Vortrag in Grundlagen der Technischen Informatik
Thema: Codes

Ulrich Berger
Oualid Benabdallah

Inhalt

- Paritätscodes
  - -Definition: Parität
  - -Ein- und Zweidimensionale Paritätscodes
- Hamming Codes
  - -Grundidee des Hammingcodes
  - -Beispiel
  - -Andere Hammingcodes
- Zyklische Codes
  - -Rechnen Modulo 2
  - -Polynomdivision mit XOR
  - -CRC: Cyclic Redundancy Check
  - -CRC Fehlererkennung
- Aufgaben

Parität einer Bitfolge

Definition:

Die Parität einer Bitfolge bezeichnet die Anzahl der mit 1 belegten Bits im Informationswort und heißt gerade(„0“),

wenn die Anzahl dieser Bits gerade ist, andernfalls ungerade(„1“).

Beispiele:

100111 -> Parität 0

1101 -> Parität 1

10101 -> Parität 1

Paritäten lassen sich sehr einfach mit XOR berechnen.

Eindimensionale Paritätscodes

Summe der Einsen	Paritätsbit bei gerader Parität	Paritätsbit bei ungerader Parität
gerade	0	1
ungerade	1	0

Gerade Parität

0000 0
0001 1
0010 1
0011 0
0100 1
0101 0
0110 0
0111 1

Ungerade Parität

0000 1
0001 0
0010 0
0011 1
0100 0
0101 1
0110 1
0111 0

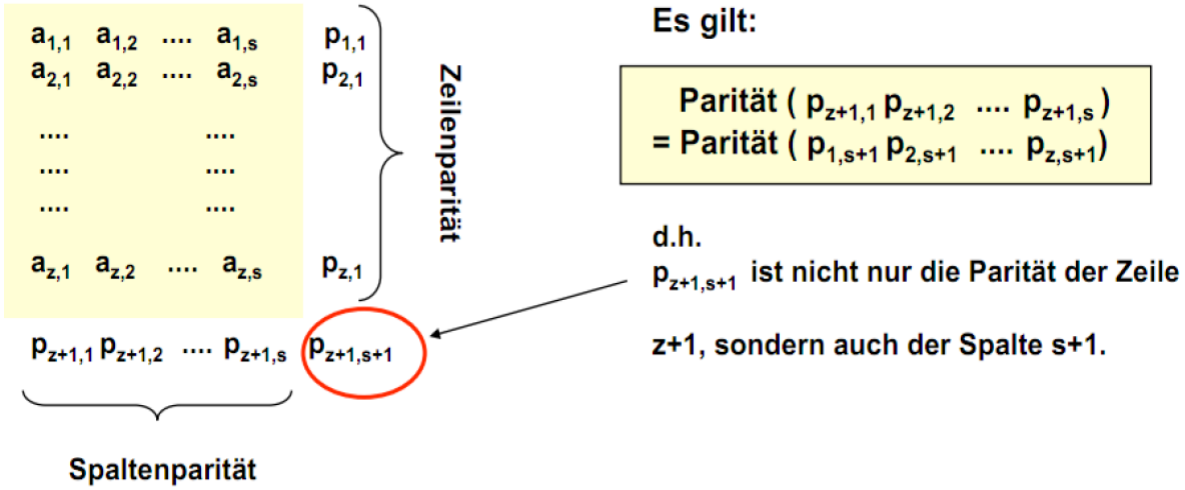
- Overhead: Konstant (1 Bit)
- Fehlerkennung: Ein-Bitfehler werden erkannt
- Fehlerkorrektur: nicht möglich

Zweidimensionale Paritätscodes

Grundidee: Verwende ein Paritätsbit für Zeilen (also für die einzelnen Codeworte) und eins für die Spalten (also für die einzelnen Bits in den Codeworten).

Beispiel:

1 0 0 1 0 1 1
0 1 1 0 1 1 0
1 0 1 0 1 1 0
1 0 0 0 0 0 1
1 0 1 1 1 1 1
1 1 1 0 1 0 0
1 0 0 0 0 0 1



- Overhead: 2n +1 zusätzliche Bits für n^2 Datenbits
- 3-fehlererkennend
- 1-fehlerkorrigierend

Der Hammingcode

- Grundidee:

Ein Paritätsbit je Binärstelle des Codeworts

→ (Hamming-)Codeworte haben (optimale) Länge von $N = 2^k-1$

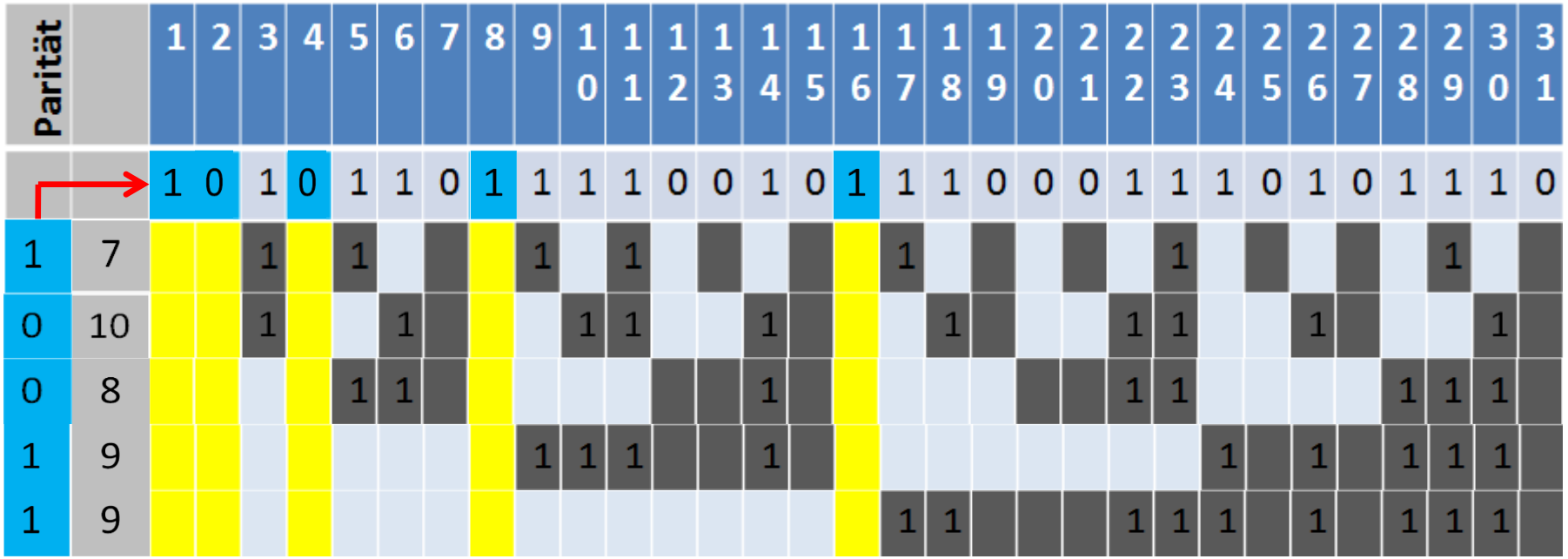
→davon sind $n=2^k-1-k$ Bits Nutzdaten

→die k-Paritätsbits werden an den Stellen $2^0, 2^1, 2^2$, … eingefügt

- Beispiel (k=5): $N = 2^5 - 1 = 31$, $n = 31 - 5 = 26$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
×		×		×		×		×		×		×		×		×		×		×		×		×		×		×		×
	×	×			×	×			×	×			×	×			×	×			×	×			×	×			×	×
			×	×	×	×					×	×	×	×					×	×	×	×					×	×	×	×
							×	×	×	×	×	×	×	×									×	×	×	×	×	×	×	×	×
															×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×

Beispiel


Parität
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
1 0 1 0 1 1 0 1 1 1 1 0 0 1 0 1 1 1 0 0 0 1 1 1 0 1 0 1 1 1 0
1 7
0 10
0 8
1 9
1 9

Beispiel(2)

XOR	Parität		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
			1	0	1	0	1	1	0	1	1	1	1	0	0	1	0	1	1	1	0	0	0	1	1	1	0		0	1	1	1	0
0	1	7			1		1				1		1						1						1						1		
1	1	9			1			1				1	1			1				1				1	1							1	
0	0	8					1	1								1								1	1					1	1	1	
1	0	8									1	1	1			1										1				1	1	1	
1	0	8																	1	1				1	1	1				1	1	1	

Andere Hammingcodes

- Hammingcodes existieren in beliebigen Größen.

Man gibt diese in der Regel als Tupel (N,n) aus Codewortgröße N und Größe der Nutzdaten n an.

- Beispiele:

H(31,26), H(7,4) oder auch H(3,1)

- In der Praxis weit verbreitet ist H(63,57)

Rechnen Modulo 2

- Addition: „normal Addieren dann mod. 2 rechnen“

  0 + 0 = 0

  0 + 1 = 1

  1 + 0 = 1

  1 + 1 = 0

entspricht XOR

- Multiplikation: „normal Multiplizieren, dann mod. 2 rechnen“

  0 * 0 = 0

  0 * 1 = 0

  1 * 0 = 0

  1 * 1 = 1

entspricht AND

Polynomdivision

Polynomdivision in F_2 kann effizient mit XOR umgesetzt werden:

- Schritt 1: Stelle Polynome als Binärzahlen dar
- Schritt 2: Verschiebe den Divisor soweit nach links bis die führenden Stellen übereinstimmen
- Schritt 3: Berechne ein XOR zwischen Dividend und dem verschobenen Divisor
- Schritt 4: Falls das Ergebnis des XOR einen geringeren Grad als der Divisor hat ist dies der Rest der Division, andernfalls weiter mit Schritt 2 wobei das Ergebnis des XOR den neuen Dividend bildet.

Beispiel

f: $x^7+x^3+x+1 = 1*x^7 + 0*x^6 + 0*x^5 + 0*x^4 + 1*x^3 + 0*x^2 + 1*x^1 + 1*x^0$
→10001011

Analog: g: $x^5+x^3+x^1+1$ →101011

f : g → 10001011 : 101011 = 101→ **Faktor: x^2+1**

```
        10101100
   XOR ----------
        00100111
           101011
   XOR ----------
           001100
```

→ **Rest: x^3+x^2**

f= g*Faktor + Rest
$x^7+x^3+x+1= (x^5+x^3+x^1+1)*(x^2+1)+x^3+x^2$

CRC: Cyclic Redundancy Check

- Grundidee: Fasse Nachrichten als Polynome auf, vereinbare ein gemeinsames "Prüfpolynom" und berechne jeweils die Reste der Polynomdivision als Prüfsumme.

- Algorithmus (Codieren):
  - Schritt 1: Gegeben sei eine Nachricht t der Länge k. Diese wird als Polynom f des Grades k-1 aufgefasst.
  - Schritt 2: Wähle ein Generatorpolynom (Prüfpolynom) g vom Grad m.
  - Schritt 3: Multipliziere f mit x^m, d.h. hänge m Stellen mit "0" an die Nachricht. Das resultierende Polynom hat nun den Grad k+m.
  - Schritt 4: Dividiere das so gewonnene Polynom durch das Generatorpolynom und bilde das Restpolynom+ r.
  - Schritt 5: Übertrage die Bitfolge für das Polynom $f * x^m + r$

- CRC-Überprüfung:
  Teile das eingehende Polynom durch das Generatorpolynom,
  falls Rest 0 liegt kein Fehler vor.

CRC Fehlererkennung

- Ein-Bit-Fehler:

werden erkannt, falls g zwei oder mehr Terme enthält

- Zwei-Bit-Fehler:

werden erkannt, falls g keinen Faktor $(1 + x^k)$ enthält

- Alle ungeraden Anzahl an Fehler:

werden erkannt, falls g einen Faktor (1+x) enthält

- Alle Bündelfehler :

werden erkannt, falls nicht mehr Bits als die Länge des CRC-Polynoms fehlerhaft sind