



# Test und Verlässlichkeit Foliensatz 3: Überwachung

Prof. G. Kemnitz

Institut für Informatik, TU Clausthal (TV\_F3)

June 15, 2017



## Inhalt Foliensatz TV\_F3: Überwachung

### Informationsredundanz

- 1.1 Fehlererkennende Codes
- 1.2 Prüfkennzeichen
- 1.3 Fehlerkorr. Codes
- 1.4 Hamming-Codes
- 1.5 RAID Systeme

### Formatkontrollen

- 2.1 Syntaxtest

- 2.2 Typ und Wertebereich

- 2.3 Signalüberwachung

### Wertekontrollen

- 3.1 Mehrfachber. & Vergleich

- 3.2 Diversität

- 3.3 Loop-Back Test

- 3.4 Kontrollkriterien für Richtigkeit



## Fehlfunktionen im laufenden Betrieb

Mögliche Reaktionen auf erkannte Fehlfunktionen:

- Schadensbegrenzung: Ergebnis nicht verwenden, Überführung in einen sicheren Zustand, ...
- Neuinitialisierung, Wiederholung, diversitäre Berechnung, ...
- Fehlertoleranz (Korrektur eigener, selbst erkannter Fehlfunktionen).

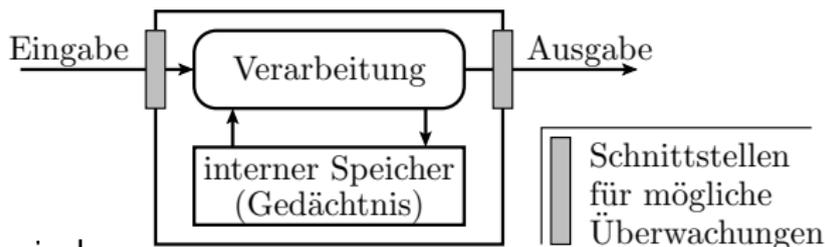
Jede sinnvolle Reaktion setzt voraus, dass die Fehlfunktionen erkannt werden. Erfordert Überwachung der Service-Leistungen:

- Zeitüberwachung,
- Formatkontrollen,
- Wertekontrollen.

Dieser Foliensatz behandelt die unterschiedlichen Arten der Überwachung und wie gut diese sind. Reaktionen auf erkannte FF behandelt später Foliensatz F6.

## Überwachungsmöglichkeiten

Laut F1 Abschn. 1.1 bildet ein Service aus Eingaben Ausgaben. Die Ein- und Ausgaben sind ganz allgemein bedatete Objekte mit einem auf die Art des Services abgestimmten Format. Ein Service kann ein Gedächtnis haben.



Überwachbar sind:

- Ausführungszeit,
- das Format der Ein- und Ausgabe (Kontrolle auf Zulässigkeit),
- die Ausgabewerte (Kontrolle auf Richtigkeit).

Die höchsten Erkennungswahrscheinlichkeiten haben  
Formatkontrollen auf Basis von Informationsredundanz.

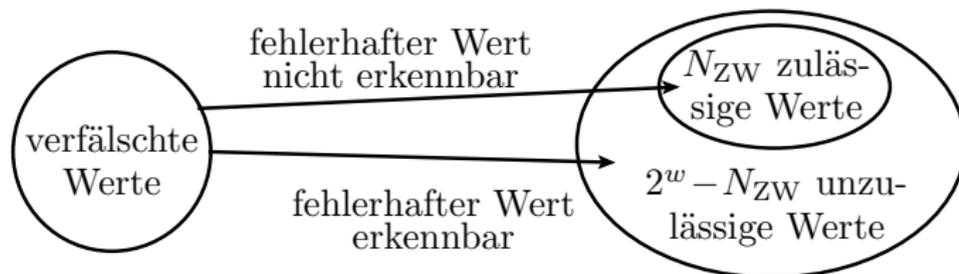


## Informationsredundanz



## Informationsredundanz

Die binäre Darstellung von  $N_{ZW}$  zu unterscheidenden zulässigen Werten verlangt mindestens  $w \geq \log_2(N_{ZW})$  Bits. Bei  $2^w > N_{ZW}$  (mehr darstellbare als darzustellende Werte) weisen unzulässige Werte auf Fehlfunktionen hin.



Eine Darstellung mit  $w \gg \log_2(N_{ZW})$  (die überwiegende Mehrheit der darstellbaren Werte ist unzulässig) erlaubt das Erkennen und bei geschickter Codierung sogar die Korrektur von Datenverfälschungen.



## Erkennungswahrscheinlichkeit

- Wenn sich fehlerhafte Werte gleichmäßig auf zulässige und unzulässige Werte abbilden, Erkennungswahrscheinlichkeit:

$$p_E \approx 1 - \frac{N_{ZW}}{2^w} \quad (1)$$

( $N_{ZW}$  – Anzahl der zulässigen Werte;  $w$  – Bitanzahl zur Darstellung).

- Durch Wahl einer ausreichenden Bitanzahl  $w$  praktisch eins.
- Wenn fehlerhafte Werte unverhältnismäßig oft zulässig sind, kann  $p_E$  deutlich kleiner sein als nach Gl. 1.
- Wenn fehlerhafte Werte unverhältnismäßig oft unzulässig sind, ist  $p_E$  größer als nach Gl. 1.



## Beispiel Rechtschreibtest

Wort im Wörterbuch enthalten?

- Maskierung: falsches Wort, das im Wörterbuch enthalten ist, z.B. »Maus« statt »Haus«.
- Phantom-FF: zulässiges Wort nicht im Wörterbuch.

- 
- Anzahl der mit  $N_{\text{Byte}}$  darstellbaren Zeichenketten:

$$2^{8 \cdot N_{\text{Byte}}}$$

Anteil der gültigen Worte fast null. Nach Gl. 1  $p_E \approx 1$ .

- Tatsächlich  $p_E \approx 80\%$ , weil Schreibfehler viel öfter als zufällige Bitverfälschungen gültige Worte sind.
- Es gibt auch relativ viele Worte, die im Wörterbuch fehlen. Phantom-FF-Wahrscheinlichkeit typisch  $p_{\text{Phan}} > 1\%$ .



## Einzelbitfehler bei Übertragung und Speicherung

Bei der Datenspeicherung und Übertragung sind Bitfehler äußerst selten.

Wenn die Daten so auf Datenobjekte aufgeteilt werden, dass jede Verfälschungsursache, z.B. ein Störimpuls, nur in jedem Datenobjekt ein Bit verfälscht, genügt der Nachweis aller Einzelbitfehler, um fast alle Datenverfälschungen zu erkennen.

Für die Einzelbitfehlererkennung genügt ein Code, in dem nur Werte mit gerader (ungerader) Anzahl von Einsen gültig sind (siehe später Erweiterung um ein Paritätsbit).

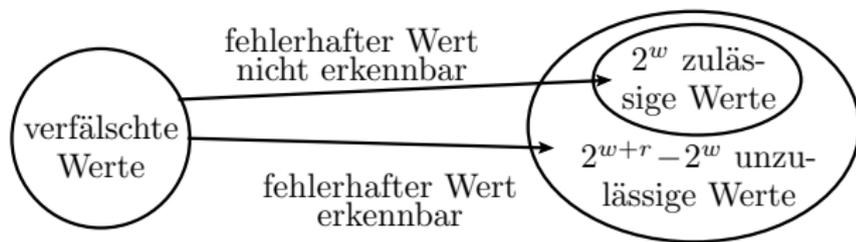
Anteil gültiger Codeworte 50%, Erkennungswahrscheinlichkeit

$$p_E \approx 1 \gg 1 - 50\%$$



## Fehlererkennende Codes

## Fehlererkennende Codes (FEC)



Bei einem fehlererkennenden Code werden die zulässigen Werte pseudo-zufällig auf eine viel größere Menge darstellbarer Werte verteilt. Pseudo-zufällig bedeutet hier, Zuordnung nach einem umkehrbaren Algorithmus so, dass Verfälschen weder bevorzugt auf zulässige noch auf unzulässige Datenworte abgebildet werden und Gl. 1 gilt. Mit  $2^w$  zulässigen Werten und  $2^{w+r}$  darstellbaren Werten beträgt die Erkennungswahrscheinlichkeit:

$$p_E = 1 - \frac{2^w}{2^{w+r}} = 1 - 2^{-r} \quad (2)$$



## Arithmetische Codes

Arithmetische Codes werden durch eine Menge von arithmetischen Operationen gebildet. Beispiel Multiplikation der Datenworte mit einer ganzzahligen Konstanten:

$$s = 34562134 \cdot x$$

Von  $s$  sind nur die Vielfachen von 34562134 gültig. Die Anzahl der darstellbaren Werte ist 34562134 mal so groß wie die der gültigen Werte. Zu erwartende Verfälschungen werden nicht vorzugsweise auf Vielfache von 34562134 abgebildet. Erkennungswahrscheinlichkeit

$$p_E \approx 1 - \frac{1}{34562134}$$

und damit fast eins. Bei sehr großen unbekanntem Primzahlen als Multiplikatoren ist es selbst vorsätzlich kaum möglich, gültige Codeworte in andere gültige Codeworte zu verfälschen. Einsatz auch zur kryptographischen Verschlüsselung.



## Zyklische Codes

Codierung durch die Multiplikation mit einer Konstanten, allerdings nicht arithmetisch, sondern modulo-2. In Hard- oder Software einfacher als arithmetische Multiplikation:

Codierung

Decodierung

$$\begin{array}{r}
 10010101101 \\
 \oplus \quad \quad \quad 10011 \\
 \hline
 \oplus 10010101101 \\
 \oplus 10010101101 \\
 \oplus 00000000000 \\
 \oplus 00000000000 \\
 \oplus 10010101101 \\
 \hline
 100011100100111
 \end{array}$$

$$\begin{array}{r}
 100011100100111 : 10011 = 100101101 \\
 \oplus 10011 \\
 \hline
 10110 \\
 \oplus 10011 \\
 \hline
 10101 \\
 10011 \\
 11000 \\
 \oplus 10011 \\
 \hline
 10111 \\
 \oplus 10011 \\
 \hline
 10011 \\
 \oplus 10011 \\
 \hline
 \text{Rest: } 00000
 \end{array}$$

(unverfälscht)



Mathematisch werden die zu multiplizierenden Faktoren als Polynome dargestellt:

- $10011 \Rightarrow 1 \cdot x^4 \oplus 0 \cdot x^3 \oplus 0 \cdot x^2 \oplus 1 \cdot x^1 \oplus 1 \cdot x^0 = x^4 \oplus x \oplus 1$
- $10010101101 \Rightarrow x^{10} \oplus x^7 \oplus x^5 \oplus x^3 \oplus x^2 \oplus 1$

Eine Multiplikation mit  $x$  beschreibt eine Verschiebung um eine Bitstelle. Die Multiplikation mit null oder eins ist eine UND-Verknüpfung und  $\oplus$  die modulo-2-Addition (EXOR). Das Produkt beider Polynome

$$(x^{10} \oplus x^7 \oplus x^5 \oplus x^3 \oplus x^2 \oplus 1) \cdot (x^4 \oplus x \oplus 1) = x^{14} \oplus x^{10} \oplus x^9 \oplus x^8 \oplus x^5 \oplus x^2 \oplus x \oplus 1$$

repräsentiert denselben Bitvektor, der für die Multiplikation der Folgen auf der Folie zuvor berechnet wurde. Die Polynomdivision:

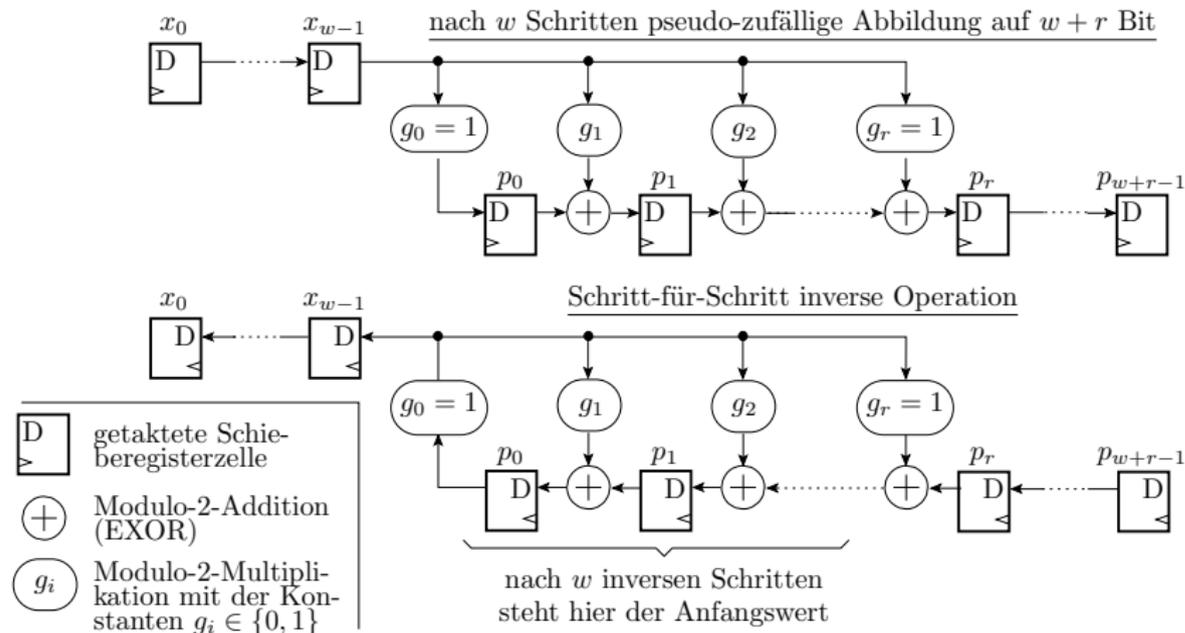
$$(x^{14} \oplus x^{10} \oplus x^9 \oplus x^8 \oplus x^5 \oplus x^2 \oplus x \oplus 1) : (x^4 \oplus x \oplus 1) = x^{10} \oplus x^7 \oplus x^5 \oplus x^3 \oplus x^2 \oplus 1$$

liefert ohne Rest das Polynom der Originalfolge.



## Linear rückgekoppelte Schieberegister

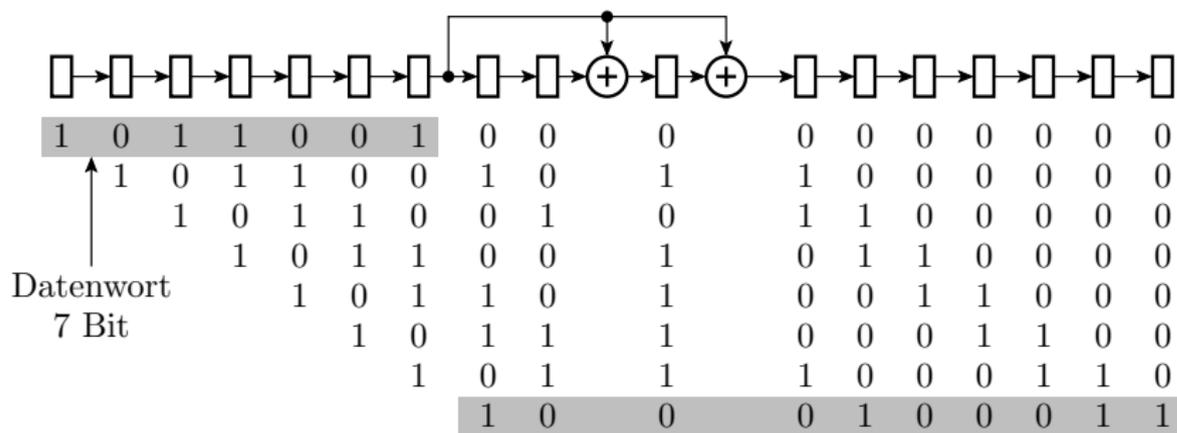
Die Codierung und Decodierung erfolgt mit Schieberegistern.



Erkennungswahrscheinlichkeit:  $p_E = 1 - \frac{2^w}{2^{w+r}} = 1 - 2^{-r}$



## Beispiel für die Codierung



3 Bit längeres codiertes Wort

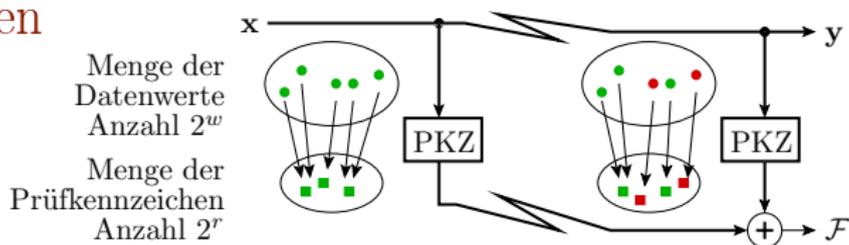
- Das Ergebnis ist 3 Bit länger und pseudo-zufällig umcodiert.
- Anzahl der zulässigen Codeworte bleibt  $2^7$ .
- Anzahl der möglichen Codeworte vergrößert sich auf  $2^{10}$ .





## Prüfkennzeichen

## Prüfkennzeichen



- Jedem  $w$ -Bit-Datenwort wird pseudo-zufällig genau eines der  $r$ -Bit-Prüfkennzeichen zugeordnet ( $w \gg r$ ).
- Nach der Übertragung oder Speicherung wird das Prüfkennzeichen ein zweites mal gebildet.
- Wenn weder die Daten noch das Prüfkennzeichen verfälscht sind, stimmen beide Prüfkennzeichen überein.

Für pseudo-zufällig gebildete Prüfkennzeichen gilt:

- Anzahl der zulässigen Prüfkennzeichen-Werte-Paare  $2^w$ ,
- Anzahl darstellbarer Paare  $2^{w+r}$ . Erkennungswahrscheinlichkeit:

$$p_E \approx \frac{2^{w+r} - 2^w}{2^{w+r}} = 1 - 2^{-r} \quad (3)$$



## Prüfsummen

Prüfkennzeichenbildung durch Aufsummierung (arithmetisch, bitweises EXOR, ...).

einfache Genauigkeit	doppelte Genauigkeit	bitweises EXOR
1011 11	1011 11	1011
0010 2	0010 2	0110
1101 13	1101 13	1101
0100 4	0100 4	1100
(1) <span style="border: 1px solid black; padding: 2px;">11110</span> 30	<span style="border: 1px solid black; padding: 2px;">0001 11110</span> 30	<span style="border: 1px solid black; padding: 2px;">1100</span>

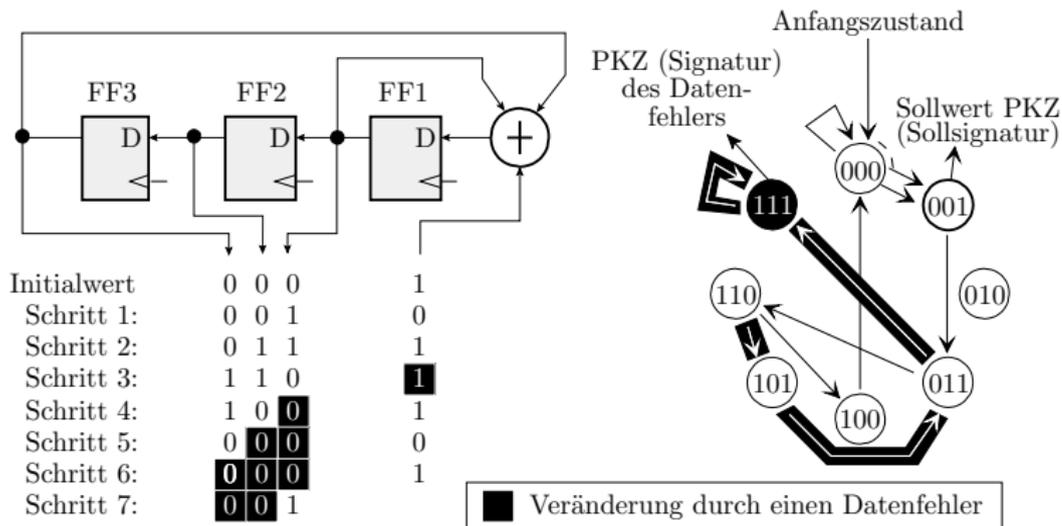
Bei »einfacher Genauigkeit« und »bitweisem EXOR« erscheint die Annahme »pseudo-zufällige Abbildung« gerechtfertigt<sup>1</sup>:

$p_E \approx 1 - 2^{-4}$ . Bei »doppelter Genauigkeit« bilden sich Verfälschungen vorzugsweise auf die niederwertigen Bits ab.

Maskierungswahrscheinlichkeit:  $2^{-4} > 1 - p_E \gg 2^{-8}$ .

<sup>1</sup>Kein Nachweis für vertauschte Summationsreihenfolge.

## Prüfkennzeichenbildung mit LFSR<sup>2</sup>



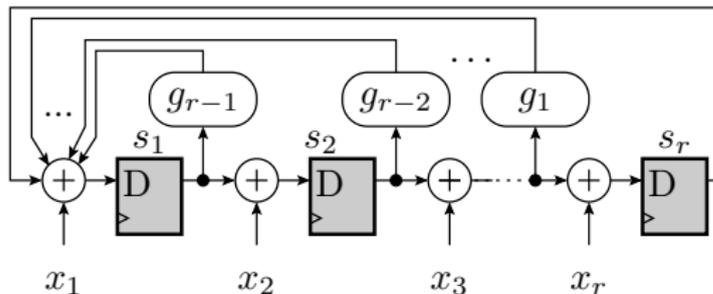
Das Prüfkennzeichen wird wie bei der CRC-Decodierung mit einem linear rückgekoppelten Schieberegister (LFSR) gebildet. Im Beispiel hat das LFSR im Gegensatz zur Polynomdivision Folie 17 eine zentrale Rückführung. Abbildung auch pseudo-zufällig.

<sup>2</sup>LFSR – Linear Feedback Shift Register.

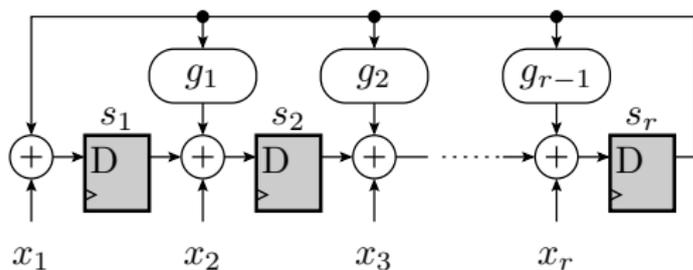


## LFSR für parallele Datenströme

Für die Bildung von Prüfkennzeichen ist es nur wichtig, dass die Abbildung pseudo-zufällig hinsichtlich der zu erwartenden Verfälschungen erfolgt. Diese Eigenschaft hat auch ein rückgekoppeltes Schieberegister, bei dem die Daten modulo-2 als Bitvektoren zu den Registerzuständen addiert werden (paralleles Signaturregister).



Die Rückführung darf dabei auch wie bei der Polynom-Division dezentral sein.



Die Koeffizienten  $g_i$  der Rückführung, bei der Polynom-Division das Divisor-Polynom, bestimmen die autonome Zyklusstruktur<sup>3</sup>. Die autonome Zyklusstruktur ist bei zentraler und dezentraler Rückführung mit denselben Rückführkoeffizienten gleich. Bevorzugt werden lange Zyklen, insbesondere sog. primitive Polynome, bei denen alle Zustände außer »alles null« einen  $2^r - 1$  langen Maximalzyklus bilden. Gebräuchliche Rückführungen:

- USB (CRC-5): 5 Bit, nur  $g_2 = 1$
- Ethernet (CRC-32, IEEE802.3): 32 Bit,  $g_{26} = g_{23} = g_{22} = g_{16} = g_{12} = g_{11} = g_{10} = g_8 = g_7 = g_5 = g_4 = g_2 = g_1 = 1$

<sup>3</sup>Zyklusstruktur ohne Eingaben.



## Experiment Fehlererkennungssicherheit von LFSR

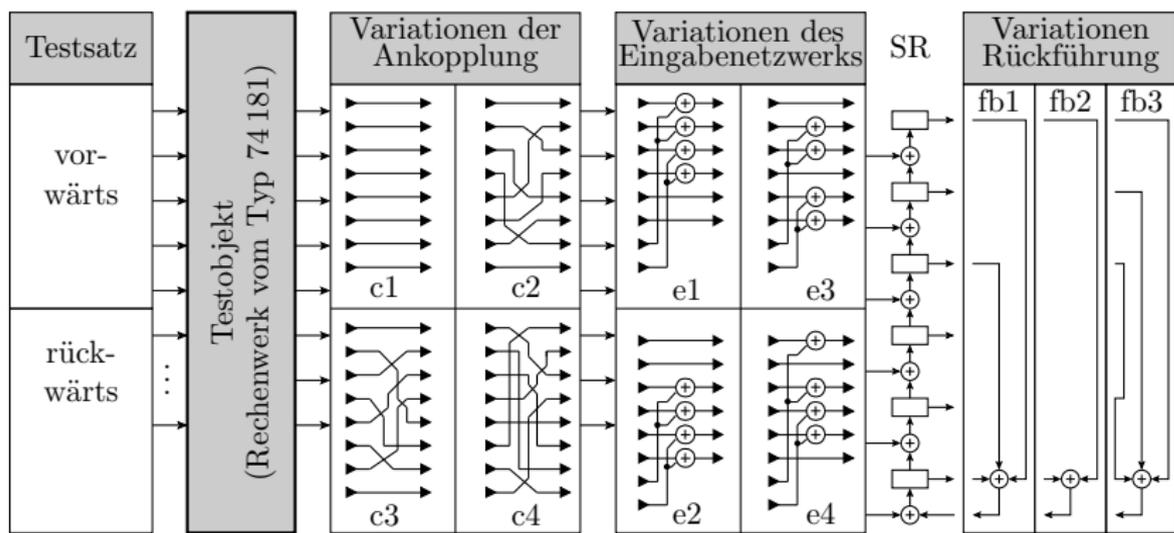
Es ist schwer zu glauben, dass

- mit  $r$ -Bit Prüfkennzeichen beliebige Verfälschung mit einer Wahrscheinlichkeit  $p_E = 1 - 2^{-r}$  erkannt werden und
- die Schaltungsstruktur, die Rückführung etc. kaum Einfluss auf die Erkennungswahrscheinlichkeit haben sollen.

Deshalb ein Experiment:

- Simulation einer Schaltung (4-Bit-Rechenwerk) mit einem Testsatz und 250 verschiedenen Haftfehlern. Berechnung des Prüfkennzeichens für jeden Fehler.
- Variation der Testsatzreihenfolge,
- Variation der Ankopplung an das LFSR und
- Variation der Rückführung.

Zählen der nachweisbaren Fehler für jede Konfiguration.



Aus  $r = 6$  bit folgt, dass jeder Fehler mit einer Wahrscheinlichkeit  $p_E = 1 - 2^{-6} = 98,44\%$  erkenn- und mit einer Wahrscheinlichkeit  $p_F = 2^{-6} = 1,36\%$  nicht erkennbar sein müsste. Definition einer Zufallsgröße  $X_i$  zum Zählen der nicht erkennbaren Fehler:

$$P(X_i = 0) = 1 - 2^{-6} \text{ Fehler } i \text{ nachweisbar}$$

$$P(X_i = 1) = 2^{-6} \text{ Fehler } i \text{ nicht nachweisbar}$$



Wenn die Theorie stimmt, müsste die Anzahl der maskierten Fehler

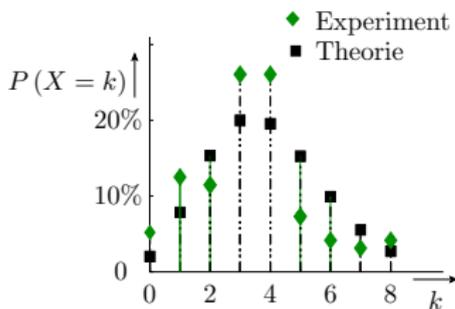
$$X = \sum_{i=1}^{250} X_i$$

binomialverteilt sein:

$$P(X = k) = \binom{N}{k} \cdot 2^{-r \cdot k} \cdot (1 - 2^{-r})^{N-k}$$

Anzahl der maskierten Fehler

		e1				e2				e3				e4			
		c1	c2	c3	c4												
vorwärts	fb1	3	4	1	2	3	4	3	3	4	2	4	3	4	3	4	6
	fb2	3	4	1	7	2	2	1	4	2	1	1	3	2	5	3	7
	fb3	5	2	2	8	4	5	3	4	3	6	3	7	5	3	3	4
rückwärts	fb1	6	4	4	2	3	4	3	4	3	4	3	4	4	8	4	5
	fb2	2	0	0	1	4	1	4	1	0	0	0	1	1	1	4	1
	fb3	2	4	3	4	4	8	5	8	3	3	3	6	3	3	4	3

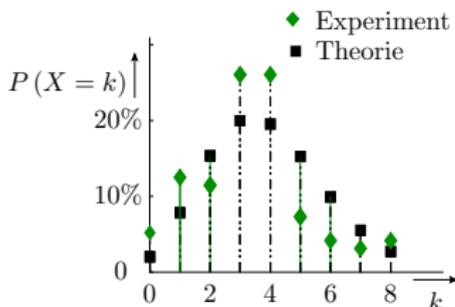


Für die Stichprobengröße sind die Abweichungen zwischen Vorhersage und Experiment nicht signifikant.



Anzahl der maskierten Fehler

		e1				e2				e3				e4			
		c1	c2	c3	c4												
vorwärts	fb1	3	4	1	2	3	4	3	3	4	2	4	3	4	3	4	6
	fb2	3	4	1	7	2	2	1	4	2	1	1	3	2	5	3	7
	fb3	5	2	2	8	4	5	3	4	3	6	3	7	5	3	3	4
rückwärts	fb1	6	4	4	2	3	4	3	4	3	4	3	4	4	8	4	5
	fb2	2	0	0	1	4	1	4	1	0	0	0	1	1	1	4	1
	fb3	2	4	3	4	4	8	5	8	3	3	3	6	3	3	4	3



Struktur, Ankopplung und Rückführung des LFSR haben kaum Einfluss auf die Maskierungswahrscheinlichkeit.

Es gibt in der Literatur gegenteilige Behauptungen<sup>4</sup>, z.B.

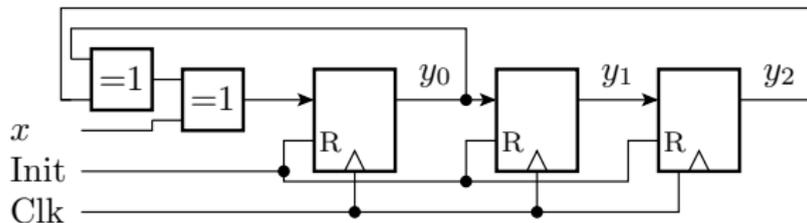
- [GrWil...] Primitive Rückführungen seien viel besser als andere. Mysteriöse Beweisführung über die Z-Transformation ohne wirklichen Bezug zur Erkennungswahrscheinlichkeit.
- [Jar..] Test einer 2-Gatterschaltung mit einem 4-Bit-Signaturregister, in dem keiner der 4 Haftfehler erkannt wird. Beispiel konstruiert oder zufällig gefunden?

<sup>4</sup>In manchen Veröffentlichungen ist der Wunsch der Vater des Gedanken.

## Beispielaufgabe



Gegeben ist folgendes linear rückgekoppelte Schieberegister:

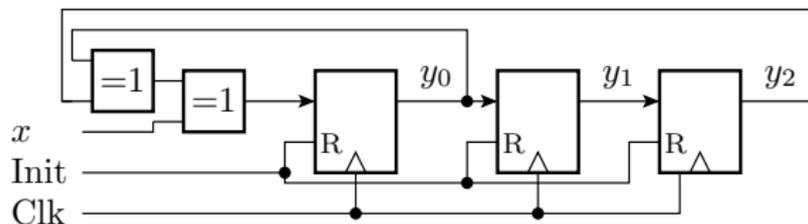


	$x$	$y_2$	$y_1$	$y_0$
0	1	0	0	0
1	0			
2	0			
3	1			
4	1			
5	0			
6	0			
7	1			
8	0			
9	1			
10	1			
11	1			
12	1			
13	0			
14	1			
15	0			
PKZ:				

- 1 Welches Prüfkennzeichen  $\mathbf{y} = y_2y_1y_0$  hat die Datenfolge »1001100101111010« bei Abbildung beginnend mit dem höchstwertigen Bit. Startwert 000.
- 2 Wie hoch ist Fehlererkennungswahrscheinlichkeit?



## Lösung



	$x$	$y_2$	$y_1$	$y_0$
0	1	0	0	0
1	0	0	0	1
2	0	0	1	1
3	1	1	1	1
4	1	1	1	1
5	0	1	1	1
6	0	1	1	0
7	1	1	0	1
8	0	0	1	1
9	1	1	1	1
10	1	1	1	1
11	1	1	1	1
12	1	1	1	1
13	0	1	1	1
14	1	1	1	0
15	0	1	0	0
PKZ:		0	0	1

Erkennungswahrscheinlichkeit:

$$p_E \approx 1 - 2^{-3} = 87,5\%$$



## Zusammenfassung

Datensicherung mit fehlererkennenden Codes / Prüfkennzeichen:

- Geringer Berechnungsaufwand.
- Geringer Zusatzaufwand für Datenübertragung und -speicherung ( $r$  zusätzlich gespeicherte / übertragene Bits für eine Datenobjekt beliebiger Größe).
- Maskierungswahrscheinlichkeit  $2^{-r}$ . Mit ausreichendem  $r$  immer vernachlässigbar klein.

Dateien, Nachrichten etc. werden sehr oft mit einem Prüfkennzeichen geschützt.

Fehlererkennende Codes sind weniger gebräuchlich.

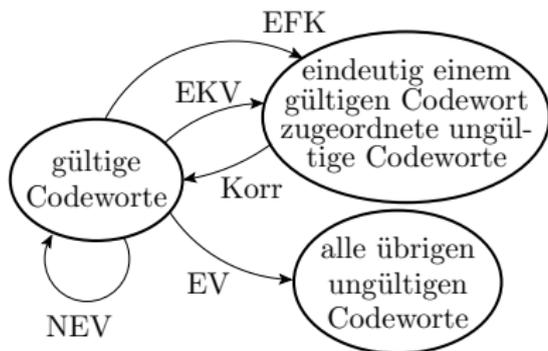


## Fehlerkorr. Codes



## Fehlerkorrigierende Codes

- EFK erkennbar, aber falsch korrigierte Datenverfälschung
- EKV erkennbare und korrigierbare Datenverfälschung
- EV erkennbare, nicht korrigierbare Datenverfälschung
- NEV nicht erkennbare Datenverfälschung
- Korr Korrektur



Erweiterung der Menge der darstellbaren Codeworte um eine viel größere Menge korrigierbarer Codeworte und optional um unzulässige nicht korrigierbare Codeworte. Mindestbitanzahl:

$$2^{N_{\text{Bit}}} \geq N_{\text{CWG}} + N_{\text{CWG}} \cdot N_{\text{CWK/CWG}}$$

( $N_{\text{CWG}}$  – Anzahl gültige Codeworte;  $N_{\text{CWK/CWG}}$  – Anzahl korrigierbare Codeworte je gültiges Codewort). Die Erkennungswahrscheinlichkeit als Anteil der übrigen ungültigen Codeworte verringert sich durch Korrekturmöglichkeiten.



## Beispiel: Korrektur von Einzelbitfehler

Anzahl korrigierbare Codeworte je gültiges Codewort gleich Bitanzahl:

$$N_{\text{CWK/CWG}} = N_{\text{Bit}}$$

Mindestbitanzahl:

$$2^{N_{\text{Bit}}} \geq N_{\text{CWG}} + N_{\text{CWG}} \cdot N_{\text{Bit}}$$

Für  $N_{\text{CWG}} = 256$  gültige Codeworte:

$$2^{N_{\text{Bit}}} \geq 256 \cdot (1 + N_{\text{Bit}})$$

$$N_{\text{Bit}} \geq 12$$

Probe:

$$2^{12} > 2^8 \cdot (1 + 12)$$

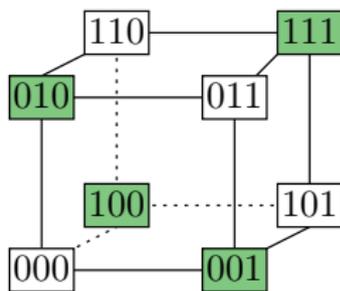


## Hamming-Codes



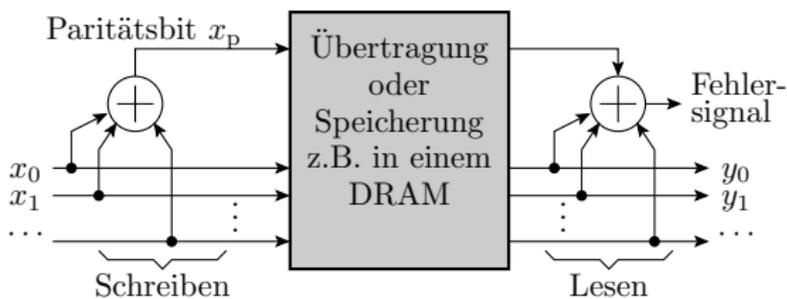
## Hamming-Distanz

Die Hamming-Distanz ist die Anzahl der Bitpositionen, in denen sich zwei Codeworte unterscheiden. Distanz von 2 oder mehr garantiert, dass ein 1-Bit Fehler nicht zu einem anderen gültigen Codewort führt.



- Erkennen von  $k$ -Bit Fehlern verlangt eine Hamming-Distanz von mindestens  $k + 1$ .
- Um  $k$ -Bit Fehler korrigieren zu können, ist eine Hamming-Distanz von  $\geq 2k + 1$  erforderlich.

## Parität als Prüfkennzeichen (Hamming-Distanz 2)



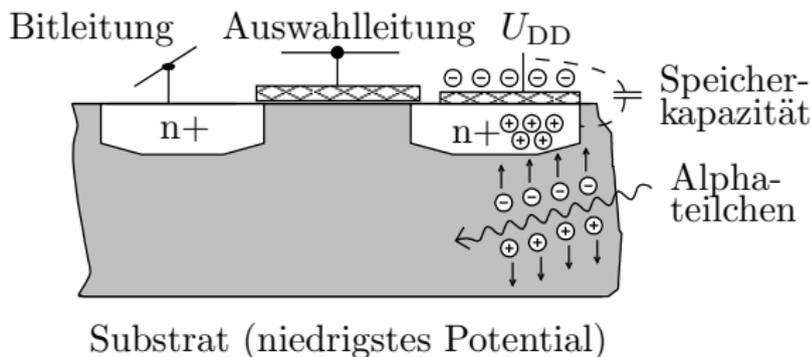
Einzelprüfbit, modulo-2 Summe (EXOR-Verknüpfung):

$$x_p = x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_1 \oplus x_0$$

bei gerader Anzahl von Einsen »0« sonst »1«.

- Erkennt jede ungeradzahlige Anzahl von Bitverfälschungen.
- Wenn geradzahlige und ungeradzahlige Bitfehler gleichhäufig auftreten:  $p_E \approx 50\%$
- Überwiegend Einzelbitfehler:  $p_E \gg 50\%$

## Paritätstest für DRAMs und Speicherriegel



- Informationsspeicherung in winzigen Kapazitäten.
- Häufigste Ursache für Datenverfälschungen: Alphastrahlung.
- Deren Quellen radioaktiver Zerfall von Uran und Thorium, die als Spurenelemente im Gehäuse und im Aluminium der Leiterbahnen enthalten sind, und Kernprozesse im Silizium durch Höhenstrahlung. Seltene Ereignisse.

- Energie eines Alpha-Teilchen: 5 MeV. Energieverlust bei der Generierung eines Elektronen-Loch-Paares

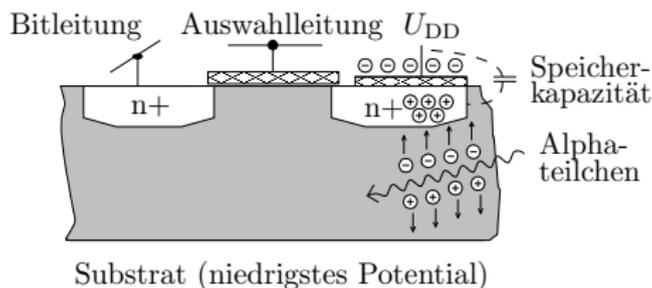
$\approx 3,6 \text{ eV} \Rightarrow$  Generierung

von  $\approx 10^6$  Ladungsträgerpaaren. Reichweite  $\approx 89 \mu\text{m}$ .

gespeicherte Ladung  $\approx 10^5$  Ladungsträger. Datenverlust einer oder mehrerer benachbarter Zellen möglich.

- Mittlerer Zeitabstand zwischen zwei Datenverfälschungen Stunden. Gleichzeitige Verfälschung durch zwei Alphateilchen unwahrscheinlich.
- Geometrische Trennung der Zellen eines Datenworts (getrennte Schaltkreise oder Speichermatrizen)  $\Rightarrow$  Einzelbitverfälschung je gelesenes Datenwort.

- 100%iger Nachweis durch Paritätskontrolle.





## Kreuzparität (Fehlerkorrigierender Paritätscode)

Daten sind in einem 2-dimensionalen Array organisiert. Paritätsbildung für alle Zeilen und Spalten. Erlaubt Lokalisierung und Korrektur von 1-Bit Fehlern. Einsatz in redundanten Festplatten-Arrays (RAID 3 und RAID 5).

...						
1	0	0	1	0	0	
0	1	0	0	1	1	1
0	0	0	1	0	1	0
1	0	0	1	1	1	0
1	1	0	0	1	1	0
0	1	1	0	1	0	1
1	0	0	0	0	1	0
1	1	1	0	0	1	
...						
0 1 1 0 1 0 1						

← Querparität

← Längsparität



## Beispielaufgabe



Kontrollieren Sie für die nachfolgenden Bitfelder mit Kreuzparität, ob eine erkennbare oder eine erkenn- und korrigierbare Verfälschung vorliegt und führen Sie, wenn möglich, die Korrektur durch.

1011010011000010	1	Querparität
1011011010010100	0	
1001011010010101	0	
1000010011111110	1	
1101101100110100	0	
0010110000110111	0	
0101011001000001	0	
1100100010011000	0	
0111101111100111		
		Längsparität



## Lösung

1011010	0011000010	1
1011011	010010100	0
1001011	010010101	0
1000010	0011111110	1
1101101	100110100	0
0010110	000110111	0
0101011	001000001	0
1100100	010011000	0
0111101	111100111	

Längsparität

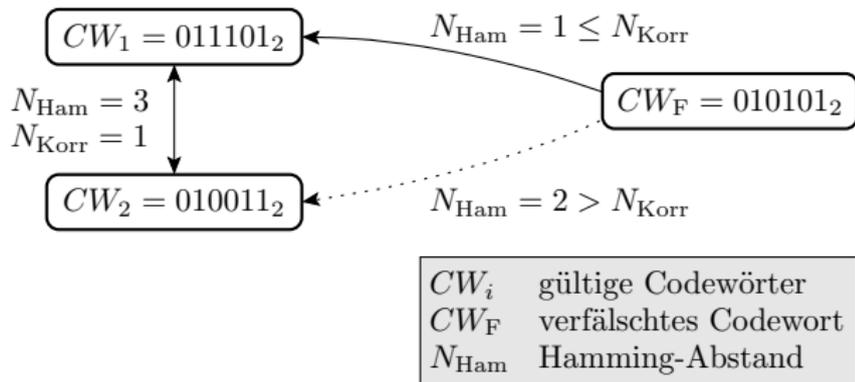
Querparität

Korrektur: Bit in Zeile 5, Spalte 7 invertieren (null setzen).



## 1-Bit fehlerkorrigierende Hamming-Codes

Ab einem Hamming-Abstand  $\geq 3$  ist jeder 1-Bit-Verfälschung eindeutig einem gültigen Codewort zugeordnet.



Korrektor durch Ersatz des verfälschten Codeworts durch das mit Hamming-Distanz  $N_{\text{Ham}} = 1$ . Bei Hamming-Distanz  $N_{\text{Ham}} = 3$  werden 2-Bit-Verfälschungen bei der Korrektur falschen gültigen Codeworten zugeordnet.



## Konstruktion eines 1-Bit fehlerkorrigierenden Codes

Zusammensetzen des Gesamtcodeworts aus

- einem Datenwort mit minimaler Bitanzahl

$$w \geq \log_2(N_{\text{CWG}})$$

und einem Prüfkennzeichen der Größe

$$r = N_{\text{Bit}} - w$$

Bit, das aus dem Datenwort mit mod-2 Summen berechnet wird.

- Wahl der mod-2 Summen so, dass bei einer Bitverfälschung die mod-2 Summe des gesendeten und empfangenen Prüfzeichens die binärcodierte Bitnummer der Verfälschung ist:

verfälschtes Bit	1	2	3	4	5	...
Prüfkennzeichendifferenz $\Delta \mathbf{q}$	0001	0010	0011	0100	0101	...



verfälschtes Bit	1	2	3	4	5	...
Prüfkennzeichendifferenz $\Delta \mathbf{q}$	0001	0010	0011	0100	0101	...

Beispiel:  $w = 8$ ,  $r = 4$ ,  $\mathbf{q} = q_3q_2q_1q_0$

$$\Delta q_0 = b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11}$$

$$\Delta q_1 = b_2 \oplus b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11}$$

$$\Delta q_2 = b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_{12}$$

$$\Delta q_3 = b_8 \oplus b_9 \oplus b_{10} \oplus b_{11} \oplus b_{12}$$

Das erste Bit jeder Summe sei das Prüfbit  $q_i$ . Die restlichen sind Datenbits. Ohne Verfälschung ist die Differenz null.

Beispielzuordnung:

$b_1$	$b_2$	$b_3$	$b_4$	$b_5$	$b_6$	$b_7$	$b_8$	$b_9$	$b_{10}$	$b_{11}$	$b_{12}$
$q_0$	$q_1$	$x_0$	$q_2$	$x_1$	$x_2$	$x_3$	$q_3$	$x_4$	$x_5$	$x_6$	$x_7$



$b_1$	$b_2$	$b_3$	$b_4$	$b_5$	$b_6$	$b_7$	$b_8$	$b_9$	$b_{10}$	$b_{11}$	$b_{12}$
$q_0$	$q_1$	$x_0$	$q_2$	$x_1$	$x_2$	$x_3$	$q_3$	$x_4$	$x_5$	$x_6$	$x_7$

Für die erste Summe gilt:

$$\begin{aligned}\Delta q_0 &= b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} \\ 0 &= q_0 \oplus x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6 \\ q_0 &= x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6\end{aligned}$$

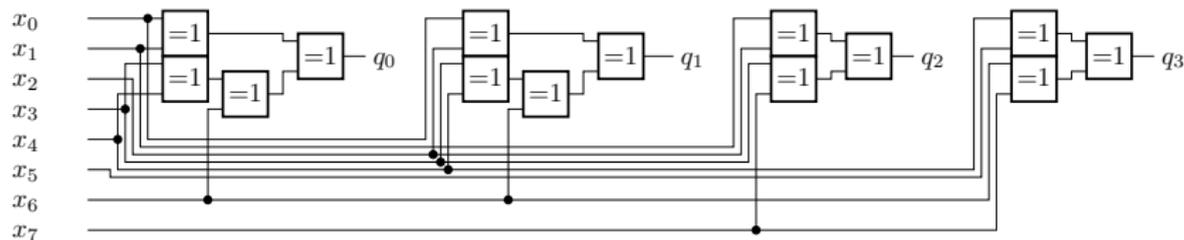
Wie lauten die Bildungsregeln für  $q_1$  bis  $q_3$ ? (an der Tafel anhand der Folie zuvor herleiten)

---

$$\begin{aligned}q_1 &= x_0 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 \\ q_2 &= x_1 \oplus x_2 \oplus x_3 \oplus x_7 \\ q_3 &= x_4 \oplus x_5 \oplus x_6 \oplus x_7\end{aligned}$$



Codierschaltung:



Die Korrekturschaltung besteht aus demselben Coder wie zur Bildung von  $\mathbf{q} = q_3 \dots q_0$ . Vergleich durch bitweises EXOR des empfangenen und des im Empfänger gebildeten Prüfzeichens. Invertierung des verfälschten Bits.

Bitfehlerkorrektur als VHDL-Case-Anweisung:

```
case (q_Empf xor q_berechnet) is
  when "0011" => x(0) <= not x(0);
  when "0101" => x(1) <= not x(1);
  ...
  when others => null;
end case;
```



## Beispielaufgabe



$b_{12}$	$b_{11}$	$b_{10}$	$b_9$	$b_8$	$b_7$	$b_6$	$b_5$	$b_4$	$b_3$	$b_2$	$b_1$
$x_7$	$x_6$	$x_5$	$x_4$	$q_3$	$x_3$	$x_2$	$x_1$	$q_2$	$x_0$	$q_1$	$q_0$

$$q_0 = x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6$$

$$q_2 = x_1 \oplus x_2 \oplus x_3 \oplus x_7$$

$$q_1 = x_0 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6$$

$$q_3 = x_4 \oplus x_5 \oplus x_6 \oplus x_7$$

- 1 Bilden Sie für den Werte  $w_1 = 0x8B$  das Codewort.
- 2 Bestimmen Sie für das Codewort  $c_2 = 0xA9B$  den Wert.

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	$x_7$	$x_6$	$x_5$	$x_4$	$q_3$	$x_3$	$x_2$	$x_1$	$q_2$	$x_0$	$q_1$	$q_0$	
Kontrollbits	=	=	=	=	=	=	=	=	=	=	=	=	
$w_1 = 0x8B$													$c_1 =$
$c_2 = 0xA9B$													$d q_2 =$



## Lösung

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	$x_7$	$x_6$	$x_5$	$x_4$	$q_3$	$x_3$	$x_2$	$x_1$	$q_2$	$x_0$	$q_1$	$q_0$	
Kontrollbits	<u>—</u>												
$w_1 = 0x8B$	1	0	0	0	1	1	0	1	1	1	0	1	$c_1 = 0x8DD$
$c_2 = 0xA9B$	1	0	1	0	1	0	0	1	1	0	1	1	$d q_2 = 12_{10}$

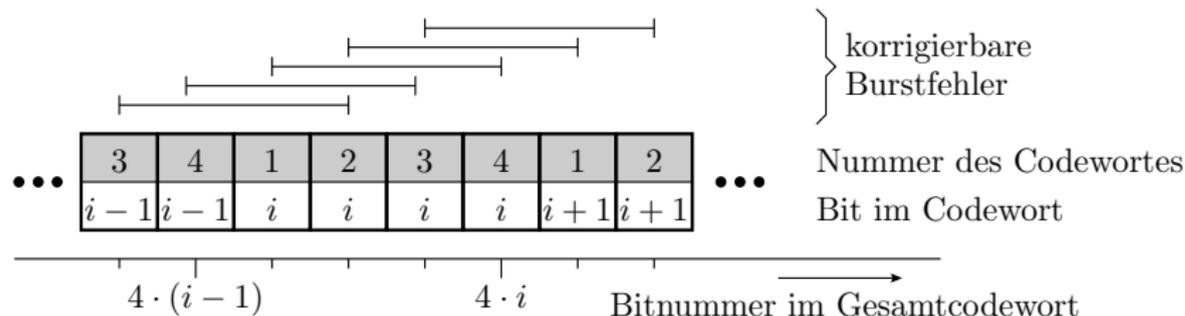
$w_2$  : Wert  $0xA2$  mit verfälschtem  $x_7 \Rightarrow w_2 = 0x22$



## Korrektur von Burstfehlern

- Bei der Datenübertragung, beim Lesen von CDs, ... ist oft eine Folge aufeinanderfolgender Bits verfälscht.
- Burst-Fehler: In einer Bitfolge sind an einer Stelle bis zu  $m$  aufeinanderfolgende Bits verfälscht.

Zusammensetzen eines fehlerkorrigierenden Codes für  $m$ -Bit-Burst-Fehler für eine  $m \cdot n$  Bit lange Folgen aus  $m$  fehlerkorrigierenden Codeworten für 1-Bit-Fehler für  $n$  Bit lange Folgen durch Verschränkung:





## Beispielaufgabe



- 1 Codierung Datenfolge 0x8B, 0x22, 0x9C so, dass bis zu 3-Bit lange Burstfehler korrigierbar sind, durch Verschränkung von je drei aufeinanderfolgenden H8-12-Codeworten.
- 2 Zeigen Sie, dass eine Invertierung der Bits 30 bis 32 korrigiert wird.

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1
Zuordnung	$x_7$	$x_6$	$x_5$	$x_4$	$q_3$	$x_3$	$x_2$	$x_1$	$q_2$	$x_0$	$q_1$	$q_0$
Kontrollbits	=	=	=	=	=	=	=	=	=	=	=	=
$w_1 = 0x8B$												
$w_2 = 0x22$									1			
$w_3 = 0x9C$												

■ Bits 30 bis 32



## Lösung

1

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1
Zuordnung	$x_7$	$x_6$	$x_5$	$x_4$	$q_3$	$x_3$	$x_2$	$x_1$	$q_2$	$x_0$	$q_1$	$q_0$
Kontrollbits	<u>—</u>											
$w_1 = 0x8B$	1	0	0	0	1	1	0	1	1	1	0	1
$w_2 = 0x22$	0	0	1	0	1	0	0	1	1	0	1	1
$w_3 = 0x9C$	1	0	0	1	0	1	1	0	1	0	0	0

2

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	$x_7$	$x_6$	$x_5$	$x_4$	$q_3$	$x_3$	$x_2$	$x_1$	$q_2$	$x_0$	$q_1$	$q_0$	
Kontrollbits	<u>—</u>												
$w_1 = 0x8B$	1	0	0	0	1	1	0	1	1	1	0	1	$d q_1 = 11_{10}$
$w_2 = 0x22$	0	0	1	0	1	0	0	1	1	0	1	1	$d q_2 = 11_{10}$
$w_3 = 0x9C$	1	0	0	1	0	1	1	0	1	0	0	0	$d q_3 = 10_{10}$

■ Durch Burstfehler invertierte Bits 30 bis 32,



# RAID Systeme



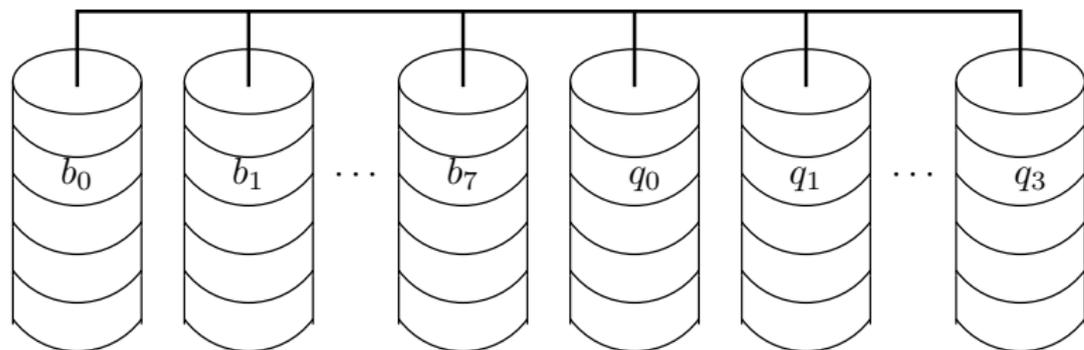
## RAID, RAID Level 1

RAID – **R**edundant **A**rray of Independent **D**isks. Anwendung der behandelten Codes zur Korrektur bei Datenspeicherung auf Festplatten.

RAID Level 1: Zwei gespiegelte Festplatten. Die Daten werden versetzt geschrieben, so dass das Schreiben etwas länger dauert, aber mit nahe doppelter Geschwindigkeit gelesen werden kann. Bei Ausfall einer Platte existieren alle Daten noch auf der zweiten Festplatte. Die Lesegeschwindigkeit reduziert sich, aber das System bleibt funktionsfähig.

## RAID Level 2

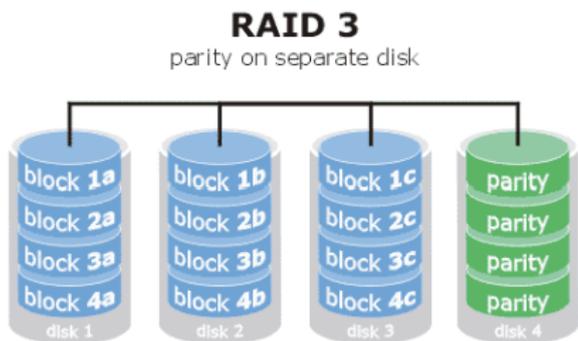
Bei RAID Level 2 werden die Daten in einem 1-Bit-korrigierenden Hamming-Code gespeichert, und zwar jedes der  $w$  Daten- und der  $r$  Kontrollbits auf einer anderen Platte, z.B.  $w = 8$  Datenbit- und  $r = 4$  Kontrollbitplatten. Im Vergleich zu RAID 1 werden statt der doppelten Plattenanzahl nur 50% mehr Platten benötigt.



Gilt als aufwändig und ungebräuchlich.

## RAID Level 3

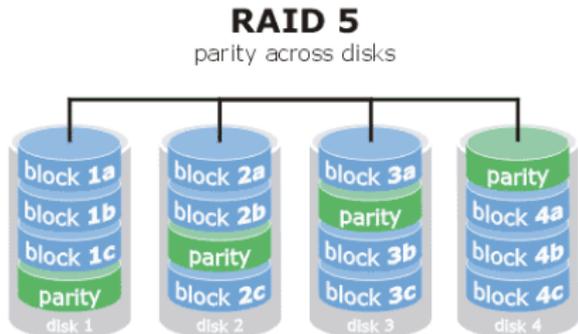
Auf einer Extra-Platte wird bitweise die Querparitätsbits der anderen Platten gebildet. Zusätzlich werden auf jeder Platte die Längsparitätsbits (oder Prüfkennzeichen) gespeichert. 1-Bit-Fehlerkorrektur nach dem Prinzip der Kreuzparität. Erlaubt die Tolerierung eines einzelnen Plattenausfalls.





## RAID Level 5

Fehlertoleranz ähnlich wie Level 3, nur dass Datenzugriffe durch unabhängige Lese- und Schreiboperationen (statt ausschließlich parallel) erlaubt sind. Größere schreibbare Datenblöcke. Die Paritätsinformation verteilt sich auf alle Platten. Gleichfalls tolerant gegenüber einem einzelnen Plattenausfall. Am häufigsten genutzte RAID-Struktur.





## RAID ist kein Backup-Ersatz

Backup: Sicherungskopien von (wichtigen / aufwändig neu zu erzeugenden) Daten. Typisch:

- Tägliche automatische Erstellung durch das Rechenzentrum.
- Nur Änderungen zum letzten Backup.
- Aufbewahrung mehrerer Versionen an einem getrennten Ort.

Wird benötigt zur Datenwiederherstellung nach

- gleichzeitiger Zerstörung aller Platten z.B. durch Überspannungsspitzen, Feuer, ...
- Diebstahl von Datenträgern,
- einem versehentlichen Löschen, das erst nach Stunden oder Wochen bemerkt wird.

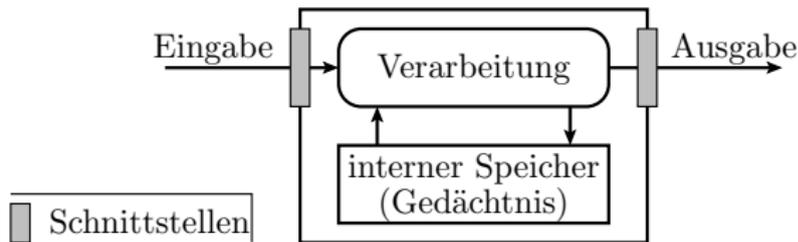


# Formatkontrollen



### Service-Modell (Wiederholung)

Laut TV-F1, Abschn. 1.1 bildet ein Service aus Eingaben Ausgaben. Die Ein- und Ausgaben sind ganz allgemein bedatete Objekte mit einem auf die Art des Services abgestimmten Format. Ein Service kann, aber muss kein Gedächtnis haben<sup>5</sup>.

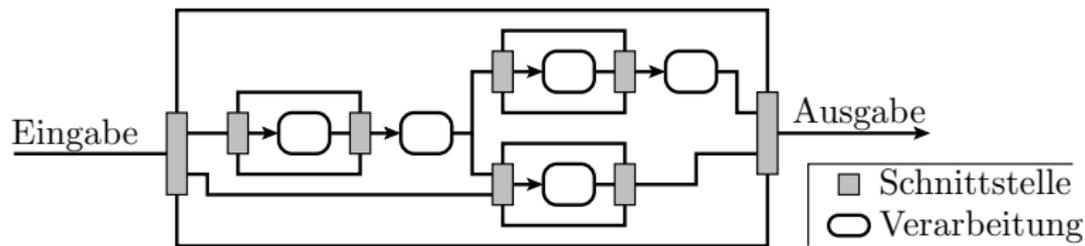


Die Formate erlauben die werteunabhängige Überwachung von Ein- und Ausgaben. Kontrollen auf Zulässigkeit, nicht auf Richtigkeit.

<sup>5</sup>Modell anwendbar auf Hardware, Software bis hin zu cyber-physikalischen Systemen.

### Schnittstellen und Kontrollen

Komplexe Service-Leistungen nutzen bei ihrer Abarbeitung Teil-Service-Leistungen. Dabei »erbt« der übergeordnete Service die Fehler, Fehlfunktionen und Ausfälle der Teil-Service-Leistungen.



Jeder Service und Teil-Service hat Ein- und Ausgabeschnittstellen mit einem Datenformat, das Kontrollmöglichkeiten bietet.

Die besten Formatkontrollen bieten fehlererkennende Codes und Prüfkennzeichen mit  $p_E = 1 - 2^{-r}$  ( $r$  – Anzahl der zusätzlich zu speichernden oder zu übergebenden Bits, vergl. Gl. 11 und 19).



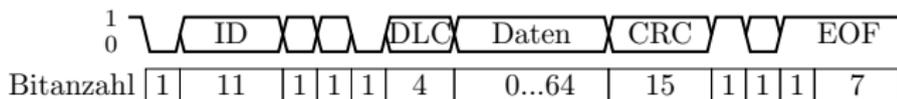
# Datenpaketformate mit Prüfkennzeichen

### Ethernet-Paket:

Sicherungsschicht			MAC-Empfänger	MAC-Absender	Protokolltyp	Nutzlast max. 1500 Bytes	Prüfkennz. CRC	
Bitübertragungsschicht	Präambel	Startbyte						Lücke zum nächsten Paket
Byteanzahl	7	1	6	6	2	46 bis 1500	4	12

- Anteil des Prüfkennzeichens an der Gesamtbitanzahl: 0,3...10%
- Fehlererkennungssicherheit:  $p_E = 1 - 2^{-32} \approx 1 - 2 \cdot 10^{-10}$

### CAN-Bus-Nachricht<sup>6</sup>:



ID    Nachrichtennummer                      CRC    Prüfkennzeichen  
 DLC    Datenlängencode                      EOF    Ende des Datenrahmens

- Anteil des Prüfkennzeichens an der Gesamtbitanzahl: 13...34%
- Fehlererkennungssicherheit:  $p_E = 1 - 2^{-15} \approx 1 - 3 \cdot 10^{-5}$

<sup>6</sup>Lokaler Bus zur Vernetzung von z.B. Steuergeräten in Autos.



## 2. Formatkontrollen

Fehlererkennende Codes und Prüfkennzeichen haben verfahrensbedingte Grenzen:

- Sie sind nur für größere Datenpakete zweckmäßig und
- erkennen nur Verfälschungen, die nach der Berechnung und Codierung entstehen.

Weitere Möglichkeiten für Formatkontrollen:

- Syntaxtest: Definition der Formate manueller Eingabedaten als formale Sprache. Kontrolle mit spracherkennenden Automaten.
- Typ- und Wertebereichskontrollen der Ein- und Ausgaben von Verarbeitungsfunktionen.
- Überwachung der Zeit- und Wertetoleranzfenster (insbesondere bei Signalen<sup>7</sup>).

---

<sup>7</sup>Ein Signal ist ein zeitlicher Werteverlauf.



# Syntaxtest



## Formale Sprachen

Eine formale Sprache definiert zulässige Worte durch Syntaxregeln.  
 Beispiel EBNF (Erweiterte Backus-Naur-Form<sup>8</sup>):

Verwendung	Zeichen
Definition	NTS = Ersetzungsregel
Aufzählung	..., ...
Endezeichen	...;
Alternative	... ...
Option	[...]
Wiederholung	{...}
Gruppierung	(...)
Zeichenkette (Terminalsymbolfolge)	"..." oder '...'

(NTS – zu ersetzendes Symbol, Nicht-Terminalsymbol).

<sup>8</sup>Siehe <http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf>.



## Beispiele für EBNF-Syntaxregeln

```
Zahl = ["-"], (ZiffernAusserNull, {Ziffer}) | "0";
```

```
ZiffernAusserNull = "1"|"2"|"3"| ... |"9";
```

```
Ziffer = ZiffernAusserNull | "0";
```

```
Bezeichner = Buchstabe, {Buchstabe | Ziffer};
```

- 
- Syntaxregeln sind Formatvorgaben.
  - Die Kontrolle erfolgt mit spracherkennenden Automaten.
  - Erkennbar sind alle Regelverletzungen.
  - Ein Syntaxtest erkennt einen erheblichen Teil der durch Menschen verursachten Eingabefehler, Programmierfehler, ...

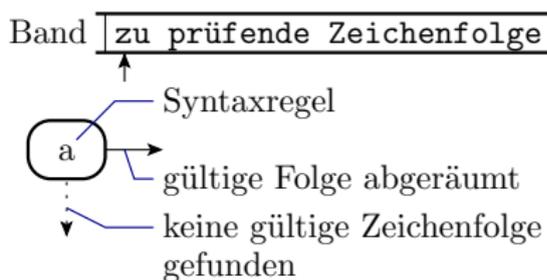
# Spracherkennende Moore-Automaten

Die zu prüfende Zeichenfolge liegt auf einem Band mit einem Zeiger auf dem Anfang.

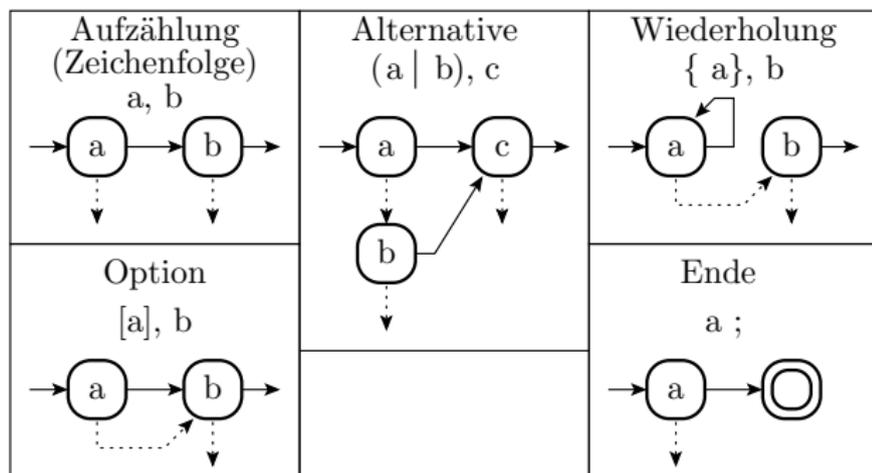
In jedem Automatenzustand wird versucht, eine Zeichenfolge nach der definierten Syntaxregel abzuräumen:

- Wenn möglich, wandert der Zeiger zum ersten Zeichen nach der abgeräumten Folge und der Knoten wird über  $\rightarrow$  verlassen.
- Sonst bleibt der Zeiger und der Knoten wird über  $\downarrow$  verlassen.

$\downarrow$ -Übergänge ohne Alternative enden im Fehlerzustand.



## Von der EBNF zum Automaten



Beispiel:

```
Zahl = [ '- ' ], (z1-9, {z}) | '0';
```

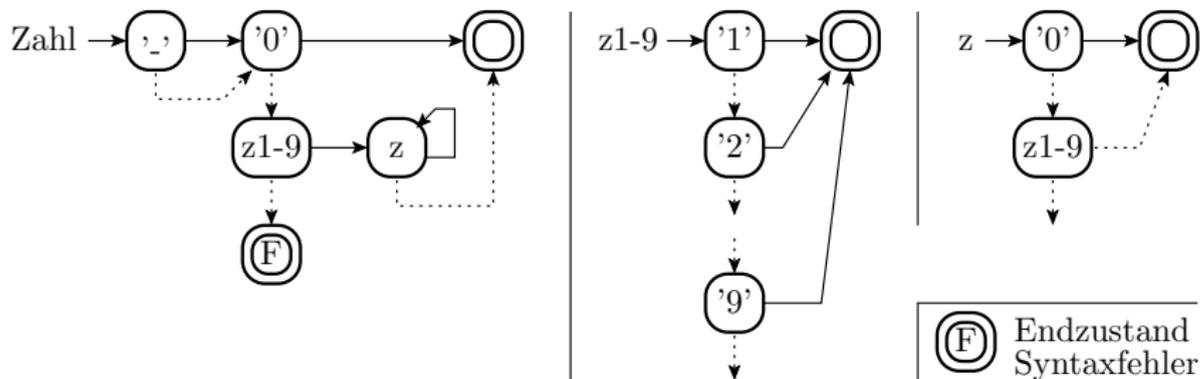
```
z1-9 = '1' | '2' | ... | '9';
```

```
z    = z1-9 | '0';
```

Zahl = ['-', ''], (z1-9, {z}) | '0';

z1-9 = '1' | '2' | ... | '9';

z = z1-9 | '0';



Welche Zustände durchläuft der Automat. Was erkennt er?

- "125\_": '-' ↓, '0' ↓, z1-9 →, z →, z →, z ↓, Endzustand »ok«
- "k89": '-' ↓, '0' ↓, z1-9 ↓, Endzustand Syntaxfehler
- "-0701": '-' →, '0' →, Endzustand »ok« (»-0« gefunden)

## Spracherkennender Mealy-Automat

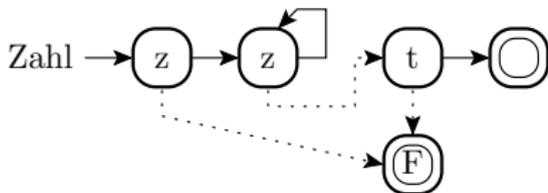
- Abräumen der Zeichen an den Kanten.
- Beschreibung derselben Syntaxregeln mit weniger Zuständen.
- Die »Sonst-Kanten« zum Fehlerzustand können weggelassen werden.

Beschreibung einer Zahl:

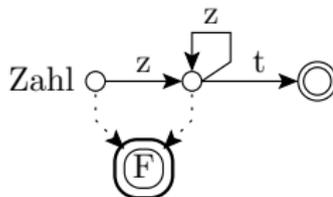
Zahl = z, {z}, t;       $z = '0' \mid '1' \mid \dots \mid '9'$ ;

$t = ' ' \mid ', ' \mid \dots$ ; (Trennzeichen)

Moore-Automat



Mealy-Automat





Mit den einfachen Ersetzungsregeln »darf sein«, »darf  $n$ -mal vorkommen« lassen sich viele Datenformate und auch Formate für Programme, die genau wie Daten gespeichert und übertragen werden, beschreiben.

Ein Programm für die Spracherkennung und Datenextraktion, das auch noch verständliche Fehlermeldungen generiert, wird zwar schnell groß und kompliziert, lässt sich aber automatisch aus der Sprachbeschreibung generieren.

Umgekehrt lässt sich eine Sprache auch so definieren, dass die Erkennung sehr einfach ist.

Ein Syntaxtest erkennt alle Verletzungen der Syntaxregeln. Idealerweise sind alle syntaktisch korrekten Daten weiterverarbeitbar, aber nicht unbedingt richtig.

Ein Syntaxtest erkennt grob geschätzt die Hälfte der Eingabefehler durch Menschen bei der Datenerfassung.



### Standardisierte Sprachen

Sprachen für die manuelle Datenerfassung:

- Programmiersprachen (C, Java, ...)
- XML (Extensible Markup Language) zur Darstellung hierarchischen Darstellung strukturierter Daten in Form von Textdateien.
- CVS: Beschreibung tabellarischer Daten, ...

Für automatisch generierte Daten, die nur aufzubewahren oder weiterzugeben, aber nicht für eine manuelle Bearbeitung vorgesehen sind, eignen sich fehlererkennende Codes und Prüfkennzeichen besser. Weniger Programmier- und Rechenaufwand. Höhere Fehlererkennungssicherheit.



## Typ und Wertebereich



## Datentypen

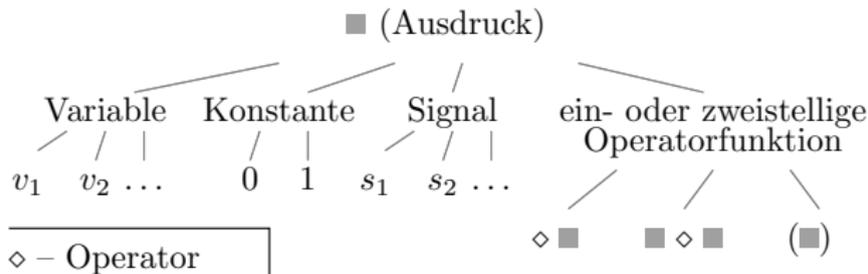
Der Datentyp beschreibt in einer Hochsprache alle von der Übersetzung und der Zielarchitektur unabhängigen Formateigenschaften. Datentyparten:

- Elementare Typen:
  - Zahlentypen (ganzzahlig mit/ohne Vorzeichen, Festkomma- und Gleitkommazahlen) und
  - Aufzählungstypen.
- Zusammengesetzte Typen:
  - Felder: Zusammenfassung typgleicher Elemente
  - Strukturen: Elemente unterschiedlichen Typs.
  - Klassen: Datenstrukturen mit Bearbeitungsmethoden z.B. Listen mit den Methoden Element hinzufügen, löschen, ...

Eine Kernidee der höheren Programmiersprachen ist, dass für jede Verarbeitungsoperation festgelegt ist, welchen Typ jeder Operand haben muss und welchen dann das Ergebnis erhält.

## Typenkontrollen

- Kontrolle in Zuweisungen: gleicher/zuordenbarer Typ von Zuweisungsziel und zugewiesenem Wert.
- Kontrollen in Ausdrücken<sup>9</sup>:



- Operationen sind nur mit bestimmten Operandentypen erlaubt.
  - Aus den Operandentypen ergibt sich der Ergebnistyp.
- Typenkontrolle an Schnittstellen: Typengleichheit zwischen übergebenem und übernommenem Wert.

<sup>9</sup>Rekursive Berechnungsvorschrift.



## Wertebereichskontrolle

Daten in einem Programm / einer Schaltungsbeschreibung haben in der Regel viel kleinere Wertebereiche als Darstellungsbereiche:

- Altersangabe für eine Person mit einer 32-Bit-Integer-Zahl ( $\approx 120$  zulässige und  $2^{32}$  darstellbare Werte)
- Zeiger auf ein  $n$  Elemente großes Feld in einem 32-Bit-System; zulässig sind nur die Werte

$$w = a + g \cdot i \quad \text{mit} \quad i \in \{0, 1, \dots, n-1\}$$

( $a$  – Feldanfang;  $g$  – Elementgröße); darstellbar  $2^{32}$  Werte

Wertebereichskontrollen bestehen aus Fallunterscheidungen und Anweisungen zur Fehlerbehandlung:

```
if (alter < 0 || alter > 120) {  
    <Anweisungen zu Fehlerbehandlung>  
}
```

Hauptprogrammieraufwand Fehlerbehandlung, i.allg. separate Service-Leistungen oder Hardware-Funktionen, z.B. Systemrufe.



## Erkennungswahrscheinlichkeit

Beispiel: Darstellung des Alters einer Person,  $WB \in [0, 119]$  mit einer 32-Bit-Integer-Variablen:

Fehlerannahmen:

- Verwechslung mit dem Alter einer anderen Person: meist falsch, immer zulässig, nie nachweisbar.
- Verwechslung mit der Hausnummer: meist falsch, oft zulässig<sup>10</sup>, selten nachweisbar.
- Verwechslung mit anderem gespeichertem Wert. Kleine positive Datenwerte treten überproportional häufig auf.
- Grob geschätzt:

$$p_E < 50\% \ll 1 - \frac{120}{2^{32}}$$

Die Erkennungswahrscheinlichkeit einer Wertebereichskontrolle ist schwer abschätzbar. Die Methode über die Größe der Mengen zulässiger und darstellbarer Werte meist viel zu optimistisch.

<sup>10</sup>Hausnummern sind immer größer null und meist kleiner 119.



## Erhöhung der Erkennungswahrscheinlichkeit

Zur Erhöhung der Erkennungswahrscheinlichkeit von Wertebereichskontrollen sind die zulässigen Werte ähnlich wie bei einem fehlererkennenden Code »zufällig« in der Menge der darstellbaren Werte zu verteilen:

- Verschiebung der Wertebereiche um einen konstanten Wert,
- Multiplikation mit einer Konstanten,
- ...

## Fehlererkennender Code für Aufzählungstypen

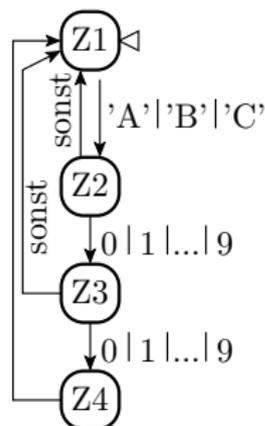
Für Aufzählungstypen, für die es nur die Operationen Zuweisung und Vergleich gibt, bieten sich fehlererkennende Codes an, d.h. die Darstellung mit zufälligen Werten innerhalb einer viel größeren Menge möglicher Werte. Statt der üblichen Zustandskodierung

```
#define EmpfAut_Z1 1
#define EmpfAut_Z2 2
...
```

kostet eine Zufallscodierung der Form

```
#define EmpfAut_Z1 0x35
#define EmpfAut_Z2 0x58
...
```

kaum Mehraufwand und vergrößert die Wahrscheinlichkeit, dass falsche Werte erkannt werden, erheblich.





## Anmerkungen zu Typenkontrollen

Typ- und Wertebereichskontrollen auch manuell einprogrammierbar, z.B. in C für eine Struktur:

```
struct tFeld {
    uint8_t typ; //Konstante für Typ, z.B. 0x4D
    uint8_t len; //Wertebereich 0 bis 100
    int16_t *feld; //Feld Indexbereich 0 bis len-1
} //WB Elemente -10000 bis 10000
```

könnte die Kontrollfunktion lauten:

```
uint8_t check_tFeld(struct tFeld f){
    if (f.typ!=0x4D) return 1;
    if (f.len>100) return 2;
    for (i=0;i<100;i++)
        if ((f.feld(i)<-10000) || (f.feld(i)>10000))
            return i+3;
    return 0;}

```



Die zusätzliche Typ- und Längeninformation würde im Beispiel weiterhin erlauben, beim Lesen und Schreiben den Index oder Zeigerwert auf Zulässigkeit zu kontrollieren, ...

Bei Objektorientierung könnte man alle Schnittstellenklassen von einer Basisklasse »Schnittstelle« mit den Grundfunktionen der Fehlerbehandlung ableiten und für alle abgeleiteten Klassen die Kontrollfunktion und schnittstellenspezifischen Fehlerbehandlungen überladen.

- 
- Einzelne Typen- und Wertebereichskontrollen haben selten hohe Erkennungswahrscheinlichkeiten.
  - Die Leistungsfähigkeit des Verfahrens liegt in der Vielzahl der Kontrollmöglichkeiten.

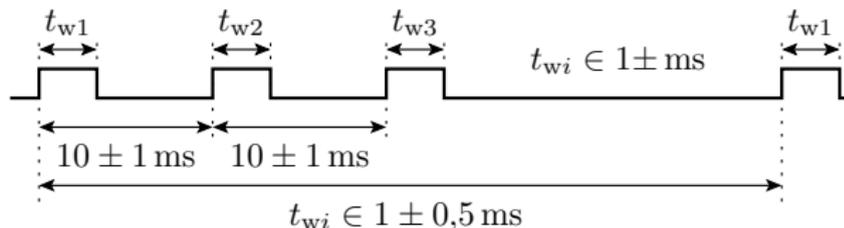


# Signalüberwachung



Auf der Schaltungsebene kommunizieren IT-Systeme über Signale. Ein Signal ist ein zeitlicher Werteverlauf einer physikalischen Größe. Die zu übergebende Information steckt in Merkmalen: Zeitverlauf der Amplitude, Frequenz, ... Auch hier teilt sich die Darstellung ein in

- Format (Gültigkeitsmerkmale) und
- Merkmale, die Werte repräsentieren.

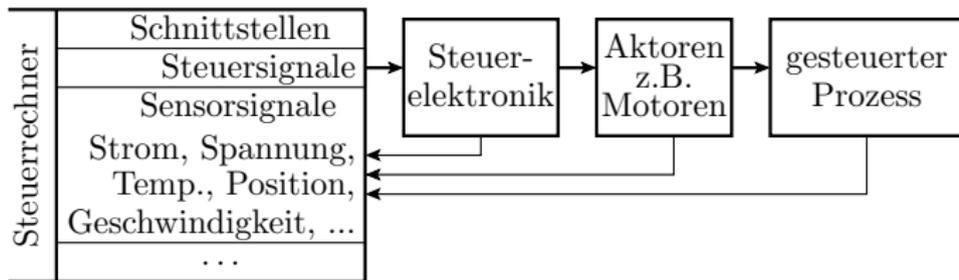


Das dargestellte PWM-Signal besteht aus einer Pulsfolge, bei der der Low-Pegel, der High-Pegel, der Pulsabstand und die Pulsbreite alle in einem bestimmten und damit kontrollierbaren Toleranzbereich liegen müssen. Die Information steckt typisch in der Pulsbreite, dem Abstand oder beidem.



## Überwachung analoger Signale

Analoge Signale hat man im Wesentlichen als Sensoreingaben.



- Bereichsüberwachung.
- Überwachung von Pulsbreiten.

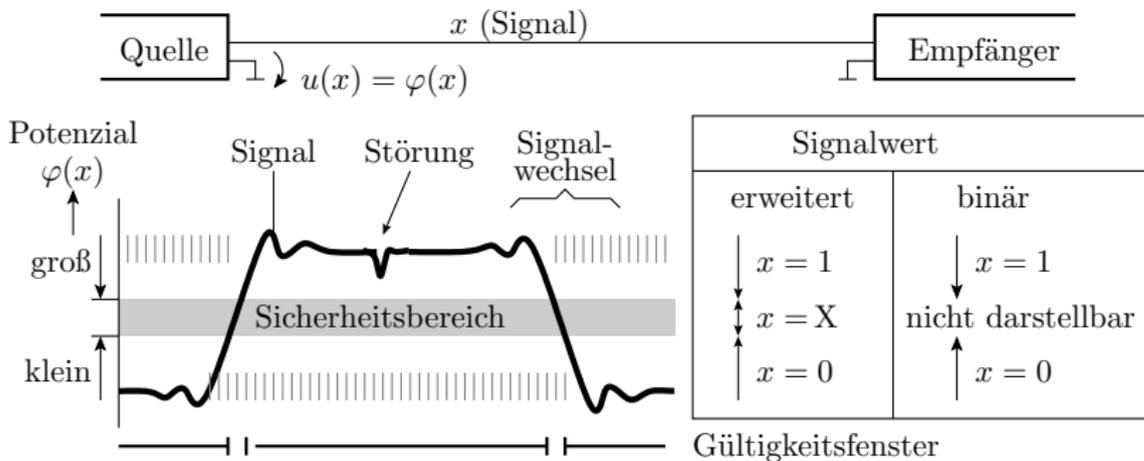
Die Überwachung komplexer Merkmale, z.B. der Stromsignaturen an Schrittmotoren auf Schrittfehler, erfolgt erst nach der Digitalisierung und ist z.T. sehr rechenintensiv<sup>11</sup>.

<sup>11</sup>Bei CP-Systemen sind Überwachungsfunktion zur Sicherung der Verlässlichkeit oft Bestandteil der Zielfunktion.



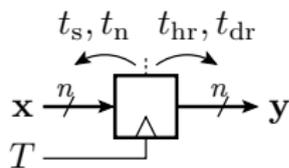
## Digitale vs. analoge Signale

Die Digitalisierung schafft einen Zwischenbereich zwischen den gültigen Werten. Jedes Gatter im Signalfluss korrigiert alle tolerierbaren Eingabeverfälschungen, die z.B. durch Rauschen und andere Störungen entstehen. Erweiterung um eine Erkennung ohne Korrektur ist möglich.



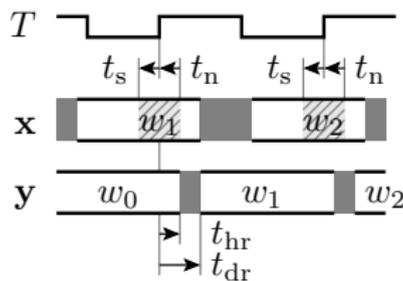


Abtastregister sorgen zusätzlich dafür, dass alle Signalverfälschungen außerhalb der Abtastfenster toleriert werden.



$t_s$  Vorhaltezeit  
 $t_n$  Nachhaltezeit  
 $t_{hr}$  Haltezeit

$t_{dr}$  Verzögerungszeit  
 // Abtastfenster  
 ■ Wert ungültig



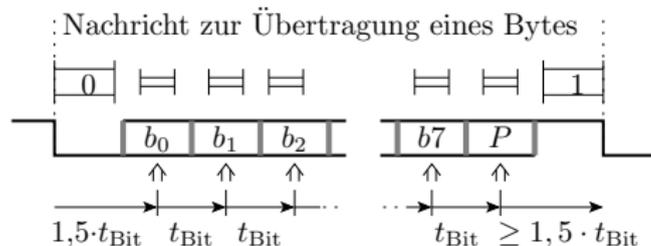
Bei analogen Signalen summieren sich die Verfälschungen durch Störungen, Rauschen, ... Die nachträgliche Trennung der Nutzsignale von überlagernden (zufälligen) Störsignalen ist schwierig, aufwändig, bzw. nur begrenzt möglich.

Auch deshalb erfolgt Informationsverarbeitung wenn möglich digital.

## Überwachung für ein serielles Protokoll

Das alte Protokoll für serielle Schnittstellen, heute noch bei Mikrorechnern verbreitet, beschreibt eine byteweise Übertragung als Folge aus

- Startbit,
- acht Datenbits,
- optionalem Paritätsbit und
- Stoppbit(s).



Überwachungsmöglichkeiten:

0	muss 0 sein
≡	muss konstant sein
1	muss 1 sein
$P$	muss $b_0 \oplus \dots \oplus b_7$ sein
↑	Abtastzeitpunkte
	ungültig

Um keine richtigen Signalverläufe als falsch zu klassifizieren, auch Werteüberwachung nur innerhalb der Gültigkeitsfenster.



# Wertekontrollen



### Wertekontrollen

Die bisher behandelten Überwachungsverfahren überprüfen das Format, d.h. die Zulässigkeit der Ein- und Ausgaben:

- Fehlererkennende Codes: Zulässiges Codewort?
- Syntaxtest: Ist die Eingabe ein Wort der Eingabesprache?
- Typkontrolle: Zulässiger Datentyp?
- Wertebereichskontrolle: Zulässiger Wert?
- Signalüberwachung: Zulässige Zeit- und Wertfenster?

In diesem Abschnitt werden Überwachungsverfahren für Ergebnisse auf Richtigkeit behandelt:

- Vergleich mit Sollwerten<sup>12</sup>,
- Mehrfachberechnung und Vergleich,
- Rückrechnung Eingabe aus Ausgaben + Vergleich und
- Kontrolle von Richtigkeitskriterien.

---

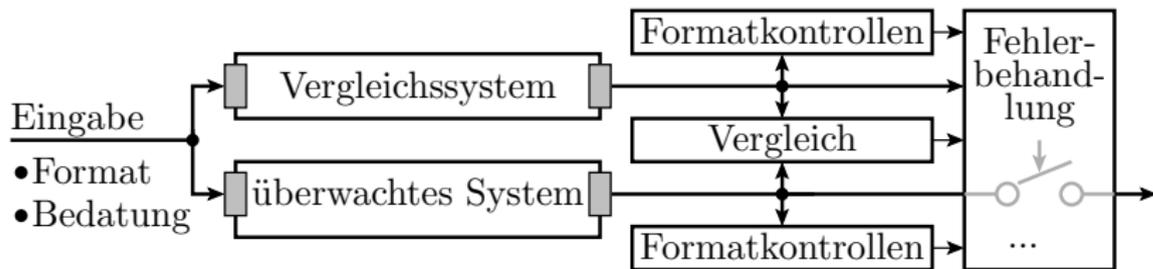
<sup>12</sup>Erfordert Testbedingungen (siehe später Foliensatz F5).



## Mehrfachber. & Vergleich

## Mehrfachberechnung und Vergleich

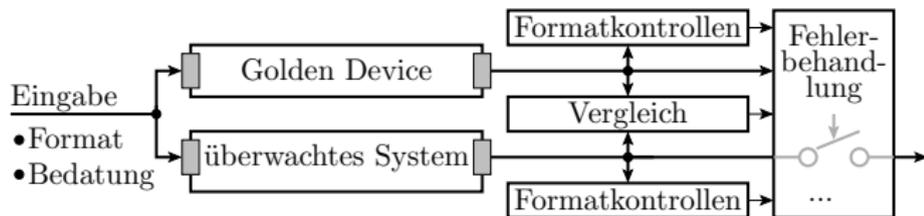
Die Ergebnisse werden mehrfach berechnet. Übereinstimmende Ergebnisse gelten als fehlerfrei. Bei Abweichung kann über einen Mehrheitsentscheid oder Zusatzkontrollen ein Ergebnis als richtig gewertet und weitergereicht werden.



Das Vergleichssystem kann sein:

- ein »Golden Device« (Ausgaben per Definition richtig) oder
- ein reales System, dessen Ausgaben genau wie die des überwachte System fehlerhaft sein können.

## Vergleich mit einem »Golden Device«

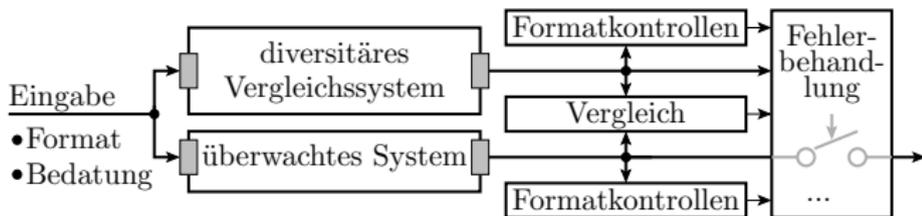


Ein »Golden Device« ist ein gründlich getestetes System, ein Mustergerät oder bei einem Regressionstest die Vorgängerversion. Seine Ausgaben gelten per Definition als richtig. Mit einem »Golden Device« und einem idealen Vergleicher<sup>13</sup> werden alle Wertabweichungen erkannt.

Ein wirklich immer korrekt arbeitendes System gibt es nicht und Vergleich verlangt korrekte Formate. Deshalb Zusatzkontrollen.

<sup>13</sup>Vergleicher, der übereinstimmende Werte immer als übereinstimmend und abweichende Werte immer als abweichend klassifiziert. In der Realität sind Vergleichsfehler nicht vollständig ausschließbar.

## Vergleich diversitärer Systeme



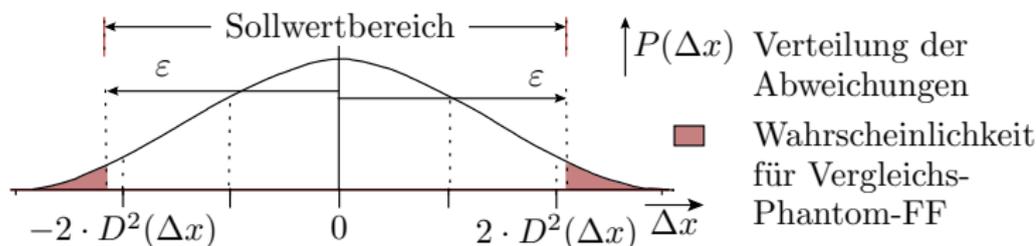
Ein reales System als Vergleichssystem liefert ähnlich häufig falsche Ergebnisse wie das überwachte System. Problematisch sind gleiche Fehler, die die Ausgaben übereinstimmend verfälschen. Denn die werden von einem Vergleich nicht erkannt. Deshalb muss das Vergleichssystem diversitär (verschiedenartig) sein. Diversität entsteht durch unabhängige Entwürfe, unterschiedliche Algorithmen, ... (Fortsetzung Folie 99).

Zufällige Übereinstimmungen sind bei größeren zu vergleichenden Bitvektoren, die fast nie falsch sind, praktisch ausgeschlossen.

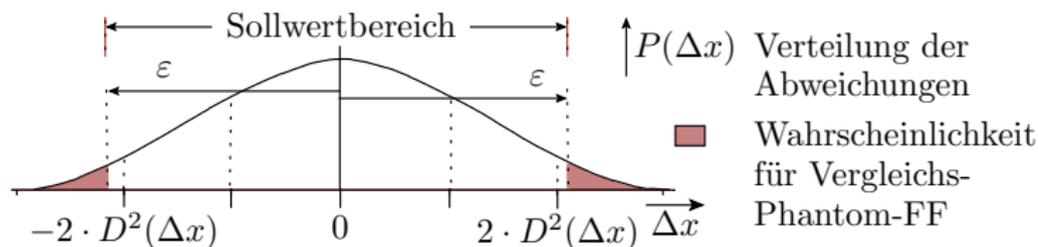
## Exakter Vergleich und Fenstervergleich

Die Sollausgaben eines Systems können exakte Werte oder von zufälligen Einflüssen überlagerte Werte sein:

- Exakte Werte: Ergebnisse von Logik- und ganzzahligen Operationen. Vergleich:  
**if** ( $x \neq \text{Sollwert}$ ) <Fehlerbehandlung>;
- Zufallsgrößen: Analoge Eingaben, Ergebnisse mit Rundungsfehlern, ... verlangen Bereichskontrollen:  
**if** ( $\text{abs}(x - \text{Sollwert}) > \text{eps}$ ) <Fehlerbehandlung>;



## Festlegung des Vergleichsfensters



Für normalverteilte Abweichungen wird für den Sollwertbereich zweckmäßiger Weise  $\pm\epsilon$  mit  $\epsilon \gg \sqrt{D^2(\Delta x)}$  gewählt ( $\sqrt{D^2(\Delta x)}$  – Standardabweichung). Die Phantom-FF-Wahrscheinlichkeit ergibt sich aus der Irrtumswahrscheinlichkeit  $\alpha$  der beideseitigen Bereichsschätzung. Für eine normalverteilte Zufallsgröße:

$\alpha$	4,54%	0,26%	0	4%	2%	1%	0,4%	0,2%
$\Phi^{-1}\left(1 - \frac{\alpha}{2}\right)$	2	3	4	2,05	2,33	2,57	2,88	3,10

(vergl. Foliensatz F2, Abschn. 3.2).



## Quantisierungs- und Rundungsfehler

Bei Quantisierung und Rundung ist ein Fehler von  $\pm 0,5 \text{ LSB}^{14}$  unvermeidbar. Die Standardabweichung als die Wurzel aus der mittleren quadratischen Abweichung ist die reichliche Hälfte davon:

$$\sqrt{D^2(X)} = \sqrt{\int_0^{0,5} x^2 \cdot dx} \cdot \text{LSB} = \frac{1}{\sqrt{12}} \cdot \text{LSB} = 0,29 \cdot \text{LSB}$$

Analog-Digital-Wandler haben oft zusätzlich Linearitätsfehler, ...  
> 1 LSB.

---

<sup>14</sup>LSB (least significant bit) Wert einer Quantisierungsstufe.



## Rundung und Fehlerfortpflanzung

Rundung auf 8 Nachkommabits:

Wert	nächster darstellbarer Wert	Rundungsfehler
125,4380	0x7D,70=125,4375	-0,05%
2,7130	0x2,B7=2,7148	+0,18%
28,2000	0x= 28,1992	-0,08%

Multiplikation der drei auf 8 Nachkommabits gerundeten Werte der Folie zuvor:

$$\begin{aligned}
 125,438 \cdot 2,713 \cdot 28,2 &= 9596,8349 \\
 0x7D,70 \cdot 0x2,B7 \cdot 0x1C,33 &= 0x2569,C7 = 9577,7773 \\
 \text{Abweichung :} &= -19,0575
 \end{aligned}$$

Einige Regeln der Fortpflanzung der Fehlergrenzen:

- Add., Sub.: Addition der absoluten Fehlergrenzen.
- Mult., Div.: Addition der relativen Fehlergrenzen.



Problematisch sind Differenzen, bei denen sich die Werte gegenseitig aufheben, aber die absoluten Fehler addieren.



# Diversität



### Erkennbare Fehlfunktionen

Mehrfachberechnung und Vergleich erkennt fast immer

- FF durch unterschiedliche Eingaben,
- FF durch (unterschiedliche) Störungen,
- FF durch unterschiedliche Hardware (bei Fertigungsfehlern und Ausfällen auch baugleicher HW),
- FF von Berechnungen nach unterschiedlichen Algorithmen,
- FF bei gleichem Algorithmus mit unterschiedlichen Fehlern.

Maskiert werden in der Regel

- FF durch HW-Fehler bei Berechnung mit derselben HW,
- FF durch HW-Entwurfsfehler bei Berechnung mit typgleicher HW,
- FF durch Programmfehler bei Berechnung mit demselben Programm.

(FF – Fehlfunktion; HW – Hardware).



## Diversität

Maskierung durch Zufall ist extrem unwahrscheinlich. Beispiel:

---

Anteil der verfälschten Ergebnisse:	$10^{-4}$
Mögliche Verfälschungen (z.B. > 100 Ergebnisbits):	$\gg 2^{100}$
Wahrscheinlichkeit gleichzeitiger Verfälschung:	$10^{-8}$
Wahrscheinlichkeit zufälliger Übereinstimmung:	$\ll 2^{-100}$
gleichzeitige zufällige Übereinstimmung:	$\ll 10^{-8} \cdot 2^{-100}$

---

Praktisch alle Maskierungen entstehen durch mangelnde  
»Verschiedenartigkeit«, Fachbegriff Diversität<sup>15</sup>.

Diversität sei im weiteren der Kehrwert der  
Nichtererkennungswahrscheinlichkeit durch Verdopplung und Vergleich.

---

<sup>15</sup>Von lateinisch »diversitas« (Vielfalt) abgeleitet.



## Schätzen der Diversität

### Definition der Diversität:

Mittlere Anzahl der aufgetretenen Fehlfunktionen zwischen zwei durch Verdopplung und Vergleich nicht erkennbaren Fehlfunktionen.

$$Div = \frac{1}{1 - p_E} = \frac{N_{FF}}{N_{FF.NErk}}$$

$p_E$  – Erkennungswahrscheinlichkeit;  $N_{FF}$  – Anzahl aller Fehlfunktionen;  $N_{FF.NErk}$  – Anzahl der durch Dopplung und Vergleich nicht erkannten FFs.

Praktische Probleme bei der Abschätzung der Diversität:

- Zählen der nicht erkannten FFs erfordert eine Kontrolle mit  $p_E = 100\%$ , z.B. Soll-/Ist-Vergleich mit Golden Device.
- Eine aussagekräftige Schätzung verlangt ausreichend große Zählwerte für  $N_{FF.NErk}$  (siehe Foliensatz F2, Abschn. 2.8).



### Arten von Diversität

Jede Berechnung hat eine natürliche Diversität durch zufällig wirkende Einflüsse:

- Bitverfälschungen bei der Speicherung und Übertragung.
- FF durch Übertaktung, Überspannung, ...
- FF durch andere auf dem Rechner laufende Programme.

FF mit natürlicher Diversität sind durch Wiederholung und Vergleich erkenn- und korrigierbar.

Nutzbare Arten geplanter Diversität:

- Hardware-Diversität: Berechnung auf unterschiedlicher Hardware. Erkennbar sind zusätzlich FF durch Fertigungsfehler und Ausfälle der HW.
- Hardware-Entwurfsdiversität: Berechnung mit unabhängig voneinander entworfener HW. Zusätzlich FF durch HW-Entwurfsfehler.



- Syntaktische Diversität: Berechnung mit unterschiedlich übersetzter Software. Zusätzlich FF durch den Compiler.
- Software-Diversität: Berechnung mit unabhängig voneinander entworfener SW. Zusätzlich FF durch SW-Entwurfsfehler.

Nicht als »Diversität« nutzbare Verschiedenartigkeiten:

- Berechnung mit unterschiedlichen Eingaben und/oder
- nach unterschiedlichen Algorithmen,

bei denen unterschiedliche richtige Ergebnisse entstehen.



### Schaffung von Diversität

Der praktische Gestaltungsspielraum für diversitäre Entwürfe:

- Unterschiedliche Programmiersprachen und -werkzeuge,
- unterschiedliche Programmübersetzung,
- unterschiedliche Zielhardware,
- unterschiedliche Funktionsbibliotheken,
- unterschiedliche Entwurfsmethoden,
- unterschiedliche Algorithmen
- unterschiedliche Projektteams, ...



## Maximal erzielbare Diversität

Die ursprüngliche euphorische Meinung, durch Diversität lassen sich praktisch alle Fehler mit gleicher Wirkung, außer denen in der Spezifikation ausschließen, hat sich nicht bestätigt. Die direkte oder indirekte Kommunikation der Entwicklungsteams über die Interpretation der Spezifikation, während des Test etc. trägt Gemeinsamkeiten in die Entwürfe. Auch die Neigung von Menschen, gewisse Fehler zu wiederholen, trägt dazu bei. Erzielbare Erkennungssicherheit laut<sup>16</sup>  $p_E \leq 90\%$ :

$$Div = \frac{1}{1 - p_E} \leq 10$$

---

<sup>16</sup>U. Voges, Software-Diversität und ihre Modellierung - Software-Fehlertoleranz und ihre Bewertung durch Fehler- und Kostenmodelle, Springer (1989)



## Syntaktische Diversität

Systematische und automatisch durchführbare Modifikationen an einem funktionsfähigen Programm. Nach<sup>17</sup>:

- Variation der Register- und Speicherzuordnung.
- Variation der Anweisungsreihenfolge bei parallelisierbaren Anweisungen.
- Variation der Bitcodierung von Datentypen.
- Variation der vom Compiler einzubindenden Code-Bausteine. Entwurfsaufwand kaum höher als bei Beschränkung auf natürliche Diversität. Erkennungssicherheit soll höher als bei einfacher Wiederholung und Vergleich sein. Experimentelle Untermauerung fehlt.

---

<sup>17</sup>Heidrun Dücker, Ergebnisvalidierung und nebenläufige Hardwarefehlererkennung mittels systematisch erzeugter Diversität., Diss. Universität Karlsruhe Institut für Rechnerentwurf und Fehlertoleranz.

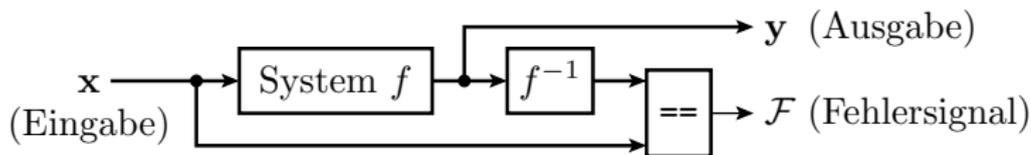


# Loop-Back Test



## Loop-Back Test (Rückrechnung der Eingabe + Vgl.)

Für umkehrbare Funktion  $f(x)$  mit  $f^{-1}(f(x)) = x$  lässt sich das Ergebnis auch dadurch kontrollieren, dass aus dem Ergebnis die Eingabe zurückberechnet und mit den ursprünglichen Service-Eingaben verglichen wird:



Beispiele für Funktionen mit Umkehrfunktion:

- Quadrierung  $\leftrightarrow$  Wurzelberechnung,
- Addition  $\leftrightarrow$  Subtraktion,
- Multiplikation  $\leftrightarrow$  Division,
- Analog/Digital-Wandlung  $\leftrightarrow$  Digital/Analog-Wandlung,
- Daten versenden  $\leftrightarrow$  Daten empfangen, ...



- Im Vergleich zu »Verdopplung und Vergleich« haben Umkehrfunktionen einen anderen Algorithmus und dadurch eine höhere natürliche Diversität. Das lässt eine höhere Erkennungswahrscheinlichkeit erwarten.
- Für Service-Leistungen, für die ein Service mit Umkehrfunktion ohnehin vorhanden ist, z.B. außer dem seriellen Sender auch der passende serielle Empfänger, ist ein Loop-Back Test die naheliegendste und einfachste Lösung.
- Bei zu erwartenden Mess-, Quantisierungs- und numerischen Fehlern Fenstervergleich.



## Kontrollkriterien für Richtigkeit



### Kontrolle auf Richtigkeit

Es gibt Aufgaben, die sind durch Kontrollkriterien für die Richtigkeit der Ergebnisse spezifiziert:

- Suche eine Funktion, die eine Differenzialgleichung erfüllt.  
Kontrolle durch Einsetzen der Funktion in die DGL.
- Suche einen Test, der einen Fehler nachweist. Kontrolle durch Simulation des Systems mit und ohne Fehler und Vergleich der berechneten Ausgaben.
- Suche einen Weg, der alle Knoten eines Graphen verbindet.

Die vorgegebenen Kontrollfunktionen sind oft einfacher und dadurch fehlerärmer zu realisieren als die Suchalgorithmen. Im Entwurfsprozess ist es zweckmäßig, diese zuerst zu programmieren, eventuell sogar diversitär, und sehr gründlich zu testen, um sie später gleichermaßen für die Suche und die Kontrollen des Suchergebnisses zu verwenden.



## Königsberger Brückenproblem

Aufgabe: Suche einen Weg, der alle sieben Brücken über den Pregel genau einmal überquert.

Ergebnis sei eine verkettete Liste der Nummern der Brücken, die zu überqueren sind.

Probe:

- Setze für alle Brücken die Zahl der Überquerungen null
- Wiederhole für jedes Listenelement
  - Anzahl der überquerten Brücke +1
- Wiederhole für jede Brücke
  - Kontrolle, dass die Anzahl eins ist.

