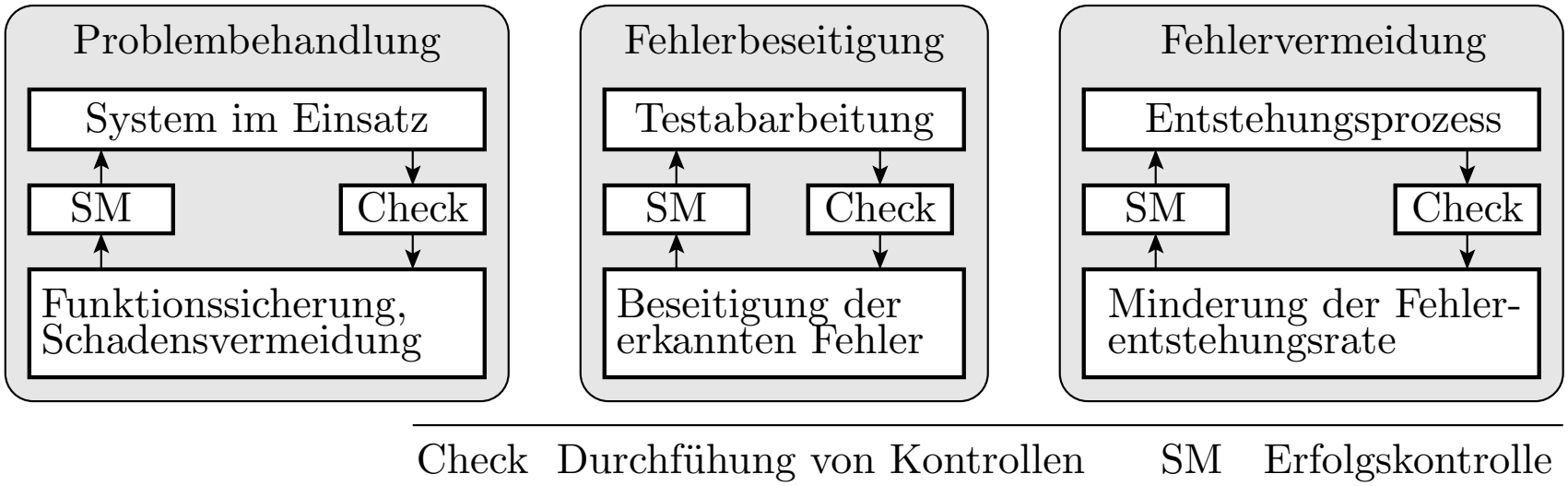
Test und Verlässlichkeit 5: Mehr zu Tests und Kontrollen

Prof. G. Kemnitz

Institut für Informatik, TU Clausthal
17. Dezember 2024

Inhalt Foliensatz 5

5.2 Gefährdungsabwendung (Folie 1.10)



Verlässlichkeit wird durch Problembeseitigung auf drei Ebenen gesichert:

- Überwachung und Problembehandlung während des Betriebs,
- Test und Fehlerbeseitigung.
- Fehlervermeidung durch Fehlerbeseitigung in den Entstehungsprozessen.

Mit der unterstellten Fehlerkultur, dass erkannte Probleme beseitigt werden, entscheiden die Tests und Kontrollen über die Verlässlichkeit.

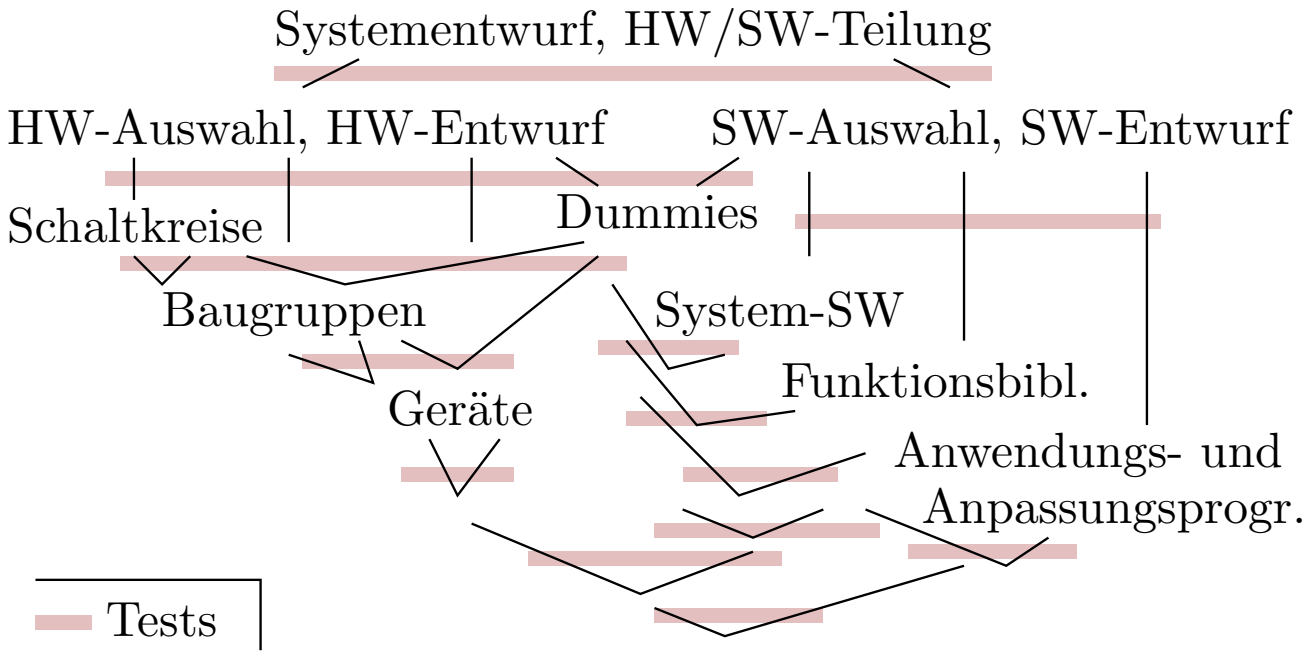
5.3 Dieser Foliensatz

... beschäftigt sich weiter mit Tests und Kontrollen, speziell mit

- Inspektion: Gütemaße, Inspektionstechniken
- Dynamische Tests: Durchführung, Systemgestaltung, prüfgerechter Entwurf.
- Codebasierte Kontrollen: Fehlererkennende und fehlerkorrigierende Codes, Prüfkennzeichen, RAIDs.
- Ergebniskontrollen: Wertebereiche, Syntax, Invarianten, ...

Inspektion

5.4 Vielfalt der Test (Abschn. 2.1.4)



In Entstehungsprozessen für IT-Systeme erfolgen in der Regel eine Vielzahl verschiedener Test:

- während und nach jeder Entwurfsphase,
- hierarchisch aufsteigend Teilsysteme dann Zusammenwirken, ...

Am Anfang hauptsächlich Inspektion entstandener Dokumente.

5.5 Inspektion (Review)

Inspektion, Sichtprüfungen (von lat. inspicere = besichtigen, betrachten). Angewendet auf:

- Dokumentationen (Spezifikation, Nutzerdokumentation, ...),
- Programmcode, Testausgaben,
- Schaltungsbeschreibungen, Konstruktionspläne, ...
- Auch Testausgaben, bevor Sollwerte festgelegt oder Kontrollen programmiert sind.

Eigenenschaften von Kontrollen durch Inspektion:

- Fast alles kontrollierbar,
- großer manueller Arbeitsaufwand,
- geringere Güte als automatisierte Kontrollen.
- Nachweis auch nicht funktionaler Fehler: Verstöße gegen Vereinbarungen und Standards, Antipattern*, ...
- Know-How-Weitergabe als positiver Zusatzeffekt.

* Beschreibungselemente, die die Übersichtlichkeit beeinträchtigen, die Kontrolle erschweren und die Fehlerentstehung begünstigen.

Kenngrößen

5.6 Kenngrößen einer Inspektion

Kenngrößen wie bei jedem anderen Test:

- Fehlerabdeckung:

  (2.1) $$FC = \frac{\#DF}{\#F}\Big|_{\mathrm{ACR}}$$

- Phantomfehlerrate:

  (2.2) $$\zeta_{\mathrm{PF}} = \frac{\#PM}{N}\Big|_{\mathrm{ACR}}$$

Weitere Kenngrößen zur Bewertung von Inspektionsprozessen [3]:

- Effizienz (EFC): Gefundene Fehler pro Mitarbeiterstunde.
- Effektivität (EFT): Gefundene Fehler je 1000 NLOC.

Kenngrößenschätzung getrennt für funktionale und andere Fehler.

FC	Fehlerabdeckung (fault coverage), Anteil der nachweisbaren Fehler.
$\#F$, $\#DF$	Fehleranzahl, Anzahl der davon nachweisbaren Fehler.
ζ_{PF}	Phantomfehlerrate des Tests.
N, $\#PM$	Testanzahl, Anzahl der Phantomfehler.
ACR	Geeignete Zählwertgrößen, typ. 100 ... 1000 ein- und nicht eingetretene Zählereignisse.
EFC	Effizienz, gefundene Fehler pro Mitarbeiterstunde.
EFT	Effektivität, gefundenen Fehler je 1000 NLOC.
NLOC	Netto Lines of Code, Anzahl der Code-Zeilen ohne Kommentar und Leerzeilen.

Beispiel 5.1: Inspektion

Programmgröße: 10.000 NLOC, Arbeitsaufwand: 200 Stunden, 228 gefundene Fehler, davon 156 funktionale. Geschätzte Gesamtfehleranzahl (vor der Inspektion): 300, davon 200 funktionale. Wie groß sind:

a) *Fehlerabdeckung FC?*

b) *Effizienz und Effektivität?*

NLOC Netto Lines of Code, Anzahl der Code-Zeilen ohne Kommentar und Leerzeilen.

Programmgröße: 10.000 NLOC, Arbeitsaufwand: 200 Stunden, 228 gefundene Fehler, davon 156 funktionale. Geschätzte Gesamtfehleranzahl (vor der Inspektion): 300, davon 200 funktionale. Wie groß sind:

a) *Fehlerabdeckung FC?*

b) *Effizienz und Effektivität?*

	gesamt	funktionale Fehler	sonstige Fehler
$FC = \frac{\#DF}{\#F}$	$\frac{228}{300}$	$\frac{156}{200}$	$\frac{72}{100}$
Effizienz (EFC)	$\frac{228\text{ Fehler}}{200\text{ h}}$	$\frac{156\text{ Fehler}}{200\text{ h}}$	$\frac{72\text{ Fehler}}{200\text{ h}}$
Effektivität (EFT)	$\frac{228\text{ Fehler}}{10.000\text{ NLOC}}$	$\frac{156\text{ Fehler}}{10.000\text{ NLOC}}$	$\frac{72\text{ Fehler}}{10.000\text{ NLOC}}$

- Effizienz und Effektivität bewerten den Inspektionsprozess und berechnen sich aus tatsächlich zählbaren Werten.
- Die Fehlerabdeckung FC hängt von der nicht zähl-, sondern nur abschätzbaren Gesamtfehleranzahl ab.

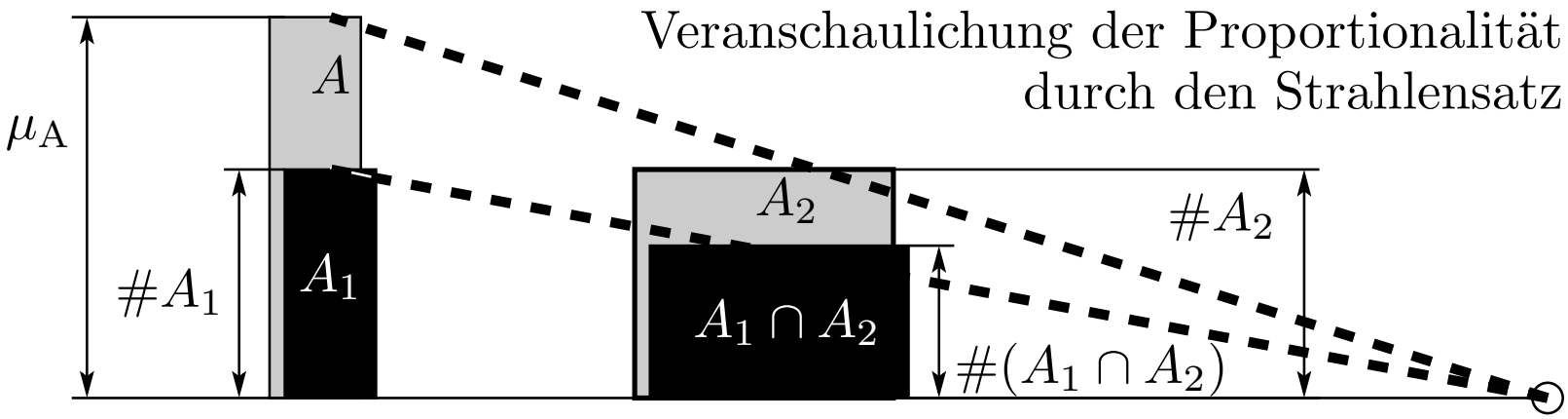
Wie lässt sich die nicht zählbare Gesamtfehleranzahl schätzen?

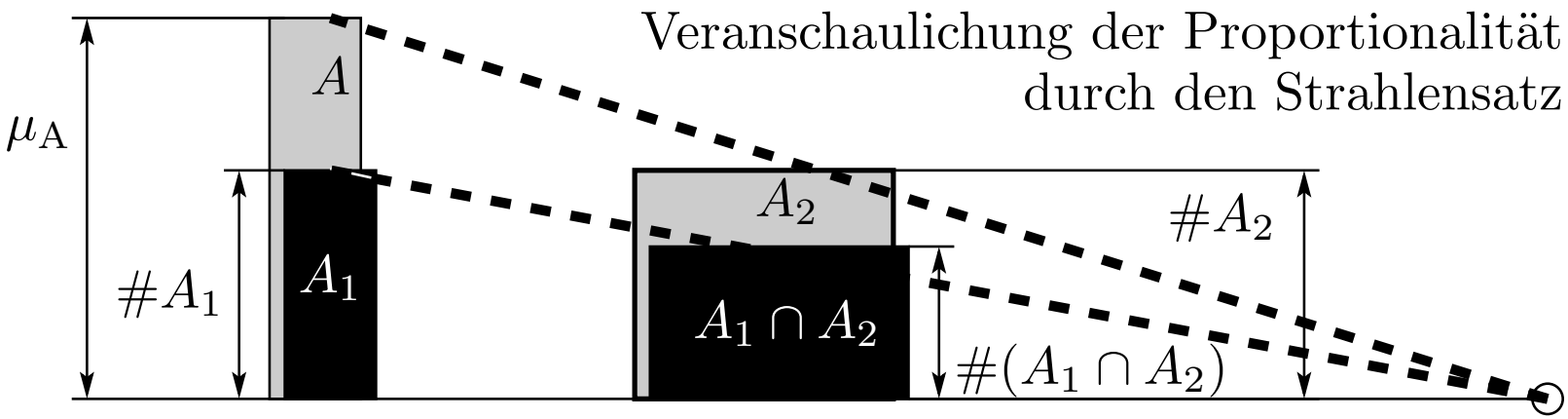
5.8 Capture-Recapture

Abgeleitet von einem Schätzer für die Größe von Tierpopulationen (z.B. von Vögeln in einem Gebiet) [2, 5, 4].

- Aus einer Menge A unbekannter Größe wird eine Menge A_1 von Tieren eingefangen, gekennzeichnet und freigelassen.
- Nach Vermischung der Population wird eine Menge A_2 von Tieren einfangen. Gekennzeichnete Tiere werden gezählt.

Bei tierunabhängiger Einfangwahrscheinlichkeit ergibt sich der Anteil der Tiere, die beim zweiten Einfangen gekennzeichnet sind, über den Strahlensatz:





$$\left.\frac{\hat{\mu}_A}{\#A_1} = \frac{\#A_2}{\#(A_1 \cap A_2)}\right|_{ACR}$$

Zu erwartende Größe der Tierpopulation:

$$\left.\hat{\mu}_A = \frac{\#A_1 \cdot \#A_2}{\#(A_1 \cap A_2)}\right|_{ACR}$$

$\hat{\mu}_A$	Zu erwartende Anzahl aller Tiere.
A_1, A_2	Menge der beim ersten bzw. zweiten mal eingefangenen Tiere.
$A_1 \cap A_2$	Menge der Tiere, die beide Male eingefangen werden.
#...	Anzahl der Elemente der Mengen.
ACR	Brauchbare Schätzwerte nur bei geeigneten Zählwertgrößen.

5.9 Fehler statt Tiere

Zwei Inspekteure $i \in \{1, 2\}$ finden jeweils eine Menge von F_i Fehlern, drunter $F_1 \cap F_2$ gleiche Fehler:

$$\hat{\mu}_{\mathrm{F}} = \left. \frac{\#F_1 \cdot \#F_2}{\#\left(F_1 \cap F_2\right)} \right|_{\mathrm{ACR}} \tag{5.1}$$

Die zu erwartende Fehlerabdeckung ist das Verhältnis der Anzahl der insgesamt von beiden Inspekteuren gefundenen Fehler $\#\left(F_1 \cup F_2\right)$ zur zu erwartenden Geamtfehleranzahl μ_{F}:

$$\hat{\mu}_{\mathrm{FC}} = \left. \frac{\#\left(F_1 \cup F_2\right)}{\hat{\mu}_{\mathrm{F}}} \right|_{\mathrm{ACR}} = \left. \frac{\#\left(F_1 \cap F_2\right) \cdot \#\left(F_1 \cup F_2\right)}{\#F_1 \cdot \#F_2} \right|_{\mathrm{ACR}} \tag{5.2}$$

$\hat{\mu}_{\mathrm{F}}$	Geschätzter Erwartungswerte der Gesamtfehleranzahl.
$\#F_1, \#F_2$	Anzahl der von Inspektor 1 bzw. Inspektor 2 gefundenen Fehler.
$\#(F_1 \cap F_2)$	Anzahl von beiden Inspekteuren gefundenen Fehler.
$\hat{\mu}_{\mathrm{FC}}$	Schätzwert der zu erwartenden Inspektionsfehlerabdeckung.

Beispiel 5.2: Capture-Recapture

- Inspekteur 1: 228 gefundene Fehler.
- Inspekteur 2: 237 gefundene Fehler.
- Übereinstimmend: 105 Fehler.

Wie groß ist die zu erwartende Gesamtfehleranzahl und die Inspektionsfehlerabdeckung?

- Inspekteur 1: 228 gefundene Fehler.
- Inspekteur 2: 237 gefundene Fehler.
- Übereinstimmend: 105 Fehler.

Wie groß ist die zu erwartende Gesamtfehleranzahl und die Inspektionsfehlerabdeckung?

(5.1) $$\hat{\mu}_{\mathrm{F}} = \left.\frac{\#F_1 \cdot \#F_2}{\#(F_1 \cap F_2)}\right|_{\mathrm{ACR}}$$

(5.2) $$\hat{\mu}_{\mathrm{FC}} = \left.\frac{\#(F_1 \cup F_2)}{\hat{\mu}_{\mathrm{F}}}\right|_{\mathrm{ACR}} = \left.\frac{\#(F_1 \cap F_2) \cdot \#(F_1 \cup F_2)}{\#F_1 \cdot \#F_2}\right|_{\mathrm{ACR}}$$

Schätzwert der zu erwartenden Gesamtfehleranzahl:

$$\hat{\mu}_{\mathrm{F}} = \frac{228 \cdot 237}{105} = 515$$

Schätzwert der zu erwartenden Inspektionsfehlerüberdeckung:

$$\hat{\mu}_{\mathrm{FC}} = \frac{228 + 237 - 105}{515} = 70\%$$

$\hat{\mu}_{\mathrm{F}}$	Geschätzter Erwartungswerte der Gesamtfehleranzahl.
$\hat{\mu}_{\mathrm{FC}}$	Schätzwert der zu erwartenden Inspektionsfehlerabdeckung.
$\#F_1, \#F_2$	Anzahl der von Inspektor 1 bzw. Inspektor 2 gefundenen Fehler.
$\#(F_1 \cap F_2)$	Anzahl von beiden Inspekteuren gefundenen Fehler.

5.11 Schätzgenauigkeit

Die zu erwartende Gesamtfehleranzahl nach (Gl. 5.1) ist das Produkt zweier Zählwerte geteilt durch einen dritten. Für grobe Abschätzungen addieren sich bei Multiplikation und Division die Quadrate der Varianzkoeffizienten. Das bedeutet, dass die drei Zählwerte für dieselbe Schätzgenauigkeit dreimal so groß wie Zählwerte zur Schätzung von Eintrittswahrscheinlichkeiten nach (Gl. 4.72) und (Gl. 4.73) sein müssen:

$$x_{\mathrm{AV}} \gtrsim 3 \cdot \frac{\kappa \cdot \left(\Phi^{-1}\left(1-\frac{\alpha}{2}\right)\right)^2}{\varepsilon_{\mathrm{r}}^2} \cdot (1-\hat{p}) \text{ für } \hat{p} \leq 50\%$$

$$n - x_{\mathrm{AV}} \geq 3 \cdot \frac{\kappa \cdot \left(\Phi^{-1}\left(1-\frac{\alpha}{2}\right)\right)^2}{\varepsilon_{\tilde{\mathrm{r}}}^2} \cdot \hat{p} \text{ für } \hat{p} > 50\%$$

Zählwerte um hundert wie in der Beispielaufgabe sind eigentlich viel zu klein.

ε_{r}, $\varepsilon_{\tilde{\mathrm{r}}}$	Intervallradius realtiv zum erwarteten Eintritts- bzw. Nichteintritts-Zählwert.
κ	Varianzerhöhung durch Abhängigkeiten, für unabhängige Zählwerte $\kappa \leq 1$.
$\Phi^{-1}(..)$	Inverse Funktion zur Verteilungsfunktion der standardisierten Normalverteilung.
n, x_{AV}	Anzahl der Zählversuche, Experimentell bestimmter Ist-Zählwert.
$\hat{p}$	Schätzwert der Eintrittswahrscheinlichkeit.

5.12 Systematische Schätzfehler

Zu den zufällige kommen systematische Schätzfehler:

- Capture-Recaptur unterstellt für alle Fehler dieselbe Erkennungswahrscheinlichkeit. In der Praxis reicht diese aber von *»Fehler kaum übersehbar«* bis *»Fehler fast nicht erkennbar«*.
- Capture-Recaptur verbietet Informationsaustausch zwischen den Inspekteuren. Falls es doch einen Informationsaustausch gibt, vergrößert der die Menge der von beiden Inspekteuren gefundenen Fehler $F_1 \cap F_2$ gegenüber einer unabhängigen Suche.
- Wenn die Inspekteure ihre Fehlerlisten voneinander abschreiben

$$\#F_1 = \#F_2 = \#\left(F_1 \cap F_2\right) = \#\left(F_1 \cup F_2\right)$$

$$\hat{\mu}_{\mathrm{FC}} = \frac{\#(F_1 \cap F_2) \cdot \#(F_1 \cup F_2)}{\#F_1 \cdot \#F_2} = 1$$

  Ein völlig unsinniger Schätzwert.

$\#F_1, \#F_2$	Anzahl der von Inspektor 1 bzw. Inspektor 2 gefundenen Fehler.
$\#(F_1 \cap F_2)$	Anzahl von beiden Inspekteuren gefundenen Fehler.
$\hat{\mu}_{\mathrm{FC}}$	Schätzwert der zu erwartenden Inspektionsfehlerabdeckung.

5.13 Kontrollfehler

Einbau von Kontrollfehlern (Mutationen) in das zu inspizierende Datenmaterial und Abschätzung der zu erwartenden Fehlerabdeckung aus dem Anteil der gefundenen Kontrollfehler. Wie bei einem Zufallstests tendiert die zu erwartende Modellfehlerabdeckung gegen die Fehlerabdeckung des c_{MF}-fachen Inspektionsaufwands (Gl. 2.37):

$$\mu_{\mathrm{FCM}}(t) = \mu_{\mathrm{FC}}(c_{\mathrm{MF}} \cdot t)$$

Mit typisch zu findenen Fehlern als Kontrollfehler (vergleichbare mittlere Fehlernachweiszeit) $c_{\mathrm{MF}} \approx 1$:

$$\hat{\mu}_{\mathrm{FC}} = \left.\frac{\#F_{\mathrm{DM}}}{\#F_{\mathrm{M}}}\right|_{\mathrm{ACR}} \tag{5.3}$$

Geschätzte Gesamtfehleranzahl:

$$\hat{\mu}_{\mathrm{F}} = \left.\frac{\#F_{\mathrm{D}}}{\hat{\mu}_{\mathrm{FC}}}\right|_{\mathrm{ACR}} = \#F_{\mathrm{D}} \cdot \left.\frac{\#F_{\mathrm{DM}}}{\#F_{\mathrm{M}}}\right|_{\mathrm{ACR}} \tag{5.4}$$

μ_{FCM}	Zu erwartende Mutationsabdeckung.
μ_{FC}	Zu erwartende Fehlerabdeckung.
c_{MF}, t	Mutationsspezifische Skalierung des Inspektionszeit, Inspektionszeit.
$\#F_{\mathrm{M}}$, $\#F_{\mathrm{DM}}$	Anzahl der untersuchten Mutationen, Anzahl der davon erkannten Mutationen.
μ_{F}, $\#F_{\mathrm{D}}$	Zu erwartende Gesamtfehleranzahl, Anzahl der erkannten Fehler.

5.14 Vergleich mit Capture-Recapture

Zufällige Schätzfehler:

- Nur ein zufälliger Zählwert, erforderliche Zählwertgröße:

(4.72) $$x_{\mathrm{AV}} \geq \frac{\kappa \cdot \left(\Phi^{-1}\left(1-\frac{\alpha}{2}\right)\right)^2}{\varepsilon_{\mathrm{r}}^2} \cdot (1-\hat{p}) \text{ für } \hat{p} \leq 50\%$$

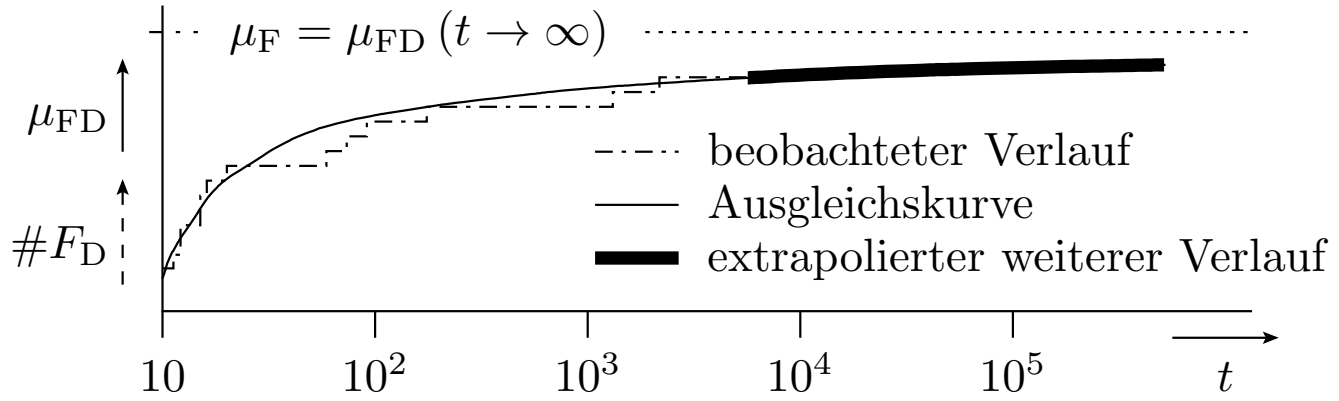
(4.73) $$n - x_{\mathrm{AV}} \geq \frac{\kappa \cdot \left(\Phi^{-1}\left(1-\frac{\alpha}{2}\right)\right)^2}{\varepsilon_{\tilde{\mathrm{r}}}^2} \cdot \hat{p} \text{ für } \hat{p} > 50\%$$

Auch geringere systematische Schätzfehler:

- Entfall der unzutreffenden Annahme gleichgroßer Nachweiswahrscheinlichkeiten und der daraus resultierenden systematischen Schätzfehler.
- Informationsaustausch zwischen Fehlereinbau und Inspektion besser unterbindbar als zwischen zwei Inspekteuren.

ε_{r}, $\varepsilon_{\tilde{\mathrm{r}}}$	Intervallradius realtiv zum erwarteten Eintritts- bzw. Nichteintritts-Zählwert.
κ	Varianzerhöhung durch Abhängigkeiten, für unabhängige Zählwerte $\kappa \leq 1$.
$\Phi^{-1}(..)$	Inverse Funktion zur Verteilungsfunktion der standardisierten Normalverteilung.
n, x_{AV}	Anzahl der Zählversuche, Experimentell bestimmter Ist-Zählwert.
$\hat{p}$	Schätzwert der Eintrittswahrscheinlichkeit.

5.15 Extrapolation der Inspektionszeit



Auch für Inspektionen gilt in der Regel das Pareto-Prinzip. Mit einem kleinen Teil des Inspektionszeit t wird die Mehrheit der Fehler gefunden. Deutet auf eine Pareto-Verteilung (vergl. Gl. 4.88):

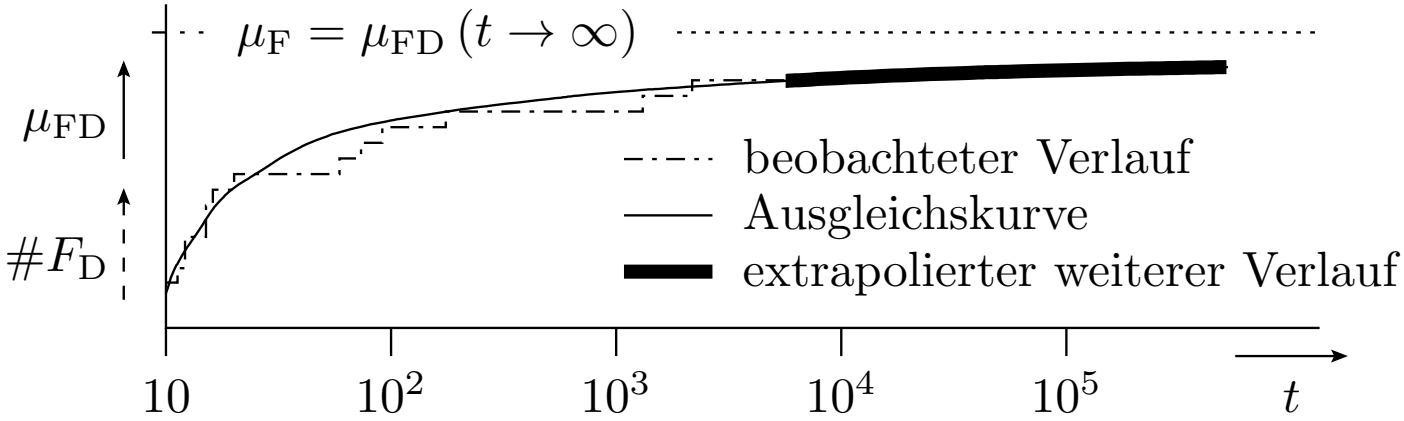
$$F_X(t) = \mathbb{P}[X \leq t] = \mu_{FC}(t) = \begin{cases} 0 & t \leq t_{min} \\ 1 - \left(\frac{t_{min}}{t}\right)^K & \text{sonst} \end{cases} \tag{5.5}$$

Schätzung $\mu_F\,(t \to \infty)$ aus Verlauf für kleine t wie im Bild unsicher.

$\#F_D$, μ_{FD}	Anzahl der erkannten Fehler, zu erwartenden Anzahl der erkennbaren Fehler.
t, t_{min}	Inspektionszeit, Skalenparameter der Pareto-Verteilung.
X	Zufallsgröße der Inspektionszeit, um einen Fehler zu finden.
$K > 0$	Formfaktor der Pareto-Verteilung.

Inspektionstechniken

5.16 Inspektionszeitverteilung und Effizienz



Die zu erwartende Effizienz nimmt proportional mit dem Anstieg der Verteilungsfunktion, d.h. mit der Dichte der Inspektionszeit ab:

$$\frac{\mu_{\mathrm{EFC}}(t)}{\mu_{\mathrm{F}} \cdot 1\,\mathrm{h}} = f_X(N) = \frac{\mathrm{d}F_X(t)}{\mathrm{d}t} = \frac{K \cdot t_{\min}^K}{t^{K+1}} \tag{5.6}$$

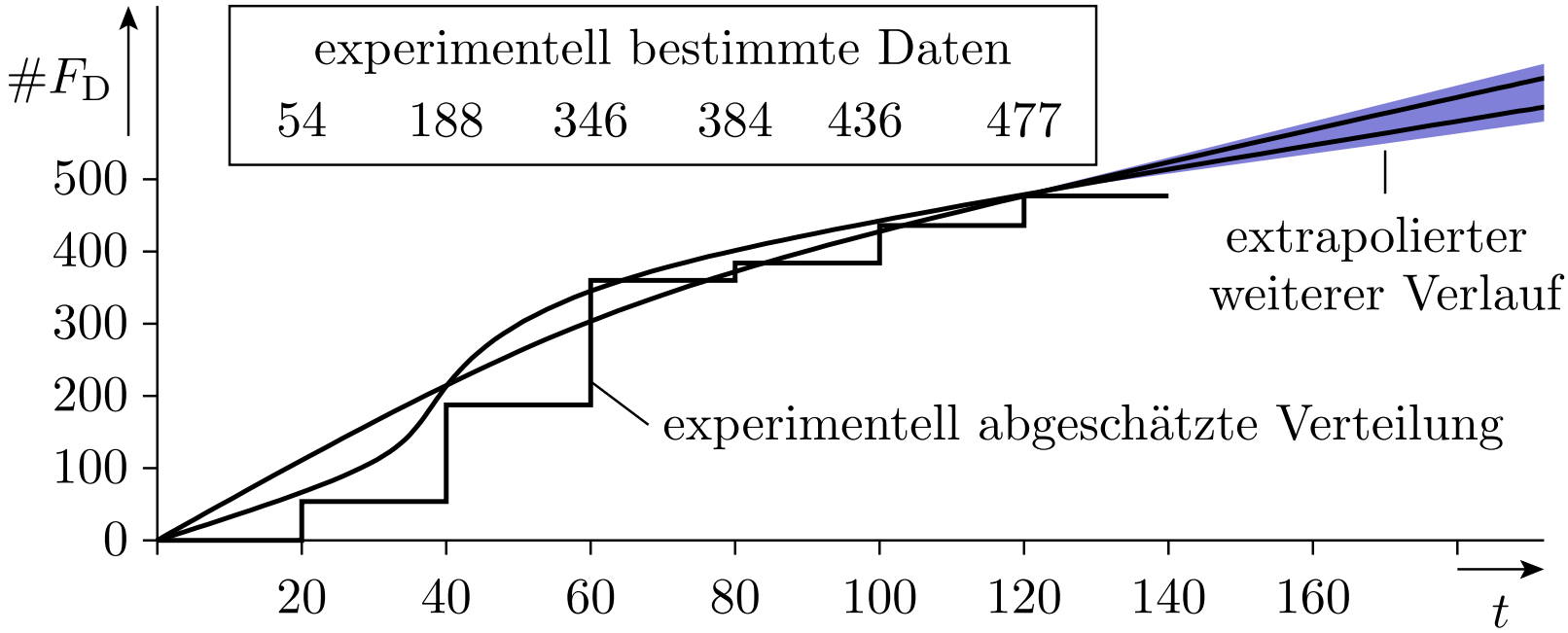
Bei pareto-verteilter Inspektionszeit mehr als umgekehrt proportionale Effizienzabnahme mit der Inspektionszeit.

μ_{EFC}	Zu erwartende Effizienz in gefundene Fehler pro Mitarbeiterstunde.
$f_X(t)$	Dichtefunktion der Inspektionszeit.
μ_{F}	Zu erwartende Anzahl der vorhandenen Fehler.
$1\,\mathrm{h}$	Eine Mitarbeiterstunde.

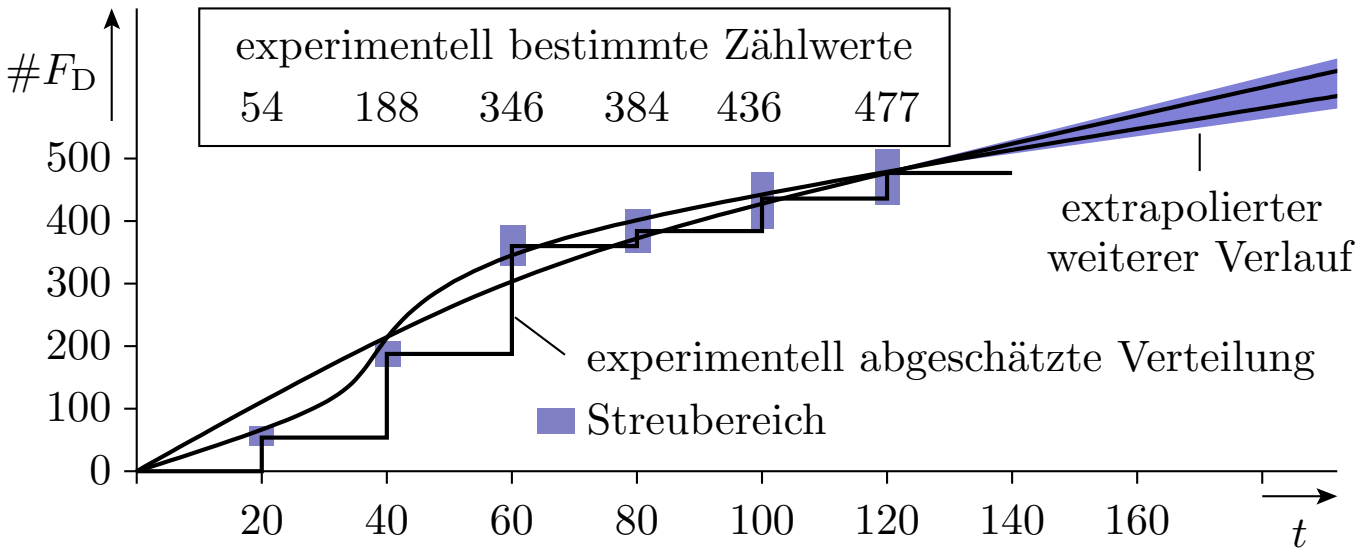
5.17 Experiment mit einem Inspekteur

Inspektion eines Buchmanuskripts* plus Beispielprogramme:

- Anzahl der gefunden Fehler in Abhängigkeit von der Inspektionszeit.



$\#F_D$ Anzahl der gefundenen Fehler (Number of detectable faults).
t Inspektionszeit in Mitarbeiterstunden.
* Bachelor-Arbeit von Yu Hong.



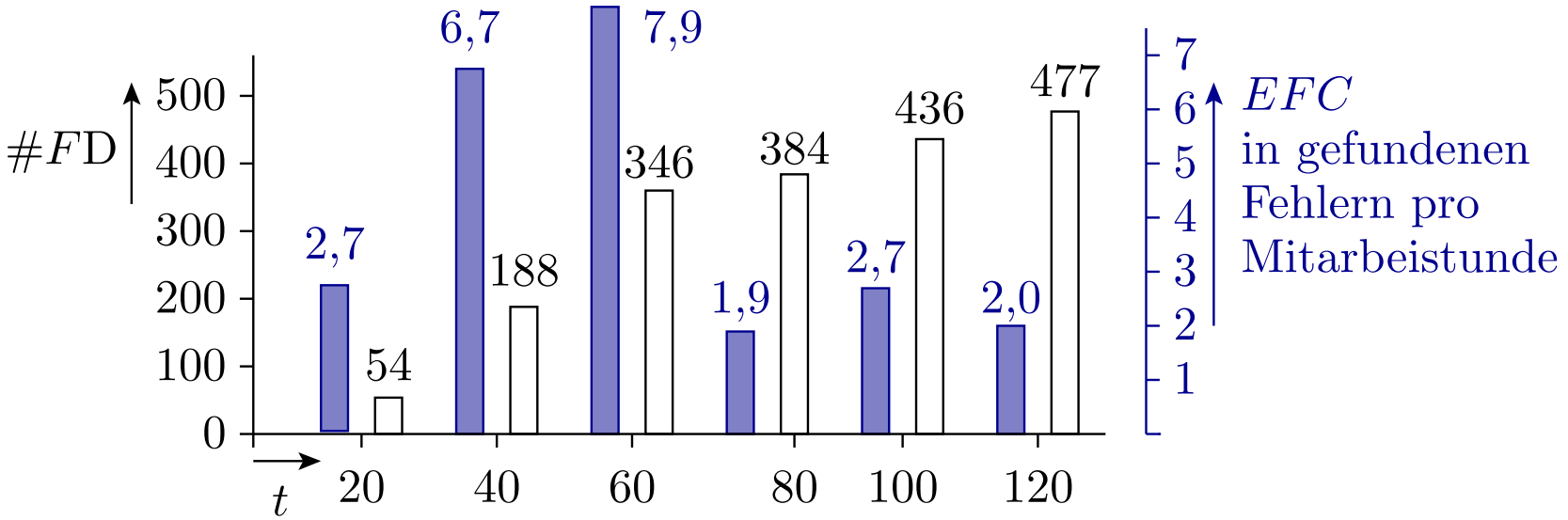
Die Breite des wahrscheinlichen Bereich von Zählwerten überschlagsweise proportional zur Wurzel aus dem Wert. Geschätzte Erwartungswertverläufe im Bereich der untersuchten Inspektionszeit unsicher und Extrapolation für längere Inspektionszeiten noch viel unsicherer. Im Experiment ist nicht einmal die mehr als umgekehrt proportionale Effizienzabnahme mit Inspektionszeit t, die für eine pareto-verteilte Nachweiszeit sprechen würde, zu erkennen.

Die nächste Folie untersucht den Zusammenhang zwischen Inspektionszeit und Effienz deshalb genauer.

$\#F_D$ Anzahl der gefundenen Fehler (Number of detectable faults).
t Inspektionszeit in Mitarbeiterstunden.

5.19 Untschiede Inspektion und Zufallstest

Die Effizienz wurde für jeweils 20 Inspektionsstunden aus dem Zuwachs der Anzahl der gefundenen Fehler geschätzt. Sie wuchst zu Beginn und nahm nach 60 bis 80 Stunden deutlich ab, obwohl erst die Hälfte der Fehler erkannt war.



Es gibt offenbar eine *»Anlernphase«*, mit zunehmender Effizienz, eine effiziente Phase und eine Ermüdungsphase mit geringer Effizienz.

$\#DF$	Anzahl der nachweisbaren Fehler.
EFC	Effizienz, gefundene Fehler pro Mitarbeiterstunde.

- Beim dritten und vierten mal »Lesen des Buchs und der Aufgabentexte« nahm im Experiment nicht nur die Effizienz, sondern auch die Zeit dafür deutlich ab, obwohl erst etwa die Hälfte der Fehler gefunden war.

Anzahl, wie oft gelesen	1	2	3	4
Anzahl der gefundenen Fehler	251	126	79	4
Zeitaufwand	50 h	70 h		

- Ein Mensch als Inspekteur ermüdet offenbar nach einiger Zeit und wird blind für Fehler, ...

These

Ein gute Inspektionstechnologie minimiert ineffiziente Anlernphasen bzw. nutzt sie zu Weiterbildung und vermeidet ineffiziente Ermüdungsphasen.

5.21 Inspektionstechniken

Arbeit *geschickt* auf mehrere Inspekteure mit unterschiedlichen Rollen verteilen. Diversität ausnutzen: »*Inspekteur ungleich Autor*«, »Vier Augen sehen mehr als zwei«, ...

Einteilung der Inspektionstechniken

- Review in Kommentartechnik: Dokumente Korrekturlesen und mit Anmerkungen versehen.
- Informales Review in Sitzungstechnik: Lösungsbesprechung in der Gruppe, Vier-Augen-Prinzip. Nimmt die Monotonie, steigert die Aufmerksamkeit, fördert den Wissensaustausch.
- Formales Review in Sitzungstechnik: Festlegen von Rollen (Leser, Moderator, Autor, Inspekteure) und Abläufen, ...

Stellschrauben einer Inspektionstechnologie:

- Lesegeschwindigkeit, Rollendisziplin,
- Vermeidung Langeweile und überhitzter Emotionen,
- Gruppennormen, ...

Zusammefassung

5.22 Anwenundung Güteabschätzung

Inspektion bedeutet in der Regel manuelle Kontrolle von Dokumentationen (Entwurfergebnisse, Testausgaben, ...) und wird eingesetzt, wenn keine automatisierten Kontrollen verfügbar.

Anzahl der nicht nachweisbaren Fehler und der Fehlerabdeckung:

- Capture-Recapture: Zählen der von zwei Inspekteueren einzeln und gemeinsam gefundenen Fehler. Schätzer:

(5.1) $$\hat{\mu}_{\mathrm{F}} = \left.\frac{\#F_1 \cdot \#F_2}{\#(F_1 \cap F_2)}\right|_{\mathrm{ACR}}$$

(5.2) $$\hat{\mu}_{\mathrm{FC}} = \left.\frac{\#(F_1 \cup F_2)}{\hat{\mu}_{\mathrm{F}}}\right|_{\mathrm{ACR}} = \left.\frac{\#(F_1 \cap F_2) \cdot \#(F_1 \cup F_2)}{\#F_1 \cdot \#F_2}\right|_{\mathrm{ACR}}$$

- Inspektionsmaterial mit eingebauten Kontrollfehlern. Fehlerabdeckung etwa Kontrollfehlerabdeckung. Fehleranzahl:

(5.3) $$\hat{\mu}_{\mathrm{FC}} = \left.\frac{\#F_{\mathrm{DM}}}{\#F_{\mathrm{M}}}\right|_{\mathrm{ACR}}$$

(5.4) $$\hat{\mu}_{\mathrm{F}} = \left.\frac{\#F_{\mathrm{D}}}{\hat{\mu}_{\mathrm{FC}}}\right|_{\mathrm{ACR}} = \#F_{\mathrm{D}} \cdot \left.\frac{\#F_{\mathrm{DM}}}{\#F_{\mathrm{M}}}\right|_{\mathrm{ACR}}$$

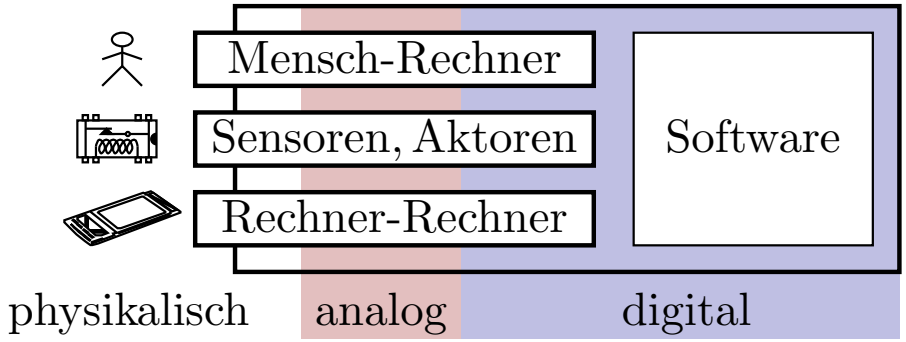
  Weniger Schätzfehler als Capture-Recapture.

5.23 Inspektionstechniken

Inspektion hat ineffiziente Einarbeitungs- und Ermüdungsphasen. Eine gute Inspektionstechnik vermeidet diese durch geschickte Abeitsorganisation.

Dynamische Tests

5.24 Typische Struktur eines IT-Systems

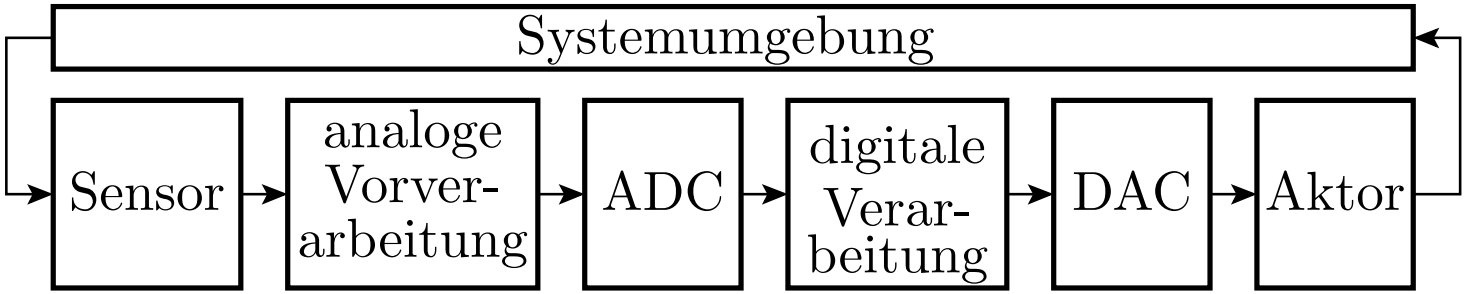


Ein IT-System kommuniziert mit der Welt über analoge Signale, arbeitet aber intern überwiegend digital. Getestet werden idealerweise alle Funktionsbausteine einzeln und hierarchisch aufsteigend die daraus zusammengesetzten Funktionsblöcke bis zum Gesamtsystem.

Einteilung der Tests nach der Art der Ein- und Ausgabe:

- physikalische Signale (Weg, Druck, Kraft, ...),
- elektrische analoge Signale (Spannung oder Strom),
- digitale Signale (Bit- und Bitvektorfolgen),
- Daten ohne physikalische Repräsentation (Software).

5.25 Systemarchitektur und Aufgabenteilung



- Physikalische und analoge Größen werden praktisch nur vorverarbeitet, gewandelt und ausgegeben.
- Die Informationsverknüpfung und Speicherung erfolgt digital.
- Komplexe Systeme haben progr. digitale Hardware, z.B. Rechner.
- Berechnungen in Schritten mit Fallunterscheidungen, Schleifen, ... in Software, z.B. auch die Fehlfunktionsbehandlung.

Diese Struktur hat sich auch aus Testbarkeitsgründen so entwickelt.

Ein IT-System und alle sein Komponenten sind so zu entwerfen, dass alles ausreichend getestet werden kann.

Physikalisch

5.26 Test mit physikalischen Ein- und Ausgaben

1. Die Bereitstellung und Messung physikalischer Ein- und Ausgaben (Weg, Kraft, Beschleunigung, Temperatur, auch elektrische Signale, ...) verlangt spezielle, oft teure Prüftechnik.
2. Diese Prüftechnik hat begrenzte Funktionalität: Wertebereich, Bandbreite, bereitstellbare Signalformen, Messfehler, ...
3. Messaufbauten sind weitere Fehlerquellen: Fremdeinwirkungen, Störsignale Rauschen, ...
4. Systeme mit physikalischen Ein- und Ausgaben dienen of zur Steuerung oder Reglung und müssen auch im Zusammenwirken mit ihrer Umgebung getestet werden.

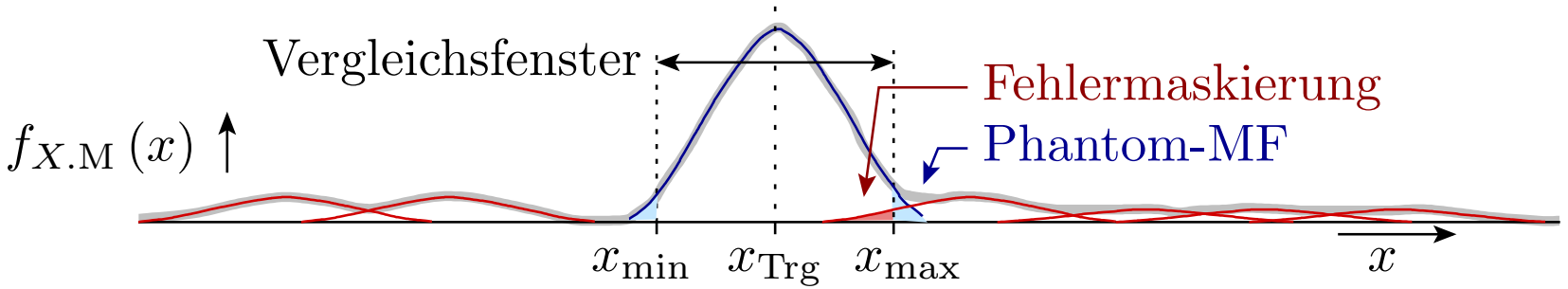
Es gibt für ausgewählte Prüf- und Messaufgaben gute Lösungen, für andere nicht.

Physikalische Verarbeitung nur:

- wenn es die Möglichkeit gibt, sie ausreichend zu testen und
- wenn nicht durch digitale Verarbeitung ersetzbar.

5.27 Kontrolle von Merkmalen

$f_{X.\mathrm{M}}(x)$ Dichtefunktion der Mischverteilung aus
der Verteilung der korrekten Werte
den Verteilungen der Werte für unterschiedliche Fehler



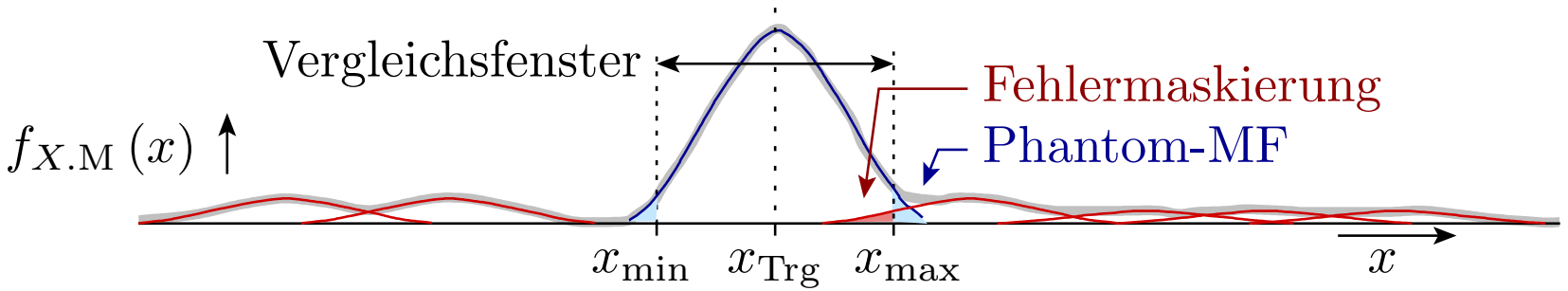
Aus Messwerten werden zu kontrollierende Merkmale gebildet:

- Verstärkung, Klirrfaktor (Maß der Linearität),
- Rauschen, Bandbreite, Sprungantwort, ...

und die Merkmalswerte einem Fenstervergleich unterzogen.

Systematische Verfälschungen der Signalerzeugung, Ankopplung und Messung beeinflussen die Sollwerte, zufällige die erforderliche Vergleichsfensterbreite bei gleicher Phantomfehlfunktionsrate.

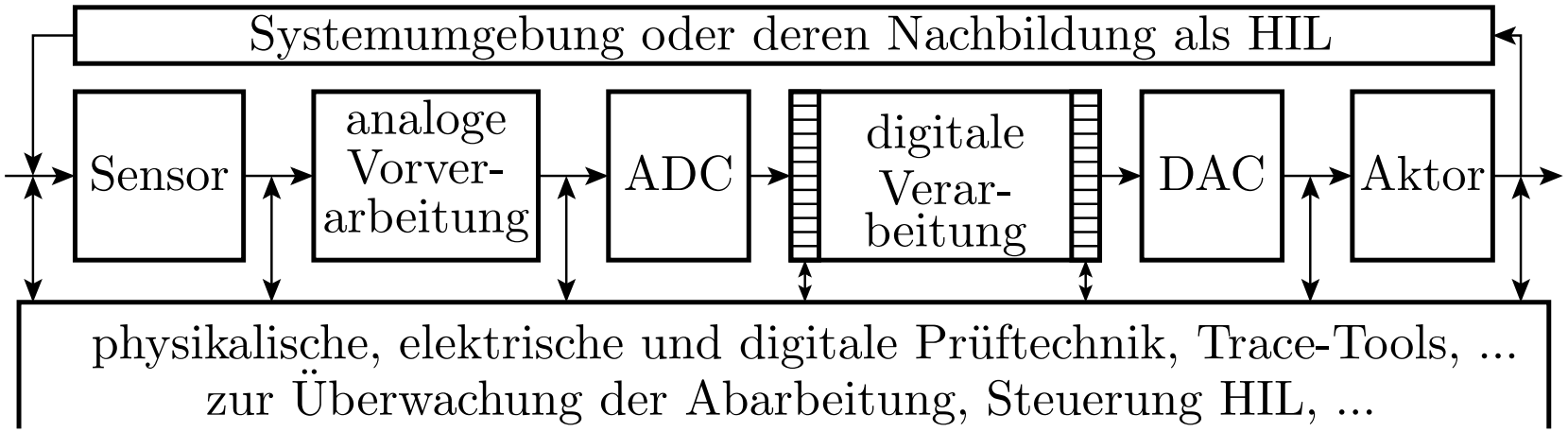
$f_{X.\mathrm{M}}(x)$ Dichtefunktion der Mischverteilung aus
der Verteilung der korrekten Werte
den Verteilungen der Werte für unterschiedliche Fehler



Die Varianzen zufälligen Einflüsse der einzelnen Komponenten der Testdatenerzeugungs, Übertragungs- und Auswertungsketten addieren sich und die Standardabweichungen addieren sich nach Pythagoras. Die einzelnen zufälligen Einflüsse dürfen nur gering sein im Vergleich zur Vergleichsfensterbreite und damit zum Toleranzbereich der betrachteten Parameter.

Prüftechnik muss viel genauer arbeiten als das zu testende System.

5.29 Möglichkeiten der Testdurchführung



Ganzheitlicher Test in der Systemumgebung:

- Überwachter Betrieb in Systemumgebung,
- HIL (Hardware in the Loop), Attrappe oder Simulationder Systemumgebung.

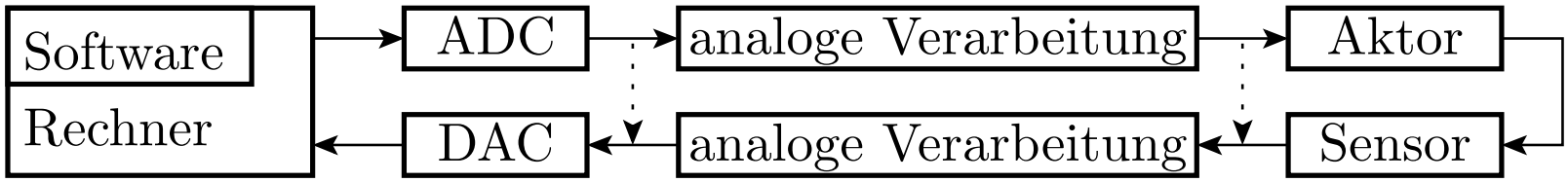
Ganzheitlicher Test mit Eingabevorgabe ohne Systemumgebung:

- Signalgeneratoren für Eingabe, Ausgabeaufzeichung, ...

Isolierter Test der physikalischen bzw. analogen Ein- und Ausgabe:

- spezielle Testmodie für Anschluss externer Prüftechnik,
- Loop-Test (testspezifische Signalflussumschaltung).

5.30 Loop-Test



- Messung der Aktorausgaben mit den Sensoren des Systems bzw. der analogen Ausgaben über die analogen Eingänge.
- Isolierter Test der Übertragungsfunktionen digital $\Rightarrow$ analog $\Rightarrow$ [physikalisch $\Rightarrow$ analog] $\Rightarrow$ digital.
- Fenstervergleich der Ergebnisswerte mit den Vorgabewerten,
- Einbindbar in Selbsttests.
- Die Wandler, Sensoren und Aktoren, müssen dafür deulich genauer arbeiten als das zu testende System, von dem sie ein Teil sind (höhere Bitauflösung, höhere Bandbreite, ...).

5.31 Hardware-in-the-Loop (HIL)

Die Systemumgebung

- Reglungsstrecke,
- physikalischer Prozess,
- Verhaltes eines Autos oder eines Flugzeugs, ...

werden für den Test durch eine Attrappe oder Simulation ersetzt.

Erlaubt gründlichere Untersuchung des Systemsverhaltens, insbesondere gefährlicher Situationen, z.B. für das Verhalten von Flugzeugen in provozierten Fehlersituationen.

5.32 Prüfgerechter Entwurf

- Die externe Prüftechnik muss anschließbar sein, mechanisch (Stecker, Kontaktflächen) und elektisch (Eingangswiderstand, ...),
- Triggersignal für Aufzeichungsbeginn, ...
- Testmodie z.B. für Loop-Tests müssen steuerbar sein, ...
- Schnittstelle für den HIL, der auch eine Simulation sein kann, die über eine Datenschnittstelle statt über Signalverlverläufe kommuniziert.
- Isolationsmöglichkeit für isoliert zu testende Bausteine.

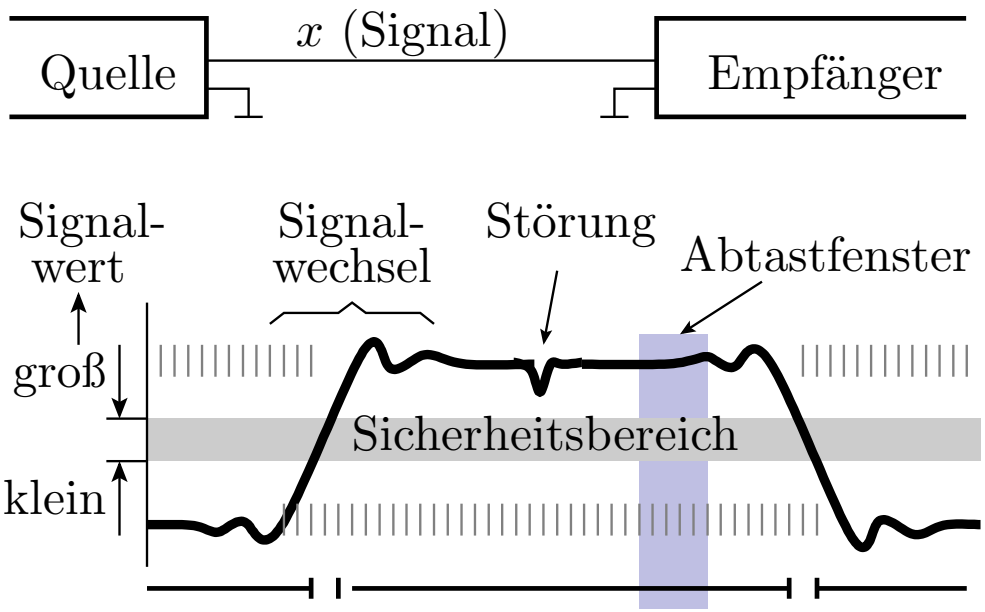
Man kann sich viel verbauen. Überlegungen zum Test müssen den gesamten Entwurfsprozess ab der Spezifilation begleiten.

Prüfgerechter Entwurf als Leitfaden zur vorbeugenden Vermeidung von Testbarkeitsproblemen umfasst

- Sammlungen gut funktionierender Lösungen,
- typische Probleme und Workarounds,
- Checklisten, was dabei nicht vergessen werden darf, ...

Digitale Bausteine

5.33 Digitale Verarbeitung (Folie 1.102)



Signalwert	
erweitert	binär
$x = 1$	$x = 1$
$x = \mathrm{X}$	nicht darstellbar
$x = 0$	$x = 0$

Gültigkeitsfenster

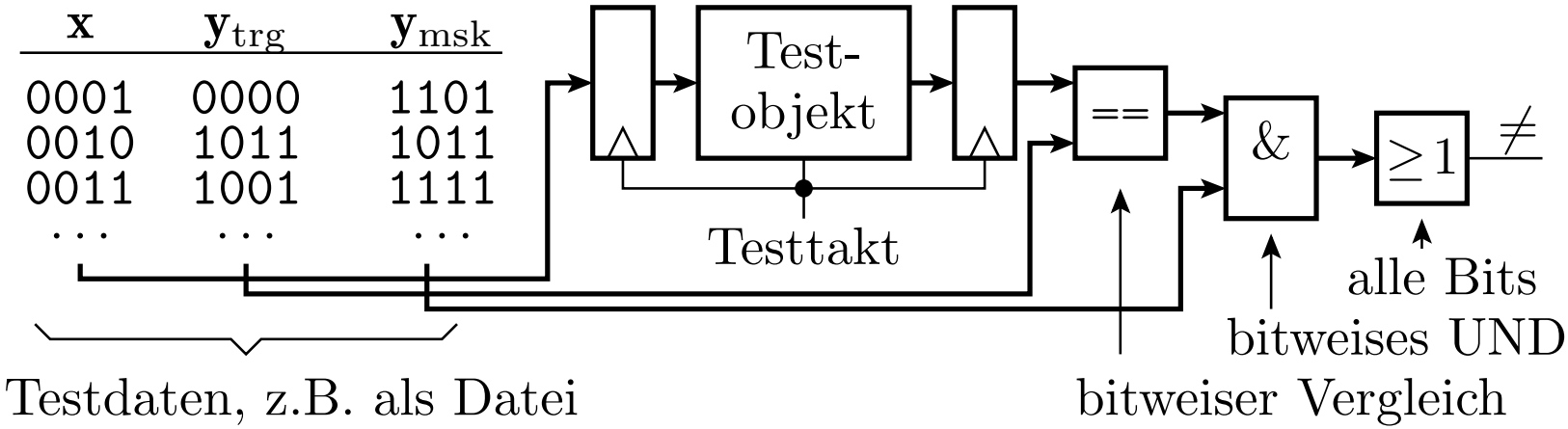
Informationsweitergabe durch Bits:

- Werteunterteilung in groß, klein und ungültig,
- Abtastung im Gültigkeitsfenstern, ...

Immun gegen Störungen und Messfehler. Einfacherere Prüftechnik, ...

0, 1, X Logische Signalwerte für klein, groß und ungültig.

5.34 Test mit digitalen Ein und Ausgaben

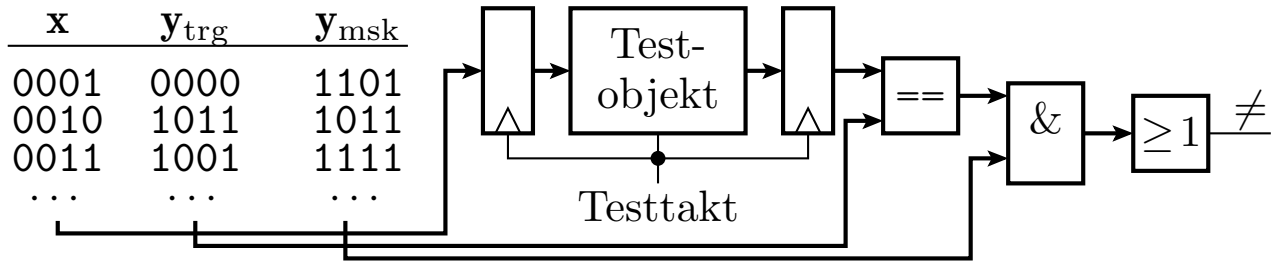


Wiederhole für jeden Taktschritt des Tests:

- Bereitstellung logischer Eingabewerte und
- Abtasten und Auswertung der vorherigen Ausgaben.

Auswertung vorzugsweise Vergleich mit Sollwerten, Ausmaskierung von der Ausgabebits mit undefinierten Werten.

$\mathbf{x}$	Testeingaben.
$\mathbf{y}_{\mathrm{trg}}$	Sollwerte der Testausgaben.
$\mathbf{y}_{\mathrm{msk}}$	Maskenwerte zum Ausschluss von Testausgaben vom Soll-Ist-Vergleich.
$\neq$	Vergleichsfehler.



Abwandlungen und Ergänzungen der Testanordnung:

- Kontrolle der Signalverzögungen durch Ergebnisabtastung mit mehrfacher Aufzeichnungsfrequenz.
- Tester auch als Kombination Signalgenerator Logikanalysator.

Einige Regeln des prüfgerechten Entwurfs:

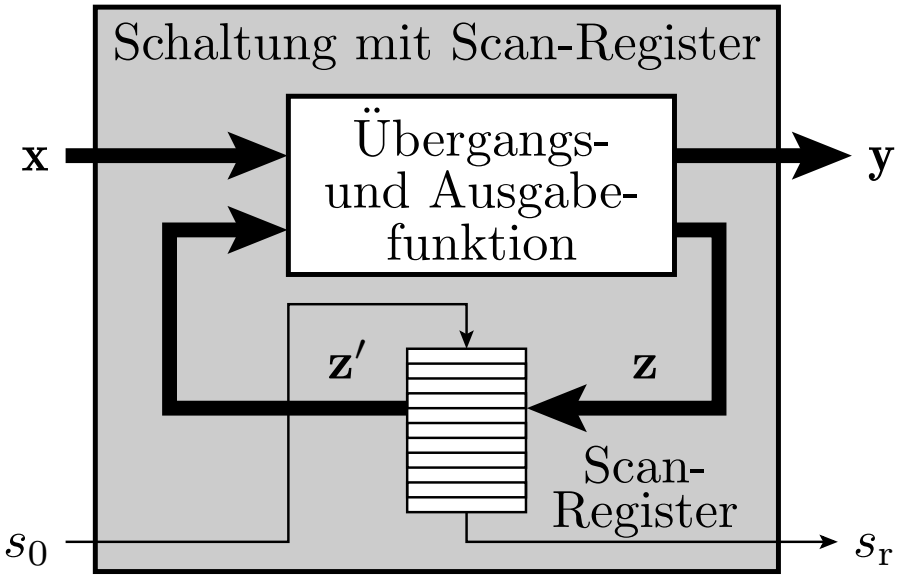
- Initialisierungsmöglichkeit für Speicherzellen.
- Isolierter Test komplexer Funktionsbausteine, insbesondere von großen Speicherblöcken (Abschn. 6.2.7),
- Scan-Verfahren (Abschn. 6.2.6).

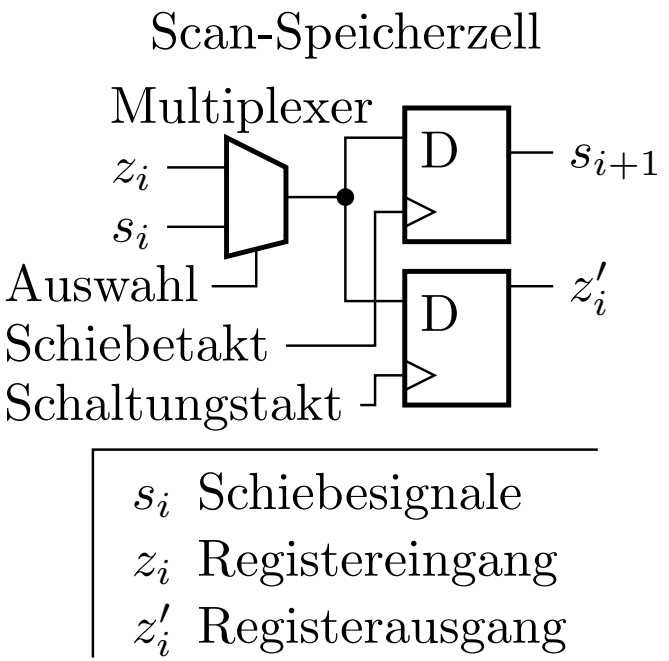
Kostenfaktoren bei Nutzung externer Prüftechnik:

- Anschlussanzahl, Größe der Testdatenspeicher,
- zeitliche Genauigkeit und Testgeschwindigkeit.
- Anschluss der Prüftechnik.

Alternative zu externer Prüftechnik ist Selbsttest (Abschn. 6.3).

5.36 5.36 Scan-Verfahren

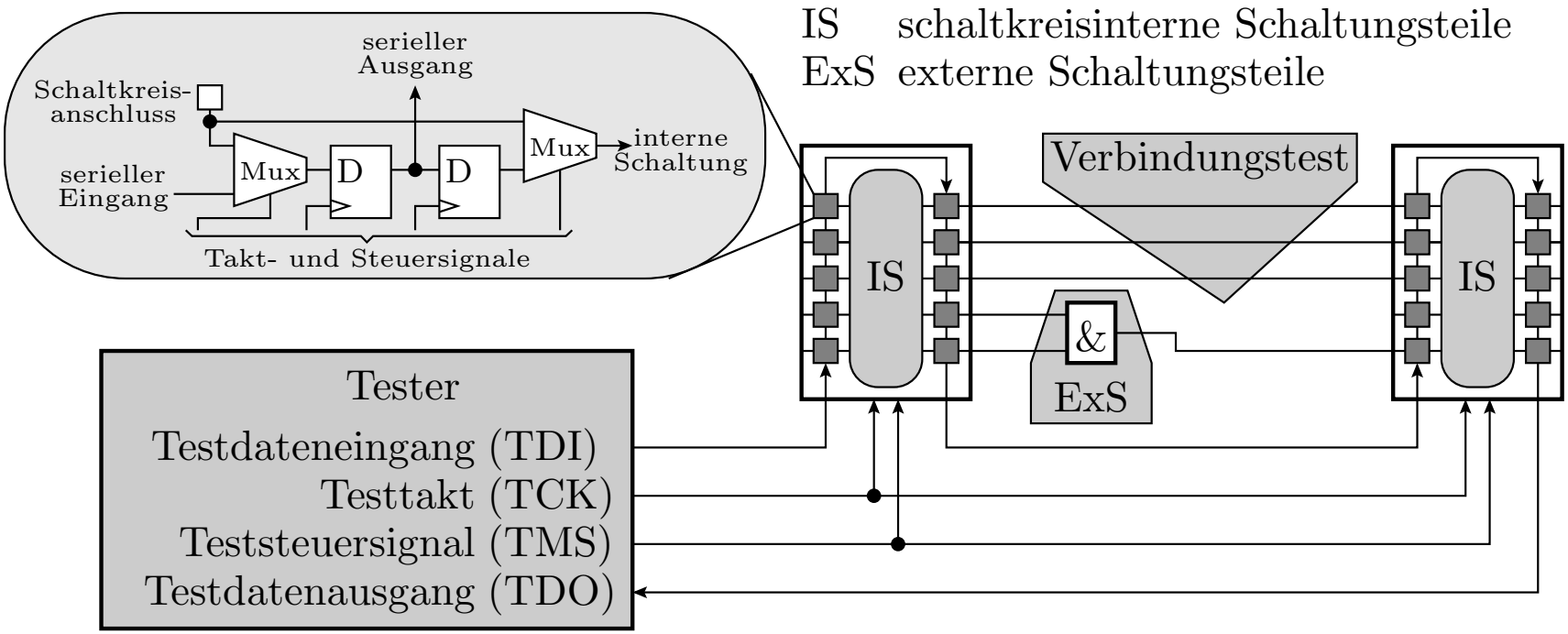




Fehlender Lese- und Schreibzugriff auf interne Signale von Schaltkreisen, insbesondere Zustandsbits, können die Erstellung von Tests erheblich erschweren. Problemumgehung mit Scan-Registern (Abschn. 6.2.6). Im Bild ist jede Speicherzelle um einen Multiplexer und eine Schiebezelle erweitert zur Bereitstellung der Funktionen:

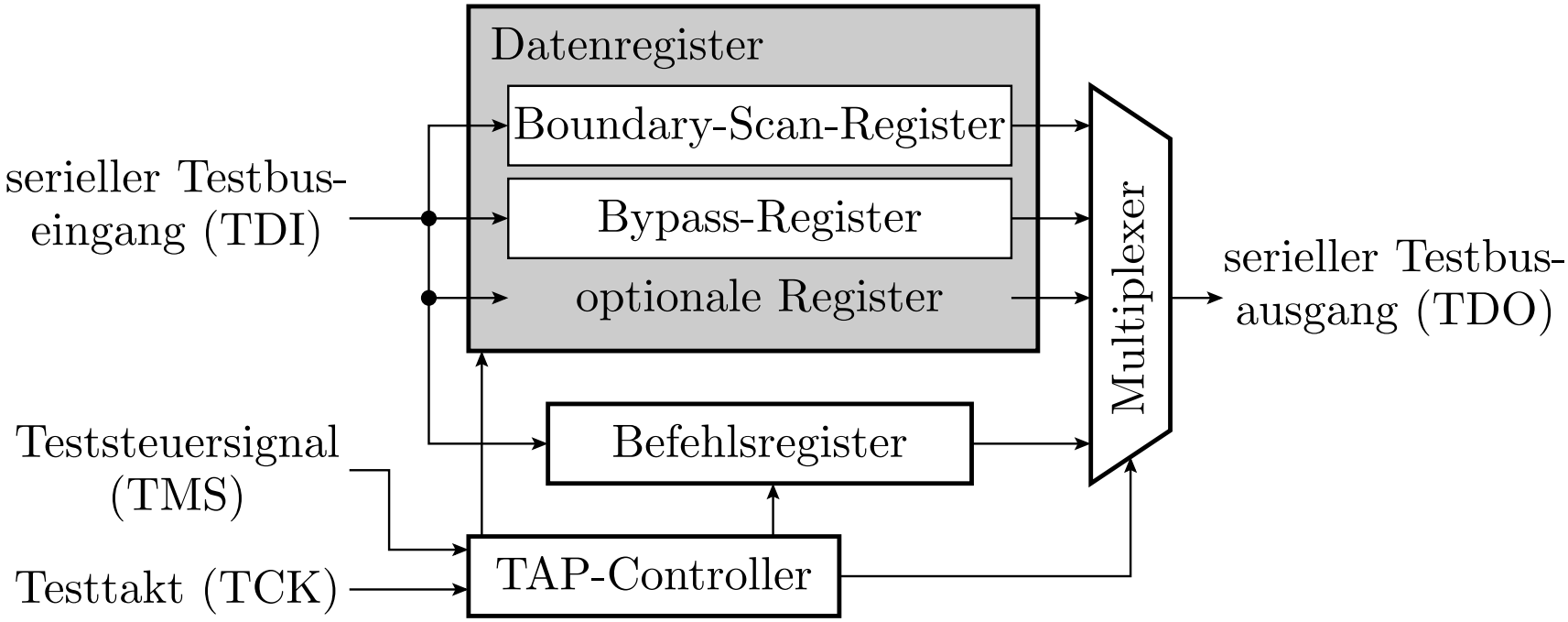
- Capture: Übernahme aus der Schaltung in die Schiebezellen,
- Shift: serielles Auslesen und neu beschreiben und
- Update: Übergabe seriell eingelesene Daten an Schaltung.

5.37 Boundary-Scan



Mit zunehmender Packungsdichte wird die mechanische Kontaktierung der Leitungen auf Baugruppen immer schwieriger (Abschn. 6.4). Heute meist ersetzt durch Boundary Scan, einem Scan-Register an den Schaltkreisanschlüssen, mit denen die logischen Pegel an den Schaltkreisanschlüssen gesetzt und gelesen werden können. Damit sind Verbindungen und Restschaltungen ohne Boundary-Scan isoliert testbar.

5.38 JTAG-Testbus



Der JTAG-Standard standardisiert zum Boundary-Scan-Prinzip einen Testbus, über den auch zahlreiche weiterer Test-, Debugg- und Programmierfunktion steuerbar sind. Eine JTAG-Testbusimplementierung umfasst:

- den TAP- (Test Access Port) Controller
- ein Befehls- und mehrere Testdatenregister.

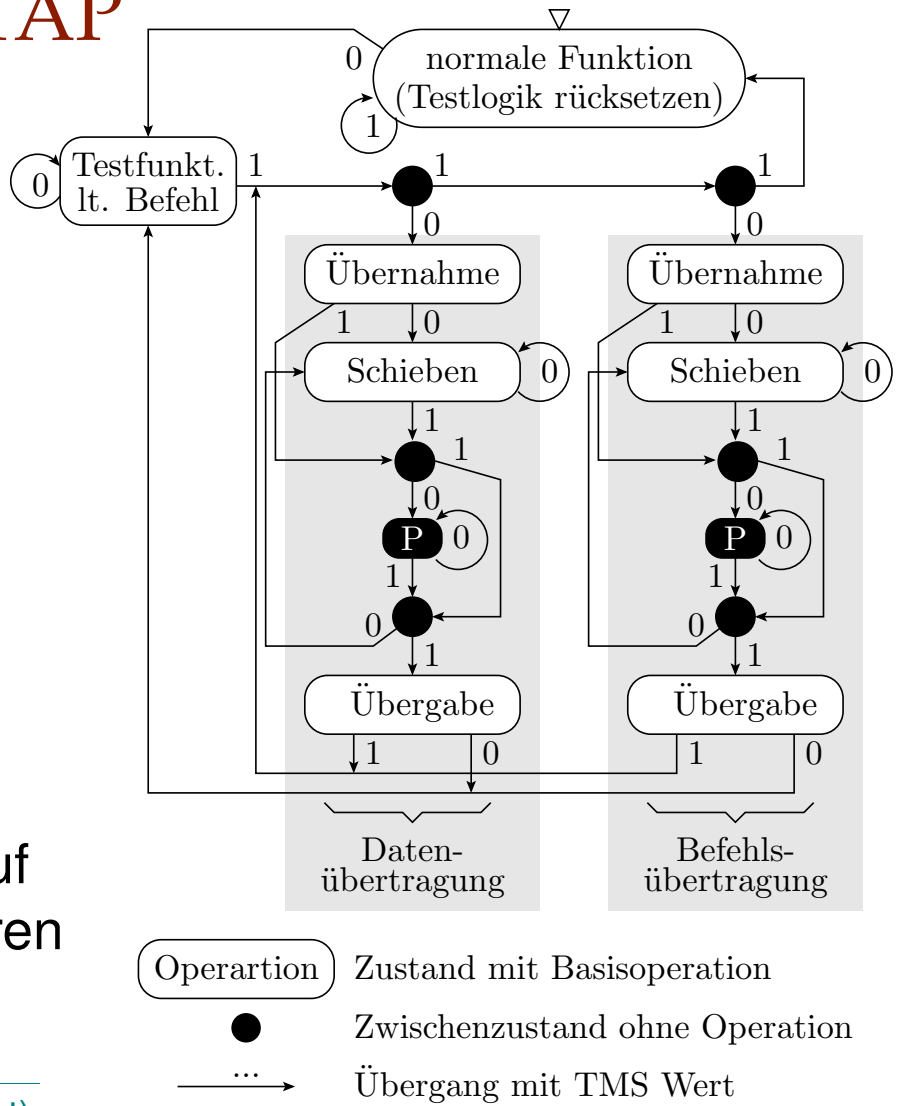
5.39 Busprotokoll und TAP

TAP-Controller

- Automat mit 16 Zuständen
- Kantenauswahl über TMS zum Wechsel zwischen 8 Basisoperationen.

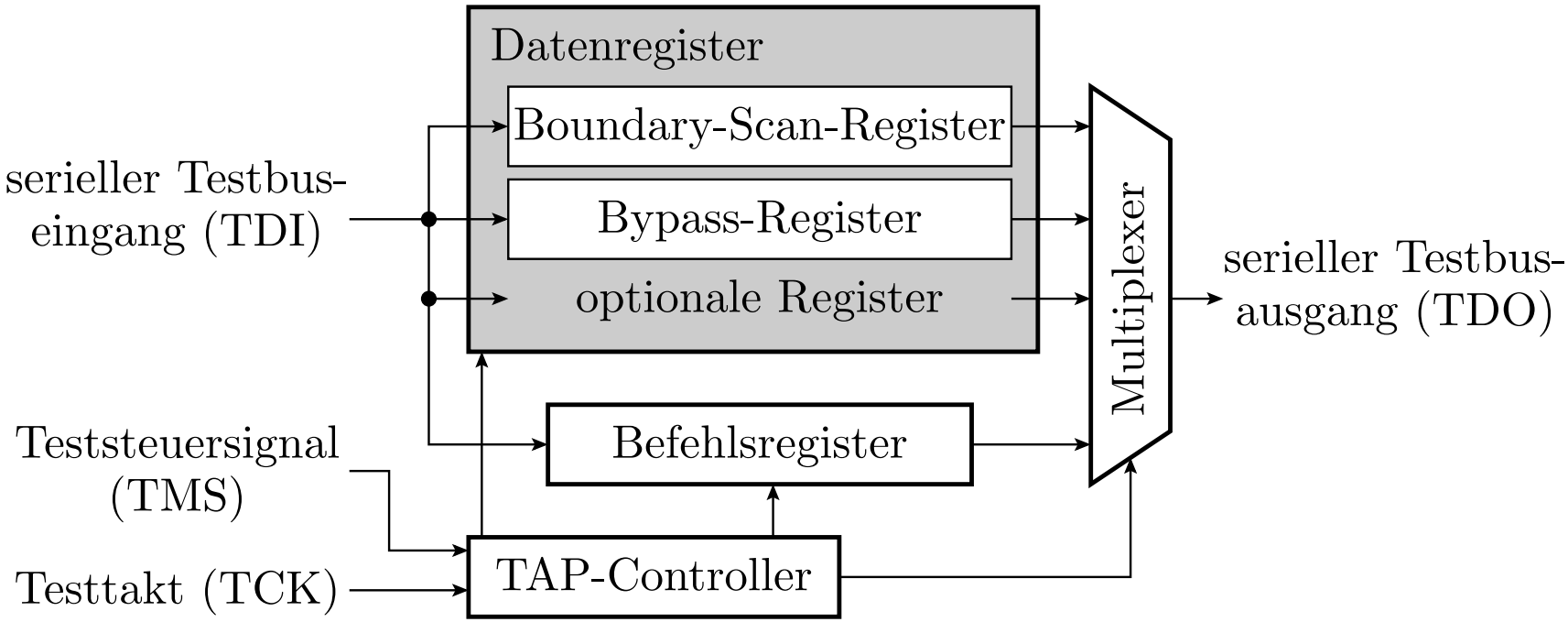
Typischer Testbeginn:

- Befehlsregister lesen (Kontrolle JTAG-Kette*),
- Bauteilnummern lesen (Bestückungskontrolle),
- Einen Teil der Schaltkreise auf Bypass setzen. Für die anderen ein Datenregister auswählen.



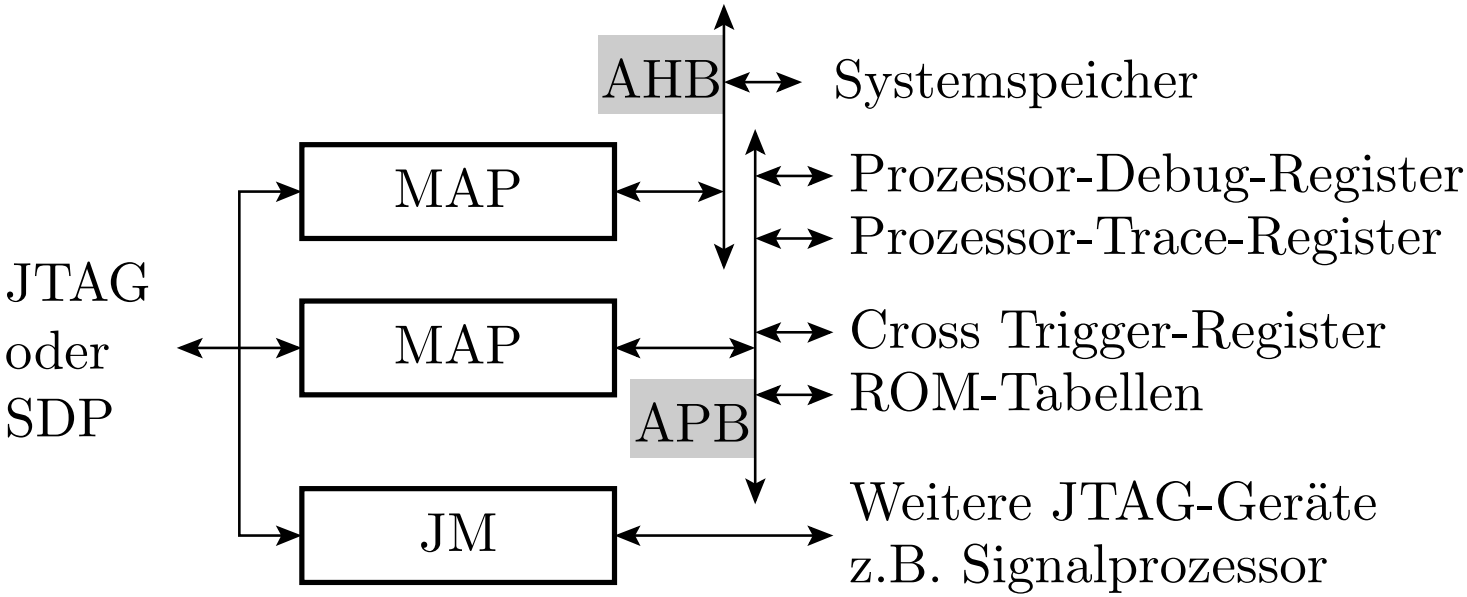
TAP	Testbussteuerung (Test access port).
TMS	Testoperationsauswahlsignal (Test mode select signal).
*	Zwei Bits des Befehlsregisters liefern beim Lesen null und eins.

5.40 Optionale Testregister und Funktionen



- Lesen einer Hersteller- und Bauteil-ID (Bestückungstest),
- Flashen von Programm- und Konfigurationsspeichern,
- Lese- und Schreibzugriff auf Programmdaten,
- Debugger-Register für Schrittbetrieb, Haltepunkte, Trace-Steuerung, ...

5.41 ARM-Testbusarchitektur



Die ARM-Testbusarchitektur definiert zusätzlich über den AHB Funktionalität für Lese- und Schreibzugriff auf den kompletten Speicher und über den APB auf die Debug-Register, ROM-Tabellen, ...

ARM	Verbreitetste Mikroprozessor-Architektur für Embedded-Systems (Smartphones, ..).
SDP	Serieller Debug-Port.
JM	JTAG-Master für weiterer Rechnerbausteine auf dem Chip, z.B. Co-Prozessoren.
MAP	Speicherzugriffsport mit serieller Adress- und Datenübergabe.
AHB, APB	Advanced High Performance Bus, Advanced Peripherial Bus.

5.42 Test- und Debug-Register (ARM)

- 32-Bit Debug-Control und Status-Register (DSCR).
- Instruction-Transfer-Register (ITR): 32 Befehlsbit + ein Statusbit, zur Ausführung von Prozessorbefehlen im Debug-Modus.
- Debug Communications Channel (DCC): 32-Bit-Datenwort + 2 Statusbits für den bidirektionalen Datentransfer mit Prozessorkern.
- Embedded Trace Module (ETM): 7 Adressbits + 32-Bit-Datenwort + 1 R/W-Bit zur Steuerung von Trace-Operationen*.
- Debug-Modul: 7 Adressbits + 32-Bit-Datenwort + 1 R/W Bit. Zugriff auf die Register für Hardware Breakpoints, Watchpoints etc..

ARM Befehle für Zusammenarbeit Programm und Debugger:

- HALT für den Übergang in Debug-Modus, in dem über das ITR Befehle eingefügt werden können und
- RESTART zum Verlassen des Debug-Modus.

* Trace-Aufzeichnung entweder in einen eingebetteten Trace-Buffer (ETB) auf dem Chip oder Ausgabe über einen High-Speed-Port.

Software

5.43 Hardware und Software

Als digitale Schaltung lassen lassen sich gut realisieren:

- Register-Transfer-Funktionen, die in jedem oder den meisten Takt ausgeführen werden,
- Blockspeicher, Addierer, Multiplizierer, ...

Damit lassen sich auch programmierbare Strukturen realisieren:

- speicherprogrammierbare Steuerungen,
- Universal- und Spezialprozessoren,
- programmierbare Logikschaltungen, ...

Software benötigt programmierbare Hardware und kann kompliziertere Funktionen realisieren, nämlich alles, was beschreibar ist mit:

- Berechnungsfolgen, Fallunterscheidungen, Schleifen,
- Unterprogrammen, Rekursionen, Nebenläufigkeit, ...

Software-Bausteine haben vorzugsweise eine Service-Struktur, d.h. sie berechnen auf Anforderung aus Eingaben Ausgaben. Auch Software ist prüfgerecht zu entwerfen (Foliensatz 7).

5.44 Testrahmen

Softwaretest in der Regel ohne externe Prüftechnik. Zu testende Programmbausteine werden in einen Testrahmen eingebettet, der Testeingaben bereitstellt und Testausgaben auswertet.

Im einfachsten Fall ist der Testrahmen ein ausführbares Programm und das Testobjekt ein aufgerufenes Unterprogramm.

Wünschenswerte Software-Unterstützung: Erstellung und Verwaltung der Testrahmen, »operationsprofilbasierte Testdatenauswürfeln« (Abschn. 3.2.4 *Operationsprofil*), ...

Sollwertbereitstellung läßt sich unterstützen durch inspektionsgerechte Istausgabeaufzeichung, Inspektion und anschließende Verwendung als Sollwerte (Abschn. 5.1).

Hilfreich für Fehlersuche (Abschn. 2.1.2 *Fehlerdiagnose & -isolation*):

- Schrittbetrieb, Haltepunkte,
- Lesen und Verändern von Variablen im Haltezustand,
- Trace-Aufzeichung von Variablen und Ausgabesignalen, ...

Ein Testobjekt und sein Testrahmen

Beispieltestsobjekt: Unterprogramm zur Quadrierung:

```
uint32_t quad(int16_t a){ // Square calculation
  return (uint32_t)a * a; // Why with Typcast?
};
```

Testbeispiele sind Tupel aus Eingaben und Sollausgaben. Man kann dafür einen neuen Datentyp definieren:

```
typedef struct{
  int16_t x;                    // Input
  uint32_t y;                   // Target output
} test_t;
```

Ein »struct« ist eine Zusammenfassung aus bereits definierten Datentypen.

Testsatz

Eine Testsatz als Menge von Tests ist im einfachsten Fall ein initalisiertes Feld von Testbeispielen:

```
test_t testset[] ={{<Tupel1>},{...},...};
```

Testbeispiele für die Quadratberechnung:

```
test_t testset[] ={          // Input   Target output
  {0, 0},                    // 0x0000   0x00000000
  {1, 1},                    // 0x0001   0x00000001
  {9, 81},                   // 0x0009   0x00000051
  {-5, 25},                  // 0xFFFB   0x00000019
  {463, 214369},             // 0x01CF   0x00034561
  {0x7FFF, 1073676289}       // 0x7FFF   0x3FFF0001
};
```

Welche Testbeispiele signalisieren Fehler?*

* Vermutlich ergibt -5 konvertiert in uint32_t mal -5 keinen positiven Wert.

Testrahmen

Programm, das in einer Schleife alle Testbeispiele abarbeitet und die Ergebnisse kontrolliert oder zur Kontrolle ausgibt:

```
int main(){
 uint8_t idx, err_ct=0;
 uint32_t erg;
 for (idx=0; idx<6;idx++){
 erg = quad(testsatz[idx].x);// Target output
 if (erg != testsatz[idx].y) // Comparison
  err_ct++;                  // Failed test
 }                           // counter
}
```

Zur Untersuchung, welchere Test versagt und zur Fehlerlokalisierung:

- Unterbrechungspunkt vor dem Fehlerzähler.
- Bei jeder Fehlfunktion Halt am Unterbrechungspunkt.
- Trace-Aufzeichung Zwischenergebnisse vor Vergleichsfehler.
- Erfordert Harware-Unterstützung, wie (Folie 5.41 *ARM-Testbusarchitektur*).

Zusammenfassung

5.45 Zusammenfassung

Die Vielzahl der notwendigen Tests müssen durchführbar sein. Um das sicherzustellen, müssen Testentscheidung praktisch ab der Spezifikation erfolgen. Checks auf Testbarkeit begleiten dann den gesamten weiteren Entwurfsprozess.

Am meisten Sorgfalt erfordern die Tests mit physikalischen Signalen:

- Geeignete Prüftechnik verfügbar und an das Testobjekt anschließbar?
- Dasselbe für benötigte Attrappen oder Simulationsmodelle der zu steuernden Umgebung?
- ...

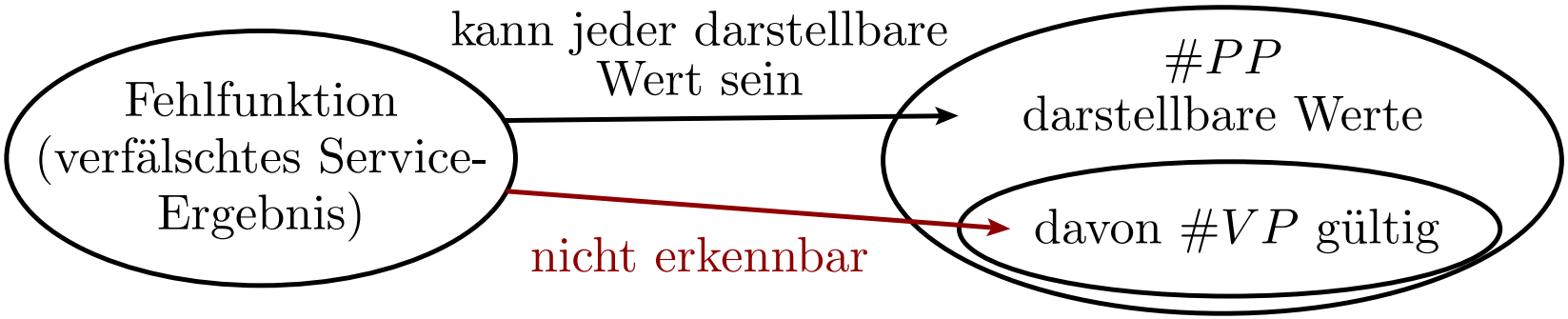
Der Test der digitalen Schaltungsteile ist wesentlich einfacher, aber nicht trivial: Bereitstellung der Eingabebits, Abtastung der Ausgabebits eine definierte Zeit später und Ausgabe.

Alternative zur Kontaktierung aller Signale auf einer Baugruppe bieten Boundary-Scan und schaltkreisinterne Scan-Register plus Testbus. Der Schritt und der Hardware-Zusatzaufwand von Scan-Registern zum Selbsttest ist nicht mehr groß (Abschn. 6.3).

Software realisiert die kompliziertesten Funktionen in IT-Systemen und lässt sich am einfachtsen testen. Im einfachsten Fall wird das zu testende Modul in einen Testrahmen eingebunden, der die Eingaben bereitstellt und die Ausgaben auswertet. Software-Unterstützung für die Testrahmenerstellung und Testdurchführung sowie Hardware-Unterstützung für Schrittbetrieb, Trace-Aufzeichnung wünschenwert.

Code & Kontrolle

5.47 Wiederholung Informationsredundanz



Wenn Verfälschungen weder unzulässige noch zulässige Werte bevorzugen und alle unzulässigen Werte erkannt werden, zu erwartende MF-Abdeckung:

(1.33) $$\mu_{\mathrm{MC}} = 1 - \frac{\#VP}{\#PP}$$

(1.35) $$\mu_{\mathrm{MC}} \geq 1 - 2^{-r}$$

Der Ausschluss einer Bevorzugungung zulässiger bzw. unzulässiger Werte verlangt eine geeignete Wertecodierung.

μ_{MC}, r	Zu erwartende Fehlfunktionsabdeckung, Anzahl der redundanten Bits.
$\#VP$	Anzahl der gültigen Bitmuster (Number of valid bit patterns).
$\#PP$	Anzahl der darstellbaren Bitmuster (Number of presentable bit patterns).

Fehlererkennende Codes

5.48 Prinzip der fehlererkennenden Codes

Umkehrbar eindeutige pseudo-zufällige Zuordnung der zulässigen Werte zu möglichen Werte.

Wenn die Verfälschungsursache »den Code-Schlüssel« nicht kennt, entstehen durch Fehlfunktionen weder bevorzugt zulässige noch unzulässige Werte. Damit gilt

(1.33) $$\mu_{\mathrm{MC}} = 1 - \frac{\#VP}{\#PP}$$

Beispiel: Multiplikation mit einer ganzzahligen großen Konstanten C

$$y = C \cdot x$$

Erkennungswahrscheinlichkeit

$$\mu_{\mathrm{MC}} = 1 - \frac{\#x}{\#x \cdot C} = 1 - \frac{1}{C} \tag{5.7}$$

Vor der Verarbeitung und Speicherung Multiplikation der zu überwachenden Werte mit C. Verfälschungen werden am Divisionsrest $y\%C \neq 0$ erkannt.

C Große ganzzahlige Konstante, bevorzugt eine Primzahl.

5.50 Polynom-Multiplikation und -division

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

Codierung

$$
\begin{array}{rr}
 & 10010101101 \\
\odot & 10011 \\
\hline
\oplus & 10010101101 \\
\oplus & 10010101101 \\
\oplus & 00000000000 \\
\oplus & 00000000000 \\
\oplus & 10010101101 \\
\hline
 & 1000111\,0\,0100111
\end{array}
$$

Decodierung

$$
\begin{array}{l}
1000111\,0\,0100111 : 10011 = 10010101101 \\
\oplus\ 10011 \\
\quad 10110 \\
\quad \oplus\ 10011 \\
\quad\ 10101 \\
\quad\ 10011 \\
\quad\ 11000 \\
\quad \oplus\ 10011 \\
\quad\ 10111 \\
\quad \oplus\ 10011 \\
\quad\ 10011 \\
\quad \oplus 10011 \\
\text{Rest:}\quad 0000 \qquad \text{(unverfälscht)}
\end{array}
$$

5.51 Warum Polynommultiplikation?

Die zu multiplizierenden Folgen lassen sich auch als Polynome des Operators z, der eine Bitverschiebung beschreibt, dargestellt:

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

Die Multiplikation »$\cdot$« und die Addition »$\oplus$« erfolgen bitweise modulo-2, also durch UND und EXOR. Potenzen z^i beschreiben Verschiebungen um i Bitstellen. Das Produkt beider Polynome

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

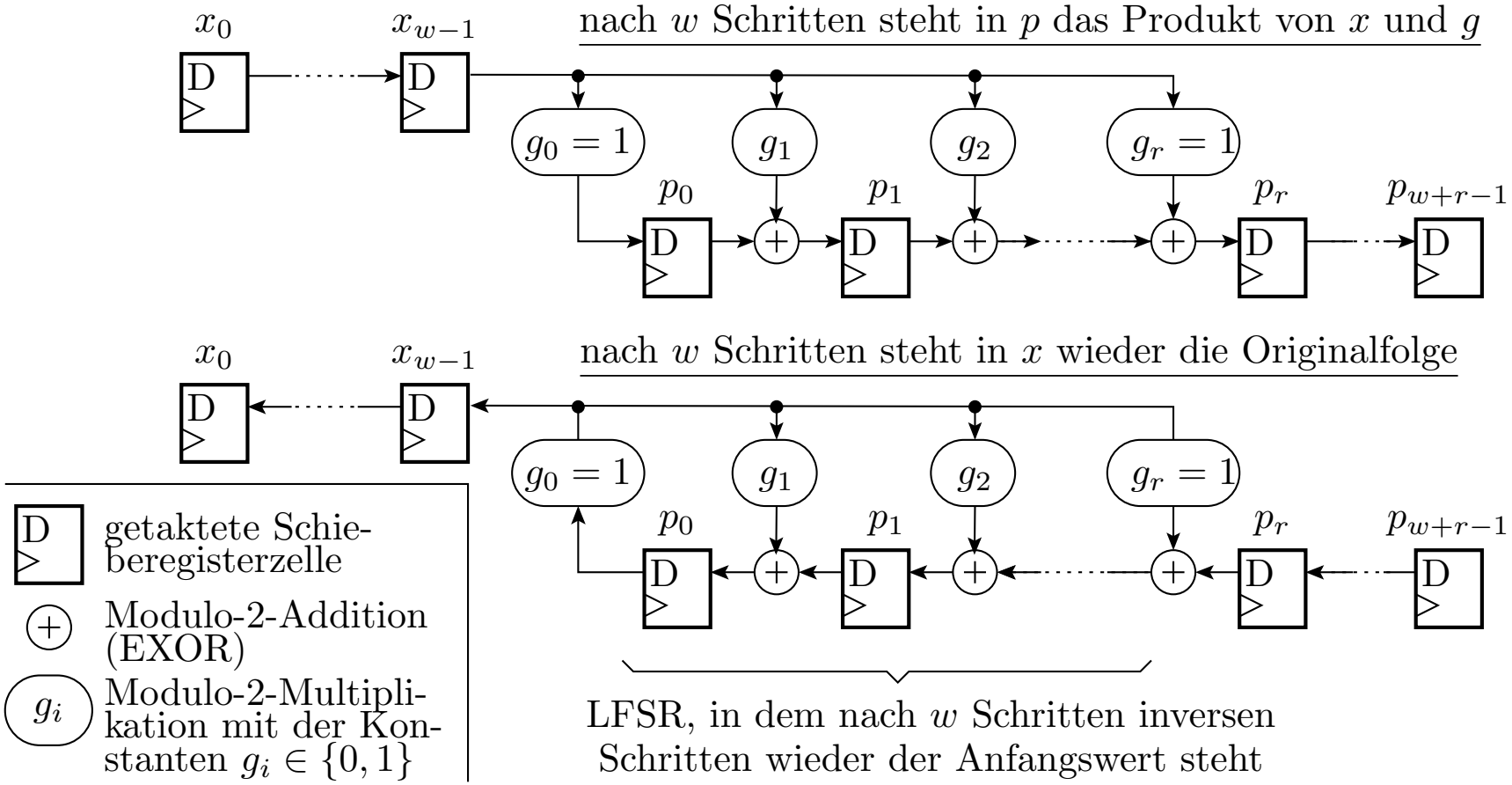
repräsentiert denselben Bitvektor, wie auf der Folie zuvor berechnet. Die Polynomdivision:

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

liefert ohne Rest das Polynom der Originalfolge.

z^i Operator für eine Verschiebung um i Bitstellen.

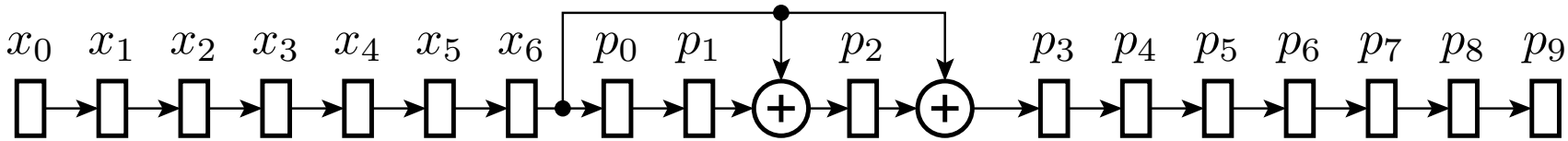
5.52 Linear rückgekoppelten Schieberegister



Schieberegister, vorwärts Multiplikation, rückwärts Division (modulo-2).

LFSR Linear rückgekoppeltes Schieberegister (Linear feeback shift register).

5.53 Zahlenbeispiel

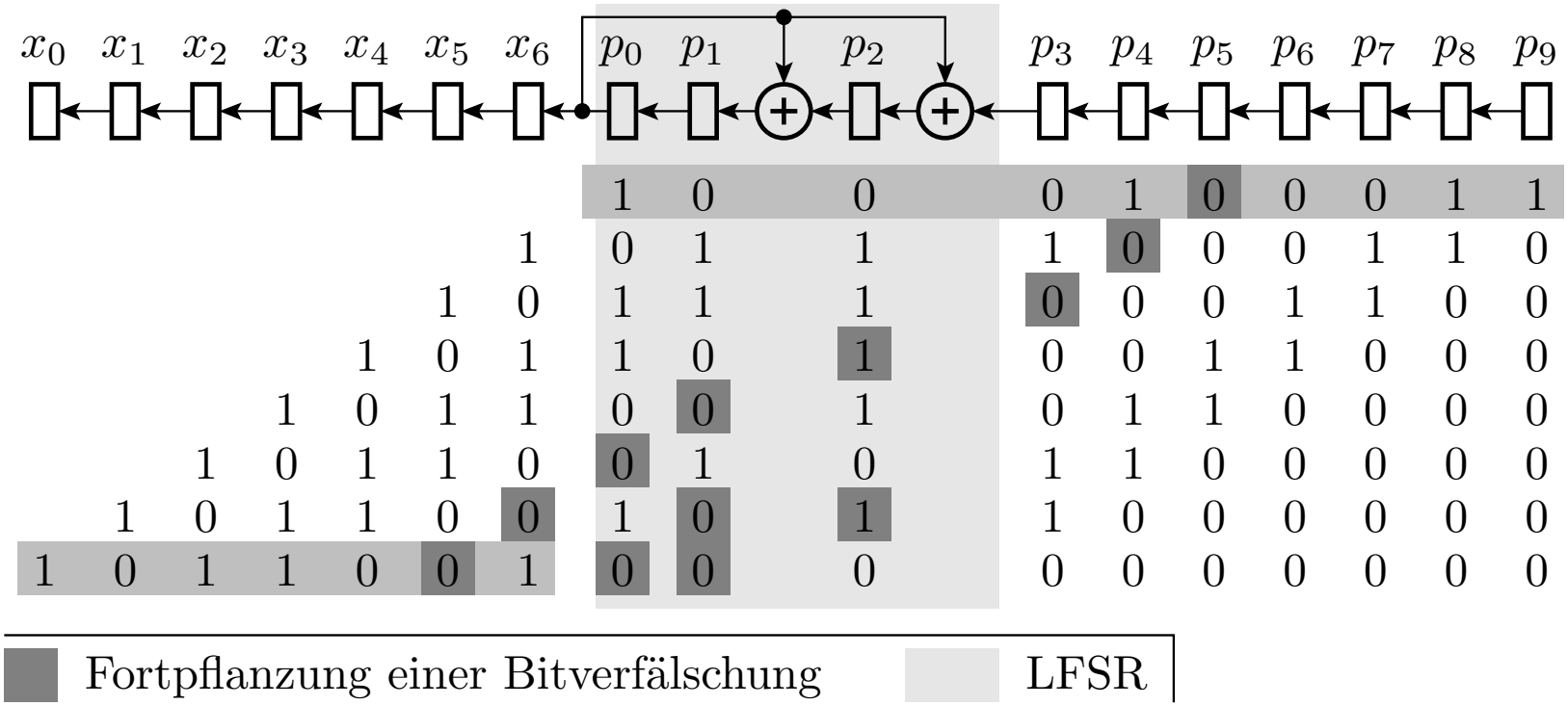


1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
	1	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0
		1	0	1	1	0	0	1	0	1	1	0	0	0	0	0
			1	0	1	1	0	0	1	0	1	1	0	0	0	0
				1	0	1	1	0	1	0	0	1	1	0	0	0
					1	0	1	1	1	0	0	0	1	1	0	0
						1	0	1	1	1	0	0	0	1	1	0
							1	0	0	0	1	0	0	0	1	1

Originalfolge:	1001101	$\left(x^6 \oplus x^3 \oplus x^2 \oplus 1\right)$
Generatorfolge:	1101	$\left(x^3 \oplus x^2 \oplus 1\right)$
Produktfolge:	1100010001	$\left(x^9 \oplus x^8 \oplus x^4 \oplus 1\right)$

Produktfolge um $r = 3$ redundante Bits länger, also 2^3 mal so viele darstelbare wie zulässige Werte. Pseudo-zufällig Umcodierung.

5.54 Rückgewinnung durch Polynomdivision

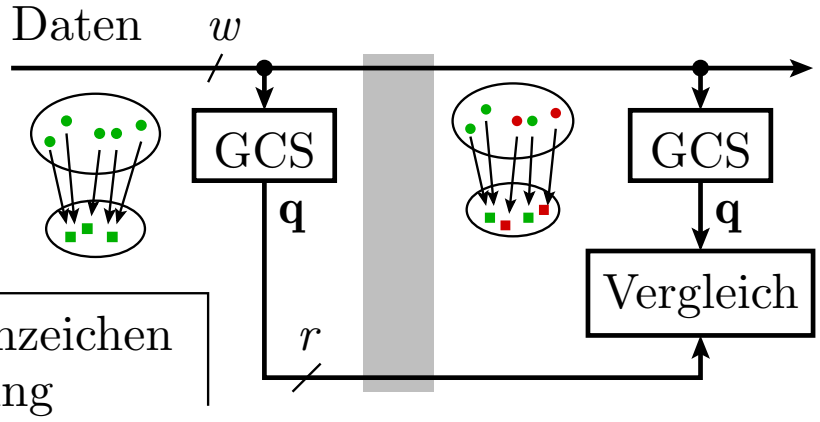


Im LFSR eingegangene Bitverfälschungen zirkulieren dort und werden durch weitere Bitverfälschungen nur mit Wahrscheinlichkeit von 2^{-3} ausgelöscht $(\mu_{\mathrm{MC}} = 1 - 2^{-3})$.

LFSR Linear rückgekoppeltes Schieberegister (Linear feeback shift register).
μ_{MC} Zu erwartende Fehlfunktionsabdeckung.

Prüfkennzeichen

5.55 Prüfkennzeichen



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

Für pseudo-zufällig gebildete Prüfkennzeichen gilt:

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

(1.35)
$$\mu_{\mathrm{MC}} \geq 1 - 2^{-r}$$

r Anzahl der redundanten Bits.
w Anzahl der Datenbits (Number of data bits).

5.56 Prüfsummen

Prüfkennzeichbildung durch Byteaddition oder EXOR:

einfache Genauigkeit	doppelte Genauigkeit	bitweises EXOR
1011 11	1011 11	1011
0010 2	0010 2	0110
1101 13	1101 13	1101
0100 4	0100 4	1100
(1) 1110 14 (+16)	0001 1110 30	1100

Bei »einfacher Genauigkeit« und »bitweisem EXOR« erscheint die Annahme »pseudo-zufällige Abbildung« gerechtfertigt*:

$$\mu_{\mathrm{MC}} = 1 - 2^{-4}$$

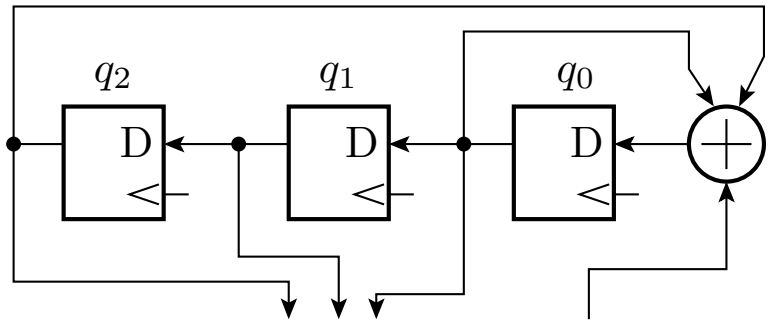
Bei »doppelter Genauigkeit« bilden sich Verfälschungen vorzugsweise auf die niederwertigen Bits ab:

$$1 - 2^{-4} \leq \mu_{\mathrm{MC}} < 1 - 2^{-8}$$

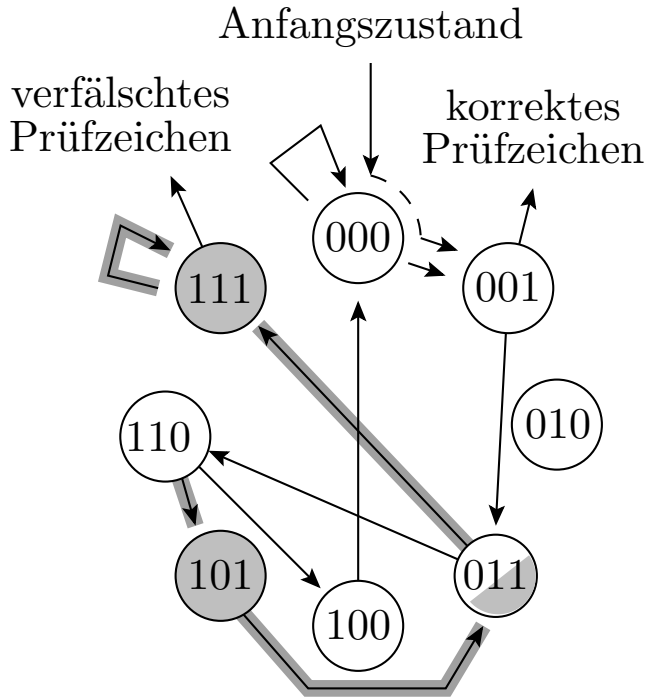
Kein Nachweis für vertauschte Summationsreihenfolge.

μ_{MC} Zu erwartende Fehlfunktionsabdeckung.

5.57 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	

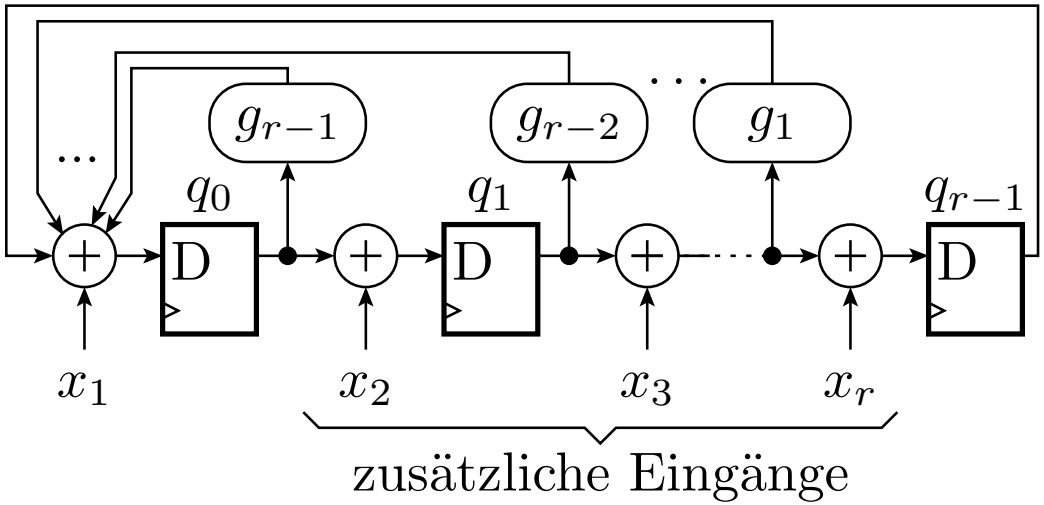


Im Beispiel hat das LSFR abweichend von Polynomdivision (Abschn. 5.3.1) eine zentrale Rückführung. Abbildung auch pseudo-zufällig.

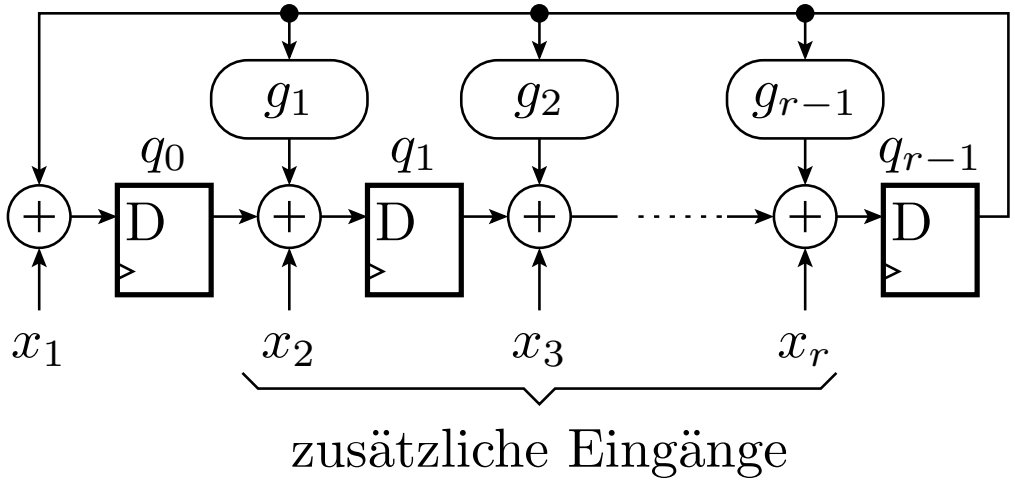
q_i	Registerbit i des linear rückgekoppelten Schieberegisters.
$\mathbf{q}$	Prüfkennzeichen (gesamter Registerzustand).
LFSR	Linear rückgekoppeltes Schieberegister (Linear feeback shift register).

5.58 Allgemeine Struktur von LFSR

Für die Bildung auf Prüfkennzeichen ist es nur wichtig, dass die Abbildung pseudo-zufällig hinsichtlich der zu erwartenden Verfälschungen erfolgt. Diese Eigenschaft hat auch ein rückgekoppeltes Schieberegister, bei dem in jedem Zeitschritt mehrere Eingabebits modulo-2 zu den Registerzuständen addiert werden.



x_i	Mehrere parallel eingespeiste LFSR-Eingabebits.
g_i, q_i	Rückführkoeffizienten und Registerbit i des linear rückgekoppelten Schieberegisters.
r	Bitanzahl des linear rückgekoppelten Schieberegisters.



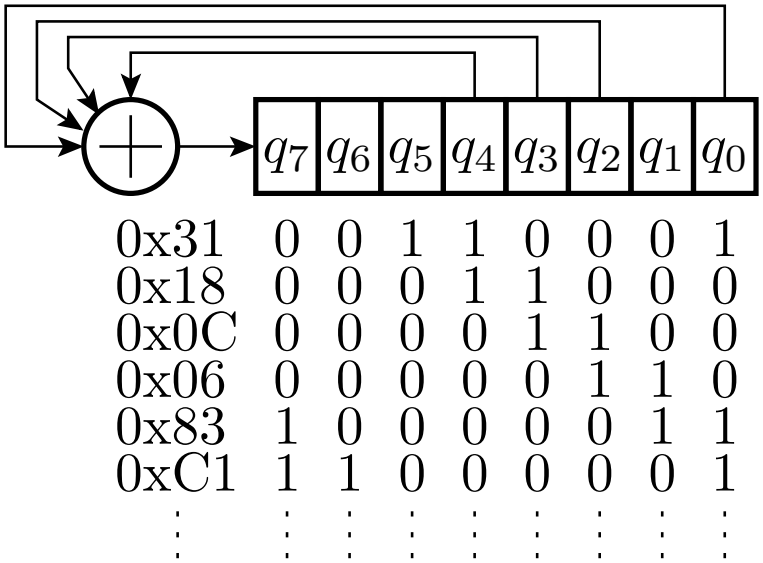
Die Rückführung darf dabei auch wie bei der Polynom-Division dezentral sein. Die Koeffizienten g_i der Rückführung, bei der Polynom-Division das Divisor-Polynom, bestimmen die autonome Zyklusstruktur*. Die autonome Zyklusstruktur ist bei zentraler und dezentraler Rückführung mit denselben Rückführkoeffizienten gleich.

Bevorzugt werden lange Zyklen, insbesondere sog. primitive Polynome, bei denen alle Zustände außer »alles null« einen $2^r - 1$ langen Maximalzyklus bilden.

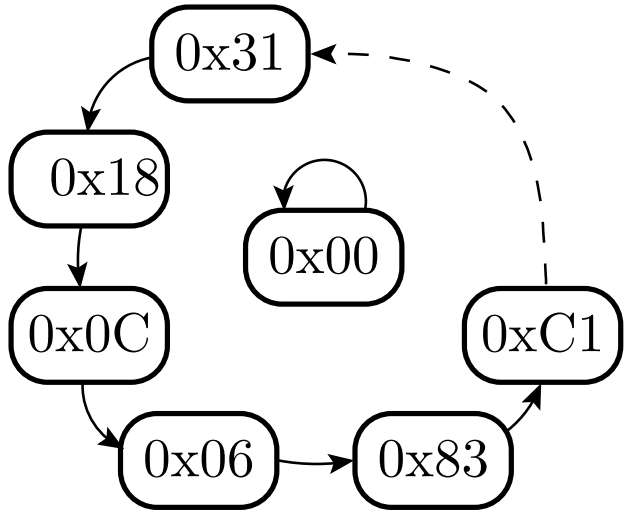
r Bitanzahl des linear rückgekoppelten Schieberegisters.
* Zyklusstruktur ohne Eingaben.

5.60 Zykluslänge und Struktur von LFSR

Die Zyklusstrukur bezieht sich auf Eingabefolge »alles 0« oder für LFSR ohne Eingänge. Beispiel 8-Bit LFSR ohne Eingänge:



0x31	0	0	1	1	0	0	0	1
0x18	0	0	0	1	1	0	0	0
0x0C	0	0	0	0	1	1	0	0
0x06	0	0	0	0	0	1	1	0
0x83	1	0	0	0	0	0	1	1
0xC1	1	1	0	0	0	0	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

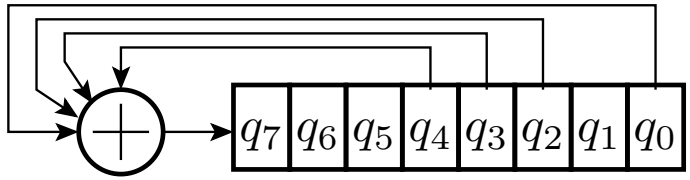


Übergangsfunktion:

$$\begin{aligned} q_7 &= q_4 \oplus q_3 \oplus q_2 \oplus q_0 \\ q_i &= q_{i+1} \text{ für } i \in \{0, 1, 2, \ldots, 6\} \end{aligned}$$

q_i Registerbit i des linear rückgekoppelten Schieberegisters.

Bestimmung der Zykluslänge durch Simulation

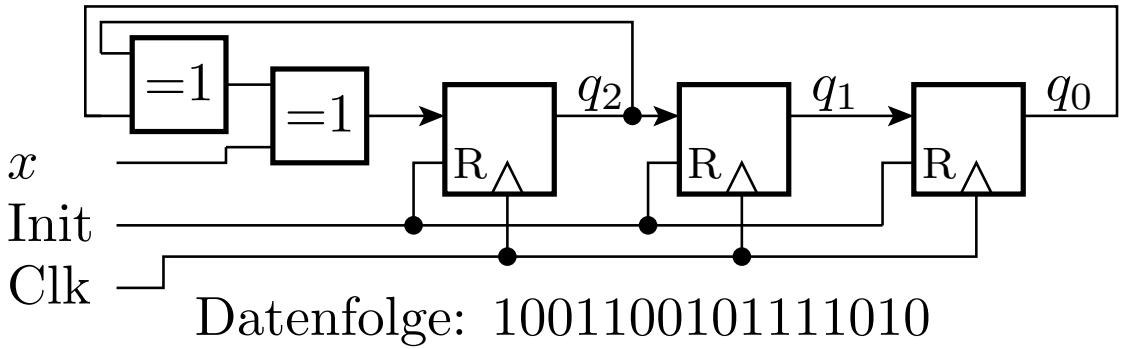



```
...
#define SA 0x31
uint8_t q = SA;          // Startwert setzen
while(1){
  q = (q>>1) ^ (q<<7) ^ ((q<<5)&0x80)
      ^ ((q<<4)&0x80) ^ ((q<<3)&0x80);
  <Ausgabe von s>
  if (q==SA) break;  // bis wieder Anfangswert
  ...                // weiter mit nicht enthal-
}                    // tenem Zustand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

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

Beispiel 5.3: Prüfkennzeichen mit LFSR

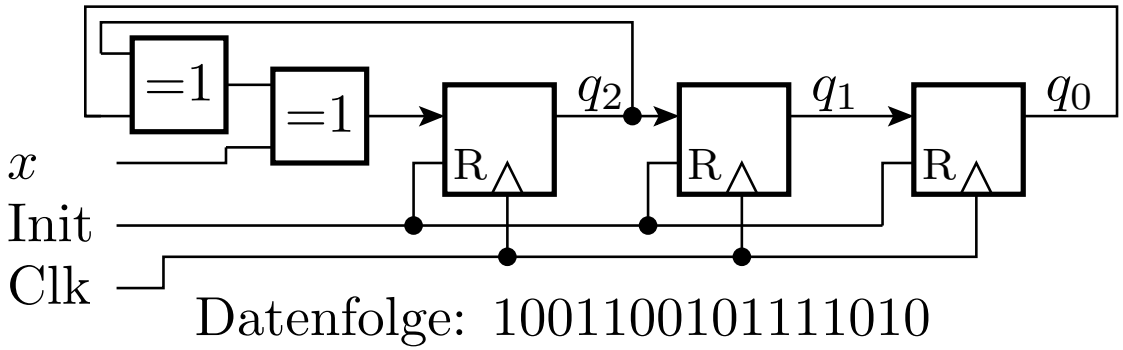
Gegeben ist folgendes linear rückgekoppelte Schieberegister:



a) *Welches Prüfkennzeichen* $\mathbf{q} = q_2 q_1 q_0$ *hat die Datenfolge bei Abbildung beginnend mit dem linken Bit und Startwert 000?*

b) *Wie hoch ist die zu erwartende Fehlfunktionsabdeckung?*

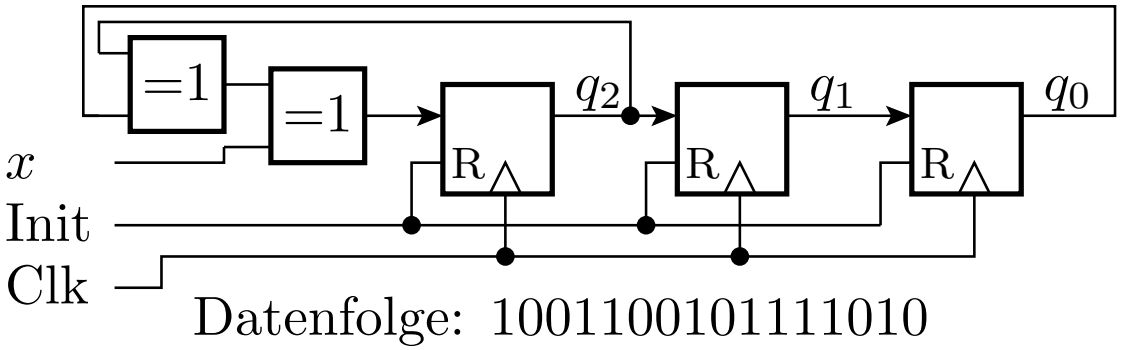
q_i	Registerbit i des linear rückgekoppelten Schieberegisters.
x	Eingang zur Abtastung der Bitfolge am Testobjektausgang.
R	Rücksetzeingang.
Init	Initialisierungssignal.
Clk	Taktsignal (Clock signal).



a) *Welches Prüfkennzeichen* $\mathbf{q} = q_2 q_1 q_0$ *hat die Datenfolge bei Abbildung beginnend mit dem linken Bit und Startwert 000?*

	x	q_2	q_1	q_0
0	1	0	0	0
1	0	1	0	0
2	0	1	1	0
3	1	1	1	1
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	1	1	0	1

	x	q_2	q_1	q_0
8	0	1	1	0
9	1	1	1	1
10	1	1	1	1
11	1	1	1	1
12	1	1	1	1
13	0	1	1	1
14	1	0	1	1
15	0	0	0	1
Prüfkennzeichen: 16		1	0	0

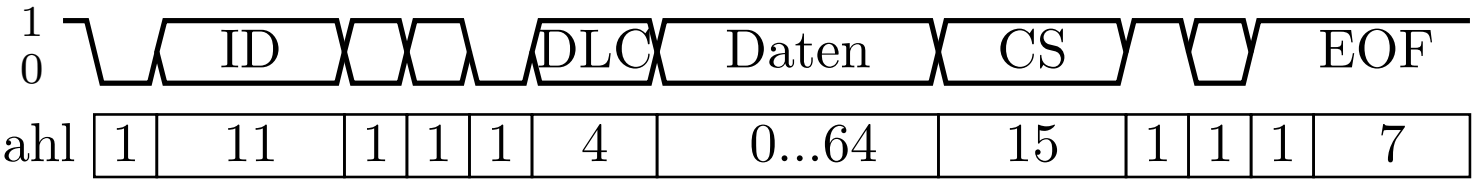


b) *Wie hoch ist die zu erwartende Fehlfunktionsabdeckung?*

$$\mu_{\mathrm{MC}} = 1 - 2^{-3} = 87{,}5\%$$

μ_{MC} Zu erwartende Fehlfunktionsabdeckung.

5.62 CAN-Bus



ID	Nachrichtennummer	CS	15-Bit Prüfkennzeichen
DLC	Datenlängencode	EOF	Ende des Datenrahmens

Eingesetzt z.B. zur Vernetzung von Fahrzeugsteuergeräten.

Erkennbare Formatfehler:

- $r = 15$ Bit Prüfkennzeichen. Erkennungssicherheit:

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

- unzulässige ID, ...

Fehlerfunktionsbehandlung:

- Wiederholung bei Nachrichtenkollision,
- Notfallreaktion bei Ausfall anderer Steuergeräte,
- Fehlerspeicher für aufgetretene Fehlfunktionen, ...

5.63 Ethernet-Protokoll

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

Erkennbare Formatfehler:

- 4-Byte-Prüfkennzeichen:

$$\mu_{\mathrm{FC}} = 1 - 2^{-32} \approx 1 - 2 \cdot 10^{-10}$$

- korrekte Präambel, korrektes Startbyte, ...
- Wiederholanforderung bei fehlerhaft empfangenen oder nicht erhaltenen Datenpaketen, ...

Mittlere Zeit zwischen dem Empfang nicht erkannter verfälschter Datenpakete bei 10^5 Datenpaketen pro s und 10^{-4} MF je Datenpaket:

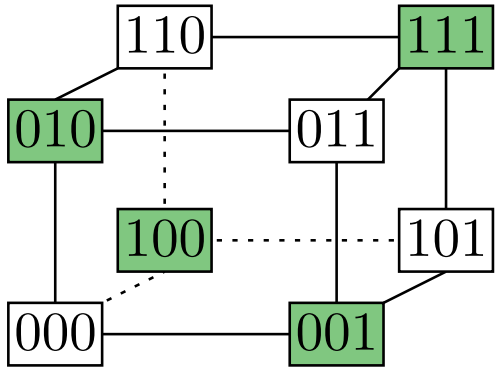
$$\frac{1}{10^5 \ \left[\frac{\mathrm{DP}}{\mathrm{s}}\right] \cdot 10^{-4} \ \left[\frac{\mathrm{MF}}{\mathrm{DP}}\right] \cdot 2^{-32}} > 13{,}5\,\mathrm{Jahre}$$

$\left[\frac{\mathrm{DP}}{\mathrm{s}}\right]$ Anzahl der Datenpakete pro Sekunde.

$\left[\frac{\mathrm{MF}}{\mathrm{DP}}\right]$ Zählwertverhältnis in Fehlfunktion je Datenpaket.

Hamming-Codes

5.64 Hamming-Codes



Hammingcodes werden durch die Hamming-distanz $\#HD$ beschrieben. Das ist die Anzahl der Bitpositionen, in denen sich zwei zulässige Codeworte mindestens unterscheiden.

Hammingdistanzen von 2 oder mehr garantiert, dass eine 1-Bit Verfälschung nicht zu einem anderen gültigen Codewort führt. Erforderliche Hamming-Distanz zum Erkennen von Verfälschungen mit bis zu $\#DB$ verfälschten Bits:

$$\#HD \geq \#DB + 1 \tag{5.8}$$

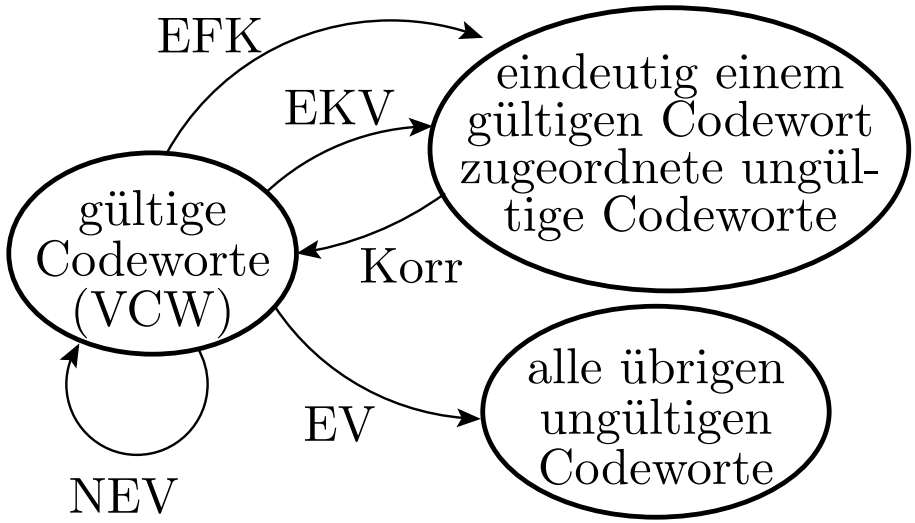
Erforderliche Hamming-Distanz zur Korrektur von Verfälschungen mit bis zu $\#CB$ verfälschten Bits:

$$\#HD \geq 2 \cdot \#CB + 1 \tag{5.9}$$

$\#HD$ Hamming-Distanz.

5.65 Fehlerkorrektur mit Hamming-Code

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



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

$$2^{\#\mathrm{Bit}} \geq \#VCW + \#VCW \cdot \#CVC \tag{5.10}$$

$\#Bit$	Bitanzahl des Codeworts.
$\#VCW$	Anzahl der gültigen Codeworte.
$\#CVC$	Anzahl korrigierbare Codeworte je gültiges Codewort.

5.66 Beispiel: Korrektur von Einzelbitfehler

Anzahl korrigierbare Codeworte je gültiges Codewort gleich Bitanzahl:

$$\#CVC = \#\text{Bit}$$

Mindestbitanzahl:

$$2^{\#\text{Bit}} \geq \#VCW + \#\text{Bit} \cdot \#VCW = (\#\text{Bit} + 1) \cdot \#VCW$$

Für $\#\text{VCW} = 256$ gültige Codeworte:

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

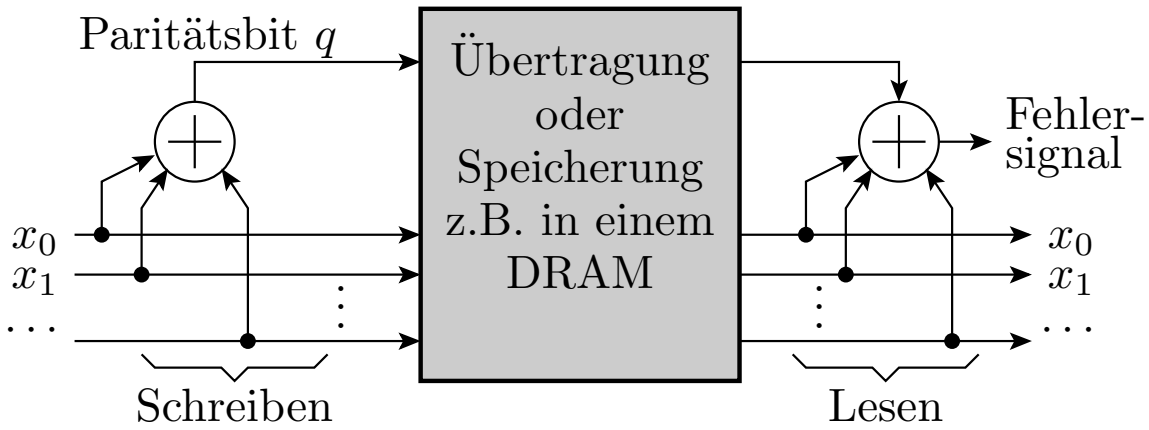
Probe:

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

$\#Bit$	Bitanzahl des Codeworts.
$\#VCW$	Anzahl der gültigen Codeworte.
$\#CVC$	Anzahl korrigierbare Codeworte je gültiges Codewort.

Paritätstest

5.67 Paritätsbit als Prüfkennzeichen ($\#HD = 2$)



Paritätsbit q gleich modulo-2 Summe (EXOR) der Datenbits:

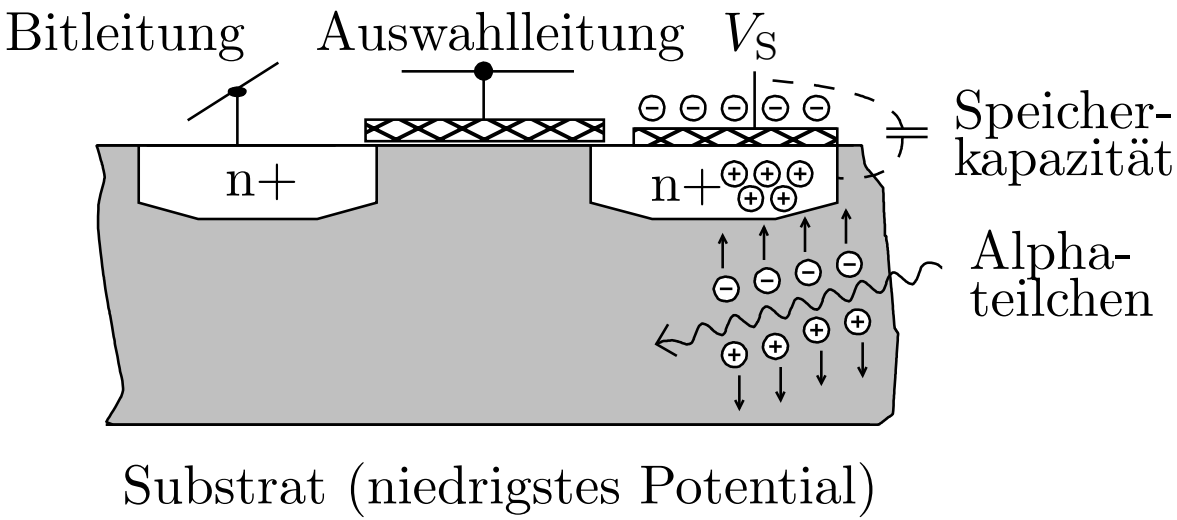
$$q = x_{n-1} \oplus x_{n-2} \oplus \ldots \oplus x_1 \oplus x_0$$

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

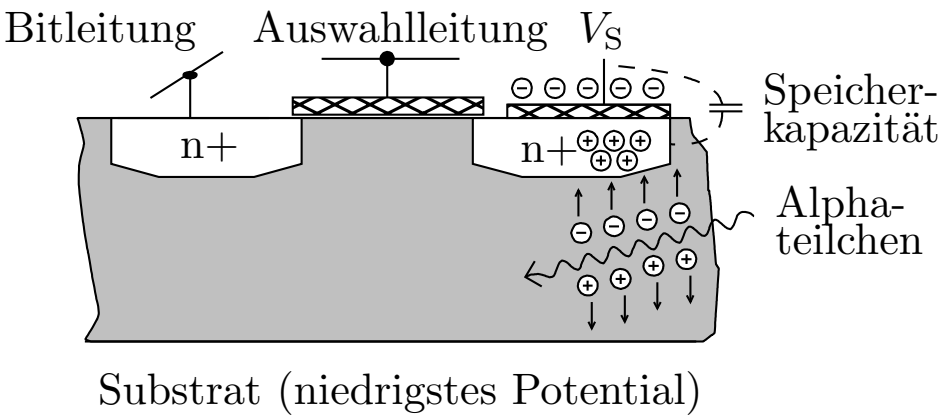
Erkannt werden alle ungeradzahligen Bitverfälschungen. Wenn Fehlfunktionen weder geradzahlige noch ungeradzahlige Verfälschunge bevorzugen, beträgt die zu erwartende Fehlfunktionsabdeckung nach (Gl. 1.35) $\mu_{\mathrm{MC}} = 50\%$. In seinen Anwendungsbereichen ist die Fehlfunktionsabdeckung viel größer.

μ_{MC}, r Zu erwartende Fehlfunktionsabdeckung, Anzahl der redundanten Bits.

5.68 Paritätstest für DRAMs und Speicherriegel



- Informationsspeicherung in winzigen Kapazitäten.
- Häufigste Ursache für Datenverfälschungen: Alphastrahlung.
- Deren Quellen radioaktiver Zerfall von Uran und Thorium, enthalten als Spurenelemente im Gehäuse und im Aluminium der Leiterbahnen oder Kernprozesse im Silizium durch Höhenstrahlung.
- Mittlerer Ereignisabstand: viele Stunden oder Tage.



- Energie eines Alphateilchen: $5\,\mathrm{MeV}$. Energieverlust bei der Generierung eines Elektronen-Loch-Paares $\approx 3{,}6\,\mathrm{eV} \Rightarrow$ Generierung von $\approx 10^6$ Ladungsträgerpaaren. Reichweite $\approx 89\,\mu\mathrm{m}$. gespeicherte Ladung $\approx 10^5$ Ladungsträger. Datenverlust einer oder mehrerer benachbarter Zellen möglich.
- Gleichzeitige Verfälschung durch zwei Alphateilchen unwahrscheinlich.
- Geometrische Trennung der Zellen eines Datenworts (getrennte Schaltkreise oder Speichermatrizen) $\Rightarrow$ Je Zerfall Einzelbitverfälschung je gelesenes Datenwort.

Auch bei der Übertragung sind Verfälschungen selten. Datenobjekte so verschränken, das auch Fehler-Bursts auf Einzelbitfehler je Datenobjekt abgebildet werden.

5.70 Nachweis seltenere Bitverfälschungen

Für seltene unabhängige Bitverfälschungen ist die Anzahl X der verfälschten Bits je Datenobjekt poisson-verteilt:

(4.40)
$$\mathbb{P}\left[X = k\right] = \mathrm{e}^{-\lambda} \cdot \frac{\lambda^k}{k!}$$

Die zu erwartende Fehlfunktionsüberdeckung ist mindests so groß, wie die bedingte Wahrscheinlichkeit, dass genau ein Bit verfälscht ist, wenn nicht null Bits verfälscht sind:

$$\mu_{\mathrm{MC}} \geq \frac{\mathbb{P}[X=1]}{1-\mathbb{P}[X=0]} = \frac{\mathrm{e}^{-\lambda}\cdot\lambda}{1-\mathrm{e}^{-\lambda}\cdot} \overset{(\mu_{\mathrm{F}} \ll 1)^*}{=} \frac{(1-\lambda)\cdot\lambda}{1-(1-\lambda)} = 1-\lambda$$

Zu erwartende Anzahl der Bitfehler je Datenwort:

$$\lambda = \zeta_{\mathrm{Bit}} \cdot w$$

$$\mu_{\mathrm{MC}} \geq 1 - \zeta_{\mathrm{Bit}} \cdot w \tag{5.11}$$

Bei geringer Bitfehlerrate ζ_{Bit} und geringer Bitanzahl $w \gg 50\%$.

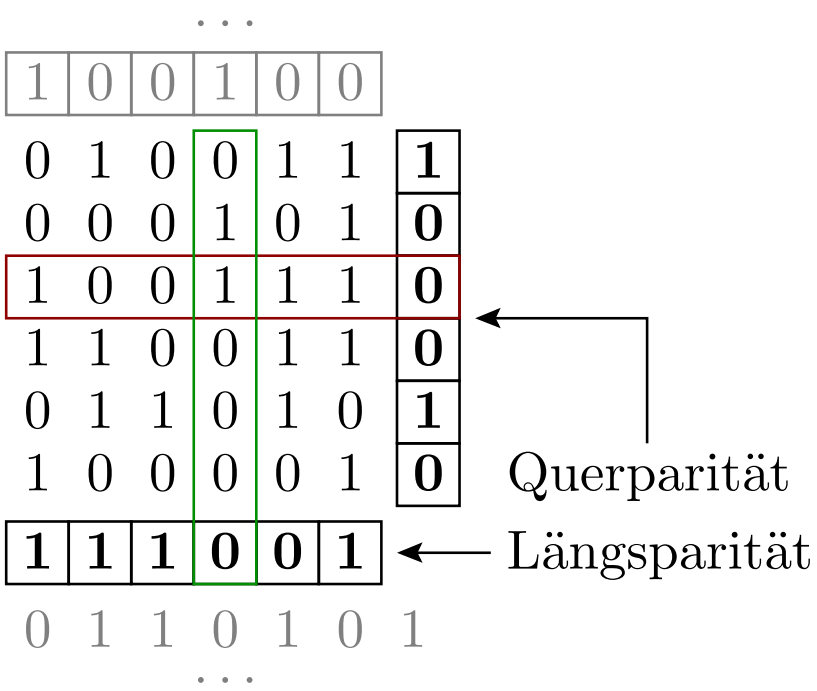
λ	Zu erwartende Anzahl der Bitfehler je Datenwort.
μ_{MC}	Zu erwartende Fehlfunktionsabdeckung.
ζ_{Bit}, w	Bitfehlerrate, Bitanzahl Datenwort ohne Paritätsbit.

*

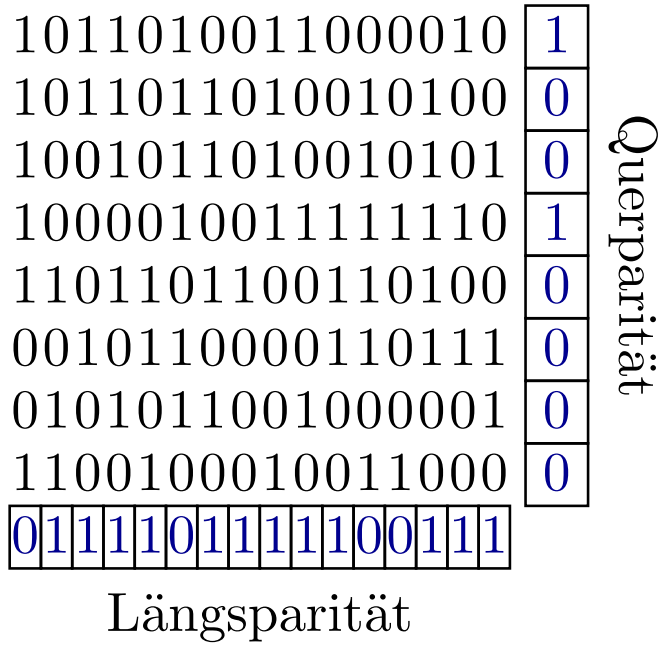
Einzelbitkorrektur

5.71 Kreuzparität

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



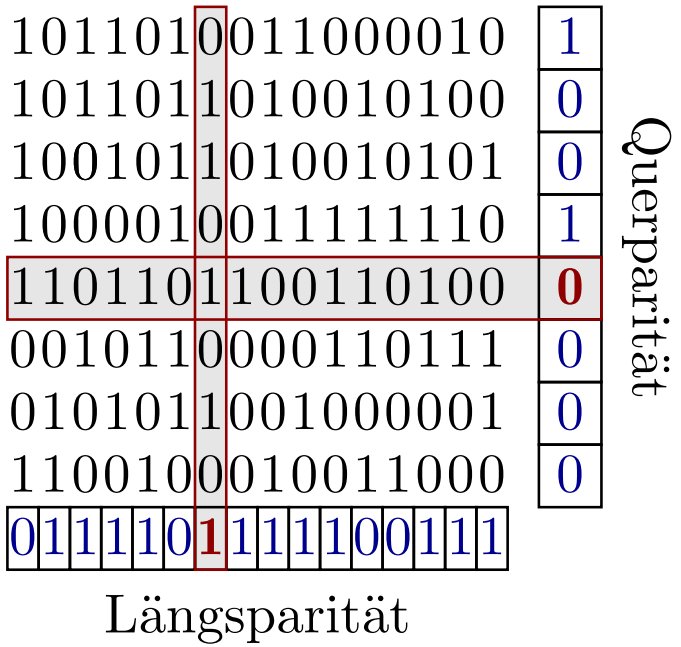
Beispiel 5.4: Kreuzparität



a) *Kontrolle der Zeilen- und Spaltenparität?*

b) *Liegt keine, eine erkennbare nicht korrigierbare oder eine erkenn- und korrigierbare Verfälschung vor?*

a) *Kontrolle der Zeilen- und Spaltenparität?*

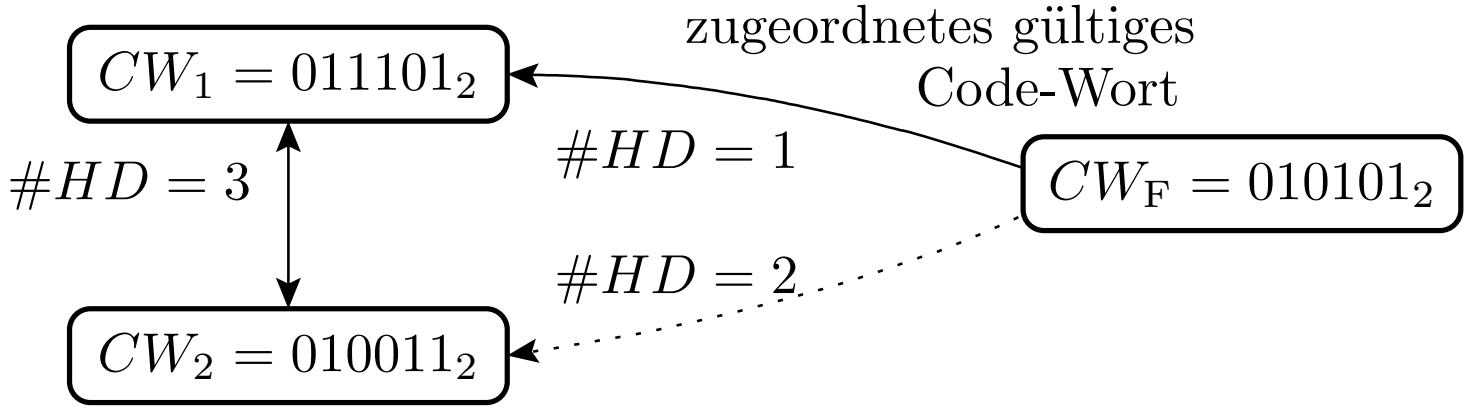


b) *Liegt keine, eine erkennbare nicht korrigierbare oder eine erkenn- und korrigierbare Verfälschung vor?*

Korrigierbar durch Invertierung des Bits in Zeile 5, Spalte 7 (null setzen).

5.73 1-Bit fehlerkorrigierende Hamming-Codes

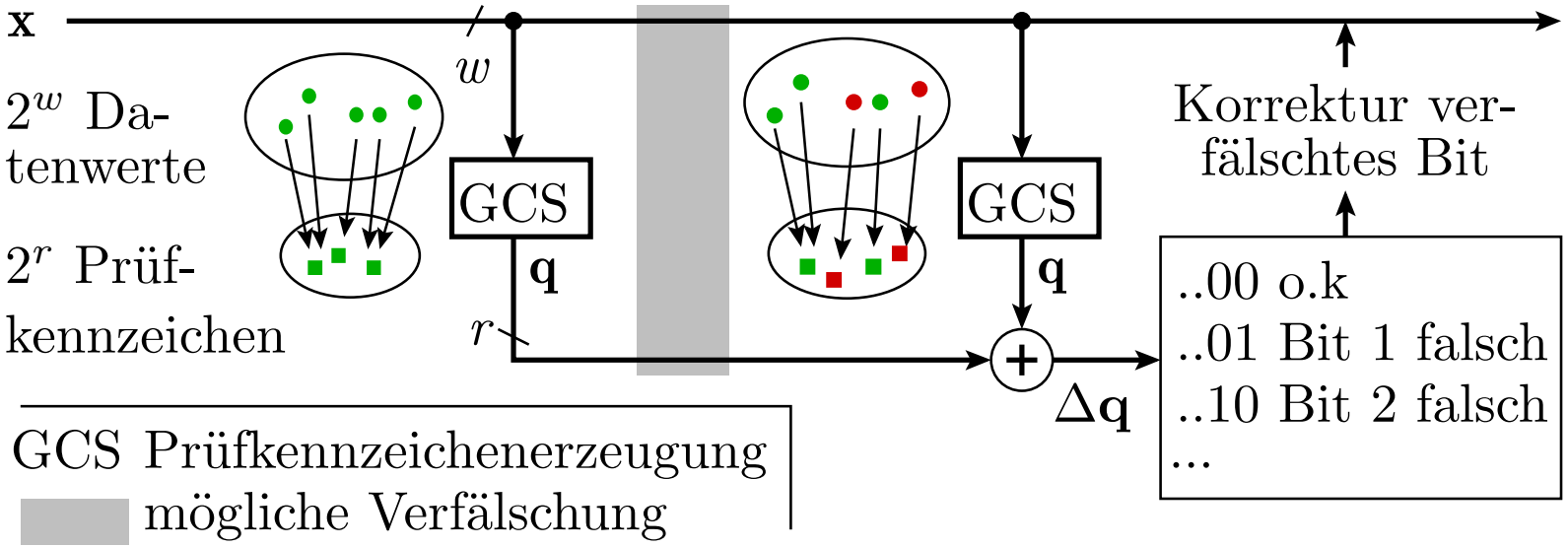
Ab einem Hamming-Abstand $\#HD \geq 3$ ist jede 1-Bit-Verfälschung eindeutig einem gültigen Codewort zugeordnet.



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

$\#HD$	Hamming-Distanz.
CW_i	Code-Wort i.
CW_F	verfälschtes Code-Wort.

5.74 Beispiel für die Codekonstruktion



Erzeugung des r-Bit Prüfkennzeichens **q** so aus dem w-Bit Datenwort **x**,: dass die mod-2 Summen des übertragenen und des danach gebildeten Prüfkennzeichens die Nummer des verfälschten Bits ergibt:

verfälschtes Bit	1	2	3	4	5	...
Prüfkennzeichendifferenz $\Delta\mathbf{q}$	..001	..010	..011	..100	..101	...

q r-Bit Prüfkennzeichen.
$\Delta\mathbf{q}$ EXOR-Differenz der erhaltenen und der neu berechneten Prüfbits.

5.75 Konstruktion der Prüfkennzeichen

Anzahl der Prüfbits r nach Gl. 5.10 für $w = 8$:

$$2^{\mathrm{w+r}} \geq \left(1 + \underbrace{w + r}_{\text{Anz. Einzelbitfehler}}\right) \cdot \underbrace{2^w}_{\#VCW} = (1 + 12) \cdot 2^8;\ r = 4$$

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

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

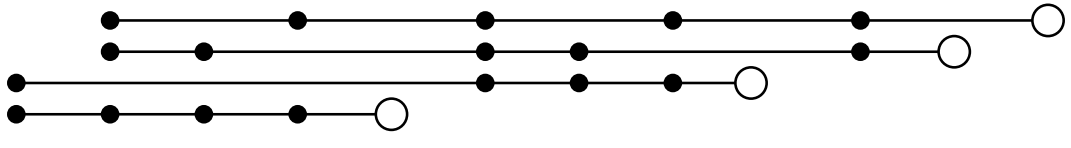
Die unterstrichenen Bits jeder Prüfsumme seinen die Prüfbits q_i, die anderen Datenbits. Datenbitnummern sind noch zuzuordnen.

w, r	Anzahl der Datenbits, Anzahl der Prüfbits.
b_i	Bit i des Gesamtcodeworts.
Δq_i	Prüfkennzeichendifferenz Bit i.

5.76 Zuordnung der Bitnummern

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

○ Prüfbit
• Datenbit



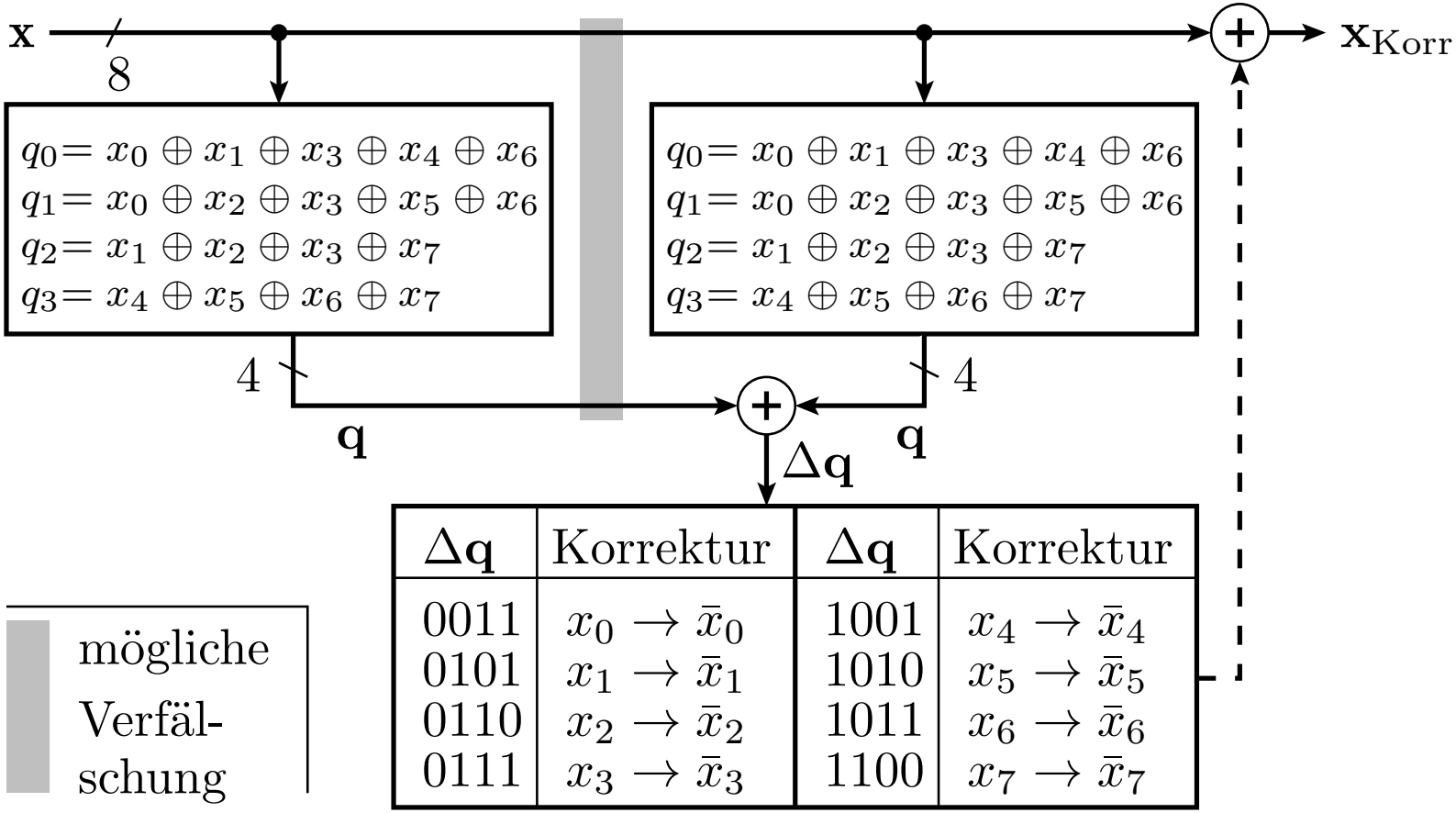
$$
\begin{aligned}
\Delta q_0 &= b_1 \oplus b_3 \oplus b_5 \oplus b_7 \oplus b_9 \oplus b_{11} &&= q_0 \oplus x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6 \\
\Delta q_1 &= b_2 \oplus b_3 \oplus b_6 \oplus b_7 \oplus b_{10} \oplus b_{11} &&= q_1 \oplus x_0 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 \\
\Delta q_2 &= b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_{12} &&= q_2 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_7 \\
\Delta q_3 &= b_8 \oplus b_9 \oplus b_{10} \oplus b_{11} \oplus b_{12} &&= q_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7
\end{aligned}
$$

Auflösung nach q_i für $\Delta\mathbf{q} = 0$:

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

b_i, x_i, q_i Gesamtbit i, Datenbit i, Prüfbit i.

Codier-, Erkennungs- und Korrekturschaltung



Δq	Korrektur	Δq	Korrektur
0011	$x_0 \to \bar{x}_0$	1001	$x_4 \to \bar{x}_4$
0101	$x_1 \to \bar{x}_1$	1010	$x_5 \to \bar{x}_5$
0110	$x_2 \to \bar{x}_2$	1011	$x_6 \to \bar{x}_6$
0111	$x_3 \to \bar{x}_3$	1100	$x_7 \to \bar{x}_7$

- Gleiches Prüfbit-Bildung vor und nach Speichern/ Übertragung.
- Differenzbildung durch bitweises EXOR.
- Wenn $\Delta\mathbf{q} \neq 0$ und Datenbit verfälscht, Bit »$\Delta\mathbf{q}$« invertieren.

Beispiel 5.5: Fehlerkorrigierender Code

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

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

a) *Bestimmung des Codeworts* $\mathbf{b}$ *für den Datenwert* $\mathbf{x} = 0x8B$.

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	x_7	x_6	x_5	x_4	q_3	x_3	x_2	x_1	q_2	x_0	q_1	q_0	
Kontrollbits													
$\mathbf{x} = 0x8B$													$\mathbf{b} =$
$\mathbf{b} = 0xA9B$													$\Delta\mathbf{q} =$
korrigiert													$\mathbf{x} =$

b) *Extrahieren des Werts* $\mathbf{x}$ *aus dem Codewort* $\mathbf{b} = 0xA9B$*?*

a) *Bestimmung des Codeworts* $\mathbf{b}$ *für den Datenwert* $\mathbf{x} = 0x8B$.

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	x_7	x_6	x_5	x_4	q_3	x_3	x_2	x_1	q_2	x_0	q_1	q_0	
Kontrollbits	=	=	=	=	=	=	=	=	=	=	=	=	
$\mathbf{x} = 0x8B$	1	0	0	0	1	1	0	1	1	1	0	1	$\mathbf{b} = 0x8DD$

Der Wert 0x8B hat das Codewort 0x8DD.

b) *Extrahieren des Werts* $\mathbf{x}$ *aus dem Codewort* $\mathbf{b} = 0xA9B$*?*

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	x_7	x_6	x_5	x_4	q_3	x_3	x_2	x_1	q_2	x_0	q_1	q_0	
Kontrollbits	=	= =	=	=		= =	=	= =	=	=	=	=	
$\mathbf{b} = 0$xA9B	1	0	1	0	1	0	0	1	1	0	1	1	$\Delta\mathbf{q} = 12_{10}$
korrigiert	0	0	1	0		0	0	1		0			$\mathbf{x} = 0$x22

Im Codewort 0x49B steckt der Wert 0xA2 mit $\Delta\mathbf{q} = 0\text{b}1100 = 12$. Das verfälschte Bit 12 im Codewort ist Datenbit x_7. Invertierung von x_7 ergibt den Datenwert 0x22.

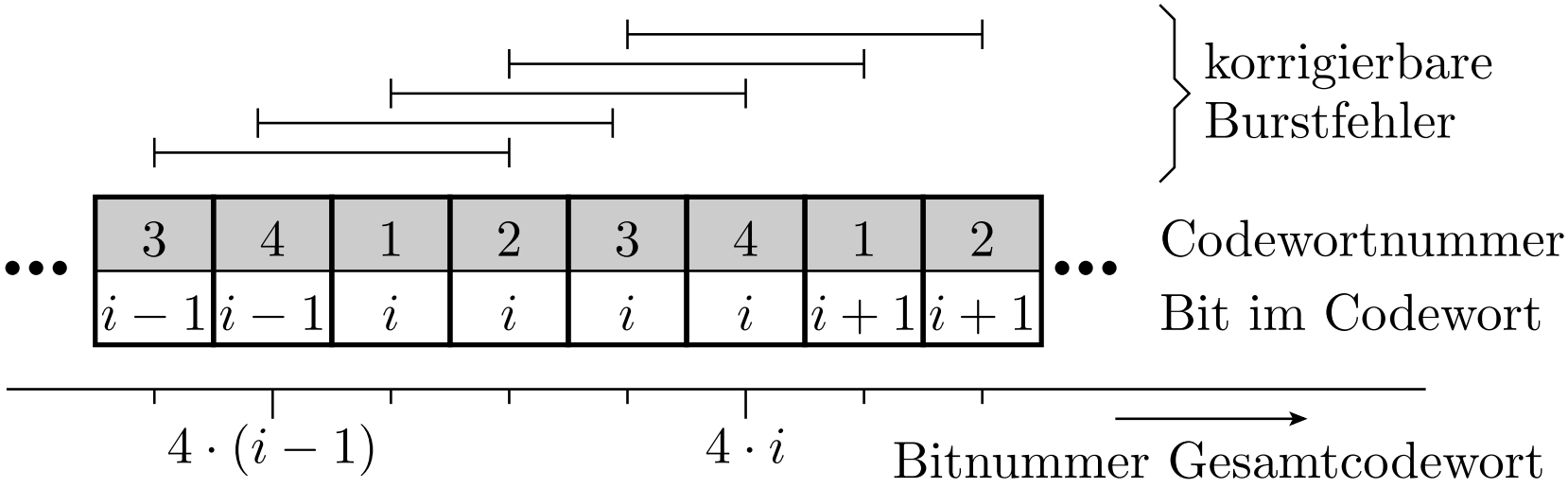
Burstfehler

5.78 Burstfehler, Verschränkung

Burstfehler

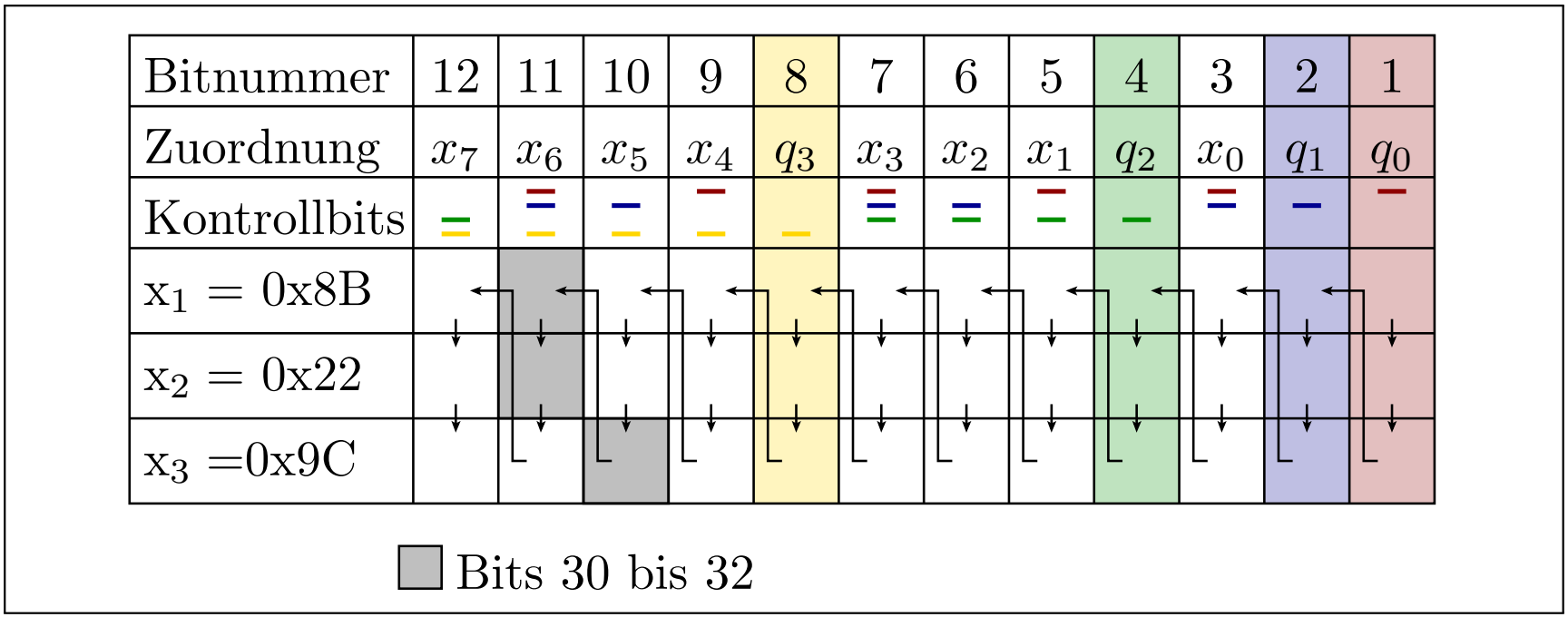
Verfälschung von bis zu m aufeinanderfolgende Bits. Typisch für Lesefehler von CDs und Übertragungsfehler.

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



Beispiel 5.6: Burstfehler

a) *Codierung Datenfolge 0x8B, 0x22, 0x9C so, dass bis zu 3-Bit lange Burstfehler korrigierbar sind, durch Verschränkung von je drei aufeinanderfolgende Codeworten für Einzelbitfehlerkorrektur?*



b) *Zeigen Sie, dass eine Invertierung der Bits 30 bis 32 korrigiert wird?*

a) *Codierung Datenfolge 0x8B, 0x22, 0x9C so, dass bis zu 3-Bit lange Burstfehler korrigierbar sind, durch Verschränkung von je drei aufeinanderfolgende Codeworten für Einzelbitfehlerkorrektur?*

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

b) *Zeigen Sie, dass eine Invertierung der Bits 30 bis 32 korrigiert wird?*

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	x_7	x_6	x_5	x_4	q_3	x_3	x_2	x_1	q_2	x_0	q_1	q_0	
Kontrollbits	=	=	=	=	=	=	=	=	=	=	=	=	
$x_1 = 0x8B$	1	0	0	0	1	1	0	1	1	1	0	1	$\Delta\mathbf{q} = 1011_2$
$x_2 = 0x22$	0	0	1	0	1	0	0	1	1	0	1	1	$\Delta\mathbf{q} = 1011_2$
$x_3 = 0x9C$	1	0	0	1	0	1	1	0	1	0	0	0	$\Delta\mathbf{q} = 1010_2$

Durch Burstfehler invertierte Bits 30 bis 32,

RAID und Backup

5.80 Redundante Datenspeicherung

Raid (Redundant Array of Inexpensive Disks): Organisation mehrerer physischer Massenspeicher (Festplatten oder SSD) zu einem logischen Laufwerk. Mit RAID können Systeme Ausfälle einer oder mehrerer Festplatten oder SSD ohne Datenverlust und in vielen Fällen sogar ohne Ausfallzeiten überstehen.

Backup: Speicherung von Daten auf einem externen Massenspeicher zur Wiederherstellung bei Datenverlust.

RAID	Redundantes Array unabhängiger Festplatten.
SSD	Solid State Drive, Festplattennachbildung mit Halbleiterspeichern.

5.81 RAID mit Paritätsblöcken (RAID 2 bis 7)

	Paritätsbildung	Ausfall Disc 2	Ausfall Disc 4
Disc 1: Daten	0001 0011 0010	0001 0011 0010	0001 0011 0010
Disc 2: Daten	1101 0101 0000	xxxx xxxx xxxx	1101 0101 0000
Disc 3: Daten	1101 1111 1100	1101 1111 1100	1101 1111 1100
Disc 4: Parität	**0001 1001 1110**	0001 1001 1110	xxxx xxxx xxxx
	Wiederherstellung:	**1101 0101 0000**	**0001 1001 1110**

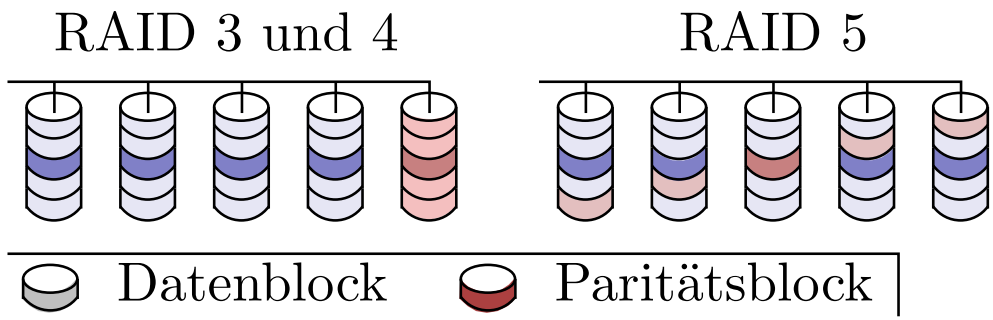
Datenverteilung wie bei RAID 0 blockweise über mehrere Platten.
Bildung und Speicherung von Paritätsblöcken so, dass

- nur Blöcke unterschiedlicher Platten zusammengefasst werden,
- der Paritätsblock auf noch einer anderen Platte gespeichert wird.

Nach Ausfall einer Platte lassen sich alle Blöcke auf der ausgefallenen Platte aus denen auf anderen Platten gespeicherten Blöcken durch bitweise EXOR rekonstruieren, vorausgesetzt es ist bekannt

- welche Platte ausgefallen ist und
- und wo die zugehörigen ver-EXOR-ten Blöcke stehen.

5.82 Verteilung der Paritätsblöcke auf Platten

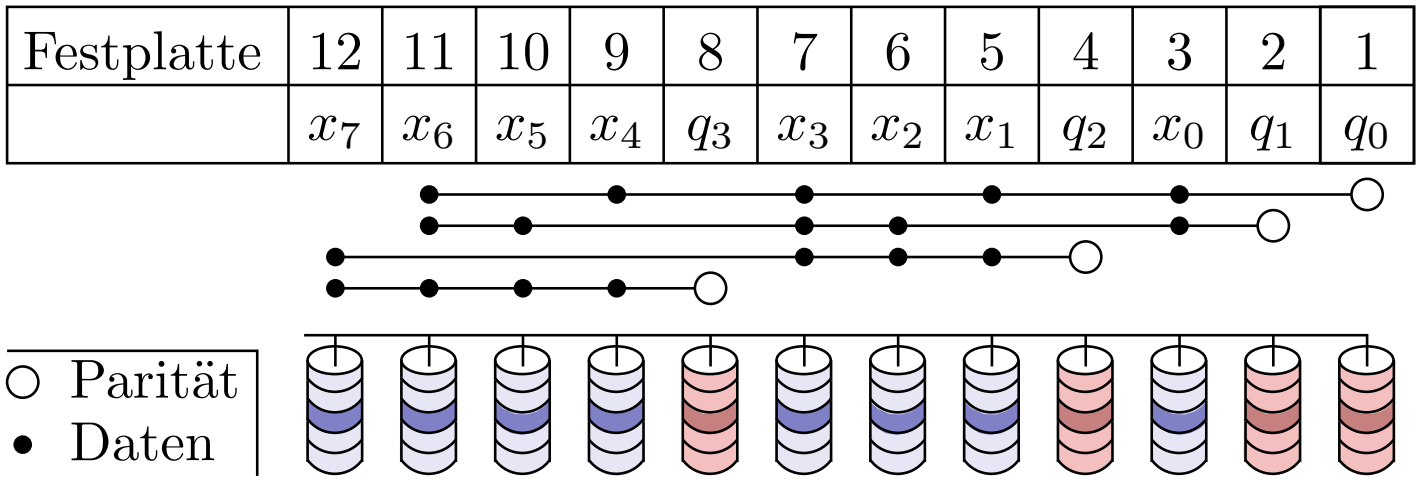


Schreiben eines Blocks:

- Block lesen, Paritätsänderung bestimmen, schreiben,
- Paritätsblock lesen, ändern, schreiben.

Wenn die Paritätsblöcke alle auf derselben Festplatte stehen (RAID 3 und 4), viel mehr Schreib-Lese-Zugriffe auf diese. Verkürzte Lebensdauer. Alternative ist gleichmäßige Verteilung der Paritätsblock auf alle Platten (RAID 5).

5.83 Erweiterte Fehlertoleranz



Statt einem Paritätsblock, mehrere nach dem Prinzip der 1-Bit-Fehlerkorrektur. Toleriert nicht nur komplette Plattenausfälle, sondern auch einzelne Bitverfälschungen (RAID 2).

Weitere Ansätze:

- Zusätzliche Längsparitätsblöcke zur Wiederherstellung einzelner verfälschter Blöcke nicht komplett ausgefallener Platten nach dem Prinzip der Kreuzparität.
- Fehlertoleranz für mehr als einen zeitgleichen Plattenausfall, ...

5.84 Weitere Schutzmaßnahmen

Ersatz der ausgefallenen Platte vor dem nächsten Ausfall:

- Laufwerkwechsel in der Regel im ausgeschalteten Zustand. Rebuilt (Rekonstruktion der verlorenen Daten) bei Systemstart.
- Hot-Fix: Reserveplatte, die bei Ausfall für die ausgefallene Platte in das RAID eingebunden wird.
- Hot-Swap: Plattenaustausch und Rebuilt im Betrieb.

Neben HW-Ausfällen gibt es weitere Ursachen für den Verlust schwer wiederzubeschaffende Daten. Auftrittshäufigkeiten:

- 59% Hardwareprobleme
- 26% Anwenderfehler
- 9% Softwarefehler
- 2% Schadware (Vieren, ...)
- 2% Naturkatstrophen
- 2% sonstiges.

Selbst ausgefeilte RAIDs tolerieren nur HW-Probleme. Den einzig wirklich zuverlässigen Schutz gegen Datenverluste bieteten konsequent geplante und durchgeführte Backups.

Zusammenfassung

5.85 Fehlererkennende Codes, Prüfkennzeichen

Formatkontrollen mit redundanten Bits funktionieren perfekt, wenn garantiert werden kann, dass Verfälschungen durch Fehlfunktionen gleichmäßig auf zulässige und erkennbar unzulässige Werte abgebildet und alle unzulässigen Werte erkannt werden:

(1.33) $$\mu_{\mathrm{MC}} = 1 - \frac{\#VP}{\#PP}$$

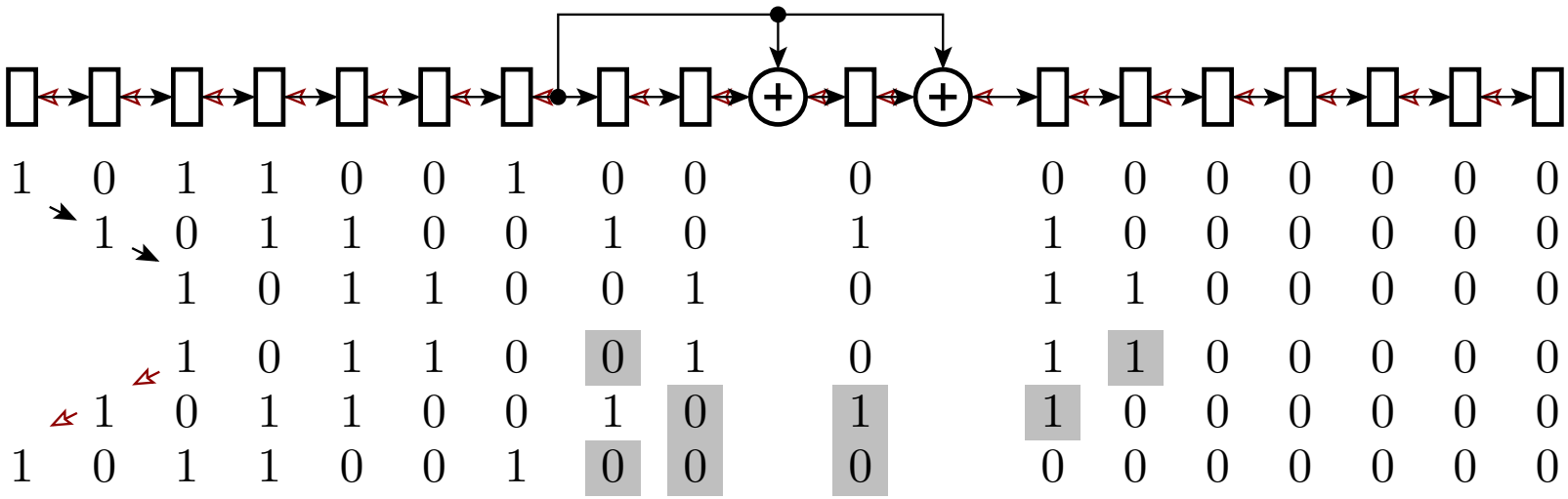
(1.35) $$\mu_{\mathrm{MC}} \geq 1 - 2^{-r}$$

(1.34) $$\zeta_{\mathrm{PM}} = 0$$

Erfordert in der Regel geeignete Codierung oder Prüfkennzeichen:

- Fehlererkennede Codes: Pseudozufällige Codierung unter Erhöhung der Bitanzahl. Dekodierung und Kontrolle mit Umkehralgorithmus der Codierung.
- Prüfkennzeichen: Pseudozufällige Bildung eines r-Bit Kennzeichens aus dem viel größeren Datenobjekt. Anhängen an das Datenobjekt. Kontrolle durch wiederholte Kennzeichbildung und Vergleich mit dem angehängten Kennzeichen.

5.86 Schieberegister als Coder und Decoder



Für umgekehrbare pseudo-zufällige Abbildung mit Bitanzahlerhöhung gibt viele Möglichkeiten, z.B. die Multiplikation mit einer großen Konstanten. Aufwandsgünstiger und genauso wirkungsvoll ist die Codierung und Decodierung mit Schieberegistern. Auch mit den Maschinenbefehlern von Rechnern programmierbar.

Zur Prüfkennzeichenbildung eignet sich der Rückwärtsbetrieb im Bild, d.h. das linear rückgekoppeltes Schieberegister. Es gibt viel mehr Lösungen, die genauso gut funktionieren.

5.87 Hamming-Codes

Gültigen Codeworte eines Hammingcodes unterscheiden unterscheiden sich um den Hammingabstand $\#HD > 1$, d.h. in mindestens $\#HD$ Bits. Dazu werden uncodierte Codeworte um lineare Summen ausgewählter Bits ergänzt.

Der Nachweis von bis zu $\#DB$ verfälschten Bits bzw. für die Korrektur von bis zu $\#CB$ Bits verlangt Hamming-Distanzen von:

(5.8) $$\#HD \geq \#DB + 1$$

(5.9) $$\#HD \geq 2 \cdot \#CB + 1$$

Genutzt werden hauptsächlich

- Paritätstest für den Nachweis von Einzelbitverfälschungen und
- fehlerkorrigierende Codes für Einzelbitverfälschungen.

Beides setzt geringe Bitverfälschungsraten und Unabhängigkeit der Verfälschungen voraus, sicherzustellen durch geringe Fehlfunktionsraten und geeignete Zusammenfassung physikalischer Bits zu Code-Worten.

5.88 Paritätstest

Ergänzung aller Datenworte um eine Paritätsbit gleich EXOR aller Datenbits. Hamming-Distanz $\#HD = 2$. Kontrolle durch EXOR aller Datenbits und Vergleich mit dem Paritätsbit.

Erkannte werden Einzelbit- und auch alle anderen ungeradzahligen Verfälschungen. Für unabhängige Bitverfälschungen ist die Fehlfunktionsüberdeckung mindestens:

(5.11)
$$\mu_{\mathrm{MC}} \geq 1 - \zeta_{\mathrm{Bit}} \cdot w$$

Einsatzbeispiel:

- Erkennen umgekippte Bits in DRAMs durch Radioaktivität. Verlangt das die Codeworte aus Bits räumlich getrennter Speicherblöcke zusammengesetzt werden.
- Übertragung kleiner Datenobjekte mit geringer Bitfehlerrate.

Kreuzparität: Mit einer Zeilen- und Spaltenparität für zweidimensionale Bitfelder lässt sich sogar eine einzelne Bitverfälschung lokalisieren und korrigieren. Anschaulichster fehlerkorrigierender Code.

5.89 Fehlerkorrigierende Hamming-Codes

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	x_7	x_6	x_5	x_4	q_3	x_3	x_2	x_1	q_2	x_0	q_1	q_0	
Kontrollbits	=	=	=	=	=	=	=	=	=	=	=	=	
$\mathbf{b} = 0\text{xA9B}$	1	0	1	0	1	0	0	1	1	0	1	1	$\Delta\mathbf{q} = 12_{10}$
korrigiert	0	0	1	0		0	0	1		0			$\mathbf{x} = 0\text{x}22$

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

(5.10) $$2^{\#\text{Bit}} \geq \#VCW + \#VCW \cdot \#CVC$$

Beispielcode:

- 8 Datenbits, $\#VCW = 256$, 4 Kontrollbits, $\#\text{Bit} = \#CVW = 12$

$$12 + 256 \cdot 12 = 3328 > 2^{12} = 4096\surd$$

- Zahl der Kontrobitabweichung gleich Nummer verfälschtes Bit.

5.90 Burstfehler, RAID, Backup

Wenn Ursachen zu erwarten sind, die mehre Bits verfälschen:

- Kratzer auf einer CD,
- Ausfall einzelner Festplatten und Speicherschaltkreise,
- Alpha-Teilchen, die räumlich benachbarte Bits verfälschen, ...

sind Bits mit erhöhtem Risiko einer gleichzeitigen Verfälschung getrennten Codeworten zuzuordnen, Dann genügen die behandelten Codes zur Korrektur von Einzelbitverfälschungen.

RAIDs sind logisches Laufwerk aus mehreren physischen Festplatten oder SSDs, die ab RAID1 Einzelplattenausfälle tolerieren. Das erfordert im einfachsten Fall, dass alle Bits eines Datenworts und das Paritätsbit je Datenwort auf einer anderen Platte stehen und die ausgefallene Platte davon unabhäng lokalisierbar ist.

Selbst ausgefeilte RAID-Technilen tolerieren nur HW-Probleme. Wirklich zuverlässigen Schutz gegen Datenverluste bieteten Backups, die ausreichend gegen Fehlfunktionen des Systems, Bedinfehler, Angriffe, Katastrophen, ... geschützt sind.

Berechnungkontrolle

5.91 Kontrollmöglichkeiten

Fehlererkennende Codes und Prüfkennzeichen erlauben perfekte Kontrollen, aber nur für übertragene und gespeicherte Daten, nicht für berechnete Ergebnisse. Auch für Berechnungen sind Kontrollen möglich:

- Eingaben, Zwischenergebnisse,
- Ablauf, Ausgabe.

Gut kontrollierbare Merkmale von Eingaben, Zwischenergebnissen und Ausgaben sind die zulässigen Wertebereiche.

Manulle Eingaben sind in der Regel in einer formale Sprache beschrieben und mit einem spracherkennenden Automaten kontrollierbar.

Für Ablauf- und Aufrufreihenfolgen gibt es Regeln, kontrollierbar mit Beobachterautomat, und Invarianten, die sich nicht ändern dürfen.

Bei der Übergabe von Datenobjekten empfielt sich eine Typkontrolle.

Das sind alles Kontrollen auf Zulässigkeit für die Einprogrammierung in die Software. Alternativ Problemausschluss durch statische Tests.

5.92 Güte der Kontrollen

Für die Kontrollverfahren auf der Folie zuvor lässt sich in der Regel nur garantieren, dass sie fehlerfrei implementiert alle betrachteten Regelverletzungen für das zu kontrollierrnde Merkmals erkennen.

Verallgemeinerbare Abschätzungen für den Einfluss auf die Fehlfunktionsüberdeckung und die Phantomfehlfunktionsrate sind schwierig. Die Leistungsfähigkeit liegt in der Vielzahl der Kontrollmöglichkeiten.

Berechnungskontrollen gehören, sofern ihre Notwendigkeit nicht durch statische Tests ausgeschlossen wird, in die Software. Der Umfang, in dem die Kontrollen tatsächlich einprogrammiert oder durch statische Tests überflüssig gemacht werden, hängt erheblich ab

- von der Programmiersprache (Abschn. 7.1),
- der Softwarearchitektur (Abschn. 7.2.1) und
- der Art, wie im Entwurfsprozess Anforderungen und Zielfunktionen gesammelt werden (Abschn. 7.2.3).

5.93 Kontrollen auf Richtigkeit (Abschn. 1.2.3)

- Mehrfachberechung und Vergleich als Universalverfahren,
- Looptest, aber nur für umkehrbar eindeutige Abbildungen und
- aufgabenspezifische Korrektheitstests.

Für Datenübertragung und -speicherung wegen besserer Alternative uninteressant.

Für Berechnungsergebnisse teilweise interessant:

- Loop-Test und aufgabenspezifische Korrektheitstests, können, wenn anwendbar, perfekte Kontrollen sein.
- Verdopplung und Vergleich ist auf alle deterministischen Berechnungen mit eindeutiger Eingabe-Ausgabe-Zuordnung, anwendbar. Gemeinsam mit den Maßnahmen zur Sicherstellung ausreichender Diversität, Relation zur erzielbaren Fehlfunktionsüberdeckung, sehr aufwändig.

Wertebereich

Wertebereichskontrolle mit Beispiel

Eingaben und Berechnungsergebnisse haben in der Regel einen zulässigen Wertebereich, der viel kleiner ist als die Menge der darstellbaren Werte.

Die Überwachung zusammenhängender Wertebereiche ist, wenn eine Funktion oder ein Systemruf für die Fehlfunktionsbehandlung bereitsteht, einfach (Abschn. 1.2.6):

```
uint32_t a;          // Age of employee, range 18...67
                     // WB(u32):0...0xFFFFFFFF
if (a < 18 || a > 67) <malfunction handling>;
```

Die Abschätzung aus der Informationsredundanz

(1.33)
$$\mu_{\mathrm{MC}} = 1 - \frac{\#VP}{\#PP}$$

$$\mu_{\mathrm{MC}} = 1 - \frac{67-18+1}{2^{32}} \approx 1 - 10^{-8}$$

ist jedoch viel zu optimistisch. Denn Fehlfunktionen verursachen insbesondere bei zusammenhängenden Zahlenbereichen überverhältnismäßig oft zulässige falsche Werte.

5.94 Fehlerüberdeckung Wertebereichskontrolle

- In Programmen werden kleine Werte, allen voran null und eins, viel öfter erzeugt als die einzelnen großen Werte, z.B. der uint32_t-Wert:

$$0\text{x}92\text{A}4.5\text{F}1\text{B} = 2.460.245.787$$

- Eine Verwechslung mit dem Alter eines anderen Angestellten ist immer zulässig.
- Eine Verwechslung mit einer Hausnummer wird meist einen zulässigen Wert ergeben, ...

Die zu erwartende Fehlfunktionsabdeckung ist auf keinen Fall fast 100% nach (Gl. 1.33). Abdeckungen unter 90% sind viel wahrscheinlicher. Verallgemeinerbare Abschätzungen sind schwierig.

5.95 Verbesserte Wertebereichskontrolle

Die offensichtlichtlichen Schwachstellen von Wertebereichskontrollen sind zum Teil vermeidbar:

- Verschiebung der Wertebereiche in selten genutzte Zahlenbereiche durch Addition einer Konstanten,
- Vermeidung zusammenhängender Zahlenbereiche, z.B. durch Multiplikation mit einer Konstanten, ...

Für Aufzählungswerte, die nur zugewiesen und gelesen werden, ist eine Verschiebung in seltener genutzte Zahlenbereiche auch bei manueller Programmierung kein großer Zusatzaufwand.

Eine typgebundenen Wertebereichstransformation zur Verbesserung der Fehlfunktionsabdeckung von Wertebereichskontrollen könnte eine interessante Idee für den Einbau in Compiler sein.

Syntax

5.96 Syntaxtest

Syntaxkontrollen sind nicht nur für Programme vor der Übersetzung, sondern für jede Form der manuellen Texteingabe zweckmäßig. Voraussetzung, Textbeschreibung in einer formalen Sprache.

Formale Sprache: Definition zulässiger Zeichenfolgen durch Syntaxregeln, deren Einhaltung sich durch einen spracherkennenden Automaten kontrollieren lassen.

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

Syntaxtests erkennen viele menschengemachter Fehler. Für maschinell erzeugte Daten, die nicht manuell bearbeitet werden, haben Prüfkennzeichen, fehlererkennende oder korrigierenden Codes ein viel besseres Verhältnis von Aufwand zu Nutzen.

5.97 EBNF zur Beschreibung formaler Sprachen

Beschreibungsmittel der EBNF zur Definition von Sprachregeln:

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

EBNF Erweiterete Backus-Naur-Form.
NTS Nicht-Terminalsymbol, zu ersetzendes Symbol.

5.98 Beispiele für EBNF-Syntaxregeln

```
Zahl = ['-'],((ZiffernAusserNull,{Ziffer})|'0');
ZifferAusserNull = '1'|'2'|'3'| ... |'9';
Ziffer = ZifferAusserNull | '0';
```

Beispielzeichenfolgen:

```
'-1001': zulässig
'3,23' : nur bis Komma zulässig
'010'  : nur vor 1 zulässig
```

```
Bezeichner = Buchstabe,{Buchstabe|Ziffer},TZ;
TZ = ','|';'|Leerzeichen|...
```

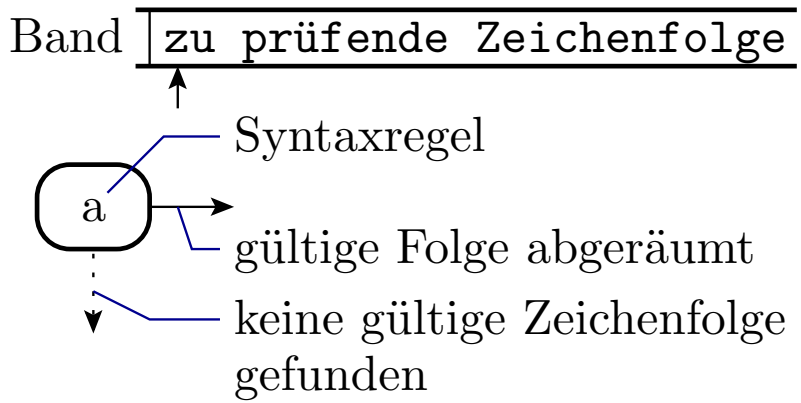
Beispielzeichenfolgen:

```
'a100;': zulässig,
'aa_3,': unzulässig wegen '_'
'1A'   : unzulässig wegen führender Ziffer
```

5.99 Spracherkennende Automaten

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

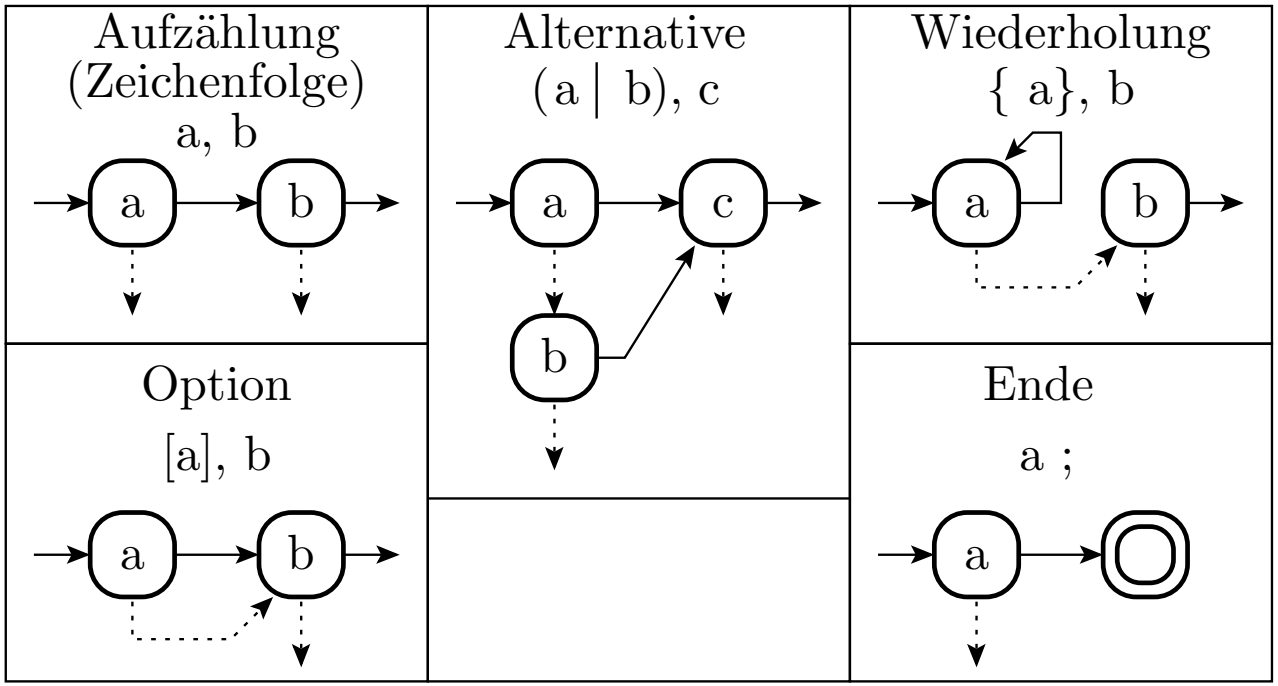
In jedem Automatenzustand wird versucht, eine Zeichenfolge nach der Syntaxregel a des Zustands abzuräumen:



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

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

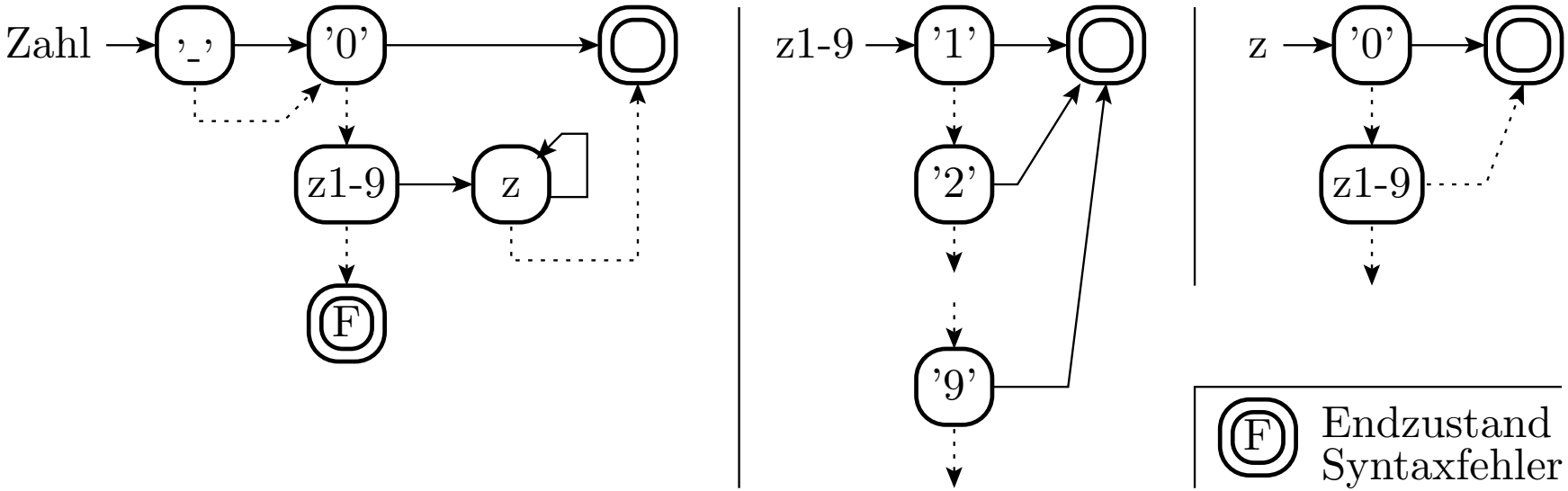
5.100 Von der EBNF zum Automaten



Beispiel:

```
Zahl = ['-'],((z1-9, {z}) | '0');
    z1-9 = '1' | '2' | ... | '9';
    z    = z1-9 | '0';
```

```
Zahl = ['-'],((z1-9, {z}) | '0');
z1-9 = '1' | '2' | ... | '9';
z    = z1-9 | '0';
```



Welche Zustände durchläuft der Automat. Was erkennt er?

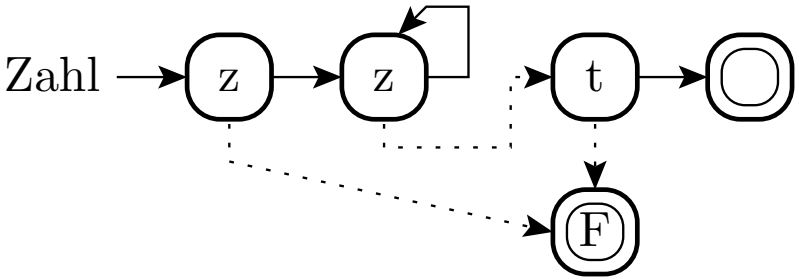
- "125_": '-'↓, '0'↓, z1-9→, z→, z→, z↓, Endzustand: $125_{\sqrt{\downarrow}}$
- "k89": '-'↓, '0'↓, z1-9↓, Endzustand: Syntaxfehler, $\overset{\downarrow}{k}$
- "-0701" : '-'→, '0' →, Endzustand: $0\sqrt{\overset{\downarrow}{7}}$

5.102 Abräumen an den Kanten

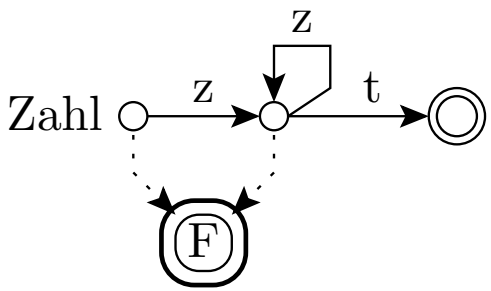
Beschreibung einer Zahl:

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

Abräumen in den Zuständen



Abräumen an den Kanten



- Das Abräumen kann auch an den Kanten erfolgen.
- Dann kann jeder Knoten auch mehr als zwei abgehende Kanten haben.
- Beschreibung derselben Syntaxregeln mit weniger Zuständen.
- Die »Sonst-Kanten« zum Fehlerzustand müssen im Syntaxgraphen nicht mit gezeichnet werden.

Invarianten

5.103 Invarianten

Invarianten: Bedingungen, die unabhängig von den zu verarbeitenden Daten immer erfüllt sein müssen. Überprüfbar

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

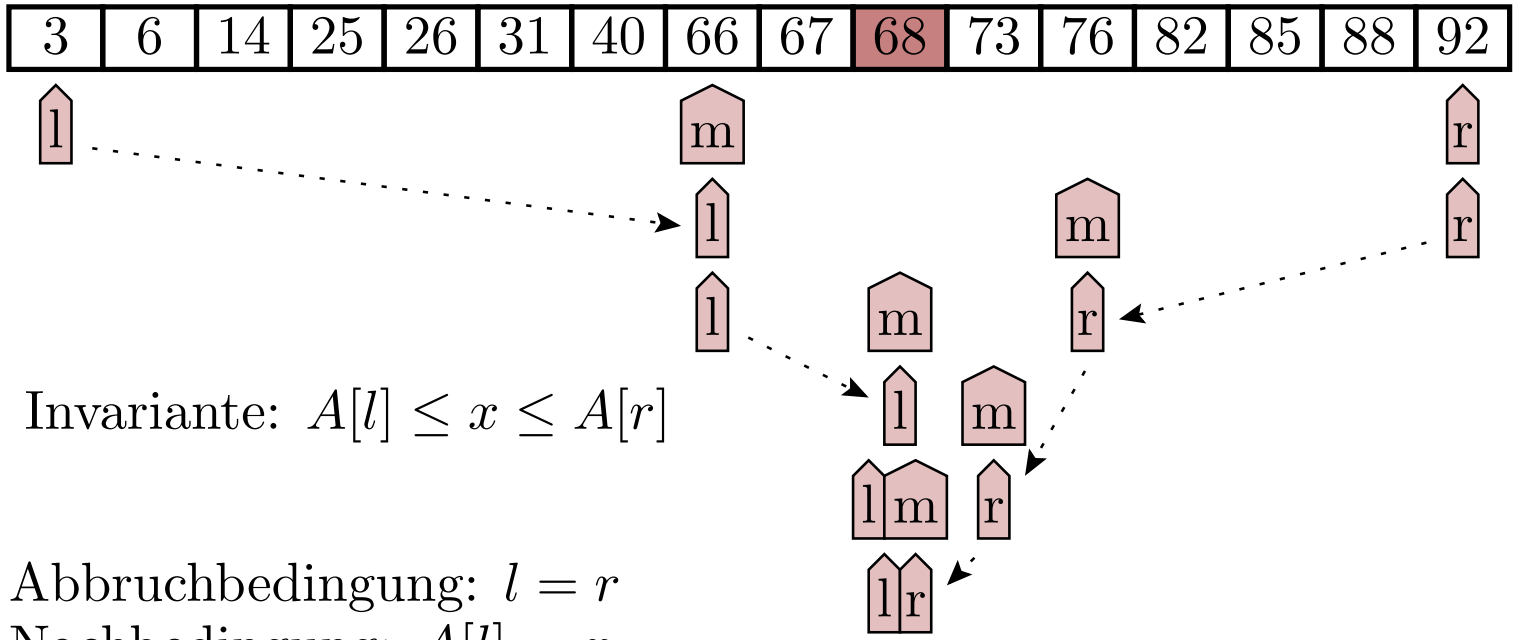
Beispiele für Invarianten:

- Sortieren einer Liste: Anzahl und Summe der Elemente.
- Zeiger: null oder Adresse eines Objekts mit zulässigem Typ.
- Schleifeninvariante.
- ...

5.104 Beispiel mit einer Schleifeninvarianten

gesucht: Position Schlüssel x

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



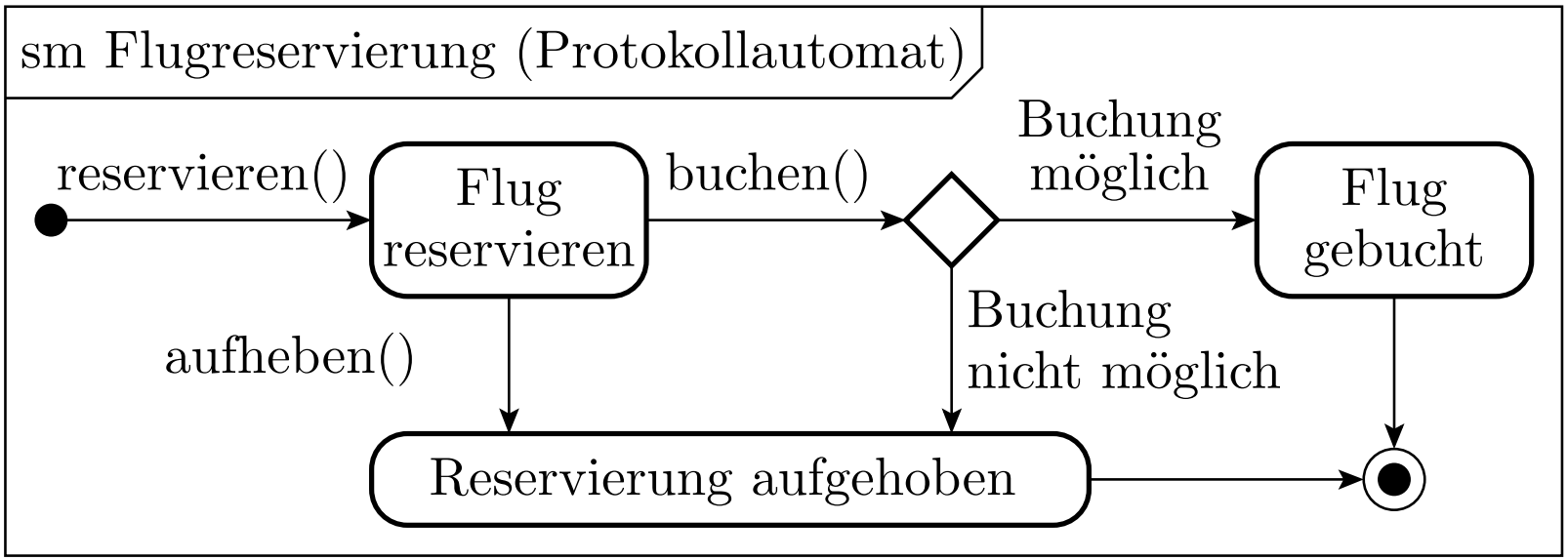
Abbruchbedingung: $l = r$

Nachbedingung: $A[l] = x$

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

Ablaufkontrolle

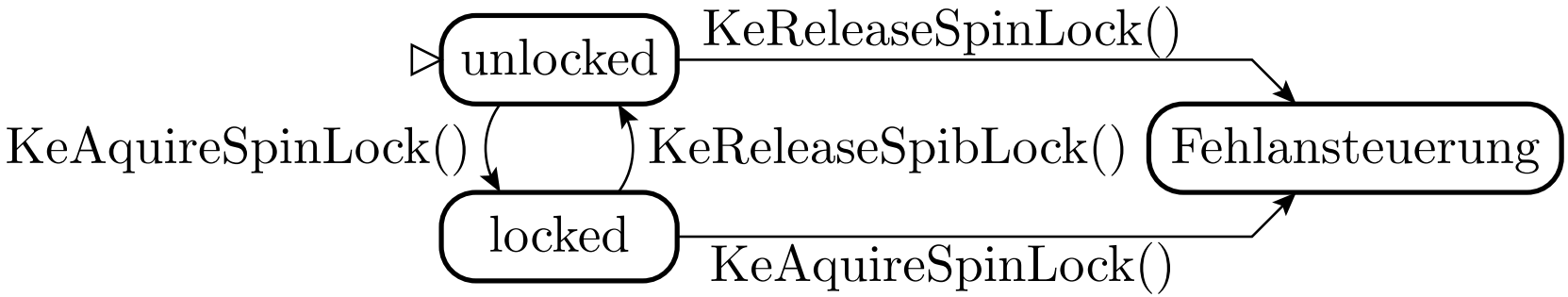
5.105 Protokollautomat



Protokollautomaten sind ein UML-Beschreibungsmittel zur Spezifikation zulässiger Aktionsreihenfolgen. Mögliche Aktionen im Beispiel sind die Methodenaufrufe »reservieren()«, »aufheben()« und »buchen()«. Aus dem Protokollautomat im Beispiel geht hervor, dass ein Flug nur nach erfolgreicher Reservierung gebucht und dass ein einmal gebuchter Flug nicht gestrichen werden kann.

UML Grafische Modellierungssprache zur Spezifikation von Software-Teilen.

5.106 API-Benutzungsregeln



Eine API definiert u.a. Benutzerregeln für die bereitgestellten Funktionen. Beispiel »spinlock« aus der Windows-API [1]:

- Spinlocks müssen alternierend reserviert und freigegeben werden
- durch Aufruf der Funktionen »KeAquire..« und »KeRelease«

beschreibbar durch einen Protokollautomaten. Protokollautomaten können als Überwachungsfunktionen zusammen mit einer Fehlfunktionsbehandlung für Protokollverletzungen in das System einprogrammiert werden. Alternative ist der Ausschluss durch statische Tests.

API — Eine API (Application Programming Interface) ist ein Satz von Befehlen, Funktionen, Protokollen und Objekten, die Programmierer verwenden können, um eine Software zu erstellen oder um mit einem externen System zu interagieren.

5.107 Eine zu testende Treiberfunktion

Eine Treiberfunktion ruft »KeAquire..« und »KeRelease...« u.U. mehrfach auf, in Fallunterscheidungen, Schleifen, ...

Für jeden Kontrollpfad muss der Spinlock alternierend bedient werden.

Fehlerausschluss erfordert Kontrolle für alle Pfade.

Reale Treiberfunktionen haben hunderte von Code-Zeilen. Kontrolle selbst so einfacher Regeln nicht trivial.

```
void example() {
 do {
   KeAcquireSpinLock();
   nPacketsOld = nPackets;
   req = devExt->WLHV;
   if(req && req->status){
    devExt->WLHV = req->Next;
    KeReleaseSpinLock();
    irp = req->irp;
    if(req->status > 0){
     irp->IoS.Status = SUCCESS;
     irp->IoS.Info = req->Status;
    } else {
     irp->IoS.Status = FAIL;
     irp->IoS.Info = req->Status;
    }
    SmartDevFreeBlock(req);
    IoCompleteRequest(irp);
    nPackets++;
   }
 } while(nPackets!=nPacketsOld);
 KeReleaseSpinLock();
}
```

Spezielle Kontrollen

5.108 Klassischer Rechtschreibtest

Wort im Wörterbuch enthalten?

- Maskierung, wenn Verfälschung zulässig und im Wörterbuch, z.B. »Maus« statt »Haus«.
- Phantom-MF, zulässiges Wort nicht im Wörterbuch.

Fehlfunktionsüberdeckung: Anteil der zulässigen Worte an den darstellbaren Zeichenfolgen fast null, aber beim Schreiben entstehen überdurchschnittlich oft zulässige Worte:

$$\mu_{\mathrm{MC}} = 60\% \ldots 90\% \ll 1 - \frac{\#VP}{\#PP}$$

Phantomfehlfunktionsrate: Viele zulässigen Zusammensetzungen und Abänderungen von Worten fehlen im Wörterbuch, typisch:

$$\zeta_{\mathrm{PM}} = \frac{1 \ldots 10\,[\mathrm{PM}]}{1000\,[\mathrm{DS}]} \gg 0$$

μ_{MC}	Zu erwartende Fehlfunktionsabdeckung.
ζ_{PM}	Phantom-Fehlfunktionsrate.
$[\mathrm{PM}]$	Zählwert in Phantom-Fehlfunktionen.
$[\mathrm{DS}]$	Zählwert in erbrachten Service-Leistungen.

Zusammenfassung

5.109 Kontrolle von Berechnungsergebnissen

Fehlererkennende Codes und Prüfkennzeichen sind ungeeignet. weil Codierung bzw. Prüfkennzeichenbildung erst nach der Berechnung der Datenobkelte erfolgen. Statt dessen:

- Wertebereichstest für Eingaben, Zwischenergebnisse und Ausgaben.
- Syntaxtest für manuelle Eingaben.
- Kontrolle von Ablaufregeln und Invarianten und
- spezielle auf die Zielfunktion zugeschnittene Kontrollen.

Die Fehlfunktionsabdeckungen dieser Kontrollen sind weitem nicht so gut wie bei den codebasierten Verfahren und auch schwer vorhersagbar. Dafür gibt es sehr viele Kontrollmöglichkeiten, mit denen sich insgesamt akzeptable Abdeckungen erzielen lassen.

Die Nutzung der Kontrollmöglichkeiten hängt maßgeblich von der Programmiersprache, der Softwarearchitektur etc. ab und wie einfach sich die Kontrollen incl. Problembehandlung beschreiben lassen.

Literatur

5. Literatur

[1] T. Ball.
Thorough static analysis of device drivers.
In *EuroSys*, pages 73–85, 2006.

[2] Nader B. Ebrahimi.
On the statistical analysis of the number of errors remaining in a software design document after inspection.
IEEE Transactions on Software Engineering, 23(8):529–532, 1997.

[3] Peter Liggesmeyer.
Software-Qualität: Testen, Analysieren und Verifizieren von Software.
Spectrum, 2002.

[4] Frank Padberg, Thomas Ragg, and Ralf Schoknecht.
Using machine learning for estimating the defect content after an inspection.
IEEE Transactions on Software Engineering, 30(1):17–28, 2004.

[5] Qinbao Song, Martin Shepperd, Michelle Cartwright, and Carolyn Mair.
Software defect association mining and defect correction effort prediction.
IEEE Transactions on Software Engineering, 32(2):69–82, 2006.