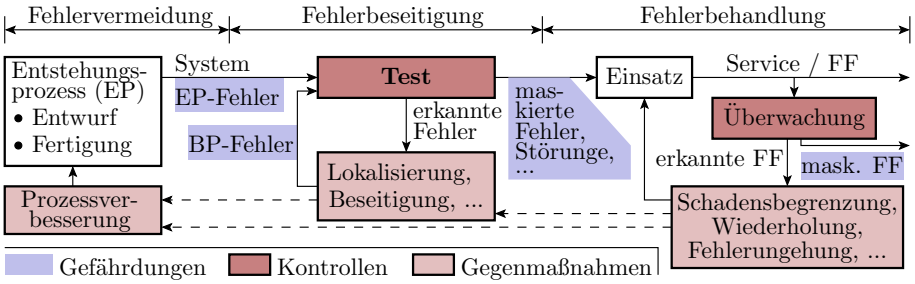
Test und Verlässlichkeit Foliensatz 4: Tests und Kontrollen

Prof. G. Kemnitz

April 10, 2022

Contents

Die 3 Ebenen zur Sicherung der Verlässlichkeit



Auf allen drei Ebenen zur Sicherung der Verlässlichkeit

1. Fehlervermeidung, Minderung und Beseitigung von Ursachen für die Fehlerentstehung.
2. Test und Fehlerbeseitigung.
3. Ergebnisüberwachung und FF-Behandlung.

erfolgt die Problembeseitigung durch Iterationen aus Kontrollen (Tests), Beseitigung erkannter Probleme und Testwiederholung.

Bei einer vernünftige Fehlerkultur werden alle erkennbaren relevanten Probleme beseitigt, so dass die Verlässlichkeit von der Anzahl und schwere der verborgenen Probleme bestimmt wird.

Die Tests und Kontrollen sind die Filter dafür, was unerkannt bleibt und damit Hauptansatzpunkt zur Schaffung verlässlicher Systeme.

Was lässt sich kontrollieren?

- physikalische Größen: elektrische, mechanische, ...
  (Entstehungsprozess, statische Tests, zu steuernder Prozess)
- analoge Werte: Signalverläufe von Sensoren und Aktoren,
- digitale Werte: Rechner Eingaben, Zustände und Ausgaben

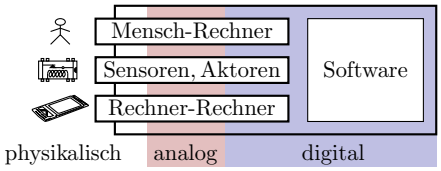
Einteilung der Kontrollen nach Ziel und Vorgehen:

- Inspektion (Sichtkontrolle), geht immer, aber eingeschränkt automatisierbar,
- auf Zulässigkeit (Ausnutzung von Informationsredundanz),
- auf Richtigkeit, insbesondere Vergleich mit Sollwerten.

1 Analog, physikalisch

1.1 Analog vs. digital

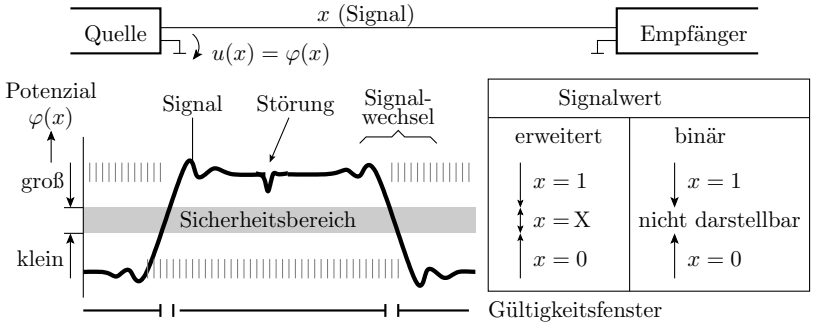
Struktur heutiger Hardware-Systeme



Die Hardware von IT-Systemen besteht aus physikalischen, analogen und digitalten Verarbeitungselementen:

- Physikalisch und analog in der Regel nur an den Schnittstellen zur Ein- und Ausgabe, Spannungversorgung, ... Separat zu testende Bausteinen. Überschaubare Funktionsvielfalt.
- Kompliziertere und spezielle Funktionen werden digital realisiert.
- Noch kompliziertere Funktionen: programmierbare Schaltungen und Rechnerstrukturen + Software.
- Überwachung und Fehlerbehandlung gehört zu den komplizierten und damit typisch in Software realisierten Funktionen.

Digital vs. Analog



Digitalisierung: Informationstdarstellung durch Bits

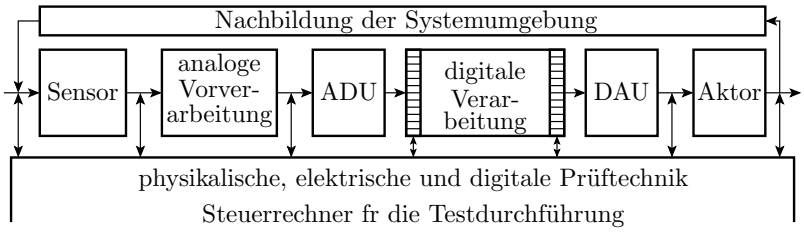
- physikalische Werteunterscheidungt nur groß, klein und ungültig,
- Abtastung im Gültigkeitsfenster.
- Robust gegen Verfälschungen durch Rauschen, induktives und kapazitives Übersprechen, Fertigungsstreuungen, Alterung, ...

Vorteile der digitalen Verarbeitung

- Information ist speicherbar.
- Speicherung und Verarbeitung ohne Informationsverlust
- größe relative Fertigungstoleranzen erlauben viel größere Integrationsdichten und höhere Geschwindigkeiten.
- Größerer funktionaler Gestaltungsspielraum.
- programmierbare und Rechnerstrukturen erlauben Funktionsfestlegung durch Software.
- Komplexe Digitalschaltungen werden heute wie Software entworfen.

1.2 Funktionstest

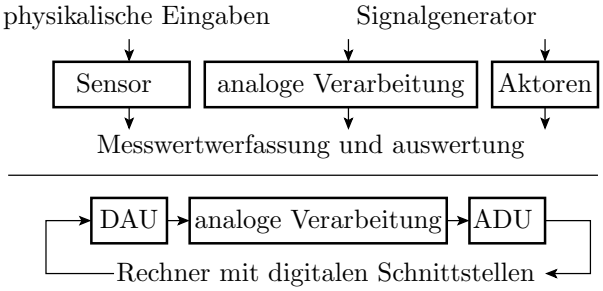
Modularar Test



Analoge Komponenten lassen sich entweder isoliert oder oder eingebettet in die Funktionsumgebung testen. Erfordert:

- Möglichkeiten der Kontaktierung und der Isolation von der Funktionsumgebung.
- Testhardware zur Bereitstellung von Eingaben und zur Auswertung von Ausgaben (physikalisch elektrisch, digital, Busprotokolle).
- Nachbildung der Systemumgebung (HIL – Hardware in the Loop).
- Teststeuerrechner.

Test analoger Bausteine



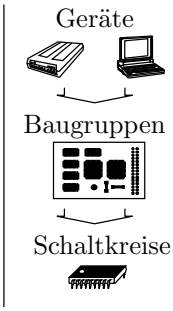
Der Sensortest verlangt physikalische Eingaben, der Test von analogen Verarbeitungseinheiten und Aktoren elektrische Eingabesignale. Die Ausgabesignale sind aufzuzeichnen und auf zu testende Merkmal zu kontrollieren.

In einem System mit Rechner ADU, analoger Verarbeitung und DAU kann man Rechner, ADU und DAU nachdem sie getestet sind, mit als Testhardware nutzen.

1.3 Statischen BG-Tests

Hierarchie und Test

Rechnerhardware besteht aus tauschbaren Komponenten:



- tauschbare Teilsystem.
- tauschbare Steckeinheiten,
- austauschbare Bauteile.

Die Komponenten werden vor Einbau in das übergeordnete System gründlich getestet.

Wiederholungsfragen:

1. Warum tauschbare Komponenten bzw. unter welcher Bedingung nicht erforderlich (nicht zweckmäßig)?
2. Warum ist ein gründlicher Komponententest zu fordern?
3. Warum beschränkt man sich beim Baugruppen- und Gerätetesttest teilweise auf einen Bestückungs- und Verbindungstest?

Bestückungs- und Verbindungstests

Hauptfehler auf Baugruppen sind Kurzschlüsse und Unterbrechungen. Nachweis durch Widerstandsmessungen zwischen und entlang der Verbindungen.
In der Serienfertigung erfolgt die Kontaktierung mit einem mit Unterdruck angesaugten Nadeladapter.

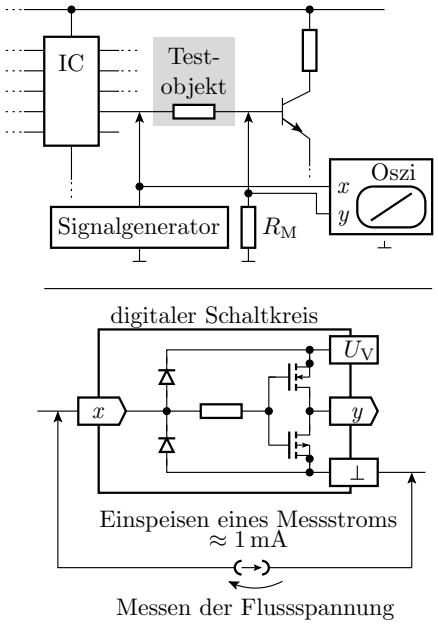
Die Nadeln sind mit reine Relais-Matrix zur Verbindung mit den Prüfgeräten angeschlossen. Auch Bestückungsfehler lassen sich überwiegend mit Zweipunktmessungen von Strom-Spannungsbeziehungen erkennen. Fehlerlokalisierung im Vergleich zu der für einen dynamischen Test, der versagt, sehr einfach.

Bestückungstests

Kontrolle auf Bestückungsfehler durch Überprüfung ausgewählter Zweipunktmerkmale:

- Widerstandswerte,
- Kapazitäten,
- Flussspannungen von Dioden, ...

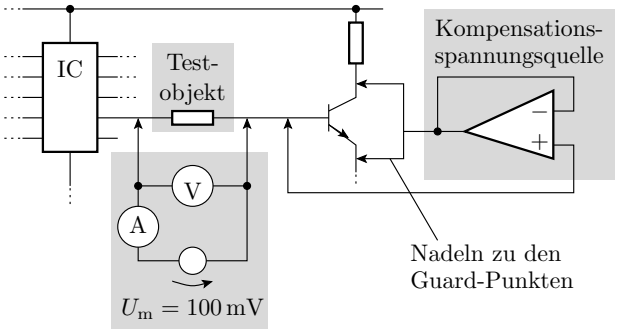
Zu Kontrolle, dass Schaltkreisanschlüsse mit dem Kontakt-Pad verbunden sind, werden die Schutzdioden zur Versorgungsspannung und Masse ausgemessen.



Analoger In-Circuit Test

Die Strom-Spannungs-Beziehung zwischen zwei Punkten hängt nicht nur vom Bauteil zwischen den Nadeln, sondern von allen Strompfaden, im Beispiel durch Transistor und Schaltkreis ab. Problematisch:

- die Toleranzbereiche der Sollwerte mit allen Bauteilstreuungen,
- die Erkennungssicherheit für Fehlbestückungen, z.B. bei sehr kleinen Kapazitäten.



Unterdrückung von Parallelströmen zum Testobjekt durch Kompensation der Spannungsabfälle über den wegführenden Bauteilen auf einer Testobjektseite auf null über »Guard-Punkte«.

Erlaubt einen isolierten Zweipoltest.

- Vereinfacht die Testauswahl, Sollwertfestlegung, ...
- Mindert die Häufigkeit von Fehlklassifikationen.

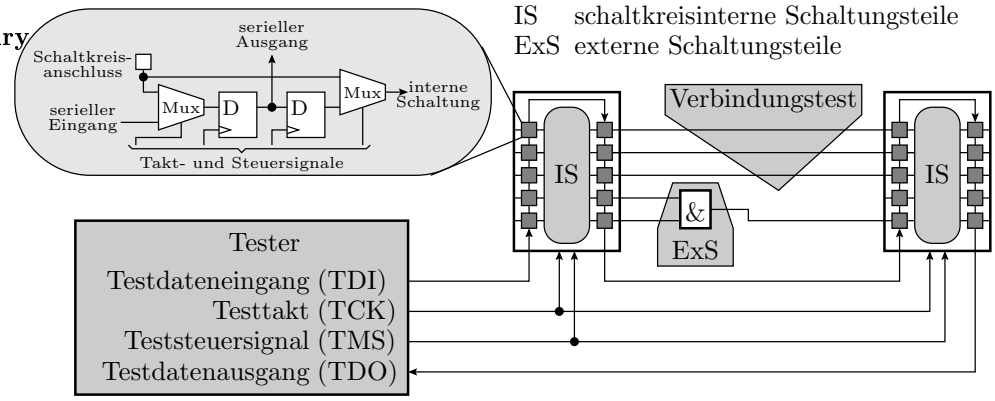
Optische Inspektion

Es gibt Bestückungsfehler, die sind optisch, aber nicht elektrisch erkennbar. Bild links korrekt bestückter SMD-Widerstand, rechts Lötfläche durch Kleber verschmutzt. Elektrisch leitende aber keine feste Lötverbindung:

Bilddatei a18.../B4?

Nachweis nur durch visuelle Kontrolle möglich. Besonderes Problem: Nach einem Ausfall der Baugruppe z.B. bei Vibration in einem Fahrzeug ist sofort erkennbar, dass es sich um einen Fertigungsfehler handelt, der (optisch) erkennbar gewesen wäre. Fall für Produkthaftung.

Boundary



Ersatz der mechanischen Nadeln durch »silicon nails« (seriell beschreibbare Register an den Schaltkreisanschlüssen, im Normalbetrieb überbrückt). Alternative zu den teuren, für jede Baugruppe speziell anzufertigenden Nadeladaptern.

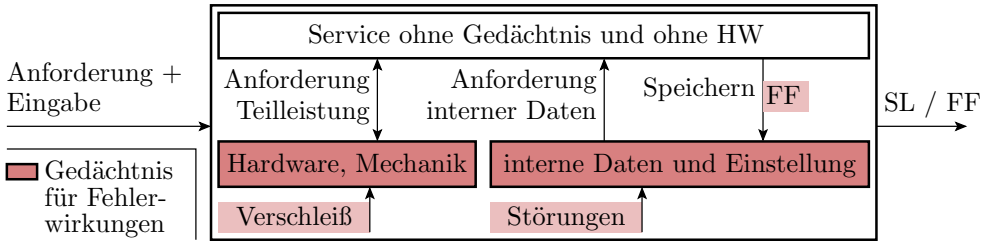
Ablauf eines Testschritts für den Baugruppentest:

- BS-Register aller Schaltkreise auf der Baugruppe seriell beschreiben,
- Datenübergabe (Update),
- Datenübernahme (Capture),
- serielle Ausgabe der übernommen Wert und Laden des Eingabevektors für den nächsten Testschritt.

Fortsetzung Abschn. # »Testbus«.

1.4 Zeitüberwachung

Burst-FF und Ausfälle



System mit Gedächtnis für Fehlerwirkungen:

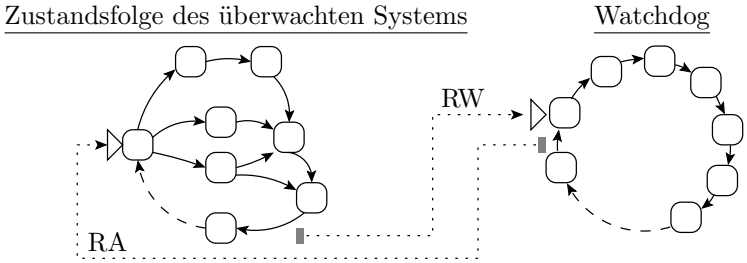
- Hardware und Mechanik, in der durch Verschleiß in der Einsatzphase neue Fehler entstehen können.
- Verfälschung service-interner Daten und Einstellungen durch FF.

Typische Fehlfunktion:

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

Fehlerbehandlung: [Reparatur,] Neuinitialisierung, Wiederholung.

Watchdog



RA Ein Überlauf des Watchdogs initialisiert den Automaten neu.

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

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

Das Watchdog-Prinzip wird angewendet für

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

Prozessoren haben in der Regel einen Hardware-Watchdog

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

2 Inspektion

Inspektion (Review)

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

- Dokumentationen (Spezifikation, Nutzerdokumentation, ...),
- Programmcode, Testausgaben,
- Schaltungsbeschreibungen, Konstruktionspläne, ...

Einordnung, Merkmale und Besonderheiten:

- wenn manuell, arbeitsaufwändig,
- zufälliger Fehlernachweis mit subjektiver Güte,
- auch Nachweis nicht funktionaler Fehler (Standardverletzungen, unsichere Beschreibungsmittel, ...),
- für frühe Entwurfsphasen geeignet,
- Know-How-Weitergabe.

Automatisierung anstrebenswert.

Kenngrößen einer Inspektion

Inspektionsfehlerüberdeckung:

$$\hat{IFC} = \frac{\#EF}{\#F}$$

($\#EF$ – Anzahl der nachweisbaren; $\#F$ – Anzahl aller (entstandenen) Fehler). Insgesamt oder getrennt für funktionale und sonstige Fehler.

Abschätzmöglichkeiten:

- Capture-Recapture-Verfahren (klassischer Ansatz).
- Modell Zufallstest.

Weitere Bewertungsgrößen für Inspektionen nach [3]:

- Effizienz: Gefundene Abweichungen pro Mitarbeiterstunde.
- Effektivität: Gefundene Abweichungen je 1000 NLOC[1].

Beispiel 1. Zähl- und Zeitwerte zur Bewertung einer 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

1. die Inspektionsfehlerüberdeckung,
2. die Effizienz und
3. die Effektivität

der Inspektion?

	gesamt	funktionale Fehler	sonstige Fehler
$\hat{IFC} = \frac{\#EF}{\#F}$	$\frac{228}{300}$	$\frac{156}{200}$	$\frac{72}{100}$
Effizienz	$\frac{228\text{ Fehler}}{200\text{ h}}$	$\frac{156\text{ Fehler}}{200\text{ h}}$	$\frac{72\text{ Fehler}}{200\text{ h}}$
Effektivität	$\frac{228\text{ Fehler}}{10.000\text{ NLOC}}$	$\frac{156\text{ Fehler}}{10.000\text{ NLOC}}$	$\frac{72\text{ Fehler}}{10.000\text{ NLOC}}$

Effizienz und Effektivität ergeben sich ausschließlich aus Zähl- und gemessenen Zeitwerten. IFC basieren auf schlecht überprüfbaren Schätzwerten für die Gesamtfehleranzahl.

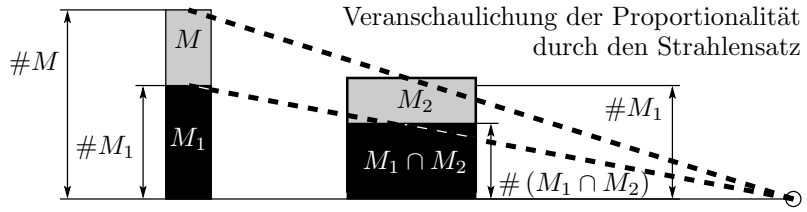
[1] NLOC: **N**etto **L**ines **o**f **C**ode. Anzahl der Code-Zeilen ohne Kommentar- und Leerzeilen.

Capture-Recapture-Verfahren

Abgeleitet von einem Schätzer für die Größe von Tierpopulationen (z.B. von Vögeln in einem Gebiet) [1, 5, 4].

- Aus einer Menge M unbekannter Größe wird eine Menge M_1 von Tieren eingefangen, gekennzeichnet und freigelassen.
- Nach Vermischung der Population Menge M_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:



$$\frac{\#M_1}{\#M} \approx \frac{\#(M_1 \cap M_2)}{\#M_2}$$

(#... – Größe der Mengen, hier Anzahl der Tiere; M – Menge aller Tiere, M_1, M_2 – beim ersten bzw. zweiten mal eingefangene Tiere; $M_1 \cap M_2$ – Menge der beide Male eingefangenen Tiere). Geschätzte Größe der Tierpopulation:

$$\#M \approx \frac{\#M_1 \cdot \#M_2}{\#(M_1 \cap M_2)}$$

Fehler statt Tiere

Zwei Inspekteure i finden jeweils eine Menge von M_i Fehlern:

$$\#M \approx \frac{\#M_1 \cdot \#M_2}{\#(M_1 \cap M_2)}$$

($\#(M_1 \cap M_2)$ – Anzahl der von beiden Inspekteuren unabhängig voneinander gefundenen gleichen Fehler; $\#M$ – geschätzte Anzahl der vorhandenen Fehler). Die geschätzte Fehlerüberdeckung ist das Verhältnis der Anzahl der insgesamt von beiden Inspekteuren gefundenen Fehler $\#(M_1 \cup M_2)$ zur geschätzten Gesamtfehleranzahl $\#M$:

$$IFC \approx \frac{\#(M_1 \cup M_2)}{\#M} \approx \frac{\#(M_1 \cap M_2) \cdot \#(M_1 \cup M_2)}{\#M_1 \cdot \#M_2}$$

Gebunden an die Annahmen:

- Inspekteure erkennen die Fehler unabhängig voneinander.
- Alle Fehler haben dieselbe Erkennungswahrscheinlichkeit.

Beispiel 2. Inspektionsergebnisse für ein Programm:

- Inspekteur 1: 228 gefundene Fehler, davon 156 funktionale.
- Inspekteur 2: 237 gefundene Fehler, davon 163 funktionale.

Schnittmenge: 105 Fehler, davon 73 funktionale.
Welche Schätzwerte ergeben sich nach dem Capture-Recapture-Verfahren für

1. die Gesamtfehleranzahl,
2. die Inspektionsfehlerüberdeckung?

1. Gesamtfehleranzahl:
$$\#M \approx \frac{\#M_1 \cdot \#M_2}{\#(M_1 \cap M_2)}$$
2. Inspektionsfehlerüberdeckung:
$$IFC \approx \frac{\#(M_1 \cap M_2) \cdot \#(M_1 \cup M_2)}{\#M_1 \cdot \#M_2} \tag{1}$$

Fehler	$\#M_1$	$\#M_2$	$\#(M_1 \cap M_2)$	$\varphi = \#M$	IFC
alle	228	237	105	515	70%
funktional	156	163	73	348	71%
sonstige	72	74	32	166	68%

Vertrauenswürdigkeit der Schätzung

Zufälliger Fehler:

1. Als Richtwert ist die Intervallbreite, um die im Mittel Zählwerte von ihren Schätzwerten abweichen, das zwei bis fünffache der Wurzel aus dem Zählwert (FS3, Abschn. Bereichsschätzung). Für einen Schätzwert 100 nicht erkannte Fehler beträgt das Intervall für den Erwartungswert $[80, 120]$ bis $[50, 150]$.
2. Bei Multiplikationen und Divisionen addieren sich die relativen Schätzfehler. In Gl. 1
$$IFC \approx \frac{\#(M_1 \cap M_2) \cdot \#(M_1 \cup M_2)}{\#M_1 \cdot \#M_2}$$
von vier Zählwerten, d.h. man benötigt 4^2 mal so große Zählwerte um die IFC mit derselben Genauigkeit/Irrtumswahrscheinlichkeit zu schätzen.

Die Zählwerte für die Fehleranzahlen müssten bei 10^3 bis 10^4 liegen, was sie in der Praxis nicht tun werden. Hinzu kommen systematische Fehler ...

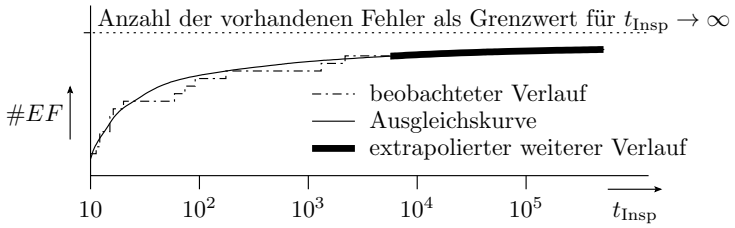
Systematische Fehler:

- Capture-Recaptur unterstellt für alle Fehler gleiche Erkennungswahrscheinlichkeit. Wenn gut erkennbare Fehler viel besser als schlecht erkennbare nachgewiesen werden, gelten die Schätzwerte nur für die gut erkennbaren.
- Capture-Recaptur verbietet Informationsaustausch zwischen den Inspekteuren. Falls es doch einen Informationsaustausch gibt, vergrößert der die Menge der gleichen gefundenen Fehler $M_1 \cap M_2$ gegenüber einer unabhängen Suche.
- Wenn die Inspekteure ihre Fehlerlisten voneinander abschreiben $M_1 = M_2$, ergibt sich als Schätzwert für die Inspektionsfehlerüberdeckung 100%.

Inspektion als Zufallstest

Einbeziehung, dass Fehlernachweiswahrscheinlichkeiten auch bei einer Inspektion um Größenordnungen variieren.

- Aufzeichnung der Anzahl der gefundenen Fehler in Abhängigkeit von der Inspektionsdauer.
- Abschätzen des weiteren Verlaufs.
- Gesamtfehleranzahl ist der Grenzwert für eine unendliche Inspektionsdauer[2]:

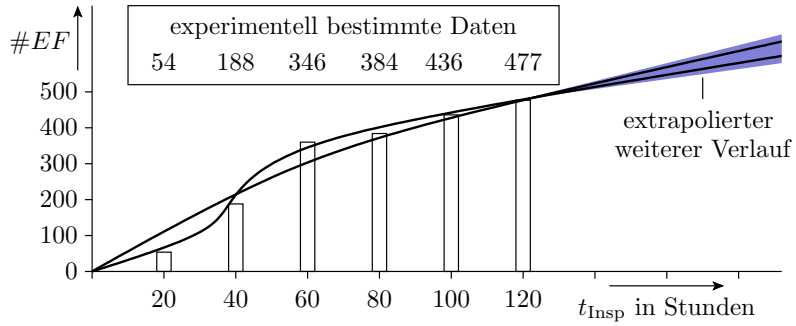


[2] Untersuchungen in dieser Richtung in der Literatur noch nicht gefunden.

Experiment mit einem Inspekteur[3]

Inspektion des Buchmanuskripts [2] plus Beispielprogramme:

- Anzahl der gefunden Fehler in Abhängigkeit von der Inspektionsdauer.



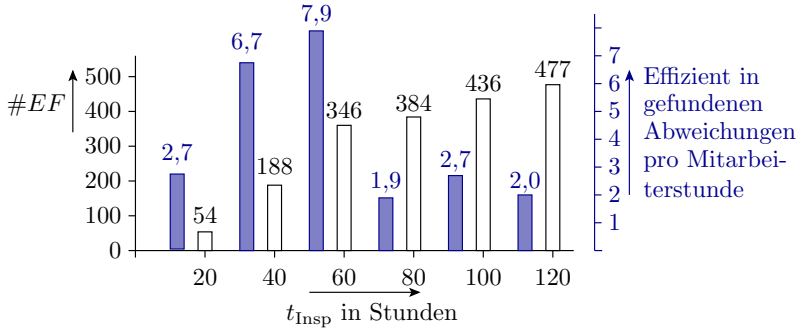
Unterschiedliche Approximationsmöglichkeiten für die weitere Abnahme der zu erwartenden Anzahl der nicht gefundenen Fehler, z.B. Abnahme der zu erwartenden Anzahl nicht gefundenen Fehler mit einem Exponenten $0 < k < 1$ (vergl. TV_F1):

$$\mathbb{E}\left[X\left(t_{\text{Insp}}\right)\right] = \mathbb{E}\left[X\left(t_0\right)\right] \cdot \left(\frac{t_{\text{Insp}}}{t_0}\right)^{-k}$$

(X – Anzahl der nicht nachweisbaren Fehler). Auch dieses Verfahren hat erhebliche Schätzfehler.

Untschiede zwischen Inspektion und Zufallstest

Bei einem Zufallstest nimmt die Effizienz (gefundene Abweichungen pro Mitarbeiterstunde) mit der Testdauer ab, weil nicht erkannte Fehler tendenziell schlechter als erkannte Fehler nachweisbar sind.



Die Beispielinspektion hatte offenbar eine »Anlernphase«, in der die Effizienz mit der Inspektionsdauer zugenommen hat.

- 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 ein erheblicher Anteil (ca. 25%) der Fehler noch nicht 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 vermeidet die uneffizienten Einarbeitungs- und Ermüdungsphasen.

[3] Bachelor-Arbeit von Yu Hong.

Inspektionstechniken

- Arbeit »geschickt« auf mehrere Inspekteure mit unterschiedlichen Rollen verteilen.
- Know-How-Weitergabe (Inspekteur ungleich Autor).
- Diversität ausnutzen »Vier Augen sehen mehr als zwei«.

Einteilung der Inspektionstechniken

- Review in Kommentartechnik: Korrekturlesen und Dokument 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, Inspektionstechnologie ...

3 Kontrolle »zulässig«

Formatüberwachung

Die Darstellungsmerkmale von SL unterteilen sich in:

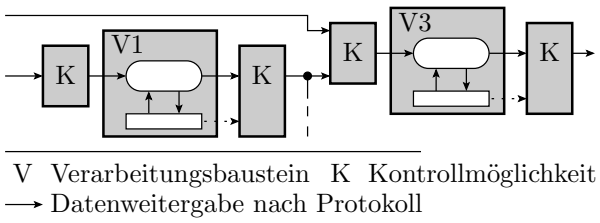
- Formatmerkmale: Konstante, für korrekte Informationen immer erfüllte Merkmale und
- Daten: Variable Merkmale.

Die Überwachung von SL während des Betriebs beschränkt sich oft auf Formatmerkmale (nur Kontrolle auf Zulässigkeit):

- Typ, Anzahl und Größe der Datenobjekte,
- Zeitschranken und andere Protokoll- und Signaleigenschaften,
- Wertebereiche, Invarianten,
- Syntax von Texteingaben, ...

Eine SL mit Formatfehler ist immer eine FF. Eine SL ohne Formatfehler kann, muss aber nicht ok. sein.

3.1 Schnittstellen, Protokolle



Komplexe Systeme, insbesondere mit Komponenten unterschiedlicher Hersteller, verteilte Systeme, Open-Source-Projekte, ... tauschen ihre Daten über ein wohldefinierte Schnittstellen und Protokolle aus:

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

Oft hunderte Seiten Doku

- welche Daten wie, mit welchen Befehlen in welcher Reihenfolge bei der Anforderung welcher SL zu übergeben sind,
- welche Vorbedingungen die Daten zu erfüllen haben,
- welche überprüfbaren Bedingungen korrekte Ergebnisse erfüllen,
- Antwortzeitschranken , ...

Festlegung von Schnittstellenformaten

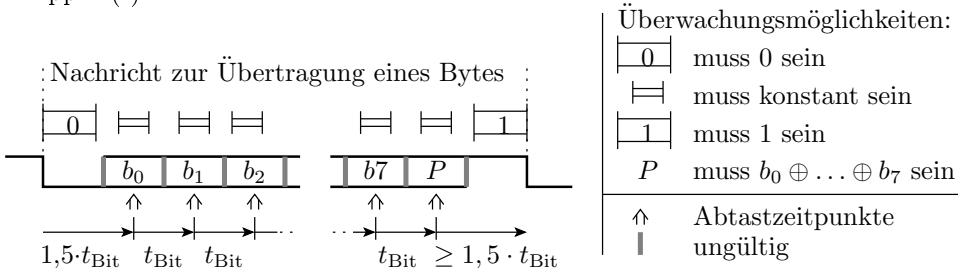
Der Entwurf einer IT-Funktionseinheit beginnt in der Regel damit. An den Schnittstellen entstehen bei nachträglicher Änderung und schlechter Dokumentation häufig Fehler. Eine geschickte Schnittstellenwahl hilft bei Vermeidung, Fehlererkennung, Fehlerisolation[4], Überwachung, ... Einige Regeln für die Festlegung:

- verständlich: Vermeidung fehlerhafter Ansteuerung durch Fehlinterpretation der Doku, ...
- Berücksichtigung eingeplanter Erweiterung: Vermeidung von Fehlern durch Nachbesserungen.
- Unterstützung statischer Tests bei Integration in übergeordnete Systeme: Syntax, Typenverträglichkeit, WB für Konstanten, ...
- Abfangen unzulässiger Werte durch Kontrollen während der Laufzeit zur Fehlerisolation: Wertebereiche, Invarianten, ...
- Standardisierte Formate, Protokolle, Sprachen, ...

Busprotokoll, Beispiel UART

Das UART-Protokoll ist der Klassiker zur bytweisen Datenübertragung zwischen Mikrorechner über eine Leitung je Übertragungsrichtung:

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



[4] Unterbindung der Beeinträchtigung der Funktion anderer Teilsysteme durch FF in einem Teilsystem.

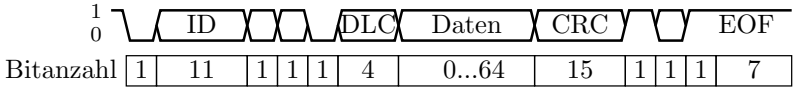
Kontrollierbare Formateigenschaften

Erkennbare Formatfehler

- Paritätsfehler: 1-Bit-CRC, $p_E \approx 50\%$ (siehe später Abschn. 3.65).
- Datenrahmenfehler: Datenwechsel in Zeitfenstern, in denen keine Änderung erlaubt. Zu kurze Start- oder Stoppbitzeit.
- Datenüberlauf: mehr ankommende Bytes als der Empfänger verarbeiten kann.

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

CAN-Bus: Vernetzung Fahrzeugsteuergeräte



ID	Nachrichtennummer	CRC	Prüfkennzeichen
DLC	Datenlängencode	EOF	Ende des Datenrahmens

Erkennbare Formatfehler:

- 15-Bit CRC als PKZ. Erkennungssicherheit (siehe später Abschn. 3.65)

$$p_E = 1 - 2^{-15} \approx 1 - 3 \cdot 10^{-5}$$

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

Fehlerbehandlung:

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

Ethernet-Paket

Ethernet-Paket:

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

Erkennbare Formatfehler:

- Prüfkennzeichenfehler. 4-Byte CRC, Erkennungssicherheit (siehe später Abschn. 3.65)

$$p_E = 1 - 2^{-32} \approx 1 - 2 \cdot 10^{-10}$$

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

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

- Wiederholte Anforderung nicht erhaltener Pakete, ...

3.2 Syntaxtest

Syntaxtest

Kontrolle, ob eine Zeichenfolge ein Wort einer formalen Sprache ist.

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

Anwendungen:

- wichtigster statischer Test für manuelle erstellte oder bearbeitete Programme.
- Kontrolle manueller Programmeingaben.

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

Syntaxtests erkennen viele menschengemachter Fehler. Für maschinell erzeugte Daten, die nicht manuell zu bearbeiten sind, eignet sich die Weitergabe mit Prüfkennzeichen oder verschlüsselt mit fehlererkennenden oder korrigierenden Codes besser.

EBNF als Beispielregelwerk für formale Sprachen

Beschreibungsmittel der EBNF (Eweiterte Backus-Naur-Form[5]) zur Definition von Sprachregeln:

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

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

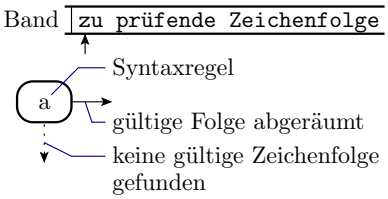
Beispiele für EBNF-Syntaxregeln

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

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

Spracherkennende Automaten

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



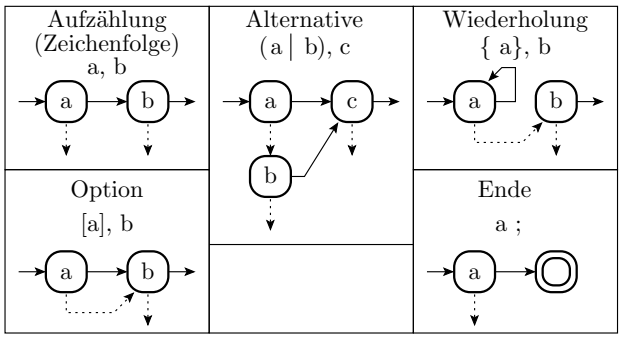
In jedem Automatenzustand wird versucht, eine Zeichenfolge nach der definierten Syntaxregel abzuräumen:

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

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

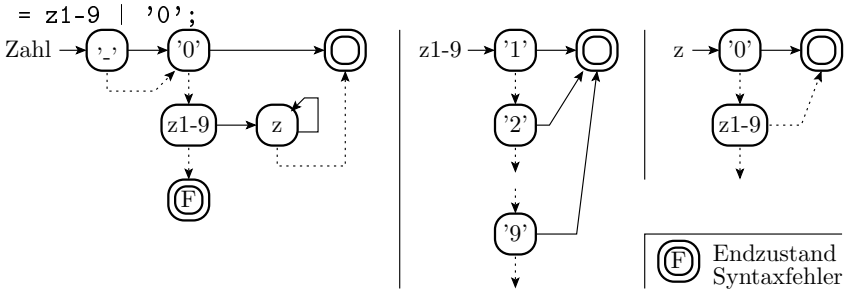
[5] Siehe http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf.

Von der EBNF zum Automaten



Beispiel:

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



Welche Zustände durchläuft der Automat. Was erkennt er?

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

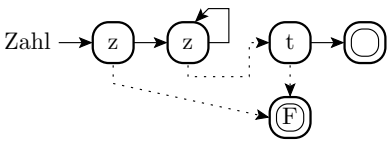
Abräumen an den Kanten

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

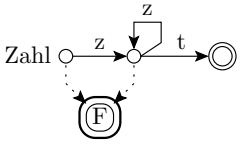
Beschreibung einer Zahl:

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

Abräumen in den Zuständen



Abräumen an den Kanten



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

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

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

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

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

3.3 Invarianten, WB

Invarianten

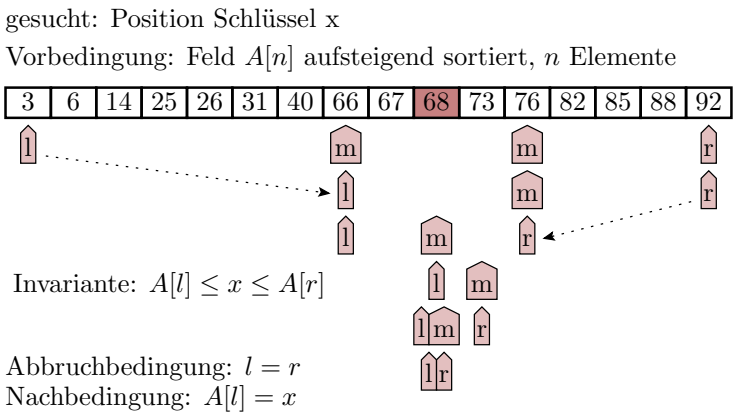
Invariante: Aussage, die über die Ausführung bestimmter Programmbefehle hinweg gilt. Überprüfbar

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

Beispiele für Invarianten:

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

Beispiel mit einer Schleifeninvarianten



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

Wertebereichskontrolle

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

```
u32 a;            // Alter Angestellter WB:18...67
                  // WB(u32):0...0xFFFFFFFF
if (a < 18 || a > 67) <Fehlerbehandlung>;
```

Wenn bei Verfälschung alle darstellbaren Werte gleichwahrscheinlich sind, Erkennungswahrscheinlichkeit:

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

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

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

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

Tatsächliche Erkennungswahrscheinlichkeit der WB-Überwachung:

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

Verbesserbar durch

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

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

Die Leistungsfähigkeit des Verfahrens liegt in der Vielzahl der Kontrollmöglichkeiten.

3.4 Fehlererk. Codes

Wiederholung Informationsredundanz

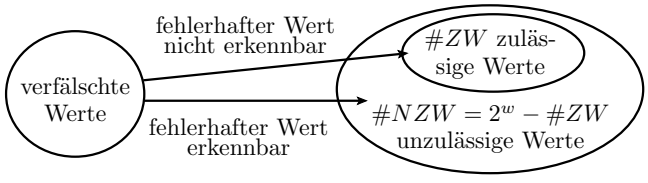
Die Unterscheidung von $\#ZW$ zulässigen Werten verlangt

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

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

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

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



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

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

$$p_{\mathrm{E}} = \mathbb{E}\left[FFC\right] = 1 - \frac{\#ZW}{2^w} \tag{2}$$

($\#ZW$ – Anzahl der zulässigen Werte; w – Bitanzahl zur Darstellung). Mit einer Informationsredundanz von r zusätzlichen (redundanten) Bits:

$$r = w - w_{\min}$$
$$p_{\mathrm{E}} = 1 - \underbrace{\frac{\#ZW}{2^{w_{\min}}}}_{\leq 1} \cdot 2^{-r} \geq 1 - 2^{-r}$$

Für eine Erkennswahrscheinlichkeit von praktisch eins genügen $r \approx 10 \ldots 20$ redundaten Bits. Voraussetzung:

- Erkennen aller unzulässiger Werte und
- Abbildung fehlerhafte Werte gleichmäßig auf zulässige und unzulässige Werte.

Verfälschungen überproportional oft

- zulässige Werte (z.B. bei Rechtschreibtest) p_{E} deutlich geringer
- unzulässige Werte (z.B. Paritätstest DRAM) p_{E} deutlich höher.

Beispiel Rechtschreibtest

Wort im Wörterbuch enthalten?

- Maskierung: falsches Wort, das im Wörterbuch enthalten ist, z.B. »Maus« statt »Haus«. Größenordnung der Erkennungswahrsch.
$$p_{\mathrm{E}} \approx 80\%$$
- Phantom-FF: zulässiges Wort nicht im Wörterbuch. Größenordnung der Phantom-FF-Rate:
$$\zeta_{\mathrm{Phan}} \approx \frac{1\ \mathrm{Phantom-FF}}{100\ \mathrm{kontrollierte\ Worte}}$$

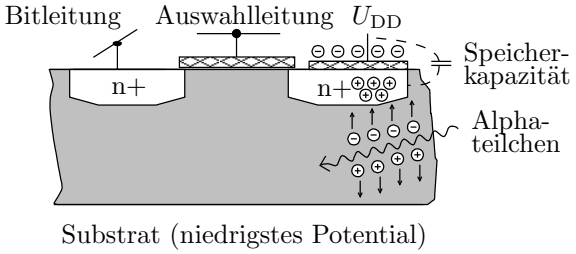
Bei gleichmäßiger Abbildung auf zulässige und unzulässige Werte Anzahl der mit #Bytes darstellbaren Zeichenketten:

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

z.B. 2^{80} für 10 Zeichen große Worte, davon zulässig ca. 10^6 Worte:

$$p_{\mathrm{E}} \approx 1 - \frac{10^6}{2^{80}} \approx 1 - \frac{10^6}{10^{24}} = 1 - 10^{18} \gg 80\%$$

Paritätstest für DRAMs und Speicherriegel



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

- Energie eines Alphateilchen: 5 MeV. Energieverlust bei der Generierung eines Elektronen-Loch-Paares $\approx 3{,}6\,\mathrm{eV} \Rightarrow$ Generierung von $\approx 10^6$ Ladungsträgerpaaren. Reichweite $\approx 89\,\mu$m. gespeicherte Ladung $\approx 10^5$ Ladungsträger. Datenverlust einer oder mehrerer benachbarter Zellen möglich.

- Mittlerer Zeitabstand zwischen zwei Datenverfälschungen Stunden. Gleichzeitige Verfälschung durch zwei Alphateilchen unwahrscheinlich.

- Geometrische Trennung der Zellen eines Datenworts (getrennte Schaltkreise oder Speichermatrizen) $\Rightarrow$ Je Zerfall Einzelbitverfälschung je gelesenes Datenwort.

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

Einzelbitfehler bei Übertragung und Speicherung

Wenn die Daten so auf Datenobjekte aufgeteilt werden, dass jede der seltenen Verfälschungsursachen nur in jedem Datenobjekt ein Bit verfälscht, genügt ein fehlererkennender Code für Einzelbitfehler (Paritätsbit).

Für unabhängige Bitverfälschungen und eine zu erwartene Anzahl je Datenwort λ ist die Anzahl der verfälschten Bits poisson-verteilt:

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

Erkennungswahrscheinlichkeit:

$$p_\mathrm{E} = \frac{\mathbb{P}\left[X = 1\right]}{\mathbb{P}\left[X \geq 1\right]} = \frac{e^{-\lambda} \cdot \lambda}{1 - e^{-\lambda}}$$

Für $\lambda \ll 1$ und Taylor-Reihe $\mathrm{e}^{-\lambda} = 1 - \lambda + \frac{\lambda^2}{2} - \ldots$:

$$p_\mathrm{E} = \frac{(1-\lambda) \cdot \lambda}{1 - (1-\lambda)} = 1 - \lambda$$

Gleichm. Abb. auf zul. und unzul. Werte und $r = 1$ Paritätsbit $p_\mathrm{E} = 50\%$

Arithmetische Codes

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

$$s = 34562134 \cdot x$$

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

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

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

Zyklische Codes, Polynommultiplikation

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

Codierung

```
          10010101101
⊙             10011
-------------------
⊕       10010101101
⊕      10010101101
⊕     00000000000
⊕    00000000000
⊕ 10010101101
-------------------
  10001110100111
```

Decodierung

```
 100011100100111 : 10011 = 10010101101
⊕10011
-------
  10110
 ⊕10011
 -------
  10101
  10011
 -------
   11000
  ⊕10011
  -------
   10111
  ⊕ 10011
  -------
    10011
   ⊕10011
   -------
Rest: 0000        (unverfälscht)
```

Mathematisch werden die zu multiplizierenden Faktoren als Polynome dargestellt:

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

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

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

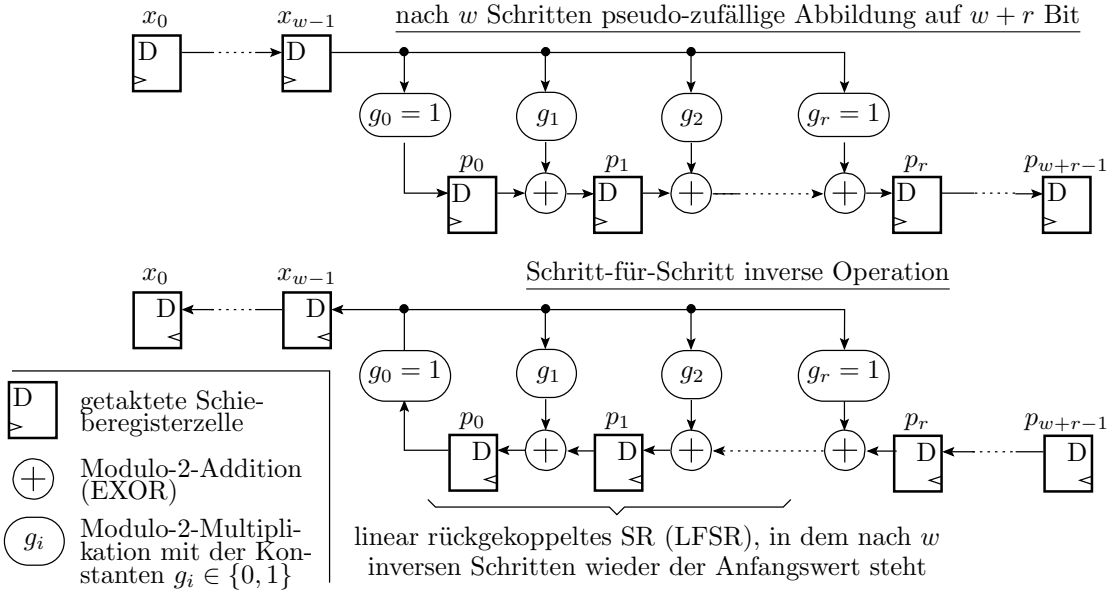
repräsentiert denselben Bitvektor, der für die Multiplikation der Folgen auf der Folie zuvor berechnet wurde. Die Polynomdivision:

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

liefert ohne Rest das Polynom der Originalfolge.

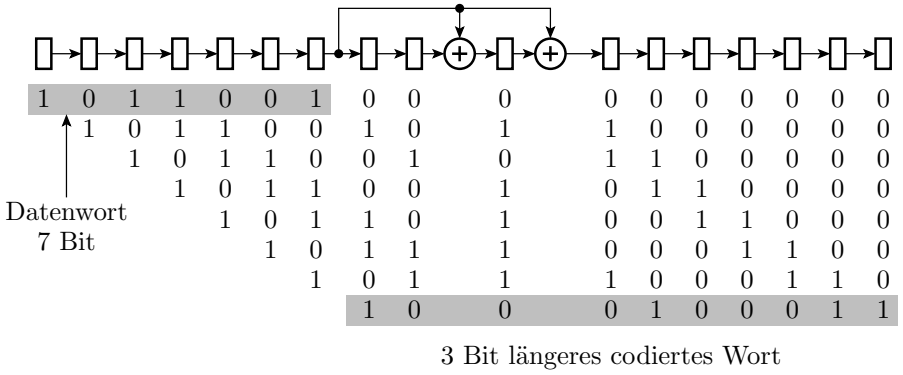
Polynommultiplikation und -Division mit SR

Codierung mit SR (**S**hift **R**egister), Decodierung mit LFSR (**L**inear **F**eedbeack **S**hift **R**egisters).



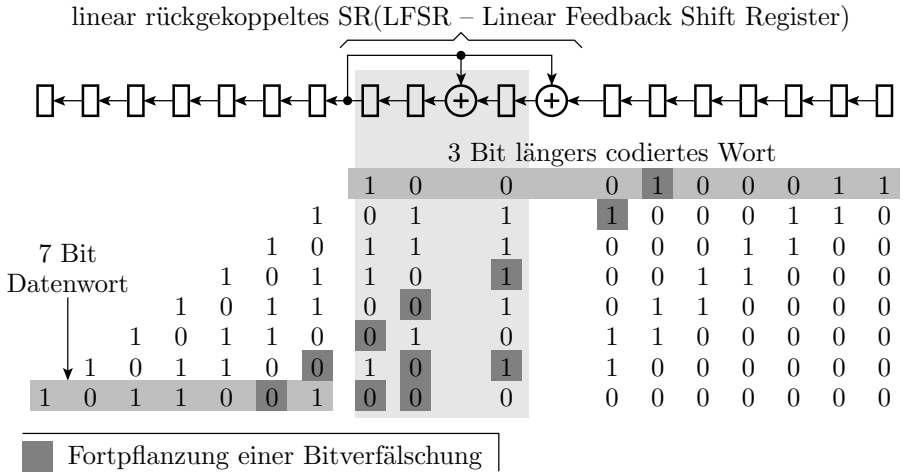
Erkennungswahrscheinlichkeit: $p_E = 1 - \frac{2^w}{2^{w+r}} = 1 - 2^{-r}$

Beispiel für die Codierung



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

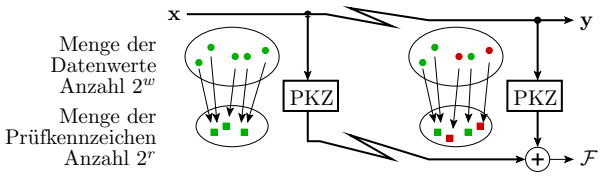
Rückgewinnung und Kontrolle



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

3.5 Prüfkennzeichen

Prüfkennzeichen



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

Für pseudo-zufällig gebildete Prüfkennzeichen gilt:

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

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

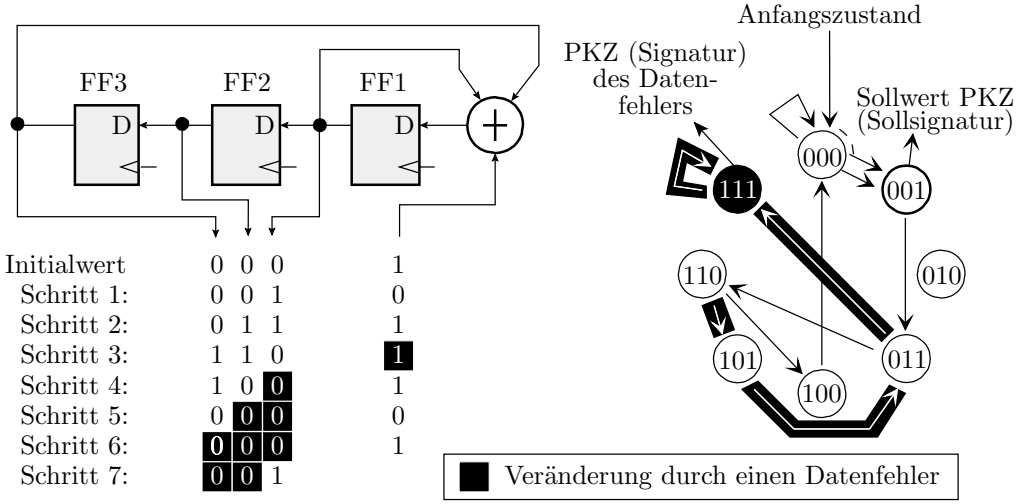
Prüfsummen

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

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

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

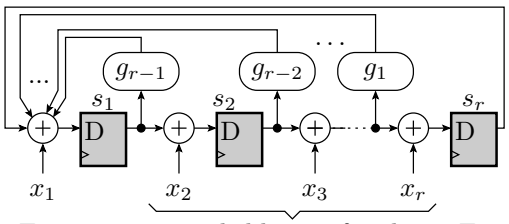
Prüfkennzeichenbildung mit LFSR[7]



Das Prüfkennzeichen wird wie bei der CRC-Decodierung mit einem linear rückgekoppelten Schieberegister (LSFR) gebildet. Im Beispiel hat das LSFR im Gegensatz zur Polynomdivision Seite 64 eine zentrale Rückführung. Abbildung auch pseudo-zufällig.

Allgemeine Struktur von LFSR

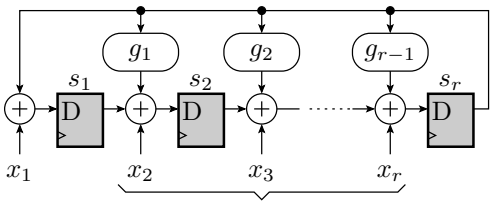
Für die Bildung auf Prüfkennzeichen ist es nur wichtig, dass die Abbildung pseudo-zufällig hinsichtlich der zu erwartenden Verfälschungen erfolgt. Diese Eigenschaft hat auch ein rückgekoppeltes Schieberegister, bei dem die Daten, ein oder mehrere Bits, modulo-2 als zu den Registerzuständen addiert werden.



Erweiterungsmöglichkeit auf mehrere Eingänge

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

[6] Kein Nachweis für vertauschte Summationsreihenfolge.
[7] LFSR – **L**inear **F**eeback **S**hift **R**egister.

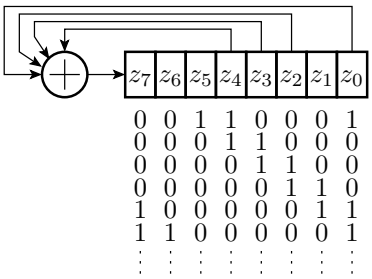


Erweiterungsmöglichkeit auf mehrere Eingänge

Die Koeffizienten g_i der Rückführung, bei der Polynom-Division das Divisor-Polynom, bestimmen die autonome Zyklusstruktur[8]. 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.

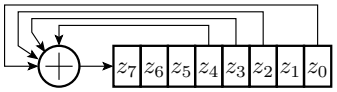
Autonome Zykluslänge und -struktur von LFSR

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



Übergangsfunktion:

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



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

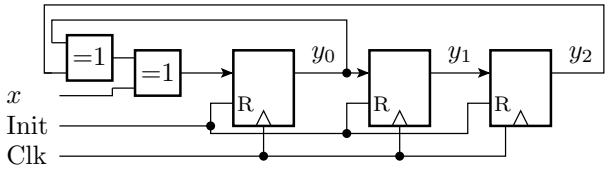
```
...
#define ZA 0x31
uint8_t z = ZA;      // Startwert setzen
while(1){
  z = (z>>1) ^ (z<<7) ^ ((z<<5)&0x80)
      ^ ((z<<4)&0x80) ^ ((z<<3)&0x80);
  <Ausgabe von z>
  if (z==ZA) break; // bis wieder Anfangswert
  ...               // weiter mit nicht enthal-
}                   // tenem Zustand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

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

[8] Zyklusstruktur ohne Eingaben.

Beispielaufgabe

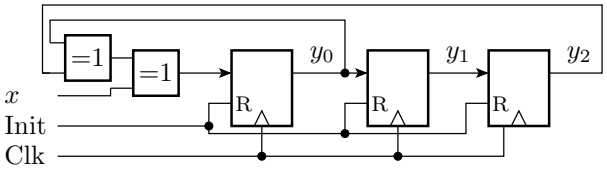
Gegeben ist folgendes linear rückgekoppelte Schieberegister:



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

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

Lösung



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

Erkennungswahrscheinlichkeit:

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

Zusammenfassung

Datensicherung mit fehlererkennenden Codes / Prüfkennzeichen:

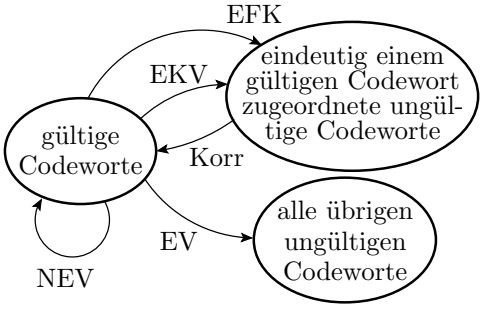
- Geringer Berechnungsaufwand.
- Geringer Zusatzaufwand für Datenübertragung und -speicherung (r zusätzlich gespeicherte / übertragene Bits für eine Datenobjekt beliebiger Größe).
- Maskierungswahrscheinlichkeit 2^{-r}. Mit ausreichendem r immer vernachlässigbar klein.

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

3.6 Fehlerkorr. Codes

Fehlerkorrigierende Codes

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



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

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

($b = w + r$ – Bitanzahl; $\#CWG$ – Anzahl gültige Codeworte; $\#CWK/CWG$ – Anzahl korrigierbare Codeworte je gültiges Codewort). Die Erkennungswahrscheinlichkeit als Anteil der übrigen ungültigen Codeworte verringert sich durch Korrekturmöglichkeiten.

Beispiel: Korrektur von Einzelbitfehler

Anzahl korrigierbare Codeworte je gültiges Codewort gleich Bitanzahl:

$$\#CWK/CWG = b$$

Mindestbitanzahl:

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

Für #CWG = 256 gültige Codeworte:

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

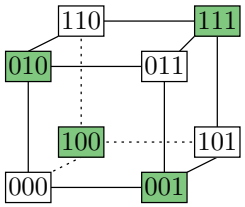
Probe:

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

3.7 Hamming-Codes

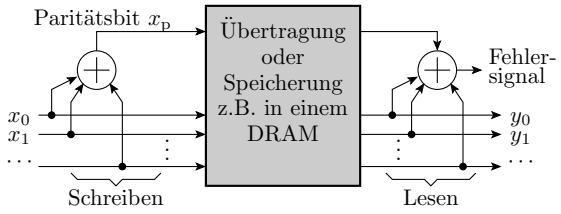
Hamming-Distanz

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



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

Parität als Prüfkennzeichen (Hamming-Dist. 2)



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

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

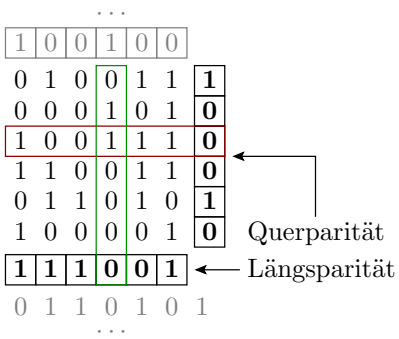
bei gerader Anzahl von Einsen »0« sonst »1«.

- Erkennt jede ungeradzahlige Anzahl von Bitverfälschungen.
- Wenn geradzahlige und ungeradzahlige Bitfehler gleichhäufig auftreten: $p_{\mathrm{E}} \approx 50\%$
- Bei unabhängiger Verfälschung mit Erwartungswert $\lambda \ll 1$ (typ. für DRAMs, vergl. Seite 58):

$$p_{\mathrm{E}} = 1 - \lambda$$

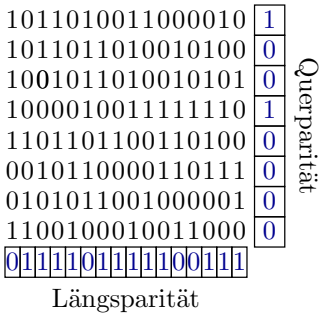
Kreuzparität (Fehlerkorrigierender Paritätscode)

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

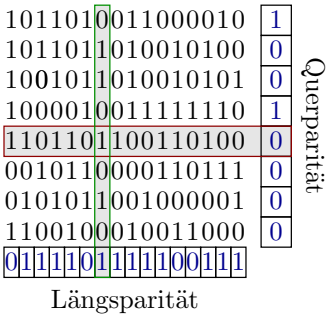


Beispielaufgabe

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



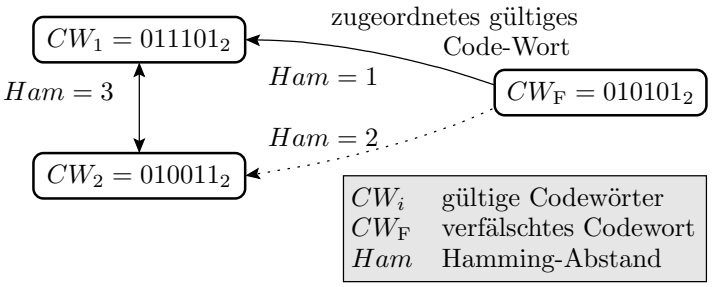
Lösung



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

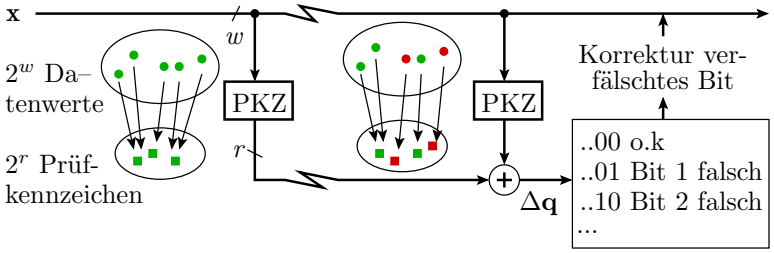
1-Bit fehlerkorrigierende Hamming-Codes

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



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

Konstruktion 1-Bit fehlerkorrigierender Code



Ergänzung des w-Bit Datenworts $\mathbf{x}$ um ein r-Bit Prüfkennzeichen so, dass die mod-2 Summen des übertragenen und des nach der Übertragung gebildeten Prüfkennzeichens die Bitnummer des verfälschten Bits ist.

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

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

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

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

Beispielzuordnung:

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

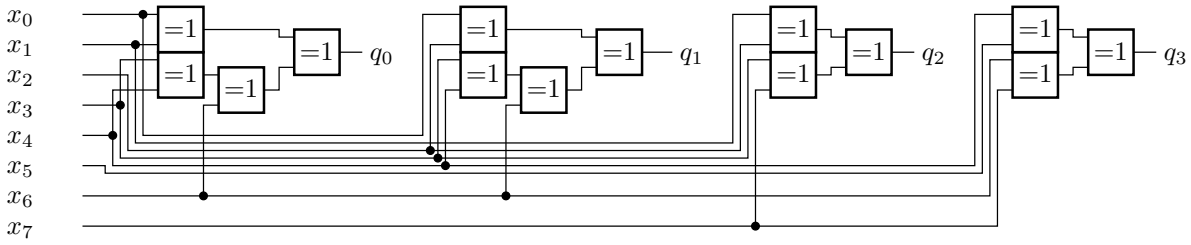
Für die erste Summe gilt:

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

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

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

Codierschaltung:



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

Bitfehlerkorrektur als VHDL-Case-Anweisung:

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

Beispielaufgabe

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

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

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

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

Lösung

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

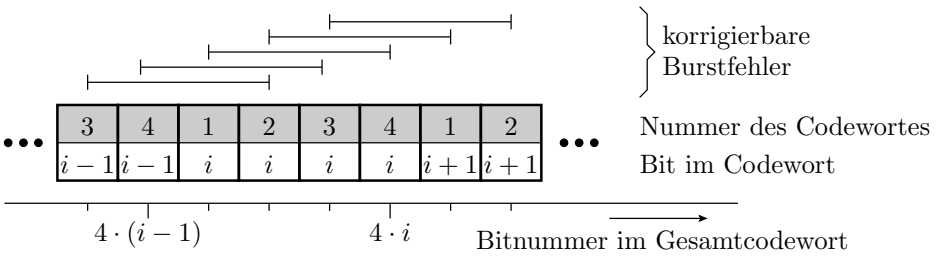
w_2 : Wert 0xA2 mit verfälschtem $x_7 \Rightarrow w_2 =$0x22

3.8 Burstkorrektur

Korrektur von Burstfehlern

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

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



Beispielaufgabe

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

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

Bits 30 bis 32

Lösung

1.

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

2.

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

Durch Burstfehler invertierte Bits 30 bis 32,

4 Kontrolle »richtig«

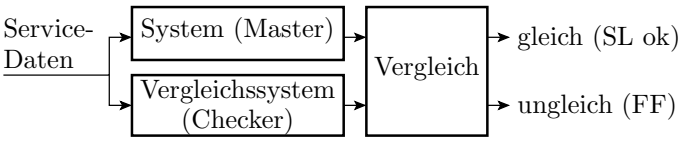
Kontrolle auf Richtigkeit

Eine Kontrollen auf Richtigkeit kontrolliert zusätzlich zum Format (wertunabhänge Merkmale von SL) die Werte der SL.

Möglichkeiten zur Wertekontrolle:

- Vergleich mit vorab berechneten Sollwerten (nur unter Testbedingungen),
- Vergleich mit»Golden Device« (Referenzsystem, dessen Ausgaben per Definition als korrekt gelten),
- Mehrfachberechnung und Vergleich während der Laufzeit,
- Rückgewinnung der Eingaben und Vergleich,
- Funktionsspezifische Kontrollen.

Verdopplung und Vergleich



FF-Überdeckung von Mehrfachberechnung und Vergleich (vergl. Foliensatz F1, Absch. 3.2 Fehlerbehandlung. Überwachungsverfahren):

$$FFC = Div = \frac{\#DFF}{\#FF}$$

(Div– Diversität; $\#DFF$ – Anzahl der Master-FF, bei denen eine Wiederholung zum korrekten Ergebnis oder zu einer geänderten FF führt). Phantom-FF-Rate:

$$\zeta_{\text{Phan}} \approx \zeta_{\text{VS}} \cdot (1 - Div)$$

ζ_{Phan} – FF-Rate Vergleichssystem, ζ_{VS} – FF-Rate Vergleichssystem.

Nicht erkennbare, d.h. übereinstimmende haben praktisch immer eine gemeinsame Ursache. *Warum?*

Man überschlage, wie wahrscheinlich es ist, dass zwei SL bei geringer FF-Rate und vielen Ergebnisbits zufällig übereinstimmen.

Mögliche Ursachen für übereinstimmende FF:

- gleicher Entwurfs- oder Fertigungsfehler,
- dieselbe Störung, Eingabe- oder Zustandsverfälschung,
- Fehler in oder bei der Interpretation der Spezifikation,
- Fehler in der Entwurfssoftware (Compiler, ...), in genutzten Bibliotheken, Dokumentationen, ...

Mögliche Ursachen für erkennbare FF:

- Störungen, Fehler in nur in einer Berechnung, Ausfall eines Teilsystems,
- gemeinsame Ursachen, aber nicht deterministische Systemverhalten, abweichender Berechnungsweg, ...

Mehrfachberechung und Vergleich ist brauchbar:

- Vorabberechnung von Sollwerten für den Test,
- mit gründlich getesteten Referenzsystem »Golden Device« oder Regressionstest (Vorversion als Referenzsystem)
- Störungen und Ausfälle als FF-Ursache.

Mehrfachberechung und Vergleich ist auch kritisch in Kombination mit FF-Korrektur durch Wiederholung, weil da nicht nur Maskierung, sondern auch Umwandlung von Phantom-FF in richtige (siehe auch später Foliensatz F7, Ausfälle und Fehlertoleranz).

Schaffung von Diversität

Störungen und FF durch Fehler ohne deterministische Wirkung bewirken eine kleine natürliche Diversität. Diversität gegenüber weiteren FF-Ursachen verlangt Zusatzmaßnahmen:

1. Hardware-Diversität: Unterschiedliche HW.
2. Hardware-Entwurfsdiversität: Unabhängig entworfene HW.
3. Syntaktische Diversität: Unterschiedlich übersetzte Software.
4. Software-Diversität: Unabhängig entworfene SW.
5. Diversitärer Service Anforderung oder Funktion[9].

Ursachen, für die FF korrigierbar sind:

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

Anmerkungen zur Software-Diversität

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

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

Ursprüngliche euphorische Meinung, dass so Diversität gegenüber allen Fehlern, außer denen in der Spezifikation erzielbar sei, nicht bestätigt. Die direkte oder indirekte Kommunikation der Entwicklungsteams über die Interpretation der Spezifikation, während des Test etc. trägt Gemeinsamkeiten in die Entwürfe. Neigung von Menschen, gewisse Fehler zu wiederholen, ... Erzielbare Divisität laut[10]

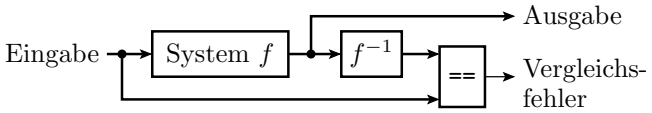
$$Div \leq 90\%$$

[9] Ungeeignet für Verdopplung und Vergleich, da abweichende Sollwerte. .

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

Rückrechnung und Vergleich

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



Beispiele für Funktionen mit Umkehrfunktion:

- Quadrierung $\leftrightarrow$ Wurzelberechnung,
- Analog/Digital-Wandlung $\leftrightarrow$ Digital/Analog-Wandlung,
- Daten versenden $\leftrightarrow$ Daten empfangen, ...

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

Funktionsspezifische Kontrollen

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

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

Solche funktionsspezifischen Kontrollen umgehen oft das Diversitätsproblem durch Verschiedenartigkeit von Berechnung und Kontrolle. Hohe Erkennungswahrscheinlichkeiten erzielbar. Leider gibt es für die meisten Zielfunktion keine derartige Kontrollmöglichkeit für die SL auf Richtigkeit.

5 Literatur

Literatur

[1] 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.

[2] Günter Kemnitz. *Technische Informatik 2: Entwurf digitaler Schaltungen.* Springer, 2011.

[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.