Test und Verlässlichkeit (F4)
Foliensatz 4:
Test und Fehlerbeseitigung

Prof. G. Kemnitz

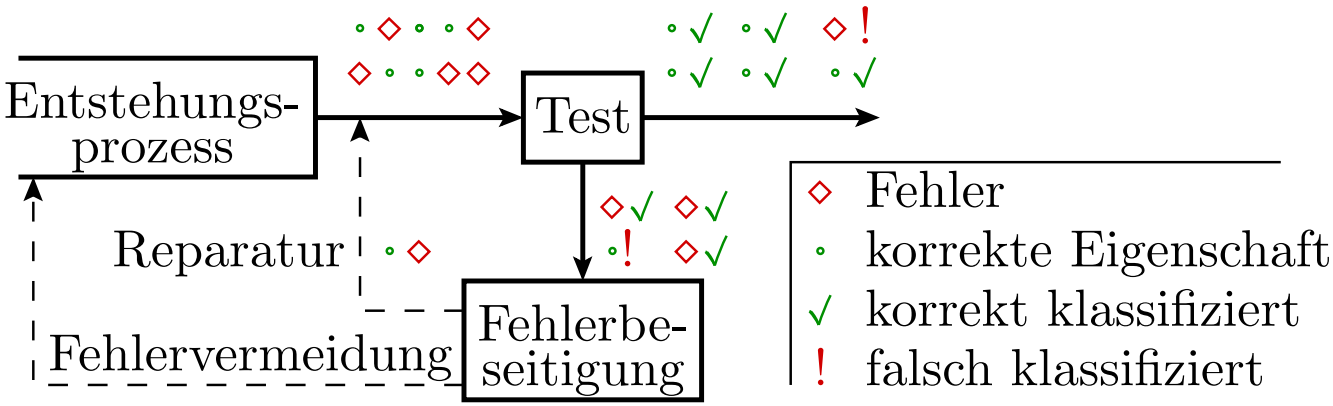
Institut für Informatik, Technische Universität Clausthal

29. Juni 2014

Inhalt F4: Test und Fehlerbeseitigung

Überblick

Test und Fehlerbeseitigung



Tests sind Kontrollen der Eigenschaften[1] des Systems. Fehlerhafte Eigenschaften werden mit einer gewissen Wahrscheinlichkeit erkannt und einer Fehlerbeseitigung zugeleitet. Beseitigung bedeutet in der Regel Reparatur oder Ersatz der kleinsten lokalisierbaren fehlerhaften Einheit. Erkannte Fehler liefern Hinweise auf ihre Entstehungsursache und Möglichkeiten für eine künftige Fehlervermeidung.

[1] Nicht wie auf dem vorherigen Foliensatz der Ergebnisse.

Zu testende Anforderungen

Sprichwort: Vertrauen ist gut, Kontrolle ist besser.

Nach diesem Motto: Kontrolle, was nicht zusicherbar ist:

- funktionale Anforderungen, Bedienbarkeit,
- selbstverständliche Anforderungen, z.B. dass eine Software installierbar sein muss,
- Einhaltung von Standards,
- Funktionsfähigkeit unter vorgeschrieben Umgebungsbedingungen (nur im Labor, auch im Freien, in Fabrikhallen, ...),
- Verlässlichkeitsanforderungen (Wartbarkeit, Zuverlässigkeit, Sicherheit, ...), ...

Erkannte Fehler und Probleme erfassen. Fehler beseitigen. Für Probleme, die keine Fehler sind, Lösungen suchen. Der Blickwinkel des Foliensatzes: Kontrolle funktionaler Anforderungen und Beseitigung funktionaler Fehler.

Statische und dynamische Test

Tests werden grob eingeteilt in:

- dynamische Tests (D): stichprobenweises Ausprobieren der Service-Leistungen und
- statische Tests (S): alle anderen Arten von Kontrollen am System (Syntaxtest, Korrekturlesen von Entwurfsbescheibungen, Sichtkontrollen an Baugruppen, ...).

Statische und dynamische Tests weisen unterschiedliche Fehler nach und werden in der Regel kombiniert, z.B.

- Kontrolle der Spezifikation durch den Auftraggeber (S),
- Kontrolle der Lösungsidee durch einen Kollegen (S),
- Syntaxtest des Programms (S),
- Ausprobieren eigener Testbeispiele (D),
- Ausprobieren beim Anwender (D),
- Kontrolle der Dokumentationen (S).

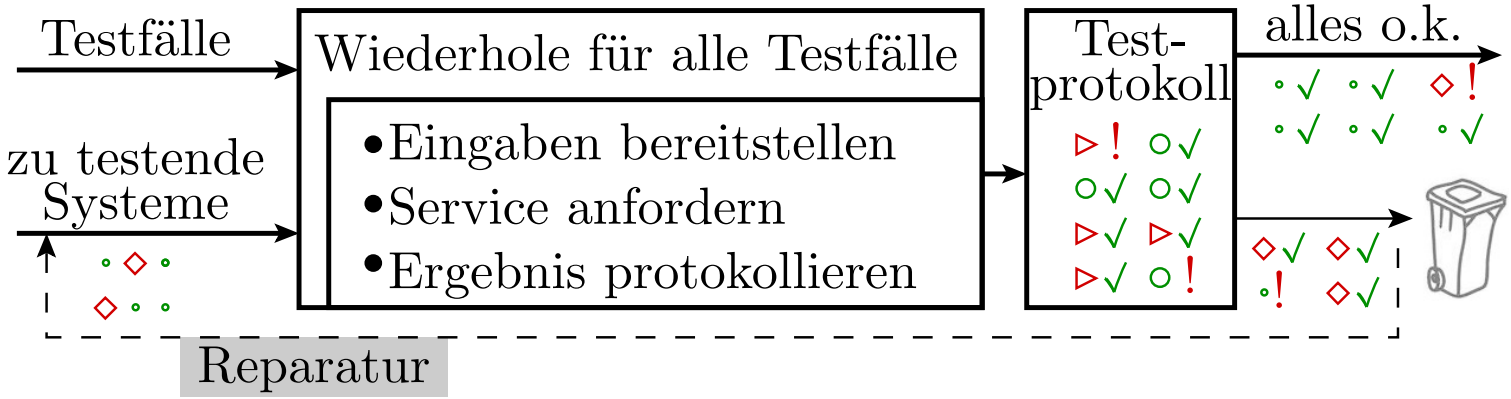
Beispiel Programmieraufgabe in einem Praktikum:

- Erstellen der Aufgabenstellung: Korrekturlesen durch den Lehrenden (S).
- Beispielimplementierung durch den Lehrenden, Syntaxtest (S), Beispiele ausprobieren (D).
- Lösungssuche durch den Studierenden und Diskussion über die Lösung mit dem Praktikumspartner (S).
- Programmieren und Syntaxtest (S), Ausprobieren (D).
- Praktikumsbetreuer schaut sich den Code an (S).
- Praktikumsbetreuen lässt sich Beispiel vorführen (D).

Statische Tests (S) dienen meist als Vorfilter und dynamische Tests (D) zur gründlichen Kontrolle.

Die gebräuchlichsten automatisierten statischen Tests für IT-Systeme, Syntaxtest und Typenkontrolle, wurden auf Foliensatz F3 behandelt. Die andere große Klasse, Inspektionen, werden wegen ihrem starken Bezug zur Fehlervermeidung erst auf Foliensatz F5 behandelt.

Dynamische Tests



◇ Fehler ▷ fehlerhaftes Ergebnis ! falsch klassifiziert
◦ korrekte Eigenschaft ○ korrektes Ergebnis √ korrekt klassifiziert

Ein dynamischer Test arbeitet eine Menge von Testfällen ab und erstellt ein Protokoll, welche Tests das Testobjekt bestanden hat und welche nicht. Fehlklassifizierungen, dass negative Testergebnisse als gut und positive als Versagen klassifiziert werden, nicht ausgeschlossen. Objekte, für die die Tests versagt haben, werden aussortiert oder repariert und nochmal getestet.

Testsatz und Testfall

Ein Testsatz besteht aus einer Menge von Testfällen. Eine Testfallbeschreibung umfasst mindestens

- die Testeingaben,
- zu kontrollierende Ergebnisse und
- Mittel zu Testdurchführung (Hardware, Testrahmen).

Die Ergebniskontrolle ist in der Regel ein Vergleich mit Sollwerten (exakt oder Fenstervergleich (siehe F3, Abschn. 4.1). Die Hardware für die Testdurchführung kann ein normaler Rechner mit Betriebssystem und bestimmten Mindestanforderungen oder ein spezieller Tester sein. Der Testrahmen ist das Programm, das die Testsätze abarbeitet und das Testprotokoll erzeugt und hat idealerweise für die Fehlersuche auch die Möglichkeit, Testfälle, Teiloperationen und Einzeloperationen schrittweise auszuführen und Zwischenergebnisse zu kontrollieren.

Testfall nach [ANSI/IEEE-Standard 829]

Testfälle dienen nicht nur zur Aufdeckung von Fehlern, sondern auch zur Vertrauensbildung und zur Absicherung des Herstellers, dass er seine Pflicht, den Anwender vor Schaden zu bewahren, erfüllt hat. Der Standard [ANSI/IEEE-Standard 829] empfielt dazu eine Ergänzung der Testfallbeschreibung um:

- Testfall-Identifikation: eindeutiger Bezeichner
- Testgegenstand: Referenz auf die Bescheibung, aus der Anforderungen überprüft werden,
- Zweck: Anforderung, deren Erfüllung der Test bestätigt,
- Testfallstatus: spezifiziert, durchgeführt, ...

und die Erzeugung eines Testprotokolls für jedes als gut befundene Objekt, aus dem hervorgeht, aus welchen Dokumenten welche Anforderungen mit welchem Ergebnis kontrolliert wurden.

Prüfgerechter und testgetriebener Entwurf

Der Aufwand für den Testentwurf ist im Vergleich zum eigentlichen Entwurf sehr hoch:

- Testfälle erstellen und verwalten
- Testrahmen schreiben,
- Tests durchführen, Ergebnisse auswerten,
- Testprotokolle verwalten.

Hinzu kommt, dass bei der alt hergebrachten Herangehensweise »erst das System und dann die Testfälle entwerfen« Letzteres nicht immer lösbar ist, so dass Anforderungen und Systembestandteile ungetestet bleiben. Der moderne Ansatz lautet, parallel in jeder Entwurfsphase und zur Spezifikation jeder Anforderung gleich Testfälle sammeln bzw., Anfordereungen wie Testfälle spezifizieren. Bei Hardware wird diese Strategie als prüfgerechter und bei Software als testgetriebener Entwurf bezeichnet.

Produkthaftung und Standards

Größere IT-Systeme haben statistisch gesehen immer Fehler und andere Schwachstellen, die in der Anwendungsumgebung zum Teil erheblichen Schaden verursachen können. Nach Schadenseintritt lässt sich oft rückwirkend nachweisen, dass die Ursache ein Entwurfs- oder Fertigungsfehler war.

- Wer haftet?

Der Hersteller kann sich nur der Haftung entziehen, wenn er nachweisen kann, dass er alle dem Stand der Technik entsprechenden Maßnahmen zur Schadensabwendung ergriffen hat, d.h. den Fehler trotz ausreichend gründlichem Test übersehen hat. Um diesen Nachweis erbringen zu können, gibt es Standards, die in durchführbarer und überprüfbaren Weise festlegen, was für Tests und anderen verlässlichkeitssichernde Maßnahmen als ausreichend gelten und wie deren Erbringung zu dokumentieren bzw. nachzuweisen ist.

Aktuelle Standarts und Normen

- ISO 9126/DIN 66272: Qualitätsmerkmale für Software.
- V-Modell XT: Leitfaden zum Planen und Durchführen von Entwicklungsprojekten unter Berücksichtigung des gesamten Systemlebenszyklus.
- ANSI/IEEE Std 829-1998: Standard für Software Test Dokumentation.
- ANSI/IEEE Std 1008-1993: Standard für Software Unit Test.
- ANSI/IEEE Std 1012-1998: Standard für Software Verfication and Validation Plans.
- DIN 66 285: Anleitung für die Vergabe von Gütesiegeln für Software-Produkte.
- Die ISO/IEC 14598:1999-2001:v Sechbändige Norm, die den Prozess der Bewertung eines Software-Produkts beschreibt.
- ...

Fehlermodellierung

Zufallstest

Ein größeres System hat statistisch gesehen Fehler. Welche ist unbekannt. Erkannte Fehler werden beseitigt. Der Ausschluss aller Fehlermöglichkeiten durch Kontrolle aller Anforderungen ist ausgeschlossen. Jeder Test ist deshalb nur ein Filter, der einen Teil der Fehler erkennt. Der Anteil der erkennbaren Fehler ist die Fehlerüberdeckung:

$$FC = \frac{\varphi_{\mathrm{N}}}{\varphi_{\mathrm{E}}}$$

(φ_{N} – Anzahl der nachweisbaren; φ_{E} – Anzahl aller (entstandenen) Fehler, Zufallsgrößen).

Abschätzmöglichkeiten:

- aus statistischen Erfahrungswerten oder
- über Modellfehlermengen.

Am einfachsten abschätzbar für einen Zufallstest, d.h., wenn die Eingaben zufällig ausgewählt werden.

Nachweisbedingungen

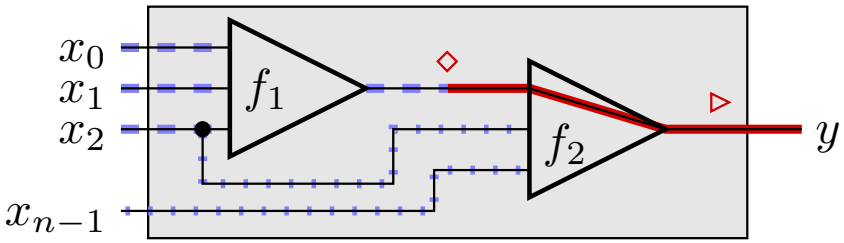
Modellfehler, Nachweismengen und -bedingungen

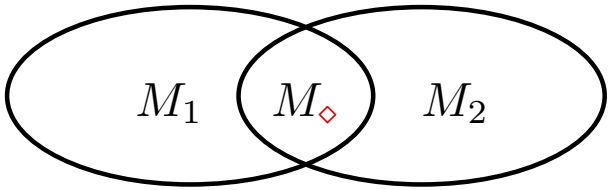
Vorhersagen der Fehlerüberdeckung verlangen Annahmen über die möglicherweise vorhandenen Fehler, deren Nachweismengen und/oder Nachweisbedingungen.

- Modellfehler: Annahmen für mögliche Fehler.
- Nachweismenge: Menge von Eingaben, mit denen ein Fehler nachweisbar ist.
- Nachweisbedingungen: Voraussetzungen für den Fehlernachweis, z.B., dass für den Nachweis eines Fehlers in einem Teilservice, diese ausgeführt und seine Ergebnisse beobachtbar sein müssen.

Der Nachweis eines lokalen Fehlers in einem System verlangt Testeingaben, die

- den Fehler anregen[2] und
- einen Beobachtungspfad erzeugen, entlang dem sich die Verfälschung zu einem beobachtbaren Ausgang fortplanzt.



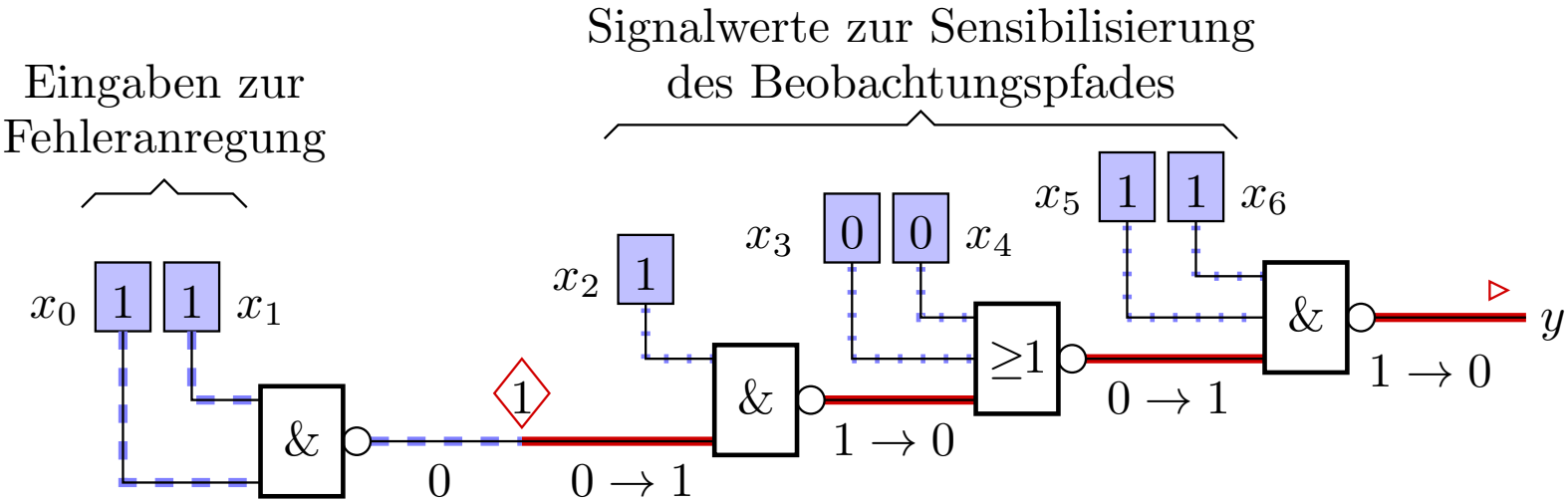


◇	Fehler
▷	Fehlfunktion (Datenverfälschung)
- -	Eingaben zur Fehleranregung
· · ·	Einstellen der Beobachtbarkeit
———	Beobachtungspfad
M_1	Eingabemenge, mit der der Fehler angeregt wird
M_2	Eingabemenge, bei der der Fehlerort beobachtbar ist
$M_\diamond$	Nachweismenge des Fehlers

Die Nachweismenge eines (Modell-) Fehlers ist die Schnittmenge der Eingabemengen, die die einzelnen Nachweisbedingungen erfüllen.

[2] Eingaben, bei denen der Fehler eine lokale Datenverfälschung bewirkt.

Nachweisbedingungen in einer Gatterschaltung



- Signalwerte für den Fehlernachweis
- Fehler (ständig 1, stuck-at 1)
- Fehlfunktion (Ausgabeinvertierung)
- Eingaben zur Fehleranregung
- Einstellung der Beobachtbarkeit
- Beobachtungspfad

Eingabemenge Fehleranregung: $M_1 = \{\text{-----11}\}$

Eingabemenge Beobachtbarkeit: $M_2 = \{\text{11001--}\}$

Fehlernachweismenge: $M_1 \cap M_2 = \{1100111\}$

Verallgemeinerung

- Der Fehlernachweis kann auch von gespeicherten Zuständen abhängen. Anregung/Beobachtung über eine Eingabefolge.
- Der Fehlernachweis kann weiterhin von eingabeunabhängigen Bedingungen abhängen, z.B. Bereich der Versorgungsspannung, ...

Aufspaltung des Fehlernachweises in mehrere Einzelbedingungen:

$$\mathrm{x} \in (M_1 \cap M_2 \cap \ldots \neq \varnothing) \wedge B_1 \wedge B_2 \wedge \ldots$$

(M_i – Eingabemenge einer notwendigen Anregungs- oder Beobachtungsbedingung; B_i – eingabeunabhängige Nachweisbedingung).

Modellfehler und Fehlermodelle

- Ein Modellfehler ist eine einfach nachzubildende oder zu simulierende Änderung an der korrekten Systembeschreibung, die sich mit möglichen Fehlern viele Nachweisbedingungen teilt.
- Ein Fehlermodell ist ein Algorithmus/Rezept/Regelwerk, mit dem für das Testobjekt eine Menge von Modellfehlern ausgewählt wird.
- Modellfehler dienen entweder zur Suche geigneter Testeingaben oder zur Abschätzung der Fehlerüberdeckung für einen gegebenen (meist zufällig ausgewählten) Testsatz.

Wiederhole für jeden Modellfehler
Suche einen oder mehrere Tests, mit denen er nachweisbar ist.

Wiederhole für jeden Modellfehler
Test, ob vom Testsatz nachweisbar Zählen, wenn nachweisbar.

Ein denkbares Fehlermodell für Programme

Jeder ganzzahlige Operand soll einmal um eins erhöht und einmal um eins verringert sein.

- fehlerfreies Testobjekt:

```
1: int a, b, c, d;
2: a=b-c;
3: if (a>2) d=c;
4: else d=b;
```

- Modellfehler (die übrigen Anweisungen bleiben unverändert)

```
M1:  Veränderung Zeile 2: a=(b+1)-c;
M2:  Veränderung Zeile 2: a=(b-1)-c;
M3:  Veränderung Zeile 2: a=b-(c+1);
M4:  Veränderung Zeile 2: a=b-(c-1);
```

```
M5:  Veränderung Zeile 3: if ((a+1)>2) d=c;
M6:  Veränderung Zeile 3: if ((a-1)>2) d=c;
M7:  Veränderung Zeile 3: if (a>2) d=(c+1);
M8:  Veränderung Zeile 3: if (a>2) d=(c-1);
M9:  Veränderung Zeile 4: else d=(b+1);
M10: Veränderung Zeile 4: else d=(b-1);
```

Identisch nachweisbare Modellfehler können zu einem Modellfehler zusammengefasst werden, im Beispiel M1 und M4,:

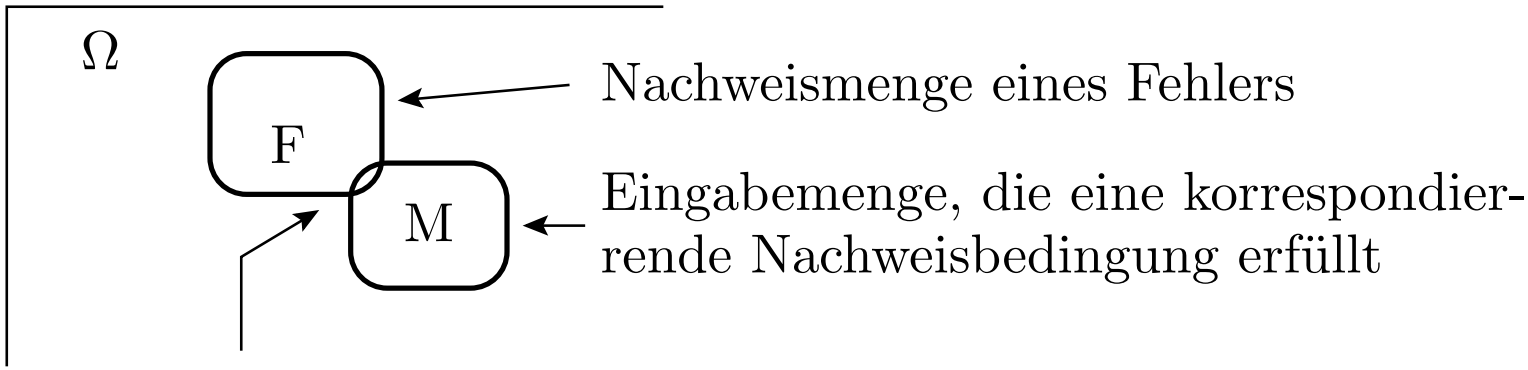
$$(b+1)-c = b-(c-1)$$

Ein Fehlermodell ist so zu wählen, dass die Modellfehlerüberdeckung, die sich bestimmen läßt, Rückschlüsse auf die tatsächliche Fehlerüberdeckung erlaubt. Der Zusammenhang zwischen tatsächlicher und Modellfehlerüberdeckung hängt von den Größenverhältnissen der Nachweismengen, deren Überschneidungen, aber auch von der Art der Testauswahl ab.

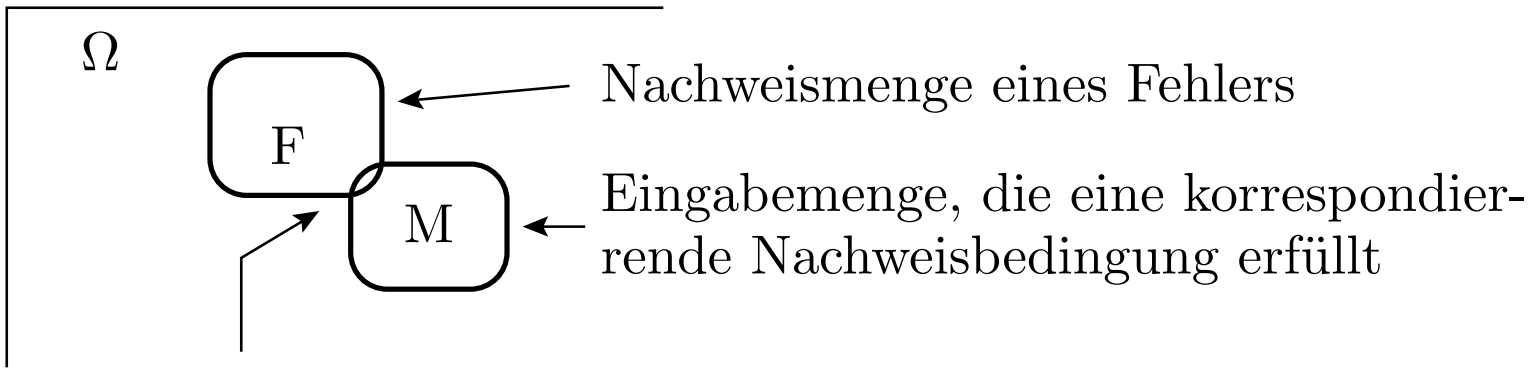
Testauswahlstrategie

Gezielte Testauswahl

Für jeden Modellfehler wird eine oder werden mehrere Eingaben gesucht, die ihn nachweisen.



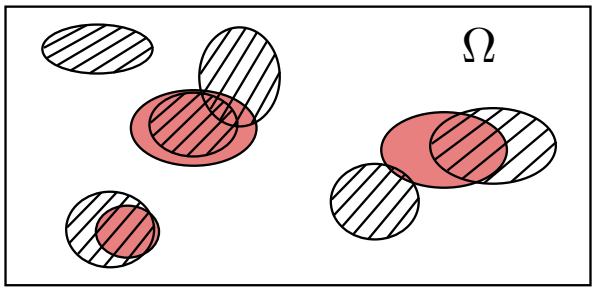
- Der Nachweis der unbekannten Fehler ist Zufall.
- Für alle Modellfehler, die sich Nachweisbedingungen mit einem Fehler teilen, impliziert jeder gefundene Test mit einer Wahrscheinlichkeit den Nachweis des Fehlers.



relative Größe der Schnittmenge als
Schätzwert für die Wahrscheinlichkeit,
dass ein Test aus M den Fehler nachweist

Die Nachweiswahrscheinlichkeit für tatsächliche Fehler ist um so größer,

- je größer der Anteil der Schnittmenge der Nachweismenge des Modellfehlers mit der des tatsächlichen Fehlers ist,
- je mehr Modellfehler sich Nachweisbedingungen mit dem tatsächlichen Fehler teilen und
- je mehr Tests für jeden Modellfehler gesucht werden.



Ω Menge der Eingabewerte /Teilfolgen die einen Fehler nachweisen können

Nachweismenge eines Modellfehlers

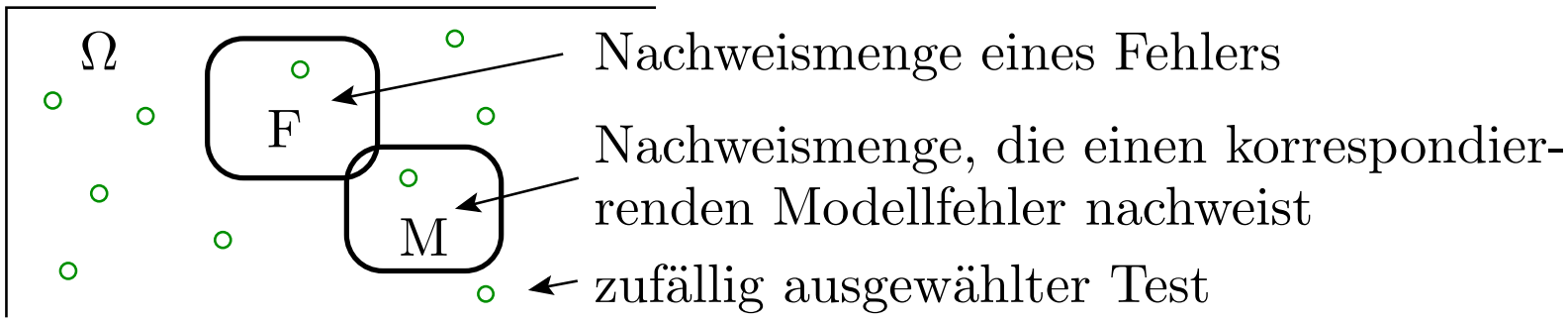
Nachweismenge eines tatsächlichen Fehlers

- Ein Fehlermodell erzeugt viele Modellfehler.
- Alle potenziellen Fehler sollten eine größere Nachweismengenüberscheidung mit mehreren Modellfehlern haben.
- Die Überschneidungen entstehen durch gleiche Anregungs- und Beobachtungsbedingungen.

⇒ Modellierung von Anregungs- und Beobachtungsbedingungen für alle Signal-, Kontroll- oder Datenflusspfade.

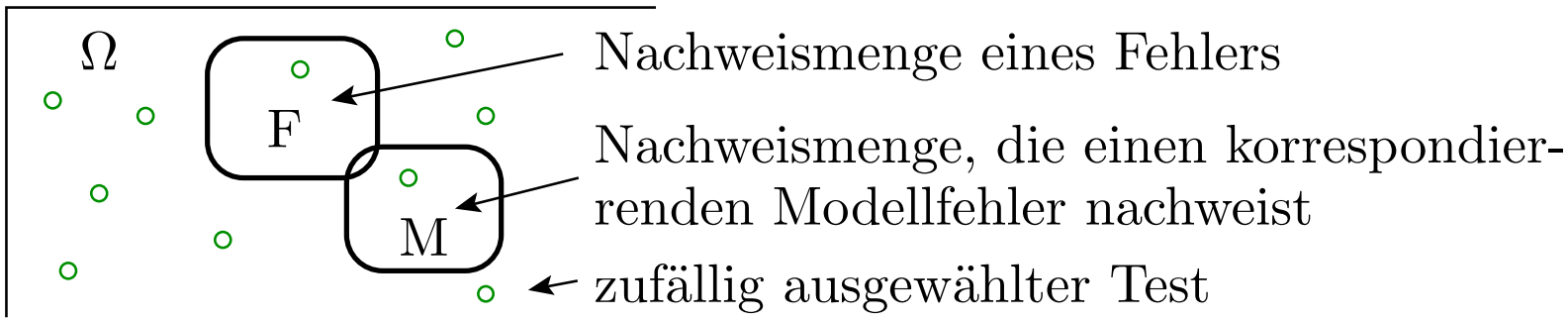
Zufällige Testauswahl

Zufällige, von den Nachweismengen der Modellfehler unabhängige Auswahl von Eingabewerten. Modellfehler dienen nur zur Bewertung.



Gedankenmodell:

- Füllen des Eingaberaums solange mit Punkten, bis zufällig die Menge M getroffen wird. Wenn F genauso groß ist, bekommt F mit derselben Wahrscheinlichkeit ein Treffer ab.
- Keine Überschneidungen, sondern nur ähnliche / umrechenbare Größen der Nachweismengen erforderlich.



Gedankenmodell 2:

- Fehlermodell zur Abschätzung der Fehlernachweisdichte.
- Über die Fehlernachweisdichte wird die Anzahl der nachweisbaren Fehler als Funktion der Testsatzlänge bzw. die erforderliche Testsatzlänge abgeschätzt.

Vorteile zufälliger gegenüber gezielter Auswahl:

- Geringere Anforderungen an das Fehlemodell.
- Genauere Vorhersage der tatsächlichen Fehlerüberdeckung.

Nachteil eines Zufallstests:

- Erfordert wesentlich längere Testsätze für dieselbe Fehlerüberdeckung.

Mischformen

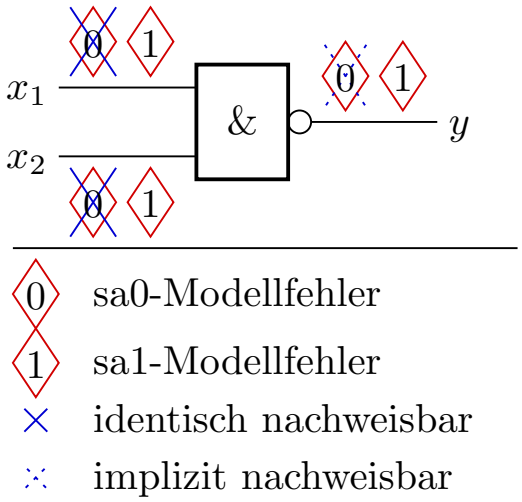
- Bevorzugung von Testeingaben, die auch Modellfehler nachweisen.
- Kombination eines Zufallstests, der die einfach nachweisbaren Modellfehler nachweist, mit gezielt ausgewählten Tests für die restlichen Modellfehler.
- Intuitive Auswahl basierend auf Erfahrungen über Fehler in anderen Objekten.
- ...

- Vergleich mit Zufallstestsätzen gleicher Modellfehlerüberdeckung: viel kürzer und geringere Fehlerüberdeckung.
- Vergleich mit geziel ausgewählten Testsätzen gleicher Modellfehlerüberdeckung: deutlich länger und höhere tatsächliche Fehlerüberdeckung.

Digitale Schaltungen

Haftfehlermodell

Das gebräuchlichste Fehlermodell ist das bereits auf Foliensatz F2 eingeführte Haftfehlermodell. Es unterstellt für jedes binäre Datensignal, dass es durch einen Fehler ständig auf null (stuck-at 0) oder ständig auf eins (stuck-at-1) gehalten wird.



x_2	x_1	$\overline{x_2 \wedge x_1}$	sa0(x_1)	sa1(x_1)	sa0(x_2)	sa1(x_2)	sa0(y)	sa1(y)
0	0	1	1	1	1	1	0	1
0	1	1	1	1	1	0	0	1
1	0	1	1	0	1	1	0	1
1	1	0	1	0	1	0	0	1

Nachweisidentität (gleiche Nachweismenge)

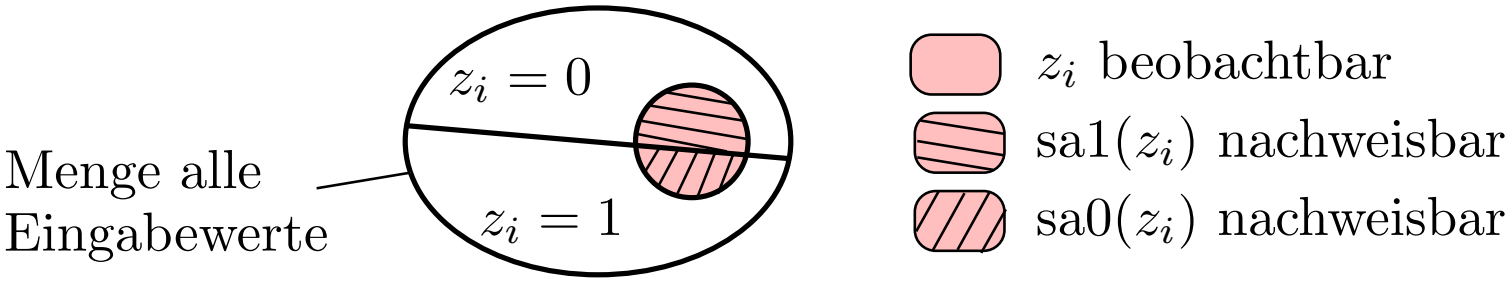
Nachweisimplikation

zugehörige Eingabe ist Element der Nachweismenge

Aus der so berechneten Anfangsmenge werden redundante Fehler, implizit nachweisbare und identisch nachweisbare Fehler gestrichen.

Toggle-Test

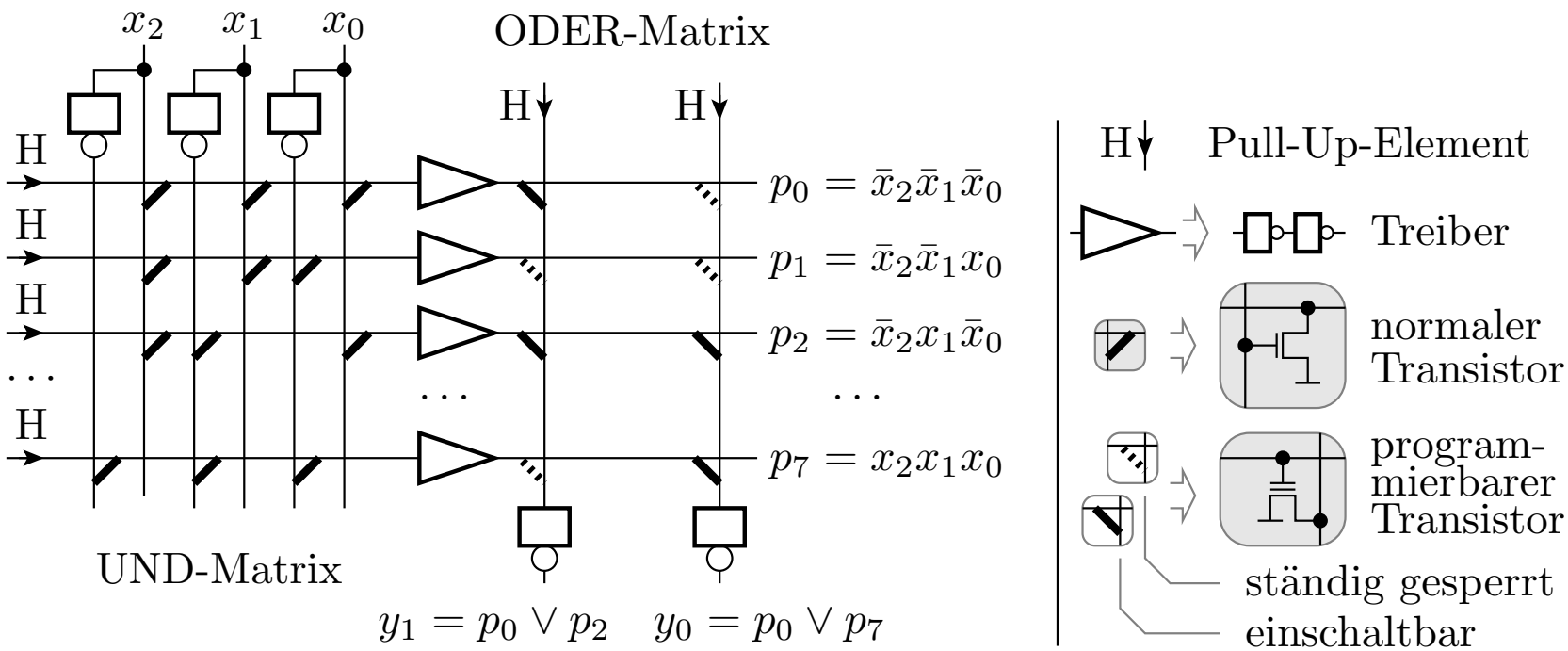
Ein Toogle Test kontrolliert, dass bei Abarbeitung des Testsatzes jedes logische Signal mindestens einmal »0« und einmal »1« ist.



- Garantiert Steuerbarkeit für alle Haftfehler. Gleichzeitige Beobachtbarkeit ist Zufall.
- Eine vergleichbare Haftfehlerüberdeckung verlangt einen wesentlich längeren Zufallstestsatz bzw. einen Testsatz, der für jede einzustellende null bzw. eins eine Vielzahl von Tests enthält (Anzahl ca. Kehrwert der Beobachtbarkeit[3]).

[3] Ähnlich wie bei Anweisungsüberdeckung, nur dass logische Zwischenergebnisse oft schlecht und Programmzwischenergebnisse gut beobachtbar sind.

Zellenfehlermodell (ROM, LUTs, ...)



Für jede Programmierstelle, Annahme falsch gesetzt oder für jedes Ausgabebit in der Wertetabelle ausgabe invertiert. Ein Testsatz mit 100% Zellenfehlerüberdeckung weist jede kombinatorische Funktionsabweichung und auch jeden Haftfehler nach.

Sollfunktion

$x_2\,x_1\,x_0$	$y_1\,y_0$
0 0 0	1 1
0 0 1	0 0
0 1 0	0 1
0 1 1	1 1

Fehler	verfälschtes Bit
1	y_0 für $\mathbf{x} = 000$
2	y_1 für $\mathbf{x} = 000$
3	y_0 für $\mathbf{x} = 001$
4	y_1 für $\mathbf{x} = 001$

Anzahl der Modellfehler:

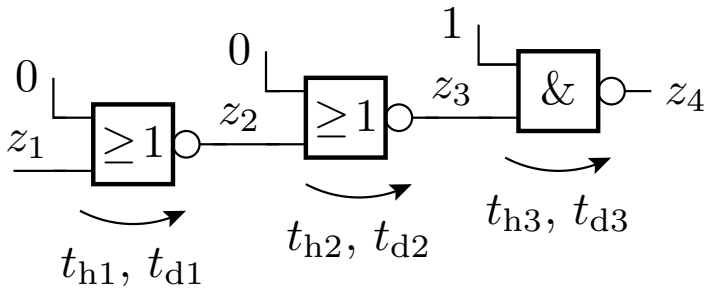
$$\varphi_{\mathrm{M}} = N_{\mathrm{A}} \cdot 2^{N_{\mathrm{E}}}$$

(N_{E} – Anzahl der Eingänge; N_{A} – Anzahl der Ausgänge).
Vergleich Zellenfehler und Haftfehler für freistrukturierte Schaltungen (Realisierung aus Gattern statt durch programmierte Speicher):

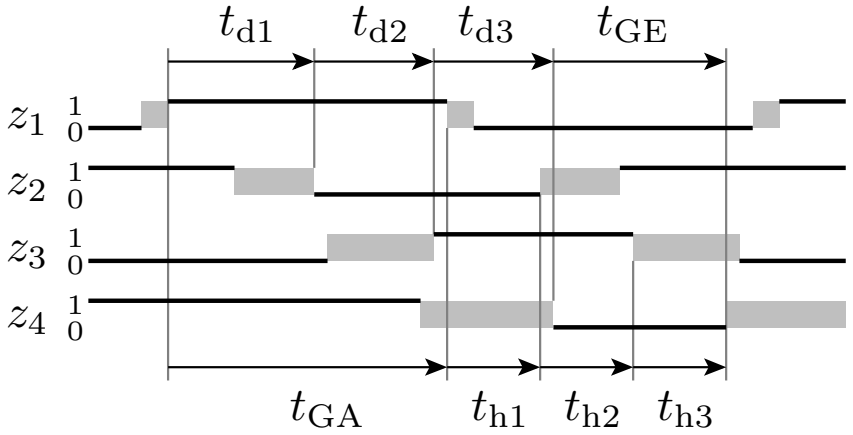
- u.U. wesentlich mehr Modellfehler,
- viele davon im Systemverbund redundant,
- Nachweis der Redundanz schwer zu erbringen.

Geeignet für LUTs, Volladdierer, ... Für Gatterschaltungen aus UND, ODER, ... ist das Haftfehlermodell besser geeignet.

Gatterverzögerungsfehler



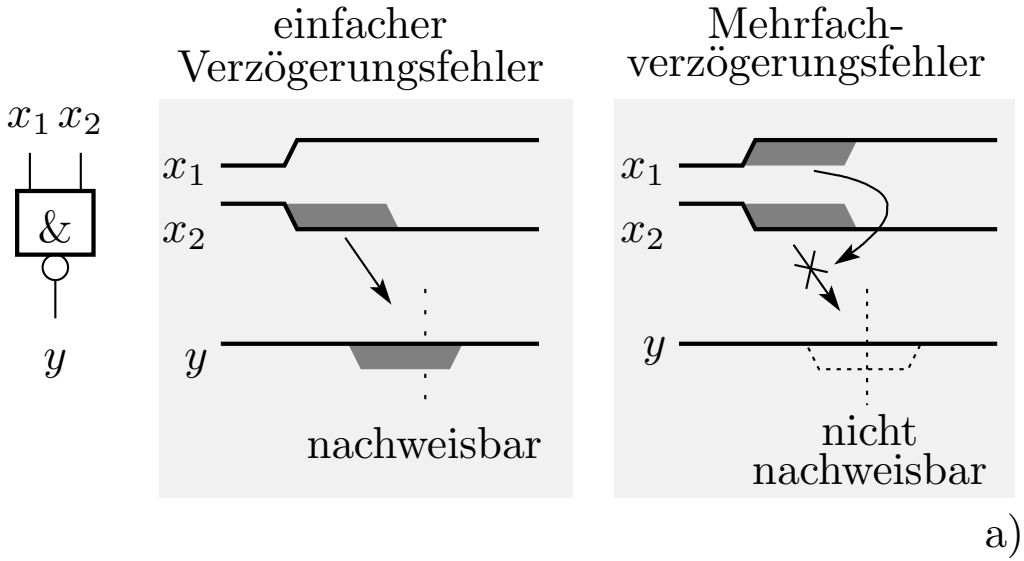
t_{GA}	Gültigkeitsdauer am Pfadanfang
t_{GE}	Gültigkeitsdauer am Pfadende



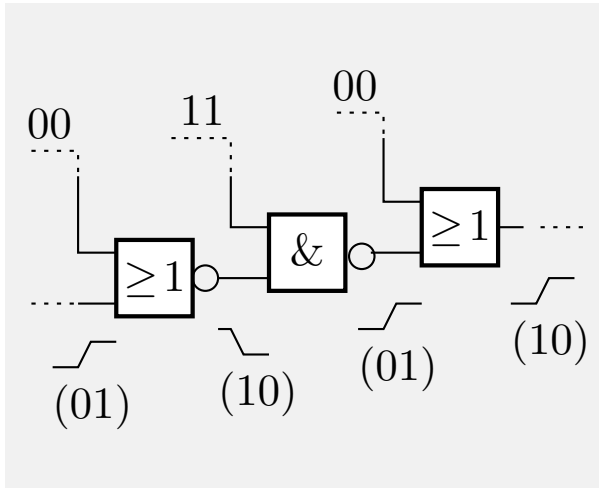
- Modellfehler: Zu kurze Haltezeit t_h, zu lange Verzögerungszeit t_d. Abtastung ungültiger Werte.
- 2-Pattern-Test: Eine Eingabe zur Initialisierung und eine zum Nachweis
- Haftfehlertest + Initialisierungseingabe
- Ein Testsatz mit hohe Haftfehlerüberdeckung besitzt in der Regel auch eine hohe Verzögerungsfehlerüberdeckung.

Pfadverzögerungsfehler

- Modellfehler: Für alle Pfade durch die Schaltung
  - slow-to-rise-Fehler (erhöhte 01-Verzögerung)
  - slow-to-fall-Fehler (erhöhte 10-Verzögerung)
- Robuster Test: Maskierungsausschluss für Mehrfachverzögerungsfehler



a)

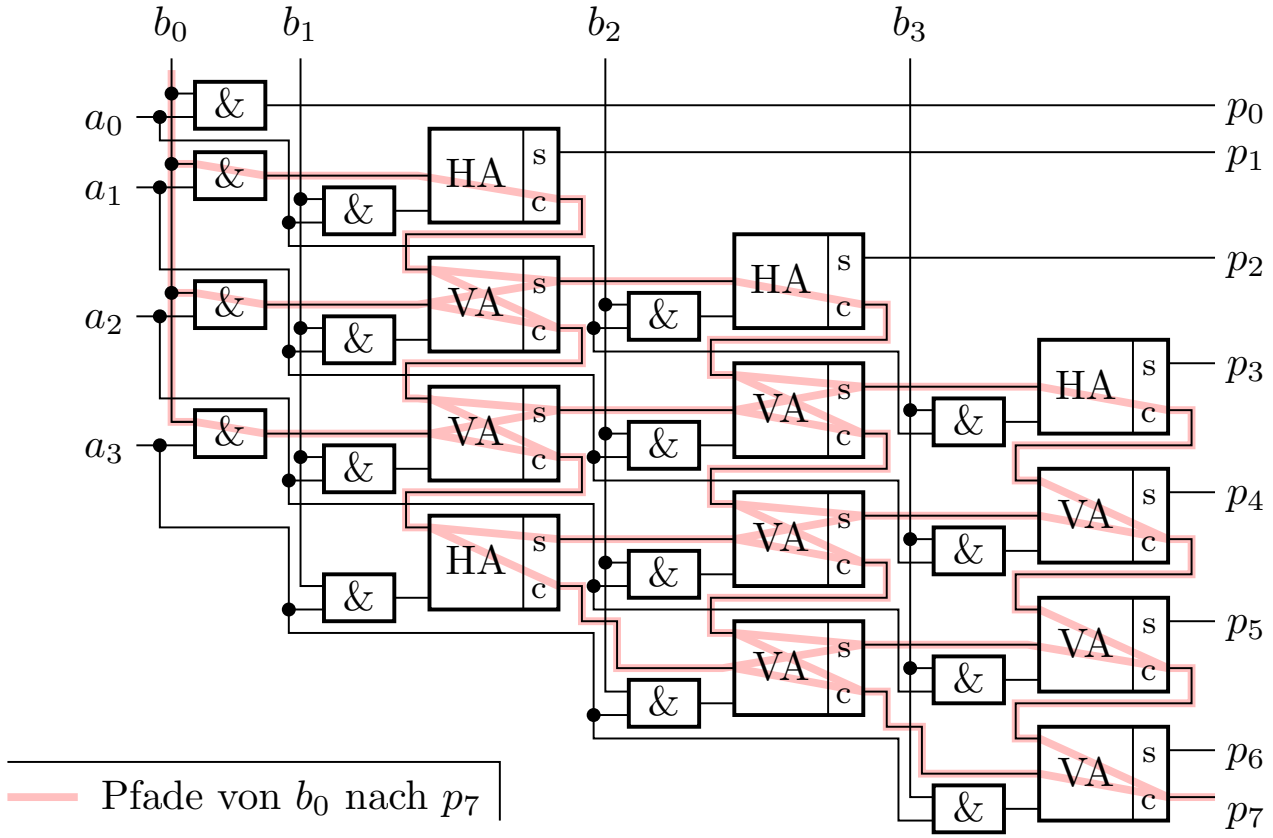


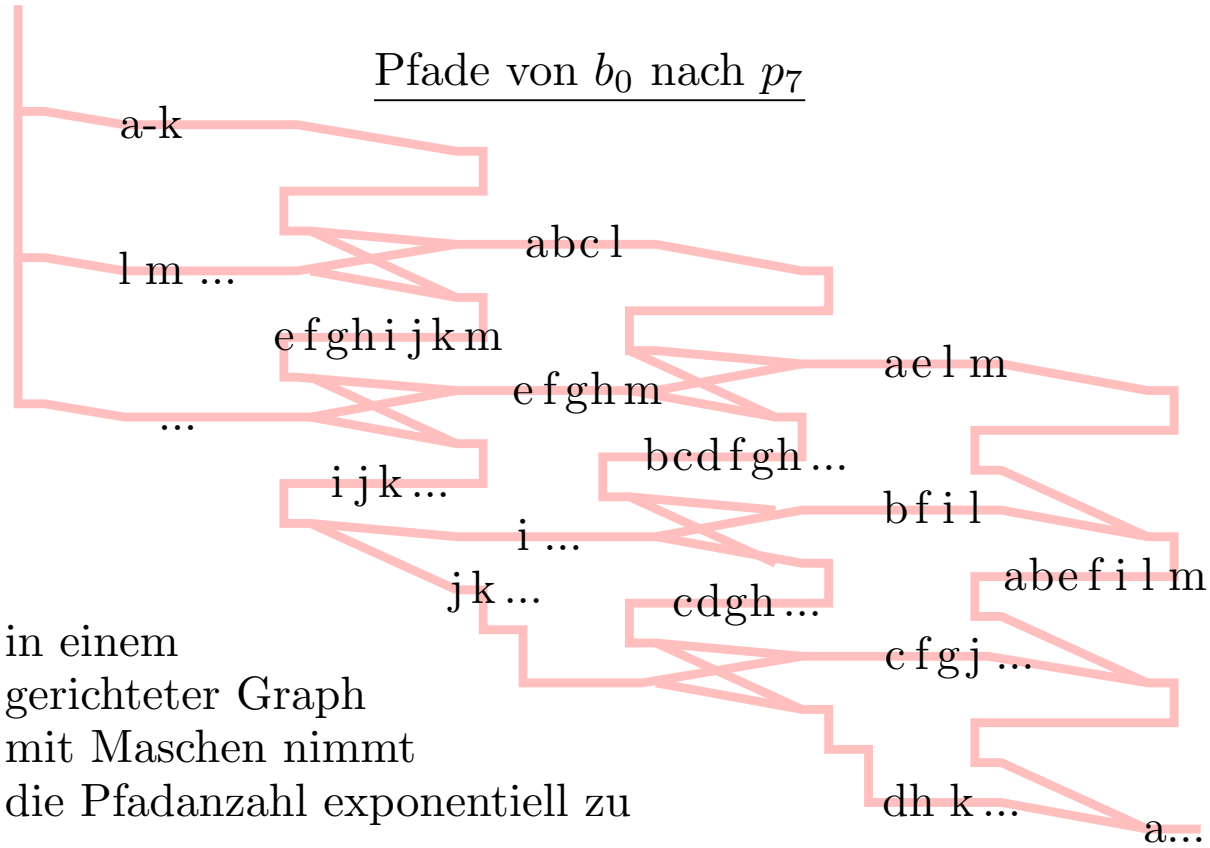
b)

- Problem Pfadanzahl; Beispiel Matrixmultiplizierer

$$(a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0) \cdot (b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0) =$$

$$\begin{array}{r}
a_3b_0 \cdot 2^3 + a_2b_0 \cdot 2^2 + a_1b_0 \cdot 2^1 + a_0b_0 \cdot 2^0 \\
a_3b_1 \cdot 2^4 + a_2b_1 \cdot 2^3 + a_1b_1 \cdot 2^2 + a_0b_1 \cdot 2^1 \\
a_3b_2 \cdot 2^5 + a_2b_2 \cdot 2^4 + a_1b_2 \cdot 2^3 + a_0b_2 \cdot 2^2 \\
a_3b_3 \cdot 2^6 + a_2b_3 \cdot 2^5 + a_1b_3 \cdot 2^4 + a_0b_3 \cdot 2^3 \\
\hline
p_7 \quad p_6 \quad p_5 \quad p_4 \quad p_3 \quad p_2 \quad p_1 \quad p_0
\end{array}$$

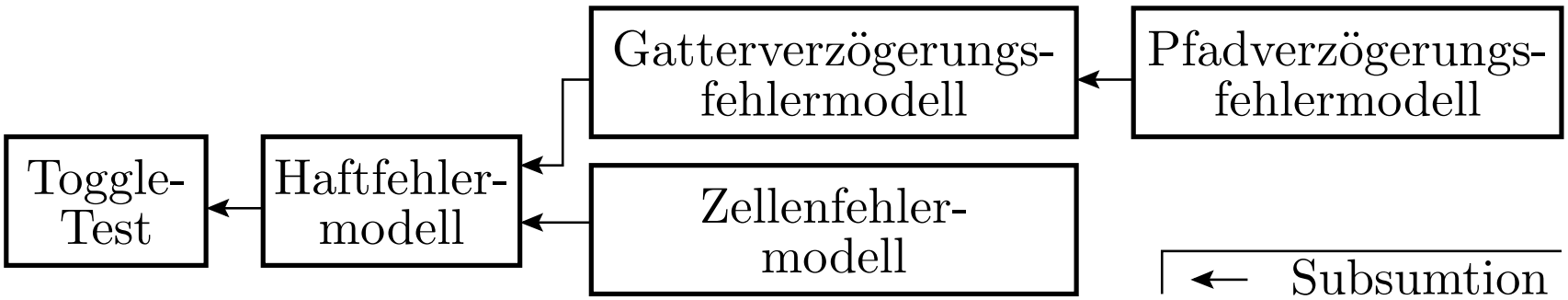




- Ausschluss verlängerter 01- und 10-Wechsels entlang aller Pfade schließt überhöhte Verzögerungen aus.
- In Schaltungen mit rekonvergenten Auffächerungen wächst die Pfadanzahl exponentiell mit der Schaltungsgröße.
- Exponenziell wachende Fehlermengen ⇒ nicht praxistauglich

Subsumtionshierarchie von Fehlermodellen

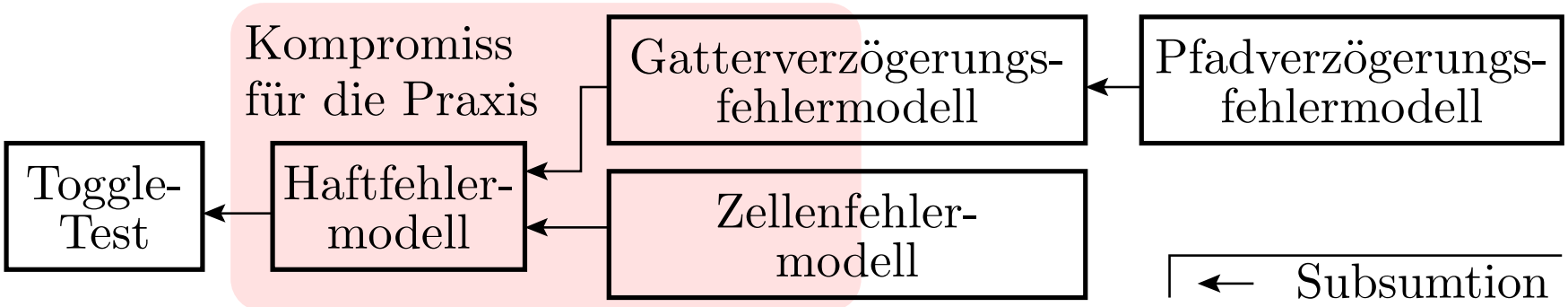
- Subsumption bedeutet in den Beschreibungslogiken, dass ein Konzept (eine eindeutig beschriebene Menge von Objekten) eine Teilmenge eines anderen Konzepts ist.
- Fehlermodell A subsumiert Fehlermodell B, wenn ein Testsatz, der alle Modellfehler von A nachweist, garantiert auch alle Modellfehler von B nachweist.



Bei 100% Modellfehlerüberdeckung haben auch die Modellfehlermengen aller subsumierten Fehlermodelle 100% Überdeckung.

Je höher ein Fehlermodell in der Subsumtionshierarchie steht:

- desto geringer ist tendentiell die Modellfehlerüberdeckung für denselben Testsatz,
- desto größer ist tendenziell die Fehlerüberdeckung bei gleicher Modellfehlerüberdeckung,
- desto größer ist der Aufwand für die Testsuche,
- desto länger muss ein Zufallstestsatz für eine angestrebte Fehlerüberdeckung sein.

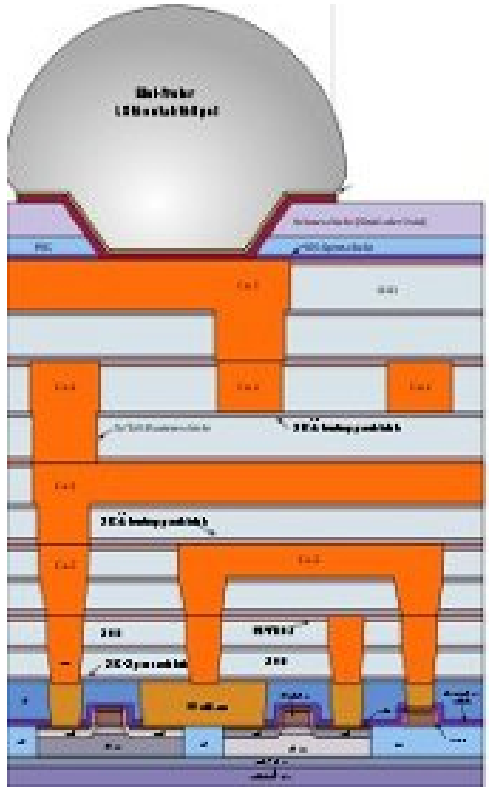


Fertigungsfehler ICs

Fertigungsfehler integrierter Schaltkreisen

Schaltkreise werden schichtenweise hergestellt:

- Auftragen von Schichten (z.B. Fotolack oder Metall),
- Belichten des Fotolacks durch eine Maske, die die Geometrie der zu erzeugenden Schichtelemente festlegt,
- Entfernen der belichteten (unbelichteten) Bereiche des Fotolacks,
- Fortätzen der freiliegenden Schichten neben dem Fotolack und entfernen des Fotolacks.

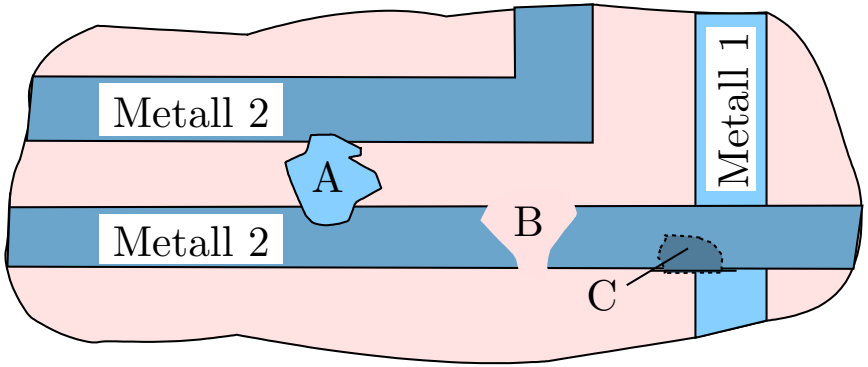


Typische Herstellungsfehler:

- fehlendes (zu wenig aufgetragenes zu viel weggeätztes) und
- überflüsiges (zu viel aufgetragenes, zu wenige weggeätztes)

Leitungs- oder Isolationsmaterial in einer Schicht.

Wirkung: Kurzschlüsse, Unterbrechungen, nicht richtig ein- oder ausschaltende oder zu langsam schaltende Transistoren.



A zuätzliches Metall
B fehlendes Metall
C fehlende Isolation

- Mehrfachfehler durch einzelne Fehlerfläche möglich.
- Einzelfehlerannahme genügt, weil ein Test für Einzelfehler auch die meisten Mehrfachfehler nachweist.

Notwendige Nachweisbedingungen:

- Anregung: Einstellung einer 0 oder 1 am Fehlerort.
- Beobachtung: Sensibilisierung eines Beobachtungspfads vom Fehlerort zu einem Ausgang.

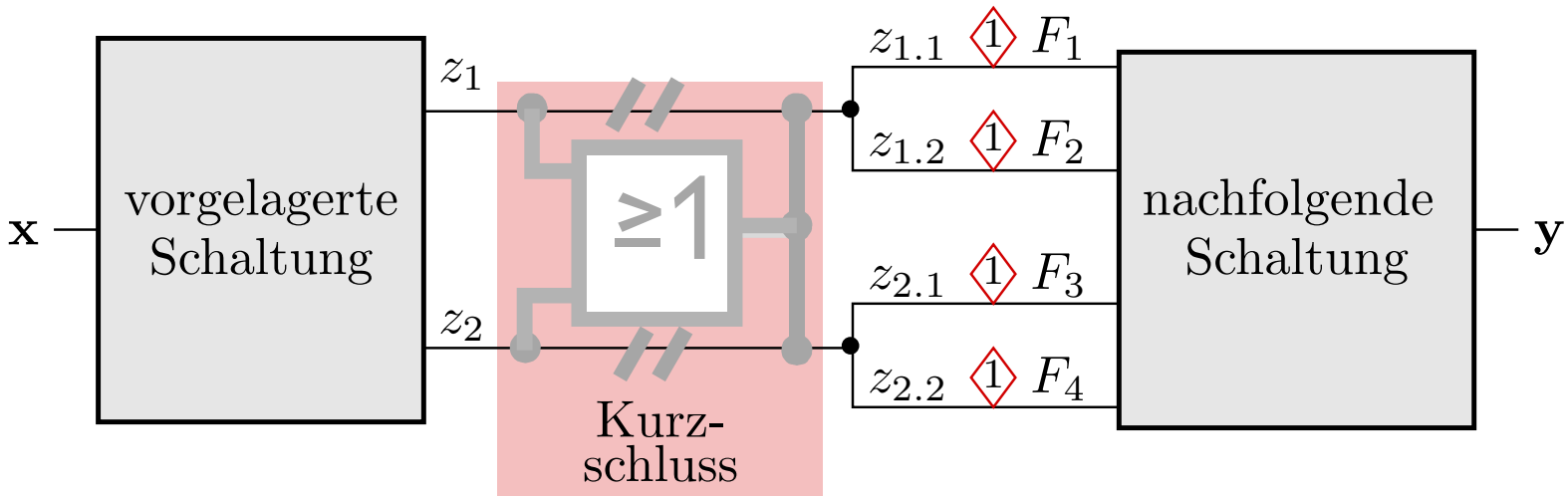
Alternative Beobachtungsmöglichkeiten:

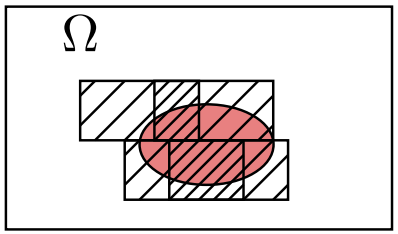
- erhöhte Verzögerung und Stromaufnahme während des Tests.

Kurzschlussnachweis mit einem Haftfehlertestsatz

- Zum Nachweis eines Kurzschlusses müssen auf den beteiligten Leitungen unterschiedliche Werte eingestellt und das dabei verfälschte Signal beobachtet werden.
- Die Menge der möglichen Kurzschlüsse in einer Schaltung ist viel größer als die Menge der Haftfehler. (Im ungünstigsten Fall quadratische Zunahme mit der Schaltungsgröße.)
- Die Fehlersimulation und die Testsatzberechnung sind auch etwas aufwändiger als für Haftfehler.
- Für Haftfehler ausgewählte Testsätze erkennen die meisten Kurzschlussmöglichkeiten, so das explizite Modellierung nicht zwingend ist.

Kurzschlussnachweis mit einem Haftfehlertestsatz



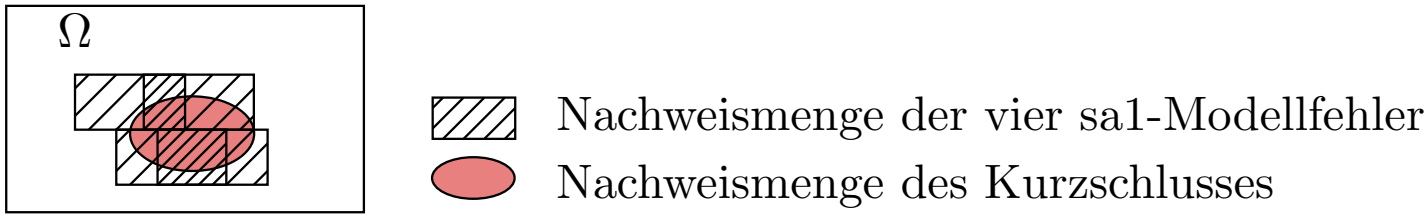


1 Modellfehler (ständig 1)

Nachweismenge der vier Modellfehler

Nachweismenge des Kurzschlusses

- Für jeden Haftfehler wird mindestens ein Test gesucht
- Wie wahrscheinlich ist der tatsächliche Fehlernachweis?



Der Kurzschluss ist nachweisbar

- $z_2 = 0$ und F_1 oder F_2 nachweisbar oder
- $z_1 = 0$ und F_3 oder F_4 nachweisbar ist.

Überschläge:

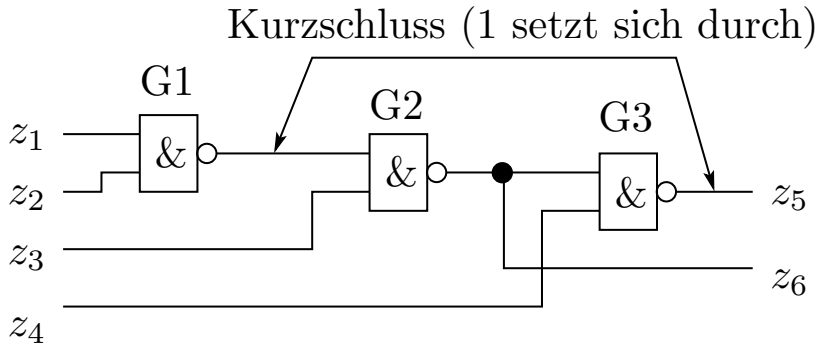
- Ein gezielt gesuchter Testsatz mit $FC = 1$ und $N \geq 4$ Versuchen mit Kurzschlussnachweiswahrscheinlichkeit von je 50%:

$$p_{\mathrm{E}} \geq 1 - 0{,}5^4 = 93{,}7\%$$

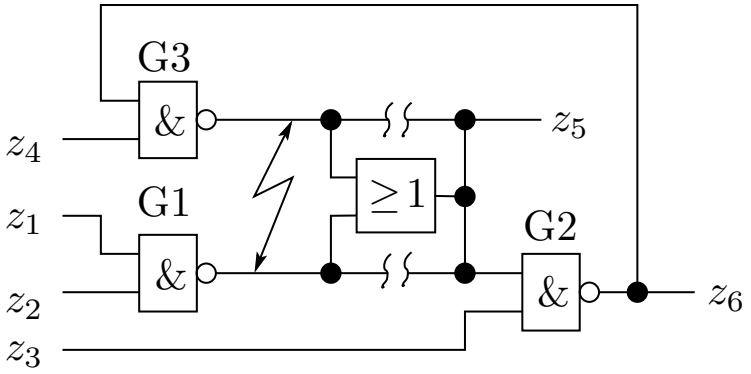
- Bei zufälliger Testauswahl ist die Nachweismenge des Kurzschlusses etwa der doppelte Mittelwert der Größe der Nachweismengen der vier Haftfehler.
- Grobabschätzung: Kurzschlussüberdeckung etwa Haftfehlerüberdeckung des halben Testsatzes.

Ein Kurzschluss kann auch ein zusätzliches Speicherverhalten verursachen.

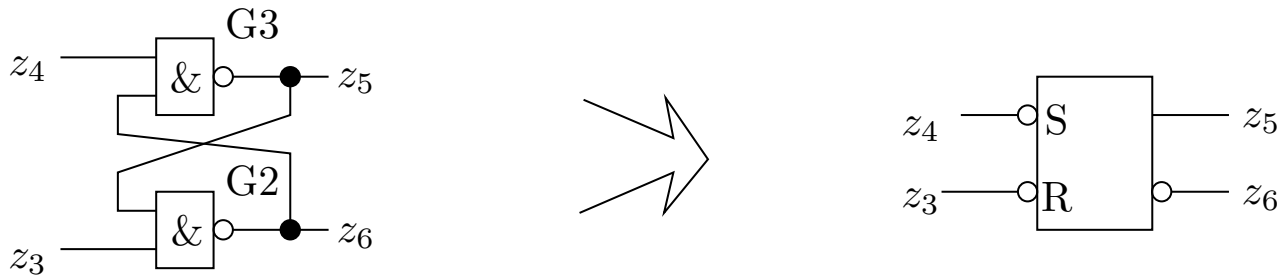
Schaltung mit Kurzschluss



Ersatz des Kurzschlusses durch ein ODER



Ersatzschaltung für $z_1 = z_2 = 1$

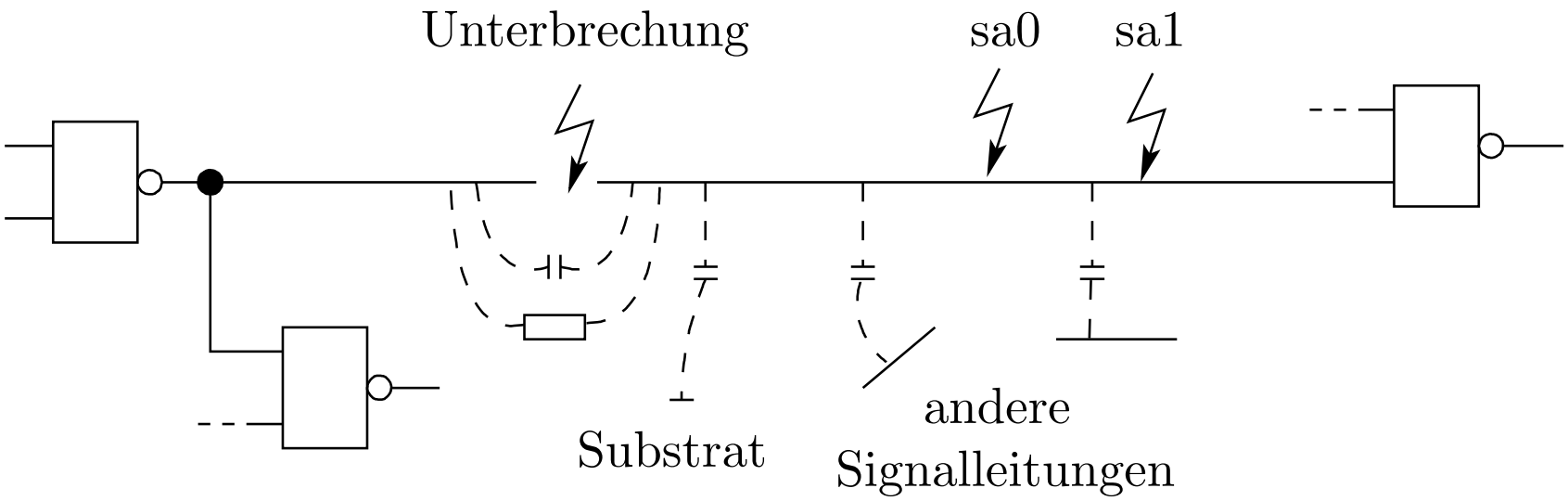


Für Fehler mit Speicherverhalten ist die Fehlersimulation und Testberechung deutlich aufwändiger/schwieriger als für Haftfehler.

Zusammenfassung, Trendabschätzungen

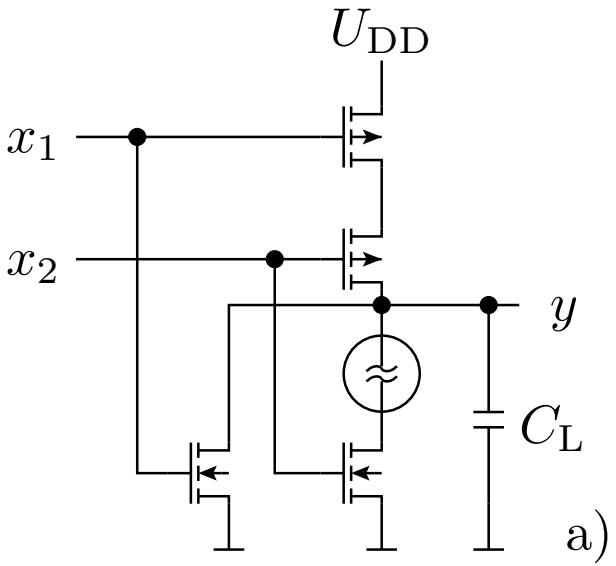
- Kurzschlüsse können sehr vielfältige Fehlerwirkungen haben.
- Berücksichtigung aller Möglichkeiten in der Regel weder notwendig, noch vom Aufwand her zu rechtfertigen.
- Gezielt berechnete Haftfehlertestsätze mit hoher Fehlerüberdeckung erkennen die meisten Kurzschlüsse.
- Die Kurzschlussüberdeckung hängt dabei jedoch weniger von der Haftfehlerüberdeckung, sondern mehr von der Anzahl der Tests, die je Haftfehler gesucht werden ab.
- Bei zufälliger Testauswahl ist die zu erwartende Kurzschlussüberdeckung etwa gleich der zu erwartenden Haftfehlerüberdeckung für einen kürzeren Testsatz.
- Dieser Zusammenhang erlaubt vertrauenswürdigere Vorhersagen, als der bei gezielter Testauswahl.

Fehlerwirkung von Unterbrechungen



- Die abgetrennten Gattereingänge können dauerhaft auf null oder eins liegen, driften oder den korrekten Wert erst nach erheblicher Verzögerung annehmen.
- Überwiegender Nachweis mit den Haftfehlertests für die nachfolgenden Gattereingänge.
- Sicherer Nachweis durch Kontrolle der Signalverzögerungen.

Stuck-open-Fehler



a)

x_2	x_1	y
0	0	1 (setzen)
0	1	0 (rücksetzen)
1	0	speichern
1	1	0 (rücksetzen)

b)

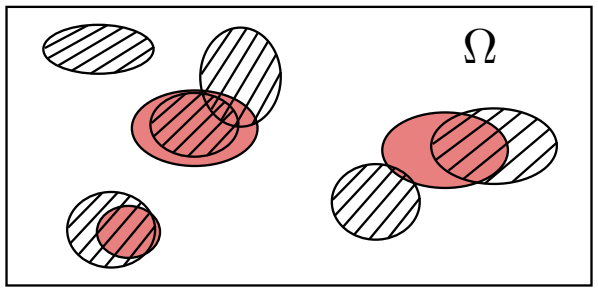
x_2	x_1	y
0	0	1
0	1	0
0	0	1
1	0	0

c)

Unterbrechung innerhalb eines Gatters, so dass die Gatterausgangskapazität für bestimmte Eingaben nicht auf- bzw. entladbar ist.

- Überwiegender Nachweis mit den Haftfehlertests.
- Sicherer Nachweis durch Kontrolle der Signalverzögerungen.

Reale Schaltkreisfehler und Haftfehler



Ω Menge der Eingabewerte /Teilfolgen die einen Fehler nachweisen können

Nachweismenge eines Modellfehlers

Nachweismenge eines tatsächlichen Fehlers

- Die möglichen Wirkungen von Schaltkreisfehlern sind wesentlich vielfältiger als gezeigt (null setzt sich bei Kurzschluss durch, ...)
- Fast jeder logisch nachweisbare Schaltkreisfehler teilt sich mit einem oder mehreren Haftfehlern den Beobachtungspfad und einen Teil der Anregungsbedingungen.
- Eine hohe Haftfehlerüberdeckung impliziert auch bei gezielter Testauswahl eine hohe Überdeckung für Fertigungsfehler.

Einschätzung der Güte von Schaltkreistests

Der typische Schaltkreistest hat eine Haft- oder Verzögerungsfehlerüberdeckung von 95% bis 100% und erkennt etwa 99,9% der Herstellungsfehler. Der Wert 99,9% ist keine publizierte Zahl, sondern über folgenden Überschlag mit typ. Werten abgeschätzt:

Von 10^6 gefertigten Schaltkreisen

- sind etwa 10% bis 50% fehlerhaft
- werden etwa 99,9% der defekten Schaltkreise von den Fertigungstests erkannt und aussortiert.
- Jeder 1000ste bis 10.000ste eingesetzte Schaltkreis hat einen kaum nachweisbaren Fehler.
- Jeder 10te Arbeitsplatzrechner enthält einen defekten Schaltkreis, erkennbar an abweichenden seltenen[4] Fehlfunktionen bei baugleichen Rechnern mit gleicher Software.

[4] Fehler, die oft Fehlfunktionen verursachen, erkennt der Schaltkreistest.

Fehlersimulation

Fehlersimulation

Wiederhole für jede Testeingabe
Berechne die Ausgabe des fehlerfreien Systems
Wiederhole für jeden Modellfehler
Berechne die Ausgabe des fehlerhaften Systems
Vergleiche die Ausgaben des fehlerfreien und des fehlerhaften Systems und protokolliere, welche Modellfehler nachweisbar sind

- Testauswahl zufällig oder manuell[5]
- Testeingaben, die keine weiteren Modellfehler nachweisen, werden entweder
  - beibehalten (fehlerunabhängige Auswahl, lange Testsätze)
  - oder gestichen (fehleranhängige Auswahl, kurze Testsätze).

[5] Auch eine bezüglich der unbekannten Fehler zufällige Auswahl, aber oft mit größeren Fehlernachweiswahrscheinlichkeiten.

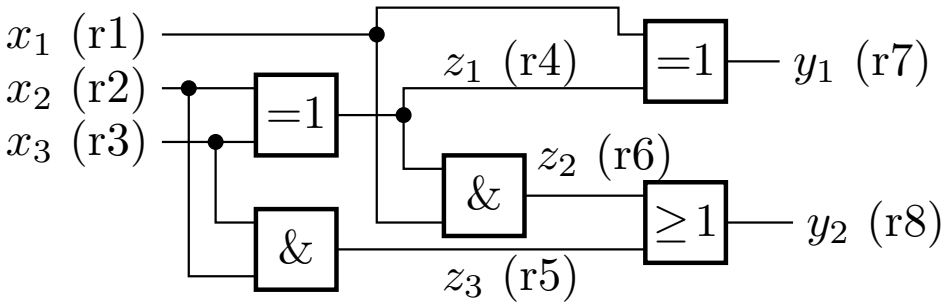
Aufwandsabschätzung:

- Schaltungsgröße: 10^4 Gatter
- Anzahl der Testschritte / Testeingaben: 10^4
- Anzahl der Modellfehler: 10^4
- Simulationsaufwand je Gatter: 10 ns

gesamter Rechenaufwand: 10^4 s, ca. 3 Stunden

Bitparalle Logiksimulation

Schaltung eines Volladdierers



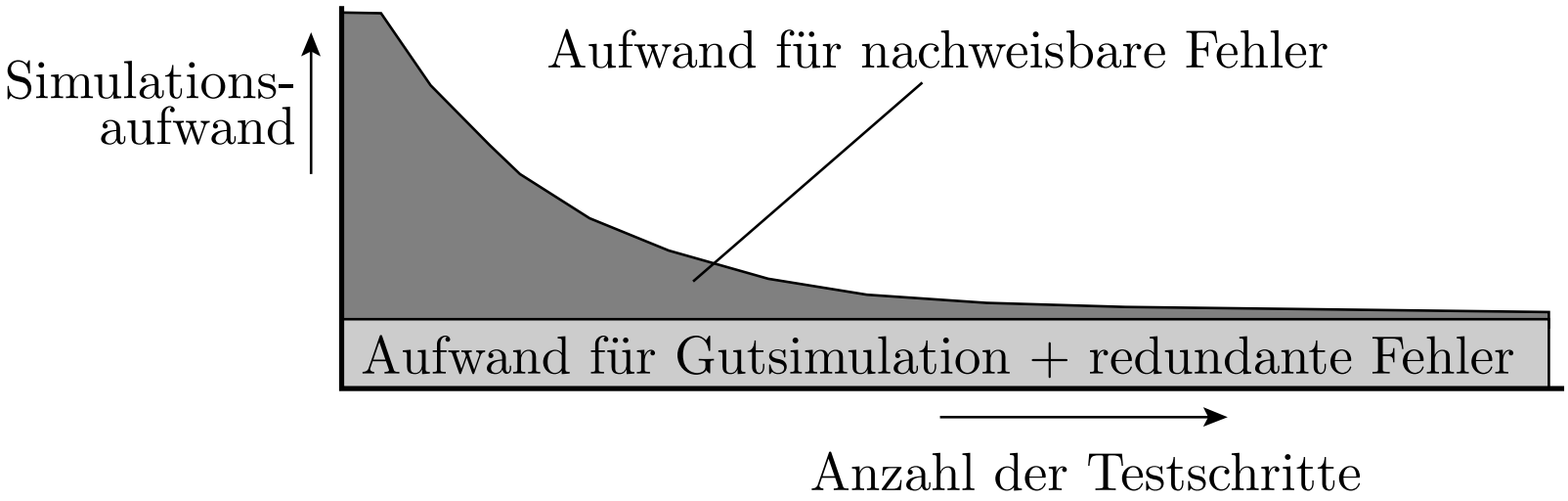
r1 bis r8 Prozessorregister

Programm für die Gutsimulation

lade x_1 in Register r1
lade x_2 in Register r2
lade x_3 in Register r3
r4 = r2 xor r3
speichere Inhalt r4 in z_1
r5 = r2 and r3
speichere Inhalt r5 in z_3
r6 = r1 and r4
speichere Inhalt r6 in z_2
r7 = r1 xor r4
speichere Inhalt r7 in y_1
r8 = r5 or r6
speichere Inhalt r8 in y_2

- In jedem Bit eine andere Eingabe simulieren; $\frac{1}{32}$ oder $\frac{1}{64}$ des Rechenaufwands. Für Beispiel zuvor nur 3 bzw. 6 Minuten.

Simulation bis zum ersten Fehlernachweis



- Nur noch ≈10% des Aufwands der Simulation aller Fehler mit allen Eingaben.
- Den meisten Rechenaufwand verursachen redundante (nicht nachweisbare) Fehler.
- Wichtiges Gütemerkmal von Fehlermodellen: geringer Anteil redundanter Fehler.

Entwurfsfehler

Entwurfsfehler

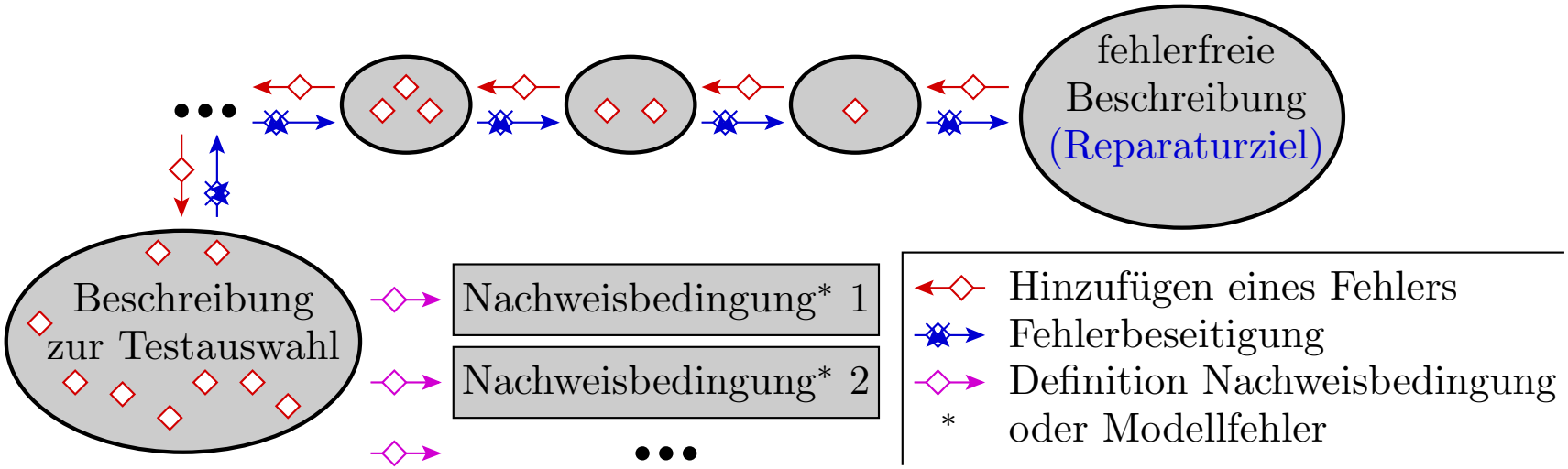
Hard- und Software werden heute ähnlich entworfen:

1. Zusammenstellung der Anforderungen,
2. Konzeption der Gesamtarchitektur und Aufspaltung in Teilentwufsaufgaben,
3. Beschreibung der Zielfunktion in einer Hochsprache,
4. Übersetzung (in ein Programm oder eine Schaltung).

In allen vier Phasen entstehen Beschreibungen und Fehler. Basis für die Testauswahl sind die Bescheibungen und Erfahrungen über die in bisherigen Entwürfen gefundenen Fehler.

- Positiv motivierte Testauswahl: Auswahl typischer Funktionsstichproben zur Rückversicherung, dass diese erfüllt werden. (Grobtest)
- Negativ motivierte Testauswahl: Suche nach Fehlern und Schwachstellen (gründlicher Test).

Mutationstest



- Für Entwürfe gibt es keine korrekte Sollbeschreibung.
- Statt der Modellfehler lassen sich nur Mutationen einer potenziell fehlerhaften Beschreibung konstruieren.
- Die Konstruktion ähnlich nachweisbarer Modellfehler setzt ein überwiegend funktionsfähiges System voraus.
- Für Spezifikationsfehler, vergessene Anforderung, ... lassen sich keine ähnlich nachweisbaren Mutationen ableiten.

Typische Mutationen auf der Hochsprachenebene:

- Verfälschung arithmetischer Ausdrücke (x=a+b ⇒ x=a*b)
- Verfälschung boolescher Ausdrücke (if(a>b){} ⇒ if(a<b){})
- Definition einer Variablen (value=5 ⇒ value=50)
- Referenz auf eine Variable (ref=obj1 ⇒ ref=obj2)
- Entfernen von Schlüsselworten (static int x=5 ⇒ int x=5)

(Abhängig von der Programmierprache. Gewinnbar aus der statistischen Analyse typischer Programmierfehler.)

Mutationstest ist vom Prinzip her auch auf die Ergebnisse der drei anderen Entwurfsphasen anwendbar:

- die zusammengestellten Anforderung,
- den Architekturentwurf und
- die übersetzte Beschreibung.

Gibt es dazu schon aussagekräftige Forschungsergebnisse?

Bei zufälliger Testauswahl genügt es, wenn die Mutationen vergleichbare Nachweiswahrscheinlichkeiten mit den unbekannten tatsächlichen Fehlern haben.

Geziele Auswahl setzt große Schnittmengen der Nachweismengen voraus.

Stand der Technik scheint noch zu sein, je Mutation nur einen Testfall zu suchen. Nach den Betrachtungen in dieser Vorlesung unzureichend.

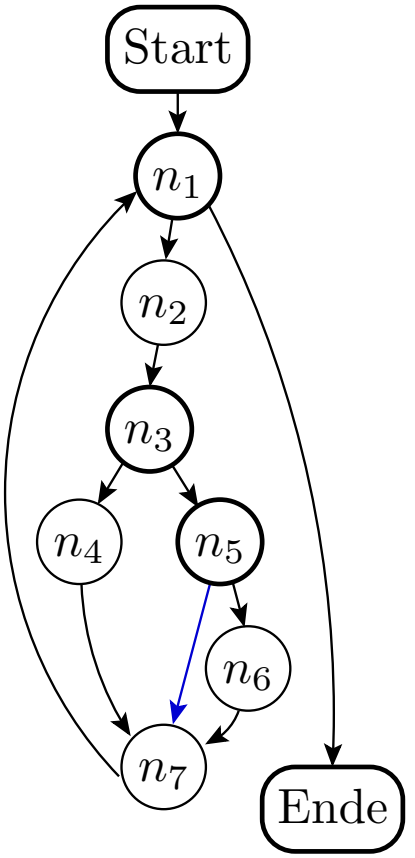
> Gibt es Forschungsergebnisse zur Beziehung zwischen Mutations- und tatsächlicher Fehlerüberdeckung?

Klassische Überdeckungsmaße

Die klassischen Überdeckungsmaße / -anforderungen sind:

- 100% Anweisungsüberdeckung,
- 100% Zweigüberdeckung und
- 100% Bedingungsüberdeckung.

Gibt es vergleichende Forschungsergebnisse zur Mutations- oder tatsächlichen Fehlerüberdeckung von Testsätzen, die diese Überdeckungsanforderungen erfüllen?

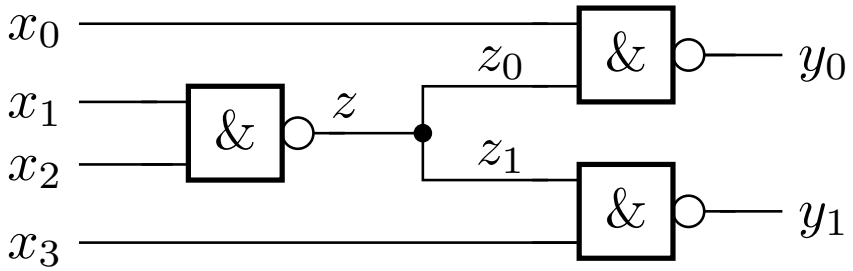


bei 100% Anweisungsüberdeckung möglicher weise ungetestete Kante

Aufgaben

Aufgabe 2.1: Haftfehlermenge

Stellen Sie für die nachfolgende Schaltung die Menge aller Haftfehler auf und streichen Sie aus jede Teilmenge identisch nachweisbarer Haftfehler alle außer einem.

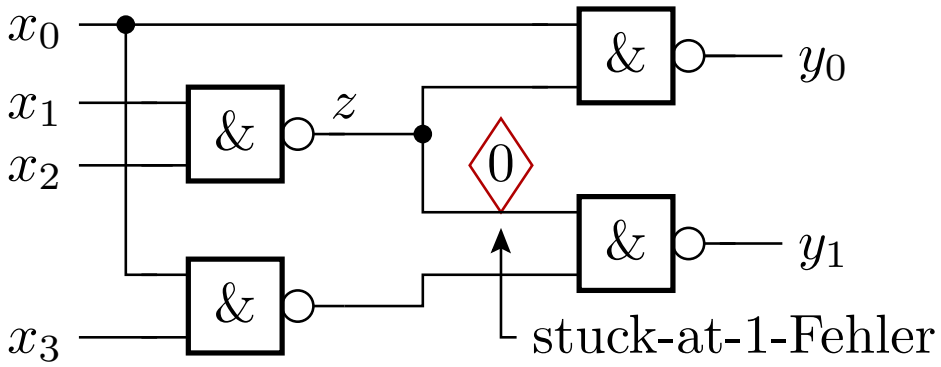


Hinweis: Notation der Haftfehler sa0(<Signalname>) bzw. sa1(<Signalname>).

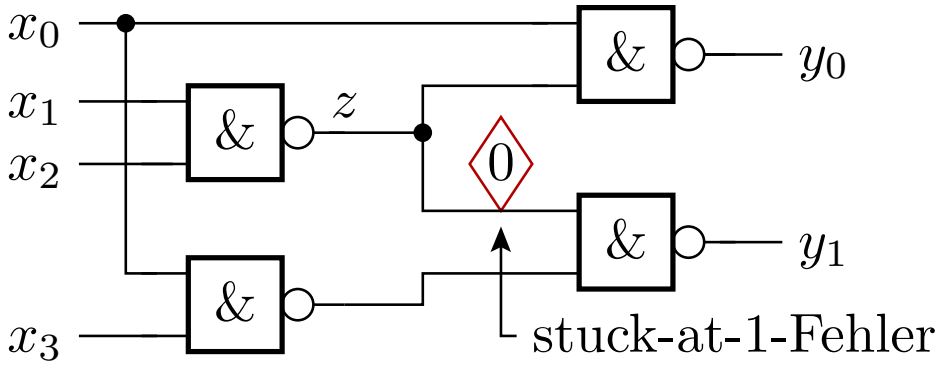
Aufgabe 2.2: Fehlernachweismengen

Bestimmen Sie für den in der nachfolgenden Schaltung eingezeichneten Haftfehler die Mengen von Eingaben

1. M_{A}mit denen der Fehler angeregt wird,
2. M_{B} mit denen der Fehler beobachtbar ist und
3. M_{N} mit denen der Fehler nachweisbar ist.



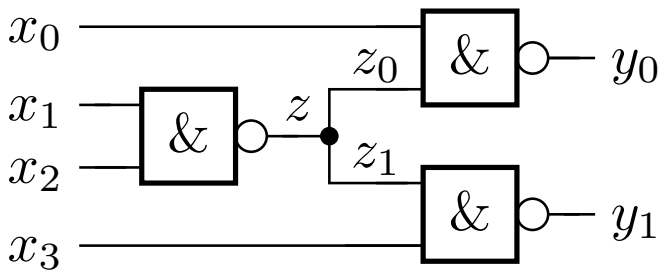
Hinweis: Notation der Eingabemengen als Kreuze in der Wertetabelle (siehe nächste Folie).



x_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
M_{A}																
M_{B}																
M_{N}																

Aufgabe 2.3: Toogle-Test

1. Kontrollieren Sie für die nachfolgende Schaltung und den angegebenen Testsatz, welche Signale noch nicht mindestens einmal den Wert null und einem den Wert eins annehmen.
2. Erweitern Sie den Testsatz um zusätzliche Eingaben so, dass eine 100%ige Toggle-Test-Überdeckung erzielt wird.

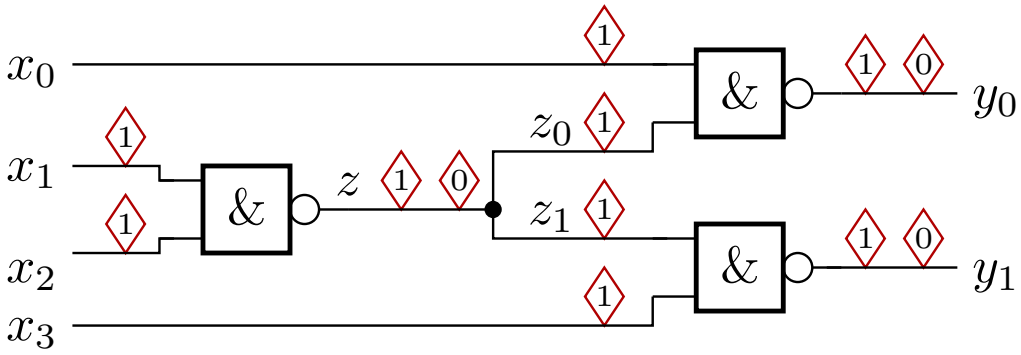


x_3	x_2	x_1	x_0	z	y_1	y_0
0	1	1	0			
1	1	0	0			
1	1	1	0			

Hinweis: Verwenden Sie als Hilfmittel die Wertetabelle rechts.

Aufgabe 2.4: Haftfehlerüberdeckung

1. Bestimmen Sie für jede Eingabe des Toggle-Test aus der Aufgabe zuvor, welche der eingezeichneten Haftfehler sich nachweist.
2. Wie groß ist die Haftfehlerüberdeckung des Toggle-Tests?



x_3	x_2	x_1	x_0	z	y_1	y_0	sa1(x_0)	sa1(x_1)	sa1(x_2)	sa1(x_3)	sa1(z)	sa0(z)	sa1(z_0)	sa1(z_1)	sa1(y_0)	sa0(y_0)	sa1(y_1)	sa0(y_1)
0	1	1	0															
1	1	0	0															
1	1	1	0															

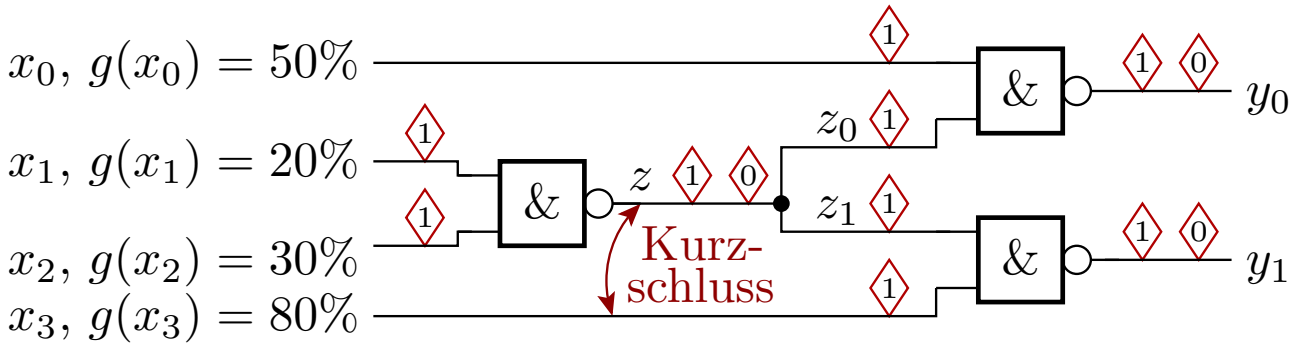
Aufgabe 2.5: Gatterverzögeungsfehler

1. Für welche der Haftfehler in der Aufgabe zu vor wird der korrespondierende Gatterverzögerungsfehler nachgewiesen.
2. Wie groß ist die Gatterverzögerungsfehlerüberdeckung des betrachtten Toggle-Tests?

Hinweis: Der korrespondierende Gatterverzögerungsfehler zu »stuck-at-0« ist »slow-to-rise« und zu »stuck-at-1« »slow-to-fall«. Zusätzliche Nachweisbedingung ist ein Signalwechsel am Fehlerort.

Aufgabe 2.6: Kurzschlussnachweis

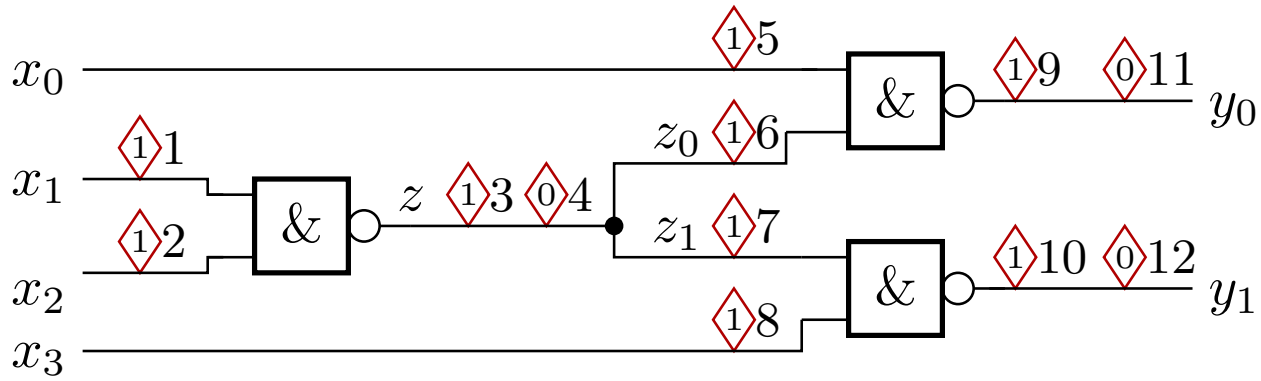
Bei dem eingezeichneten Kurzschluss soll sich null durchsetzen. Welche der eingezeichneten Haftfehler teilen sich Nachweisbedingungen mit dem Kurzschluss und wie groß ist für jeden dieser Haftfehler die Wahrscheinlichkeit, dass wenn ein Test ihn nachweist, auch der Kurzschluss nachgewiesen wird?



Hinweis: $g(...)$ – Signalwichtungen, Auftrittshäufigkeit einer Eins. Die bedingte Wahrscheinlichkeit, das ein Kurzschluss von einem Haftfehlertest nachgewiesen wird ist die Wichtung $g(...)$ der anderen beteiligten Leitung oder deren Gegenwahrscheinlichkeit.

Aufgabe 2.7: Fehlersimulation

Ergänzen Sie in der nachfolgenden Skizze eines C-Programms die Anweisungen für eine fehlerparallele Simulation der nachfolgenden Schaltung mit eingezeichneten Haftfehlern.




```
unit16_t x0, x1, x2, x3, z, z0, z1, y;
<wiederhole für jeden Testschritt>
 <lade Testeingaben in x0 bis x3, 0x0 oder 0xFF>
 <zu ergänzende Anweisungsfolge>
 <Protokollierung der erkannten Fehler>
```

Hinweise: siehe nächste Folie.

Zu verwenden sind die bitweise C-Operatoren ~ (Negation), & (UND) und | (ODER. Die Nummern hinter den Fehlern geben die Bitstelle an, mit der der Fehler zu simulieren ist. In Bit null soll die Gut-Simulation erfolgen.

Zufallstest

Erforderliche Testsatzlänge

Festlegung der erforderlichen Testsatzlänge

Die Auswahl des Test erfolgt zufällig. Nur die Testsatzlänge ist zu bestimmen. Möglichkeiten:

1. direkte Vorgabe (z.B. Speichergröße Testautomat),
2. erfülltes Überdeckungsziel (z.B. 100% Zweigüberdeckung),
3. geforderte Modell-, z.B. Haftfehlerüberdeckung,
4. geforderte tatsächliche Fehlerüberdeckung und
5. geforderte Zuverlässigkeit[6] .

Für 2 bis 5 werden solange zufällige ausgewählte Testbeispiele hinzugenommen, bis das Kriterium erfüllt ist. Für Entwürfe ist 2 (Abhaken von Überdeckungsriterien) gebräuchlich, für digitale Schaltungen auch 3 (Fehlersimulation). 4 und 5 wären wünschenswert, sind aber noch nicht Stand der Technik.

[6] Zuverlässigkeit sei hier definiert als mittlere Zeit zwischen Fehlfunktionen im Einsatz.

Geforderte tatsächliche Fehlerüberdeckung

Falls es einmal möglich sein wird, Vorhersagen über die Fehlernachweisdichte eines zu testenden Systems zu treffen (z.B. über die Simulation einer Stichprobe von Modellfehlern oder Mutationen), gilt nach Foliensatz F2, Abschn. 3.2)

$$E(\varphi, n) = E(\varphi_{\mathrm{E}}) \cdot \int_0^1 h(p) \cdot e^{-n \cdot p} \cdot dp$$

($E(\varphi_{\mathrm{E}})$ – zu erwartende Anzahl der Entwurf und Fertigung entstandenen Fehler; $h(p)$ – Fehlernachweisdichte; n – Testsatzlänge). Zu erwartende Fehlerüberdeckung:

$$E(FC(n)) = 1 - \frac{E(\varphi, n)}{E(\varphi_{\mathrm{E}})} = \int_0^1 h(p) \cdot e^{-n \cdot p} \cdot dp$$

Auflösung nach $n(E(FC), h(p))$, notfalls numerisch, liefert für eine geforderte Fehlerüberdeckung und eine angenommene Fehlernachweisdichte die notwendige Testsatzlänge n.

Mit einer Potenzfunktion als Fehlernachweisdichte:

$$h(p) = k \cdot p^{k-1}$$

verringert sich die zu erwartende Fehleranzahl umgekehrt proportional zur k-ten Potenz $(0 < k < 1)$:

$$E(\varphi, n) \approx E(\varphi, n_0) \cdot \left(\frac{n}{n_0}\right)^{-k}$$

Eine Verringerung der zu erwartenden Fehleranzahl um eine Zehnerpotenz verlangt

- bei $k = 1$ die 10-fache,
- bei $k = 0{,}5$ die 100-fache und
- bei $k = 0{,}333$ die 1000-fache Testsatzlänge.

Für die zu erwartende Fehlerüberdeckung beträgt

$$E(FC(n)) = 1 - \frac{E(\varphi, n)}{E(\varphi_E)} = 1 - \left(\frac{n}{n_0}\right)^{-k}$$

Unter Kenntnis von k und n_0 ergibt sich für die erforderliche Testsatzlänge:

$$n \geq n_0 \cdot (1 - E(FC(n)))^{-\frac{1}{k}}$$

Problem: für eine geforderte hohe Fehlerüberdeckung und ein pessimistisch abgeschätztes k ergibt sich eine unrealistisch lange Testzeit von vielen Jahren, die sich der Hersteller nicht leisten kann

Ausweg Zuverlässigkeitswachstumsprozess:

- Fortsetzung des Tests beim Anwender mit Eingaben aus der Anwendungsumgebung als Zufallswerte.
- Erfassung der beobachteten Fehlfunktionen.
- Konstruktion von Testfällen für ihren Nachweis.
- Suche und Beseitigung der zugrunde liegenden Fehler.

Zuverlässigkeitswachstum

Zuverlässigkeitswachstumsprozess

- Zufallstest, beim dem sich die Nutzungsdauern bei allen Anwendern, bei denen die bobachteten Fehlfunktionen erfasst werden, als Testdauer akkumulieren.
- Erlaubt um Zehnerpotenzen längere Testdauern.
- Erfordert Kontrollfunktionen im System und organisatorische Maßnahmen, die die aufgetretenden Fehlfunktionen erfassen und an den Hersteller zur Erstellung von Tests für ihren Nachweis weiterleiten.
- Die Fehlerbeseitigung ist unsicherer, als wenn der Hersteller selbst testet. Beseitigungswahrscheinlichkeit für nachweisbare Fehler $p_{\mathrm{B}} \ll 1$ (z.B. 10%).

Die Zuverlässigkeit in Service-Anforderungen je Fehlfunktion

$$Z = N_{\xi} / E\left(\xi_{\mathrm{F}}\right)$$

nimmt trotzdem zu.

Die zu erwartende Anzahl der Fehlfunktionen je N_ξ Servic-Anforderung ist die Summe der Auftrittshäufigkeiten mal der Nachweiswahrscheinlichkeit je Service-Anforderung aller potenzieller Fehler N_φ:

$$\frac{E\left(\xi_\mathrm{F}\right)}{N_\xi} = \sum_{i=1}^{N_\varphi} h_i \cdot p_i$$

und beträgt ausgedrückt durch die Fehlernachweisdichte

$$\frac{E\left(\xi_\mathrm{F}\right)}{N_\xi}\left(\xi_\mathrm{F}\right) = E\left(\varphi\right) \cdot \int_0^1 h\left(p\right) \cdot p \cdot dp \tag{1}$$

Die Wahrscheinlichkeit, dass ein Fehler nach einem Service-Aufruf beseitigt wird, ist die Wahrscheinlichkeit, dass er nachgewiesen, und wenn nachgewiesen, auch beseitigt wird:

$$p \cdot p_\mathrm{B}$$

Die Beseitigungswahrscheinlichkeit bei Abarbeitung von n Service-Leistungen ist:

$$1 - e^{-n \cdot p \cdot p_\mathrm{B}}$$

Änderung der Fehlernachweisdichte mit n:

$$h(p,n) = \frac{h(p) \cdot e^{-n \cdot p \cdot p_{\mathrm{B}}}}{\int_0^1 h(p) \cdot e^{-n \cdot p \cdot p_{\mathrm{B}}} \cdot dp}$$

($h(p)$ – Fehlernachweisdichte des ungetesteten Systems). Gegenüber F2, Abschn. 3.3 ist, wenn nur ein Anteil p_{B} der erkennbaren Fehler beseitigt wird, n durch $n \cdot p_{\mathrm{B}}$ zu ersetzen:

$$E(\xi_{\mathrm{F}}) = N_\xi \cdot E(\varphi_{\mathrm{E}}) \cdot \int_0^1 h(p) \cdot p \cdot e^{-n \cdot p_{\mathrm{B}} \cdot p} \cdot dp$$

Mit der Potenzfunktion

$$h(p) = k \cdot p^{k-1}$$

ergibt sich für die zu erwartende Anzahl der durch Fehler verursachten Fehlfunktionen:

$$\begin{aligned} E(\xi_{\mathrm{F}}) &= N_\xi \cdot E(\varphi_{\mathrm{E}}) \cdot \int_0^1 k \cdot p^{k-1} \cdot p \cdot e^{-n \cdot p_{\mathrm{B}} \cdot p} \cdot dp \\ &= N_\xi \cdot E(\varphi_{\mathrm{E}}) \cdot \int_0^1 k \cdot p^{k} \cdot e^{-n \cdot p_{\mathrm{B}} \cdot p} \cdot dp \end{aligned}$$

Substitution $p = \frac{x}{p_\mathrm{B} \cdot n}$ und $dp = \frac{dx}{p_\mathrm{B} \cdot n}$

$$E\left(\xi_\mathrm{F}\right) = \frac{k \cdot N_\xi \cdot E\left(\varphi_\mathrm{E}\right)}{\left(p_\mathrm{B} \cdot n\right)^{k+1}} \cdot \underbrace{\int_0^n x^k \cdot e^{-x} \cdot dx}_{\approx \Gamma(k+1) \approx 1 \text{ für } 0 < k \leq 1}$$

Die Zuverlässigkeit in Service-Leistungen pro Fehlfunktion wächst überproportional mit dem Produkt $p_\mathrm{B} \cdot n$:

$$Z\left(n\right) = \frac{N_\xi}{E\left(\xi_\mathrm{F}\right)} = \frac{\left(p_\mathrm{B} \cdot n\right)^{k+1}}{k \cdot N_\xi \cdot E\left(\varphi_\mathrm{E}\right)}$$

Mit der Zuverlässigkeit einer Bezugsanzahl von Service-Aufrufen n_0:

$$Z\left(n\right) = Z\left(n_0\right) \cdot \frac{\left(p_\mathrm{B} \cdot n\right)^{k+1}}{\left(p_\mathrm{B} \cdot n_0\right)^{k+1}} = Z\left(n_0\right) \cdot \left(\frac{n}{n_0}\right)^{k+1}$$

Die Zuverlässigkeit nimmt überproportional mit der Anzahl der Service-Aufrufe, bei denen erkannte Fehlfunktionen Fehlerbeseitigungsversuche auslösen, zu.

Nutzungsdauer und Zuverlässigkeit

Unter der Annahme, dass sich die Anzahl der Service-Aufrufe n proportional zur Reifezeit[7] t verhält:

$$Z(t) = Z(t_0) \cdot \left(\frac{t}{t_0}\right)^{k+1}$$

t_0 und k – Modellparameter, vorzugsweise abzuschätzen aus der zu beobachtenden Zunahme der Zeit zwischen den beobachteten Fehlfunktionen.

[7] Große IT-Systeme müssen nach ihrer Entstehung eine Weile reifen, bevor sie ausreichend zuverlässig für den Einsatz sind. Eine Strategie hierfür ist die kostenlose Freigabe als Beta-Software.

Abschätzung der noch erforderlichen Reifezeit

Die Zuverlässigkeit eines Systems sei nach eine Reifedauer von 1 Jahr 10 Stunden pro Fehlfunktion. Wie lange muss das System noch reifen, um die Zuverlässigkeit auf 50 Stunden je Fehlfunktion zu erhöhen? Schätzwert für $k \approx 0{,}5$.
Lösung:

$$t = 1\,\text{Jahr} \cdot \left(\frac{50\,\text{h}}{10\,\text{h}}\right)^{\frac{1}{1,5}} = 2,92\,\text{Jahre}$$

Das System müsste noch zwei Jahre weiter reifen.

Überwachung eines Reifeprozesses

Gegeben: Für $i = 1$ bis 4 Versionen eines Softwareprodukts Erscheinungsdatum, die Zeitdifferenz t_i zur Herausgabe der ersten Version und die im Mittel pro Woche gezählten Fehlfunktionen ζ_i für zehn eingesetzte Systeme:

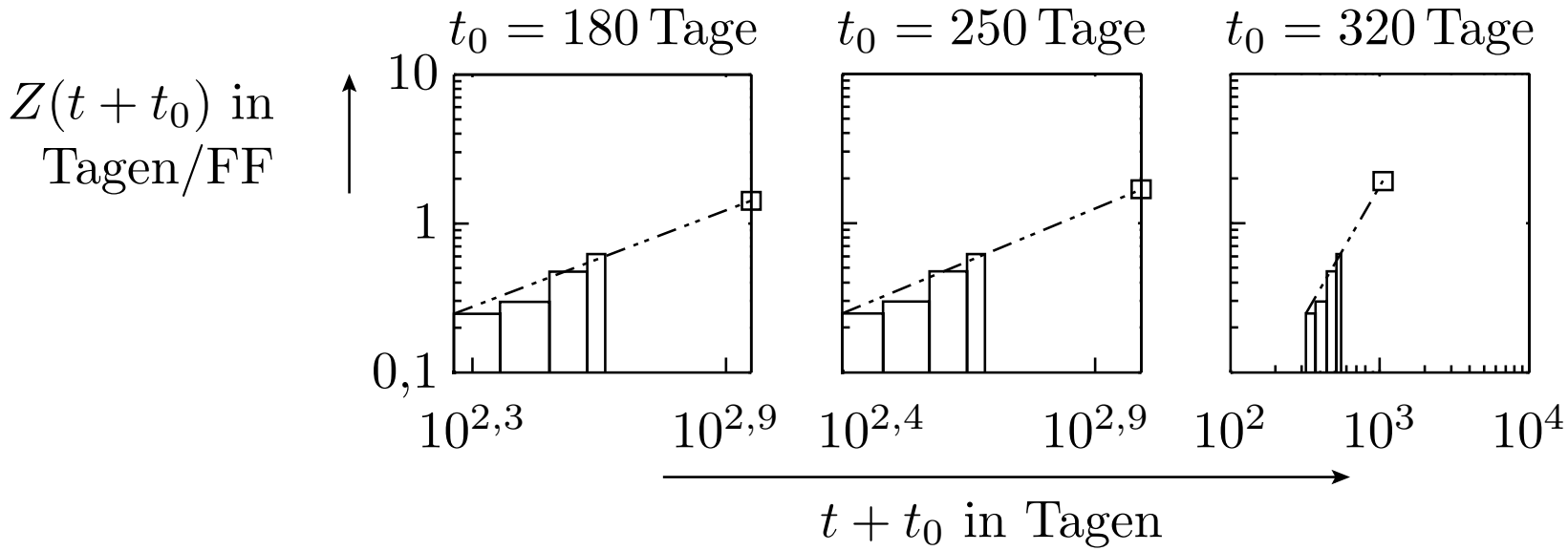
i	1	2	3	4	Ende
Datum	01.07.11	22.08.11	01.11.11	10.01.12	30.01.12
ζ_i	290,5	238,4	147,0	32,3	

Gesucht: t_0 und k zur Modellierung des Reifeprozesses.

Umrechnung die Daten in Reifezeiten und Zuverlässigkeiten:

i	1	2	3	4
t_i in Tagen	t_0	$t_0 + 52$	$t_0 + 123$	$t_0 + 193$
$Z(t_i)$ in Tagen	0,248	0,298	0,476	0,619

Vorgabe von t_0 und Abschätzung von k aus dem Anstieg in der doppellogarithmischen Darstellung der Zuverlässigkeit als Funktion der Testdauer:



Approx.	t_0	k	$Z\,(t_0 + 2\,\text{Jahre})$
1	180 Tage	0,08	1,42 Tage
2	250 Tage	0,68	1,68 Tage
3	320 Tage	0,92	1,92 Tage

Mit den Schätzwerten kann die Zuverlässigkeit zu einem späteren Zeitpunkt geschätzt werden, im Bsp. für eine Version, die 2 Jahre nach der ersten erscheint. Erstaunlicherweise sind die Ergebnisse für alle drei Paramtersätze sehr ähnlich.

Approx.	t_0	k	$Z\,(t_0 + 2\,\text{Jahre})$
1	180 Tage	0,08	1,42 Tage
2	250 Tage	0,68	1,68 Tage
3	320 Tage	0,92	1,92 Tage

Fakt 1

Es ist möglich, mit Modellrechungen auf ein Zuverlässigkeitswachstum zu schließen. Das erfordert Annahmen über die Nachweiseigenschaften der nicht gefundenen Fehler. Der Ansatz, eine Potenzfunktion als Fehlernachweisdichte zu nehmen, erscheint vielversprechend.

Modell von Musa bzw. Goel-Okumoto [1]

Am häufigsten zitiertes Zuverlässigkeitswachstumsmodell. Unterstellter Zusammenhang für die Anzahl der nachweisbaren Fehler in Abhängigkeit von der Test- oder Reifezeit t:

$$\varphi(t) = a\left(1 - e^{-bt}\right)$$

(a, b – experimentell zu bestimmende Parameter). Was für eine Fehlernachweisdichte müsste das IT-System haben?

$$\varphi(t) = a\left(1 - e^{-b \cdot t_{\mathrm{T}}}\right) \Rightarrow E\left(\varphi_{\mathrm{N}}(n)\right) = E\left(\varphi_{\mathrm{E}}\right) \cdot \int_0^1 h(p) \cdot \left(1 - e^{-n \cdot p}\right) \cdot dp$$

Das Modell unterstellt offenbar, dass alle Fehler mit gleicher Wahrscheinlichkeit nachweisbar sind:

$$h(p) = \begin{cases} 1 & \text{für } p = b \\ 0 & \text{sonst} \end{cases}$$

Untypisch für IT-Systeme!

Pseudo-Zufallsgeneratoren

Erzeugung von Pseudo-Zufallszahlen

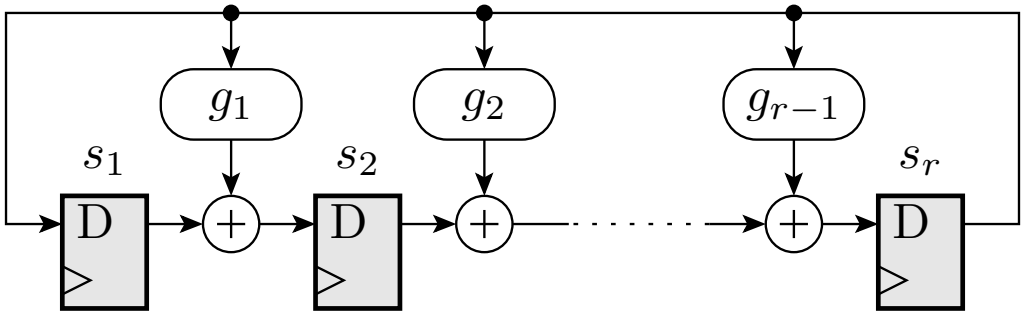
Testbeispiele, auch zufällige müssen wiederholbar sein, um Sollwerte zuordnen und den Reparaturerfolg kontrollieren zu können. Pseudo-zufällig bedeutet, dass die erzeugte Folge Zufallscharakter hat, bei Wiederholung aber immer dieselbe Folge von Testeingaben erzeugt wird. Beispielalgorithmus aus [?]:

```
#define pi 3.141592654
double Zufallszahl=0.230720081928;
neue_Zufallszahl() {
  Wiederhole 10 mal   {
      Zufallszahl = Nachkommastellen(Zufallszahl + pi)^8;
  }
}
```

Programmierumgebungen stellen dafür in der Regel Unterprogramme bereit zur Festlegung zur Initialisierung und zur Berechnung des nächsten Wertes.

Linear rückgekoppelte Schieberegister

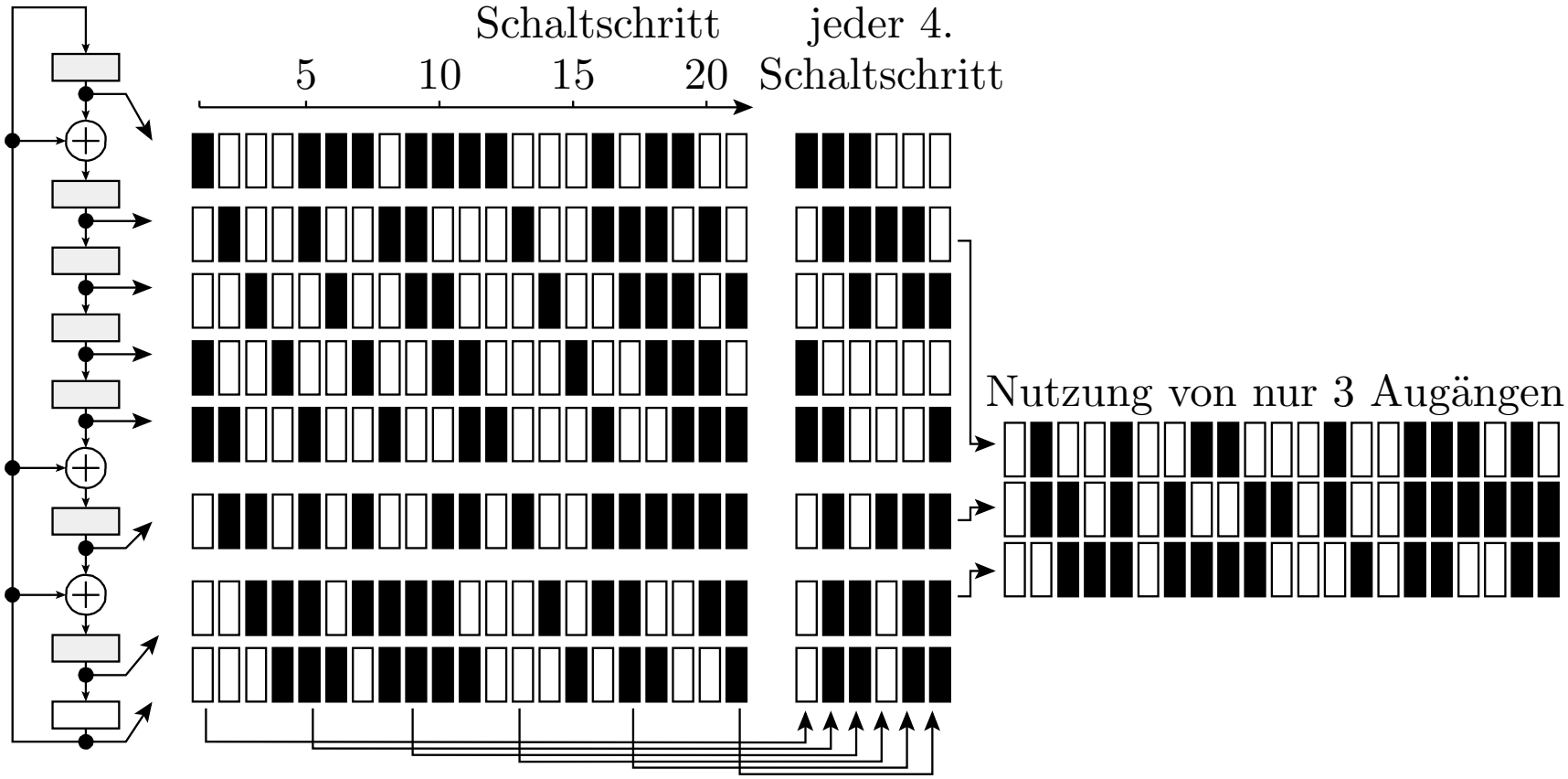
Einfachste und zur Testdatenbereitstellung meist ausreichende Lösung sind linear rückgekoppelte Schieberegister (LFSR linear feeback shift register) bzw. Algorithmen aus logischen und Verschiebeoperationen ähnlich CRC-Bildung:



Für die Rückführung g_i gehen nur bestimmte Werte, bei denen große Zyklen entstehen. Internet Suchbegriff »Primitive Polynome«. Beispiel für ein 16-Bit LFSR (vergl. [KeTV]:

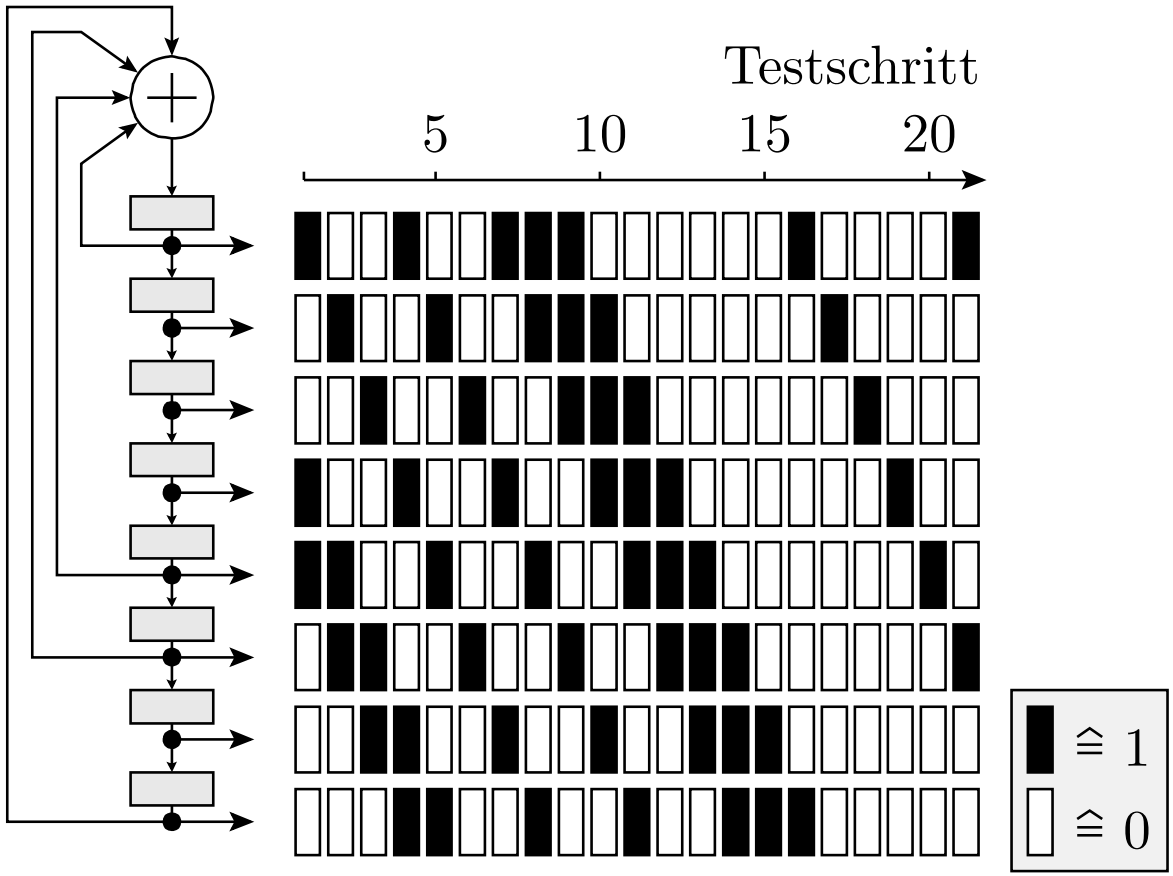
$$v^{16} \oplus v^5 \oplus v^3 \oplus v \oplus 1$$

Es bedeutet $g_1 = g_3 = g_5 = 1$, alle anderen null / weglassen.

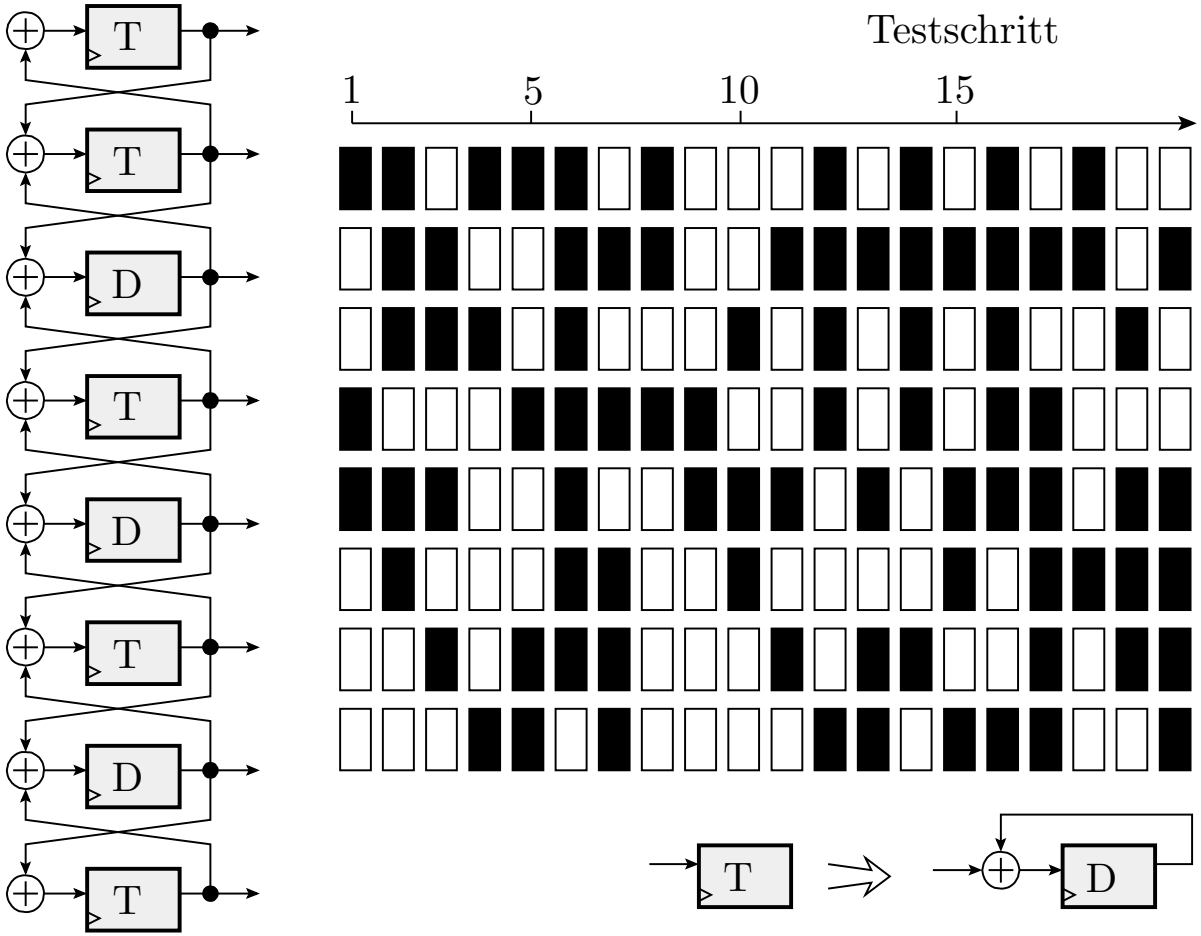


Falls die »Streifenmuster« durch die Schiebeoperationen stören, Generator für jede neue Testeingabe mehrere Schritte weiterschalten oder nur einen Teil der Ausgänge nutzen.

Statt einer Rückkopplung des Ausgangs auf mehrere Bitstellen können auch mehrere Bitstellen auf den Eingang rückgekoppelt werden:



Es gibt viele weitere lineare Automaten, die auch zyklisch Bitfolgen in zufälliger reihenfolge erzeugen, Zellenautomaten, bei denen jedes Folgebit aus dem eigenen und den Zuständen der Nachbarbits gebildet werden:

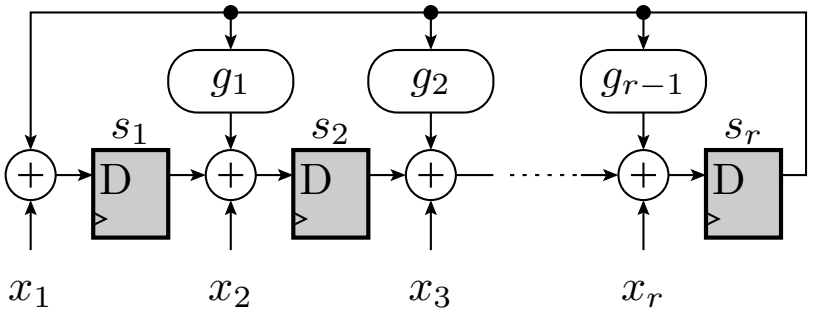


Selbsttest mit LFSR

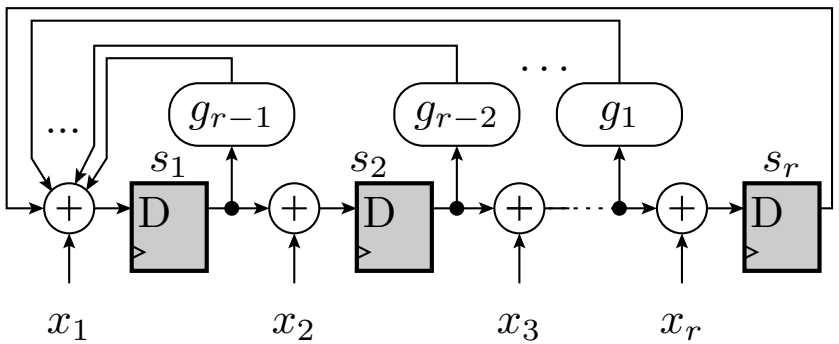
LFSR als Signaturregister

Register lassen sich nicht nur einfach zu Pseudo-Zufallsgeneratoren, sondern auch zu sog. Signaturanalysatoren erweitern. Das sind Schaltungen zur pseudo-zufälligen Bildung von Prüfkennzeichen, z.B. durch Polynomdivision.

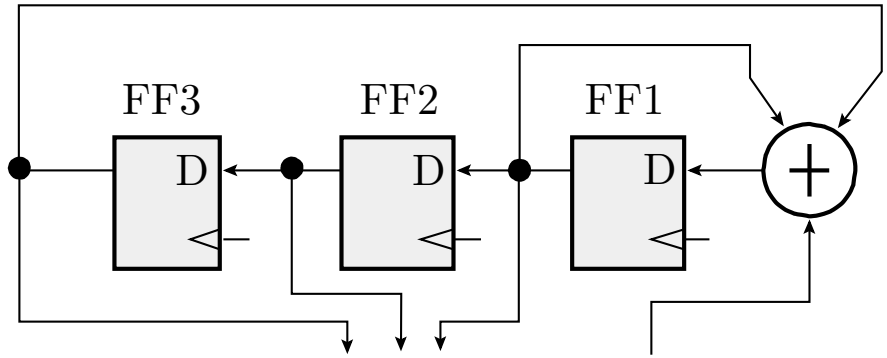
Parallels Signaturregister mit dezentraler Rückführung



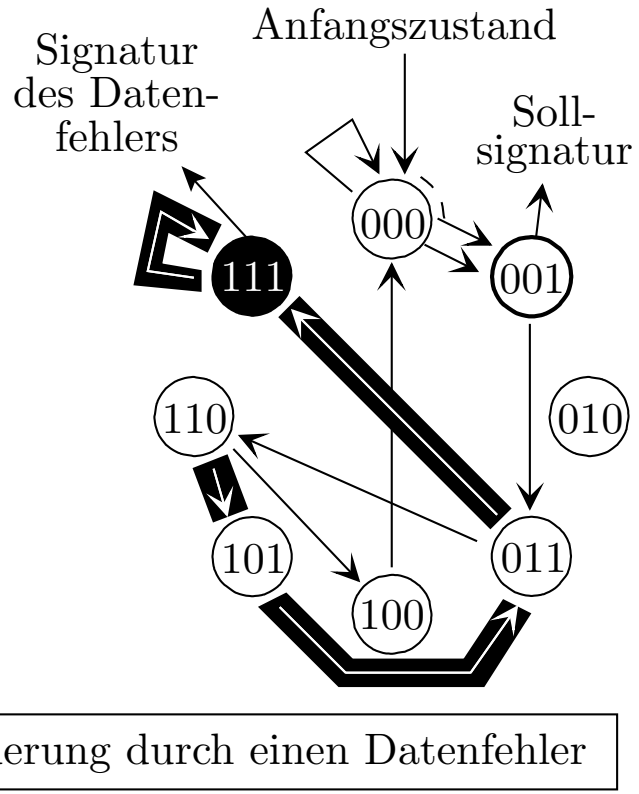
Parallels Signaturregister mit zentraler Rückführung



Fälschungen zirkulieren als pseudo-zufällige Differenzfolgen:



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	

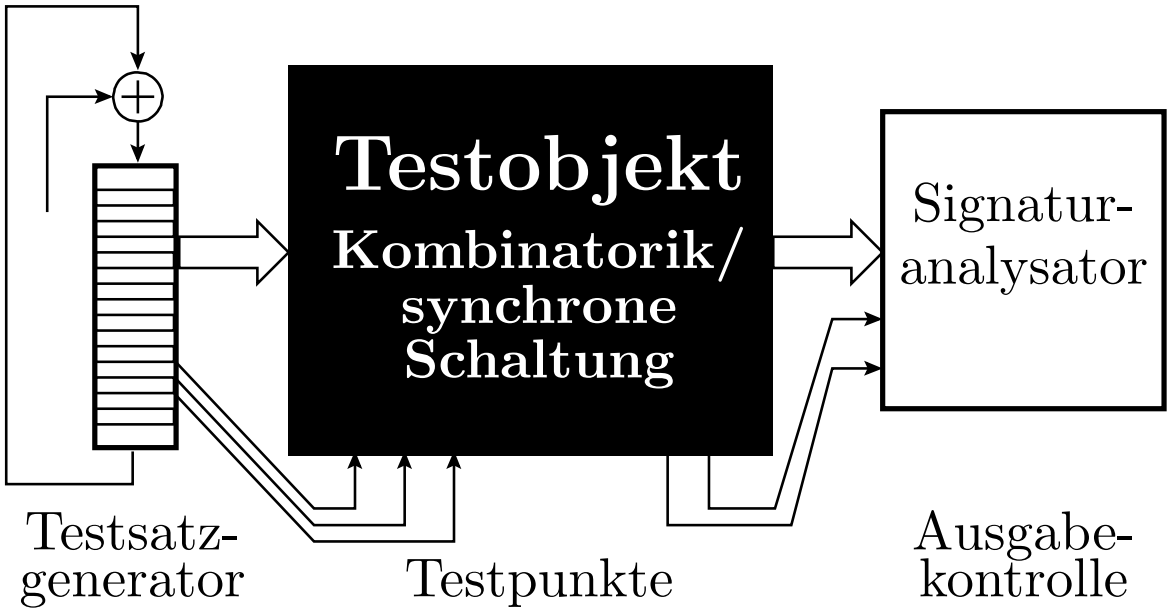


Wahrscheinlichkeit, dass weitere Verfälschungen ein Differenzfolge wieder auf null setzen (die Verfälschung maskieren) ist

$$2^{-r}$$

(r – Registerlänge) und kann beliebig klein gehalten werden.

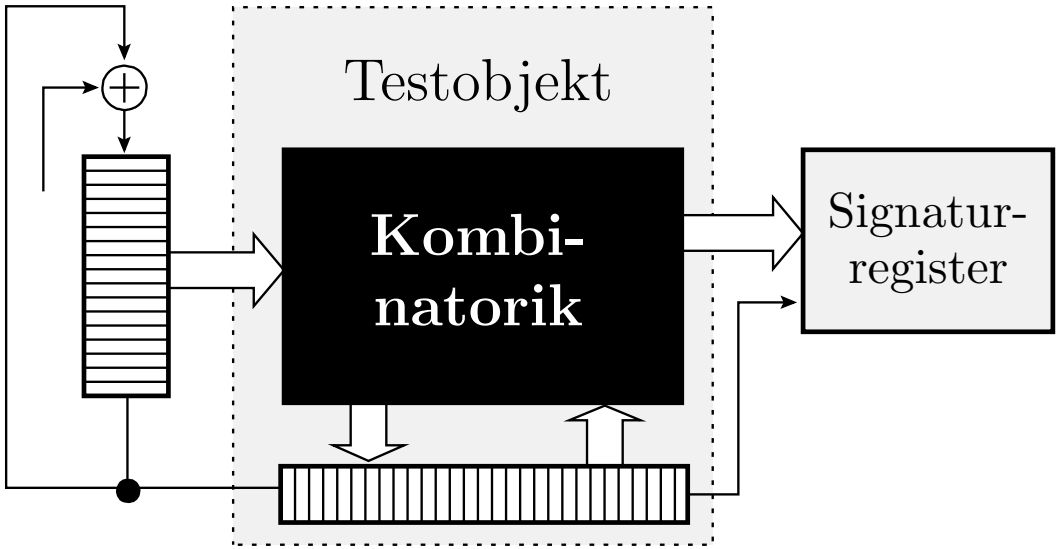
Prinzip eines Hardware-Selbsttests



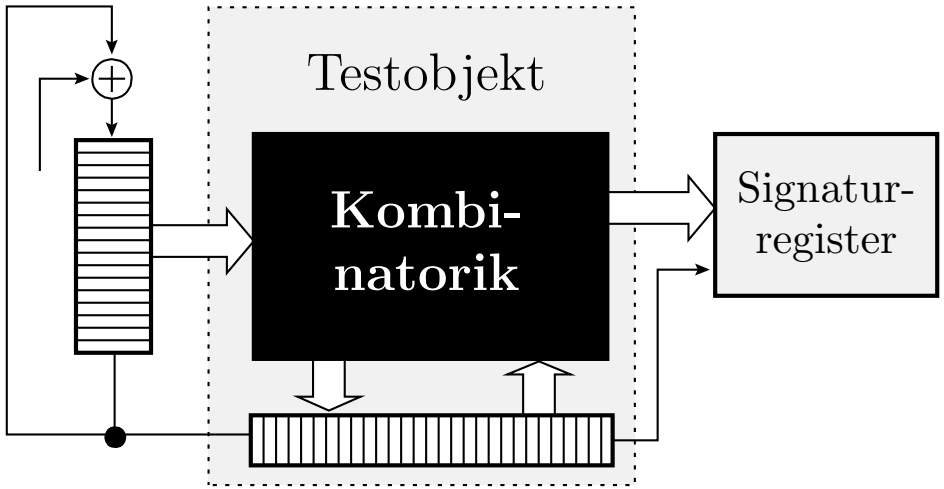
- Das Eingaberegister wird in einen Pseudo-Zufallsgenerator,
- das Ausgaberegister in ein Signaturregister umgeschaltet.
- Optionale Stimulierung und Beobachtung weiterer Punkte.
- Ein Testschritt pro Takt. Zum Schluss Signatur auswerten.

Kombination mit Scan-Registern

Die Fehlerüberdeckung in Abhängigkeit von der Testdauer lässt sich oft erheblich verbessern, wenn zusätzlich die internen Speicherzellen mit Pseudo-Zufallswerten belegt und ihre Folgezustände mit in die Prüfsummenbildung einbezogen werden:



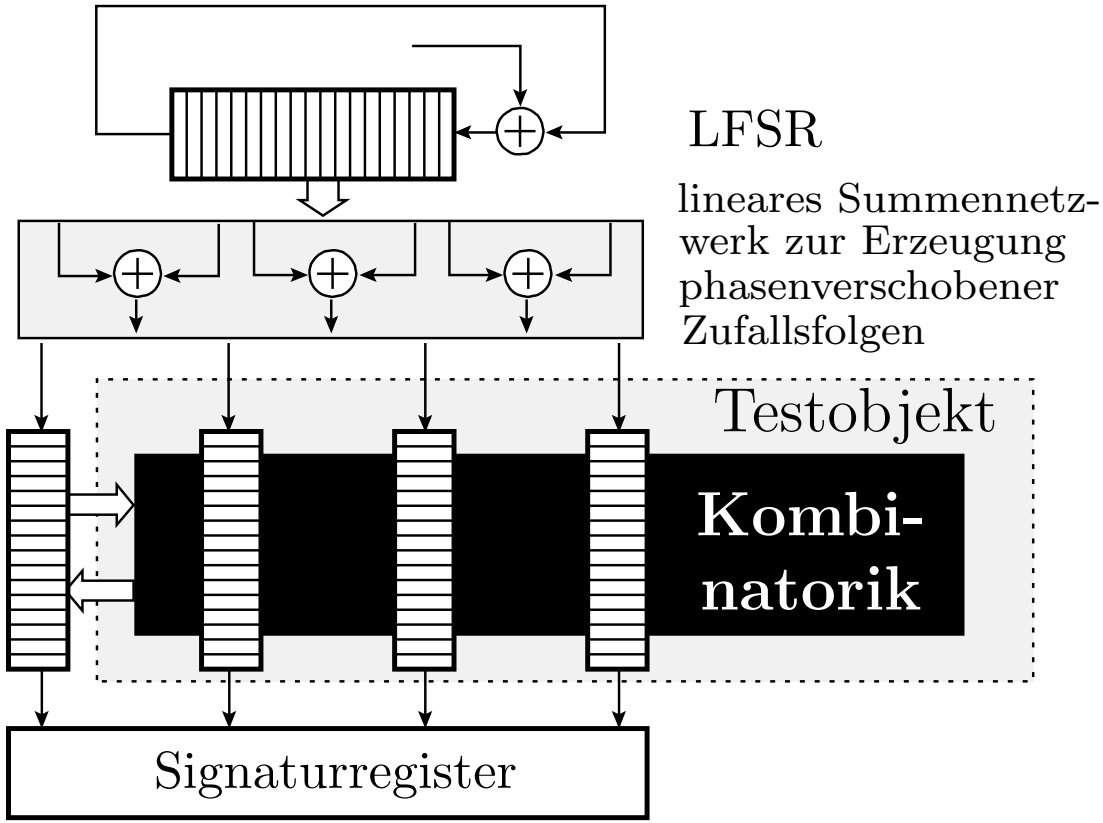
Die übliche Lösung Umschaltung der internen Register in ein Scan-Register.



Ein Scan-Register ist ein Register, das im Testmodus zwischen den normalen Operationsschritten serielle gelesen und neu beschrieben wird. Erweiterter Testablauf:

- Wiederhole für jeden Testschritte (z.B. 1 Million mal)
  - Schiebe den Scan-Registerzustand in das Signaturregister und beschreibe das Scan-Register mit Pseudo-Zufallswerten, die vom Eingaberegister gebildet werden.
  - Ausführung eines Testschritt mit Pseudo-Zufallswerten am Eingang und in den internen Registern und Ergebnisabildung in das Scan- und Signaturregister.

Für sehr große Systeme, z.B. Multi-Chip-Module gibt es vergleichbare Lösungen mit mehreren Scan-Registern, die zwischen den Testschritten parallel gelesen und mit neuen Zufallswerten beschrieben werden.



Weiterführende Literatur [Ke07]

Gewichteter Zufallstest

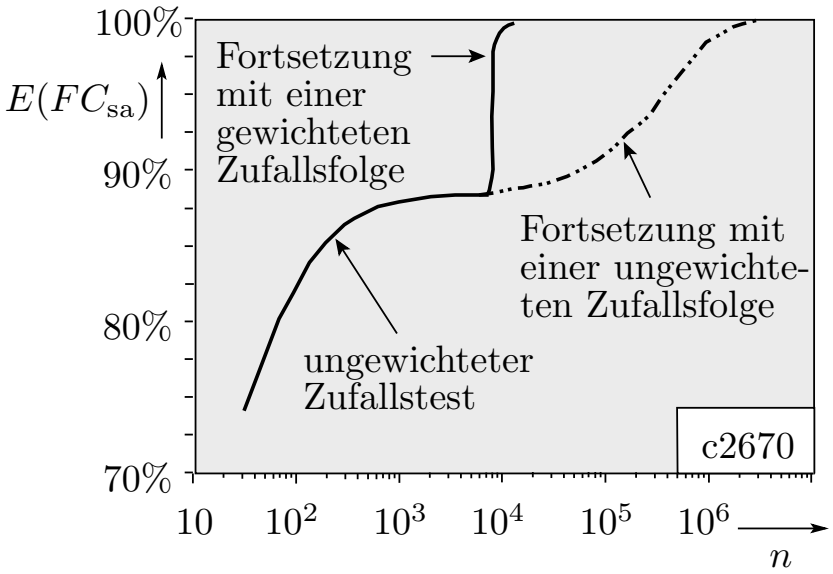
Gewichteter Zufallstest

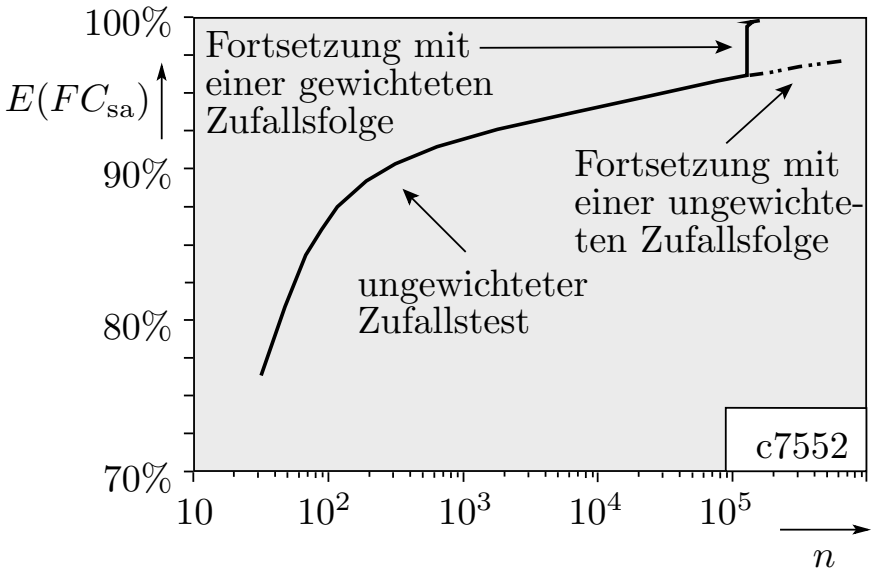
Ein Möglichkeit, die Fehlerüberdeckung von Zufallstestsätzen erheblich zu erhöhen, ist die Umschaltung des Operationsprofils während des Tests. Pragmatischer Ansatz:

1. Festlegung einer größeren Menge von Modellfehlern.
2. Längerer Test mit ungewichteten Zufallswerten und Abhaken aller damit nachweisbaren Modellfehler.
3. Suche für die restlichen Modellfehler eine Eingabewichtung, die deren Nachweiswahrscheinlichkeiten erheblich erhöht.
4. Längerer Test mit den so gewichteten Zufallswerten und Abhaken aller damit nachweisbaren Modellfehler.
5. Wenn erforderlich, Wiederholung von Schritt 3 und 4.

Experiment mit den Schaltungen c2670 und c7552[8]

- Test mit 10^4 bzw. 10^5 ungewichteten Zufallsmustern, die 90% bzw. 95% der Haftfehler nachweisen.
- Gezielte Testberechnung für die restlichen Haftfehler.
- Individuelle Wichtung aller Eingabebits zur Maximierung der mittleren Auftrittshäufigkeit der berechneten Testeingaben.





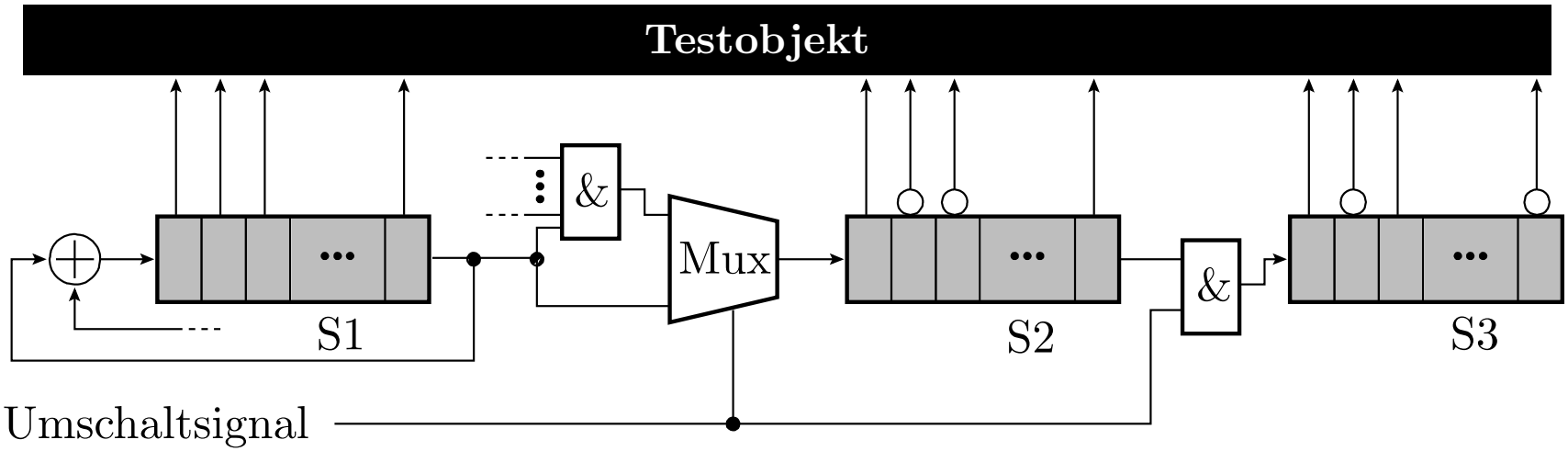
[8] Kombinatorische Benchmakschaltungen zum Vergleich von Testlösungen. Die Zahl hinter dem »c« ist die Anzahl der Signalleitungen.

Implementierung als Selbsttest

Im Experiment wurde der Wertebereich für die Wichtung auf die schaltungstechnisch einfach einstellbare Werte begrenzt:

$$g_i \in \left\{0, 2^{-k}, 0{,}5, 1 - 2^{-k}, 1\right\}$$

Diese werden mit wenigen UND-Gattern erzeugt und mit Scan-Registern an die Eingänge weitergeleitet.



(Details siehe [2]).

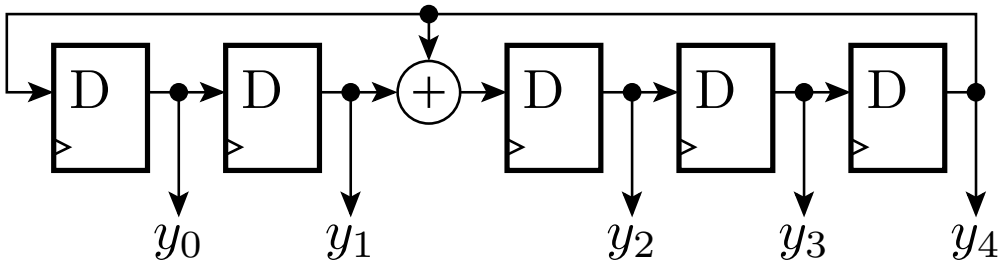
Aufgaben

Aufgabe 3.1: Ideensammlung

1. Schaltung mit Haftfehlern vorgeben. Berechnung der Nachweiswahrscheinlichkeiten.
2. Fehlerprofil und gewünsche zu erwartende Fehlerüberdeckung vorgeben. Erforderliche Testdauer bestimmen. Häufigkeit der falschen Ergebnisse bestimmen.
3. Änderung der Nachweiswahrscheinlichkeit durch Wichtung / anderes Operationsprofil.

Aufgabe 3.2: LFSR-Zykluslänge

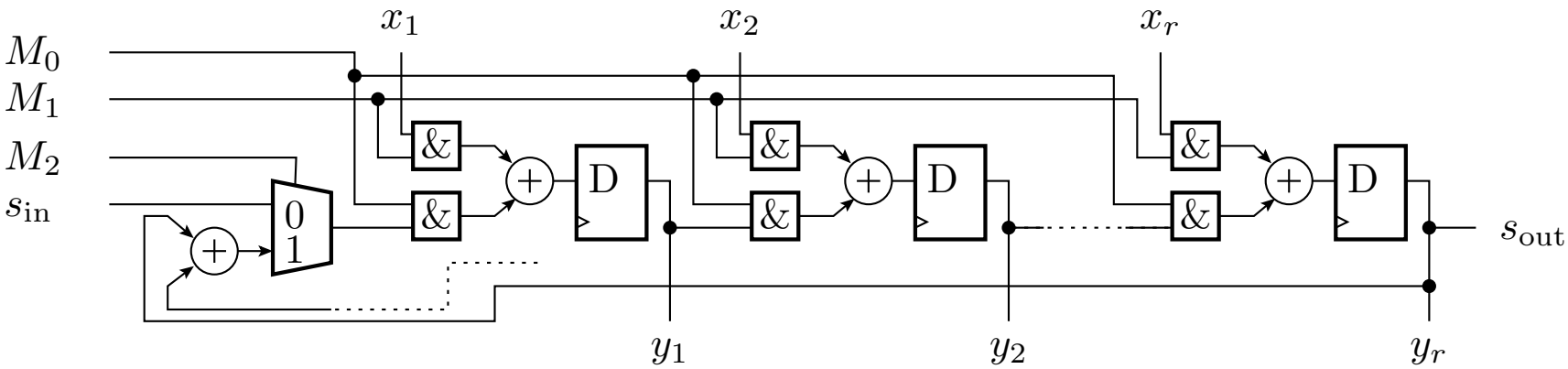
Untersuchen Sie für das nachfolgende 5-Bit linear rückgekoppelte Schieberegister die Zyklusstruktur.



1. Bestimmen Sie für jeden der 32 möglichen Zustände den Folgezustand.
2. Wie viele unterschiedliche Testeingaben lassen sich maximal hintereinander erzeugen?
3. Wie lautet die zyklisch generierte Testeingabefolge, wenn der Generator mit $y_4y_3y_2y_1y_0 = 01011$ initialisiert und jeder zweite Zusand als Testeingabe verwendet wird?

Aufgabe 3.3: Built-in Logic Block Observer

Die gezeigte Schaltung ist ein Built-in Logic Block Observer (BILBO) und führt in Abhängigkeit von den Steuersignalen M_0 bis M_2 die Funktionen aus: Intialisierung, normales Register, Schieberegister, Pseudo-Zufallsgenerator oder Signaturegister.



1. Welche Steuersignalbelegung steuert welche Funktion?
2. Vereinfachen Sie für jede dieser Steuersignalbelegungen die Schaltung durch Konstanteneliminierung (Vereinfachen bzw. Weglassen der logischen Verknüpfungen mit Konstanten.)

Aufgabe 3.4: xxx

Ein Fehler erzeugt im Mittel eine Fehlfunktion pro Stunde.
Welche Testzeit ist erforderlich, damit der Fehler mit einer Wahrscheinlichkeit von 95% nachgewiesen wird.
Derselbe Fehler wird, wenn ihn der Testsatz nachweist, nur in $MFC = 70\%$ der Fälle erkannt. Wie lang muss die Testzeit mindestens sein, damit der Fehler mindestens mit einer Wahrscheinlichkeit von 95% von den Kontrollfunktionen erkannt wird?

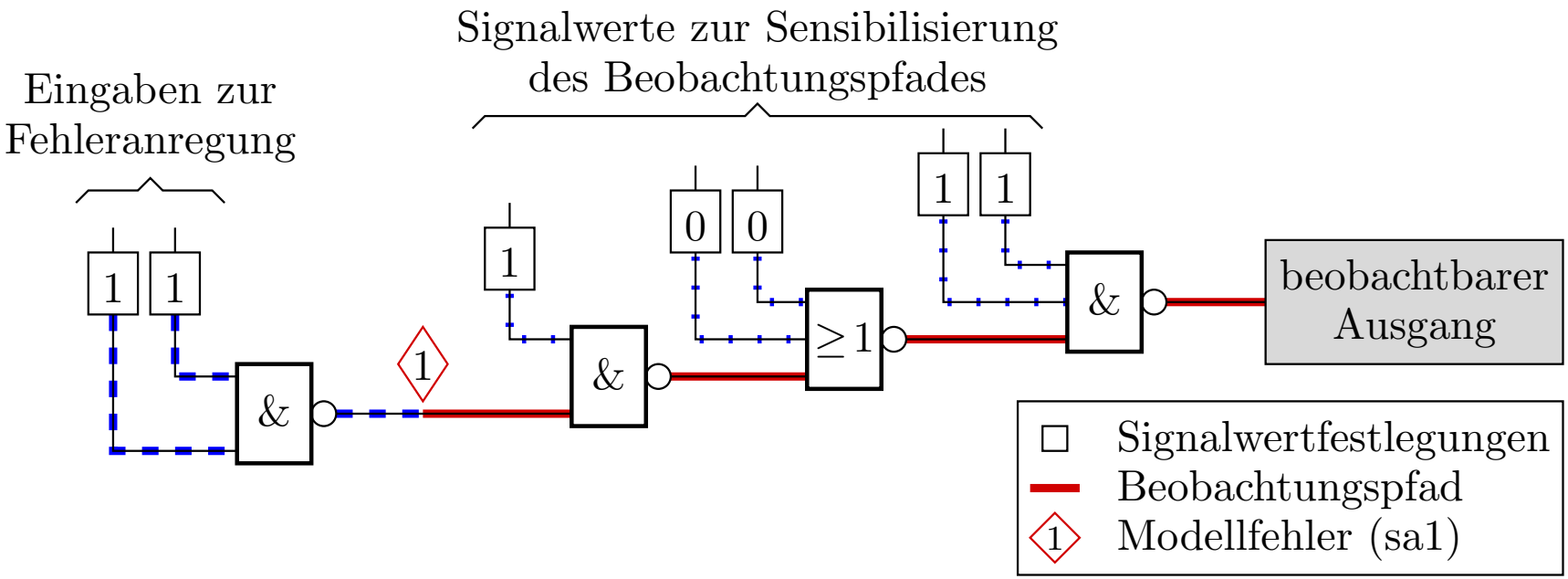
Aufgabe 3.5: Exponenten der Fehlerdichte

Für einen bestimmten Systemtyp sei bekannt, dass sich bei einer Verzehnfachung der Testzeit die beobachtbare Häufigkeit der falschen Ergebnisse auf etwa ein Fünfzehntel verringert.

1. Welchen Exponent k hat eine Potenzfunktion der Fehlernachweisdichte unter dieser Annhame?
2. Um welchen Faktor verringert sich die Fehleranzahl bei einer Verzehnfachung der Testzeit?

Testberechnung

Pfadalgorithmen



Vom (Modell-) Fehlerort werden durch Festlegung von Signalwerten

- in Signalflussrichtung Beobachtungspfade sensibilisiert und
- entgegen der Signalflussrichtung Steuerbedingungen eingestellt.

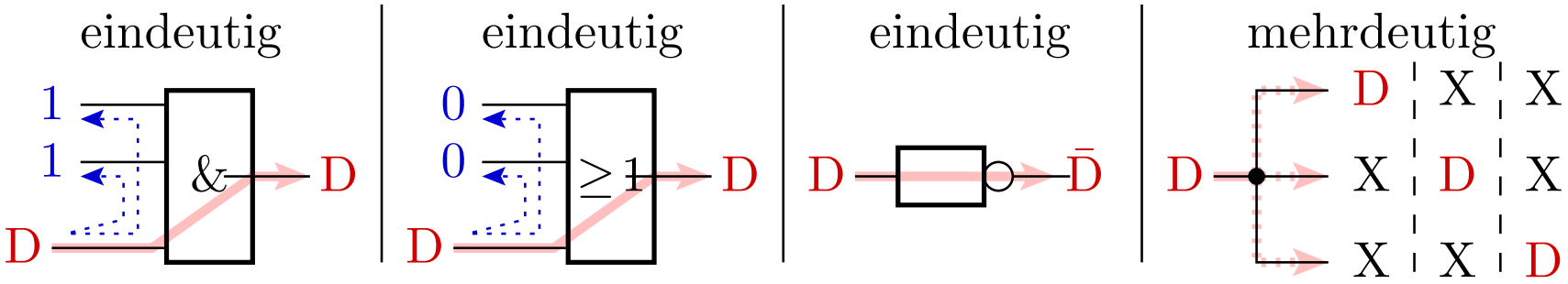
D-Algorithmus

D-Algorithmus

Erweiterung der Logikwerte um Pseudo-Werte:

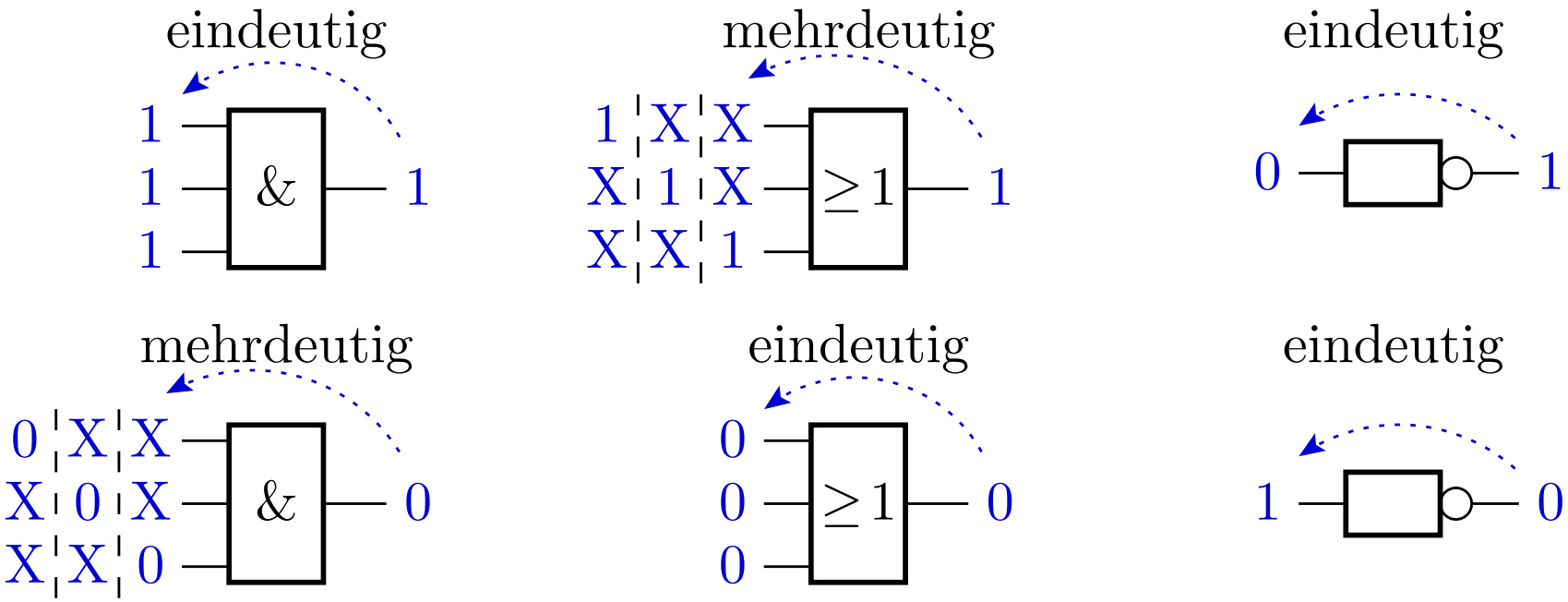
- D Signalwert ist gleich dem Logikwert am Fehlerort.
- $\bar{D}$ Signalwert ist gleich dem inversen Signalwert am Fehlerort.
- X Signalwert ist ungültig oder für den Fehlernachweis ohne Bedeutung.

Regeln für die Sensibilisierung eines Beobachtungspfades:

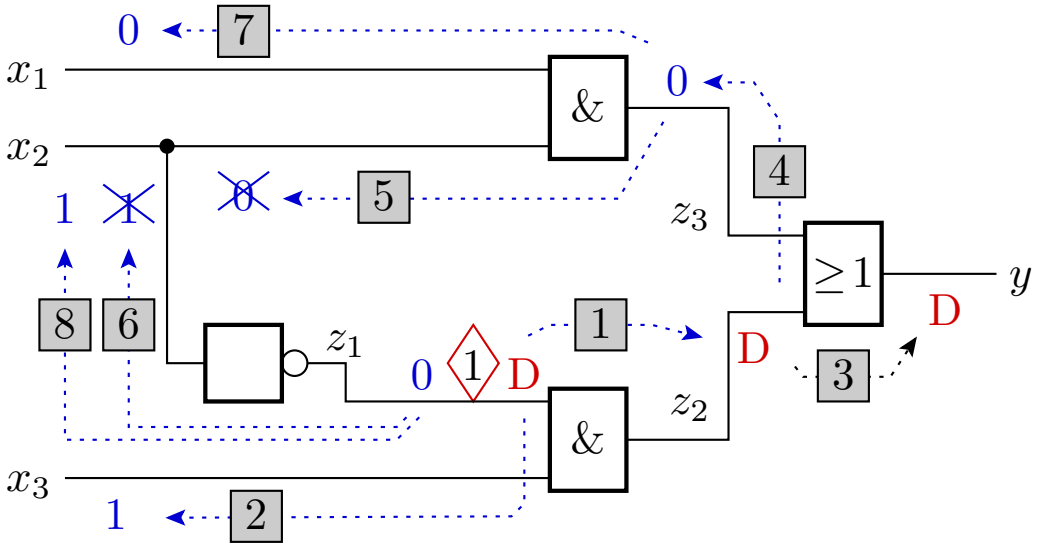


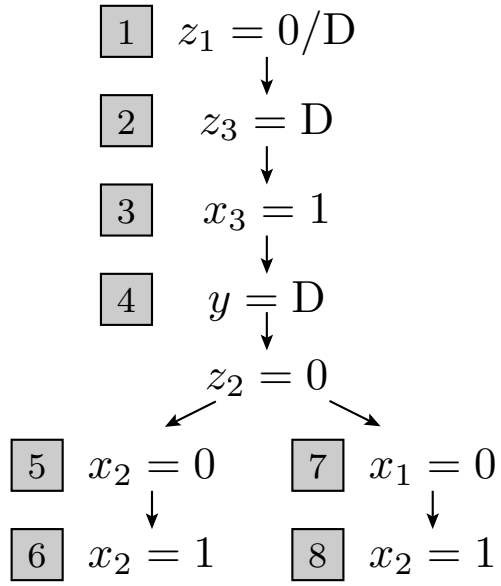
- Für $\bar{D}$ analog.

Einstellung von Steuerwerten



- Jede kombinatorische Schaltung kann in eine Schaltung aus AND, OR, NOT nachgebildet werden.
- Später Verallgemeinerung auf LUT (look-up table, Tabellenfunktionen).





(Widerspruch)

	x_3	x_2	x_1	z_3	z_2	z_1	y
0	X	X	X	X	X	0D	X
1	1	X	X	X	X	0D	X
2	1	X	X	X	D	0D	X
3	1	X	X	X	D	0D	D
4	1	X	X	0	D	0D	D
5	1	0	X	0	D	0D	D
6	1	~~01~~	X	0	D	0D	D
7	1	X	0	0	D	0D	D
8	1	1	0	0	D	0D	D

Baumsuche:

- Bei der Wertefestlegung können Widerspüche auftreten.
- Zurück zur letzten mehrdeutigen Entscheidung.
- Keine Lösung nach Durchmusterung des gesamten Baums. ⇒ Fehler nicht nachweisbar

Erfolgsrate der Testberechnung:

- Anteil der Fehler, für die ein Test gefunden oder für die der Beweis »nicht nachweisbar« erbracht wird.

- Die Testsuche für einen Fehler kann hunderte von Wertefestlegungen beinhalten.
- Der Suchraum wächst exponentiell mit der Anzahl der mehrdeutigen Festlegungen.; Suchräume der Größen $> 2^{30...40}$ nicht mehr vollständig durchsuchbar.
- Abbruch der Suche nach einer bestimmten Rechenzeit.

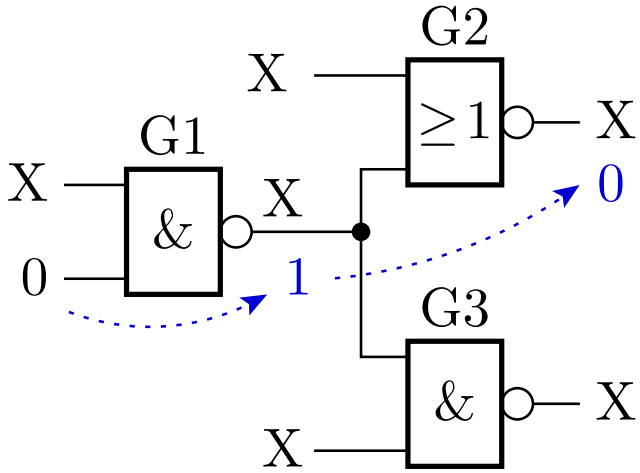
Heuristigen:

- Frühe Erkennung von Widersprüchen,
- Suchraumbegrenzung und
- gute Suchraumstrukturierung.

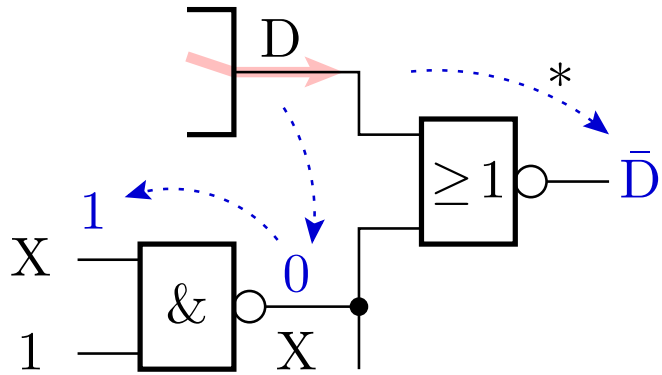
Implikationstest (Widerspruchsfrüherkennung)

- Aus den berechneten Wertefestlegungen alle eindeutig folgenden Werte berechnen.

Implikation in Signalflussrichtung



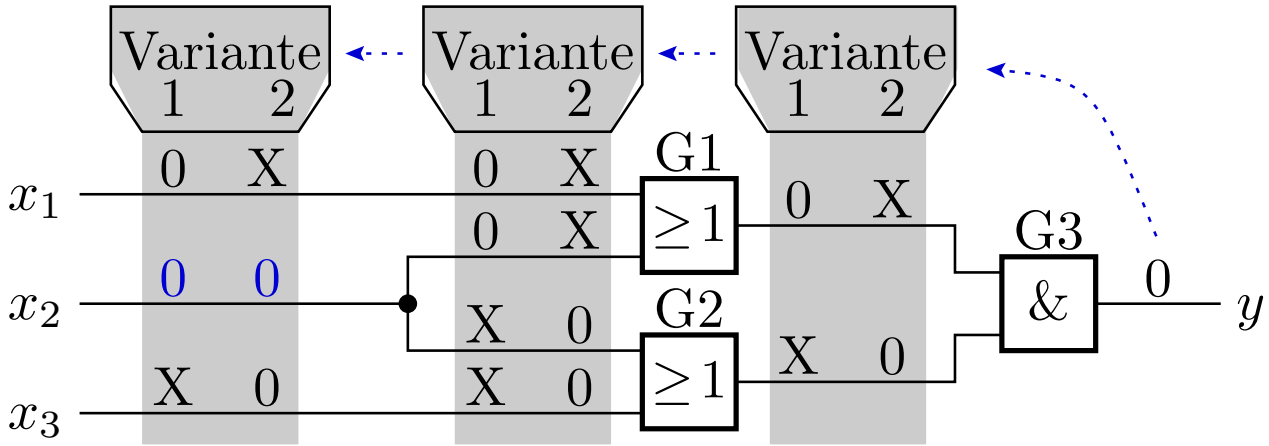
D-Pfad- und Rückwärtsimplikation



* eindeutige Weiterführung

- Mindert die Entscheidungsbaumtiefe.

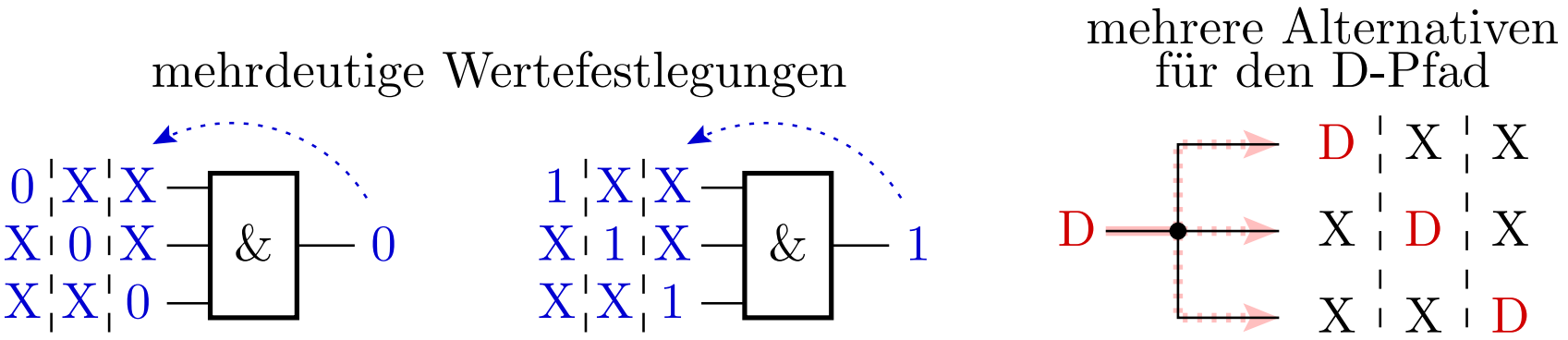
- Rückwärtsimplikation über mehrere Gatterebenen:



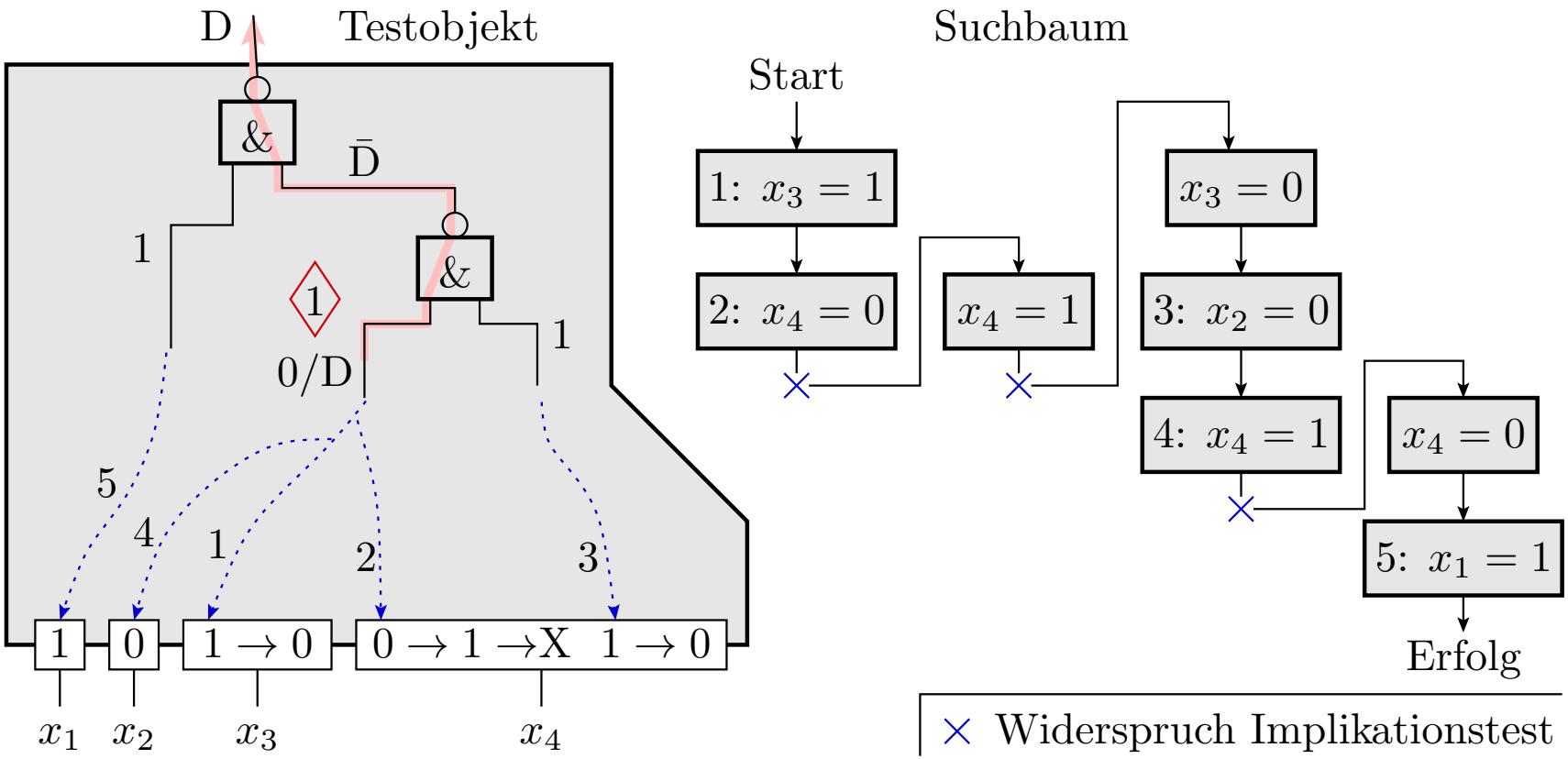
- Für $y = 1$ gibt es zwei Einstellmöglichkeiten.
- Für beide Möglichkeiten muss $x_2 = 1$ sein.
- Das Erkennen von Implikationen dieser Art mindert die Backtracking-Häufigkeit um bis zu 80 %.

Suchraumbegrenzung

- Der D-Algorithmus baut den Suchbaum über alle mehrdeutigen Wertefestlegungen auf:

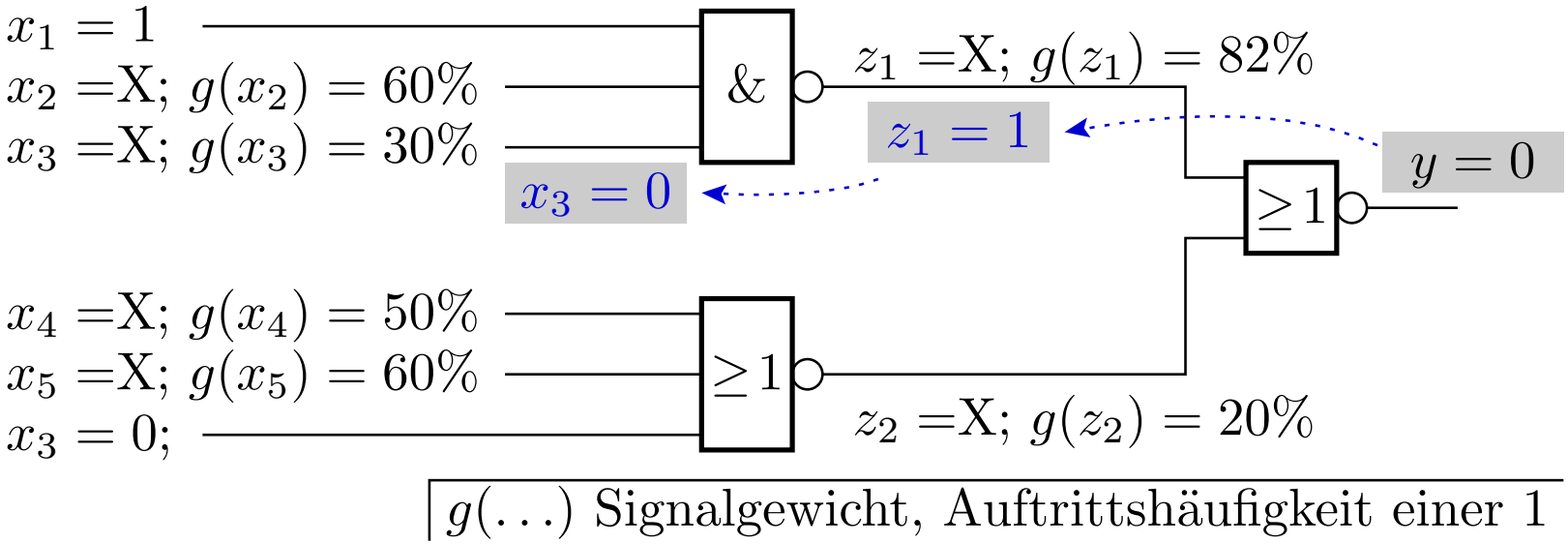


- Nur die Schaltungseingänge können unabhängig voneinander alle Wertevariationen annehmen.
- Es genügt, den Suchbaum mit den Eingabewertefestlegungen aufzubauen.
- Das begrenzt den Suchraum auf 2^{N_E} (N_E – Eingangsanzahl). Verringerung des Rechenaufwands um Zehnerpotenzen.



- Lange Steuerpfade vom Fehlerort und vom D-Pfad zu Eingängen.
- Aufbau des Suchbaums über Eingangssignale.
- Wenn Implikationstest-Widerspruch, letzte Eingabefestlegung invertieren.

Geschätzte Erfolgswahrscheinlichkeiten



- Schätzen der Signalgewichte (Auftrittshäufigkeit einer »1«) über ein kurze Simulation mit Zufallswerten oder analytisch.
- Wahl der Steuerwerte / Beobachtungspfade, die mit größerer Wahrscheinlichkeit aktivierbar / sensibilisierbar sind.

Komplexe Funktionsbausteine

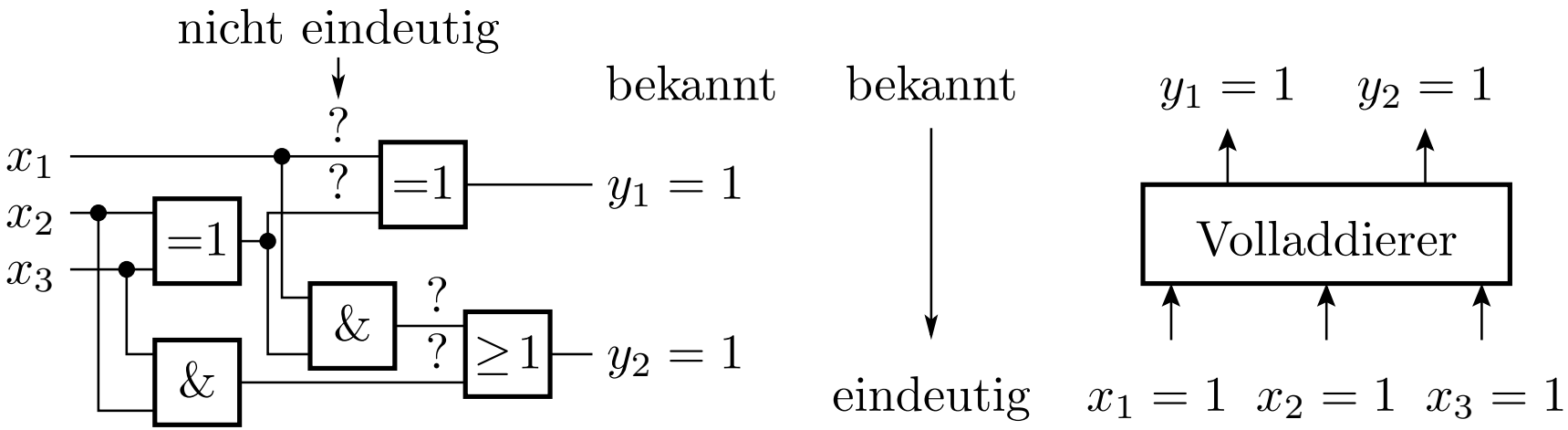
- Beschreibung durch Tabellenfunktion (Bsp. Volladdierer):

x_2	x_1	x_0	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

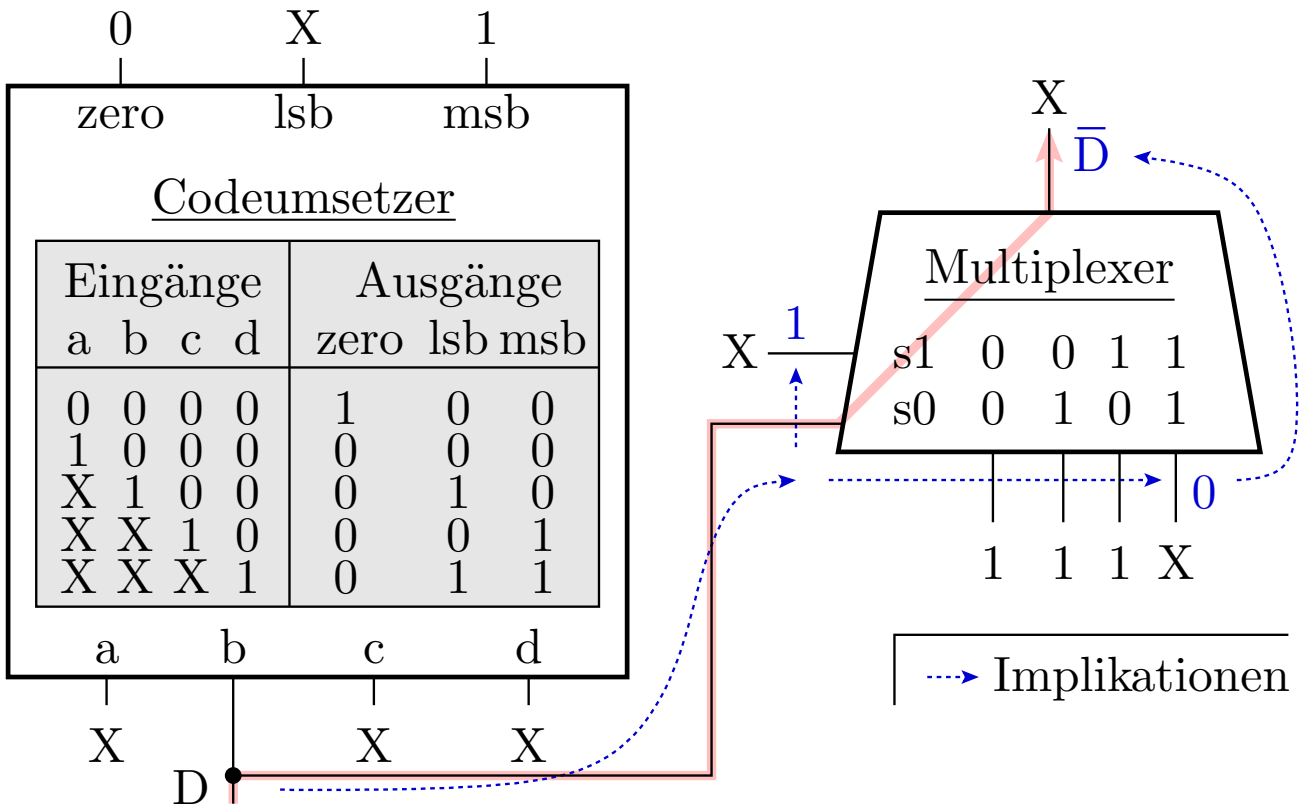
gegeben		Lösungsmenge
XXX00	$\Rightarrow$	00000
01DXX	$\Rightarrow$	01D$\bar{\text{D}}$D
1XXXD	$\Rightarrow$	10D$\bar{\text{D}}$D, 1D0$\bar{\text{D}}$D
11XX1	$\Rightarrow$	11111, 111001

- Vervollständigung des Vektors der gegebenen Anschlusswerte durch Vergleich mit allen Tabellenzeilen:
  - »1« und »0« passen nur auf »1« und »0«
  - »X« passt immer
  - »D« muss für »D=0« und für »D=1« passen

Implikationstest an einem Volladdierer



- An der Gatterbeschreibung eines Volladdierers ist die Implikation $y_1 = y_2 = 1 \Rightarrow x_1 = x_2 = x_3 = 1$ nicht zu erkennen. Lösungsfindung über Baumsuche.
- Bei Zusammenfassung zu einer Tabellenfunktion wird die Lösung bereits bei der Anschlusswertevervollständigung erkannt.



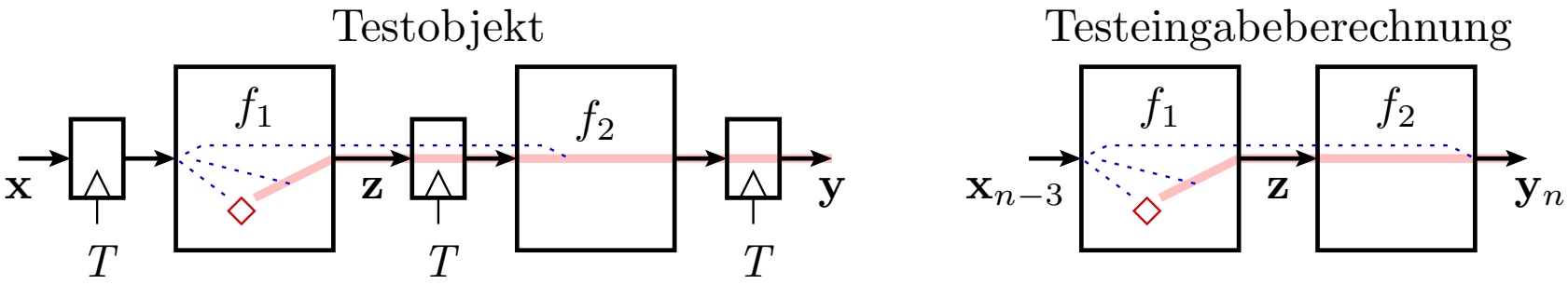
- »lsb« hängt bei »zero=0« und »msb=1« nicht von »b« ab. Eindeutiger D-Pfad über Multiplexer.
- Tabelleneingabewerte »X« (Eingang beeinflusst nicht die Ausgabe) führt zu Tabellen mit $\ll 2^{A_E}$ Tabellenzeilen.

Sequentielle Schaltungen

Sequenzielle Schaltungen

Zurückführung auf kombinatorische Schaltungen:

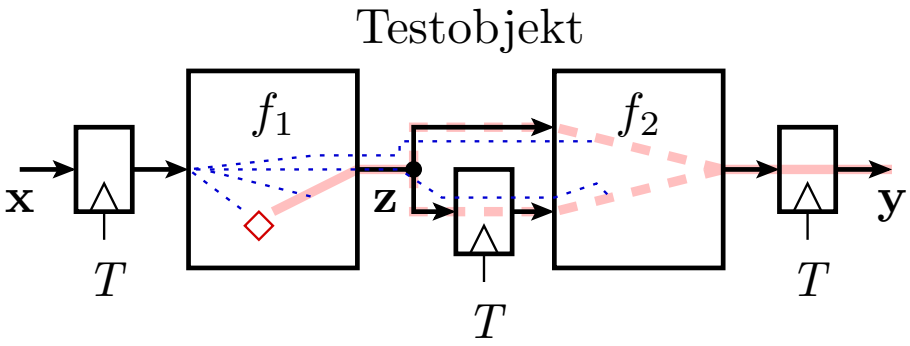
- nur Abtastung

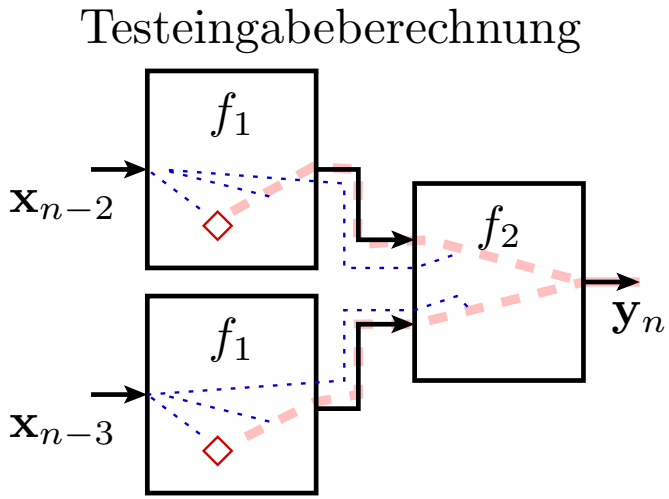


- Testberechnung wie für eine kombinatorische Schaltung
- Zeitversatz zwischen Ein- und Ausgabe berücksichtigen

```
x <= Eingabe_1; wait on RISING_EDGE(T);
x <= Eingabe_2; wait on RISING_EDGE(T);
x <= Eingabe_3; wait on RISING_EDGE(T);
x <= Eingabe_4; wait for tP; assert y=Ausgabe_1 ...;
x <= Eingabe_5; wait for tP; assert y=Ausgabe_2 ...;
```

Verarbeitung in mehreren Zeitebenen



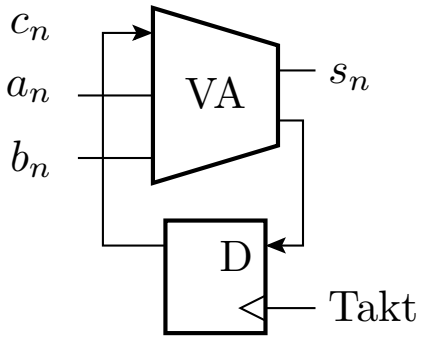


- Die kombinatorisch Ersatzschaltung enthält mehrere Kopien derselben Schaltung.
- Die Haftfehler sind in jeder Kopie.
- Eingaben mehrerer Zeitebenen / Mehr-Pattern-Test:

```
x <= Eingabe_1A; wait on RISING_EDGE(T);
x <= Eingabe_1B; wait on RISING_EDGE(T);
... wait on RISING_EDGE(T); assert y=Ausgabe_1 ...;
```

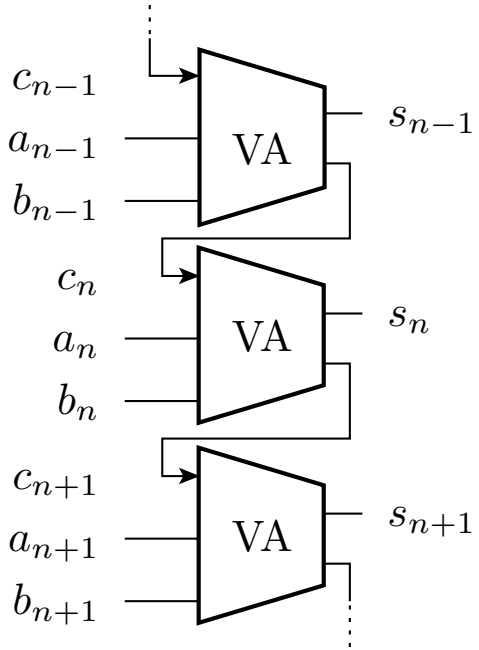
Schaltungen mit Rückführung

serieller Addierer



VA - Volladdierer

aufgerollter Addierer



- Ersatzschaltung aus unbegrenzt vielen Kopien.
- Regeln zur Begrenzung der Länge der Steuer- und Beobachtungspfade.

Zusammenfassung

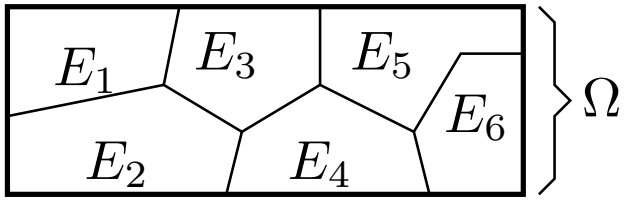
- Das Haftfehlermodell ist das an meisten angewendete Fehlermodell.
- Modellfehleranzahl etwa proportional zur Systemgröße.
- Erlaubt relativ einfache und effiziente Algorithmen für die Berechnung der Fehlermengen, die Fehlersimulation und die Testberechnung für speicherfreie digitale Schaltungen.
- Anwendbarkeit der Tessatzauswahl- und Bewertungstechniken durch prüfgerechten Entwurf sichern.

Andere Testauswahl und -bewertungstechniken werden im Weiteren mit den Haftfehler-basierten Techniken verglichen.

Äquivalenzklassen

Testauswahl mit Äquivalenzklassen

- Äquivalenzklasse: Menge ähnlich verarbeiteter Eingabewerte
- Annahme, dass die meisten Fehler mit Werten an den Bereichsgrenzen nachweisbar sind.



Ω Eingaberaum
E_i Äquivalenzklasse

- Gezielte Suche von Werten an Bereichsgrenzen.
- optionale Wichtung der Testanzahl innerhalb der Bereiche nach Wichtigkeit, Kompliziertheit, ... der Funktion

---

- gehört zu den Funktionstests
  - Testerstellung bereits vor oder unabhängig von der Codierung möglich
  - nicht ganz unabhängige (zufällige) Auswahl in Bezug auf zu erwartende Fehler
- Auswahltechnik für Programmmodule mit EVA-Struktur

praktisches Vorgehen:

- Beschreibung der gesamten Funktion durch Bedingungen $b_i(\mathbf{x})$ und Funktionen $f_i(\mathbf{x})$, nach denen unter diesen Bedingungen die Ausgabe zu berechnen ist:

  `wenn` $b_1(x)$ `dann` $y = f_1(x)$
  `wenn` $b_2(x)$ `dann` $y = f_2(x)$

- Bedingungen bestehen aus Vergleichen und logischen Operationen, z.B.:

```
int a, b, c; if ((a>3)|(b+c>5))&(a<34) ...
```

- Suche Wertetupel, bei denen geringe Werteänderungen einer Variablen den Wahrheitswert ändern:

a	b	c	nachweisbarer Verfälschungen
3	-3	4	a+[1...30]
34	7	9	a-[1...]

- Ergänzen einiger Zufallstests innerhalb des Bereiches

```
int a, b, c;
...
if ((a>3)|(b+c>5))&(a<34)
  {...
```

a	b	c	nachweisbarer Verfälschungen
3	-3	4	a+[1...30]
34	7	9	a-[1...]
1	2	3	b+[1...]; c+[1...]
2	4	2	b-[1...]; c-[1...]
33	14	-15	a+[1...]
34	-2	28	a-[1...]

Beispiel einer abschnittsweise definierte Funktion

x	y	Beispiel Testauswahl für x
$0 \leq x < 1$	$y := x$	0, 0.5, 0.999
$1 \leq x < 3$	$y := (x-2)^2$	1, 1.5, 2, 2.7, 2.98
$3 \leq x \leq 4$	$y := 3 - x$	3, 3,2, 4
sonst	Fehlermeldung	-100, -0.1, 4.01, 345

Beschreibung als Programm:

```
function f(x: REAL) return REAL is
 variable y: REAL;
begin
 assert x>0.0 and x<=4.0 report "Eingabefehler";
 if    x<1.0 then y := x;              -- AK1
 elsif x<3.0 then y := (x-2.0)**2;     -- AK2
 else             y := 3.0-x;          -- AK3
 end if;
 return y;
end function;
```

x	y	Beispiel Testauswahl für x
$0 \leq x < 1$	$y := x$	0, 0.5, 0.999
$1 \leq x < 3$	$y := (x-2)^2$	1, 1.5, 2, 2.7, 2.98
$3 \leq x \leq 4$	$y := 3 - x$	3, 3,2, 4
sonst	Fehlermeldung	-100, -0.1, 4.01, 345

Testrahmen mit Testeingaben:

```
type tRealArray is array (NATURAL range <>) of REAL;
constant xVektor: tRealArray := (0.0, 0.5, ..., 345.0);
variable x, y: REAL;
...
for idx in xVector'RANGE loop
 x := xVector(idx);
 y := f(x);
 write("x=" & str(x) & "  y=" & str(y));
end loop;
```

Die Testausgaben sind hier manuell zu kontrollieren.

Richtlinien zur Festlegung der Äquivalenzklassen:

- überschaubare Anzahl
- einfach beschreibbare Einzelfunktionen[9]

Abschlussbemerkungen

- Über Äquivalenzklassen ausgewählte Tests erkennen in der Regel mehr Fehler als Zufallstestsätze derselben Länge.
- Den Aufwand, eine Funktionsbeschreibung in die »Äquivalenzklassenform« zu bringen, ist relativ hoch.
- Aus einer Äquivalenzklassenbeschreibung lässt sich relativ einfach (auch automatisiert) ein Programm erzeugen;
- Über den Weg »Nachbildung durch Äquivalenzklassenform« ⇒ »Abbildung in ein Programm« lassen sich diversitäre Beschreibungen für Mehrversionsvergleich ableiten.

[9] gut strukturiert, kurz und übersichtlich sind auch Grundregeln zur Fehlervermeidung

Speichertest

Speichertest

Schreib-Lese-Speicher (RAM) besteht aus der Speichermatrix, dem Adressdecoder, der Eingabelogik und der Ausgabelogik. Fehler in der Decodier-, Eingabe- und Ausgabelogik werden als Fehler in der Speichermatrix modelliert. Modellfehler für Speicher:

- Haftfehler (Lesewert ist ständig null oder ständig eins),
- Übergangsfehler,
- Stuck-open-Fehler (Ausgabe des zuletzt gelesenen Wertes),
- zerstörendes Lesen (Löschen des Inhalts beim Lesen) und
- gegenseitige Beeinflussung unterschiedlicher Zellen.

Aufsetzend auf den Fehlerannahmen gibt es algorithmisch beschriebene Testabläufe mit der Speicherorganaisation als Parameter, die alle unterstellen Modellfehler nachweisen.

beteiligte Zellen	Name	Definition	Fälle	Testfolge für den Nachweis
1	Haftfehler	Wert der Speicherzelle ist nicht setzbar	stuck-at-0 stuck-at-1	$W(i)1$, $R(i)$ $W(i)0$, $R(i)$
	Übergangsfehler	Wert der Speicherzelle i ist nur in einer Richtung änderbar	kein Übergang $1 \to 0$ $0 \to 1$	 $W(i)1$, $R(i)$, $W(i)0$, $R(i)$ $W(i)0$, $R(i)$, $W(i)1$, $R(i)$
	Stuck-open-Fehler	kein Zugriff auf Speicherzelle i (Ausgabe des Wertes der vorherigen Leseoperation)		$W(i)0$, $R(j)1$, $R(i)$, $W(i)1$, $R(j)0$, $R(i)$
	zerstörendes Lesen	Inhalt von Speicherzelle i wird beim Lesen verändert	$R(i) \Rightarrow C(i) = \overline{C(i)}$	$W(i)0$, $R(i)$, $R(i)$ $W(i)1$, $R(i)$, $R(i)$
2	Kopplung Typ 1	Veränderung des Inhalts von Zelle i bestimmt Zustand in Zelle j	$W(i)0 \Rightarrow C(j) = 0$ $W(i)0 \Rightarrow C(j) = 1$ $W(i)1 \Rightarrow C(j) = 0$ $W(i)1 \Rightarrow C(j) = 1$	$W(i)0$, $R(j)$, $W(i)1$, $R(j)$, $W(i)0$, $R(j)$
	Kopplung Typ 2	Veränderung des Inhalts von Zelle i bewirkt eine Änderung in Zelle j	$C(i) = \overline{C(i)} \Rightarrow C(j) = \overline{C(j)}$	$R(j)$, $W(i)0$, $R(j)$, $W(i)1$, $R(j)$

$W(i)1$ Schreibe 1
$W(i)0$ Schreibe 0
$R(j)$ Lese eine beliebige andere Zelle
$R(i)$ Lese Inhalt und Vergleiche mit Sollwert
$R(j)0$ Lese eine andere Zelle, in der 0 steht
$R(j)1$ Lese eine andere Zelle, in der 1 steht

Marching Test

Adresse i	Initialisierung	March 1	March 2	March 3
0	$W(i)0$	$R(i)0, W(i)1$	$R(i)1, W(i)0$	$R(i)0, W(i)1$
1	$W(i)0$	$R(i)0, W(i)1$	$R(i)1, W(i)0$	$R(i)0, W(i)1$
2	$W(i)0$	$R(i)0, W(i)1$	$R(i)1, W(i)0$	$R(i)0, W(i)1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$N-1$	$W(i)0$	$R(i)0, W(i)1$	$R(i)1, W(i)0$	$R(i)0, W(i)1$

	March 4		March 1a		March 2a
0	$R(i)1, W(i)0$	Wartezeit	$R(i)0, W(i)1$	Wartezeit	$R(i)1$
1	$R(i)1, W(i)0$		$R(i)0, W(i)1$		$R(i)1$
2	$R(i)1, W(i)0$		$R(i)0, W(i)1$		$R(i)1$
$\vdots$	$\vdots$		$\vdots$		$\vdots$
$N-1$	$R(i)1, W(i)0$		$R(i)0, W(i)1$		$R(i)1$

Mehrfachen Durchwandern des Speichers in unterschiedlicher Reihenfolge mit der Operationsfolge Zelle Lesen, Wert kontrollieren und inversen Wert zurückschreiben.

Aufgaben

Ideensammlung

1. D-Algorithmus per Hand.
2. Schaltung mit festgelegten Werten vorgeben. Alle implizit festgelegten Werte bestimmen.
3. Beispielschaltung als Schaltplan vorgeben. Wie sieht die aufgerollte Schaltung für die Testsuche aus?

Fehlerbeseitigung

Fehlerbeseitigung

Die Reparatur alter Fehler kostet oft mehr als die Anschaffung neuer. (Wieslaw Brudzinski, 1920*)

Vorabtests vor der Fehlersuche:

- Fehler oder Störung? ⇒ Fehlverhalten reproduzierbar? Lassen sich Tests finden, die bei Wiederholung dasselbe Fehlverhalten anregen? Fehlerwirkung beseitigbar?
- Entwurfs- oder Fertigungsfehler? ⇒ Unterschiedliches Verhalten baugleicher Systeme?

Möglichkeiten der Fehlerbeseitigung:

- Ersatz

Reparatur (oft Ersatz von Teilsystemen)

Ersatz

Ersatz, Fehleranzahl und Fehleranteil

Eine Einheit wird getauscht, wenn sie mindestens einen nachweisbaren Fehler enthält. Zu erwartende Fehleranzahl ungetesteter Systeme nach F2, Abschn.3.2:

$$E(\varphi) = \sum_{i=1}^{N_\varphi} h_i = N_\varphi \cdot \bar{h}$$

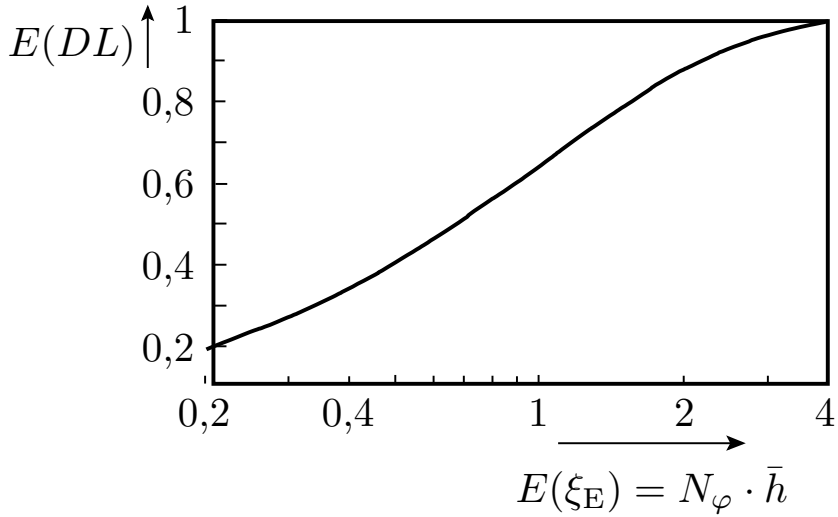
(N_φ – Anzahl der potentiellen Fehler; h_i – Auftrittshäufigkeiten der einzelnen potenziellen Fehler; $\bar{h}$ – mittlere Fehlerauftrittshäufigkeit). Der zu erwartende Fehleranteil ist die Wahrscheinlichkeit, dass das System mindestens einen Fehler enthält:

$$E(DL) = 1 - \prod_{i=1}^{N_\varphi} (1 - h_i) = 1 - e^{-N_\varphi \cdot \bar{h}}$$

Ersatz ist nur bei Verfügbarkeit fehlerarmer Ersatzteile oder beim Fertigungstest für hinreichend kleine Bausteine mit im Mittel nicht viel mehr als einem Fehler sinnvoll.

$$E\left(DL\right) = 1 - e^{-N_\varphi \cdot \bar{h}}$$

Für den Fertigungstest steht die Anzahl der potenziellen Fehler N_φ für die Systemgröße und $\bar{h}$ für die Güte des Entstehungsprozesses. Derzu erwartende Fehleranteil ist gleichzeitig der zu erwartende Ausschuss.



Durch Aussortieren werden alle erkennbaren Fehler beseitigt. Übrig bleiben nur nicht erkannte Fehler:

$$E\left(\varphi_\mathrm{T}\right) = \left(1 - E\left(FC\right)\right) \cdot E\left(\varphi\right)$$

(FC – Fehlerüberdeckung). Fehleranteil nach Ersatz erkannter fehlerhafter Systeme:

$$E\left(DL_\mathrm{T}\right) = 1 - e^{-(1-E(FC)) \cdot N_\varphi \cdot \bar{h}}$$

Fehleranteil nach Aussortieren aller fehlerhaften Objekte in dpu (defects per unit) bzw. dpm (defects per million):

vor dem Test	$E(\varphi) = 2$	$E(\varphi) = 1$	$E(\varphi) = 0{,}5$	$E(\varphi) = 0{,}2$
ohne Test	0,865 dpu	0,632 dpu	0,393 dpu	0,181 dpu
$FC = 70\%$	0,45 dpu	0,26 dpu	0,14 dpu	0,058 dpu
$FC = 90\%$	0,18 dpu	0,095 dpu	0,049 dpu	0,020 dpu
$FC = 99{,}7\%$	0,058 dpu	0,030 dpu	0,015 dpu	6000 dpm
$FC = 99\%$	0,020 dpu	0,010 dpu	5000 dpm	2000 dpm
$FC = 99{,}7\%$	5000 dpm	3000 dpm	1500 dpm	600 dpm
$FC = 99{,}9\%$	2000 dpm	1000 dpm	500 dpm	200 dpm

Fehleranteil und -überdeckung für Schaltkreise

vor dem Test	$E(\varphi) = 2$	$E(\varphi) = 1$	$E(\varphi) = 0{,}5$	$E(\varphi) = 0{,}2$
Ausbeute	13,5%	36,8%	60,1%	81,9%
$FC = 90\%$	0,18 dpu	0,095 dpu	0,049 dpu	0,020 dpu
$FC = 99{,}7\%$	0,058 dpu	0,030 dpu	0,015 dpu	6000 dpm
$FC = 99\%$	0,020 dpu	0,010 dpu	5000 dpm	2000 dpm
$FC = 99{,}7\%$	5000 dpm	3000 dpm	1500 dpm	600 dpm
$FC = 99{,}9\%$	2000 dpm	1000 dpm	500 dpm	200 dpm

- Gründlich getestete Schaltkreise haben einen Fehleranteil von 100...1000 dpm (grün gekennzeichnet).
- Verlangt etwa eine Ausbeute $Y \approx 1 - DL = e^{-E(\varphi)}$ von 50% und eine Fehlerüberdeckung von $FC \approx 99{,}9\%$.

vor dem Test	$E(\varphi) = 2$	$E(\varphi) = 1$	$E(\varphi) = 0{,}5$	$E(\varphi) = 0{,}2$
Ausbeute	13,5%	36,8%	60,1%	81,9%
$FC = 99{,}7\%$	5000 dpm	3000 dpm	1500 dpm	600 dpm
$FC = 99{,}9\%$	2000 dpm	1000 dpm	500 dpm	200 dpm

Die angestebten Haftfehlerüberdeckungen sind typisch nur $FC_{\text{sa}} \approx 98\%$ und die erzielten Ausbeuten nicht viel größer als 50%. Die Zahlen passen nicht zusammen!

Ansätze für die Forschung:

- Sind die Abschätzungen für den Fehleranteil in der Literatur zu optimistisch?
- Ist der Anteil der nicht nachweisbaren tatsächlichen Fehler eine Zehnerpotenz kleiner als der der Haftfehler und wenn ja, warum?

Fehleranteil von Bauteilen und Baugruppen

- Gründlich getestete Schaltkreise: 100...1000 dpm = $10^{-4} \ldots 10^{-3}$ dpu
  (dpm – defects per million; dpu – defects per unit)
- Rechner aus 10^2 Schaltkreisen mit $DL_{\mathrm{IC}} = 10^{-4}$ dpu:
- Wie viele Rechner enthalten (mindestens) einen defekten Schaltkreis?

$$DL = 1 - \left(1 - 10^{-4}\right)^2 \approx 10^{-2}$$

Jeder hundertste Rechner.

- Vom Herstellertest übersehene Schaltkreisfehler beeinträchtigen die Funktion fast nicht / kaum zu erkennen.
- Ein HW-Fehler im Einsatz auf 100 Rechner ist eine glaubhafte Größenordnung.

Input-Workaround

Alternativen zum Ersatz des Gesamtsystems

Bei einem Fehleranteil nahe eins ist Austausch defekter Objekte unbezahlbar. Alternativen:

- System so nutzen, dass die internen Fehler hinreichend selten Fehlfunktionen verursachen (input work-around).
- Reparatur (z.B. Ersatz von Teilsystemen).

Anfrage-Umformulierung (input work-around):

- Geänderte Anfrage, die im fehlefreien Fall dasselbe Ergebnis liefert (anderen Bedatung, andere Schnittstellen, andere Ausführungsreihenfolge, ...)
- Für Benutzer von IT-Systemen intuitiv erlernte Technik.
- Auch bei langer Systemnutzung ohne Beseitigung erkannter Fehler nimmt die Häufigkeit der Fehlfunktionen dadurch in der Regel ab.

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

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

Andere Beispiele:

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

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

Reparatur

Reparatur

Die Reparaturmöglichkeiten legt der Entwurf fest. Einige Prinzipien des reparaturgerechten Entwurfs:

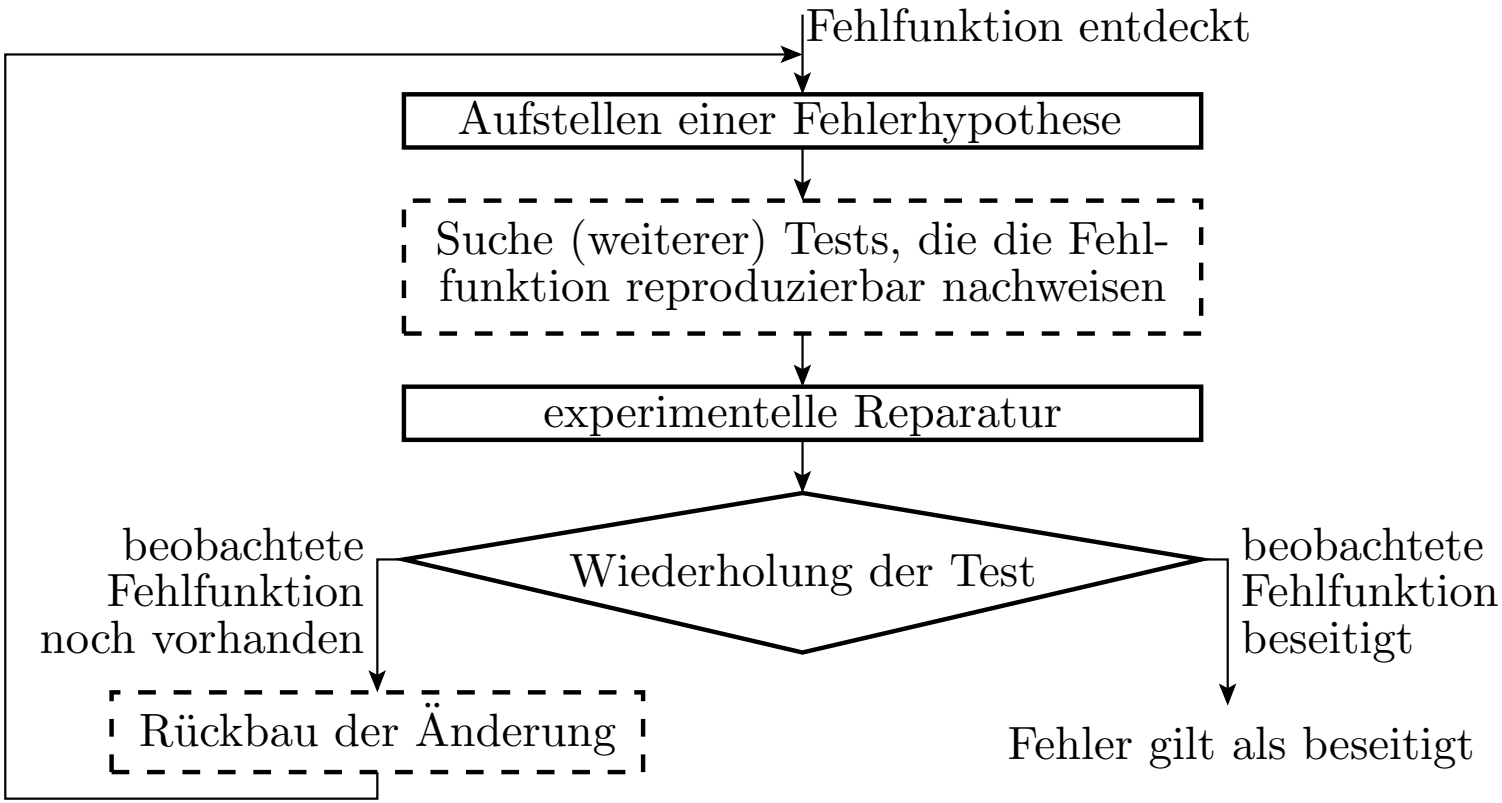
- Zusammensetzten aus austauschbaren Einheiten:
- Software aus Bibliothekselementen, Funktionen, Anweisungen, ...
- Geräte aus Baugruppen, Verbindungen, ...
- Baugruppen aus Bauteilen, ...
- hochintegrierte Schaltkreise: über Programmierelemente abschaltbare Einheiten und zuschaltbare Reserveeinheiten

Unterstützung der Fehlerlokalisierung:

- auftrennbare Steck- und Lötverbindungen, ... (Baugruppen),
- Assertanweisungen, Fehlercodes, ... (Software),
- Einschalttest (Rechner),
- Diagnosebus, lesbarer Fehlerspeicher (Fahrzeugsteuergeräte).

Experimentelle Reparatur

Intuitives iteratives Vorgehen:



Der Reparaturversuch ist die Kontrolle, ob die aufgestellte Hypothese über die Ursache der Fehlfunktion richtig war.

- Jede Fehlerbeseitigungsiteration startet mit einer beobachteten Fehlfunktion.
- Für die Fehlfunktion werden Testbeispiele festgelegt, die sie nachweisen[10].
- Wiederhole, bis die Fehlfunktion nicht mehr auftritt:
  - Ersatz von Einheiten, Reparatur von Verbindungen, ...[11]
  - Erfolgskontrolle mit Testbeispielen.
    - kein Erfolg: (möglichst) Rückbau vorheriger Zustand[12]
    - Erfolg: Iterationsabbruch.

---

[10] Für beim Test bemerkte Fehlfunktionen ist das der Testsatz. Für im Einsatz erkannte Fehlfunktionen müssen diese u.U. erst konstruiert werden.

[11] Aufwandsminimierung: Zusätzliche Tests zur Einschränkung der Fehlermöglichkeiten und damit der zu erwartenden Iterationsanzahl, z.B. Ausschluss von Unterbrechungen oder Kurzschlüssen durch Widerstandsmessungen. Beginn mit einfachen Reparaturmöglichkeiten. ...

[12] Z.B. Ersatz einer geänderten Programmdatei durch ein Backup, wenn die Programmänderung des Fehlersymptom nicht beseitigt hat. Ohne gewissenhaften Rückbau kann sich durch die Reparaturiterationen die Fehleranzahl vergrößern statt abnehmen.

Fakt 2

Eine Fehlerbeseitigung auf Verdacht beseitigt alle erkennbaren Fehler, auch wenn sich der Verdacht, welche Einheit oder Verbindung defekt ist, mehrmals nicht bestätigt.

Für die Fehlerlokalisierung

- genügt eine Erfolgswahrscheinlichkeit weit unter 100%
- wesentlich weniger qualifiziertes Personal als für die Testauswahl.[13]

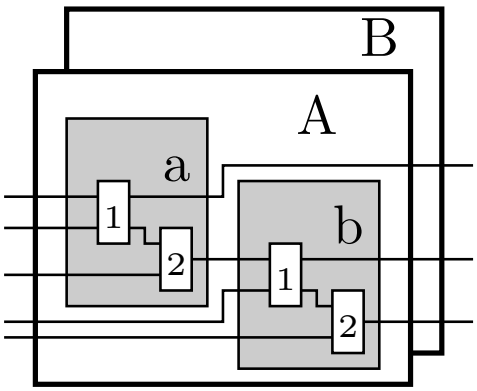
Praktische Lokalisierungstechniken:

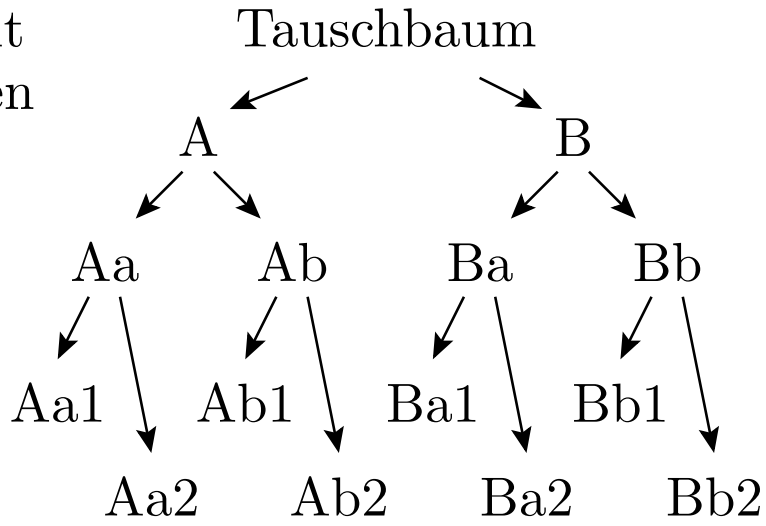
- Systematisches Tauschen.
- Erfahrungsbasierte Reparaturentscheidung.
- Rückverfolgung.

[13] Braucht im Studium deshalb auch nicht unterrichte zu werden.

Systematisches Tauschen

hierarchisches System mit tauschbaren Komponenten

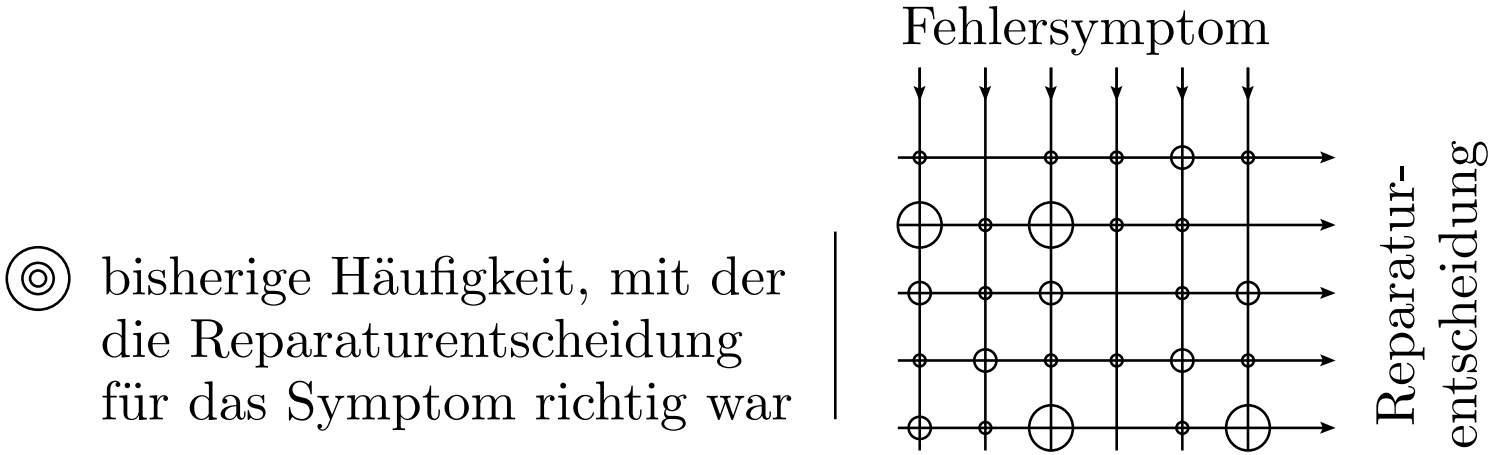




- Nach jedem Tausch, Erfolgskontrolle durch Testwiederholung.
- Ideal: binärer Suchbaum, Tausch der Hälfte, eines Viertel, ... der Komponenten.

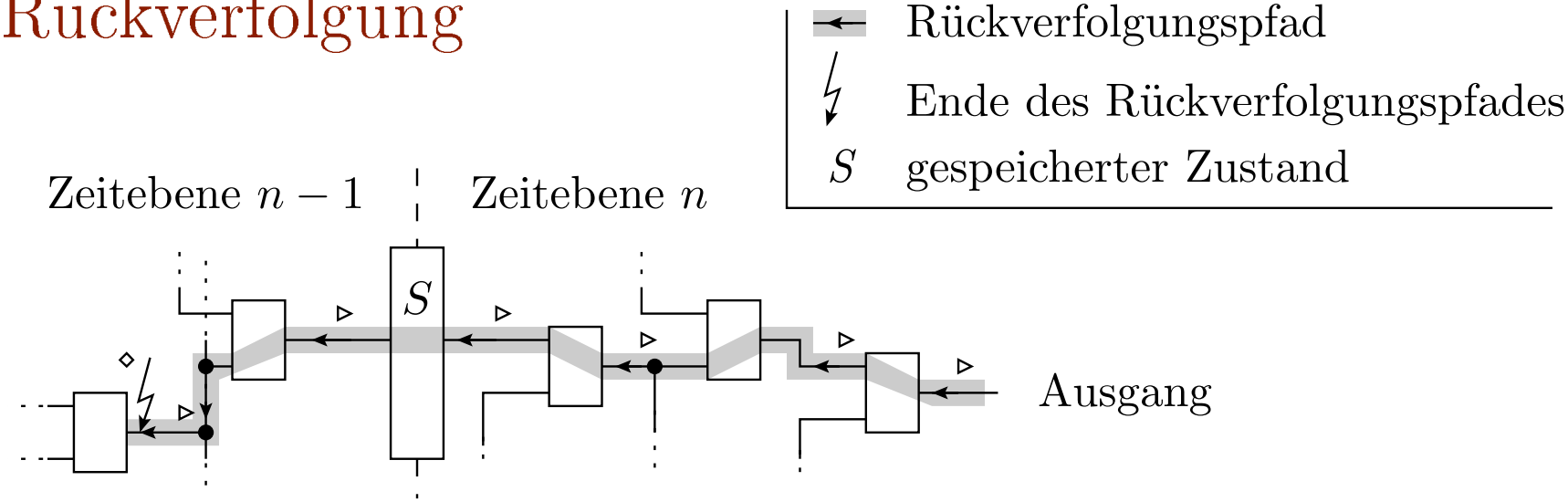
Erfahrungsbasierte Reparaturentscheidung

- Pareto-Prinzip: Produkte haben Schwachstellen. Richtwert: 80% der Probleme geht auf 20% der Ursachen zurück.
- Zählen der erfolgreichen und erfolglosen Reparaturversuche.
- Bei Alternativen, Beginn mit der erfolgsversprechenden Reparaturmöglichkeit.



- Nach erfolglosen Reparaturversuchen Vorzustand wieder herstellen.

Rückverfolgung



- Aufzeichnung der Zeitverläufe potenziell verfälscher Signalverläufe oder Variablenwerte (Simulation, Logikanalyse, Programm-Trace)
- Ausgehend von einer erkannten falschen Ausgabe Rückwärtsuche nach dem Entstehungsort.
- Entstehungsort: Funktionsbaustein, der aus richtigen Eingaben falsche Ausgaben erzeugt.

Am Fehlerort erfahrungsbasierte Reparaturentscheidung:

- Nicht unbedingt der Funktionsbaustein mit richtigen Eingaben und falschen Ausgaben verursacht den Fehler.
- Weitere potenzielle Ursachen:
  - Kurzschluss zu einem anderen Signal, Unterbrechung, defekter Schaltkreiseingang, ... (Baugruppen),
  - Schreiboperation auf eine falsche Variable, ... (Programme).
- Spezielle Fehlerausschlusstests z.B.
  - Widerstandsmessungen zwischen und entlang von Verbindungen.
  - Suche nach falscher Schreiboperation auf die Variable.

Reparatur der Hardware eines PCs

Typisches Mechaniker-Vorgehen:

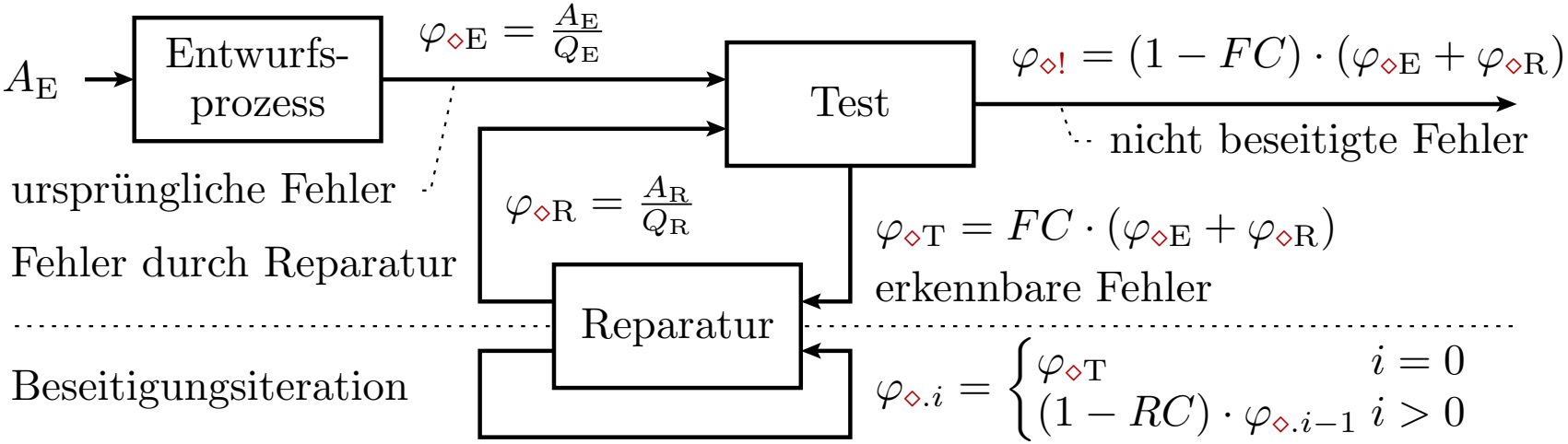
- Grobabschätzung, welcher Rechnerteil defekt sein könnte aus den Fehlersymptomen.
- Kontrolle der Steckverbinder auf Kontaktprobleme durch Abziehen, Reinigen, Zusammenstecken, Ausprobieren.[14]
- Tausch möglicherweise defekter Baugruppen gegen Ersatzbaugruppen, Ausprobieren, ...

Voraussetzungen:

- Wiederholbare Tests, die den Fehler nachweisen.
- Ausreichend Ersatzteile.
- Verlangt nur allgemeine Mechnikerkenntnisse, aber keine Kenntnis der Funktionsweise des zu reparierenden Systems.

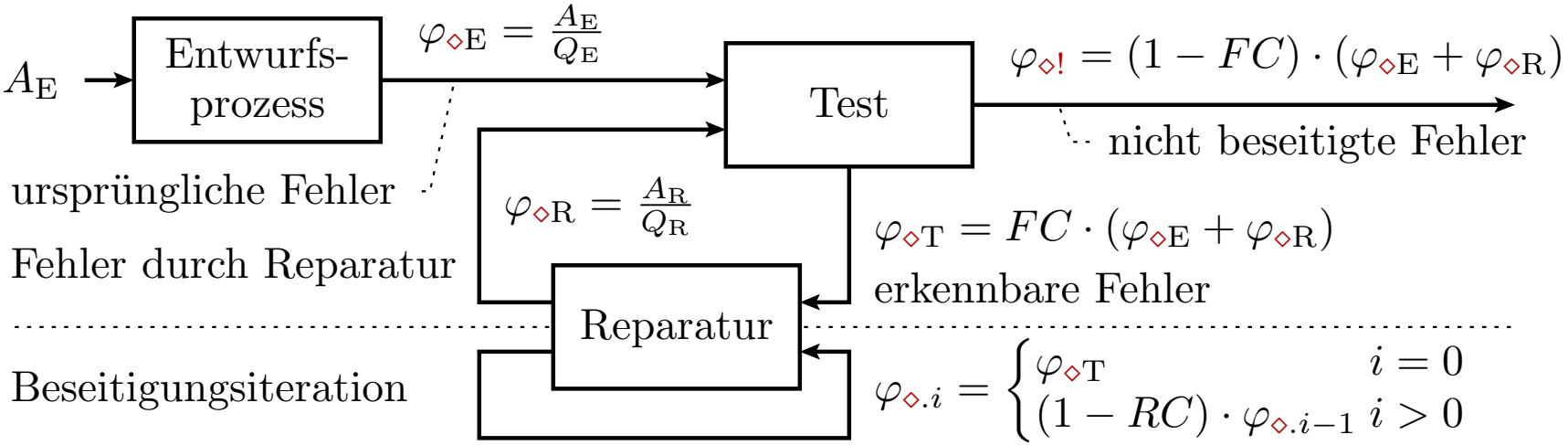
[14] Ein cleverer Mechaniker bauen getauschte vermutlich ganze Teile statt in denselben, in einen anderen Rechner ein. Warum sollte das unterbleiben?

Abschätzung der Anzahl nicht beseitigten Fehler



A_E	Entwurfsaufwand	Q_R	Güte des Reparaturprozesses
A_R	Reparaturaufwand	FC	Fehlerüberdeckung des Tests
Q_E	Güte des Entwurfsprozesses	RC	Erfolgsrate der Reparatur

- Die Fehleranzahl sei proportional zum Aufwand und umgekehrt proportional zur Prozessgüte.
- Fehler, die der Test durchlässt, erreichen nie die Reparatur
- Nachweisbare Fehler werden iterativ beseitigt.
- Je mehr Beseitigungsiterationen, desto mehr neue Fehler.



- Fehleranzahl je Beseitigungsschritt $\Rightarrow$ geometrische Reihe:

$$\varphi_{\diamond .i} = (1 - RC)^i \cdot \varphi_{\diamond T}$$

- Anzahl der Reparaturen gleich Summe aller $\varphi_{\diamond .i}$:

$$A_R = \sum_{i=0}^{\infty} \varphi_{\diamond .i} = \varphi_{\diamond T} \cdot \sum_{i=0}^{\infty} (1 - RC)^i = \frac{\varphi_{\diamond T}}{RC}$$

- Anzahl der durch Reparaturen verursachten Fehler

$$\varphi_{\diamond R} = \frac{\varphi_{\diamond T}}{Q_R \cdot RC} = \frac{FC \cdot (\varphi_{\diamond E} + \varphi_{\diamond R})}{Q_R \cdot RC}$$

$$\varphi_{\diamond \mathrm{R}} = \frac{\varphi_{\diamond \mathrm{T}}}{Q_\mathrm{R} \cdot RC} = \frac{FC \cdot (\varphi_{\diamond \mathrm{E}} + \varphi_{\diamond \mathrm{R}})}{Q_\mathrm{R} \cdot RC}$$

$$Q_\mathrm{R} \cdot RC = \frac{\text{Anz. Reparaturen}}{\text{neue Fehler}} \cdot \frac{\text{beseitigte Fehler}}{\text{Anz.Reparaturen}} = \frac{\text{beseitigteFehler}}{\text{neue Fehler}}$$

- Wenn die Anzahl der nachweisbaren Fehler abnimmt:

$$Q_\mathrm{R} \cdot RC > FC$$

- Gesamtanzahl der durch Reparatur verursachen Fehler:

$$\varphi_{\diamond \mathrm{R}} = \varphi_{\diamond \mathrm{E}} \cdot \frac{FC}{Q_\mathrm{R} \cdot RC - FC}$$

- Ursprüngliche + reparaturbedingte Fehler

$$\varphi_{\diamond \mathrm{E}} + \varphi_{\diamond \mathrm{R}} = \varphi_{\diamond \mathrm{E}} \cdot \left(1 + \frac{FC}{Q_\mathrm{R} \cdot RC - FC}\right) = \varphi_{\diamond \mathrm{E}} \cdot \frac{Q_\mathrm{R} \cdot RC}{Q_\mathrm{R} \cdot RC - FC}$$

- Gesamtanzahl der nicht beseitigten Fehler $(\ldots \cdot (1 - FC))$

$$\boxed{\varphi_{\diamond !} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-FC) \cdot Q_\mathrm{R} \cdot RC}{Q_\mathrm{R} \cdot RC - FC}}$$

Idealer Reparaturprozess

- Es werden viel mehr Fehler beseitigt als neu eingebaut:

$$Q_\mathrm{R} \cdot RC = \frac{\text{beseitigteFehler}}{\text{neue Fehler}} \gg 1$$

Fehlerbeseitigung ist so gut wie der Test

$$\varphi_{\diamond !} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-FC) \cdot Q_\mathrm{R} \cdot RC}{Q_\mathrm{R} \cdot RC - FC} \approx (1-FC) \cdot \varphi_{\diamond \mathrm{E}}$$

Wenn im Mittel für jeden beseitigten Fehler ein neuer eingebaut wird: $Q_\mathrm{R} \cdot RC = 1$

$$\varphi_{\diamond !} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-FC) \cdot Q_\mathrm{R} \cdot RC}{Q_\mathrm{R} \cdot RC - FC} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-FC) \cdot 1}{1-FC} = \varphi_{\diamond \mathrm{E}}$$

- Werden alle nachweisbaren Fehler beseitigt und
- entstehen etwas genauso viele nicht nachweisbare Fehler.

Typische studentische Programmierarbeiten

$$\boxed{\varphi_{\diamond !} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-FC)\cdot Q_\mathrm{R}\cdot RC}{Q_\mathrm{R}\cdot RC - FC}}$$

Fall A: wenige Testbeispiele, brauchbarer Reparaturprozess

- Beispiel: $FC = 30\%$ erkennbare Fehler, $Q_\mathrm{R} \cdot RC = 2$ beseitigte je neuer Fehler

$$\varphi_{\diamond !} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-0.,3)\cdot 2}{2-0.,3} \approx 82\% \cdot \varphi_{\diamond \mathrm{E}}$$

- Reduktion der Fehleranzahl auf 82%. Davon sind 70% nicht erkannte ursprüngliche und $12\% \cdot \varphi_{\diamond \mathrm{E}}$ bei der Reparatur entstandene Fehler.
- Erkannt und beseitigt werden die am meisten störenden Fehler (siehe Zufallstest). Es bestehen Chancen, dass das System einen Abnahmetest mit 1 bis 2 neuen zufälligen Testbeispielen erfolgreich passiert.

$$\boxed{\varphi_{\diamond !} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-FC)\cdot Q_\mathrm{R}\cdot RC}{Q_\mathrm{R}\cdot RC - FC}}$$

Fall B: Entwurf wird beherrscht, aber Test und Reparaturtechniken nicht

- Beispiel: $FC = 25\%$ erkennbare Fehler, $Q_\mathrm{R} \cdot RC = 0{,}5$ beseitigte je neuer Fehler

$$\varphi_{\diamond !} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-0,25)\cdot 0,5}{0,5-0,25} = 2{,}5 \cdot \varphi_{\diamond \mathrm{E}}$$

- System enthält nach Test und Fehlerbeseitigung viel mehr Fehler als zuvor.
- Reparatur versteckt die Fehler (ersetzt alle erkennbaren durch nicht erkennbare Fehler); Testbeispiele entwertet
- Begleitsymtom: übermäßig lange Test- und Reparaturphase.
- Abnahmetest mit 1 bis 2 neuen zufälligen Testbeispielen findet in der Regel Fehler.

$$\boxed{\varphi_{\diamond !} = \varphi_{\diamond \mathrm{E}} \cdot \frac{(1-FC)\cdot Q_\mathrm{R}\cdot RC}{Q_\mathrm{R}\cdot RC - FC}}$$

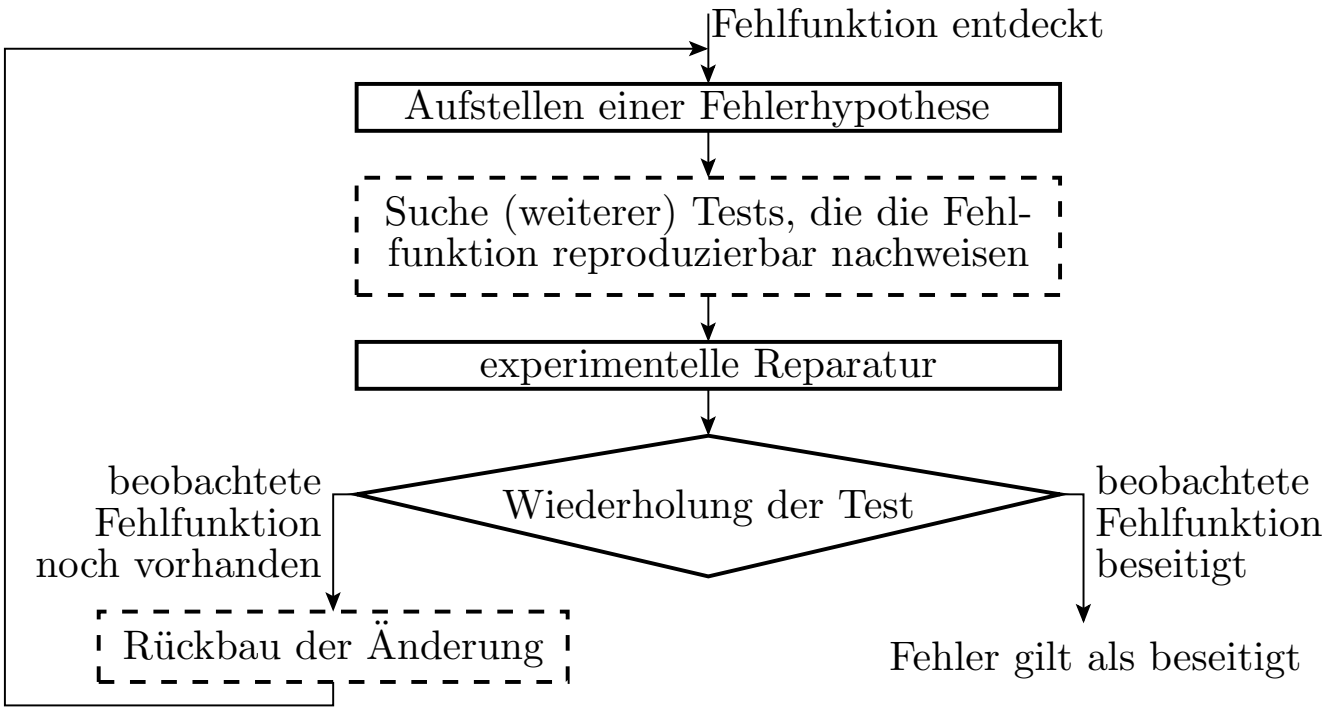
Fall C: Studierender ist mit seiner Aufgabe überfordert

$$FC > Q_\mathrm{R} \cdot RC$$

- Zunahme der Anzahl der nachweisbaren Fehler mit Fortschreiten der Beseitigungsversuche.
- Projekt wird nie fertig.

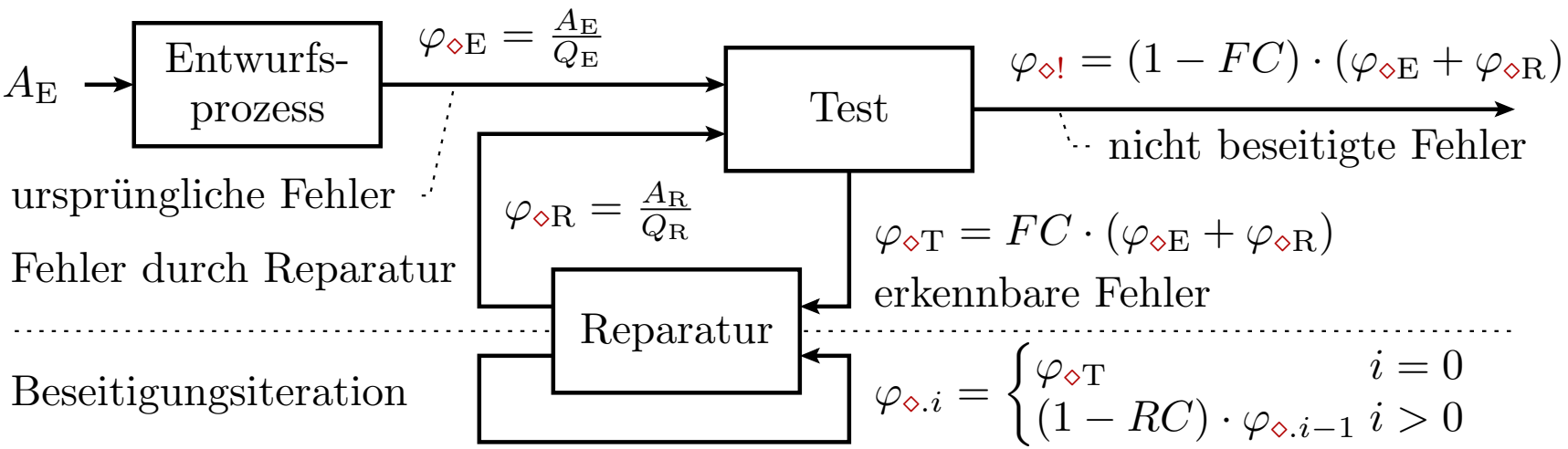
Vor der Übertragung einer Entwicklungsaufgabe sollte der personengebundenen Parameter $Q_\mathrm{R} \cdot RC$ (beseitigte Fehler je neuer Fehler) kontrolliert und, wenn schlecht, Einarbeitungszeit verlängert werden.

Reparaturempfehlungen



Zur Minimierung der Iterationszahl:

- Ausschluss von Hypothesen durch zusätzliche Tests.
- Reparaturentscheidungen so wählen, dass bei Nicht-Erfolg weitere Hypothesen ausgeschlossen werden. ...



Fehlervermeidung bei der Reparatur:

- Sorgfältig reparieren (hohe Güte Q_R und hohe Erfolgsrate RC des Reparaturprozesses).
- Sorgfältiger Rückbau. (Beseitigt auch entstandene Fehler, die der Test nicht erkennt.)
- Zahl der erfolglosen Reparaturversuche je Fehler begrenzen, z.B. auf drei, dann Ersatz.
- Von Systemen mit sehr vielen Änderungen nur die Zielfunktion und die Testbeispiele übernehmen. System selbst neu entwerfen.

Regel für den Einkauf von IT-Systemen:

- Zur Kontrolle, ob der Hersteller eine akzeptable Prüf- und Reparaturtechnologien hat, neu angeschaffte IT-System mit einer Stichprobe zufälliger Eingaben testen.
- Wenn ein System diesen Test nicht besteht, zurückgeben und von diesem Hersteller nie wieder etwas kaufen. Denn das ist ein sicherer Hinweis darauf, dass dieser Hersteller seine Entwurfs- und Reparaturprozesse nicht beherrscht.

Akzeptiere nie ein IT-System, ohne vorher selbst gewählte, dem Entwerfer unbekannte Tesetbeispiele auszuprobieren.

Aufgaben

Aufgabe 5.1: Fehleranteil

1. Die zu erwartende Fehleranteil eines Systemtyps sei vor dem Test 0,7 dpu und nach dem Test 0,01 dpu. Welche Fehlerüberdeckung hat der Testsatz unter der Annahme, dass die Fehleranzahl vor und nach dem Test poison-verteilt ist?
2. Ein Schaltkreistest hat 60% aller gefertigten Schaltkreise als fehlerhaft aussortiert. Aus dem Garantierückläufen seitens der Anwender wurde für die getesteten Schaltkreise ein Fehleranteil von 200 dpm abgeschätzt. Auf welche Ausbeute und auf welche Fehlerüberdeckung lassen diese Zahlen schließen, wenn eine poisson-verteile Fehleranzahl unterstellt wird?

Aufgabe 5.2: Input Workaround

1. Beschreiben Sie in der Programmiersprache C einen Work-Around zur Ausführung der Anweisung

```
int a, b, c;
...
a = b * c;
```

für einen Rechner, der bei einer Multiplikation mit null aufgrund eines Fehlers eins berechnet so, dass das Programm auch funktioniert, wenn der Fehler in späteren Rechnergenerationen nicht mehr vorhanden ist.

2. Welche Dokumente eines Software-Prototypentwurfs, an denen sehr viel geändert und in dem bereits sehr viele Fehler beseitigt wurden, sollten weiterverwendet und welche neu geschrieben werden?

Aufgabe 5.3: Fehlerlokalisierung

Für die nachfolgende Schaltung wurden die Signalwerte in der Tabelle während des Tests aufgezeichnet.

1. Welche Bauteile oder Verbindungen könnten die Ursache der Fehlfunktion sein?
2. Wie ließen sich die Fehlermöglichkeiten vor dem ersten Reparaturversuch weiter einschränken?

Aufgabe 5.4: Reparaturprozess

Ein Student macht angenommen beim Programmieren im Mittel 5 Fehler auf Hundert Programmzeilen, die weder vom Syntaxtest noch von den gewählten Testbeispielen erkannt werden. Auf jeden nicht erkannten Fehler kommt im Mittel ein Syntaxfehler und ein erkannter semantischer Fehler. Die Beseitigung eines erkannten Fehlers erfordert im Mittel drei Versuche. Bei jedem zweiten Versuch entsteht ein neuer Fehler. Von den neuen Fehlern wird ein Drittel vom Syntaxtest, ein Drittel von den Testbeispielen und ein Drittel nicht erkannt. Wie hoch ist die zu erwartende Fehleranzahl nach Beseitigung aller erkennbaren Fehler eines 200 Codezeilen großen Programms?

Aufgabe 5.5: Fehlerbeseitigungsiteration

Wie groß ist die zu erwartende Fehleranzahl nach einer Iteration aus Test und Fehlerbeseitigung?

- Fehleranzahl vor der Fehlerbeseitigungsiteration 20;
- Fehlerüberdeckung $FC = 60\%$ und
- im Mittel einem neu entstehenden Fehler je fünf Beseitigungsversuche.

Unter welcher Bedingung nimmt die Anzahl der nachweisbaren Fehler in einer Fehlerbeseitigungsiteration denoch ab, wenn bei jedem erfolgreichen Beseitigungsversuch im Mittel 1,2 neue Fehler in das System eingebaut werden?

Aufgabe 5.6: Fehleranteil Baugruppen

1. Eine Leiterplatte wird aus folgenden Bauteilen mit bekanntem Fehleranteil zusammengesetzt. Wie hoch ist der Fehleranteil der Baugruppe nach Beseitigung aller Verbindungsfehler?

Typ	Anzahl	Fehleranteil
Widerstände, Kondensatoren, ...		
Schaltkreise		
Steckkontakte		
Leiterplatte		

2. Begründen Sie aus Reparatursicht, warum auf Baugruppen auch manchmal Teilbaugruppen als vorgefertigte, seperat testbare Module gesteckt werden.

Aufgabe 5.7: Fehleranteil Rechner

Ein Rechner besteht aus Leiterplatten, Schaltkreisen, diskreten Bauteilen (Widerstände, Kondensatoren, ...) und Lötstellen.

Typ	Anzahl	$E(DL_{\mathrm{BT}})$
Leiterplatten	10	10 dpm
Schaltkreise	100	200 dpm
diskrete Bauteile	200	10 dpm
Lötstellen	10000	1 dpm

Fehleranteil des gesamten Rechners [15]?

[15] Für die Bauteile und den kompletten Rechner sei unterstellt, dass die Fehleranzahl poisson-verteilt ist.

Aufgabe 5.8: Objektüberdeckung, Ausbeute, ...

- Ist die Objektfehlerüberdeckung größer oder kleiner als die Fehlerüberdeckung?
- Die Objektfehlerüberdeckung sei 50% und die Ausbeute 80%. Wie hoch ist der Fehleranteil, wenn die Fehleranzahl Poisson-verteilt ist?
- Für einen bestimmten Schaltkreistyp, z.B. Speicherschaltkreise sei die Ausbeute 80%. Um welchen Faktor kann die Chipfläche vergrößert werden, damit die Ausbeute nicht unter 20% absinkt? Annahmen: Die Fehleranzahl sei Poisson-verteilt und verhalte proportional zu Chip-Fläche.

Testumgebungen

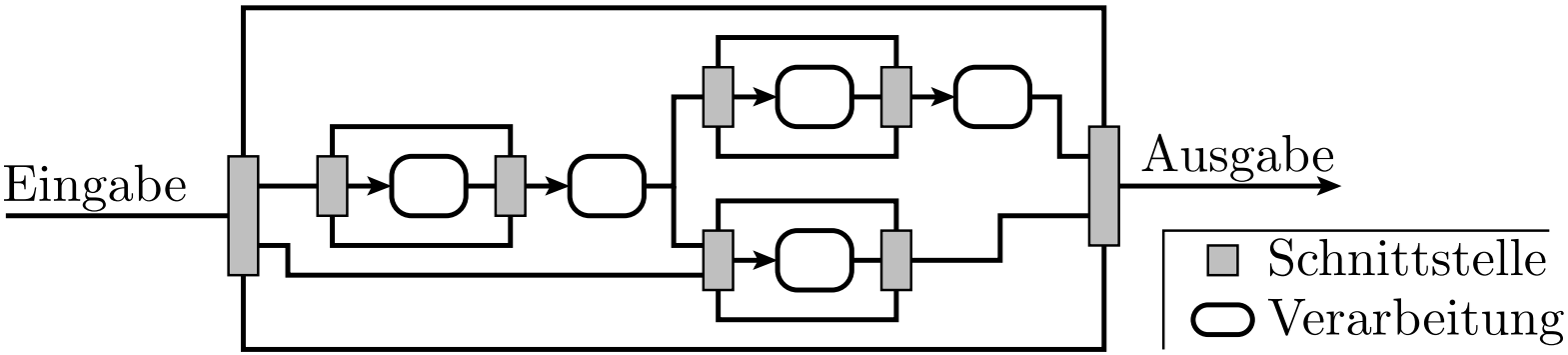
Testumgebungen

Zur Durchführung eines dynamischen Tests sind Eingaben bereitzustellen und die Ergebnisse zu kontrollieren. Für Software-Module erfordert das nur einen Rechner und ein Programm (Testrahmen), in den das Testobjekt eingebunden ist. Interaktive Programme brauchen eine kompliziertere Testumgebung, im ungünstigsten Fall einen Bediener. Für den ganzheitlichen Test reaktiver Systeme sind Teile der Umgebung nachzubilden bis hin zur Simulation des gesteuerten Objekts (Antrieb, Fahrzeug, ...). Hardware verlangt oft Spezialtester, die in Echtzeit Testeingaben bereitstellen und Ergebnisse kontrollieren.

Modultest

Modultest

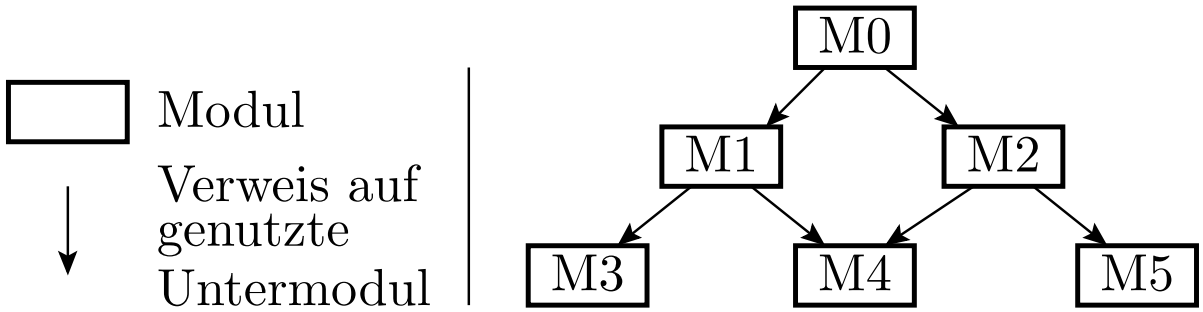
IT-Systeme sind modular aufgebaut. Module sind die kleinsten sinnvoll testbaren Systembausteine. Nach dem Service-Modell hat jedes Modul Schnittstellen für die Ein- und Ausgabe und eine interne Funktion.



Für den Modultest wird jedes Modul aus dem Kontext gelöst, in einen Testrahmen eingebettet, der Testeingaben bereitstellt und die Testausgaben kontrolliert. Die Modularisierung legt der Entwurf fest und ist fundamental für die Testbarkeit.

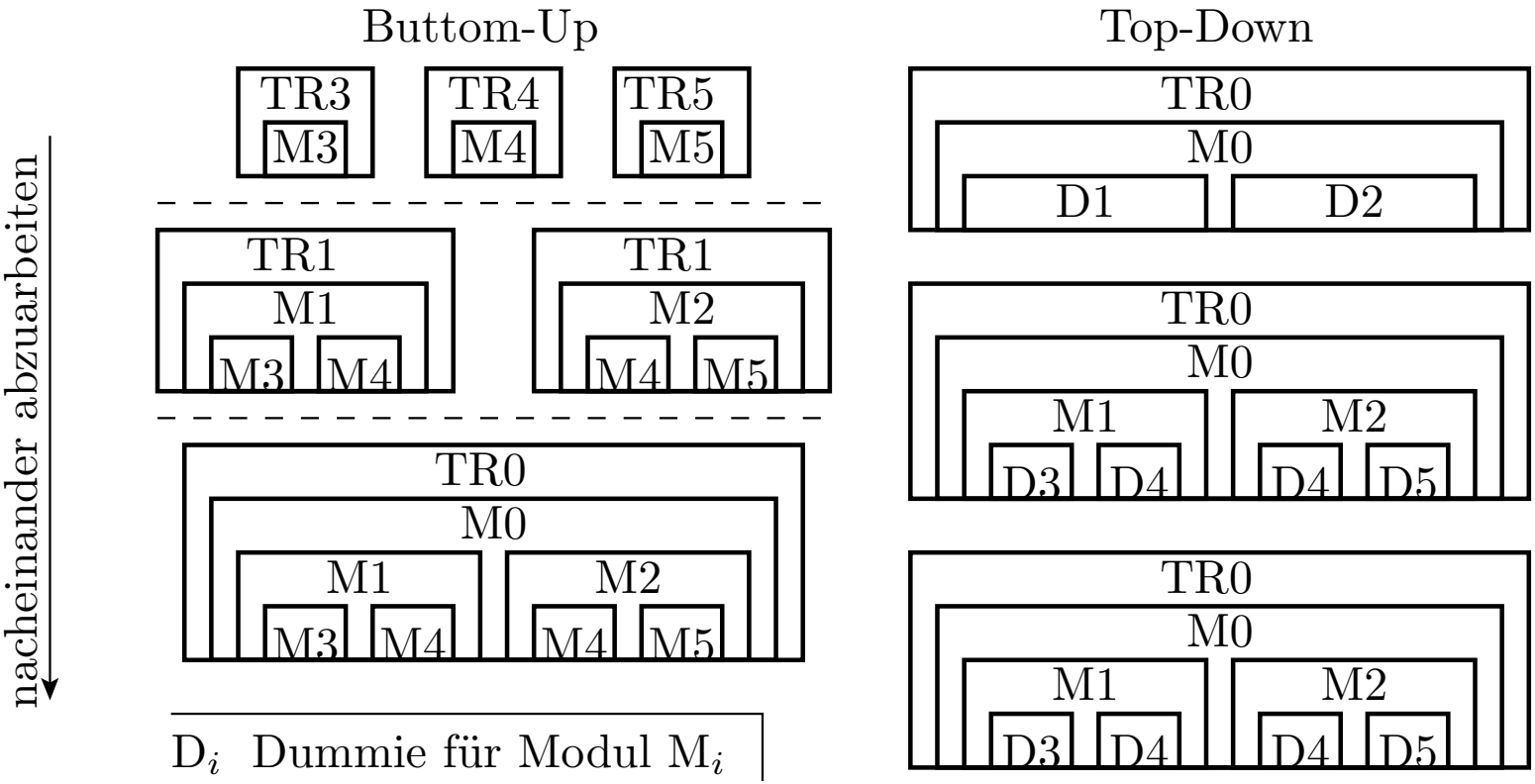
Entwurfs- und Testreihenfolge

Software hat eine hierarchische Aufrufstruktur, in der das Hauptprogramm Unterprogramme nutzt, die wiederum Unterprogramme nutzen.



Strategien für die Entwurfs- und Testreihenfolge:

- Bottom-Up: Beginn mit dem Entwurf und Test der untersten Module. Test der übergeordneten Module mit den bereits getesteten Untermodulen.
- Top-Down: Beginn mit dem Entwurf übergeordneter Module und Test mit Dummies für die Untermodule. Schrittweise Ersatz der Dummies durch getestete Untermodule.



Der Buttom-Up-Entwurf verlangt für jedes Modul einen eigenen Testrahmen. Der Top-Down-Entwurf kommt mit einem Testrahmen für oberste Modul aus, verlangt aber die Entwicklung von Dummies für die zu Beginn noch nicht eingebundenen Module.

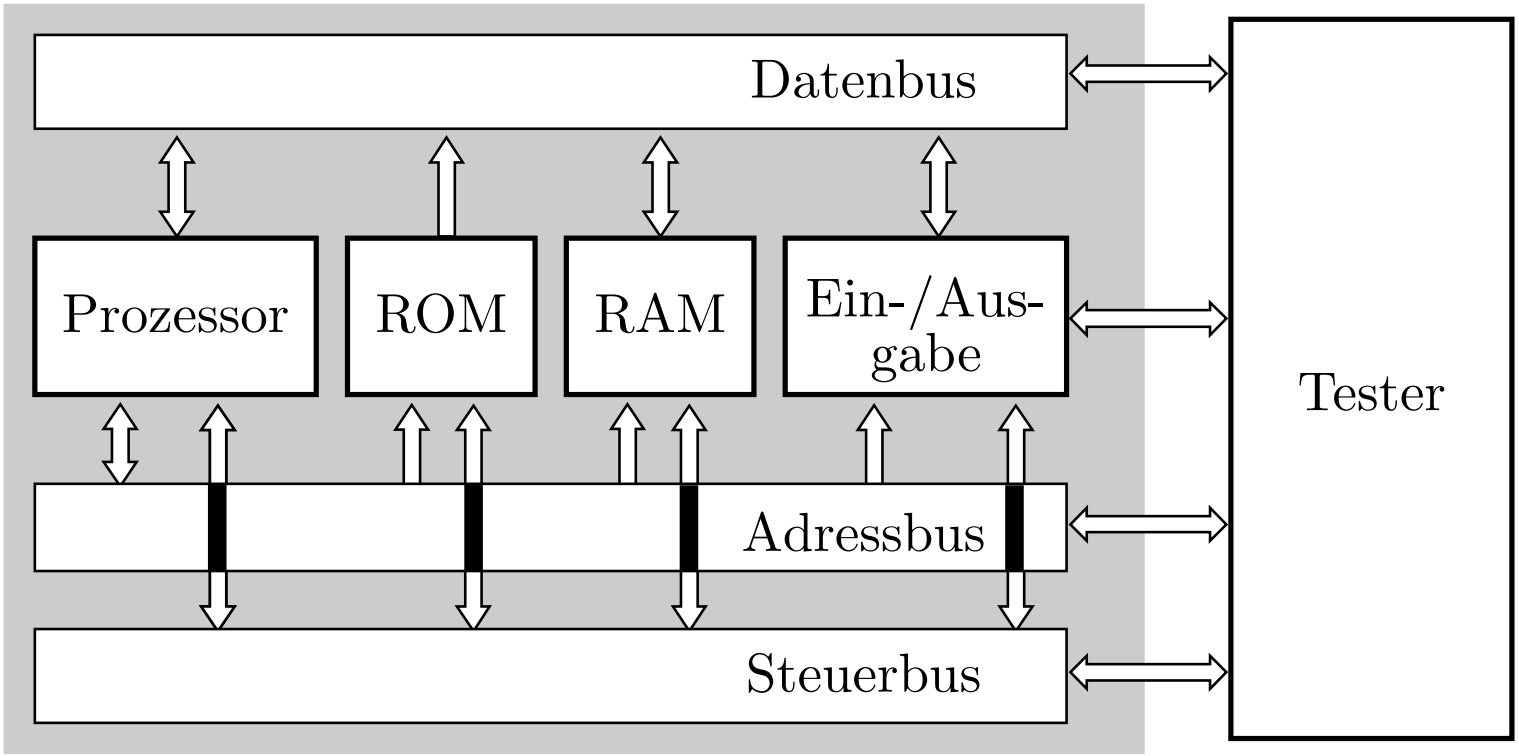
Testrahmen für Software

Für Software-Module und Hardware-Simulationsmodelle ist der Testrahmen ein Programm, dass die Eingaben bereitstellt und die Ausgaben auswertet oder aufzeichnet. Beim Top-Douw-Entwurf kommen die Dummies hinzu, die die noch nicht entworfenen Untermodule ersetzen.

Im Prinzip lassen sich die Testrahmen bis auf die Bedatung und Sollwerte automatisch aus den Schnittstellenbeschreibungen generieren. Sprachen wie VHDL unterstützen das. Konstrukte wie die Datenübergabe mit Zeigern oder über globale Variablen erschweren die Testrahmenprogrammierung.

Testfälle, -daten, Sollwerte, ... möglichst getrennt vom Testrahmen in einer Datei oder Datenbank abspeichern.

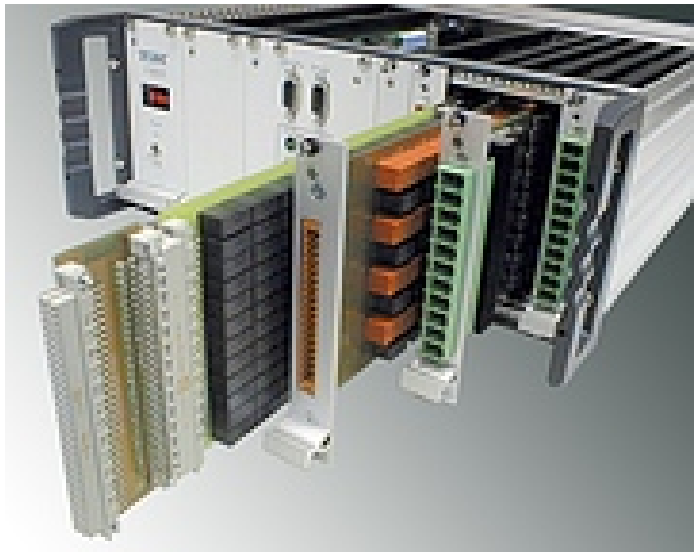
Modularer Hardware-Test



Der modulare Hardware-Test verlangt Entwurfsvorkehrungen, die dem Tester Zugriff auf die Schnittstellen zwischen dem einzelnen Teilsystemem erlauben. Für den Tester zugängliche Busstrukturen sind dafür besonders geeignet.

Funktionstester

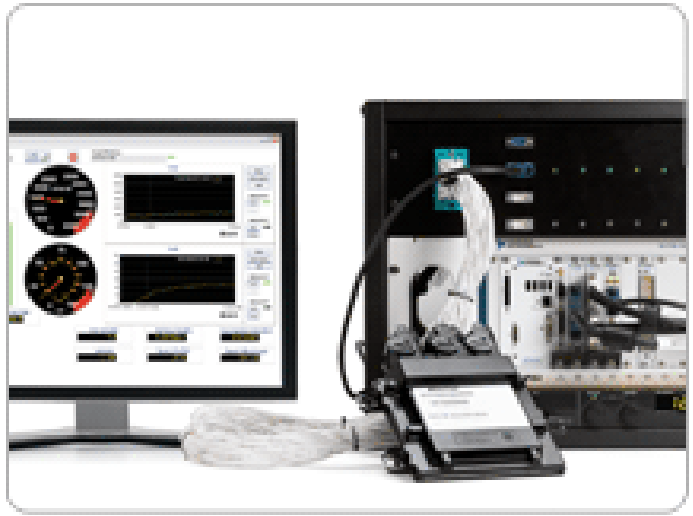
Modularer Funktionstester



Der komplette Test reaktiver Systeme verlangt die Bereitstellung von Testverläufen für die Sensorsignale und die Kontrollen der Ausgabesignale. Typische Lösung ist ein Rechner mit einem modular zusammensetzbaren System aus

- Logikgenerator- und Logikanalysatorbaugruppen,
- DAU- und ADU-Baugruppen,
- programmierbaren Spannungsversorgungen,
- Baugruppen für Busschnittstellen (RS232, SPI, CAN, ...),
- Lastschaltungen, Adapter, ...

HIL- (Hardware in the Loop) Tester



Nachbildung der Systemumgebung (Antrieb, Sensoren, Prozesse, komplette Fahrzeuge, ...) physikalisch, als Simultionsmodelle oder gemischt. Abarbeitung/Simulation echter Betriebssituationen als Tests.

#

Einfaches Beispiel:

<Bild>

Testobjekt ist ein Regler mit der Schrittfunktion ... (als Matlab-Funktion)

HIL- (Hardware in the Loop) Tester

Nachbildung der Systemumgebung physikalisch, als Simultionsmodell oder gemischt. Maschinen und Anlagenbau:

- Physikalische Simulation der gesteuerten Maschine oder Anlage,
- 3D-Visualisierung des physikalischen Verhaltens,
- Untersuchung von Grenzwert- und Gefahrensituationen.

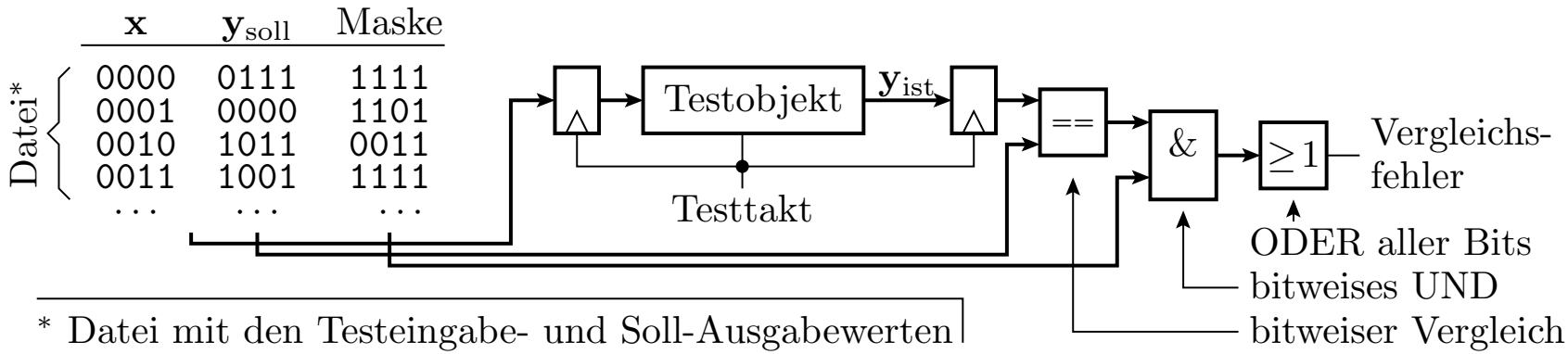
Fahrzeugbau, Luft- und Raumfahrt

- physikalische Simulationen von Motoren, Lenksystemen bis hin zu kompletten Flugzeugen,
- Nachstellung komplizierter Testsituationen im Labor (fahrendes Auto, Flugzeug in der Luft, ...)

Jedes Simulationsmodell hat Genauigkeitsgrenzen. Kein vollständiger Ersatz für den Test in der Anwendungsumgebung

Schaltungstester

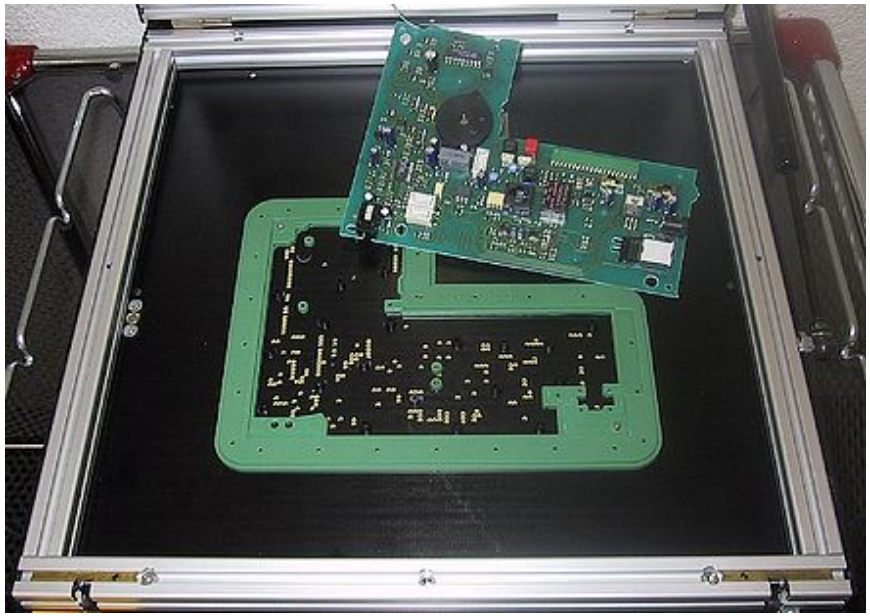
Prinzip eines Digitaltesters



Das Testprogramm beschreibt:

- Eingabebitvektoren
- Sollausgaben
- Maskenbitvektoren: Festlegung der zu kontrollierenden (gültigen) Bitwerte je Testschritt
- Testtakt zur Festlegung der Zeitpunkte der Eingabesignalwechsel und der Ausgabeabtastung

Baugruppentester



Elektronische Baugruppen werden für den Test meist auf ein Nadelbett gelegt und mit Unterdruck angesaugt. Über die Nadeln sind Prüfgeräte angeschlossen. Die Kontaktierung schaltungsinterener Punkte erlaubt einen modularen Test und eine Lokalisierung von Kurzschlüssen, Unterbrechungen und defekten Baussteinen.

Weitere Unterteilung:

- MDA (manufacturing defect analyzer),
- ICT (in-circuit tester).

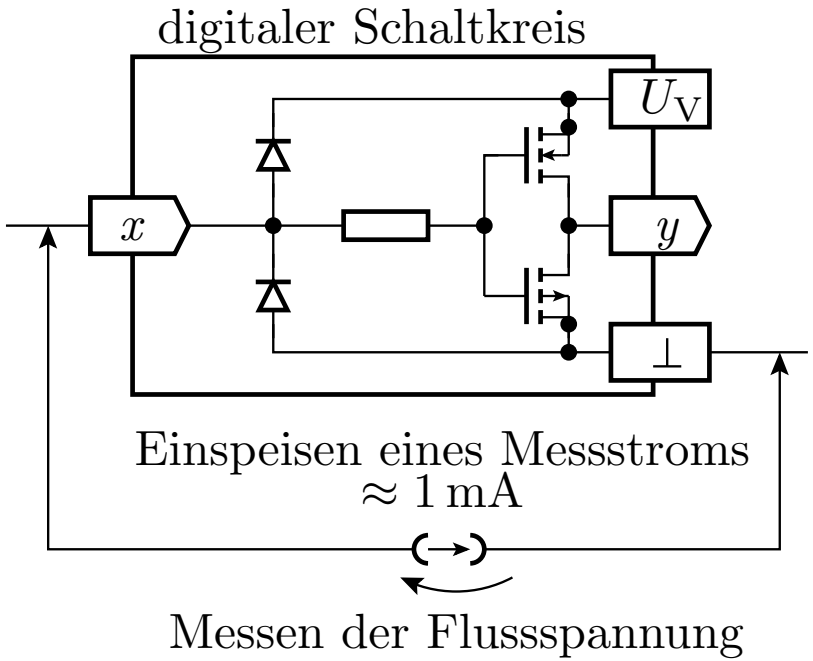
MDA

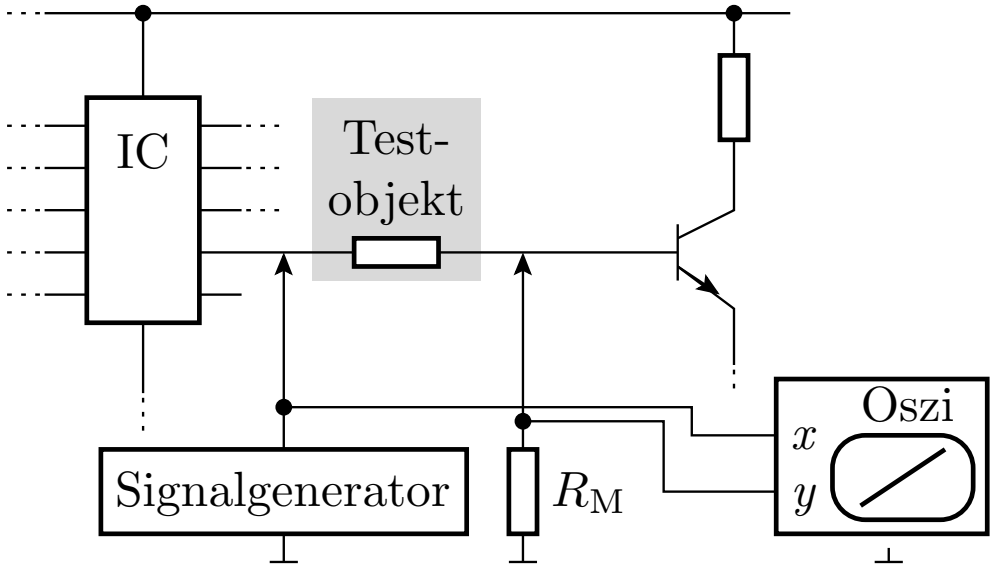
Suche potenzieller Bestückungs- und Verdrahtungsfehler mit elektrischen Messungen, in der Regel zwischen zwei Punkten:

- Stromeinspeisung und Messung der Spannung,
- Spannungseinspeisung und Strommessung.

Bauteiltypische Strom-Spannungsbeziehungen:

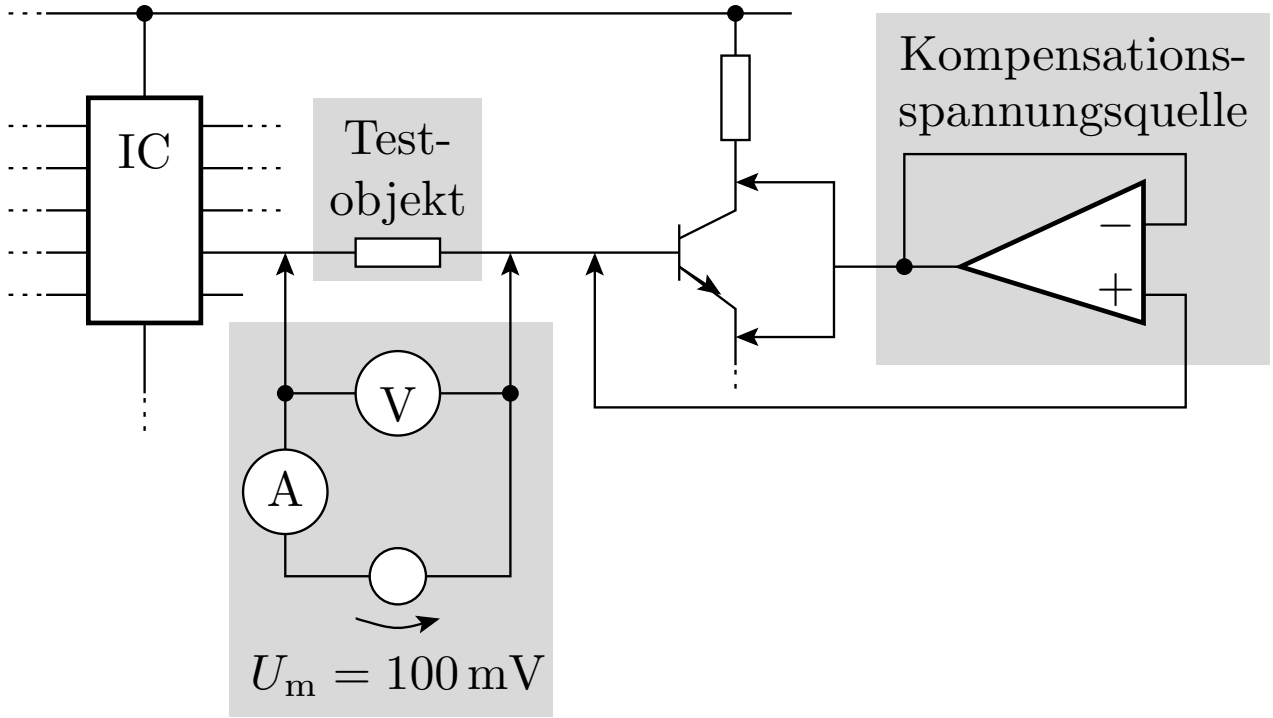
- Widerstand: Gerade,
- Kondensator: Elypse,
- Diode: Kennlinie mit Knick,
- Schaltkreise: Ausmessen der Schutzdioden,
- Unterbrechung: kein Strom,
- Kurzschluss: kein Spannungsabfall,
- ...





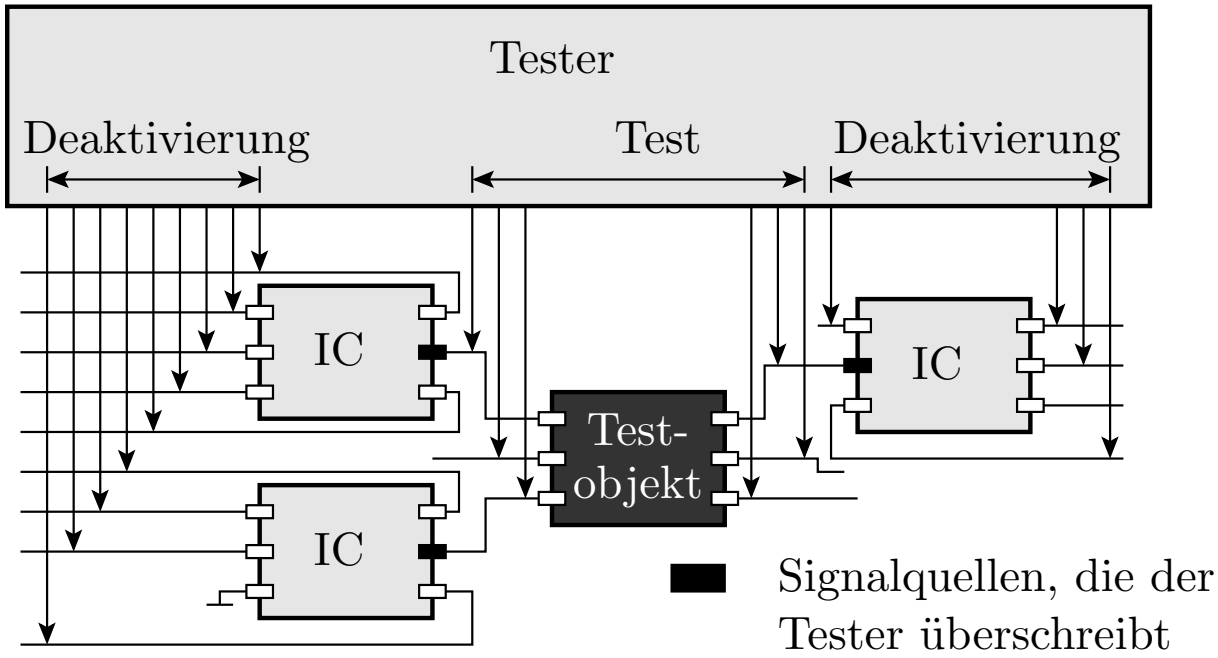
- Die Strom-Spannungs-Beziehung zwischen zwei Punkten in einer Schaltung hängt von der kompletten Schaltung, nicht nur einem einzelnen Bauteilen.
- Bestimmbar durch Ausprobieren an einem »Golden Device«.
- Problematisch kann sein die
  - die Festlegung der Toleranzbereiche für die Signatur aus den Bauteilstreuungen,
  - die Erkennungssicherheit, z.B. Fehlbestückungen bei kleinen Kondensatoren.

Analoger In-Circuit Test



Unterdrückung von Parallelströmen zum Testobjekt durch Kompensation der Spannungsabfälle über den wegführenden Bauteilen auf einer Testobjektseite auf null. Erlaubt einen isolierten Zweipoltest.

Digitaler In-Circuit-Tester



- Überschreiben digitaler Eingabesignale des Testobjekts mit stromstarken Treibern.
- Andere Schaltkreise werden möglichst deaktiviert (Anschlüsse hochohmig).
- Erlaubt isolierten Bestückungstest der Schaltkreise.

Voraussetzung für ICT-Einsatz »prüfgerechter Entwurf«:

- Bei Vakuumansaugen luftdichter Rand, keine Löcher.
- Geeignete Kontaktflächen.
- Deaktivierungsmöglichkeit der Schaltkreise, ...

Automatisch Generierung der Testvorschrift möglich:

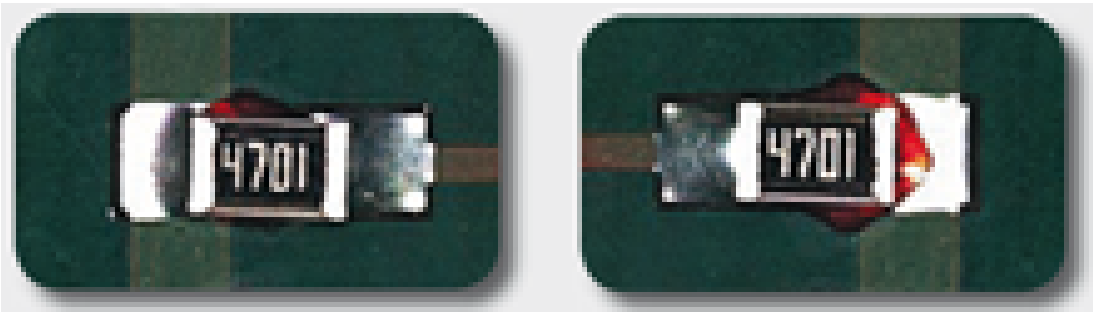
- Zusammensetzen aus Test- und Deaktivierungsvorschriften für alle Bauteile (gut gestellt).

Fehlererkennung und Lokalisierung:

- Erkennt fast alle Kurzschlüsse, Unterbrechungen und Fehlbestückungen und gibt den genauen Fehlerort an.

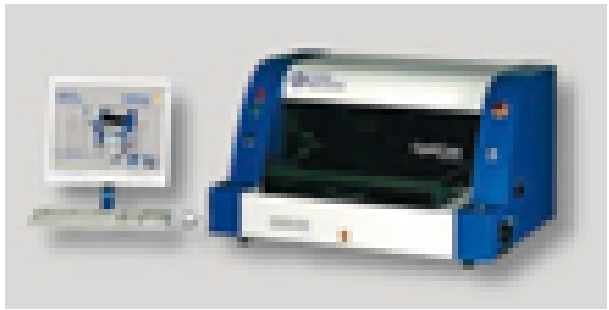
Optische Inspektion

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



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, so dass der Hersteller auch für alle Folgeschäden, z.B. Unfallschaden, haftet.

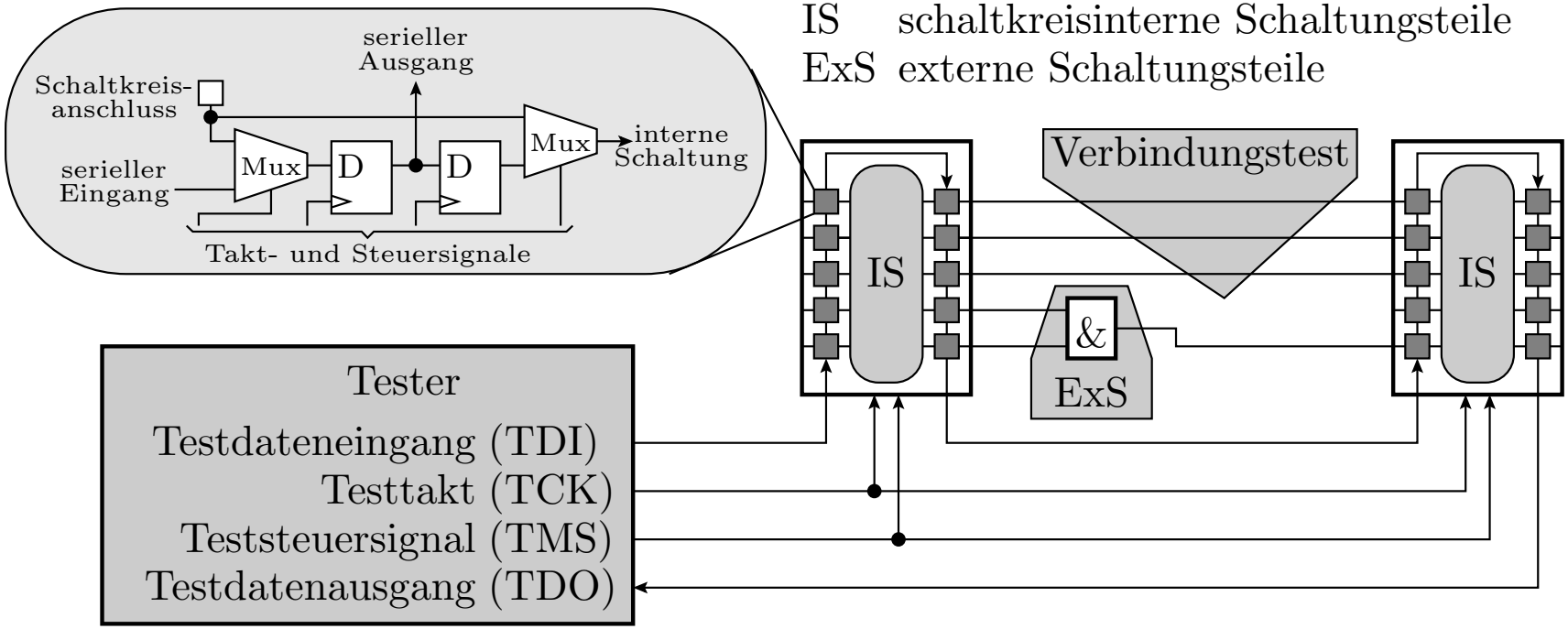
Automatisierte Baugruppeninspektion



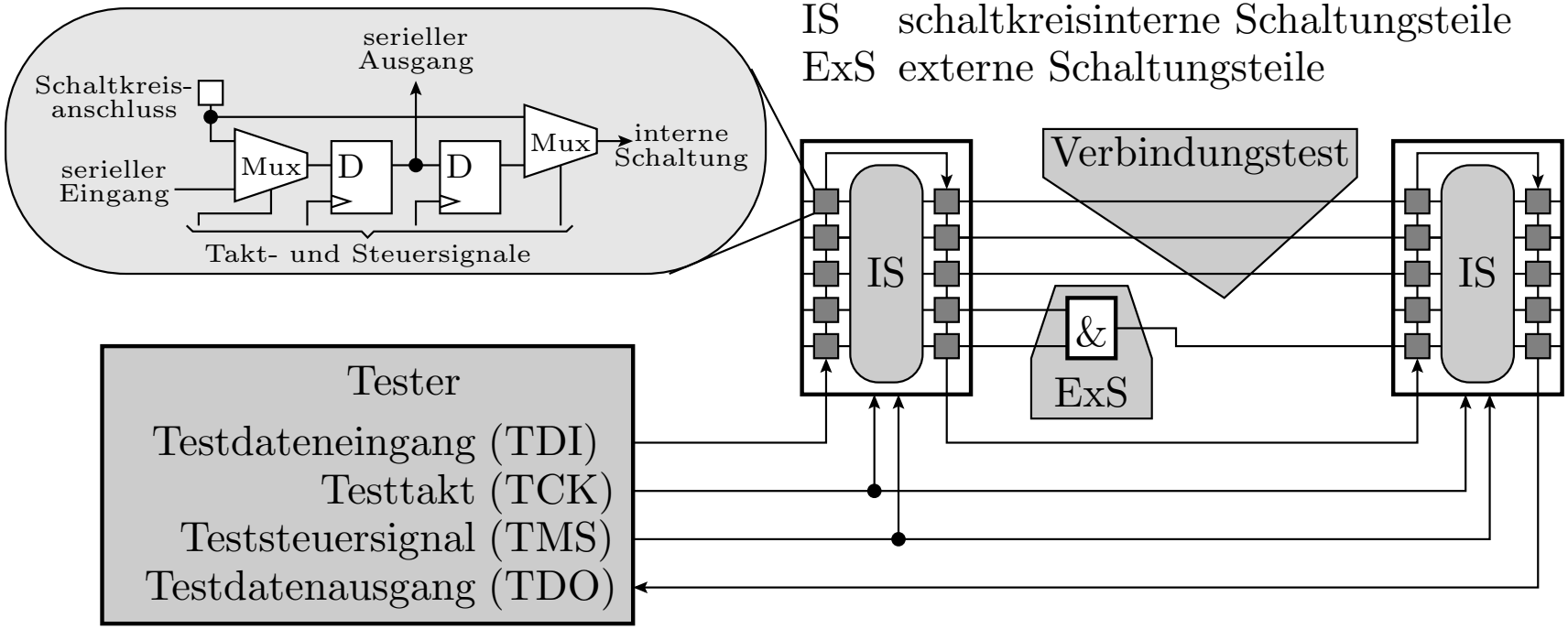
- Bildverarbeitungssystem mit Beleuchtung, Kamera, Verarbeitung, Monitor.
- Lernen von Bildern mit Fehlern und korrekten Bauteilen.
- Generierung des Prüfprogramms aus einer geometrischen Beschreibung und einer Bilddatenbank.
- Pflicht für sicherheitskritische Baugruppen (Automotive).
- Gehört zu den statischen Tests.

Boundary Scan / JTAG

Das Prinzip von Boundary-Scan



- Ersatz der mechanischen Nadeln durch »silicon nails« (seriell beschreibbare Register an den Schaltkreisanschlüssen, im Normalbetrieb überbrückt)
- Aufspaltung in viele kleine, separat testbare Teilschaltungen.

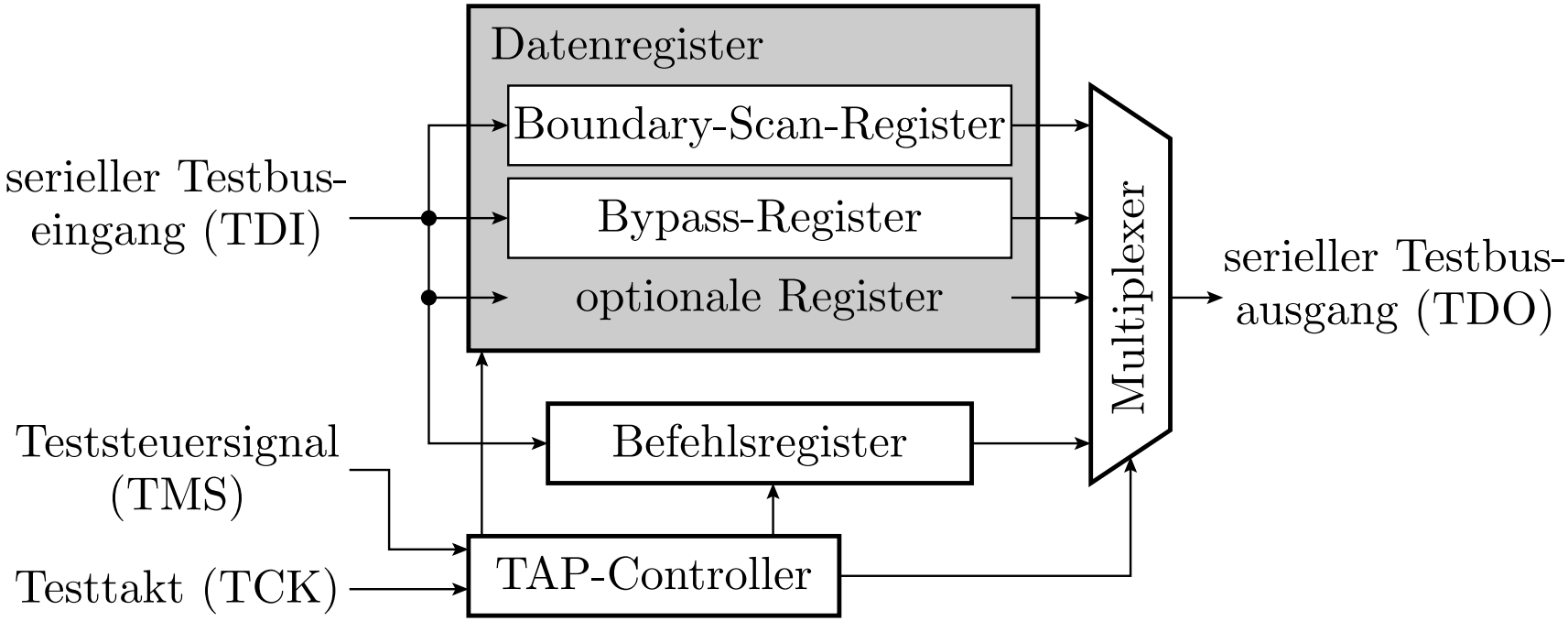


Ablauf eines Testschritts für den Baugruppentest:

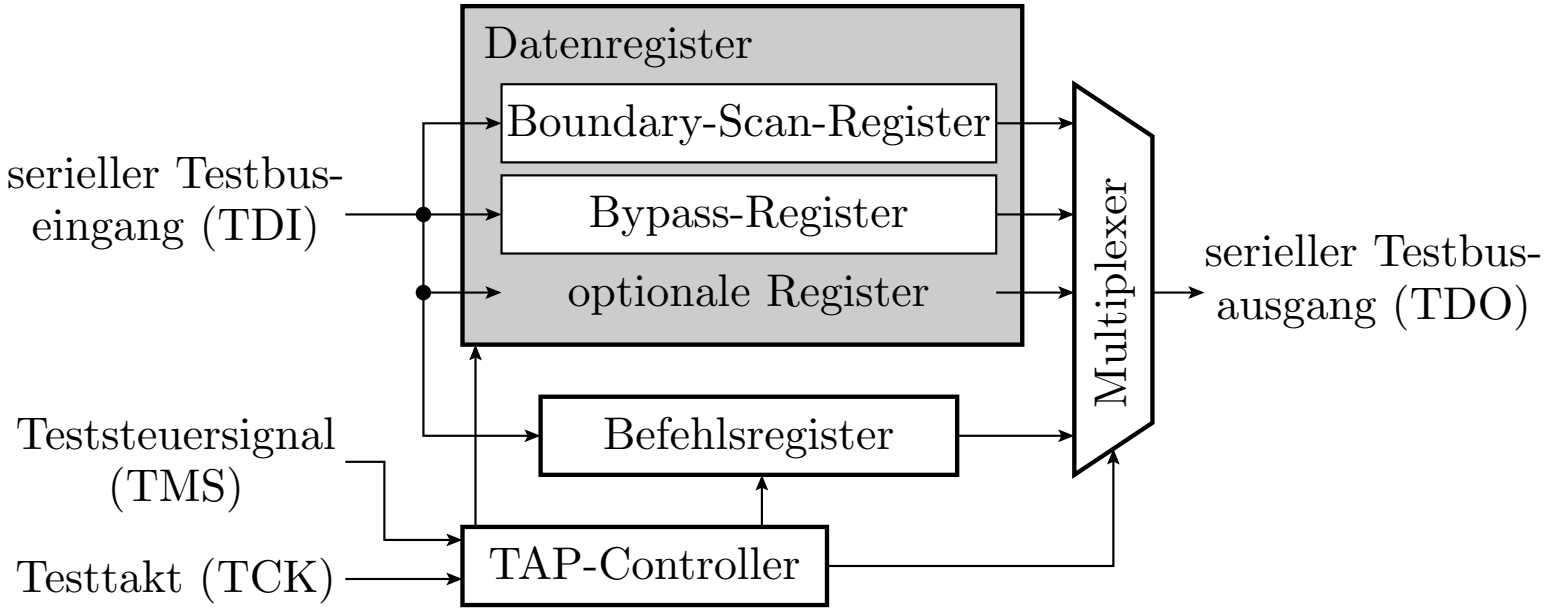
- BS-Register aller Schaltkreise auf der Baugruppe seriell beschreiben,
- einen Arbeitsschritt ausführen,
- die im Arbeitsschritt in den BS übernommenen Werte seriell an den Tester ausgeben und zeitgleich nächsten Eingabevektor laden.

- Ursprungsidee: Alternative zu den teueren, für jede Baugruppe speziell anzufertigenden Nadeladaptern.
- Als IEEE 1149.1 standardisierter serieller Testbus.
- Nutzbar für weitere Test-, Diagnose- und Rekonfigurationsfunktionen (z.B. für die Xilix-FPGA und die ATMEGA-Mikroprozessoren aus den Übungen zur Programmierung, zur Kommunikation mit dem In-Circuit-Debugger (Atmel) und eines intergrierbaren Logikanalysators (Xilinx-ChipsScope).

Die Testbusarchitektur der Schaltkreise

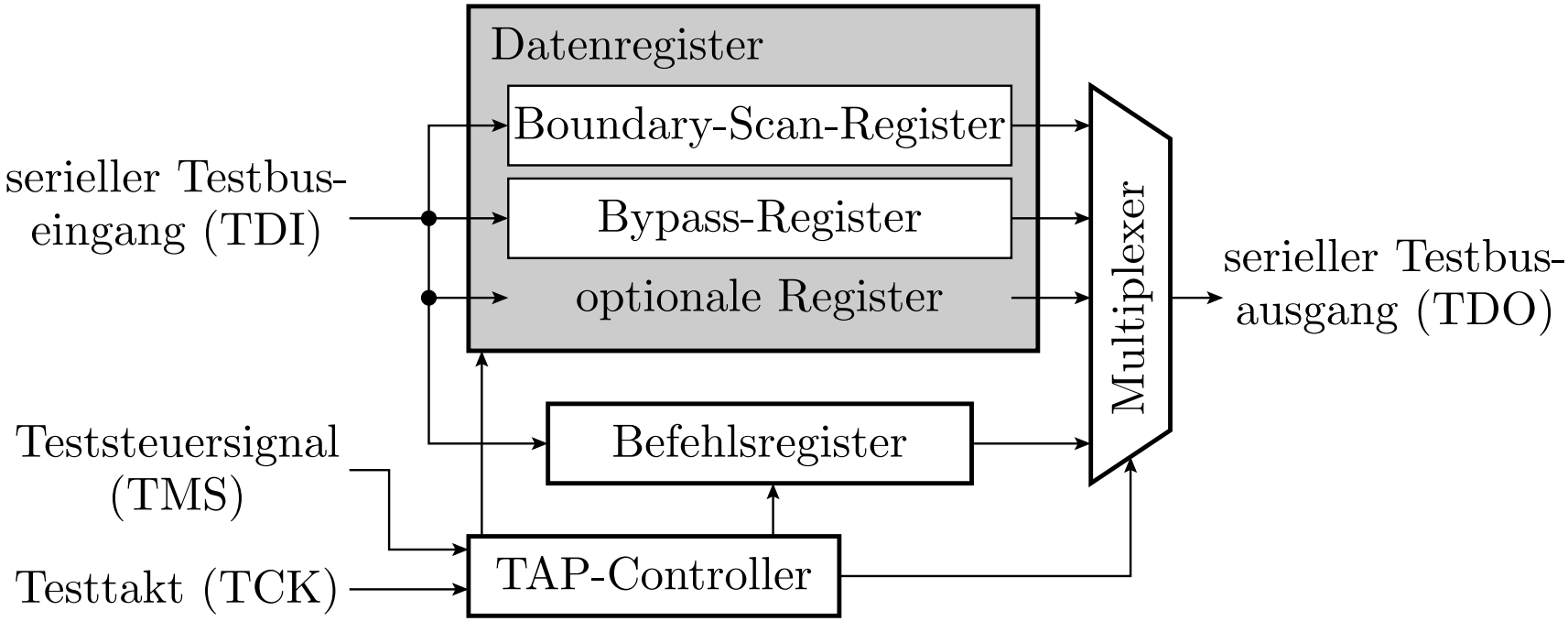


- den TAP- (test access port) Controller
- ein Befehlsregister
- mehrere Testdatenregister (mindestens das Boundary-Scan- und das Bypassregister).



Über TMS und TAP-Controller steuerbare Funktionen:

- Capture: Übernahme von Daten aus der Schaltung in das Befehlsregister
- Shift: Werte im Befehlsregister eine Position weiter schieben
- Update: eingeschobenes Bitmuster in das Befehlsregister übernehmen
- dieselben drei Funktion für ein über das Befehlswort ausgewähltes Datenregister, ...



Datenregister:

- Boundary-Scan: Register am Schaltkreisrand
- Bypass: 1-Bit-Register zur Überbrückung des Schaltkreises in der Schieberegisterkette der Baugruppe

Optionale Erweiterungen:

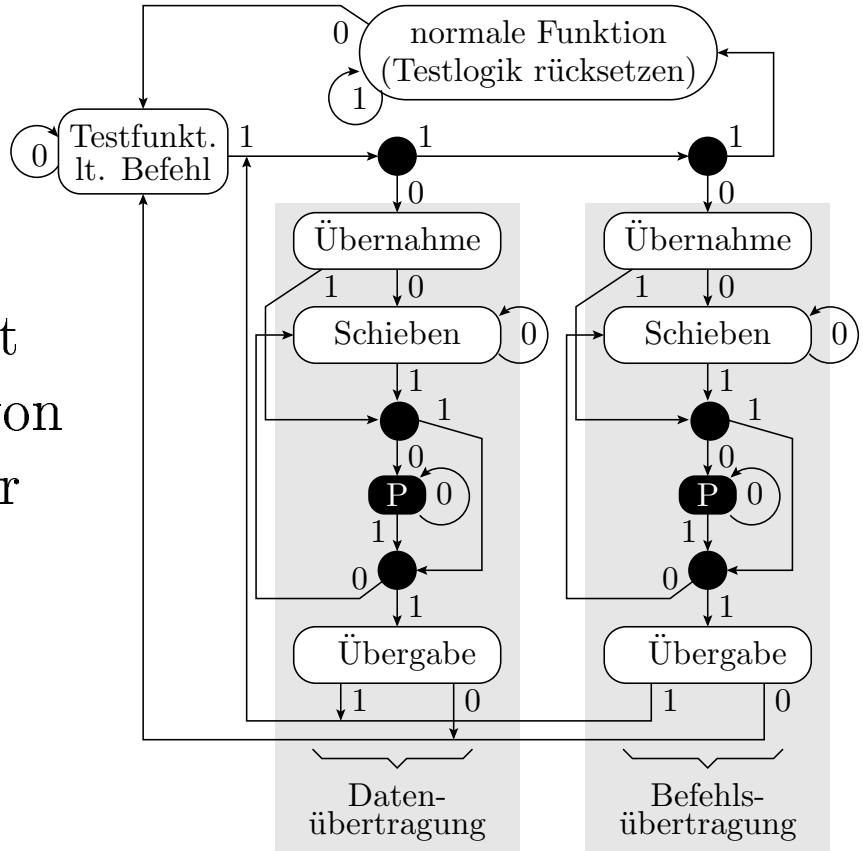
- Hersteller- und Bauteilidentifikationsregister
- weitere Test-, Programmier- oder Debug-Register

TAP-Controller, Busprotokoll

- Automat mit 16 Zuständen
- Kantenauswahl über TMS- (test mode select) Signal

Typischer Testablauf:

- Befehlsregister lesen (enthält ein Muster zur Erkennung von Busunterbrechungen und der Größe der Befehlsworte je Schaltkreis)
- Bauteilnummern lesen (Bestückungskontrolle)
- ein Teil der Schaltkreise auf
- Bypass setzen, für die anderen
- Datenregister auswählen, ...



Aufgaben

Aufgabe 6.1: Ideensammlung

- Testrahmen schreiben: Gegeben: Schnittstelle des Testobjekts, Datei-Lesefunktion für Testfälle. Testprotokoll auf der Standardausgabe nur die Tests, die versagt haben in der Form <Test-ID>: Testzweck, Eingabe, Sollwert, Istwert \n
- Digitaltest: Eingabe, fehlerhaftes Objekt, Sollausgabe, erkennbar?
- HIL: Funktion der Strecke und eines Reglers vorgeben. Kontrolle Toleranzschema für einen Sprung der Stellgröße. Mit Matlab kontrollieren.
- Test für MDA Übung mit Spice/.op mit Toleranzen?

Literatur

[1] Benedikte Elbel.
Zuverlässigkeitsorientiertes Testmanagement.
www.systematic-testing.com/.../Zuverlaessigkeitsorientiertes_Testmanagement.pdf, 2003.

[2] J. Hartmann and G. Kemnitz.
How to do weighted random testing for BIST?
pages 568–571, 1993.