisierbar ist.

Wie bereits oben erwähnt, wird für die Rücktransformation aus der Fourierbasis in die Standardbasis die inversen Quanten-Fourier-Transformation angewendet. Um einen Quantenschaltkreis aus unitären Gattern zu invertieren, wird die inverse der verwendeten Gatter in umgekehrter Reihenfolge der originalen Schaltung angewendet. Die Swap Operationen stehen somit bei der inversen Quanten-Fourier-Transformation am Anfang. In Abbildung 3 ist beispielhaft die inverse Quanten-Fourier-Transformation für drei Qubits abgebildet.

5.6 Quanten-Phase-Estimation

Im nachfolgenden Abschnitt wird die Anwendung und Funktionsweise des Quantum-Phase-Estimation Quantenalgorithmus erläutert. Die Quantum-Phase-Estimation ist ein Bestandteil einiger fortgeschrittener Quantenalgorithmen und eine integrale Komponente des spezifischen Quantenalgorithmus, der in dieser Arbeit implementiert wird. Genauer gesagt basiert der implementierte Algorithmus auf den Prinzipien der Quantum-Phase-Estimation und verwendet die selbe Methodik für einen spezialisierten Kontext.

Schwerpunktmäßig konzentriert sich die Erklärung primär auf das Verständnis der Funktionsweise der Quantum-Phase-Estimation und nimmt an, dass einige Voraussetzungen gegeben sind. Diese Voraussetzungen hängen vom spezifischen Kontext ab, in dem die Quantum-Phase-Estimation angewendet wird. Im weiteren Verlauf der Arbeit werden diese Voraussetzungen im Hinblick auf den Anwendungsfall des implementierten Quantenalgorithmus konkretisiert.

Voraussetzungen ist, dass ein Eigenvektor $|x\rangle_n$ von einer unitäre Transformation $U^{n\times n}$ bekannt ist. Wendet man die Transformation $U^{n\times n}$ auf $|x\rangle_n$ an, so gilt: $U^{n\times n}|x\rangle_n = \lambda_x |x\rangle_n$ [NC10]. Dabei erhält man, abhängig vom gewählten Eigenvektor $|x\rangle_n$, einen der Eigenwert λ_x von $U^{n\times n}$. Ein Eigenwert λ_x besitzt die Form eines Phasenfaktors: $e^{2\pi i\varphi}$ mit $0 \leq \varphi < 1$. Im Prinzip wird durch die unitäre Transformation also eine globale Phasenverschiebung auf den Eigenvektor angewendet.

Wie bereits im Kapitel zu den Grundlagen gezeigt wurde, ist es nicht möglich eine globale Phase durch eine gewöhnliche Messung der Qubits zu bestimmen. Das liegt daran, dass eine globale Phase die Amplituden eines Qubits nicht verändert und somit die Wahrscheinlichkeiten der Messergebnisse unverändert bleiben. Stattdessen muss man die Qubits manipulieren, so dass die globale Phase doch Einfluss auf die Amplituden hat.

Der Quanten-Phase-Estimation Quantenalgorithmus ist in der Lage, den Eigenwert aus $U^{n\times n}|x\rangle_n = \lambda_x |x\rangle_n$, also die Phasenverschiebungen, repräsentiert durch $\lambda_x = e^{2\pi i\varphi}$, auf die Amplitude eines anderen Qubits zu verschieben. Um λ_x auf ein anderes Qubit zu übertragen wird der Effekt des **Phase-Kickback** genutzt. Anschließend wird der Wert φ des Eigenwertes durch die inverse Quanten-Fourier-Transformation in einen messbaren Zustand überführt.

Der Phase-Kickback tritt auf, wenn eine unitäre Transformation $U^{n\times n}$ kontrolliert durch ein Qubit $|y\rangle_1$ in Superposition, auf einen Eigenvektor $|x\rangle_n$ von $U^{n\times n}$ anwendet wirkt. Dabei wird der Eigenwert, beziehungsweise λ_x, auf den $|1\rangle$-Anteil von $|y\rangle_1$ übertragen.

Sei: $|y\rangle_1 \equiv \alpha|0\rangle + \beta|1\rangle$ mit $\alpha, \beta \neq 0$, dann:
$CU^{(n+1)\times(n+1)}(|y\rangle_1 \bigotimes |x\rangle_n) = (\alpha|0\rangle + \lambda_x\beta|1\rangle) \bigotimes |x\rangle_n$.

Es folgt ein Beispiel welches den Effekt verdeutlicht: Beachtet werden zwei Qubits im Zustand $|+\rangle_1 \bigotimes |-\rangle_1$ auf die ein kontrolliertes X-Gatter angewendet wird. Dabei ist $|-\rangle_1$ der Eigenvektor einer X-Transformation mit zugehörigen Eigenwert $-$, also $X^{1\times1}|-\rangle_1 = -|-\rangle_1$.

Auf das zweite Qubit $|-\rangle_1$ wirkt ein $CX^{2\times2}$ Gatter welches durch das erste Qubit $|+\rangle_1$

kontrolliert wird:

$$CX^{2\times 2}(|+\rangle_1 \bigotimes |-\rangle_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$$

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \bigotimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = |-\rangle_1 \bigotimes |-\rangle_1$$

Mithilfe des Phase-Kickback kann man also den Eigenwert einer unitären Transformation in die Phase eines Kontrollqubits verschieben. Der Vorteil davon ist, dass diese Phasenverschiebung nur den Vorfaktor von $|1\rangle$ des Kontrollqubits betrifft und keine globale Phase darstellt.

Der Aufbau eines Quanten-Phase-Estimation Quantenschaltung sieht beispielsweise wie folgt aus:

Beachtet wird die unitäre Transformation eines Phase-Gatter(P) mit einer variablen Phasenverschiebung von $P(2i\pi\varphi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{2i\pi\varphi} \end{pmatrix}$.

Ein zugehöriger Eigenvektor dieser Transformation ist $|1\rangle_1$ den $P(2i\pi\varphi)\,|1\rangle_1 = e^{2i\pi\varphi}\,|1\rangle_1$.

Um den Effekt das Phase-Kickback nutzen zu können, muss sich das Kontrollqubit in einer Superposition befinden. Dafür wird das Kontrollqubit im Zustand $|0\rangle_1$ initialisiert und anschließend mit einem Hadamard-Gatter(H) in die gleichmäßige Superposition $|+\rangle_1$ versetzt.

$$|0\rangle_1 \bigotimes |1\rangle_1 \xrightarrow{H^{\otimes 1}} |+\rangle_1 \bigotimes |1\rangle_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \bigotimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

Über das kontrollierte Phase-Gatter wird der Eigenwert auf das Kontrollqubit verschoben und befindet sich deswegen nicht mehr im Zustand $|+\rangle_1$:

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \xrightarrow{CP^{2\times 2}(2i\pi\varphi)} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2i\pi\varphi} \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 0 \\ e^{2i\pi\varphi} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ e^{2i\pi\varphi} \end{pmatrix} \bigotimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Anschließend wird auf die Kontrollqubits die inverse Quanten-Fourier-Transformation angewendet. Die inverse Quanten-Fourier-Transformation sorgt dafür, dass der Eigenwert die Amplituden der Kontrollqubits beeinflusst. Die Qubits mit dem Eigenvektor sind für den weiteren Ablauf der Quanten-Phasen-Estimation nicht mehr relevant und werden nicht weiter beachtet. Im Beispiel entspricht die inverse Quanten-Fourier-Transformation einem Hadamard-Gatter da nur ein einzelnes Kontrollqubit existiert:

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ e^{2i\pi\varphi} \end{pmatrix} \xrightarrow{H^{\otimes 1}} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ e^{2i\pi\varphi} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 + e^{2i\pi\varphi} \\ 1 - e^{2i\pi\varphi} \end{pmatrix}$$

Anhand des Ergebnisses des Beispiels ist zu erkennen, dass der Eigenwert praktisch auf die Amplitude vom Kontrollqubit transformiert wird.

Verwendet man ein Phase-Gatter mit $\varphi = 0.075$ sieht der Quantenschaltkreis wie in Abbildung **??** aus. Mit dieser Phasenverschiebung sollte man bei einer Messung mit einer Wahrscheinlichkeit von ungefähr 0.9455 den Zustand $|0\rangle$ erhalten und $|1\rangle$ mit der Wahrscheinlichkeit 0.0544. Die Ergebnisse von 20.000 Messungen in Abbildung **??** bestätigen die Größenordnung der Wahrscheinlichkeiten. Jedoch ergeben die Messungen aus Abbildung **??** nicht ganz genau den ausgerechneten Wahrscheinlichkeiten. Dies liegt an der probabilistischen Natur der Messung. Bei einer zunehmenden Anzahl an Messungen würden die Ergebnisse an die Wahrscheinlichkeitswerte konvergieren. Somit benötigt man sehr viele Durchläufe des Quantenalgorithmus um anhand der Messungen ein verlässliches Ergebnis zu erhalten.

Es ist möglich die Präzision der Quanten-Phase-Estimation zu verbessern, indem mehr Qubits verwendet werden. Diese Qubits werden dann als weitere Kontrollqubits verwendet. Die Anzahl der Qubits für den Eigenvektor bleibt gleich der Bitanzahl, die ausreicht, um den Wert des Eigenvektors zu definieren. Jedes einzelne Kontrollqubit kontrolliert ein U^{2^x}-Gatter. Bei n Kontrollqubits kontrolliert das least-significant-bit ein U^{2^0}-Gatter, das darauffolgende ein U^{2^1}-Gatter, während das letzte Kontrollqubit ein $U^{2^{n-1}}$-Gatter kontrolliert. Dabei kann U^{2^x} als 2^x viele U-Gatter realisiert werden oder als ein einzelnes Gatter, welches den Eigenwert λ mit 2^x multipliziert anwendet. Anschließend wirkt die inverse Quanten-Fourier-Transformation auf alle Kontrollqubits. Anhand der Messung kann dann φ bestimmt werden. Der Aufbau der Schaltung ist in Abbildung **??** abgebildet.

Wird die Quanten-Fourier-Transformation wie in Abbildung **??** realisiert, kann man den Zustand der Kontrollqubits vor der inversen Quanten-Fourier-Transformation wie folgt beschreiben:

$$\frac{1}{\sqrt{N}}[(|0\rangle + CU^{2^0}|1\rangle) \bigotimes (|0\rangle + CU^{2^1}|1\rangle) \bigotimes ... \bigotimes (|0\rangle + CU^{2^{n-1}}|1\rangle)]$$

Mit $U^{n\times n}|x\rangle_n = e^{2\pi i\varphi}|x\rangle_n$ wird der Eigenwert $e^{2\pi i\varphi}$ wegen des Phase-Kickbacks über die CU-Gatter auf die Kontrollqubits übertragen:

$$\frac{1}{\sqrt{N}}[(|0\rangle + e^{2\pi i 2^0\varphi}|1\rangle) \bigotimes (|0\rangle + e^{2\pi i 2^1\varphi}|1\rangle) \bigotimes ... \bigotimes (|0\rangle + e^{2\pi i 2^{n-1}\varphi}|1\rangle)]$$

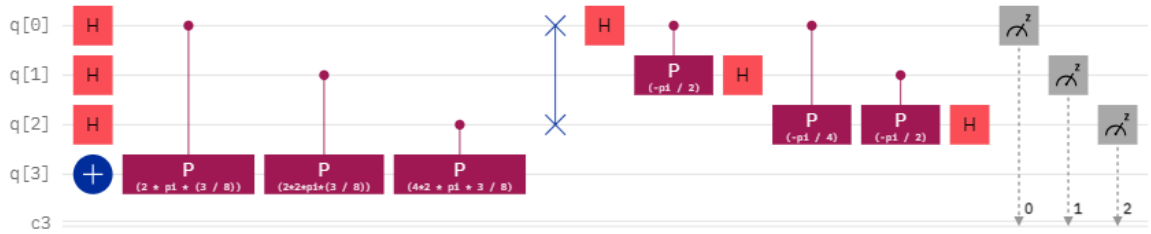
Schreibt man φ als Binärbruch:

$$\varphi = \frac{\varphi_n}{2^1} + \frac{\varphi_{n-1}}{2^2} + ... + \frac{\varphi_1}{2^n}$$

Kann die Formel in einer ähnlichen Form wie die Quanten-Fourier-Transformation umgeformt werden [NC10]:

$$\frac{1}{\sqrt{N}}[(|0\rangle + e^{2\pi i(\frac{\varphi_n}{2})}...e^{2\pi i(\frac{\varphi_2}{2^{n-1}})}e^{2\pi i(\frac{\varphi_1}{2^n})}|1\rangle)...(|0\rangle + e^{2\pi i(\frac{\varphi_2}{2})}e^{2\pi i(\frac{\varphi_1}{4})}|1\rangle) \bigotimes (|0\rangle + e^{2\pi i(\frac{\varphi_1}{2})}|1\rangle)]$$

Abbildung 4: 3-Kontroll-Qubit QPE



Die Formel besitzt die gespiegelte Struktur wie die Quanten-Fourier-Transformation ohne Swap-Gatter:

$$QFT_N \left|x\right\rangle_n = \frac{1}{\sqrt{N}}(\left|0\right\rangle + e^{2\pi i \frac{(x_1)}{2}} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{2\pi i \frac{(x_2)}{2} \frac{(x_1)}{4}} \left|1\right\rangle) ... (\left|0\right\rangle + e^{2\pi i (\frac{x_n}{2})} ... e^{2\pi i (\frac{x_2}{2^{n-1}})} e^{2\pi i (\frac{x_1}{2^n})} \left|1\right\rangle)$$

Durch die Verwendung der Swap Gatter kann die Reihenfolge der quanten-Fourier-Transformation gespiegelt werden, anschließend sind beide Formeln strukturell identisch.

Wie im Kapitell zur Quanten-Fourier-Transformation erklärt, transformiert die Quanten-Fourier-Transformation den Zustand der Eingangsqubits $\left|x\right\rangle_n$ in die Phasen der Ausgangsqubits. Hingegen kehrt die inverse Quanten-Fourier-Transformation diesen Vorgang um, indem die Phaseninformationen der Eingangsqubits in Zustände der Standartbasis transformiert werden.

Die Anwendung der inversen Quanten-Fourier-Transformation, inklusive Swap-Gatter, bewirkt also:

$$iQFT(\frac{1}{\sqrt{N}}[(\left|0\right\rangle + e^{2\pi i (\frac{\varphi_n}{2})} ... e^{2\pi i (\frac{\varphi_2}{2^{n-1}})} e^{2\pi i (\frac{\varphi_1}{2^n})} \left|1\right\rangle) ... (\left|0\right\rangle + e^{2\pi i (\frac{\varphi_2}{2})} e^{2\pi i (\frac{\varphi_1}{4})} \left|1\right\rangle) \bigotimes (\left|0\right\rangle + e^{2\pi i (\frac{\varphi_1}{2})} \left|1\right\rangle)])$$

$$= \left|\varphi_1 \varphi_2 ... \varphi_n\right\rangle_n = \left|2^n \varphi\right\rangle_n$$

Schließlich kann φ mit einer division durch 2^n bestimmt werden.

Als Beispiel wird die Quanten-Phase-Estimation für $U^{1\times1} \left|1\right\rangle_1 = e^{2\pi i \frac{3}{8}} \left|1\right\rangle_1$ also mit $\varphi = \frac{3}{8}$ beachtet. Damit der Quantenschaltkreis in Abbildung 4 möglichst gut erkennbar ist, werden die kontrollierten U^{2^x}-Gatter werden als Phase-Gatter mit $P(e^{2^x 2\pi i \frac{3}{8}})$ realisiert. Die Messung in Abbildung 5 ergibt konsistent bei allen Durchläufen den Zustand $\left|3\right\rangle_3$. Da $\left|3\right\rangle_3 = \left|2^3 \varphi\right\rangle_3$ entspricht, kann mit einer Division $\varphi = \frac{3}{8}$ bestimmt werden. Es ist zu beachten, dass die Reihenfolge der Wertigkeit der Qubits gleich bleibt. Normalerweise vertauscht die inverse wie auch die normale Quanten-Fourier-Transformation die Wertigkeiten. Jedoch wird dieser Effekt mit Swap-Gattern korrigiert.

Im vorherigen Beispiel liefern die Messungen aller Durchläufe den gleichen Zustand mit dem φ eindeutig bestimmbar ist. Diese Eindeutigkeit tritt auf wenn die verwendete Anzahl an Kontroll-Qubits ausreicht, um φ eindeutig zu repräsentieren. Im oberen

Abbildung 5: 3-C-Qubit QPE Messergebnis

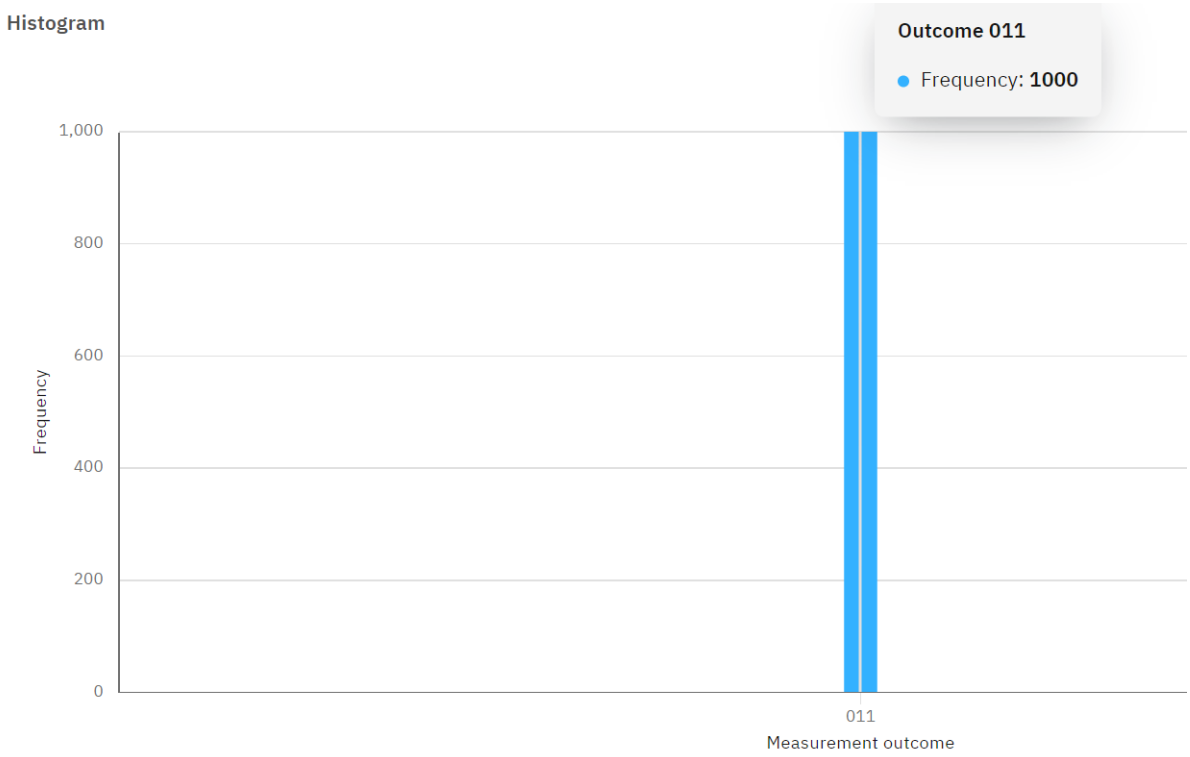
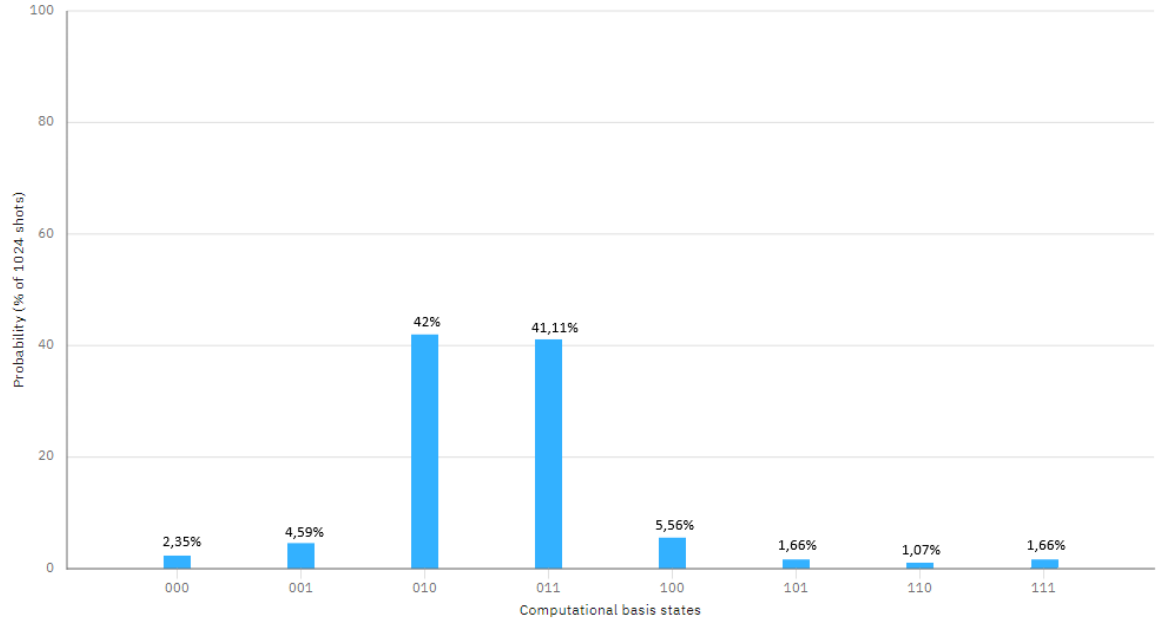


Abbildung 6: QPE unpräzises Messergebnis



Beispiel kann φ eindeutig mit den drei verwendeten Kontrollqubits dargestellt werden: $\frac{3}{8} = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}$. Verwendet man nicht ausreichend Kontroll-Qubits kann φ nicht eindeutig repräsentiert werden. Als Konsequenz wird die Messung ungenau. Mit hoher Wahrscheinlichkeit kollabieren die Qubits bei einer Messung in die darstellbaren Zustände, die den genauen Wert am besten approximieren. In Abbildung 6 sind die Messergebnisse von einer Quanten-Phase-Estimation abgebildet, die die Phase $\varphi = \frac{5}{16}$ bestimmen soll. Da $\frac{5}{16}$ nicht mit 3-Qubits darstellbar ist, gibt es kein eindeutiges Messergebnis. Anhand der Messergebnisse ist aber erkennbar, dass die Messungen mit hoher Wahrscheinlichkeit, zu den bestmöglichen Zuständen kollabieren. Diese entsprechen $\frac{2}{8} = \frac{4}{16}$ und $\frac{3}{8} = \frac{6}{16}$, also genau die Werte um $\frac{5}{16}$.

6 Shor-Algorithmus

Im folgenden Kapitel wird der Shor-Algorithmus zum Faktorisieren von Zahlen beschrieben. Der Inhalt bezieht sich auf den Zweck, die Funktionsweise und den Aufbau des Algorithmus. Der Aufbau beinhaltet nicht die konkrete Implementierung der Bestandteile des Algorithmus. Stattdessen werden die Details bezüglich der Implementierung im nächsten Kapitel behandelt.

6.1 Zweck

Der Shor-Algorithmus wurde mit dem spezifischen Ziel entwickelt, große zusammengesetzte Zahlen effizient auf Quantencomputern in ihre Primfaktoren zu faktorisieren. Im Gegensatz zu klassischen Faktorisierungsverfahren, die exponentielle Zeit erfordern [KL23], ermöglicht Shor's Ansatz die Faktorisierung in polynomialer Zeit, in Bezug auf die Anzahl an Bits der zu faktorisierenden Zahl [Sho97]. Dies stellt eine signifikante Beschleunigung gegenüber den besten bekannten klassischen Faktorisierungsverfahren dar. Der Shor-Algorithmus bekräftigt die These, das Quantencomputer bestimmte Probleme wesentlich schneller lösen können als ihre klassischen Gegenstücke.

6.2 Funktionsweise

Der Shor-Algorithmus verwendet zwei Teilberechnungen, die zusammen die Faktorisierung berechnen. Die erste Teilberechnung erfolgt mittels eines Quantenalgorithmus. Der zweite Teil basiert auf einem klassischen Algorithmus. Im quantenmechanischen Teil des Shor-Algorithmus geht es um die Bestimmung der Ordnung in der multiplikativen Gruppe modulo N. Hierbei ist anzumerken, dass der Quantenalgorithmus nicht direkt die Primfaktoren der zu faktorisierenden Zahl berechnet. Stattdessen wird die Eigenschaft ausgenutzt, dass das Problem der Faktorisierung äquivalent zu dem Problem der Ordnungsbestimmung ist [NC10]. Daher impliziert eine effiziente Lösung für die Berechnung der Ordnung eine ebenso effiziente Methode zur Faktorisierung. Der nachfolgende klassische Algorithmus verwendet die ermittelte Ordnung, um die Primfaktoren abzuleiten. Beide Teilberechnungen, sowohl die quantenmechanische als auch die klassische, führen ihre Berechnungen in polynomialer Zeit durch. Daher liegt die gesamte Laufzeit des Shor-Algorithmus ebenfalls in einer polynomialen Größenordnung.

6.3 Ordnungsbestimmung

Zu bestimmen sind die Primfaktoren der Zahl N. Zuerst wird ein a mit $0 < a < N$ gewählt. Falls der ungewöhnlichen Fall eintritt, dass a nicht teilerfremd zu N ist, entspricht a einem der Primfaktoren. Anschließend wird die Ordnung beziehungsweise Periode p der Funktion $f(x) = a^x \mod N$ mit einem Quantenalgorithmus bestimmt:

Die Periode p beschreibt das kleinste ganzzahlige Element mit $p > 0$, für das gilt: $f(p) = 1 \mod N$.

x		0	1	2	3	4	5	$\longmapsto p = 4$
7^x	mod 15	1	7	4	13	1	7	

Im Wesentlichen handelt es sich bei der quantenmechanischen Berechnung des Shor-Algorithmus um die Quanten-Phase-Estimation. Die Architektur dieser spezifischen Quanten-Phase-Estimation korrespondiert weitgehend mit der in Abschnitt 5.6 vorgestellten Struktur. Hierbei ersetzen speziell für die gegebene Anwendung definierte U-Gatter die allgemeinen.

Für den konkreten Kontext der Periodenberechnung, realisieren die U-Gatter die Transformation:

$$U\left|y\right\rangle = \left|ay \mod N\right\rangle$$

Die Ausführung der Quanten-Phase-Estimation erfordert die Erzeugung eines Eigenvektors der Transformation U. Da die Quanten-Phase-Estimation φ aus dem Eigenwert extrahiert, darf der Eigenvektor nicht den trivialen Eigenwert 1 besitzen. Stattdessen ist es notwendig, dass die Periode der Transformation im Eigenwert enthalten ist.

Wie in [NC10, S. 227] gezeigt wird, gibt es zu U Eigenvektoren $\left|u_s\right\rangle$, mit $0 \leq s \leq p-1$:

$$\left|u_s\right\rangle \equiv \frac{1}{\sqrt{p}} \sum_{k=0}^{p-1} e^{\frac{-2\pi i s k}{p}} \left|a^k \mod N\right\rangle$$

Für $a = 7$ bei $N = 15$ entspricht $r = 4$. Ein Eigenvektor zu $s = 1$ lautet dann:

$$\left|u_1\right\rangle_4 = \frac{1}{\sqrt{4}}(\left|1\right\rangle_4 + e^{-\frac{2\pi i}{4}} \left|7\right\rangle_4 + e^{-\frac{4\pi i}{4}} \left|4\right\rangle_4 + e^{-\frac{6\pi i}{4}} \left|13\right\rangle_4)$$

$$U\left|u_1\right\rangle_4 = \frac{1}{\sqrt{4}}(\left|7\right\rangle_4 + e^{-\frac{2\pi i}{4}} \left|4\right\rangle_4 + e^{-\frac{4\pi i}{4}} \left|13\right\rangle_4 + e^{-\frac{6\pi i}{4}} \left|1\right\rangle_4)$$

$$U\left|u_1\right\rangle_4 = e^{\frac{2\pi i}{4}} \cdot \frac{1}{\sqrt{4}}(e^{-\frac{2\pi i}{4}} \left|7\right\rangle_4 + e^{-\frac{4\pi i}{4}} \left|4\right\rangle_4 + e^{-\frac{6\pi i}{4}} \left|13\right\rangle_4 + e^{-\frac{8\pi i}{4}} \left|1\right\rangle_4)$$

$$U\left|u_1\right\rangle_4 = e^{\frac{2\pi i}{4}} \cdot \frac{1}{\sqrt{4}}(e^{-\frac{8\pi i}{4}} \left|1\right\rangle_4 e^{-\frac{2\pi i}{4}} \left|7\right\rangle_4 + e^{-\frac{4\pi i}{4}} \left|4\right\rangle_4 + e^{-\frac{6\pi i}{4}} \left|13\right\rangle_4) = e^{\frac{2\pi i}{4}} \cdot \left|u_1\right\rangle_4$$

Das kann verallgemeinert werden:

$$U\left|u_s\right\rangle = e^{\frac{2\pi i s}{p}} \left|u_s\right\rangle$$

Wie man in der Definition von $\left|u_s\right\rangle$ sieht, benötigt die Initialisierung eines Eigenvektors $\left|u_s\right\rangle$ mit einem konkreten s die Periode p. Man kann diese Problematik jedoch umgehen indem man anstelle eines einzelnen Eigenvektors $\left|u_s\right\rangle$ eine Superposition verwendet, die alle $\left|u_s\right\rangle$ umfasst. Die Superposition entspricht [NC10, S. 227]:

$$\frac{1}{\sqrt{p}} \sum_{s=0}^{r-1} \left|u_s\right\rangle = \left|1\right\rangle$$

Also wird das Qubit, welches das Least-Significant-Bit des für den Eigenvektor bestimmten Qubit-Registers repräsentiert, mit $\left|1\right\rangle$ initialisiert.

Die Superposition der Eigenvektoren hat zur Folge, dass nach der inversen Quanten-Fourier-Transformation, also am Ende der Quanten-Phase-Estimation, ebenfalls eine Superposition mit dem φ_s der Eigenwerte λ_{u_s} aller möglichen Eigenvektoren $\left|u_s\right\rangle$ existiert. Sei k die Anzahl an Kontroll-Qubits dann:

$$\frac{1}{\sqrt{p}} \sum_{s=0}^{p-1} \left|2^k \cdot \frac{s}{p}\right\rangle_k = \frac{1}{\sqrt{p}}(\left|0\right\rangle_k + \left|2^k \cdot \frac{1}{p}\right\rangle_k + \left|2^k \cdot \frac{2}{p}\right\rangle_k ... + \left|2^k \cdot \frac{p-1}{p}\right\rangle_k)$$

Bei einer Messung wird also zufällig eines der insgesamt p vielen $\varphi_s \approx \frac{s}{p}$ gemessen.

Die Genauigkeit des Algorithmus, ausgedrückt durch den Parameter k, ist direkt abhängig von der Anzahl der verwendeten U^{2^x}-Gatter und der damit assoziierten Kontroll-Qubits im Quantenschaltkreis. Die Genauigkeit des Messergebnisses gegenüber dem φ_s hängt von k ab.

Wenn genügend Kontroll-Qubits verwendet werden, wird bei einer Messung ein Zustand gemessen, in welchem $\frac{s}{p}$ enthalten ist.

Falls nicht ausreichend Kontroll-Qubits vorhanden sind und der Wert von φ_s somit nicht ausreichend umfasst werden kann, wird die Messung mit hoher Wahrscheinlichkeit in einen der möglichen Zustand kollabieren, der dem genauen Ergebnis am nächsten ist. Dadurch bilden sich ein Peak rund um den korrekten Wert, welcher Vergleichbar ist mit Abbildung 6. Genau wie in Abbildung 6, haben die Werte nah dem genauen Wert eine höhere Chance gemessen zu werden. Da aber eine Messung von probabilistischer Natur ist, kann eine Messung auch zu einem Ergebnis führen, welches entfernt vom genauen Ergebnis liegt.

Anders als in Abbildung 6 gibt es bei der Periodenbestimmung insgesamt p Peaks. Das liegt daran, dass es insgesamt p viele Zustände in der Superposition des Endzustandes gibt.

Anhand des Messergebnisses erfolgt die Primfaktorzerlegung in einer Nachberechnung.

6.4 Klassische Nachberechnung

Die Nachberechnung benötigt keine Quantenbrechnung und wird somit mit einem klassischen Algorithmus durchgeführt.

Ein einzelner Zustand der finalen Superposition, entspricht einem genauen Ergebnis und besitzt die Form $\left|2^k \cdot \frac{s}{p}\right\rangle_k$ für $0 \leq s < p$. Mit einer Division durch 2^k kann $\frac{s}{p}$ extrahiert werden. Zum Beispiel ist die finale Superposition für $a = 7;\ N = 15$ mit $p = 4$ bei einer Genauigkeit $k = 4$:

$$\frac{1}{\sqrt{4}}(|0\rangle_4 + \left|2^4 \cdot \frac{1}{4}\right\rangle_4 + \left|2^4 \cdot \frac{2}{4}\right\rangle_4 + \left|2^4 \cdot \frac{3}{4}\right\rangle_4) = \frac{1}{\sqrt{4}}(|0\rangle_4 + |4\rangle_4 + |8\rangle_4 + |12\rangle_4)$$

Mit Ausnahme der Zustände $|0\rangle_4$ und $|8\rangle_4$, gewähren die beiden anderen Zustände nach einer Division mit 2^4 eine Kommazahl, die als Bruch die gesuchte Periode $p = 4$ als Nenner enthält.

Im vorherigen Beispiel handelt es sich bei der Periode um eine 2er Potenz, weswegen die Messergebnisse eindeutig sind. Jedoch ist dies ein ideales Szenario welches bei immer größeren N seltener auftretet. Das nächste Beispiel beachtet $a = 11;\ N = 21$ mit der Periode $p = 6$ bei einer Genauigkeit $k = 4$:

$$\frac{1}{\sqrt{6}}(|0\rangle_4 + \left|2^4 \cdot \frac{1}{6}\right\rangle_4 + \left|2^4 \cdot \frac{2}{6}\right\rangle_4 + \left|2^4 \cdot \frac{3}{6}\right\rangle_4) + \left|2^4 \cdot \frac{4}{6}\right\rangle_4 + \left|2^4 \cdot \frac{5}{6}\right\rangle_4$$

$$= \frac{1}{\sqrt{6}}(|0\rangle_4 + |2,\overline{6}\rangle_4 + |5,\overline{3}\rangle_4 + |8\rangle_4 + |10,\overline{6}\rangle_4 + |13,\overline{3}\rangle_4)$$

Die Kontroll-Qubits, welche gemessen werden, können keine Kommazahlen darstellen. Als Folge davon, bildet sich um die benachbarten Zustände eine Wahrscheinlichkeitsverteilung, die bei dem ganzzahligen Zustände, der am nächst liegt, ihren Hochpunkt hat, vergleichbar mit 6. Im konkreten Beispiel wird für $|2,\overline{6}\rangle_4$ der Zustand $|3\rangle_4$ mit der höchsten Wahrscheinlichkeit gemessen. Eine Extraktion der Periode aus $|3\rangle_4$ mit $\frac{3}{2^4}$ liefert als Ergebnis die Kommazahl $0,1875$. Mit dem Kettenbruch-Algorithmus wird zu der Kommazahl der nächstgelegene Bruch gefunden, im konkreten Fall die $\frac{3}{16}$. Der Bruch $\frac{3}{16}$ enthält im Nenner nicht die gesuchte Periode $p = 6$. Das Gleiche gilt auch für andere Zustände, die mit hoher Wahrscheinlichkeit gemessen werden können, wie $|5\rangle_4$, $|8\rangle_4$, $|11\rangle_4$ und $|13\rangle_4$.

Das Versagen der vorausgegangenen Berechnung ist auf eine unzureichende Genauigkeit zurückzuführen. In der vorangegangenen Darstellung werden $k = 4$ Kontroll-Qubits eingesetzt, welche für die gesuchte Periode einen zu kleinen Wertebereich bilden.

Wiederholt man das ganz, mit $k = 7$, erhält man folgende Superposition:

$$\frac{1}{\sqrt{6}}(|0\rangle_7 + |21,\overline{3}\rangle_7 + |42,\overline{6}\rangle_7 + |64\rangle_7 + |85,\overline{3}\rangle_7 + |106,\overline{6}\rangle_7)$$

Es ist ersichtlich, dass die genauen Zustände weiterhin über Nachkommastellen verfügen. Weswegen mit einer geringeren Wahrscheinlichkeit auch andere Werte gemessen werden können. Bei einer Messung ist statt des Zustands $|21,\overline{3}\rangle_7$ der Zustand $|21\rangle_7$ das wahrscheinlichste Ergebnis. Für $\frac{21}{2^7}$ erhält man eine Kommazahl, deren genauer Bruch $\frac{21}{128}$ entspricht. Bekanntlich entspricht die Periode jedoch einer Zahl p die kleiner ist als N. Wendet man auf $\frac{21}{128}$ den Kettenbruch-Algorithmus an und limitiert den Nenner auf N, wird der nächstgelegene Bruch gefunden, welcher ein Nenner besitzt, der kleiner ist als N. Im konkreten Beispiel folgt für $\frac{21}{128}$ der Bruch $\frac{1}{6}$, welcher die Periode im Nenner offenbart. Bei einer Messung kann auch der benachbarte ganzzahlige Zustand $|22\rangle_7$ gemessen werden, allerdings mit einer geringeren Wahrscheinlichkeit im Vergleich zu $|21\rangle_7$. Der Zustand $|22\rangle_7$ ermöglicht nicht die Extraktion der Phase, da $\frac{22}{128}$ mit dem Kettenbruch-Algorithmus auf einen für N limitierten Nenner zu $\frac{3}{17}$ wird.

Bedeutet mit einer hinreichenden Genauigkeit, kann die Periode nur aus dem Zustand extrahiert werden, welcher den Wert des genauen Zustandes am besten approximiert.

Mit zusätzlichen Kontroll-Qubits wird das Messergebnis Fehlertolerant. Wiederholt man das Beispiel erneut, diesmal jedoch mit $k = 10$, ist der zweite Zustand in der finalen Superposition: $|2^{10} \cdot \frac{1}{6}\rangle_{10} = |170,\overline{6}\rangle_{10}$. Im Gegensatz zu vorher liefert jetzt nicht nur das am besten approximierte Ergebnis, in dem Fall die $|170\rangle_{10}$, mit dem Kettenbruch-Algorithmus die Periode, sondern auch alle benachbarten Zustände von $|167\rangle_{10}$ bis $|175\rangle_{10}$.

Durch den Einsatz von zusätzlichen Kontroll-Qubits erhöht sich der Wertebereich. Aufgrund des größeren Wertebereiches existieren mehr mögliche Zustände, die bei einer

Messung vorkommen könnten und ebenfalls auch mehr Zustände, die die Extraktion der korrekten Periode ermöglichen. Dabei ist die Wahrscheinlichkeitsverteilung der ausschlaggebende Faktor. Diese verändert sich mit einer wachsenden Anzahl an Kontroll-Qubits zugunsten der Zustände, welche die korrekte Periode offenbaren. Die Abbildungen 7 und 8 heben den Effekt zusätzlicher Kontroll-Qubits auf die Wahrscheinlichkeitsverteilung der Messungen hervor. Die grünen Säulen im Diagram zeigen die Messergebnisse, welche die Extraktion der Periode erlauben.

Abbildung 7: Messergebnisse für a=11, N=21, k=7 bei 1024 Messungen

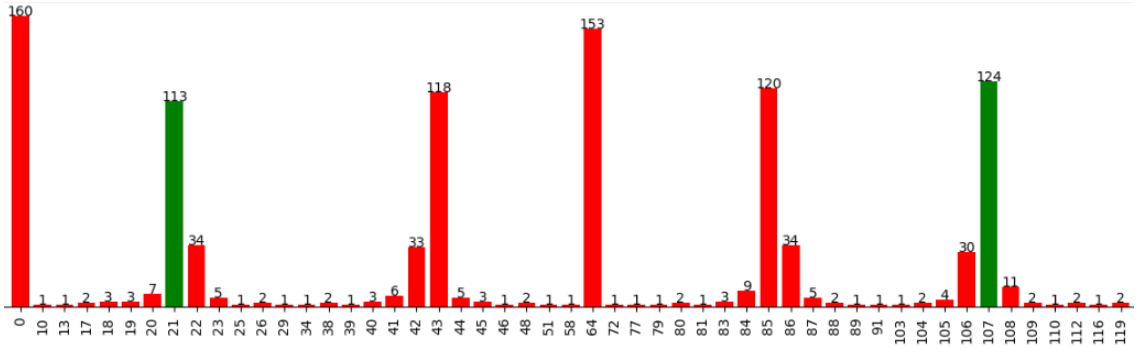
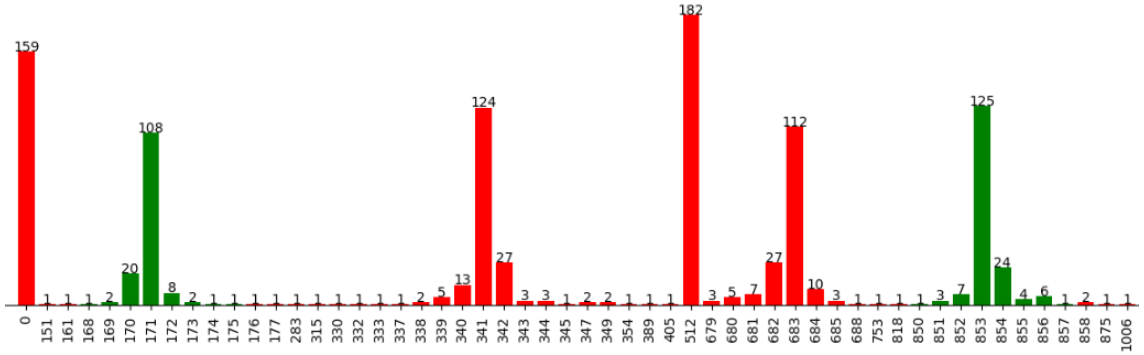


Abbildung 8: Messergebnisse für a=11, N=21, k=10 bei 1024 Messungen



Für eine bekannte Periode, kann man zeigen, dass die Verwendung von $k > 2\text{ld}(p)$ eine ausreichende Genauigkeit liefert, so dass ein Messergebnis unter Verwendung des Kettenbruch-Algorithmus zu einem Näherungsbruch $\frac{s}{p}$ von φ_s konvergiert [Sho97]. Im Normalfall kann für eine Elementes a für den Modulus N vorab nicht gesagt werden, wie viele Kontroll-Qubits man exakt braucht, da dies von der unbekannten Periode p abhängt. Stattdessen, verwendet man einen Wert der auf alle Fälle größer als $2\text{ld}(p)$ ist, wie zum Beispiel $2\text{ld}(N)$ [Sho97][ME99].

Wie man in Abbildungen 8 sieht, gibt es auch bei der Verwendung von $2\text{ld}(N)$ Kontroll-Qubits drei Szenarien, bei denen eine Zustand mit einer ungültigen Periode gemessen werden kann.

Im Falle des ersten Szenarios, entspricht mindestens einer der Zustände der finalen Superposition einer Kommazahl. Dadurch kann es zu Messungen von falschen Werten kommen. Anhand dieses Ergebnisses führt der Kettenbruch-Algorithmus nicht zu $\frac{s}{p}$, weswegen eine neue Messung benötigt wird.

Das zweite Szenario ähnelt dem ersten und tritt ebenfalls auf, sobald mindestens einer der Zustände der finalen Superposition eine Kommazahl ist. Im Unterschied zum ersten Szenario liegt dieses Messergebnis jedoch nah bei einem Peak, also in der Nähe eines Messergebnisses, aus dem die Periode extrahiert werden kann. Indem man die Nachbarzustände des Messergebnisses testet, kann man gegebenenfalls einen Zustand finden, der die Extraktion der Periode ermöglicht.

Das zweite Szenario tritt beispielsweise für das Messergebnis $|176\rangle_{10}$ in Abbildung 8 auf. Die Anwendung des Kettenbruch-Algorithmus auf $\frac{176}{2^{10}}$ liefert nicht die korrekte Periode, stattdessen ergibt sich der Bruch $\frac{3}{17}$. Anstatt eine neue Messung durchzuführen, ist es in diesem Fall zielführend, die ganzzahligen Nachbarn des Messergebnisses zu überprüfen. Im konkreten Beispiel ermöglicht bereits die geringfügige Anpassung des Messergebnisses durch Subtraktion von 1, also zu $|175\rangle_{10}$, die erfolgreiche Extraktion der Periode.

Die Erhöhung der Genauigkeit k auf Werte über $2\mathrm{ld}(N)$ hinaus hat den Effekt, dass die Wahrscheinlichkeit für das Eintreten des ersten oder zweiten Falls abnimmt. Dadurch verbessern sich gleichzeitig die Wahrscheinlichkeit, direkt ein korrektes Ergebnis zu messen. Allerdings führt dies zu einem komplexeren Quantenschaltkreis und somit erhöhen sich die Kosten an zusätzlichen Ressourcen [NC10, S. 231].

Der dritte und letzte Fall kann immer auftreten. In diesem Szenario wird ein $\frac{s'}{p'}$ gemessen, welches entsteht, wenn s und p einen gemeinsamen Teiler haben und deswegen von $\frac{s}{p}$ auf $\frac{s'}{p'}$ gekürzt wurden. In diesem Fall kann ein vielfaches von p', also $2p'$,$3p'$,.. zum richtigen p führen. Des Weiteren besteht die Möglichkeit, dass bei einer zweiten Messung erneut ein gekürztes $\frac{s}{p}$ als $\frac{s''}{p''}$ gefunden wird. Wenn s' und s'' keine Faktoren teilen, kann aus dem kleinsten gemeinsamen Vielfachen von p' und p'' das p berechnet werden [Sho97].

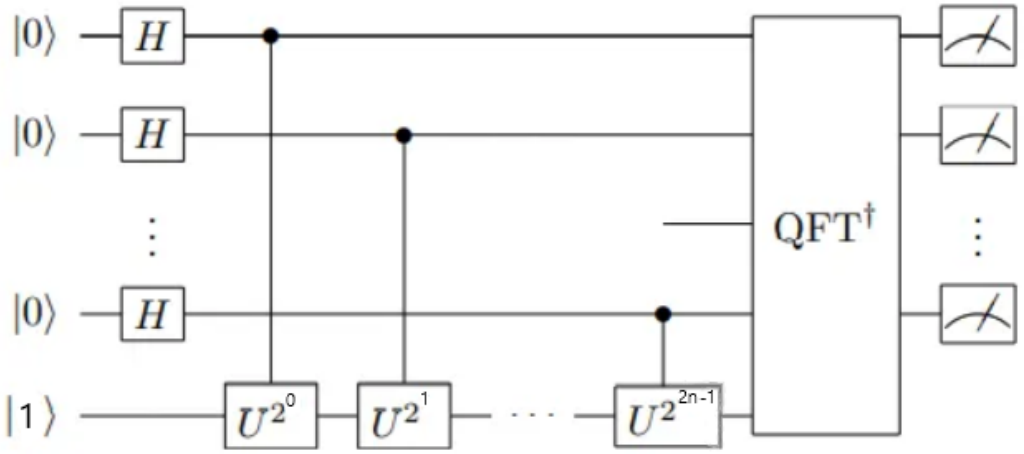
In Abbildung 8 sind die beiden Peaks um die Zustände $|341\rangle_{10}$ und $|683\rangle_{10}$ rot gefärbt, was auf den dritten Fall zurückzuführen ist. Sowohl $|341\rangle_{10}$ als auch $|683\rangle_{10}$ und ihre benachbarten Zustände enthalten ursprünglich im Nenner die korrekte Periode, wurden jedoch gekürzt. Die Brüche wurden auf $\frac{1}{3}$, beziehungsweise $\frac{2}{3}$, gekürzt. Eine Überprüfung der nächsten Vielfachen, in diesem Fall $2 \cdot 3 = 6$, führt zur korrekten Periode.

Man kann zeigen, dass man mit mindestens einer Wahrscheinlichkeit von $\frac{1}{2\mathrm{ld}(N)}$, bei einer Messung ein s bekommt, welches eine Primzahl ist und somit zu p teilerfremde ist. Das bedeutet, dass bei insgesamt $2\mathrm{ld}(N)$ Messungen, die Wahrscheinlichkeit sehr hoch ist, mindestens einmal ein $\frac{s}{p}$ zu messen, bei dem s und p nicht gekürzt sind [NC10, S. 231].

Ob die korrekte Periode gefunden wurde, kann mit $a^p = 1 \mod N$ geprüft werden.

Sobald die korrekte Periode p gefunden wurde, können die Primfaktoren von N mit

Abbildung 9: QPE für Shor [Ano23]



dem größten gemeinsamen Teiler(gcd) berechnet werden:

$$gcd(a^{\frac{p}{2}} - 1, N), gcd(a^{\frac{p}{2}} + 1, N)$$

Dies schlägt nur fehl falls p ungerade ist oder falls $a^{\frac{p}{2}} = -1 \mod N$ erfüllt [Sho97]. Die Wahrscheinlichkeit dass einer der beiden genannten Fälle eintritt beträgt $1 - \frac{1}{2^f}$, wobei f die Anzahl an unterschiedlicher Primfaktoren von N angibt [Sho97]. In einem solchen Fall, wiederholt man die Periodenberechnung mit einem anderen a.

7 Implementierung

7.1 Quantenalgorithmus

Wie im Kapitel 6.2 zur Funktionsweise erklärt, wird für die Periodenbestimmung die Quanten-Phase-Estimation genutzt.

Um den Quanten-Phase-Estimation Algorithmus für die Periodenberechnung zu nutzen, benötigt man ein Gatter U welches die modulare Multiplikation $U|y\rangle = |ay \mod N\rangle$, als eine unitär Transformation realisiert. Mit den passenden U-Gatter wird der Quantenschaltkreis wie in Abbildung 9 strukturiert.

Die Realisierung der Transformation bedingt die Implementierung einiger arithmetischer Operationen in Form eines Quantenschaltkreises. Diese fungieren als Bausteine, die zusammengesetzt zur Konstruktion des übergeordneten Quantenschaltkreises für die modulare Multiplikation beitragen. Zu den erforderlichen arithmetischen Operationen gehört die Addition, Subtraktion sowie die modulare Addition.

In den folgenden Abschnitten werden die untergeordneten arithmetischen Operationen bis hin zur modularen Multiplikation implementiert.

7.1.1 Addition

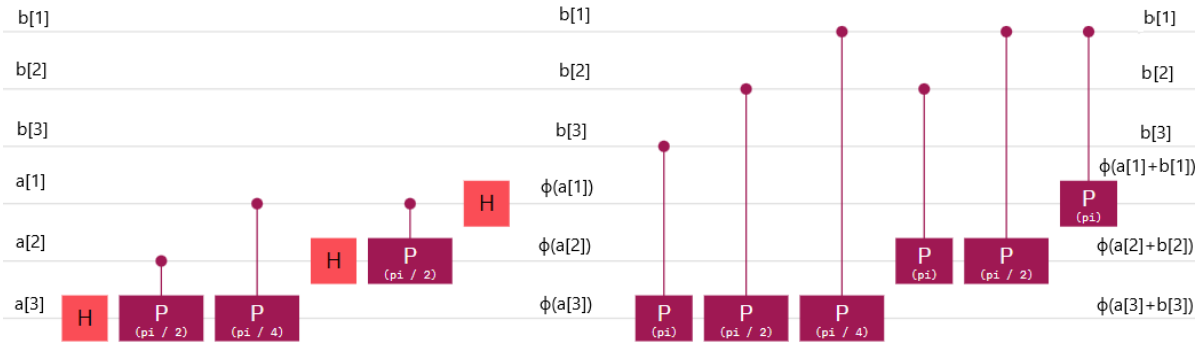
Der Quantenschaltkreis für die Addition bildet das Fundament der U-Gatter und stellt einen der am häufigsten verwendeten Bausteine dar. Deswegen hat die Implementierung der Addition einen erheblichen Einfluss auf den Ressourcenbedarf des gesamten Quantenalgorithmus und sollte daher möglichst effizient implementiert werden.

Eine Möglichkeit, die Addition als Quantenschaltkreis zu realisieren, besteht im Nachbau eines klassischen Schaltkreises aus Volladdierern. Da es nicht möglich ist, die notwendige klassischen Gatter wie AND und OR als unitäre Transformation mit nur zwei Qubits darzustellen [Hoe23b], werden zusätzliche Hilfsqubits benötigt. Die zusätzlichen Hilfsqubits bewirken, dass der Nachbau eines klassischen Schaltkreis für die Addition zweier n-Bit Zahlen, also solche der Größenordnung 2^n, mindestens $3n$ Qubits benötigt [Zal98].

Eine effizientere Methode, die ohne Hilfsqubits auskommt, ist die Quanten-Addition [Dra00]. Die Quanten-Addition führt die Berechnung auf quantenmechanische Weise durch. Im Wesentlichen wird dabei die Addition in der Fourier-Basis berechnet, wobei die Phasen der Qubits eines Summanden mit kontrollierte Phasenverschiebungen auf die Qubits des anderen Summanden wirkt.

Im Folgenden wird ein Beispiel für die Quanten-Addition zweier Qubit-Register $|a\rangle_3$ und $|b\rangle_3$, jeweils bestehend aus drei Qubits, betrachtet:

Abbildung 10: Quantum-Addition



Die Registermarkierungen in der Mitte von Abbildung 10 unterteilen die Darstellung in zwei Hälften. Die linke Hälfte repräsentiert die Quanten-Fourier-Transformation, während die rechte Hälfte die Quanten-Addition zeigt.

Wie man an der Struktur der Quanten-Addition erkennen kann, ist die Anordnung der Gatter fast identisch mit der Quanten-Fourier-Transformation. Ein Unterschied besteht darin, dass die Hadamard-Gatter durch kontrollierte $P(\pi)$ Phasen-Gatter ersetzt

wurden. Sowohl das Hadamard-Gatter als auch das $P(\pi)$ Phasen-Gatter erzeugen eine relative Phase von $e^{\pi i}$. Ein weiterer Unterschied zur Quanten-Fourier-Transformation besteht darin, dass die Phasen-Gatter nicht durch das gleiche Register$|a\rangle_3$ kontrolliert werden, auf das die Gatter auch wirken. Stattdessen kontrollieren die Qubits des Registers $|b\rangle_3$ die Phasen-Gatter. Dabei wird das $P(\pi)$ Phasen-Gatter durch das Qubit des $|b\rangle_3$ kontrolliert, welches die selbe Wertigkeit hat wie das Zielqubit des $|a\rangle_3$ Registers. Jedes weitere kontrollierte Phasen-Gatter für das gleiche Zielqubit wird fortlaufend von dem nächstkleineren Qubit von $|b\rangle_3$ kontrolliert.

Im Prinzip handelt es sich bei dieser Quantenschaltung um eine Anwendung derselben Phasenverschiebungen wie bei der Quanten-Fourier-Transformation. Der grundlegende Unterschied liegt darin, dass diese Phasenverschiebungen kontrolliert auf ein anderes Quantenregister angewendet werden.

Die Wirkung der Quanten-Addition wird anhand der Abbildung 10 verdeutlicht: Am Anfang der linken Hälfte befinden sich beide Register in der Standardbasis. Auf das Zielregister $|a\rangle_3$ wirkt die Quanten-Fourier-Transformation ohne Swap Gatter. Dadurch befindet sich $|a\rangle_3$ nun in der Fourier-Basis Φ, also $|\Phi(a)\rangle_3$:

$$|\Phi(a)\rangle_3 = \frac{1}{\sqrt{8}}[(|0\rangle+e^{\frac{2\pi i(2^0a_1)}{2^1}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^1a_2+2^0a_1)}{2^2}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^2a_3+2^1a_2+2^0a_1)}{2^3}}|1\rangle)]$$

Anschließend wirkt auf das hinterste Tensorprodukt ein $P(\pi)$ Phasen-Gatter, welches durch $|b_3\rangle_1$ kontrolliert wird. Wenn sich $|b_3\rangle_1$ im Zustand $|0\rangle_1$ befindet, passiert nichts. Wenn es sich im Zustand $|1\rangle_1$ befindet, dass das Phasen-Gatter angewendet wird. Dieses Verhalten kann man für beide Fälle mit den entsprechenden Matrizen $\begin{pmatrix}1 & 0\\ 0 & e^{\pi i b_3}\end{pmatrix}$ beziehungsweise $\begin{pmatrix}1 & 0\\ 0 & e^{\frac{2\pi i(2^2b_3)}{2^3}}\end{pmatrix}$ beschreiben. Schreibt man das hinterste Tensorprodukt als Vektor, ergibt sich die folgende Formulierung:

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{\frac{2\pi i(2^2a_3+2^1a_2+2^0a_1)}{2^3}}|1\rangle) \equiv \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ e^{\frac{2\pi i(2^2a_3+2^1a_2+2^0a_1)}{2^3}}\end{pmatrix}$$

Dann wird durch das Ergebnis der Verrechnung mit dem Phasen-Gatter deutlich, dass die Addition im Wesentlichen in der Phase des Quantenzustands stattfindet:

$$\begin{pmatrix}1 & 0\\ 0 & e^{\frac{2\pi i(2^2b_3)}{2^3}}\end{pmatrix}\cdot\frac{1}{\sqrt{2}}\begin{pmatrix}1\\ e^{\frac{2\pi i(2^2a_3+2^1a_2+2^0a_1)}{2^3}}\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ e^{\frac{2\pi i(2^2(a_3+b_3)+2^1a_2+2^0a_1)}{2^3}}\end{pmatrix}$$

Wie in der Abbildung 10 erkenntlich, wirken auf das hinterste Tensorprodukt auch noch die beiden Phasen-Gatter $P(\frac{\pi}{2})$ und $P(\frac{\pi}{4})$ mit:

$$P(\frac{\pi}{2}) = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{\pi}{2}ib_2}\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{2\pi i(2^1b_2)}{2^3}}\end{pmatrix}\ ;\ P(\frac{\pi}{4}) = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{\pi}{4}ib_1}\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{2\pi i(2^0b_1)}{2^3}}\end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i(2^0 b_1)}{2^3}} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i(2^1 b_2)}{2^3}} \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ e^{\frac{2\pi i(2^2(a_3+b_3)+2^1 a_2+2^0 a_1)}{2^3}} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ e^{\frac{2\pi i(2^2(a_3+b_3)+2^1(a_2+b_2)+2^0(a_1+b_1))}{2^3}} \end{pmatrix}$$

Wendet man alle weiteren Phasen-Gatter auf das vollständige Tensorprodukt an, erhält man:

$$\frac{1}{\sqrt{8}}[(|0\rangle+e^{\frac{2\pi i(2^0(a_1+b_1))}{2^1}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^1(a_2+b_2)+2^0(a_1+b_1))}{2^2}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^2(a_3+b_3)+2^1(a_2+b_2)+2^0(a_1+b_1))}{2^3}}|1\rangle)]$$

Setzt man in diese Formel zwei Zahlen in Binärschreibweise ein, wird man den selben Zustand erhalten, wie wenn man die Summe der beiden Zahlen in die Formel der Quanten-Fourier-Transformation einsetzt. Beispielsweise sei $a = 3$ also binär $a_3 = 0$, $a_2 = 1$,$a_1 = 1$ und $b = 1$ also $b_3 = 0$, $b_2 = 0$,$b_1 = 1$:

$$\frac{1}{\sqrt{8}}[(|0\rangle+e^{\frac{2\pi i(2^0(1+1))}{2^1}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^1(1+0)+2^0(1+1))}{2^2}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^2(0+0)+2^1(1+0)+2^0(1+1))}{2^3}}|1\rangle)]$$

$$= \frac{1}{\sqrt{8}}[(|0\rangle+|1\rangle)\bigotimes(|0\rangle+|1\rangle)\bigotimes(|0\rangle+e^{\pi i}|1\rangle)]$$

Das dies tatsächlich die Summe in Fourier-Basis entspricht, wird deutlich wenn man das selbe Tensorprodukt aus der Quanten-Fourier-Transformation bildet:

$$QFT(|c\rangle_3)\frac{1}{\sqrt{8}}[(|0\rangle+e^{\frac{2\pi i(2^0(c_1))}{2^1}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^1(c_2)+2^0(c_1))}{2^2}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^2(c_3)+2^1(c_2)+2^0(c_1))}{2^3}}|1\rangle)]$$

Die Summe von a und b entspricht $c = 4$ also $c_3 = 1,\ c_2 = 0,\ c_1 = 0$:

$$QFT(|4\rangle_3)\frac{1}{\sqrt{8}}[(|0\rangle+e^{\frac{2\pi i(2^0(0))}{2^1}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^1(0)+2^0(0))}{2^2}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^2(1)+2^1(0)+2^0(0))}{2^3}}|1\rangle)]$$

$$= \frac{1}{\sqrt{8}}[(|0\rangle+e^{\frac{2\pi i(0)}{2^1}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(0)}{2^2}}|1\rangle)\bigotimes(|0\rangle+e^{\frac{2\pi i(2^2(1))}{2^3}}|1\rangle)]$$

$$= \frac{1}{\sqrt{8}}[(|0\rangle+|1\rangle)\bigotimes(|0\rangle+|1\rangle)\bigotimes(|0\rangle+e^{\pi i}|1\rangle)]$$

Das Ergebnis der Quanten-Addition zweier Summanden $a = 3,\ b = 1$, jeweils in einem Register mit drei Qubits, ist somit identisch mit dem Zustand, der durch die Anwendung der Quanten-Fourier-Transformation auf ein Register aus ebenfalls drei Qubits mit der Summe der beiden Zahlen entsteht.

Mit einer anschließenden inversen Quanten-Fourier-Transformation, kann die Summe in die Standardbasis und somit in einen Messbaren Zustand transformiert werden:

$$iQFT(|\Phi(4)_3\rangle) \equiv iQFT(\frac{1}{\sqrt{8}}[(|0\rangle+|1\rangle)\bigotimes(|0\rangle+|1\rangle)\bigotimes(|0\rangle+e^{\pi i}|1\rangle)]) = |4\rangle_3$$

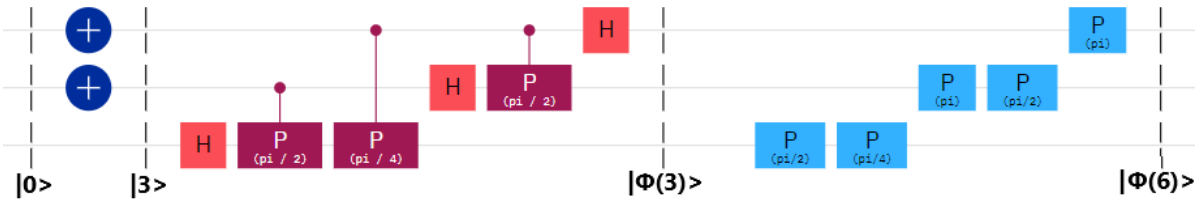
Das Zielregister sollte aus genügend Qubits bestehen, damit die Summe vollständig erfasst werden kann. Andernfalls kommt es zum overflow mit $a+b \mod 2^n$, wobei n die Anzahl an Qubits des Zielregisters beschreibt [Bea03].

Für die Realisierung der modularen Multiplikation wird zu keinem Zeitpunkt der Berechnung eine Addition zweier Zwischenergebnisse benötigt. Genauer gesagt, ist es nicht nötig, ein Quantenregister auf ein anderes zu addieren. Stattdessen wird die Quanten-Addition benutzt, um eine vorab bekannte Zahl auf ein Quantenregister zu addieren.

Bei der Quanten-Addition mit zwei Register wie in Abbildung 10 erfolgt die Phasenverschiebung kontrolliert, also in Abhängigkeit des Inhaltes von Register $|b\rangle_3$. Ist der Inhalt von Register $|b\rangle_3$ vorab bekannt, können gewöhnliche Phasen-Gatter anstelle von kontrollierten verwendet werden [Bea03].

Wenn bei der Quanten-Addition mit zwei Registern ein Phasen-Gatter aufgrund des zugehörigen Kontrollqubits im Zustand $|1\rangle$ angewendet wird, wird es in der Variante mit einem einzelnen Register als gewöhnliches Phasen-Gatter verwendet. Ist das Kontrollqubit hingegen im Zustand $|0\rangle$, wodurch das Phasen-Gatter bei der Quanten-Addition mit zwei Registern nicht zur Anwendung kommt, wird dieses Phasen-Gatter in der Variante mit nur einem Register weggelassen.

Abbildung 11: Quantum-Addition fixierte Phasenverschiebungen

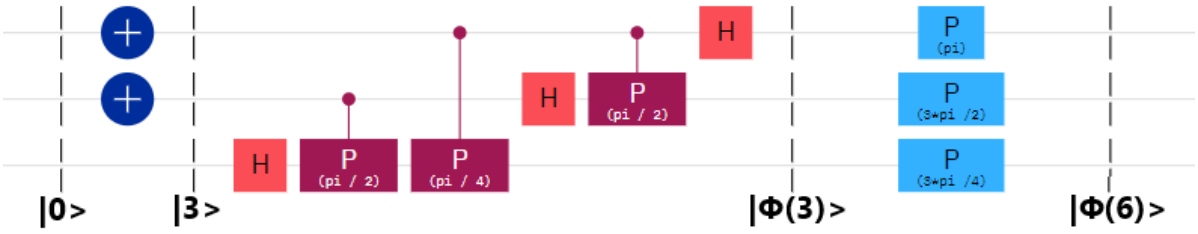


In Abbildung 11 ist die Quanten-Addition für ein 3-Qubit Register abgebildet. Die blauen Phasen-Gatter sorgen für die Quanten-Addition mit einem fixierten Wert von 3. Vergleicht man die Abbildung 11 mit der Abbildung 10 fällt auf, dass das aller erste Phasen-Gatter der Quanten-Addition nicht vorkommt. Im Quantenschaltkreis der Abbildung 10 würde ein Registerinhalt von $b = 3$ das Kontrollqubit b_3 nicht setzen. Somit kommt das erste Phasen-Gatter der Quanten-Addition nicht zum Einsatz und wird deswegen in der Variante wie in Abbildung 11 weggelassen.

Des weiteren ist es möglich, den Quantenschaltkreis der Quanten-Addition für eine vorab festgelegte Zahl ressourcensparender zu realisieren. Die Optimierung bezieht sich dabei auf die Anzahl der verwendeten Gatter. Wird die Quanten-Addition für eine feste Zahl realisiert, können für jedes einzelne Qubit die angewendeten Phasenverschiebungen vorab zusammengerechnet werden. Demnach bietet es sich an, die zusammengerechneten Phasenverschiebung in ein einzelnes Phasen-Gatter zusammen zu fassen. Nach diesem

Prinzip, benötigt man für eine Quanten-Addition maximal ein einzelnes Phasen-Gatter pro Qubit.

Abbildung 12: Ressourcensparende Quantum-Addition



Der Quantenschaltkreis in Abbildung 12 führt die identische Berechnung durch, wie der Quantenschaltkreis aus Abbildung 11. Darüber hinaus, werden die benötigten Phasenverschiebungen mit einem einzigen Phasen-Gatter pro Qubit realisiert.

Die Implementierung nutzt die ressourceneffiziente Variante der Quanten-Addition. In der Funktion `A_Gate` wird ein Gatter erzeugt, das sämtliche für die Quanten-Addition erforderlichen Phasen-Gatter umfasst. Der Code zur Funktion ist in Abbildung 13 dargestellt.

Abbildung 13: Quantum-Addition in Qiskit

```
1  def A_Gate(a_bin: list[int]) -> qiskit.circuit.gate:
2      A_Gate = qiskit.QuantumCircuit(len(a_bin))
3      theta_list = [0.0]*len(a_bin)
4      for target_bit in range(len(a_bin)):
5          exponent = 1
6          for control_bit in reversed(range(target_bit+1)):
7              if a_bin[control_bit] == 1:
8                  theta_list[target_bit]+= 2*pi/(2**(exponent))
9              exponent+=1
10     for qubit_index in range(len(a_bin)):
11         A_Gate.append(P_Gate(theta_list[qubit_index]),[qubit_index])
12     A_Gate = A_Gate.to_gate()
13     A_Gate.name = " Add(" + str (binToDez(a_bin) )+ ")"
14     return A_Gate
```

Der Funktion `A_Gate` wird der zu addierende Summand im Binärformat übergeben, wobei am Index Null das Least-Significant-Bit liegt. Abhängig von der Anzahl der Bits des Summanden wird ein Quantenschaltkreis mit der gleichen Anzahl an Qubits erzeugt. In den Zeilen 4 bis 9 werden die Phasenverschiebungen, die auf ein einzelnes Qubit

wirken sollen, berechnet und akkumuliert. Anschließend wird in den Zeilen 10 bis 11 auf jedes Qubit des Quantenschaltkreises ein individuelles Phasen-Gatter angewendet. Die Phasenverschiebung eines einzelnen Phasen-Gatters ergibt sich aus der vorherigen Akkumulation in den Zeilen 4 bis 9. Abschließend wird der Quantenschaltkreis in ein Gatter umgewandelt und mit einer passenden Bezeichnung versehen.

7.1.2 Subtraktion

Die Quanten-Addition besteht ausschließlich aus unitären Gattern und ist daher selbst auch unitär. Diese Eigenschaft vereinfacht die Implementierung der Subtraktion. Indem die Quanten-Addition invertiert wird, ergibt sich ein Quantenschaltkreis, der eine Subtraktion in der Fourier-Basis ausführt. Damit hat man praktisch einen Quantenschaltkreis der Quanten-Subtraktion.

Genau wie bei der Quanten-Addition kann es auch bei der Quanten-Subtraktion zu einem Overflow, beziehungsweise im konkreten Kontext, zu einem Underflow kommen. Bei der Quanten-Subtraktion tritt dieser Effekt ein, falls der Minuend im Zielregister kleiner als der Subtrahend ist.

Dieser Effekt kann verwendet werden, um herauszufinden, ob der Subtrahend größer ist als der Minuend [Bea03]. Angenommen man hat zwei Zahlen die maximal der Größenordnung 2^n entsprechen und ein Zielregister welches aus $n+1$ Qubits besteht. Der Subtrahend b befindet sich im Zielregister also $|\phi(b)\rangle_{n+1}$, worauf die Quanten-Subtraktion mit dem Minuend a wirkt. Dann existieren zwei mögliche Fälle:

$$b \geq a \;\rightarrow\; |\phi(b-a)\rangle_{n+1}\,;\; b < a \;\rightarrow\; \left|\phi(2^{n+1}-(a-b))\right\rangle_{n+1}$$

Das Most-Significant Bit ist ausschließlich im zweiten Fall gesetzt und kann somit, nachdem das Register in die Standardbasis transformiert wird, als eine Art Borrow-Bit entspricht.

Die Implementierung der Quanten-Subtraktion ist aufgrund der `inverse` Funktion von Qiskit unkompliziert. Wendet man die `inverse` Funktion auf ein Gatter der Quanten-Addition der `A_Gate` Funktion an, wird dieses in die Quanten-Subtraktion invertiert. Der Code davon ist in 14 abgebildet.

Abbildung 14: Quantum-Subtraktion in Qiskit

```
1 def S_Gate(subtrahend_bin: list[int]) -> qiskit.circuit.gate:
2     S_Gate = A_Gate(subtrahend_bin).inverse()
3     S_Gate.name = "  Sub(" + str (binToDez(subtrahend_bin) )+ ")"
4     return S_Gate
```

7.1.3 Modulare Addition

Die Gatter der Quanten-Addition und der Quanten-Subtraktion ermöglichen die Konstruktion einer unitären Transformation, die die Berechnung der modularen Addition

realisiert. Gemeinsam bilden sie einen Quantenschaltkreis, der das Ergebnis der Berechnung $|\Phi(a+b) \mod N\rangle$ für $a, b < N$ im Zielregister speichert.

Abbildung 15: Modulare Addition nach Beauregard [Bea03]

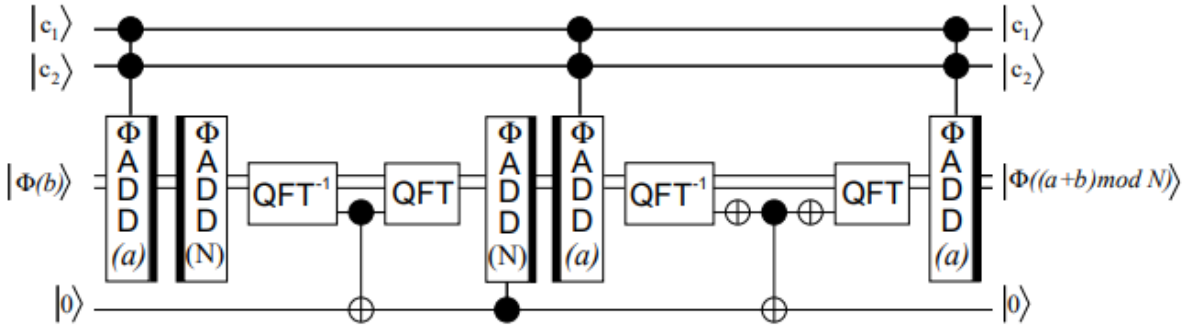


Abbildung 15 zeigt das verwendete Konzept, das für die Implementierung der modularen Addition als Bauplan diente. In der Grafik sind zwei Arten von ϕADD-Gattern zu sehen. Dabei handelt es sich um die Quanten-Addition, erkennbar mit dem fettgedruckten Strich auf der rechten Seite und der Quanten-Subtraktion mit einem fettgedruckten Strich auf der linken Seite. Sowohl die Quanten-Addition als auch die Quanten-Subtraktion addieren beziehungsweise subtrahieren eine fixe Zahl und sind somit die zuvor vorgestellten Varianten mit gewöhnlichen Phasen-Gattern. Der Quantenschaltkreis zur modularen Addition agiert als Baustein in einem größeren Gesamtbild, weshalb die zwei Kontroll-Qubits $|c_1\rangle$ und $|c_2\rangle$ eingebaut werden, die im weiteren Verlauf Anwendung finden. Das Eingaberegister stellt auch das Zielregister dar und ist initial bereits in der Fourier-Basis mit $|\phi(b)\rangle$. Das Zielregister wird um ein weiteres Qubit erweitert, welches das Most-Significant-Bit des Zielregisters dargestellt. Dieses Qubit hat einen besonderen Anwendungszweck und wird im weiteren als Borrow-Qubit bezeichnet. Insgesamt besteht das Zielregister also aus $n+1$ Qubits wenn der Modulus N einer Größenordnung von 2^n entspricht. Alle nachhaltigen Veränderungen durch den Quantenschaltkreis betreffen ausschließlich das Zielregister. Des Weiteren verwendet die Quantenschaltung noch ein Qubit, welches initial im Zustand $|0\rangle$ beginnt. Der Zustand dieses einzelnen Qubits bedingt eine Fallunterscheidung in der Berechnung des Quantenschaltkreises und wird daher im Weiteren als Bedingungs-Qubit bezeichnet.

Um die Berechnung des Quantenschaltkreises nachvollziehen zu können, wird die Auswirkung der verwendeten Gatter im einzelnen erklärt. Die erste Quanten-Addition sorgt dafür, dass auf den initialen Zielregister $|\phi(b)\rangle$ eine Addition mit a erfolgt und dadurch $|\phi(b+a)\rangle$ entspricht. Darauf folgt eine Quanten-Subtraktion die mit dem Subtrahend N auf das Zielregister mit $|\phi(b+a-N)\rangle$ wirkt. Daraus resultieren zwei mögliche Zustände:

$$1.\ (b+a) \geq N \ \rightarrow\ |\phi((b+a)-N)\rangle_{n+1}\ ;\ 2.\ (b+a) < N \ \rightarrow\ \left|\phi(2^{n+1}-(N-(a+b)))\right\rangle_{n+1}$$

Im erste Fall entspricht das Ergebnis der korrekten Berechnung von $a+b \mod N$. In

diesem Fall ist der Registerinhalt mit $a + b \mod N$ kleiner als N, weshalb das Borrow-Bit nicht gesetzt ist. Im Gegensatz dazu ist das Ergebnis im zweiten Fall fehlerhaft. Da bei $(b+a) < N$ bereits der Rest der modularen Restklasse im Register steht, wird durch die Quanten-Subtraktion ein N zu viel vom Registerinhalt abgezogen. Dadurch entsteht ein underflow im Zielregister wodurch das Borrow-Bit gesetzt wird.

Als nächstes wirkt die inverse Quanten-Fourier-Transformation auf das Zielregister und führt eine Transformation in die Standartbasis durch. Aufgrund des Basiswechsels in die Standartbasis beschreiben die Zustände der Qubits des Zielregisters nun das Ergebnis in der Binärdarstellung. In der Binärdarstellung befindet sich im ersten Fall das Borrow-Bit im Zustand $|0\rangle$ und im zweiten Fall im Zustand $|1\rangle$. Anhand dieser Unterscheidung kann man eine bedingte Operation mittels eines kontrollierten X-Gatter realisieren. Dafür kontrolliert das Borrow-Bit ein X-Gatter welches auf das Bedingungs-Qubit wirkt. Dadurch wird das Bedingungs-Qubit ausschließlich im zweiten Fall in den Zustand $|1\rangle$ versetzt. Danach wird das Zielregister wieder in die Fourier-Basis transformiert, indem die Quanten-Fourier-Transformation angewendet wird. Anschließend kontrolliert das Bedingungs-Qubit eine Quanten-Addition mit dem Summanden N auf das Zielregister. Die kontrollierte Quanten-Addition wird also nur im zweiten Fall angewendet und korrigiert das vorher ungültige Ergebnis, zu dem korrekte:

$$\left|\phi(2^{n+1} - (N - (a+b)))\right\rangle_{n+1} \xrightarrow{+N} \left|\phi(2^{n+1} + (a+b))\right\rangle_{n+1} \xrightarrow{\text{overflow } 2^{n+1}} \left|\phi(a+b)\right\rangle_{n+1}$$

Nun befinden sich in beiden Fällen das korrekt berechnete Ergebnis im Zielregister mit $|\Phi(a+b) \mod N\rangle$. Somit ist die Berechnung der modularen Addition abgeschlossen.

Je nach dem, welcher der beiden Fälle eingetreten ist, befindet sich das Bedingungs-Qubit in einem anderen Zustand. Um zu verhindern, dass sich das Bedingungs-Qubit in einem ungewissen Zustand befindet und dadurch zum „Trash"-Qubit wird, wird der initiale Zustand $|0\rangle$ des Bedingungs-Qubit durch die restlichen Gatter des Quantenschaltkreises wiederhergestellt. Ein eindeutiger Zustand ermöglicht, dass das Bedingungs-Qubit bei weiteren Berechnungen wiederverwendet werden kann.

Um das Bedingungs-Qubit zurückzusetzen, wird zuerst eine Quanten-Subtraktion mit a auf das Zielregister angewendet. Um die Auswirkungen dieser Quanten-Subtraktion hervorzuheben, wird der Registerinhalt für beide Fälle untersucht: Im ersten Fall war $(b + a) \geq N$ weswegen die Subtraktion von N zum Ergebnis der modularen Addition führte. Da $a, b < N$ gilt, ist das Ergebnis der modularen Addition kleiner als a. Die Quanten-Subtraktion mit a führt also zu einem Underflow, wodurch das Borrow-Bit gesetzt wird. Im zweiten Fall, war $(b+a) < N$. Deswegen wurde die Subtraktion von N nicht durchgeführt, beziehungsweise wieder rückgängig gemacht, da $(b + a)$ bereits das Ergebnis der modularen Addition ist. Für den zweiten Fall gilt als Ergebnis also $(b+a)$ und dies ist größer als a, darum kommt es zu keinem Underflow und das Borrow-Bit ist deswegen nicht gesetzt. Der Zustand des Borrow-Bits ist nun also im Vergleich zu den Fällen des vorherigen Abschnitts der Quantenschaltung invertiert.

Um das Borrow-Bit auslesen zu können, wird analog zum vorherigen Abschnitt des Quantenschaltkreises eine inverse Quanten-Fourier-Transformation durchgeführt. An-

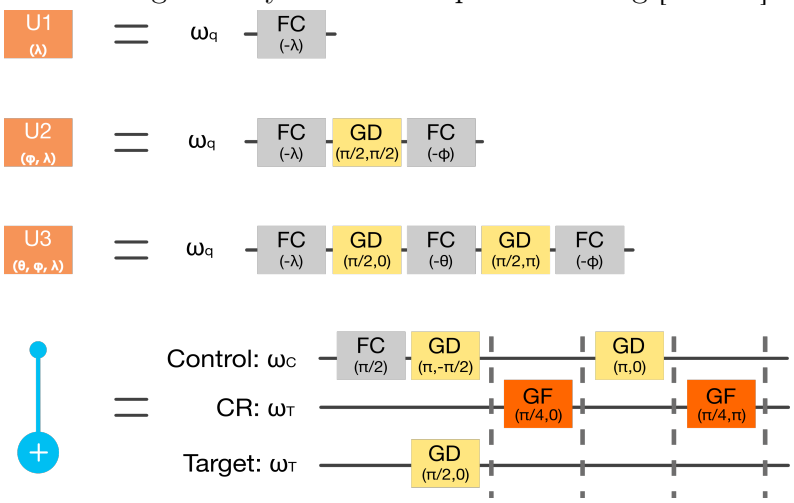
schließend wird ein X-Gatter auf das Borrow-Bit angewendet. Dadurch befindet sich das Borrow-Bit nun in denselben Zuständen wie in den beiden Fällen des ersten Abschnitts der Quantenschaltung. Im nächsten Schritt kontrolliert das Borrow-Bit ein X-Gatter, das auf das Bedingungs-Qubit wirkt. Dadurch wird es wieder in den initialen Zustand $|0\rangle$ versetzt.

Um den Zustand $|\Phi(a+b) \mod N\rangle$ des Zielregister wiederherzustellen, werden die Gatter bis zur Quanten-Subtraktion mit a inverse angewendet. Zunächst wird ein selbstinverses X-Gatter auf das Borrow-Bit angewendet. Danach folgt die Quanten-Fourier-Transformation, die die Inverse der inversen Quanten-Fourier-Transformation darstellt. Abschließend folgt die Quanten-Addition mit a, um die zuvor erfolgte Quanten-Subtraktion von a auf das Zielregister zu revertieren.

Das Zielregisters beinhaltet nun also den gewünschten Zustand $|\Phi(a+b) \mod N\rangle$ für $a, b < N$. Des weiteren befindet sich das Bedingungs-Qubit wieder im initialen Zustand und kann für weitere Rechnungen wiederverwendet werden.

Wenn man den Quantenschaltkreis in Abbildung 15 betrachtet, fällt auf, dass nicht alle Quanten-Additionen und Quanten-Subtraktionen kontrolliert durchgeführt werden. Die modulare Addition soll nur dann ausgeführt werden, wenn die Kontroll-Qubits $|c_1\rangle$ und $|c_2\rangle$ gesetzt sind. Dies ist zurückzuführen auf die Bedingung $b < N$, die dafür sorgt, dass der Rest des Quantenschaltkreises lediglich die Identitätstransformation durchführt, wenn die Kontroll-Qubits nicht aktiviert sind [Bea03]. Der Grund, weshalb nicht alle Quanten-Additionen, Quanten-Subtraktion und gegebenenfalls sogar die (inversen) Quanten-Fourier-Transformationen kontrolliert sind, liegt in der erhöhten Komplexität, die dadurch im Quantenschaltkreis entstehen würde [Bea03]. Wenn ein Kontroll-Qubit nicht aktiviert ist, wird das kontrollierte Gatter zwar ausgeführt, jedoch ohne praktische Wirkung. Abbildung 16 zeigt die physikalische Implementierung dreier Single-Qubit-Gatter im Vergleich zu einem kontrollierten X-Gatter. Wie aus der Abbildung erkennbar, benötigt das kontrollierte Gatter mehr Hardware-Elemente als die drei Single-Qubit-Gatter. Somit ist es nicht möglich die Komplexität des Quantenschaltkreis zu verringern, indem zusätzliche Gatter kontrolliert angewendet werden.

Abbildung 16: Physikalische Implementierung [IBM23]



Ein weiterer Aspekt, der die Komplexität der Quantenschaltung für die modulare Addition erhöht, ist der Abschnitt, der das Bedingungs-Qubit zurücksetzt. Wäre es möglich, diesen Teil auszulassen, könnten zwei der insgesamt fünf Quanten-Additionen bzw. Quanten-Subtraktionen sowie die Hälfte der (inversen) Quanten-Fourier-Transformationen eingespart werden.

Es gibt die Möglichkeit, bei einem Qubit einen Reset durchzuführen, wodurch dieses den Zustand $|0\rangle$ annimmt. In Qiskit kann dies mit der Funktion `Reset` [Tea23] realisiert werden. Die Verwendung ist jedoch nicht zielführend, da diese durch keine unitäre Abbildungsmatrix beschrieben werden kann. Somit ist die Reset Transformation nicht unitär und würde deswegen dazu führen, dass alle Quantenalgorithmen, die diese Funktion nutzen, ebenfalls nicht mehr unitär wären. Da die Quanten-Phase-Estimation den Eigenwert einer unitären Transformation extrahiert, ist diese Funktion für die Anwendung im Shor-Algorithmus ungeeignet.

Abbildung 17: Qiskit modulare Addition

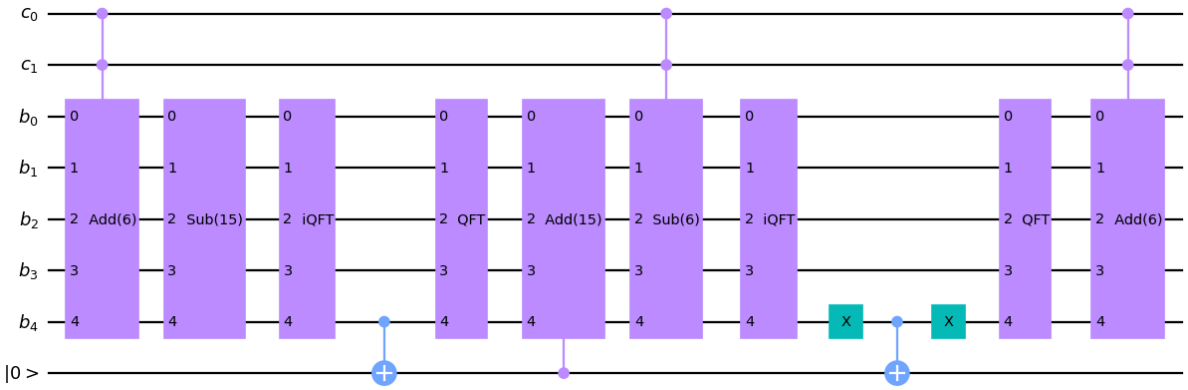


Abbildung 17 zeigt die Implementierung eines Gatters für die modulare Addition mit $a = 6$ und $N = 15$ in Qiskit. Der zugehörige Code, der ein Gatter wie in Abbildung 17 generiert, ist in der Funktion `Modular_Adder_Gate` 18 enthalten.

Die Funktion `Modular_Adder_Gate` erwartet den Summand a und die Zahl N als Parameter, jeweils in der Form einer Liste, welche die Zahl in Binärdarstellung beschreibt. Dabei repräsentiert der erste Index beider Listen das Least-Significant-Bit. Wenn N der Größenordnung 2^n entspricht, müssen die Listen jeweils von der Länge $n+1$ sein wobei das Most-Significant-Bit immer der 0 entspricht.

Die Funktion definiert drei Wertebereiche, die die Positionen der Qubits beschreiben. *c_ qbits* definiert die Position der beiden Kontroll-Qubits, *b_ qbits* die Qubits, welche den Summanden b in der Fourier-Basis enthalten und *cond_ qbit* repräsentiert die Position des Bedingungs-Qubit. Dadurch das *b_ qbits* genau so groß definiert wird, wie die Liste des Summand a lang ist, wird sichergestellt, dass das *b_ qbits* Register ein extra Qubit für das Borrow-Bit enthält. In Zeile 5 der Funktion, wird ein Quantenschaltkreis definiert, der aus so viele Qubits besteht, wie es insgesamt an Positionen in *c_ qbits*, *b_ qbits* und *cond_ qbit* gibt.

In den Zeilen 6 bis 18 werden alle benötigten Gatter, in der selben Reihenfolge wie in Abbildung 15 angewendet. Zum hinzufügen eines eigens programmierten Gatters, also für die Quanten-Addition, Quanten-Subtraktion und die Quanten-Fourier-Transformation, wird die Qiskit Methode `append` genutzt.

Abschließend wird der Quantenschaltkreis in Zeile 19 in ein Gatter umgewandelt.

Abbildung 18: Modulare Addition in Qiskit

```
1  def Modular_Adder_Gate(a_bin: list[int],N_bin: list[int]) -> qiskit.circuit.gate:
2    c_qbits = [0,1]
3    b_qbits = list(range(2, len(a_bin)+2))
4    cond_qbit = len(a_bin)+2
5    m_a_g = qiskit.QuantumCircuit(2 + len(a_bin) + 1)
6    m_a_g.append(A_Gate(a_bin).control(2), c_qbits + b_qbits)
7    m_a_g.append(S_Gate(N_bin),b_qbits)
8    m_a_g.append(QFT_Gate(len(a_bin),inverse = True, MSB_first = False), b_qbits)
9    m_a_g.cnot(b_qbits[-1],cond_qbit)
10   m_a_g.append(QFT_Gate(len(a_bin),inverse = False, MSB_first = False), b_qbits)
11   m_a_g.append(A_Gate(N_bin).control(1), [cond_qbit] + b_qbits)
12   m_a_g.append(S_Gate(a_bin).control(2), c_qbits + b_qbits)
13   m_a_g.append(QFT_Gate(len(a_bin),inverse = True, MSB_first = False), b_qbits)
14   m_a_g.x(b_qbits[-1])
15   m_a_g.cnot(b_qbits[-1],cond_qbit)
16   m_a_g.x(b_qbits[-1])
17   m_a_g.append(QFT_Gate(len(a_bin),inverse = False, MSB_first = False), b_qbits)
18   m_a_g.append(A_Gate(a_bin).control(2), c_qbits + b_qbits)
19   m_a_g = m_a_g.to_gate()
20   m_a_g.name = "Add " + str(binToDez(a_bin)) + " Mod " + str(binToDez(N_bin))
21   return m_a_g
```

7.1.4 Kontrollierte Multiplikation

Als nächstes wird ein Gatter konstruiert, welches auf vier Quantenregister $|c\rangle_1 |x\rangle_n |b\rangle_{n+1} |0\rangle_1$ eine Transformation nach $|c\rangle_1 |x\rangle_n |b + (ax) \mod N\rangle_{n+1} |0\rangle_1$ durchführt. Das Gatter umfasst ein Kontroll-Qubit $|c\rangle_1$, ein Quantenregister $|x\rangle_n$ mit dem Faktor x, das Zielregister $|b\rangle_{n+1}$ mit initialem Wert b und ein Hilfsqubit $|0\rangle_1$ welches als Bedingungs-Qubit dient.

Die Spezifikation dieses Gatters besteht darin, auf den Registerinhalt des Zielregisters $|b\rangle_{n+1}$ mit initialen Wert b , das Produkt der Faktoren x und a modulare aufzuaddieren.

Dabei handelt es sich bei dem Faktor a um eine vorab bekannte Zahl die während der Berechnung nicht variiert. Anders sieht dies bei x aus, da es sich bei dieser Zahl um den Registerinhalt von $|x\rangle_n$ handelt. Das Gatter berechnet also das Produkt einer klassischen Zahl a und dem Inhalt eines Quantenregisters x. Das Register $|x\rangle_n$ umfasst mehrere Qubits die x im Binärsystem beschreiben, also zum Beispiel in Binärschreibweise für drei Qubits $|x\rangle_3 = 2^0x_0+2^1x_1+2^2x_2$. Das Produkt kann dementsprechend umgeformt werden:

$$(x \cdot a) = (2^0x_0 \cdot a + 2^1x_1 \cdot a + 2^2x_2 \cdot a)$$

Beziehungsweise bezogen auf die Transformation des $|b\rangle_{n+1}$ Registers:

$$|b + (x \cdot a) \mod N\rangle_4 = \big|b + (2^0x_0 \cdot a + 2^1x_1 \cdot a + 2^2x_2 \cdot a) \mod N\big\rangle_4$$

Bedeutet, dass die Transformation realisiert werden kann, indem die einzelnen Produkte $2^ix_i \cdot a$ einzeln auf $|b\rangle_{n+1}$ addiert werden. Um große Zwischenergebnisse zu vermeiden, die gegebenenfalls für einen Overflow im $|b\rangle_{n+1}$ Register sorgen, kann man die Berechnung weiter umformen [Bea03]:

$$= \big|(((b + 2^0x_0 \cdot a) \mod N + 2^1x_1 \cdot a) \mod N + 2^2x_2 \cdot a) \mod N\big\rangle_4$$

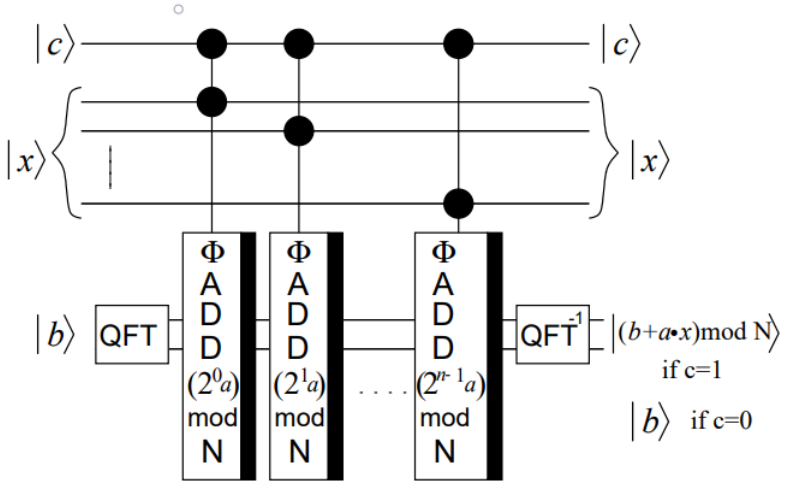
Dies wird als Quantenschaltkreis realisiert, indem eine Gatter zur modularen Addition verwendet wird, um $(b+2^ix_i \cdot a) \mod N$ zu berechnet. Dafür wird das modulare Addition Gatter mit dem Summanden beziehungsweise Parameter $2^i \cdot a$ versehen und für die Abhängigkeit von x_i durch das Qubit $|x_i\rangle_1$ kontrolliert. Für die gesamte Summe muss jedes Qubit des $|x\rangle_n$ Registers ein Gatter zur modularen Addition kontrollieren, welche auf das Register $|b\rangle_{n+1}$ wirken. Jedes Gatter zur modularen Addition bekommt die zugehörigen Potenz von 2^i zugewiesen, die auch der Wertigkeit des zugehörigen Kontroll-Qubits $|x_i\rangle_1$ entspricht. Also wirken alle Gatter zur modularen Addition nacheinander auf das $|b\rangle_{n+1}$ Register und Transformieren dies so sukzessiv nach $|b + (x \cdot a) \mod N\rangle_{n+1}$.

Die verwendeten Gatter zur modularen Addition führen alle Berechnungen in der Fourier-Basis durch. Deswegen muss sich das $|b\rangle_{n+1}$ Register auch in der Fourier-Basis befinden. Das Gatter für die kontrollierte Multiplikation wirkt dazu als erste Operation eine Quanten-Fourier-Transformation auf das Zielregister $|b\rangle_{n+1}$ und als letzte Operation die inverse Quanten-Fourier-Transformation.

Zusätzlich verfügt das Gatter zur kontrollierten Multiplikation noch über das Kontroll-Qubit $|c\rangle_1$. Dieses Kontroll-Qubit kontrolliert alle der verwendeten modularen Addition Gatter. Das Kontroll-Qubit $|c\rangle_1$ findet beim nächsten Gatter Gebrauch.

Alles zusammen führt zu einem Quantenschaltkreis wie in Abbildung 19.

Abbildung 19: Kontrollierte Multiplikation nach Beauregard [Bea03]



Der Code der einen Quantenschaltkreis als Gatter nach dem Konzept in Abbildung 19 generiert, wird in der Funktion `cmult_gate` 20 definiert. Die Funktion erwartet sowohl a als auch N als Parameter, jeweils in der Form einer Liste, welche die Zahl in Binärdarstellung beschreibt. Dabei repräsentiert der erste Index beider Listen das Least-Significant-Bit. Damit die unterliegenden modularen Addition Gatter für die korrekte Anzahl an Qubits gebildet werden, müssen die Listen jeweils von der Länge n + 1 sein, wenn N der Größenordnung 2^n entspricht. Das Most-Significant-Bit ist dementsprechend für beide Listen 0. Des Weiteren erwartet die Funktion den Parameter *x_ bits_ amount*, welcher angibt, wie viele Qubits das Register $|x\rangle_n$ umfasst und somit also n entspricht.

Um die Position der Qubits zu beschreiben, werden in der Funktion vier Wertebereiche definiert. *c_ qbit* liegt an der ersten Stelle des kontrollierten Multiplikation Gatters und ist für das Kontroll-Qubit $|c\rangle_1$ vorhergesehen. Die darauffolgenden Qubits umfassen das $|x\rangle_n$ Register und bestehen aus insgesamt n Qubits, deren Position in *c_ qbit* definiert ist. Die Position des $|b\rangle_{n+1}$ Registers wird in *b_ qbits* beschrieben. Aufgrund der unterliegenden Gatter umfasst das $|b\rangle_{n+1}$ Register ein extra Qubit, welches als Borrow-Bit dient. Die Position des letzten Qubit $|0\rangle_1$ wird mit *cond_ qbit* definiert. Dieses Qubit ist das Bedingungs-Qubit für die unterliegenden modularen Addition Gatter.

Im Code 20 wird in Zeile 9 die Quanten-Fourier-Transformation auf das $|b\rangle_{n+1}$ Register anwendet. In Zeile 10 bis 13 folgen die Gatter zur modularen Addition. Da $(b + a \cdot x) \mod N$ äquivalent ist zu $(b + (a \cdot x) \mod N) \mod N$ [Koe21], wird einem Gatter der modularen Addition nicht der Parameter $2^i \cdot a$ übergeben, sondern $(2^i \cdot a) \mod N$. Andernfalls wäre es nötig, zusätzliche Qubits zu verwendet, damit $2^i \cdot a$ gespeichert werden kann. Wie man in Zeile 13 sieht, teilen sich alle modularen Additions Gatter das selbe

Bedingungs-Qubit $|0\rangle_1$, welches sich auf der Position von *cond_qbit* befindet. Genau aus diesem Grund, wird bei dem Gatter zur modularen Addition des Bedingungs-Qubit zurückgesetzt. Abschließend wird noch die inverse Quanten-Fourier-Transformation auf das $|b\rangle_{n+1}$ Register anwendet und von einem Quantenschaltkreis in ein Quantengatter umgeformt.

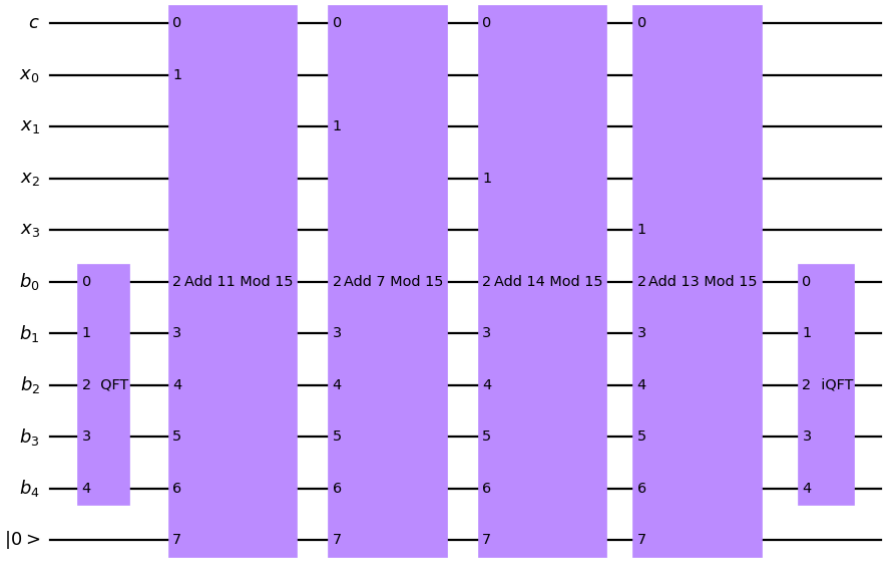
Abbildung 20: Kontrollierte Multiplikation in Qiskit

```
def cmult_gate(x_bits_amount: int,a_bin: list[int],N_bin: list[int]) -> qiskit.circuit.gate:
  a_dez = binToDez(a_bin)
  N_dez = binToDez(N_bin)
  c_qbit = [0]
  x_qbits =  list(range(1, x_bits_amount + 1))
  b_qbits = list(range(1+x_bits_amount, 1 + x_bits_amount + len(a_bin)))
  cond_qbit = [1 + x_bits_amount + len(a_bin)]
  cmult_gate = qiskit.QuantumCircuit(1 + x_bits_amount + len(a_bin) + 1)
  cmult_gate.append(QFT_Gate(len(b_qbits),inverse = False, MSB_first = False), b_qbits)
  for i in range(0, len(x_qbits)):
      a_i = ((2**(i)) *  a_dez) % N_dez
      a_i_bin = dezToBin(a_i, len(a_bin))
      cmult_gate.append(modular_adder_gate(a_i_bin, N_bin),c_qbit+[x_qbits[i]]+b_qbits+cond_qbit)
  cmult_gate.append(QFT_Gate(len(b_qbits),inverse = True, MSB_first = False), b_qbits)
  cmult_gate = cmult_gate.to_gate()
  cmult_gate.name = "cmult " + str(a_dez) + " Mod " + str(N_dez)
  return cmult_gate
```

In Qiskit wird der durch den Code gebildete Quantenschaltkreis wie in Abbildung 21 in Qiskit dargestellt. Die Visualisierung der Kontroll-Abzweigungen ist anders als in den vorherigen Abbildungen dargestellt. Dies liegt daran, dass die Kontroll-Qubits bereits in den unterliegenden Gattern, also der modularen Addition, definiert wurden. Es ist möglich, die Kontroll-Qubits bei der Definition der modularen Adder Gatter auszulassen und stattdessen erst bei der Anwendung im kontrollierten Multiplikation Gatter anzuwenden, indem die Qiskit Funktion *control* verwendet wird. Dadurch wird der Kontrolleffekt auf das gesamte Gatter zur modularen Addition angewendet, wodurch man einen ineffizientere Quantenschaltkreis erhält, wie bereits im Abschnitt 7.1.3 zur modularen Addition erwähnt.

Es ist zu beachten, dass die Gatter zur modularen Addition in Abbildung 21 zwar über alle Qubits des $|x\rangle_n$ Registers verlaufen, jedoch ausschließlich durch das Qubit kontrolliert werden, bei dem das modularen Additions Gatter den Index 1 verzeichnet. Die Abbildung repräsentiert den Quantenschaltkreis für $a = 11;\ N = 15$.

Abbildung 21: Schaltkreis der kontrollierten Multiplikation in Qiskit



Insgesamt wirkt das Gatter auf $2n+3$ Qubits. Diese setzen sich aus dem Kontroll-Qubit $|c\rangle_1$, den n Qubits von $|x\rangle_n$, den $n+1$ Qubits von $|b\rangle_{n+1}$ und dem Bedingungs-Qubit $|0\rangle_1$zusammen.

Damit die Inverse des kontrollierten Multiplikation Gatters eine Eindeutige Benennung hat, wird dafür eine zusätzliche Funktion `inv_cmult_gate` definiert.

Abbildung 22: Inverse kontrollierte Multiplikation in Qiskit

```
def inv_cmult_gate(x_bits_amount: int,a_bin: list[int],N_bin: list[int]) -> qiskit.circuit.gate:
  inv_cmult_gate = cmult_gate(x_bits_amount,a_bin,N_bin).inverse()
  inv_cmult_gate.name = "inv cmult " + str(binToDez(a_bin)) + " Mod " + str(binToDez(N_bin))
  return inv_cmult_gate
```

7.1.5 Vollstände Transformation

Das letzte zu konstruierende Gatter realisiert die benötigte Transformation $U|y\rangle = |ay \mod N\rangle$ und wird im weiteren als U_a-Gatter bezeichnet. Mit dem U_a-Gatter wird die Quanten-Phase-Estimation, wie in Abbildung 9, für den Shor-Algorithmus gebildet. Dafür nimmt das U_a-Gatter die Quantenregister $|c\rangle_1 |x\rangle_n |0\rangle_{n+1} |0\rangle_1$ entgegen und führt eine Transformation nach $|c\rangle_1 |(a \cdot x) \mod N\rangle_n |0\rangle_{n+1} |0\rangle_1$ durch.

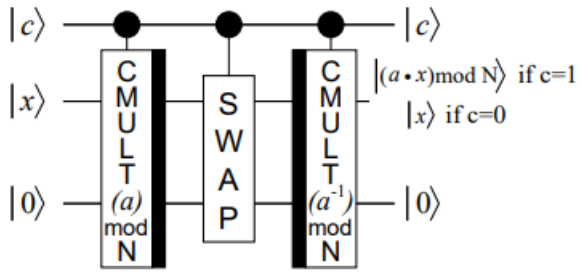
Um ein U_a-Gatter zu realisieren, benötigt man im Grunde lediglich zwei Gatter zur kontrollierten Multiplikation und kontrollierte Swap-Gatter.

Im folgenden wird die Auswirkung der einzelnen Bestandteile des U_a-Gatter erklärt. Dabei werden von den Quantenregistern $|c\rangle_1 |x\rangle_n |0\rangle_{n+1} |0\rangle_1$ nur die beiden Quantenregis-

ter $|x\rangle_n |0\rangle_{n+1}$ beachtet. Sowohl $|c\rangle_1$ also auch $|0\rangle_1$ verzeichnen während der Transformation keine Veränderung, welche für das Verständnis des U_a-Gatters relevant ist. Der Ablauf des U_a-Gatter ist der folgende [Bea03]: Initial befindet sich die beiden Quantenregistern in dem Zustand $|x\rangle_n |0\rangle_{n+1}$, anschließend wird ein Gatter zur kontrollierten Multiplikation mit a ausgeführt, welches den Zustand zu $|x\rangle_n |(ax) \mod N\rangle_{n+1}$ transformiert. Darauf folgen kontrollierte Swap-Gatter die beide Zustände zu $|(ax) \mod N\rangle_n |x\rangle_{n+1}$ vertauschen. Dabei ist zu beachten, dass das zweite Quantenregister aus $n+1$ Gattern besteht, also ein Qubit mehr besitzt, als das erste Quantenregister. Da das Most-Significant-Bit des zweiten Quantenregisters, zu dem Zeitpunkt, garantiert dem Zustand $|0\rangle$ entspricht, kann dieses Vernachlässigt werden und wird keiner Swap-Operation unterzogen. Nach dem Swap wird ein inverses Gatter zur kontrollierten Multiplikation angewendet. Des Weiteren wird dem inverses Gatter zur kontrollierten Multiplikation statt dem Element a, das inverse von a sein multiplikatives Inverses in den Einheiten des Restklassenrings N übergeben, also $a^{-1} \mod N$. Als Folge davon, verändert sich der Zustand der Quantenregister von $|(ax) \mod N\rangle_n |x\rangle_{n+1}$ zu $|(ax) \mod N\rangle_n |(x - a^{-1}ax) \mod N\rangle_{n+1}$. Der Ausdruck $(a^{-1}a) \mod N$ entspricht dem neutralen Element 1, weswegen der Zustand der Quantenregister zu $|(ax) \mod N\rangle_n |0\rangle_{n+1}$ vereinfacht werden kann.

Beachtet man also den ersten und letzten Zustand der beiden Quantenregister, so bewirkt das U-Gatter also eine Transformation von $|x\rangle_n |0\rangle_{n+1}$ zu $|(ax) \mod N\rangle_n |0\rangle_{n+1}$. Das Konzept des Aufbaus des Quantenschaltkreises ist in Abbildung 23 dargestellt.

Abbildung 23: U_a-Gatter [Bea03]



Die Funktion `U_gate` 24 enthält den Code um ein U_a-Gatter zu generieren. Dazu benötigt die Funktion die selben Parameter wie die Funktion `cmult_gate`, also *a_ bin* und *N_ bin*, welche jeweils a und N binär im Listenformat beschreibt und *x_ bits_ amount*, die im Grunde n repräsentiert.

Im Code 24 werden insgesamt fünf Wertebereiche definiert. Das Kontroll-Qubit $|c\rangle_1$ befindet sich auf der Position *c_ qbit*. Das $|x\rangle_n$ Quantenregister wird durch *x_ qbits* beschrieben, gefolgt von *b_ qbits* für die Qubits von $|0\rangle_{n+1}$. Das Bedingungs-Qubit wird auf der letzten Position des Gatters platziert, die in *cond_ qbit* festgelegt ist. Des Weiteren wird noch *full_ range* verwendet, in welcher alle der vier vorherigen Quantenregister umfasst sind.

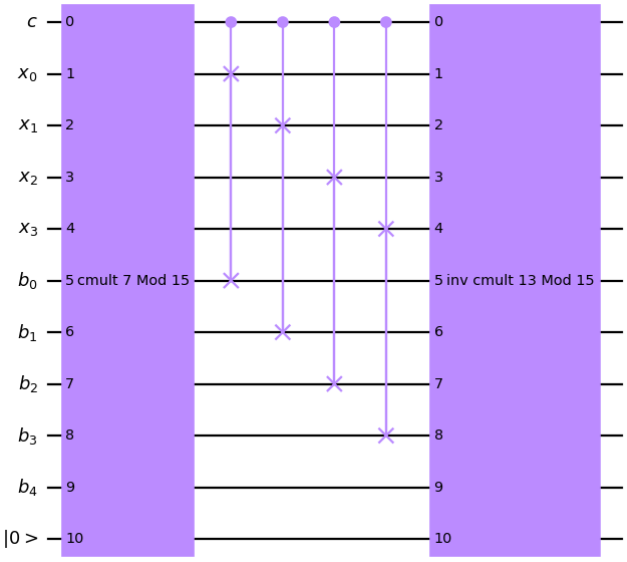
Wie im Code in Zeile 7 zu sehen ist, wird zuerst ein Gatter zur kontrollierten Multiplikation eingesetzt. Dazu bekommt die Funktion `cmult_gate` die Parameter *a_ bin*, *N_ bin* und *x_ bits_ amount* übergeben. Das Gatter zur kontrollierten Multiplikation wird über den Bereich *full_ range* angewendet, somit umfasst das U_a-Gatter auch $2n + 3$ Qubits. In Zeile 9 und 10 werden die kontrollierten Swap-Gatter eingesetzt. Wie man in Zeile 10 erkennt, bezieht sich die Wirkung eines Swap-Gatters auf den selben index von *x_ qbits* und *b_ qbits*. Somit werden die Qubits beider Quantenregister vertauscht, die die identische Wertigkeit besitzen. In Zeile 11 wird $a^{-1} \mod N$ mit dem erweiterte euklidische Algorithmus Berechnet. Anschließend wird das Berechnete $a^{-1} \mod N$ als Parameter an ein inverses kontrolliertes Multiplikation Gatter übergeben. Abschließend wird der Quantenschaltkreis zu einem Gatter umgeformt und passend benannt.

Abbildung 24: U-Gatter in Qiskit

```
1  def U_gate(x_bits_amount: int,a_bin: list[int],N_bin: list[int]) -> qiskit.circuit.gate:
2    c_qbit = [0]
3    x_qbits =  list(range(1, x_bits_amount + 1))
4    b_qbits = list(range(1+x_bits_amount, 1 + x_bits_amount + len(a_bin)))
5    cond_qbit = [1 + x_bits_amount + len(a_bin)]
6    full_range = c_qbit + x_qbits + b_qbits + cond_qbit
7    U_a_gate = qiskit.QuantumCircuit(len(full_range))
8    U_a_gate.append(cmult_gate(x_bits_amount,a_bin ,N_bin), full_range)
9    for i in (range(x_bits_amount)):
10       U_a_gate.cswap(c_qbit,x_qbits[i],b_qbits[i])
11   a_inv_bin = dezToBin(gcd(binToDez(a_bin),binToDez(N_bin)),len(a_bin))
12   U_a_gate.append(inv_cmult_gate(x_bits_amount,a_inv_bin ,N_bin), full_range)
13   U_a_gate = U_a_gate.to_gate()
14   U_a_gate.name = "  U_" + str(binToDez(a_bin))
15   return U_a_gate
```

Der Quantenschaltkreis der durch den Code 24 gebildet wird, wird in Qiskit wie in Abbildung dargestellt. Die Abbildung 25 zeigt ein Beispiel für $a = 7$ und $N = 15$.

Abbildung 25: U_a-Gatter in Qiskit



7.1.6 QPE für Shor

Das U_a-Gatter führt die benötigte unitäre Transformation von $U\,|y\rangle = |ay \mod N\rangle$ aus. Der letzte Schritt, um den Quantenschaltkreis des Quantenalgorithmus zu bilden, der in der Lage, ist die Periode zu bestimmen, besteht darin, die Quanten-Phase-Estimation wie in Abbildung 9 mit den U_a-Gatter nachzubauen.

In Abbildung 9 werden unter anderem ${U_a}^{2^x}$-Gatter eingesetzt. Diese können realisiert werden, indem ein U_a-Gatter insgesamt 2^x mal hintereinander geschaltet wird oder indem anstelle von a den Parameter a^{2^x} an ein einzelnes U_a-Gatter übergibt. Die zweite Variante ist möglich wegen [Bea03]:

$$(a^n x) \mod N = \underbrace{(a...(((a(ax) \mod N) \mod N)...) \mod N)}_{n \text{ mal}}$$

Bei der Wahl zwischen den beiden Varianten ist das entscheidende Kriterium bei der Ressourceneffizienz. Die zweite Variante schneidet dabei deutlich besser ab, da sie lediglich $2n$ U_a-Gatter benötigt, im Gegensatz zur ersten Variante, die $2^n - 1$ U_a-Gatter erfordert.

Ein weiterer wichtiger Aspekt für den Aufbau der Quantenschaltung betrifft die Anzahl der benötigten Kontroll-Qubits. Beauregard erklärt in seinem Paper [Bea03] nicht, warum er eine Konstruktion mit insgesamt $2n$ Kontroll-Qubits beziehungsweise ${U_a}^{2^x}$-Gatter wählt. Es gibt Quellen, die mehr als $2n$ ${U_a}^{2^x}$-Gatter verwendeten [NC10]. Nach dem Paper von Peter Shor [Sho97] sind jedoch $2n$ ${U_a}^{2^x}$-Gatter ausreichen, wie auch im Abschnitt 6.4 erläutert wird. Deswegen verwendet die hier implementierte Variante $2n$ Kontroll-Qubits.

Der Code für einen Quantenschaltkreis wie in Abbildung 9 ist in der Funktion `Shor` 26 enthalten. Die Funktion erwartet die beiden Zahlen a und N als Parameter. Des Weiteren kann der Funktion über den Parameter *number_shots* mitgeteilt werden, mit wie vielen Durchläufen der Quantenalgorithmus ausgeführt werden soll. Jeder Durchlauf gewährt ein weiteres Messergebnis. Der Parameter *backend* ist dazu da, um Festzulegen auf welchem System der Quantenalgorithmus ausgeführt werden soll. Standardmäßig wird der Simulator von Qiskit genommen, falls Zugang zu einem Quantensystem von IBM besteht, würde man hier die Bezeichnung des entsprechenden Quantum-Backend verwenden. Die Funktion `Shor` definiert die beiden Wertebereiche *c_qbits* und *ev_qbits* um die Positionen der beiden Quantenregister zu beschreiben. Der Wertebereich *c_qbits* enthält die insgesamt $2n$ Positionen der Kontroll-Qubits. Der Eigenvektor wird in den Qubits gespeichert, deren Position in *ev_qbits* definiert ist. Der Quantenschaltkreis wird in der 5 Zeile definiert und umfasst $4n + 2$ Qubits und $2n$ klassische Bits, welche zum Messen verwendet werden. In Zeile 6 und 7 wird auf jedes Kontroll-Qubit mit ein Hadamard-Gatter angewendet. In Zeile 8 wird der Eigenvektor gebildet. Da dieser dem Zustand $|1\rangle$ entspricht, genügt es, wenn das Least-Significant-Qubit mit einem X-Gatter versehen wird. Die Zeile 9 und 10 wendet das Zugehörige ${U_a}^{2^x}$-Gatter mit dem passenden x auf das Register mit dem Eigenvektor an. Die Funktion `mod_exp` berechnet $(a^{2^x}) \mod N$. In Zeile 11 wird die inverse Quanten-Fourier-Transformation auf die Kontroll-Qubits angewendet. Da die Funktion `Shor` Messungen durchführen soll, wird in Zeile 12 definiert, welche Qubits gemessen werden. Anschließend werden in Zeile 13-14 *number_shots* viele Messungen durchgeführt und das Ergebnis als Rückgabewert der Funktion zurückgeben.

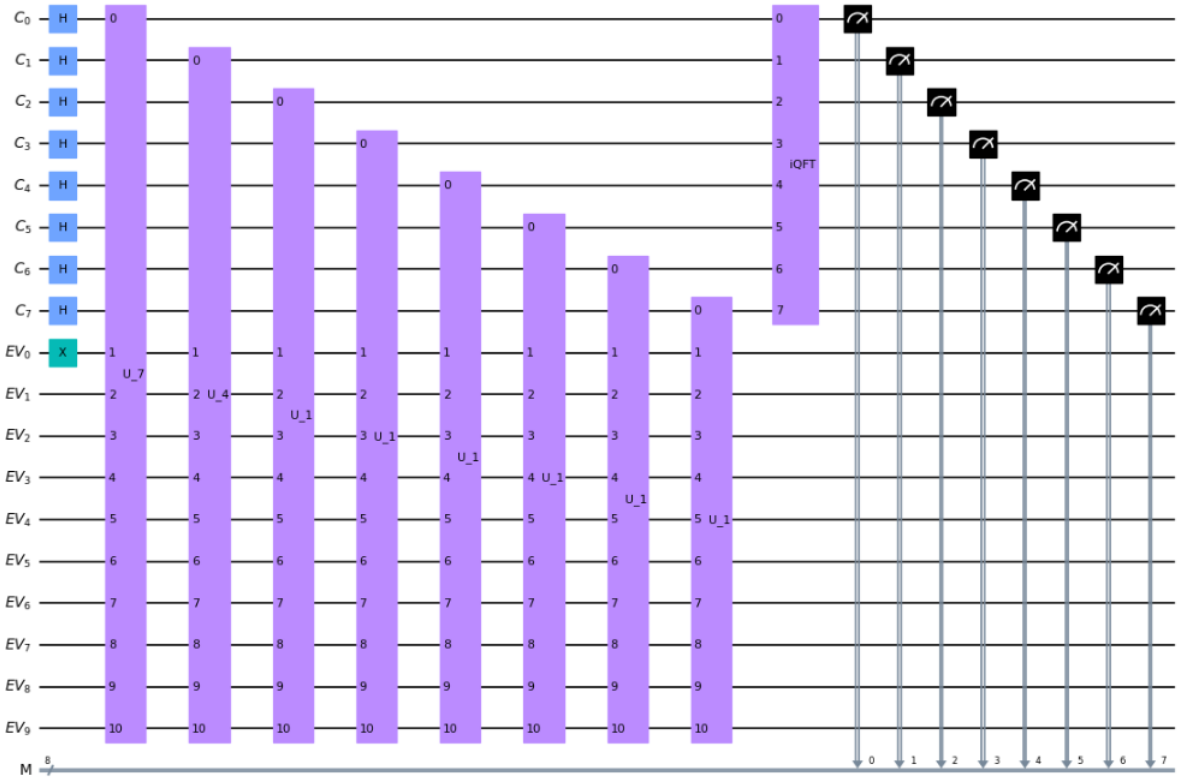
Wie bei der Definition des Quantenschaltkreis in Zeil 5 erkennbar, benötigt der Quantenschaltkreis $4n + 2$ Qubits anstatt $2n + 3$. Die Reduktion der benötigten Qubits wird in der Sektion 7.3 thematisiert.

Abbildung 26: Periodenbestimmung in Qiskit

```
1  def Shor(a: int, N: int,number_shots: int = 1000, backend: str = "aer_simulator"):
2    n = N.bit_length()
3    c_qbits = range(2*n)
4    ev_qbits = list(range(len(c_qbits),4*n+2))
5    qc = qiskit.QuantumCircuit(4*n+2,len(c_qbits))
6    for c_bit in c_qbits:
7      qc.h(c_bit)
8    qc.x(ev_qbits[0])
9    for c_bit in c_qbits:
10     qc.append(U_gate(n,dezToBin(mod_exp(a,2**c_bit,N),n+1),dezToBin(N,n+1)),[c_bit]+ev_qbits)
11   qc.append(QFT_Gate(len(c_qbits),inverse = True, MSB_first = False,swaps = True), c_qbits)
12   qc.measure(c_qbits,c_qbits)
13   simulator = qiskit.Aer.get_backend(backend)
14   sim_result = qiskit.execute(qc, backend=simulator, shots=number_shots).result()
15   return sim_result.get_counts(qc)
```

In Abbildung 27 ist der Quantenschaltkreis zu sehen, welcher durch die Funktion `Shor` mit $a = 7;\ N = 15$ generiert wird. Bei $N = 15$ tritt der Effekt auf, das für $x > 1$ die Rechnung $(a^{2^x}) \mod N = 1$ ergibt, weswegen auch die ${U_a}^{2^x}$-Gatter zu U_1 werden.

Abbildung 27: Quantenalgorithmus von Shor in Qiskit



7.2 Faktorisierungsalgorithmus

Der Faktorisierungsalgorithmus kombiniert die Quanten-Phase-Estimation zur Periodenberechnung mit der klassischen Nachberechnung. Der Hauptfokus der Implementierung liegt darauf, die Schwächen der Messergebnisse der Quanten-Phase-Estimation mithilfe klassischer Methoden zu kompensieren. Es ist zu erwarten, dass die Ausführung von Quantenberechnungen auch mit zukünftig fortschrittlichen Quantencomputern kostenintensiv sein wird [Sho97]. Daher wäre es ineffizient, die die Quanten-Phase-Estimation so oft zu wiederholen, bis ein Ergebnis erzielt wird, das die Periode direkt offenbart. Stattdessen ist es ressourcensparender, ein erhaltenes Messergebnis unter Berücksichtigung der drei möglichen Szenarien die in Abschnitt 6.4 erklärt werden, mit klassischen Methoden aufzubereiten. Als Resultat können so weitere Durchläufe der Quanten-Phase-Estimation eingespart und durch klassische Rechenleistung ersetzt werden.

Die Grafik 28 repräsentiert insgesamt 512 Messungen für $a = 19;\ N = 119$, mit

einer Genauigkeit k von $2\text{ld}(119) \approx 14$ und soll die Besonderheiten, welche bei Messergebnissen auftreten können, hervorheben. Zur Verdeutlichung sind die Messergebnisse in Abbildung 28 in vier Kategorien eingeteilt. Die Kriterien für die Einteilung in diese Kategorien basieren darauf, ob nach der Division durch 2^k und der Anwendung des Kettenbruch-Algorithmus mit einem Limit von N, die Periode erfolgreich extrahiert werden kann.

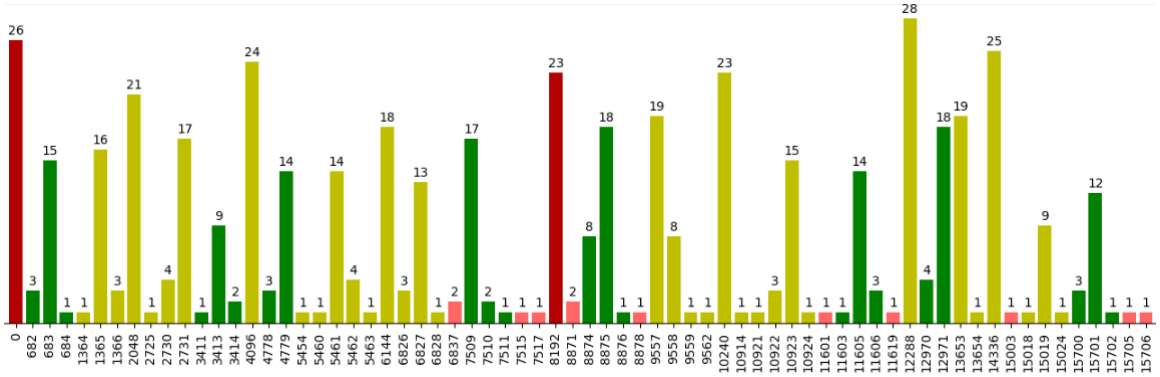
Die grünen Ergebnisse offenbaren direkt die gesuchte Periode von $p = 24$. Ein Faktorisierungsalgorithmus der solange die Quanten-Phase-Estimation wiederholt, bis ein Messergebnis die ungekürzte Periode enthält, wird ausschließlich bei den grünen Ergebnissen erfolgreich sein.

Anhand der gelben Ergebnisse kann die gesuchte Periode nicht direkt extrahiert werden. Dabei ist der Fall eingetreten, dass der Zähler und Nenner des Bruches einen gemeinsamen Teiler hatte, wodurch der Bruch gekürzt werden konnte. Gegebenenfalls kann die Periode mit klassisch ausführbaren Methoden rekonstruiert werden.

Die hellroten Ergebnisse sind zu weit vom Peak entfernt, weswegen diese die gesuchte Periode weder vollständig noch gekürzt enthalten. Womöglich kann von diesem Messergebnis die Periode gefunden werden, wenn man die benachbarten Zustände überprüft.

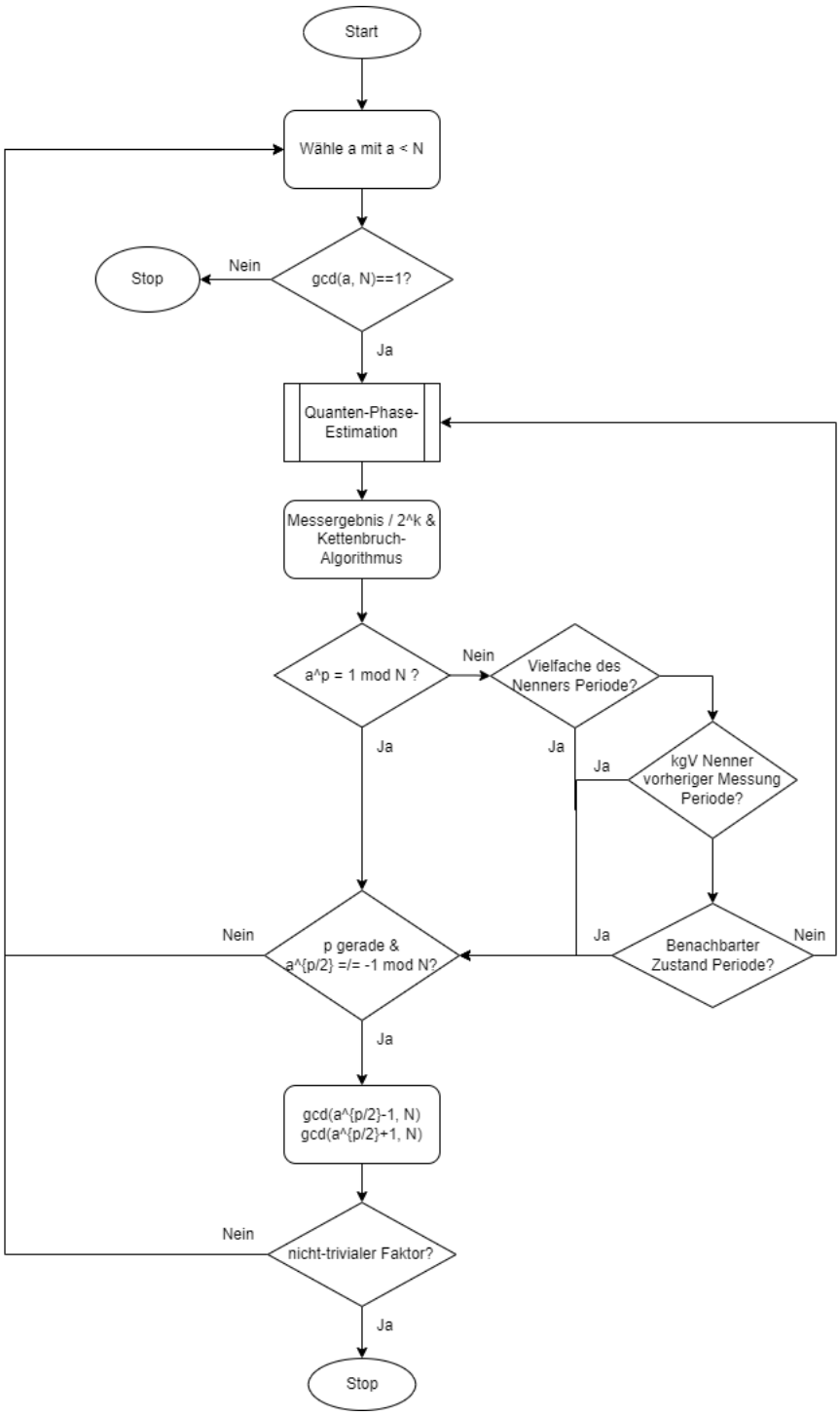
Die beiden dunkelroten Zustände sind die beiden Zustände der finale Superposition $\left|2^k \cdot \frac{0}{p}\right\rangle_k$ und $\left|2^k \cdot \frac{p/2}{p}\right\rangle_k$. Der Zustand $\left|2^k \cdot \frac{0}{p}\right\rangle_k$ existiert für jede Zahl und erlaubt keine Rückschlüsse auf die Periode. Der andere Zustand $\left|2^k \cdot \frac{p/2}{p}\right\rangle_k$ existiert ausschließlich wenn p gerade ist. Auch dieser Zustand gewehrt keine hilfreichen Rückschlüsse auf die Phase, da $\frac{p/2}{p}$ immer zu $\frac{1}{2}$ gekürzt wird. Eine notwendige Bedingung um die Faktoren aus p herleiten zu können, ist, das p gerade ist [Sho97], dies wird zwar durch die Messung von $\left|2^k \cdot \frac{p/2}{p}\right\rangle_k$ bestätigt, jedoch kann aus dieser Information keine hilfreiche Konsequenz gezogen werden.

Abbildung 28: Beispiel Messung



Das Flussdiagramm in Abbildung 29 beschreibt den

Abbildung 29: Flussdiagramm Faktorisierung



7.3 Optimierung

8 Resultate

9 Verlauf

9.1 Rückblick

9.2 Ausblick

Literaturverzeichnis

[23a] *Gemeinsame Umfrage von BSI und KPMG in Deutschland zu „Kryptografie und Quantencomputing“*. 2023. URL: https://www.bsi.bund.de/DE/Service-Navi/Presse/Alle-Meldungen-News/Meldungen/BSI_KMPG_Quanten_230418.html (besucht am 02. 08. 2023).

[23b] *IBM Quantum Development Roadmap*. 2023. URL: https://www.ibm.com/quantum/roadmap (besucht am 02. 08. 2023).

[23c] *Unveiling our new Quantum AI campus*. 2023. URL: https://blog.google/technology/ai/unveiling-our-new-quantum-ai-campus/ (besucht am 02. 08. 2023).

[Ano23] AnonymousKet. *Quantum Phase Estimation*. Zugriff am: 14.08.2023. 2023. URL: https://anonymousket.medium.com/quantum-phase-estimation-e2910e8ef8ec.

[Bea03] Stephane Beauregard. *Circuit for Shor's algorithm using 2n+3 qubits*. 2003. arXiv: quant-ph/0205095 [quant-ph].

[Ben80] Paul Benioff. „The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines“. In: *Journal of Statistical Physics* 22.5 (1980), S. 563–591. DOI: 10.1007/BF01011339.

[Cor+09] Thomas H. Cormen u. a. *Introduction to Algorithms*. 3rd. MIT Press, 2009, S. 963.

[Deu85] David Deutsch. „Quantum theory, the Church–Turing principle and the universal quantum computer“. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400 (1985), S. 97–117. DOI: 10.1098/rspa.1985.0070.

[DH76] W. Diffie und M. Hellman. „New directions in cryptography“. In: *IEEE Transactions on Information Theory* 22.6 (1976), S. 644–654. DOI: 10.1109/TIT.1976.1055638.

[Dir39] P. A. M. Dirac. „A new notation for quantum mechanics“. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 35.3 (1939), S. 416–418. DOI: 10.1017/S0305004100021162.

[DiV00] David P. DiVincenzo. „The Physical Implementation of Quantum Computation“. In: *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. URL: https://doi.org/10.1002%2F1521-3978%28200009%2948%3A9%2F11%3C771%3A%3Aaid-prop771%3E3.0.co%3B2-e.

[Dra00] Thomas G. Draper. *Addition on a Quantum Computer*. 2000. arXiv: quant-ph/0008033 [quant-ph].

[Fey82] Richard P. Feynman. „Simulating physics with computers“. In: *International Journal of Theoretical Physics* 21.6 (1982), S. 467–488. DOI: 10.1007/BF02650179. URL: https://doi.org/10.1007/BF02650179.

[Gro96] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: quant-ph/9605043 [quant-ph].

[Hoe22] Georg Hoever. *Kryptologie*. Skript zur Vorlesung Kryptologie, Fachhochschule Aachen. 2022, S. 28.

[Hoe23a] Georg Hoever. *Münzbeispiel*. Mündlich/Symbolisch in der Vorlesung Quantencomputing, Fachhochschule Aachen. 2023.

[Hoe23b] Georg Hoever. *Quanten Computing*. Skript zur Vorlesung Quanten Computing, Fachhochschule Aachen. 2023, S. 107.

[Hom] Matthias Homeister. *Quantum Computing verstehen*. 5. Aufl. Springer, S. 215.

[IBM23] IBM. *IBM Q 16 Rueschlikon Backend Information*. Zugriff am: 14.08.2023. 2023. URL: https://github.com/Qiskit/ibmq-device-information/tree/master/backends/rueschlikon/V1.

[JM98] J. A. Jones und M. Mosca. „Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer“. In: *The Journal of Chemical Physics* 109.5 (Aug. 1998), S. 1648–1653. DOI: 10.1063/1.476739. URL: https://doi.org/10.1063%2F1.476739.

[KL23] Jonathan Katz und Yehuda Lindell. *Introduction to modern cryptography*. 2. Aufl. Boca Raton, FL: CRC Press, 2023. ISBN: 9781466570269.

[Koe21] Wolfram Koepf. *Elementare Arithmetik und Algebra. Modulare Arithmetik*. Abgerufen am: 20.08.2023. 2021. URL: https://www.mathematik.uni-kassel.de/~koepf/ElAriAl/modular.pdf.

[ME99] Michele Mosca und Artur Ekert. *The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer*. 1999. arXiv: quant-ph/9903071 [quant-ph].

[NC10] Michael A. Nielsen und Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[RG17] Lidia Ruiz-Perez und Juan Carlos Garcia-Escartin. „Quantum arithmetic with the quantum Fourier transform“. In: *Quantum Information Processing* 16.6 (2017), S. 152. ISSN: 1573-1332. DOI: 10.1007/s11128-017-1603-1. URL: https://doi.org/10.1007/s11128-017-1603-1.

[Sho97] Peter W. Shor. „Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer“. In: *SIAM Journal on Computing* 26.5 (Okt. 1997), S. 1484–1509. DOI: 10.1137/s0097539795293172. URL: https://doi.org/10.1137%2Fs0097539795293172.

[Tea23] Qiskit Development Team. *Reset — Qiskit 0.x documentation*. Accessed: 19.08.2023. 2023. URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.Reset.html.

[Zal98] Christof Zalka. *Fast versions of Shor's quantum factoring algorithm*. 1998. arXiv: quant-ph/9806084 [quant-ph].

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Aachen, den 8. September 2023