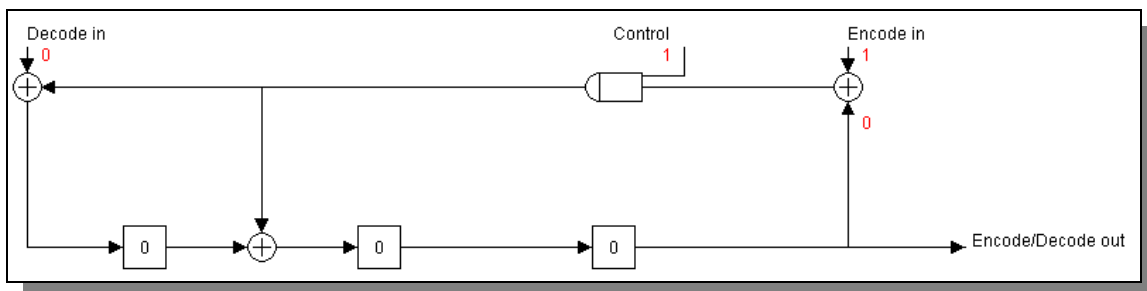




Untersuchungen an Cyclic Redundancy Checks (CRC)



Inhaltsverzeichnis

1 Cyclic Redundancy Checks (CRC)	3
1.1 CRC-Applet initialisieren	3
1.2 Verlauf einer Codierung und einer Decodierung	4
1.3 Fehlerarten beim Decodieren	6
2 Quellenverzeichnis	7
2.1 Quellen aus dem Internet	7
3 Anhang	7
3.1 Abbildungen	7
3.2 Tabellen	7

1 Cyclic Redundancy Checks (CRC)

1.1 CRC-Applet initialisieren

Auf der Homepage der Universität EECE in Australien [1] gibt es eine Java-Applikation, die eine Simulation mit CRC ermöglicht. Alle Informationen für die Benutzung des Programms sind auf die Homepage zu finden [2]. Bevor man das Applet startet, muss man folgende Werte einstellen:

- Das Applet kann eine Bitfolge Codieren oder Decodieren. Zuerst möchten wir sie codieren, somit muss oben links (oberhalb von „Polynomial“) der Knopf auf „Encode“ eingestellt sein.
- Polynomial CRC-12: $1+x+x^2+x^3+x^{11}+x^{12}$ (Wichtig: Knopf „Enter“ drücken!)
- Input bits (k): 6
- Input String: 001011

Das Applet sieht dann folgendermassen aus:

The diagram shows the CRC-12 applet interface. It features a polynomial input field with the value $1+x+x^2+x^3+x^{11}+x^{12}$, an input bits field with the value 6, and an input string field with the value 001011. The interface also includes buttons for Encode, Restart, Delta Back, Back, Step, and Delta Step. The Step button is highlighted, indicating that the calculation has been performed.

Encode	Restart	Delta Back	Back	Step	Delta Step
Polynomial:	$1+x+x^2+x^3+x^{11}+x^{12}$		Input String:	001011	
Input bits (k)	6		Input String:		
Code bits (n)	18		Output String:		

Abbildung 1: Java-Applet mit den Anfangsparametern

Danach muss man mit dem Knopf „Delta Step“ das 18-Bit Codewort (12 + 6 bit) berechnen. Das Resultat (Output String), wie auf die Abbildung 2 zu sehen ist, entspricht „001011100001001011“.

The diagram shows the CRC-12 applet interface after the calculation. The Step button is now highlighted, and the Output String field displays the calculated 18-bit codeword: 001011100001001011.

Encode	Restart	Delta Back	Back	Step	Delta Step
Polynomial:	$1+x+x^2+x^3+x^{11}+x^{12}$		Input String:	001011	
Input bits (k)	6		Input String:	001011	
Code bits (n)	18		Output String:	001011100001001011	

Abbildung 2: Java-Applet mit dem 18-Bit Codewort

1.2 Verlauf einer Codierung und einer Decodierung

In der nachfolgenden Tabelle gibt es eine Schritt-zu-Schritt Codierung mit den wichtigsten Eigenschaften. Die Anfangsparametern sind gleich wie auf die Abbildung 1. (Polynom CRC-12, $k = 6$, Input = 001011).

Schritt	Input	Control	Encode	Decode	CRC												Output
					1	2	3	4	5	6	7	8	9	10	11	12	
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	1	1
4	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0
5	1	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	1
6	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	1	1
7	-	1	1	0	0	1	1	0	1	0	0	1	0	0	0	0	1
8	-	1	1	0	0	0	1	1	0	1	0	0	1	0	0	0	0
9	-	1	1	0	0	0	0	1	1	0	1	0	0	1	0	0	0
10	-	1	1	0	0	0	0	0	1	1	0	1	0	0	1	0	0
11	-	1	1	0	0	0	0	0	0	1	1	0	1	0	0	1	0
12	-	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1
13	-	1	1	0	0	0	0	0	0	0	0	1	1	0	1	0	0
14	-	1	1	0	0	0	0	0	0	0	0	0	1	1	0	1	0
15	-	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1
16	-	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
17	-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
18	-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Tabelle 1: Codierung der Bitfolge „001011“ mit CRC-12

Als Output bekommt man folgendes: 001011100001001011

In der nachfolgenden Tabelle gibt es eine Schritt-zu-Schritt Decodierung mit CRC-12 und als Input „001011100001001011“. Die ersten 6 Bits bilden die Nutzdaten, die restlichen 12 Bits werden für die Check-Summe gebraucht, um Fehler zu identifizieren.

Schritt	Input	Control	Encode	Decode	CRC												Output
					1	2	3	4	5	6	7	8	9	10	11	12	
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-
3	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	-
4	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	-
5	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	-
6	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	-
7	1	1	0	1	1	1	1	0	1	0	0	0	0	0	0	0	-
8	0	1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	-
9	0	1	0	0	0	0	1	1	1	0	1	0	0	0	0	0	-
10	0	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	-
11	0	1	0	0	0	0	0	0	1	1	1	0	1	0	0	0	-
12	1	1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	-
13	0	1	0	0	0	1	0	0	0	0	1	1	1	0	1	0	-
14	0	1	0	0	0	0	1	0	0	0	0	1	1	1	0	1	-
15	1	1	0	1	0	1	1	0	0	0	0	0	1	1	1	1	-
16	0	1	0	0	1	1	0	0	0	0	0	0	0	1	1	0	-
17	1	1	0	1	1	1	1	0	0	0	0	0	0	0	1	1	-
18	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	-
19	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
20	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
21	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
22	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
23	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
24	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
25	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
26	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
27	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

28	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
29	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
30	-	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabelle 2: Decodierung mit CRC-12

Ab Schritt 19 werden die Bits nach rechts verschoben und als Output eingetragen (engl. Flushing). Das Resultat ist „000000000000“, also die Bitfolge enthält keinen Fehler.

1.3 Fehlerarten beim Decodieren

Nun werden verschiedene Bit-Fehler beim Codewort manuell eingefügt. Die Bitfolge ohne Fehler ist gleich wie in die vorherigen Aufgaben, und zwar „001011100001001011“.

Zuerst wird 1-Bit Fehler getestet und dann 4-/8- und 12-Bit Fehler (Die verfälschte Bits sind unterstrichen):

Input	Output
<u>1</u> 01011100001001011	100101000101
0010111000010010 <u>1</u> 0	000000000001

Tabelle 3: 1-Bit Fehler

Input	Output
<u>110</u> 111100001001011	000110011000
001011100001000 <u>100</u>	000000001111

Tabelle 4: 4-Bit Fehler

Input	Output
<u>1101000</u> 10001001011	110110000110
0010111000 <u>10110</u> 100	000011111111

Tabelle 5: 8-Bit Fehler

Input	Output
<u>11010001111</u> 0001011	111001000110
001011 <u>1011110</u> 10100	111111111111

Tabelle 6: 12-Bit Fehler

Man sieht, dass die Fehler immer erkannt werden. Wenn alle Output-Bits nicht gleich Null sind, dann gibt es im Input ein oder mehrere verfälschte Bits.

Ein wichtiger Faktor bei der Fehlererkennung spielt das Generatorpolynom; je ein Generatorpolynom grösser ist, desto mehrere Fehler erkannt werden können. Laut Wikipedia [3] können sogar 1-Bit Fehler korrigiert werden, wenn der Datenblock gewisse Faktoren mit dem Grad des Polynoms in Zusammenhang hat.

2 Quellenverzeichnis

2.1 Quellen aus dem Internet

- [1] The Cyclic Code Encoder / Decoder
[<http://www.ee.uwa.edu.au/%7Eroberto/teach/itc314/java/CRC/crc.html>], 2008
- [2] Simulation instructions for the CRC
[<http://www.ee.uwa.edu.au/%7Eroberto/teach/itc314/java/CRC/>], 2008
- [3] Theorie CRC
[http://de.wikipedia.org/wiki/Cyclic_Redundancy_Check], 2008

3 Anhang

3.1 Abbildungen

Abbildung 1: Java-Applet mit den Anfangsparametern	3
Abbildung 2: Java-Applet mit dem 18-Bit Codewort	3

3.2 Tabellen

Tabelle 1: Codierung der Bitfolge „001011“ mit CRC-12	4
Tabelle 2: Decodierung mit CRC-12	6
Tabelle 3: 1-Bit Fehler	6
Tabelle 4: 4-Bit Fehler	6
Tabelle 5: 8-Bit Fehler	6
Tabelle 6: 12-Bit Fehler	6