Technische Universität Graz

Quantenfehlerkorrekturcodes

Christian Hartler

22. September 2009

Bachelorarbeit

Betreuer: Univ.-Prof. Dr.rer.nat. Arrigoni, Enrico

Inhaltsverzeichnis

1 Vorwort **1**

2 Einleitung **2**

2.1 Unterschiede zwischen klassischen und Quantencomputer[1] 2

2.2 Quantengatter[2] 4

3 Fehler im Quantencomputer[3] **5**

4 Quantenfehlerkorrektur[4] **7**

4.1 Bitfehlerkorrektur 7

4.2 Phasenfehlerkorrektur 10

5 9QECC - 9 Quantum Error Correcting Code[5] **12**

5.1 Kodierung 12

5.2 Messung 13

5.3 Dekodierung 16

6 7QECC - 7 Quantum Error Correcting Code[6] **17**

6.1 Klassische Theorie der Fehlerkorrekturcodes 17

6.2 Umsetzung 20

6.2.1 Kodierung 20

6.2.2 Fehlererkennung 21

6.2.3 Dekodierung 25

7 Simulation der Fehlerkorrekturcodes mit Matlab **26**

7.1 Matlabfunktionen 26

7.2 Änderungen in den QECCs für die Simulation 30

7.2.1 9QECC 30

7.2.2 7QECC 31

8 Anhang- Verzeichnisse **33**

Abbildungsverzeichnis

1 Redundanz [1] 7
2 Schaltplan: Bitfehler [1] 9
3 Schaltplan: Phasenfehler [1] 11
4 Schaltplan: Kodierung des 9QECCs [1] 13
5 Schaltplan: Messung im 9QECC [1] 15
6 Schaltplan: Dekodierung des 9 QECCs [1] 16
7 Schaltplan: Kodierung des 7QECCs [1] 21
8 Schaltplan zur Messung von Eigenwerten [1] 23
9 Schaltplan: Messung im 7 QECC [1] 24
10 Schaltplan: Dekodierung des 7 QECCs [1] 25
11 Diese Abbildung zeigt wie mit mehreren Toffoli Gatter und QBits im Basiszustand (die unteren zwei QBits in der Abbildung) drei QBits (die oberen drei Qbits in der Abbildung) auf $|1>$ überprüft werden kann. 26
12 Schaltplan: Simulation der Messung und Korrektur des Bitfehlers im 9QECC mit Matlab 30
13 Schaltplan: Simulation der Messung und Korrektur des Phasenfehlers im 9QECC mit Matlab 31
14 Schaltplan: Simulation der Messung und Korrektur im 7QECC mit Matlab . . . 32

1 Vorwort

Mit dieser Bachelorarbeit möchte ich einen Einblick in die Problematik der auftretenden Fehler in einem Quantencomputer und deren Korrektur geben. Die sogenannten Fehlerkorrekturcodes oder auf Englisch Quantum Error Correcting Codes (kurz: QECC), die diese Korrektur ermöglichen, sind jedoch keine neuartige Entwicklung, sondern basieren auf Fehlerkorrekturcodes, die bereits für den klassischen Computer entwickelt wurden. Sie werden in der Nachrichtentechnik als Kanalkodierung bezeichnet, da es sich hierbei um die Übertragung von digitalen Daten über gestörte Kanäle handelt.

Beim Quantencomputer werden diese Störungen durch Wechselwirkungen des Quantenmechanischen Bits (kurz: QBit) mit der Umgebung, aber auch durch Gattern verursacht. Die Realisierung dieser Codes musste umgestellt werden, da der Quantencomputer andere Gattern verwendet als der klassische Computer.

Ziel dieser Arbeit ist es, sowohl die Funktionsweise der beiden Korrekturcodes 7QECC und 9QECC (die Zahl vor dem QECC gibt die Anzahl der verwendeten QBits für die Kodierung an) zu recherchieren, als auch diese mit dem Programm Matlab zu simulieren.

2 Einleitung

In diesem Kapitel wird die Logik und die Gatter des Quantencomputers anhand eines Vergleichs mit dem klassischen Computer erklärt.

2.1 Unterschiede zwischen klassischen und Quantencomputer[1]

Klassischer Computer

- Bits werden über die angelegte Spannung bestimmt:
  Bit 0 = 0 bis 1 Volt
  Bit 1 = 3 bis 5 Volt
- Es tritt nur ein Zustand auf entweder das Bit 0 oder das Bit 1
- Gatter überschreiben die Zustände (Irreversibilität)
- Da es nur einen Zustand gibt, muss ein Operationsschritt auf alle Zustände angewendet werden.

Quantencomputer

- QBits können auf 3 verschiedene Arten bestimmt werden:
  - Spin (in meiner Arbeit werde ich hauptsächlich diese Art verwenden):
    Bit 0 $\triangleq$ QBit $|0> = |\uparrow>$
    Bit 1 $\triangleq$ QBit $|1> = |\downarrow>$
  - Polarisationszustände von Photonen:
    QBit $|0> = |Rechtspolarisiert>$
    QBit $|1> = |Linkspolarisiert>$
  - 2 metastabile Zustände eines Atoms:
    QBit $|0> = |Grundzustand>$
    QBit $|1> = |metastabilerZustand>$

[1] Vgl. Quelle [2] , Kapitel 1.2, 3.1 bzw. 4, Seite 6, Seite 24 bzw. Seite 29

- Es treten mehrere Zustände auf (Superpositionsprinzip):
  $|\Phi >= a|0 > +b|1 >$
  a und b sind Koeffizienten, Φ ist der quantenmechanische Zustand und es gilt die Normierung: $|a|^2 + |b|^2 = 1$

- Gatter sind reversibel, d.h. Zustände können wieder rücktransformiert werden

- Datendurchsatz ist um ein Vielfaches höher, da mehrere Zustände verwendet werden und 1 Operationschritt sie alle verändern kann.

- Messung zerstört den Quantenmechanischen Zustand. Die Konsequenz daraus ist, dass es unmöglich ist einen Quantenzustand komplett zu bestimmen (Unschärferelation).
  Bsp.: Die Messung von $|\Phi >= a|0 > +b|1 >$ lässt diesen Zustand mit einer gewissen Wahrscheinlichkeit in $|0 >$ oder $|1 >$ zerfallen.

- Quantenmechanische Bits besitzen im Gegensatz zu klassischen Bits Phasen. Die relative Phase kontrolliert die Orientierung des Spins und steckt in den Koeffizienten der QBits und wird über die Exponentialfunktion beschrieben: $e^{i\psi x}|x >$ (diese e-Funktion wird als Phasenfaktor bezeichnet)
  Beispiel: Spin Polarisationen in X- bzw. Y Richtung

  - X-Richtung:
    $|S_x, + >= | \rightarrow >= \frac{1}{\sqrt{2}}(| \uparrow > +| \downarrow >)$, $e^{i0} = 1$
    $|S_x, - >= | \leftarrow >= \frac{1}{\sqrt{2}}(| \uparrow > -| \downarrow >)$, $e^{i\Pi} = -1$
  - Y-Richtung:
    $|S_y, + >= \frac{1}{\sqrt{2}}(| \uparrow > +i| \downarrow >)$, $e^{i\frac{\Pi}{2}} = +i$
    $|S_y, - >= \frac{1}{\sqrt{2}}(| \uparrow > -i| \downarrow >)$, $e^{i\frac{\Pi}{2}} = -i$

- Das Wechselwirken zweier Zustände führt zu deren Verschränkung. Im Quantencomputer geschieht dies über Gatter.
  Bsp. für einen Verschränkten Zustand: $|\Phi >= a|00 > +b|01 > +c|10 > +d|11 >$ mit $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$
  Gegenbeispiel: $|\Phi > |\Psi >= (a|0 > +b|1 >) * (c|0 > +d|1 >)$ mit $|a|^2 + |b|^2 = 1$ bzw. $|c|^2 + |d|^2 = 1$

- Es treten Interferenzen auf, die bei geschickter Anwendung schnellere Algorithmen erzeugen.

2.2 Quantengatter[2]

Gatter die nur auf 1 QBit angewendet werden

- Not Gatter
- Phase-Shift Gatter
- Hadamard Gatter

Gatter die auf mehrere QBits angewendet werden

- C-Not Gatter
- Toffoli Gatter

Not Gatter

Das Not Gatter entspricht einem Bit-Flip (oder Spin Flip):

$$Not|0>\rightarrow|1> \tag{1}$$

bzw.

$$Not|0>\rightarrow|1> \tag{2}$$

Hadamard Gatter

Das Hadamard Gatter bewirkt eine Drehung des Spins um 90°. Dies wäre äquivalent zur Bitschreibweise:

$$H|0>\rightarrow \frac{1}{\sqrt{2}}(|0>+|1>) \tag{3}$$

$$H|1>\rightarrow \frac{1}{\sqrt{2}}(|0>-|1>) \tag{4}$$

Phase-shift Gatter

Das Phase-Shift-Gatter ist ein 1-QBit Gatter, das die relative Phase zwischen $|1>$ und $|0>$ verschiebt:

$$U\psi|x>=e^{i\psi x}|x> \tag{5}$$

[2]Vgl. Quelle [2] , Kapitel 4.1 bzw. 4.2, ab Seite 29 und [4], Kapitel 5.2,Seite 22

C-Not Gatter

Das C-Not Gatter ist ein 2-QBit Gatter. Es wendet auf das 2. QBit nur dann Not an, wenn das 1. QBit $|1>$ ist.

Toffoli Gatter

Das Toffoli Gatter ist eine Erweiterung des C-Not Gattern. Es werden 2 Eingänge betrachtet und falls beide QBits 1 sind, dann wendet es auf das 3. QBit Not an, andernfalls macht es nichts. Bei geeignetem Aufbau (z.B. durch das Hinzufügen eines Hilfs-QBits im Zustand $|0>$ und einem weiterem Toffoli Gatter) kann das Toffoli Gatter mehr als 2 Eingänge überprüfen, ob alle $|1>$ sind. Dies wird in meinem Programm bei der Fehlerkorrektur ausgenutzt.

3 Fehler im Quantencomputer[3]

Ein großes Problem bei Quantencomputern stellt die hohe Fehleranfälligkeit dar. Wir unterscheiden 2 Arten:

1. **Fehler durch Gattern**
   z.B. Dauer und Phase eines Laserimpulses. Es handelt sich hier um ein unitären Fehler.

2. **Dekohärenz**
   bezeichnet die Wechselwirkung des quantenmechanischen Systems mit der Umgebung. Dieser Fehler entspricht einer nicht unitären Transformation.

In einem klassischen Computer wäre der Fehler eine Bitdrehung. Hierbei nutzt die Fehlerkorrektur die Redundanz aus, das heißt jedes Bit wird in N-Kopien gespeichert. Nach einer gewissen Zeit wird das Bit kontrolliert und falls eine Bitdrehung stattgefunden hat nach der Mehrheitsregel korrigiert. In einem Quantencomputer kann diese Methode der Redundanz nicht angewendet werden, weil es nicht möglich ist quantenmechanische Zustände zu kopieren. Es gilt das sogenannte No cloning Theorem: "Es ist nicht möglich ein System zu bauen, das jedes beliebige QBit perfekt auf ein anderes QBit kopiert, ohne dabei das ursprüngliche zu verändern." (Ohne das No Cloning Theorem könnte der Ort eines Zustandes x beliebig genau gemessen werden und bei der Kopie der Impuls p, dies ist aber eine Verletzung der Heisenberg'schen Unschärferelation).
Hinzukommt, dass neben dem Bitfehler auch Phasenfehler auftreten können:

[3]Vgl. Quelle [1], Kapitel 10.2.1 bzw. 10.2.2, ab Seite 196 und Quelle [2], Kapitel 8, ab Seite 87

$$|0> \rightarrow |0>$$
$$|1> \rightarrow -|1>$$

Bit- und Phasenfehler können auch gleichzeitig auftreten. Diese 3 Fehlertypen lassen sich durch die Paulimatrizen darstellen:

- Bitfehler
  entspricht $\sigma_X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \triangleq |1><0| + |0><1| = X$
- Phasenfehler
  entspricht $\sigma_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \triangleq |0><0| - |1><1| = Z$
- Bit- und Phasenfehler
  entspricht $i * \sigma_Y = \sigma_Z * \sigma_X = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \triangleq |0><1| - |1><0| = Y$

Die später behandelte Fehlerkorrektur wird noch zusätzlich erschwert, da die 3 Fehlertypen nicht diskret, sondern kontinuierlich sind. Das heißt: wäre ein Startzustand $a|0> + b|1>$ gegeben, können kleine Fehler in den Koeffizienten a und b auftreten. Diese Fehler sind nicht diskret, das bedeutet sie sind nicht eindeutig aufzuspüren und sie bauen sich auf. Daher braucht die Korrektur eine Messung, die wiederum das quantenmechanische System stört.

4 Quantenfehlerkorrektur[4]

4.1 Bitfehlerkorrektur

Wenden wir uns zunächst dem klassischen Bitfehler zu, hierbei muss auf die Redundanz eingegangen werden. Im Gegensatz zum klassischen Computer kann ein quantenmechanischer Zustand nicht perfekt kopiert werden, ohne den zu kopierenden Zustand zu verändern (No Cloning Theorem). Stattdessen wird die Verschränkung ausgenutzt. Dabei werden die QBits auf folgende Weise ersetzt:

$$|0> \rightarrow |000>$$
$$|1> \rightarrow |111>$$
$$\text{also}$$
$$|\psi> = a|0> + b|1> \rightarrow a|000> + b|111>$$

Dies stellt keine Kopie des Zustands dar. Eine Kopie des vorigen Zustands hätte die Form:

$$|\psi> \rightarrow (a|0> + b|1>)(a|0> + b|1>)(a|0> + b|1>)$$

Das Netzwerk zur Erstellung dieses Zustandes ist in Abb.1 zu sehen.

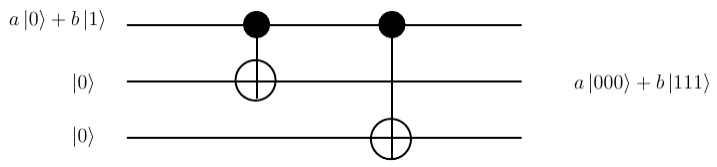

Abbildung 1: Schaltplan zur Erzeugung einer Redundanz für ein QBit ohne das No-Cloning-Theorem zu verletzen. Es werden zwei C-Not Gatter vom ersten QBit ausgehend auf die beiden anderen angewendet, um die drei Zustände zu verschränken

[4]Vgl. Quelle [1], Kapitel 10.2.1 bzw. 10.2.2, ab Seite 196 und [2], Kapitel 8.2, ab Seite 88

Tabelle 1: Es sind die verschiedenen Wahrscheinlichkeiten für das Auftreten von Fehlern in QBits, die durch einen Kanal, der diese Fehler auslöst, übertragen werden, angegeben. p stellt die Wahrscheinlichkeit für das Auftreten eines Fehlers in einem QBit dar, während 1-p die Gegenwahrscheinlichkeit angibt und dass dieser Fehler nicht aufgetreten ist. Daher errechnet sich die Wahrscheinlichkeit für drei zu übertragende QBits beispielsweise für Fehler, die alle drei QBits befallen, aus der Wahrscheinlichkeit, dass das erste QBit befallen wird mal der Wahrscheinlichkeit, dass das zweite QBit befallen wird, usw. Damit ergibt sich eine Gesamtwahrscheinlichkeit von p^3, dass alle drei QBits befallen werden. Daher sollte p möglichst klein sein. Hier ist das Start-QBit $|\Phi >= a|000 > +b|111 >$

QBit nach der Übertragung	Wahrscheinlichkeit
$a\|000> +b\|111>$	$(1-p)^3$
$a\|100> +b\|011>$	$p(1-p)^2$
$a\|010> +b\|101>$	$p(1-p)^2$
$a\|001> +b\|110>$	$p(1-p)^2$
$a\|110> +b\|001>$	$p^2(1-p)$
$a\|101> +b\|010>$	$p^2(1-p)$
$a\|011> +b\|100>$	$p^2(1-p)$
$a\|111> +b\|000>$	p^3

Nun schicken wir dieses kodierte QBit durch einen Kanal. Dort kann zu einer gewissen Wahrscheinlichkeit (siehe Tabelle 1) ein Fehler auftreten. Nehmen wir an, dass im ersten QBit tatsächlich ein Bit-Flip Fehler aufgetreten wäre. Im klassischen Fall wird der Zustand gemessen, doch hier würden wir den Zustand verändern und die Information über die lineare Kombination (Koeffizienten a,b,usw.) ginge verloren. Darum werden 2 Hilfs-QBits eingeführt, die die 3 Zustände auf Ungleichheit überprüfen sollen. Dies geschieht folgendermaßen:
Bezeichnen wir die Basiszustände als

$$|b1, b2, b3>$$

dann werden die Größen

$$(b1 + b2, b1 + b3) \tag{6}$$

betrachtet, wobei die Summen als modulo 2 zu betrachten sind. Dies ist äquivalent zur Aussage:

$$(b1 \stackrel{?}{=} b2, b1 \stackrel{?}{=} b3)$$

Die 2 Hilfs-QBits werden im Grundzustand verwendet und auf ihnen wirken CNOT-Gatter, wie in Abb.2 gezeigt wird, um Gl.6 zu erfüllen. Diese 2 Hilfs-QBits werden danach gemessen

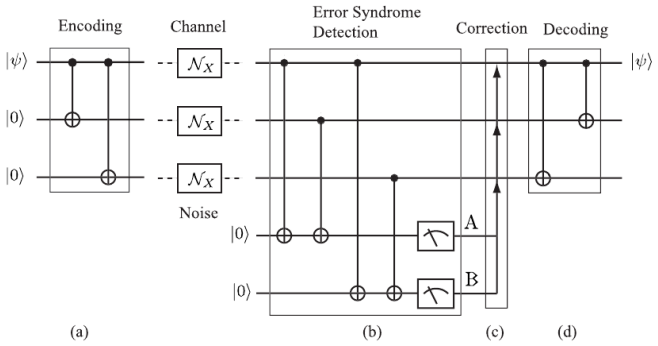

Abbildung 2: Schaltplan zur Korrektur eines Bitfehlers. Im ersten Segment wird das QBit $|\Psi>$ durch Verschränkung mit zwei QBits im Grundzustand kodiert. Danach wird es durch einen Kanal geschickt, der einen Bitfehler (entspricht dem Flippen des Spins in der X-Basis) in einem der drei QBits erzeugt geschickt. Im zweiten Segment werden zwei Hilfs-QBits hinzugenommen. Durch die C-Not Gatter und den darauf folgenden Messungen kann überprüft werden, in welchem QBit der Fehler aufgetreten ist, um ihn danach zu korrigieren. Im letzten Segment wird das QBit $|\Psi>$ wieder dekodiert

und in unserem Fall ergibt die Messung $(|1>,|1>) \Longleftrightarrow (b1 \neq b2, b1 \neq b3)$. Dadurch wissen wir, dass ein Fehler im QBit b1 aufgetreten ist und können diesen korrigieren. Die Messung (nichtlokal) dieser beiden Hilfs-QBits hat die Grundinformation der anderen 3 QBits (lokal) nicht zerstören. Im diskutierten Fall handelte es sich um einen diskreten Fehler. Nun gehen wir auf einen kontinuierlichen Fehler ein. Durch solch einen Fehler wird das QBit auf folgende Weise transformiert

$$|0> \rightarrow cos\alpha|0> -i*sin\alpha|1> \tag{7}$$

bzw.

$$|1> \rightarrow -i*sin\alpha|0> +cos\alpha|1> \tag{8}$$

$$\Rightarrow a|000> +b|111> \rightarrow cos\alpha(a|000> +b|111>) - i*sin\alpha(|100> +b|011>) \tag{9}$$

(siehe Phase-shift Gatter)

Betrachten wir nun das gleiche Problem, dass im 1. QBit ein Bit-Flip Fehler aufgetreten ist. Durch die Messung können wir 2 Ergebnisse bekommen:

- $(|0>,|0>)$ mit einer Wahrscheinlichkeit von $P = cos^2\alpha$
  $|\psi>$ zerfällt in $a|000> +b|111>$

- $(|1>,|1>)$ mit einer Wahrscheinlichkeit von $P = sin^2\alpha$
  $|\psi>$ zerfällt in $a|100> + b|011>$

Im 1. Fall wurde der Fehler durch die Messung korrigiert. Im 2. Fall hat sich der Fehler verschlechtert, aber dadurch wissen wir, dass der 1. QBit korrigiert werden muss.

4.2 Phasenfehlerkorrektur

Auch hier können diskrete bzw. kontinuierliche Fehler auftreten. Betrachten wir dazu das kodierte QBit von der Bitfehlerkorrektur auf das ein diskreter Phasenfehler wirkt:

$$a|000> + b|111> \rightarrow a|000> - b|111>$$

Der Phasenfehler, der auf eines der QBits b_i wirkt, hat auch Einfluss auf die restlichen QBits. Die Lösung des Problems besteht darin diese Methode nicht auf die QBits, sondern die Phase anzuwenden. Dazu wird auf jedes einzelne QBit ein Hadamard Gatter angewandt:

$$\begin{gathered} H_3H_2H_1(a|000> + b|111>) \\ \rightarrow H_3H_2(\tfrac{a}{\sqrt{2}}(|0> + |1>)|00> + \tfrac{b}{\sqrt{2}}(|0> - |1>)|11>) \\ \rightarrow \tfrac{a}{\sqrt{8}}|+++> + \tfrac{b}{\sqrt{8}}|---> \end{gathered}$$

Gehen wir davon aus, dass der Fehler nur auf einer Phase stattfindet. Im Fall eines diskreten, unitären Phasenfehlers beispielsweise auf den ersten Zustand:

$$|0> \pm|1> \rightarrow |0> \mp|1>$$

Nach dem Fehler modifiziert sich der Zustand:

$$\tfrac{a}{\sqrt{8}}|+++> + \tfrac{b}{\sqrt{8}}|---> \rightarrow \tfrac{a}{\sqrt{8}}|-++> + \tfrac{b}{\sqrt{8}}|+-->$$

Danach werden wieder Hadamard-Gattern auf alle QBits angewendet:

$$H_3H_2H_1(\tfrac{a}{\sqrt{8}}|-++> + \tfrac{b}{\sqrt{8}}|+-->) \rightarrow a|100> + b|011>$$

Dadurch wird erreicht, dass ein Phasenfehler zu einem Bitfehler wird und dieser kann mit der Bitfehlerkorrektur korrigiert werden. (siehe Abb.3)

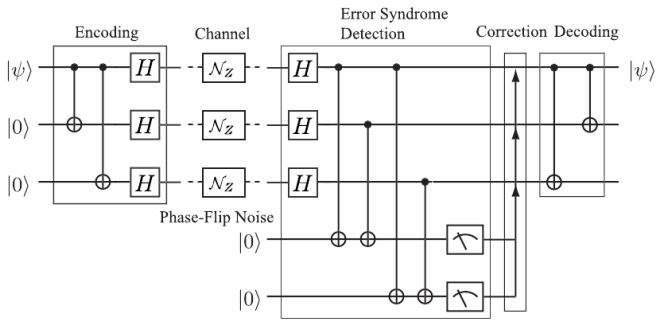

Abbildung 3: Schaltplan für die Korrektur eines Phasenfehlers. Der Aufbau und die Funktionsweise sind gleich wie sie in Abb.2 beschrieben werden, mit dem Unterschied, dass vor und nach dem Kanal der den Fehler erzeugt Hadamard Gatter platziert werden, die den Spin der QBits innerhalb des Kanals in die Z-Basis legen, da der Phasenfehler den Spin der Z-Basis flippt.

Wie würde das Problem bei einem kontinuierlichen Fehler aussehen? Gehen wir wieder davon aus, dass im ersten Zustand ein Fehler aufgetreten ist, diesmal aber ein kontinuierlicher:

$$\frac{a}{\sqrt{8}}|+++>+\frac{b}{\sqrt{8}}|---> \\ \rightarrow \frac{a}{\sqrt{8}}(|0>+e^{i2\psi}|1>)|++>+\frac{b}{\sqrt{8}}(|0>-e^{i2\psi}|1>)|-->$$

Der kontinuierliche Phasenfehler hat die gleiche Auswirkungen wie ein Phase-Shift-Gatter: $U_\Phi = e^{i\Phi x}|x>$

Wir wenden wieder auf alle QBits Hadamard Gattern an:

- $H_1H_2H_3(\frac{a}{\sqrt{8}}(|0>+e^{i2\psi}|1>)|++>+\frac{b}{\sqrt{8}}(|0>-e^{i2\psi}|1>)|-->)$

$= H_1[\frac{a}{\sqrt{2}}(|0>+e^{i2\psi}|1>)|00>+\frac{b}{\sqrt{2}}(|0>-e^{i2\psi}|1>)|11>]$

$= \frac{a}{\sqrt{4}}[|0>+|1>+e^{i2\psi}(|0>-|1>)]|00>+\frac{b}{\sqrt{4}}[|0>+|1>-e^{i2\psi}(|0>-|1>)]|11>$

$= e^{i\psi}\frac{a}{\sqrt{4}}[e^{-i\psi}(|0>+|1>)+e^{i\psi}(|0>-|1>)]|00>+$
$e^{i\psi}\frac{b}{\sqrt{4}}[e^{-i\psi}(|0>+|1>)+e^{i\psi}(-|0>+|1>)]|11> \rightarrow a(cos\psi|0>-i*sin\psi|1>)|00>$
$+b(cos\psi|1>-i*sin\psi|0>)|11>$

$$= cos\psi(a|000>+b|111>)-i*sin\psi(a|100>+b|011>) \quad (10)$$

Damit sind wir wieder beim kontinuierlichen Bitfehler angelangt. Hiermit wurde gezeigt, dass kontinuierliche Fehler in QBits durch Messungen entweder in diskrete Fehler verwandelt oder ganz zerstört werden. Deshalb werden in den QECCs nur mehr diskrete Fehler betrachtet.

5 9QECC - 9 Quantum Error Correcting Code[5]

5.1 Kodierung

Da wir vorher die Korrekturen nur für Bit-Flip bzw. Phase-Flip Fehler einzeln betrachtet haben, versuchen wir nun die beiden Methoden zu kombinieren. Wir machen dazu wieder zuerst die Phase Redundant und danach die Bits der einzelnen Phasen:

- $a|0> +b|1> \rightarrow a|000> +b|111>$
- $a|000> +b|111> \rightarrow \frac{a}{\sqrt{8}}|+++> +\frac{b}{\sqrt{8}}|--->$
- $\frac{a}{\sqrt{8}}(|0> +|1>)|++> +\frac{b}{\sqrt{8}}(|0> -|1>)|-->$

$\rightarrow \frac{a}{\sqrt{8}}(|000> +|111>)|\bar{+}> |\bar{+}> +\frac{b}{\sqrt{8}}(|000> -|111>)|\bar{-}> |\bar{-}>$

$= \frac{a}{\sqrt{8}}|\bar{+}\bar{+}\bar{+}> +\frac{b}{\sqrt{8}}|\bar{-}\bar{-}\bar{-}>$

Durch diese Maßnahme haben wir den Shore 95 Code bzw. den 9QECC erzeugt.
Das Netzwerk zur Kodierung ist in Abb.4 zu sehen.

[5]Vgl. Quelle [1], Kapitel 10.3, ab Seite 203

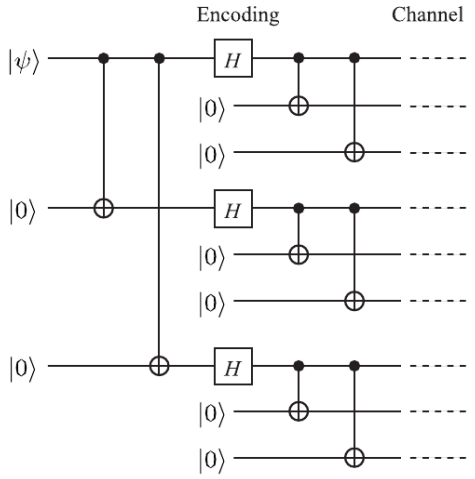

Abbildung 4: Schaltplan zur Kodierung des 9QECC. Dies stellt eine Kombination aus der in Abb.2 und 3 beschriebenen Kodierung dar. Es wir daher zuerst die Phase (Z-Basis) redundant gemacht (vergleiche Abb.3) und nach den Hadamard Gattern die Bits(X-Basis) der kodierten Phase (vergleiche Abb.2). Damit erhalten wir drei logische QBits, die jeweils mit drei QBits kodiert wurden. Dabei entsprechen ungerade Anzahlen an Einsen $|1>_L$ und gerade Anzahlen an Einsen $|0>_L$.

5.2 Messung

Es sei ein Bit- und Phasen-Fehler in einem QBit aufgetreten, beispielsweise das letzte QBit im Netzwerk:

$$\frac{a}{\sqrt{8}}|\bar{+}\bar{+}> (|000>+|111>) + \frac{b}{\sqrt{8}}|\bar{-}\bar{-}> (|000>-|111>)$$
$$\rightarrow \frac{a}{\sqrt{8}}|\bar{+}\bar{+}> (|001>-|110>) + \frac{b}{\sqrt{8}}|\bar{-}\bar{-}> (|001>+|110>)$$

(Bemerkung: Der Phasenfehler hat die gesamte letzte Gruppe betroffen)

Als nächstes werden die einzelnen dreier Gruppen auf Bitfehler untersucht

$$\frac{a}{\sqrt{8}}|\bar{+}\bar{+}> \underline{(|001>-|110>)} + \frac{b}{\sqrt{8}}|\bar{-}\bar{-}> \underline{(|001>+|110>)}$$

⇒Die letzte Gruppe liefert das Syndrom (01) und ein Bitflipoperator muss auf das letzte QBit angewandt werden.

$$\rightarrow \frac{a}{\sqrt{8}}|\bar{+}\bar{+}> \underline{(|000>-|111>)} + \frac{b}{\sqrt{8}}|\bar{-}\bar{-}> \underline{(|000>+|111>)}$$
$$= \frac{a}{\sqrt{8}}|\bar{+}\bar{+}\bar{-}> + \frac{b}{\sqrt{8}}|\bar{-}\bar{-}\bar{+}>$$

Bleibt noch der Phasenfehler. Dazu werden auf alle QBits Hadamard Gattern angewandt, dadurch ändern sich die Gruppen wie folgt:

$$|\bar{+}> \rightarrow \frac{1}{2}(|000> + |011> + |101> + |110>) = |0>_L$$

...gerade Anzahl von 1en

$$|\bar{-}> \rightarrow \frac{1}{2}(|001> + |010> + |100> + |111>) = |1>_L$$

...ungerade Anzahl von 1en

(Die Einteilung in gerader bzw. ungerader Anzahl von 1en in logisch 0 bzw. 1 wird später eine Rolle spielen, da die Netzwerke groß sind und der Überblick bei einer genauen Aufzählung verloren geht)

Diesmal werden die 3 Gruppen gegenübergestellt, dies erfolgt durch das Anlegen einzelner CNOT-Gattern die von einzelnen Gruppenteilen auf die Vergleichs-QBits wirken:

$$\frac{a}{\sqrt{8}}|001>_L + \frac{b}{\sqrt{8}}|110>_L$$

Dadurch entsteht das Syndrom (01) und ein Bitflipoperator muss auf die gesamte letzte Gruppe angewendet werden.

Bei einem kontinuierlichen Fehler verhält es sich hier gleich wie in den Kapiteln 4.1 und 4.2 beschrieben wird. Der Zustand wird durch die Messung in den Hilfs-QBits entweder verbessert (Fehler verschwindet) oder verschlechtert (kontinuierlicher Fehler wird zum diskreten Fehler). Dies gilt auch für die Messungen beim 7QECC. Das Netzwerk zu den beschriebenen Messungen ist in Abb.5 abgebildet.

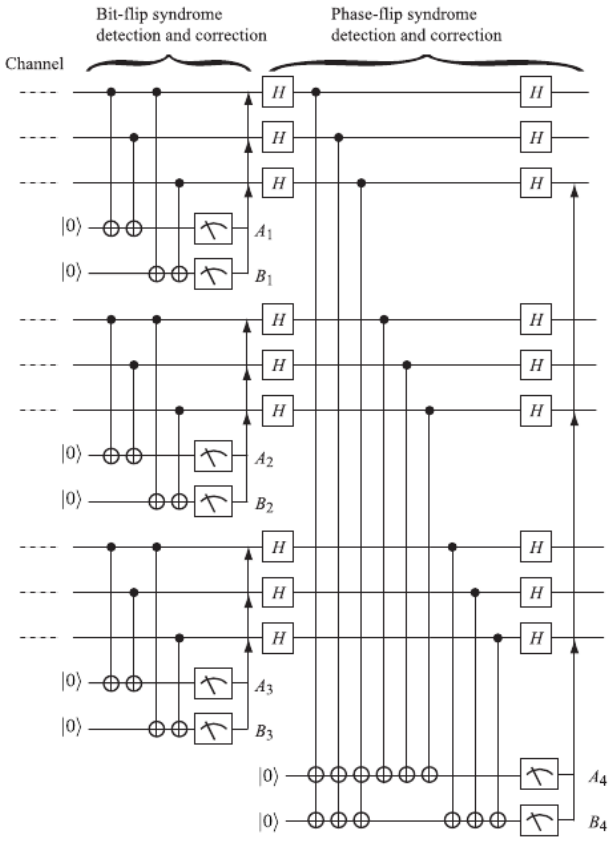

Abbildung 5: Schaltplan zur Messung von Bit- und Phasenfehler für den 9QECC. Im ersten Segment ist die Messung des Bitfehlers aufgezeichnet, wobei innerhalb der logischen QBits (siehe Abb.4) überprüft wird, ob alle QBits den gleichen Spin in der X-Basis haben (diese Messungen laufen gleich ab wie die Bitfehlermessung in Abb.2). Im zweiten Segment werden die Phasenfehler gemessen. Die C-NOT Gatter, die von den einzelnen QBits eines mit drei QBits kodierten logische QBits ausgehen, wirken auf das jeweilige Hilfs-QBits, als wären sie nur ein C-NOT Gatter und ein $|1>_L$ verursacht ein Bitflip im Hilfs-QBit und ein $|0>_L$ nicht. Die Phasenfehlermessung verhält sich gleich, wie sie in Abb.3 beschrieben wird. Nur hier werden anstatt einzelne QBits logische QBits, bestehend aus mehreren QBits, verwendet

5.3 Dekodierung

Bleibt noch die Dekodierung des Zustands, das durch die Inversion der Kodierung erfolgt (siehe Abb.6).

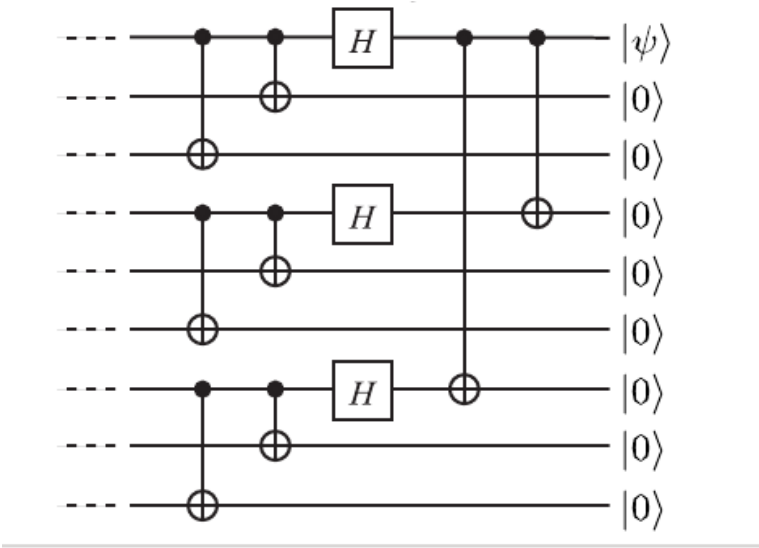

Abbildung 6: Schaltplan für die Dekodierung des 9QECC. Die Dekodierung erfolgt über die umgekehrte Anwendung der Gatter der Kodierung des 9 QECCs (siehe Abb. 4).

6 7QECC - 7 Quantum Error Correcting Code[6]

6.1 Klassische Theorie der Fehlerkorrekturcodes

Nehmen wir an ein Sender, namens Alice, kodiert k Bits an Informationen in einen Code c der aus n ($n > k$) Bits besteht und schickt ihn zu einem Empfänger mit dem Namen Bob durch einen Kanal, der einen Fehler im Code hervorrufen kann. Bob überprüft den Code auf Fehler mit einer Paritätskontrollmatrix H. Wenn kein Fehler auftritt dann sollte

$$Hc^t = 0 \tag{11}$$

sein. Bei Ungleichheit ist ein Fehler aufgetreten und das Ergebnis Hc^t wird das Syndrom genannt. Es sollen nun k=4 Bits an Informationen und einen Code der Länge n=7 übertragen werden. Des weiteren tritt nur ein Bitfehler auf. Nach dem Kanal muss das fehlerbehaftete Bit identifiziert werden. Dazu werden 3 Bits benötigt (Dies kommt durch n-k zustande). 000 ist eine fehlerfreie Übertragung, 001 ist ein Fehler in x_1, 010 ist ein Fehler in x_2 und so weiter. Wenn wir nun die Liste der dreier Bits in eine Matrix aufschreiben, außer den fehlerfreien Fall 000

$$\begin{pmatrix} 001 \\ ... \\ 111 \end{pmatrix}$$

und diese Matrix transponieren, erhalten wir die Paritätskontrollmatrix:

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{12}$$

Bob erhält nun einen fehlerfreien Code $c_1 = (0, 1, 1, 0, 0, 1, 1)$ und wendet H auf c^t an:

$$Hc_1^t = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Wenn nun der erhaltene Code $c_2 = (0, 1, 1, 1, 0, 1, 1)$ ist und die Paritätskontrollmatrix darauf angewendet wird:

$$Hc_1^t \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

[6]Vgl. Quelle [1], Kapitel 10.4, Seite 209 und [3], Kapitel 7.5 bzw. Kapitel 7.9, Seite 21 bzw. Seite 34

Dadurch weiß Bob, dass ein Fehler aufgetreten ist und der erhaltene Binärvektor gibt das fehlerbehaftete Bit an (in unserem Fall das vierte Bit), dieser wird auch als das (Fehler-)Syndrom bezeichnet. Wenden wir auf den Bitstrang $c = (x_1, x_2, ..., x_7)$, den Bob erhält, H an, erhalten wir das Syndrom:

$$Hc^t = \begin{pmatrix} x_4 \oplus x_5 \oplus x_6 \oplus x_7 \\ x_2 \oplus x_3 \oplus x_6 \oplus x_7 \\ x_1 \oplus x_3 \oplus x_5 \oplus x_7 \end{pmatrix} \tag{13}$$

Das Syndrom hängt also nur davon ab, welches Bit durch einen Fehler betroffen wurde und nicht vom Code selbst, den Alice gesendet hat. Die korrekten Codewörter C werden durch den Kern (mod 2) von H gekennzeichnet, C = Kern(H) Die Bedingung $Hc^t = 0$ führt drei Beziehungen zwischen 7 Variablen x_i ein, von denen wir dim C = 7 - 3 = 4 erhalten, mit der Übereinkunft, dass k=4 ist. Die Ordnung der Menge C ist deshalb $2^4 = 16$. Der Coderaum C wird erzeugt durch die Matrix:

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{14}$$

Die Matrix M wird als erzeugende Matrix des Fehlerkorrekturcodes C und der Code der durch diese Matrix erzeugt wird, wird als Hamming Code bezeichnet. Bemerkung: M wird durch hinzufügen einer Zeile, bestehend aus Einsen, zur Paritätskontrollmatrix H gebildet. Vektoren in C werden durch das Anwenden eines Vektors v der Größe k=4 auf die erzeugende Matrix (c=v*M) gebildet (siehe Tabelle 2).

Tabelle 2: Es werden die erzeugten Codes für den 7QECC angezeigt. Diese entstehen aus v*M, wobei v ein Vektor der Größe k=4 Bits und M die erzeugende Matrix (siehe Gl.14) darstellt.

v	v*M	v	v*M
(0000)	(0000000)	(1000)	(0001111)
(0001)	(1111111)	(1001)	(1110000)
(0010)	(1010101)	(1010)	(1011010)
(0011)	(0101010)	(1011)	(0100101)
(0100)	(0110011)	(1100)	(0111100)
(0101)	(1001100)	(1101)	(1000011)
(0110)	(1100110)	(1110)	(1101001)
(0111)	(0011001)	(1111)	(0010110)

Die erhaltenen Codes können in gleiche Anteile an gerade (logisch 0) und ungerade (logisch 1) Anzahlen von Einsen aufgeteilt werden. Interessant ist, dass die Codes für logisch 0 durch eine gerade Binärzahl v entstehen also $v = (i_1, i_2, i_3, 0)$, des weiteren sind diese (mod 2)-orthogonal zu allen Codes in C und werden als $C^{\perp}$ bezeichnet zum Beispiel:

$$(1010101).(0111100) = 0 mod 2,$$
$$(1010101).(0010110) = 0 mod 2.$$

Nehmen wir nun nochmal die Paritätskontrollmatrix her, so stellen wir fest, dass sie aus Zeilenvektoren mit gerader Anzahl von Einsen besteht und diese gleich sind wie einige Teile aus $C^{\perp}$. Damit wissen wir, dass jede Zeile von H (mod 2)-orthogonal zu jedem erzeugtem Code ist, und durch das Anwenden von $HM^t = 0$ (M besteht aus Zeilenvektoren die den Codes in C entsprechen, die wiederum mod2-orthogonal zu allen Zeilen von H sind) ist. Daraus folgt, dass $HM^t v^t = Hc^t = 0$ ist.
Gehen wir nun davon aus $c_1 = (x_1, x_2, ..., x_n)$ und $c_2 = (y_1, y_2, ..., y_n)$ sind Codes in C. Der Hammingabstand $d_H(c_1, c_2)$ zwischen c_1 und c_2 ist definiert als die Anzahl der Indizes i bei denen $x_i \neq y_i$ ist. Also jene Anzahl an Bits die zwischen 2 Codes abweichen. Der Hammingabstand muss den Axiomen der Abstände im metrischen Räumen genügen[7]:

1. $d_H(c, c) \geq 0$ für alle c $\in$ C.
2. $d_H(c_1, c_2) = d_H(c_2, c_1)$ für alle $c_1, c_2 \in$ C.
3. $d_H(c_1, c_3) \leq d_H(c_1, c_2) + d_H(c_2, c_3)$ für alle $c_1, c_2, c_3 \in$ C.

Der Minimalabstand $d_H(C)$ des Codes C ist definiert als das Minimum des Hammingabstandes zwischen verschiedenen Elementen von C:

$$d_H(C) = \min_{c,c' \in C} d_H(c, c')$$

In unserem Fall ist der Minimalabstand $d_H(C) = 3$ und dieser muss gegeben sein um zumindest ein fehlerbehaftetes Bit zu korrigieren.
Die maximale Fehleranzahl, die ein Fehlerkorrekturcode korrigieren kann ist gegeben durch:

$$Maxfehler = \lfloor \frac{d_H(C) - 1}{2} \rfloor \tag{15}$$

wobei $\lfloor x \rfloor$ die Abrundungsfunktion bezeichnet.

[7] Vgl. Quelle [6] Kapitel Metrik

6.2 Umsetzung

Aufbauend aus dem klassischen Fehlerkorrekturcode wurde der 7 QECC entwickelt.

6.2.1 Kodierung

Hierbei versuchen wir die Codes aus der klassischen 7 QECC Theorie zu erreichen. Dazu benötigen wir folgende Operatoren für die Bits:

$$M_0 = X_4X_3X_2X_1, M_1 = X_5X_3X_2X_0, M_2 = X_6X_3X_1X_0 \tag{16}$$

und für die Phasen:

$$N_0 = Z_4Z_3Z_2Z_1, N_1 = Z_5Z_3Z_2Z_0, N_2 = Z_6Z_3Z_1Z_0 \tag{17}$$

Dabei gelten für sie folgende Regeln:

$$M_i^2 = N_i^2 = I \tag{18}$$

$$M_i(I + M_i) = I + M_i, [M_i, M_j] = [N_i, N_j] = 0 \tag{19}$$

und

$$[M_i, N_j] = 0 \tag{20}$$

Nun kodieren wir unsere Superposition von 7-QBits

$$\begin{aligned} |0>_L &= \frac{1}{\sqrt{8}}(I + M_0)(I + M_1)(I + M_2)|0>^{\otimes 7} \\ &= \frac{1}{\sqrt{8}}(|0000000> + |1010101> + |0110011> + |1100110> + \\ &\quad |0001111> + |1011010> + |0111100> + |1101001>) \end{aligned} \tag{21}$$

und

$$\begin{aligned} |1>_L &= \frac{1}{\sqrt{8}}(I + M_0)(I + M_1)(I + M_2)|1>^{\otimes 7} \\ &= \frac{1}{\sqrt{8}}(I + M_0)(I + M_1)(I + M_2)\overline{X}|0> \\ &= \frac{1}{\sqrt{8}}(|1111111> + |0101010> + |1001100> + |0011001> + \\ &\quad |1110000> + |0100101> + |1000011> + |0010110>) \end{aligned} \tag{22}$$

Dadurch konnte erreicht werden, dass $|0>_L$ mit $C^\perp$ und die $|1>_L$ mit C-$C^\perp$ aus dem klassischen Hammingcode übereinstimmen. Zusätzlich sind $|0>_L$ und $|1>_L$ Eigenvektoren von M_i und N_i mit den Eigenwerten +1:

$$M_i|x>_L = N_i|x>_L = |x> (x \in GF(2)) \tag{23}$$

Die Umsetzung dieser Kodierung ist in Abb.7 zu sehen. In dieser Schaltung treten vor den algebraischen Operatoren 2 CNOT-Gattern auf. Diese werden für die Kodierung eines QBits $|1>$ in $|1>_L$ benötigt:

$$\begin{aligned} X_0X_1X_2|0>_L &= M_0M_1M_2\overline{X}\frac{1}{\sqrt{8}}(I+M_0)(I+M_1)(I+M_2)|0>^{\otimes 7} \\ &= \frac{1}{\sqrt{8}}(I+M_0)(I+M_1)(I+M_2)\overline{X}|0>^{\otimes 7} \\ &= |1>_L \end{aligned} \tag{24}$$

Bei der Kodierung des QBits $|0>$ in $|0>_L$ sind die CNOT-Gattern inaktiv und damit formt sich die vorige Gleichung in Gleichung 21 um.

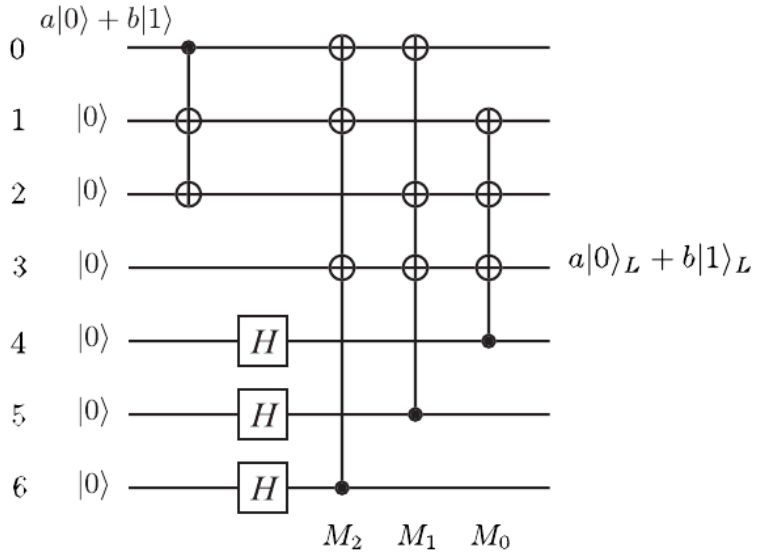

Abbildung 7: Schaltplan zur Kodierung des 7QECC. Die Hadamard Gatter und die C-Not Gatter nach den Hadamard Gattern werden zur Kodierung der $|0>_L$ und $|1>_L$ benötigt (siehe Gl.21 und Gl.22) Die beiden C-Not Gatter am Beginn der Schaltung werden benötigt, um $|1>_L$ zu kodieren, da die QBits eins bis sieben sich im Basiszustand befinden (siehe Gl.(24)).

6.2.2 Fehlererkennung

Die Erkennung des fehlerbehafteten QBits wird anhand eines Beispiels erklärt. Nehmen wir also an, dass ein Fehler, der dem Operator X_0 entspricht, auf das logische QBit $|0>_L$ wirkt. Der resultierende Wert ist ein Eigenwert von N_1 mit dem Wert -1:

$$N_1X_0|0>_L = -X_0N_1|0>_L = -X_0|0>_L \tag{25}$$

Es bestätigt sich, dass folgendes gilt:

$$\begin{aligned} N_2X_0|0>_L = -X_0|0>_L, N_0X_0|0>_L = X_0|0>_L, \\ M_iX_0|0>_L = X_0|0>_L \ (i=0,1,2) \end{aligned} \tag{26}$$

Demnach wird der gestörte Wert $X_0|0>_L$ durch die Eigenwerte (1,1,1;1,-1,-1) der Operatoren $(M_0, M_1, M_2, N_0, N_1, N_2)$ charakterisiert. Diese Tatsache gilt ebenso für $|1>_L$ QBits:

$$N_1 X_0|1>_L = N_1 X_0 \overline{X}|0>_L = X_0 \overline{X} N_1|0>_L = -X_0|1>_L \tag{27}$$

wobei $[\overline{X}, N_i] = 0$ gilt. Dadurch wird gezeigt, dass $X_0|\Psi>$, mit $|\Psi> = a|0>_L + b|0>_L$ $(\forall a, b \in \mathbb{C})$ ein Eigenvektor $(M_0, M_1, M_2; N_0, N_1, N_2)$ mit den Eigenwerten (1,1,1;1,-1,-1) ist. Zur Übersicht der Eigenwerte zu den verschiedenen Fehlersyndromen siehe Tabelle 3,4 und 5.

Tabelle 3: Es werden die Syndrome für Bitfehler aufgelistet. Dabei wird gleich vorgegangen wie in Gl.25, 26 und 27 um die Fehler mit den Operatoren von Gl.17 zu charakterisieren. Da Bitfehler die den Operatoren X_i erzeugen sie zusammen mit den Operatoren von Gl.16 nur die Eigenwerte +1. Das Symbol * steht für den Eigenwert -1.

	X_0	X_1	X_2	X_3	X_4	X_5	X_6
N_0		*	*	*	*		
N_1	*		*	*		*	
N_2	*	*		*			*

Tabelle 4: Es werden die Syndrome für Phasenfehler aufgelistet. Dabei wird gleich vorgegangen wie in Gl.25, 26 und 27 um die Fehler mit den Operatoren von Gl.16 zu charakterisieren. Da Bitfehler die den Operatoren Z_i erzeugen sie zusammen mit den Operatoren von Gl.17 nur die Eigenwerte +1. Das Symbol * steht für den Eigenwert -1.

	Z_0	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6
M_0		*	*	*	*		
M_1	*		*	*		*	
M_2	*	*		*			*

Tabelle 5: Es werden die Syndrome für Bit- und Phasenfehler aufgelistet. Die Fehlercharakteristiken ergeben sich aus den Tabellen 3 und 4 und aus der Beziehung Y=XZ (bzw. $Y_i = X_i Z_i$) folgen die Eigenwerte von $Y_i|\Phi>$. Das Symbol * steht für den Eigenwert -1.

	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6
M_0		*	*	*	*		
M_1	*		*	*		*	
M_2	*	*		*			*
N_0		*	*	*	*		
N_1	*		*	*		*	
N_2	*	*		*			*

Da QBit-Messmethoden keine Phasen ($\triangleq$ Eigenwerte) messen können, muss eine entsprechende Operation angewendet werden, um die Phasenänderung auf eine Bitänderung über zuführen. Dazu muss nur das kontrollierende- und das Ziel-QBit vertauscht werden (siehe Abb.8).

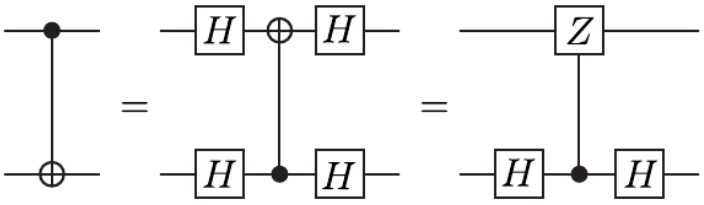

Abbildung 8: Vertauschung des kontrollierenden- und des Ziel-QBits. Damit können die in den Tabellen 3, 4 und 5 beschriebenen Eigenwerte der zugehörigen Fehlercharakteristiken in Bits umgewandelt werden.

Die Schaltung in Abb.8 ist äquivalent zur Aussage:

$$U_{CNOT} = (I \otimes U_H)(I \otimes |0><0| + Z \otimes |1><1|)(I \otimes U_H) \qquad U_H X U_H = Z \tag{28}$$

Wenden wir diese Überlegung auf unser Beispiel an und ersetzen Z durch die Operatoren M_i und N_i, um die Eigenwerte zu erhalten, so lässt sich die Schaltung wie in Abb.9 realisieren. Zur Überprüfung betrachten wir nun den 1. Teilkreis der Abb.9 über die Formel 28.

$$\begin{aligned}&(I \otimes U_H)(I \otimes |0><0| + M_0 \otimes |1><1|)(I \otimes U_H)\\ &= \frac{1}{2}(I + M_0)|\Psi>|0> + \frac{1}{2}(I - M_0)|\Psi>|1>\end{aligned} \tag{29}$$

In der Gleichung ist $|\Psi>$ das kodierte 7-QBit System mit einem möglichen einzelnen Fehler und das letzte QBit ist das Hilfs-QBit. $|\Psi>$ ist ein Eigenvektor von M_i mit den Eigenwerten +1 oder -1. Falls der Eigenwert +1 beträgt, wäre das Ergebnis der Gleichung $|\Psi>|0>$, aber beim Eigenwert -1 hingegen, wäre es $|\Psi>|1>$. Diese Aussage trifft auch auf die restlichen Operatoren M_i und N_i zu. Der Hauptgrund, dass diese Überlegung funktioniert, liegt in Antikommutativität einiger Operatoren M_i und N_i mit dem Fehleroperator (dieser ist ein Tensorprodukt der Paulimatrizen). Die Operatoren M_i und N_i bilden als Menge $\{M_i, N_i\}$ eine kommutierende Abel'sche Gruppe und werden als 'stabilizer generators' [9,10], während der Untervektorraumm C(S)=$\{|\Phi>\in \mathbf{C}^2 | M_i|\Psi>= N_i|\Psi>= |\Psi>\}$ als 'stabilizer code' von S bezeichnet wird. Durch diese Maßnahmen erhalten wir eines der in den Tabellen 3,4 und 5 beschriebene Syndrome.

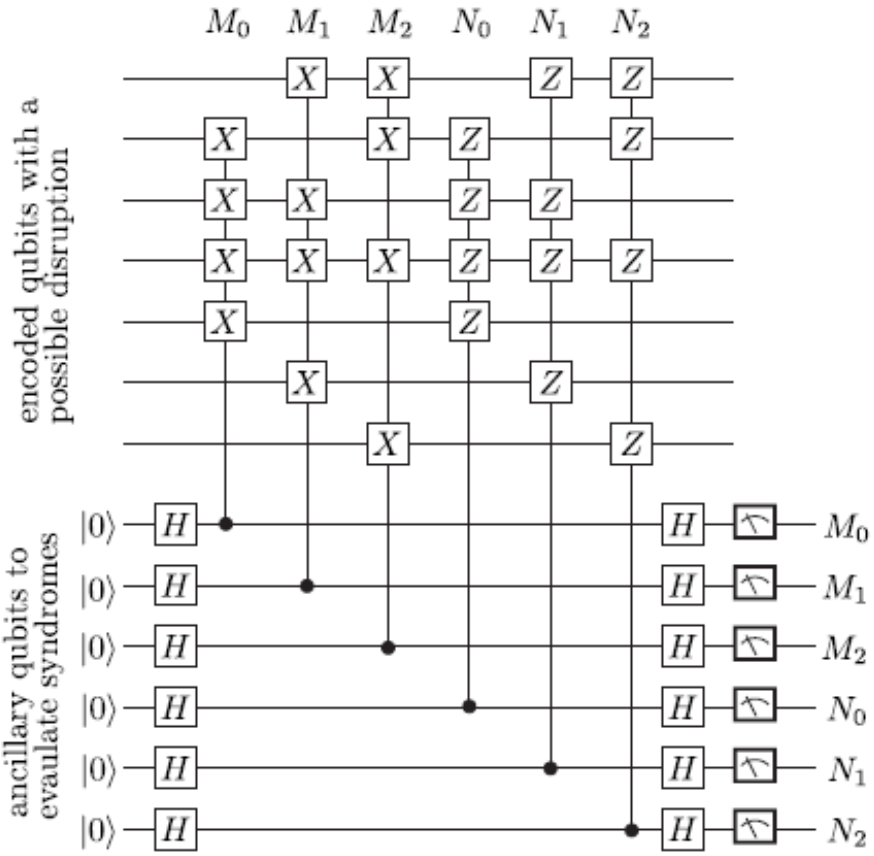

Abbildung 9: Schaltplan zur Messung des Fehlersyndroms. Es werden die in den Tabellen 3, 4 und 5 Fehlercharakteristiken, über die in Abb.8 gezeigte Methode, auf Bitfehler umgewandelt, damit sie messbar werden

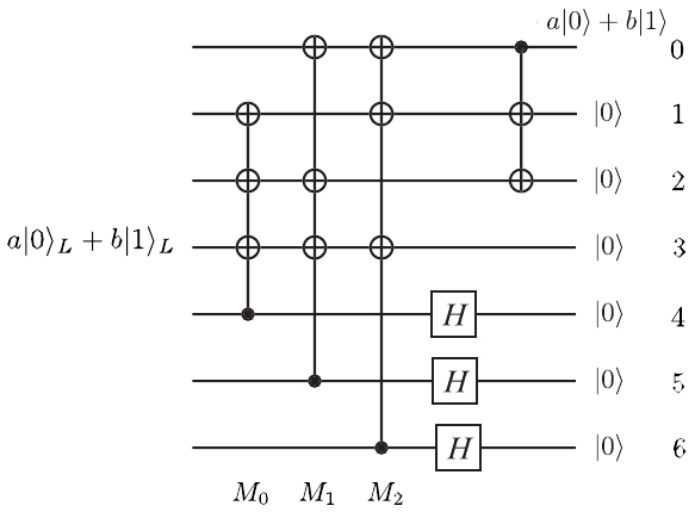

Abbildung 10: Schaltplan zur Dekodierung des 7QECCs. Diese erfolgt durch die umgekehrte Anwendung der Gatter der Kodierung des 7 QECCs(siehe Abb.7).

6.2.3 Dekodierung

Die Dekodierung des 7QECC erfolgt wie beim 9QECC durch Anwendung der invertierten Operatoren der Kodierung (siehe Abb.10).

7 Simulation der Fehlerkorrekturcodes mit Matlab

7.1 Matlabfunktionen

Die Realisierung des 9QECC und des 7QECC in Matlab erfolgte über das Simulieren der einzelnen Quantengatter, die entsprechend für die Fehlerkorrekturcodes zusammengeschaltet wurden. Zunächst möchte ich jedoch erklären wie die einzelnen QBits in meiner Simulation aufgebaut sind:
In der Dirac'schen Schreibweise sieht ein QBit folgendermaßen aus

$$a|xxxx> +b|yyyy> +...$$

und wird in meiner Simulation über die Matrix

$$\begin{pmatrix} xxxxa \\ yyyyb \\ ... \end{pmatrix}$$

dargestellt.
Die korrespondierende Matlabfunktionen der einzelnen Gattern sind in Tabelle 6 angegeben.
Bemerkung zur Tabelle 6: die Matlabfunktion cnot kann auch als Toffoli Gatter verwendet werden wenn, das kontrollierende QBit als Vektor mit verschiedenen kontrollierenden QBits angegeben wird und diese sind nicht begrenzt. Dies kann auch praktisch funktionieren durch das Zusammenschalten von mehrerer Toffoli Gatter und QBits im Basiszustand (siehe als Bsp. Abb.11.

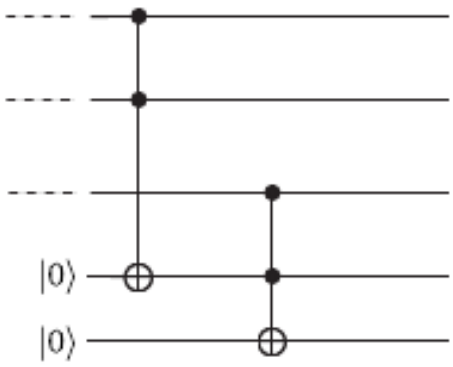

Abbildung 11: Diese Abbildung zeigt wie mit mehreren Toffoli Gatter und QBits im Basiszustand (die unteren zwei QBits in der Abbildung) drei QBits (die oberen drei Qbits in der Abbildung) auf $|1>$ überprüft werden kann.

Tabelle 6: Es sind die programmierten Matlabfunktionen und deren Funktionsweise zu den korrespondierenden Gattern angegeben

Gatter	Matlabfunktion	Funktionsweise der Parameter
NOT	qnot(controlled,qbit)	controlled...Ziel-QBit QBit...QBit in dem die Operation stattfindet
CNOT	cnot(control,controlled,qbit)	control...kontrollierendes QBit controlled...Ziel-QBit QBit...QBit in dem die Operation stattfindet
Hadamard	hadamard(controlled,qbit)	controlled...Ziel-QBit QBit...QBit in dem die Operation stattfindet
Phase-Shift	ugate(phi,controlled,qbit)	phi...Phasenwinkel in rad controlled...Ziel-QBit QBit...QBit in dem die Operation stattfindet
Stern-Gerlach-Messinstrument	messung(phi,mess,qbit)	phi...Phasenwinkel bei einer zuvor ausgeführten Drehung in rad Grund: Zur Wiederherstellung des Koeffizienten wenn aber phi=[], dann bleibt der Koeffizient unangetastet und dessen Anpassung muss händisch erfolgen. mess...zu messendes QBit QBit...QBit in dem die Operation stattfindet

Weitere wichtige Matlabfunktionen sind:

- einf(qbit):

  Nach jedem Hadamard- bzw. Phase-Shift Gatter werden die Zeilen der darstellende Matrix für die QBits verdoppelt. Diese Funktion durchsucht die darstellende Matrix nach gleichen Verschränkungen, addiert die Koeffizienten und verringert dadurch die Anzahlen der Zeilen.

  QBit...QBit in dem die Operation stattfindet

  In dieser Funktion wurde for-Schleifen verwendet und ist auch gleichzeitig die einzige

Funktion in der ich diese anwende. Der Grund liegt darin, dass Matlab zur Berechnung von for-Schleifen viel länger benötigt als andere Programmiersprachen wie zum Beispiel C/C++. Darum muss diese Funktion, obwohl sie nötig ist, mit bedacht eingesetzt werden, weil sonst die Performance der Simulation darunter leidet.

- verschr(wahl,qbit)
  Es wird dabei ein QBit dem bereits bestehenden QBit an letzter Stelle hinzugefügt. Der Eingabeparameter wahl gibt an welche Art von QBit hinzugefügt werden soll. Es steht zur Verfügung:

  0 ...für $|0>$

  1 ...für $|1>$

  2 ...für $|0>+|1>$

  QBit...QBit in dem die Operation stattfindet

- qnorm(qbit):
  Normierung des QBits (Die Funktion ist jedoch nicht fähig, wie im Kap.4.2 durch geschicktes herausheben Real- und Imaginärteil zu trennen)
  QBit...QBit in dem die Operation stattfindet

- deleter(wahl,qbit):
  Der Eingabeparameter wahl, gibt an welches QBit (welche Spalte der darstellenden Matrix) aus dem bestehenden System entfernt werden soll. Wird hauptsächlich zur Entfernung nicht mehr benötigter Hilfs-QBits verwendet.
  QBit...QBit in dem die Operation stattfindet

- read(qbit,figurenumber,title) und readmore(qbit,figurenumber,title):
  Dienen zur Darstellung der darstellenden Matrix in einem Figure und in Dirac'scher Form. Die Funktion readmore ist dabei besser geeignet, da die Funktion read von mir an die verschiedenen Fehlerkorrekturcode angepasst wurde, um die Hilfs-QBits besser hervorzuheben.
  QBit...QBit in dem die Operation stattfindet
  figurenumber...Gibt an in welcher figure die Darstellung erfolgen soll
  title...Titel der Oberhalb der QBits angezeigt wird (dient zur Übersicht)

- qbit9 und qbit7:
  sind die eigentlichen Simulationen und werden mit dem Aufruf gleichnamiger in Matlab gestartet. Weitere Matlab Funktionen die mit qbit9 bzw. mit qbit7 anfangen dienen zur Unterteilung und besseren Übersicht der Schaltpläne. Genaueres wird im nächsten Kapitel erklärt.

- noisychannel(a,phi,fehler,qbit):
  dient zum einbauen des Fehlers.
  a...Winkel der beim kontinuierlichen Bitfehler auftritt (siehe Gl.9)
  phi...Winkel der beim kontinuierlichen Phasenfehler auftritt (siehe Gl.10)
  fehler...Art des Fehlers:

  1 Bitfehler

  2 Phasenfehler

  3 Bitfehler und Phasenfehler

  QBit...QBit in dem die Operation stattfindet

Alle Matlabfunktionen bis auf read, readmore und noisychannel geben nur das veränderte QBit zurück, während noisychannel noch zusätzlich das fehlerbehaftete QBit zurück gibt, um die Schaltung zu kontrollieren. Die Funktionen read und readmore verändern das QBit nicht und haben deshalb keinen Rückgabewert.

7.2 Änderungen in den QECCs für die Simulation

Da die Simulation rein mit den programmierten Funktionen ablaufen sollte, mussten einige Änderungen in den ursprünglichen QECC Schaltkreisen vorgenommen werden.

7.2.1 9QECC

Die Kodierung und Dekodierung hatten den gleichen Aufbau wie in den Abb.4 und 6. Änderungen wurden aber für die Messungen vorgenommen. Nach der Messung des Bitfehlers wurden Toffoli und NOT Gattern eingebaut, um das Fehlersyndrom abzufragen und zu korrigieren (siehe Abb.12 ein Vergleich siehe Abb.5).

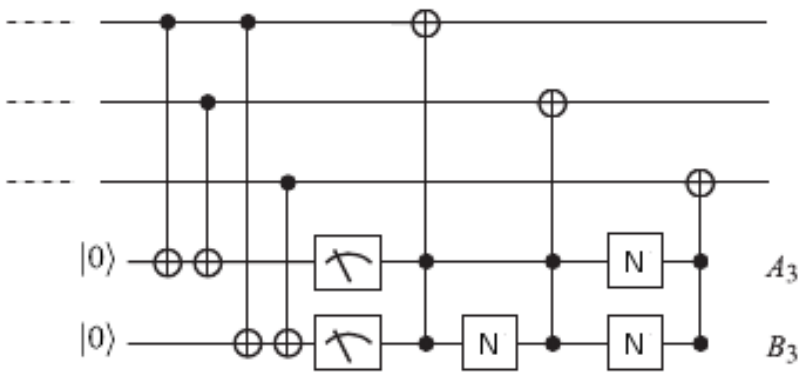

Abbildung 12: Schaltplan zur Simulation der Messung und Korrektur eines Bitfehlers des 9QECCs in Matlab. Nach der Messung wie sie Abb.2 für einzelne Bits, bzw. wie sie in Abb.5 in der ersten Sektion für alle logische QBits gezeigt wird, werden Toffoli Gatter angebracht, um das Fehlerbehaftete QBit zu korrigieren. Da das Auftreten eines bzw. keines Fehlers in den Hilfs-QBits eindeutig bestimmt ist, kann über NOT Gatter die Hilfs-QBits derart umgeändert werden, dass die Toffoli Gatter das richtige QBit flippen.

Die gleiche Änderung betraf auch die Messung des Phasenfehlers und es wurde zusätzlich zur Steigerung der Performance des Matlab Programms die Vertauschung des kontrollierendem und des Ziel-QBits, wie sie in Abb.8 gezeigt wird durchgeführt (siehe Abb.13 zum Vergleich zu Abb.5).

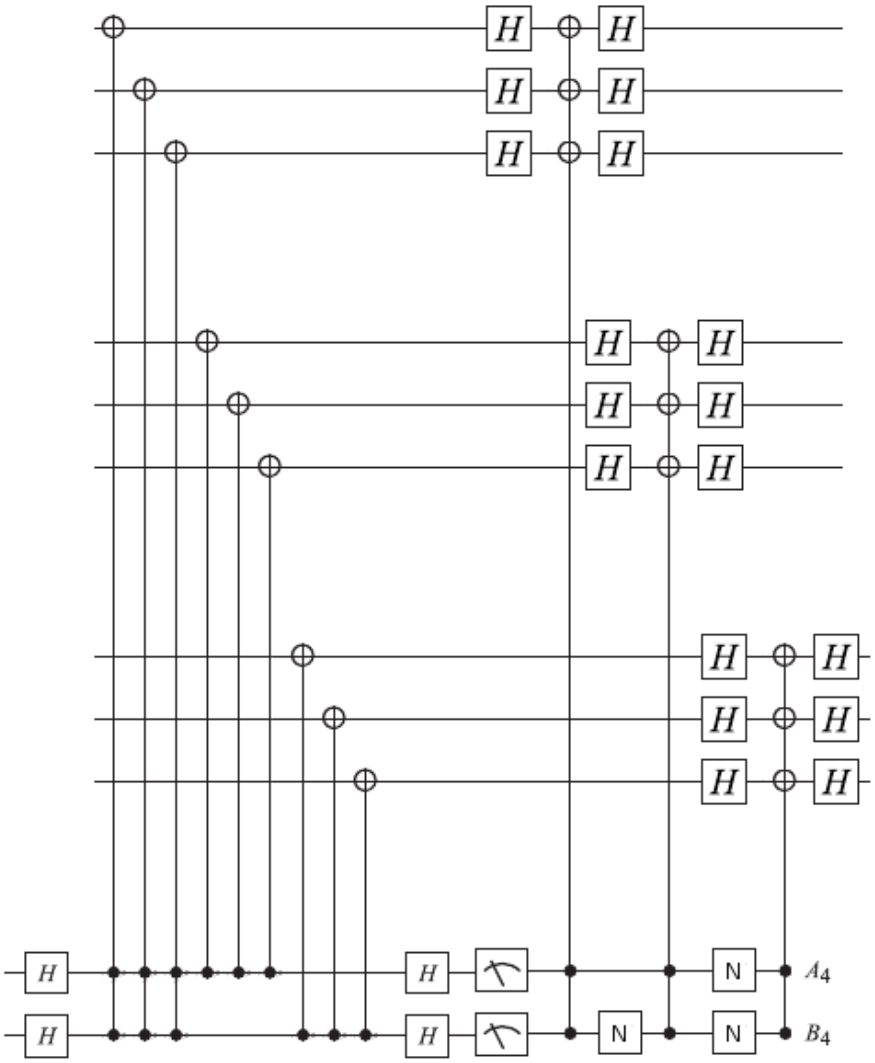

Abbildung 13: Schaltplan zur Simulation der Messung und Korrektur eines Phasenfehlers des 9QECCs in Matlab. Hierbei wurde die ursprüngliche Schaltung wie sie im zweiten Segment der Abb.5 gezeigt wird mit Hilfe der in Abb.8 gezeigtem Gesetz, das zu überprüfende QBit und das Ziel-QBit vertauscht, um die Performance der Matlabsimulation zu steigern. Nach der Messung wird wie es in Abb.12 gezeigt wird, das fehlerbehaftete logische QBit korrigiert.

7.2.2 7QECC

Beim 7 QECC wurden der Aufbau für die Kodierung und Dekodierung ebenfalls beibehalten (siehe Abb.7 und 10). Die Messung musste verändert werden. Dabei wurde wieder aus Gründen der Performancesteigerung die Vertauschung des kontrollierendem und des Ziel-QBits, wie sie in Abb.8 gezeigt wurde durchgeführt und nach der Messung Toffoli und NOT Gattern eingebaut, um den jeweiligen Fehler zu korrigieren (siehe Abb.14 zum Vergleich zu Abb.9).

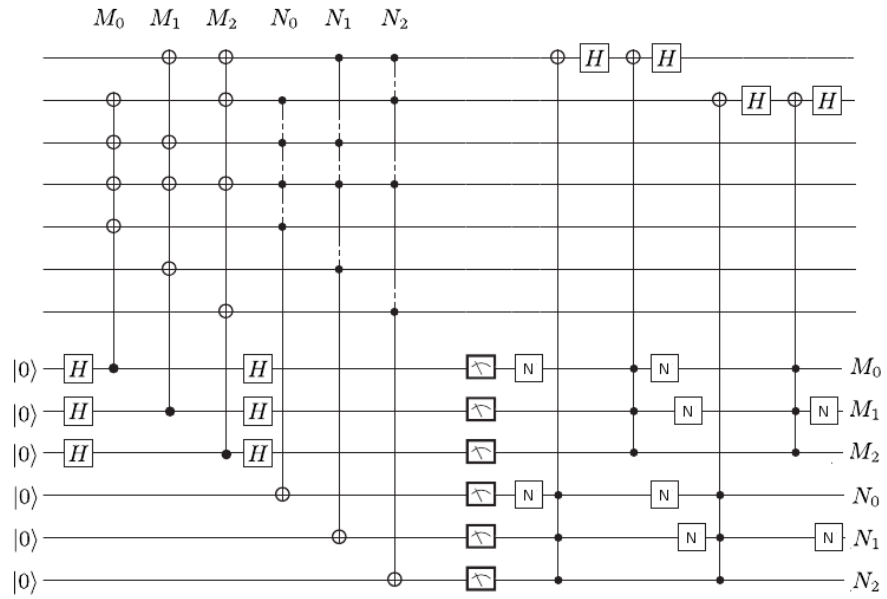

Abbildung 14: Schaltplan zur Simulation der Messung und Korrektur eines Bit- und Phasenfehlers des 7QECCs in Matlab. Die Ursprüngliche Schaltung, wie sie in Abb.9 gezeigt wird, wurde mit der in Abb.8 gezeigten Gesetzmäßigkeit umgeändert, um die Performance der Matlabsimulation zu steigern. Nach der Messung wird wie es in Abb.12 gezeigt wird, das fehlerbehaftete logische QBit korrigiert. Bemerkung: Es sind nicht alle Korrekturen aufgezeichnet. Die möglichen Korrekturen der restlichen kodierten QBits wurden wie sie in dieser Abbildung für das 1. und 2. dargestellt werden und unter Anpassung der NOT- und Toffoli Gatter an die Tabelle 5 aufgebaut

8 Anhang- Verzeichnisse

Literatur

[1] Quantum Computing - From Linear Algebra to Physical Realizations von Mikio Nakahara und Tetsuo Ohmi

[2] Quantencomputer - Eine Einführung von Univ.-Prof. Dr.rer.nat. Arrigoni, Enrico

[3] Vorlesungsskript über Quantum Computation von John Preskill am Caltech `http://theory.caltech.edu/people/preskill/ph229/`

[4] Quantum Computing - Andrew Steane
`http://xxx.lanl.gov/PS_cache/quant-ph/pdf/9708/9708022v2.pdf`

[5] Simulation eines Quantencomputers - Björn Butscher und Hendrik Weimer
`http://www.libquantum.de/files/libquantum.pdf`

[6] Analysis 3 (Integraltransformationen, Hilberträume, partielle Differentialgleichungen) von Ao.Univ.-Prof. Dr. H. Wallner

[7] Eine Auflistung der Quantengatter
`http://de.wikipedia.org/wiki/Liste_der_Quantengatter`