



Test und Verlässlichkeit (F3)
Foliensatz 3:
Kontrollen und Fehlerbehandlung
Prof. G. Kemnitz

Institut für Informatik, Technische Universität Clausthal
1. Juli 2014



Inhalt F3: Kontrollen und Fehlerbehandlung

Überblick

- 1.1 Problembeseitigung
- 1.2 PB im Betrieb
- 1.3 Aufgaben

Informationsredundanz

- 2.1 Fehlererkennende Codes
- 2.2 Prüfkennzeichen
- 2.3 Fehlerkorr. Codes
- 2.4 Hamming-Codes
- 2.5 RAID Systeme
- 2.6 Aufgaben

Formatkontrollen

- 3.1 Syntextest
- 3.2 Typ und Wertebereich

3.3 Signalüberwachung

3.4 Aufgaben

Wertekontrollen

4.1 Mehrfachber. & Vergleich

4.2 Diversität

4.3 Loop-Back Test

4.4 Probe

4.5 Aufgaben

Fehlerbehandlung

5.1 Fail-Safe/-Fast/-Slow

5.2 Neustart, Wiederholung

5.3 Fehlerisolation

5.4 Fehlertoleranz

5.5 Aufgaben



Überblick



Problembeseitigung



Kontrolle und Problembeseitigung

Der Schlüssel zu objektiv verlässlichen Systemen sind Kontrollen und eine Nachbesserungsiteration zur Beseitigung der dabei erkannten Probleme auf drei Ebenen:

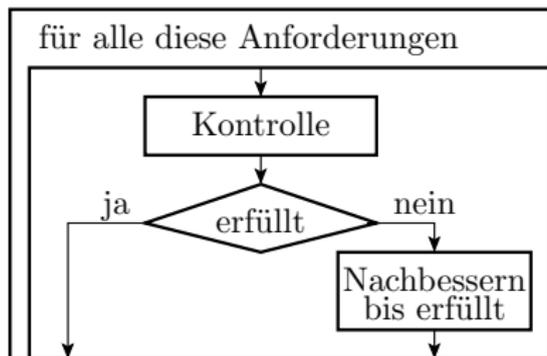
- während Entwurf und Fertigung (Fehlervermeidung),
- vor dem Einsatz und zur Wartung (Fehlerbeseitigung) und
- im laufenden Betrieb (Fehlertoleranz, Schadensvermeidung).

Zusammenstellung überprüfbarer Anforderungen:

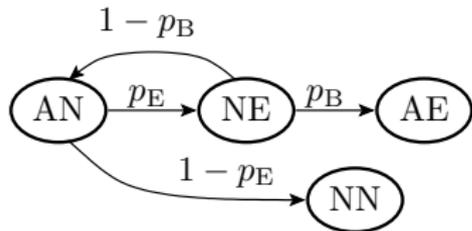
- für Entwurf und Fertigung
- für den Test
- während des Betriebs

Nachbesserung:

- Veränderung am System
- Änderung der Anforderung
- Änderung der Nutzung



Problembeseitigung als Markov-Kette



AN Anforderung nicht erfüllt.

NE Nicht erfüllte Anforderung erkannt.

NN Nicht erfüllte Anforderung nicht erkannt, Problem bleibt bestehen.

AE Anforderung erfüllt. Problem beseitigt.

p_E Erkennungswahrscheinlichkeit

p_B Beseitigungswahrscheinlichkeit

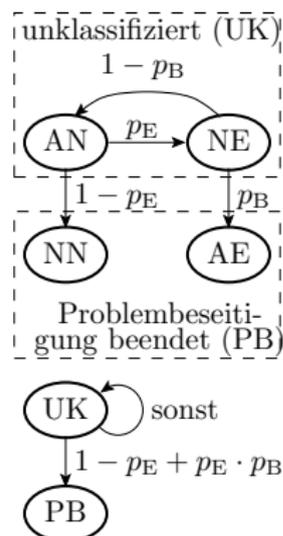
Zustand der Markov-Kette nach n Kontroll- und Beseitigungsschritten:

$$\begin{pmatrix} p_{AN} \\ p_{NE} \\ p_{NN} \\ p_{AE} \end{pmatrix} = \begin{pmatrix} 0 & 1 - p_B & 0 & 0 \\ p_E & 0 & 0 & 0 \\ 1 - p_E & 0 & 1 & 0 \\ 0 & p_B & 0 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Beispielsimulationen

Simulation der Wahrscheinlichkeiten p_{AE} (Problem erfolgreich beseitigt), p_{NN} (Problem nicht (mehr) erkennbar), $p_{AN} + p_{NE}$ (unklassifiziert) und $\frac{p_{AE}}{p_{NN} + p_{AE}}$ (wenn klassifiziert, dann beseitigt) nach jedem Doppelschritt mit $p_E = 99\%$ und $p_B = 20\%$:

n	$p_{AN} + p_{NE}$	p_{NN}	p_{AE}	$\frac{p_{AE}}{p_{NN} + p_{AE}}$
2	79.2	1.0	19.8	95.2
4	62.7	1.8	35.5	95.2
6	49.7	2.4	47.9	95.2
8	39.3	2.9	57.7	95.2
10	31.2	3.3	65.5	95.2
12	24.7	3.6	71.7	95.2
14	19.5	3.9	76.6	95.2
16	15.5	4.1	80.5	95.2
18	12.3	4.2	83.5	95.2



Beseitigungsaufwand und -wahrscheinlichkeit

Die Wahrscheinlichkeit, das ein erkanntes Problem nach einem Beseitigungsversuch beseitigt oder nicht mehr zu erkennen ist, beträgt:

$$p_{PB}(1) = 1 - p_E + p_E \cdot p_B$$

Nach m Beseitigungsversuchen beträgt sie:

$$p_{PB}(m) = 1 - (p_E \cdot (1 - p_B))^m$$

Die mittlere Anzahl der Beseitigungsversuche ist der Kehrwert der Beseitigungswahrscheinlichkeit je Versuch:

$$\bar{m} = \frac{1}{1 - p_E + p_E \cdot p_B}$$

Bei einer Erkennungswahrscheinlichkeit nahe 100% ist die Beseitigungswahrscheinlichkeit abschätzungsweise $p_B \approx 1/\bar{m}$, bei bei 1,25 ca. 80%. und bei 5 Versuchen ca. 20%.



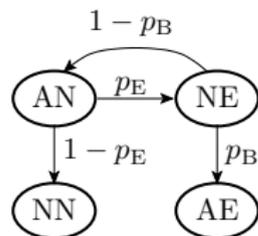
Erfolgswahrscheinlichkeit

Die Wahrscheinlichkeit, dass das betrachtete Problem nach einer Beseitigungsiteration nicht mehr existiert, beträgt unabhängig von der Anzahl der Beseitigungsversuche:

$$p_{AE} \setminus p_{PB} = \frac{p_E \cdot p_B}{(1 - p_E) + p_E \cdot p_B}$$

Für die Verlässlichkeit ist eine hohe Erkennungs- und eine moderate Beseitigungswahrscheinlichkeit wichtig. Viele

Beseitigungsiterationen ($p_B \ll 1$) erhöhen das Risiko, dass aus erkennbaren nicht erkennbare Probleme werden.



$p_B \setminus p_E$	80%	90%	95%	99%
20%	44,4%	64,3%	79,2%	95,2%
50%	66,7%	81,8%	90,5%	98,0%
80%	76,2%	87,8%	93,8%	98,8%



Problembeseitigungsiterationen in dieser Grundform und in erweiterten Formen gibt es auf allen Ebenen der Sicherung der Verlässlichkeit:

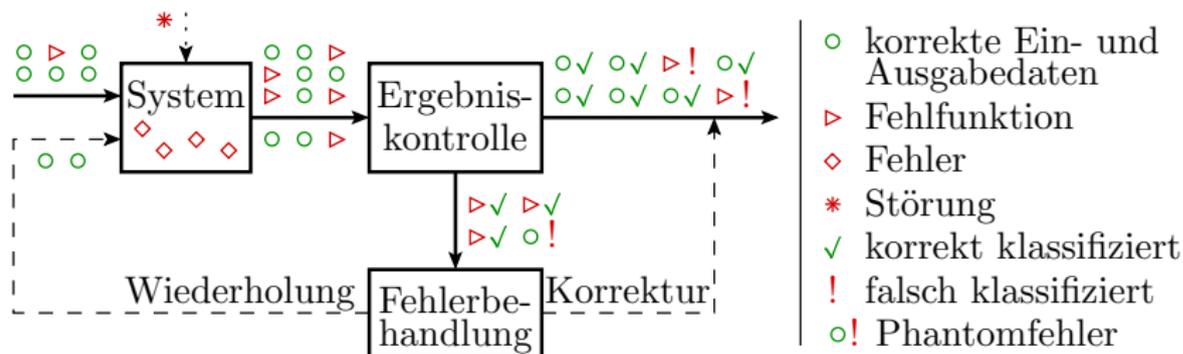
- Fehlervermeidung (Kontrolle der Entstehungsprozesse und Problembeseitigung),
- Test und Fehlerbeseitigung sowie
- Überwachung im laufenden Betrieb und Fehlerbehandlung.

Dieser Foliensatz wird sich im weiteren nur mit der letzten Ebenen, der Überwachung im Betrieb befassen.



PB im Betrieb

Problembeseitigung während des Betrieb



- Das System erhält fortlaufend Service-Anforderungen,
- reagiert mit richtigen, keinen oder falschen Ergebnissen.
- Kontrollen filtern als falsch erkannte Ergebnisse aus und lösen zur Problembeseitigung eine Fehlerbehandlung aus.
- Die Fehlerbehandlung kann eine Korrektur oder Wiederholung beinhalten.



Kontrollmöglichkeiten während des Betriebs

Die Kontrollmöglichkeiten während des Betriebs eines Informationsverarbeitenden Systems lassen sich einteilen in:

- Kontrolle der Datenformate der Service-Anforderungen und Ergebnisse,
- Kontrolle Ergebniswerte und Zustände,
- Überwachung von Zeitschranken,
- von den Service-Leistungen unabhängige Größen
(Versorgungsspannung, Stromverbrauch, Temperatur, ...)

Hinzu kommen zyklische Kontroll- und Wartungstests z.B. zur Prüfung der Konsistenz von Dateisystemen oder Datenbanken.



Who Checks the Checker

Eine Kontrollsystem ist auch ein informationsverarbeitendes System, das aus Eingaben Ausgaben bildet.

- Als reales System kann seine Ausgabe, das Kontrollergebnis auch durch Fehler, Störungen und Bedienfehler verfälscht sein.
- Das erfordert theoretisch eine Kontrolle der Kontrolle, ... (Endlosrekursion).

Eine rekursive Kontrolle der Kontrolle ist natürlich unrealistisch.
Praktische Lösungen:

- Kontrolle einfach halten und gründlich testen, damit sie deutlich zuverlässiger als der Rest des Systems ist.
- Selbst kontrollierende Kontrollsysteme,
- doppelte Kontrolle, mehrere unabhängige Kontrollen,
- ...



Reaktion auf erkannte Fehlfunktionen

Problemen, die während des Betriebs auftreten können, sind:

- Ausfälle der Versorgungsspannung, Übertemperatur, ...
- Eingabefehler,
- Service nicht verfügbar (Zeitschrankenüberschreitung),
- Ergebnis fehlerhaft (Format oder Wertefehler),
- Dateninkonsistenzen, ...

Vorrang bei der Problembehandlung während des Betriebs hat der Übergang in einen gefahrenfreien Zustand:

- bei Unterspannung Daten retten und ausschalten,
- bei Übertemperatur und Überstrom Leistung reduzieren/Lüftergeschwindigkeit erhöhen/...
- Festplatten haben heute einen Beschleunigungssensor, der einen freien Fall erkennt, um den Lese/Schreib-Kopf in eine sichere Position zu bringen.
- ...



Reaktion auf Eingabefehler, Zeitüberschreitung, ...

Ein IT-System sollte robust gegenüber Eingabefehlern sein, d.h. jede unzulässige Eingabe mit einer aussagekräftigen Fehlermeldung zurückweisen.

Auf Überschreitung von Zeitschranken ist meist die gewünschte Reaktion Bearbeitungsabbruch und das System für andere Service-Leistungen verfügbar halten.

Erkannte Fehlfunktionen sind zu protokollieren und einer Fehlersuche zugänglich zu machen (siehe später Foliensatz F4).

Fehlertolerante Systeme korrigieren zusätzlich fehlerhafter Ergebnisse. Dafür gibt es nur zwei Verfahren:

- Datenkorrektur nach Übertragung und Speicherung mit fehlerkorrigierenden Codes und
- Wiederholung von Berechnungen, Neuanforderung Übertragungen oder Eingaben.



Korrektur durch Wiederholung

Bei Übertragungsfehlern und fehlerhaften Ergebnissen von Systemen ohne Gedächtnis sind die ersten Korrekturversuche einfache Wiederholungen.

Bei Systemen mit Gedächtnis ist vor der Wiederholung ein fehlerfreier Zustand herzustellen, z.B. durch Neuinitialisierung oder Laden eines gesicherten fehlerfreien Zustands (Backups).

Fehler im Einsatz verursachen in der Regel nur für einen geringen Anteil der Bedatungen Fehlfunktionen und es gibt mehrere Wege, dieselbe Service-Leitung anzufordern. Eine (manuelle) Umformulierung der Service-Anforderung ist deshalb meist der nächste Versuch, wenn eine einfache Wiederholung nicht hilft.



Fehlertoleranz

Aufwändige Systeme haben Redundanzen:

- Reserve-Hardware zu Rekonfiguration bei Ausfällen,
- Informationsredundanz zum Erkennen und zu Korrektur von Datenfehlern (vergl. fehlerkorrigierende Codes) und
- diversitäre¹ Mehrfachberechnung mit Ergebnisauswahl.

Eine automatische Fehlerbehandlung verlangt Fehlerisolation, d.h. Verhinderung der Ausbreitung verfälschter Daten über das betroffene Teilsystem hinaus. Insbesondere dürfen die Teilsysteme, die das System in einen sicheren Zustand versetzen, Fehlfunktionen korrigieren, z.B. der Backup-Speicher, nicht selbst durch die Fehlfunktionen beeinträchtigt sein.

Ein System, das eigene interne Fehlfunktionen erkennt und automatisch korrigiert, ist fehlertolerant.

¹Berechnung auf verschiedenen Weise.



Aufgaben



Aufgabe 1.1: Fehlerbeseitigungsiteration

Es seien 1000 Anforderungen zu überprüfen. Bei 20% treten Probleme auf. Die Problembeseitigung erfordert im Mittel 3 Versuche. Wie groß muss die Erkennungswahrscheinlichkeit der Kontrolle mindestens sein, damit nach der Problembeseitigungsiteration im Mittel nicht mehr als zehn Anforderung problembehaftet sind?

Hinweis: Die gegebene mittlere Anzahl der Reparaturversuche und der indirekt gegebene Anteil der zu beseitigenden Probleme sind beides Funktionen der Beseitigungs- und Erkennungswahrscheinlichkeit. Die Erkennungswahrscheinlichkeit muss überschlagsweise fast eins sein, so dass sie in einer der beiden Gleichungen praktisch keinen Einfluss hat, so dass sich Beseitigungs- und Erkennungswahrscheinlichkeit nacheinander über jeweils eine Gleichung bestimmen lassen.



Aufgabe 1.2: Fehlerbeseitigungsiteration

Bei einer praktischen Fehlerbeseitigung, z.B. bei der Beseitigung von Programmfehlern, ist es nicht untypisch, dass neu Probleme in Form von anderen danach nicht mehr erfüllten Anforderungen entstehen. Es sei angenommen, dass jeder Beseitigungsversuch für jede der bis dahin erfüllten Anforderung mit einer Wahrscheinlichkeit von 10% ein neues Problem verursacht.

- 1 Wie viele Probleme entstehen unter Beibehaltung aller anderen Werte aus der Aufgabenstellung zuvor zusätzlich durch die Problembeseitigungsversuche?
- 2 Auf welchen Erwartungswert erhöht sich die Anzahl der Probleme, die nach der Beseitigungsiteration noch vorhanden sind, ohne von den Kontrollen erkannt zu werden?



Aufgabe 1.3: Anzahl der nicht erkannten Fehlfunktionen

Eine Kontrolle habe einer Erkennungswahrscheinlichkeit $p_E = 99\%$. Es werden

- 1 1000
- 2 10000
- 3 100000

Service-Leistungen kontrolliert, von denen 10% fehlerhaft sind. Wie hoch ist die Wahrscheinlichkeit, das weniger als 95% der fehlerhaften Service-Leistungen erkannt werden?

Hinweise: Bei einem Erwartungswert kleiner zehn ist die Anzahl der nicht erkannten falschen Service-Leistungen näherungsweise Poisson- und bei einem Erwartungswert ab zehn normalverteilt.

Aufgabe 1.4: Beispiele für die Fehlerbehandlung

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

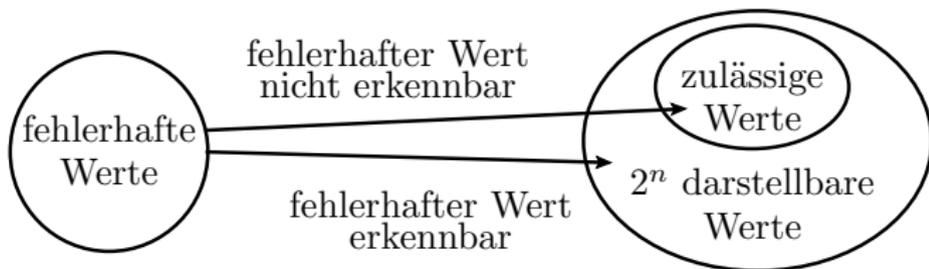
- 1 Zeitüberwachung mit Service-Abbruch bei Zeitüberschreitung.
- 2 Wiederholungsanforderung nach fehlerhaftem Datenempfang.
- 3 Systemen, bei denen sich Fehlverhalten durch andere Eingabereihenfolgen, Nutzung andere Eingabemenüs etc. umgehen lassen.
- 4 Systeme, die beim Ausschalten automatisch ihre 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.



Informationsredundanz

Informationsredundanz

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



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



2. Informationsredundanz

Anwendungsbeispiel Rechtschreibtest: Wort im Wörterbuch enthalten?

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

Wenn fehlerhafte Werte sich gleichmäßig auf zulässige und unzulässige Werte abbilden, ist die erzielbare Erkennungswahrscheinlichkeit²:

$$p_E \approx 1 - \frac{N_{Zul}}{2^n} \quad (1)$$

(N_{Zul} – Anzahl der zulässigen Werte). Ein Rechtschreibtest erkennt Größenordnung 80% der Rechtschreibfehler, obwohl der Anteil der Zeichenfolgen, die zulässige Worte darstellen, verschwindend gering ist. Viel schlechtere Erkennungswahrscheinlichkeit als Gl. 1:

$$p_E \gg 1 - \frac{N_{Zul}}{2^n}$$

²Wahrscheinlichkeit, dass fehlerhafte Werte als fehlerhaft erkannt werden.



2. Informationsredundanz

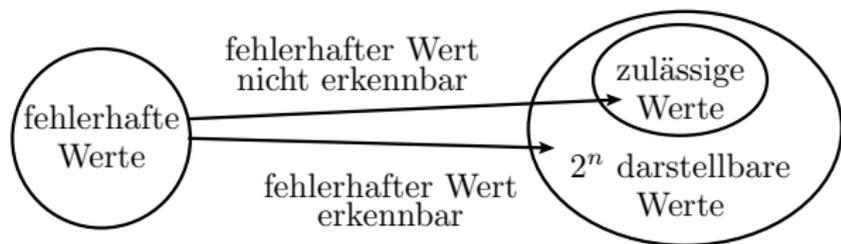
Bei der Datenspeicherung und Übertragung sind Bitfehler äußerst selten. Wenn die Daten so auf Datenobjekte aufgeteilt werden, dass jede Verfälschungsursache, z.B. ein Störimpuls, nur in jedem Datenobjekt ein Bit verfälscht, genügt der Nachweis aller Einzelbitfehler, um fast alle Datenverfälschungen zu erkennen. Einzelbitfehler sind bereits mit einem Code mit $N_{\text{Zul}} = 50\% \cdot 2^n$ zulässigen Codeworten nachweisbar (vergl. später Paritätstest Folie 52). Viel bessere Erkennungswahrscheinlichkeit als Gl. 1:

$$p_E \ll 1 - 50\%$$



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:

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



Arithmetische Codes

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

$$s = 34562134 \cdot x$$

Von s sind nur die Vielfachen von 34562134 gültig. Die Anzahl der zulässigen Werte ist mindestens 34562134 mal so groß wie die der gültigen Werte. Da zu erwartende Verfälschungen kaum vorzugsweise auf Vielfache von 34562134 abgebildet werden, ist die Erkennungswahrscheinlichkeit

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

fast eins. Bei sehr großen unbekanntenen Primzahlen als Multiplikatoren ist es selbst vorsätzlich kaum möglich, gültige Codeworte so zu verfälschen, dass andere gültige Codeworte entstehen.

Daher auch für kryptographische Verschlüsselungen geeignet.



Zyklische Codes

Codierung erfolgt durch die Multiplikation mit einer Konstanten, allerdings nicht arithmetisch, sondern in einer in Hard- oder Software viel einfacher ausführbaren Form, nämlich modulo-2:

Codierung

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

Decodierung

$$\begin{array}{r}
 100011100100111 : 10011 \\
 \oplus 10011 \\
 \hline
 10110 \\
 \oplus 10011 \\
 \hline
 10101 \\
 10011 \\
 11000 \\
 \oplus 10011 \\
 \hline
 10111 \\
 \oplus 10011 \\
 \hline
 10011 \\
 \oplus 10011 \\
 \hline
 00000 \text{ (Rest null, fehlerfrei)}
 \end{array}$$



Mathematisch werden die zu multiplizierenden Faktoren als Polynome repräsentiert:

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

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

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

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

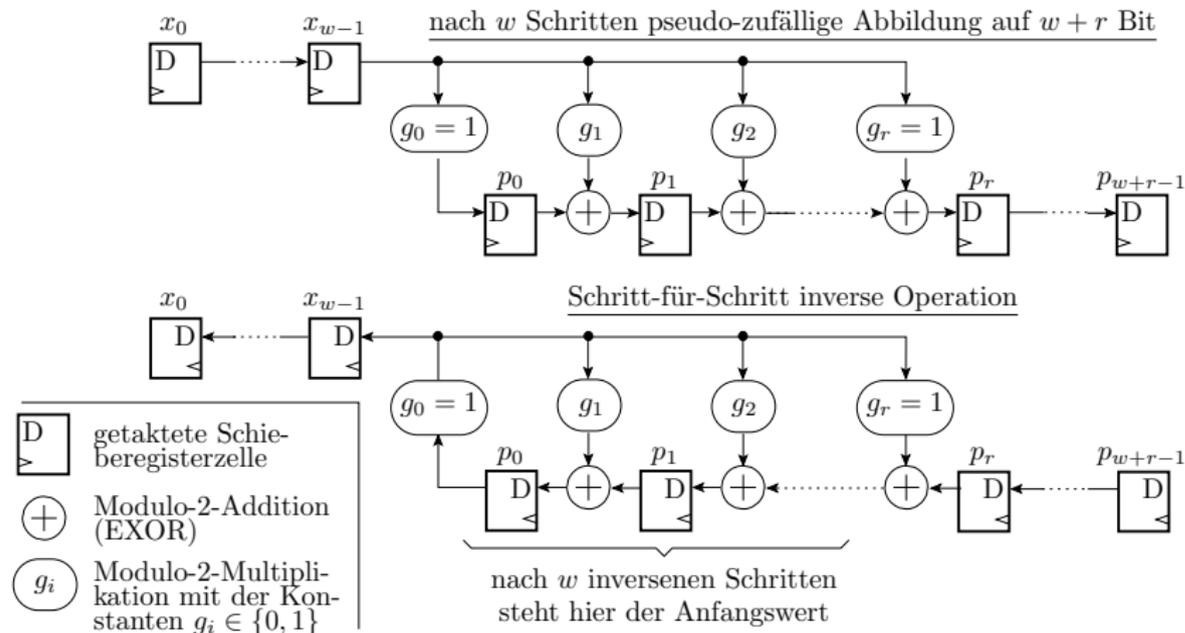
$$(x^{14} \oplus x^{10} \oplus x^9 \oplus x^8 \oplus x^5 \oplus x^2 \oplus x \oplus 1) : (x^4 \oplus x \oplus 1) = x^{10} \oplus x^7 \oplus x^5 \oplus x^3 \oplus x^2 \oplus 1$$

liefert ohne Rest das Polynom der Originalfolge.



Linear rückgekoppelten Schieberegistern

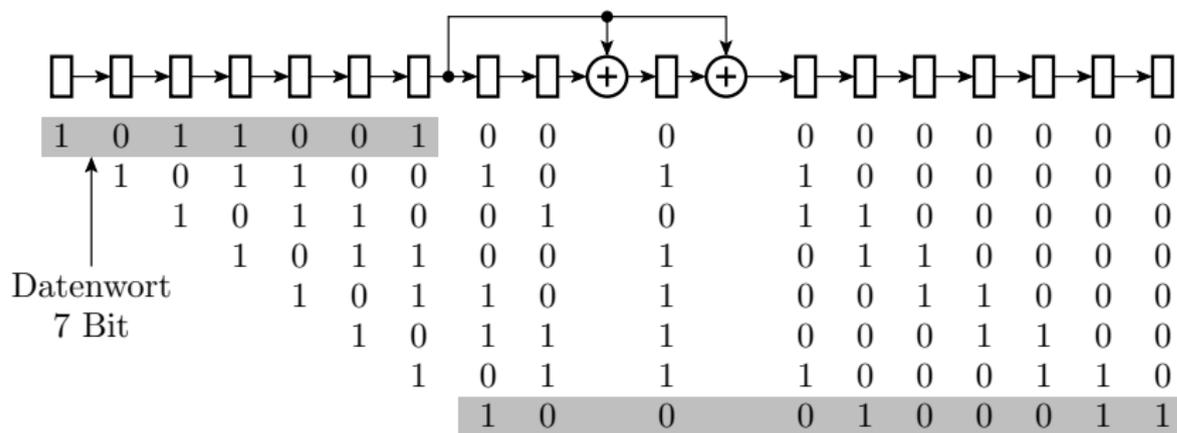
Die Codierung und Decodierung erfolgt mit Schieberegistern.



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



Beispiel für die Codierung

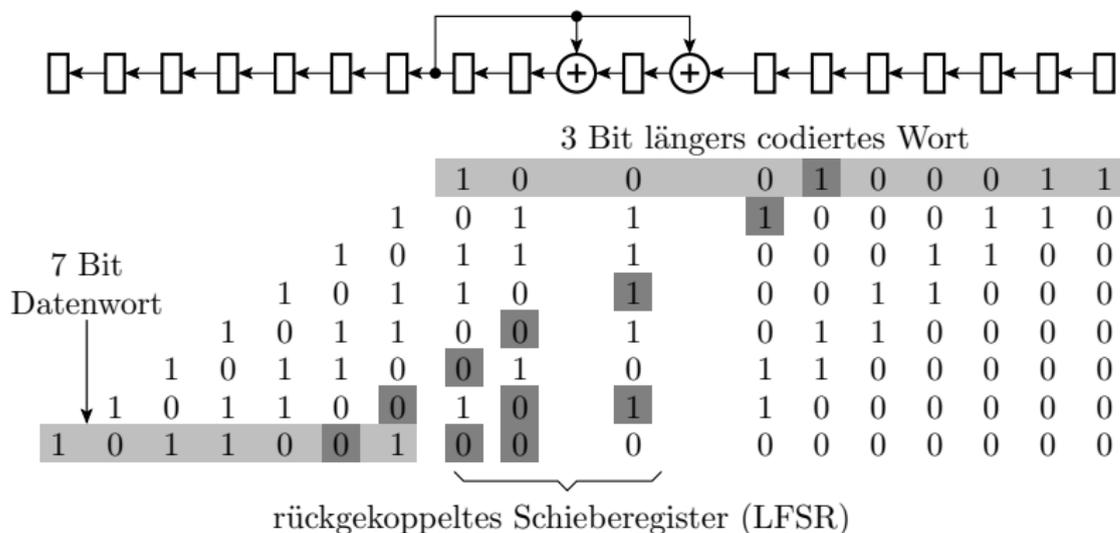


3 Bit längeres codiertes Wort

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



Rückgewinnung und Kontrolle



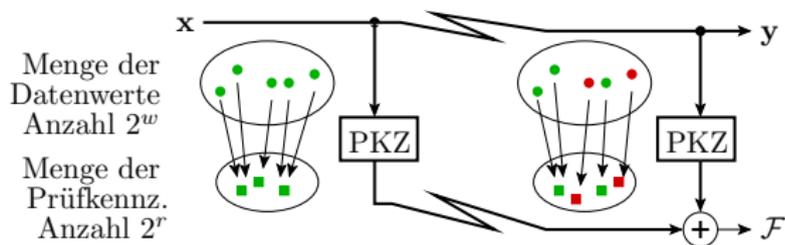
■ Fortpflanzung einer Bitverfälschung

- Eine Bitverfälschung verursacht mit der Wahrscheinlichkeit $p_E = 1 - 2^{-3}$ 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.:

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



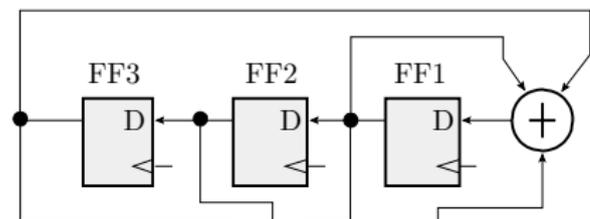
Prüfsummen

Prüfkennzeichenbildung durch Aufsummierung der zu übertragenden oder zu speichernden Daten (arithmetisch, bitweises EXOR, ...).

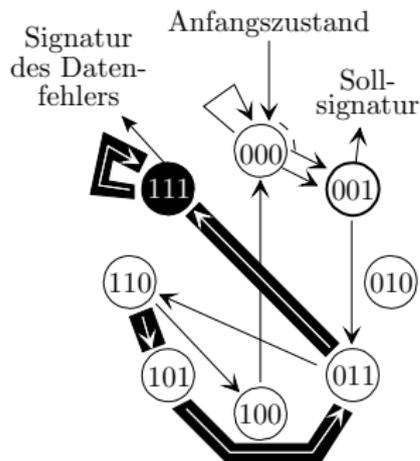
einfache Genauigkeit	doppelte Genauigkeit	bitweises EXOR
1011 11	1011 11	1011
0010 2	0010 2	0110
1101 13	1101 13	1101
0100 4	0100 4	1100
(1) 11110 30	0001 11110 30	1100

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

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	

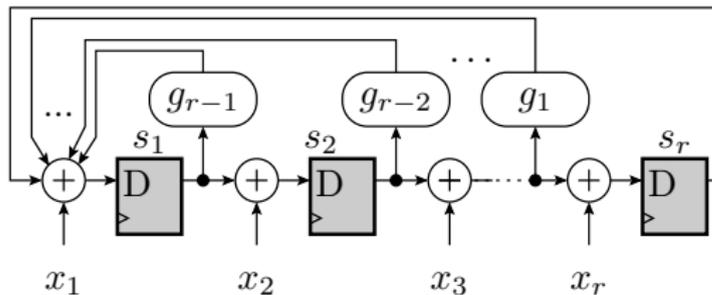


■ Veränderung durch einen Datenfehler

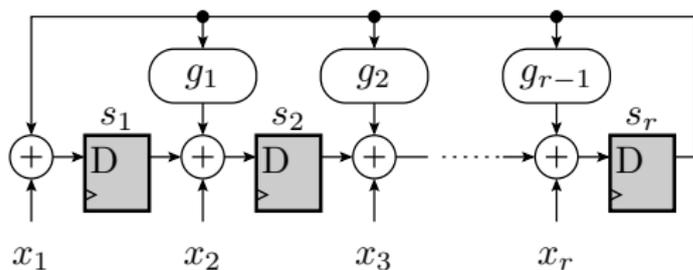
Das Prüfkennzeichen wird wie bei der CRC-Decodierung mit einem linear rückgekoppeltes Schieberegister (LFSR) gebildet. Im Beispiel hat das LFSR im Gegensatz zur Polynomdivision Folie 8 zentrale Rückführung. Abbildung auch pseudo-zufällig.

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 Zyklusstruktur³.

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

³Zyklusstruktur bei Einspeisung einer Folge aus nur Nullen.



Experiment zur Fehlererkennungssicherheit mit LFSRs

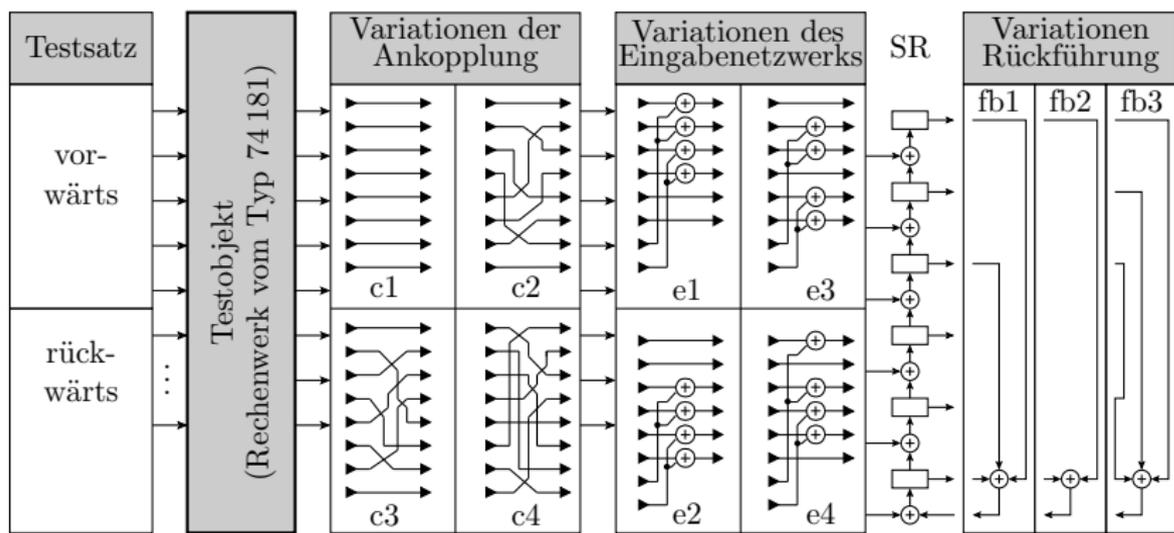
Es ist schwer zu glauben, dass

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

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

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



Wenn die Theorie stimmt, müsste die Anzahl der maskierten Fehler

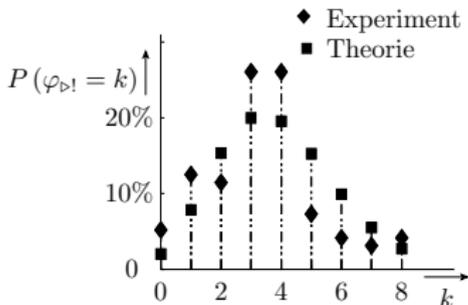
$$N = \sum_{i=1}^{250} X_i$$

binomialverteilt sein:

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

Anzahl der maskierten Fehler

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

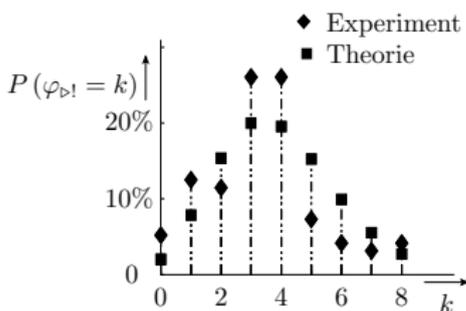


Die experimentell bestimmte Maskierungswahrscheinlichkeit scheint sogar unwesentlich kleiner als 2^{-6} zu sein.



Anzahl der maskierten Fehler

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



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

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

- [GrWil...] Primitive Rückführungen seien viel besser als andere. Mysteriöse Beweisführung, unglaubliches Ergebnis, aber »Best Paper Award 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.

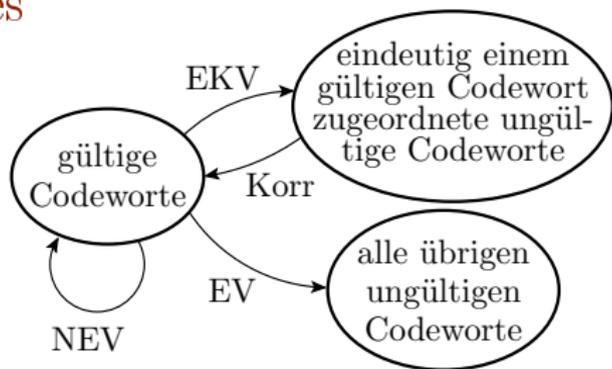


Fehlerkorr. Codes



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:

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

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



Beispiel: Korrektur von Einzelbitfehler

Anzahl korrigierbare Codeworte je gültiges Codewort gleich Bitanzahl:

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

Mindestbitanzahl:

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

Für $N_{\text{CWG}} = 256$ Bit:

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

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

Probe:

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

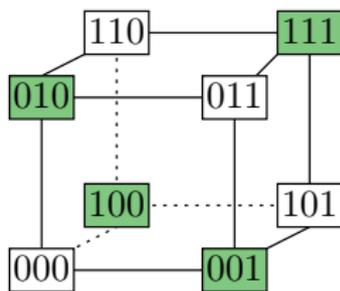


Hamming-Codes



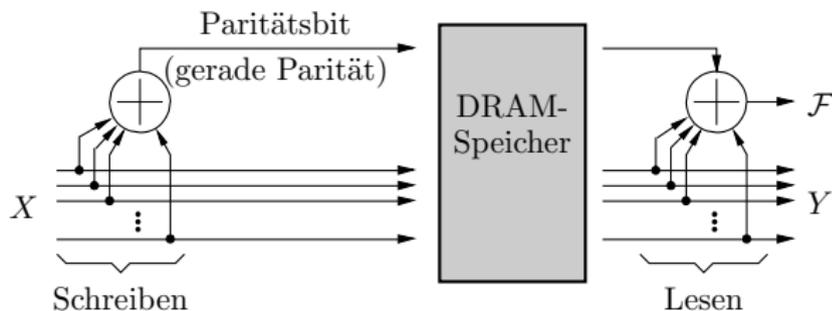
Hamming-Distanz

Die Hamming-Distanz ist die Zahl 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)



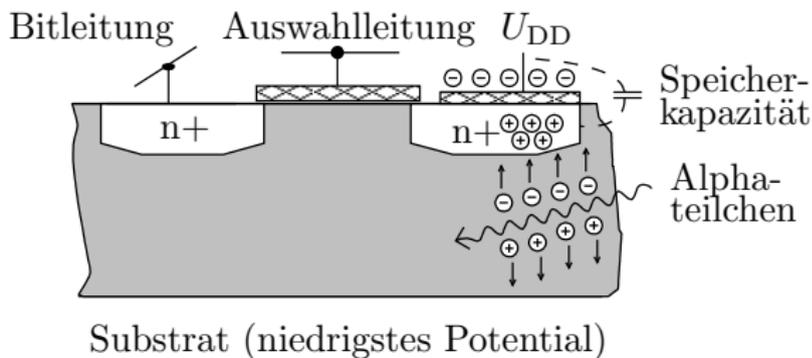
Einzelprüfbit, modulo-2 Summe (EXOR-Verknüpfung):

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

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

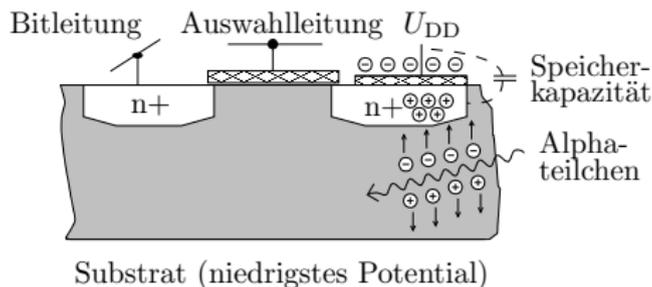
- Erkennt jede ungeradzahlige Anzahl von Bitverfälschung.
- Wenn bei Verfälschungen geradzahlige und ungeradzahlige Bitfehler gleichhäufig auftreten: $p_E \approx 50\%$
- Überwiegend Einzelbitfehler: $p_E \gg 50\%$

Paritätstest für DRAMs und Speicherriegel



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

- Energie eines Alpha-Teilchen: 5 MeV. Energieverlust bei der Generierung eines Elektronen-Loch-Paares $\approx 3,6 \text{ eV} \Rightarrow$ Generierung von $\approx 10^6$ Ladungsträgerpaaren. Reichweite $\approx 89 \mu\text{m}$. gespeicherte Ladung $\approx 10^5$ Ladungsträger. Datenverlust einer oder mehrerer benachbarter Zellen möglich.
- Mittlerer Zeitabstand zwischen zwei Datenverfälschungen Stunden. Gleichzeitige Verfälschung durch zwei Alphateilchen unwahrscheinlich.
- Geometrische Trennung der Zellen eines Datenworts (getrennte Schaltkreise oder Speicher­matritzen) \Rightarrow Einzelbitverfälschung je gelesenes Datenwort.
- 100%iger Nachweis durch Paritätskontrolle.





Kreuzparität (Fehlerkorrigierender Paritätscode)

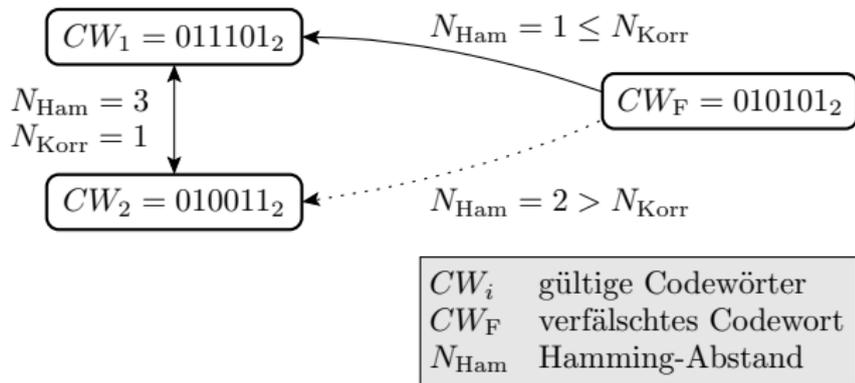
Daten sind in einem 2-dimensionalen Array organisiert. Paritätsbildung für alle Zeilen und Spalten. Erlaubt Lokalisierung und Korrektur von 1-Bit Fehlern. Einsatz in redundanten Festplatten-Arrays (RAID 3 und RAID 5).

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



1-Bit fehlerkorrigierende Hamming-Codes

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



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 des Codes

Beispielrezept:

- Zusammensetzen des Gesamtcodeworts aus einem Wort mit minimaler Bitanzahl

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

und einem

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

Prüfkennzeichen, das aus diesem 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: $w = 8$, $r = 4$, $\mathbf{q} = q_3q_2q_1q_0$

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

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

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

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

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:

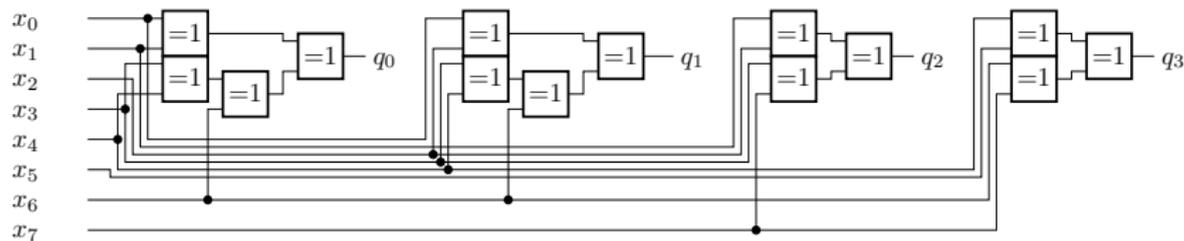
$$\begin{aligned}\Delta q_0 &= b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} \\ 0 &= q_0 \oplus x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6 \\ q_0 &= x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6\end{aligned}$$

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 $\mathbf{q} = q_3 \dots q_0$ der bitweise EXOR-Verknüpfung des empfangen und des im Empfänger gebildeten Prüfzeichens. Invertierung des verfälschten Bits (als Case-Anweisung in VHDL):

```

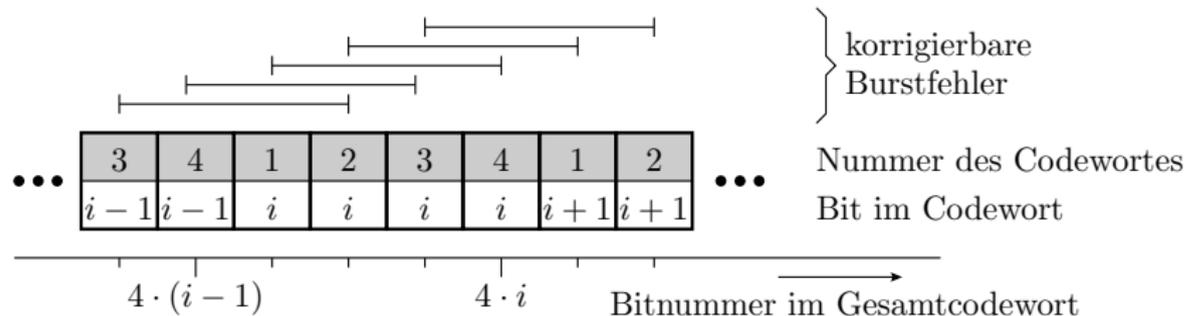
case (q_Empf xor q_berechnet) is
  when "0011" => x_korr(0) <= not x(0);
  when "0101" => x_korr(1) <= not x(1);
  ...
  when others => null;
end case;

```

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



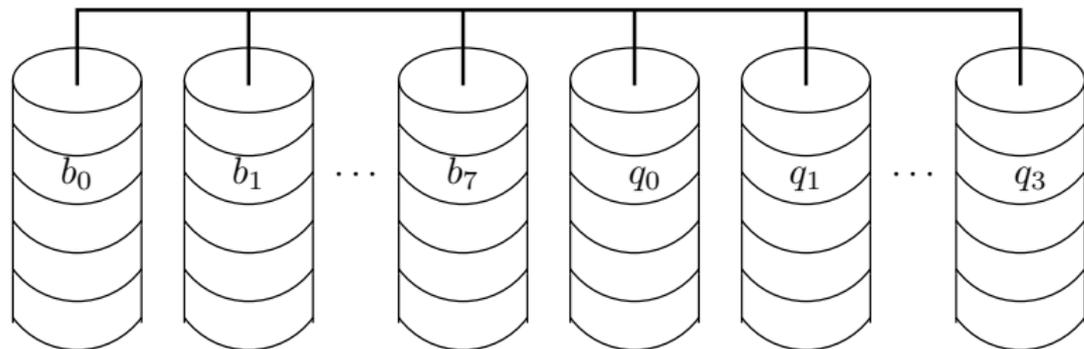
RAID, RAID Level 1

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

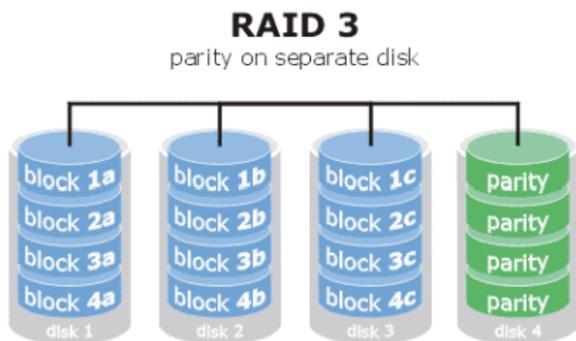
Bei RAID Level 2 werden die Daten in einem 1-Bit fehlerkorrigierenden den Hamming-Code gespeichert, und zwar jedes der w Daten- und der r Kontrollbits auf einer eigenen Platte, z.B. $w = 8$ Datenbit- und $r = 4$ Kontrollbitplatten. Im Vergleich zu RAID 1 werden statt der doppelten Plattenanzahl nur 50%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



Gilt als aufwändig und ungebräuchlich.

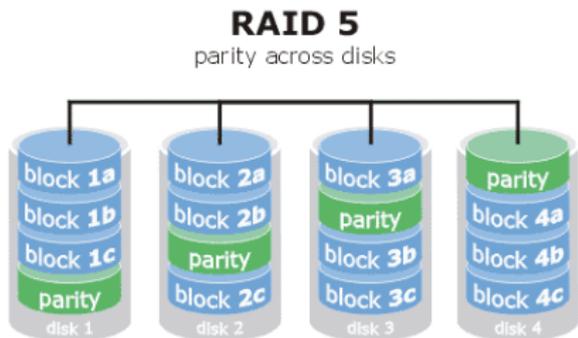
RAID Level 3

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.





Ein RAID ist kein Ersatz für ein Backup:

- That back-up will come in handy if all drives fail simultaneously because of a power spike. It is a safeguard if the storage system gets stolen.
- Back-ups can be kept off-site at a different location. This can come in handy if a natural disaster or fire destroys your workplace.
- The most important reason to back-up multiple generations of data is user error. If someone accidentally deletes some important data and this goes unnoticed for several hours, days or weeks, a good set of back-ups ensure you can still retrieve those files.



Aufgaben



Aufgabe 2.1: Arithmetischer Code

- 1 Bilden Sie für den Bitvektor

110010001000011101

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 2.2: 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 2.3: 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.

1011001001101000	<input type="checkbox"/>	Längsparität																				
1100001110010011	<input type="checkbox"/>																					
0110010010101101	<input type="checkbox"/>																					
1000100001100101	<input type="checkbox"/>																					
1101001011010011	<input type="checkbox"/>																					
1101000010011110	<input type="checkbox"/>																					
1010011000010101	<input type="checkbox"/>																					
1011010010100110	<input type="checkbox"/>																					
<table border="1" style="display: inline-table; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 20px; height: 20px;"></td> </tr> </table>																						Querparität



Aufgabe 2.4: 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.

1011010011000010	<input type="checkbox"/>
1011011010010100	<input type="checkbox"/>
1001011010010101	<input type="checkbox"/>
1000010011111110	<input type="checkbox"/>
1101101100110100	<input type="checkbox"/>
0010110000110111	<input type="checkbox"/>
0101011001000001	<input type="checkbox"/>
1100100010011000	<input type="checkbox"/>
<u>01111101111100111</u>	

Längsparität

Querparität

a)

1001011100110011	<input type="checkbox"/>
0101011101000101	<input type="checkbox"/>
0011001011010010	<input type="checkbox"/>
1111000010011111	<input type="checkbox"/>
1001101101100100	<input type="checkbox"/>
1000010110000111	<input type="checkbox"/>
0000101011001001	<input type="checkbox"/>
0011100100010000	<input type="checkbox"/>
<u>011111100110001</u>	

Längsparität

Querparität

b)



Aufgabe 2.5: 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 2.6: Hamming-Code

- 1 Bilden Sie für den ab Folie 57 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:

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

$$q_1 = x_0 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_6$$

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

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



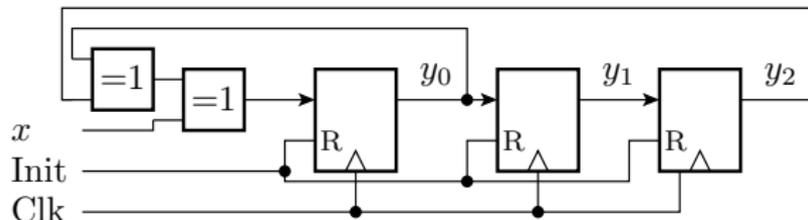
Bitzuordnung mit Beispiel:

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 2.7: 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 $\mathbf{y} = y_2y_1y_0$ hat die Datenfolge »1001100101111010« bei Abbildung beginnend mit dem höchstwertigen Bit. Startwert 000. Füllen Sie dazu die Tabelle in der Abbildung aus.

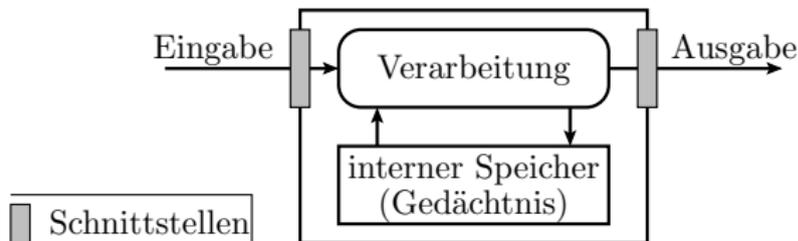


Formatkontrollen



Service-Modell

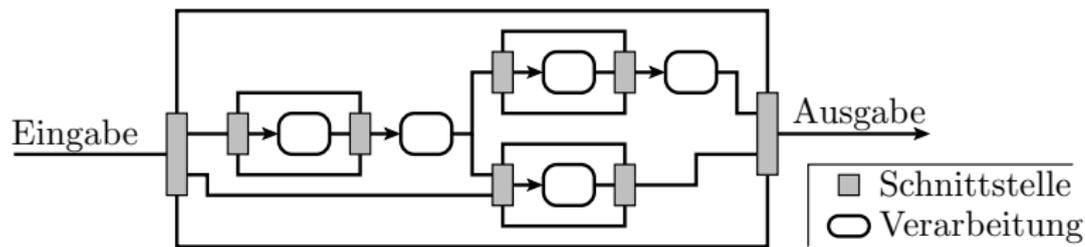
Laut F1, Abschn. 1.2 ist ein Service ein Berechnungsablauf, der aus Eingaben Ausgaben bildet. Die Ein- und Ausgaben sind ganz allgemein bedatete Objekte mit einem auf die Art des Services abgestimmten Format. Ein Service kann, aber muss kein Gedächtnis haben.



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 »erben« die komplexen Service-Leistungen die Fehler, Fehlfunktionen, Fehlerdichten, Ausfälle etc. ihrer Teil-Service-Leistungen.



Schnittstellen sind bevorzugte Kontrollpunkte, insbesondere

- für den Test der Teil- und Gesamtsysteme,
- die Fehlerlokalisierung und

die Fehlerisolierung in fehlertoleranten Systemen.



Formate und Kontrollen

An den Schnittstellen zwischen Teil-Service-Leistungen können kontrolliert werden:

- die Daten,
- Aufwandskenngrößen wie Antwortzeiten, ... und
- das Format.

Im laufenden Betrieb werden hauptsächlich die Formate und Antwortzeiten überwacht. Formatkontrollen sind Tests auf Zulässigkeit. Ein- und Ausgaben mit einem falschen Format sind fehlerhaft. Daten mit korrektem Format sind zulässig, müssen aber nicht richtig sein.

Die sichersten Formatkontrollen bieten fehlererkennende Codes und Prüfkennzeichen. Mit ihrer Erkennungssicherheit von $p_E = 1 - 2^{-r}$ (r – Anzahl der zusätzlich zu speichernden oder zu übergebenden Bits, vergl. Gl. 29 und 37).



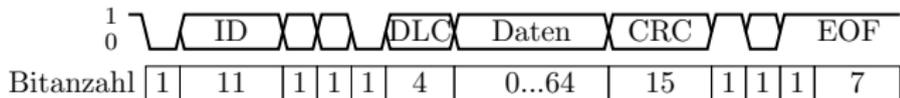
Datenpaktfomate mit Prüfkennzeichen

Datensicherungsschicht von Ethernet-Paketen:

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

- Anteil zusätzlicher Bits 0,3...10%
- Fehlererkennungssicherheit $p_E = 1 - 2^{-32} \approx 1 - 2 \cdot 10^{-10}$

CAN-Bus⁴-Nachricht:



ID Nachrichtennummer CRC Prüfkennzeichen (CRC)
 DLC Datenlängencode EOF Ende des Datenrahmens

- Anteil zusätzlicher Bits 13...34%
- Fehlererkennungssicherheit $p_E = 1 - 2^{-15} \approx 1 - 3 \cdot 10^{-5}$

⁴Einsatz z.B. als lokaler Bus zur Vernetzung von Steuergeräten in Autos.



3. Formatkontrollen

Fehlererkennende Codes und Prüfkennzeichen haben verfahrensbedingte Grenzen:

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

Weitere Möglichkeiten für Formatkontrollen:

- Syntaxtest: Definition der Formate manueller Eingabedaten als formale Sprache. Kontrolle mit spracherkennenden Automaten.
- Typ- und Wertebereichskontrollen der Ein- und Ausgaben von Verarbeitungsfunktionen.
- Überwachung der Zeit- und Wertetoleranzfenster von Signalen.



Syntaxtest

Formale Sprachen

Eine formale Sprache definiert zulässige Worte durch Textersetzungsregeln:

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

Beispiel:

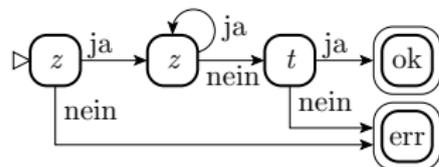
- Anweisung \Rightarrow Zuweisung | if_anweisung | ...
- Zuweisung \Rightarrow Variable = Ausdruck ;
- Ausdruck \Rightarrow uk_Ausdruck | (uk_Ausdruck)
- uk_Ausdruck \Rightarrow Konst. | Variab. | Ausdruck OP Ausdruck
- ...



Spracherkennender Automat

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

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



Band ... 1 3 5 □ 5 8 k ...	erwartet	Wert
Start 1	↑	z✓ 1
	↑	z✓ 13
	↑	z✓ 135
	↑	z (nein)
	↑	t✓ 135
Endzustand	↑	ok 135
Start 2	↑	z✓ 5
	↑	z✓ 58
	↑	z (nein)
	↑	t (nein)
Endzustand	↑	err ungültig

- ▷ Startzustand
- z Ziffer erwartet
- t Trennzeichen erwartet
- ✓ zulässiges Zeichen
- nein kein zulässiges Zeichen
- ok Wort erfolgreich abgeräumt
- err Eingabefehler



Mit den einfachen Ersetzungsregeln »darf sein«, »darf ein- oder n -mal vorkommen« lassen sich viele Daten und auch Programme, die ja letztendlich auch nur Daten sind, beschreiben.

Ein Programme 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 Fehler durch Menschen bei der Eingabe und Datenerfassung.



Standardisierte Sprachen

Sprachen für die manuelle Datenerfassung:

- Programmiersprachen (C, Java, ...)
- XML (E**x**tensible M**a**rku**p** L**a**nguage) zur Darstellung hierarchischen Darstellung strukturierter Daten in Form von Textdateien.
- CVS: Beschreibung tabellarischer Daten.

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



Typ und Wertebereich



Datentypen

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

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

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



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;
```

Wochentag $\in \{1, 2, 3, 4, 5, 6, 7\}$; 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
```

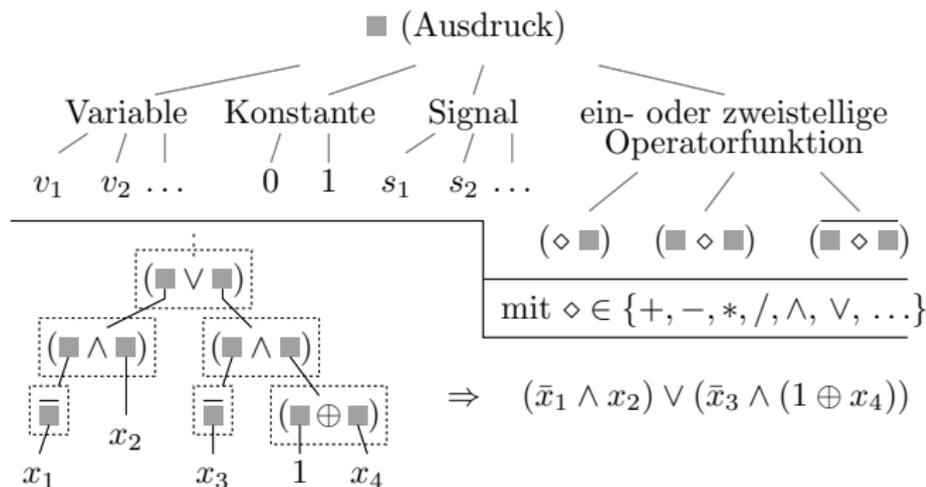
- Kontrollen bei Signalzuweisungen

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

Gleicher Typ des Ausdruck *W* und des Signal. Ausdruck t_d muss Typ TIME haben; $t_d \geq 0$.



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



- Arithmetische Operatoren (+, -, ...) verknüpfen identische Zahlentypen⁵. Ergebnis Operandentyp.

⁵Ausnahme »**« (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, ...)
- $BS \Rightarrow BZ\{, BS \Rightarrow BZ\}$ 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:** $\text{apfel_2} + 11 \mapsto \text{tAepfel}$; typengleich mit apfel_1 ;
Ergebnis 12, nicht darstellbar
 - A2:** $\text{apfel_1} * \text{apfel_2} \mapsto \text{tAepfel}$; 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

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

(a – Feldanfang; g – Elementgröße); 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

Es ist schwierig, Erkennungswahrscheinlichkeiten zu garantieren. Die Methode über die Größe der Mengen zulässiger und darstellbarer Werte ist in der Regel viel zu optimistisch.

Beispiel: Darstellung des Alters einer Person mit einer 32-Bit-Integer-Variable:

- zulässige Werte: 0 bis 120 (Anzahl 121)
- darstellbare Werte: -2^{31} bis $2^{31} - 1$ (Anzahl 2^{32}).

Erkennungswahrscheinlichkeit bei einer fehlererkennenden (zufälligen) Codierung:

$$p_{\text{FF}} \approx \frac{2^{32} - 120}{2^{32}} = 99,99 \dots \%$$

Die Erkennungswahrscheinlichkeit, geschätzt durch Zählen der aufgetretenen und erkannten Verfälschungen der Variablen wird eher unter 50% liegen.



Fehlerannahmen:

- Verwechslung mit dem Alter einer anderen Person: meist falsch, immer zulässig, nie nachweisbar.
- Verwechslung mit der Hausnummer: meist falsch, oft zulässig⁶, selten nachweisbar.
- Verwechslung mit einem beliebigen Wert im Rechner-Speicher: Kleine positive Datenwerte treten überproportional häufig auf. Nachweishäufigkeit deutlich unter 99%.

Es ist allein schwierig, eine repräsentative Menge zu erwartender Fehler zusammenzustellen und deren Häufigkeit zu wichten.

Es gibt offensichtlich Verbesserungspotential gegenüber der Nutzung der Datenbereiche ab oder um null in Bezug auf die Fehlererkennungswahrscheinlichkeit von Wertebereichstests:

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

⁶Hausnummern sind immer größer null und meist kleiner 119.

Fehlererkennender Code für Aufzählungstypen

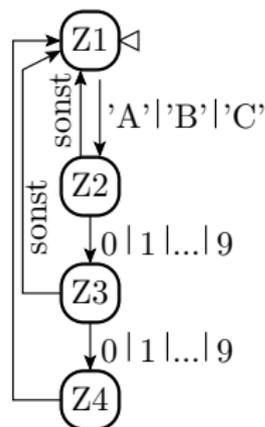
Für Aufzählungstypen, für die es nur die Operationen Zuweisung und Vergleich gibt, bieten sich fehlererkennende Codes an, d.h. ihre Darstellung mit zufälligen Werten innerhalb einer viel größeren Menge möglicher Werte. Statt der üblichen Zustandskodierung

```
#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.





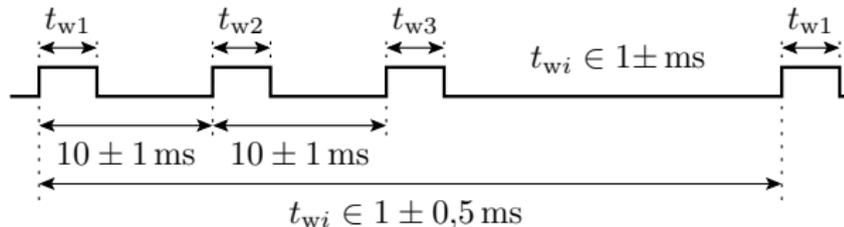
Signalüberwachung



Auf der Schaltungsebene kommunizieren IT-Systeme über Signale. Ein Signal ist ein zeitlicher Werteverlauf eine 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.

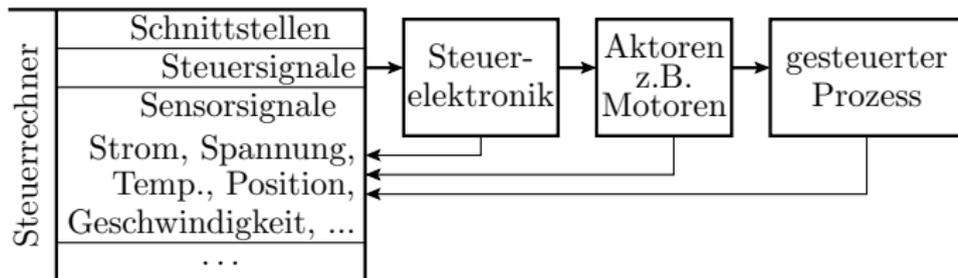
Das nachfolgende dargestellte PWM-Signal besteht aus einer Pulsfolge, bei denen Low-Pegel, der High-Pegel, der Pulsabstand und die Pulsbreite alle in einem bestimmten und damit kontrollierbaren Toleranzbereich liegen müssen. Die Information steckt typisch in der Pulsbreite, dem Abstand oder beidem.





Überwachung analoger Signale

Analoge Signale hat man im Wesentlichen als Sensoreingaben.



- Bereichsüberwachung.
- Überwachung von Pulsbreiten.

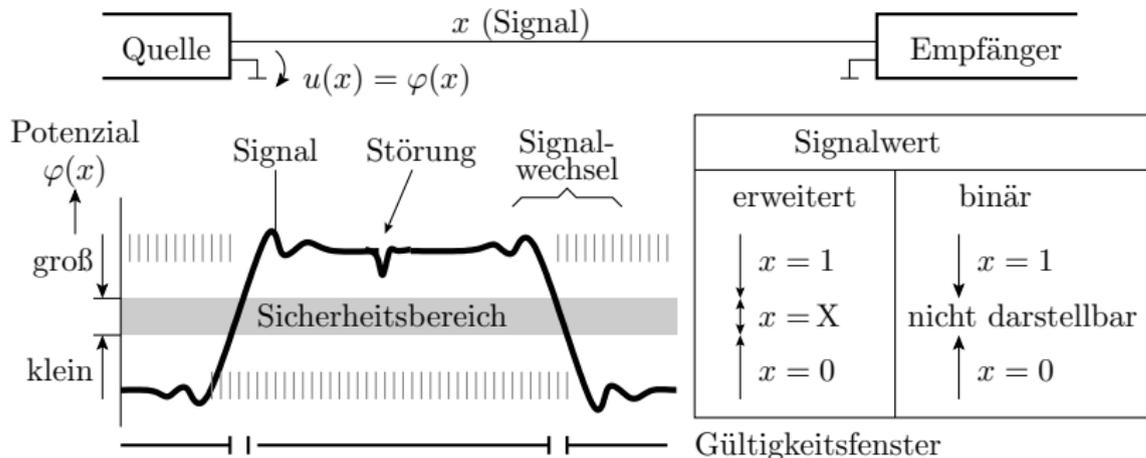
Die Überwachung komplexer Merkmale, z.B. der Stromsignaturen an Schrittmotoren auf Schrittfehler, erfolgt erst nach der Digitalisierung und ist z.T. sehr rechenintensiv⁷.

⁷Bei mechatronischen Systemen ist Übergang zwischen Zielfunktion und Kontrollfunktion zur Sicherung der Verlässlichkeit oft fließend.



Digitale vs. analoge Signale

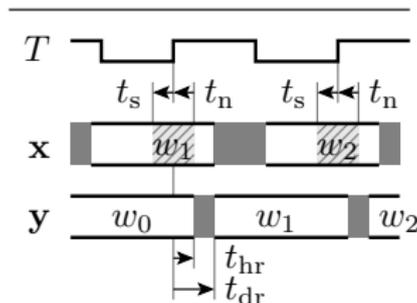
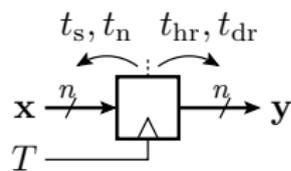
Die Digitalisierung schafft einen Zwischenbereich zwischen den gültigen Werten. Jedes Gatter im Signalfluss korrigiert alle tolerierbaren Eingabeverfälschungen, die z.B. durch Rauschen und andere Störungen entstehen. Erweiterung um eine Erkennung ohne Korrektur ist möglich.





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



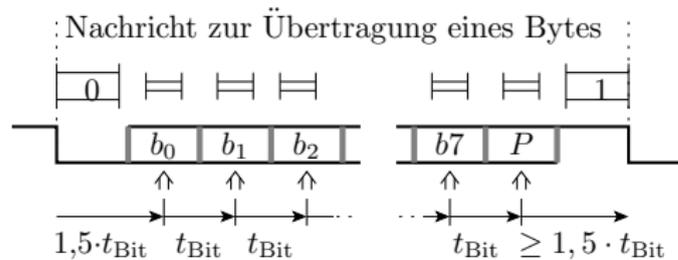
Bei analogen Signalen summieren sich die Verfälschungen durch Störungen, Rauschen, ... Die nachträgliche Trennung der Nutzsignale von überlagernden (zufälligen) Störsignalen ist schwierig, aufwändig, bzw. nur begrenzt möglich.

Deshalb erfolgt die Informationsverarbeitung mit Ausnahme der Wertaufnahme 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).



Kontrollmöglichkeiten:

0	muss 0 sein
1	muss konstant sein
1	muss 1 sein
P	muss $b_0 \oplus \dots \oplus b_7$ sein

↑ | Abtzeitpunkte, ungültig

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



Aufgaben

Aufgabe 3.1: Mikrorechner Ein- und Ausgabe

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

$$\langle \text{NTyp} \rangle = \text{'U' | 'V' | 'W'}$$

gefolgt von zwei Hexadezimalziffern:

$$\langle \text{NTyp} \rangle = \text{'0' | '1' | \dots | 'a' | 'b' | \dots | 'f'}$$

- 1 Beschreiben Sie dieses Eingabeformat als formale Sprache mit den Ersetzungsregeln auf Folie 84.
- 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 3.2: Eingabe von Festkommazahlen

Eine Festkommazahl in einer Datei soll aus einer bis drei Dezimalziffern⁸ 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 84.
- 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.

⁸Darstellung als ASCII-Zeichen '0'→0x30, ..., '9'→0x39, ','→0x2C



Aufgabe 3.3: 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 3.4: Zugriffsmethoden mit Kontrollen

Schreiben Sie für die Feld-Datenstruktur auf der Folie zuvor

- 1 eine Funktion

```
uint8_t9 writeSDatx(uint8_t idx, uint16_t Wert)
```

zum Beschreiben eines Feldelementes und

- 2 eine Funktion

```
uint8_t10 CheckSDatx()
```

zur Formatkontrolle.

⁹Rückgabewert 1 bei Wertebereichsfehler für den Index, 2 bei Wertebereichsfehler des Wertes, sonst null.

¹⁰Rückgabewert 201 bei Wertebereichsfehler der Länge, i bei Wertebereichsfehler von Wert i , 202 bei Prüfkennzeichenfehler, sonst null.



Aufgabe 3.5: 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.



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 Sollwerten¹¹,
- Mehrfachberechnung und Vergleich,
- Rückrechnung der Eingabe aus den Ausgaben und Vergleich und
- die Kontrolle der Suchkriterien bei Suchproblemen.

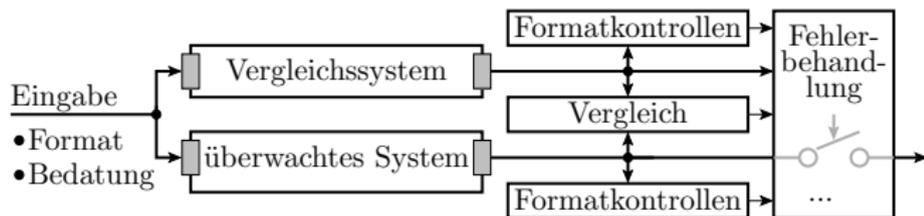
¹¹Erfordert Testbedingungen und wird auf Foliensatz Test behandelt.



Mehrfachber. & Vergleich

Mehrfachberechnung und Vergleich

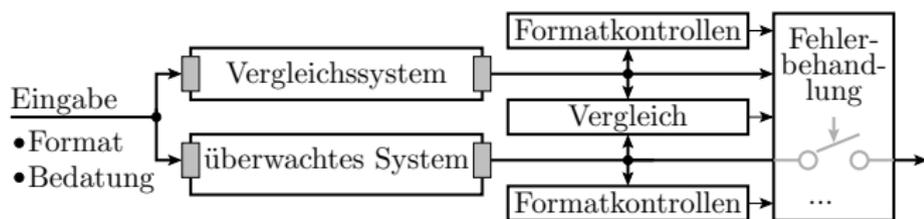
Die Ergebnisse werden mehrfach berechnet. Übereinstimmende Ergebnisse gelten als fehlerfrei. Bei Abweichung kann über einen Mehrheitsentscheid oder Zusatzkontrollen ein Ergebnis als richtig gewertet und weitergereicht werden.



Das Vergleichssystem kann sein:

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

Vergleich mit einem »Golden Device«

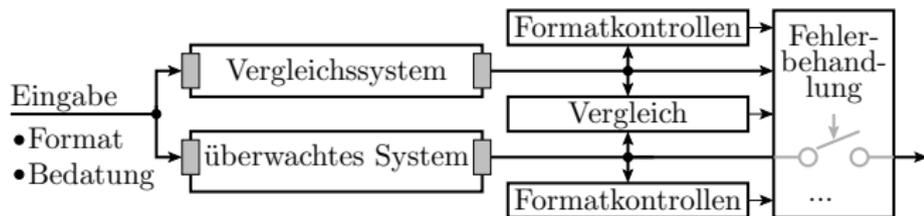


Ein »Golden Device« ist ein gründlich getestetes System, ein Mustergerät oder bei einem Regressionstest die Vorgängerversion. Seine Ausgaben gelten per Definition als richtig. Mit einem »Golden Device« und einem idealen Vergleich¹² werden alle Werteabweichungen erkannt.

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

¹²Vergleich, der übereinstimmende Werte immer als übereinstimmend und abweichende Werte immer als abweichend klassifiziert. In der Realität sind Vergleichsfehler nicht vollständig ausschließbar.

Vergleich diversitärer Systeme



Ein reales System als Vergleichssystem liefert ähnlich häufig falsche Ergebnisse wie das überwachte System. Problematisch sind gleiche Fehler, die die Ausgaben übereinstimmend verfälschen. Denn die werden von einem Vergleich nicht erkannt. Deshalb muss das Vergleichssystem diversitär (verschiedenartig) sein. Diversität entsteht durch unabhängige Entwürfe, unterschiedliche Algorithmen, ... (Fortsetzung Folie 127).

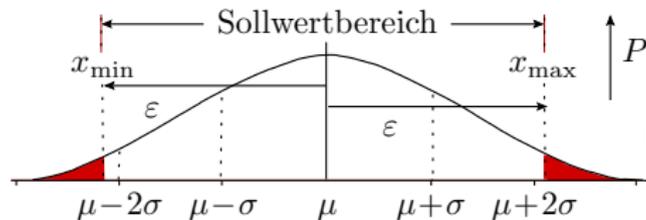
Zufällige Übereinstimmungen sind bei größeren zu vergleichenden Bitvektoren, die fast nie falsch sind, praktisch ausgeschlossen.

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:

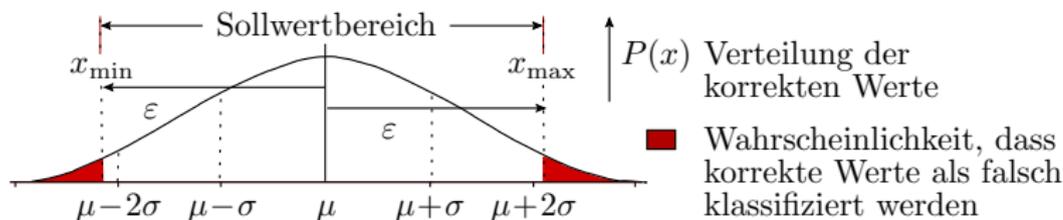
if (x<minWert) || (x>maxWert) <Fehlerbehandlung>;



$P(x)$ Verteilung der korrekten Werte

■ Wahrscheinlichkeit, dass korrekte Werte als falsch klassifiziert werden

Festlegung des Vergleichsfensters



Für normalverteilte Zufallsgrößen wird für den Sollwertbereich zweckmäßiger Weise $\mu \pm \epsilon$ mit $\epsilon \geq 2\sigma$ 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 der Quantisierung und Rundung entsteht ein Fehler von maximal $\pm 0,5 \text{ LSB}^{13}$. Die Standardabweichung als die Wurzel aus der mittleren quadratischen Abweichung ist die reichliche Hälfte davon:

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

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

Wert	nächster darstellbarer Wert	Rundungsfehler
125,4380	7D,70h=125,4375	-0,05%
2,7130	2,B7h=2,7148	+0,18%
28,2000	1C,33h= 28,1992	-0,08%

¹³LSB (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:

$$125,438 \cdot 2,713 \cdot 28,2 = 9596,8349$$

$$7D,70h \cdot 2,B7h \cdot 28,2h = 2569,C7h = 9577,7773$$

$$\text{Abweichung :} \quad -19,0575$$

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



Diversität

Das Hauptproblem von Mehrfachberechnung und Vergleich ist, dass eine Berechnung mit denselben Fehlern nach demselben Algorithmus dieselben falschen Ergebnisse liefert. Übereinstimmende falsche Ergebnisse erkennt ein Vergleich nicht.

Diversität bedeute Verschiedenartigkeit¹⁴:

- verschiedene Eingaben,
- verschiedene Algorithmen,
- Ausschluss gleicher Fehler und Störquellen,
- getrennte Hardware.

»Mehrfachberechnung und Vergleich« beschränkt die zulässige Diversität. Denn er verlangt:

- Gleiche Eingaben je Service-Aufruf und
- übereinstimmende Sollausgaben.

¹⁴Von lateinisch *diversitas* „Vielfalt“ abgeleitet.



Nicht nutzbare Diversität

- Eingabediversität: Ein gewünschtes Service-Ergebnis lässt sich in der Regel mit verschiedenartigen Eingaben und Service-Anforderungen erzielen. Bei getesteten, überwiegend funktionierenden Systemen gibt es für fehlerhaft ausgeführte Service-Leistungen meist Alternativen mit anderen Bedatungen und Formaten, die richtig ausgeführt werden, aber deren Ergebnisse nicht direkt vergleichbar sind.
- Diversität der Problemlösung: Für die meisten Aufgaben (PC, Drucker, Textverarbeitung, Webbrowser, ...) gibt es alternative Hard- und Software. Nur stimmen auch hier die Bedatungen und Formate der Service-Anforderungen und Ergebnisse praktisch nie so gut überein, dass eine Kontrolle über Vergleich möglich ist.

Entwurfsdiversität

Die Mindestvoraussetzung für die Entstehung von Systeme mit identischem Sollverhalten sind identische Entwurfsvorgaben für Schnittstellen und die Service-Funktionen. Resultierende Probleme:

- Die Fehlfunktionen durch falsche Vorgaben stimmen in allen Systemen, die die Vorgaben erfüllen, überein und sind somit nicht erkennbar.
- Mehrdeutigkeiten in den Vorgaben führen zu abweichendem korrekten Verhalten und beim Ergebnisvergleich zu Phantomfehlern¹⁵. Die Vermeidung/Beseitigung von Phantomfehlern verlangt weitere Absprachen und Vorgaben, die die Diversität mindern.
- Vervielfachung des Entwurfsaufwands.

Laut Literatur $\geq 10\%$ nicht erkennbare Fehler [6].

¹⁵Klassifizierung richtiger Ergebnisse als falsch.

Natürliche Diversität

Einige Quellen von Fehlfunktionen haben eine natürliche Diversität, die identische Fehlerwirkungen praktisch ausschließt, z.B.

- Störungen paralleler Berechnungen auf unterschiedlichen Rechnern oder bei Wiederholung auf demselben Rechner.
- Hardware-Fertigungsfehler und Ausfälle unterschiedlicher Rechner.
- Hardware-Entwurfsfehler bei Verwendung unterschiedlicher Rechnertypen mit unterschiedlichen Schaltkreisen.
- Fehler in voneinander unabhängig entwickelten Compilern, Synthese-Programmen, Entwurfsbibliotheken, ...

Es gibt Systeme mit Mehrfachberechnung und Vergleich im Einsatz, die nur natürliche Diversität ausnutzen¹⁶, insbesondere zur Erzielung einer Fehlertoleranz gegenüber Ausfällen und Störungen der Hardware (siehe später Abschn. 5.4).

¹⁶Ohne aufwändige Entwurfsdiversität.



Syntaktische Diversität

Syntaktisch erzeugte Diversität erhält man durch systematische und automatisch durchführbare Modifikationen eines funktionsfähigen Programms [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. Die zu erwartende Fehlererkennungswahrscheinlichkeit soll etwas höher sein als beim Vergleich von Programmversionen ohne diese Modifikationen. Fehlfunktionen durch Entwurfsfehler sind nicht erkennbar.



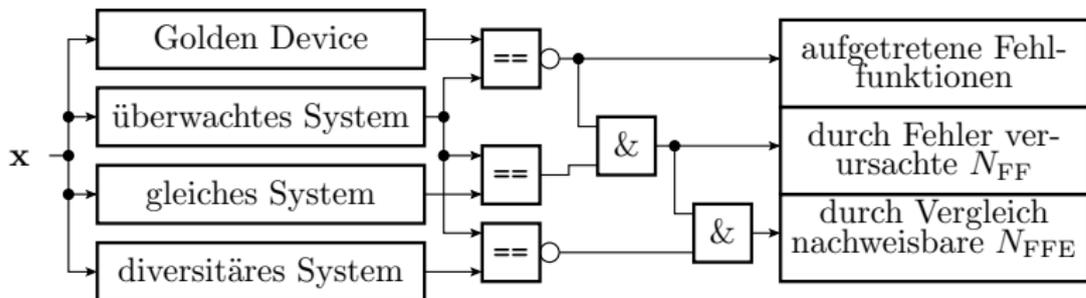
Abschätzung der Diversität

Ein sinnvolles Maß für die Diversität ist das Verhältnis aus der Anzahl der durch Fehler verursachten Fehlfunktionen N_{FFE} , die durch Verdopplung und Vergleich erkennbar sind, zur Anzahl aller durch Fehler verursachten Fehlfunktionen N_{FF} :

$$E(\text{Div}) = \frac{N_{\text{FFE}}}{N_{\text{FF}}}$$

($E(\dots)$ – Erwartungswert). Das Zählen, wie oft eine Service durch einen Fehler versagt, verlangt eine ideale Kontrolle, z.B. einen Vergleich mit den Ergebnissen eines »Golden Device«, sowie die Kontrolle, dass das System reproduzierbar oder zumindest häufig in derselben Weise versagt, z.B. eine parallele Berechnung mit einem identischen System.

Zähler für Fehlfunktionen:



$$E(Div) = \frac{N_{FFE}}{N_{FF}}$$

Eine vertrauenswürdige Schätzung verlangt, dass mindestens so lange Fehlfunktionen gezählt werden, bis einige (mindestens zehn) durch Fehler verursachte nicht erkennbare Fehlfunktionen aufgetreten sind:

$$N_{FF} - N_{FFE} > 10$$

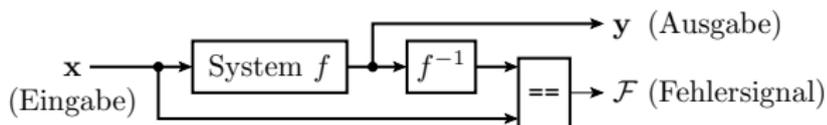
Ergebnisse eines solchen Experiments wurden bisher noch nicht in der Literatur veröffentlicht.



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 naheliegendste 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.

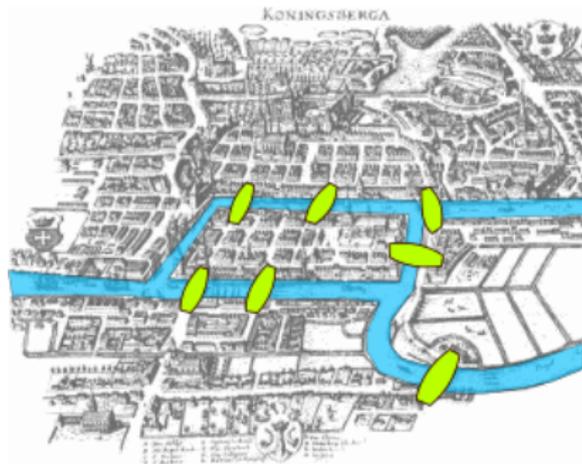
Die vorgegebenen Kontrollfunktionen sind meist einfacher und dadurch fehlerärmer zu realisieren als die Suchalgorithmen. Im Entwurfsprozess ist es zweckmäßig, diese zuerst zu programmieren, eventuell sogar diversitär, und sehr gründlich zu testen, um sie später gleichermaßen für die Suche und die Kontrollen 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 4.1: 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 zu

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

- Zwangsbremmung, 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[4]

- Angabe der Fehlerschwere in den Assertion-Anweisung
- Umschaltung des Exception-Handlers, dass nur bei schweren Fehlern ein Abbruch erfolgt.
- Protokollierung und Versendung aller Assertion-Ausgaben.¹⁷

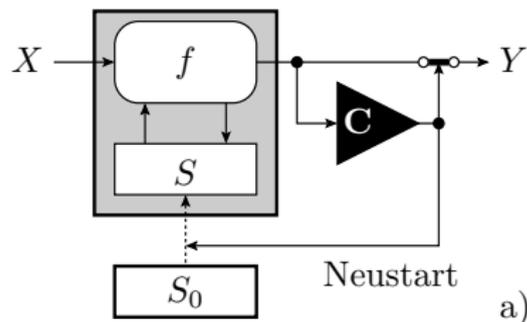
¹⁷Höfliche Programme fragen bzw. informieren zumindest, bevor sie Mails nach Hause schicken.



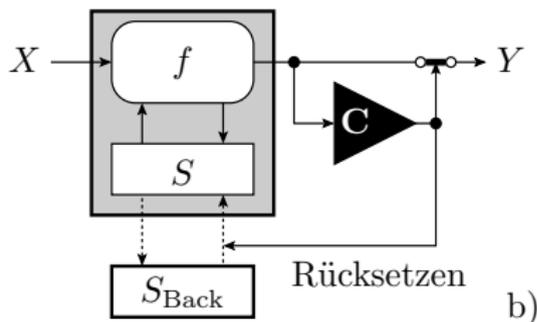
Neustart, Wiederholung

Neuinitialisierung

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



a)



b)

- 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, Regelungen, ...).
- 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 protokollieren¹⁸.

¹⁸So 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 Prüfzeichenfehler:

- 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 sinnvolle Reaktion auf Fehlfunktion:

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

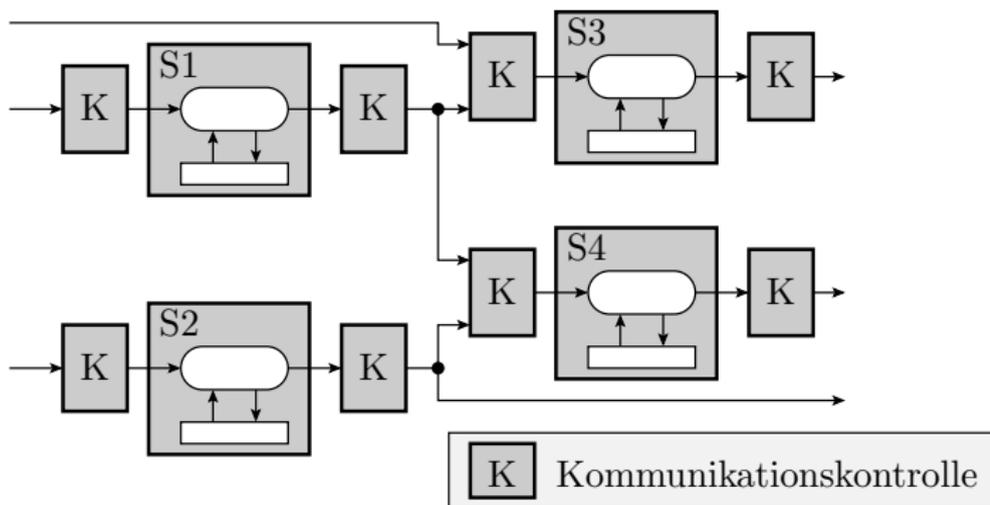
erfordern, dass der Zustand des Systems, das die Fehlerbehandlung durchführt, unverfälscht ist:

- Die Reset-Logik 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.

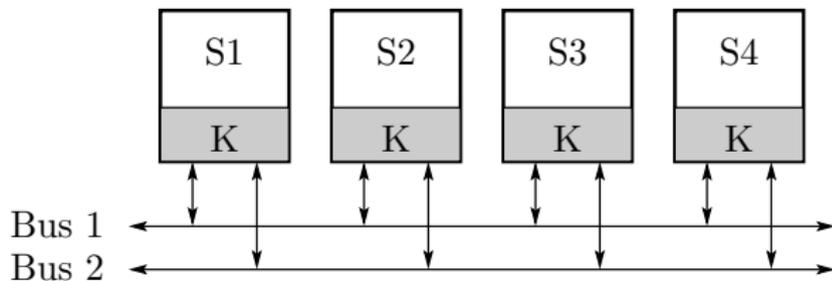
Prinzip der Fehlerisolation

- Modulares System mit Kontrollen an den Schnittstellen, die keine kontaminierten Daten weitergeben.



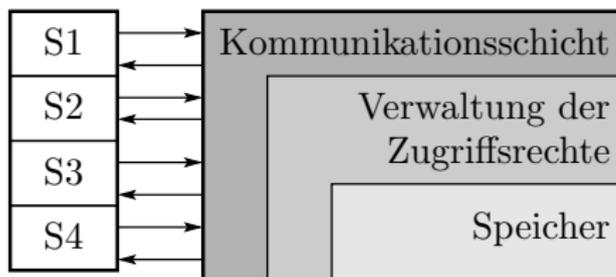
Fehlerisolation in Bussystemen

- 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 Betriebssystem



- 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 Ressourcen (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 redundanter¹⁹ Funktionseinheiten:

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

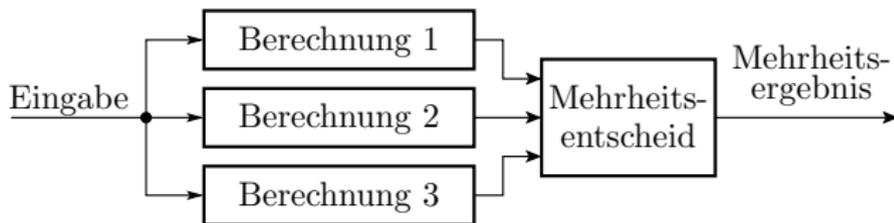
Im Bereich der diversitärer Redundanz haben sich im wesentlichen zwei Ausprägungen durchgesetzt:

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

¹⁹Redundanz ist die Existenz von Teilen, die nicht unbedingt für die eigentliche Funktion eines System notwendig sind



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

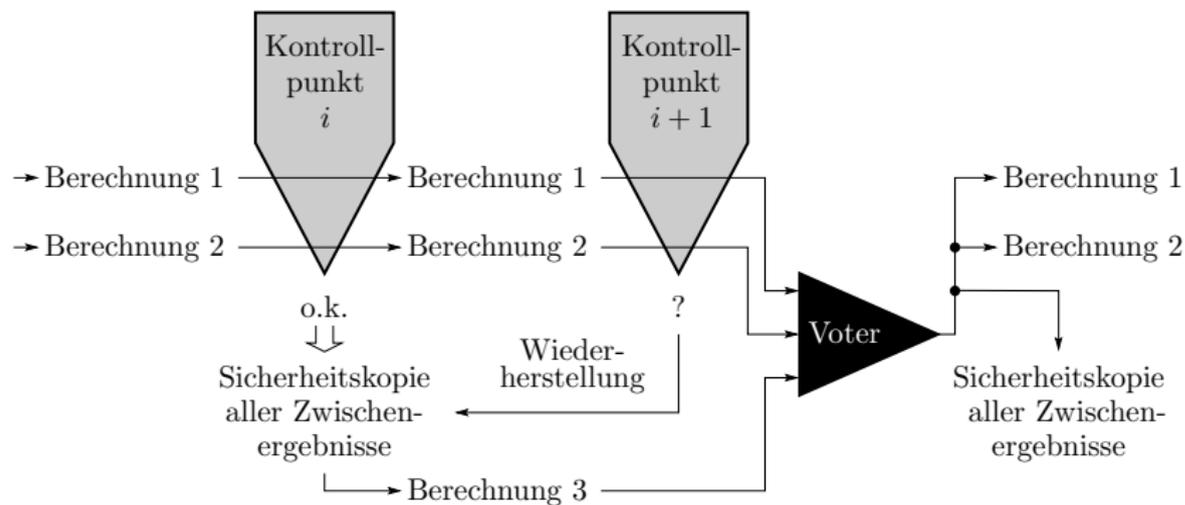
- Hoher Aufwand: Das originale n-Versionssystem verlangt mindestens drei statt nur einem Rechner. Abhilfe schafft, die Berechnungen nacheinander auf demselben oder nur zwei Systemen auszuführen. Das spart Hardware und erhöht den Rechenaufwand.
- Ein Mehrversionssystem in dieser Form toleriert keine übereinstimmenden 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 einer Vielzahl weiterer Kontrollen und einer Vielzahl einprogrammierten Verhaltensweisen bei Fehlfunktionen.

Check-Point-Roll-Back-Recovery [3]



- Nur zwei parallel ausgeführte Berechnungen.
- An einprogrammierten Kontrollpunkten im Programm Vergleich der Bearbeitungszustände (Werte aller Variablen und Register)



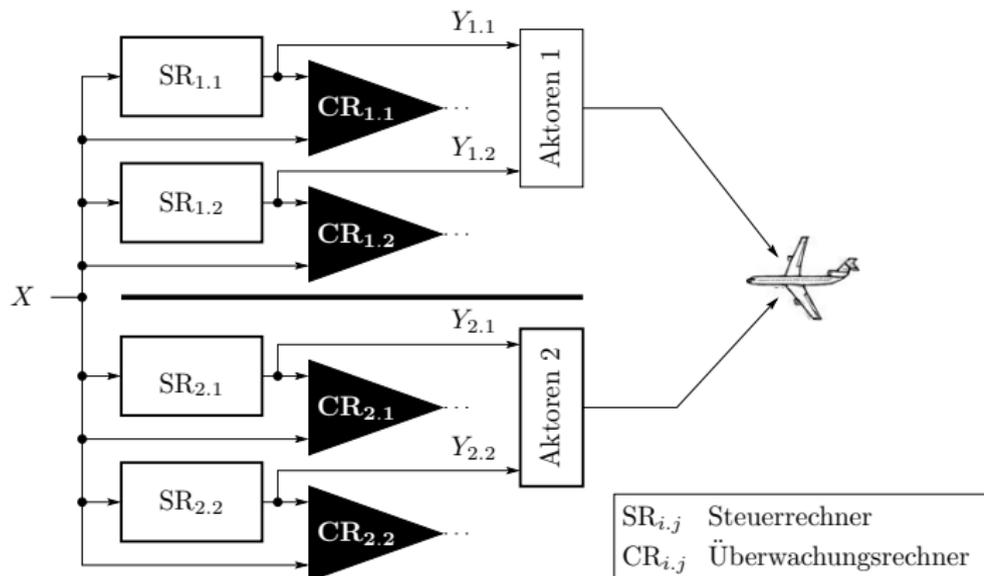
- Bei Übereinstimmung Speicherung des Bearbeitungszustands in einem geschützten Speicher.
- Bei Abweichung, Laden der letzten Sicherheitskopie und Berechnungswiederholung (Roll-Back Recovery).
- Nach Roll-Back Recovery am Kontrollpunkt Mehrheitsentscheid.
- Wenn Mehrheitsergebnis, diesen als gesicherten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

Sequoia-System [1]:

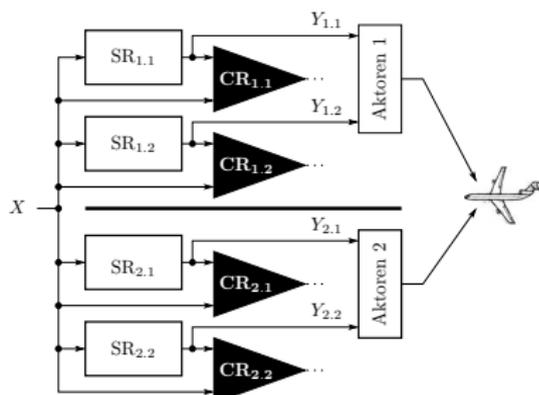
- Berechnung auf zwei Prozessoren mit eigenem Write-Back-Cache.
- Vergleich in jedem Takt.
- Zustands-Backup bei Ereignissen wie Stack-Überlauf und Prozesswechsel.
- Hauptspeicher hat die Funktion des stabilen Speichers.

Flugsteuersystem Airbus A3XX [5]

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 Komplettsensor- oder Aktorausfällen System 1 übernimmt System 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.



Aufgaben

Aufgabe 5.1: 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);
int20 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.

²⁰Rückgabewert bei Abweichung null, sonst eins.



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] 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.