Douglas Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.): INFORMATIK 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

Möglichkeiten der Modellierung von Fehlern in MLC-Flash-Speichern durch Fehlergraphen

Günther Nieß[1] Thomas Kern[2] Michael Gössel[1]

Abstract: In dieser Arbeit werden beliebige kombinatorische Fehler mit Hilfe eines Fehlergraphen modelliert. Die Knoten des Fehlergraphen sind n-Tupel mit q-ären Komponenten. Zwei Knoten v und v' sind durch eine Kante von v nach v' verbunden, wenn das Wort v durch einen Fehler in das Wort v' verfälscht werden kann. Sollten die Fehler mit bekannten Wahrscheinlichkeiten oder Häufigkeiten auftreten, kann dies durch gewichtete Kanten der Fehlergraphen modelliert werden. Es wird gezeigt, wie ein optimaler fehlererkennender Code mit $m \geq 1$ Prüfbits und $k = n - m$ Datenbits mit einem Fehlergraphen bestimmt werden kann. Für kleine n und m können optimale Lösungen berechnet werden, für größere Blocklängen wird eine effiziente Heuristik vorgeschlagen. Als Beispiel wurden Retention- und Program-Interference-Fehler in Multi-Level NAND-Flash-Speicherzellen modelliert, für die mit Hilfe der vorgestellten Heuristik fehlererkennende Codes bestimmt wurden. Die neuen Codes wurden mit fehlererkennenden Codes für unidirektionale Fehler mit einer beschränkten Auslenkung und mit linearen Blockcodes mit Hilfe experimenteller Ergebnisse verglichen.

Keywords: Fehlermodellierung, Fehlermodell, Fehlererkennung, Flash, MLC, Code

1 Einführung

In dieser Arbeit wird dargestellt, wie $m \geq 1$ optimale Prüfbits eines fehlererkennenden Codes für ein beliebig vorgegebenes Fehlermodell bestimmt werden können. Fehler werden durch einen Fehlergraphen modelliert, in dem die Knoten aus q-ären Worten bestehen. Zwei Knoten v_1 und v_2 sind in dem Fehlergraphen durch eine Kante von v_1 nach v_2 verbunden, wenn ein Fehler des Fehlermodells das Wort v_1 in das Wort v_2 verfälscht. Dieses auf Graphen basierte Fehlermodell erlaubt es, alle kombinatorischen Fehler, die durch unterschiedliche Fehlerursachen ausgelöst werden können, auszudrücken. Fehlerauftrittswahrscheinlichkeiten können den entsprechenden Kanten als Kantengewichte zugeordnet werden.

Für ein gegebenes Fehlermodell werden $m \geq 1$ Prüfbits eines fehlererkennenden Codes mit Hilfe eines Graphen-basierten Algorithmus bestimmt, der durch Entfernen von Knoten die Kantengewichte bzw. die Anzahl der verbleibenden Kanten minimiert. Da die Kanten unterschiedliche Fehler des Fehlermodells entsprechen, ist die Fehlerauftrittswahrscheinlichkeit der verbleibenden nicht erkennbaren Fehler nach dem Bestimmen der optimalen Prüfbits minimal. Es wird weiterhin beschrieben, wie für einen bereits gegebenen fehlererkennenden Codes zusätzliche Prüfbits bestimmt werden können.

[1] Universität Potsdam, AG Fehlertolerantes Rechnen, August-Bebel-Str. 89, 14482 Potsdam, {niess,mgoessel}@cs.uni-potsdam.de
[2] Infineon Technologies AG, Am Campeon 1-12, 85579 Neubiberg, thomas.kern@infineon.com

Die Arbeit ist wie folgt gegliedert. In Abschnitt 2 werden Fehlergraphen eingeführt und es wird dargestellt wie verschiedene Arten von Fehlern ausgedrückt werden können. Abschnitt 3 stellt dar wie Fehlergraphen genutzt werden um fehlererkennende Codes oder m zusätzliche Prüfbits für einen bereits gegebenen Code zu bestimmen. Um auch größere Blocklängen zu bewältigen wird eine Heuristik eingeführt, die eine m-stellige Boolesche Prüfbitfunktion bestimmt. Als mögliche Anwendung der vorgestellten Methode werden in Abschnitt 4 für multi-level Flash-Speicherzellen und Retention-Fehler neu bestimmte fehlererkennende Codes mit bereits bekannten Codes verglichen. Abschnitt 5 schließt die Arbeit mit einer Zusammenfassung ab.

2 Fehlergraph

In diesem Abschnitt wird beschrieben, wie Fehler in Fehlergraphen modelliert werden können. Die Knoten des Fehlergraphen sind n-äre Wörter. Wenn es einen Fehler gibt, der das Wort v in das Wort v' verfälscht, dann wird der Knoten v mit einer gerichteten Kante von v nach v' verbunden. Tritt ein Fehler mit einer Wahrscheinlichkeit p auf, so hat die Kante von v nach v' das Gewicht p. Unter der Annahme, dass es für alle Fehler e, die v in v' verändern auch Fehler gibt, die v' in v verfälschen, ist der Fehlergraph ungerichtet.

Viele Codes wurden zur Korrektur und Erkennung von asymmetrische und unidirektionalen Fehler mit einer beschränkten Auslenkung veröffentlicht [Ca10], [KBE11], [MK12], [EBE13]. Die häufigste Anwendung für diese Codes ist die Fehlerkorrektur und Fehlererkennung in multi-level Flash-Speicherzellen [KBE11], [MK12]. Derartige Speicherzellen haben mehrere Schwellenwerte für Spannungen und somit verschieden programmierbare Speicherzustände. Ein unidirektionaler Fehler mit einer beschränkten Auslenkung verändert den programmierten Wert einer Speicherzelle um einen bestimmten absoluten Betrag in eine Richtung, zunehmend oder abnehmend. Bei asymmetrischen Fehlern mit einer beschränkten Auslenkung werden hingegen nur Fehler betrachtet, bei denen im Voraus schon bekannt ist, ob die Fehler den Wert der Speicherzellen bis zu einem absoluten Betrag erhöhen oder absenken.

In Abb. 1 wird ein Fehlergraph für 2 multi-level Speicherzellen und asymmetrischen Fehlern der maximalen Auslenkung $l = 1$ gezeigt. Dabei werden in den Knoten die möglichen Speicherzustände abgebildet. Jede Speicherzelle kann mit 2 Bits programmiert werden und dadurch die Werte 0 (00), 1 (01), 2 (10) und 3 (11) abspeichern. Der Speicherzustand wird durch die Anzahl der Elektronen in der Speicherschicht definiert. Es gibt hauptsächlich zwei Methoden, die benutzt werden um die programmierte Ladung in den multi-level Speicherzellen in binäre Werte abzubilden. Sie sind in Tab. 1 dargestellt. Die erste Methode

Tab. 1: Multi-level Flash-Speicherzellen und deren Speicherzustände

Gespeicherte Elektronen	kleinste Anzahl			größte Anzahl
Binäres Encoding	11	10	01	00
Gray Encoding	10	11	01	00

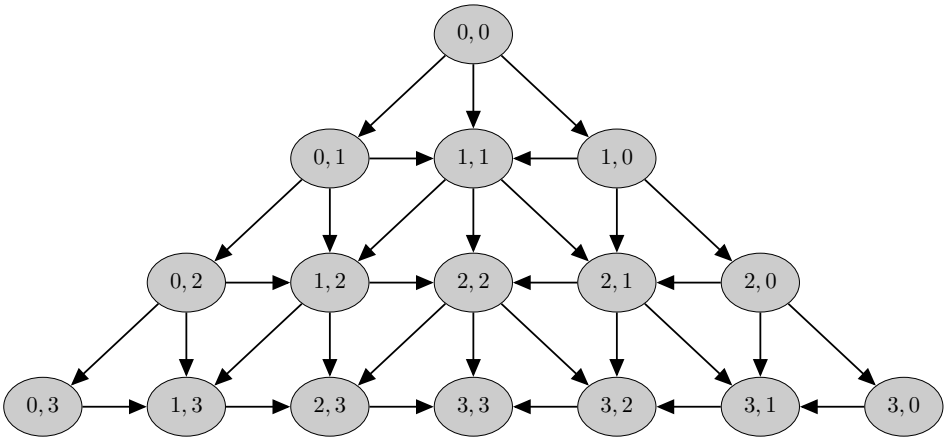

Abb. 1: Fehlergraph für zwei multi-level Speicherzellen mit asymmetrischen Fehlern mit einer beschränkten Auslenkung $l = 1$

benutzt den Gray-Code. Beim Gray-Code unterscheiden sich zwei aufeinander folgende Speicherzustände einer Speicherzelle in nur einem Bit. Wenn man diese Methode benutzt, stellt man sicher, dass eine kleine Veränderung in der Ladungsmenge einer Speicherzelle sich nur in einer kleinen Anzahl von Bit-Fehlern im abgespeicherten Wert auswirkt. Die zweite Methode, das sogenannte binäre Encoding aus Tab. 1, zählt die Schwellwerte einer multi-level Speicherzelle beginnend mit der höchsten Anzahl an gespeicherten Elektronen als 0. Verliert eine Speicherzelle über einen längeren Zeitraum eine geringe Anzahl an Elektronen so verändert sich bei dieser Methode der abgespeicherte Wert ebenso nur um einen kleinen Betrag. Sind zum Beispiel zwei multi-level Speicherzellen mit dem Wert $(0,0)$ programmiert, so halten sie die maximale Anzahl an Elektronen in den Speicherzellen. Tritt ein Fehler mit einer beschränkten Auslenkung $l = 1$ auf, so können die Speicherzellen in die Werte $(0,1)$, $(1,0)$ oder $(1,1)$ verfälscht werden. Dabei verliert im ersten Fall, bei dem der Speicherzustand $(0,0)$ in den Zustand $(0,1)$ verfälscht wird, die zweite Speicherzelle gespeicherte Elektronen, und der Fehler wird im Fehlergraphen aus Abb. 1 durch die Kante abgebildet, die den Knoten $(0,0)$ mit $(0,1)$ verbindet. Werden nun alle asymmetrischen Fehler mit einer Auslenkung $l = 1$ betrachtet, wenn die Speicherzellen mit den Werten $(0,0)$ programmiert wurden, so sind aus dem Fehlergraphen aus Abb. 1 und den drei Kanten, die vom Knoten $(0,0)$ ausgehen, die drei möglichen Fehlerübergänge ersichtlich.

Werden anstatt asymmetrischen Fehlern nun unidirektionale Fehler betrachtet, so werden aus den gerichteten Kanten aus Abb. 1 ungerichtete Kanten. Da für jeden Fehler, der einen Speicherzustand um die Auslenkung $l = 1$ erhöht, wir nun auch den Fehler betrachten, der den Speicherzustand um die Auslenkung $l = 1$ absenkt. Erhöhen wir die Auslenkung auf $l = 2$, kommen zu den Kanten aus Abb. 1 noch zusätzlich die Kanten aus Abb. 2 hinzu. Das jetzt betrachtete Fehlermodell beschreibt ein fehlerhaftes auf- oder entladen von Flash-Speicherzellen um bis zu zwei Stufen. Die schlägt sich in dem grafisch dargestellten Fehlergraphen so nieder, dass nicht nur bestimmte Nachbarknoten durch Fehlerkanten verbunden sind, wie in Abb. 1, sondern dass die Kanten über einen Knoten hinweg zwei

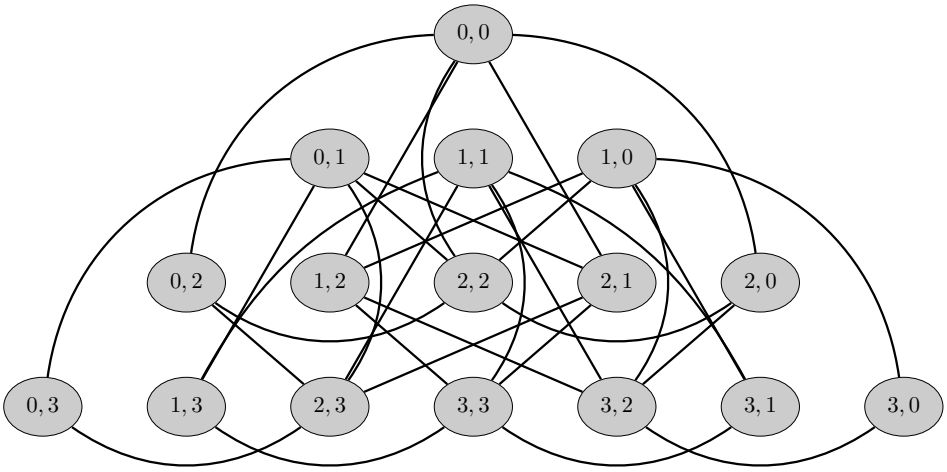

Abb. 2: Fehlergraph mit zusätzlichen Kanten für unidirektionale Fehler mit einer beschränkten Auslenkung $l = 2$

Speicherzustände miteinander verbinden, wie in Abb. 2. Dabei modellieren Kanten, die von oben nach unten, bzw. von außen nach innen verlaufen einen Ladungsverlust in den Speicherzellen und Kanten die von unten nach oben, bzw. von innen nach außen verlaufen eine Ladungszunahme.

Eine ausführliche Analyse von einem multi-level NAND-Flash-Speicherchip machte deutlich, dass Retention-Fehler von den programmierten Speicherzuständen der Flash-Zellen abhängen [Ca12]. Retention-Fehler sind Fehler, bei denen sich der abgespeicherte Wert einer Flash-Zelle über die Zeit verändert. Normalerweise geschieht dies, wenn die programmierte Ladung einer Speicherzelle mit der Zeit über Leckströme entweicht. Eine andere Fehlerart sind die Program-Interference-Fehler. Sie sind nach den Retention-Fehlern die am häufigsten vorkommende Fehlerart [Ca12]. Program-Interference-Fehler treten auf, wenn der Speicherzustand einer Zelle sich verändert, während eine Nachbarzelle beschrieben wird. Dies geschieht durch kapazitive Kopplungseffekte der Nachbarzellen beim Programmieren auf die Speicherzelle, die den Schwellwert verändern. Durch die kleiner werdenden Strukturgrößen und geringeren Abständen zwischen den Zellen wird dieser Effekt zunehmend dominant.

Für die in [Ca12] betrachteten Chips beträgt die Wahrscheinlichkeit von Retention-Fehlern mit ausgeschalteter Fehlerkorrektureinheit (ECC) nach 3 Jahren Betrieb und nach 3.000 Programmier- und Löschzyklen ungefähr $q \approx 10^{-4}$ und die häufigsten Retention-Fehler sind $00 \to 01$, $01 \to 10$, $01 \to 11$ und $10 \to 11$ mit einer relativen Verteilung aller Retention-Fehler von 46%, 44%, 5% und 2%. Die Program-Interference-Fehler besitzen nach demselben Betrieb die Auftrittswahrscheinlichkeit von ungefähr $q \approx 3,5 \cdot 10^{-7}$ und die häufigsten Program-Interference-Fehler sind $11 \to 10$, $10 \to 01$, $10 \to 00$, $11 \to 01$ und $01 \to 00$ mit einer relativen Verteilung aller Program-Interference-Fehler von 70%, 24%, 2,2%, 1,5% und 0,4%. Wenn wir annehmen, dass die Fehlerrate von Flash-Zellen statistisch unabhängig ist und die programmierten Speicherzustände gleichverteilt sind, können wir die

Wahrscheinlichkeiten der Fehlerübergänge von Speicherzellen berechnen. Die verbleibenden drei Prozent der Retention-Fehler, bzw. zwei Prozent der Program-Interference-Fehler wurden nicht genauer in der Arbeit [Ca12] spezifiziert und deshalb hier nicht weiter modelliert.

Abb. 3 zeigt einen Fehlergraphen für 2 Flash-Speicherzellen der mit modellierten Retention- und Program-Interference-Fehlern aus [Ca12] bestimmt wurde. Die Fehlerverteilung wird durch gewichtete Kanten modelliert, die aufgrund der Übersichtlichkeit durch die Linienbreite der Kanten dargestellt wurde, wobei sie der Fehlerauftrittswahrscheinlichkeit des dargestellten Fehlers entspricht. Der Graph stellt ausschließlich Retention- oder Program-Interference-Fehler dar, eine Kombination der Fehler, also dass beispielsweise in der ersten Speicherzelle ein Retention-Fehler und in der zweiten Speicherzelle ein Program-Interference-Fehler aufgetreten ist wird in der Abb. 3 nicht dargestellt. Somit sind alle dargestellten Fehler unidirektional und der Graph ist ein Teilgraph des Fehlergraphen für unidirektionale Fehler mit einer maximalen Auslenkung von $l = 2$. Ein Code der alle unidirektionalen Fehler mit einer maximalen Auslenkung von $l = 2$ erkennt oder korrigiert, kann somit auch alle in Abb. 3 modellierten Fehler erkennen oder korrigieren. Vergleichen wir die Gewichte der Kanten, also die Fehlerauftrittswahrscheinlichkeiten, kann man erkennen, dass einige Kanten mit einer Auslenkung von $l = 2$, z.B. zwischen den Knoten $(01,01)$ und $(01,11)$ ein größeres Gewicht besitzen, als manche Kanten mit einer Auslenkung von $l = 1$, wie z.B. der Fehler zwischen den Knoten $(10,10)$ und $(11,11)$.

Betrachten wir nun Fehler, die beide Speicherzellen betreffen und wobei eine Speicherzelle einen Retention-Fehler und die Andere einen Program-Interference-Fehler aufweist, erhalten wir den Fehlergraph aus Abb. 4. Keine Kante des Graphen aus Abb. 4, die die modellierten Fehler darstellen, ist in dem Fehlergraphen für unidirektionale Fehler mit einer maximalen Auslenkung von $l = 2$ enthalten. Somit werden die dargestellten Fehler von einem Code, der nur unidirektionale Fehler mit einer maximalen Auslenkung von $l = 2$ erkennt oder korrigiert, nicht behandelt und somit nicht erkannt oder eventuell falsch korrigiert. Die modellierten Fehlerauftrittswahrscheinlichkeiten liegen zwischen den Größenordnungen $q \approx 10^{-11}$ und $q \approx 10^{-15}$. Sie unterliegen somit nicht einer so breiten Streuung wie die modellierten Fehler aus Abb. 3. Wie weitere Fehlerarten, wie z.B. Bit-Flips oder Stuck-At-Fehler, durch Fehlergraphen modelliert werden können, wird in [NKG15] beschrieben.

3 Bestimmung von fehlererkennenden Prüfbits

In diesem Abschnitt beschreiben wir, wie zusätzliche Prüfbits für einen fehlererkennenden Blockcode bestimmt werden können. Wir betrachten zwei Fälle.

1. Im ersten Fall werden m optimale Prüfbits c_1, c_2, …, c_m für k Datenbits u_1, u_2, …, u_k bestimmt. Dabei nehmen wir an, dass wir 2^k unterschiedliche k-Tupel codieren können. Wenn die Prüfbits c_1, c_2, …, c_m durch k-äre Boolesche Funktionen f_1, f_2, …, f_m aus den Datenbits u_1, u_2, …, u_k durch $c_1 = f_1(u_1,u_2,\ldots,u_k)$, $c_2 = f_2(u_1,u_2,\ldots,u_k)$, …, $c_m = f_m(u_1,u_2,\ldots,u_k)$ bestimmt sind, bilden die Daten-

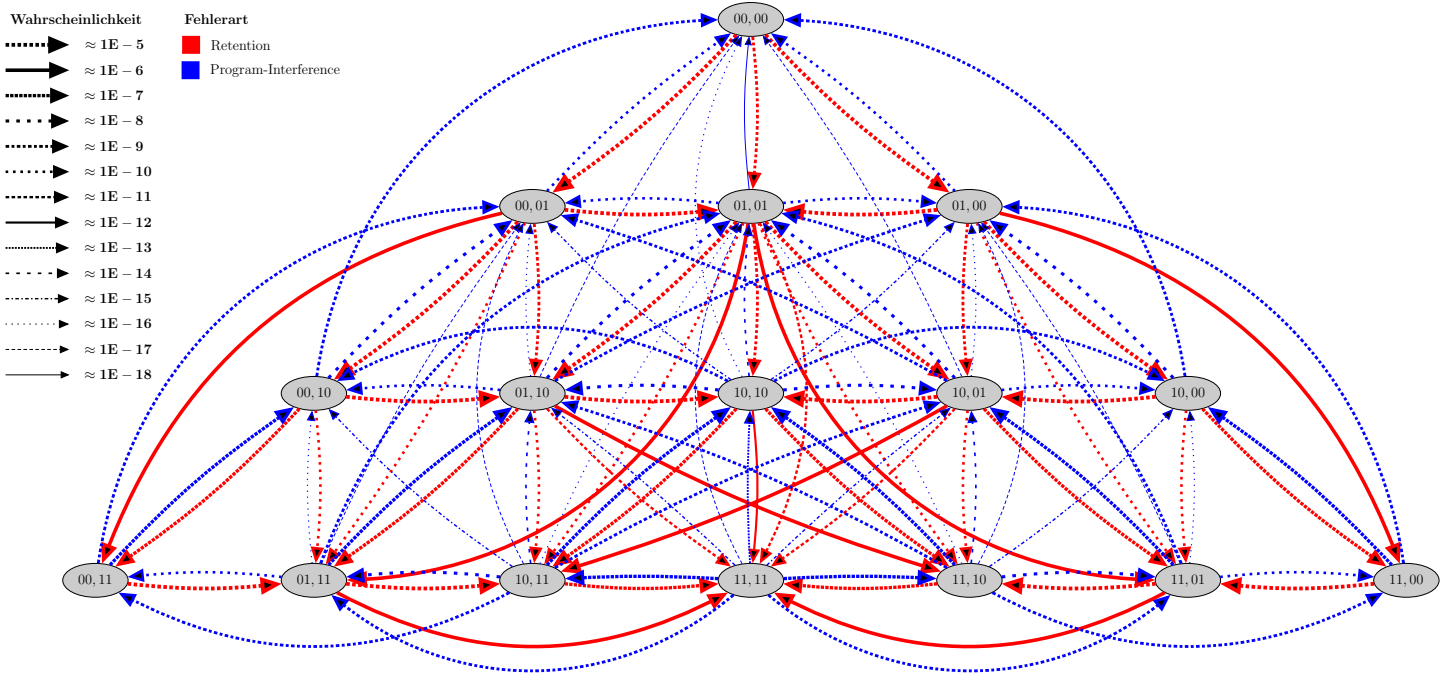

Abb. 3: Fehlergraph für multi-level Flash-Speicher mit unidirektionalen Retention- und Program-Interference-Fehler

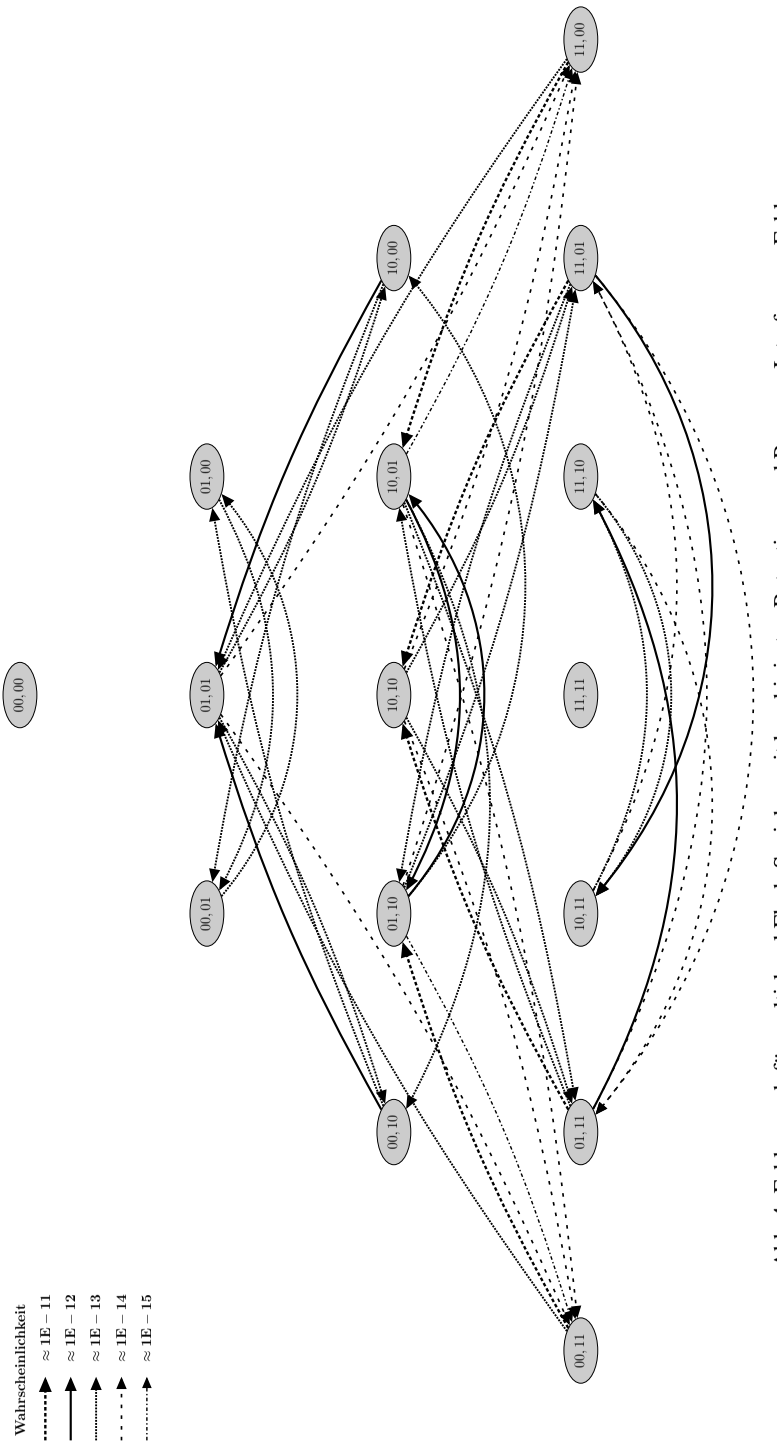

Abb. 4: Fehlergraph für multi-level Flash-Speicher mit kombinierten Retention- und Program-Interference-Fehler

bits mit den entsprechenden Prüfbits $u_1, u_2, \ldots, u_k, c_1, c_2, \ldots, c_m$ ein Codewort des Codes C. Dabei ist der Code C optimal, wenn die Anzahl nicht erkennbarer Fehler oder die Summe der Fehlerauftrittswahrscheinlichkeiten von nicht erkennbaren Fehlern minimal ist.

2. Im zweiten Fall nehmen wir an, dass die Bits $(v_1, v_2, \ldots, v_{n'})$ durch zusätzliche Prüfbits gegenüber Fehlern geschützt werden sollen. Dazu nehmen wir weiterhin an, dass die Bits $(v_1, v_2, \ldots, v_{n'})$ bereits Codewörter eines gegebenen fehlererkennenden Codes C_{out} sind. Durch Hinzufügen von m optimalen zusätzlichen Prüfbits $c_1, c_2, \ldots, c_m$ sollen die Fehlererkennungseigenschaften des Codes C_{out} verbessert werden. Die zusätzlichen Prüfbits werden durch die Funktionen $c_1 = g_1(v_1, v_2, \ldots, v_{n'})$, $c_2 = g_2(v_1, v_2, \ldots, v_{n'})$, $\ldots$, $c_m = g_m(v_1, v_2, \ldots, v_{n'})$ für $(v_1, v_2, \ldots, v_{n'}) \in C_{out}$ bestimmt. Dabei kann der Code C_{out} als äußerer Code betrachtet werden [Fo66], [Bo98]. Sind die Prüfbits $c_1, c_2, \ldots, c_m$ durch Boolesche Funktionen bestimmt, so bilden die Bits $(v_1, v_2, \ldots, v_{n'}, c_1, c_2, \ldots, c_m)$ ein Codewort des neu zu bestimmenden inneren Codes C_{in}.

Betrachten wir zunächst den ersten Fall und nehmen an, dass ein initialer Fehlergraph mit allen 2^n möglichen n-ären binären Knoten gegeben ist, in dem Fehler sowohl in den Daten als auch in den zu bestimmenden Prüfbits modelliert sind.

Für ein gegebenes Datenwort $u = (u_1, u_2, \ldots, u_k)$ bezeichnen wir die Menge aller Knoten mit den gleichen Datenbits u und verschiedenen Prüfbits als Equal-Data-Nodes

$$ED(u_1, u_2, \ldots, u_k) := \{(v_1, v_2, \ldots, v_{k+m}) \in V \mid u_1 = v_1, u_2 = v_2, \ldots, u_k = v_k\}$$

wobei V der Knotenmenge des Fehlergraphen entspricht. Für einen initialen Fehlergraphen besteht die Menge $ED(u)$ aus 2^m unterschiedlichen Wörtern mit 2^m unterschiedlichen Werten für die mögliche Prüfbitbelegung.

Für jeden Knoten $v = (u, c) = (u_1, u_2, \ldots, u_k, c_1, c_2, \ldots, c_m)$ eines Fehlergraphen ist die Menge der komplementären Knoten $CV(v)$ definiert durch

$$CV(u, c) := ED(u) \setminus \{(u, c)\}.$$

Für initiale Fehlergraphen besteht die Menge von komplementären Knoten eines Knotens v aus allen $2^m - 1$ Knoten, die dieselben Datenbits haben wie der Knoten v aber unterschiedliche Prüfbits aufweisen.

Um einen fehlererkennenden Code C zu bestimmen, wird für jedes Datenwort u ein Knoten $v \in ED(u)$ als Codewort $v \in C$ ausgewählt und alle komplementären Knoten $v' \in CV(v)$ werden aus dem Fehlergraphen entfernt. Wenn zum Beispiel der Knoten $(u_1, u_2, \ldots, u_k, 1, 0, \ldots, 0)$ aus dem Fehlergraphen als Codewort ausgewählt wird, erhalten wir die Prüfbitfunktion $1 = f_1(u_1, u_2, \ldots, u_k)$, $0 = f_2(u_1, u_2, \ldots, u_k)$, $\ldots$, $0 = f_m(u_1, u_2, \ldots, u_k)$. Um die Booleschen Funktionen $f_1, f_2, \ldots, f_m$ vollständig zu bestimmen muss für jedes Datenwort u ein Knoten als Codewort ausgewählt werden.

Dies kann auf $(2^m)^{2^k}$ verschiedene Arten erfolgen. Für kleine k und m, $m \cdot 2^k \leq 64$, ist es möglich, alle Fälle zu betrachten und dann den besten oder optimalen Code zu bestimmen.

Für größere k und m wurde eine Heuristik entwickelt. In jedem Schritt des Algorithmus wird mit Hilfe einer Bewertungsfunktion ein Knoten als Codewort ausgewählt und $2^m - 1$ Knoten werden aus dem Fehlergraphen entfernt.

Die Heuristik weist jedem Knoten $v = (u,c) \in V$ ein Gewicht $weight(v)$ zu, welches aus der Summe der Kantengewichte eines Knotens v besteht, die zu dem Knoten v des aktuellen Fehlergraphen führen oder vom Knoten v ausgehen und nicht Knoten der Menge $ED(u)$ enthalten. Ist der Graph nicht gewichtet, so wird jede Kante mit dem Gewicht 1 gezählt. Wird ein Knoten $v = (u,c) \in ED(u)$ als Codewort des Codes C ausgewählt, so verbleiben die Kanten, die mit v und Knoten außerhalb von $ED(u)$ verbunden sind unverändert. Aber alle komplementären Knoten $v' \in CV(u)$ werden zusammen mit ihren Kanten aus dem aktuellen Fehlergraphen entfernt. Um den am besten geeigneten Knoten als Codewort auszuwählen, führen wir den Rang eines Knotens ein

$$rank(v) := \left(\sum_{v' \in CV(v)} weight(v') \right) - weight(v).$$

Die Summe über alle komplementären Knoten $v' \in CV(v)$ beschreibt die Reduzierung der Gewichte, die durch das Entfernen aller komplementären Knoten $CV(v)$ entsteht und der Rang $rank(v)$ berücksichtigt dabei noch die durch die Auswahl verbleibenden Kanten im veränderten Fehlergraphen. Stellen die Kanten die Fehlerauftrittswahrscheinlichkeiten dar, so beschreibt der Rang *rank* eines Knotens v die durch Auswahl des Knotens v zusätzlich gewonnene Fehlererkennungswahrscheinlichkeit minus der Wahrscheinlichkeit das zusätzliche Fehler nicht erkannt werden.

Die Heuristik wählt in jedem Schritt einen Knoten v mit einem maximalen Rang $rank(v)$ als Codewort aus und entfernt seine komplementären Knoten $CV(v)$. In jedem Schritt wird ein neues Codewort des Code C bestimmt, so dass nach 2^k Schritten der Code C vollständig ermittelt ist.

Zum Beispiel sind in Abb. 3 für das Datenwort 01 die Equal-Data-Nodes $ED(01) = \{(01,00), (01,01), (01,10), (01,11)\}$. Die komplementären Knoten des Knoten $(01,01)$ sind $CV(01,01) = \{(01,00), (01,10), (01,11)\}$. Das Gewicht der Retention-Fehler, die den Knoten $(01,01)$ betreffen wird berechnet durch

$$\begin{aligned} weight(01,01) &= 2,12E-9+1,25E-6+1,1E-5+2,2E-10+1,94E-9 \\ &\quad +2,5E-11+2,2E-10+1,15E-5 \\ &= 2,3754525E-5 \end{aligned}$$

und der Rang des Knotens ist

$$\begin{aligned} rank(01,01) &= \left(\sum_{v \in \{(01,00),(01,10),(01,11)\}} weight(v) \right) - weight(01,01) \\ &= 2,375225E-5+2,3752118E-5+2,3750322E-5-2,3754525E-5 \\ &= 4,7500165E-5. \end{aligned}$$

Alle Ränge, die in unserem Beispiel durch die Retention-Fehler des Fehlergraphen aus Abb. 3 betrachten, sind in Tab. 2 aufgeführt.

Tab. 2: Knotenränge der Retention-Fehler des initialen Fehlergraphen aus Abb. 3

Knoten v	$(00,00)$	$(00,01)$	$(00,10)$	$(00,11)$
$rank(v)$	$2,30002E-5$	$2,30000E-5$	$2,30043E-5$	$2,30045E-5$
Knoten v	$(01,00)$	$(01,01)$	$(01,10)$	$(01,11)$
$rank(v)$	$4,75047E-5$	$4,75002E-5$	$4,75050E-5$	$4,75086E-5$
Knoten v	$(10,00)$	$(10,01)$	$(10,10)$	$(10,11)$
$rank(v)$	$2,30043E-5$	$2,30002E-5$	$2,30006E-5$	$2,30038E-5$
Knoten v	$(11,00)$	$(11,01)$	$(11,10)$	$(11,11)$
$rank(v)$	$3,50068E-6$	$3,50004E-6$	$3,50006E-6$	$3,50058E-6$

Entsprechend der Heuristik hat nach Tab. 2 der Knoten $v = (01,11)$ den maximalen Rang und wird demzufolge als Codewort ausgewählt. Seine komplementären Knoten $CV(01,11) = \{(01,00), (01,01), (01,10)\}$ wie auch alle Kanten die mit einem komplementären Knoten $CV(01,11)$ verbunden sind, werden aus dem Fehlergraphen entfernt. Als Ergebnis entsteht

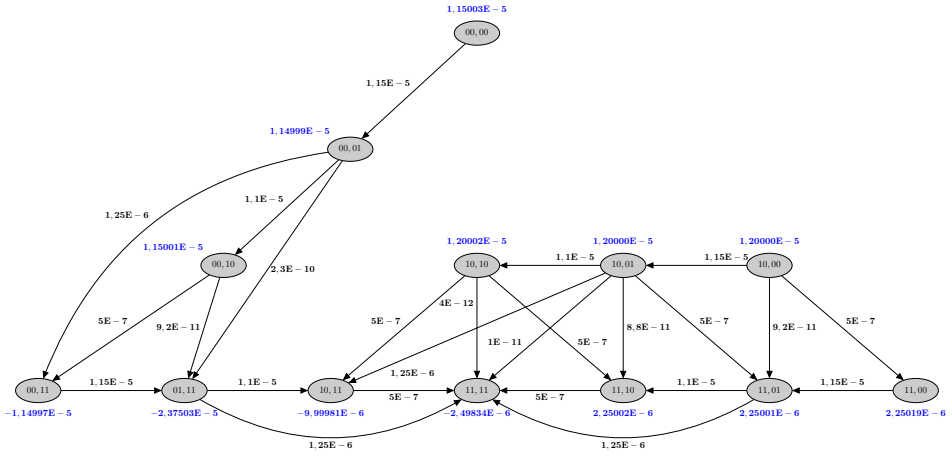

Abb. 5: Gewichteter Fehlergraph nachdem der Knoten $(01,11)$ als Codewort ausgewählt wurde

der in Abb. 5 dargestellte Fehlergraph, bei dem die neu berechneten Ränge über und unter den Knoten abgebildet sind. Nun hat der Knoten $v = (10,10)$ den größten Rang und wird deshalb als Codewort gewählt. Anschließend entfernt der Algorithmus alle Kanten die mit den komplementären Knoten $CV(10,10) = \{(10,00), (10,01), (10,11)\}$ verbunden sind und die komplementären Knoten $CV(10,10)$ selbst aus dem aktuellen Fehlergraphen. In dem resultierenden Fehlergraphen hat der Knoten $(00,00)$ den maximalen Rang $rank(00,00) = 1,15003E-5$ und wird als Codewort gewählt. Nachdem alle komplementären Knoten $CV(00,00) = \{(00,01), (00,10), (00,11)\}$ und die mit den Knoten verbundenen Kanten entfernt wurden, ergibt sich der in Abb. 6 dargestellte Fehlergraph. Der Fehlergraph aus Abb. 6 führt dazu, dass die Knoten $(11,01)$ und $(11,00)$ beide

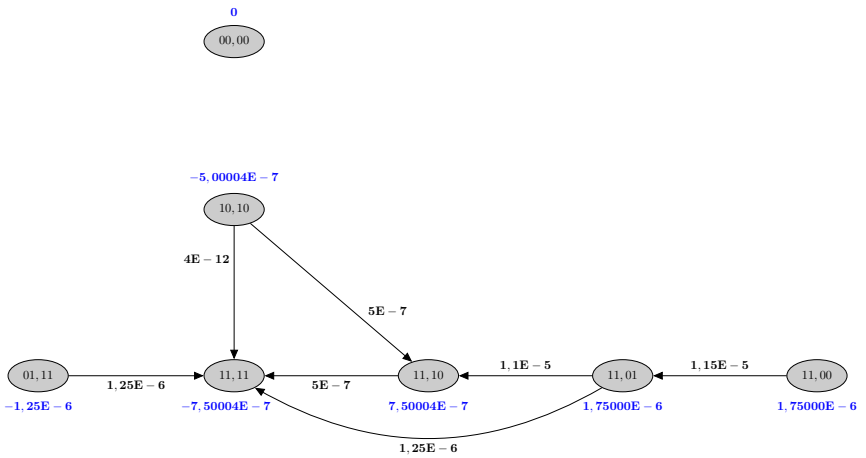

Abb. 6: Gewichteter Fehlergraph nachdem der Knoten $(00,00)$ als Codewort ausgewählt wurde

einen maximalen Rang von $1,75000E-6$ besitzen und wir einen der Knoten als Codewort wählen können. Nachdem ein Knoten als Codewort ausgewählt wurde und seine komplementären Knoten und die mit deren Kanten vom Fehlergraphen entfernt wurden, erhalten wir einen fehlererkennenden Code z.B. $C = \{(00,00), (01,11), (10,10), (11,00)\}$ bei dem die Prüfbitfunktionen f_1 und f_2 vollständig bestimmt sind.

Der zweite Fall ist ähnlich, mit dem Unterschied, dass nur Fehler betrachtet werden, die ein Codewort des inneren Codes C_{in} in ein Codewort des inneren Codes C_{in} verfälschen. Fehler die durch den äußeren Code erkannt werden, können vom initialen Fehlergraphen durch Entfernen der Knoten $(v_1, v_2, \ldots, v_{n'}, c_1, c_2, \ldots, c_m)$ bei denen die ersten n' Bits nicht einem Codewort des äußeren Codes $v \notin C_{out}$ entsprechen nicht weiter betrachtet werden. Die Implementierung des Algorithmus wird in [NKG15] beschrieben.

4 Fehlererkennung für Multi-Level NAND-Flash-Speicher

In diesem Abschnitt werden Codes, die mit der vorgestellten Heuristik erzeugt wurden, mit bereits bekannten fehlererkennenden Codes verglichen. Als beispielhaftes Anwendungsgebiet betrachten wir multi-level NAND-Flash-Speicher mit Retention- und Program-Interference-Fehler die anhand von experimentellen Ergebnissen modelliert wurden [Ca12]. Der Fehlergraph für zwei Speicherzellen setzt sich aus den Graphen aus Abb. 3 und Abb. 4 zusammen. Das Fehlermodell wurde am Ende des Abschnitts 2 beschrieben.

Unidirektionale und asymmetrische fehlererkennende Codes für Speicherzellen, die mehr als $q > 2$ Werte speichern können, werden mit Hilfe des Restklassenrings $\mathbb{Z}/q\mathbb{Z}$ definiert. In praktischen Anwendungen werden hauptsächlich die Werte $q = 2^a$ mit $a \in \{2,3\}$ benutzt. Die Codes besitzen normalerweise ganzzahlige Blocklängen mit $K = k/a$ Datenzellen und $M = m/a$ redundante Prüfbitzellen. Es wird das binäre Encoding von Tab. 1 benutzt um die gespeicherten Elektronen von Flash-Speichern zu interpretieren. Es wird demzufolge ein binäres Codewort $v = (u_1, u_2, \ldots, u_k, c_1, c_2, \ldots, c_m)$ mit k Datenbits

und m Prüfbits mit dem Restklassenring $\mathbb{Z}/2^a\mathbb{Z}$ interpretiert, so dass wir Codewörter der Form $V = (U_1, U_2, \ldots, U_K, C_1, C_2, \ldots, C_M)$ mit $U_1, U_2, \ldots, U_K, C_1, C_2, \ldots, C_M \in \mathbb{Z}/2^a\mathbb{Z}$ betrachten.

Der optimale systematische fehlererkennende Code für unidirektionale Fehler in $\mathbb{Z}/q\mathbb{Z}$ benötigt $\lceil \log_q K(q-1)+1 \rceil$ Prüfsymbole und wird durch

$$c = \sum_{i=1}^{K} (q - 1 - U_i)$$

berechnet, wobei U_1, U_2, …, $U_K \in \mathbb{Z}/q\mathbb{Z}$ den abgespeicherten Werten der Datenzellen entsprechen und die Prüfsymbole $C_1, C_2, \ldots, C_M$ die q-äre Repräsentation von c darstellen [BP82]. In dem beschriebenen Fall von $q = 2^2$ wird für $K = 1$ Datenzelle eine redundante Speicherzelle für die Prüfbits benötigt. Für $K \leq 5$ Datenzellen werden zwei redundante Zellen benötigt um alle unidirektionalen Fehler erkennen zu können. Wenn wir für mehr als $K > 5$ Datenzellen $M = 2$ Prüfsymbole benutzen möchten, empfiehlt sich der Code mit den Prüfsymbolen

$$c = \left(\sum_{i=1}^{K} (q - 1 - U_i) \right) \mod q^2. \tag{1}$$

Der Code erkennt bis zu $t = \frac{q^2}{l} - q$ asymmetrische Fehler mit einer maximalen Auslenkung l [EBE13]. Wir möchten zunächst Codes mit einer redundanten Speicherzelle vergleichen. Diese Codes sollen die am häufigsten vorkommenden Retention- und Program-Interference-Fehler in Flash-Speichern erkennen. Demzufolge können wir zwei Prüfbits oder ein Prüfsymbol im Restklassenring $\mathbb{Z}/4\mathbb{Z}$ definieren. Den Konstruktionsverfahren in [BP82] und [EBE13] folgend, können wir ein Prüfsymbol durch

$$c = \left(\sum_{i=1}^{K} (3 - U_i) \right) \mod 4 \tag{2}$$

bestimmen und es zur Fehlererkennung nutzen.

Eine andere Methode um multi-level Flash-Speicher zu schützen ist, das Gray-Encoding aus Tab. 1 einzusetzen um die Menge der gespeicherten Elektronen in Speicherzellen zu interpretieren und mit einem linearen Blockcode die Fehlererkennung umzusetzen. Der lineare fehlererkennende Code sollte Fehler erkennen, die eine geringe Anzahl von Bit-Fehlern aufweisen. Mit $m = 2$ Prüfbits kann der Code mit

$$\begin{aligned} c_1 &= u_1 \oplus u_2 \oplus u_3 \oplus \ldots \oplus u_k \\ c_2 &= u_1 \oplus u_3 \oplus u_5 \oplus \ldots \oplus u_{k-1} \end{aligned} \tag{3}$$

für die Datenbits u_1, u_2, …, u_k bestimmt werden. Das Prüfbit c_1 entspricht hierbei der Gesamtparität und die XOR-Summe von c_2 enthält ein Bit aus jeder Flash-Zelle. Der Code kann somit alle Fehler erkennen, die nur eine Speicherzelle betreffen oder Fehler die mehrere Speicherzellen betreffen und eine ungerade Anzahl an Bits verfälschen.

Die zwei oben aufgeführten Verfahren werden in Tab. 3 mit den experimentellen Ergebnissen der in Abschnitt 3 vorgeschlagenen Heuristik verglichen. Dabei ist die Tab. 3 in drei

Tab. 3: Die Fehlererkennung von verschiedenen Codes für Multi-Level NAND-Flash-Speicher mit Retention- und Program-Interference-Fehlern

Codewortlänge (n)		4	6	8	10	12	14
# modellierten Fehler		153	2.133	28.305	370.269	4.822.713	62.732.133
Fehlerwahrscheinlichkeit pro Wort		$2,0069E-4$	$3,0102E-4$	$4,0134E-4$	$5,0165E-4$	$6,0195E-4$	$7,0224E-4$
Wahrsch. eines nicht modellierten Fehlers		$6,0133E-6$	$9,0199E-6$	$1,2027E-5$	$1,5033E-5$	$1,8040E-5$	$2,1046E-5$
# Datenzellen (K)		1	2	3	4	5	6
Heuristik	Erkannte Fehler	98,039%	96,906%	96,283%	96,088%	95,696%	95,406%
	Wahrsch. der Erkennung	$1,9468E-4$	$2,9200E-4$	$3,8931E-4$	$4,8640E-4$	$5,8354E-4$	$6,8009E-4$
	Wahrsch. von nichterkannten Fehlern	$1,0963E-11$	$2,6273E-10$	$3,4741E-9$	$2,1708E-7$	$3,7157E-7$	$1,0973E-6$
Unidirektional	Erkannte Fehler	94,771%	94,374%	93,937%	93,800%	93,766%	93,755%
	Wahrsch. der Erkennung	$1,9468E-4$	$2,9200E-4$	$3,8931E-4$	$4,8662E-4$	$5,8391E-4$	$6,8119E-4$
	Wahrsch. von nichterkannten Fehlern	$3,0015E-11$	$6,4497E-11$	$1,2899E-10$	$2,1506E-10$	$3,2272E-10$	$4,5198E-10$
Linear	Erkannte Fehler	95,425%	94,327%	93,913%	93,803%	93,766%	93,755%
	Wahrsch. der Erkennung	$1,9468E-4$	$2,9200E-4$	$3,8931E-4$	$4,8661E-4$	$5,8390E-4$	$6,8117E-4$
	Wahrsch. von nichterkannten Fehlern	$8,9907E-11$	$2,3687E-9$	$5,1636E-9$	$9,0318E-9$	$1,3973E-8$	$1,9987E-8$
# Datenzellen (K)			1	2	3	4	5
Heuristik	Erkannte Fehler		99,953%	99,855%	99,774%	99,766%	99,733%
	Wahrsch. der Erkennung		$2,9200E-4$	$3,8931E-4$	$4,8662E-4$	$5,8391E-4$	$6,8119E-4$
	Wahrsch. von nichterkannten Fehlern		$2,7367E-12$	$4,8717E-12$	$4,9343E-11$	$4,2058E-11$	$4,8925E-11$
Unidirektional	Erkannte Fehler		99,625%	99,576%	99,538%	99,529%	99,527%
	Wahrsch. der Erkennung		$2,9200E-4$	$3,8931E-4$	$4,8662E-4$	$5,8391E-4$	$6,8119E-4$
	Wahrsch. von nichterkannten Fehlern		$7,5029E-12$	$1,1421E-11$	$2,2839E-11$	$3,8062E-11$	$5,7087E-11$

Abschnitte gegliedert. Der erste Abschnitt beschreibt die modellierten Fehler, im zweiten Abschnitt werden die drei unterschiedlichen Codes mit einer redundanten Speicherzelle verwendet und im unteren Teil der Tabelle werden zwei redundante Speicherzellen zur Fehlererkennung genutzt.

Die erste Zeile führt die Blocklänge n des Codes in Bits auf, entsprechend umfasst ein Codewort $n/2$ Speicherzellen. In der nächsten Zeile kann man die Anzahl der modellierten Fehler finden. Dabei entspricht ein Fehler einer Kante im Fehlergraphen. Ein solcher Fehler ist das Ergebnis, wenn sich eine oder mehrere Speicherzellen unbeabsichtigt auf- oder entlädt. In der dritten Zeile der Tab. 3 wird die Wahrscheinlichkeit aufgeführt, dass ein Retention- oder ein Program-Interference-Fehler innerhalb eines Codeworts aufgetreten ist. Da die Arbeit [Ca12] drei Prozent der Retention- und zwei Prozent der Program-Interference-Fehler nicht weiter spezifiziert, wird in der nächsten Zeile die Wahrscheinlichkeit, dass ein nicht modellierter Fehler auftritt aufgelistet. Die ersten vier Zeilen der Tab. 3 beschreiben also das benutzte Fehlermodell und in unserem Falle den initialen Fehlergraphen.

In den folgenden drei Zeilen (5, 6, 7) werden die experimentellen Ergebnisse der neuen fehlererkennenden Codes beschrieben, die mit Hilfe der vorgestellten Heuristik erzeugt wurden. Die erste Zeile zeigt wie viele der modellierten Fehler mit Hilfe der Heuristik erkannt werden können. In der nächsten Zeile wird die Wahrscheinlichkeit, dass ein Fehler aufgetreten ist und erkannt werden kann aufgeführt. Und in der letzten Zeile der Fehlererkennungsergebnisse der beschriebenen Heuristik wird die Wahrscheinlichkeit dargestellt, dass ein Retention- oder Program-Interference-Fehler auftritt und nicht durch den vorgeschlagenen Code erkannt werden kann. Der fehlererkennende Code für unidirektionale Fehler, der für eine redundante Speicherzelle durch die Gleichung (2) bestimmt wird, wird in Tab. 3 als Unidirektional bezeichnet und seine Ergebnisse werden in den Zeilen 8, 9 und 10 aufgeführt. Die Ergebnisse der linearen Codes sind in den Zeilen 11, 12 und 13 aufgelistet, dabei wird die Menge von Elektronen einer Speicherzelle mit Hilfe des Gray-Codes interpretiert und mit dem binären fehlererkennenden Code aus den Gleichungen (3) werden Fehler erkannt. Im unteren Abschnitt der Tabelle wird die Gleichung (1) aus [EBE13] für die Codes zur Erkennung unidirektionaler Fehler genutzt und mit Codes verglichen, die mit Hilfe der in dieser Arbeit vorgestellten Heuristik erstellt wurden.

Tab. 3 zeigt, dass die auf unidirektionale Fehler spezialisierten Codes in vielen Fällen die kleinste Wahrscheinlichkeit aufweisen, dass ein modellierter Fehler nicht erkannt werden kann. Die mit der vorgestellten Heuristik bestimmten Codes können hingegen die größte Anzahl der modellierten Fehler erkennen, was aus den Prozenten der erkannten Fehler ersichtlich ist. Wenn wir die ganze Tabelle in Augenschein nehmen, können wir erkennen, dass für alle untersuchten Codes die Wahrscheinlichkeit, dass ein nicht modellierter Fehler auftritt um Größenordnungen höher ist, als die Wahrscheinlichkeit, dass ein modellierter Fehler auftritt und nicht erkannt werden kann.

5 Zusammenfassung

In dieser Arbeit wurde gezeigt, wie Fehler durch Fehlergraphen adäquat beschrieben werden können. Als Anwendungsbeispiel wurde das unidirektionale Fehlermodell mit beschränkter Auslenkung einem Fehlermodell gegenübergestellt, dass aus experimentellen Ergebnissen für multi-level Flash-Speicher nach [Ca12] abgeleitet wurde.

Es wurde demonstriert, wie das Bestimmen von Prüfbits eines optimalen fehlererkennenden Codes mit Hilfe eines einfachen graphentheoretischen Algorithmus erfolgen kann, der die Anzahl von Kanten in einem Fehlergraphen minimiert. Zur Berechnung der Prüfbits für größere Blocklängen wurde eine effiziente Heuristik bestimmt.

Ein ausführlicher Vergleich der Fehlererkennungseigenschaften von bekannten Codes und den neu bestimmten Codes wurde für Retention- und Program-Interference-Fehler in multi-level NAND-Flash-Speicherzellen durchgeführt. Es wurde gezeigt, dass ein Fehlergraph für unidirektionale Fehler mit der maximalen Auslenkung $l = 2$ nicht alle Fehler für multi-level Flash-Zellen abdeckt. Es erwies sich, dass das in der Literatur verwendete Fehlermodell von unidirektionalen Fehlern mit beschränkter Auslenkung nur einen Teil der häufig auftretenden Retention- und Program-Interference-Fehler nicht adäquat modelliert. Experimentell konnte gezeigt werden, dass die in der Arbeit verwendeten Codes zur Erkennung von unidirektionalen Fehlern auch zur Erkennung von Retention- und Program-Interference-Fehlern geeignet sind.

Literaturverzeichnis

[Bo98] Bossert, Martin: Kanalkodierung. B.G. Teubner Stuttgart, 1998. 2. erw. Auflage.

[BP82] Bose, B.; Pradhan, D.K.: Optimal Unidirectional Error Detecting/Correcting Codes. IEEE Transactions on Computers, C-31(6):564–568, June 1982.

[Ca10] Cassuto, Y.; Schwartz, M.; Bohossian, V.; Bruck, J.: Codes for Asymmetric Limited-Magnitude Errors With Application to Multilevel Flash Memories. IEEE Transactions on Information Theory, 56(4):1582–1595, April 2010.

[Ca12] Cai, Yu; Haratsch, Erich F; Mutlu, Onur; Mai, Ken: Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012. IEEE, S. 521–526, 2012.

[EBE13] Elarief, Noha; Bose, Bella; Elmougy, Samir: Limited Magnitude Error Detecting Codes over Z_q. IEEE Transactions on Computers, 62(5):984–989, 2013.

[Fo66] Forney, David G.: Concatenated Codes. Technical report 440, MIT Press, 1966.

[KBE11] Klove, T.; Bose, B.; Elarief, N.: Systematic, Single Limited Magnitude Error Correcting Codes for Flash Memories. IEEE Transactions on Information Theory, 57(7):4477–4487, July 2011.

[MK12] Manzor, Yifat; Keren, Osnat: Amalgamated q-ary codes for multi-level flash memories. In: IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) 2012. IEEE, S. 98–103, 2012.

[NKG15] Nieß, G.; Kern, T.; Gössel, M.: Error Detection Codes for Arbitrary Errors Modeled by Error Graphs. In: 11th Workshop on Dependability and Fault Tolerance - VERFE 2015. Porto, Portugal, March 2015.