Dr. Jürg M. Stettbacher
Neugutstrasse 54
CH-8600 Dübendorf

Telefon: +41 43 299 57 23
Fax: +41 43 299 57 25
E-Mail: dsp@stettbacher.ch

CRC

Zyklische Redundanzprüfung

Version 1.24
2015-09-11

Zusammenfassung: Das Dokument erläutert die Grundlagen des CRC-Verfahrens sowie der mathematischen Werkzeuge. Insbesondere werden die Codierung und Decodierung anhand von Beispielen im Detail erklärt, ausserdem wird auf Grenzen hingewiesen.

Inhaltsverzeichnis

1 Zweck

Dieses Dokument verfolgt die nachstehenden Ziele:

- Einführen des CRC-Verfahrens und dessen Verwendung.
- Erklären der Polynom-Darstellung von binären Vektoren.
- Aufzeigen, wie CRC-Prüfbits berechnet werden.
- Aufzeigen, wie die CRC-Prüfung ausgeführt wird.
- Beschreiben der Grenzen des CRC-Verfahrens.

2 Einleitung

CRC (Cyclic Redundancy Check, zyklische Redundanzprüfung) ist eine Art der Kanalcodierung. Das heisst, es wird bei der Übertragung von Daten zum Schutz von Übertragungsfehlern verwendet. Im Grunde ist CRC ein Blockcode: Aus jeweils K Nutzbits, werden P Prüfbits berechnet, welche an die Nutzbits angehängt werden. Das zu übertragende Datenwort besteht somit aus $N = K + P$ Bits.

Die Idee besteht darin, dass das N-stellige CRC-Codewort 2^N verschiedene Bitmuster annehmen kann, davon aber nur 2^K Möglichkeiten benutzt werden. Tritt bei einer Übertragung ein Fehler auf und erhält der Empfänger ein Bitmuster, das keinem gültigen Codewort entspricht, so erkennt er daran den Übertragungsfehler.

Ein vergleichbares Verfahren besteht in der Verwendung eines *Parity-Bits*. Hier ist $P = 1$ und das Parity-Bit sorgt entweder dafür, dass die Anzahl Einsen im gesamten Datenwort gerade oder ungerade wird. Falls bei der Übertragung ein Bit des Codeworts verfälscht wird, so schlägt auf der Empfängerseite der Parity-Check fehl. Dagegen können zwei Bitfehler pro Codewort auf diese Weise nicht erkannt werden. Generell werden alle ungeradzahligen Bitfehler pro Codewort mit dem Parity-Check erkannt, alle geradzahligen bleiben unerkannt.

Der Trick von CRC besteht nun darin, dass die Prüfbits so gewählt werden, dass möglichst viele typische Arten von Übertragungsfehlern erkannt werden. In dieser Hinsicht ist CRC besonders erfolgreich und wird entsprechend oft eingesetzt, zum Beispiel bei Ethernet, USB und einer ganzen Anzahl weiterer Standards der Datenübertragung. CRC-Codes sind linear und zyklisch.

3 Grundlagen

Die Berechnung von CRC-Prüfbits kann mathematisch beschrieben werden. Dabei werden die binären Datenworte nicht mehr als Bit-Vektoren aufgefasst, sondern als Polynome einer beliebigen Variable z. Die einzelnen Bits entspechen gerade den Koeffizienten des Polynoms. Zum Beispiel entspricht bei dieser Betrachtung das Datenwort $\underline{u} = (101001)$ dem Polynom $U(z)$:

$$\begin{aligned} U(z) &= 1 \cdot z^5 + 0 \cdot z^4 + 1 \cdot z^3 + 0 \cdot z^2 + 0 \cdot z^1 + 1 \cdot z^0 \\ &= z^5 + z^3 + 1 \end{aligned}$$

Beachte, dass die Koeffizienten immer nur null oder eins sein können. Der Vorteil besteht darin, dass die Polynomrechnung deutlich einfacher und schneller durchzuführen ist als beispielsweise die Matrizenrechnung. Häufig schreiben wir statt $U(z)$ einfach nur U mit einem Grossbuchstaben.

3.1 Binäre Polynom-Arithmetik

Da bei der Polynomdarstellung von binären Datenworten die Koeffizienten nur die Werte null oder eins annehmen können, wird die sogenannte MOD-2 Arithmetik[1] mit nur einem Bit eingesetzt. Wir nennen dies manchmal auch eine 1-Bit Arithmetik.

Addition und Subtraktion Die Vorstellung, eine Zahlengerade auf einen Zahlenkreis abzubilden, hilft im Folgenden, um die MOD-2 Arithmetik[2] zu verstehen. Auf diesem Zahlenkreis befinden sich bei der 1-Bit Arithmetik nur die zwei Elemente 0 und 1. Eine Addition entspricht einer Bewegung auf diesem Zahlenkreis im Uhrzeigersinn. Von der 0 gelangt man zur 1, dann wieder zur 0, usw. Analog entspricht der Subtraktion eine Bewegung gegen den Uhrzeigersinn. Von der 0 gelangt man wiederum zur 1, dann (so wie bereits bei der Addition) zurück zur 0, usw. In dieser speziellen Arithmetik sind demnach Subtraktion und Addition identisch. Die Tabelle 1 zeigt aus diesem Grund nur eine Wahrheitstabelle für beide Operationen. Es ist ersichtlich, dass die Addition und die Subtraktion beide der XOR-Verknüpfung entsprechen.

Beispiel:

$$\begin{aligned} \left(z^3 + z^2 + 1\right) \pm \left(z^2 + z + 1\right) &= z^3 \cdot (1 \pm 0) + z^2 \cdot (1 \pm 1) + z^1 \cdot (0 \pm 1) + z^0 \cdot (1 \pm 1) \\ &= z^3 + z \end{aligned}$$

[1] MOD-2 steht im Folgenden für die Modulo-2 Operation, die den Rest der Ganzzahl-Division mit 2 ergibt.

[2] Unter *Arithmetik* versteht man die Zahlenlehre, vor allem die Lehre der natürlichen Zahlen und dem Rechnen damit. Sie ist damit ein Teilgebiet der Algebra.

a	b	$a \pm b$
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 1: Wahrheitstabelle der 1-Bit Addition und der 1-Bit Subtraktion.

Multiplikation und Division Analog wird die Multiplikation von Polynomen ausgeführt:

$$\begin{aligned}
\left(z^2+z+1\right)\cdot\left(z^2+z\right) &= z^4+z^3+z^3+z^2+z^2+z \\
&= z^4+\left(z^3+z^3\right)+\left(z^2+z^2\right)+z \\
&= z^4+z^3\left(1+1\right)+z^2\left(1+1\right)+z \\
&= z^4+z
\end{aligned}$$

Die Wahrheitstabelle (Tabelle 2) der 1-Bit Multiplikation entspricht einer AND-Verknüpfung.

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

Tabelle 2: Wahrheitstabelle der 1-Bit Multiplikation.

Mathematiker beschreiben die Zusammenhänge so: Unsere gewohnte Algebra[3] ist (streng genommen) definiert für Zahlenkörper. Ein Zahlenkörper ist eine Menge von Zahlen, die gewisse Eigenschaften erfüllen:

- Ein Körper ist abgeschlossen bezüglich der Addition. Das heisst, die Summe zweier Zahlen aus dem Körper ist ebenfalls ein Element des Körpers.
- Der Körper ist abgeschlossen bezüglich der Multiplikation.

[3] Unter einer Algebra versteht man grob einen Satz von Rechenregeln, die in gewissen Mengen, zum Beispiel in Zahlmengen oder in Vektorräumen Gültigkeit haben.

- Es existiert ein Neutralelement für die Addition. Gemeint ist, dass die Summe einer Zahl aus dem Körper mit dem Neutralelement die ursprüngliche Zahl ergibt. In unserem Fall ist 0 das Neutralelement der Addition.
- Es existiert ein Neutralelement für die Multiplikation. Bei uns ist das die Zahl 1.
- Es gilt das Distributivgesetz: $a \cdot (b+c) = a \cdot b + a \cdot c$.
- Die Division muss definiert sein.

Typische Beispiele von Zahlenkörpern sind die Menge der reellen Zahlen $\mathbb{R}$ und die Menge der komplexen Zahlen $\mathbb{C}$. Diese Körper umfassen unendlich viele Elemente, sonst wären sie nicht abgeschlossen. Sie heissen darum unendliche Körper. Für alle Körper gilt dieselbe Algebra, die Rechenregeln sind darum identisch für die reellen und die komplexen Zahlen. Mit binären Zahlen der Wortlänge n lassen sich jedoch nur 2^n verschiedene Wörter bilden. Mit solchen Datenworten lässt sich kein unendlicher Körper bilden. Ein Trick schafft Abhilfe: Anstelle einer Zahlengerade verwendet man einen Zahlenkreis.

Beispiel: In einem Körper mit den Elementen 0 bis 6 wird $4+5=2$.

Das ist nichts anderes als die Modulo-7 Arithmetik (MOD-7), wobei 7 die Anzahl Elemente im endlichen Zahlenkörper ist. Allerdings können nicht endliche Zahlenkörper mit einer beliebigen Anzahl von Elementen gebildet werden, ohne die obigen Regeln zu verletzen. Der französische Mathematiker Évariste Galois (1811 bis 1832) hat bewiesen, dass die Anzahl Elemente in einem endlichen Raum eine Primzahl p oder eine Primzahlpotenz p^n sein muss, wobei n eine natürliche Zahl ist. Man spricht darum von *Galois-Körpern* $\mathrm{GF}(p)$ bzw. $\mathrm{GF}(p^n)$. In der englischen Sprache heisst ein Zahlenkörper *Field*, einen Galoiskörper nennt man *Galois Field*, abgekürzt GF. Auch in der deutschsprachigen Literatur trifft man häufig den Ausdruck Galois-Feld an.

Für eine konkrete Anwendung muss man p und n festlegen. In unserem Fall ist $p=2$ und $n=1$, so dass wir vom Galois-Körper $\mathrm{GF}(2)$ sprechen, der die Elemente 0 und 1 umfasst. Diese sind gleichzeitig die Neutralelemente sowie die logischen Werte, die ein Bit annehmen kann. Ein Galois-Körper $\mathrm{GF}(2^n)$ umfasst 2^n Elemente, also wird n gerade gleich der Länge (Stellenzahl) eines entsprechenden binären Vektors gesetzt. Der geniale Trick bei der Anwendung der Galois-Felder liegt also darin, dass man die bekannte Algebra anwendet auf einen Zahlenkörper der massgeschneidert ist für unsere digitalen Codeworte.

Interessant sind noch die Folgen für die Division. Es sei daran erinnert, dass in $\mathrm{GF}(2)$ alle Operationen bitweise betrachtet werden können. Die Berechnung der Division von z^3durch (z^3+z) führt demnach nicht zu null, sondern zu 1 mit einem Rest.

```
1000 : 1010 = 1
1010
----
  10    <== Rest P
```

4 Polynome für CRC

Anstelle einer Generatormatrix wie bei den Block-Codes, wird für CRC ein Generatorpolynom $G(z)$ verwendet, um aus dem Datenwort $\underline{u}$ das Codewort $\underline{c}$ zu erzeugen. Die Idee ist die, dass jedes Codewort ein Vielfaches (eine Linearkombination) des Generatorpolynoms $G(z)$ ist. Dadurch sind derartige Code automatisch zyklisch und linear. Die Tabelle 3 verdeutlicht dies anhand eines Beispiels.

0	0	1	0	0	1	0	1	$G(z)$ ist ein gültiges Codewort.
1	0	0	1	0	1	0	0	Permutation ergibt ein anderes Codewort (zyklischer Code).
1	0	1	1	0	0	0	1	Die Summe beider Codeworte ist ein Codewort (linearer Code).

Tabelle 3: Linearer zyklischer Code.

4.1 Beispiele

Die Wahl von $G(z)$ ist entscheidend für die entstehenden Codewort und deren Eigenschaften. Tabelle 4 zeigt eine Auswahl von gängigen Generatorpolynomen. Häufig verwendet werden CRC-32, CRC-16(-IBM) und CRC-16-CCITT.

4.2 Eigenschaften

CRC-16 Das CRC-16-IBM-Polynom hat im wesentlichen dieselben Eigenschaften wie das Polynom CRC-16-CCITT. Unter der Voraussetzung, dass der Datenblock nicht länger ist als 4095 Byte gilt:

- Alle 1-Bit Fehler werden erkannt.
- Alle Fehler mit einer ungeraden Anzahl von Bitfehlern werden erkannt.
- Alle 2-Bit Fehler werden erkannt.
- Alle Fehlerbursts, die nicht länger als 16 Bits sind, werden erkannt.
- Für Fehlerburts der Länge 17 Bits oder mehr liegt die Wahrscheinlichkeit für das Erkennen bei über 99.997%.

Name	Polynom G(z)	Bemerkungen
CRC-1	$z+1$	Parity-Bit (gerade)
CRC-4	z^4+z+1	Identisch zu (15,11)-Hamming-Code
CRC-5-USB	z^5+z^2+1	Identisch zu (31,26)-Hamming-Code
CRC-5-ITU	$z^5+z^4+z^2+1$	
CRC-7	z^7+z^3+1	Identisch zu (127,120)-Hamming-Code
CRC-8-CCITT	z^8+z^2+z+1	
CRC-12	$z^{12}+z^{11}+z^3+z^2+z+1$	
CRC-16-CCITT	$z^{16}+z^{12}+z^5+1$	Verwendet für X.25, SD, Bluetooth.
CRC-16-IBM	$z^{16}+z^{15}+z^2+1$	Verwendet z.B. für Modbus, USB
CRC-32	$z^{32}+z^{26}+z^{23}+z^{22}+z^{16}+z^{12}+z^{11}$ $+z^{10}+z^8+z^7+z^5+z^4+z^2+z+1$	Verwendet für Ethernet, ZIP, PNG, SATA, MPEG-2, ZMODEM
CRC-64-ISO	$z^{64}+z^4+z^3+z+1$	

Tabelle 4: Generatorpolynome für CRC.

CRC-32 Das Generatorpolynom hat eine ungerade Anzahl von Termen und kann somit nicht alle ungeraden Bitfehler erkennen (die Begründung folgt weiter unten).

Die Wahrscheinlichkeit, einen Fehler nicht zu entdecken, hängt auch von den verwendeten Daten ab, sowie von der Fehlercharakteristik des verwendeten Kanals. Empirische Untersuchungen für gute Fehlercodes benötigen normalerweise sehr große Stichproben. In einem Beispiel wurden bei 157'000 Blöcken der Länge 2048 Bits keine unentdeckten Fehler beobachtet. Eine Berechnung der Wahrscheinlichkeit für unentdeckte Fehler ergab, dass diese kleiner als 10^{-14} ist.

CRC-8 CRC-8 ist ein Prüfpolynom, welches von der CCITT in der Empfehlung I.432 für die Prüfung des vier Byte langen Headers einer ATM-Zelle vorgeschlagen wird. Mit der angehängten Prüfsumme wird der Header fünf Byte lang. Man kann zeigen, dass sämtliche 1, 2 und 3-Bit Fehler in dem fünf Byte langen Header mit Sicherheit erkannt werden.

5 Verfahren

Es wird nun gezeigt, wie die Codeworte erzeugt und überprüft werden.

5.1 Senderseite

Mit dem Polynom $U(z)$ der Nutzbits und dem Generatorpolynom $G(z)$ von Grad p lässt sich das Verfahren wie folgt beschreiben. U ist um p Stellen nach links zu schieben und mit Nullen aufzufüllen. Anschliessend wird die Polynomdivision mit dem Generatorpolynom G durchgeführt, wobei der Rest der Division den p Prüfbits P entspricht. Mathematisch ausgedrückt:

$$\frac{z^p \cdot U}{G} = Q + \frac{P}{G} \tag{1}$$

Die Multiplikation von U mit z^p entspricht dem Schieben um die p Stellen nach links.

Das Polynom für das Codewort C wird gebildet, indem das Prüfbit-Polynom P zu den Nutzdaten U addiert wird. Dadurch werden die zuvor angehängten Nullen überschrieben:

$$C = z^p \cdot U + P \tag{2}$$

Beispiel: Die Folge der zu übertragenden Bits sei $\underline{u} = (100101101)$. Dies entspricht somit dem Polynom $U = z^8 + z^5 + z^3 + z^2 + 1$. Als Generatorpolynom wird das CRC-5-ITU Polynom $G = z^5 + z^4 + z^2 + 1$ gewählt (siehe Tabelle 4). Der Grad ist $p = 5$, die Anzahl der Koeffizienten somit 6. Die Polynom-Division wird wie gewohnt durchgeführt. Der Einfachheit halber schreiben wir statt den Polynomen aber nur die Koeffizienten. Alle Operationen verstehen sich als MOD-2 Arithmetik. Das heisst, es gibt keine Überträge (Carry). Die Subtraktion ist identisch zur Addition und entspricht genau einer bitweisen XOR-Verknüpfung.

```
10010110100000 : 110101 = 111101010
110101
------
 100001
 110101
 ------
  101000
  110101
  ------
   111011
   110101
   ------
     111000
     110101
     ------
       110100
       110101
       ------
            10    <== Rest P
```

Der Grad des Prüfpolynoms muss p entsprechen. Ist der Rest (wie in diesem Beispiel) zu kurz, so muss also links mit Nullen aufgefüllt werden. Der berechnete Rest (die gesuchte Prüfsumme) ergibt sich zu $\underline{p} = (00010)$ und das generierte Codewort wird demnach $\underline{c} = (10010110100010)$.

5.2 Übertragung

Bei der Übertragung können einzelne Bits verfälscht werden. Der Empfänger wird daher nicht mehr das korrekte Codewort $\underline{c}$ empfangen, sondern eine allenfalls modifizierte Form davon. Wir bezeichnen das empfangene, möglicherweise fehlerhaft Codewort mit $\underline{\widetilde{c}}$. Den Fehlervektor nennen wir $\underline{e}$. Er hat die selbe Länge wie $\underline{c}$ und $\underline{\widetilde{c}}$ und hat eine 1 an jenen Stellen, wo ein Fehler vorliegt. Falls $\underline{e} = (00\ldots0)$, so ist kein Fehler aufgetreten. Unter Verwendung des Fehlervektors $\underline{e}$ können wir schreiben:

$$\underline{\widetilde{c}} = \underline{c} + \underline{e}$$

Oder in Polynom-Schreibweise:

$$\widetilde{C}(z) = C(z) + E(z)$$

5.3 Empfängerseite

Der Empfänger nimmt das empfangene Wort $\widetilde{C}$ und dividiert es durch G. Da erwartet wird, dass $\widetilde{C}$ ein Vielfaches von G sei, sollte die Division keinen Rest ergeben. Dass C ein Vielfaches von G ist, sieht man auch daran, dass der Rest der Division U/G im Sender zu $z^p \cdot U$ addiert (resp. subtrahiert) wurde. Entsteht trotzdem ein Rest, so ist ein Übertragungsfehler aufgetreten. Im anderen Fall nehmen wir an, die Übertragung sei korrekt verlaufen. Allerdings kann der äusserst seltene Fall auftreten, wo die Division trotz fehlerhaften Daten $\widetilde{C}$ ohne Rest duch G teilbar ist. In diesem Fall entsprechen die fehlerhaften Nutzdaten zufällig gerade wieder einem Vielfachen von G.

```
10010110100010 : 110101 = 111101010
110101
------
 100001
 110101
 ------
  101000
  110101
  ------
   111011
   110101
   ------
     111000
     110101
     ------
       110101
       110101
       ------
            00    <== Rest P
```

5.4 Fehlererkennung

Ist das CRC-Polynom gut gewählt, können mit dem CRC-Verfahren alle Einbitfehler, jede ungerade Anzahl von verfälschten Bits, sowie alle Burstfehler erkannt werden, deren Länge den Grad des CRC-

Polynoms nicht übersteigen. Zudem werden alle Fehler erkannt, deren Polynomdarstellung einen kleineren Grad als das CRC-Polynom hat. Dies wird in den folgenden Abschnitten gezeigt. Dazu verwenden wir ein einfaches Beispiel (kurze Polynome):

$$\begin{aligned} \underline{u} &= (11001) \\ \underline{g} &= (110101) \\ \underline{p} &= (11010) \\ \underline{c} &= (1100111010) \end{aligned}$$

Ferner gilt:

$G(z)$	CRC- oder Generatorpolynom
$C(z)$	Codewort, Datenwort plus Prüfbits, als Polynom
$E(z)$	Fehlerpolynom

Der Empfänger wertet $\widetilde{C}(z)$ aus, bzw. die Summe von $C(z)$ und $E(z)$:

$$\frac{C(z)+E(z)}{G(z)} = \frac{C(z)}{G(z)} + \frac{E(z)}{G(z)} \tag{3}$$

Um das empfangene Codewort zu beurteilen (fehlerhaft oder nicht) wird der Rest aus Formel 3 betrachtet. Der erste Term geht per Definition ohne Rest auf. Somit stammt ein allfälliger Rest nur vom zweiten Term mit dem Fehlerpolynom.

Fehlerfreie Übertragung Bei einer fehlerfreien Übertragung ergibt die Division von $(C(z)+E(z))/G(z)$ null:

```
1100111010 : 110101 = 10010
110101
------
   110101
   110101
   ------
        00    <== Rest P
```

Eine fehlerfreie Übertragung hat stattgefunden, wenn der Rest der Division verschwindet.

Ein Fehler in den Nutzdaten Es sei das Bit 1 der Nutzbits $\underline{u}$ falsch übertragen worden (es entspricht dem Bit 6 des Datenpaketes $\widetilde{c}$). Im folgenden Beispiel ist das fehlerhafte Bit mit einem Punkt darüber markiert:

```
    .
1101111010 : 110101 = 10001
110101
------
    101010
    110101
    ------
     11111    <== Rest P
```

Durch den Fehler ergibt sich bei der Division ein Restterm.

Ein Fehler in der Prüfsumme Es sei das Bit 1 der Prüfsumme $\underline{p}$ fehlerhaft:

```
         .
1100111000 : 110101 = 10010
110101
------
   110100
   110101
   ------
        10    <== Rest P
```

Wieder zeigt ein Rest an, dass die Übertragung fehlerhaft war.

Wenn ein 1-Bit Fehler aufgetreten ist, gilt $E(z) = z^q$, wobei q bestimmt, welches Bit invertiert ist. Wenn nun $G(z)$ zwei oder mehr Terme enthält, kann die Division von $E(z)/G(z)$ niemals aufgehen (siehe Formel 3), denn jedes Vielfache von $G(z)$ kann durch Shift- und Addition-Operationen gebildet werden. Auf diese Weise ist es unmöglich einen Wert mit nur einer gesetzten Eins zu erzeugen. Dies ist für alle Polynome aus Tabelle 4 der Fall. Das heisst, dass alle aufgelisteten CRC-Polynome 1-Bit Fehler (unabhängig von deren Position) erkennen können.

Ungerade Anzahl von Fehlern Bei einer ungeraden Anzahl von Fehlern enthält $E(z)$ eine ungerade Anzahl von Termen. Wählt man das CRC-Polynom so, dass es $(z+1)$ als Faktor hat, werden alle Fehler mit einer ungeraden Anzahl von verfälschten Bits erkannt. Dies ist auch der Fall, wenn das Generatorpolynom eine gerade Anzahl von Koeffizienten enthält.

Zu betrachten sind also die zwei Fälle:

- $G(z)$ enthält $(z+1)$ als Faktor.
- $G(z)$ hat gerade Parität.

▷ Erklärung für den 1. Fall: Ein Fehler wird nicht erkannt, wenn $E(z)$ ein Vielfaches von $G(z)$ ist. In diesem Fall hiesse das, ein Fehler wird nicht erkannt, wenn $E(z)$ den Faktor $(z+1)$ enthält. Für den Beweis gehen wir von der folgenden Annahme aus: $E(z)$ hat ungerade Parität (eine ungerade Anzahl Fehler) und $(z+1)$ als Faktor. (Die Annahme impliziert, dass ein Bitfehler nicht als Fehler erkannt würde.) Damit folgt:

$$E(z) = (z+1) \cdot T(z)$$

Nun wählen wir $z = 1$. Wegen der ungeraden Parität von $E(z)$ wird $E(1) = 1$. Weiter folgt:

$$E(z) = (z+1) \cdot T(z) = (1+1) \cdot T(z) = 0$$

Es resultiert ein Widerspruch. Folglich ist die oben getroffene Annahme falsch. Wenn $E(z)$ eine ungerade Parität hat, kann es den Faktor $(z+1)$ nicht enthalten.

▷ Erklärung für den 2. Fall: Da das Codewort eine Linearkombination von $G(z)$ ist, hat ein gültiges Codewort mit einem Generatorpolynom gerader Parität ebenfalls gerade Parität. Ein Fehler ungerader Parität führt zu einem verfälschten Codewort, das somit ungerade Parität bekommt. Der Fehler wird also erkannt. In Tabelle 4 weisen nicht alle Generatorpolynome diese Eigenschaft auf (z. B. das CRC-4 Polynom). Interessant sind vor allem die Polynome mit ungerader Anzahl von Koeffizienten. Dazu mehr folgt im Kapitel 5.4.

Burstfehler Als Beispiel seien in $\widetilde{\underline{c}}$ die Bits 4 bis 6 fehlerhaft:

```
   ...
1101001010 : 110101 = 10000
110101
------
     11010    <== Rest P
```

Wieder zeigt der Rest, dass die Übertragung fehlerhaft war. Ein Burstfehler wird erkannt wenn die Länge des Bursts b den Grad p des CRC-Polynoms nicht übersteigt.

Erklärung: Ein Burstfehler der Länge $b \leq p$ lässt sich schreiben als $E(z) = z^q \cdot E_0(z)$ mit $E_0(z) = \left(z^{b-1} + ... + 1\right)$, wobei q bestimmt, an welcher Stelle im Codewort der Burstfehler liegt. Der Fehlervektor $\underline{e}$ ist also von der Form von (0...01...10...0). Wenn der Fehler erkannt werden soll, muss die Division von $E(z) = z^q \cdot E_0(z)$ durch $G(z)$ einen Rest ergeben. Da gute $G(z)$ immer den Term $z^0 = 1$ enthalten (siehe Tabelle 4), hat $G(z)$ den Term z^q sicher nicht als Faktor. Das heißt, wenn $G(z)$ ein Teiler des Fehlerterms $E(z)$ ist, dann muss $G(z)$ zwangsläufig ein Teiler sein von $E_0(z)$. Dies ist jedoch nicht möglich, da per Annahme der Grad von $E_0(z)$ kleiner ist (nämlich $b-1$) als der Grad p von $G(z)$. Der Rest kann somit niemals null sein und der Burstfehler wird erkannt.

Zwei isolierte Einbit-Fehler Ob zwei isolierte Einbit-Fehler erkannt werden ist nicht nur vom Generatorpolynom sonderen auch von der Länge der Nachricht abhängig.

Erklärung: Sind zwei isolierte 1-Bit Fehler aufgetreten, gilt $E(z) = z^q + z^r$, wobei $q > r+1$. Klammert man z^r aus, lässt sich dies auch als $E(z) = z^r \cdot (z^{q-r} + 1)$ schreiben. Da $G(z)$ nicht durch z teilbar ist (der kleinste Term von $G(z)$ ist z^0), reicht es zu fordern, dass $G(z)$ nicht $\left(z^b + 1\right)$ teilt (für alle b bis zum maximalen Wert von $(q-r)$, das heisst der maximalen Rahmenlänge). Für $G(z)$ sind also Vielfache von (11), (101), (1001), (10...01) zu vermeiden. Einfache Polynome geringen Grades, die eine sichere Übertragung für lange Rahmen ermöglichen, sind bekannt. Zum Beispiel teilt $G_{15} =$

$\left(z^{15}+z^{14}+1\right)$ den Term $\left(z^b+1\right)$ nicht für jedes $b < 32767$. Der CRC mit dem Generatorpolynom G_{15} kann also alle 2-Bit Fehler erkennen, wenn das zu übertragende Frame nicht länger ist als 32767 Bits.

Pech Betrachte den folgenden Fall mit sechs Fehlern:

```
   . .....
1101100101 : 110101 = 10010
110101
------
    110101
    110101
    ------
         0    <== Rest P
```

Obwohl die Division keinen Rest ergibt war die Übertragung fehlerhaft. Die fehlerhaften Bits finden sich im Nutz- und im Prüfbereich. Vom Codewort C wurden nur die Bits 5, 7, 8 und 9 korrekt übertragen. Jedes Fehlerpolynom $E(z)$, das ein Vielfaches ist von $G(z)$ wird zu einer Division ohne Rest führen.

Ebenfalls Pech kann man haben, wenn das Generatorpolynom eine ungerade Anzahl von Koeffizienten hat und den Faktor $(z+1)$ nicht enthält. So kann nicht mehr jede ungerade Anzahl von Bitfehlern erkannt werden (siehe Kapitel 5.4). Das ist u. a. beim CRC-4 und beim CRC-5-USB Polynom der Fall (siehe Tabelle 4). In beiden Fällen werden 3-Bit Fehler in der Form $E(z) = z^q \cdot G(z)$ nicht erkannt.

5.5 Zusammenfassung

1. Wenn das empfangene Bitmuster $\widetilde{C}(z)$ ohne Rest durch das Generatorpolynom $G(z)$ dividierbar ist, wird eine fehlerfreie Übertragung angenommen.
2. Ein geeignetes[4] Generatorpolynom $G(z)$ erkennt sämtliche Burstfehler, die nicht länger sind als der Grad p des Generatorpolynoms. Das beinhaltet natürlich auch 1-Bit Fehler als Burstfehler der Länge eins. Wenn das Generatorpolynom also mehr als einen Term enthält, werden alle 1-Bit Fehler erkannt. Ausserdem erkennt das Polynom alle Burstfehler des Grads $p+1$, ausser jenem, wo das Fehlermuster dem Polynom entspricht.
3. Ein Generatorpolynom mit gerader Anzahl von Termen erkennt jede ungerade Anzahl von Bitfehlern. Ebenso ist dies der Fall, wenn der Faktor $(z+1)$ im Generatorpolynom enthalten ist.
4. Man kann ferner zeigen, dass 2-Bit Fehler nur erkannt werden, wenn eine gewisse Nachrichtenlänge nicht überschritten wird. Bei optimal gewählten Generatorpolynomen vom Grad p mit gerader Anzahl von Termen ist diese Länge $L_{Max} = 2^{p-1} - 1$. Ist beispielsweise $p = 16$ beträgt diese Länge immerhin 32767, also mehr als 4000 Bytes.
5. Entspricht ein Fehlerpolynom $E(z)$ einem beliebigen Vielfachem des Generatorpolynoms $G(z)$, kann der Empfänger im Bitmuster $\widetilde{C}(z)$ keinen Fehler erkennen.

[4] Geeignet meint, dass das Polynom keinen Faktor z enthält, resp. dass es den Term $+1$ enthält.

6 Fehlerkorrektur

Es kam bisher nur darauf an, Fehler zu erkennen. Dies ist auch die Hauptanwendung von CRC. Es stellt sich aber die Frage, ob das CRC-Verfahren darüber hinaus auch für die Fehlerkorrektur geeignet wäre. Lässt sich aus dem Rest der Division eventuell auf den aufgetretenen Fehler schließen?

Damit Fehler korrigiert werden können, muss aus dem Rest der Polynomdivision eindeutig auf einen (oder ggf. mehrere) Fehler geschlossen werden können. Wir wollen hier nur einfache Bitfehler untersuchen, d. h. $E(z) = z^q$. Wenn jetzt der Rest aus der Division $E(z)/G(z)$ für jedes q eindeutig ist, so können wir aus dem Rest eindeutig auf den Fehler z^q schließen und dieses falsche Bit natürlich korrigieren. Man kann zeigen, dass alle 1-Bit Fehler korrigiert werden können, wenn der Datenblock (das Codewort) nicht länger ist als eine gewisse kritische Länge L_{krit}.

Anhand eines Beispiels soll dies verdeutlicht werden: Das CRC-5-ITU Generatorpolynom hat die bereits bekannte Polynomdarstellung $G(z) = z^5 + z^4 + z^2 + 1$, resp. $\underline{g} = (110101)$, siehe Tabelle 4. Bei diesem Generatorpolynom ist $L_{krit} = 15$. Somit können 1-Bit Fehler erkannt und eindeutig einem bestimmten Bit zugeordnet werden, wenn das empfangene Bitmuster $\widetilde{C}(z)$ nicht länger ist als 15 Bit. Die Tabelle 5 zeigt für jede Fehlerposition den eindeutigen Rest und zwar unabhängig vom Codewort, das übertragen wurde.

Fehler bei Bit	Rest
–	00000
0	00001
1	00010
2	00100
3	01000
4	10000
5	10101
6	11111
7	01011

Fehler bei Bit	Rest
8	10110
9	11001
10	00111
11	01110
12	11100
13	01101
14	11010
15	00001
16	00010

Tabelle 5: Der Rest bei einem 1-Bit Fehler mit dem CRC-5-ITU Polynom.

Beispiel: $\underline{u} = (111) \implies \underline{c} = (11100001)$. Mit einem 1-Bit Fehler an der Position 6 folgt:

```
 .
10100001 : 110101 = 110
110101
------
```

```
111010
110101
------
  11111    <== Rest P
```

7 Implementierung

Weiter oben wurde gezeigt, dass in GF(2) die Addition der XOR-Verknüpfung entspricht und die Multiplikation der AND-Verknüpfung (siehe die Tabellen 1 und 2). Wir brauchen in erster Linie Additionen, darum lässt sich das CRC-Verfahren mit XOR-Verknüpfung und Schiebeoperationen in Hardware oder auch in Software realisieren.

Als Beispiel betrachten wir einen CRC-5-ITU Decoder mit $\underline{g} = (110101)$, der das empfangene Bitmuster $\underline{\widetilde{c}} = (1011001101010)$ auf Übertragungsfehler überprüfen soll. Dazu benötigen wir:

- Ein Schieberegister mit $p+1$ Zellen (6 Stück bei CRC-5).
- Total $p+1 = 6$ XOR-Gatter[5].

Die Polynom-Division $\widetilde{C}(z)/G(z)$ wird nun folgendermassen ausgeführt: Es wird die Bitfolge $\underline{\widetilde{c}}$ schrittweise von rechts nach links durch die $p+1$ Speicherzellen geschoben. Nach jedem Schritt wird geprüft, ob das Bit ganz links eine 1 ist. Falls das zutrifft, so wird vom Inhalt des Schieberegisters das Generatorpolynom subtrahiert. Dazu werden die $p+1$ XOR-Gatter verwendet. Ist das Bit ganz links eine 0, so bleibt der Inhalt des Schieberegisters unverändert. Sobald die Division fertig ausgeführt ist (alle Bits von $\underline{\widetilde{c}}$ wurden ins Register geschoben), steht der gesuchte Rest im Schieberegister. Betrachte die folgenden Darstellung (SR steht für Schieberegister, MSB steht für das Bit ganz links im SR) und beachte auch, dass das Resultat der Division gar nicht gespeichert wird, da wir nur am Rest interessiert sind:

```
   SR       Bitmuster       Kommentar
-------------------------------------------------------------
 000000 <-- 1011001101010   Ausgangslage, SR ist leer.
 000001 <-- 011001101010    Erstes Bit ins SR geschoben,   MSB = 0.
 000010 <-- 11001101010     Nächstes Bit ins SR geschoben, MSB = 0.
 000101 <-- 1001101010      Nächstes Bit ins SR geschoben, MSB = 0.
 001011 <-- 001101010       Nächstes Bit ins SR geschoben, MSB = 0.
 010110 <-- 01101010        Nächstes Bit ins SR geschoben, MSB = 0.
 101100 <-- 1101010         Nächstes Bit ins SR geschoben, MSB = 1.
 011001 <-- 1101010         XOR mit G = (110101) auf SR angewendet.
 110011 <-- 101010          Nächstes Bit ins SR geschoben, MSB = 1.
 000110 <-- 101010          XOR mit G = (110101) auf SR angewendet.
 001101 <-- 01010           Nächstes Bit ins SR geschoben, MSB = 0.
```

[5] Im Grunde reichen p Zellen und p XOR-Gatter. Aus didaktischen Gründen zeigen wir das Verfahren mit $p+1$ Zellen.

```
011010 <-- 1010          Nächstes Bit ins SR geschoben, MSB = 0.
110101 <-- 010           Nächstes Bit ins SR geschoben, MSB = 1.
000000 <-- 010           XOR mit G = (110101) auf SR angewendet.
000000 <-- 10            Nächstes Bit ins SR geschoben, MSB = 0.
000001 <-- 0             Nächstes Bit ins SR geschoben, MSB = 0.
000010 <--               Letztes Bit ins SR geschoben,  MSB = 0.
000010                   Ende, CRC-Prüfsumme steht im SR.
```

Das Decodieren der Empfängerdaten $\widetilde{c}$ zeigt, dass der Rest nicht null ist. Die Nachricht $\widetilde{c}$ ist somit kein gültiges Codewort. Es ist ein Übertragungsfehler aufgetreten.

Der Encoder kann im Wesentlichen den selben Algorithmus verwenden. An das Nutzdatenwort $\underline{u}$ müssen rechts die p Nullen angehängt werden. Dann ermittelt man auf dieselbe Art und Weise die Prüfbits der Nachricht c.

7.1 Realisierung in Hardware

Der oben dargestellte Algorithmus lässt sich einfach in Hardware realisieren. In der Abbildung 1 ist ein Beispiel für CRC-7 angegeben. Die mit z^{-1} bezeichneten Blöcke sind die Verzögerungselemente, resp. die 1-Bit Speicher des Schieberegisters. Davon gibt es nun p Stück statt $p+1$. Die Bits werden von links nach rechts ins Register geschoben, das MSB (Most Significant Bit) der Daten zuerst. Das Ausgangsbit des Schieberegisters wird verwendet, um via XOR an den notwendigen Stellen, die Bits zu invertieren. Nachdem alle Datenbits in das Schieberegister hinein geschoben wurden, steht im Schieberegister die CRC-Prüfsumme.

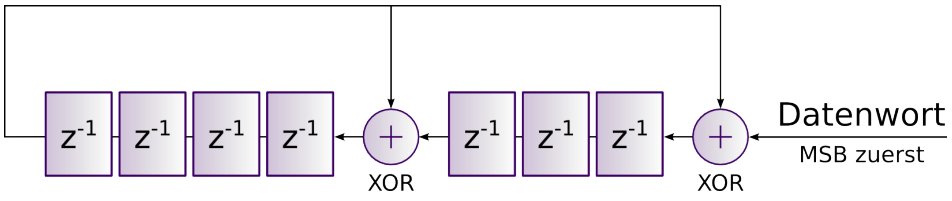

Abbildung 1: Die Realisierung von CRC-7 in Hardware $\left(G(z) = z^7 + z^3 + 1\right)$.

Ein weiteres Beispiel ist in Abbildung 2 für CRC-4 mit dem Polynom $G(z) = z^4 + z + 1$ dargestellt. Damit soll das Bitmuster $\underline{\widetilde{c}} = (1011111)$ überprüft werden. Beachte, dass wir in der folgenden Liste wieder wie oben das Eingangsbit zum Schieberegister (SR) zählen. Die Prüfsumme wird 0, damit ist das Bitmuster $\underline{\widetilde{c}}$ ein gültiges Codewort.

```
  SR       Bitmuster       Kommentar
-------------------------------------------------------
 00000 <-- 1011111         Ausgangslage, SR ist leer.
 00001 <-- 011111          Erstes Bit liegt am SR an,     MSB = 0.
 00010 <-- 11111           Nächstes Bit ins SR geschoben, MSB = 0.
 00101 <-- 1111            Nächstes Bit ins SR geschoben, MSB = 0.
 01011 <-- 111             Nächstes Bit ins SR geschoben, MSB = 0.
 10111 <-- 11              Nächstes Bit ins SR geschoben, MSB = 1.
 00100 <-- 11              XOR mit G = (10011) auf SR angewendet.
 01001 <-- 1               Nächstes Bit ins SR geschoben, MSB = 0.
 10011 <--                 Nächstes Bit ins SR geschoben, MSB = 1.
 00000 <--                 XOR mit G = (10011) auf SR angewendet.
 00000                     Ende, CRC-Prüfsumme steht im SR.
```

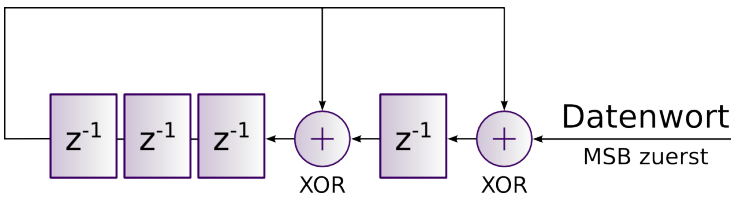

Abbildung 2: CRC-4 in Hardware $\left(G(z) = x^4 + x + 1\right)$.

8 Aufgaben

8.1 Einfache Fragen

1. Wie lautet die CRC-7 Prüfsumme $\underline{p}$ und das Codewort $\underline{c}$ zu $\underline{u} = (11)$?
2. Überprüfen Sie das Codewort $\underline{c} = (1000101100)$ mit CRC-5-ITU und decodieren Sie es, wenn möglich.
3. Welche Vorteile und Nachteile hat CRC-32 gegenüber CRC-12?

8.2 Knifflige Fragen

1. Was liefert das Generatorpolynom $G(z) = z + 1$ als CRC-Prüfsumme zurück? Begründen Sie.
2. Können mit dem CRC-4-Polynom $G(z) = z^4 + z + 1$ Fehler korrigiert werden? Begründen Sie.
3. Wie lange muss das CRC-Polynom sein, um Burstfehler der Länge b zu erkennen?
4. Was geschieht, wenn die Burstfehler zu lange werden?
5. Weshalb ist das Polynom $G(z) = z^2 + z$ kein geeignetes Generatorpolynom?
6. Mit einem CRC-Polynom gerader Parität kann jede ungerade Fehleranzahl erkannt werden. Begründet wird dies mit der Parität des Codewortes und des Restes (nach der Division). Siehe dazu Kapitel 5.4. Können mit einem CRC-Polynom ungerader Parität alle gerde Anzahl von Bitfehlern erkannt werden? Begründen Sie.