CRC Cyclic Redundancy Check

Analyseverfahren mit Bitfiltern

Prof. Dr. W. Kowalk
Universität Oldenburg
Fachbereich Informatik
05.09.06

Sie erreichen den Autor unter der E-Mail-Adresse:
kowalk@informatik.uni-oldenburg.de

Der Text wurde mit Staroffice erstellt und ins PDF-Format gewandelt.

Inhaltsverzeichnis

1 Einführung

1.1 Einleitung

Die Untersuchungen in diesem Bericht betreffen das heutzutage weit verbreitete Verfahren zur Prüfung einer Folge von Bits auf Fehler. **CRC** steht für „*Cyclic Redundancy Check*“. Das CRC-Verfahren wird in vielen Standards vorgeschrieben und hat den Vorteil, einfach in „Hardware“ integriert werden zu können. Das Verfahren kann am einfachsten als „Polynom-Division“ beschrieben werden und soll im folgenden etwas ausführlicher erläutert werden. Ziel dieser Arbeit ist es jedoch, eine einfache Theorie vorzustellen, mit welcher die Art der erkennbaren Fehler des CRC-Verfahrens charakterisiert werden kann und somit einiges zu dessen Mächtigkeit ausgesagt werden kann.

1.2 Übersicht

Die hier vorgestellte Technik verwendet sogenannte Bitfilter, um die wichtigsten Eigenschaften des CRC-Verfahrens zu beschreiben. Ein Bitfilter betrachtet nur einen Teil der Terme eines Fehlerpolynoms und leitet aufgrund seiner Struktur die gesuchten Eigenschaften her. Da außerdem der Bitfilter unabhängig analysiert werden kann, erhält man auf diese Weise neue und wichtige Aussagen über die Mächtigkeit des CRC-Verfahrens.

Dieser Bericht führt zunächst allgemein in die Grundlagen der Polynomdarstellung von Bitfolgen ein, stellt die benötigten Operationen auf solchen Polynomen vor und definiert dann die Bitfilter, zunächst allgemein und dann spezielle Bitfilter, welche für jedes Generatorpolynom spezifisch definiert werden. Die Analyse der Eigenschaften solcher Bitfilter mit gerader Parität bezüglich eines Generators ermöglicht dann die hier vorgestellten Aussagen. Dazu gehören präzise Angaben über die Art der erkannten Fehler sowie eine vollständige Beschreibung der Möglichkeiten, Gruppenbitfehler zu erkennen, und einzelne Bitfehler auch zu korrigieren.

2 Die Anwendung

Daten werden in der Regel in Bytes gespeichert und diese wiederum bestehen aus jeweils 8 Bits. Man kann somit Daten als Folgen von Bits beschreiben. Bei der Speicherung oder der Übertragung von Daten können einzelne Bits verfälscht werden, was zu Fehlern führt, die bei vielen Anwendungen nicht hingenommen werden können. Aus diesem Grunde wird der in Form von Bits gespeicherten Information weitere Information hinzugefügt, die zwar redundant ist in Bezug auf die Bedeutung der Information, aber die hilft festzustellen, ob Information möglicherweise verfälscht wurde. Das CRC-Verfahren ist eine Methode, derartige Redundanz zu berechnen.

Es fasst prinzipiell die Bits als Summe von Potenzen X^k (also als Polynome) auf, indem es den Term X^k hinzufügt, wenn das k-te Bit den Wert 1 hat. Genauer wird das erste Bit mit $X^0=1$, das zweite mit $X^1=X$, das dritte mit X^2, usw. bezeichnet, d.h. also das k-te mit X^{k-1} und das k+1-te mit X^k.

Dieses Polynom über X wird dann durch ein vereinbartes Polynom, das aus historischen Gründen häufig als *Generatorpolynom* bezeichnet wird, dividiert; da wir auch Bitfolgen statt Polynome zur Beschreibung verwenden, benutzen wir auch die Bezeichnung *Generator*.

Der Rest dieser Division ist der Prüfwert (gelegentlich auch als Prüfsumme, Prüfrest und ähnlich bezeichnet). Der Empfänger kann im Prinzip die gleiche Operation durchführen, und wenn die Reste verschieden sind liegt ein Fehler vor (der u.U. auch im Prüfwert stehen könnte). Sind die Reste gleich, so wurde kein Fehler erkannt, d.h. entweder es liegt tatsächlich kein Fehler vor, oder der vorhandene Fehler ist mit diesem Verfahren nicht erkennbar.

Die notwendigen Operationen auf Polynomen benötigen einen speziellen Zahlkörper (0,1), so dass diese Operationen in dem Ring der Polynome durchgeführt werden können; man nennt die Menge der Polynome den Ring über dem Körper (0,1), da die Polynome im Sinne der formalen Strukturen einen Ring bilden, in welchem addiert, subtrahiert und multipliziert werden kann; dividiert werden kann nur mit einem Rest, so dass die Division in dem Ring der Polynome nicht abgeschlossen ist. Allerdings spielt gerade dieser Rest bei der Fehlererkennung mit dem CRC-Verfahren eine wichtige Rolle.

Man kann auch zeigen, dass das CRC-Verfahren geeignet ist, bestimmte Fehler zu korrigieren. Solche fehlerkorrigierenden Verfahren sind in einer Reihe von Anwendung wichtig, z.B. wenn die Datenübertragung möglichst verzögerungsarm arbeiten soll, weil zeitkritische Dienste (wie Audio) übermittelt werden soll. Auch diese Möglichkeiten sollen hier genauer untersucht werden.

3 Mathematische Grundbegriffe und Bezeichnungen

3.1 Bits

Ein **Bit** ist ein Objekt mit einem von zwei möglichen Werten (binäres Objekt). Wir bezeichnen diese Werte mit „0“ und „1“. Eine **Bitfolge** wird durch die Folge ihrer Werte 0 oder 1 beschrieben, z.B. 0010000101. Wir lesen diese Bitfolgen in der Regel von links nach rechts, d.h. links steht das erste Bit, gefolgt vom zweiten usw., bis zum letzten Bit. Daher werden Bitfolgen hier in der Regel nicht als Dualzahlen interpretiert. Werden die Bitfolgen anders aufgefasst, so wird dieses ausdrücklich erwähnt.

3.2 Der Zahlkörper (0,1)

Das CRC-Verfahren stellt eine Folge von Bits als Polynom über dem einfachsten Körper dar.

Definition 1

Ein „Körper“ ist eine formale Struktur mit zwei Operatoren, „+“ und „·“. Bezüglich des „+“-Operators bilden die Elemente eine kommutative Gruppe, bezüglich des „·“-Operators eine Gruppe. In unserem Falle besitzt der Körper nur zwei Elemente {0, 1}, die im wesentlichen den Operationen der üblichen Algebra entsprechen bis auf die Beziehung: 1+1 = 0. Die Ergebnisse der jeweiligen Operationen sind in der folgenden Tabelle dargestellt.

+	0	1
0	0	1
1	1	0

*	0	1
0	0	0
1	0	1

3.3 Bitfolgen und Polynome über dem Zahlenkörper (0,1)

Eine Bitfolge kann prinzipiell unbeschränkt lang sein. Offenbar reicht es aus, nur eine Art, etwa die 1-Werte zu beschreiben, da sich die anderen entsprechend ergeben. Aus diesen und anderen Gründen wird eine Bitfolge daher auch als Polynom $P(X)$ über einem Körper (0,1) geschrieben, wobei

$$P(X)=\sum_k a_k \cdot X^k$$

mit den k aus der Menge {0,1,2,..} und a_k aus der Menge {0,1}.

Dabei ordnen wir dem Bit '1' den Zahlenwert 1 des Zahlkörpers zu und dem Bit '0' den Zahlenwert 0. In der Regel ergeben sich aus der gleichen Darstellung von Bitwert und Zahlenwert keine Verständnisschwierigkeiten.

Beispiel 2

Die Bitfolge 0010000101 lässt sich als Polynom $P_1(X) = X^2+X^7+X^9$ darstellen. Die Bitfolge 1110011000 entspricht dem Polynom $P_2(X) = 1+X+X^2+X^5+X^6$.

Hinweis 1: Es ist lediglich eine Frage der Definition, ob das linke oder das rechte Bit einer solchen Folge als Term $X^0=1$ interpretiert wird. D.h. die obigen Bitfolgen könnten auch korrekt durch die Polynome: $P_1'(X) = 1+X^2+X^7$ bzw. $P_2'(X) = X^3+X^4+X^7+X^8+X^9$ dargestellt werden. Wir verwenden hier in der Regel die erste Darstellung.

Hinweis 2: Das Generatorpolynom kann offenbar Nullen für hohe Potenzen nicht darstellen. Die Bitfolge 1110011000 kann daher nicht eindeutig durch das Polynom $P_2(X) = 1+X+X^2+X^5+X^6$ dargestellt werden, da auch 111001100000000 auf dieses Polynom abgebildet wird. Zur eindeutigen Darstellung als Polynom ist immer die Länge der Bitfolge mit anzugeben.

Hinweis 3: Es sei darauf hingewiesen, dass die Bitfolgen in der Praxis sehr viel länger sind als in diesem und den meisten folgenden Beispielen (z.B. bis zu 36.000 Bits beim FDDI-Protokoll).

3.4 Länge und Abstand von Bitfolgen

Unter der ***Länge*** einer Bitfolge wird die Anzahl aller Terme (0- und 1-Terme) verstanden. In Polynomschreibweise wird die Differenz zwischen dem höchsten und dem niedrigsten Exponenten plus 1 des jeweils gemeinten Polynoms als Länge genommen.

Unter dem ***Abstand*** zweier Bits wird die Differenz der Position der Bits verstanden, also in der Polynomschreibweise die Differenz der Exponenten der beiden Bits. Der Abstand der Bits (oder Terme) X^m und X^n ist also $|m–n|$. Z.B. ist Abstand von X^5 und X^5 offenbar 0, der Abstand von X^6 und X^4 ist 2. Sowohl Abstand als auch Länge werden immer positiv angegeben.

3.5 Polynomaddition und Polynomsubtraktion

Derartige Polynome lassen sich addieren, was termweise geschieht. Da die Terme in einer Summe vertauscht werden können (Kommutativität), können wir diese beliebig umordnen. Für die letzten beiden Polynome erhalten wir

$$X^2+X^7+X^9 \; + \; 1+X+X^2+X^5+X^6 = 1+X+X^2+X^2+X^5+X^6+X^7+X^9.$$

Da in dem Körper (0,1) die Addition abgeschlossen sein muss, gilt 1+1=0. Daher gilt auch $X^2+X^2=0$, so dass wir für die Summe erhalten

$$X^2+X^7+X^9 + 1+X+X^2+X^5+X^6 = 1+X+X^5+X^6+X^7+X^9.$$

Man beachte, dass die Umkehrung zur Addition die Subtraktion ist und diese hier genauso funktioniert wie die Addition. D.h. $1 + 1 = 1 – 1 = 0$ usw. Im folgenden verwenden wir somit nicht mehr die Subtraktion, da diese vollständig durch die Addition ersetzt werden kann.

3.6 Polynommultiplikation

Solche Polynome lassen sich multiplizieren, wobei die üblichen Gesetze, insbesondere das Distributivgesetz gelten. Beispielsweise ist

$(1+X)\cdot(1+X) = 1\cdot(1+X) + X\cdot(1+X) = 1+X+X+X^2 = 1+X^2$,

oder

$(1+X)\cdot(1+X^2) = 1\cdot(1+X^2) + X\cdot(1+X^2) = 1+X^2+X+X^3 = 1+X+X^2+X^3$.

3.7 Polynomdivision

Um jetzt eine redundante Information zu einer Bitfolge $N(X)$ zu ermitteln, verwendet das CRC-Verfahren den Rest nach der Polynomdivision durch ein so genanntes **Generatorpolynom**. Dieses ist eine kurze Bitfolge, meist 8, 16 oder 32 Bit lang, und da im Ring der Polynome eine Division so ähnlich wie in den ganzen Zahlen durchgeführt werden kann, bleibt ein Rest, der wiederum ein Polynom ist, welches jedoch von echt kleinerem Grad ist als das Generatorpolynom.

Definition **3**
Ein *Generatorpolynom* hat mindestens die Terme X^0=1 und X^n, wobei n der Grad dieses Polynoms ist. Im folgenden schreiben wir stets G(X) für ein Generatorpolynom und nur für dieses.

Beim CRC-Verfahren wird also ein $R(X)$ berechnet, so dass $N(X) = G(X)\cdot Q(X)+R(X)$, wobei $R(X)<G(X)$; dieses bedeutet (bei Polynomen über (0,1)), dass der Grad von $R(X)$ kleiner ist als der von $G(X)$. Der Quotient $Q(X)$ interessiert im folgenden nicht mehr.

Dieser Rest R(X) wird mit der zu übertragenden Nachricht $N(X)$ an den Empfänger übermittelt. Der Empfänger führt jetzt die gleiche Operation durch und vergleicht die beiden Reste. Sind diese verschieden, so wurde ein Fehler erkannt. Sind diese jedoch nicht verschieden, so kann nicht garantiert werden, dass kein Fehler vorliegt. Es stellt sich damit die Frage, welche Typen von Fehlern man erkennen kann bzw. welche Typen von Fehlern u.U. übersehen werden könnten.

3.8 Technik der Polynomdivision

Der Ausdruck $A(X)\cdot Z(X)=B(X)$ soll für beliebige Polynome $A(X)$ und $B(X)$ nach $Z(X)$ aufgelöst werden. Man kann dieses als Division schreiben: $Z(X) = B(X) / A(X)$, wobei dieses jedoch nicht für beliebiges $A(X)$ und $B(X)$ ohne Rest durchgeführt werden kann. Allgemein gilt: $B(X) / A(X) = Q(X)$ Rest $R(X)$, oder in multiplikativer Schreibweise:

$A(X)\cdot Q(X) + R(X) = B(X)$.

Beispiele **4**
$(1+X+X^2+X^3) / (1+X) = 1+X^2$ (siehe obiges Beispiel im Abschnitt 3.6).
$(1+X^2+X^3) / (1+X) = X^2$ Rest 1, da $(1+X)\cdot X^2 = X^2+X^3$., d.h. $(1+X)\cdot X^2+1 = 1+X^2+X^3$.

Praktisch lässt sich die Division in der Regel durch einen Algorithmus durchführen, der ähnlich dem der Division ganzer Zahlen funktioniert. Da dieser im folgenden noch benötigt wird, soll er hier etwas genauer demonstriert werden.

Dazu soll zunächst die Multiplikation einer Polynoms mit einer einfachen Potenz betrachtet werden $(1+X^3)\cdot X^2 = X^2+X^5$. Die Wirkung einer solchen Operation ist ein 'Verschieben' des Polynoms um 2 Stellen in Richtung der höheren Werte (als nach rechts in Bitfolgen: 1001·001 = 001001. Dieses wird benutzt, um beispielsweise einen Term X^k des Divisors zu finden, damit $G(X)\cdot X^k$ von dem Dividenden subtrahiert werden kann. Auf diese Weise werden nach und nach alle Terme des Divisors eliminiert, bis zum Schluss nur noch Terme übrig bleiben, deren Grad kleiner ist als der von $G(X)$.

Allgemein kann der Ausdruck $A(X)\cdot X^k$ verstanden werden als das 'Verschieben' von $A(X)$ um k Stellen in die Richtung größerer Exponenten. Man kann analog unter $A(X)/X^k$ die Verschiebung von $A(X)$ um k Stellen in die Richtung kleinerer Exponenten verstehen, wobei in der Regel ein Rest bleibt, der jedoch ggf. ignoriert werden könnte, wenn er nicht benötigt wird.

Kommen wir jetzt zum Beispiel der Division von Polynomen. Wir wählen als Beispiel den Dividenden $X^{11}+X^{10}+X^8+X^7+X^5+X^2+X+1$ und den Divisor X^3+X+1 und führen die Division aus wie bei der manuellen Division üblich:

```
X11+X10+X8+X7+X5+X2+X+1 / X3+X+1 = X8+X7+X6+X5+X4+X2+X+1  Rest X2+X
X11+X9 +X8
    X10+X9+X7+X5+X2+X+1
  X10+X8+x7
      X9+X8+X5+X2+X+1
    X9+X7+x6
        X8+X7+X6+X5+X2+X+1
      X8+X6+x5
          X7+X2+X+1
        X7+X5+x4
            X5+X4+X2+X+1
          X5+X3+x2
              X4+X3+X+1
            X4+X2+x
                X3+X2+1
              X3+X +1
                  X2+X
```

Es wird also der Divisor (X^3+X+1) mit einem geeigneten Faktor X^k (hier ist zunächst k=8 zu wählen) multipliziert und zu dem Dividenden addiert. Das Ergebnis dieser Operation findet sich in dem Beispiel in der dritten Zeile. Da der höchste Term des Polynoms (X^{11}) jetzt verschwunden ist, kann das gleiche Verfahren für das Polynom geringeren Grads durchgeführt werden. Das Ergebnis erscheint dann in der ersten Zeile auf der rechten Seite hinter dem Gleichheitszeichen.

Irgendwann ist der höchste Term des restlichen Polynoms einmal kleiner als der höchste Term des Divisors. Dann kann man nicht mehr weiter dividieren und man erhält einen Rest. Der Rest kann dann hinter das Ergebnis geschrieben werden. Zur Überprüfung lässt sich der Ausdruck

$(X^8+X^7+X^6+X^5+X^4+X^2+X+1)\cdot(X^3+X+1) + X^2+X$ berechnen. Er muss wieder den Dividenden ergeben.

Man kann diese Division auch mit Bitfolgen durchführen. Dann erhält man die anschauliche Vorstellung, dass der Divisor unter dem Dividenden entlang geschoben wird und ungleiche Bits zu einer 1, gleiche Bits zu einer 0 führen. Dann wird der Divisor weiter geschoben, bis sein linkes Bit mit der nächsten 1 des restlichen Dividenden übereinstimmt. Wir verwenden hier die Darstellung der Bits in umgekehrter Reihenfolge als oben festgelegt, d.h. das höchstwertige Bit steht links.

```
1 1 0 1 1 0 1 0 0 1 1 1 / 1 0 1 1 = 1 1 1 1 1 0 1 1 1 Rest 1 1 0
1 0 1 1
  1 1 0 1 0 1 0 0 1 1 1
  1 0 1 1
    1 1 0 0 1 0 0 1 1 1
    1 0 1 1
      1 1 1 1 0 0 1 1 1
      1 0 1 1
        1 0 0 0 0 1 1 1
        1 0 1 1
            1 1 0 1 1 1
            1 0 1 1
              1 1 0 1 1
              1 0 1 1
                1 1 0 1
                1 0 1 1
                  1 1 0
```

Wir sehen hieran z.B. auch, dass die Anzahl der Bits, die sich im restlichen Dividenden ändern, jeweils genau der Anzahl der 1 Bits im Divisor entspricht. Diese Eigenschaft wird später noch sehr wichtig werden.

4 Der Operator '&'

Um aus einer Bitfolge, die etwa durch ein Polynom $V(X)$ beschrieben ist, gewisse Bits herauszufiltern, führen wir den Operator '&' ein. Sei $F(X)$ ein anderes Polynom über X und sollen genau die Terme X^k erhalten bleiben, die in $V(X)$ **und** in $F(X)$ vorkommen, so verwenden wir den Operator '&'.

4.1 Definition des Operators '&'

$$F(X)=\sum_k f_k\cdot X^k,$$
$$V(X)=\sum_k v_k\cdot X^k,$$
$$F(X)\&V(X)=\sum_k f_k\cdot v_k\cdot X^k.$$

Das Ergebnis dieses Operators ist also wieder ein Polynom, welches höchstens so viele Terme hat wie in F(X) oder in V(X) vorkommen. Man kann ihn als bit- oder termweise *Konjunktion* interpretieren, wenn 0 als falsch und 1 als wahr betrachtet werden. Logisch soll dieser Ausdruck gerade bedeuten, das nur die 'wahren' Terme ausgesondert werden.

Betrachten wir einige Eigenschaften des Operators '&'.

Satz 5

a) Der Operator „&“ ist kommutativ und assoziativ.

b) Der Operator „&“ ist distributiv über „+“.

c) Der Operator „&“ ist nicht distributiv über „·“.

d) Weder „+“ noch „·“ sind distributiv über „&“.

Beweis

Man braucht jeweils nur ein X^k für ein festes k zu betrachten, da die Operation '&' unabhängig von allen anderen Stellen ist.

Zu a) Da der Operator „&“ der Multiplikation bzw. Konjunktion oder dem logischen „Und“ entspricht, ist dieser Operator offenbar **kommutativ** und **assoziativ**.

Zu b) Offenbar enthält $R(X)\&(S(X)+T(X))$ für ein festes k den Term X^k nur dann, wenn dieser sowohl in $R(X)$ als auch in genau einem der Terme $S(X)$ oder $T(X)$ vorkommt. Gleiches gilt offenbar auch für $R(X)\&S(X)+R(X)\&T(X)$.

Die letzten Aussagen des Satzes zeigt man durch Gegenbeispiele.

Zu c) Es ist mit $k \neq 0$: $X^k \& (X^k \cdot X^k) = X^k \& X^{2k} = 0$ und nicht gleich $(X^k \& X^k) \cdot (X^k \& X^k) = X^{2k}$.

Zu d) Es ist $X^k + (0 \cdot X^k \& X^k) = X^k + 0 = X^k$ ungleich $(X^k + 0 \cdot X^k) \& (X^k + X^k) = (X^k) \& (0 \cdot X^k) = 0$.

Die folgende Tabelle stellt alle Fälle zusammen.

a	b	c	$a \cdot X^k + (b \cdot X^k \& c \cdot X^k)$	$(a \cdot X^k + b \cdot X^k)$ & $(a \cdot X^k + c \cdot X^k)$	gleich
0	0	0	0	(0)&(0) = 0	=
1	0	0	X^k	$(X^k)\&(X^k) = X^k$	=
0	1	0	0	$(X^k)\&(0) = 0$	=
1	1	0	$X^k + (0) = X^k$	$(0)\&(X^k) = 0$	≠
0	0	1	0	$(0)\&(X^k) = 0$	=
1	0	1	$X^k + (0) = X^k$	$(X^k)\&(0) = 0$	≠
0	1	1	$0 + (X^k) = X^k$	$(X^k)\&(X^k) = X^k$	=
1	1	1	$X^k + (X^k) = 0$	(0)&(0) = 0	=

Es ist $(1+X) \cdot (1 \& X) = (1+X) \& 0 = 0$ ungleich $(1+X) \& (X+X^2) = X$.

5 Bitfilter

5.1 Definition des Bitfilters

Ein **Bitfilter** beschreibt eine Teilmenge einer Folge von Bits. Um mehrere Bits einer Bitfolge zu kennzeichnen, eignet sich wieder eine Bitfolge, bei der das *k*-te Bit gesetzt ist, wenn das *k*-te Bit (einer anderen Bitfolge) gekennzeichnet werden soll. Wir benutzen daher Bitfolgen, um bestimmte Bits aus einer anderen Bitfolge "herauszufiltern". Eine solche Bitfolge wird als **Bitfilter** bezeichnet.

Natürlich kann auch ein Bitfilter durch ein Polynom beschrieben werden, welches dann entsprechend als "Filterpolynom" bezeichnet wird. Wir bezeichnen einen Bitfilter in der Polynomschreibweise meistens mit $F(X)$.

In der Regel ist die Länge eines Bitfilters nicht beschränkt. Allerdings betrachten wir im folgenden nur die untersten Bits eines Bitfilters, also einen endlichen Abschnitt. Die hier betrachteten speziellen Bitfilter besitzen stets eine endliche Periode p, bei der also die Bits an den Stellen r, $r+p$, $r+2\cdot p$ usw. für alle r alle gleich sind. Es genügt also, die ersten ein oder zwei Perioden zu betrachten, die anderen ergeben sich dann kanonisch.

Bitfilter haben natürlich spezielle Eigenschaften, die jetzt zunächst allgemein untersucht werden sollen. Danach werden die Bitfilter dann benutzt, um die Reste bei der Polynomdivision zu untersuchen.

5.2 Einspolynom und Nullpolynom

Sei $1(X) = 1+X+X^2+X^3+...+X^{k-1}+X^k+X^{k+1}+...$ (oder in Bitschreibweise: 11111111....). Wir nennen dieses auch das "Einspolynom". Entsprechend nennen wir $0(X) = 0$ das "Nullpolynom" (in Bitschreibweise: 0000000000....)

5.3 Komplement

Wir nennen $F^c(X) = F(X)+1(X)$ das Komplement zu $F(X)$. $F^c(X)$ hat genau jene Terme, die $F(X)$ nicht besitzt. Offenbar gilt $F^c(X)+F(X)=1(X)$ und $F^c(X)\&F(X)=0$.

In Bitschreibweise können wir mit $1^c=0$ und $0^c=1$ schreiben

$(111111111...)^c = 000000000....$

$(000000000...)^c = 111111111....$

$abcdefgh... + a^cb^cc^cd^ce^cf^cg^ch^c... = 11111111...$

$abcdefgh... \& a^cb^cc^cd^ce^cf^cg^ch^c... = 00000000...$

5.4 Gerade und ungerade Bitfilter

Die folgende Definition legt eine bestimmte Eigenschaft fest, die ein Bitfilter in Bezug auf einen Generator haben kann. Und zwar soll die Anzahl der 1-Bits in beiden Bitfolgen bei jeder Position

des Generators gerade (bzw. ungerade) sein. Wird dann eine Division durchgeführt, so ändert sich in der durch den Bitfilter ausgezeichneten Teilmenge des Dividenden stets eine gerade Anzahl von Bits. Diese fundamentale Eigenschaft ist besonders wichtig für die nachfolgenden Überlegungen.

Für die Polynomdarstellung erhält man die folgende formale Definition:

Definition 6

$F(X)$ hat gerade Parität bzgl. $G(X)$, falls für jedes $k \geq 0$ gilt:

$$F(X)\&(G(X)\cdot X^k)\,|_{X=1} = 0.$$

Dies bedeutet, dass $F(X)$ bei jeder Position eines Divisors $G(X)$ in einer geraden Anzahl von Stellen mit $G(X)$ einen Term gemeinsam hat. Diese gerade Anzahl kann natürlich auch null sein, oder gleich der Anzahl der Stellen von $G(X)$. Dabei kann $F(X)$ in den anderen Stellen, an denen $G(X)\cdot X^k$ also einen 0-Term hat, einen Term haben oder auch nicht. Es wird nur verlangt, dass $F(X)$ an allen Stellen, an denen $G(X)\cdot X^k$ einen Term hat, eine gerade Anzahl von Termen hat.

Die obige Schreibweise ergibt sich aus folgender Überlegung. Setzt man beispielsweise in ein Polynom $P(X) = X^2+X^7+X^9$ für X den Wert 1 ein, so erhält man: $P(1) = 1^2+1^7+1^9 = 1+1+1=1$. Ebenso gilt für $P(X) = X+X^2+X^7+X^9$ mit dem Wert X=1: $P(1) = 1+1^2+1^7+1^9 = 1+1+1+1=0$. Ist daher $P(1)$=1, so enthält $P(X)$ eine ungerade Anzahl von Termen. Ist $P(1)$=0, so ist die Anzahl der Terme in $P(X)$ gerade. Statt von "ungerader Anzahl" spricht man auch von ungerader Parität, bzw. statt von "gerader Anzahl" von gerader Parität.

Entsprechend habe $F(X)$ *ungerade* Parität bzgl. $G(X)$, falls für jedes k ≥ 0 gilt:

$$F(X)\&(G(X)\cdot X^k)\,|_{X=1} = 1,$$

d.h. wenn $F(X)\&(G(X)\cdot X^k)$ ungerade Parität besitzt. Dies bedeutet, dass $F(X)$ bei jeder Position eines Divisors $G(X)$ in einer ungeraden Anzahl von Stellen mit $G(X)$ einen Term gemeinsam hat.

Definition 7

Ein Polynom oder ein Ausdruck mit einem Polynom als Ergebnis hat ungerade Parität, wenn die Anzahl der Terme in dem Polynom ungerade ist. Entsprechend hat ein Polynom gerade Parität, wenn die Anzahl der Terme in dem Polynom gerade ist.

Hinweis: Man verwendet das Wort *Parität* nur für (un)gerade Anzahl, nicht für beliebige Zahlen (man sagt also nicht: Parität drei, Parität vier usw.).

Wird somit zu einer Bitfolge $V(X)$ an irgendeiner Stelle ein Divisor $G(X)$ hinzu addiert, so ändert sich die Parität der durch einen Bitfilter $F(X)$ mit gerader Parität bzgl. des Divisors $G(X)$ heraus gefilterten Bits nicht. Wir können somit auch bzgl. der Parität eines Restes nach einer Division durch einen entsprechenden Divisor noch aussagen machen, was das eigentliche Ziel dieser Konstruktion ist. Hat nämlich ein Bitfilter eine ungerade Anzahl von Bits aus dem Dividenden herausgefiltert, so bleibt auch im Rest eine ungerade Anzahl von Bits übrig, also mindestens eines. Damit kann der Rest nicht verschwinden.

Wiederum können wir diesen Sachverhalt mit Polynomen formal beschreiben und beweisen.

Satz 8

Habe $F(X)$ gerade Parität bzgl. $G(X)$, so gilt für jedes $V(X)$ und jedes $k \geq 0$:

$$F(X)\&(V(X)+G(X)\cdot X^k)_{|X=1} = F(X)\&V(X)_{|X=1},$$

d.h. die Addition des Terms $G(X)\cdot X^k$ ändert die Parität von $F(X)$ bzgl. $V(X)+G(X)\cdot X^k$ nicht.

Beweis

Der Beweis dieses Satzes kann durch einfaches Ausmultiplizieren des linken Terms geführt werden, wobei lediglich darauf zu achten ist, dass in jedem Körper $r + 0 = r$ ist.

$$F(X)\&(V(X)+G(X)\cdot X^k)_{|X=1} = (F(X)\&V(X) + F(X)\&(G(X)\cdot X^k))_{|X=1} =$$

$$= (F(X)\&V(X))_{|X=1} + (F(X)\&(G(X)\cdot X^k))_{|X=1} = (F(X)\&V(X))_{|X=1} + 0 =$$

$$= (F(X)\&V(X))_{|X=1}.$$

Da die Division durch ein Generatorpolynom $G(X)$ durch wiederholte Addition des Terms $G(X)\cdot X^k$ durchgeführt wird, lässt sich dieser Satz unmittelbar anwenden. Dazu nehmen wir an, wir haben ein Fehlerpolynom $E(X)$, welches bzgl. eines Bitfilters ungerade Parität hat; der Bitfilter filtert somit eine ungerade Anzahl von 1en aus dem Fehlerpolynom heraus. Zugleich habe der Bitfilter gerade Parität bzgl. $G(X)$. Die Division von $E(X)$ durch $G(X)$ lässt somit einen Rest, denn auch der Rest muss ungerade Parität haben, da er durch wiederholte Addition mit Termen der Art $G(X)\cdot X^k$ entstanden ist, und ungerade Parität bedeutet, dass es mindestens einen Term gibt. Somit lassen sich diese Art von Fehlern mit dem CRC-Verfahren erkennen. Daher spielen die Bitfilter gerader Parität bzgl. eines Generatorpolynoms eine wichtige Rolle in dieser Theorie.

Ein Beispiel soll dieses verdeutlichen. Da wir 'dividieren' müssen, schreiben wir das kleinste Bit (d.h. X^0) nach rechts! Sei der Generator **1**0**1**0**11** oder $G(X)=X^0+X^1+X^3+X^5$. Der Fehler sei 00000**1**000**111**000 oder $E(X)=X^3+X^4+X^5+X^9$. Ein Bitfilter mit gerader Parität bzgl. $G(X)$ ist ...0**11010111**10001**0**.. oder $F(X)=X^1+X^5+X^6+X^7+X^8+X^{10}+X^{12}+X^{13}+...$ (Die Konstruktion von Bitfiltern zeigen wir in Abschnitt 5.7). Dann folgt $F(X)\&E(X)=X^5$, d.h. es gibt eine ungerade Anzahl von Termen in $F(X)\&E(X)$, so dass es einen Rest geben muss, wenn $E(X)$ durch $G(X)$ dividiert wird. Z.B. ist

$$F(X)\&(E(X)+G(X)\cdot X^4)_{|X=1} = F(X)\&(X^3+X^7)_{|X=1} = (X^7)_{|X=1} = 1 \neq 0.$$

F = 0**11**0**1**0**111**0000**1**0

E = 00000**1**000**111**000 $F\&E$ = 000000000**1**00000

G_1 = **1**0**1**0**11** $G_1\&E$ = 000000**1**000**11**0000

Man überprüfe, dass $E(X)$ durch $G(X)$ einen Rest lässt (siehe auch das Beispiel in Abschnitt 6.2 auf Seite 29). Als Gegenprobe wähle man den Fehler **1**000000000000000**1** oder $E(X)=X^0+X^{15}$. Dieser lässt nach Division durch $G(X)$ keinen Rest, wie man ggf. selbst prüft. Es gibt aber auch keinen Bit-

filter, mit dem genau eines der beiden Bits ausgefiltert werden könnte, da alle Bitfilter mit gerader Parität bzgl. $G(X)$ zyklisch sind mit dem Zyklus 15; dieses wird später noch gezeigt werden.

5.5 Spezielle Bitfilter

Als nächstes müssen wir die Bitfilter mit gerader Parität bzgl. eines Generatorpolynoms genauer untersuchen, da diese die bei weitem wichtigsten Eigenschaften besitzen. Dabei interessiert zum einen die Anzahl solcher Bitfilter zum anderen jedoch auch eine Methode, wie solche Bitfilter konstruiert werden können bzw. welche Eigenschaften sie noch haben. Wir werden beispielsweise zeigen, dass Bitfilter periodisch sind, und dass es stets zwei ausgezeichnete Bitfilter gibt. Letzteres sagt der nächste Satz aus.

Satz 9

Der Bitfilter $F(X)=0(X)$ hat gerade Parität bzgl. jedes Generatorpolynoms $G(X)$, da $F(X)\&0=0$.

Der Bitfilter $F(X)=1(X)$ hat gerade Parität bzgl. jedes Generatorpolynoms $G(X)$ mit gerader Anzahl von Termen, da $1(X)\&(G(X)\cdot X^k)=G(X)\cdot X^k$.

Der Bitfilter $F(X)=1(X)$ hat ungerade Parität bzgl. jedes Generatorpolynoms $G(X)$ mit ungerader Anzahl von Termen, da $1(X)\&(G(X)\cdot X^k)=G(X)\cdot X^k$.

Während $0(X)$ keine besondere Bedeutung hat, spielt der Bitfilter $1(X)$ trotz seiner einfachen Struktur eine wichtige Rolle. Aus diesem folgt z.B. sofort, dass jede ungerade Anzahl von fehlerhaften Bits stets erkannt werden kann, wenn die Anzahl der Terme im Generatorpolynom gerade ist, da die Division durch ein entsprechendes Generatorpolynom mit gerader Anzahl von Termen einen Rest lassen muss.

5.6 Das Komplement zu einem Bitfilter

Das Komplement zu einem Bitfilter hat analoge Eigenschaften wie der Bitfilter selbst. Denn habe $G(X)$ eine gerade Anzahl von Termen und hat $G(X)\cdot X^k$ für jedes k an einer geraden Anzahl von Stellen einen Term mit $F(X)$ gemeinsam, so hat es an einer geraden Anzahl von Stellen einen Term, an welchem $F(X)$ keinen Term hat. Die gleiche Anzahl von Termen, also eine gerade, hat aber $G(X)\cdot X^k$ auch mit $F^c(X)$ gemein. Der folgende Satz sagt dieses allgemeiner und beweist es formal.

Satz 10

a) Hat $G(X)$ eine gerade Anzahl von Termen, so hat $F^c(X)$ die gleiche Parität bzgl. $G(X)$ wie $F(X)$.

b) Hat $G(X)$ eine ungerade Anzahl von Termen, so hat $F^c(X)$ die entgegengesetzte Parität wie $F(X)$.

Beweis

Es ist: $F^c(X)\&G(X)=(1(X)+F(X))\&G(X))=G(X)+F(X)\&G(X)$.

Im Falle a) ist $G(1)=0$, im Falle b) ist $G(1)=1$.

5.7 Konstruktion von Bitfiltern

Der nächste Satz sagt etwas dazu aus, wie Bitfilter konstruiert werden können und wie viele verschiedene Bitfilter es überhaupt geben kann. Wir werden allerdings später sehen, dass die Anzahl der "wesentlich" verschiedenen Bitfilter deutlich geringer ist.

Satz 11

Ist n der Grad eines Generatorpolynoms $G(X)$, so bestimmt eine Folge von n aufeinander folgenden Bits ($b_k \cdot X^k + ... + b_{k+n-1} \cdot X^{k+n-1}$) eindeutig einen Bitfilter $F(X)$ bzgl. $G(X)$ mit vorgegebener Parität. Daher ist die Anzahl verschiedener Bitfilter zu $G(X)$ mit gerader oder ungerader Parität 2^n.

Wenn n verschiedene Bits jeweils einen Bitfilter bestimmen, so bestimmt jede Kombination daraus einen anderen; denn an diesen n Stellen unterscheiden sich offenbar die Bitfilter. Es ist dann lediglich zu zeigen, dass zu n verschiedenen Bits jeweils genau ein Bitfilter konstruiert werden kann. Näheres wird in dem folgenden Beweis ausgeführt.

Beweis zu Satz 11

Wir betrachten zunächst nur gerade Parität. Sei $U(X) = \sum_k a_k \cdot X^k$ für k von r bis $r+n-1$ der *Erzeuger* des Bitfilters $F(X)$. Der nächste Faktor a_{r+n} für den Term X^{r+n} von $F(X)$ wird dann durch die Anzahl der Terme in $(G(X) \cdot X^r) \& U(X)|_{X=1} = a_{r+n}$ eindeutig bestimmt, da nur so die gerade Parität von $F(X)$ mit $G(X) \cdot X^r$ garantiert werden kann. Entsprechend wird der Faktor für den vorhergehenden Term $a_{r-1} \cdot X^{r-1}$ eindeutig durch $(G(X) \cdot X^{r-1}) \& U(X)|_{X=1} = a_{r-1}$ bestimmt, da nur so die gerade Parität von $F(X)$ mit $G(X) \cdot X^{r-1}$ garantiert werden kann.

Bei ungerader Parität setzt man $a_{r+n} = 1 + (G(X) \cdot X^k) \& U(X)\ |_{X=1}$.

Man erkennt hieraus sofort, wie man einen Bitfilter mit vorgegebener Parität konstruieren kann. Man schiebt den Generator (der aus $n+1$ Bits besteht) so unter die vorgegebenen n Bits des Bitfilters – die im folgenden als *Erzeuger* (eines Bitfilters) bezeichnet werden sollen – dass entweder links oder rechts genau ein Bit des Generators herausragt. Dieses Bit des Bitfilters ist jetzt eindeutig durch die bereits übereinstimmenden Bits zwischen Bitfilter und Generator bestimmt. Gibt es eine ungerade Anzahl Übereinstimmungen von 1 Bits in Bitfilter & Generator, so muss das nächste Bit im Bitfilter eine 1 sein, da das höchste und das niedrigste Bit des Generators jeweils eine 1 sind; wir setzen hier gerade Parität des Bitfilters bzgl. des Generators voraus. Ist jedoch die Anzahl von 1 Bits in Bitfilter & Generator gerade, so muss das nächste Bit im Bitfilter eine 0 sein. In jedem Falle sind die nächsten Bits eindeutig bestimmt. Davon ausgehend lassen sich analog die anderen Bits des Bitfilters konstruieren.

Beispiel 12

Man konstruiere einen Bitfilter mit gerader Parität bzgl. $G(X) = 1+X+X^3$, der die Bitfolge „100“ enthält.

```
  1001.. 10010.. 100101.. 1001011.. 10010111.. 1001011101.. 100101110101011..

  1101    1101     1101      1101       1101        1101          1101
```

5.8 Zyklen von Bitfiltern

Diese Überlegungen zeigen, dass die Anzahl der Bitfilter endlich ist, und dass ihre Zahl genau bestimmt werden kann. Hieraus folgt aber auch, dass die Bitfilter sich in ihrem Muster irgendwann einmal wiederholen müssen; denn wenn n Bits alle nachfolgenden Bits bestimmen, so muss irgendein Muster von n Bits sich irgendwann einmal wiederholen, und von dort aus werden wieder die gleichen Bitfolgen konstruiert, usw. Der folgende Satz macht hierzu eine Aussage.

Satz 13

Jeder Bitfilter $F(X)$ mit gerader oder ungerader Parität bzgl. $G(X)$ hat eine Periode, die nur von $G(X)$ abhängt.

Beweis zu 13

Legt man die ersten n Bits fest, so wird daraus eindeutig ein Bitfilter erzeugt; irgendwann wird sich das Bitmuster der ersten n Bits wiederholen, da n Bits höchstens 2^n verschiedene Werte annehmen können, so dass sich dann alle folgenden Bits entsprechend wiederholen. Daher kann die Periode eines Bitfilters bzgl. $G(X)$ auch niemals länger sein als 2^n.

Der Beweis des letzten Satzes geht davon aus, dass sich jedes Muster einmal wiederholt. Dieses folgt unmittelbar daraus, dass zu n gegebenen Bits nicht nur der "Nachfolger" in einer solchen Folge, sondern auch der "Vorgänger" eindeutig bestimmt ist. Es können also keine "Verzweigungen" oder "Zusammenführungen" derart konstruierter, verschiedener Bitfolgen entstehen.

Wenn sich die Bitfolgen eines Bitfilters stets zyklisch wiederholen, so kann man Fragen, welche Bitfilter lediglich Teile eines gemeinsamen Zyklus sind. Dazu geben wir die folgende Definition.

Definition 14

Zwei Bitfilter $U(X)$ und $V(X)$ heißen **äquivalent**, wenn es eine Zahl k gibt, so dass $U(X) \mathbin{//} X^k = V(X)$.

Gibt es keine solche Zahl, so heißen $U(X)$ und $V(X)$ **disjunkt**.

Wir verwenden in dieser Definition den Ausdruck $U(X) \mathbin{//} X^k$ dazu, um die Terme um k Stellen zu verschieben. Der Rest, der üblicherweise bei einer Division auftritt, wird also nicht betrachtet. Außerdem gehen wir davon aus, dass die Bitfilter unbeschränkt lang sind, so dass $U(X)$ bei der Divisi-

on nicht effektiv verkürzt wird. Daher bedeutet diese Definition, dass zwei Bitfilter äquivalent sind, wenn sie den gleichen Zyklus besitzen (d.h. die gleiche Bitfolge, die sich zyklisch wiederholt), aber nicht notwendigerweise mit den gleichen Werten anfangen.

Ist p die Periode des Bitfilters $U(X)$ so gilt offenbar $U(X) \,//\, X^p = U(X)$. Gilt außerdem $U(X) \,//\, X^k = V(X)$, so folgt $V(X) \,//\, X^{p-k} = U(X)$, da

$U(X) = U(X) \,//\, X^p = (U(X)//X^k) \,//\, X^{p-k} = V(X) \,/\, X^{p-k}$.

Die kleinere der beiden Zahlen k und $p–k$ wird als die *Phase* der beiden äquivalenten Bitfilter $U(X)$ und $V(X)$ bezeichnet; in der Regel können beide Werte k und $p–k$ als Phase bezeichnet werden, ohne dass Missverständnisse entstehen. Entsprechend kann man auch / statt // schreiben, wenn die Bedeutung aus dem Kontext klar hervorgeht.

Satz 15

Zwei äquivalente Bitfilter $U(X)$ und $V(X)$ haben stets die gleiche Periode p.

Gilt $U(X) \,//\, X^k = V(X)$, so gilt auch $U(X) = V(X) \,//\, X^{p-k}$.

Die kleinere der beiden Zahlen k und p-k wird als Phase zwischen $U(X)$ und $V(X)$ bezeichnet.

Als Phase von $U(X)$ zu $V(X)$ wird die Zahl k bezeichnet (d.h. wenn $V(X)$ um k Stellen zu kleineren Indizes verschoben wird, erhalten wir $U(X)$).

5.9 Komplementäre Bitfilter

Die Anzahl der Terme (oder „1-en“ in der Bitfolge) in dem Bitfilter $F(X)$ innerhalb irgendeiner Periode werde mit $|F(X)|_{per}$ bezeichnet. Für jeden äquivalenten Bitfilter $F(X)//X^k$ gilt offenbar $|F(X)|_{per} = |F(X)//X^k|_{per}$, da jede Bitfolge der Periodenlänge p die gleiche Anzahl von Termen hat, unabhängig von der Phase. Als Bitfolge aufgefasst ist die Anzahl der Nullen in dieser Periode offenbar

p-$|F(X)|_{per}$.

Dann gilt für das Komplement $F^c(X)$ von $F(X)$, dass die Anzahl der Terme dort p-$|F(X)|_{per}$ ist: p-$|F(X)|_{per} = |F^c(X)|_{per}$. Also erhalten wir das folgende Ergebnis.

Satz 16

$p = |F(X)|_{per} + |F^c(X)|_{per}$.

Habe $G(X)$ eine gerade Anzahl von Termen und sei die Periode p eines Bitfilters $F(X)$ ungerade. Da $G(X)$ eine gerade Anzahl von Termen hat, sind (wegen Satz 10 auf Seite 17) $F(X)$ und $F^c(X)$ Bitfilter mit gleicher Parität bzgl. $G(X)$. Die Anzahl der Terme in $F(X)$ und $F^c(X)$ muss dann jedoch verschieden sein, d.h. $F(X)$ und $F^c(X)$ können nicht äquivalent sein.

Satz 17

Habe $G(X)$ eine gerade Anzahl von Termen und sei die Periode p eines Bitfilters $F(X)$ ungerade. Dann sind $F(X)$ und $F^c(X)$ nicht äquivalente Bitfilter mit gleicher Parität bzgl. $G(X)$.

5.10 Maximale Periodenlängen von Bitfiltern

Aus dem letzten Satz folgt nun aber noch ein relevantes Ergebnis, welches eine Aussage über die maximalen Periodenlängen von Bitfiltern macht. Da unter den oben genannten Voraussetzungen die Bitfilter $F(X)$ und $F^c(X)$ disjunkt sind, ist bei $G(X)$ mit gerader Anzahl von Termen und einer geraden Parität niemals ein Bitfilter maximaler Länge zu finden. Von den 2^n Bitfiltern werden zwei für die Bitfilter $0(X)$ und $1(X)$ benötigt; ist ein weiterer vorhanden, so auch dessen Komplement, da $(2^n-2)/2=2^{n-1}-1$ eine ungerade Periodenlänge bedeutet. Tatsächlich kann bei gerader Parität nur ein maximaler Bitfilter mit „halber“ Periodenlänge gefunden werden, was für die Anwendung durchaus eine markante Einschränkung bedeutet. Ein ausführliche Begründung für diese Tatsache wird später (Abschnitt 6.7) gegeben.

5.11 Mindestzahl disjunkter Bitfilter

Habe $F(X)$ gerade Parität bzgl. $G(X)$ mit gerader Anzahl von Termen, dann gibt es mindestens vier disjunkte Bitfilter gleicher Parität bzgl. eines Generatorpolynoms $G(X)$, nämlich $0(X)$, $1(X)$, $F(X)$ und $F^c(X)$.

Beispiel 18

Die folgenden Bitfilter haben gerade Parität zu der Bitfolge 11101 (d.h. dem Generatorpolynom $1+X+X^2+X^4$):

0, 1

1000101, 0001011, 0010110, 0101100, 1011000, 0110001, 1100010

0111010, 1110100, 1101001, 1010011, 0100111, 1001110, 0011101

In diesem Beispiel wurden nur die ersten Perioden der jeweiligen Bitfilter geschrieben. Man erkennt, dass es genau vier disjunkte Bitfilter gibt, wobei jeweils zwei komplementär zueinander sind und zwei Gruppen zu je sieben äquivalenten Bitfiltern. Wir könnten also sämtliche Bitfilter durch die beiden Bitfolgen 1, 1000101 sowie die hierzu komplementären und jeweils äquivalenten Bitfilter beschreiben. Daher gibt es tatsächlich nur zwei wesentlich verschiedene Bitfilter; alle anderen lassen sich einfach aus diesen beiden Bitfiltern ableiten und haben die gleiche bzw. komplementäre Struktur.

Da für die folgenden Betrachtungen die Längen der Perioden der Bitfilter eine wichtige Rolle spielen, sind diese Aussagen besonders relevant. Generatoren mit genau vier Bitfiltern mit gerader Parität sollen als optimale Generatoren bezeichnet werden.

Definition 19

Gibt es zu einem Generator $G(X)$ mit gerader Anzahl von Termen nur zwei wesentlich verschiedene Bitfilter (d.h. alle anderen sind entweder komplementär oder äquivalent zu diesen beiden), die bzgl. $G(X)$ gerade Parität haben, so heißt $G(X)$ *optimaler Generator*.

5.12 Beispiele für Bitfilter

Einige Beispiele sollen die Eigenschaften von Bitfiltern verständlicher machen.

Beispiel 20

Das CCITT-Generatorpolynom $1+X^5+X^{12}+X^{16}$ hat genau vier äquivalente Bitfilter der Periodenlängen zweimal 1 und zweimal $2^{15}-1 = 32767$.

Das gleiche gilt für das CRC-16-Generatorpolynom $1+X^2+X^{15}+X^{16}$.

Beispiel 21

Die Bitfilter 0 (der Länge 1) und

1000000010000011100010101011010100001011001100011001010011101110100010011011110
0001101001000111110101110010010111111001111011101 (der Länge 127)

sowie die jeweils komplementären und äquivalenten Bitfilter haben gerade Parität bzgl. des HEC-Generatorpolynoms $1+X+X^2+X^8$, welches in ATM verwendet wird.

Es sind aber Generatorpolynome mit ungerader Anzahl von Termen bekannt, die nur zwei Bitfilter gerader Parität besitzen.

Beispiel 22

Die Bitfilter 0 sowie 1001011 und die sechs hierzu äquivalenten haben gerade Parität bzgl. des Generatorpolynoms $G(X)=1+X+X^3$ (Bitfolge 1101). Die komplementären Bitfolgen bilden jedoch keine Bitfilter gerader Parität bzgl. $G(X)$, da es nur acht verschiedene Bitfilter geben kann.

Die bisher vorgestellten Strukturen waren relativ einfach. Es gibt aber Generatorpolynome mit Bitfiltern sehr unterschiedlicher Periodenlänge.

Beispiel 23

Zum Generatorpolynom $G(X)=1+X+X^2+X^7$ (Bitfolge 11100001) gibt es sechzig Bitfilter der Periodenlänge 60, jeweils dreißig Bitfilter der Periodenlänge 30 und 15, vier Bitfilter der Periodenlänge 4 sowie jeweils zwei Bitfilter der Periodenlängen 2 und 1, jeweils mit gerader Parität bzgl. $G(X)$. Siehe auch Beispiel 28.

Es fällt hier auf, dass das kleinste gemeinsame Vielfache (KGV) aller Längen die Zahl 60 ist. Dieses KGV spielt eine wichtige Rolle für die Teilbarkeitseigenschaft solcher Generatorpolynome. Wir werden später zeigen, dass in jedem Falle bei geraden Bitfiltern der längste Bitfilter dem KGV der Längen aller Bitfilter – auch von sich selbst – entspricht.

Das letzte Beispiel lässt sich einfach zu einem Generatorpolynom mit Bitfiltern gleicher Länge umändern.

Beispiel 24

Das Generatorpolynom $G(X)=1+X+X^3+X^7$ (Bitfolge 11010001) erzeugt, jeweils mit gerader Parität, die Bitfilter 0 und

10000000100000111000101010110101000010110011000110010100111011101111111011111000111010101001010111101001100111001101011000100 0 (Periode 127)

sowie die jeweils komplementären und äquivalenten.

5.13 Summen von Bitfiltern

Eine weitere wichtige Eigenschaft eines Bitfilters ist durch die Längen verschiedener Bitfilter gegeben. Bisher konnte kein Beispiel gefunden werden, bei dem die Periode eines Bitfilters kein Teiler der Periode eines Bitfilters mit maximaler Periode ist. Sei also p die Periode eines maximalen Bitfilters zum Generatorpolynome $G(X)$ mit gerader Anzahl von Termen, so soll gezeigt werden, dass die Periode jedes anderen Bitfilters zu dem entsprechenden Generatorpolynome $G(X)$ Teiler von p ist.

Im folgenden sei $G(X)$ stets ein Generatorpolynom mit einer geraden Anzahl von Termen.

Satz 25

Sind $F(X)$ und $H(X)$ zwei gerade Bitfilter zu $G(X)$, so ist $F(X)+H(X)$ auch ein gerader Bitfilter zu $G(X)$.

Beweis

Ein gerader Bitfilter $F(X)$ zu $G(X)$ ist gekennzeichnet durch die Eigenschaft, dass $(G(X)\cdot X^k)\&F(X)$ eine gerade Anzahl von Termen für jedes positive k hat; auch für jeden anderen Bitfilter $H(X)$ mit gerader Parität zu $G(X)$ gilt $(G(X)\cdot X^k)\&H(X)$. Damit ist auch die Summe der Terme beider Ausdrücken gerade. Wenn an der gleichen Position X^s sowohl in $(G(X)\cdot X^k)\&F(X)$ als auch in $(G(X)\cdot X^k)\&\mathrm{H}(X)$ ein Term steht, so wird daraus in der Summe $(G(X)\cdot X^k)\&F(X) + (G(X)\cdot X^k)\&\mathrm{H}(X)$ ein 0-Term. Also verschwinden zwei Terme. Steht genau ein Term und ein 0-Term an der gleichen Position, so verbleibt genau ein Term. Und 0-Terme an der gleichen Position bleiben ebenfalls 0-Terme. Also ändert sich die gerade Anzahl von 1-Termen in $(G(X)\cdot X^k)\&F(X)$ und $(G(X)\cdot X^k)\&\mathrm{H}(X)$ in eine gerade Anzahl von 1-Termen in $(G(X)\cdot X^k)\&F(X) + (G(X)\cdot X^k)\&\mathrm{H}(X)$. Dann ist nach Definition aber $F(X)+\mathrm{H}(X)$ ein gerader Bitfilter zu $G(X)$.

Formal ließe sich kürzer schreiben:

$$(F(X)+H(X))\&(G(X)\cdot X^k)|_{X=1} = F(X)\&(G(X)\cdot X^k)|_{X=1}+H(X)\&(G(X)\cdot X^k)|_{X=1} = 0+0 = 0.$$

Beispiel 26

Ein Beispiel für diesen Satz ist zu einem Bitfilter $F(X)$ der komplementäre Bitfilter $F^c(X)$, für den natürlich gilt:

$$F^c(X) = F(X) + 1(X).$$

Da nach Voraussetzung $F(X)$ ein gerader Bitfilter zu $G(X)$ ist und $1(X)$ ebenfalls ein gerader Bitfilter zu $G(X)$ ist, ergibt die Summe beider einen geraden Bitfilter zu $G(X)$.

Man sieht aus der formalen Notation auch sofort, dass die Summe zweier Bitfilter mit ungerader Parität einen solchen mit gerader Parität ergibt, und dass die Summe zweier Bitfilter mit ungleicher Parität einen Bitfehler mit ungerader Parität ergibt. Der obige Satz ließe sich also verallgemeinern zu:

Korollar 27

Sind $F(X)$ und $H(X)$ Bitfilter mit gleicher Parität bzgl. $G(X)$, so hat $F(X)+H(X)$ gerade Parität bzgl. $G(X)$, sonst hat $F(X)+H(X)$ ungerade Parität bzgl. $G(X)$.

Die Anzahl der Terme in $G(X)$ spielt offenbar keine Rolle bei diesem Ergebnis.

5.14 Basisbitfilter

Es zeigt sich jetzt, dass der Bitfilter, der den Erzeuger $L(X)=1$ enthält, alle anderen Bitfilter durch eine geeignete Summe phasenverschobener Bitfilter generieren kann. Ehe wir dieses Verfahren allgemein beschreiben, soll anhand eines Beispiels diese wichtige Eigenschaft demonstriert werden.

Wir hatten bereits das Generatorpolynom $G(X)=1+X+X^2+X^7$ (Bitfolge 11100001) betrachtet, welches die folgenden Bitfilter generiert:

Beispiel 28

Bitfilter zu dem Generatorpolynom $G(X)=1+X+X^2+X^7$ sind

0
1000000100001110010101110100100111111011110001101010001011 01
0100000110001001011111001110 11
111000010100110
100100011110101
1001

01
1

Der längste Bitfilter wird von dem Erzeuger $L(X)=1$ generiert (wir legen in der Bitfolge die kleinste Stelle der Generatorpolynoms nach links; im Bild sind die sieben Bits des Erzeugers unterstrichen). Neben diesem gibt es noch die folgenden phasenverschobenen Bitfilter, welche wir (einschließlich den durch $L(X)=1$ erzeugten) **Basisbitfilter** nennen wollen und jeweils gesondert bezeichnen

Beispiel 29

Basisbitfilter zu dem Generatorpolynom $G(X)=1+X+X^2+X^7$

B_0 = 1000000100001110010101110100100111111011110001101010001011 01
B_1 = 1100000100001110010101110100100111111011110001101010001011 0
B_2 = 0110000010000111001010111010010011111101111000110101000101 1
B_3 = 1011000001000011100101011101001001111110111100011010100010 1
B_4 = 1101100000100001110010101110100100111111011110001101010001 0
B_5 = 0110110000010000111001010111010010011111101111000110101000 1
B_6 = 1011011000000100001110010101110100100111111011110001101010 00

Der erste Basisbitfilter $B_0(X)$ ist der durch $L(X)=1$ erzeugte Bitfilter und $L_r(X)$ geht aus $L_0(X)$ durch Phasenverschiebung mit der Phase r zu den höheren Potenzen hervor. $L_r(X) = L_r(X)/X^{p-r}$.

Es soll jetzt durch Addition ein beliebiger anderer Bitfilter konstruiert werden; wir nehmen als Beispiel den Bitfilter 1110000101001 10, mit dem unterstrichenen Erzeuger 1110000 von sieben Bits. Da die oberen vier Bits 0 sind, ist der erste Bitfilter, den wir benötigen B_2. Dieses würde als einziger Summand den Erzeuger 0110000 ergeben. Damit die erste 0 auch noch zu 1 wird, ist der Bitfilter B_0 zu addieren, d.h.

```
B0    = 1000000100001110010101110100100111111011110001101010001011 01
B2    = 0110000010000111001010111010010011111101111000110101000101 1
B0+B2= 1110000101001101110000101001101110000101001101110000101001 10
```

Die Summe beider Bitfilter ergibt somit den Bitfilter mit dem Erzeuger 1110000. Dieses ist offenbar der oben genannte Bitfilter mit der entsprechenden Periode 15; als Summe von Bitfiltern der Periode 60 hat er auch die Periode 60, also kommt viermal die gleiche Bitfolge vor.

Damit sollte das Konstruktionsprinzip klar sein. Beginnend mit dem Basisbitfilter mit dem höchsten Index wird bei Bedarf ein niedrigerer Basisbitfilter addiert, wenn das komplementäre Bit im Erzeuger benötigt wird. Der Aufwand ist offenbar gleich $O(n)$, wenn n der Grad des Generatorpolynoms ist.

Das zweite Beispiel soll den Bitfilter mit der Länge 30 erzeugen. Es gilt:

```
B0    = 1000000100001110010101110100100111111011110001101010001011 01
B1    = 1100000100001110010101110100100111111011110001101010001011 0
B0+B1 = 0100000110001001011111001110110100000110001001011111001110 11
```

Auch hier erhalten wir einen Bitfilter als Summe zweier Basisbitfilter; i.allg. können natürlich auch mehr als zwei Summanden benötigt werden.

Für den Bitfilter 1001 mit der Periode 4 (dieses muss wegen n=7 zum Erzeuger 1001100 erweitert werden) erhalten wir:

```
B0        =   10000001000011100101011101001001111110111100011010100010110 1
B1        =   11000000100001110010101110100100111111011110001101010001011 0
B4        =   11011000000100001110010101110100100111111011110001101010001 0
B0+B1+B4  =   10011001100110011001100110011001100110011001100110011001100 11001
```

Selbst der Bitfilter 1(X) läst sich auf diese Weise erzeugen. Es gilt

```
B0       = 10000001000011100101011101001001111110111100011010100010110 1
B1       = 11000000100001110010101110100100111111011110001101010001011 0
B2       = 01100000010000111001010111010010011111101111000110101000101 11
B3       = 10110000001000011100101011101001001111110111100011010100010 1
B4       = 11011000000100001110010101110100100111111011110001101010001 0
B6       = 10110110000001000011100101011101001001111110111100011010100 0
B0+B1+B2+B3+B4+B6=
           11111111111111111111111111111111111111111111111111111111111 11111
```

Um 0(X) zu erzeugen, wird die leere Summe genommen.

Damit haben wir ein allgemeines Konstruktionsprinzip gefunden, mit dessen Hilfe wir durch Addition von Basisbitfiltern jeden anderen Bitfilter konstruieren können. Die Konsequenzen dieses Ergebnisses werden im nächsten Abschnitt besprochen.

Konstruktionsprinzip 30

Konstruktionsprinzip von Bitfiltern durch Summation von Basisbitfiltern.

Um einen beliebigen Bitfilter mit Erzeuger $L(X)$ durch Addition von Basisbitfiltern erzeugen zu können, muss zu einem Bitfilter der Erzeuger konstruiert werden. Beginnend mit dem höchsten Index $r = n–1, n–2, .., 1, 0$ und dem Erzeuger $L_n(X) = 0(X)$ wird der Bitfilter $B_r(X)$ als Summand hinzugefügt, wenn an der Stelle X^r des Erzeugers die bisherige Summe ein falsches Bit erzeugen würde.

$L_r(X) = L_{r+1}(X)$, falls $L_{r+1}(X)\&X^r = L(X)\&X^r$, sonst $L_r(X) = L_{r+1}(X)+B_r(X)$.

Da die Basisbitfilter mit Index $s<r$ an der Stelle X^r bis X^{n-1} jeweils den Wert 0 haben, verändert ein Basisbitfilter mit Index $s<r$ die Stelle X^r bis X^{n-1} nicht mehr, kann also hinzuaddiert werden, um die Stelle X^s richtig zu setzen, ohne X^{s+1} bis X^{n-1} zu verändern.

Ein weiteres Beispiel soll zeigen, dass diese Technik auch für Generatoren mit ungerader Anzahl von Termen gilt. Sei also $G(X) = 1+X^2+X^4$ (10101). Dann ist der kanonische Basisbitfilter B_0=100010 mit der Periode 6. Die anderen Basisbitfilter sind B_1=010001 , B_2=101000 , B_3=010100. Man überzeugt sich leicht, dass die 16 Summen von 0 bis 4 Bitfiltern alle anderen Bitfilter ergeben. U.a. findet sich natürlich nicht der Bitfilter 1(X) unter diesen, da dieses kein gerader Bitfilter zu $G(X)$ ist. Neben B_0 und dessen äquivalenten Bitfilter haben auch 111100 und dessen äquivalente Bitfilter die Periode 6, während es noch drei Bitfilter mit Periode 3 gibt: 110, 011, 101. Insgesamt gibt es also neben 0(X) noch 15 Bitfilter, die alle aus der Summe von Basisbitfiltern erzeugbar sind.

5.15 Wichtige Resultate bzgl. Basisbitfilter

Die erste wichtige Schlussfolgerung ist, dass der Erzeuger $L(X)=1$ den längsten Bitfilter erzeugt. Denn der durch 1 erzeugte Bitfilter hat mindestens die Länge n, wenn $n>0$ der Grad des Generatorpolynoms ist; dann lassen sich die obigen n Basisbitfilter $B_0(X), B_1(X), \ldots, B_{n-1}(X)$ durch Phasenverschiebung konstruieren, und dann können hieraus sämtliche anderen Bitfilter durch Addition geeigneter Basisbitfilter berechnet werden. Hätte jetzt ein Bitfilter eine größere Periode q als der durch $L(X)=1$ erzeugte, so ließe sich dieser durch die Basisbitfilter erzeugen, welche jedoch alle die gleiche Periode p haben. Damit hätte dieser Bitfilter eine Periode p; q kann also nicht größer als p sein. Aus diesem Widerspruch folgt, dass $L(X)=1$ den längsten Bitfilter erzeugt.

Das zweite Ergebnis besagt, dass die Länge jedes Bitfilters ein Teiler der Länge des längsten Bitfilters sein muss. Denn jeder Bitfilter lässt sich aus phasenverschobenen Bitfiltern durch Addition erzeugen, und da Basisbitfilter alle die gleiche Periode haben, muss die Periode jedes Bitfilters ein Teiler des durch $L(X)=1$ generierten Bitfilters sein. Damit ist auch das kleinste gemeinsame Vielfache aller Bitfilterperioden gleich der Länge der längsten Bitfilter (es kann mehrere Bitfilter maximaler Länge geben, wie z.B. den komplementären Bitfilter).

Natürlich ist auch offensichtlich, dass sich jeder Bitfilter *eindeutig* als Summe der Basisbitfilter berechnen lässt, da der obige Konstruktionsvorgang eindeutig entscheidet, welcher Basisbitfilter benötigt wird.

Satz 31

Sei $G(X)$ ein Generator vom Grad n mit einer geraden Anzahl von Termen. Werde zu $G(X)$ durch den Erzeuger $L(X)=1$ ein Bitfilter mit gerader Parität, $B_0(X)$ mit Periode p konstruiert, so gilt:

1. $B_0(X)$ hat die größte Periode aller durch $G(X)$ erzeugten Bitfilter;
2. Für jeden Bitfilter mit einer Periode q gilt: q teilt p;
3. Jeder Bitfilter lässt sich eindeutig durch eine Summe der Basisbitfilter konstruieren.

5.16 Zusammenfassung der Eigenschaften von Bitfiltern

Da die Bitfilter recht komplexe Eigenschaften haben, sollen hier in einer Kurzübersicht verschiedene Aspekte zusammengefasst werden.

1. Bitfilter $F(X)$ sind Polynome.
2. Ein *Generatorpolynom* $G(X)$ hat mindestens die Terme $G(X)=1+\ldots+X^n$. Dabei heißt n der *Grad* von $G(X)$ und die Anzahl der Terme in $G(X)$ wird als die *Parität* von $G(X)$ bezeichnet; entsprechend spricht man von *gerader* bzw. *ungerader Parität*.
3. Ein Bitfilter $F(X)$ hat *gerade* Parität bzgl. eines Generatorpolynoms $G(X)$, wenn für jedes $k \geq 0$ gilt: $F(X)\&(G(X)\cdot X^k)\,|_{X=1} = 0$.

Ein Bitfilter $F(X)$ hat *ungerade* Parität bzgl. eines Generatorpolynoms $G(X)$, wenn für jedes $k \geq 0$ gilt: $F(X)\&(G(X)\cdot X^k)\,|_{X=1} = 1$.

4. Wir nennen $1(X)= 1+X+X^2+X^3+X^4+...$ das *Einspolynom*, $0(X)=0$ das *Nullpolynom*.

5. $1(X)$ hat die gleiche Parität bzgl. eines Generatorpolynoms $G(X)$ wie $G(X)$.
   $0(X)$ hat gerade Parität bzgl. jedes Generatorpolynoms $G(X)$.

6. Sei $F(X)$ ein Bitfilter (oder Polynom), so nennen wir $F^c(X) = 1(X)+F(X)$ das *Komplement* zu $F(X)$.

7. Ist n der Grad von $G(X)$, so gibt es 2^n verschiedene Bitfilter mit gerader Parität zu $G(X)$; entsprechend gibt es 2^n verschiedene Bitfilter mit ungerader Parität zu $G(X)$.

8. Jeder Bitfilter mit gerader Parität bzgl. eines Generators $G(X)$ besitzt einen Zyklus, d.h. eine ganze Zahl p, so dass $F(X)=F(X)/X^{k\cdot p}$ für jedes $k \geq 0$.

9. Geht ein Bitfilter $H(X)$ aus einem anderen $F(X)$ hervor, indem gilt $H(X)=F(X)/X^k$, $k \geq 0$, so werden $H(X)$ und $K(X)$ als *äquivalent* bezeichnet. k (ggf. der größere Wert von k und $p–k$) werden als Phase von $F(X)$ zu $H(X)$ bezeichnet.

10. Sind $F(X)$ und $H(X)$ Bitfilter mit gleicher Parität bzgl. $G(X)$, so hat $F(X)+H(X)$ gerade Parität bzgl. $G(X)$, sonst hat $F(X)+H(X)$ ungerade Parität bzgl. $G(X)$.

11. Jeder Bitfilter $F(X)$ kann eindeutig aus einem Erzeugerpolynom $E(X)$ konstruiert werden, dessen Grad nicht größer als $n–1$ ist, wenn n der Grad des Generatorpolynoms $G(X)$ ist. Wir schreiben $F(X) = KON[E(X)]$ oder kürzer $F_E(X)$.

12. Habe $G(X)$ gerade Parität. Der aus $L(X)=1$ erzeugte Bitfilter $F_1(X)$ mit gerader Parität bzgl. $G(X)$ hat maximale Periode p.
    Die Periode jedes anderen Bitfilters mit gerader Parität bzgl. $G(X)$ teilt die Periode p von $F_1(X)$.

13. Die äquivalenten Bitfilter $B_k(X)$, k=0..n-1 mit den Eigenschaften: $B_k(X)/X=B_{k-1}(X)$; $B_0(X)=KON[1]$ heißen *Basisbitfilter*.

14. Jeder Bitfilter mit gerader Parität bzgl. G(X) kann eindeutig als Summe bestimmter Basisbitfilter erzeugt werden.

15. Habe $G(X)$ eine gerade Anzahl von Termen und den Grad n. Dann haben die Bitfilter mit gerader Parität bzgl. G(X) eine maximale Periode $p \leq 2^{n-1}-1$ (Satz 42 auf Seite 39).

6 Die Mächtigkeit des CRC-Verfahrens

6.1 Das CRC-Prüfverfahren

Das CRC-Verfahren verwendet den Rest ein Polynomdivision, um einen Fehler zu erkennen. In der Regel wird das CRC-Verfahren folgendermaßen eingeführt:

a) Sei $N(X)$ eine Nachricht, die als Polynom dargestellt wird.

b) Sei $G(X)$ das Generatorpolynom vom Grad n.

c) Sei $R(X)$ der Rest nach Division von $N(X)\cdot X^n$ durch $G(X)$, also $G(X)\cdot Q(X) + R(X) = N(X)\cdot X^n$ oder auch $G(X)\cdot Q(X) = N(X)\cdot X^n + R(X)$; daher ist $N(X)\cdot X^n + R(X)$ durch $G(X)$ teilbar.

d) Es werde das Polynom $S(X) = N(X)\cdot X^n + R(X)$ gesendet.

e) Sei $E(X)$ das Fehlerpolynom; es enthält den Term X^k, genau wenn bei der Übertragung von $S(X)$ im k-ten Bit ein Fehler auftritt.

f) Der Empfänger erhält: $H(X) = S(X) + E(X)$.

g) Der Empfänger dividiert $H(X)$ durch $G(X)$:

$$\frac{H(X)}{G(X)}=\frac{S(X)+E(X)}{G(X)}=\frac{S(X)}{G(X)}+\frac{E(X)}{G(X)}=\frac{N(X)\cdot X^n+R(X)}{G(X)}+\frac{E(X)}{G(X)}.$$

Hier lässt der erste Quotient wegen c) keinen Rest, so dass ein möglicherweise beim Empfänger ermittelter Rest ausschließlich durch das Fehlerpolynom bewirkt wurde.

Hierzu sind einige Erläuterungen angebracht.

Wird eine Bitfolge verändert, so lässt sich dieses durch ein Fehlerpolynom $E(X)$ darstellen. Es hat genau an jenen Stellen einen Term, an denen die Bitfolge verändert wurde.

Ist $N(X)$ das Nachrichtenpolynom und $R(X)$ der Rest, der als Division von $N(X)\cdot X^n$ mit $G(X)$ entsteht. Da der Rest an das Nachrichtenpolynom angehängt wird, kann das gesendete Polynom als $N(X)\cdot X^n+R(X)$ beschrieben werden, wenn n der Grad von $G(X)$ ist. Das fehlerhaft empfangene Polynom kann dann als $(N(X)\cdot X^n+R(X)) + E(X)$ beschrieben werden. Da eine Division durch $G(X)$ auch des empfangenen Polynoms durchgeführt wird, und der Rest von $N(X)\cdot X^n+R(X)$ null ergibt, ist der Rest des empfangenen Polynoms ausschließlich von $E(X)$ abhängig. Wir verwenden also E(X) zur Beschreibung der Fehler und betrachten im folgenden lediglich dieses Fehlerpolynom zur Charakterisierung der mit diesem Verfahren erkennbaren Fehler.

6.2 Fehlerbursts

Wir gehen von einem Generator der Länge n+1: $G(X)=1+...+X^n$. Ein Fehlerburst oder Burst der Länge k ist eine Bitfolge mit k Bits, wobei das höchste und tiefste Bit jeweils 1 sind $B(X)=X^r$

$+...+X^{r+k-1}$. Fehlerbursts kommen in realen System häufig vor, da externe Störungen meist mehrere nahe beieinander stehende Bits verfälschen.

Sei $E(X)=X^r+...+X^{r+n}$ ein Fehlerburst der Länge n+1 und sei das Fehlermuster das gleiche wie das Muster des Generators, d.h. $E(X)=G(X)\cdot X^r$ für ein passendes r. Dann ist der Rest nach Division durch G(X) 0, d.h. $E(X)/G(X) = Q(X)$; ein Fehler wird nicht entdeckt. In allen anderen Fällen jedoch lassen Fehlerburst der maximalen Länge n+1 einen von null verschiedenen Rest, was jetzt gezeigt werden soll.

Um $E(X)$ durch $G(X)$ zu dividieren, wird mehrfach der Tem $G(X)\cdot X^q$ mit passenden q's addiert. Sei $E(X)=X^r+...+X^k+...X^j+...+X^{r+n}$ mit der Länge n+1, dann hat $E(X)+G(X)\cdot X^r$ eine geringere Länge als n+1, da $X^r+...+a_k\cdot X^k+...+a_j\cdot X^j+...+X^{r+n} + (X^r+...+X^{r+n}) = X^k+...X^j$, wobei k und j der kleinste und größte Term in $E(X)$ sind, die sich von den Termen in $G(X)\cdot X^r$ unterscheiden (von denen es nach Voraussetzung mindestens einen geben muss). Offenbar sind k und j weder r noch $r+n$, so dass wir $|k-j|+1<n+1$ erhalten. Die entstehende Bitfolge von Termen, die nicht null sind (und somit einen Fehlerburst darstellen) ist also kürzer als $G(X)$; außerdem ist diese Bitfolge nicht null, da es mindestens ein Bit geben sollte, welches in $E(X)$ und $G(X)\cdot X^r$ verschieden ist.

Um weiter zu dividieren, wird $G(X)\cdot X^r$ auf diesen Fehlerburst mit einer Länge kürzer als n+1 (oder auf einen Fehlerburst, der von Anfang an kürzer ist) addiert, so dass $a_{r+1}\cdot X^{r+1}+...+X^k+...+X^j+...+X^{r+n} + (X^r+...+X^{r+n}) = X^r+...+X^j$, wobei wieder $|r-j|+1<n+1$, und wiederum dieser Rest nicht verschwinden kann, denn der kleinste Term X^r ist niemals null. Wiederholt man diese Schritte, so erhält man irgendwann einen Rest, der kleiner ist als X^n, was bedeutet, dass er nicht weiter durch $G(X)$ geteilt werden kann.

Das Ergebnis dieser relativ einfachen Überlegungen ist es, dass alle (außer einem) Fehlerbursts, die nicht länger als $G(X)$ sind, mit dem CRC-Verfahren erkannt werden. Dieses hängt nicht von der Struktur von $G(X)$ ab, also gleich ob dessen Parität gerade oder ungerade ist. Dieses trifft auch auf einzelne Bitfehler (als 'Fehlerburst' der Länge 1) zu.

Wir demonstrieren hier diesen Beweis anhand eines Beispiels. Wie in der Division üblich schreiben wir die Bits hier in absteigender Reihenfolge, d.h. das kleinste Bit steht rechts. Seien

E_0 = 00000**1**00**111**00000

G = **1**0**1**0**11**

Dann erhalten wir

E_0 = 00000**1**00**111**00000
G = **1**0**1**0**11**
E_1 = 0000000**11**0000000
G = **1**0**1**0**11**
E_2 = 000000000**11**0**11**000
G = **1**0**1**0**11**
E_3 = 00000000**111**0**1**00
G = **1**0**1**0**11**
E_4 = 000000000**1**000**1**0
G = **1**0**1**0**11**
E_5 = 000000000000**1**00**1**

Man erkennt an den Resten, dass die Länge des Fehlerbursts niemals größer wird als n, obgleich er sehr wohl genau gleich n werden kann, wie bei E_2, E_3 und E_4. Daher wird das niedrigste Bit stets gesetzt, also kann der Rest niemals verschwinden!

Diese Aussagen lassen sich auch mit Bitfiltern beweisen, wobei man jedoch gerade Parität des Generators $G(X)$ annehmen muss. Da dieses weniger allgemein ist, übergehen wir den Beweis hier.

Der folgende Satz fasst das Ergebnis noch einmal zusammen.

Satz 32

Fehlerbursts, die nicht länger als der Generator $G(X)$ sind, werden vom CRC-Verfahren erkannt; einzige Ausnahme sind Fehlerbursts $E(X)=G(X)\cdot X^r$ mit dem gleichen Muster wie der Generator.

6.3 1 Bit-Fehler

Satz 33

Sei $E(X)=X^k$, so lässt die Division von $E(X)$ mit $G(X)$ einen Rest ungleich null.

Der Beweis folgt unmittelbar aus Satz 32, wobei die Länge des Bursts 1 ist. Es gibt jedoch weitere Möglichkeiten, den Beweis zu führen.

Würde der Satz nicht gelten, so wäre z.B. $X^k=Q(X)\cdot G(X)$, da dann X^k von $G(X)$ geteilt wird. Dabei habe $G(X)=1+..+X^n$ mindestens zwei Terme. Der niedrigste Term von $Q(X)$ sei X^r, der höchste X^s. Dann gilt für das Produkt: $Q(X)\cdot G(X) = X^r+..+X^{s+n}$, weil sich der niedrigste und höchste Term des Produkts nicht gegen andere aufheben lassen; dies gilt auch, wenn $r=s$! Daher hat das Produkt mindestens zwei Terme, d.h. es ist nicht möglich, irgendeinen Term X^k durch ein Polynom $G(X)$ mit zwei oder mehr Termen ohne Rest zu dividieren. Alle 1 Bit-Fehler werden somit richtig erkannt.

Einfacher lässt sich der Beweis mit Bitfiltern zeigen, wenn $G(X)$ eine gerade Anzahl von Termen hat. Dann ist $1(X)$ ein Bitfilter mit gerader Parität bzgl. $G(X)$, und da $E(X)=X^k$ ungerade Parität bzgl. $1(X)$ hat, hat es dieses auch bzgl. des Restes, der somit nicht null sein kann.

6.4 Ungerade Anzahl von Bitfehlern

Doch es gibt noch weitere Fehlerklassen, die sicher erkannt werden können. Diese sollen jetzt dargestellt werden.

Satz 34

Sei $E(X)$ ein Polynom mit einer ungeraden Anzahl von Termen und $G(X)$ ein Generatorpolynom mit einer geraden Anzahl von Termen. Dann teilt $G(X)$ nicht $E(X)$.

Beweis

Da $1(X)$ gerade Parität bzgl. $G(X)$ hat und ungerade Parität bzgl. $E(X)$, hat $1(X)$ bzgl. des Rests nach Division von $E(X)$ durch $G(X)$ ungerade Parität, d.h. der Rest ist nicht null.

6.5 Erkennbare 2 Bit-Fehler

Da wir mit einfachen Bitfehlern oder ungerader Fehlerzahl offenbar keine Probleme haben, betrachten wir jetzt die 2-Bit-Fehler, d.h. zwei fehlerhafte Bits.

Als erstes zeigen wir, welche Klassen von 2-Bit-Fehlern mit Sicherheit durch das CRC-Verfahren erkannt werden, indem wir zeigen, welche 2-Bit-Fehler nach Division durch $G(X)$ einen Rest lassen.

Satz 35

Sei $G(X)$ eine Generatorpolynom und sei p die größte Periode aller Bitfilter mit gerader Parität bzgl. $G(X)$. Dann wird für jedes k, welches kein Vielfaches von p ist, $E(X)=X^{k+r}+X^r$ nicht von $G(X)$ geteilt.

Beweis

Sei ein Polynom $E(X)=X^{k+r}+X^r$ mit zwei Termen gegeben, die nicht im Abstand p (oder einem Vielfachen davon) liegen; dieser Abstand sei k. Sei $H(X)$ der Bitfilter mit der längsten Periode. Dann gibt es einen zu $H(X)$ äquivalenten Bitfilter $F(X)$, so dass dieser die beiden Bits $X^{k+r}+X^r$ verschieden ausfiltert, d.h. das eine mit einer 0, das andere mit einer 1; es gilt also für $F(X)$ entweder $F(X)\&E(X)=X^{k+r}$ oder $F(X)\&E(X)=X^r$, aber weder $F(X)\&E(X)=0$ noch $F(X)\&E(X)=X^{k+r}+X^r$. Wäre es nicht möglich, einen solchen äquivalenten Bitfilter zu finden, so hätten alle äquivalenten Bitfilter im Abstand k das gleiche Bit, und damit hätten die Bitfilter die Periode k, also eine andere Periode als p, was nach Voraussetzung nicht der Fall ist. Dann hat das Polynom $F(X)\&E(X)$ ungerade Parität bzgl. $G(X)$.

Die Division wird durch wiederholte Addition des Terms $G(X)\cdot X^u$ auf $E(X)$ ausgeführt. Dieses ändert die Parität von $F(X)\&E(X)$ bzgl. $G(X)$ nicht. Da $F(X)$ nicht $0(X)$ ist, gibt es einen Term X^s in $F(X)$, der kleiner ist als X^n, also s<n. Wird die Division vollständig durchgeführt, so sind alle restlichen Terme kleiner als X^n, wobei eine ungerade Anzahl von Termen in $F(X)$ liegen muss, da die Parität von $F(X)\&E(X)$ bzgl. $G(X)$ ungerade bleibt, also nicht null werden kann. Daher muss ein Rest bei der Division von $E(X)$ mit $G(X)$ bleiben.

Diese Argumentation besagt also, dass die durch $F(X)$ ausgefilterten Bits auch nach Addition von $X^u\cdot G(X)$ stets die gleiche Parität haben bzgl. $G(X)$, also wenn sie zunächst eine ungerade hatten, dann ändert sich daran auch nichts mehr. Damit kann auch ein Rest nicht verschwinden und somit wird ein entsprechender 2 Bit-Fehler erkannt.

Somit lassen sich alle 2-Bit-Fehler erkennen, vorausgesetzt ihr Abstand ist nicht ein Vielfaches der maximalen Periode des jeweils längsten Bitfilters. Ist diese Periode sehr lang, so wird praktisch bei der Datenübertragung kein doppelter Bitfehler übersehen, da die Datenblöcke ja nicht so lang sind, dass zwei fehlerhafte Bits entsprechende Abstände haben könnte. Ein häufig verwendeter Grad von Generatorpolynomen ist 16, so dass hier immerhin Blöcke bis zu einer Länge von 32767 Bits oder 4095 Bytes verwendet werden können, so dass sämtliche ungeraden Bitfehler aber auch sämtliche doppelten Bitfehler sicher erkannt werden. Da die meisten Standardprotokolle deutlich

kürzere Blocklängen verwenden, können diese als relativ sicher angesehen werden. Daher ist dieses Ergebnis für die Praxis sehr relevant.

Wir demonstrieren hier anhand eines Beispiels für n=5, dass doppelte Bitfehler im Abstand $p=2^{n-1}-1=15$ nicht erkannt werden. Ein Bitfilter gerader Parität zum Generator 110101 ist 100001010011011. In der folgenden Darstellung stehe das kleinste Bit rechts.

F_0 = 1000010100110111

E_0 = **1**0000**0**0**0**00**0**00**1**

G = **11**0**1**0**1**

Dann erhalten wir

E_0 = **1**0000**0**0**0**00**0**00**1**
G = **11**0**1**0**1**
E_1 = **0****1**0**1**0**1**0000**0**00**1**
G = **11**0**1**0**1**
E_2 = **0**0**111****1****1**0**0**00**0**00**1**
G = **11**0**1**0**1**
E_3 = **0**0000**1**0**1****1**00**0**00**1**
G = **11**0**1**0**1**
E_4 = **0**0000**1****1**0**1**00**0**00**1**
G = **11**0**1**0**1**
E_5 = **0**0000000**11****1**00**0**1
G = **11**0**1**0**1**
E_6 = **0**0000000000**11**0**1**0**1**
G = **11**0**1**0**1**
E_7 = 0000000000000000

Es wurden jeweils jene Bits unterstrichen, die auch im Bitfilter stehen. Man überzeuge sich, dass die Anzahl der Bits im Bitfilter und im Rest E_i immer gerade ist. Mit dem Generator **1**0**1**0**11**, der offenbar symmetrisch zu **11**0**1**0**1** ist, erhalten wir mit dem Bitfilter 1000011101100101 mit dem Bitfilter 1000011101100101

F_0 = 1000011101100101
E_0 = **1**000000000000000**1**
G = **1**0**1**0**11**
E_1 = **0**0**1**0**11**000**0**0000**0**0**1**
G = **1**0**1**0**11**
E_2 = **0**0000**111**0**0**0000**0**0**1**
G = **1**0**1**0**11**
E_3 = **0**0000**0****1**0**0****11**00**0**0**1**
G = **1**0**1**0**11**
E_4 = **0**0000**0**000**11**0**1**0**0**0**1**
G = **1**0**1**0**11**

E_5 = 0000000001111101
G = 101011
E_6 = 0000000000101011
G = 101011
E_7 = 0000000000000000

Es wurde im Satz nicht vorausgesetzt, dass $G(X)$ eine gerade Anzahl von Termen hat, was auch nicht nötig ist; denn die Aussage, dass die Parität von $F(X)\&E(X)$ bzgl. $G(X)$ ungerade bleibt gilt auch in diesem Fall. Wir geben auch hierfür ein Beispiel an. Der Bitfilter 100101011100111 hat gerade Parität zum Generator 11001 und die Periode 15. Es ist der einzige Bitfilter mit gerader Parität bzgl. 11001, da es nicht mehr als 15 äquivalente Bitfilter geben kann. Für diesen Generator folgt für einen Fehler im Abstand 15:

F_0 = 1000100110101111
E_0 = 1000000000000001
G = 11001

Dann erhalten wir

E_0 = 1000000000000001
G = 11001
E_1 = 0100100000000001
G = 11001
E_2 = 0010110000000001
G = 11001
E_3 = 0001111000000001
G = 11001
E_4 = 0000111000000001
G = 11001
E_5 = 0000000101000001
G = 11001
E_6 = 0000000011010001
G = 11001
E_7 = 0000000000011001
G = 11001
E_8 = 0000000000000000

Es wurden jeweils jene Bits unterstrichen, die auch im Bitfilter stehen. Man sieht an diesem Beispiel, dass die Perioden bei einer ungeraden Anzahl von Termen in $G(X)$ auch maximal lang sein können, also $p=2^n-1$. Allerdings scheint der Nachteil, mit derartigen Generatoren nicht alle ungeraden Fehler erkennen zu können schwerer zu wiegen, als ein Generator vom Grad $n+1$, welcher die gleiche Bitfilterperiode hätte.

6.6 Nicht erkennbare 2-Bit Fehler

Tatsächlich gilt die Verallgemeinerung dieses Satzes, nämlich dass genau zwei Bits im Abstand von p (oder einem Vielfachen davon) ohne Rest durch den Generator geteilt werden können, wie wir im nächsten Satz zeigen.

Satz 36

Sei $G(X)$ eine Generatorpolynom mit gerader Anzahl von Termen und sei p die größte Periode aller Bitfilter mit gerader Parität bzgl. $G(X)$. Dann wird $X^r \cdot (X^p+1) = X^r + X^{r+p}$ für jedes $r \geq 0$ durch $G(X)$ geteilt.

Dieser Satz gibt also eine Bedingung an, unter der die Division stets keinen Rest lässt. D.h. dass kein 2 Bit-Fehler im Abstand von p erkannt werden kann.

Beweis zu Satz 36

Wegen $X^r \cdot (X^p+1) = X^r + X^{r+p}$ untersuchen wir nur die Teilbarkeit von $E(X)=X^p+1$. Die Division wird durch wiederholte Addition von $G(X) \cdot X^k$ zu $E(X)=X^p+1$ mit stetig fallendem k durchgeführt. Lasse jetzt X^p+1 nach Division durch $G(X)$ einen Rest $R(X)$, der u.a. den Term X^r enthalte; dabei sei r minimal gewählt. Da $r < n$, erzeugt X^r eindeutig einen Bitfilter $F(X)=KON(X^r)$ der Periode p (hat $F(X)$ eine Periode $q < p$, so teilt q p, da p alle Periodenlängen als Teiler hat, und $F(X)$ hat somit auch die Periode p).

Ist $r>0$, so hat $F(X)$ nicht den Term $X^0=1$, und wegen der Periode p auch nicht den Term X^p; also ist $F(X)\&E(X) = F(X)\&(X^p+1) = 0$, und daher hat F($X$) gerade Parität bzgl. $E(X)=X^p+1$. Nach Annahme und Konstruktion von $F(X)$ ist $R(X)\&F(X) = X^r$. Damit hat $F(X)$ ungerade Parität bzgl. des Rests $R(X)$ der Division von X^p+1 mit $G(X)$, was ein Widerspruch ist, da der Rest aus wiederholter Addition von Termen $G(X) \cdot X^k$ zu $E(X)$ entsteht, wobei die Parität von $F(X)\&E(X) = F(X)\&(X^p+1)$ jeweils nicht verändert wird. Also kann der Term X^r nicht im Rest enthalten sein.

Ist $r=0$, also der Term $X^0=1$ im Rest enthalten, so ist wegen der Periodizität von $F(X)$ auch X^p in $F(X)$ enthalten, wobei $F(X)$ durch $X^0=1$ erzeugt wird. Daher enthält $F(X)$ die Terme X^p und 1, also ist $F(X)\&(X^p+1) = X^p+1$ und $F(X)$ hat gerade Parität bzgl. X^p+1. Auch hier würde der Rest $R(X)=1$ ungerade Parität bzgl. des Bitfilters $F(X)$ haben, und man schließt wie oben.

Zusammenfassend lässt sich der Beweis folgendermaßen übersichtlich darstellen:

1. Sei $R(X)$ der Rest von (X^p+1) nach Division durch $G(X)$: $(X^p+1) \cdot G(X) = Q(X) + R(X)$.

2. Sei X^r ein 1-Term von $R(X)$.

3. X^r generiere den Bitfilter $F(X)=KON(X^r)$ mit gerader Parität bzgl. $G(X)$.

4. $F(X)\&(X^p+1)$ hat gerade Parität,

5. $F(X)\&R(X)=X^r$ hat ungerade Parität.

6. Aus 4. und 5. entsteht ein Widerspruch, da $R(X)$ aus X^p+1 durch wiederholte Addtion von $G(X)$ hervorgeht, und $F(X)$ gerade Parität bzgl. $G(X)$ hat. Also ist $R(X)=0$.

Dieses gilt auch für Abstände, die Vielfache von p sind. So ergibt etwa $(1+X^p)+(X^p+X^{2p}) = (1+X^{2p})$. Da jeder einzelne Term durch $G(X)$ teilbar ist, gilt dieses auch für die Summe. Allgemein folgt wegen der Beziehung $(1+X^{s\cdot p})=(X^0+X^p)+(X^p+X^{2p})+(X^{2p}+X^{3p})+...+(X^{(s-2)p}+X^{(s-1)p})+(X^{(s-1)p}+X^{s\cdot p})$, dass diese Aussage für jeden Vielfachen Abstand $s\cdot p$ gilt.

Wird $(1+X^r)$ durch $G(X)$ geteilt, so auch $(1+X^r)\cdot X^k$. Also gilt diese Aussage für alle 2 Bit-Fehler, soweit deren Abstand ein Vielfaches von p ist.

Corollar 37

Sei p die größte Periode aller Bitfilter mit gerader Parität bzgl. $G(X)$ (mit gerader Anzahl von Termen). Dann teilt G(X) jedes Fehlerpolynom $E(X)=X^r+X^{r+s\cdot p}$ für alle $r \geq 0$ und $s \geq 1$, d.h. alle 2 Bit-Fehler mit einem Abstand, der ein Vielfaches von p ist.

Anhand der Bitfilter lassen sich somit einfach die erkennbaren 2 Bit-Fehler klassifizieren. Die obigen Beispiele zeigen deutlich, dass auch sehr kurze Generatorpolynome $G(X)$ wie das CRC-16 sehr lange Bitfilter besitzen, die zu $G(X)$ gerade Parität haben. Das bedeutet etwa, dass bereits ein 16-Bit-Generatorpolynom Bitfolgen mit einer Länge bis zu 32767 Bits sicher auf 1- und 2 Bit-Fehler überprüfen kann. Da die üblichen Längen von Datenblöcken (außer beim FDDI) deutlich kürzer sind, erhalten wir hiermit ein effektives Verfahren zur Fehlererkennung.

6.7 Fehlerkorrektur aus den Resten

Es kam bisher nur darauf an, Fehler zu erkennen. Es stellt sich die Frage, ob das CRC-Verfahren darüber hinaus auch für die Fehlerkorrektur geeignet ist: lässt sich aus dem Rest nach Division durch ein Generatorpolynom evtl. auf den aufgetretenen Fehler schließen?

Damit Fehler korrigiert werden können, muss aus dem Rest eindeutig auf einen (oder ggf. mehrere) Fehler geschlossen werden können. Wir gehen hier davon aus, dass wir nur einen einfachen Bitfehler haben, d.h. $E(X)=X^r$. Wenn jetzt der Rest von $E(X)$ nach Division durch $G(X)$ für jedes r innerhalb gewisser Grenzen eindeutig ist, so können wir aus den Resten eindeutig auf den Fehler X^r schließen und dieses falsche Bit natürlich korrigieren.

Der folgende Satz sagt auch hierzu etwas aus:

Satz 38

Sei p die längste Periode der Bitfilter mit gerader Parität bzgl. eines Generatorpolynoms $G(X)$ (mit gerader Anzahl von Termen). Dann haben alle p Fehlerpolynome aus der Menge $\{E(X)= X^k, k=0..p-1\}$ verschiedene Reste.

Beweis

Lassen zwei Terme den gleichen Rest $R(X)$ bezüglich Division mit $G(X)$, d.h.

1. $X^s = G(X)\cdot Q_s(X) + R(X)$,

2. $X^t = G(X)\cdot Q_t(X) + R(X)$,

so erhalten wir für die Summe dieser beiden Gleichungen:

3. $X^s + X^t = G(X)\cdot Q_s(X) + R(X) + G(X)\cdot Q_t(X) + R(X) = G(X)\cdot[Q_s(X) + Q_t(X)]$.

Sei $F(X)$ der Bitfilter mit gerader Parität bzgl. $G(X)$ mit maximale Perioden p. Dann lassen nach Corollar 37 nur dann zwei Terme bei der Division mit $G(X)$ den Rest 0, wenn ihr Abstand p oder ein Vielfaches davon ist. Das bedeutet, dass auch s und t den Abstand p oder ein Vielfaches hiervon benötigen, um zu gleichen Resten bei Division mit $G(X)$ zu kommen.

Daher kann auf diese Weise eindeutig bei 1 Bit-Fehlern aus dem Rest nach Division mit $G(X)$ auf die Position des 1 Bit-Fehlers geschlossen werden, wenn p größer ist als die Länge des Datenpakets. Wir erhalten einen fehlerkorrigierenden Code für 1 Bit-Fehler.

Es folgt auch sofort, welche 1 Bit-Fehler den gleichen Rest liefern, nämlich die in einem Abstand, der ein Vielfaches von p ist.

Beispiel 39

Das HEC (Header Error Correction Code) Generatorpolynom, welches in ATM verwendet wird, nutzt diese Eigenschaft, um einen Bitfehler zu erkennen und zu korrigieren. Es lautet: $1+X+X^2+X^8$ und hat nur zwei zueinander komplementäre Bitfilter der Länge 127 mit gerader Parität bzgl. HEC (neben den Bitfiltern 0 und 1). Das HEC prüft ein Feld von 32 Bit Länge, seine Bitfilter haben aber eine Periode von 127, so dass bei 1 Bit-Fehlern ohne weiteres aus dem Rest auf die Position des Fehlers geschlossen werden kann. Allerdings gibt es auch 3-Bit-Fehler, die die gleichen Reste lassen wie 1 Bit-Fehler, so dass die Erkennung nicht eindeutig ist. Da man bei ATM jedoch davon ausgeht, dass 3-Bit-Fehler dort praktisch nicht auftreten, verwendet man dieses Verfahren beim ersten Fehler dieser Art. Tritt beim nächsten Datenpaket (dort Zelle genannt) noch einmal ein Fehler auf, dann wird eine völlig neue Synchronisation des Zellstroms durchgeführt.

Der folgende Satz beantwortet die Frage, wie man erkennen kann, ob es sich bei einem gegebenen Rest um einen 1 Bit-Fehler oder einen 2 Bit-Fehler handelt.

Satz 40

Habe $G(X)$ eine gerade Parität. Die Reste von 1 Bit-Fehler haben dann stets eine ungerade Parität, die von 2 Bit-Fehlern eine gerade.

Beweis

Die Anzahl der Bits in dem Rest bei dem 1 Bit-Fehler $E(X) = X^k$ muss natürlich ungerade sein, wenn das Generatorpolynom eine gerade Anzahl von Termen hat. Denn wählt man $1(X)$ als Bitfilter mit gerader Parität bzgl. $G(X)$, so ist die Parität von $E(X)$ zu $1(X)$ ungerade, und daran ändert auch die Division durch $G(X)$ nichts mehr.

Entsprechend folgt, dass ein 2 Bit-Fehler stets einen Rest mit gerader Parität hinterlässt, so dass allein aus dem Rest geschlossen werden kann, ob es sich um einen 1 Bit-Fehler oder einen 2 Bit-Fehler handelt. Ist die Anzahl der Terme im Rest gerade, so kann es kein 1 Bit-Fehler sein.

Die Anzahl verschiedener Reste beträgt 2^n, wenn n der Grad des Generatorpolynoms ist; davon hat die Hälfte eine gerade Anzahl von Termen und die andere Hälfte eine ungerade Anzahl von Termen, also höchstens 2^{n-1} besitzen gerade bzw. ungerade Termzahl. Ist $p=2^{n-1}-1$, wie z.B. beim CRC-16-Generatorpolynom, so werden sämtliche 1 Bit-Fehler auf sämtliche Reste mit ungerader Anzahl von Termen abgebildet. Daher gibt es zu jedem Rest mit ungerader Anzahl von Termen genau einen 1 Bit-Fehler.

Es sei hier erwähnt, dass es stets genau einen Rest gibt, der nicht durch Division eines einzelnen Terms erhalten wird. Denn wenn $G(X)$ ein Generatorpolynom vom Grad n ist mit einer geraden Anzahl von Termen, so gibt es genau ein Polynom $R(X)$ vom Grad kleiner als n, so dass gilt: $R(X)\cdot X+G(X)=R(X)$. Dann lässt $R(X)\cdot X^k$ nach Division durch $G(X)$ den Rest $R(X)$, da

$R(X)\cdot X^k+G(X)=R(X)\cdot X^{k-1}$

$R(X)\cdot X^{k-s}+G(X)=R(X)\cdot X^{k-s-1}$, $s=1,2,..,k-2$

$R(X)\cdot X^1+G(X)=R(X)\cdot X^0=R(X)$.

Beispiel 41

Sei $G(X)=1+X+X^2+X^8$, so lässt sich $R(X)$ kanonisch konstruieren.
Die Bedingung lautet allgemein $R(X)\cdot X+G(X)=R(X)$, also für diesen speziellen Fall mit allgemeinen Termen für $R(X)=a_0+a_1X+a_2X^2+a_3X^3+a_4X^4+a_5X^5+a_6X^6+a_7X^7$ erhalten wir:

$(a_0X+a_1X^2+a_2X^3+a_3X^4+a_4X^5+a_5X^6+a_6X^7+a_7X^8)+(1+X+X^2+X^8)=$

$1+(1+a_0)X+(1+a_1)X^2+a_2X^3+a_3X^4+a_4X^5+a_5X^6+a_6X^7+(1+a_7)X^8=$

$a_0+a_1X+a_2X^2+a_3X^3+a_4X^4+a_5X^5+a_6X^6+a_7X^7$

Koeffizientenvergleich ergibt eindeutig:

$a_0=1$, $a_1=1+a_0=0$, $a_2=1+a_1=1$, $a_3=a_2=1$, $a_4=a_3=1$, $a_5=a_4=1$, $a_6=a_5=1$, $a_7=a_6=1$, und $1+a_7=0$

Daher gilt $R(X)=1+X^2+X^3+X^4+X^5+X^6+X^7$.

Habe $G(X)$ eine ungerade Anzahl von Termen, so gilt jedoch etwas anderes. Sei $G(X)=1+X^2+X^8$, so lässt sich die obige Bedingung formulieren als:

$(a_0X+a_1X^2+a_2X^3+a_3X^4+a_4X^5+a_5X^6+a_6X^7+a_7X^8)+(1+X^2+X^8)=$

$1+a_0X+(1+a_1)X^2+a_2X^3+a_3X^4+a_4X^5+a_5X^6+a_6X^7+(1+a_7)X^8=$

$a_0+a_1X+a_2X^2+a_3X^3+a_4X^4+a_5X^5+a_6X^6+a_7X^7$

Koeffizientenvergleich ergibt:

$a_0=1$, $a_1=a_0=1$, $a_2=1+a_1=0$, $a_3=a_2=0$, $a_4=a_3=0$, $a_5=a_4=0$, $a_6=a_5=0$, $a_7=a_6=0$, und $1+a_7=0$.

Daher kann a_7 kein eindeutiger Wert zugewiesen werden. Es existiert in diesem Fall also kein solches $R(X)$.

Der Grund ist offenbar, dass in der Folge $\{a_k\}_{k=0,1,...,n-1}$ der Wert der Koeffizienten a_k nur einmal geändert wird, während er bei gerader Anzahl von Termen in $G(X)$ eine gerade Anzahl mal geändert wird.

Hieraus folgt aber auch, dass für Bitfilter mit gerader Parität bzgl. $G(X)$ (wobei $G(X)$ vom Grad n ist und eine gerade Anzahl von Termen besitzt) die maximale Periode p höchstens 2^{n-1}-1 sein kann. Denn wenn der Term X^p den gleich Rest lässt wie 1, und es höchstens 2^{n-1}-1 verschiedene Reste für die Terme 1, X, X^2, X^3, ... geben kann, dann kann auch die Periode p nicht länger sein. Es folgt der Satz:

Satz 42

Habe $G(X)$ eine gerade Anzahl von Termen und den Grad n. Dann hat jeder Bitfilter mit gerader Parität bzgl. $G(X)$ als längste Periode $p=2^{n-1}$-1.

Daher folgt die früher schon zitierte Behauptung, dass die Bitfilter mit gerader Parität bzgl. $G(X)$ keine Periode länger als 2^{n-1}-1 besitzen können.

6.8 Mögliche fehlerhafte Korrekturen

Allerdings stellt dieses kein sicheres Verfahren zur Korrektur von bestimmten Fehlerklassen dar, da natürlich auch ein 3-Bit-Fehler oder jede andere ungerade Zahl an Bitfehlern vorliegen könnte.

Beispiel 43

3 Bit-Fehler mit den gleichen Resten wie 1 Bit-Fehler lassen sich meistens einfach konstruieren: Wenn $G(X)$ genau vier Terme hat, dann ist bereits der Ausdruck $X^k + G(X)\cdot X^{k-n}$ ein Polynom mit drei Termen, welches offenbar nach der Division durch $G(X)$ den gleichen Rest lässt wie X^k.

6.9 Das Beispiel Bluetooth

In Bluetooth, einem neueren Standard, wird ebenfalls die fehlerkorrigierende Eigenschaften des CRC-Verfahrens genutzt.

Beispiel 44

Das (für dieses Beispiel angenehm kurze) Generatorpolynom bei Bluetooth hat die Polynomdarstellung $X^5+X^4+X^2+1$ und die Bitfolge 110101. Bitfilter sind neben 0 und 1 noch 100001010011011 und dessen Komplement. Da die Periode des längstens Bitfilters 15 beträgt, können 1 Bit-Fehler mit einem Abstand bis zu 14 erkannt und eindeutig einem bestimmten Bit zugeordnet werden (der Abstand von X^r und X^s ist $|r\text{-}s|$).

Im einzelnen erhalten wir

Fehler	Rest	Fehler	Rest	Fehler	Rest	Fehler	Rest
1	'00001'	X^4	'10000'	X^8	'10110'	X^{12}	'11100'
X^1	'00010'	X^5	'10101'	X^9	'11001'	X^{13}	'01101'
X^2	'00100'	X^6	'11111'	X^{10}	'00111'	X^{14}	'11010'
X^3	'01000'	X^7	'01011'	X^{11}	'01110'	X^{15}	'00001'

Man beachte, dass der Rest 10011 nicht auftaucht. Die Addition von $(X^4+X+1)\cdot X^{k+1}$ mit $G(X)\cdot X^k$ liefert $(X^{5+k}+X^{2+k}+X^{1+k})$ + $(X^{5+k}+X^{4+k}+X^{2+k}+X^k)=(X^{4+k}+X^{1+k}+X^k)$, also $(X^4+X+1)\cdot X^k$. Daher kann X^4+X+1 nicht der Rest nach Division von X^k durch $G(X)$ sein.

Für einige 2 Bit-Fehler erhalten wir die Werte

Fehler	Rest	Fehler	Rest	Fehler	Rest	Fehler	Rest
$1+X^{15}$	'00000'	$1+X^{11}$	'01111'	$1+X^7$	'01010'	$X+X^{14}$	'11000'
$1+X^{14}$	'11011'	$1+X^{10}$	'00110'	$1+X^6$	'11110'	X^2+X^{14}	'11110'
$1+X^{13}$	'01100'	$1+X^9$	'11000'	$1+X^5$	'10100'	X^3+X^{13}	'00101'
$1+X^{12}$	'11101'	$1+X^8$	'10111'	$1+X^4$	'10001'	X^4+X^{10}	'10111'

Man beachte in dieser Tabelle, dass beispielsweise die 2 Bit-Fehler $1+X^8$ und X^4+X^{10} die gleichen Reste lassen, also nicht anhand des Rests unterschieden werden können.

6.10 Algorithmische Berechnung des fehlerhaften Bits

Offenbar kann in einer Tabelle zu jedem Rest (mit einer ungeraden Anzahl von Termen) die Position des Bits angegeben werden, an dessen Stelle ein 1-Term diesen Rest lässt. Natürlich könnten auch bei Vorliegen eines Fehlers sämtlich 1-Terme ausprobiert werden. Das wäre allerdings mit

einem Aufwand $O(p^2)$ verbunden, da für jeden 1-Term X^s bis zu s Operationen ausgeführt werden müssten.

Alternativ kann man diesen Wert algorithmisch berechnen, indem die Divisionsoperation umgekehrt wird. Kriterium ist, dass alle Terme verschwinden, außer einem einzelnen. Wir schieben also das Generatorpolynom von rechts nach links und bringen die am weitesten rechts stehenden 1-Terme in Deckung, addieren, usw. bis einmal nur noch ein einzelner Term übrig bleibt. Damit ist der Aufwand durch $O(s \cdot n+1)$ beschränkt, wenn s die Stelle des gesuchten 1-Terms ist und n der Grad des Polynoms.

Beispiel 45

Mit dem Generatorpolynom von Bluetooth $X^5+X^4+X^2+1$ und dem Rest als Bitfolge 01011 erhalten wir:

```
000001011
  110101
000111110
 110101
001010100
 110101
010000000
```

Das Ergebnis ist also X^7. Da dieses exakt die Umkehrung des Polynomdivisiondivisionsverfahren ist und verschiedene 1-Terme auch verschiedene Reste lassen, liefert dieses Verfahren stets eindeutig das korrekte Ergebnis.

6.11 Symmetrien

Man beobachtet leicht, dass Generatoren mit symmetrischem Termaufbau ähnliche Eigenschaften besitzen. Zwei Generatoren $G_1(X)$ und $G_2(X)$ vom Grad n sollen symmetrisch heißen, wenn X^k genau dann in $G_1(X)$ liegt, wenn X^{n-k} in $G_2(X)$ liegt. Daher ist in beiden Fällen die Anzahl der Terme gleich; sollte $n–k=k$ sein, so kommt der mittlere Term (wie ja auch der höchste und tiefste, also $X^0=1$ und X^n) in beiden Generatoren vor.

Beispiele für symmetrische Generatoren sind $1+X+X^2+X^4$ und $1+X^2+X^3+X^4$ (`11101`, `10111`) oder $1+X+X^{14}+X^{16}$ und $1+X^2+X^{15}+X^{16}$ (`1100000000000101`, `1010000000000011`).

Die Eigenschaften eines Generators ergeben sich im wesentlichen aus seinen geraden Bitfiltern, und offensichtlich sind diese bei symmetrischen Generatoren identisch. Ist nämlich $F_1(X)$ ein gerader Bitfilter zu $G_1(X)$ mit Periode p, und $F_2(X)$ ein Bitfilter mit der gleichen Bitfolge in entgegengesetzter Richtung, d.h. X^k genau dann in $F_1(X)$ liegt, wenn X^{p-k} in $F_2(X)$ liegt, dann ist offenbar $F_2(X)$ ein gerader Bitfilter zu $G_2(X)$, da die Terme in $G_2(X)$ entgegengesetzt zu $G_1(X)$ liegen. Daher gibt es die gleichen Bitfilter zu beiden Generatoren, außer dass die Reihenfolge der Bits entgegengesetzt geordnet ist; es kommt jedoch nur auf die Länge der Perioden an, die natürlich in beiden Fällen gleich ist. Daher haben symmetrische Generatoren sämtliche Eigenschaften gemeinsam, soweit sich

diese durch Bitfilter bestimmen lassen (bzw. die Eigenschaften von dem Grad oder der Parität der Generatoren abhängen, die bei symmetrischen Generatoren ja auch gleich sind).

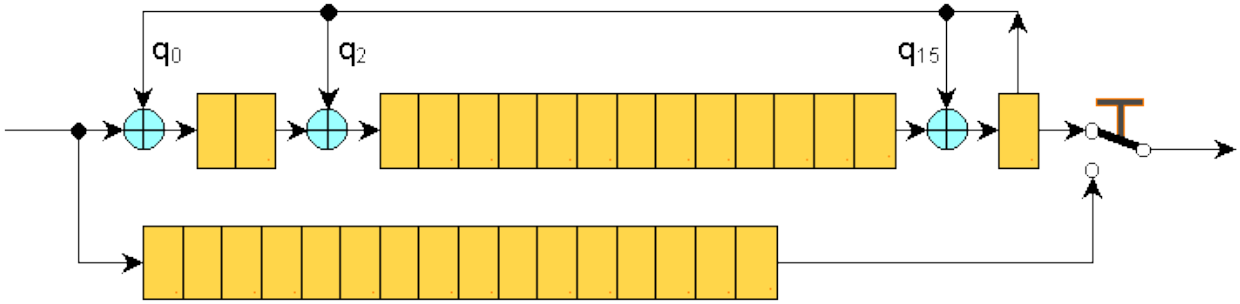
Es folgt, dass zwei symmetrische Generatoren identische Bitfolgen gleichermaßen auf Fehler analysieren und korrigieren können.

Beispiel **46**

Symmetrische Generatoren haben identische Eigenschaften, soweit diese von der Anzahl der Terme, deren Grad oder den Eigenschaften der Bitfilter zu diesen Generatoren abhängen.

7 Implementierung des CRC-Verfahrens

Das CRC-Verfahren kann durch ein einfaches Schieberegister implementiert werden. Dazu wird die Bitfolge $N(X)$ durch n 1 Bit-Speicher geschoben, welche die Bits entweder verändert oder unverändert weiterreichen. Die Änderung hängt davon ab, ob die jeweiligen Bits aufgrund der Terme im Generatorpolynom eine Änderung der entsprechenden Bits bewirken oder nicht. Nachdem das letzte Bit aus diesem Register herauskommt, verbleibt im Register ein Rest, der dem Rest der Polynomdivision von $N(X)$ mit $G(X)$ entspricht.



Schieberegister mit Divisor $Q(X) = X^{16} + X^{15} + X^2 + 1$

8 Zusammenfassung

Dieser Bericht stellt ein neues Verfahren zur Bewertung der Mächtigkeit des CRC-Verfahrens vor. Es konnte gezeigt werden, welche Fehlerklassen durch das CRC-Verfahren erkannt werden, wie gewisse Fehler sogar korrigiert werden können, und wie geeignete Generatorpolynome erzeugt bzw. getestet werden können, um ein optimales Resultat zu erhalten; insbesondere konnte auch gezeigt werden, dass ein Generatorpolynom auch sehr ungeeignet gewählt werden kann.

9 Anhang

9.1 Einige optimale Generatoren

Einige optimale Generatoren (mit 4 Termen) verschiedener Grade.

$1+X+X^2+X^4$, $1+X+X^3+X^5$, $1+X+X^2+X^6$, $1+X+X^5+X^7$, $1+X+X^6+X^8$, $1+X^2+X^6+X^9$, $1+X^2+X^5+X^{10}$, $1+X+X^3+X^{11}$, $1+X^2+X^7+X^{12}$, $1+X+X^2+X^7+X^{12}+X^{13}$, $1+X+X^2+X^{14}$, $1+X^2+X^5+X^{15}$, $1+X+X^{14}+X^{16}$, $1+X+X^{11}+X^{17}$, $1+X+X^{14}+X^{18}$, $1+X+X^{13}+X^{19}$, $1+X+X^{14}+X^{20}$, $1+X+X^{11}+X^{21}$, $1+X+X^{16}+X^{22}$, $1+X+X^{21}+X^{23}$, $1+X+X^{18}+X^{24}$, $1+X^2+X^{14}+X^{25}$, $1+X+X^{22}+X^{26}$, $1+X+X^{15}+X^{27}$, $1+X^2+X^{15}+X^{28}$, $1+X+X^{17}+X^{29}$, $1+X^2+X^{13}+X^{30}$, $1+X^2+X^{12}+X^{31}$, $1+X+X^{28}+X^{32}$.

Für den Grad 13 konnte kein Generator mit vier Termen mit maximaler Bitfilterperiode gefunden werden, so dass wir einen mit sechs Termen aufgenommen haben.

9.2 Programme

Ein Programm zur Bestimmung der maximalen Periode wurde in Java implementiert.

```
/**
* Computes period of the even bitfilter to a generator.
* The generator is a long number, where its degree is the second parameter.
* The generator highest bit is 1L of the generator field, the lowest bit is at position
* 2^(degreeGenerator) in generator field, or as bit pattern: 1L<<(degreeGenerator).
* @param generator Generator of the bitfilter, as specified above.
* @param degreeGenerator Degree of the generator
* @return Returns the period of the longest bitfilter
*/
static long bitfilterPeriod(long generator, long degreeGenerator) {
  long bitMask = 1L<<degreeGenerator;      // mask the bit to compare for even parity
  long mask = ~((-1L)<<(degreeGenerator)); // mask the result, to detect the period
  long period = 1;                         // counts the length of the bitfilter
  long bitfilter = (1L)<<(degreeGenerator-1); // bitfilter's bit pattern
  long startBitfilter = bitfilter;         // start bitfilter, to compare with
  long limit = (1L)<<(degreeGenerator);    // limit loop, if something is wrong

  for(;period<=limit;) { // cycle for all bits in bitfilter
    long m = bitMask; // copy bit mask to count number of bits in generator and bitfilter
    int cnt = 0;      // counter for bit count
    bitfilter<<=1;   // shift bitfilter one position
    long v = bitfilter&generator; // count only bits at same position in
                                  // generator and bitfilter
    while(m!=0) {         // for all bits, selected by bit mask
      if((m&v)!=0) cnt++; // if bit in generator and bitfilter is set, count it!
      m>>=1;              // next bit mask
    }
    if((cnt&1)==1)bitfilter+=1L; // if count is odd set next bit of bitfilter
    if((mask&bitfilter)==startBitfilter) return period; // if bitfilter is the same
                                                        // as start, cycle found!
    period++;             // analyse next bit of bitfilter!
  }
  System.out.println("undefined end; period = "+period);
  return 0;
}
```

```
/**
 * Computes position (0,1,2,...) of a 1 bit error, depending on remainder and generator.
 * Third parameter is degree of generator.
 * Coding is high ending, i.e. bit 0 of long generator holds term X^n of generator etc.
 * e.g. generator = 1+X+X^3+x^5 is coded as binary 110101(base 2), degree=5.
 * @param remainder value of remainder, coded high ending
 * @param generator value of generator, coded high ending, 1+..+X^degree
 * @param genDegree degree of generator
 * @return position of error; count 0,1,2,.. i.e. first bit's count is 0, etc.
 */
 int findErrorPosition(long remainder, long generator, int degree) {
   if(remainder == 0){out("No error found?\n"); return 0;}
   int count = Long.bitCount(remainder); // number of 1's
   if(count==1){
     int position = degree-1-Long.numberOfTrailingZeros(remainder); // pos of 1
     out("Error found at "+position+"!\n");
     return position;
   } else if((count&1L)==0) {
     out("Even number of errors detected!\n");
     return -1; }
   long mask = 1L<<degree; // mask Generators' 1-term
   remainder<<=1;          // matching 1's of gen and rem
   long end = (mask>>1)-1; // = 2^(degree-1)-1
   while(degree<=end) {    // repeat until maximum bitfilter cycle
     if((remainder&mask)!=0L) remainder^=generator; // is rem 1 at gen's 1: xor
     if(remainder==1L) break; // only 1 bit left, error position found
     remainder<<=1;         // shift remainder to left
     degree++;              // set position of generator's highest term
   }
   out("Error found at "+degree+"!\n");
   return degree;
 }
```