AℓZAGK-Seminar: Quantencomputer-Algorithmen IV: Von einfachen Gattern zum Addierer mod N

Claas Grenzebach

8. Mai 2001

Zusammenfassung

Quantencomputer verwenden unitäre Transformationen für ihre einzelnen Rechenschritte. Im folgenden wird dargestellt, wie man von einfachen Gattern, die unitäre Transformationen beschreiben, zu einem reversiblen n-Bit-Addierer kommt, der modulo einer vorgegebenen Zahl N rechnet.

Darauf aufbauend kann man dann auch einen Multiplizierer und einen Potenzierer für n Bits konstruieren, worauf hier nicht näher eingegangen werden soll.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Unitäre Transformationen als Schaltbilder

Wenn man logische Gatter für mehr als ein Q-Bit durch Matrizen darstellen möchte, hat man das Problem, daß deren Größe mit steigender Anzahl an Q-Bits rasch zunimmt und die Matrizen unübersichtlich werden. Es gibt jedoch eine alternative Schreibweise, nämlich als Schaltbilder.

Zur Erinnerung:

Ein Q-Bit ist ein zweidimensionaler Hilbertraum H mit der Basis $|0\rangle$ und $|1\rangle$. Bezüglich dieser Basis kann man die Vektoren $a|0\rangle + b|1\rangle$ auch als $\binom{a}{b}$ schreiben, die Basisvektoren sind dann $\binom{1}{0}$ und $\binom{0}{1}$.

Systeme von mehreren Q-Bits erhält man per Tensorprodukt: $H_1 \otimes \cdots \otimes H_n$. Die zugehörigen Basisvektoren lauten:

$$|b_1, \ldots, b_n\rangle := |b_1\rangle|b_2\rangle \ldots |b_n\rangle := |b_1\rangle \otimes \cdots \otimes |b_n\rangle,$$

wobei $|b_j\rangle$ ein Basisvektor von H_j ist.

Häufig ist es nützlich, diese Schreibweise weiter abzukürzen und die hintereinander notierten Ziffern als Binärzahl zu interpretieren:

$$b_1, \ldots, b_n \longleftrightarrow b_1 \cdot 2^{n-1} + \cdots + b_n \cdot 2^0 = \sum_{j=0}^{n-1} b_{n-j} \cdot 2^j.$$

Diese Binärzahl kann dann auch durch die zugehörige Dezimalzahl ersetzt werden.

Beispiel:

$$|9\rangle = |1001\rangle = |1\rangle \otimes |0\rangle \otimes |0\rangle \otimes |1\rangle$$

Da die betrachteten Räume der Q-Bits endlichdimensional sind, genügt es, (unitäre) Transformationen auf diesen Räumen durch ihr Verhalten auf den Basisvektoren zu charakterisieren, das heißt, durch ihre Matrixdarstellung.

1.1 Einfache Beispiele: Nicht, XOR, Toffoli

Die Schreibweise unitärer Transformationen als Schaltbild soll nun an einigen einfachen Beispielen eingeführt werden.

Ein logisches Nicht kann man sich auf dem $\mathbb{F}_2$ als Addition[1] von 1 vorstellen: $\neg a = a \oplus 1$. Für ein Q-Bit $|a\rangle$, das dieser Operation unterzogen werden soll, bedeutet das den Tausch der Basisvektoren. Das logische Nicht wird also durch die (unitäre) Transformation bewirkt, deren Matrix im folgenden angegeben ist; die Darstellung als Schaltbild ist eine Linie mit einem Kreuz ($\times$, passend zum Tausch der Basisvektoren):

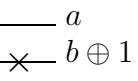
$$\text{Nicht} \quad \widehat{=} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \widehat{=} \quad$$

Diese Schreibweise läßt sich einfach auf Systeme von Q-Bits ausdehnen. Betrachten wir beispielsweise die obige Nicht-Operation, jedoch auf einem System zweier Q-Bits, wobei das Nicht lediglich auf das zweite Q-Bit wirken soll. Dann lautet die zugehörige Matrix: $I \otimes \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. Die Einheitsmatrix I wird im Schaltbild als eine bloße Linie dargestellt, und jedes Q-Bit erhält seinen eigenen „Eingang“ in die Schaltung. Hier hat man also:

$$\text{Nicht (2. Bit)} \quad \widehat{=} \quad I \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \widehat{=} \quad$$

Manchmal ist es bequem, die Eingangs- und Ausgangszustände links und rechts an die Schaltung zu schreiben. Dabei sind dann auf der linken Seite *stets* Basisvektoren gemeint. So kann man im letzteren Beispiel notieren:

$$\left(I \otimes \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)\right)\Big(|a\rangle|b\rangle\Big) = |a\rangle|b \oplus 1\rangle \quad \widehat{=} \quad$$



Da Basisvektoren klassischen Bits entsprechen, wird es auf diese Weise möglich, die Schaltbilder wie klassische Schaltdiagramme zu lesen. Der einzige Unterschied ist dabei, daß die Schaltbilder zu unitären Transformationen auch „rückwärts“ (von rechts nach links) gelesen werden können, womit man dann die jeweils inverse Transformation erhält.

Außerdem funktioniert die Schaltung bzw. Transformation im Gegensatz zur klassischen Schaltalgebra auch für Linearkombinationen als Eingaben – man braucht nur die Ausgaben der Schaltbilder entsprechend linear zu kombinieren.

Beide genannten Merkmale sind wesentlich für Quantencomputer.

[1] Addition heißt hier Rechnen modulo 2

Wohl die einfachste Transformation, die man sich auf zwei Q-Bits vorstellen kann, ist das exklusive Oder („XOR“; $\oplus$) mit der Zuordnung $|a, b\rangle \mapsto |a, a \oplus b\rangle$, im einzelnen:

$$\begin{aligned} |00\rangle &\mapsto |00\rangle, & |10\rangle &\mapsto |11\rangle, \\ |01\rangle &\mapsto |01\rangle, & |11\rangle &\mapsto |10\rangle. \end{aligned}$$

Anders betrachtet, liegt hier ein kontrolliertes Nicht vor (nur wenn das erste Bit 1 ist, wird das zweite „umgedreht“). Das führt zu der in der Abbildung 1 aufgeführten Schreibweise. Die zugehörige Matrix ist:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

wobei die Basisvektoren wie üblich nach der Größe der Binärzahlen sortiert wurden:

$$|00\rangle \mathrel{\widehat{=}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle \mathrel{\widehat{=}} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle \mathrel{\widehat{=}} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle \mathrel{\widehat{=}} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Ebenso kann man sich ein von den ersten beiden Bits kontrolliertes Nicht denken (nur wenn beide 1 sind, wird das dritte Bit „umgedreht“); es trägt den Namen *Toffoligatter*. Die zugehörige Matrix ist:[2]

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Wie man leicht nachprüfen kann, sind beide Transformationen tatsächlich unitär.

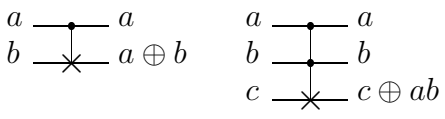


Abbildung 1: XOR- und Toffoli-Gatter

[2]Hier sieht man deutlich den Vorteil der Schreibweise als Schaltbild!

2 Klassischer Halbaddierer, Volladdierer

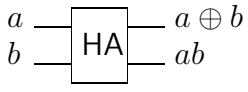
Die Addition zweier (binärer) Zahlen geschieht bei dem bekannten Verfahren der schriftlicher Addition, indem die einzelnen Stellen (von der rechten Seite angefangen) nacheinander abgearbeitet werden, wobei jedesmal ein Übertrag entsteht.

Beispiel: Addition der Binärzahlen 1010 (dezimal: 10) und 0111 (dezimal: 7)

$$\begin{array}{cccccc} & & 1 & 0 & 1 & 0 \\ + & & 0_1 & 1_1 & 1_0 & 1 \\ \hline & 1 & 0 & 0 & 0 & 1 \end{array}$$

Die einfachste Operation ist dabei die zuerst durchgeführte Addition der letzten Bits ganz rechts. Bei ihr braucht noch kein Übertrag aus einer früheren Rechnung berücksichtigt zu werden. Die letzte Stelle des Ergebnisses ist dann die Summe der beiden Bits (modulo 2), ein eventueller Übertrag ihr Produkt. Ein Schaltelement, das genau dies leistet, nennt man *Halbaddierer*. Die Summe der Eingangsbits ist nichts anderes als ein exklusives Oder („XOR“), und für den Übertrag als Produkt kann man das Toffoligatter verwenden:

$$\begin{array}{cc} & a \\ + & b \\ \hline ab & a \oplus b \end{array} \quad \widehat{=} \quad$$



$\widehat{=}$

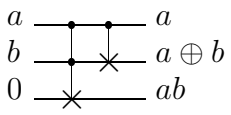
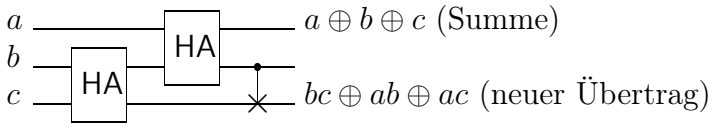


Abbildung 2: Halbaddierer

Will man nicht nur zwei einzelne Bits addieren, sondern wie in obigem Beispiel mehrstellige Dualzahlen, so ist ein weiterer Addierer notwendig, der neben zwei Eingabebits ein Übertragsbit verarbeitet. Ein derartiger *Volladdierer* läßt sich wie folgt konstruieren:

$$\begin{array}{cc} & a \\ + & b_c \\ \hline bc \oplus ab \oplus ac & a \oplus b \oplus c \end{array} \quad \widehat{=} \quad$$



Ersetzen der Halbaddierer durch die jeweiligen Gatter führt zu:

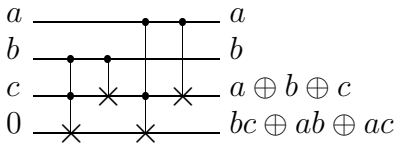


Abbildung 3: Volladdierer

Werden der Halb- und der Volladdierer, wie in den Abbildungen 2 und 3 auf der vorherigen Seite dargestellt, nur aus XOR- und Toffoligattern aufgebaut, sind beide Operationen reversibel, da sie Produkte von unitären Transformationen sind. Mit diesen Elementen lassen sich also Addierer und Multiplizierer für Quantencomputer bauen. Im nächsten Abschnitt wird der Addierer für n Bits beschrieben.

3 Addition zweier n Bit langer Zahlen

3.1 Addierer für n Bits

Im Prinzip genügte es, die im vorigen Abschnitt erstellten Halb- und Volladdierer gemäß den Vorschriften des schriftlichen Addierens zusammenzusetzen. Für ein n-Bit-Addierwerk bräuchte man dann einen Halbaddierer und $n-1$ Volladdierer.

Nun arbeitet ein Quantencomputer jedoch reversibel. Das bedeutet insbesondere, daß die Überträge jedes einzelnen Additionsschrittes bis zum Ende der Rechnung stehenbleiben. Diese nur als Hilfsgrößen verwendeten Bits werden dabei auf Werte gesetzt, die mit den anderen Ausgabegrößen verschränkt sind. Man kann die Übertragsbits also nicht einfach wieder zurück auf Null setzen.

Um diese Bits in weiteren Rechnungen verwenden zu können, müssen sie aber wieder zurück in ihren Ausgangszustand gebracht werden. Eine Möglichkeit, dies zu erreichen, ohne die anderen Bits zu beeinflussen, ist, einige Operationen ein zweites Mal, nur diesmal rückwärts, anzuwenden (siehe unten). Dadurch wird die Erstellung eines Addierers für n Bits komplizierter.

Als erstes wird der Volladdierer in zwei unabhängige Teile zerlegt; der eine berechnet nur den Übertrag ($bc \oplus ab \oplus ac$), der andere nur die Ein-Bit-Summe ($a \oplus b \oplus c$). Wie man der Abbildung 3 auf der vorherigen Seite leicht entnimmt, sind diese Teile die folgenden:

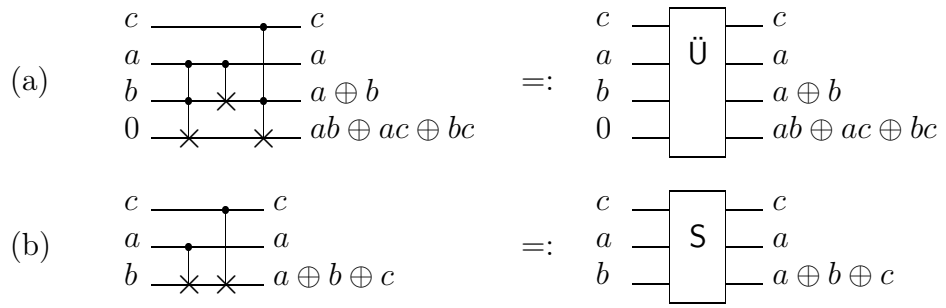


Abbildung 4: Übertrag (a) und Summe (b)

Jetzt kann man bei den einzelnen Additionsschritten zuerst den jeweiligen Übertrag[3] bestimmen. Das tut man vom letzten Bit ausgehend solange, bis man das führende Bit erreicht hat. Dort verwendet man einen kompletten Volladdierer und besitzt so bereits die beiden führenden Bits des zu bestimmenden Ergebnisses. Die übrigen Bits erhält man – schrittweise vom vordersten bis zum hintersten –, indem die jeweilige Übertragsrechnung rückgängig[4] gemacht und die Summe[5] berechnet wird.

Beispiel: Der Einfachheit halber möchte ich diesen Prozeß zunächst für zwei 3-Bit-Zahlen ($|a\rangle = |a_2a_1a_0\rangle$ und $|b\rangle = |b_2b_1b_0\rangle$) angeben, die zu dem Ergebnis $|c\rangle = |a + b\rangle$ addiert werden sollen:

	a_2	a_1	a_0
$+$	b_2	b_1	b_0
c_3	c_2	c_1	c_0

Dabei ist, wie man leicht nachrechnet:

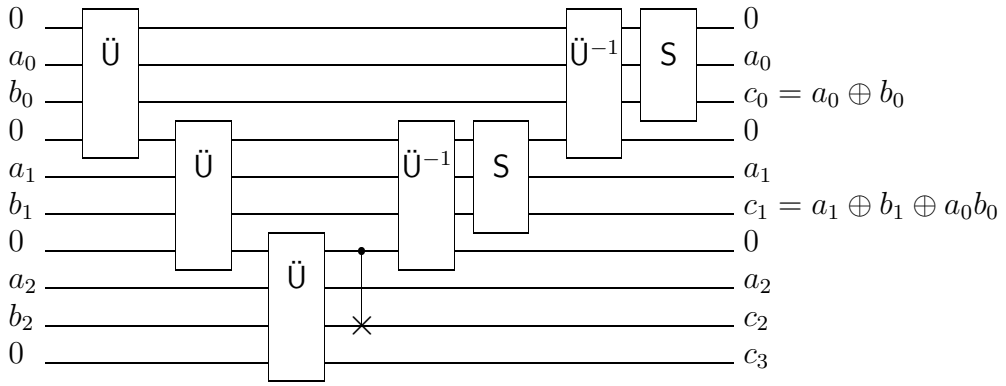
$c_0 = a_0 \oplus b_0$ Übertrag: $d_0 = a_0b_0$

$c_1 = a_1 \oplus b_1 \oplus a_0b_0$ Übertrag: $d_1 = a_1b_1 \oplus d_0(a_1 \oplus b_1)$

$c_2 = a_2 \oplus b_2 \oplus d_1$ Übertrag: $d_2 = a_2b_2 \oplus d_1(a_2 \oplus b_2)$

$c_3 = d_2 = a_2b_2 \oplus (a_1b_1 \oplus a_0b_0(a_1 \oplus b_1))(a_2 \oplus b_2)$,

und dazu gehört nach den obigen Überlegungen das folgende Schaltbild:

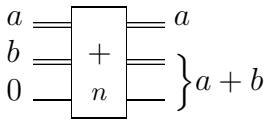


[3] gemäß Abbildung 4(a) auf der vorherigen Seite

[4] mit der Umkehrung von Abbildung 4(a) auf der vorherigen Seite – dann ist das Übertragsbit Null, und es liegen wieder die Werte der ursprünglichen Eingabe vor

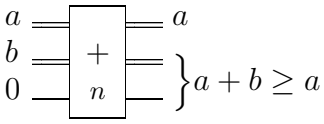
[5] gemäß Abbildung 4(b) auf der vorherigen Seite

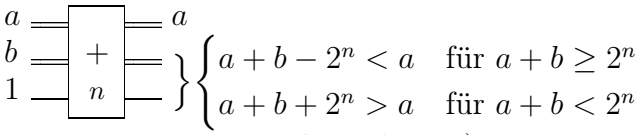
Führt man die im Beispiel bereits angedeutete Stufenform fort (mit $n-3$ weiteren vorgeschalteten Übertragsgattern und ebenso vielen nachgeschalteten inversen Übertrags- inklusive Summengattern), so erhält man einen n-Bit-Addierer. Dieser soll ab jetzt abkürzend geschrieben werden als:



Dabei bedeuten die Doppellinien, daß ein ganzes Register von je n Q-Bits übergeben wird. Die Null stellt das zusätzliche Bit dar, das zum Speichern des Übertrags an der führenden Stelle gebraucht wird (in obigem Schaltbild ganz unten). Ebenso besteht die Ausgabe $a+b$ aus $n+1$ Bits.

Was passiert, wenn das zusätzliche Bit 1 ist statt 0? Offensichtlich ändert sich nichts an den hinteren n Bits von $a+b$. Lediglich die führende Stelle wird ins Gegenteil verkehrt (aus 0 wird 1 und umgekehrt). Das bedeutet, daß das Ergebnis dann nicht mehr $a+b$ ist, sondern vielmehr um 2^n erhöht bzw. erniedrigt wurde. Im einzelnen gilt:



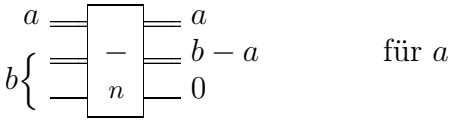


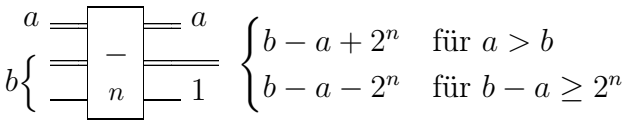
(Nach Voraussetzung: $0 \le a < 2^n$ und $0 \le b < 2^n$)

Abbildung 5: Addierer für n Bits

Als unitäre Transformation kann man den n-Bit-Addierer auch von rechts nach links lesen, er stellt dann einen n-Bit-Subtrahierer dar. Dieser erhält als Eingabe eine n-Bit-Zahl a und eine $(n+1)$-Bit-Zahl b. Ist $a \le b$ und $b-a < 2^n$, so erhält man problemlos die Ausgabe $(a, b-a, 0)$, das zusätzliche Bit ist also 0.

Ist $a \le b$ oder $b-a < 2^n$ nicht erfüllt, so wird das zusätzliche Bit 1 und zeigt so den aufgetretenen Überlauf an. Im einzelnen hat man im Umkehrung der drei beim Addierer aufgetretenen Fälle:





(Nach Voraussetzung: $0 \le a < 2^n$ und $0 \le b < 2^{n+1}$)

Abbildung 6: Subtrahierer für n Bits

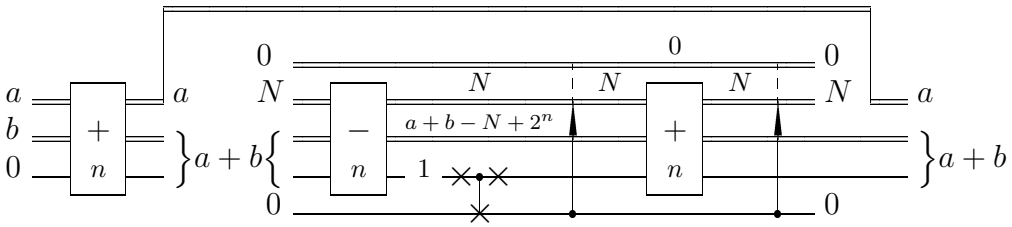
Weiß man bereits, daß beide Zahlen a und b kleiner als 2^n sind, so kann man den Subtrahierer auch zum Größenvergleich verwenden. Das zusätzliche Bit wird dann nämlich genau dann 1, wenn $a > b$ ist.

3.2 Addierer modulo *N*

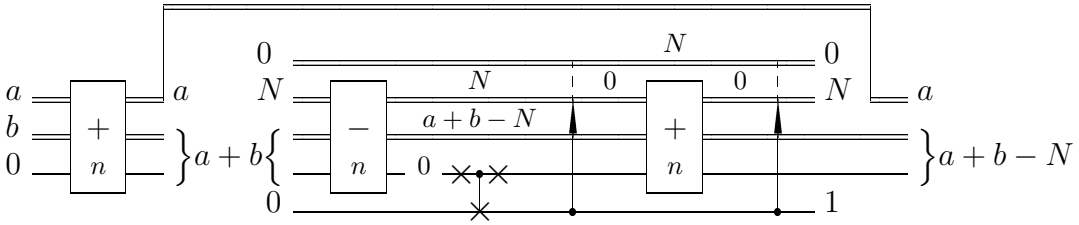
Als nächstes soll ein Addierer (mod N) gebildet werden, das heißt, ein Schaltelement, das die Operation $|a, b\rangle \mapsto |a, (a + b) \bmod N\rangle$ zu einem vorgegebenen $N < 2^n$ und $0 \leq a < N; 0 \leq b < N$ ausführt.

Zunächst muß auf jeden Fall die Summe $a + b$ gebildet werden, was mit Hilfe des im vorangegangenen Abschnitt definierten n-Bit-Addierers geschieht. Danach sind zwei Fälle zu unterscheiden: $0 \leq a+b < N$ und $N \leq a+b < 2N$; im letzteren ist von der Summe $a + b$ noch N abzuziehen. Man geht nun so vor, daß in jedem Fall N subtrahiert wird. Ist dann das eine zusätzliche Bit der Subtrahiererausgabe 0, so war $N \leq a+b < 2N$. Andernfalls (bei $0 \leq a+b < N$) wurde fälschlicherweise N abgezogen, was rückgängig gemacht werden muß. Je nach Wert des zusätzlichen Bits ist somit noch 0 oder N zu addieren. Der ganze Vorgang wird durch folgendes Schaltnetz erreicht:

Für $a + b < N$:

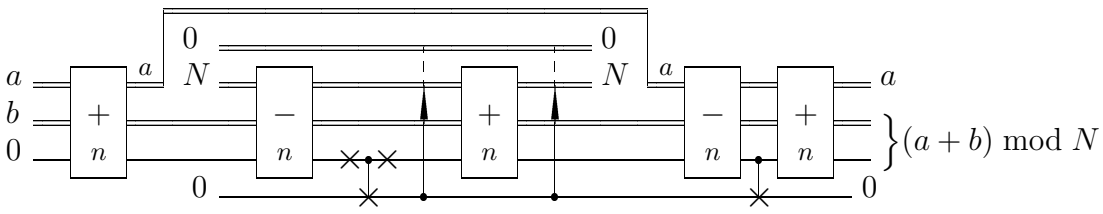


Für $N \leq a + b$:



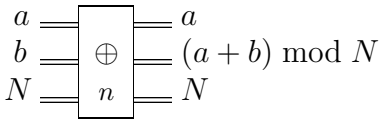
Dabei bedeutet ein Pfeil, daß der Inhalt der beiden mit einer gestrichelten Linie verbundenen Register (N und 0) getauscht wird, falls das Kontrollbit 1 ist (realisierbar mit mehreren Toffoligattern, siehe Bemerkung am Schluß). Durch die zweimalige Anwendung befindet sich das Register mit N am Ende der Rechnung wieder in seinem Anfangszustand, dafür ist das kontrollierende zusätzliche Bit eventuell nicht 0.

Um auch das zusätzliche Kontrollbit wieder auf seinen anfänglichen Wert 0 zu setzen, ist noch ein weiterer abschließender Schritt nötig:



Dabei wird von dem bereits berechneten Ergebnis $(a + b)$ mod N (das heißt, von $a + b$ bzw. $a + b - N$) noch einmal die Zahl a subtrahiert. Es ergibt sich b (ohne Überlauf) bzw. $b - N + 2^n$ (mit Überlauf). Aber das bedeutet, daß das Kontrollbit in jedem Fall wieder auf seinen Anfangswert 0 gesetzt wird – wie gewünscht. Zum Schluß wird die Subtraktion rückgängig gemacht.

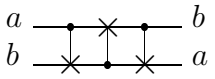
Es gilt stets: $(a + b)$ mod $N < N$; wegen $N < 2^n$ ist somit auch das Ergebnis $(a + b)$ mod N auf jeden Fall nur eine n-Bit-Zahl. Insgesamt kann der Addierer (mod N) daher abkürzend wie folgt geschrieben werden:



(Nach Voraussetzung: $0 \leq a < N$ und $0 \leq b < N$ mit $0 \leq N < 2^n$)

Abbildung 7: Addierer modulo N für n Bits

Bemerkung: Um den Inhalt zweier Register zu tauschen, verwendet man pro Q-Bit die folgende Transformation:



Setzt man anstelle der XOR-Gatter Toffoligatter ein, so wird die Vertauschung nur durchgeführt, falls ein weiteres Kontrollbit 1 ist – wie oben benötigt.

$