Test und Verlässlichkeit (F3)
Foliensatz 3:
Kontrollen und Fehlerbehandlung

Prof. G. Kemnitz

Institut für Informatik, Technische Universität Clausthal

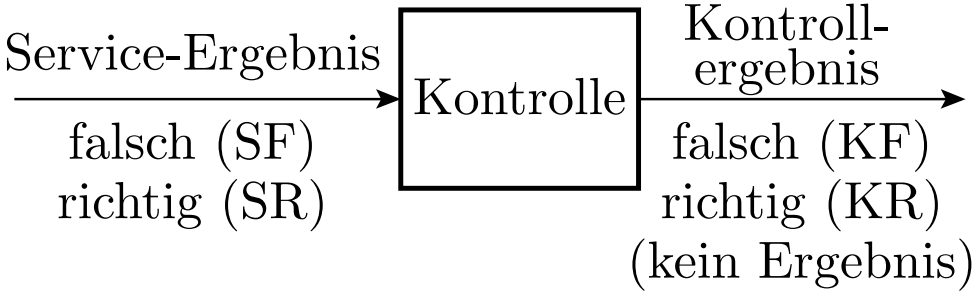
20. Juni 2015

Inhalt F3: Kontrollen und Fehlerbehandlung

Überblick

Kontrollen

Kontrollen sind Service-Leistungen zur Überprüfung der Ergebnisse anderer Service-Leistungen (vergl. Foliensatz F1):



Ergebnis	Kontroll-ergebnis	Kontroll-aussage
falsch	falsch	FF
falsch	richtig	FR
richtig	falsch	RF
richtig	richtig	RR

- Erkennungswahrscheinlichkeit:

$$p_{\mathrm{E}} \approx \frac{\mathrm{Anz}\,(\mathrm{FF})}{\mathrm{Anz}\,(\mathrm{SF})}$$

- Maskierungswahrscheinlichkeit:

$$p_{\mathrm{M}} = 1 - p_{\mathrm{E}} \approx \frac{\mathrm{Anz}\,(\mathrm{FR})}{\mathrm{Anz}\,(\mathrm{SF})}$$

- Phantomfehlerwahrscheinlichkeit:

$$p_{\mathrm{Ph}} \approx \frac{\mathrm{Anz}\,(\mathrm{RF})}{\mathrm{Anz}\,(\mathrm{SR})}$$

Kontrollen zur Sicherung der Verlässlichkeit erfolgen

- während des Entwurfs und der Fertigung zur Fehlervermeidung,
- nach dem Entwurf und der Fertigung zur Fehlerbeseitigung und
- während des Betriebs zur Schadensbegrenzung und Ergebniskorrektur.

Dieser Abschnitt behandelt hauptsächlich Kontrollen und Reaktionen auf Fehlfunktionen während des Betriebs.

Kontrollen während des Betriebs

Kontrollen auf Zulässigkeit[1]:

- Fehlererkennende Codes, Prüfkennzeichen,
- Datenformate, Syntax,
- Datentypen, Wertebereiche, Zeitschranken, ...

Kontrollen auf Richtigkeit:

- Soll-/Ist-Vergleich, Mehrversionsvergleich,
- Loop-Back-Test, Probe, ...

Wartungstests[2]:

- Einschalttests zur Kontrolle auf Hardwareausfälle, Konfigurationsfehler, ...
- Konsistenz von Dateisystemen und Datenbanken, ...

[1] Ein unzulässiges Ergebnis ist immer falsch. Ein zulässiges Ergebnis kann, aber muss nicht richtig sein.

[2] Kontrollen, die nicht ständig, sondern nur in gewissen Zeitintervallen durchgeführt werden.

Reaktion auf erkannte Fehlfunktionen

Schadensvermeidung
- Daten retten, gefahrenfreien Zustand herstellen[3], ausschalten, ...

Problemprotokollierung
- für Anwender »Input-Workarounds«
- für Entwerfer »Change Requests«

Wiederherstellung der Betriebsbereitschaft
- Neustart, Laden von Sicherungskopien, ...

Ergebniskorrektur
- fehlerkorrigierende Codes,
- (diversitäre) Mehrfachberechnung, ...

[3] Für das System selbst z.B. gegen Überhitzung Leistungsaufnahme reduzieren, bei Laptops beim Herunterfallen Schreibköpfe der Festplatte in Parkposition bringen, ... Bei Cyper-Physikalischen Systemen wie Fahrzeug- und Anlagensteuerungen Personen- und Sachschaden vermeiden.

Einige spezielle Reaktionen

Eingabefehler

- aussgekräftige Fehlermeldung, Wiederholschleife, bis Eingabe zulässig, ...

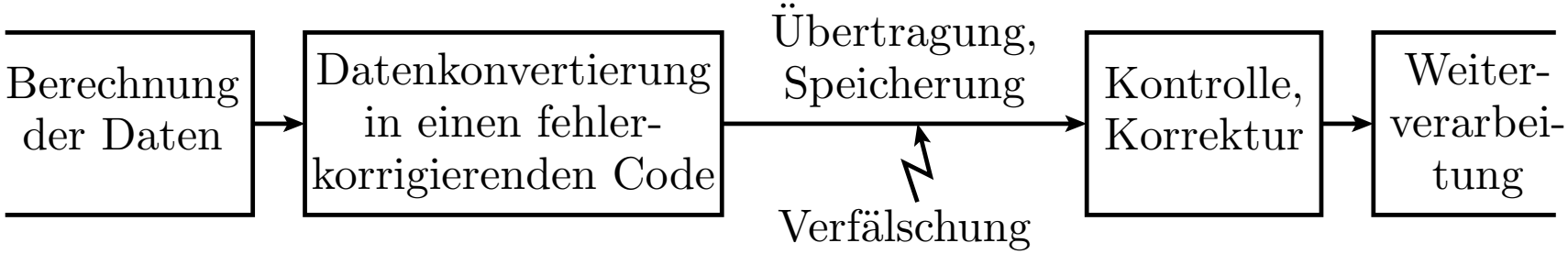
Überschreitung von Zeitschranken

- Bearbeitungsabbruch, System für andere Service-Leistungen verfügbar halten.

»Change Request«

- Zusammenstellung aller Informationen zur Nachbildung des Fehlverhaltens (Eingaben, Core-Dump, Treiberinfos, ...)

Fehlerkorrigierende Codes



Fehlertoleranz, Redundanz, Fehlerisolation

Fehlertolerante Systeme korrigieren selbständig interne Fehlfunktionen. Erfordert Redundanz[4] und Fehlerisolation[5].

- Fehlerkorrigierende Codes: Informationsredundanz. Zusätzliche zu speichernden und zu übertragende Bits.
- Wiederholung: Zeitredundanz. Zusätzliche Rechenzeit.
- (Diversitäre) Mehrfachberechnung: Funktionale Redundanz. Mehrere gleiche Service-Anbieter mit unterschiedlichen (unabhängig entstandenen Fehlern).
- Hardware-Redundanz. Reserveeinheiten, die bei Ausfall die Funktion der aktiven Komponenten übernehmen.

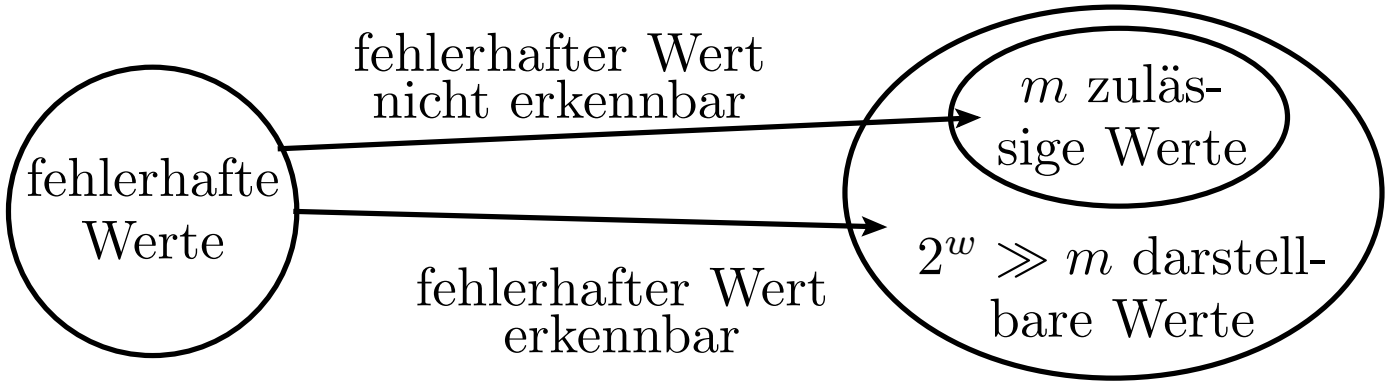
[4] Für eine fehlerfreies System nicht erforderliche Systemressourcen.

[5] Maßnamen zur Verhinderung der Ausbreitung verfälschter Daten über Teilsystemgrenzen hinaus. Im Fehlerfall nicht kontaminierte Teilsysteme: Backup-Speicher, Wiederanlauffunktionen, ...

Informationsredundanz

Informationsredundanz

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



Ein Darstellung mit $w \gg \log_2(m)$ (die überwiegende Mehrheit der darstellbaren Werte ist unzulässig) erlaubt eine Fehlererkennung und bei geschickter Codierung sogar eine Korrektur ausgewählter Datenverfälschungen.

Erkennungswahrscheinlichkeit

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

$$p_{\mathrm{E}} \approx 1 - \frac{\mathrm{Anz(ZW)}}{2^w} \tag{1}$$

  (Anz(ZW) – Anzahl der zulässigen Werte; w – Bitanzahl zur Darstellung).
- Kann durch Wahl vom w beliebig groß gewählt werden.
- Wenn fehlerhafte Werte unverhältnismäßig oft zulässig sind, ist p_{E} deutlich kleiner.
- Wenn fehlerhafte Werte unverhältnismäßig oft unzulässig sind, ist p_{E} deutlich größer.

Beispiel Rechtschreibtest

Wort im Wörterbuch enthalten?

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

- Anzahl der mit Anz(Zeichen) darstellbaren Zeichenketten:

$$2^{8 \cdot \text{Anz(Zeichen)}}$$

  Anteil der gültigen Worte fast null. Nach Gl. 1 $p_\text{E} \approx 1$.
- Tatsächlich $p_\text{E} \approx 80\%$, weil Schreibfehler viel öfter als zufällige Bitverfälschungen gültige Worte sind.
- Es gibt auch relativ viele Worte, die im Wörterbuch fehlen. Phantomfehlerwahrscheinlichkeit typisch $p_\text{Ph} \approx 1\%$.

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.

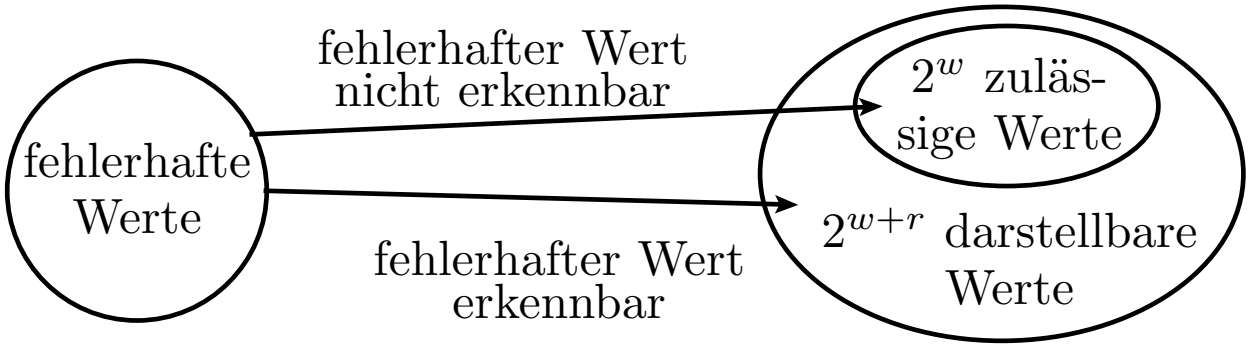
Für die Einzelbitfehlererkennung genügt ein Code, in dem nur Werte mit gerader (ungerader) Anzahl von Einsen gültig sind (Erweiterung um ein Paritätsbit).

Anteil der gültigen Codeworte 50%, Erkennugswahrscheinlichkeit

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

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, aber so, dass Verfälschen weder bevorzugt auf zulässige noch auf unzulässige Datenworte abgebildet werden, so dass Gl. 1 gilt. Mit 2^w zulässigen Werten und 2^{w+r} darstellbaren Werten beträgt die Erkennungswahrscheinlichkeit:

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

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 zulässigen Werte ist mindestens 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

```
        10010101101
⊙            10011
-----------------------
⊕     1001|0101101
⊕    10010|101101
⊕   00000|00000
⊕  000000|0000
⊕ 1001010|1101
-----------------------
  1000111|0100111
```

Decodierung

```
  1000111001001111 : 10011
⊕ 10011
-----
   10110
 ⊕ 10011
 -------
    10101
    10011
    -----
     11000
   ⊕ 10011
   -------
     10111
  ⊕  10011
  --------
      10011
    ⊕10011
    ------
     00000 (Rest null, fehlerfrei)
```

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 einer null oder eins ist eine UND-Verknüpfung und der Operator $\oplus$ eine modulo-2-Addition (EXOR). Das Produkt beider Polynome

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

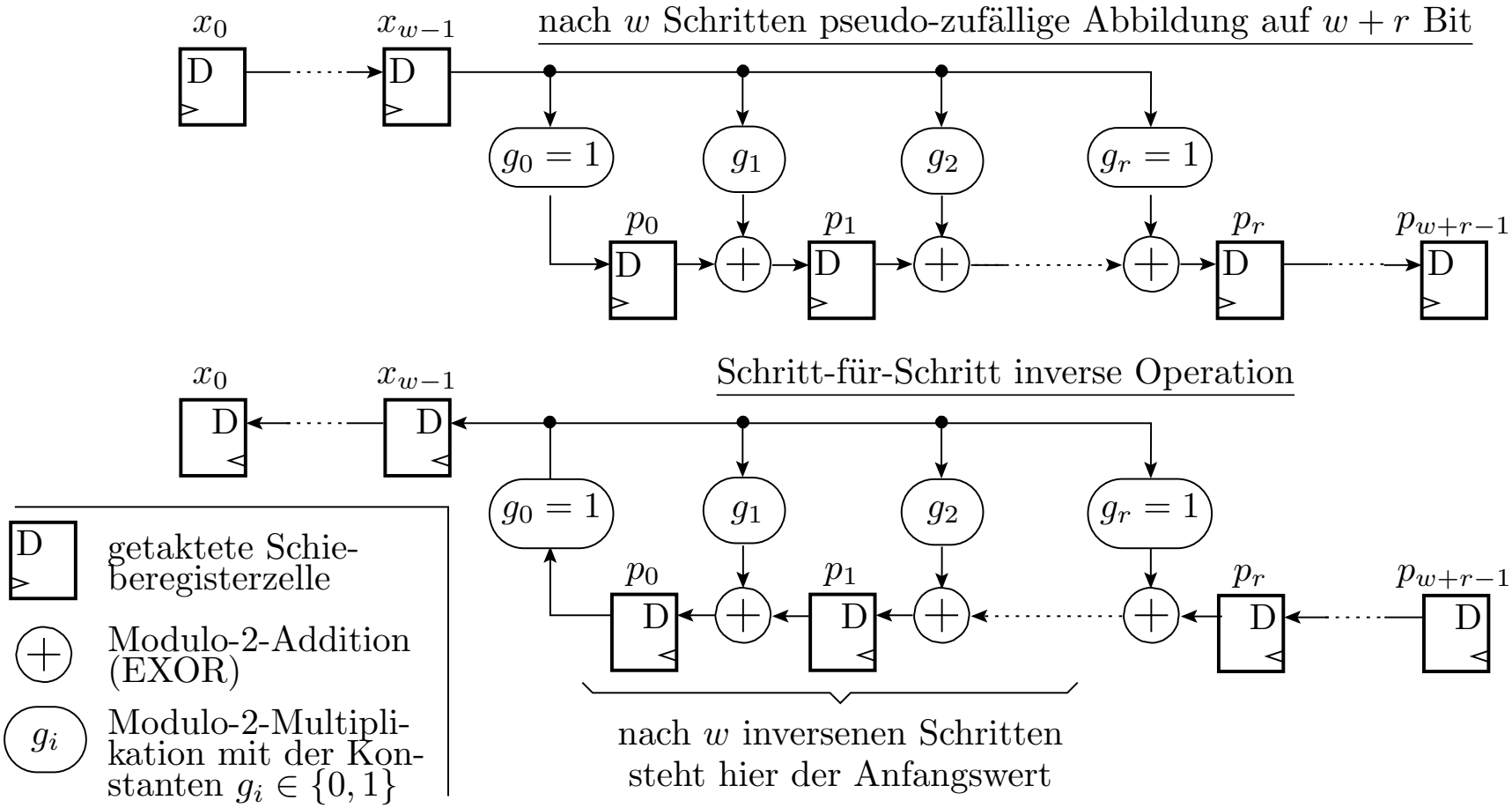
repräsentiert denselben Bitvektor, der für die Multiplikation der Folgen auf der Folie zuvor berechnet wurde. Die Polynomdivision:

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

liefert ohne Rest das Polynom der Originalfolge.

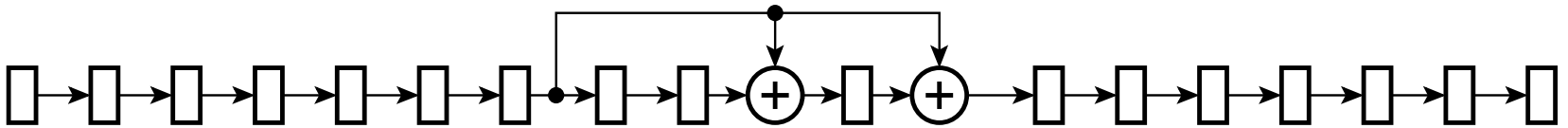
Linear rückgekoppelte Schieberegister

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



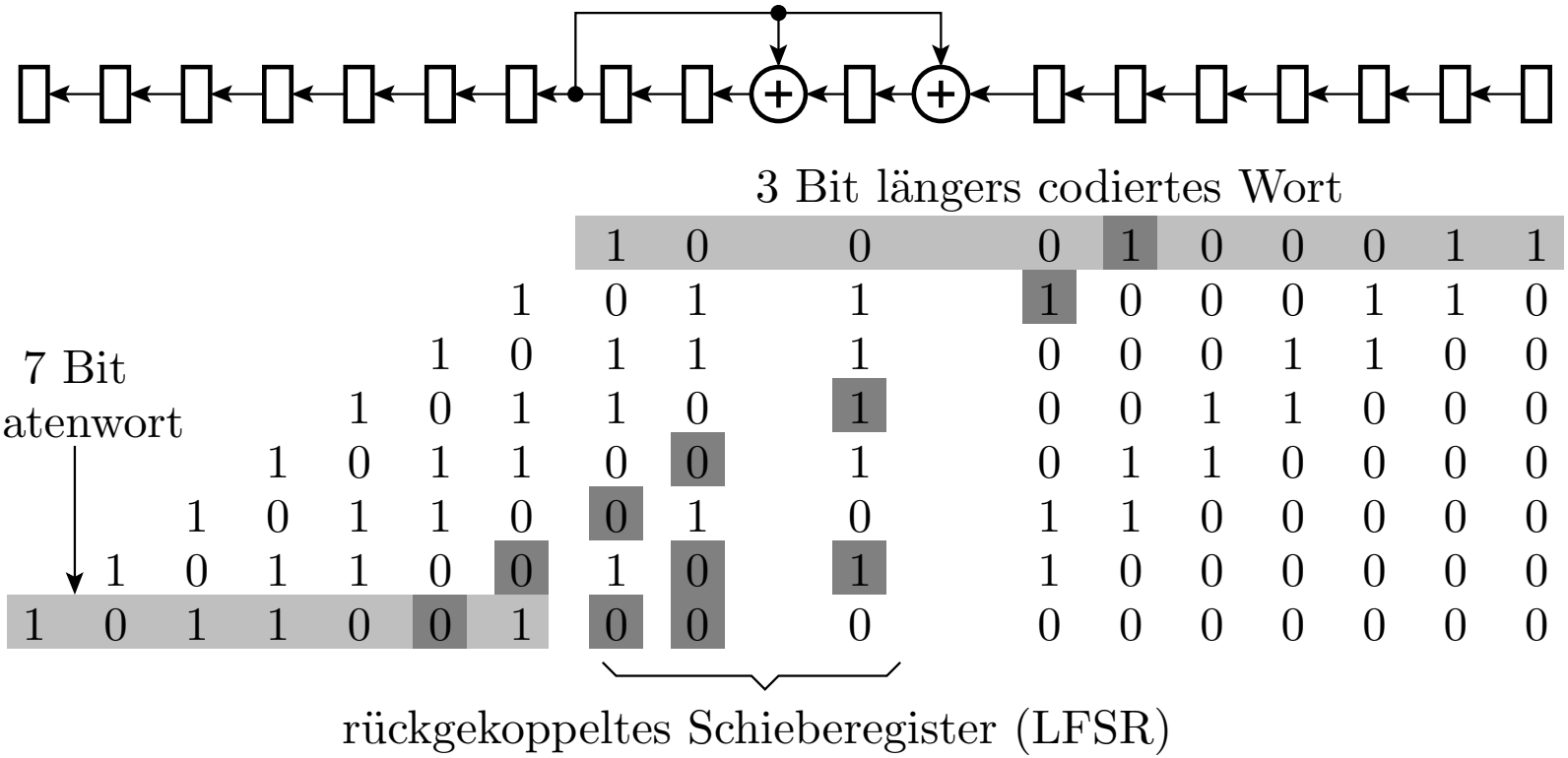
1	0	1	1	0	0	1	0	0		0		0	0	0	0	0	0	0
	1	0	1	1	0	0	1	0		1		1	0	0	0	0	0	0
		1	0	1	1	0	0	1		0		1	1	0	0	0	0	0
			1	0	1	1	0	0		1		0	1	1	0	0	0	0
				1	0	1	1	0		1		0	0	1	1	0	0	0
					1	0	1	1		1		0	0	0	1	1	0	0
						1	0	1		1		1	0	0	0	1	1	0
							1	0		0		0	1	0	0	0	1	1

Datenwort
7 Bit

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}.

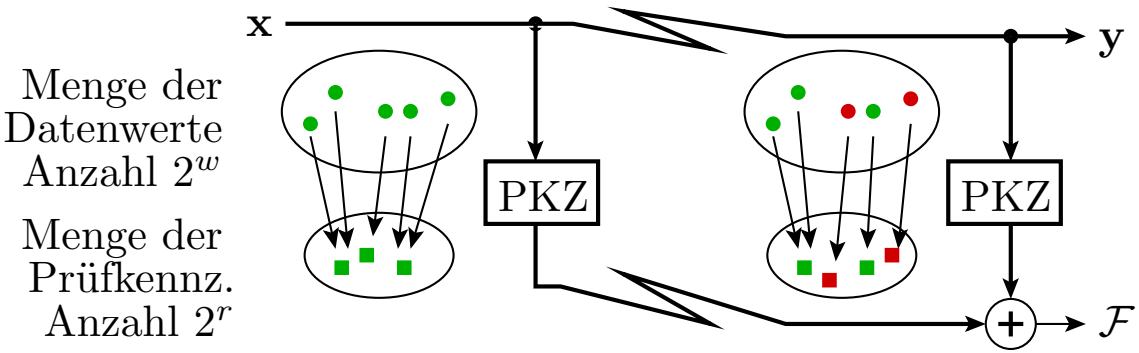
Rückgewinnung und Kontrolle



- 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 gebildeten Prüfkennzeichen gilt:

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

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

Prüfsummen

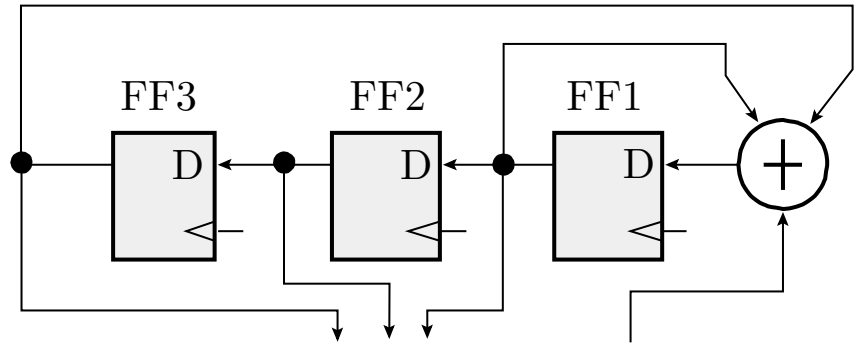
Prüfkennzeichenbildung durch Aufsummierung (arithmetisch, bitweises EXOR, ...).

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	30	0001 1110	30	1100

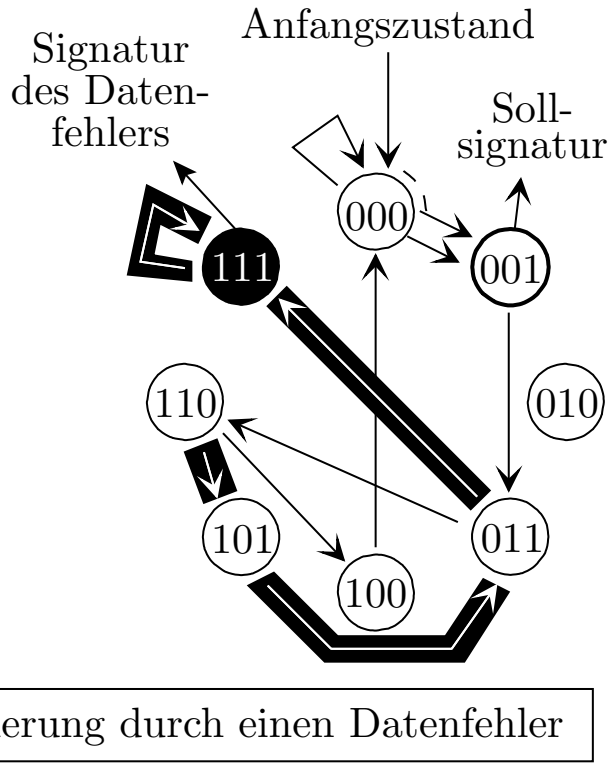
Bei »einfacher Genauigkeit« und »bitweisem EXOR« erscheint die Annahme »pseudo-zufällige Abbildung« gerechtfertigt[6] : $p_E \approx 1 - 2^{-4}$. Bei »doppelter Genauigkeit« bilden sich Verfälschungen vorzugsweise auf die niederwertigen Bits ab. Maskierungswahrscheinlichkeit: $2^{-4} > 1 - p_E \gg 2^{-8}$.

[6] Kein Nachweis für vertauschte Reihenfolge.

Prüfkennzeichenbildung mit LFSR



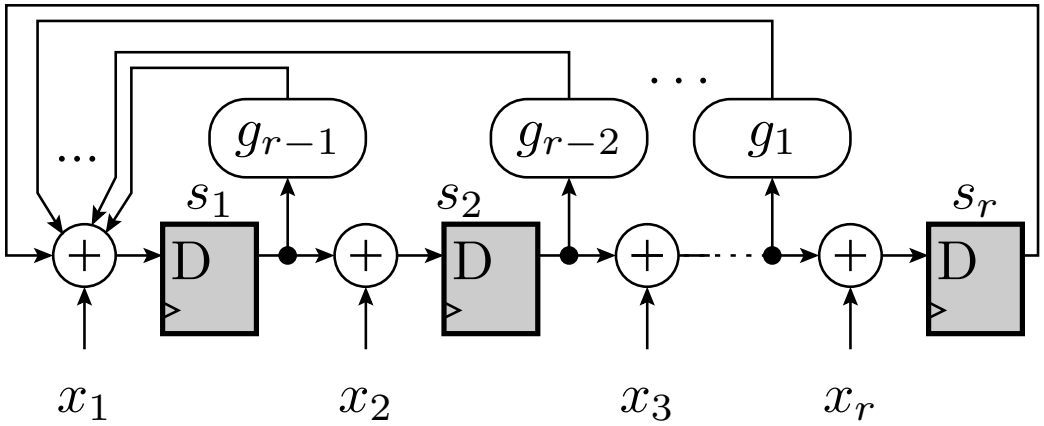
Initialwert	0	0	0	1
Schritt 1:	0	0	1	0
Schritt 2:	0	1	1	1
Schritt 3:	1	1	0	**1**
Schritt 4:	1	0	**0**	1
Schritt 5:	0	**0**	**0**	0
Schritt 6:	**0**	**0**	**0**	1
Schritt 7:	**0**	**0**	1	



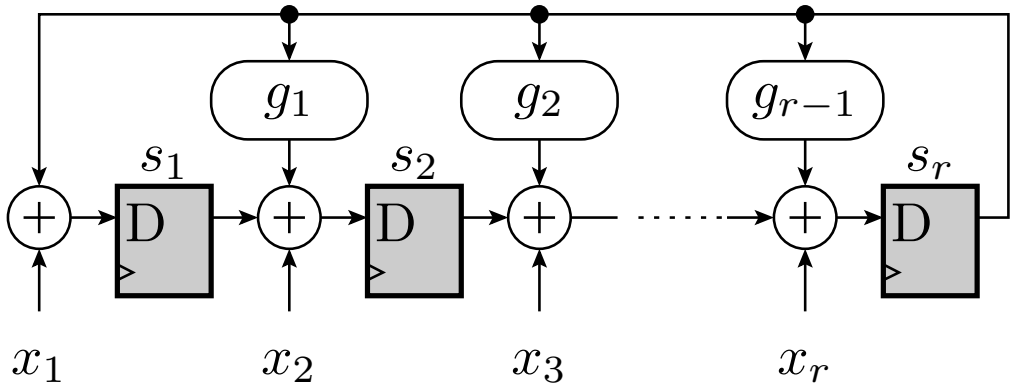
Das Prüfkennzeichen wird wie bei der CRC-Decodierung mit einem linear rückgekoppeltes Schieberegister (LSFR) gebildet. Im Beispiel hat das LSFR im Gegensatz zur Polynomdivision Folie 8 zentrale Rückführung. Abbildung auch pseudo-zufällig.

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[7]. 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, $g_{26} = g_{23} = g_{22} = g_{16} = g_{12} = g_{11} = g_{10} = g_8 = g_7 = g_5 = g_4 = g_2 = g_1 = 1$

[7] Zyklusstruktur bei Einspeisung einer Folge aus nur Nullen.

Experiment Fehlererkennungssicherheit von LFSR

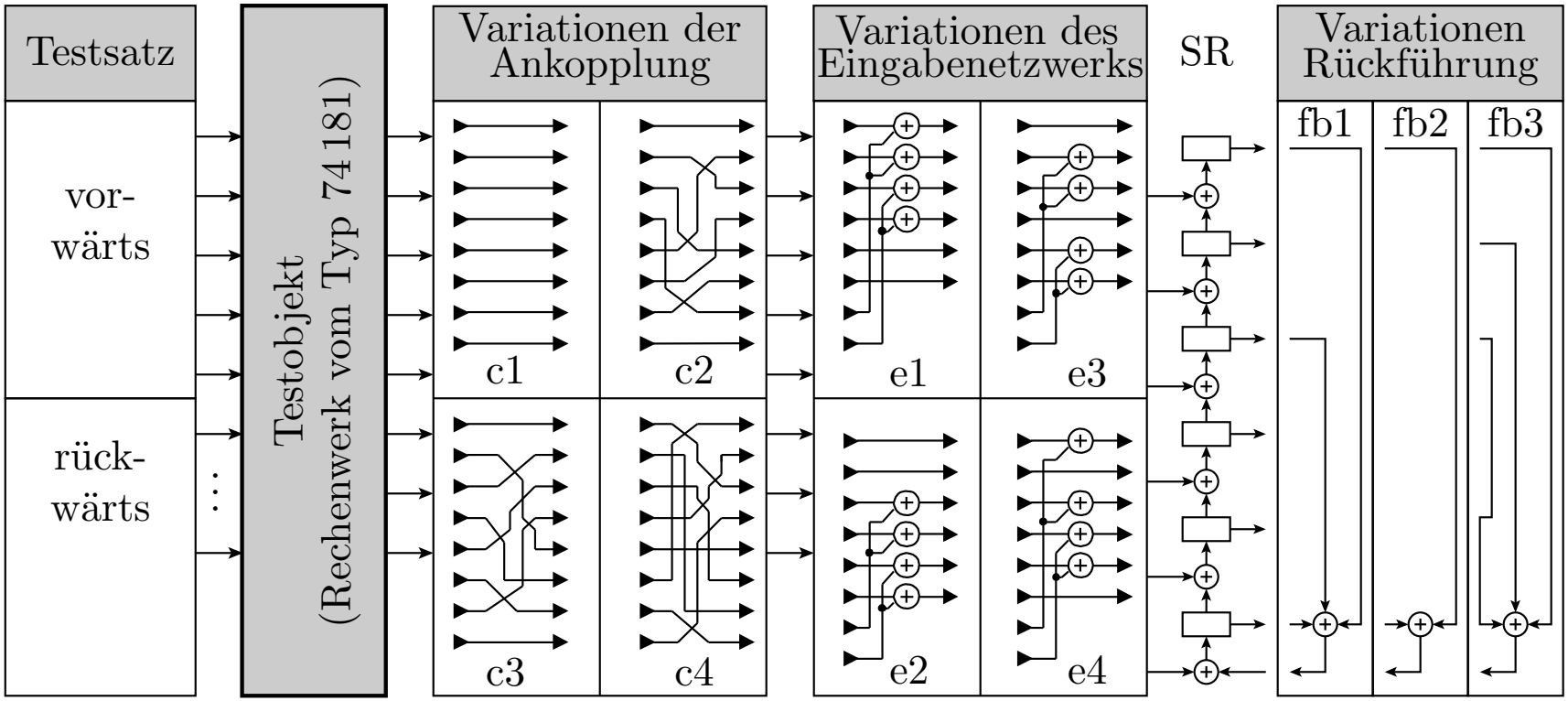
Es ist schwer zu glauben, dass

- mit r-Bit Prüfkennzeichen beliebige Verfälschung mit einer Wahrscheinlichkeit $p_{\mathrm{E}} = 1 - 2^{-r}$ erkannt werden und
- die Schaltungsstruktur, die Rückführung etc. keinen 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 $p_{\mathrm{E}} = 1 - 2^{-6} = 98{,}44\%$ erkenn- und mit einer Wahrscheinlichkeit $p_{\mathrm{F}} = 2^{-6} = 1{,}36\%$ nicht erkennbar sein müsste. Definition einer Zufallsgröße X_i zum Zählen der nicht erkennbaren Fehler:

$$\begin{aligned} P(X_i = 0) &= 1 - 2^{-6} \text{ Fehler } i \text{ nachweisbar} \\ P(X_i = 1) &= 2^{-6} \text{ Fehler } i \text{ nicht nachweisbar} \end{aligned}$$

Wenn die Theorie stimmt, müsste die Anzahl der maskierten Fehler

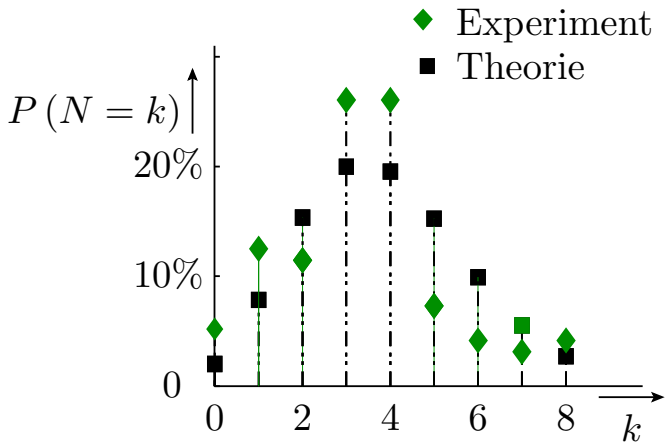
$$N = \sum_{i=1}^{250} X_i$$

binomialverteilt sein:

$$P(N = k) = \begin{pmatrix} N \\ k \end{pmatrix} \cdot 2^{-r \cdot k} \cdot (1 - 2^r)^{N-k}$$

Anzahl der maskierten Fehler

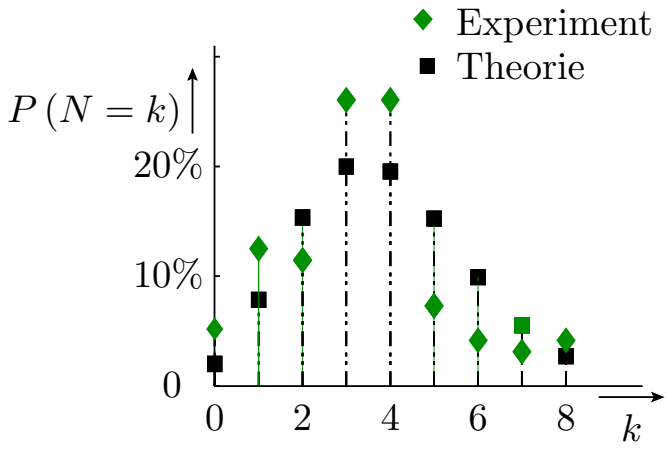
		e1				e2				e3				e4			
		c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4
vor-wärts	fb1	3	4	1	2	3	4	3	3	4	2	4	3	4	3	4	6
	fb2	3	4	1	7	2	2	1	4	2	1	1	3	2	5	3	7
	fb3	5	2	2	8	4	5	3	4	3	6	3	7	5	3	3	4
rück-wärts	fb1	6	4	4	2	3	4	3	4	3	4	3	4	4	8	4	5
	fb2	2	0	0	1	4	1	4	1	0	0	0	1	1	1	4	1
	fb3	2	4	3	4	4	8	5	8	3	3	3	6	3	3	4	3



Die experimentell bestimmte Maskierungswahrscheinlichkeit scheint sogar unwesentlich kleiner als 2^{-6} zu sein.

Anzahl der maskierten Fehler

		e1				e2				e3				e4			
		c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4
vor-wärts	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
rück-wärts	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 im Experiment kaum erkennbaren Einfluss auf die Maskierungswahrscheinlichkeit.

Es gibt in der Literatur gegenteilige Behauptungen, z.B.

- [GrWil...] Primitive Rückführungen seinen viel besser als andere. Mysteriöse Beweisführung, unglaubhaftes Ergebnis, aber »Best Paper Arward auf der Internat. Testkonferenz«.
- [Jar..] Ein Experiment mit einem 4-Bit-Signaturregister, in dem von 4 Haftfehlern keiner erkannt wird. Beispiel konstruiert oder zufällig gefunden?

Zusammenfassung

Datensicherung mit fehlererkennenden Codes / Prüfkennzeichen:

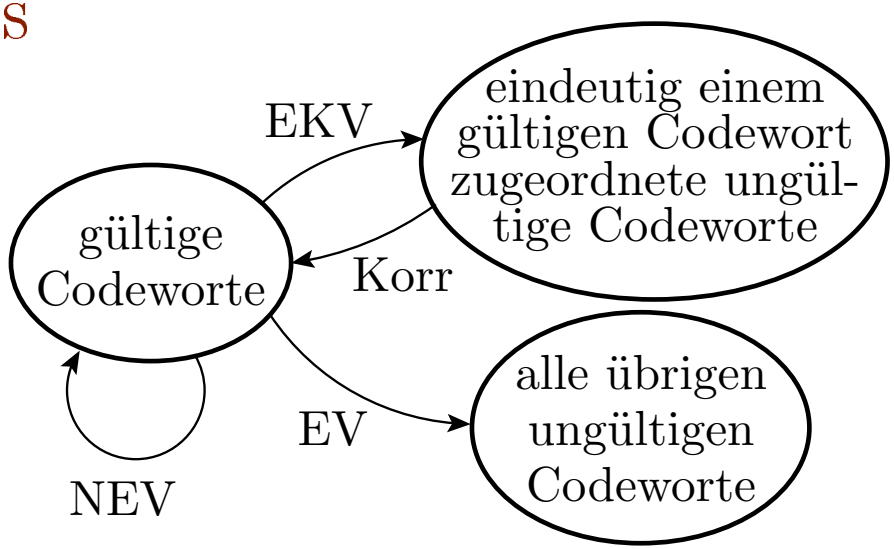
- Geringer Berechnungsaufwand.
- r-Bit-zusätzlich gespeicherte / übertragenen Information für eine Datenobjekt beliebiger Größe $\Rightarrow$ Maskierungswahrscheinlichkeit 2^{-r}. Immer vernachlässigbar klein wählbar.
- exzellente Erkennungswahrscheinlichkeit, geringer Aufwand.

Dateien, Nachrichten etc. werden fast immer mit einem Prüfkennzeichen geschützt. Fehlererkennende Codes sind weniger gebräuchlich.

Fehlerkorr. Codes

Fehlerkorrigierende Codes

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:

$$2^{\text{Anz(Bit)}} \geq \text{Anz(CWG)} + \text{Anz(CWG)} \cdot \text{Anz(CWK/CWG)}$$

(Anz(CWG) – Anzahl gültige Codeworte; Anz(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:

$$\text{Anz(CWK/CWG)} = \text{Anz(Bit)}$$

Mindestbitanzahl:

$$2^{\text{Anz(Bit)}} \geq \text{Anz(CWG)} + \text{Anz(CWG)} \cdot \text{Anz(Bit)}$$

Für $\text{Anz(CWG)} = 256\,\text{Bit}$:

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

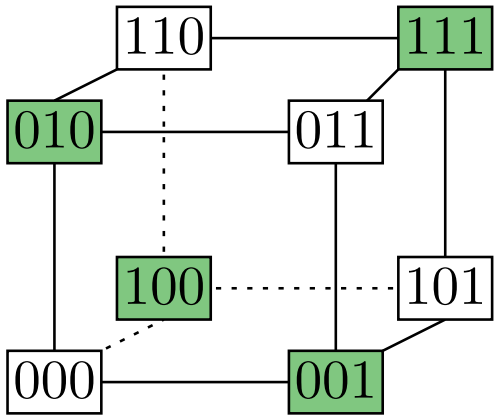
Probe:

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

Hamming-Codes

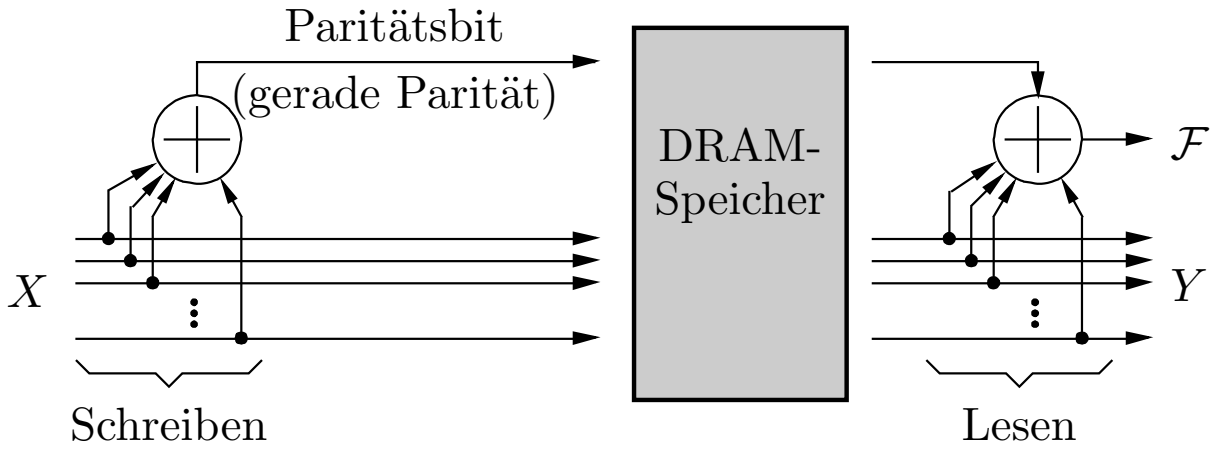
Hamming-Distanz

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)



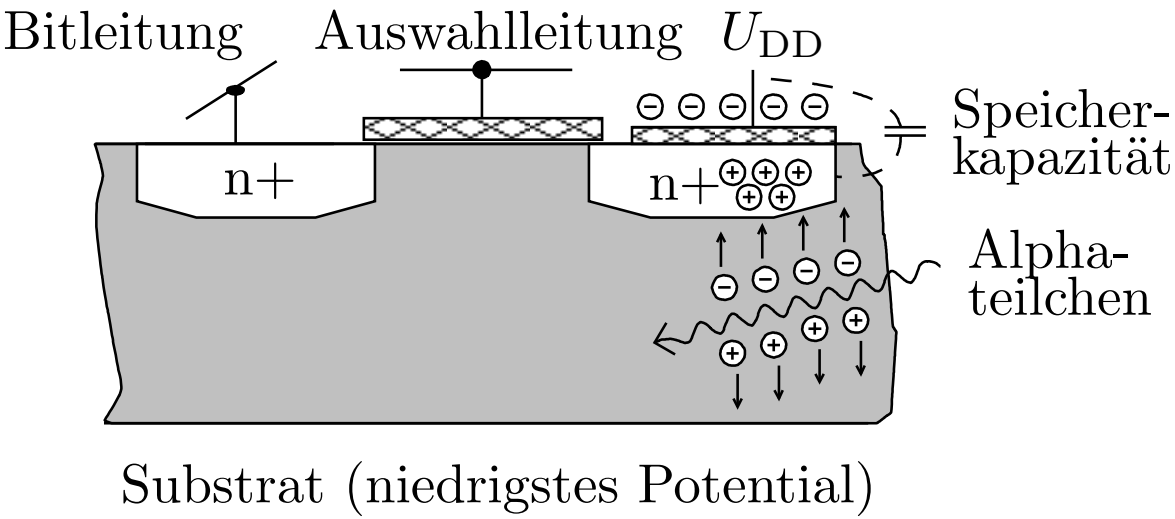
Einzelprüfbit, modulo-2 Summe (EXOR-Verknüpfung):

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

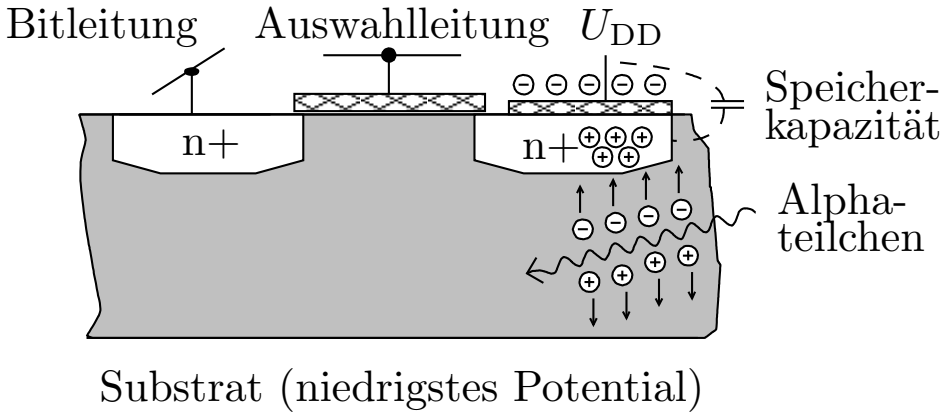
bei gerader Anzahl von Einsen »0« sonst »1«.

- Erkennt jede ungeradzahlige Anzahl von Bitverfälschungen.
- Wenn bei Verfälschungen 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 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 Speichermatritzen) $\Rightarrow$ Einzelbitverfälschung je gelesenes Datenwort.

- 100%iger Nachweis durch Paritätskontrolle.

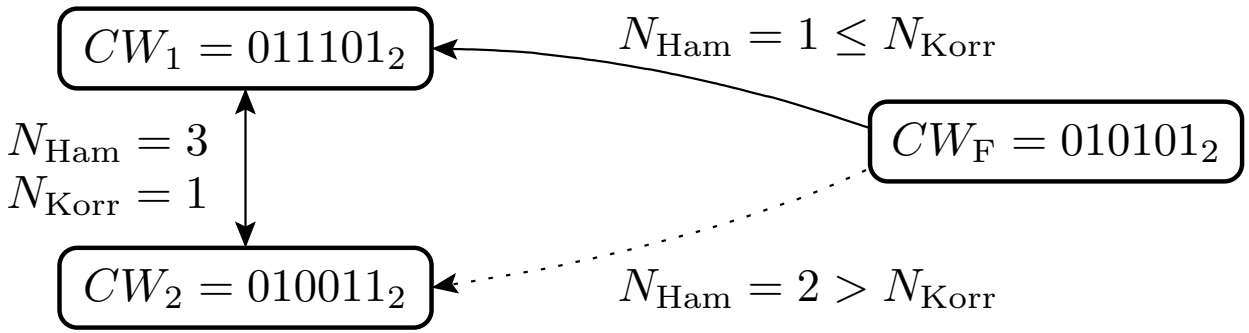
Kreuzparität (Fehlerkorrigierender Paritätscode)

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).

$$
\begin{array}{ccccccc}
 & & & \cdots & & & \\
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & \mathbf{1} \\
0 & 0 & 0 & 1 & 0 & 1 & \mathbf{0} \\
1 & 0 & 0 & 1 & 1 & 1 & \mathbf{0} \\
1 & 1 & 0 & 0 & 1 & 1 & \mathbf{0} \\
0 & 1 & 1 & 0 & 1 & 0 & \mathbf{1} \\
1 & 0 & 0 & 0 & 0 & 1 & \mathbf{0} \\
\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\
0 & 1 & 1 & 0 & 1 & 0 & 1 \\
 & & & \cdots & & &
\end{array}
$$

1-Bit fehlerkorrigierende Hamming-Codes

Ab einem Hamming-Abstand ≥ 3 ist jeder 1-Bit-Verfälschung eindeutig ein gültiges Codewort zugeordnet.



CW_i	gültige Codewörter
CW_{F}	verfälschtes Codewort
N_{Ham}	Hamming-Abstand

Korrektor durch Ersatz des verfälschten Codeworts durch das mit Hamming-Distanz eins. Bei Hamming-Distanz $= 3$ werden 2-Bit-Verfälschungen bei der Korrektur falschen gültigen Codeworten zugeordnet.

Konstruktion eines 1-Bit fehlerkorrigierendes Codes

Zusammensetzen des Gesamtcodeworts aus

- einem Datenwort mit minimaler Bitanzahl

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

und einem Prüfkennzeichen der Größe

$$r = \text{Anz}(\text{Bit}) - w$$

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	...
Prüfzeichendifferenz $\Delta\mathbf{q}$	...001	...010	...011	...100	0101	...

verfälschtes Bit	1	2	3	4	5	...
Prüfzeichendifferenz $\Delta\mathbf{q}$	...001	...010	...011	...100	0101	...

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

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

Ein Bit jeder Summe muss das Prüfbit sein. 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

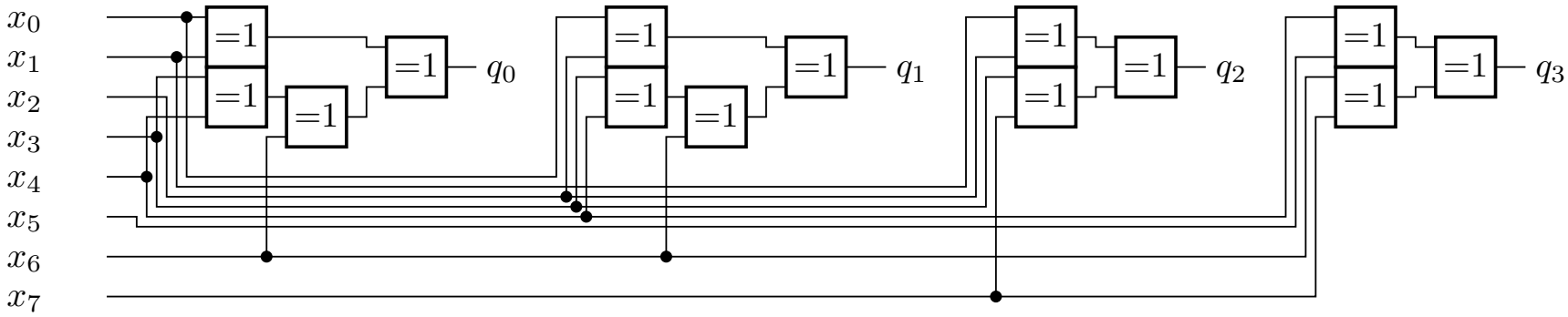
Für die erste Summe gilt:

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

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:



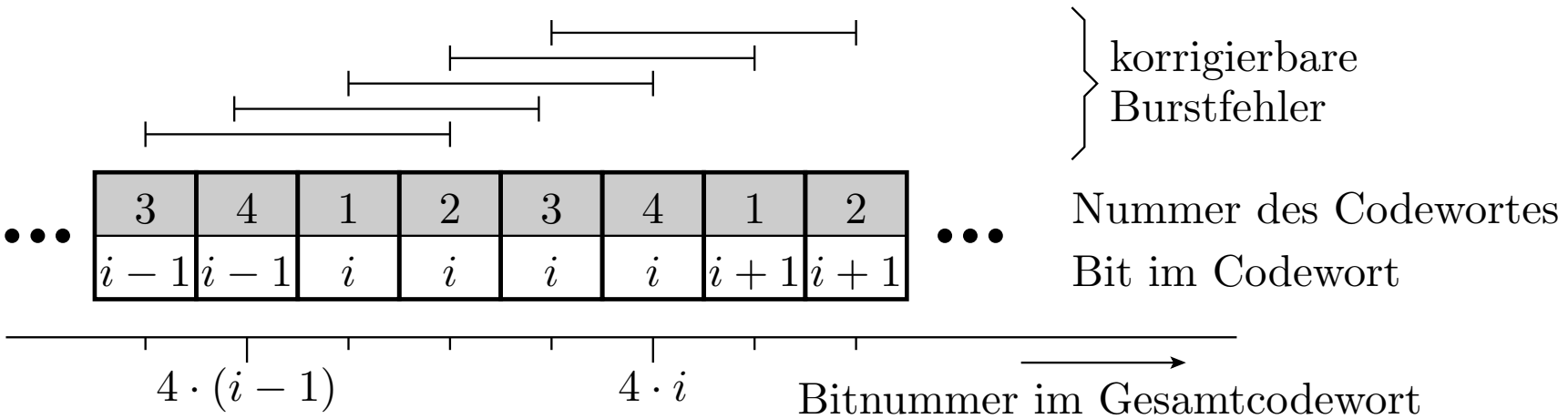
Korrekturschaltung besteht aus demselben Coder zur Bildung von $\mathbf{q} = q_3 \ldots q_0$ der bitweise EXOR-Verknüpfung des empfangen und des im Empfänger gebildeten Prüfzeichens. Invertierung des verfälschten Bits (als Case-Anweisung in VHDL):

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

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 aufeinanderfolgenden 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:



RAID Systeme

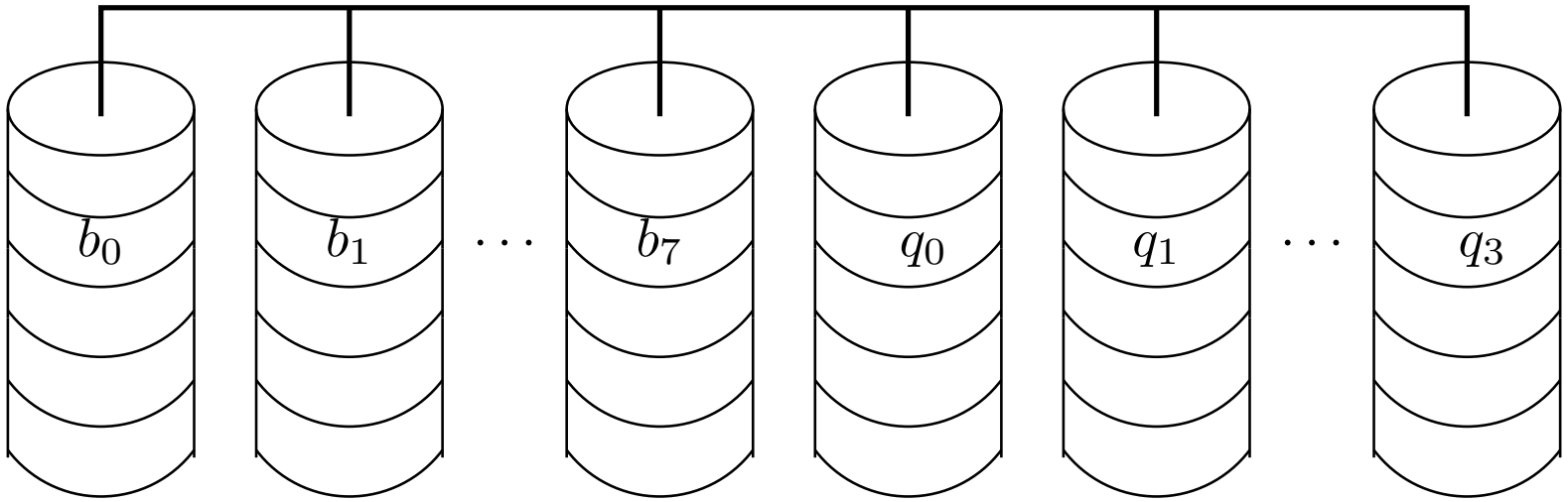
RAID, RAID Level 1

RAID – Redundant Array of Independent Disks. 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

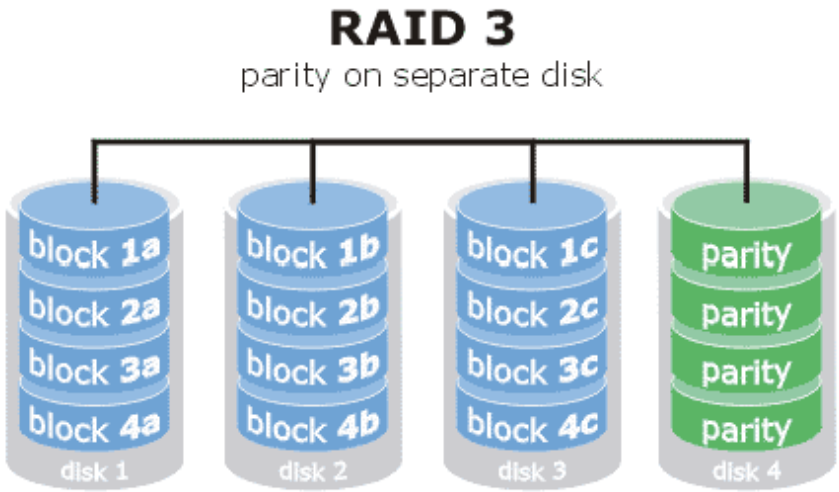
Bei RAID Level 2 werden die Daten in einem 1-Bit fehlerkorrigierenden Hamming-Code gespeichert, und zwar jedes der w Daten- und der r Kontrollbits auf einer eigenen Platte, z.B. $w = 8$ Datenbit- und $r = 4$ Kontrollbitplatten. Im Vergleich zu RAID 1 werden statt der doppelten Plattenanzahl nur 50% mehr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



Gilt als aufwändig und ungebräuchlich.

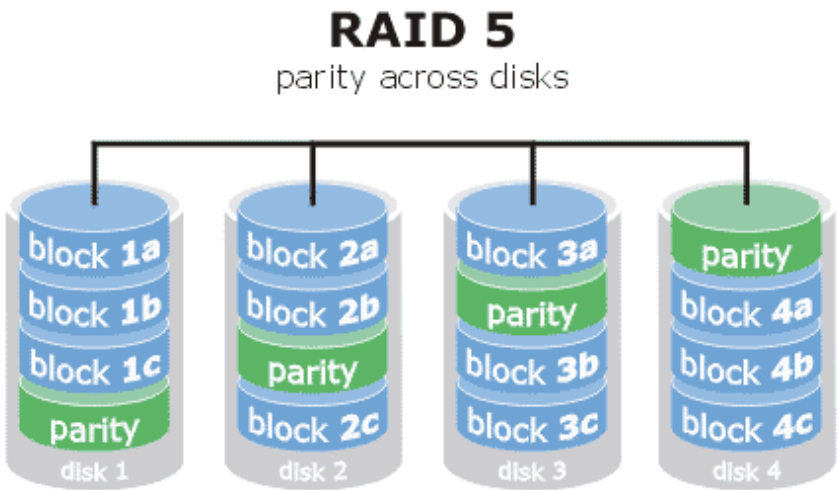
RAID Level 3

Auf einer Extra-Platte wird bitweise die Querparitätsbit der anderen Platten gebildet. Zusätzlich wird auf jeder Platte die Längsparitätsbit (oder ein 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 genutztes RAID-System.



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 versehentlichem Löschen, das erst nach Stunden oder Wochen bemerkt wird.

Aufgaben

Aufgabe 3.1: Arithmetischer Code

1. Bilden Sie für den Bitvektor

   $$1100100010000011101$$

   das fehlererkennende Codewort durch Multiplikation seines Wertes als vorzeichenfrei ganze Binärzahl mit der Primzahl 10313 (Bestimmung des Dezimalwerts, Multiplikation und Konvertierung des Produkts in einen Binärvektor).
2. Mit welcher Wahrscheinlichkeit werden mit dem gewählten fehlererkennenden Code Datenverfälschungen erkannt?
3. Werden mit dem gewählten Code Verfälschung erkannt, die die Bitstellen 3 bis 14 invertieren?

Hinweis: Der Code ist linear, so dass das Erkennen eines verfälschten Codeworts nur von der Differenz und nicht vom codierten Wert abhängt.

Aufgabe 3.2: Prüfsummen

Bilden Sie für die Bytefolge

```
0x13, 0xF2, 0x33, 0xE6, 0x8A, 0x3D, 0x30, 0x51
```

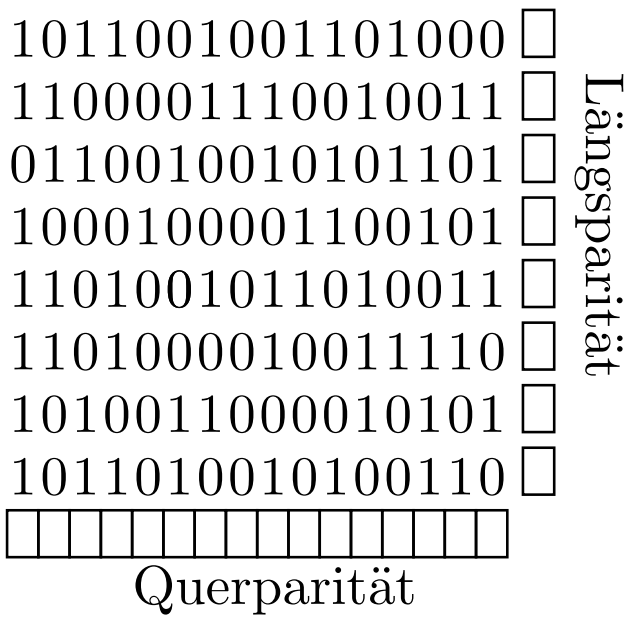
die Prüfsumme:

1. durch byteweise Aufsummieren unter Vernachlässigung der Überträge und
2. durch bitweise EXOR-Verknüpfung der Bytes.
3. Welche der beiden Prüfsummen erkennt, dass die nachfolgenden Datenfolgen verfälscht sind?

```
F1: 0x13, 0x33, 0xF2, 0xE6, 0x8A, 0x3D, 0x30, 0x51
F2: 0x13, 0xF2, 0x35, 0xE2, 0x8A, 0x3D, 0x30, 0x51
```

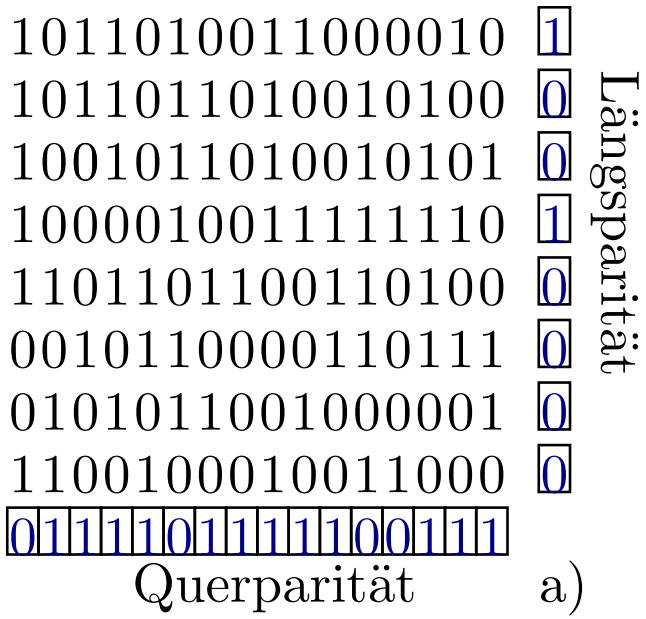
Aufgabe 3.3: Kreuzparität, Berechnung

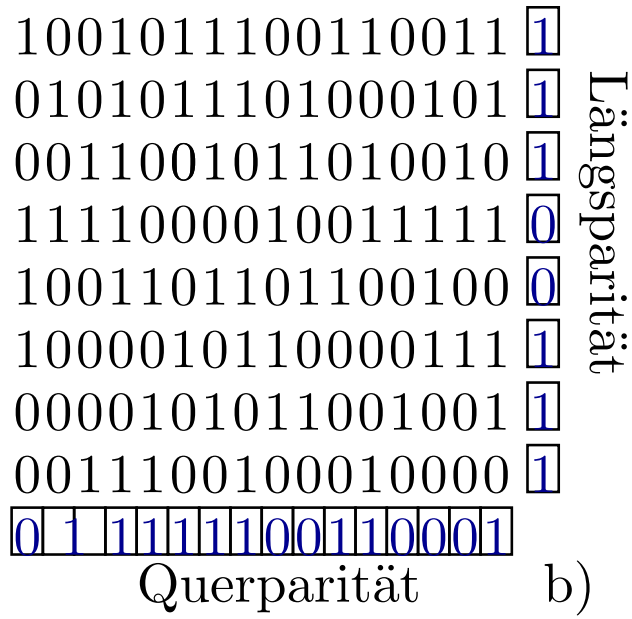
Ergänzen Sie die Bitwerte für die Längs- und Querparität so, dass die Anzahl der Einsen in jeder Zeile und Spalte incl. Paritätsbit gerade ist.



Aufgabe 3.4: Kreuzparität, Korrektur

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.





Aufgabe 3.5: Hamming-Distanz

1. Welche Hamming-Distanz ist erforderlich, damit alle 1-Bit-Fehler korrigiert und alle 2-Bit-Fehler erkannt werden?
2. Wie groß ist die minimale Hamming-Distanz zwischen den acht Codeworten 0000000, 1110001, 1001101, 0111100, 0101011, 1011010, 1100110, 0010111?

Aufgabe 3.6: Hamming-Code

1. Bilden Sie für den ab Folie 44 entwickelten (8,12)-Hamming-Code die Codeworte für die darzustellenden Werte: 0x73, 0x1D und 0xD6.
2. Handelt es sich bei den Codeworten 0xA24, 0x5D6 und 0x41 um zulässige Codeworte, Codeworte mit korrigierbaren oder Codeworte mit erkenn- aber nicht korrigierbaren Verfälschungen? Wenn sie korrigierbar sind, wie lauten die zugeordneten korrekten Werte?

Hinweise: Berechnung der Kontrollstellen:

$$
\begin{aligned}
q_0 &= x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6 \\
q_1 &= x_0 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_6 \\
q_2 &= x_1 \oplus x_2 \oplus x_3 \oplus x_7 \\
q_3 &= x_4 \oplus x_5 \oplus x_6 \oplus x_7
\end{aligned}
$$

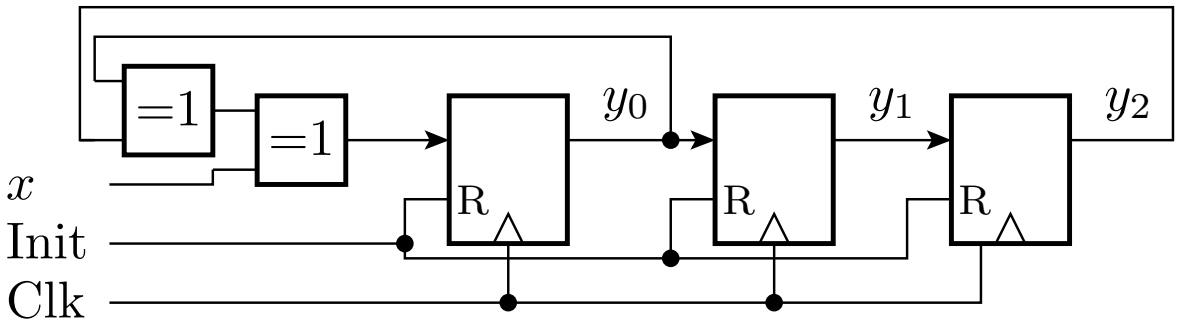
Bitzuordnung mit Beispiel:

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

- Gesamtes Codewort: 0x9B8 (0b1001 1011 0000)
- Datenwert: 0x96 (0b1001 0110)
- übertragene Prüfbits: 0xD (0b1100)
- berechnete Prüfbits: 0x40 (0b0100)
- Differenz der Prüfbits: 0x80 (0b1000)
- verfälschtes Bit: $b_8 = q_3$.

Aufgabe 3.7: Prüfkennzeichen mit LFSR

Gegeben ist folgendes linear rückgekoppelte Schieberegister:



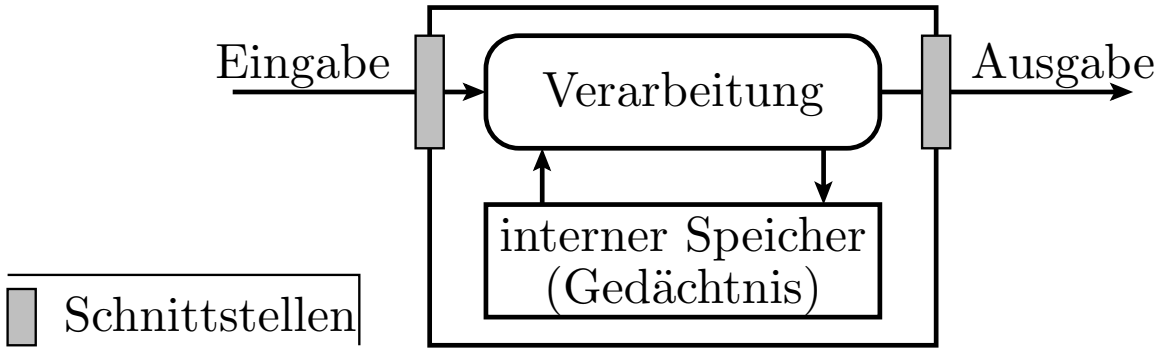
	x	y_2	y_1	y_0
0	1	0	0	0
1	0			
2	0			
3	1			
4	1			
5	0			
6	0			
7	1			
8	0			
9	1			
10	1			
11	1			
12	1			
13	0			
14	1			
15	0			
PKZ:				

1. Wie hoch ist Fehlererkennungswahrscheinlichkeit?
2. Welches Prüfkennzeichen $\mathbf{y} = y_2 y_1 y_0$ hat die Datenfolge »1001100101111010« bei Abbildung beginnend mit dem höchstwertigen Bit. Startwert 000. Füllen Sie dazu die Tabelle in der Abbildung aus.

Formatkontrollen

Service-Modell

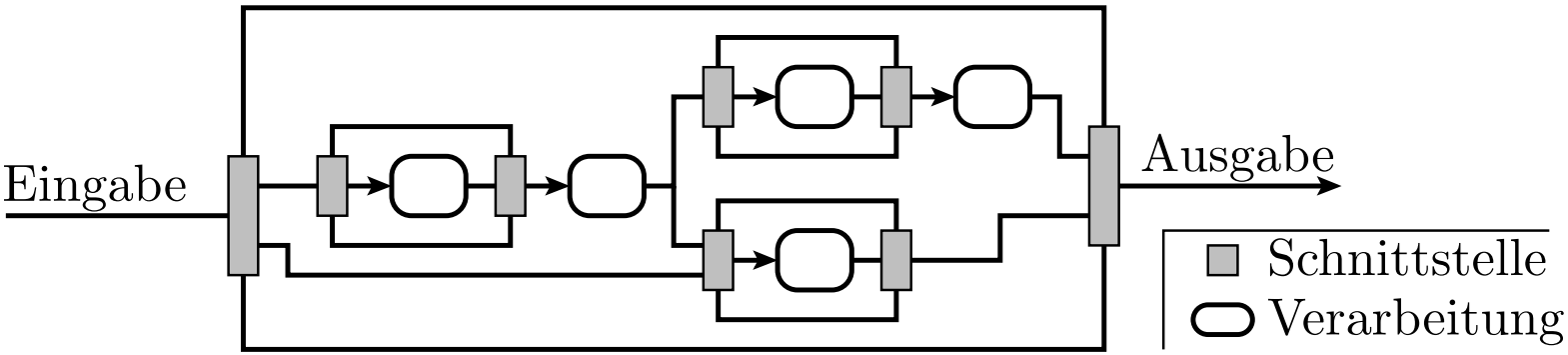
Laut F1, Abschn. 1.2 ist ein Service ein Berechnungsablauf, der aus Eingaben Ausgaben bildet. Die Ein- und Ausgaben sind ganz allgemein bedatete Objekte mit einem auf die Art des Services abgestimmten Format. Ein Service kann, aber muss kein Gedächtnis haben.



Mit diesem Modell lassen sich die Verlässlichkeitsaspekte der Hardware vom schaltenden Transistor bis zum Rechnersystem, der Software vom Maschinenbefehl bis zum Programmsystem und kompletter Geräte aus Hard- und Software beschreiben.

Schnittstellen und Kontrollen

Komplexe Service-Leistungen nutzen bei ihrer Abarbeitung Teilservice-Leitungen. 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 sichersten Formatkontrollen bieten fehlererkennende Codes und Prüfkennzeichen mit $p_{\mathrm{E}} = 1 - 2^{-r}$ (r – Anzahl der zusätzlich zu speichernden oder zu übergebenden Bits, vergl. Gl. 16 und 24).

Datenpaktformate mit Prüfkennzeichen

Ethernet-Paket:

Sicherungs-schicht			MAC-Empfänger	MAC-Absender	Protokoll-typ	Nutzlast max. 1500 Bytes	Prüfkennz. CRC	
Bitübertra-gungsschicht	Präambel	Start-byte						Lücke zum nächsten Packet
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[8]:

ID	Nachrichtennummer	CRC	Prüfkennzeichen
DLC	Datenlängencode	EOF	Ende des Datenrahmens

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

[8] Lokaler Bus zur Vernetzung von z.B. Steuergeräten in Autos.

Fehlererkennende Codes und Prüfkennzeichen habe 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[9].

[9] Ein Signal ist ein zeitlicher Werteverlauf.

Syntaxtest

Formale Sprachen

Eine formale Sprache definiert zulässige Worte durch Textersetzungsregeln:

- Terminalsymbole: Zeichen- oder Zeichenkettenkonstanten
- Nichtterminalsymbole: zu ersetzende Symbole
- Beschreibung von Ersetzungsregeln:
  - ...|... – darf der rechte oder der linke Ausdruck sein
  - [...] – der Ausdruck ... darf 0 oder einmal eingesetzt werden
  - {...} – der Ausdruck ... darf beliebig oft aufeinander folgen.

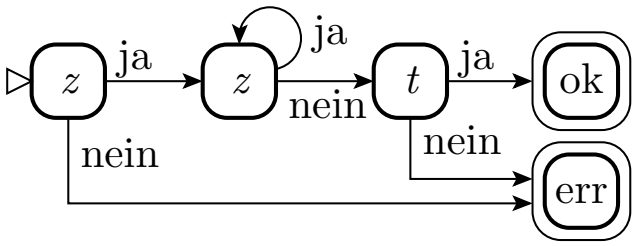
Beispiel:

- Anweisung => Zuweisung | if_anweisung | ...
- Zuweisung => Variable = Ausdruck ;
- Ausdruck => uk_Ausdruck | (uk_Ausdruck)
- uk_Ausdruck => Konst. | Variab. | Ausdruck OP Ausdruck
- ...

Spracherkennender Automat

Beispiel: Syntaxtest für die Beschreibung einer positiven ganzen Zahl. Sprachbeschreibung: Eine Ziffer gefolgt von beliebig vielen Ziffern und einem Trennzeichen:

$$Zahl \Rightarrow z\{z\}t$$



▷	Startzustand
z	Ziffer erwartet
t	Trennzeichen erwartet
$\surd$	zulässiges Zeichen
nein	kein zulässiges Zeichen
ok	Wort erfolgreich abgeräumt
err	Eingabefehler

Band . . . 1 3 5 ␣ 5 8 k . . .	erwartet	Wert
Start 1	$z\surd$	1
	$z\surd$	13
	$z\surd$	135
	z (nein)	
	$t\surd$	135
Endzustand	ok	135
Start 2	$z\surd$	5
	$z\surd$	58
	z (nein)	
	t (nein)	
Endzustand	err	ungültig

Mit den einfachen Ersetzungsregeln »darf sein«, »darf ein- oder n-mal vorkommen« lassen sich viele Datenformate und auch Programme, die ja letztendlich auch nur Daten sind, 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 die Hälfte oder auch deutlich mehr der der Fehler durch Menschen bei der Eingabe und 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 sind 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
  - Aufzählungtypen.
- 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.

VHDL – Sprache mit strenger Typkontrolle

VHDL definiert keine Datentypen, sondern Beschreibungsmittel, um Datentypen zu definieren. Beispiel seien die Zahlentypen.

- Ein Zahlentyp ist ein zusammenhängender Zahlenbereich:

  ```
  type Zahlentyp is range Bereich;
  ```

- Bereiche können auf- oder absteigend geordnet sein:

  ```
  type tWochentag is range 1 to 7;
  type tBitnummer is range 3 downto 0;
  ```

  $\text{Wochentag} \in \{1, 2, 3, 4, 5, 6, 7\}$; $\text{Bitnummer} \in \{3, 2, 1, 0\}$

- ganzzahlige Bereichsgrenzen $\Rightarrow$ diskreter Zahlentyp
- Bereichsgrenzen mit Dezimalpunkt $\Rightarrow$ reeller Zahlentyp

  ```
  type tWahrscheinlichkeit is range 0.0 to 1.0;
  ```

Kontrollmöglichkeiten

- Kontrollen bei einer Variablenzuweisung

  Variablenname := Ausdruck;

  Typenübereinstimmung. Zulässiger Ausdruckswert.

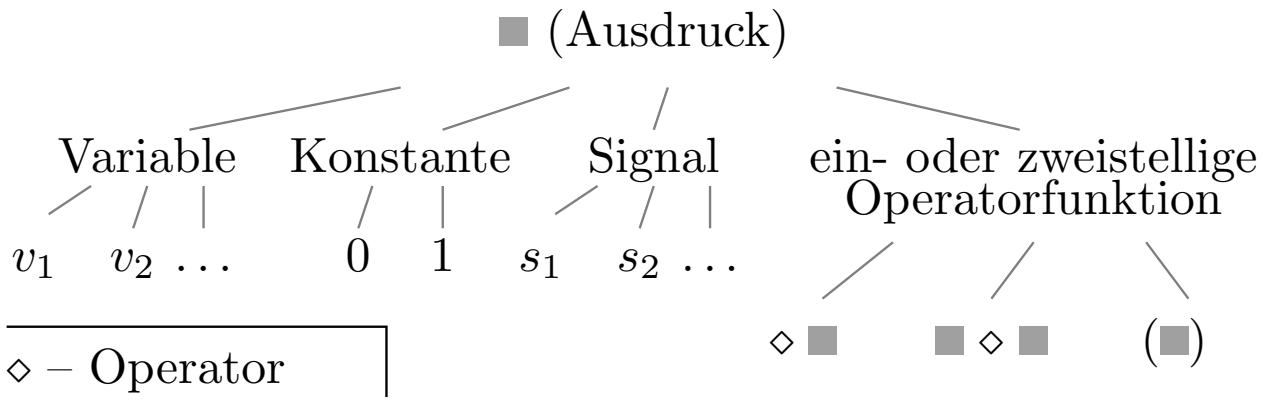
```
variable Wochentag: tWochentag;
variable Bitnummer: tBitnummer;
...
Wochentag := Bitnummer;-- Typ unzulässig
Wochentag := 9;        -- Wert unzulässig
```

- Kontrollen bei Signalzuweisungen

  Signalname <= W [after t_d]{, W after t_d};

  Gleicher Typ des Ausdruck W und des Signal. Ausdruck t_d muss Typ TIME haben; $t_\mathrm{d} \geq 0$.

- Kontrollen in Ausdrücken: Ein Ausdruck ist eine rekursive Beschreibung einer Funktion mit ein- und zweistelligen Operatorfunktionen.



- Arithmetische Operatoren (+, -, ...) verknüpfen identische Zahlentypen[10]. Ergebnis Operandentyp.

[10] Ausnahme »**« (Potenzbildung), die einen ganzzahligen Exponenten als zweiten Operanden verlangt. Ergebnis hat den Typ des ersten Operanden.

- Logische Operatoren (and, or, ...) verknüpfen identische Bit- oder Bitvektortypen. Ergebnis Operandentyp.
- Vergleichsoperatoren: Verknüpfung identischer Typen. Ergebnis Typ BOOLEAN.
- ...
- Regelabweichungen erfordern Typenkonvertierung (Typecast) oder Überladen der Operatorfunktion mit anderen Typenkombinationen.

- Kontrollen bei Zuordnungen (Schnittstellensignale, Aufrufparameter, Vektorelemente, ...)

  BS => *BZ*{, *BS* => *BZ*} oder *ZB*{, *ZB*}

  Typgleichheit zwischen zugeordneten Objekt *BZ* und den Objekten denen etwas zugeordnet wird

```
type tAepfel is range 1 to 10;
type tBirnen is range 1 to 10;
variable apfel_1, apfel_2: tAepfel := 3;
variable birne: tBirnen := 2;
```

Zulässig oder fehlerhaft?

```
A1: apfel_1 := apfel_2 + 11;
A2: birne := apfel_1 * apfel_2;
A3: birne := birne + apfel_1;
```

⇒WEB-Projekt: 3

P3.2/TestApfel.vhdl

A1: apfel_2 + 11 ↦ tAepfel; typengleich mit apfel_1; Ergebnis 12, nicht darstellbar

A2: apfel_1 * apfel_2 ↦ tAepfel; nicht an Typ tBirne zuweisbar

A3: Addition von Summanden mit den Typen tBirne und tAepfel unzulässig

```
type tWahrsch is range 0.0 to 1.0;
variable w: tWahrsch;
variable r: REAL; -- REAL ist ein vordefinierter Typ
...
... w * 2.0 ↦ tWahrsch
... w * 2 ↦ Übersetzungsfehler
... 3 + 5 ↦ universeller ganzzahliger Typ
... 3 * 2.5 ↦ Übersetzungsfehler
... w > 0.0 ↦ BOOLEAN
... w = 0 ↦ Übersetzungsfehler
... w * r ↦ Übersetzungsfehler
```

Beseitigung der Übersetzungsfehler durch Typkonvertierung:

```
... w * tWahrsch(2) ↦ tWahrsch
... 3 * INTERGER(2.5)↦ INTERGER
... INTERGER(w) = 0 ↦ BOOLEAN
... w * tWahrsch(r) ↦ tWahrsch
```

Die strenge Typenprüfung erkennt nicht nur, sondern vermeidet auch Fehler, indem sie den Programmierer zwingt, genauer über die beabsichtigte Zielfunktion nachzudenken.

Untertypen

Ableitung vom Basistyp durch Wertebereichsbeschränkung, z.B.:

```
subtype tWochenende is tWochentag range 6 to 7;
```

- Ein Untertyp wird bei der Typenprüfung wie sein Basistyp behandelt und
- erbt alle zulässigen Operationen, Funktionen etc.,
- kann aber nur Werte seines Wertebereichs annehmen.

```
variable wt: tWochentag;
variable we: tWochenende;

...
A1: we := wt;
A2: wt := ((we + 4) mod 7) + 1;
```

A1: Typenprüfung o.k.; Simulationsfehler bei $wt \notin \{6, 7\}$
A2: Typenprüfung o.k.; $we = 6 \mapsto 4$; $we = 7 \mapsto 5$ o.k.

Allgemeine Anmerkungen zu Typenkontrollen

Wenn es die Sprache nicht unterstützt, können Typ- und Wertebereichskontrollen auch manuell einprogrammiert werden, z.B. in C für eine Struktur:

```
struct tFeld {
 uint8_t typ;   // Konstante für den Typ, z.B. 0x4D
 uint8_t len;   // Wertebereich 0 bis 100
 int16_t *feld; // Feld mit 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.

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 (≈ 120 zulässige und 2^{32} darstellbare Werte)
- Zeiger auf ein n Elemente großes Feld in einem 32-Bit-System; zulässig sind nur die Werte

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

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

Wertebereichskontrollen bestehen aus Fallunterscheidung und Fehlerbehandlung:

```
if alter<0 and 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 mit einer 32-Bit-Integer-Variable:

Fehlerannahmen:

- Verwechslung mit dem Alter einer anderen Person: meist falsch, immer zulässig, nie nachweisbar.
- Verwechselung mit der Hausnummer: meist falsch, oft zulässig[11], selten nachweisbar.
- Verwechselung mit anderem gespeicherten Wert. Kleine positive Datenwerte treten überproportional häufig auf.
- Grob geschätzt:

$$p_{\mathrm{E}} < 50\% \ll 1 - \frac{120}{2^{32}}$$

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

[11] Hausnummern sind immer größer null und meist keiner 119.

Erhöhung der Erkennungswahrscheinlichkeit

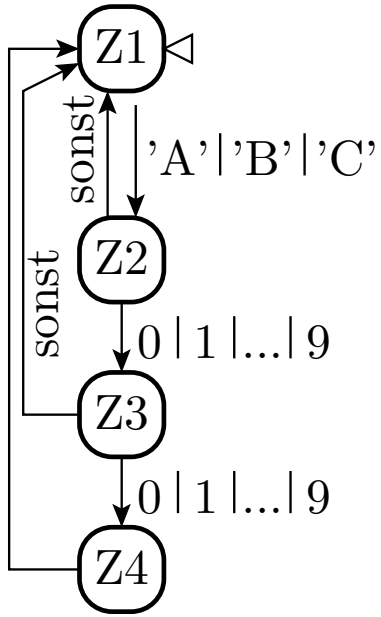
Zur Erhöhung der Erkennungswahrscheinlichkeit von Wertebereichskontrollen sind die zulässigen Werte ähnlich wie bei einem fehlererkennenden Code »zufällig« in der Menge der darstellbaren Werte zu verteilen:

- 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. ihre Darstellung mit zufälligen Werten innerhalb einer viel größeren Menge möglicher Werte. Statt der üblichen Zustandscodierung

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



kostet eine Zufallscodierung der Form

```
#define EmpfAut_Z1 0x35
#define EmpfAut_Z2 0x58
#define EmpfAut_Z2 0xD1
#define EmpfAut_Z2 0x7A
```

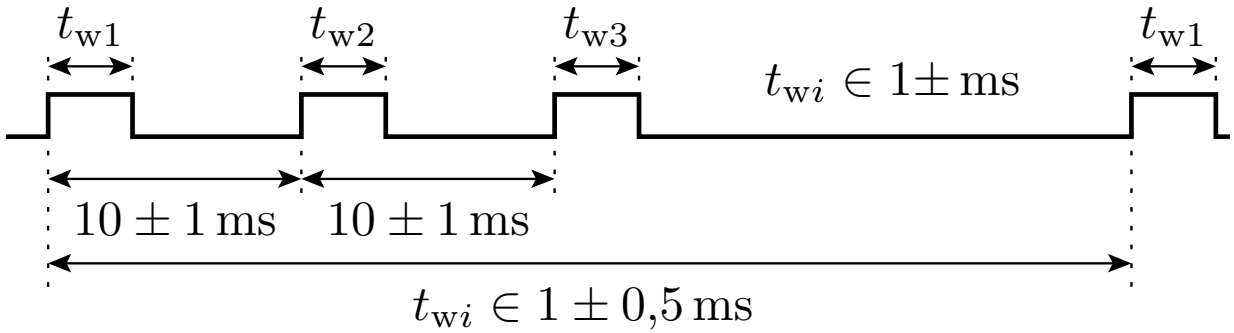
kaum Mehraufwand und vergrößert die Wahrscheinlichkeit, dass falsche Werte erkennbar sind, erheblich.

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: Amplituden, Zeiten, Frequenzen, ... Auch hier teilt sich die Darstellung ein in

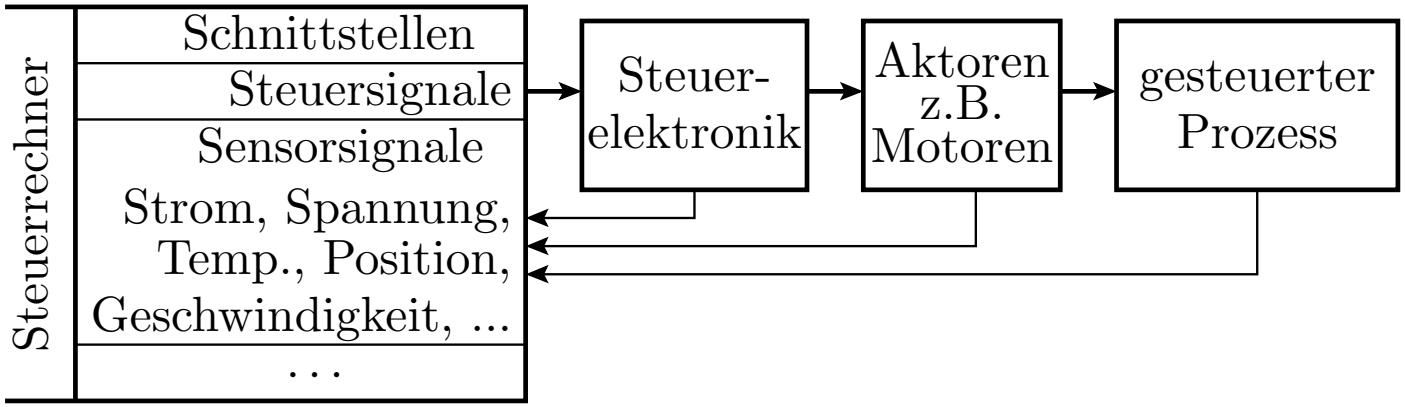
- Format (Gültigkeitsmerkmale) und
- Merkmale, die Werte repräsentieren.

Das nachfolgend 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.



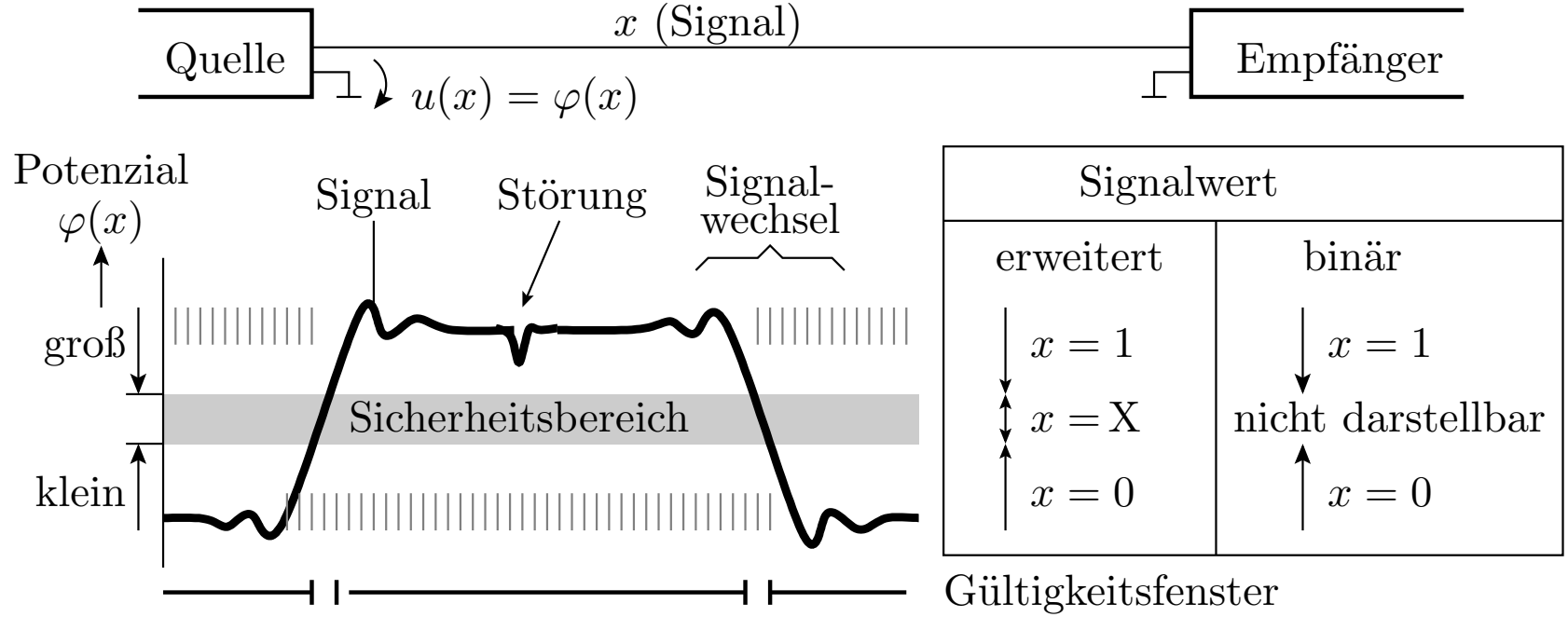
- Bereichsüberwachung.
- Überwachung von Pulsbreiten.

Die Überwachung komplexer Merkmale, z.B. der Stromsignaturen an Schrittmotoren auf Schrittfehler, erfolgt erst nach der Digitalisierung und ist z.T. sehr rechenintensiv[12].

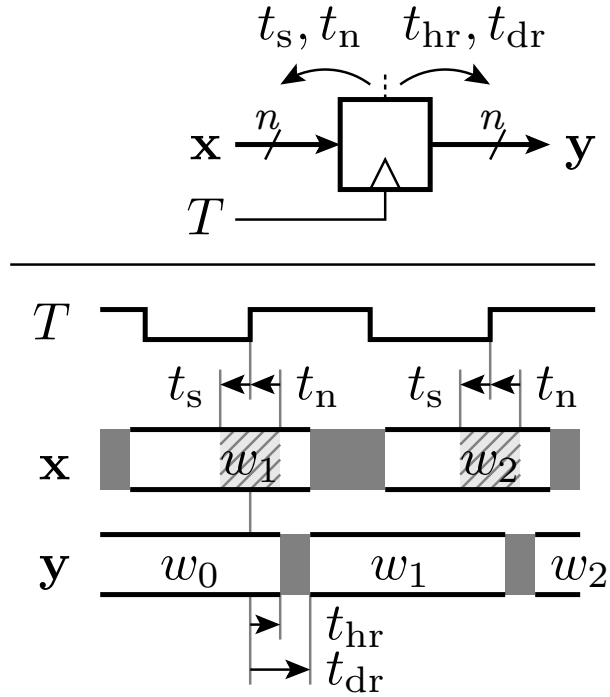
[12] Bei mechatronischen Systemen ist Übergang zwischen Zielfunktion und Kontrollfunktion zur Sicherung der Verlässlichkeit oft fließend.

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_{dr}	Verzögerungszeit
t_n	Nachhaltezeit		Abtastfenster
t_{hr}	Haltezeit		Wert ungültig

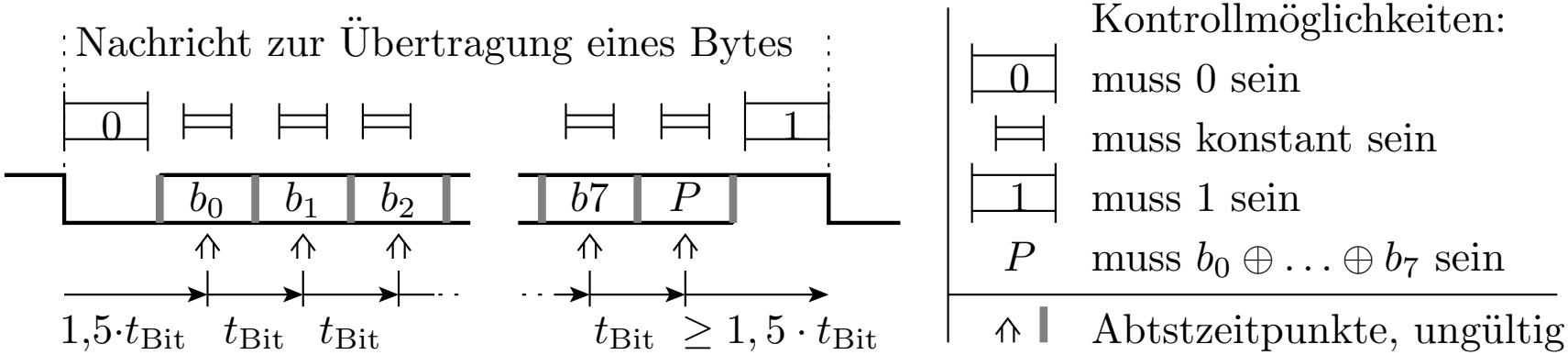
Bei analogen Signalen summieren sich die Verfälschungen durch Störungen, Rauschen, ... Die nachträgliche Trennung der Nutzsignale von überlagernden (zufälligen) Störsignalen ist schwierig, aufwändig, bzw. nur begrenzt möglich.

Deshalb erfolgt die Informationsverarbeitung mit Ausnahme der Werteaufnahme und Vorverstärkung überwiegend digital.

Serielles Protokoll – Schnittstelle mit Kontrollen

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).



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

Aufgaben

Aufgabe 3.8: Mikrorechner Ein- und Ausgabe

Ein einfaches Protokoll für die Kommunikation zwischen Mikrorechnern ist ein ASCII-Zeichen für den Nachrichtentyp

```
<NTyp> = 'U'|'V'|'W'
```

gefolgt von zwei Hexadezimalziffern:

```
<NTyp> = '0'|'1'|...|'a'|'b'|...|'f'
```

1. Beschreiben Sie dieses Eingabeformat als formale Sprache mit den Ersetzungsregeln auf Folie 70.
2. Entwerfen Sie in C eine Funktion

   ```
   uint8_t getMsg(uint8_t *MsgTyp, uint8_t *MsgWert)
   ```

   für den spracherkennenden Automaten, die mit

   ```
   uint8_t getchar();
   ```

   auf Zeichen von der seriellen Schnittstelle wartet, bei Empfang einer gültigen Nachricht mit »null«, Typ und Wert und sonst mit »eins« zurückkehrt.

Aufgabe 3.9: Eingabe von Festkommazahlen

Eine Festkommazahl in einer Datei soll aus einer bis drei Dezimalziffern[13] vor und genau zwei Ziffern nach dem Komma bestehen. Danach folgen ein oder mehrere Trennzeichen (0x20 (Leerzeichen), 0xa (Zeilenumbruch) oder 0xd (Wagenrücklauf)).

1. Beschreiben Sie dieses Eingabeformat als formale Sprache mit den Ersetzungsregeln auf Folie 70.
2. Beschreiben Sie den spracherkennenden Automaten als Graph.
3. Schreiben Sie ein C-Programm

   ```
   float getZahl()
   ```

   das mit »uint8_t getChar()« Zeichen aus einer Datei liest, bei jeder korrekt erkannten Zahl den Wert und sonst NaN (not a number) zurückgibt.

[13] Darstellung als ASCII-Zeichen '0'→0x30, ..., '9'→0x39, ','→0x2C

Aufgabe 3.10: Feld mit Prüfkennzeichen

Eine Schnittstelle habe folgende Struktur als Format:

```
struct SDatx {
 uint8_t len;  // Wertebereich 0 bis 200
 uint16_t *feld; // Feld mit Indexbereich 0 bis len-1
                 // WB Elemente 0 bis 10000
 uint32_t PKZ ; // Prüfkennzeichen
}
```

Berechnungsvorschrift für das Prüfkennzeichen:

$$PKZ = \sum_{i=0}^{len} i \cdot feld_i$$

1. Wie lautet das Prüfkennzeichen für len=5, feld(0)=37, feld(1)=9879, feld(3)=238 und feld(4)=615?
2. Wie ist das Prüfkennzeichen zu ändern, wenn das Feldelement $i \in [0, 4]$ mit einem neuen Wert w überschrieben wird?

Aufgabe 3.11: Zugriffsmethoden mit Kontrollen

Schreiben Sie für die Feld-Datenstruktur auf der Folie zuvor

1. eine Funktion

   `uint8_t`[14] `writeSDatx(uint8_t idx, uint16_t Wert)`

   zum Beschreiben eines Feldelementes und
2. eine Funktion

   `uint8_t`[15] `CheckSDatx()`

   zur Formatkontrolle.

[14] Rückgabewert 1 bei Wertebereichsfehler für den Index, 2 bei Wertebereichsfehler des Wertes, sonst null.

[15] Rückgabewert 201 bei Wertebereichsfehler der Länge, i bei Wertebereichsfehler von Wert i, 202 bei Prüfkennzeichenfehler, sonst null.

Aufgabe 3.12: Kontrollfunktion für eine Liste

Die Datenstruktur

```
struct DVListe {
 struct Element *first;
 struct Element *last;
}
```

beschreibt eine doppelt verkettete Liste von 0 bis max. 1000 Elementen

```
struct Element{
 struct Element *last;
 struct Inhalt w;
 struct Element *next;
}
```

mit aufsteigend sortierten Werten w.

Für die Sortierreihenfolge existiert eine Kontrollfunktion:

```
uint8_t groesser(struct Inhalt w1, struct Inhalt w2);
```

die eins, wenn w1 größer als w2 ist und sonst null zurückgibt.

Schreiben Sie eine Kontrollfunktion

```
uint8_t checkDVListe(struct DVListe l)
```

die alle vorgegebenen Formatregeln kontrolliert. Ohne Formatfehler soll diese null und sonst eine den Fehlertyp charakterisierende Nummer zurückzugeben.

Ideen für weitere Aufgaben

- Welche Kontrollen sind für die Zwischen- und Entergebnisse folgender Berechnungen geeignet[16] (Ausdruck, Zuweisung).
- Definieren Sie eine eigenen Funktion + Kontrolle.
- Welche Kontrollen sind für das nachfolgend definierte Signal möglich.

16

- Lösen mit gesundem Menschenverstand.

Wertekontrollen

Wertekontrollen

Die bisher behandelten Kontrollen überprüfen das Format, d.h. die Zulässigkeit der Ein- und Ausgaben, nicht deren Richtigkeit:

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

In diesem Abschnitt werden Kontrollen der Ein- und Ausgabewerte auf Richtigkeit behandelt:

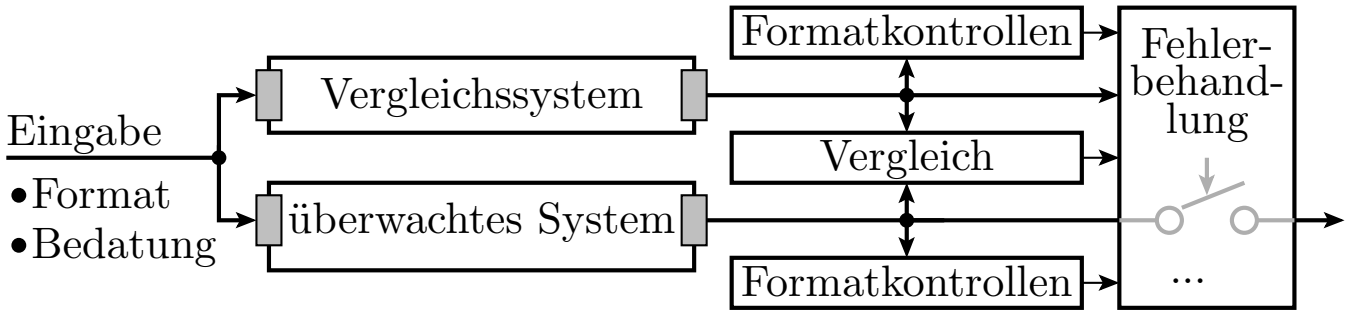
- Vergleich mit Sollwerten[17],
- Mehrfachberechnung und Vergleich,
- Rückrechnung der Eingabe aus den Ausgaben und Vergleich und
- die Kontrolle der Suchkriterien bei Suchproblemen.

[17] Erfordert Testbedingungen und wird auf Foliensatz Test behandelt.

Mehrfachber. & Vergleich

Mehrfachberechnung und Vergleich

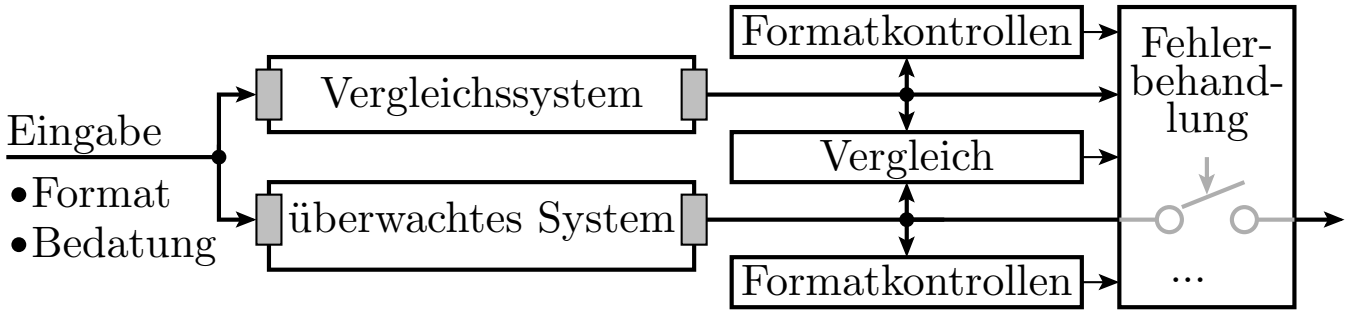
Die Ergebnisse werden mehrfach berechnet. Übereinstimmende Ergebnisse gelten als fehlerfrei. Bei Abweichung kann über einen Mehrheitsentscheid oder Zusatzkontrollen ein Ergebnis als richtig gewertet und weitergereicht werden.



Das Vergleichssystem kann sein:

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

Vergleich mit einem »Golden Device«

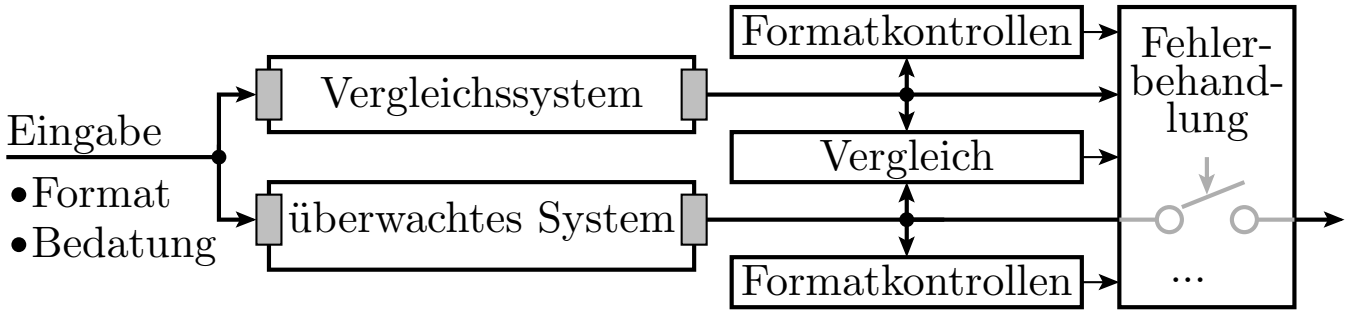


Ein »Golden Device« ist ein gründlich getestetes System, ein Mustergerät oder bei einem Regressionstest die Vorgängerversion. Seine Ausgaben gelten per Definition als richtig. Mit einem »Golden Device« und einem idealen Vergleicher[18] werden alle Werteabweichungen erkannt.

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

[18] Vergleich, 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 Systems. 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 114).

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

Exakter Vergleich und Fenstervergleich

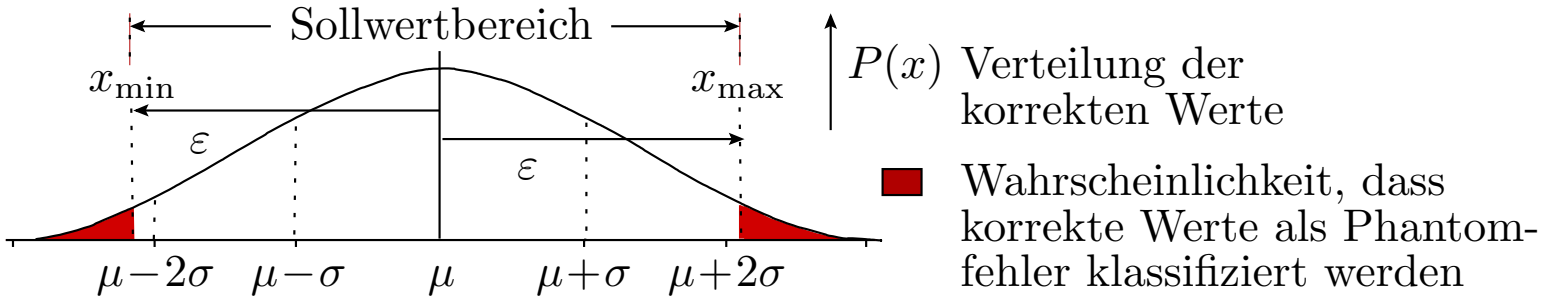
Die Sollausgaben eines Systems können exakte Werte oder von zufälligen Einflüssen überlagerte Werte sein:

- Exakte Werte: Ergebnisse von Logik- und ganzzahligen Operationen. Vergleich:

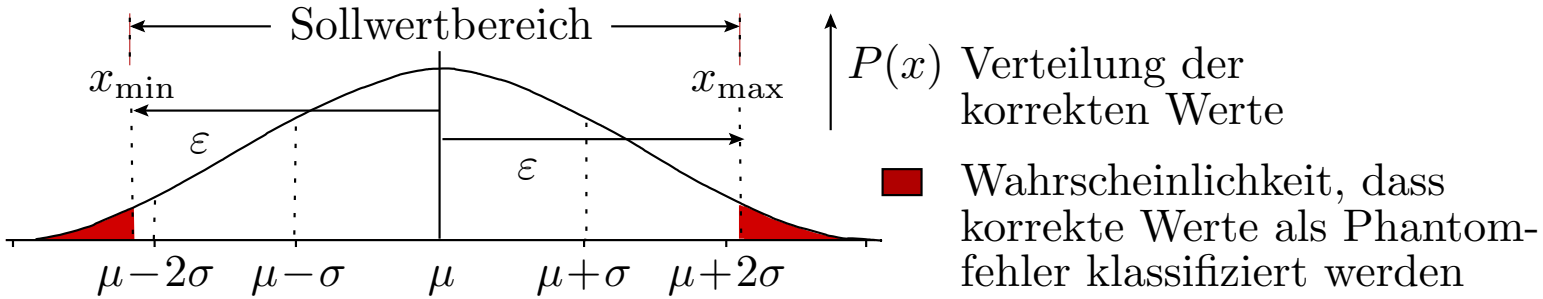
```
if (x/=Sollwert) <Fehlerbehandlung>;
```

- Zufallsgrößen: Analoge Eingabewerte, rundungsfehlerbehaftete Ergebnisse von Berechnungen mit Nachkommastellen. Bereichskontrolle:

```
if (x<minWert)||(x>maxWert) <Fehlerbehandlung>;
```



Festlegung des Vergleichsfensters



Für normalverteilte Zufallsgrößen wird für den Sollwertbereich zweckmäßiger Weise $\mu \pm \varepsilon$ mit $\varepsilon \geq 2\sigma$ gewählt (μ – Erwartungswert, σ – Standardabweichung). Wahrscheinlichkeit α, dass korrekte Werte als falsch klassifiziert werden:

ε/σ	1	2	3	4	2,05	2,33	2,57	2,88	3,10
α	15,9%	2,27%	0,13%	0	2%	1%	0,5%	0,2%	0,1%

(vergl. Foliensatz F2, Abschn. 4.2).

Quantisierungs- und Rundungsfehler

Bei Quantisierung und Rundung ist ein Fehler von $\pm 0,5\,\mathrm{LSB}$[19] unvermeidbar. Die Standardabweichung als die Wurzel aus der mittleren quadratischen Abweichung ist die reichliche Hälfte davon:

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

Beispiel Rundung auf 8 Nachkommabits. Max.Rundungsfehler $\pm 0{,}195\%$, Standardabweichung $\sigma = 0{,}056\%$:

Wert	nächster darstellbarer Wert	Rundungsfehler
125,4380	0x7D,70=125,4375	−0,05%
2,7130	0x2,B7=2,7148	+0,18%
28,2000	0x1C,33= 28,1992	−0,08%

[19] LSB (least significant bit) Wert einer Quantisierungsstufe.

Für exakte Werte genügt nach der Quantisierung ein Vergleichsfenster von ± 1 LSB. Bei Sensorwerten kommt der Messfehler hinzu, der das erforderliche Vergleichsfenster um weitere 1...2 LSB vergrößert. Eine digitale Verarbeitung mit zu wenige Nachkommastellen kann die Wirkung der Mess- und Quantisierungsfehler und damit die Größe der erforderlichen Vergleichsfenster erheblich vergrößern. Beispiel Multiplikation der drei auf 8 Nachkommabits gerundeten Werte der Folie zuvor:

$$\begin{aligned} 125{,}438 \cdot 2{,}713 \cdot 28{,}2 &= 9596{,}8349 \\ \text{0x7D,70} \cdot \text{0x2,B7} \cdot \text{0x28,2} &= \text{0x2569,C7} = 9577{,}7773 \\ \text{Abweichung:} &\quad -19{,}0575 \end{aligned}$$

Ob das akzeptabel ist, sollte aus der Spezifikation hervorgehen.

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 summieren.

Diversität

Erkennbare Fehfunktionen

Mehrfachberechnung und Vergleich erkennt

- 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 oft:

- 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.

Erkennungswahrscheinlichkeit und Diversität

Maskierung durch Zufall ist extrem unwahrscheinlich. Beispiel:

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

Praktisch alle Maskierungen entstehen durch fehlende »Verschiedenartigkeit«, Fachbegriff Diversität[20].

Definition 1

Diversität sei die Wahrscheinlichkeit, dass eine Fehlfunktion durch doppelte Berechnung und Vergleich erkannt wird.

[20] Von lateinisch diversitas „Vielfalt“ abgeleitet.

Schätzen der Diversität

$$Div = \frac{\text{Anz(DV\_Erk)}}{\text{Anz(FF)}} = 1 - \frac{\text{Anz(DV\_NErk)}}{\text{Anz(FF)}}$$

(Anz(DV_Erk) – Anzahl der durch Dopplung und Vergleich erkannten; Anz(DV_NErk) – Anzahl der durch Dopplung und Vergleich erkannten FFs).

Praktische Probleme:

- Zählen der nicht erkannten FFs erfordert eine Kontrolle mit $p_{\mathrm{E}} = 100\%$, z.B. Soll-/Ist-Vergleich mit Golden Device.
- Eine aussagekräftige Schätzung verlangt ausreichend große Zählwerte für Anz(DV_Erk) und Anz(DV_NErk) (siehe Foliensatz 2).

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:

- Unterschiedliche Eingabe: Bei getesteten, überwiegend funktionierenden Systemen gibt es für fehlerhaft ausgeführte Service-Leistungen meist Alternativen mit anderen Bedatungen und anderen richtigen Ergebnissen.
- Unterschiedliche Aufgabenlösungen: Viele Aufgaben lassen sich ganz unterschiedlich lösen, liefern dann aber für dieselben Eingaben keine direkt vergleichbaren richtigen Ergebnisse, z.B. schon, wenn die Ein- und Ausgabeformate abweichen.

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

Maximalen Ausschluss von Entstehungsursachen für gleich wirkende Fehler erhält man mit getrennten Implementierungen nach derselben Spezifikation durch unterschiedliche Entwurfsteams. Die Unterschiedlichkeit ensteht zufällig oder kann forciert werden: Vorgabe unterschiedlicher Algorithmen, Werzeuge, ...

Die ursprüngliche euphorische Meinung, durch die Diversität lassen sich praktisch alle identischen Fehler 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 Gemeisamkeiten in die Entwürfe. Auch die Neigung von Menschen, gewisse Fehler zu wiederholen, trägt dazu bei. Erzielbare Erkennungssicherheit laut [7]:

$$p_{\mathrm{E}} \leq 90\%$$

Syntaktische Diversität

Systematische und automatisch durchführbare Modifikationen an einem funktionsfähigen Programm. Nach [2]:

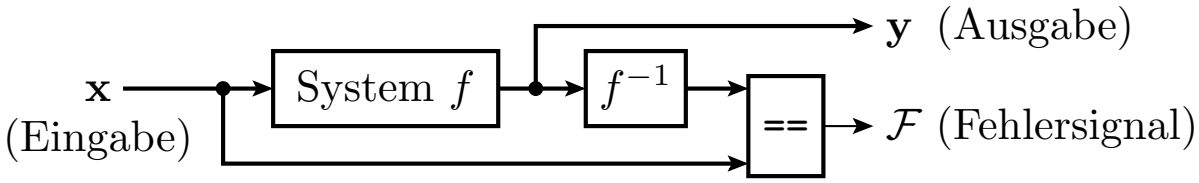
- Variation der Register- und Speicherzuordnung.
- Variation der Anweisungsreihenfolge bei parallelisierbaren Anweisungen.
- Variation der Bitcodierung von Datentypen.
- Variation der vom Compiler einzubindender Code-Bausteine.

Entwurfsaufwand kaum höher als bei Beschränkung auf natürliche Diversität. Erkennungssicherheit soll höher als bei einfacher Wiederholung und Vergleich sein. Experimentelle Untermauerung fehlt.

Loop-Back Test

Rückrechnung der Eingabe (Loop-Back Test)

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 Empfang,
- ...

- 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 naheliegenste und einfachste Lösung.
- Bei zu erwartenden Mess-, Quantisierungs- und numerischen Fehlern Fenstervergleich.

Probe

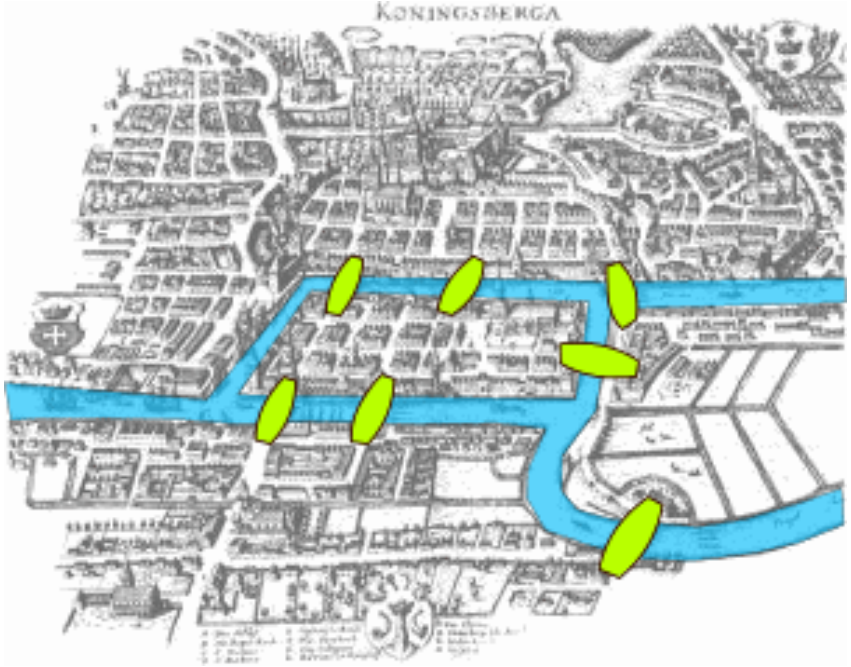
Kontrolle durch eine Probe (im math. Sinne)

Suchaufgaben sind oft durch eine Kontrollfunktion (Probe) spezifiziert:

- Suche eine Funktion, die eine Differenzialgleichung erfüllt. Kontrolle durch Einsetzen der Funktion in die Differenzialgleichung.
- 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 meist 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, bei dem man alle sieben Brücken über den Pregel genau einmal überquert werden.

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.
```

Aufgaben

Aufgabe 3.13: Kontrolle für einen Sortieralgorithmus

Eingabe- und Ergebnisformat sei folgende Struktur:

```
struct DFeld {
 uint8_t len;   // Länge des Feldes
 int32_t *feld; // Zeiger auf den Feldanfang
 int32_t PSum;  // Prüfsumme
}
```

Entwickeln Sie ein Kontrollfunktion

```
uint8_t CheckSortDFeld(
  struct DFeld u,  // unsortierte Eingabe
  struct DFeld s); // sortierte Ausgabe
```

die die Sortierreihenfolge, die Länge und die Prüfsumme kontrolliert.

Rundungsfehler:

- ADC-Wandlung 10 Bit, max. Fehler ± 1 LSB. Vor dem Vergleich Kalibrieren

$$soll - eps < (w - a) * k < soll + eps$$

Vergleich mit

$$(soll - eps)/k + a < w < (soll + eps)/k + a$$

- Differenz von zwei Messwerten vor und nach Produktbildung.

Fehlerbehandlung

Fehlerbehandlung

Beseitigung von Fehlfunktionen, bevor sie Schaden verursachen.
Typische Formen der Fehlerbehandlung:

- Schadensminimierung (gefahrenfreien Zustand herstellen, ...),
- Protokollieren der Fehlfunktionen,
- Wiederherstellung eines betriebsbereiten Zustands,
- Wiederholung der Berechnung, ...

Protokollieren der Fehlfunktionen dient zur kontinuierlichen Verbesserung der Verlässlichkeit. Betriebssysteme, Anwendungssoftware, ... versenden Fehlernachrichten zum Hersteller. Bei KfZ-Steuergeräten werden die Fehlerinformationen bei der Wartung ausgelesen und verschickt. Der Hersteller versucht die aufgetretenen Fehlersituationen mit Testfällen nachzubilden, beseitigt erkannte Fehler und verteilt neue Versionen.

Fail-Safe/-Fast/-Slow

Strategien zum Umgang mit Fehlfunktionen

- Fail-Save: Übergang in einen gefahrlosen Zustand
- Fail-Fast: Sofortiger Bearbeitungsabbruch und Fehlersuche.
- Fail-Slow: Fortsetzung der Arbeit im Fehlerfall, solange noch eine sinnvolle Möglichkeit besteht.

Beispiele für Fail-Safe-Systeme

- Zwangsbremsung, wenn Lokführer Haltesignal überfährt.
- Bei funkgesteuerten Modellen Motor aus und Bremsen ein bei schwachem Funkempfang oder schwacher Batterie.
- Bei erkanntem Sensorausfall Maschinen oder Anlagen kontrolliert anhalten.
- Bei Erkennen eines freien Falls (Laptop fällt herunter), Festplattenköpfe in Parkposition bringen.
- Ruhestromprinzip (nächste Folie)

Ruhestromprinzip (gebräuchliche Fail-Safe Technik)

- Sicherheitsbremsen: Bei Stromausfall, Drahtbruch, geplatztem Bremsschlauch, ... setzt Bremswirkung ein.
- Alarmanlagen: Bei Durchtrennen einer Signalleitung Alarm auslösen.
- Eisenbahnsignal:
  - Bei Drahtseilriss für ein Streckenfreigabesignal Strecke sperren (Signalarm fällt runter).
  - Bei fehlendem Ruhestrom auf einer Meldeleitung, Anzeige einer Störung.
- Brandmeldeanlage: Alarm bei Durchtrennen der Signalleitungen eines Brandmelders.
- Not-Aus: Bei Leitungsbruch Not-Aus einleiten.

Fail-Fast

Bei erkannter Fehlfunktionen (Zugriff auf nicht vorhanden Daten, Division durch Null, Wertebereichsverletzung ...) Abarbeitung stoppen, typ. mit Assert-Anweisung, z.B. in VHDL

```
assert <Bedingung> report <Beschreibung>
  severity note|warning|fault|error;
```

- Nutzung Systemrufe (Software-Interrupts).
- Sinnvoll während des Tests.
- Ausgabe aller für die Lokalisierung der Ursache dienlichen Daten einprogrammieren!

Fail-Slow

Fail-Slow: Das System soll bei einer erkannten Fehlfunktion solange wie möglich weiterarbeiten, z.B. beim Zugriff auf nicht vorhandene Daten, Weiterarbeit mit Standardwerten.

- weniger sichtbare Fehlfunktionen, höhere Zuverlässigkeit
- sinnvoll im Einsatz.
- Fehlerinformationen aber möglichst an die Hersteller zur Fehlersuche weiterleiten.

Umschaltung zwischen Fail-Fast und Fail-Slow[5]

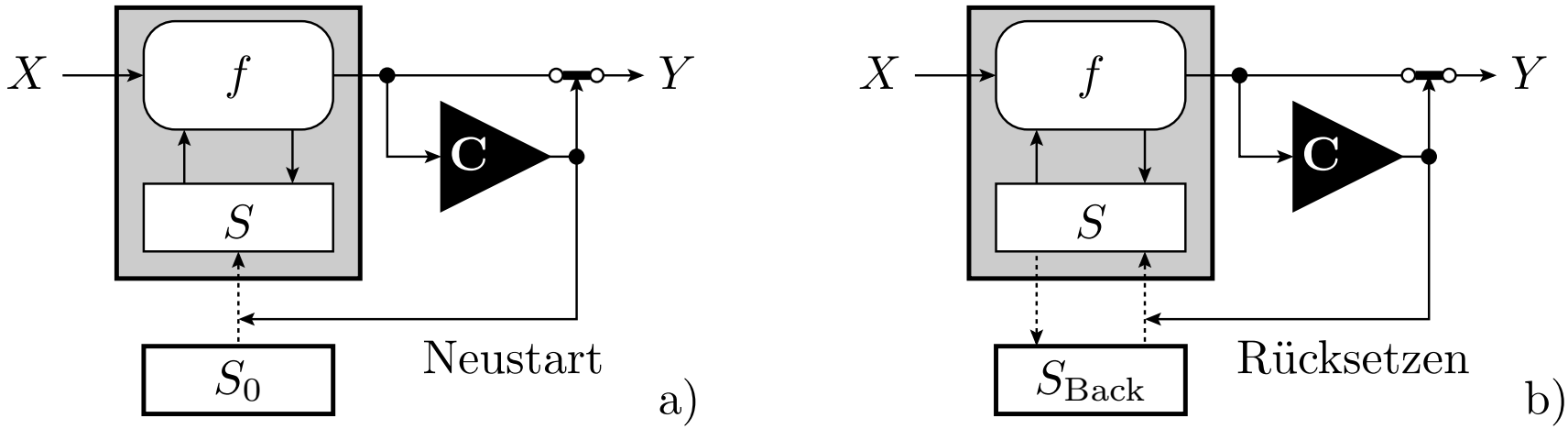
- Angabe der Fehlerschwere in den Assertion-Anweisung
- Umschaltung des Exeption-Handlers, dass nur bei schweren Fehlern ein Abbruch erfolgt.
- Protokollierung und Versendung aller Assertion-Ausgaben.[21]

[21] Höfliche Programme fragen bzw. informieren zumindest, bevor sie Mails nach Hause schicken.

Neustart, Wiederholung

Neuinitialisierung

Zur Fortsetzung der Arbeit nach einer Fehlfunktion benötigt das System einen nicht kontaminierten Bearbeitungszustand.



- Statische Neuinitialisierung (fester Anfangszustand)
- Dynamische Neuinitialisierung (zurück zum letzten gesicherten Bearbeitungszustand).

Statische Neuinitialisierung

- Typisch für Prozessrechner, die in einem festen Zeitraster Ausgabesignale aus Eingabesignalen berechnen (SPS, Steuerrechner, Reglungen, ...).
- Lösung: Zeitüberwachung mit Watchdog und Neustart bei Zeitüberschreitung. Ein Watchdog ist ein Zähler, der
  - zu Beginn und nach jeder Taskabarbeitung rückgesetzt wird
  - bei Überlauf über Reset das komplette Programm auf dem Rechner neu startet.
  - Der Watchdog wird vom Programm eingeschaltet und kann nicht vom Programm, d.h. auch nicht durch Fehlfunktionen gestoppt werden, sondern nur durch Neustart.
- Ein Rechner hat gewöhnlich eine Reset-Taste für den Neustart, wenn alle anderen Fehlerbehandlungen versagen.

Dynamische Neuinitialisierung

Erfordert regelmäßige Sicherheitskopien des Bearbeitungszustands in einem gesicherten Speicher, z.B.

- für einen PC manuell auf einer externen Festplatte oder einem Stick,
- vom Rechenzentrum mit einem Backup-System oder
- im EEPROM eines Mikrocontrollers.

Oft werden nur Daten gesichert, die sich nicht problemlos neu berechnen lassen:

- Eingabedaten, Ergebnisse langwieriger Berechnungen, ...
- Beispiele: Editoren, Logistiksystemen, Datenbank, ... die Eingaben und Aufträge seit der letzten Sicherheitskopie protokollieren[22].

[22] So kann der aktuelle Zustand aus dem letzten gesicherten Zustand und den protokollierten Eingabe- und Auftragsdaten wiederhergestellt werden.

Wiederholung

Standardfehlerbehandlung für nicht reproduzierbare Fehlfunktionen

- störungsbedingte Fehlfunktionen,
- Ausfall der Spannungsversorgung,
- Eingabefehler, Übertragungsfehler,
- Lesefehler von Festplatte, ...

Daten mit potenziellen Verfälschungen werden in der Regel mit Prüfsummen, CRC, ... gesichert (Erkennungssicherheit nahe 1).

Standardreaktion bei Prüfkennzeichenfehler:

- Wiederholung bis Erfolg oder Zeitüberschreitung
- Bei Zeitüberschreitung Fehlermeldung und Abbruch oder Weiterarbeit ohne angeforderte Daten.

Reaktion auf Speichermangel:

- Platz schaffen und wiederholen.

Fehlerisolation

Fehlerisolation

Eine automatische sinnvolle Reaktion auf Fehlfunktion:

- Fail-Safe,
- dynamische Neuinitialisierung und Wiederholung,
- andere anwendungsspezifische Reaktionen

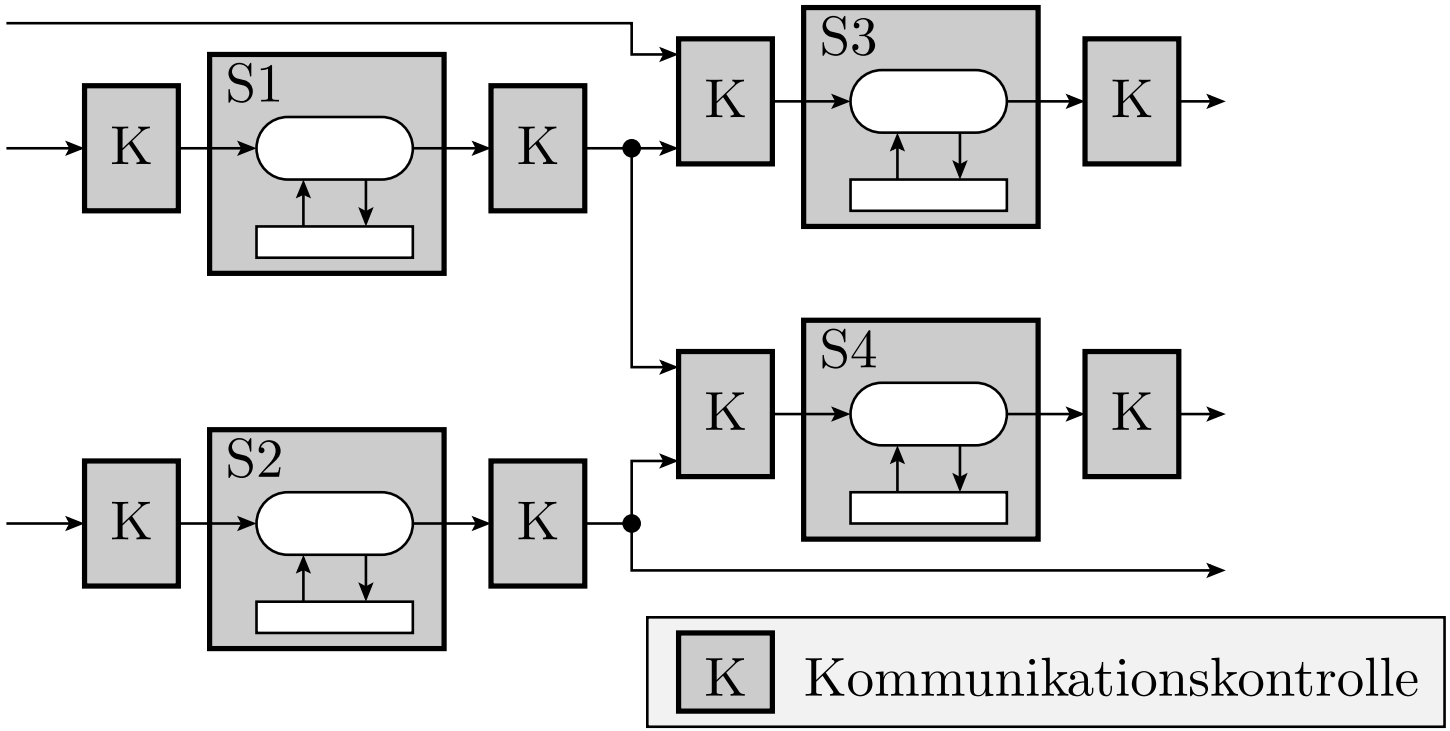
erfordern, dass der Zustand des Systems, das die Fehlerbehandlung durchführt, unverfälscht ist:

- Ein HW-Reset zur statischen Neuinitialisierung hat keine kontaminierbaren Zustände.
- Der Back-Up-Speicher zur dynamischen Neuinitialisierung benötigt eine Zugriffsschutz gegen Verfälschungen.
- Fail-Save-Funktionen, Notbetrieb, ... benötigen einen funktionierenden Rechner und geschützte Daten.

Fehlerisolation: Maßnahmen zur Verhinderung, dass sich Datenverfälschungen und andere Fehlfunktionen über Teilsystemgrenzen ausbreiten.

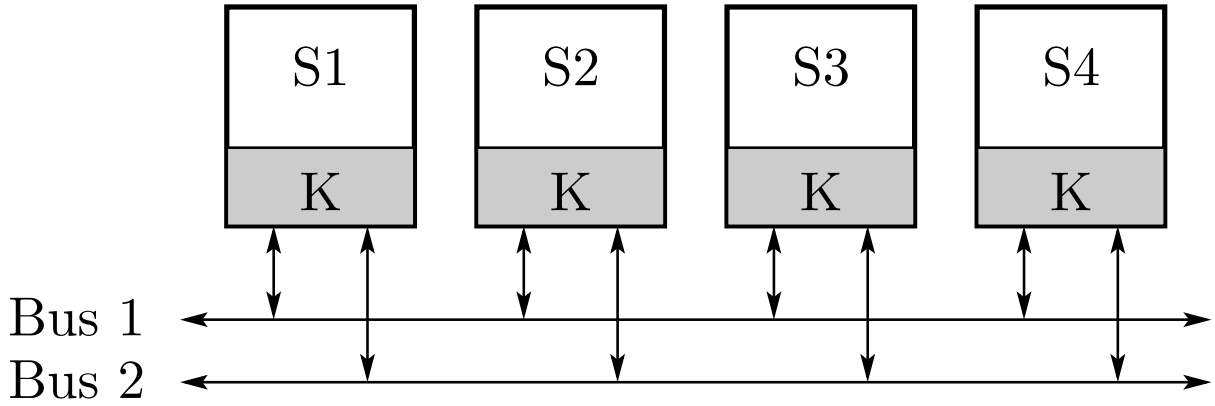
Prinzip der Fehlerisolation

- Modulares System mit Kontrollen an den Schnittstellen. Keine Weitergabe kontaminierter Daten.



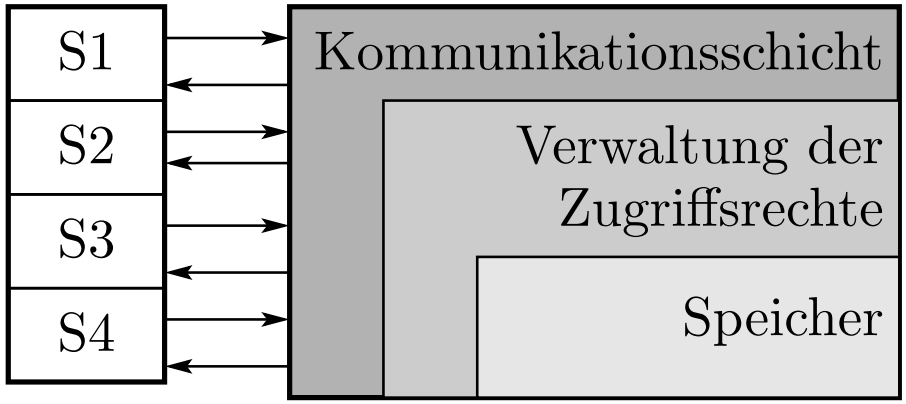
Fehlerisolation in Bussystemen

- Bus: zentrale Informationsschnittstelle.
- Jedes Teilsystem erhält Kontrollfunktionen für alle über den Bus empfangen und versendeten Ein- und Ausgabedaten.



- Fehlerbehandlung für Eingabefehler: Keine Weiterverarbeitung, Protokollierung der Fehlfunktion, Diagnose des Quellsystems, ...
- Fehlerbehandlung für Ausgabefehler: Keine Ausgabe, Neustart, Wiederholung, ...

Fehlerisolation mit einem Betriebssystems



- 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.

Die Zuordnung von Resourcen (physikalischer Speicher, Ein- und Ausgabegeräte, Kommunikation zu anderen Prozessen...) sind nur über Systemrufe möglich, bei denen das Betriebssystem Kontrolle über alle Daten hat.

Fehlertoleranz

Redundanz und Fehlertoleranz

Fehlertoleranz: Korrektur eigener, selbst erkannter Fehlfunktionen unter Verwendung redundanter[23] Funktionseinheiten:

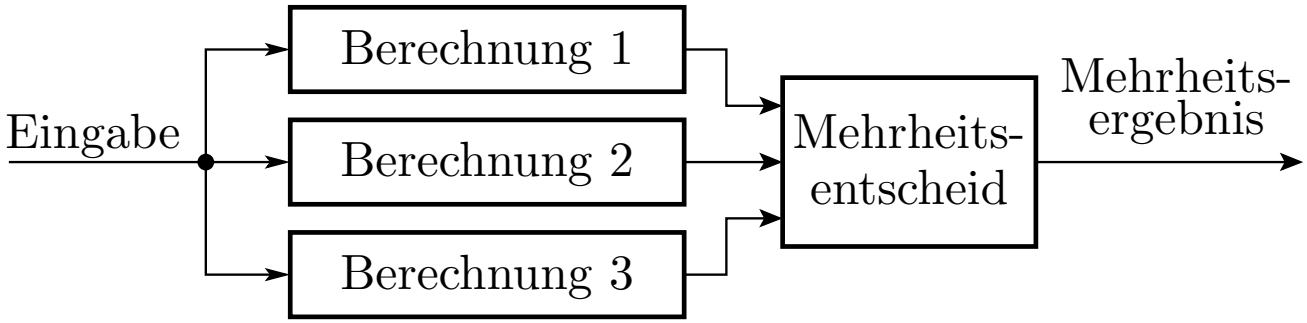
- Informationsredundanz: siehe fehlerkorrigierende Codes.
- homogene Redundanz (Redundanz mit gleichartigen Mitteln): Reserveeinheiten zur Vorbeugung gegenüber Hardware-Ausfällen z.B. zwei Glühbirnen im roten Teil einer Ampel (siehe später heiße und kalte Reserve) und
- diversitäre Redundanz (Redundanz mit ungleichartigen Mitteln) zur Korrektur von falschen Ergebnissen.

Im Bereich der diversitärer Redundanz haben sich im wesentlichen zwei Ausprägungen durchgesetzt:

- N-Versionstechnik mit Mehrheitsentscheid und
- und Systeme mit Rücksetzblöcken.

[23] Redundanz ist eine System-Resource, die für die eigentliche Funktion des Systems nicht erforderlich ist.

Mehrfachberechnung mit Mehrheitsentscheid



Klassische Architektur für ein fehlertolerantes System, bereits 1956 von »von Neumann« vorgeschlagen:

- Jede Berechnung erfolgt $n \geq 3$ mal.
- In jedem Schritt Mehrheitsauswahl.
- Ohne Mehrheitsergebnis nur Fehlererkennung.

Die originale Idee sah die zeitgleiche Berechnung auf unterschiedlichen Rechnern vor.

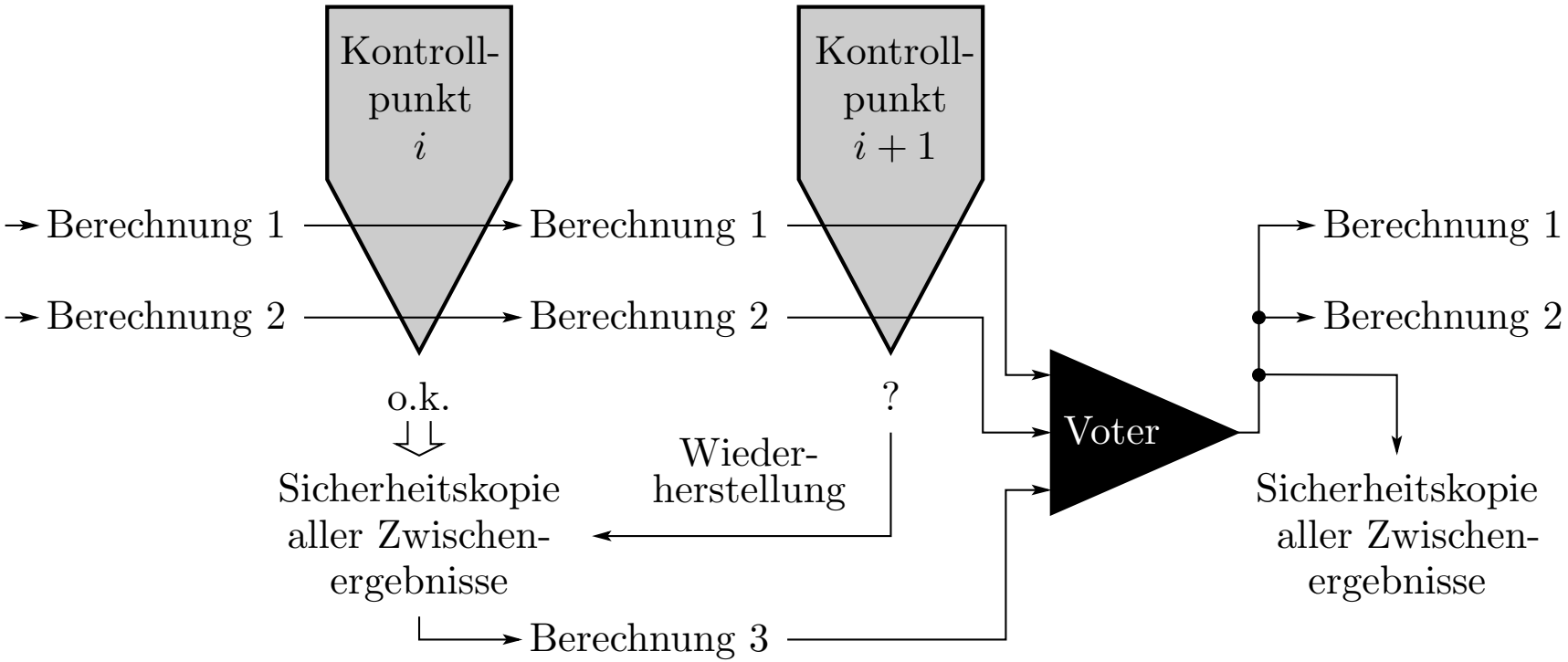
Probleme und Verbesserungsansätze

- Hoher Aufwand: Das originale n-Versionssystem verlangt mindestens drei statt nur einem Rechner.
  Abhilfe schafft, die Berechnungen nacheinander auf demselben oder nur zwei Systemen auszuführen. Das spart Hardware und erhöht den Rechenaufwand.
- Ein Mehrversionssystem in dieser Form toleriert keine übereinstimmenden Entwurfsfehler in den Systemen.
  Das Original war für die Tolerierung der häufigen Ausfälle von Röhrenrechnern konzipiert. Heute sind Entwurfsfehler die häufigste Ursache für ein Versagen. Für deren Tolerierung müssen die Berechnungen verschiedenartig (diversitär) sein (unterschiedliche Hardware, verschiedene Software-Entwürfe, ...).

- Eine Diversität, die alle Ursachen für übereinstimmende Fehlverhalten ausschließt (fehlerhaft oder unvollständig Anforderungen, gleiche Denkfehler, ...), ist für Systeme mit angestrebtem identischen Verhalten im fehlerfreien Fall unerreichbar.
- Bei diversitären Service-Leistungen können sich auch die richtigen Ergebnisse unterscheiden. Ein Vergleich diagnostiziert das als Fehlfunktion (Phantomfehler).

Praktisch eingesetzte fehlertolerante Systeme nutzen verfeinerte Techniken mit eine Vielzahl weiterer Kontrollen und einer Vielzahl einprogrammierten Verhaltensweisen bei Fehlfunktionen.

Check-Point-Roll-Back-Recovery [4]



- Nur zwei parallel ausgeführte Berechnungen.
- An einprogrammierten Kontrollpunkten im Programm werden die Bearbeitungszustände[24] verglichen.

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

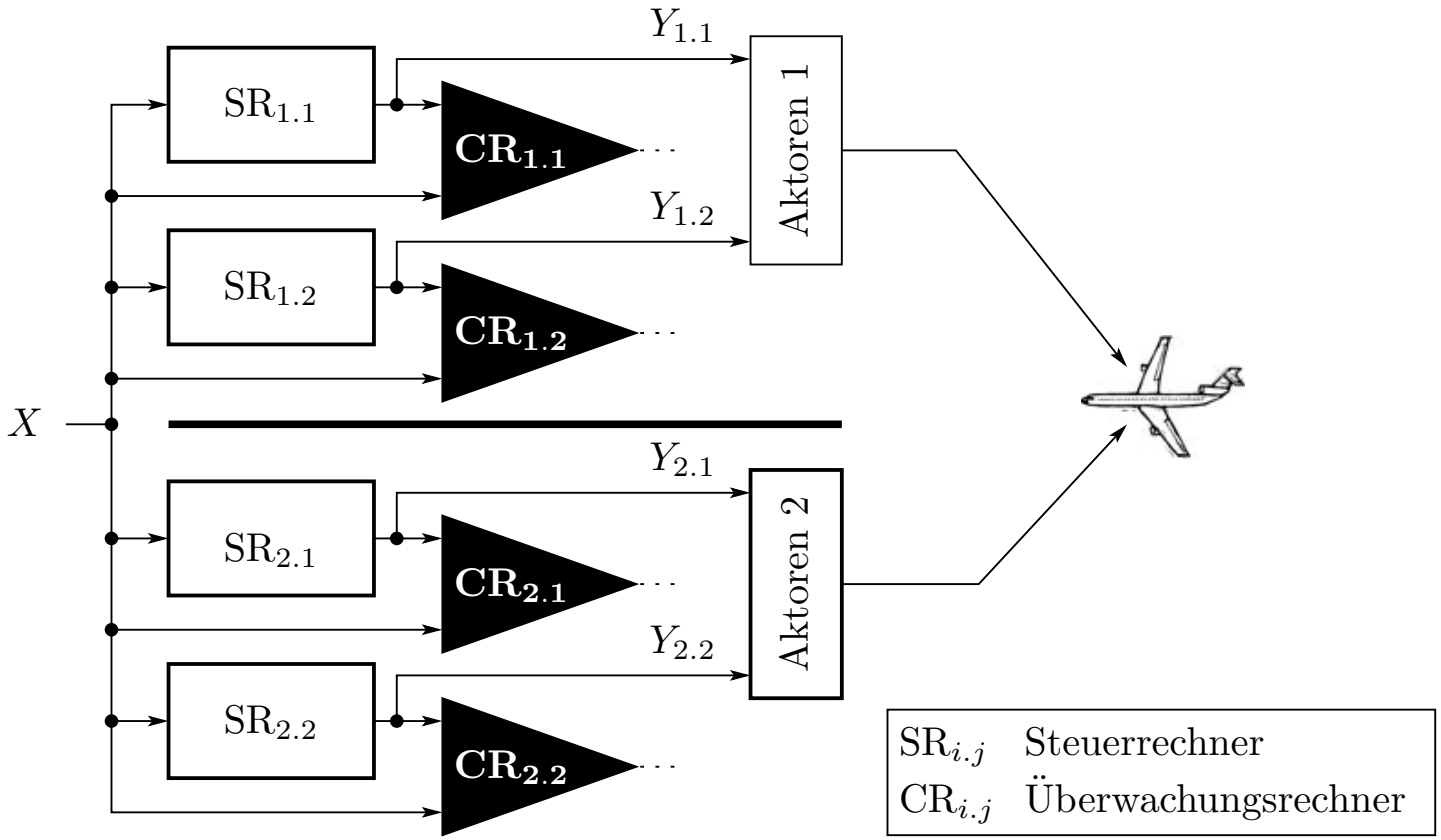
- Bei Übereinstimmung Speicherung des Bearbeitungszustands in einem geschützten Speicher.
- Bei Abweichung, Laden der letzten Sicherheitskopie und Berechnungswiederholung (Roll-Back Recovery).
- Nach Roll-Back Recovery am nächsten Kontrollpunkt wieder Mehrheitsentscheid.
- Wenn Mehrheitsergebnis, diesen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

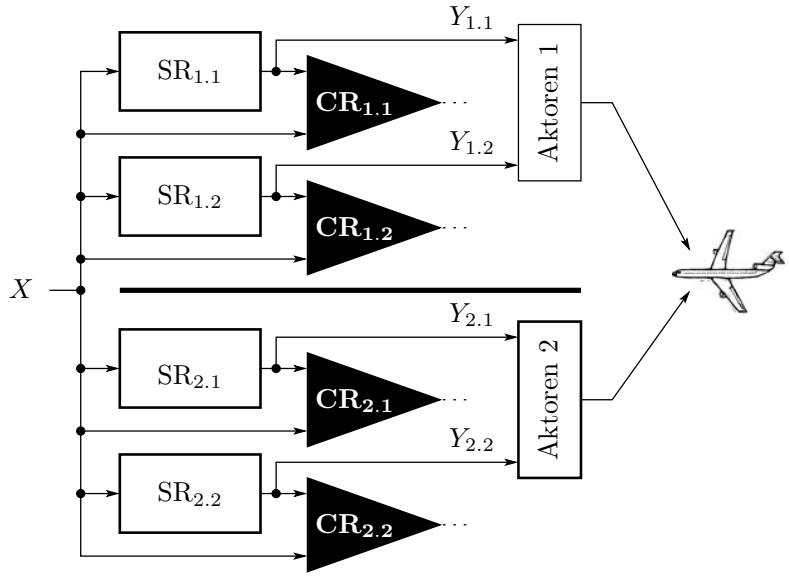
Sequoia-System [1]:

- 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 [6]

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 Rechnerpaar 1 übernimmt Rechnerpaar 2, bei Komplett- Sensor- oder Aktorausfällen System 1 übernimmt Sysem 2.
- Diversität: Rechner unterschiedlicher Hersteller, getrennte Software-Entwicklung nach Spezifikationen, die unabhängig von einer gemeinsamen Basisspezifikation abgeleitet wurden.

Zusammenfassung

- Mehrfache Berechnung kostet mindestens die doppelte Rechenzeit oder den doppelten Hardwareaufwand.
- Für nicht reproduzierbare Fehlfunktionen (verursacht z.B. durch Störungen, Eingabe- und Initialisierungsfehler) genügt eine Berechnung auf demselben oder einem gleichen System.
  - Standardfehlerbehandlung für nicht deterministische Fehlverhalten, verursacht z.B. durch Eingabe-, Übertragungs-, Initialisierungs- und Festplattenlesefehler)
- Reproduzierbare Fehlfunktionen verlangen eine Mehrfachberechnung mit diversitären Systemen. Einsatz
  - unter Testbedingungen: Regressionstest (Funktionsvergleich aufeinanderfolgender Software-Versionen) und Mehrversions-Schaltungsentwürfe
  - unter Betriebsbedingungen: Mehr als doppelter Entwurfsaufwand.

Manuelle Reaktion

Wenn die automatische Fehlerbehandlungen versagt

ist Handarbeit erforderlich:

- System ersetzen. Erfordert fehlerärmeren Ersatz. Sinnvoll nach Ausfällen. Unüblich für nicht gefundene Entwurfs- und Fertigungsfehler[25].
- Input-Workaround: Service anders nutzen, z.B. mit anderen Eingaben, in anderer Reihenfolge. (Suche eines diversitären Lösungsweges.)
- Fehlerreport (Change Request): Weitergabe der Fehlerinformationen an den Hersteller in der Hoffnung, dass der zugrund liegende Fehler beseitigt wird.

[25] Die sollten hinreichend selten Fehlfunktionen verursachen.

Ersatz

Es gibt selbst bei Software Situationen, in denen die Verlässlichkeitsprobleme und Nutzungsbeschränkungen so gravierend sind, dass ein Umstieg auf andere Produkte angemessen erscheint.

Sehr aufwändig und kann schief gehen.

Input-Workaround

- Geänderte Anfrage, die im fehlerfreien Fall dasselbe Ergebnis liefert (andere Bedatung, andere Eingabeschnittstelle, andere Ausführungsreihenfolge, ...)
- Lernprozess für IT-Benutzer: Wie lassen sich mit dem System die eigenen Aufgaben am umkompliziertesten lösen?
- Auch bei langer Systemnutzung ohne Beseitigung erkannter Fehler nimmt die Häufigkeit der Fehlfunktionen durch den Lernprozess der Nutzer ab.

Beispiele studentischer Arbeiten im Arbeitsbereich, die nur mit Input-Workarounds lösbar waren:

- CAN-Busansteuerung c167: SFR-Register mussten in einer nicht in der Doku stehenden Reihenfolge initialisiert werden.
- Sharp-Abstandssensoren: Im Datenblatt steht nicht, dass sich mehrere Sensoren im Raum gegenseitig stören, ...
- Power-Cube (Geleke des großen Laborroboters): Bei Nachrichtenkollision auf dem CAN-Bus keine Übertragungswiederholung und andere Bugs. ...

Andere Beispiele:

- Vor der Compilierung des Linux-Kernals können die Hardware-Bugs abgewählt werden, für die Input-Workarounds überflüssig sind (z.B. Float/Div-Bug).

Bei Problemen mit neue Hard- oder Software am besten Internet durchsuchen, ob schon Workarounds dafür bekannt sind.

Fehlerreport (Change Request)

Das Abstellen eines Fehlers erfordert Daten zur Lokalisierung.

- Manuelle Erfassung: detaillierte Beschreibung, bei welchen Eingaben welche Fehlfunktion aufgetreten ist. (Viel manuelle Arbeit, auch Überzeugungsarbeit am Telefon, ...)
- Automatische Generierung[26]:
  - welche Kontrolle hat die Fehlfunktion erkannt (Abbruchadresse, Übersetzungsversion)
  - Aufruf-Stack
  - Werte der Programmvariablen (Core-Dump oder Teile davon)
  - Betriebssystemversion, Hardware- und Treiberkonfiguration (Windows sammelt diese Daten z.B. in den cab-Files).

[26] Erheblicher Programmieraufwand. Der Windows Error Report Service (WER) erlaubt z.B., dass Programmier-Teams, denen Fehler zugeordnet werden, Auftrittshäufigkeiten ablesen, die zur Lokalisierung anzufordernden Daten konfigurieren können, ...[3]

Aufgaben

Aufgabe 3.14: Beispiele für die Fehlerbehandlung

Nennen Sie Beispiele (Ihnen bekannte Programme und Geräte) die folgende Techniken nutzen:

1. Zeitüberwachung mit Service-Abbruch bei Zeitüberschreitung.
2. Wiederholungsanforderung nach fehlerhaftem Datenempfang.
3. Systemen, bei denen sich Fehlverhalten durch andere Eingabereihenfolgen, Nutzung andere Eingabemenüs etc. umgehen lassen.
4. Systeme, die beim Ausschalten automatisch ihre Bearbeitungszustand sichern.
5. Systeme, die nach einer Fehlfunktion vom letzten gesicherten Zustand starten.
6. Versenden von Fehlerinformationen an die Firma, die das System entwickelt hat.

Aufgabe 3.15: Programmieren eines 3-Versionssystem

Die drei Berechnungsversionen für ein n-Versionssystem und eine Funktion für den Vergleich seien wie folgt als C-Unterprogramme vereinbart:

```
struct ergebnis fkt_Vers1(struct eingabe e);
struct ergebnis fkt_Vers2(struct eingabe e);
struct ergebnis fkt_Vers3(struct eingabe e);
int[27] Vergleich(struct ergebnis e1, struct ergebnis e2);
```

Schreiben Sie ein Programm

```
struct ergebnis fkt_3Vers(struct eingabe e);
```

das alle drei Ergebnisse berechnet und das Mehrheitsergebnis zurückgibt. Wenn es kein Mehrheitsergebnis gibt, soll ein Abbruch mit dem Funktionsaufruf

```
exit("Keine Mehrheitsergebnis");
```

erfolgen.

Literatur

6. Literatur

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

[2] Heidrun Dücker.
Ergebnisvalidierung und nebenläufige Hardwarefehlererkennung mittels systematisch erzeugter Diversität.
PhD thesis, Diss. Universität Karlsruhe Institut für Rechnerentwurf und Fehlertoleranz.

[3] K. Glerum.
Debugging in the (Very) Large: Ten Years of Implementation and Experince.
In SOSP, pages 11–14, 2009.

[4] 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.

[5] Jim Shore.
Fail fast.
In IEEE Software, pages 21–25, 2004.

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

[7] U. Voges.

Software-Diversität und ihre Modellierung - Software-Fehlertoleranz und ihre Bewertung durch Fehler- und Kostenmodelle. In Informatik-Fachberichte. Springer, 1989.