Test und Verlässlichkeit Foliensatz 5: Überwachung und Fehlertoleranz

Prof. G. Kemnitz

Institut für Informatik, TU Clausthal (TV_F5)

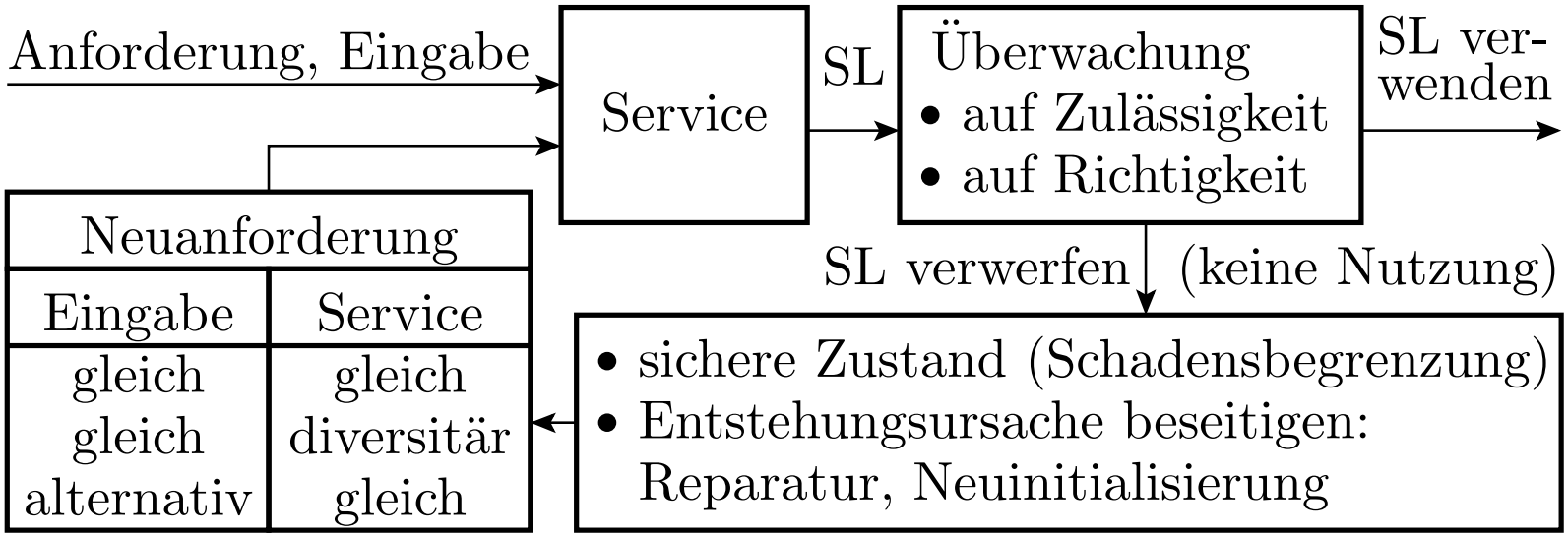
July 8, 2020

Inhalt Foliensatz TV_F5: Überwachung und Fehlertoleranz

Wiederholung
Informationsredundanz
2.1 Fehlererk. Codes
2.2 LFSR
2.3 Prüfkennzeichen
2.4 Fehlerkorr. Codes
2.5 Hamming-Codes
2.6 Burstkorrektur
Formatüberwachung
3.1 Zeitüberwachung
3.2 Protokolle
3.3 Invarianten, WB
3.4 Syntax
Überwachung auf Richtigkeit
Fehlertoleranz
5.1 Fehlerbehandlung
5.2 Redundanz
5.3 Anwendungsspez. Lösungen
5.4 RAID und Backup
Literatur

Wiederholung

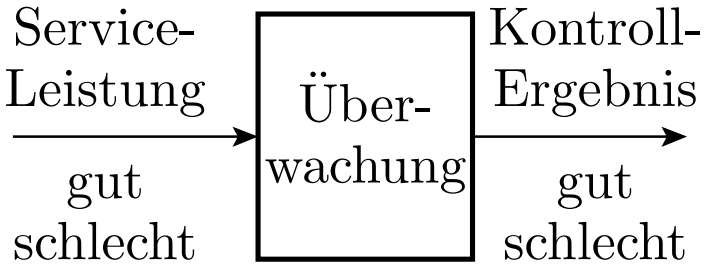
Überwachung und Reaktion auf FF



Zur Vermeidung von unvorhersehbarem Systemverhalten bei FF:

- Überwachung Eingaben, innerer Zustände und SL,
- fehlerhafte SL verwerfen,
- definierten (sicheren) Zustand herstellen,
- optional: Reparatur, Neuinitialisierung, Korrektur, ...
- Erfassung der Entstehungsbed. zur späteren Fehlerbeseitigung.

Kenngrößen der Überwachung



Service-Ergebnis	Kontroll-ergebnis	Fehlfunktion der Kontrolle
schlecht	schlecht	—
gut	gut	—
schlecht	gut	Maskierung
gut	schlecht	Phantom-FF

- FF-Überdeckung:

$$p_{\mathrm{E}} \approx F\hat{F}C = \frac{\#EFF}{\#FF}$$

$\#EFF$ – Anzahl der erkannten FF, $\#FF$ – Anzahl aller FF.

- Phantomfehlerrate:

$$\zeta_{\mathrm{Phan}} = \frac{\#PFF}{\#SL}$$

$\#PFF$ – Anzahl der Phantom-FF, $\#SL$ – Anzahl der SL.

Arten der Überwachung

Formatüberwachung (Kontrolle auf Zulässigkeit)
- fehlererkennende Codes und Prüfkennzeichen
- Zeitüberwachung, Syntaxkontrolle für manuelle Eingaben,
- Invarianten, Protokoll- und Signaleigenschaften, ...

Werteüberwachung (Kontrolle auf Richtigkeit)
- Soll-/Ist-Vergleich,
- Mehrfachberechung und Vergleich,
- inverse Funktion und Eingabevergleich,
- Invarianten für die Korrektheit.

Meisten wird nur auf Zulässigkeit überwacht, weil eine Überwachung auf Richtigkeit nicht möglich bzw. unakzeptabel aufwändig ist oder keine bessere FFC (Fehlfunktionsüberdeckung) verspricht.

Informationsredundanz

Informationsredundanz

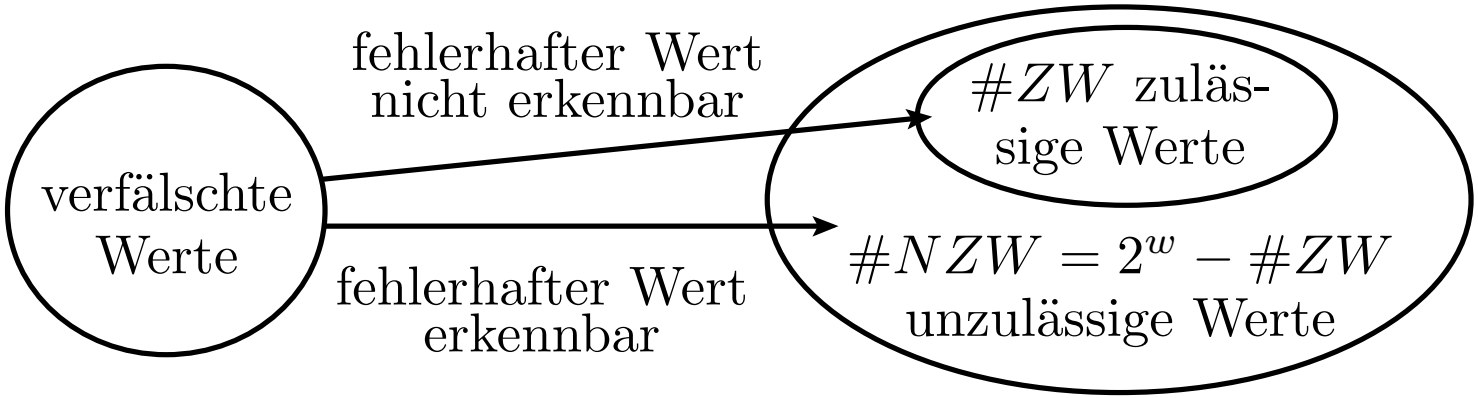
Die Unterscheidung von $\#ZW$ zulässigen Werten verlangt in der Binärdarstellung

$$w \geq \log_2 (\#ZW)$$

Bits, mit denen 2^w Werte darstellbar sind, von denen

$$\#NZW = 2^w - \#ZW$$

unzulässig sind. Eine FF verfälscht einen zulässigen Wert in entweder einen anderen zulässigen oder einen unzulässigen Wert:



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

$$p_{\mathrm{E}} = FFC \approx 1 - \frac{\#ZW}{2^w} \tag{1}$$

($\#ZW$ – Anzahl der zulässigen Werte; w – Bitanzahl zur Darstellung). Mit einer kleinen Informationsredundanz von

$$r = w - w_{\min} \approx 10 \ldots 20$$

redundaten Bits ist die Erkennswahrscheinlichkeit praktisch eins, aber:

- Wenn fehlerhafte Werte unverhältnismäßig oft zulässig sind, kann p_{E} deutlich kleiner sein als nach Gl. 1.
- Wenn fehlerhafte Werte unverhältnismäßig oft unzulässig sind, kann p_{E} deutlich größer sein als nach Gl. 1.
- Klassifizierungen zulässiger Werte als unzulässig sind Phantom-FF.

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 $\#Bytes$ darstellbaren Zeichenketten:

$$2^{8\cdot\#Bytes}$$

  Anteil der gültigen Worte fast null. Nach Gl. 1 $p_{\mathrm{E}} \approx 1$.
- Schreibfehler überdurchschnittlich oft gültige Worte, Typisch:

$$p_{\mathrm{E}} \approx 80\% \ll 1$$

- Es gibt auch relativ viele Worte, die im Wörterbuch fehlen. Phantom-FF-Wahrscheinlichkeit typisch $p_{\mathrm{Phan}} > 1\%$.

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.

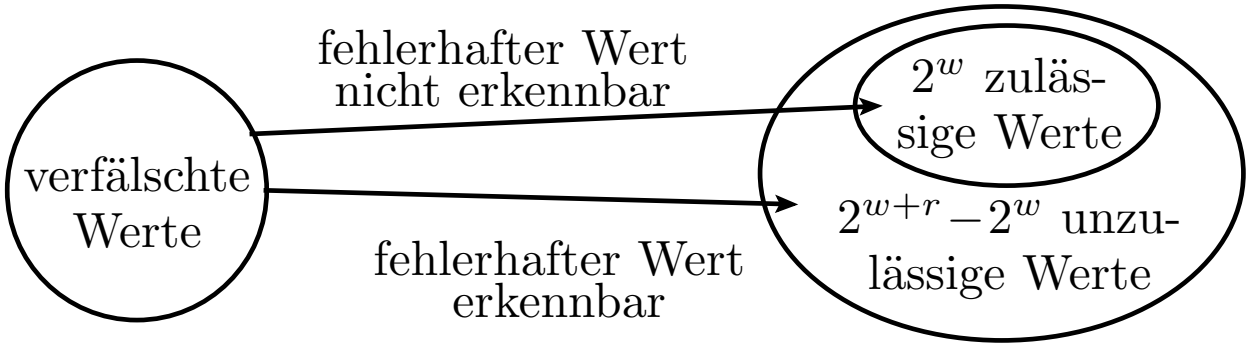
Für die Einzelbitfehlererkennung genügt ein Code, in dem nur Werte mit gerader (ungerader) Anzahl von Einsen gültig sind (siehe später Erweiterung um ein Paritätsbit).

Anteil gültiger Codeworte 50%, Erkennungswahrscheinlichkeit

$$p_{\mathrm{E}} \approx 1 \gg 1 - 50\%$$

Fehlererk. 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:

$$p_{\mathrm{E}} = 1 - \frac{2^w}{2^{w+r}} = 1 - 2^{-r} \tag{2}$$

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

$$p_{\mathrm{E}} \approx 1 - \frac{1}{34562134}$$

und damit fast eins. Bei sehr großen unbekannten 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

$$
\begin{array}{rr}
 & 10010101101 \\
\odot & 10011 \\
\hline
\oplus & 10010101101 \\
\oplus & 10010101101 \\
\oplus & 00000000000 \\
\oplus & 00000000000 \\
\oplus & 10010101101 \\
\hline
 & 10001110100111
\end{array}
$$

Decodierung

$$
\begin{array}{rl}
 & 10001110100111 : 10011 = 10010101101 \\
\oplus & 10011 \\
\hline
 & 10110 \\
\oplus & 10011 \\
\hline
 & 10101 \\
 & 10011 \\
\hline
 & 11000 \\
\oplus & 10011 \\
\hline
 & 10111 \\
\oplus & 10011 \\
\hline
 & 10011 \\
\oplus & 10011 \\
\hline
\text{Rest:} & 0000 \qquad \text{(unverfälscht)}
\end{array}
$$

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

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

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

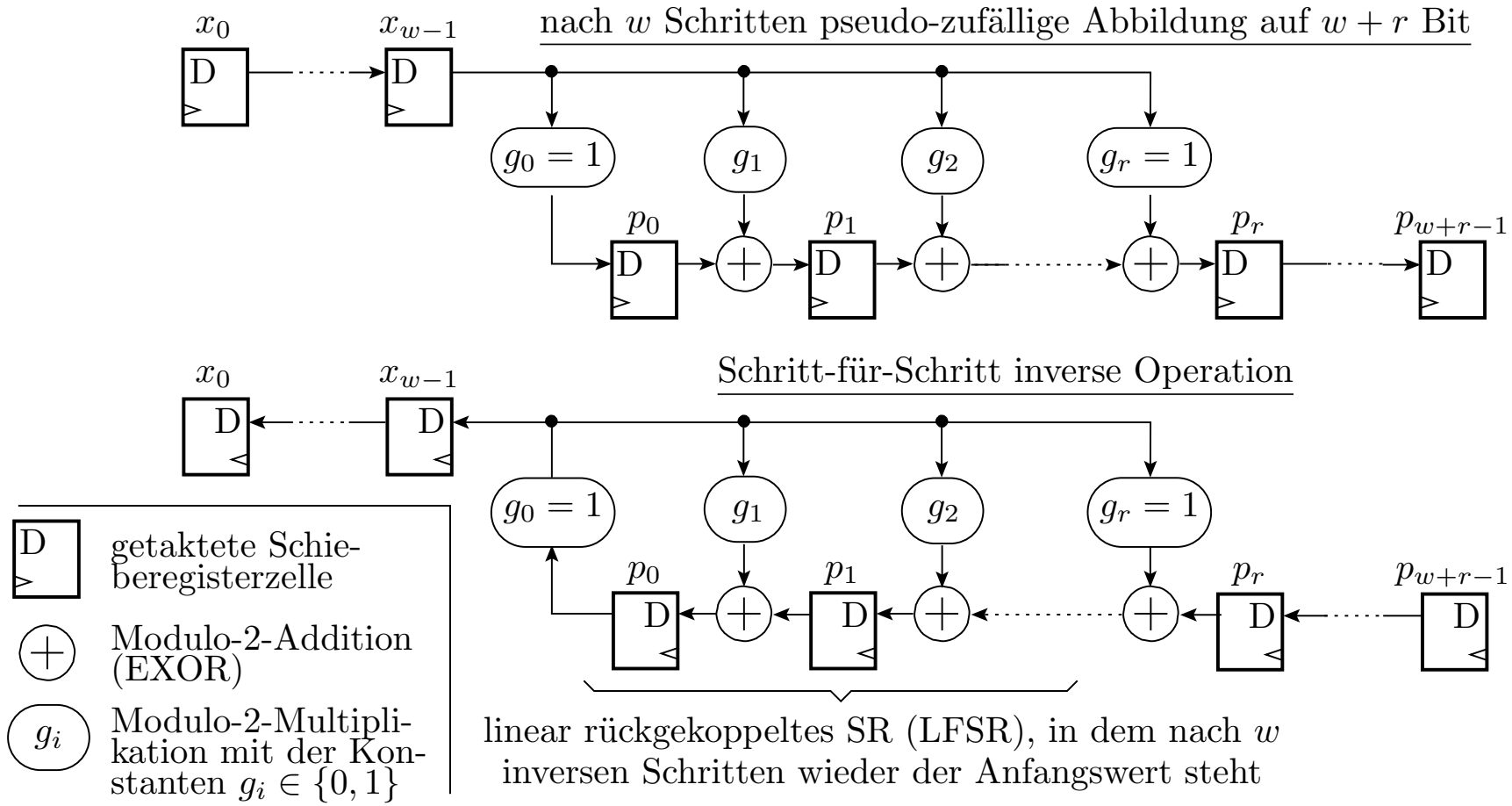
$$\left(x^{14} \oplus x^{10} \oplus x^9 \oplus x^8 \oplus x^5 \oplus x^2 \oplus x \oplus 1\right) : \left(x^4 \oplus x \oplus 1\right) =$$
$$x^{10} \oplus x^7 \oplus x^5 \oplus x^3 \oplus x^2 \oplus 1$$

liefert ohne Rest das Polynom der Originalfolge.

LFSR

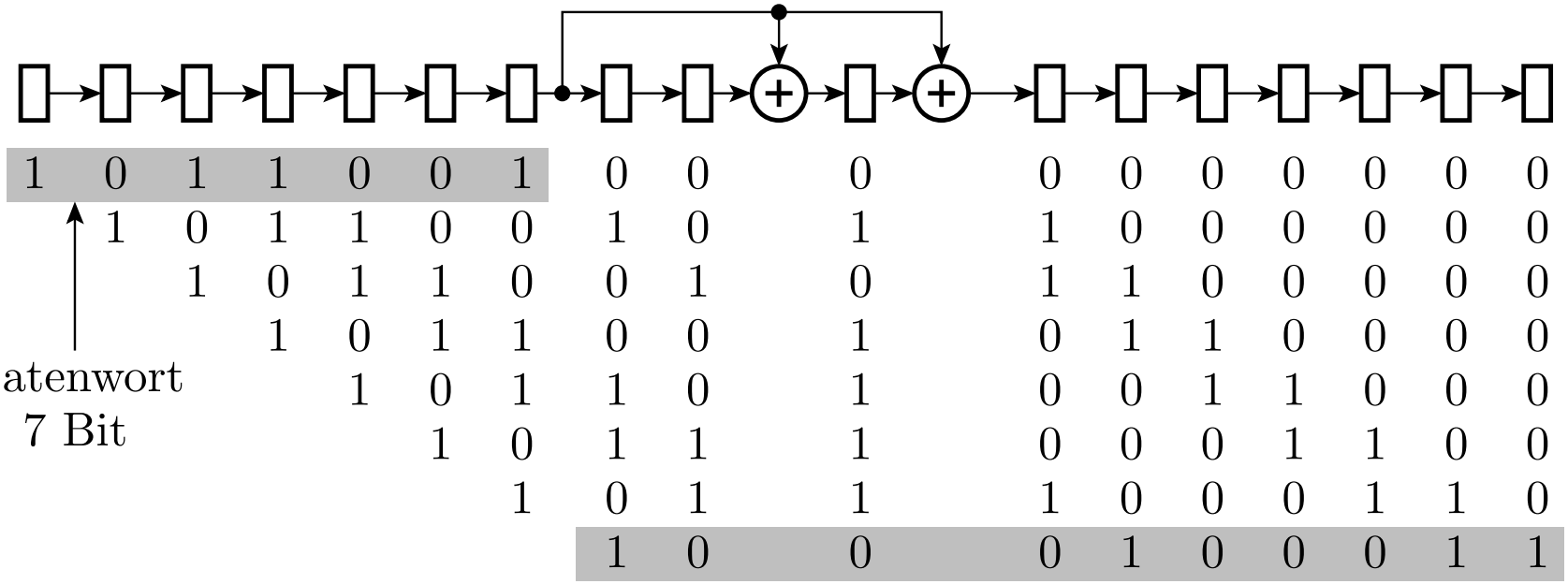
Polynommultiplikation und -Division mit SR

Die Codierung und Decodierung erfolgt mit Schieberegistern.



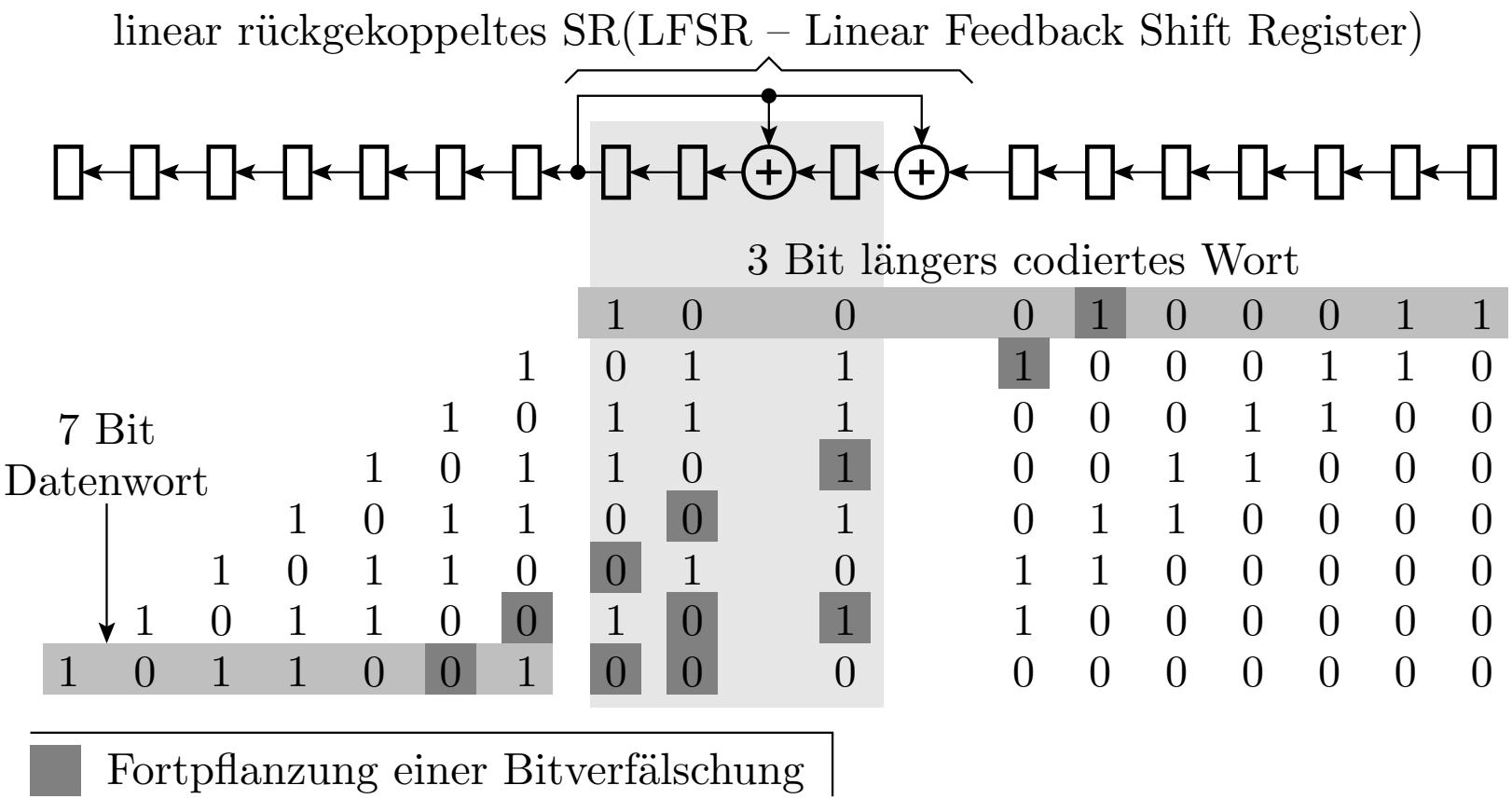
Erkennungswahrscheinlichkeit: $p_{\mathrm{E}} = 1 - \frac{2^w}{2^{w+r}} = 1 - 2^{-r}$

Beispiel für die Codierung



- 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}.

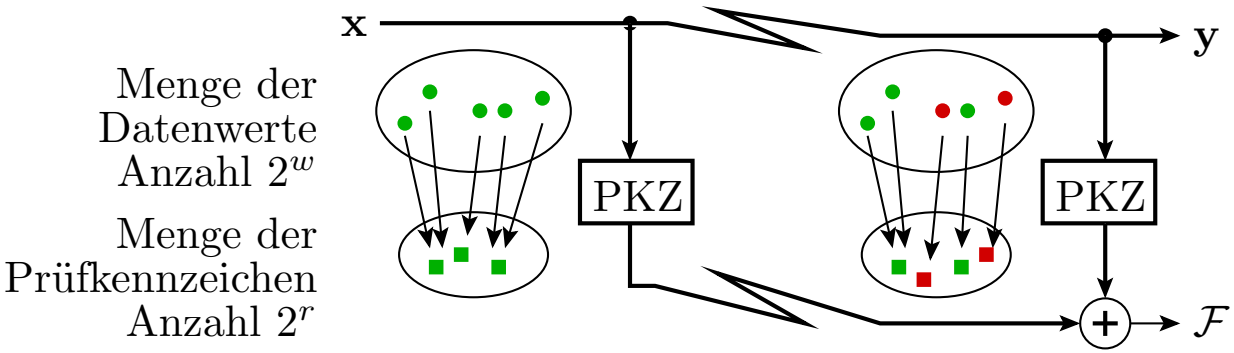
Rückgewinnung und Kontrolle



Eine Bitverfälschung verursacht mit der Wahrscheinlichkeit $p_{\mathrm{E}} = 1 - 2^{-3}$ einen von null abweichenden Endwert im LFSR.

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:

$$p_{\mathrm{E}} \approx 1 - \frac{2^w}{2^{w+r}} = 1 - 2^{-r} \tag{3}$$

Prüfsummen

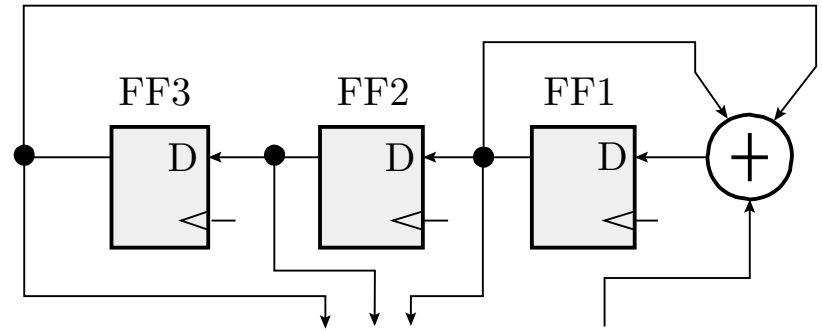
Prüfkennzeichenbildung durch Aufsummierung (arithmetisch, bitweises EXOR, ...).

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) 1110 14 (+16)	0001 1110 30	1100

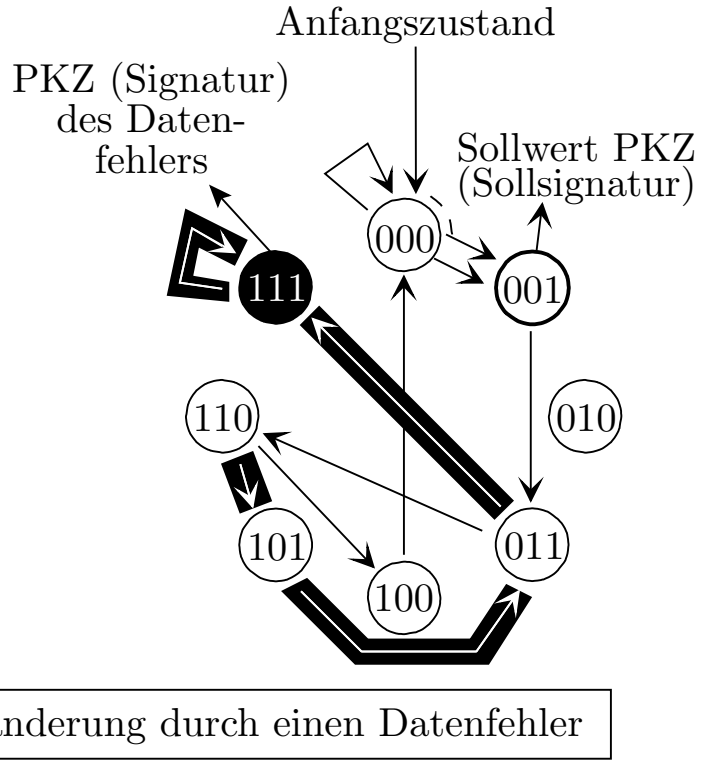
Bei »einfacher Genauigkeit« und »bitweisem EXOR« erscheint die Annahme »pseudo-zufällige Abbildung« gerechtfertigt[1]: $p_{\mathrm{E}} \approx 1 - 2^{-4}$.
Bei »doppelter Genauigkeit« bilden sich Verfälschungen vorzugsweise auf die niederwertigen Bits ab. Maskierungswahrscheinlichkeit: $2^{-4} > 1 - p_{\mathrm{E}} \gg 2^{-8}$.

[1] Kein Nachweis für vertauschte Summationsreihenfolge.

Prüfkennzeichenbildung mit LFSR[2]



Initialwert	0	0	0	1
Schritt 1:	0	0	1	0
Schritt 2:	0	1	1	1
Schritt 3:	1	1	0	**1**
Schritt 4:	1	0	**0**	1
Schritt 5:	0	**0**	**0**	0
Schritt 6:	**0**	**0**	**0**	1
Schritt 7:	**0**	**0**	1	

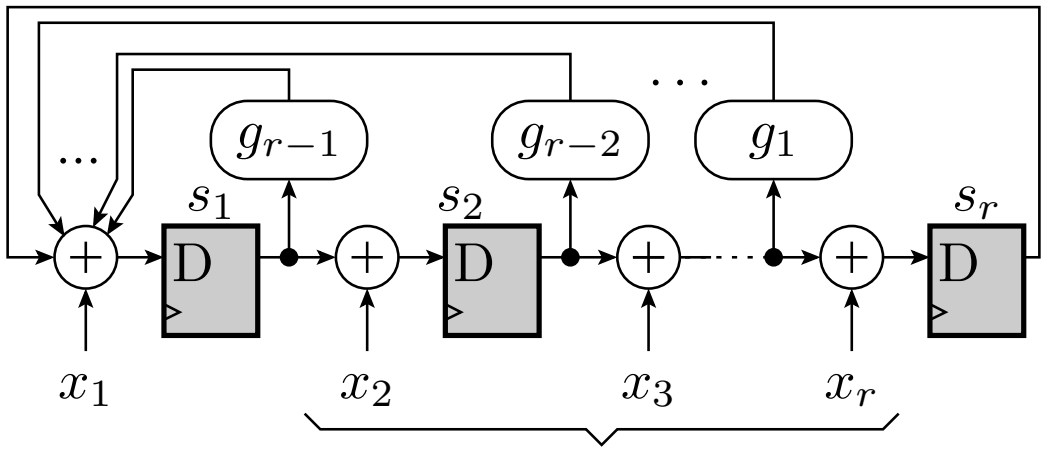


Das Prüfkennzeichen wird wie bei der CRC-Decodierung mit einem linear rückgekoppelten Schieberegister (LSFR) gebildet. Im Beispiel hat das LSFR im Gegensatz zur Polynomdivision Folie 20 eine zentrale Rückführung. Abbildung auch pseudo-zufällig.

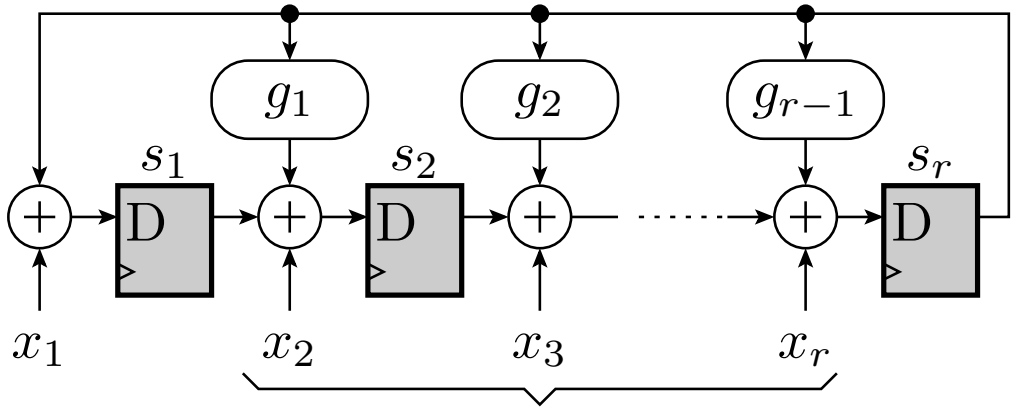
[2] LFSR – **L**inear **F**eeback **S**hift **R**egister.

Allgemeine Struktur von LFSR

Für die Bildung auf 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, ein oder mehrere Bits, modulo-2 als zu den Registerzuständen addiert werden.



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



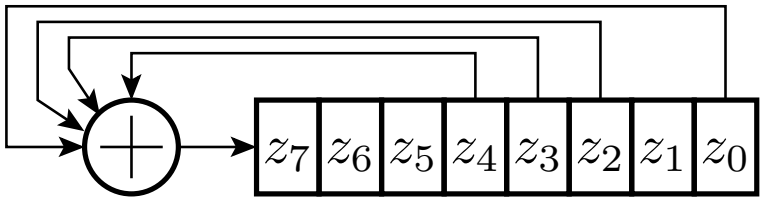
Erweiterungsmöglichkeit auf mehrere Eingänge

Die Koeffizienten g_i der Rückführung, bei der Polynom-Division das Divisor-Polynom, bestimmen die autonome Zyklusstruktur[3]. 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.

[3] Zyklusstruktur ohne Eingaben.

Autonome Zykluslänge und -struktur von LFSR

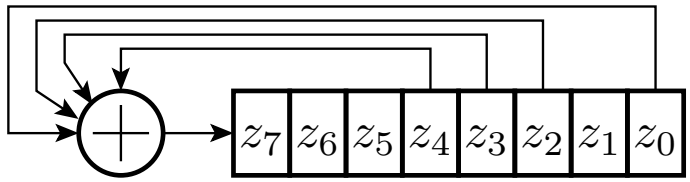
Autonom bedeutet Zyklusstrukur für Eingabewerte »alle 0« oder für LFSR ohne Eingänge. Beispiel 8-Bit autonomes LFSR:



0	0	1	1	0	0	0	1
0	0	0	1	1	0	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	1	1	0
1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Übergangsfunktion:

$$z_7 = z_4 \oplus z_3 \oplus z_2 \oplus z_0$$
$$z_i = z_{i+1} \text{ für } i \in \{0, 1, 2, \ldots 6\}$$



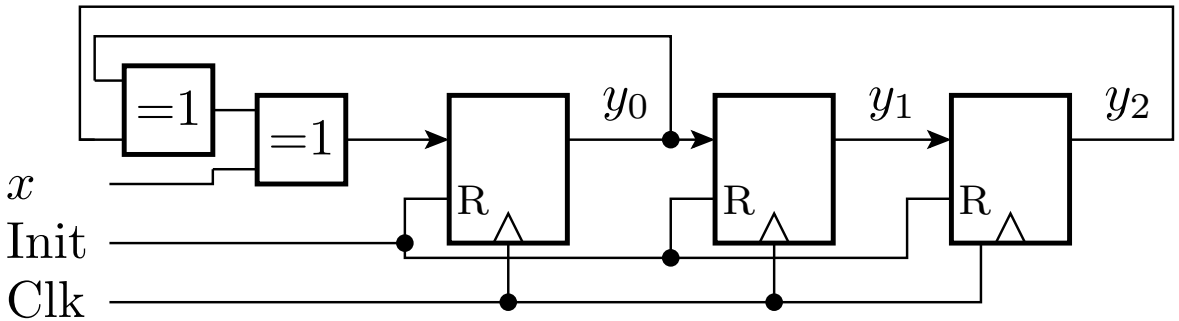
Bestimmen der Zyklusstruktur durch Simulation (z.B. C-Prograann):

```
...
#define ZA 0x31
uint8_t z = ZA;          // Startwert setzen
while(1){
   z = (z>>1) ^ (z<<7) ^ ((z<<5)&0x80)
        ^ ((z<<4)&0x80) ^ ((z<<3)&0x80);
   <Ausgabe von z>
   if (z==ZA) break;  // bis wieder Anfangswert
   ...                // weiter mit nicht enthal-
}                     // tenem Zustand als Startw.
```

- 0x00 geht in sich selbst über.
- Alle anderes 255 Zustände gehen zyklisch ineinander über.
- max. Zykluslänge $2^r - 1$: primitive Rückkopplung.

Beispielaufgabe

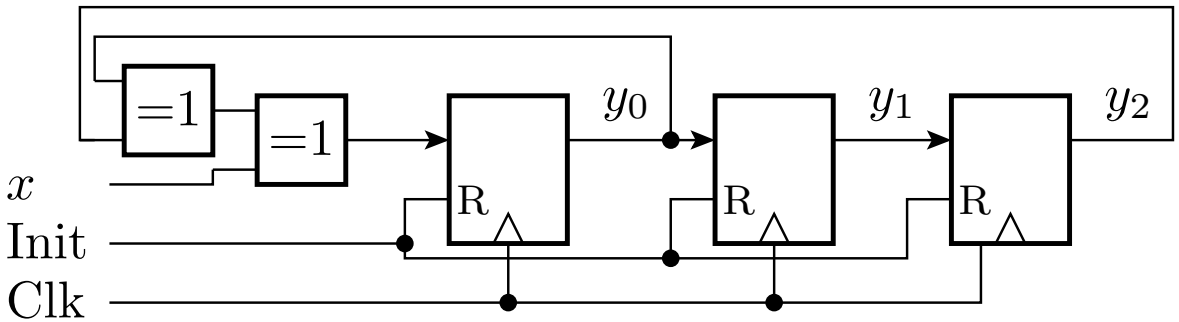
Gegeben ist folgendes linear rückgekoppelte Schieberegister:



	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 $\mathbf{y} = y_2 y_1 y_0$ 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:

$$p_{\mathrm{E}} \approx 1 - 2^{-3} = 87{,}5\%$$

Zusammenfassung

Datensicherung mit fehlererkennenden Codes / Prüfkennzeichen:

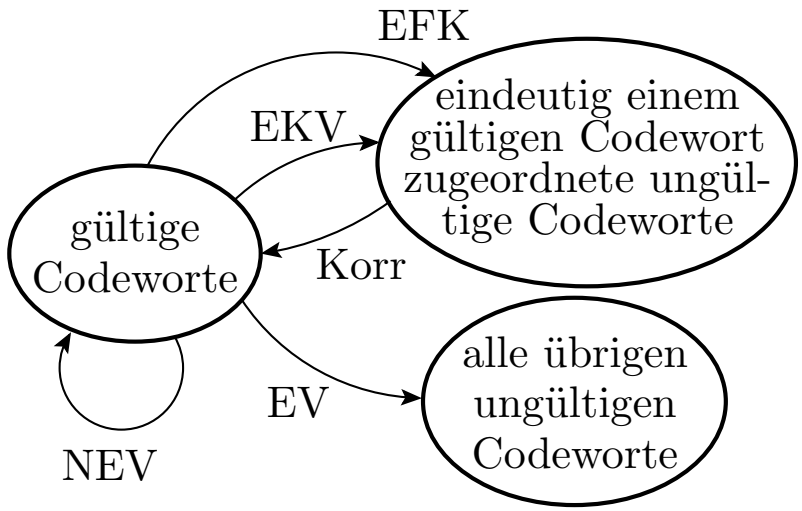
- Geringer Berechnungsaufwand.
- Geringer Zusatzaufwand für Datenübertragung und -speicherung (r zusätzlich gespeicherte / übertragene Bits für eine Datenobjekt beliebiger Größe).
- Maskierungswahrscheinlichkeit 2^{-r}. Mit ausreichendem r immer vernachlässigbar klein.

Dateien, Nachrichten etc. werden sehr oft mit Prüfkennzeichen übertragen und gespeichert. In Software sind PKZs bevorzugt Prüfsummen, in Hardware werden sie bevorzugt mit LFSR gebildet.

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:

$$2^b \geq \#CWG + \#CWG \cdot \#CWK/CWG$$

($b = w + r$ – Bitanzahl; $\#CWG$ – Anzahl gültige Codeworte; $\#CWK/CWG$ – 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:

$$\#CWK/CWG = b$$

Mindestbitanzahl:

$$2^{\mathrm{b}} \geq \#CWG + b \cdot \#CWG = (b+1) \cdot \#CWG$$

Für $\#\mathrm{CWG} = 256$ gültige Codeworte:

$$\begin{aligned} 2^b &\geq 256 \cdot (1+b) \\ b &\geq 12 \end{aligned}$$

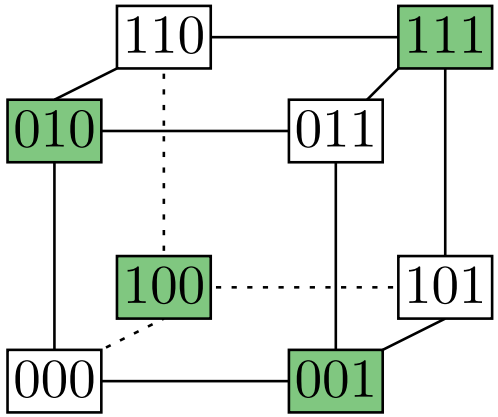
Probe:

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

Hamming-Codes

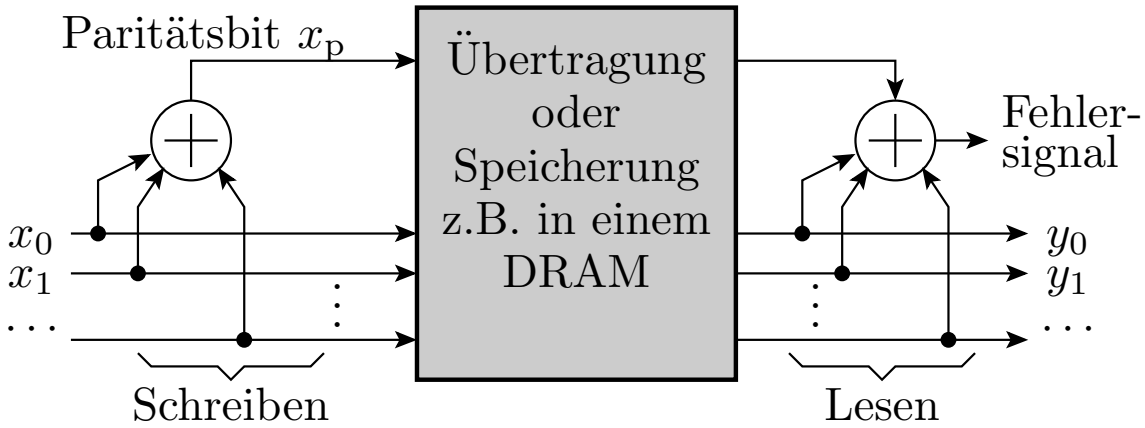
Hamming-Distanz

Die Hamming-Distanz N_{Ham} 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 $Ham \geq k + 1$.
- Um k-Bit-Fehler korrigieren zu können, ist eine Hamming-Distanz von $Ham \geq 2 \cdot k + 1$ erforderlich.

Parität als Prüfkennzeichen (Hamming-Dist. 2)



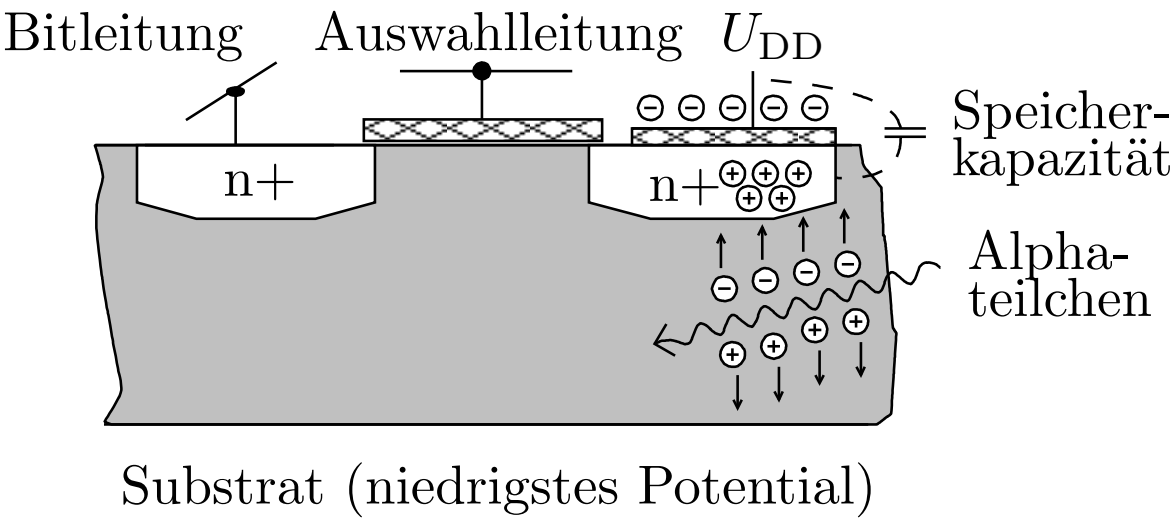
Einzelprüfbit, modulo-2 Summe (EXOR-Verknüpfung):

$$x_\mathrm{p} = x_{n-1} \oplus x_{n-2} \oplus \ldots \oplus x_1 \oplus x_0$$

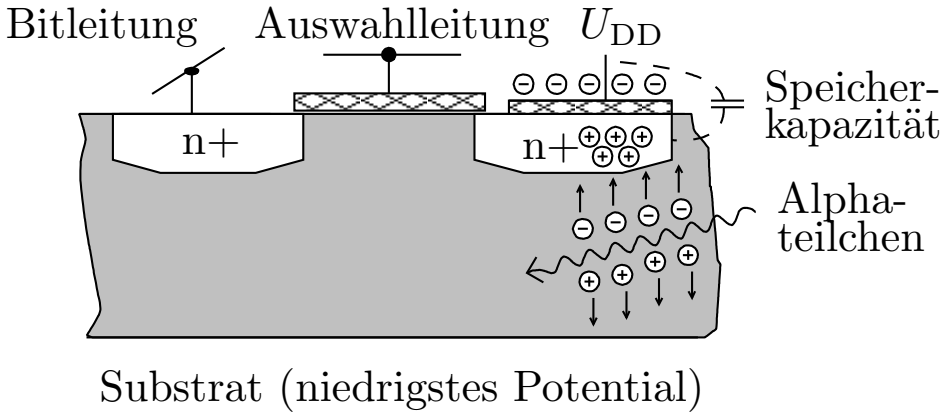
bei gerader Anzahl von Einsen »0« sonst »1«.

- Erkennt jede ungeradzahlige Anzahl von Bitverfälschungen.
- Wenn geradzahlige und ungeradzahlige Bitfehler gleichhäufig auftreten: $p_\mathrm{E} \approx 50\%$
- Überwiegend Einzelbitfehler: $p_\mathrm{E} \gg 50\%$

Paritätstest für DRAMs und Speicherriegel



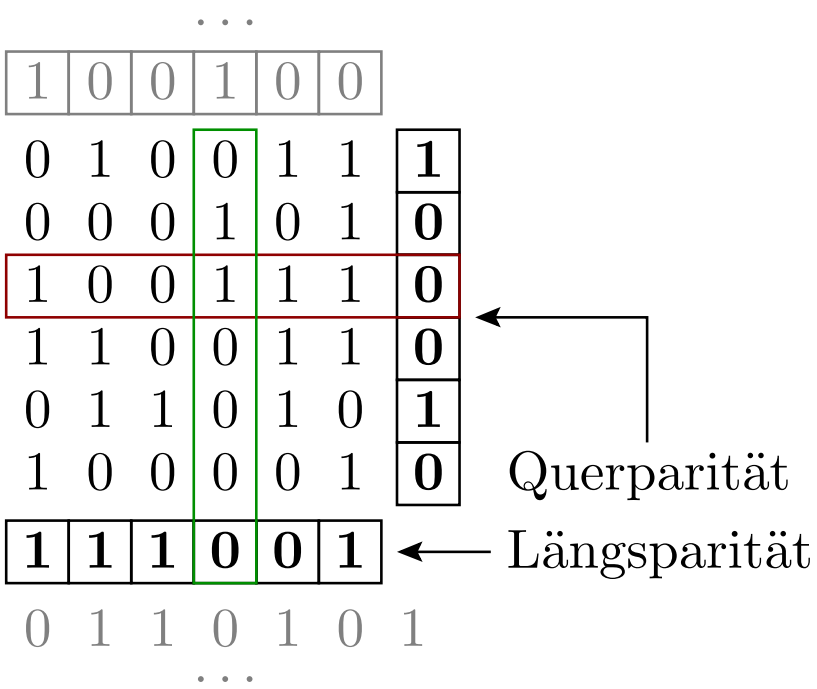
- Informationsspeicherung in winzigen Kapazitäten.
- Häufigste Ursache für Datenverfälschungen: Alphastrahlung.
- Deren Quellen radioaktiver Zerfall von Uran und Thorium, die als Spurenelemente im Gehäuse und im Aluminium der Leiterbahnen enthalten sind, und Kernprozesse im Silizium durch Höhenstrahlung.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



- Energie eines Alphateilchen: $5\,\mathrm{MeV}$. Energieverlust bei der Generierung eines Elektronen-Loch-Paares $\approx 3{,}6\,\mathrm{eV} \Rightarrow$ Generierung von $\approx 10^6$ Ladungsträgerpaaren. Reichweite $\approx 89\,\mu\mathrm{m}$. 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 Speichermatritzen) $\Rightarrow$ 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).



Beispielaufgabe

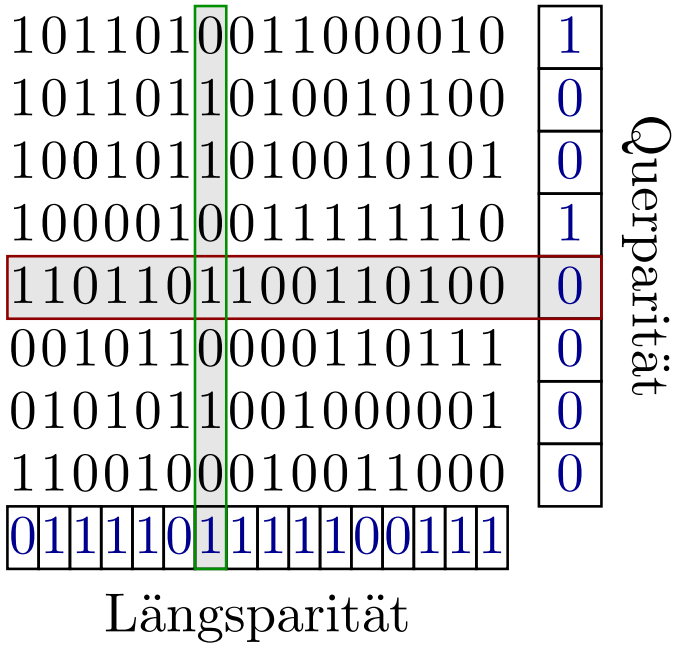
Kontrollieren Sie für die nachfolgenden Bitfelder mit Kreuzparität, ob eine erkennbare oder eine erkenn- und korrigierbare Verfälschung vorliegt und führen Sie, wenn möglich, die Korrektur durch.

1011010011000010	1
1011011010010100	0
1001011010010101	0
1000010011111110	1
1101101100110100	0
0010110000110111	0
0101011001000001	0
1100100010011000	0
0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1	

Querparität

Längsparität

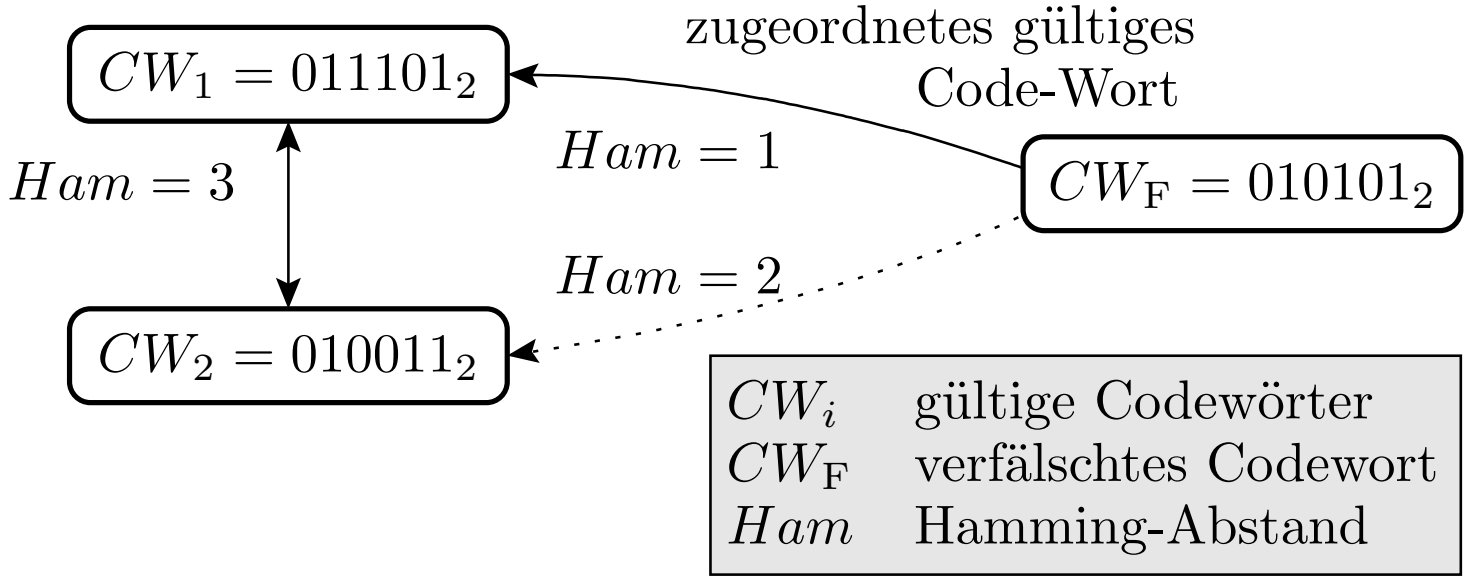
Lösung



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

1-Bit fehlerkorrigierende Hamming-Codes

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



Korrektur durch Ersatz des verfälschten Codeworts durch das mit Hamming-Distanz $Ham = 1$. Bei Hamming-Distanz $Ham = 3$ werden Codeworte mit zwei oder mehr verfälschten Bits falschen gültigen Codeworten zugeordnet.

Konstruktion eines 1-Bit fehlerkorrigierenden Codes

Zusammensetzen des Gesamtcodeworts aus

- einem Datenwort mit minimaler Bitanzahl

$$w \geq \log_2 (\#CWG)$$

und einem Prüfkennzeichen der Bitanzahl

$$r = b - w$$

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: $w = 8$, $r = 4$, $\mathbf{q} = q_3 q_2 q_1 q_0$

$$\begin{aligned}
\Delta q_0 &= b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} \\
\Delta q_1 &= b_2 \oplus b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} \\
\Delta q_2 &= b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_{12} \\
\Delta q_3 &= b_8 \oplus b_9 \oplus b_{10} \oplus b_{11} \oplus b_{12}
\end{aligned}$$

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

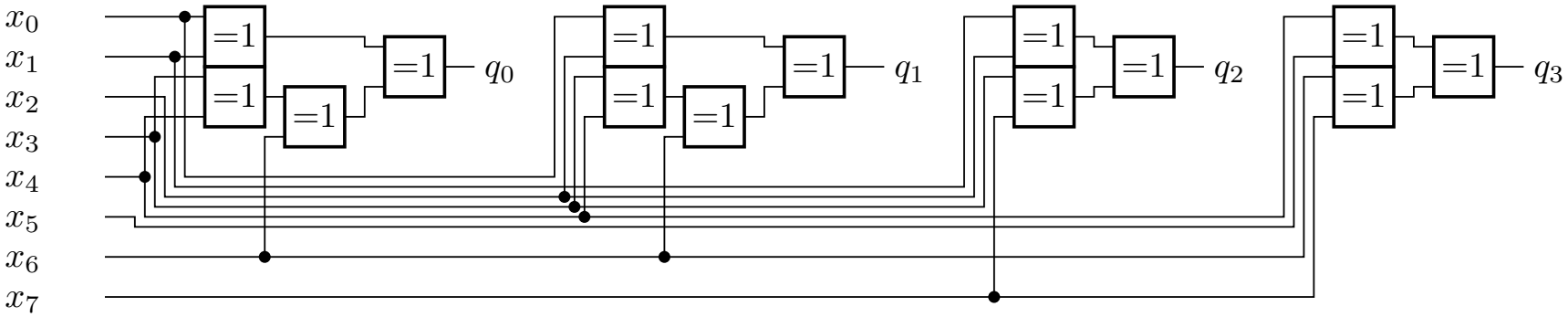
Für die erste Summe gilt:

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

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 $\mathbf{q} = q_3 \ldots q_0$. Vergleich durch bitweises EXOR des empfangenen und des im Empfänger gebildeten Prüfzeichens. Invertierung des verfälschten Bits.

Bitfehlerkorrektur als VHDL-Case-Anweisung:

```
case (q_Empf xnor 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

$$q_0 = x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6 \qquad q_2 = x_1 \oplus x_2 \oplus x_3 \oplus x_7$$
$$q_1 = x_0 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 \qquad q_3 = x_4 \oplus x_5 \oplus x_6 \oplus x_7$$

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													
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	$d\,q_2 = 12_{10}$

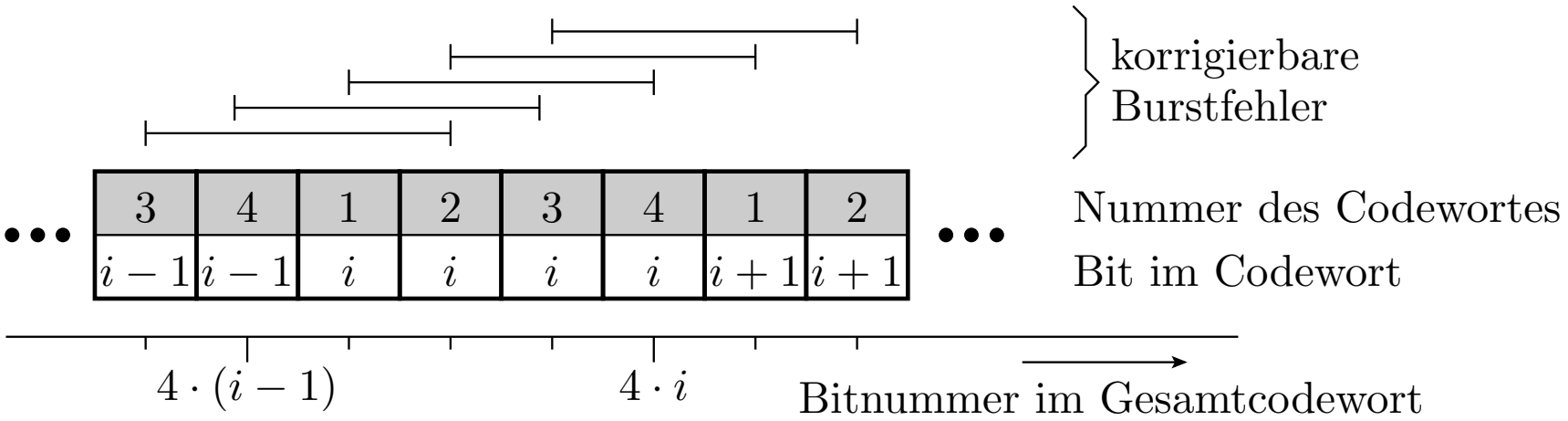
w_2 : Wert 0xA2 mit verfälschtem $x_7 \Rightarrow w_2$ =0x22

Burstkorrektur

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												
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													
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,

Formatüberwachung

Formatkontrollen

Einteilung der überwachbaren Merkmale:

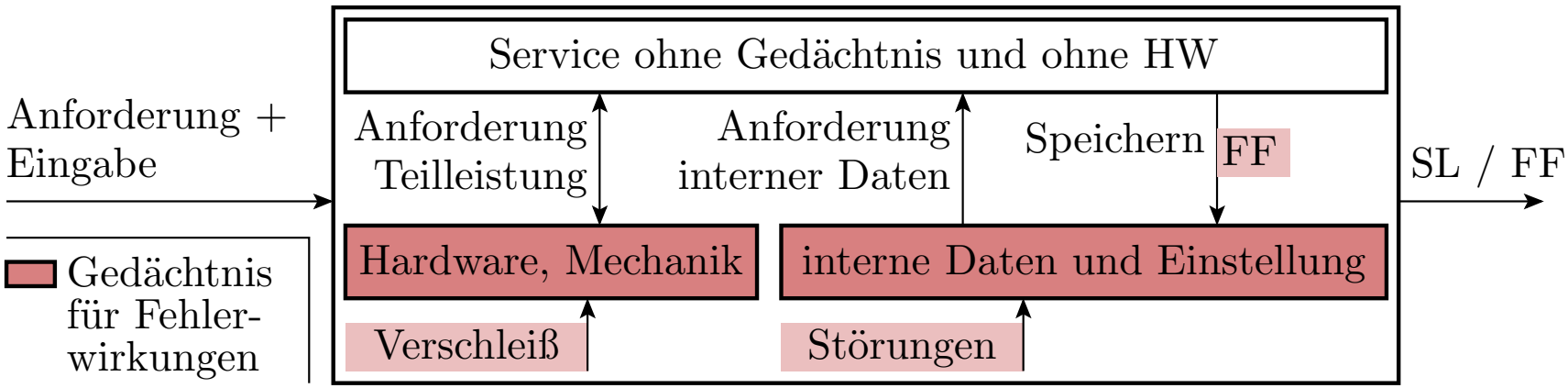
- Format: Konstante, immer erfüllte Merkmale (Zeitraster, ...).
- Daten: Variable Merkmale (Werte von Datenobjekten, ...).

Prüfkennzeichen und fehlererkennende Codes sind Formatmerkmale, an denen sich Verfälschen mit $p_{\mathrm{E}} \approx 1$ erkennen lassen. Leider nicht für alle Datenrepräsentationen geeignet. Weitere kontrollierbare Formatmerkmale:

- Zeitschranken,
- Protokoll- und Signaleigenschaften,
- Wertebereiche, Invarianten;
- Syntax von Texteingaben.

Zeitüberwachung

Burst-FF und Ausfälle



System mit Gedächtnis für Fehlerwirkungen:

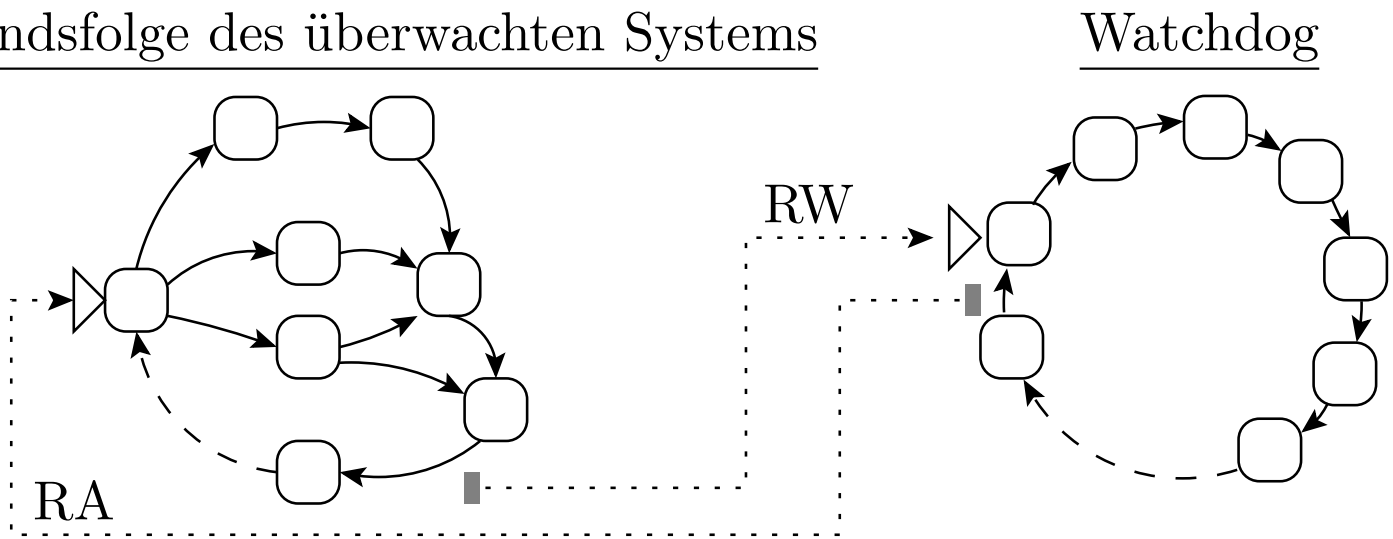
- Hardware und Mechanik, in der durch Verschleiß in der Einsatzphase neue Fehler entstehen können.
- Verfälschung service-interner Daten und Einstellungen durch FF.

Typische Fehlfunktion:

- verzögerte Ergebnisausgabe,
- Dauerhafter Übergang in unzulässige Zustände (Absturz).

Fehlerbehandlung: [Reparatur,] Neuinitialisierung, Wiederholung.

Watchdog



RA Ein Überlauf des Watchdogs initialisiert den Automaten neu.

RW Bei einem bestimmten, regelmäßig stattfindenden Zustandsübergang wird der Watchdog neu initialisiert.

Das überwachte System setzt in periodisch zu erreichenden Sollzuständen einen Zeitzähler zurück, der bei Überlauf das System neu startet und dabei auch wieder einen zulässigen Zustand herstellt.

Das Watchdog-Prinzip wird angewendet für

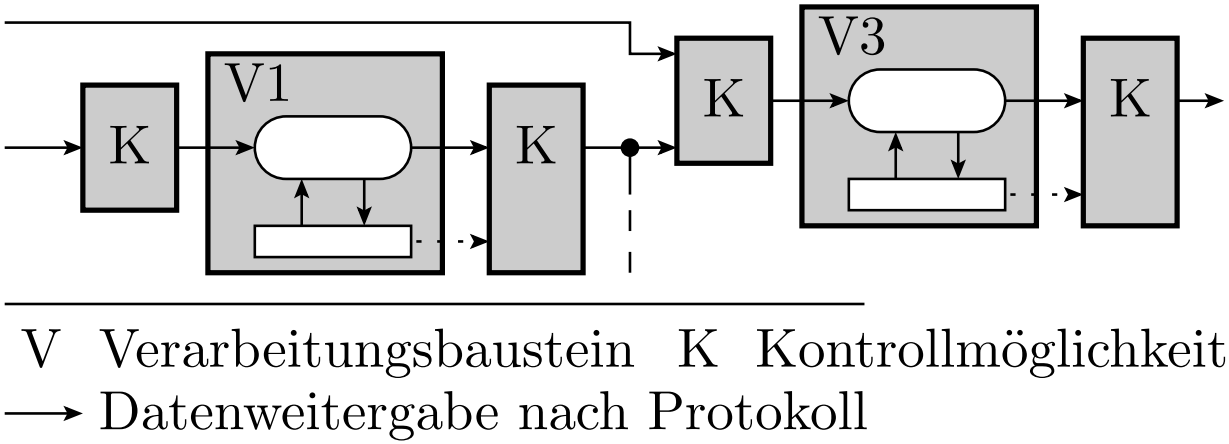
- einzelne Aufgaben,
- einzelne Programme in Multi-Task-Systemen,
- komplette Rechner, ...

Prozessoren haben in der Regel einen Hardware-Watchdog

- mit programmierbarer Zeit bis zum Überlauf,
- der, wenn eingeschaltet, nicht vom Programm, d.h. auch nicht durch Fehlfunktionen, ausschaltbar ist, sondern nur durch Neustart,
- bei Überlauf einen Betriebssystemaufruf zur Fehlerbehandlung und/oder einen Rechnerneustart auslöst.

Protokolle

Überwachung von Protokollen



Komplexe Systeme, insbesondere mit Komponenten unterschiedlicher Hersteller, verteilte Systeme, Open-Source-Projekte, ... tauschen ihre Daten über ein wohldefiniertes Regelwerk aus, genannt Protokolle:

- Software-Schnittstellen, Busprotokoll,
- Signaldefinitionen, Internet-Protokoll, ...

Protokolle definieren vielfältige Kontroll- und Überwachungsmöglichkeiten für zugesicherte und zuzusichernde Eigenschaften bieten.

Software-Schnittstellen und Fehlerbehandlung

```
// Beispiel einer Funktion in Rust
fn function_name(x: String)->Result<String, io::Error>{
 if <Eingabefehler> {panic!("Ausgabetext");} // Überwachung, bei FF
                                             // Programmabruch
 ...;                                        // Beendigung, Ergebnisrückgabe
}
```

- Schnittstellen legen Anzahl und Typ der Ein- und Ausgaben fest.
- Typen (elementar, zusammengesetzt, generisch, ...) definieren kontrollierbare Regeln.
- Über die Typdefinitionen hinausgehende Beschränkungen bieten weitere Kontrollmöglichkeiten.
- zu jeder Kontrolle gehört eine Fehlerbehandlung.
- »panic«: Makro für den kontrollierte Programmabbruch, ...

```
1  fn function_name(x: String)->Result<String, io::Error>{
2   let f = File::open(x);     // f: Result<file,io::Error>
3   let mut f = match f {      // Fallunterscheidung
4    Ok(file) => file,         // Erfolg: f <= file
5    Err(e) => return Err(e),// sonst Fehlerrücksprung
6   };
7     ...;
8  }
```

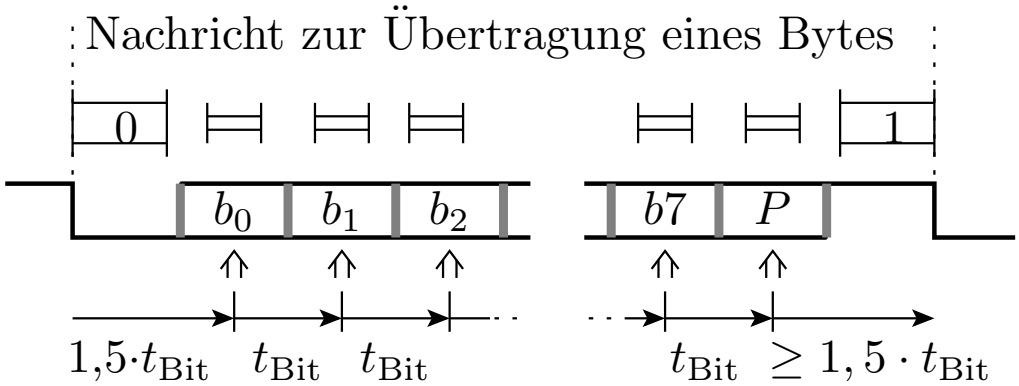
1. Eingabe ist eine Zeichenkette. Rückgabetyp ist »Ok« + Zeichenkette oder »Err« + Datenstruktur »io::Err«.
2. Die Funktion »File::open« gibt entweder Wert »Ok« + Filehandler oder Wert »Err« + Datenstruktur »io::Err« zurück.
3. »match f«: zum einen Ausdruck mit einem Wert, zum anderen Fallunterscheidung nach der Art von »f«
4. wenn »Ok« hat der Match-Ausdruck den Wert »file«
5. wenn »Err«, Rücksprung mit »Err« + Datenstruktur »io::Err«

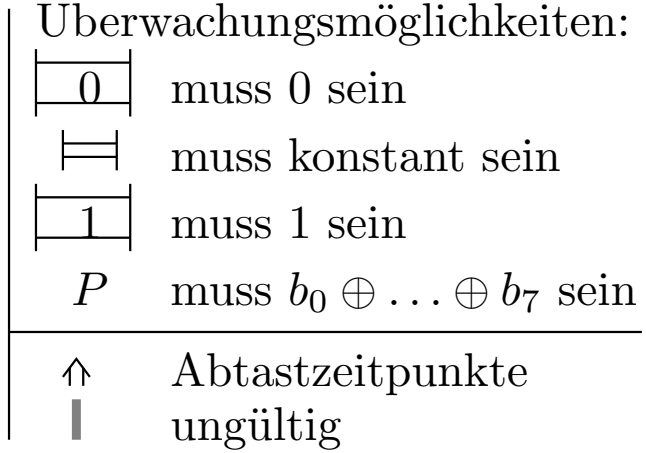
RUST ist eine Programmiersprache für effiziente und zuverlässige Programme (siehe später Foliensatz 7).

Busprotokoll, Beispiel UART

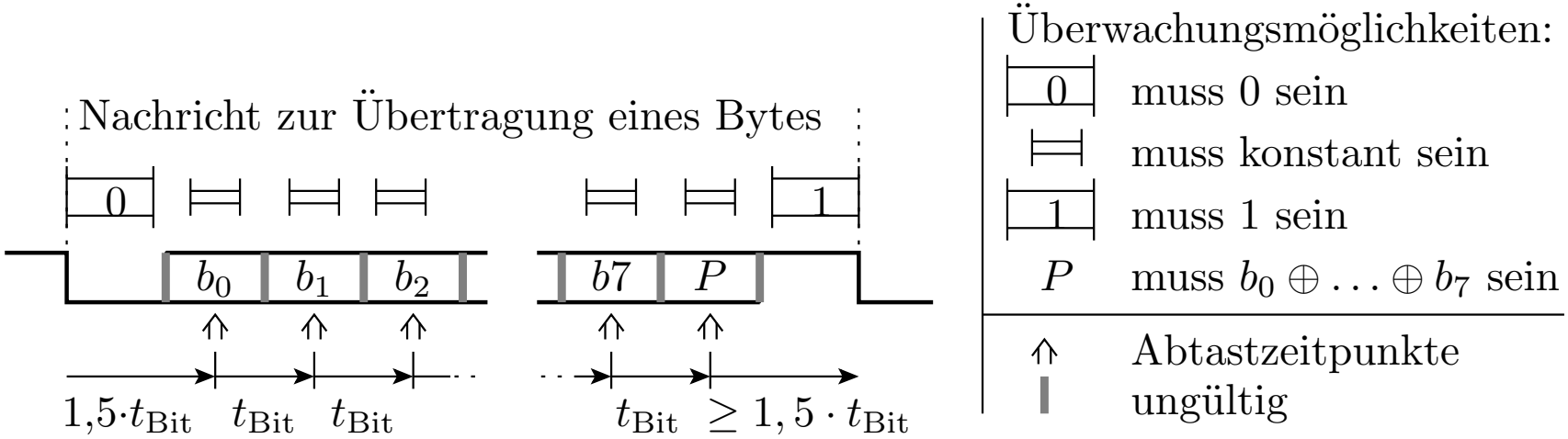
Zwischen Hardware-Modulen erfolgt der Datenaustausch in der Regel über Busse. Beispiel UART-Protokoll zur bytweisen Datenübertragung:

- Baud-Rate (Bitzeiten pro s): 4800, 9600, ...
- 1 Startbit mit Wert 0,
- 7 / 8 /9 Datenbits,
- kein /gerades / ungerades Paritätsbit und
- 1 / 2 Stoppbit(s) mit Wert 1.





Kontrollierbare Formateigenschaften

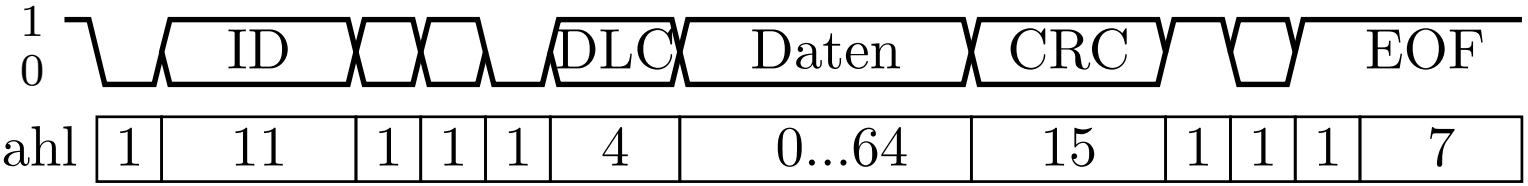


Erkennbare Formatfehler

- Paritätsfehler: 1-Bit-CRC, $p_{\mathrm{E}} \approx 50\%$
- Datenrahmenfehler: Datenwechsel in Zeitfenstern, in denen keine Änderung erlaubt. Zu kurze Start- oder Stoppbitzeit.
- Datenüberlauf: mehr ankommende Bytes als der Empfänger verarbeiten kann.

Überwachung durch die Prozessor-Hardware. Benachrichtigung der Software durch Setzen von FF-Bits in Spezialregistern.

CAN-Bus: Vernetzung Fahrzeugsteuergeräte



Bitanzahl	1	11	1	1	1	4	0...64	15	1	1	1	7

ID	Nachrichtennummer	CRC	Prüfkennzeichen
DLC	Datenlängencode	EOF	Ende des Datenrahmens

Erkennbare Formatfehler:

- 15-Bit CRC als PKZ. Erkennungssicherheit

$$p_{\mathrm{E}} = 1 - 2^{-15} \approx 1 - 3 \cdot 10^{-5}$$

- Soll-/Ist-Abweichung konstante Bits, unzul. Datenwechsel.

Fehlerbehandlung:

- Wiederholung bei Nachrichtenkollision,
- Notfallreaktion bei Ausfall anderer Steuergeräte,
- Einträge der FF-Nummern in den Fehlerspeicher, ...

Ethernet-Paket

Ethernet-Paket:

Sicherungs-schicht			MAC-Empfänger	MAC-Absender	Protokoll-typ	Nutzlast max. 1500 Bytes	Prüfkennz. CRC	
Bitübertra-gungsschicht	Präambel	Start-byte						Lücke zum nächsten Paket
Byteanzahl	7	1	6	6	2	46 bis 1500	4	12

Erkennbare Formatfehler:

- Prüfkennzeichenfehler. 4-Byte CRC, Erkennungssicherheit

$$p_{\mathrm{E}} = 1 - 2^{-32} \approx 1 - 2 \cdot 10^{-10}$$

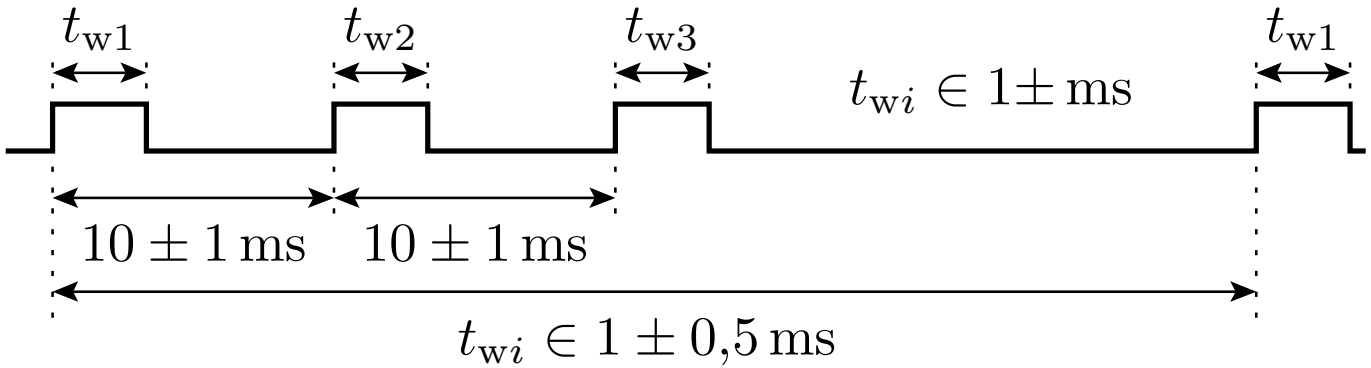
- Soll-/Ist-Abweichung konstante Bytes.
- Datenwechsel in unzul. Zeitfenstern,
- Unzul. Nutzdaten, ...

Überwachung durch die Prozessor-Hardware. Benachrichtigung der Software über Spezialbits. Fehlerbehandlung:

- Wiederholte Anforderung nicht erhaltener Pakete, ...

Signalintegrität von PWM-Signalen

Pulsweitenmodulierte Signale (PWM) kodieren die Information in den Pulsbreiten $t_{\mathrm{w}i}$ und werden u.a. zur Datenweitergabe von Sensoren an Rechner und von Rechnern an Aktoren genutzt.



Überwachbare Formateigenschaften:

- Periodendauer,
- minimale und maximale Pulsbreite,
- Amplitude der Pulse.

Invarianten, WB

Invarianten

Invariante: Aussage, die über die Ausführung bestimmter Programmbefehle hinweg gilt. Überprüfbar

- durch Beweis, bei der Compilierung,
- durch Überwachung während der Laufzeit.

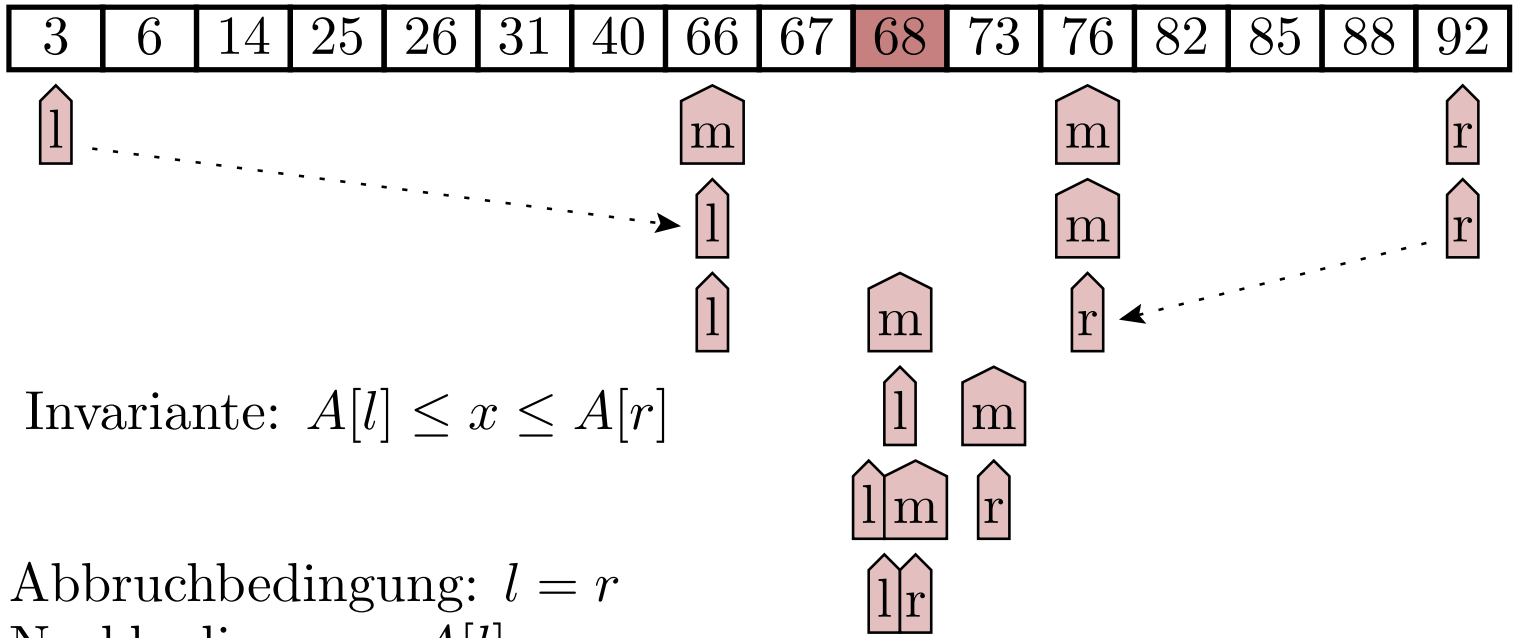
Beispiele für Invarianten:

- Sortieren einer Liste: Anzahl und Summe der Elemente.
- Task-Liste: die max. Anzahl der aktiven Tasks.
- Variable: Wertebereich,
- Zeiger: null oder Adresse eines Objekts mit zulässigem Typ.
- Geschlosses physikalisches System: Energie, Impuls, ...
- Iteration: Vorbedingungen, Schleifeninvarianten, Nachbedingungen.
- ...

Beispiel mit einer Schleifeninvarianten

gesucht: Position Schlüssel x

Vorbedingung: Feld $A[n]$ aufsteigend sortiert, n Elemente



Invariante: $A[l] \leq x \leq A[r]$

Abbruchbedingung: $l = r$

Nachbedingung: $A[l] = x$

Innerhalb der binären Suche nach dem Element mit Schlüsse x darf $A[l]$ nie größer und $A[r]$ nie kleiner als der gesuchte Schlüssel sein.

Wertebereichskontrolle

Die Menge der zulässigen Werte eines Datenobjekts ist meist viel kleiner als die Menge der darstellbaren Werte. Überwachung zusammenhängender Wertebereiche:

```
let a: u32 = ... // Alter Angestellter WB:18...67
                 // WB(u32):0...0xFFFFFFFF
assert!(a < 18 || a > 67, "Ausgabetext");
                 // identisch damit
if a < 18 || a > 67 {panic!("Ausgabetext");}
```

Wenn bei Verfälschung alle darstellbaren Werte gleichwahrscheinlich sind, Erkennungswahrscheinlichkeit:

$$p_{\mathrm{E}} = 1 - \frac{(67 - 18 + 1)}{2^{32}} = 1 - 10^{-8}$$

Würde bedeuten, dass praktisch jede Verfälschung erkannt wird, aber:

- kleine Werte stehen viel öfter als große im Speicher,
- ...

- Verwechslung mit dem Alter einer anderen Person, immer zulässig,
- Verwechslung mit der Hausnummer, oft zulässig, ...

Tatsächliche Erkennungswahrscheinlichkeit der WB-Überwachung:

$$p_{\mathrm{E}} \ll 1 - 10^{-8}$$

Verbesserbar durch

- pseudozufälligen Codierung der zulässigen Werte, ... (vergl. fehlererkennende Codes).
- Typüberwachung, ...

Eine einzelne Wertebereichskontrolle hat in der Regel nur eine geringe Erkennungswahrscheinlichkeit, aber es sind sehr viele Kontrollen (auch für Überläufe) möglich, z.T. schon zur Compile-Zeit; Laufzeit-WB-Überwachung vom Compiler automatisch einfügbar.

Die Leistungsfähigkeit des Verfahrens liegt in der Vielzahl der Kontrollmöglichkeiten.

Syntax

Syntaxtest

Ein Syntaxtest kontrolliert für eine Zeichenfolge, dass sie »ein Wort einer formalen Sprache« ist.

Die Kontrolle und Datenextraktion erfolgt mit einem Automtaten.

Ein spracherkennenden Automat ist zwar selbst kein einfaches, schnell zu schreibendes Programm, aber das Programm für einen spracherkennenden Automaten lässt sich nach bekannten Algorithmen aus den Syntaxregeln der Sprache generieren.

Anwendung: manuelle Dateneingabe.

Formale Sprachen

Eine formale Sprache definiert zulässige Worte durch Syntaxregeln.
Beispiel EBNF (Eweiterte Backus-Naur-Form[4]):

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

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

[4]Siehe http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf.

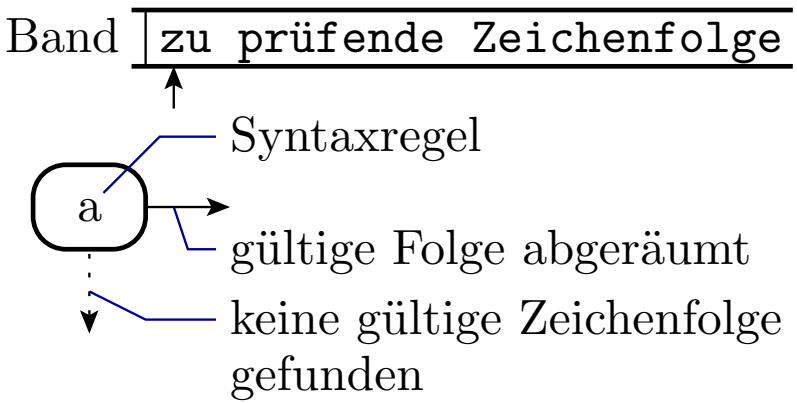
Beispiele für EBNF-Syntaxregeln

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

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

Ziffer = ZiffernAusserNull | "0";
```

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

Spracherkennende Automaten

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

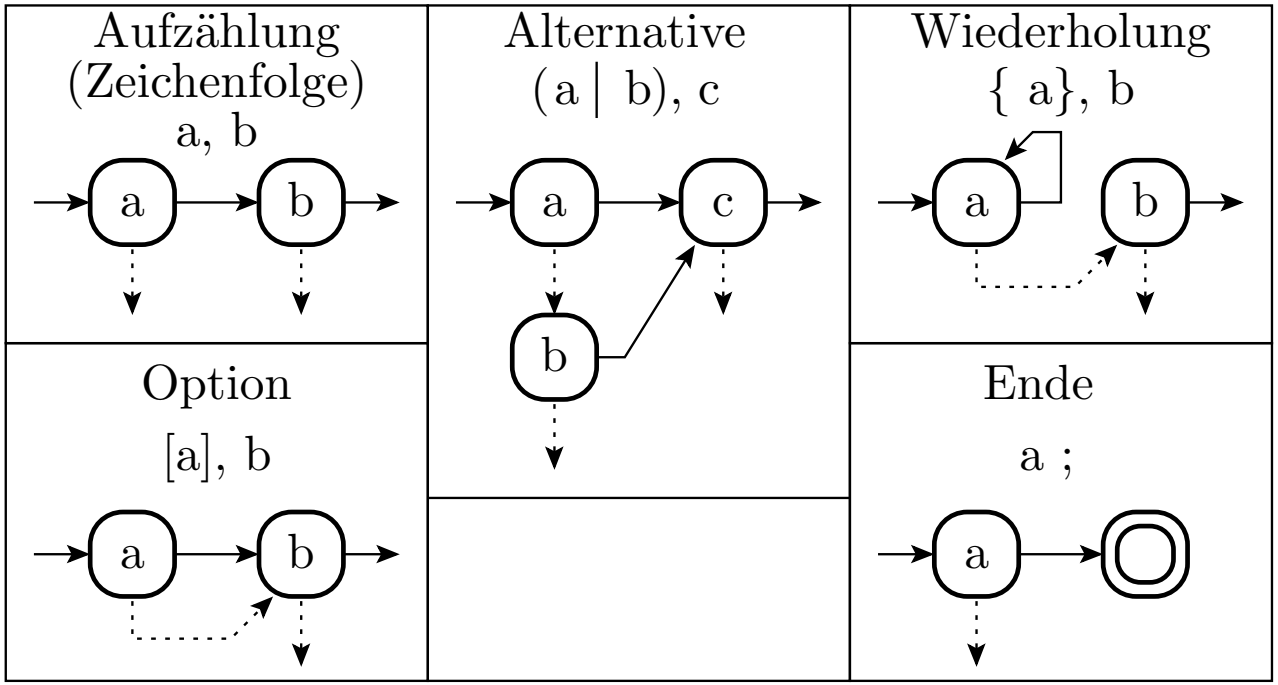


In jedem Automatenzustand wird versucht, eine Zeichenfolge nach der definierten Syntaxregel abzuräumen:

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

$\downarrow$-Übergänge ohne dargestellten Zielknoten enden im Fehlerzustand.

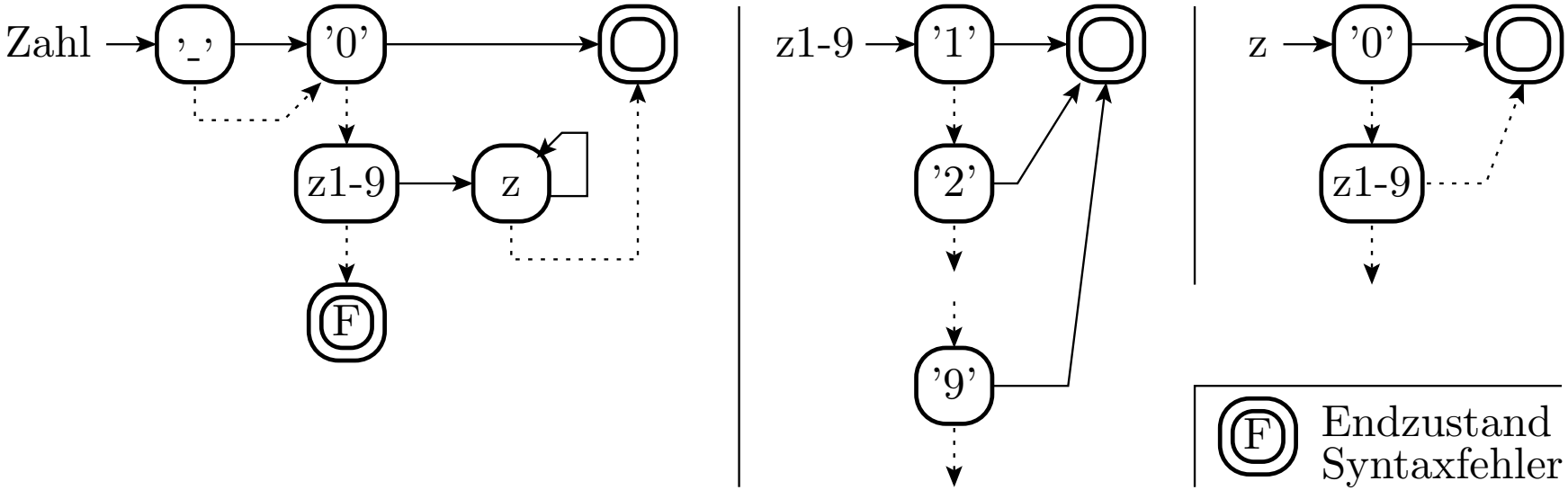
Von der EBNF zum Automaten



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)

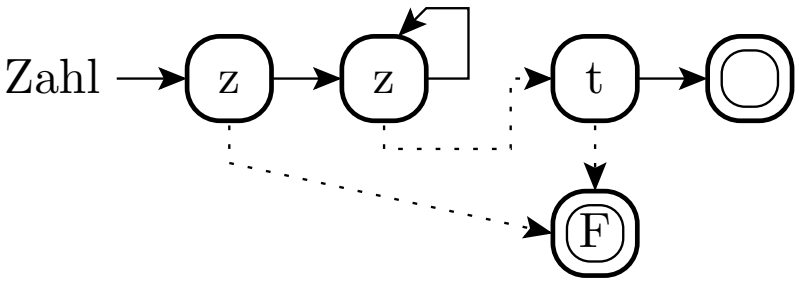
Abräumen an den Kanten

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

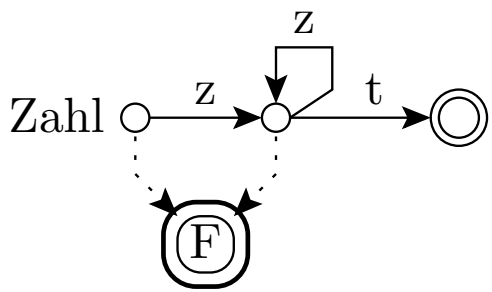
Beschreibung einer Zahl:

Zahl = z, {z}, t; z = '0' | '1' | ... | '9';
t = ' ' | ',' | ...; (Trennzeichen)

Abräumen in den Zuständen



Abräumen an den Kanten



Mit den einfachen Ersetzungsregeln »darf sein«, »darf n-mal vorkommen« lassen sich viele Datenformate und auch Formate für Programme, die in Entstehungsprozessen als Daten eingegeben werden, beschreiben.

Ein Programm für die Spracherkennung und Datenextraktion, dass 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 (E**x**tensible **M**arkup **L**anguage) 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 sind fehlererkennende Codes und Prüfkennzeichen besser. Weniger Programmier- und Rechenaufwand. Höhere Fehlererkennungssicherheit.

Überwachung auf Richtigkeit

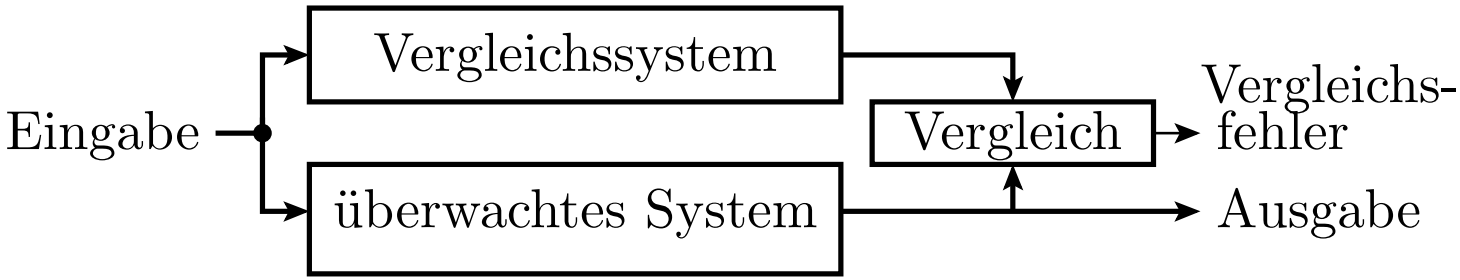
Überwachung auf Richtigkeit

Zur Überwachung auf Richtigkeit sind zusätzlich zu den werteunabhängigen Formatmerkmalen, die repräsentierten Werte zu kontrollieren:

- beim Test in der Regel durch Soll-/Ist-Vergleich oder Vergleich »Golden Device«[5],
- im Einsatz in der Regel durch Mehrfachberechnung und Vergleich.

[5] Referenzsystem, dessen Ausgaben per Definition als richtig gelten: Simulationsmodell, Prototyp, Vorgängerversion, ...)

Mehrfachberechnung und Vergleich



Erkennungswahrscheinlichkeit ist etwa die Diversität

$$p_{\mathrm{E}} \approx Div$$

und die Phantom-FF-Rate ist gleich der diversitären Checker-FF (vergl. Foliensatz 1, Abschn. 2.4 Überwachung un Fehlertoleranz).

Schaffung von Diversität

1. Hardware-Diversität: Unterschiedliche HW.
2. Hardware-Entwurfsdiversität: Unabhängig entworfene HW.
3. Syntaktische Diversität: Unterschiedlich übersetzte Software.
4. Software-Diversität: Unabhängig entworfene SW.
5. Diversitärer Service Anforderung oder Funktion[6].

Ursachen, für die FF korrigierbar sind:

1. zusätzlich Fertigungsfehler und Ausfälle der HW.
2. zusätzlich HW-Entwurfsfehler.
3. zusätzlich FF des Compilers.
4. zusätzlich SW-Entwurfsfehler.
5. zusätzlich Spezifikationsfehler.

[6] Ungeeignet für Verdopplung und Vergleich, da abweichde Sollwerte. .

Anmerkungen zur Software-Diversität

Software-Fehler als Hauptquelle für FF verlangen SW-Diversität:

- Komplette Entwicklung mindestens zweimal.
- durch getrennte Teams, keine Kommunikation,
- aus einer nicht diversitären Spezifikation, ...

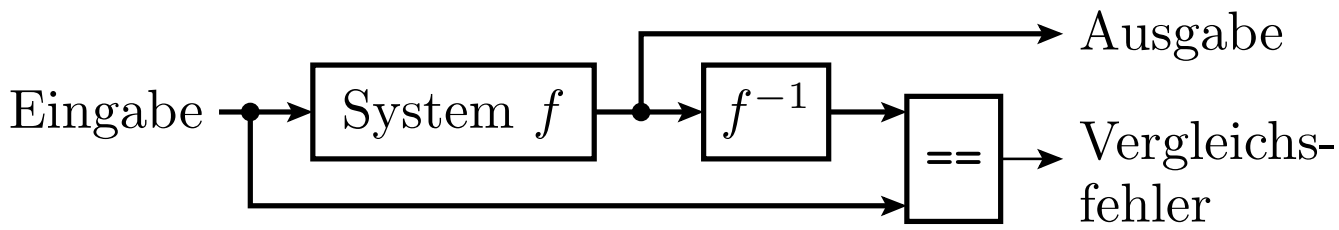
Ursprüngliche euphorische Meinung, dass so Diversität gegenüber allen Fehlern, außer denen in der Spezifikation erzielbar sei, nicht bestätigt. Die direkte oder indirekte Kommunikation der Entwicklungsteams über die Interpretation der Spezifikation, während des Test etc. trägt Gemeinsamkeiten in die Entwürfe. Neigung von Menschen, gewisse Fehler zu wiederholen, ... Erzielbare Divisität laut[7]

$$Div \leq 90\%$$

[7] U. Voges, Software-Diversität und ihre Modellierung - Software-Fehlertoleranz und ihre Bewertung durch Fehler- und Kostenmodelle, Springer (1989)

Rückrechnung und Vergleich

Für umkehrbare Funktion $f(x)$ mit $f^{-1}(f(x)) = x$ lassen sich die Ergebniswerte 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,
- Analog/Digital-Wandlung $\leftrightarrow$ Digital/Analog-Wandlung,
- Daten versenden $\leftrightarrow$ Daten empfangen, ...

Geringeres Risiko, dass sich FF von f und f^{-1} gegenseitig aufheben.
Höhere zu erwartenden Diversität und Erkennungswahrscheinlichkeit.

Funktionsspezifische Kontrollen

Eine Reihe von Zielfunktionen bieten weitere Kontrollmöglichkeiten für das Ergebnis auf Richtigkeit:

- Sortieren: Sortierte Menge enthält alle Elemente der Originalmenge in der richtigen Reihenfolge.
- Suche einen Weg durch einen Graphen: Graph unverändert und Weg erfüllt die Zielvorgaben.
- Suche einen Test für den Nachweis eines Fehlers: Test weist den Fehler nach.

Solche funktionsspezifischen Kontrollen umgehen oft das Diversitätsproblem durch Verschiedenartigkeit von Berechnung und Kontrolle. Hohe Erkennungswahrscheinlichkeiten erzielbar.

Fehlertoleranz

Fehlertoleranz

Reaktionen auf Fehlfunktionen:

- unvorhersehbares Systemverhalten.
- kontrollierter Abbruch, Systemsicherheit gewährleistet.
- Weiterer Systembetrieb mit verringerter Leistung.
- Weiterer Systembetrieb ohne Leistungsverminderung.

Robustheit (Anteil der FF ohne unvorhersehbares Systemverhalten):

$$ROB = \frac{\#FFR}{\#FF} \tag{4}$$

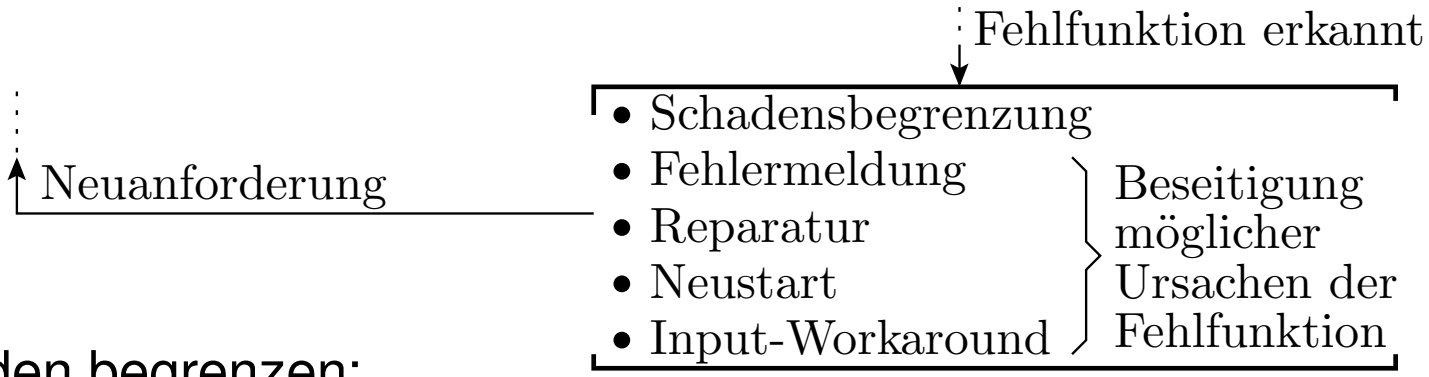
Fehlertoleranz (Anteil der FF, die das System selbst korrigiert):

$$FT = \frac{\#FFT}{\#FF} \tag{5}$$

($\#FFR$ – Anzahl der internen FF ohne unvorhersehbares Systemverhalten; $\#FFS$ – Anzahl der tolerierten internen FF). Fehlertoleranz setzt Robustheit und Robustheit Nachweisbarkeit der FF voraus.

Fehlerbehandlung

Umgang mit Fehlfunktionen im Betrieb



Schaden begrenzen:

- Bearbeitungsabbruch, Daten sichern, ...
- Herstellen eines sicheren Zustands, z.B. Notausschaltung.

Daten zur Fehlerlokalisierung erfassen:

- Fehlermeldung, Core-Dump, Cap-Datei (Windows) erzeugen.

Wiederholung / Vermeidung desselben Versagens:

- Fehlerbeseitigung: Hardware-Tausch, Updates einspielen, ...
- Neuinitialisierung.
- Diversitäre Service-Anforderung / Berechnung: Geänderte Service-Reihung, Eingaben, Berechnungsfluß, ...

Fehlerisolation

Eine sinnvolle automatische Reaktion auf eine FF benötigt

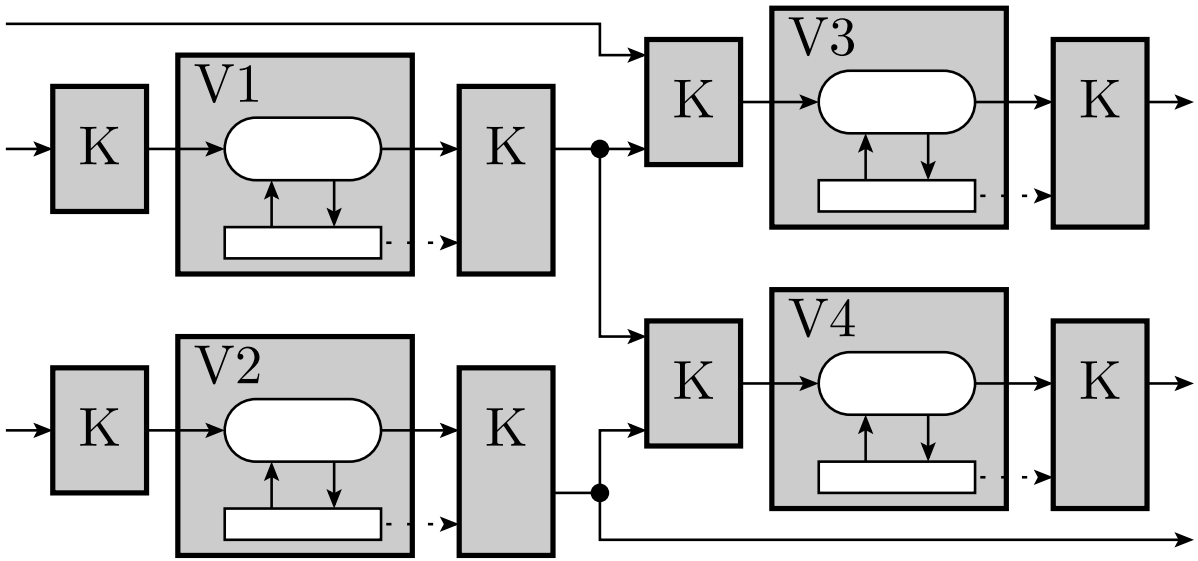
- von der Entstehungsursache der FF unbeeinträchtigte Systemteile und
- von der FF nicht kontaminierte Daten.

Techniken zur Fehlerisolation:

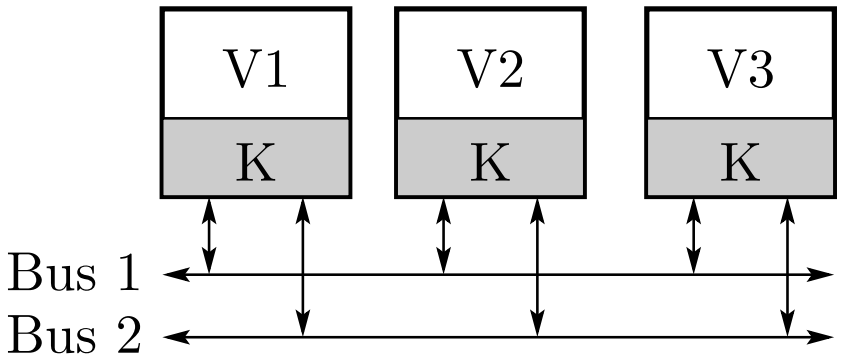
- Datenkontrolle an Schnittstellen zwischen Teilsystemen. Keine Weitergabe erkannter verfälschter Daten.
- Keine Zugriffmöglichkeit auf interne Daten fremder Funktionseinheiten.
- Physikatisch und räumlich getrennte Systeme (Risikominderung gleicher Fehler, zeitgleicher Ausfälle, ...).
- ...

Architekturen zur Fehlerisolation

- Berechnungskonoten mit gerichtetem Datenfluss



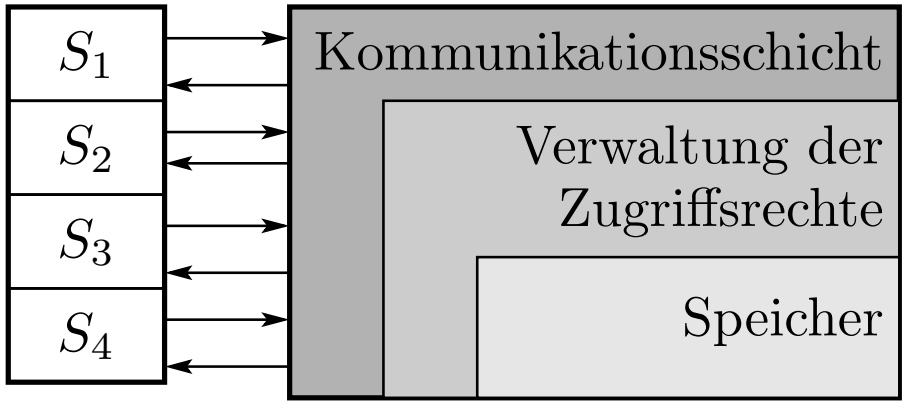
- Steuergeräte an einem redundanten Bussystem



V Verarbeitungs-baustein

K Überwachung und Fehler-isolation

Fehlerisolation in Betriebssystemen

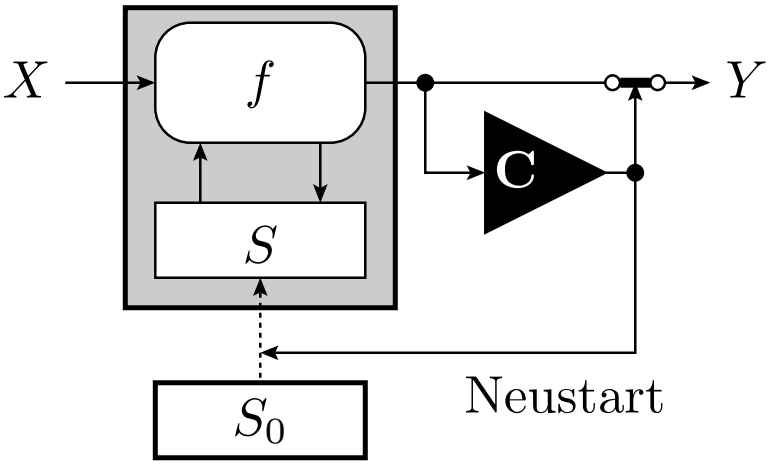


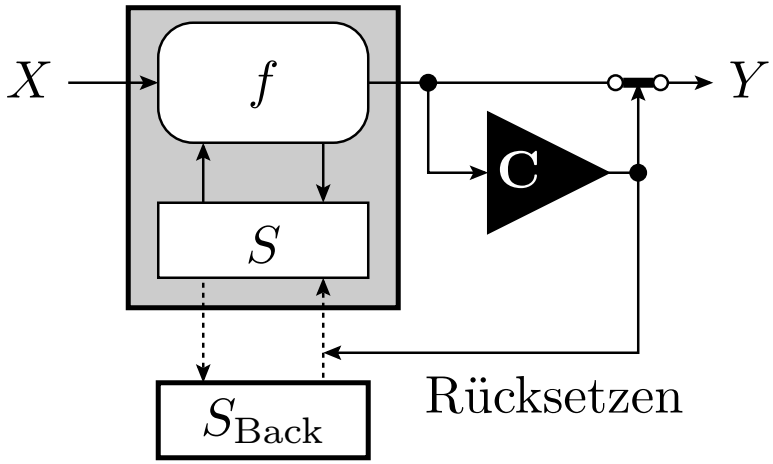
Die zu isolierenden Teilsysteme sind die Prozesse S_i.

- Jeder Prozess sieht nur seinen eigenen virtuellen Speicher,
- die ihm zugeordneten Ein- und Ausgabegeräte und
- bekommt den Prozessor zeitscheibenweise zugeteilt.

Resourcen-Zuordnung (physikalischer Speicher, Ein- und Ausgabegeräte, Kommunikation zu anderen Prozessen, ...) nur über Systemrufe möglich. Nur der Betriebssystemkern hat Universalzugriff (und kann alle Daten kontaminieren).

Neuinitialisierung





- Statische Neuinitialisierung (Reset): fester Anfangszustand,
- Dynamische Neuinitialisierung: Regelmäßiges Backup wärend des Betriebs. Laden des letzten Backups nach Crash.
- Oft werden nur Daten gesichert, die sich nicht problemlos neu berechnen lassen, bei Editoren, Logistiksysteme, Datenbanken, ... die Eingaben seit dem letzten kompletten Backup.
- Berechnungswiederholung vom letzten Backup bis Versagen.

Fail-Fast, Fail-Slow und Ruhestromprinzip

- Fail Fast: Abbruch bei erkannten FF, üblich in der Testphase zur Fehlersuche.
- Fail-Slow: Funktion so lange wie möglich aufrechterhalten, z.B
  - Ersatz fehlerhafter Daten durch sinnvolle Standardwerte,
  - Bei WB-Überlauf Begrenzung auf zulässige Werte,
  - Suche fehlender Dateien an anderen Orten,
  - ...
- Ruhestromprinzip: Konstruktionsprinzip, bei dem das System bei Versagen automatisch in einen sicheren Zustand übergeht.
  - Eisenbahnsignaltechnik: bei fehlendem Ruhestrom Störungsmeldung.
  - Brandmeldeanlage: bei Drahtbruch Alarm.
  - Fahrzeugbremse: Bremsen, wenn Bremsschlauch platzt.
  - ...

Redundanz

Gleichschrittsysteme

- Gleichschrittsysteme: Parallele Ausführung der SL auf replizierten Funktionsbausteinen. Im fehlerfreien Fall übereinstimmende SL und übereinstimmende interne Zustände.
- Sanfter Leistungsabfall (Graceful degradation): Nach Ausfall von Systemkomponenten Fortsetzung (eines Notbetriebs) mit verlängerter Verarbeitungszeit oder vermindertetem Service.

Noo$M$ (N out of M): N benötigte von M vorhandenen Replik.:

- 1oo1: Einfaches System. Nach Ausfall Reparatur.
- 1oo2: Überwachung durch Vergleich. Ab Teilsystemausfall bis Reparatur Betrieb als 1oo1.
- 2oo2: Überwachung durch Vergleich. Ab Teilsystemausfall bis Reparatur optional Betrieb als 1oo1 mit reduzierter Leistung.
- 2oo3: Mehrheitsentscheid. Ab Teilsystemausfall bis Reparatur Betrieb als 2oo2.

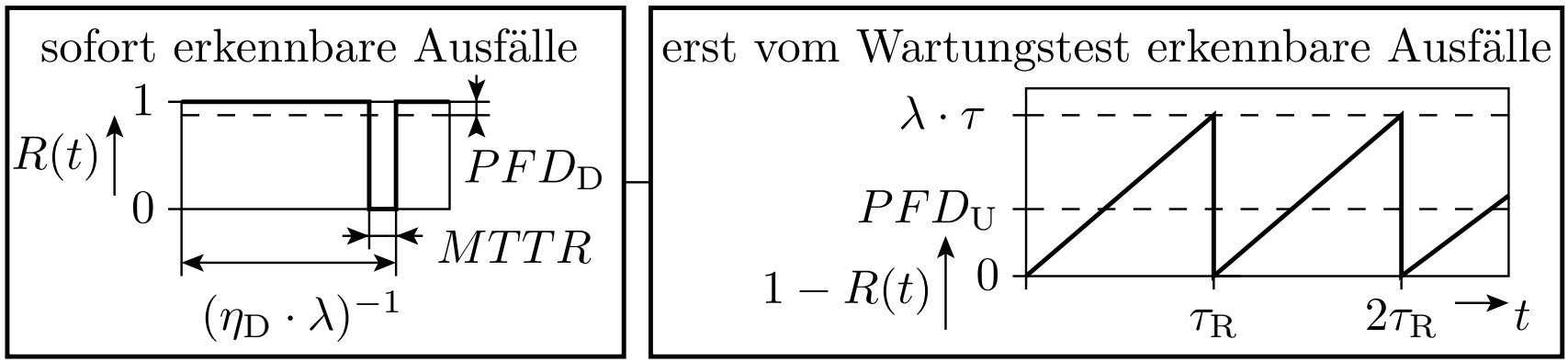
- Alle redundanzfreien Systeme, die regelmäßig gewartet werden, z.B. wie Autos zum TÜV müssen, sind 1oo1.
- Im Maschinenbau je nach Sicherheitsstufe: 1oo1- oder 1oo2.
- Prozessindustrie und Systeme ohne einen in kurzer Zeit erreichbaren sicheren Zustand (z. B.: Flugzeugsteuerungen, Atomkraftwerke, Chemiereaktoren) auch höhere Redundanzen 2oo2, 2oo3 oder 2oo4 üblich.

Sanfter Leistungsabfall

- Bei Komponentenausfall Umverteilung von Aufgaben auf andere Systembestandteile.
- Bei Notstromversorgung Abschalten von Systemteilen.
- Transmission Control Protocol (TCP): auch dann noch eine sichere Punkt-zu-Punkt-Verbindung, wenn wenn einzelne Knoten im Netzwerk überlastet, falsch eingestellt sind oder Daten verfälschen.
- HTML ist aufwärtskompatibel so aufgebaut, dass älterer Browser neue HTML-Einheiten, die sie nicht kennen, ignorieren und den Rest des Dokuments trotzdem dargestellen.
- Ausschluss / Ersatz fehlerhafter Rechner-Knoten und Aufgabenumverteilung auf die verringerte Anzahl.

Verfügbarkeit 1oo1 mit Wartungsintervall τ

1oo1 ist ein System ohne Redundanz. Verfügbarkeit und PFD (Probability of Failure on Demand) siehe Foliensatz 4, Abschnitt 5.5 Wartung.



Sofort erkennbare Ausfälle treten mit einer Rate $\eta_\mathrm{D} \cdot \lambda$ im mittleren Abstand von $(\eta_\mathrm{D} \cdot \lambda)^{-1}$ auf, haben eine mittlere Beseitigungszeit $MTTR \ll \lambda_\mathrm{DD}^{-1}$ und sind im Mittel mit Wahrscheinlichkeit:

$$PFD_{1001\mathrm{D}} = \eta_\mathrm{D} \cdot \lambda \cdot MTTR$$

vorhanden. Ausfälle, die erst bei der Wartung beseitigt werden ...

Ausfälle, die erst vom Wartungstest erkannt werden, sind im Mittel mit Wahrscheinlichkeit

$$PFD_{1001\mathrm{U}} = \frac{(1-\eta_{\mathrm{D}}) \cdot \lambda \cdot \tau}{2}$$

vorhanden. System nicht verfügbar, wenn wegen sofort erkennbarem ODER erst vom Wartungstest erkennbarem Ausfall nicht verfügbar. Für kleine $PFD_{\ldots} \ll 1$ mit guter Näherung:

$$\begin{aligned} PFD_{1\mathrm{oo}1} &= PFD_{1001\mathrm{D}} + PFD_{1001\mathrm{U}} \\ &= \eta_{\mathrm{D}} \cdot \lambda \cdot MTTR + \frac{(1-\eta_{\mathrm{D}}) \cdot \lambda \cdot \tau}{2} \end{aligned}$$

(τ – Wartungsintervall incl. mittlere Reparaturzeit; $MTTR$ – mittlere Reparaturzeit außerhalb der Wartung). Verfügbarkeit als Gegenwahrscheinlichkeit:

$$V_{1\mathrm{oo}1} = 1 - PFD_{1\mathrm{oo}1}$$

Beispiel 1

KFZ: Wartungsintervall $\tau = \frac{10.000\text{km}}{50\,\text{km/h}} = 200\text{h}$, $\lambda = 10^{-3}\,\text{h}^{-1}$, $\eta_\text{D} = 80\%$, $MTTR = 2\,\text{h}$. Gesucht:

- PFD (Probability of Failure on Demand) und
- mittlere Wahrscheinlichkeit, dass das KFZ sicher verfügbar ist:

$$\begin{aligned} PFD_\text{KFZ} &= \eta_\text{D} \cdot \lambda \cdot MTTR + \frac{(1-\eta_\text{D}) \cdot \lambda \cdot \tau}{2} \\ &= 0{,}8 \cdot 10^{-3}\,\text{h}^{-1} \cdot 2\,\text{h} + \frac{0{,}2 \cdot 10^{-3}\,\text{h}^{-1} \cdot 200\text{h}}{2} \\ &= 2{,}16\% \\ V_\text{KFZ} &= 1 - PFD_\text{KFZ} = 97{,}84\% \end{aligned}$$

Verfügbarkeit redundanter 1oo2-Systeme

Wahrscheinlichkeit, dass mindestens eine von zwei unabhängig ausfallenden Komponenten mit Überlebenswahrscheinlichkeit $e^{\lambda t}$ nicht ausgefallen ist:

$$\begin{aligned} R(t)_{\text{1oo2U}} &= 1 - \left(1 - e^{\lambda t}\right)^2 \\ &= 2 \cdot e^{\lambda t} - e^{2\lambda t} \end{aligned}$$

Verfügbarkeit als mittlere Überlebenswahrscheinlichkeit in einem Wartungsintervall τ:

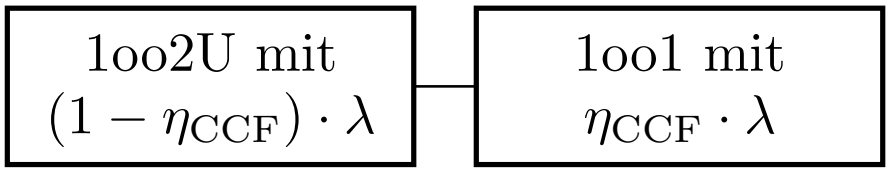
$$\begin{aligned} \bar{V}_{\text{1oo2U}} &= \frac{1}{\tau} \int_0^{\tau} \left(2 \cdot e^{\lambda t} - e^{2\lambda t}\right) \cdot \mathrm{d}\tau \\ &= \frac{2}{\lambda\tau} \cdot \left(1 - e^{\lambda t}\right) - \frac{1}{2\lambda\tau} \cdot \left(1 - e^{2\lambda t}\right) \end{aligned}$$

Mit der Näherung:

$$e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{6}$$

$$\bar{V}_{1oo2U} = \frac{2}{\lambda\tau} \cdot \left(1 - \left(1 - \lambda\tau + \frac{(\lambda\tau)^2}{2} - \frac{(\lambda\tau)^3}{6}\right)\right)$$
$$- \frac{1}{2\lambda\tau} \cdot \left(1 - \left(1 - 2\lambda\tau + \frac{(2\lambda\tau)^2}{2} - \frac{(2\lambda\tau)^3}{6}\right)\right) = 1 - \frac{(\lambda\tau)^2}{3}$$

$$PFD_{1oo2U} = 1 - \bar{V}_{1oo2U} = \frac{(\lambda\tau)^2}{3}$$



Ein Anteil η_{CCF} der Ausfälle verursacht wegen gemeinsamer Ursachen (Common Cause Failurs) den gleichzeitigen Ausfall beider Systeme. Modellierung als Reihenschaltung:

- 1oo2-System für alle unabhängigen Ausfälle und

$$PFD_{1oo2U} = \frac{(1 - \eta_{CCF}) \cdot (\lambda\tau)^2}{3}$$

- ein 1oo1-System für die gleichzeitigen Ausfälle ...

- ein 1oo1-System für die gleichzeitigen Ausfälle:

$$PFD_{1oo1} = \eta_{CCF} \cdot \eta_D \cdot \lambda \cdot MTTR + \frac{\eta_{CCF} \cdot (1 - \eta_D) \cdot \lambda \cdot \tau}{2}$$

Gesamt-PDF für $PFD_{\ldots} \ll 1$ für Reparatur, solange noch eine Komponente verfügbar ist, erst zum Wartungstermin:

$$PFD_{1oo2} = \frac{((1 - \eta_{CCF}) \cdot \lambda \cdot \tau)^2}{3} + \eta_{CCF} \cdot \eta_D \cdot \lambda \cdot MTTR + \frac{\eta_{CCF} \cdot (1 - \eta_D) \cdot \lambda \cdot \tau}{2}$$

Weitere N aus M Systeme

- 2oo2: Zwei identische Systeme (Master und Checker) im Gleichschritt mit Ergebnisvergleich. Sobald ein System versagt, nicht mehr sicher verfügbar. Verfügbarkeit und PFD wie 1oo1 mit der doppelten Ausfallrate:

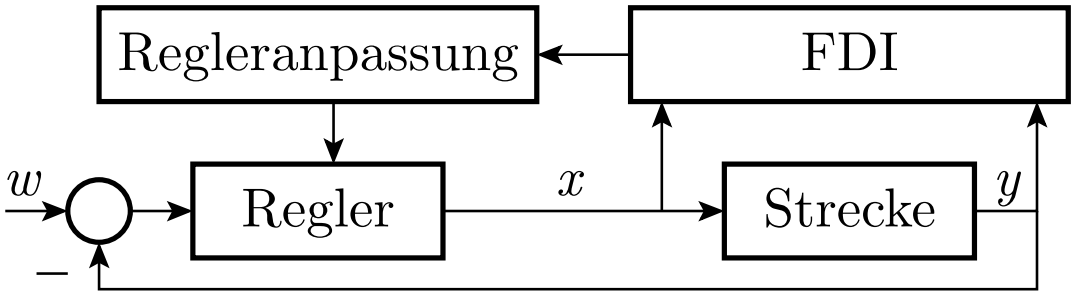
$$PFD_{2oo2} = 2 \cdot \eta_D \cdot \lambda \cdot MTTR + (1 - \eta_D) \cdot \lambda \cdot \tau$$

- PFD nach [3] für N von M funktionierende mit Ausfallrate λ unabhängig voneinander ausfallende identische Komponenten; Reparaturintervall τ:

N	Noo1	Noo2	Noo3	Noo4
1	$\frac{\lambda \cdot \tau}{2}$	$\frac{(\lambda \cdot \tau)^2}{3}$	$\frac{(\lambda \cdot \tau)^3}{4}$	$\frac{(\lambda \cdot \tau)^4}{5}$
2	-	$\lambda \cdot \tau$	$(\lambda \cdot \tau)^2$	$(\lambda \cdot \tau)^3$
3	-	-	$\frac{3 \cdot \lambda \cdot \tau}{2}$	$2 \cdot (\lambda \cdot \tau)^2$
4	-	-	-	$2 \cdot \lambda \cdot \tau$

Anwendungsspez. Lösungen

Fehlertolerantes Regelsystem

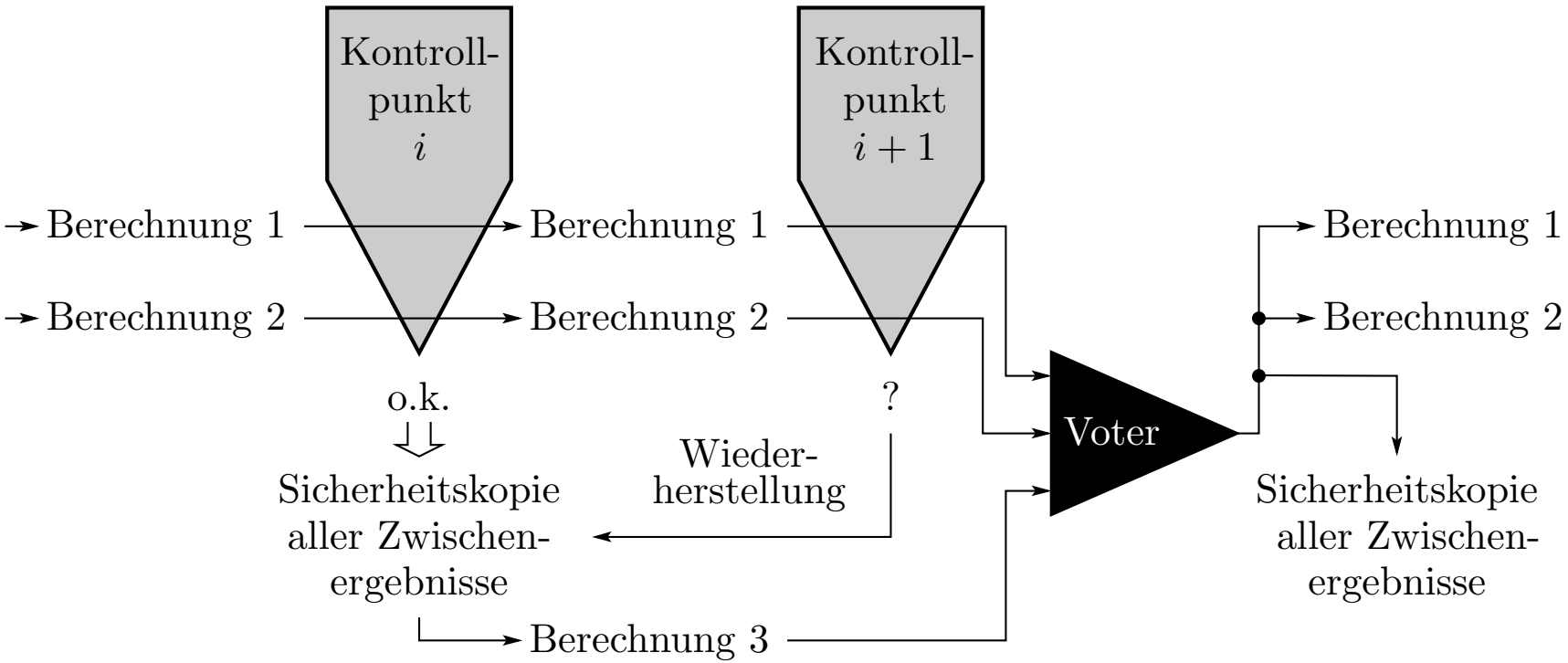


In einem Reglersystem wird vom Sollwert w der zu regelnde Ist-Wert y abgezogen. Aus der Differenz bildet der Regler den Stellwert x für die Regelstrecke (z.B. eine Heizung, wenn y eine Temperatur ist).

Fehlertoleranz gegenüber FF von Regler und Strecke:

- Zusatzmodul zur Fehlerdiagnose (Fehler Detektion, Isolation und Identifikation, FDI) überwacht Stellwert und Ist-Wert.
- Regleranpassung: Bei signalisierter FF Änderung der Reglerfunktion so, dass eine Mindestfunktionalität gewährleistet bleibt.

Check-Point-Roll-Back-Recovery [2]



- Nur zwei parallel ausgeführte Berechnungen.
- An einprogrammierten Kontrollpunkten im Programm werden die Bearbeitungszustände[8] verglichen.

[8] Werte der Variablen, Register, ...

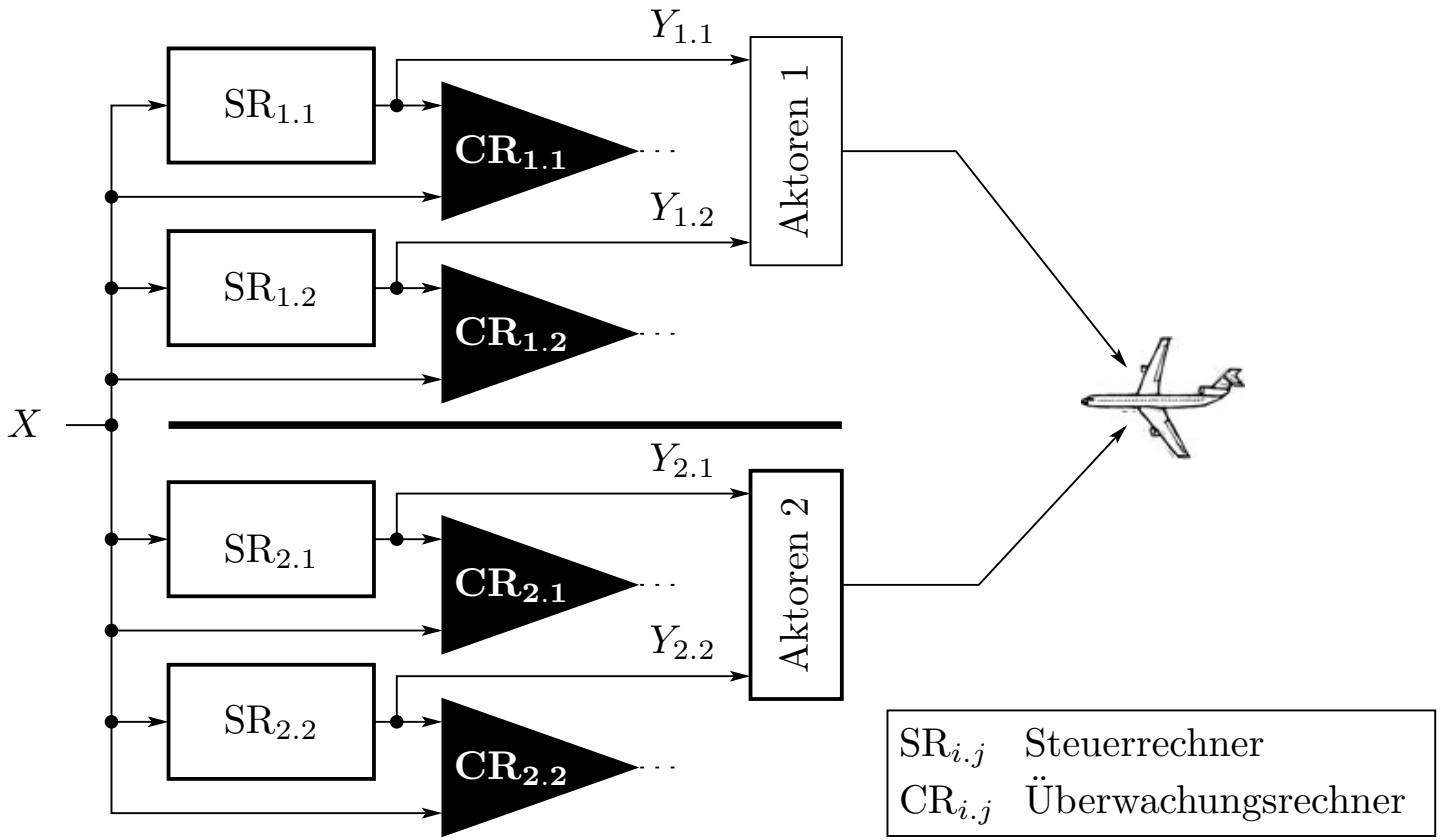
- Bei Übereinstimmung Speicherung des Bearbeitungszustands in einem geschützten Speicher.
- Bei Abweichung, Laden der letzten Sicherheitskopie und Berechnungswiederholung (Roll-Back Recovery).
- Nach Roll-Back Recovery am nächsten Kontrollpunkt wieder Vergleich.
- Wenn Übereinstimmung, diesen als gesicherten Zustand speichern, sonst Abbruch.

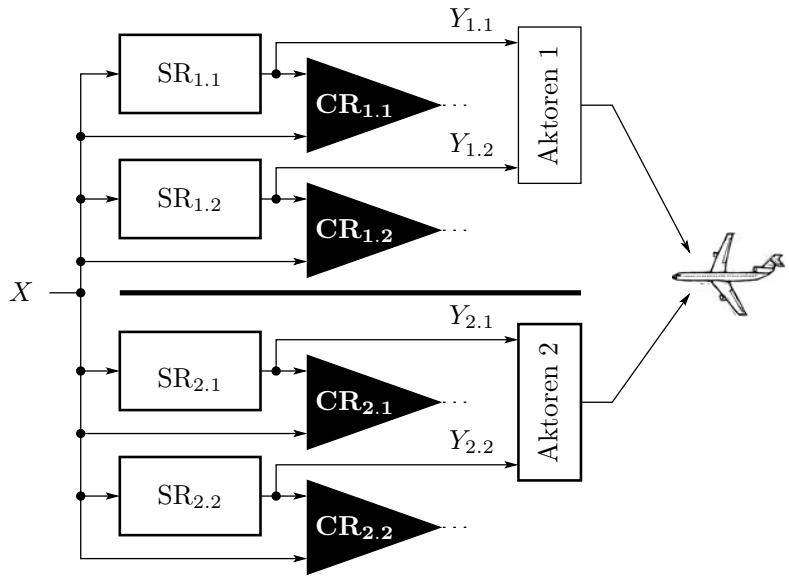
Sequoia-System [1]:

- Berechnung auf zwei Prozessoren mit eigenem Write-Back-Cache.
- Vergleich in jedem Takt.
- Zustands-Backup bei Ereignissen wie Stack-Überlauf und Prozesswechsel.
- Hauptspeicher hat die Funktion des stabilen Speichers.

Flugsteuersystem Airbus A3XX [4]

Hochsicherheitskritische Anwendungen müssen möglichst alle Fehlfunktionen, auch solche durch nicht erkannte Entwurfsfehler, nicht erkannte Fertigungsfehler und Ausfälle tolerieren.





- Zwei identische Systeme mit allen Sensoren, Aktoren und zwei Rechnerpaaren.
- Jedes Rechnerpaar besteht aus einem Steuerrechner $SR_{i.j}$, der die Aktoren ansteuert, und einem Überwachungsrechner $CR_{i.j}$.
- Normalzustand Rechner $SR_{1.1}$ steuert und $CR_{1.1}$ überwacht. Zweites Rechnerpaar Stand-By. System 2 abgeschaltet.
- Bei Ausfall übernimmt Rechnerpaar 1 von Rechnerpaar 2. Bei Komplett-, Sensor- oder Aktorausfällen übernimmt System 2 von System 1.

Diversität: Rechner unterschiedlicher Hersteller, getrennte Software-Entwicklung nach Spezifikationen, die unabhängig von einer gemeinsamen Basisspezifikation abgeleitet wurden.

RAID und Backup

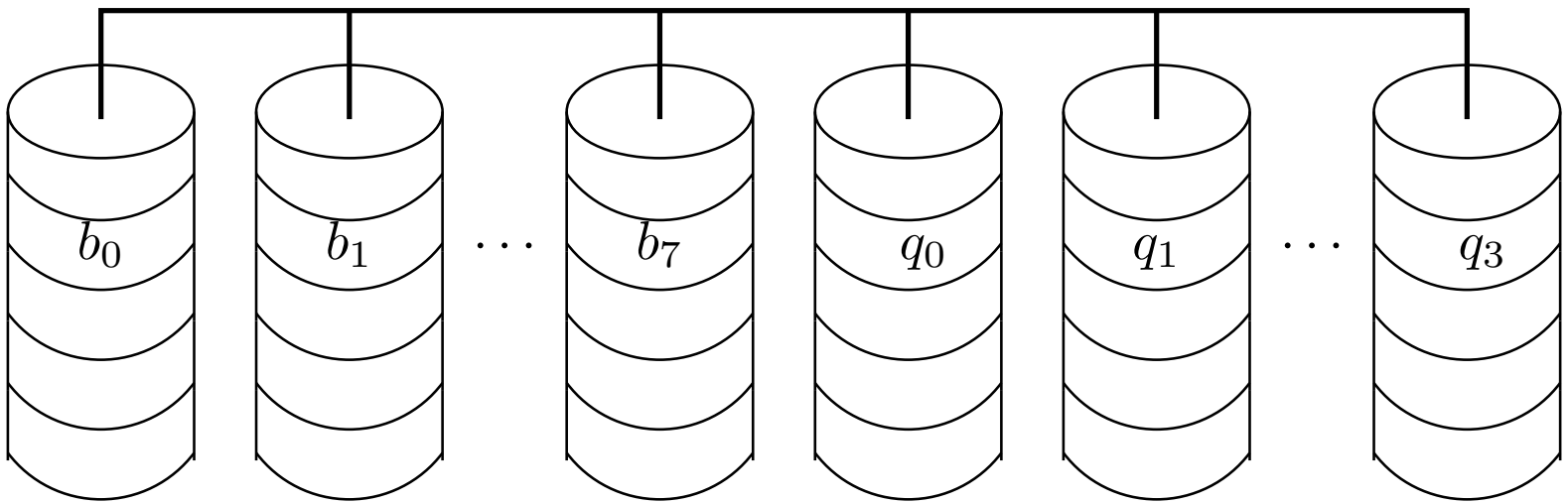
RAID, RAID Level 1

RAID – **R**edundant **A**rray of **I**ndependent **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

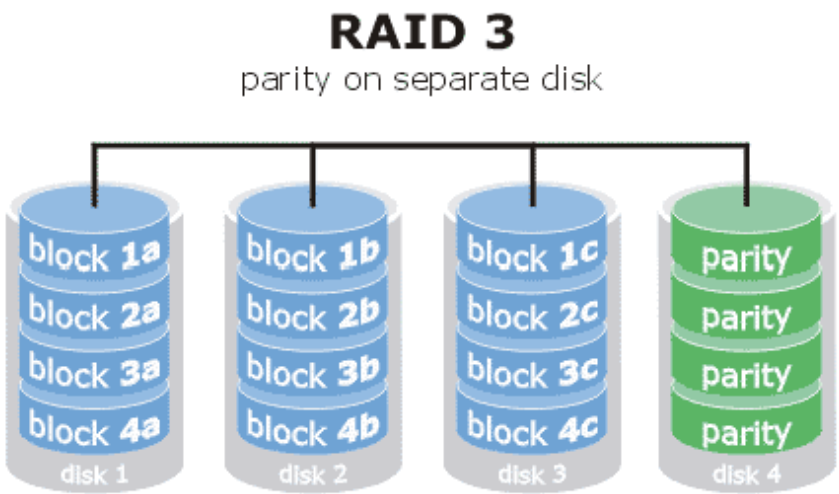
Bei RAID Level 2 werden die Daten in einem 1-Bit-korrigierenden Hamming-Code gespeichert, und zwar jedes der w Daten- und der r Kontrollbits auf einer anderen Platte, z.B. $w = 8$ Datenbit- und $r = 4$ Kontrollbitplatten. Im Vergleich zu RAID 1 werden statt der doppelten Plattenanzahl nur 50% mehr Platten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



Gilt als aufwändig und ungebräuchlich.

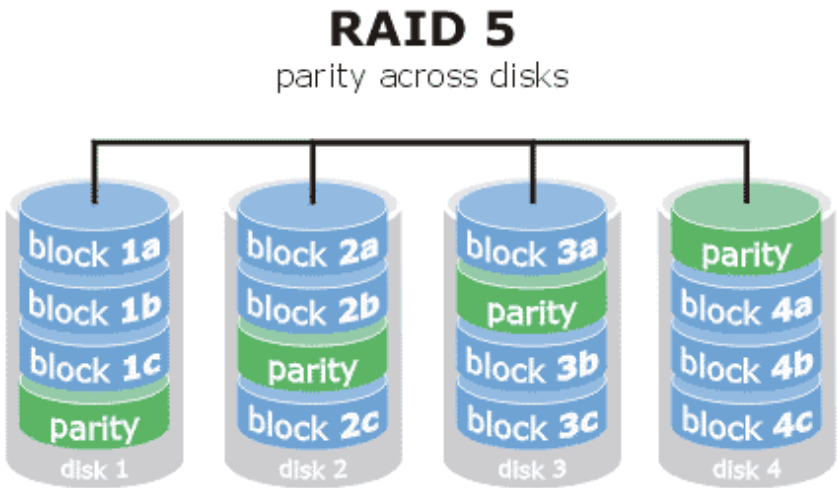
RAID Level 3

Auf einer Extra-Platte wird bitweise die Querparitätsbit der anderen Platten gebildet. Zusätzlich werden auf jeder Platte die Längsparitätsbits (oder Prüfkennzeichen) gespeichert. Mit einer zusätzlichen Blockparität ist eine 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.

Literatur

6. Literatur

[1] P.A. Bernstein.
Sequoia: a fault-tolerant tightly coupled multiprocessor for transaction processing.
Computer, 21(2):37–45, 1988.

[2] D. K. Pradhan, D. D. Sharma, and N. H Vaidya.
Roll-forward checkpointing schemes.
In *Lecture Notes in Computer Science 744*, pages 93–116. Springer Verlag, 1994.

[3] Marvin Rausand and Arnljot Hsyland.
Systems Reliability Theory, Models, Statistical Methods, and Applications.
Wiley-Interscience, 2004.

[4] Pascal Traverse.
Dependability of digital computers on board airplanes.
Dependable Computing for critical applications, 4:134–152, 1991.