Test und Verlässlichkeit (F4)
Foliensatz 4:
Test

Prof. G. Kemnitz

Institut für Informatik, Technische Universität Clausthal

1. Juli 2015

Inhalt Foliensatz F4: Test

Test

Test von IT-Systemen

Tests sind Kontrollen von Eigenschaften mit den Zielen:

- Nachweis der Nutzbarkeit,
- Aufdeckung von Problemen und Beseitigung, ...

Sie erfolgen in allen Phasen vom Entwurf bis zum fertigen Produkt im Einsatz.

- Keine Kontrolle ist perfekt, d.h. auch kein perfekter Nachweis für die Nutzbarkeit.
- Bei Beseitigungsiterationen wird nach jedem Beseitigungsversuch die Kontrolle wiederholt. Wenn Beseitigungsversuche keine neuen Fehler verursachen, hängt der Zuverlässigkeitszuwachs nur von der Testgüte ab.

$\Rightarrow$ Schwerpunkt in der Vorlesung: Test, Testgüte, ...

Zu testende Anforderungen

- 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 (Zuverlässigkeit, Sicherheit, Wartbarkeit, ...), ...

Je früher der Test, desto geringer die problembezogenen Kosten:

- »Standardkonformität« lässt sich schon bei der Zusammenstellung der Anforderungen prüfen,
- »Bedien- und Wartbarkeit schon beim Entwurf.
- Fertigungstests suchen nur noch nach Fertigungsfehlern, ...

Ziele und Grundprinzipien

Testziele

Die Testziele können positiv oder negativ formuliert sein:

positiv: Nachweis erfüllter Anforderungen.
negativ: Suche nach Fehlern.

Überprüfung der Anforderungen direkt oder an Beispielen:

statischer Test: Direkte Kontrolle von Eigenschaften, z.B. Syntax.
dynamischer Test: Ausprobieren der Service-Leistungen. Wahrscheinlichkeitsaussagen Fehlerabwesenheit und Zuverlässigkeit.

Praxis: Kombination positiv/negativ, statisch/dynamisch.

Positive Testziele

- Abhaken erbrachter Leistungen bei der Produktabnahme.
- Mindestpunktzahl zum Bestehen einer Prüfungsklausur, ...

⇒ Situationen, in den die »Pluspunkte« gezählt werden.

- Schätzen der Verfügbarkeit, Zuverlässigkeit, Sicherheit, ...
  durch Zählen der verfügbaren, korrekten, ...
  Service-Leistungen einer Stichpobe.

⇒ Experimente von Foliensatz 1

- Demonstrationen / Vorführungen

⇒ Interesse wecken, Vertrauen bilden.

- Haftungsausschluss

⇒ kein IT-System ist fehlerfrei. Kein Test ist perfekt. Genaue Abgrenzung der Sorgfaltspflicht des Herstellers vom Risiko des Nutzers.

Fehlernachweis als Testziel

Überwiegend funktionierende Systeme enthalten im Mittel Fehler. Anzahl abhängig vom Entsstehungsprozess, der Systemgröße, ... Es sollen möglichst viele Fehler davon gefunden und beseitigt werden.

Emotionale Barrieren:

- Fehler müssen als etwas »natürlich vorhandenes« betrachtet,
- Gefundende Fehler als etwas Positives zählen.
- Erkannte Fehler »wegreden« ist ganz schlecht.
- Die besten Tester: Maschinen, Pedanten, Authisten, ...

Kontraproduktiv / schlechtes Management:

- »It is not a bug, it is a feature!«
- »Grün-Reden« von Fehlern,
- weglassen von Tests, um Zeit- und Kosten zu sparen.

Statische Tests

Bei direkten Kontrollen gibt es nur Klassifizierungsfehler durch die Kontrolle selbst, aber nicht durch die Auswahl der Testbeispiele. Lokalisierung oft trivial. Nur für ausgewählte funktionale Anforderungen, dafür aber auch für einige nicht funktionale Anforderungen (die z.B. nur die Lebensdauer oder die Wartbarkeit beeinträchtigen) möglich. Beispiele:

- Elektrische Verbindungskontrolle auf Baugruppen durch Widerstandsmessungen entlang / zwischen Leitungen.
- Review: Kontrolllesen aller Arten von Entwurfsbeschreibungen (Anforderungen, Programme, Hardware-Beschreibungen, Testfallbeschreibungen, ...)
- Syntaxtest (siehe Foliensatz 3)
- Kontrolle Entwurfsregeln (z.B. Nichtverwendung fehlerträchtiger Programmkonstrukte, Regeln für Bezeichner, ...)

Dynamische Tests

Zusätzliche Ablaufschleife: »Wiederhole für alle Testfälle«

- Erfordert zusätzlich eine Testauswahl
- Steuerablauf: Eingabe, Ausführen, Ergebniskontrolle
- Kontrolle vorzugsweise Soll/Ist-Vergleich. Erkennt idealerweise alle Abweichungen.

Die Fehlerüberdeckung hängt vom Umfang und der Auswahl der Testbeispiele ab.

Auswahlprinzipien für Testbeispiele:

- funktionsorientiert (z.B. nach der Wichtigkeit der Service-Leitungen.)
- fehlerorientiert: Zusammenstellen einer Modellfehlermenge. Suche mindestens ein Testbeispiel je Modellfehler.
- zufällig: fehlerunabhängige Auswahl der Testbeispiele. Fehlerannahmen nur zur Festlegung der Testsatzlänge.

Kombination statischer und dynamischer Tests

Statische und dynamische Tests erkennen unterschiedlicher Fehler und werden in der Regle kombiniert. Typischer Entwicklungsablauf:

- 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).

(S – statischer Test; D – dynamischer Test)

Testsatz und Testfall

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

- die Testeingaben,
- Art der Kontrolle, in der Regel Sollwert/Sollverhalten,
- Mittel zu Testdurchführung (Hardware, Testrahmen).

Die Hardware für die Testdurchführung kann ein normaler Rechner, ein spezieller Tester oder ein HIL[1] sein.

Der Testrahmen ist das Programm, das die Testsätze abarbeitet und das Testprotokoll erzeugt.

[1] Hardware in the Loop Tester.

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.

Produkthaftung und Standards

Größere IT-Systeme haben statistisch gesehen Fehler, die erheblichen Schaden verursachen können.

Wer haftet für den Schaden durch IT-Systeme?

- Nach Schadenseintritt lässt sich oft rückwirkend zeigen, dass die Ursache ein Entwurfs- oder Fertigungsfehler war.
- Der Hersteller haftet bei Verletzung seiner »Sorgfaltspflicht«.
- Verletzt ist diese, wenn er den Fehler lt. Stand der Technik hätte finden müssen.
- Wie lässt sich nachweisen, dass ausreichend getestet wurde?

Genau dazu dienen Standards wie der zur Testfallbeschreibung.

Festlegungen in »abhakbarer« Weise, was für Tests und anderen verlässlichkeitssichernde Maßnahmen als ausreichend gelten und wie deren Erbringung zu dokumentieren bzw. nachzuweisen ist.

Weitere Standarts und Normen hierzu

- 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: Sechbändige Norm, die den Prozess der Bewertung eines Software-Produkts beschreibt.
- ...

Tester für Baugruppen

Arten von Baugruppentests

- Manuelle statische Tests: Reviews der Schalt- und Verdrahtungspläne, erste Inbetriebnahmeschritte.
- Automatisierte statische Tests: Serientests mit MDA, optischen Inspektionssystemen, ...
- Manuelle dynamische Tests: Ausprobieren der ersten Testbeispiele, ...
- Automatisierte dynamische Tests: Abnahmetests mit Funktionstester, HIL, ...

Manuelle Tests: flexibl, menschliche Faktor als Zusatzquelle für Fehlklassifikationen.

Automatisierte Test: hoher Vorbereitungsaufwand, einfach zu Wiederholen (wichtig für die Fehlerbeseitigung), höhere Erkennungssicherheit und weniger Phantomfehler.

Inbetriebnahme von Baugruppenprototypen

Manuelle statische Tests:

- Sichtprüfung auf Bestückungs- und Lötfehler.
- Kontrolle der Verbindungen über Widerstandsmessungen mit Multimeter.
- Anschluss der Spannung. Stromüberwachung. Kontrolle der Bauteile auf unnormale Erwärmung.

(Manuelle) dynamische Tests:

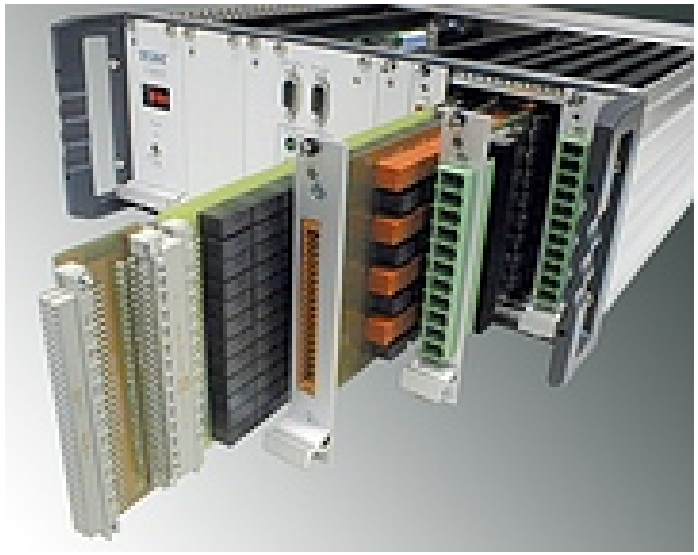
- Anschluss von Signalgeneratoren, Oszillographen, ...
- Manuelle Einstellung der Eingaben und Ausgabekontrolle.

Gründliche dynamische Tests:

- Programmgesteuerte Tests mit Funktionstestern, ...

»Wer mit Mist misst misst Mist«
(oder gute Messtechnik ist teuer.)

Modularer Funktionstester



Der komplette Test von CP-Systemen[2] 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, ...

[2] Cypher-Physikalische Systeme

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

Serientest für Baugruppen

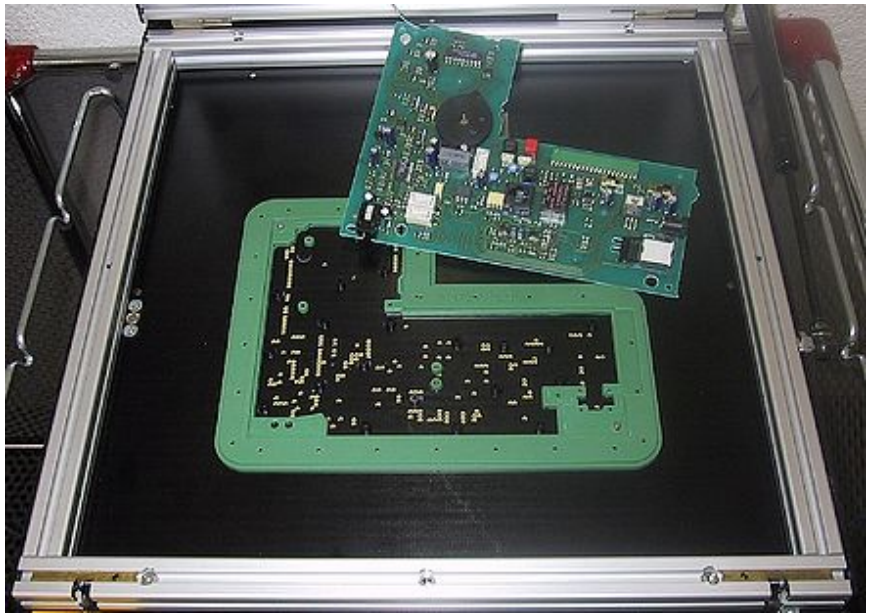
Vor der Serienfertigung sind Entwurfsfehler beseitigt. Fertigungstests von Baugruppen suchen Bestückungs- und Verbindungsfehler.

Stand der Technik verlangt automatisierte optische und elektrische Kontrollen[3].

Elektrische Tests sind vorzugsweise statisch. Adaptierung aller Verbindungen. Verbindungs- und Bestückungskontrolle durch elektrische Messungen ohne Versorgungsspannung.

[3] Irrtümer durch manuelle Kontrollen sind nicht mehr von der Produkthaftung ausschließbar.

Nadelbettadapter



Die elektronische Baugruppe wird für den Test auf ein Nadelbett gelegt und mit Unterdruck angesaugt.
Über die Nadeln sind Prüfgeräte angeschlossen. Die Kontaktierung schaltungsinterner Punkte erlaubt einen modularen Test und eine Lokalisierung von Kurzschlüssen, Unterbrechungen und defekten Bauteilen.

Weitere Unterteilung:

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

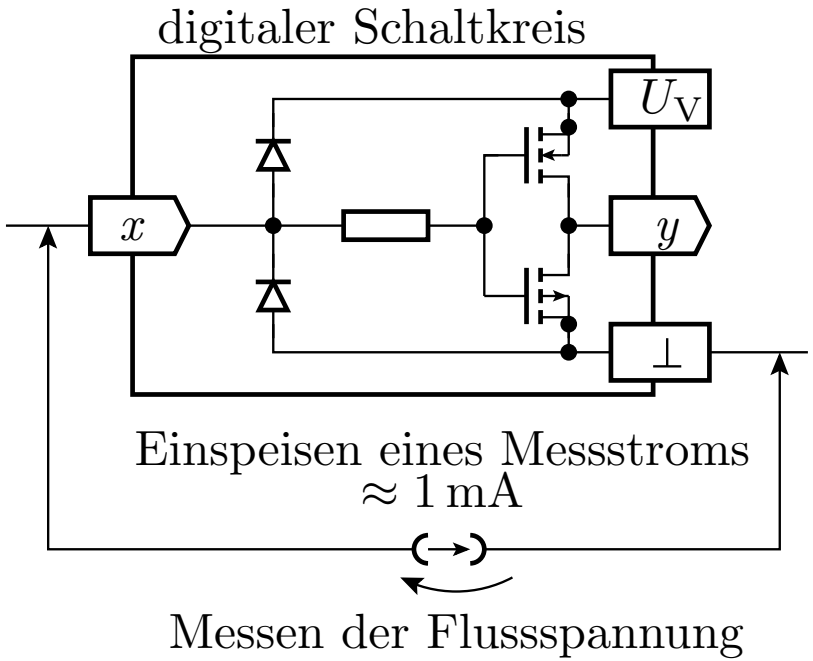
MDA

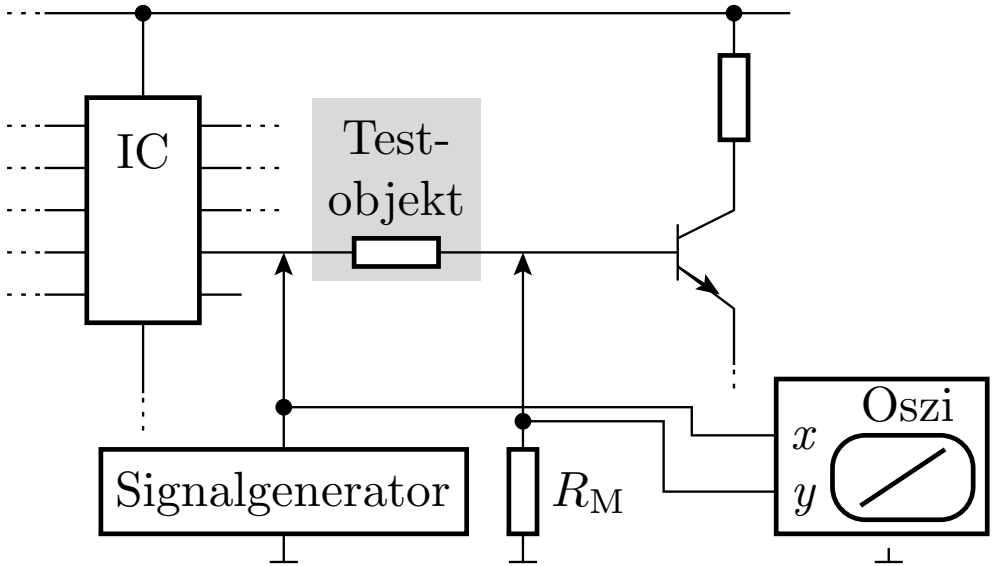
Suche potenzieller Bestückungs- und Verdrahtungsfehler mit elektrischen Zweipunktmessungen:

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

Bauteiltypische Strom-Spannungsbeziehungen für Sinuseingabe:

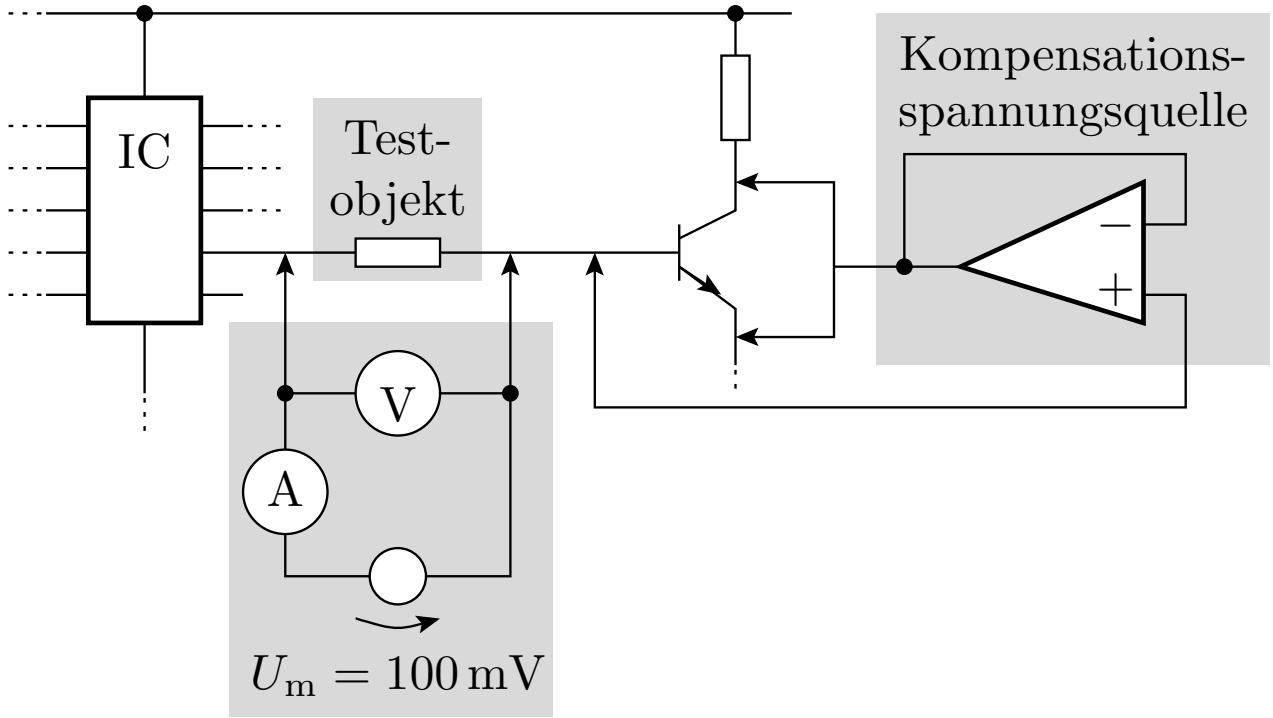
- Widerstand: Gerade,
- Kondensator: Elypse,
- Diode: Kennlinie mit Knick,
- Schaltkreise: Ausmessen der Schutzdioden,
- Unterbrechung: kein Strom,
- Kurzschluss: kein Spannungsabfall,
- ...





- Die Strom-Spannungs-Beziehung zwischen zwei Punkten in einer Schaltung hängt von der kompletten Schaltung, nicht nur einem einzelnen Bauteil ab.
- Bestimmbar durch Ausprobieren an einem »Golden Device«.
- Problematisch können sein
  - die Toleranzbereiche der Sollwerte unter Berücksichtigung der Bauteilstreuungen,
  - die Erkennungssicherheit für Fehlbestückungen, z.B. bei sehr kleinen Kapazitäten.

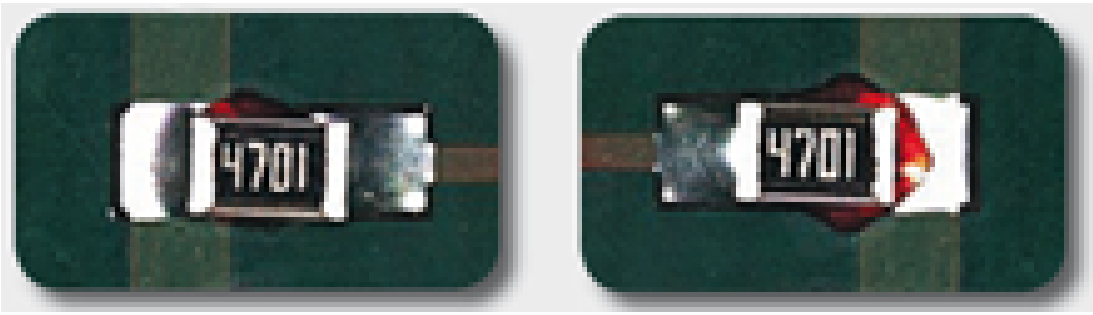
Analoger In-Circuit Test



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

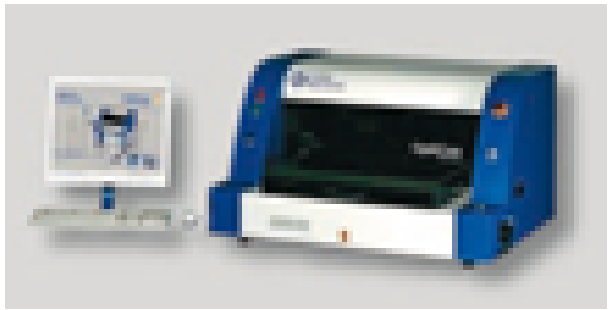
Optische Inspektion

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



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.

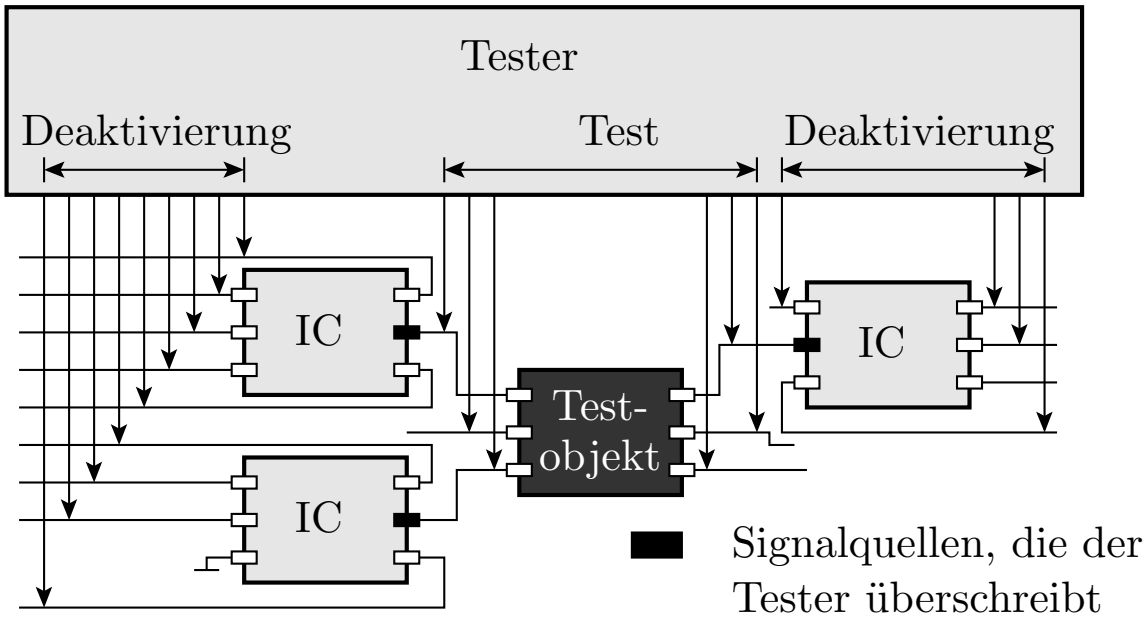
Automatisierte Baugruppeninspektion



- 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.

Digitale Systeme

Isolierter Test von Schaltkreisen ⇒ Digitaler In-Circuit-Tester:



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

Voraussetzung für ICT-Einsatz »prüfgerechter Entwurf«:

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

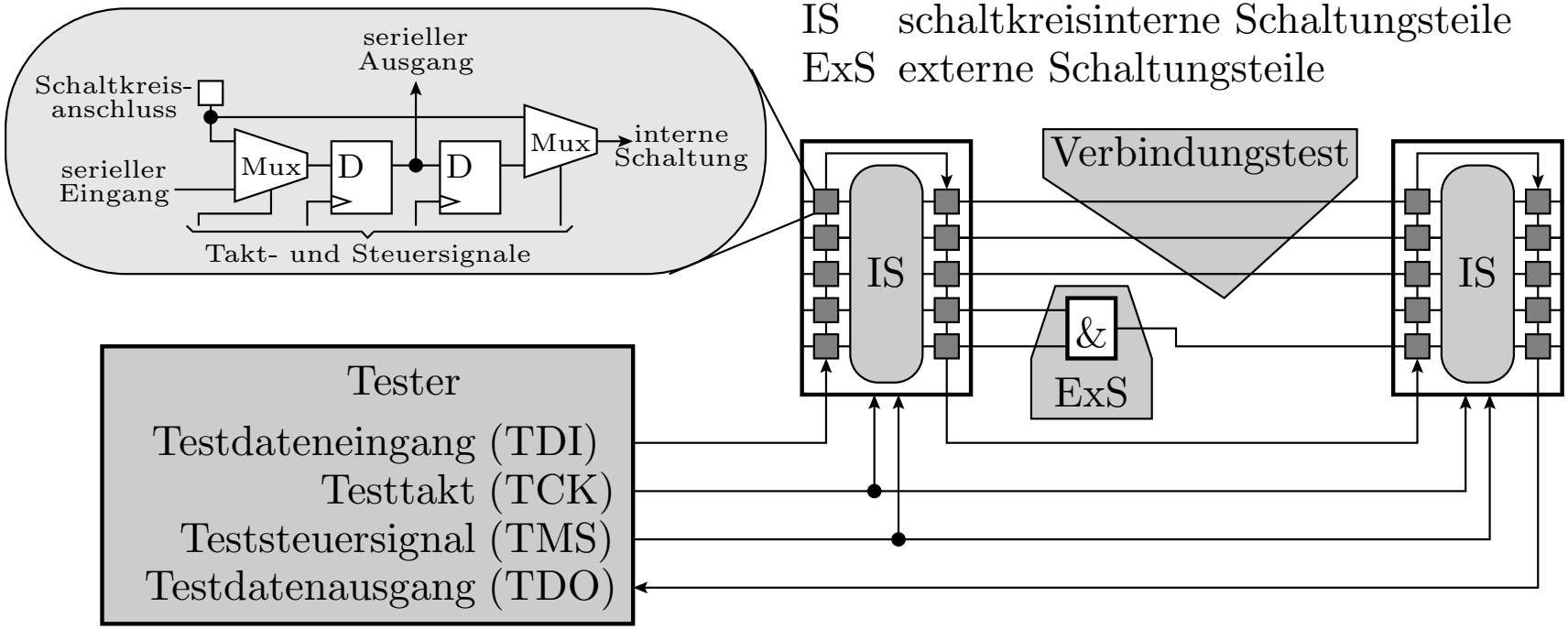
Automatische Generierung der Testvorschrift möglich:

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

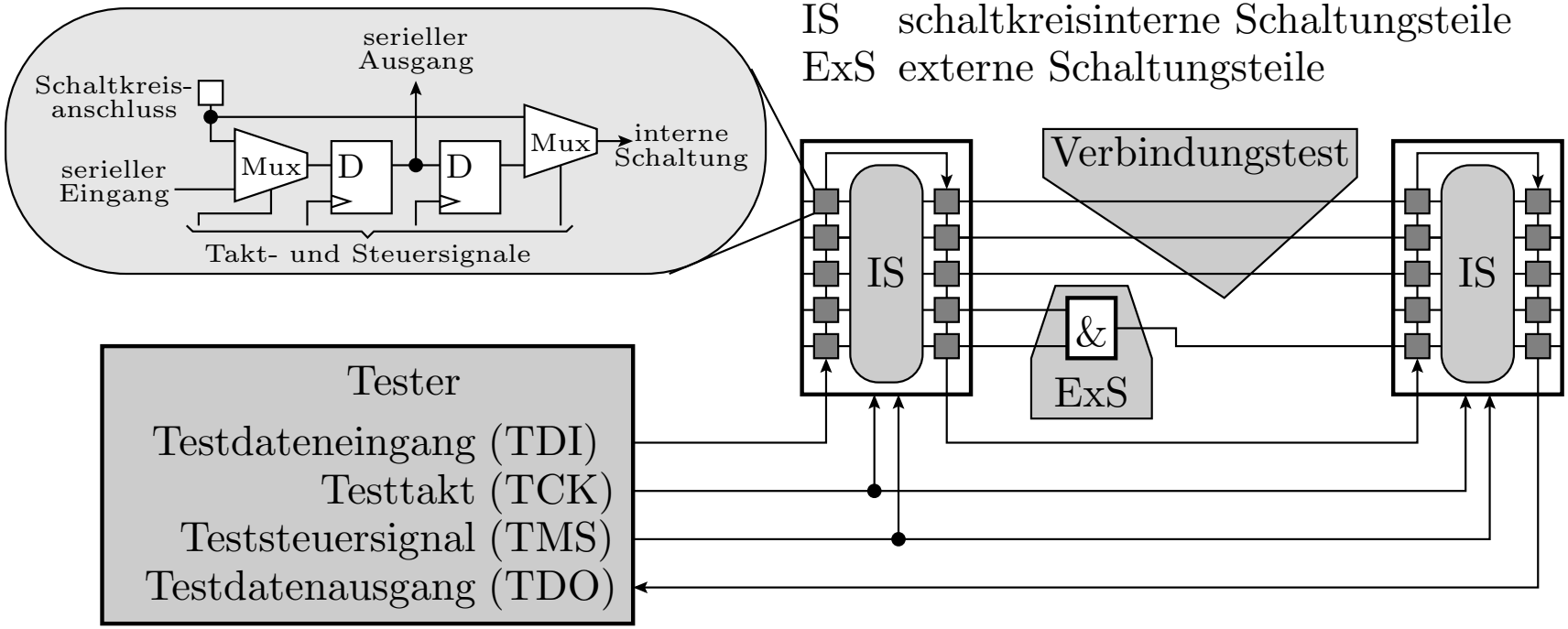
Fehlererkennung und Lokalisierung:

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

Boundary-Scan



- Ersatz der mechanischen Nadeln durch »silicon nails« (seriell beschreibbare Register an den Schaltkreisanschlüssen, im Normalbetrieb überbrückt)
- Verbindungen, Innenschaltung und »ExS's« separat testbar.

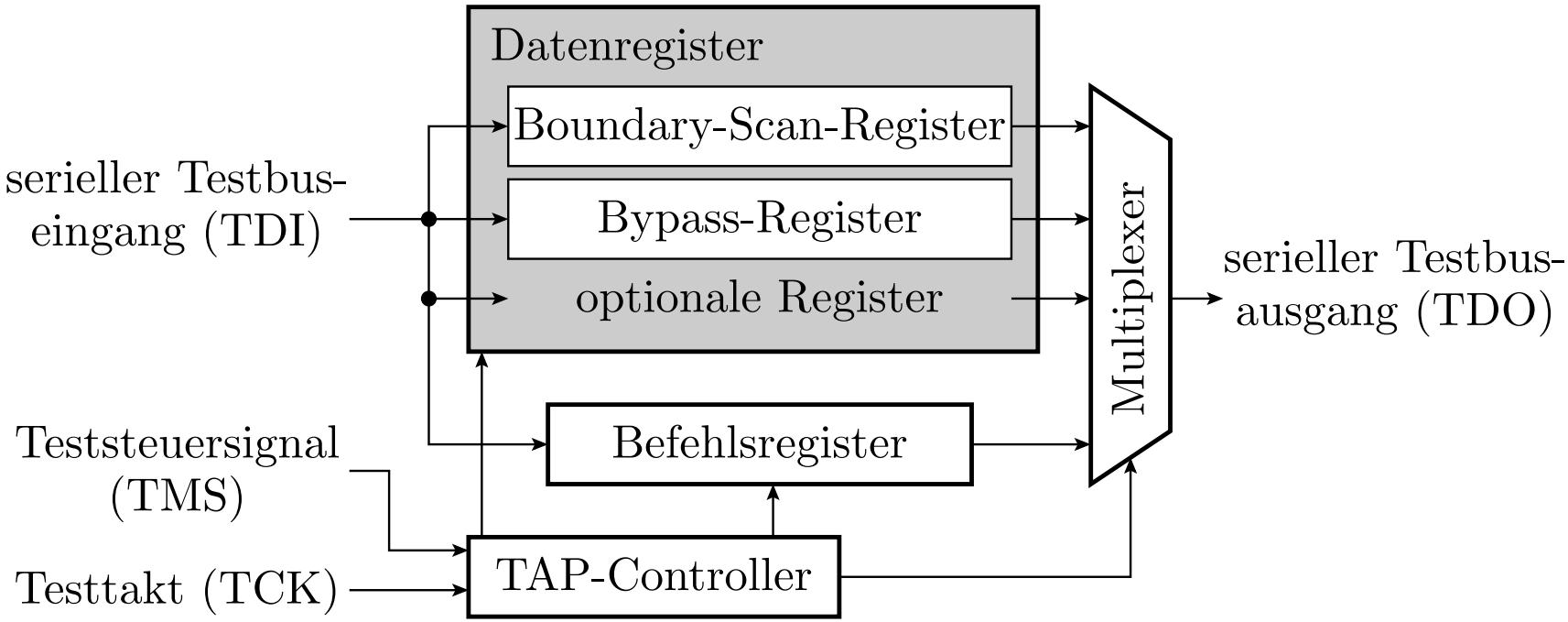


Ablauf eines Testschritts für den Baugruppentest:

- 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.

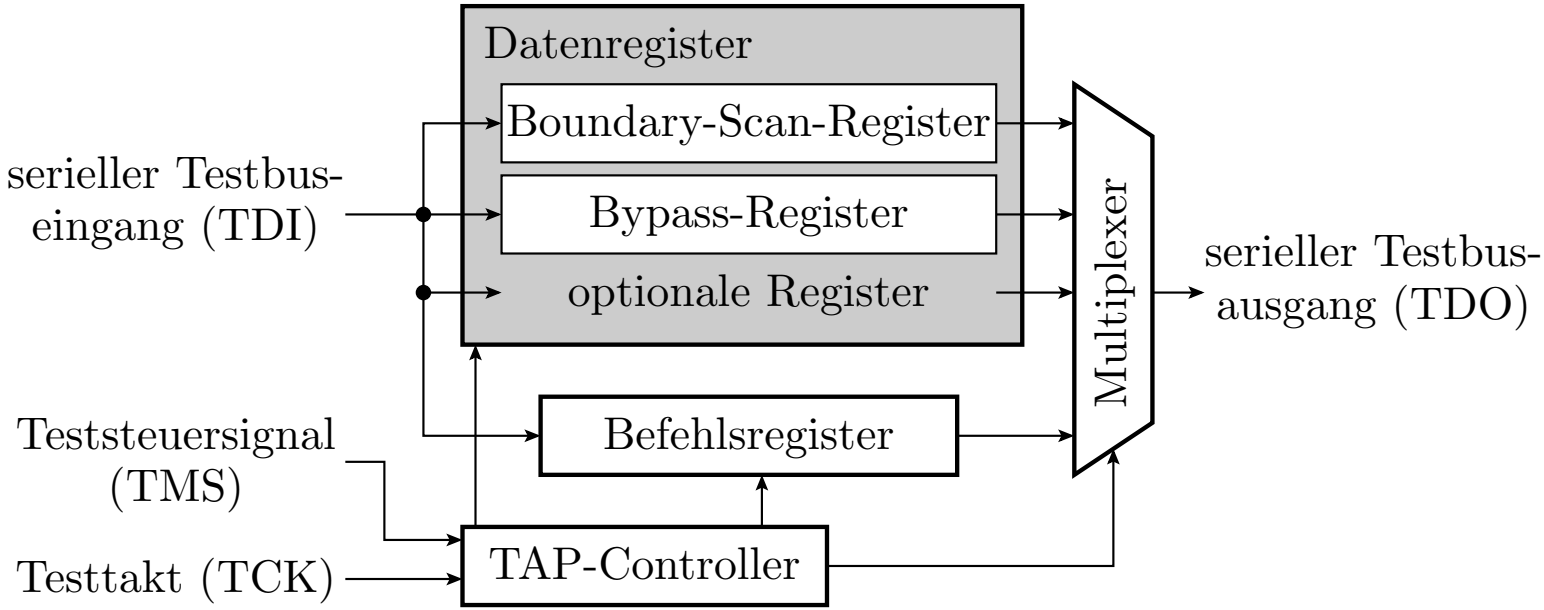
- Ursprungsidee: Alternative zu den teuren, für jede Baugruppe speziell anzufertigenden Nadeladaptern.
- Als IEEE 1149.1 standardisierter serieller Testbus.
- Nutzbar für weitere Test-, Diagnose- und Rekonfigurationsfunktionen (z.B. für die Xilix-FPGAs und die ATMEGA-Mikroprozessoren aus den Übungen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

Die Testbusarchitektur der Schaltkreise



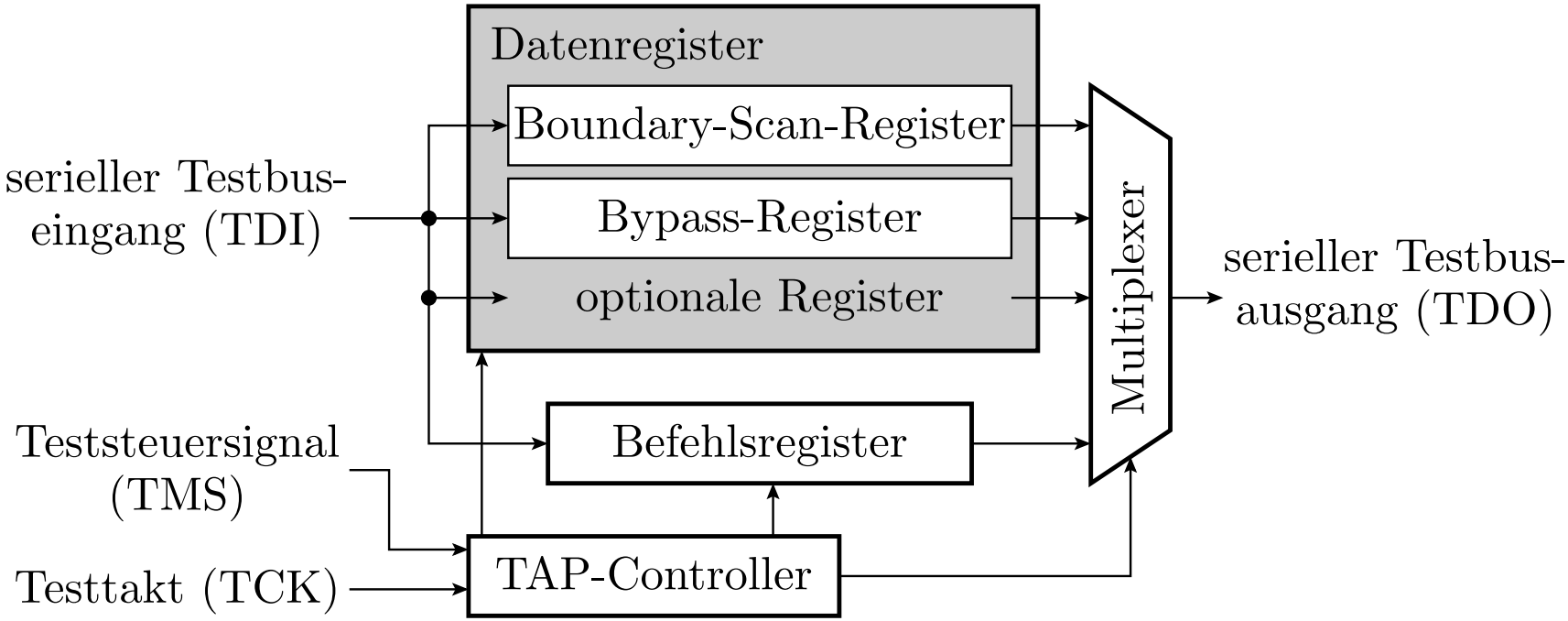
Eine Boundary-Scan-Implementierung umfasst:

- 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:

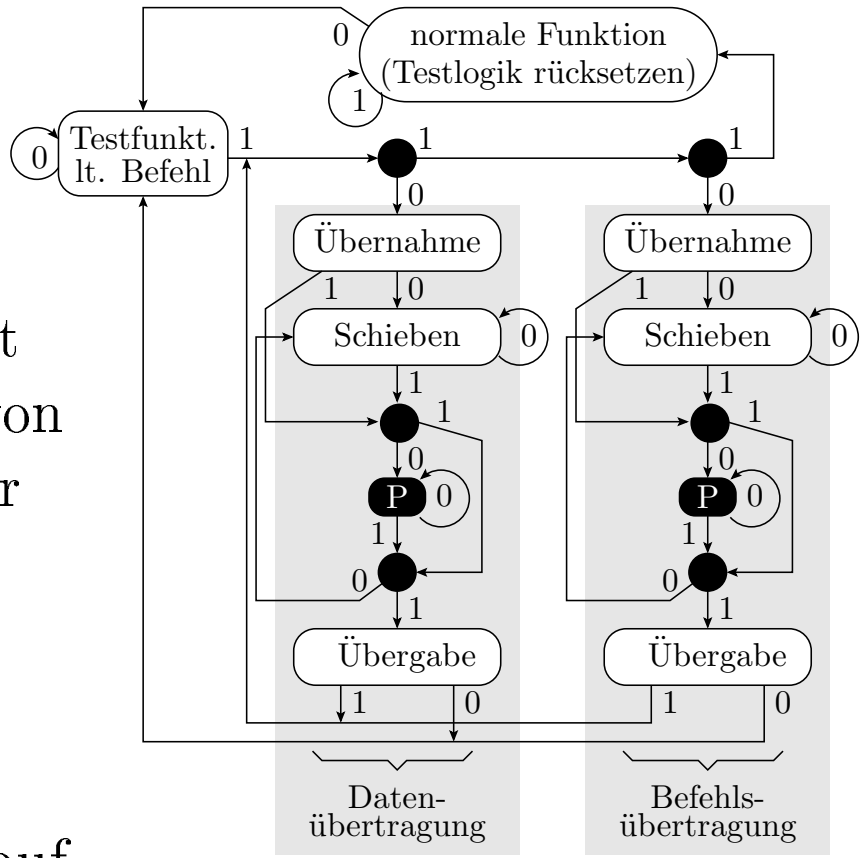
- Boundary-Scan: Register am Schaltkreisrand
- Bypass: 1-Bit-Register zur Überbrückung des Schaltkreises in der Schieberegisterkette der Baugruppe

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

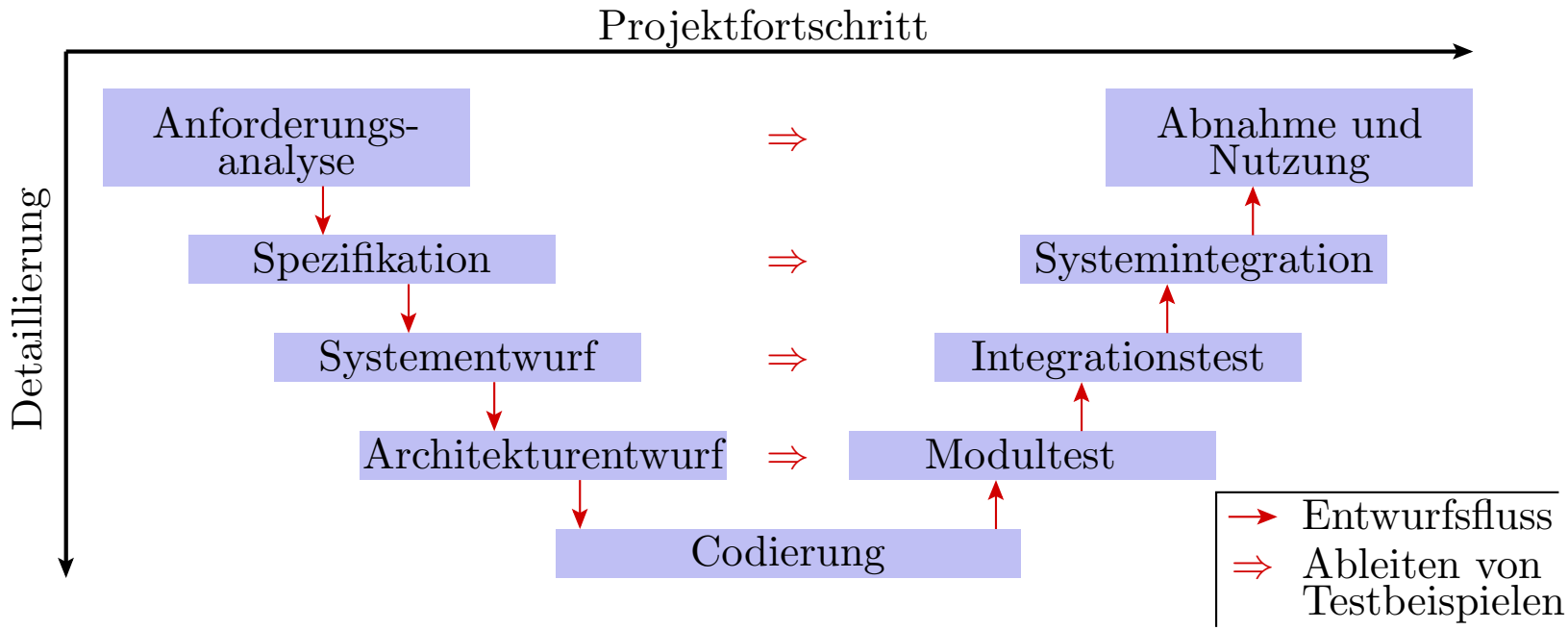


Typischer Testablauf:

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

Testrahmen für Entwurfsbeschreibungen

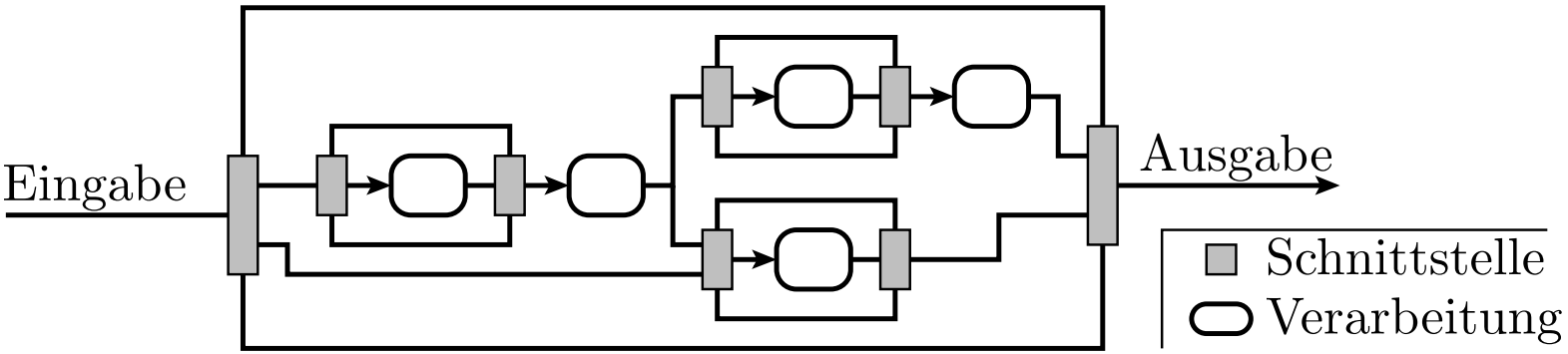
Entwurfs- und Testphasen



- Hard- und Software wird in Phasen entworfen.
- Zu jeder Entwurfsphase gehören Tests.
- Die unterste Ebene sind bei Software die Module und bei Hardware die kleinsten (selbstentworfenen) Teilschaltungen.

Modultest[4]

Test der kleinsten sinnvoll testbaren Systembausteine. Nach dem Service-Modell hat jedes Modul Schnittstellen für die Ein- und Ausgabe und eine interne Funktion.

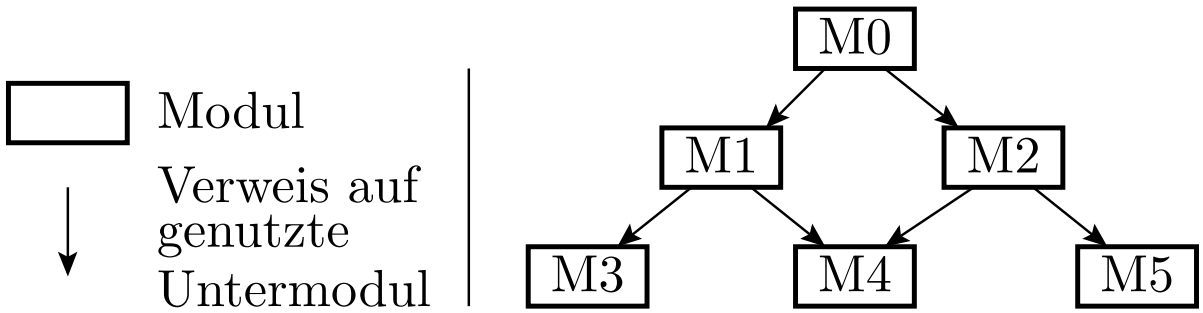


Für den Modultest wird jedes Modul aus dem Kontext gelöst, in einen Testrahmen eingebettet, der Testeingaben bereitstellt und die Testausgaben kontrolliert. Die Modularisierung legt der Entwurf fest und ist fundamental für die Testbarkeit.

[4] Für Hardware Simulation oder Test von Teilschaltungen.

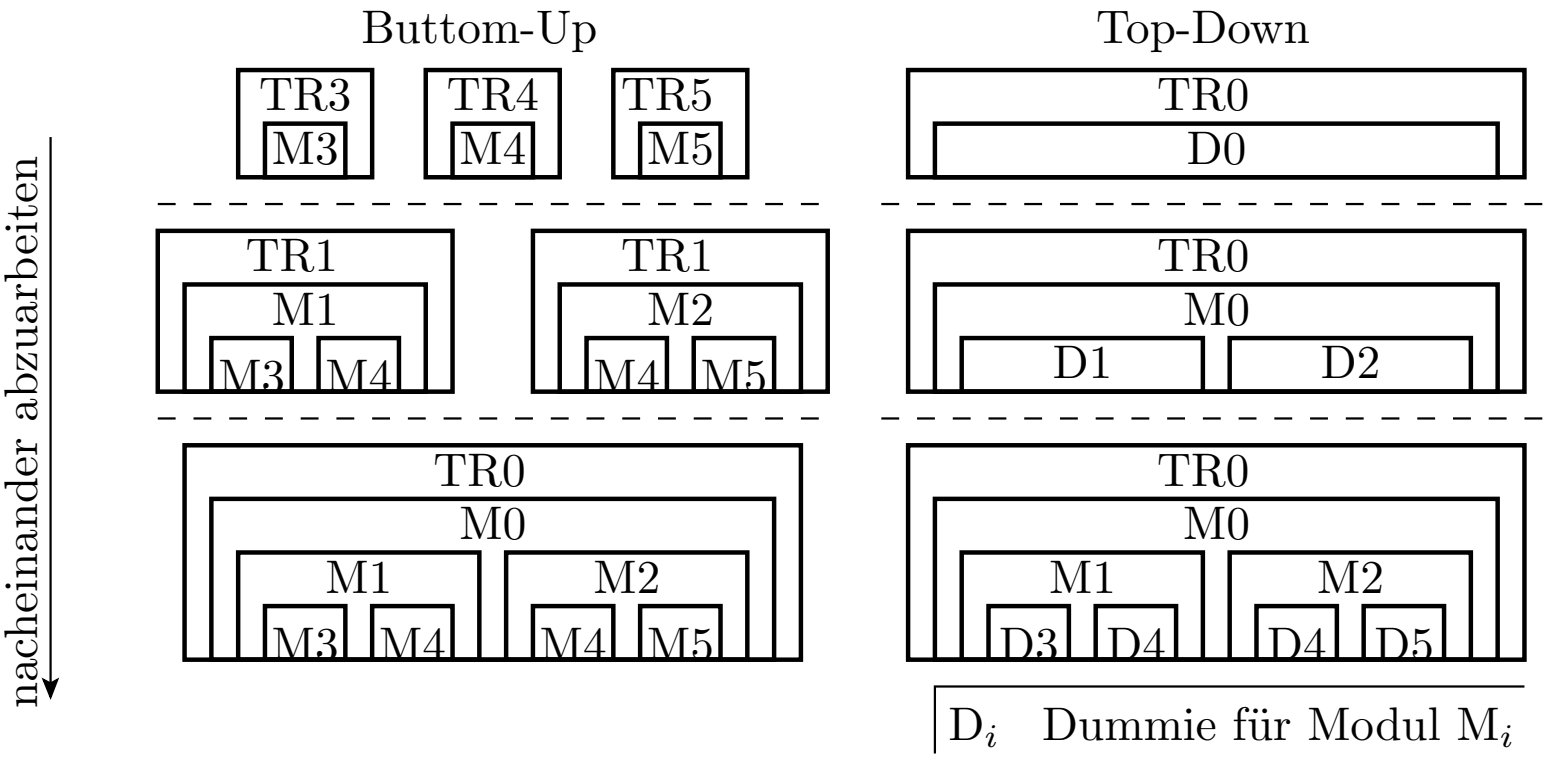
Entwurfs- und Testreihenfolge

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 das oberste Modul aus, verlangt aber die Entwicklung von Dummies für die zu Beginn noch nicht vorhandenen Module der unteren Schichten.

Testrahmen

Der Testrahmen ist ein Programm, dass die Eingaben bereitstellt und die Ausgaben kontrolliert oder zur Kontrolle bereitstellt. Beim Top-Down-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.

Stichwort »Prüfgerechter Entwurf«.

Bedatungen, Sollwerte, ... möglichst getrennt vom Testrahmen in einer Datei oder Datenbank abspeichern.

Tester und Selbsttest für digitale Schaltkreise

Test digitaler Schaltkreise

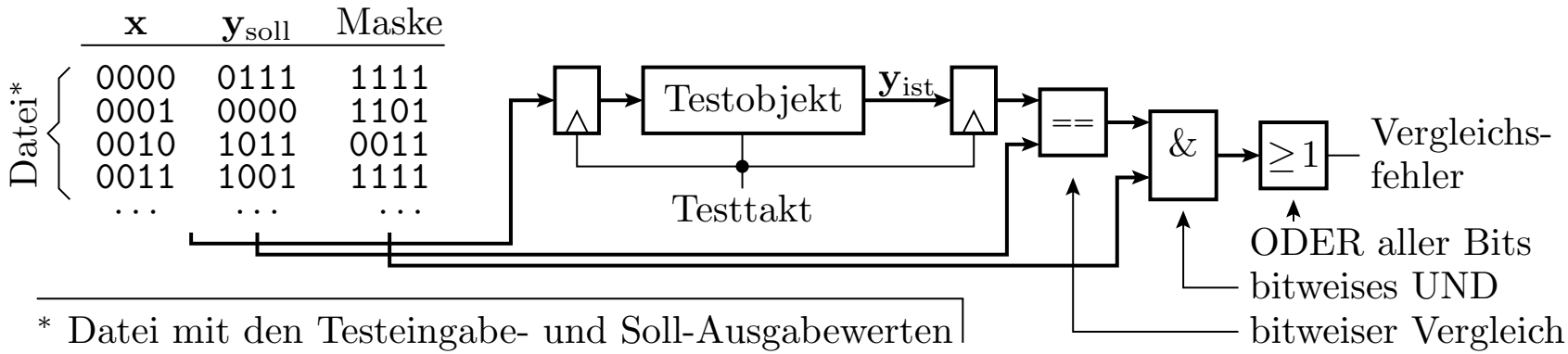
- digital ⇒ zweiwertig ⇒schlecht steuer- und beobachtbar.
- Systeme aus tausenden bis millionen Transistorschaltern.
- Eingeschränkte Kontaktiermöglichkeiten für interne Signale.
- Sehr lange Testsätze. Aufwändige Testsatzberechnung.
- Baugruppentests finden fast alle Bestückungs- und Verbindungsfehler, aber kaum Schaltkreisfehler. Fehleranzahl von Baugruppen ca. Summe der Fehleranteile der Schaltkreise.

Zusammenfassend:

- Große zusammenhängend zu testende Module.
- Hohe zu fordernde Fehlerüberdeckung.

Der Schaltkreistest ist deshalb Vorreiter in vielen Fragen von Test und Verlässlichkeit (Fehlermodellierung, Testsatzgenerierung, ...).

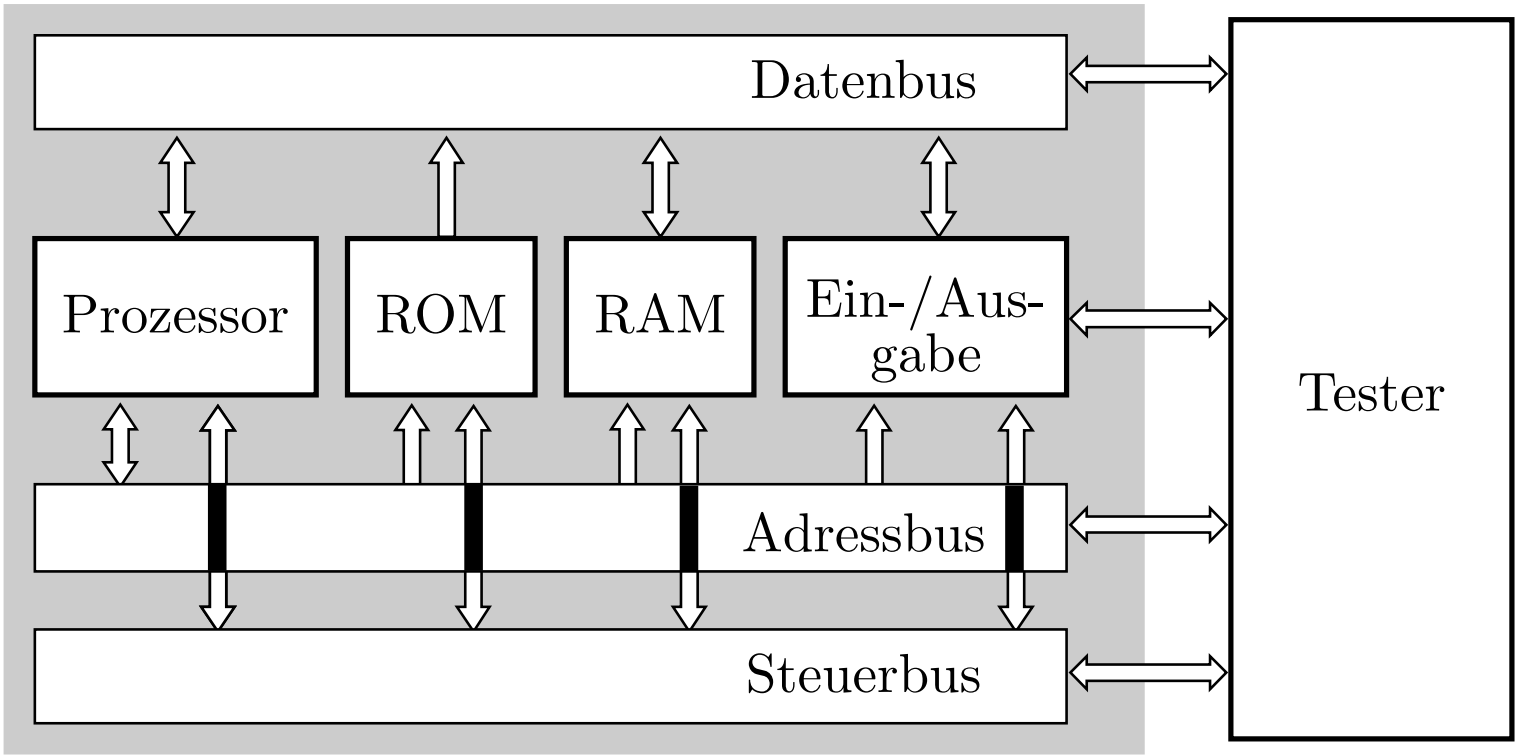
Prinzip eines Digitaltesters



Das Testprogramm beschreibt:

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

Auch Schaltkreise werden modular getestet



- Verlangt vom Entwurf Vorkehrungen für den Testerzugriff auf Schnittstellen zwischen Teilsystemen.

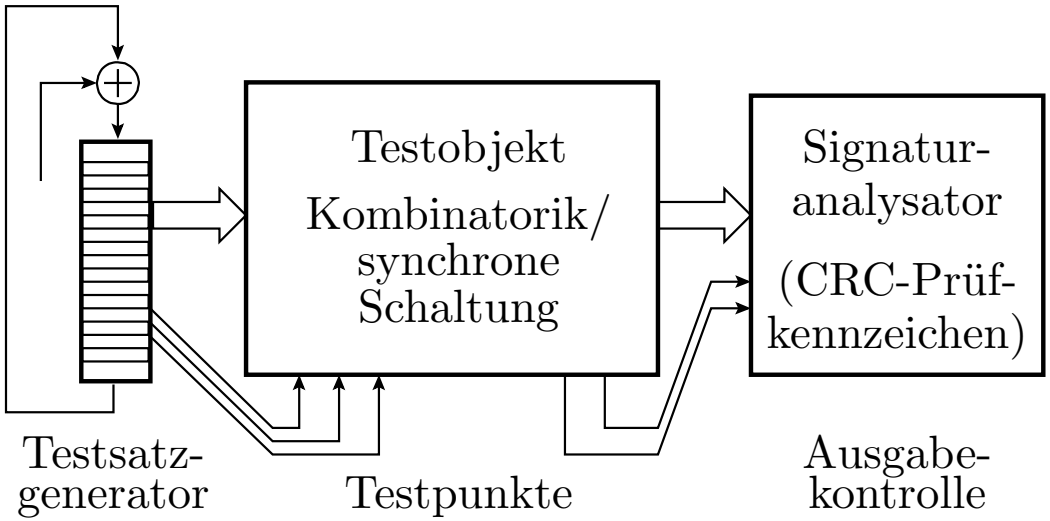
Geeignet sind Busstrukturen, Boundary-Scan, ...

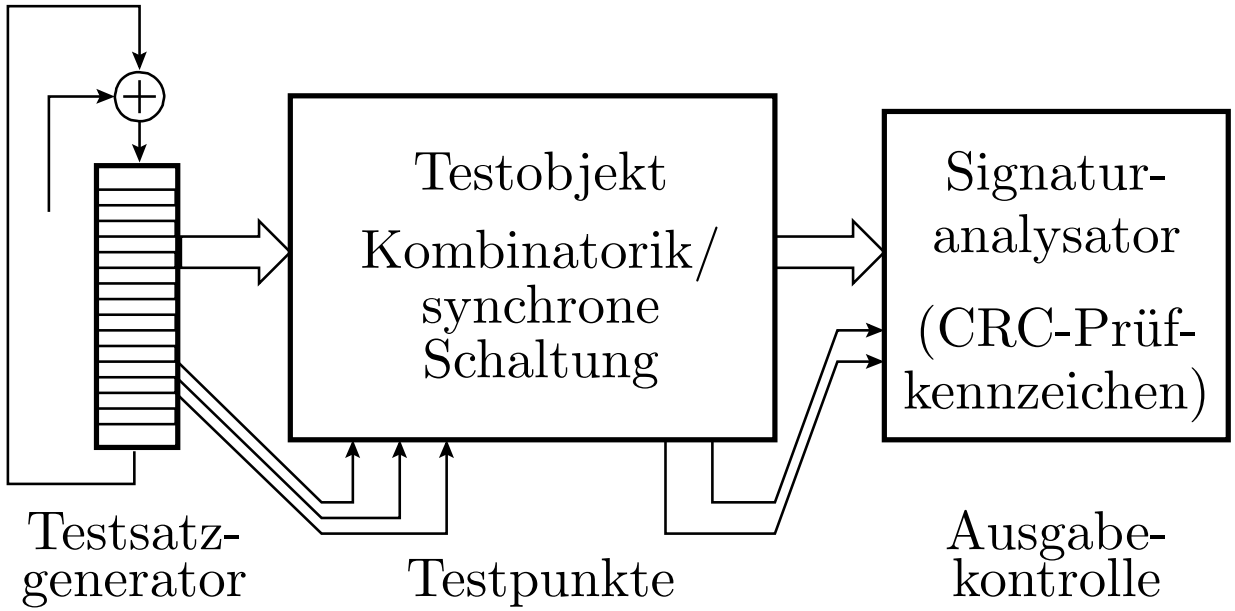
Selbsttests

Schaltkreistester sind sehr teuer:

- müssen so schnell wie die Schaltkreise sein, die Daten aber über viel längere Leitungen übertragen und Speicherzugriffe je Schritt ausführen, ...
- die Chips unverdrahtet kontaktieren (teilweise Nadeln für hunderte Pins).

Alternative Selbsttest:

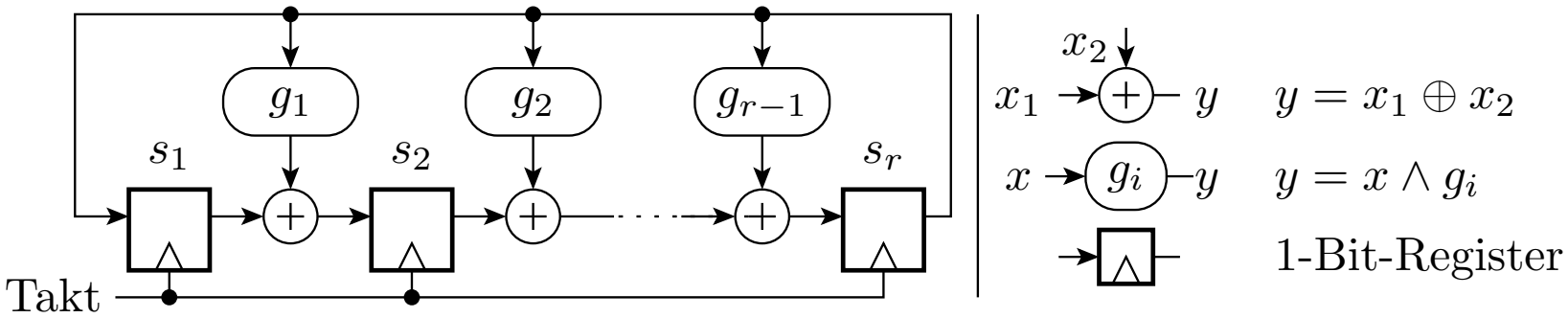




- Einrahmen der Testobjekte mit Pseudo-Zufallsgeneratoren und Signaturanalysatoren zur Prüfkennzeichenbildung.
- Zugangsmöglichkeit zu internen Schaltungspunkten. Erlaubt feinere Modularisierung.
- Integrierte Test-Hardware kann so schnell wie das Testobjekt getaktet werden.
- Zeitgleicher Test aller Chips auf einem Waver, ...

Linear rückgekoppelte Schieberegister

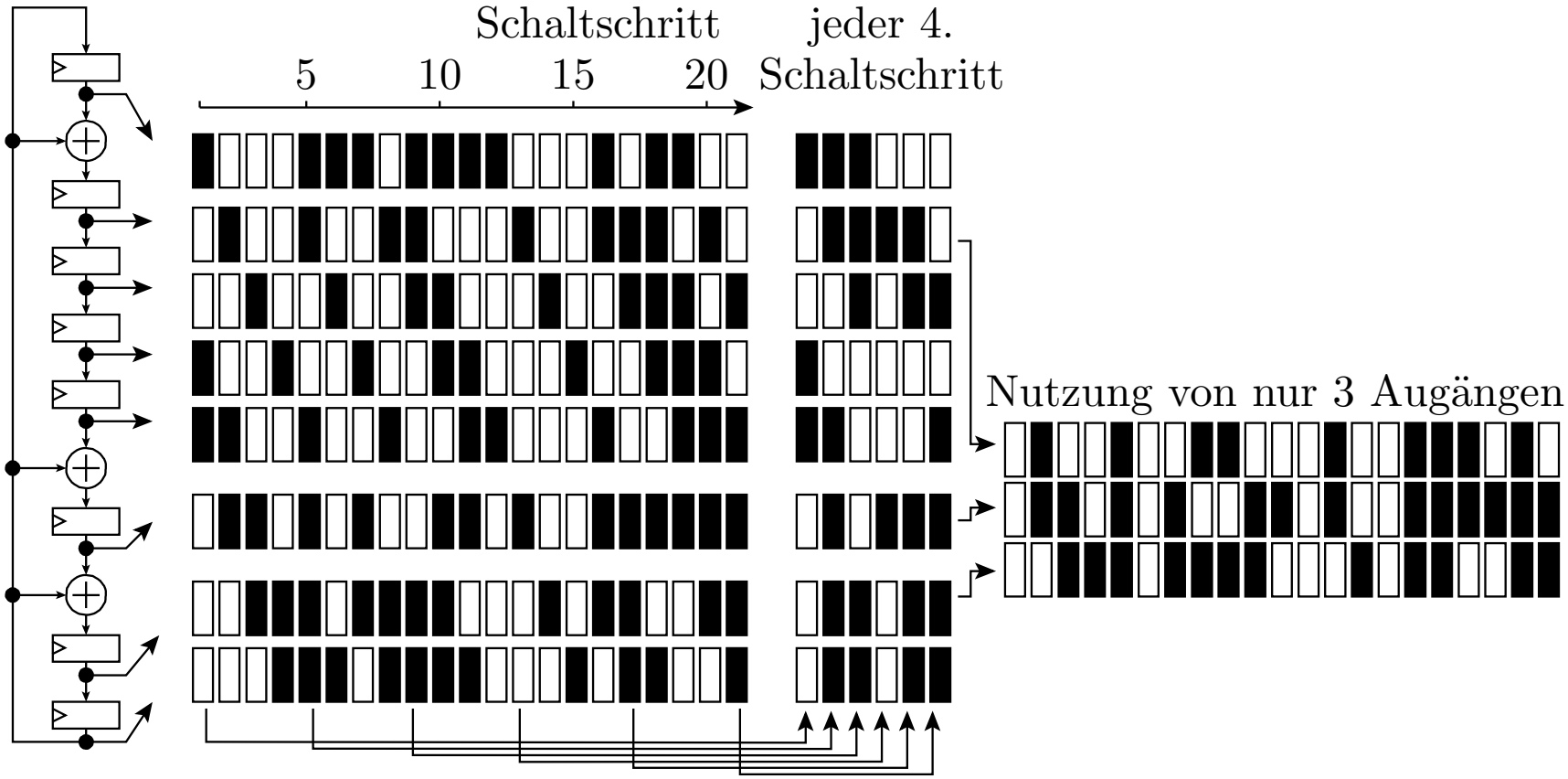
Einfachste und zur Testdatenbereitstellung meist ausreichende Lösung sind linear rückgekoppelte Schieberegister (LFSR linear feeback shift register) bzw. Algorithmen aus logischen und Verschiebeoperationen ähnlich CRC-Bildung:



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]:

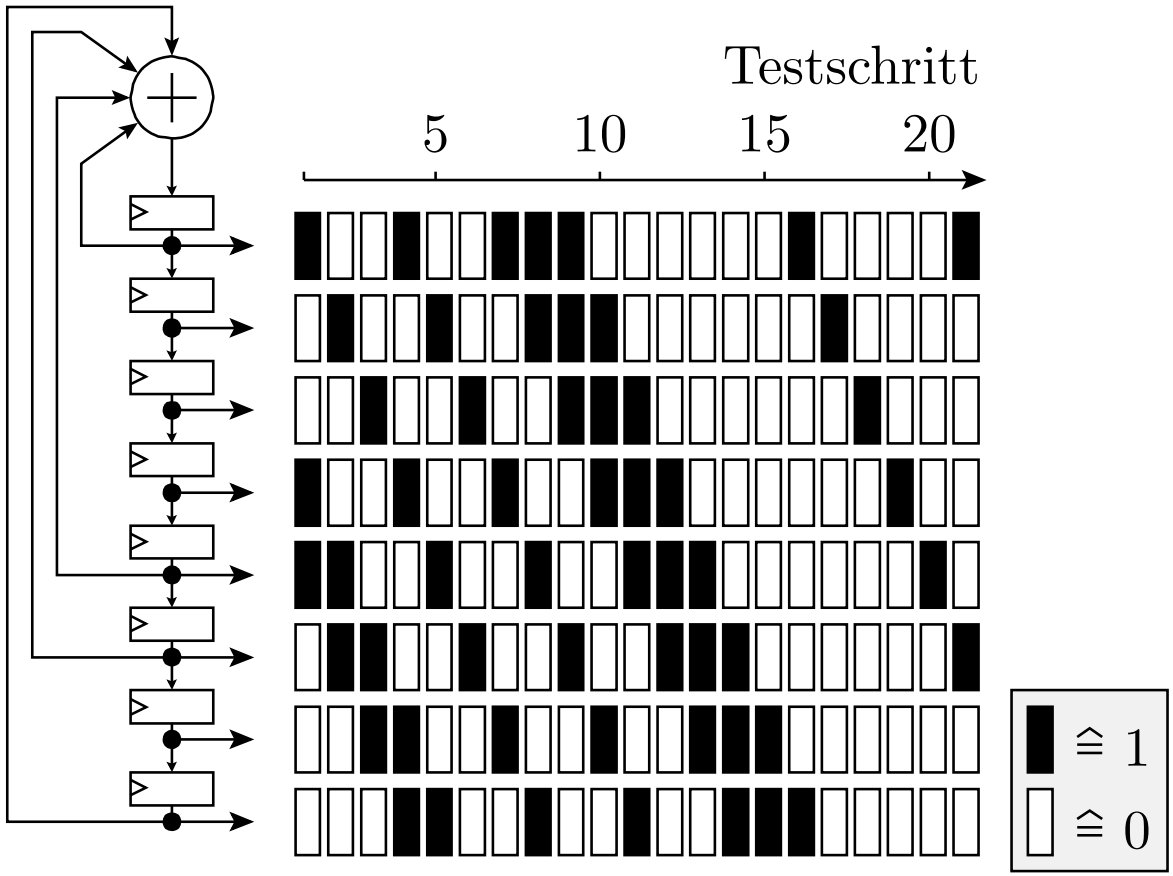
$$x^{16} \oplus x^5 \oplus x^3 \oplus x \oplus 1$$

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

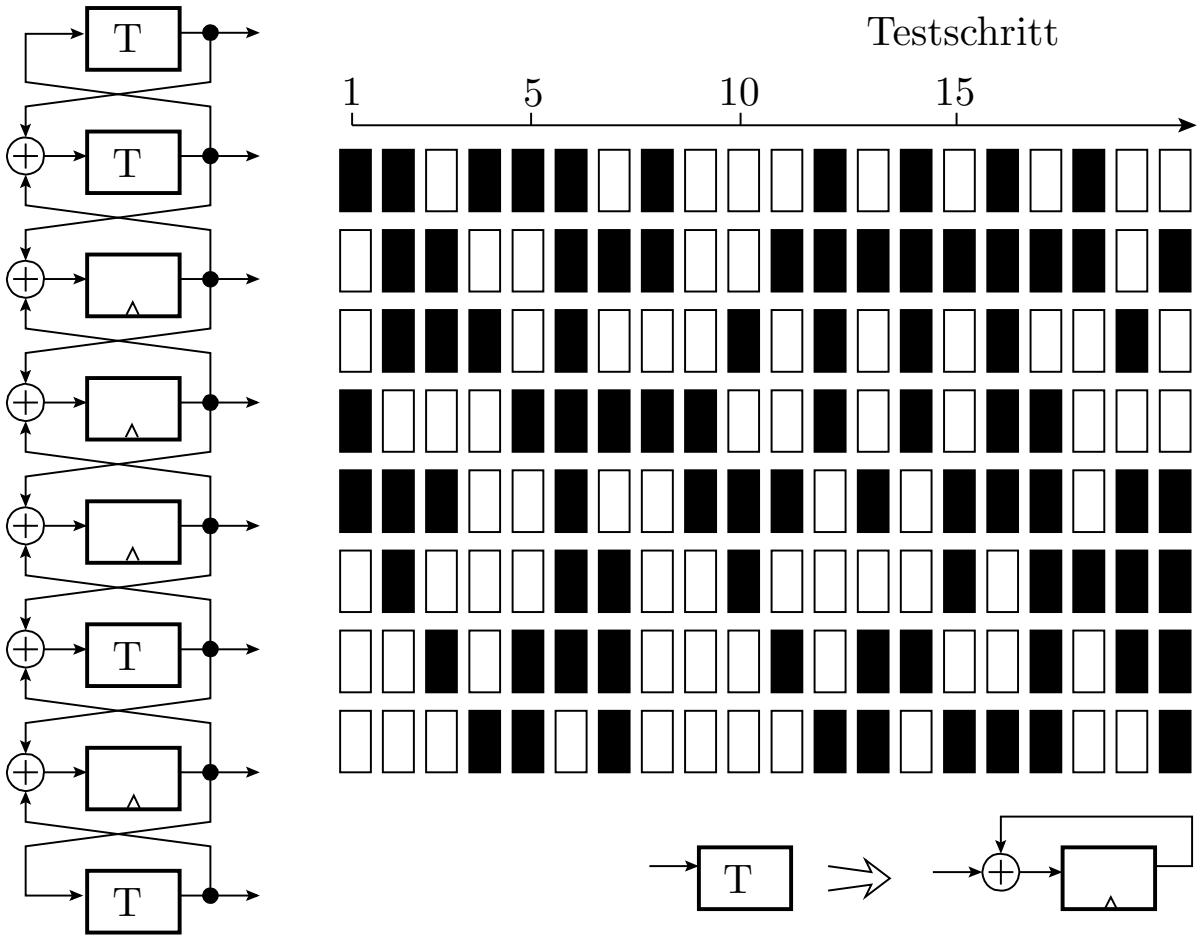


Falls die »Streifenmuster« durch die Schiebeoperationen stören, Generator für jede neue Testeingabe mehrere Schritte weiterschalten oder nur einen Teil der Ausgänge nutzen.

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



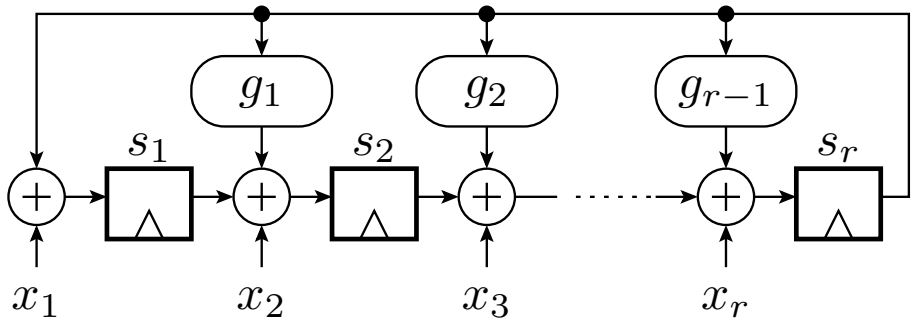
Es gibt viele weitere lineare Automaten, die auch zyklisch Bitfolgen in zufälliger Reihenfolge erzeugen. Beispiel Zellenautomaten, bei denen jedes Folgebit aus dem eigenen und den Zuständen der Nachbarbits gebildet wird:



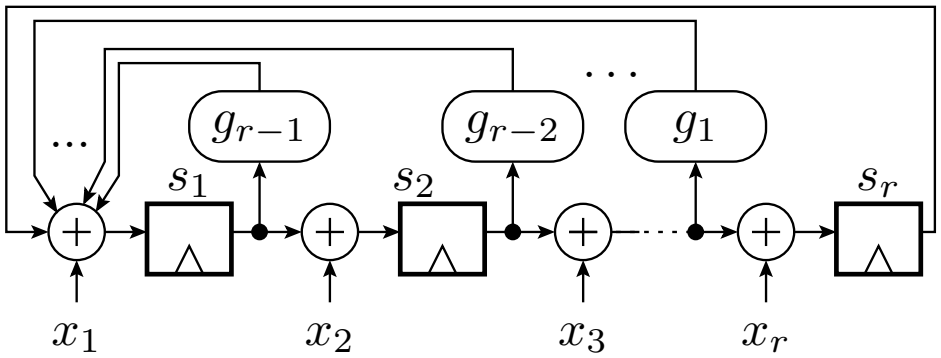
LFSR als Signaturregister

Zur Prüfkennzeichenbildung werden die Ausgaben des Testobjekts modulo-2 (EXOR) zu den LFSR-Zuständen addiert.

Parallels Signaturregister mit dezentraler Rückführung



Parallels Signaturregister mit zentraler Rückführung



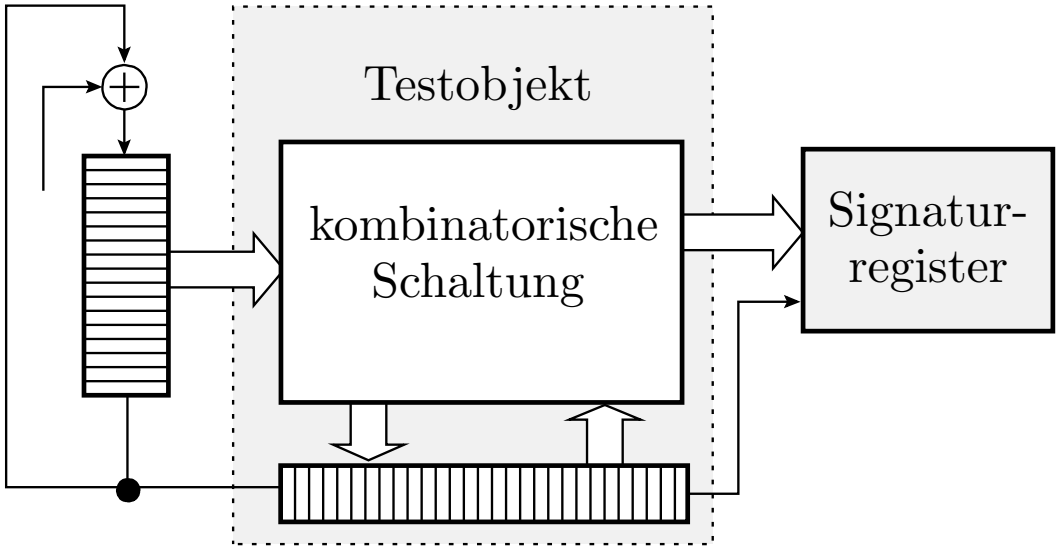
Das LSFR bildet so pseudo-zufällig ein Prüfkennzeichen.

Erkennungssicherheit für abweichende Testausgaben:

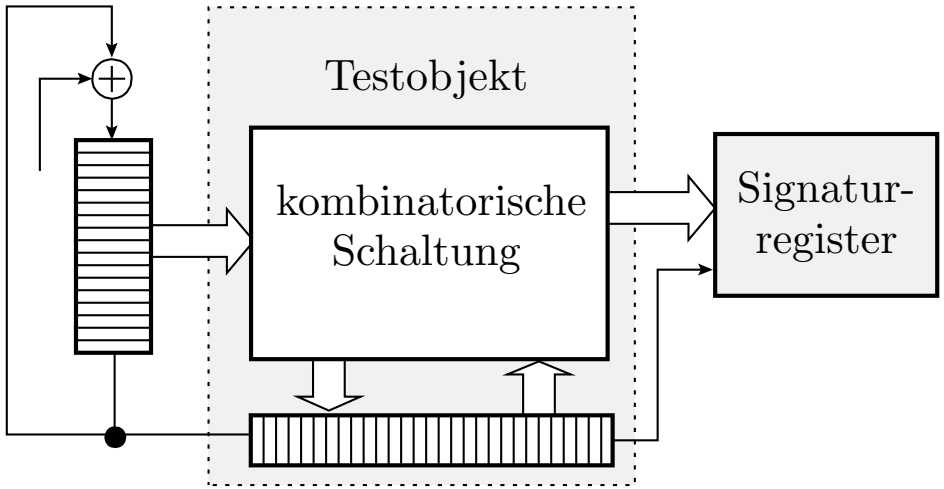
$$p_{\mathrm{E}} \approx 1 - 2^{-r}$$

(r – Registerlänge, siehe Foliensatz F3).

Pseudo-Zufallstest für die Übergangsfunktion



- Die Fehlerüberdeckung für den pseudo-zufälligen Test von Automaten ist oft deutlich höher, wenn die Übergangs- und Ausgabefunktionen isoliert getestet werden.
- Erweiterung der Eingabe-, Zustands- und Ausgaberegister um Schiebefunktionen für den Test.



- Wiederhole für jeden Testschritte (z.B. 1 Million mal)
  - Schiebe den Zustandsregisterzustand in das Signaturregister und beschreibe das Zustandsregister gleichzeitig mit Pseudo-Zufallswerten aus dem Eingaberegister.
  - Ausführung eines Testschritts mit Pseudo-Zufallswerten am Eingang und in den internen Registern und Ergebnisabildung in das Zustands- und Signaturregister.

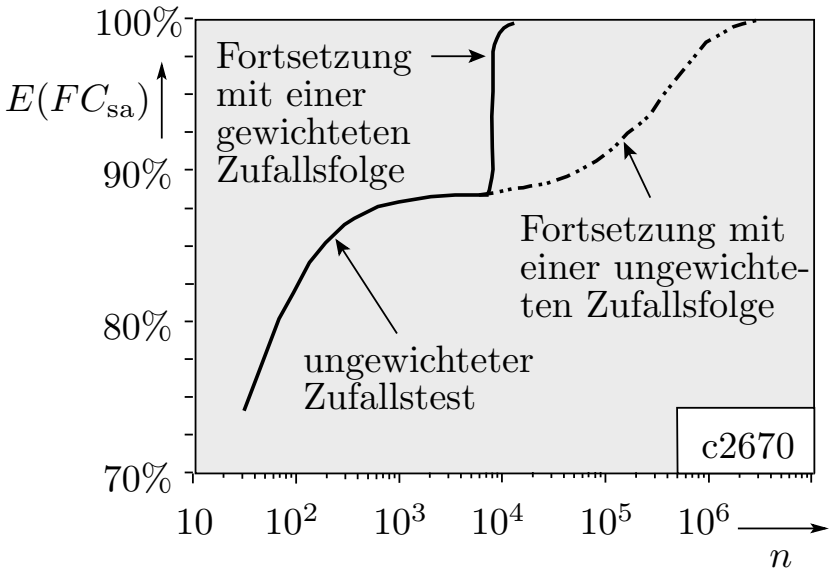
Gewichteter Zufallstest

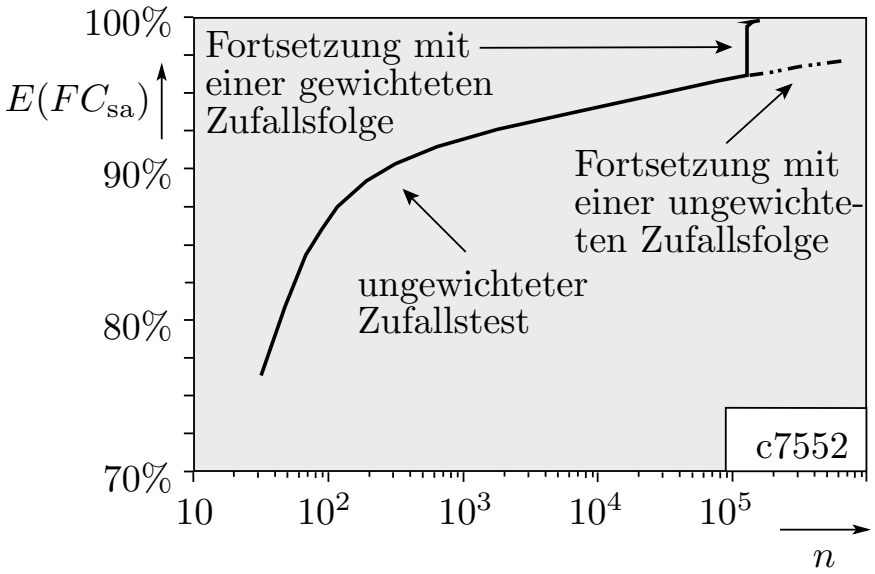
Außer durch weitere Modularisierung lässt sich die Fehlerüberdeckung von Zufallstests auch durch Änderung des Operationsprofils während des Tests erhöhen. Pragmatischer Ansatz:

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[5]

- 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.





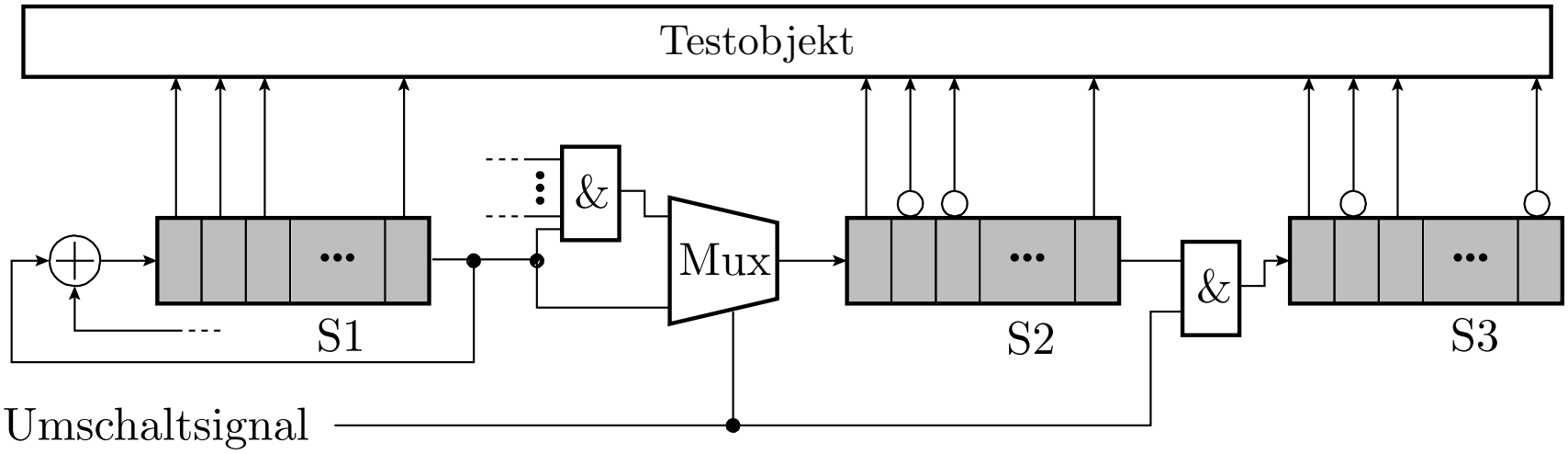
[5] Kombinatorische Benchmakschaltungen zum Vergleich von Testlösungen. Die Zahl hinter dem »c« ist die Anzahl der Signalleitungen.

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.



Nicht nennenswert aufwändiger als ohne Wichtung.

Testauswahl

Testauswahl

Für dynamische Tests ist eine Stichprobe von Service-Bedatungen auszuwählen. Bei guter Testbarkeit

- Verbindungstest mit Boundary-Scan,
- Test linearer Funktionen,
- Test kleiner Funktionsbausteine, ...

genügen kurze Zufallstests mit einfachen Vollständigkeitskontrollen (wird hier nicht weiter betrachtet).

Große ganzheitlich zu testende Systeme verlangen

- lange Testsätze,
- aufwändige Vollständigkeitskontrollen (Fehlersimulation) und
- bei zu geringen Fehlernachweiswahrscheinlichkeiten eine gezielte Testberechnung.

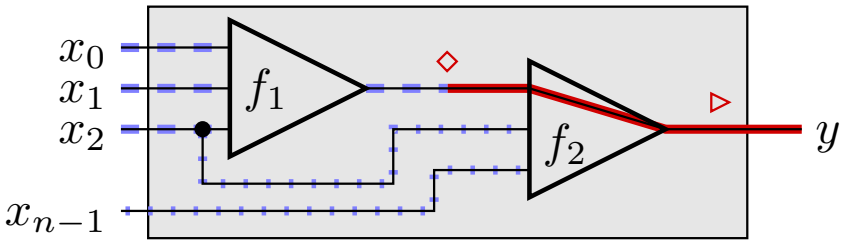
Aussagekräftige Vollständigkeitskontrollen basieren auf Annahmen möglicher Fehler und deren Nachweisbedingungen.

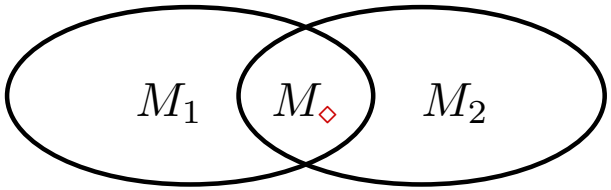
Modellfehler, Nachweismengen und -bedingungen

- Modellfehler: Beispielfehler mit exakt vorgegebenen Fehlverhalten (siehe Foliensatz F2).
- 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 Teil-Service, dieser ausgeführt und seine Ergebnisse beobachtbar sind.
- Ein Fehlermodell ist ein Algorithmus/Rezept/Regelwerk, mit dem für das Testobjekt eine Menge von Modellfehlern ausgewählt wird.
- Modellfehler dienen entweder zur Suche geeigneter Testeingaben oder zur Abschätzung der Fehlerüberdeckung für einen gegebenen (meist zufällig ausgewählten) Testsatz.

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

- den Fehler anregen[6] und
- einen Beobachtungspfad erzeugen, entlang dem sich die Verfälschung zu einem beobachtbaren Ausgang fortplanzt.





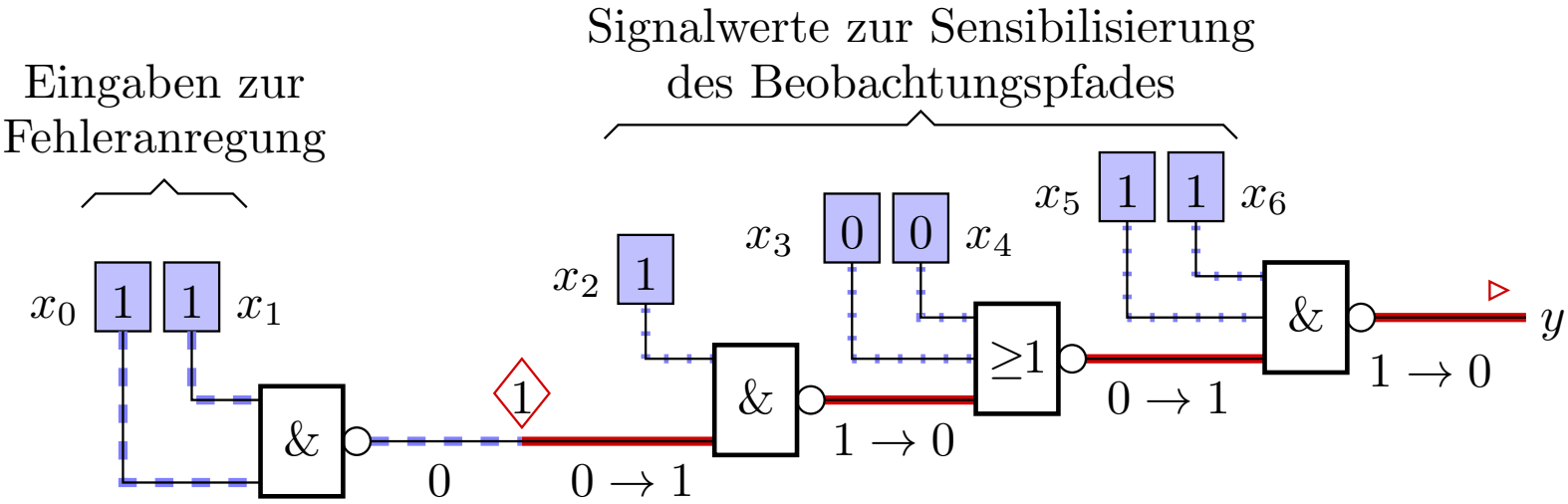
◇ Fehler
▷ Fehlfunktion (Datenverfälschung)
Eingaben zur Fehleranregung
Einstellen der Beobachtbarkeit
Beobachtungspfad

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.

[6] 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; $\varnothing$ – leere Menge).

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);
```

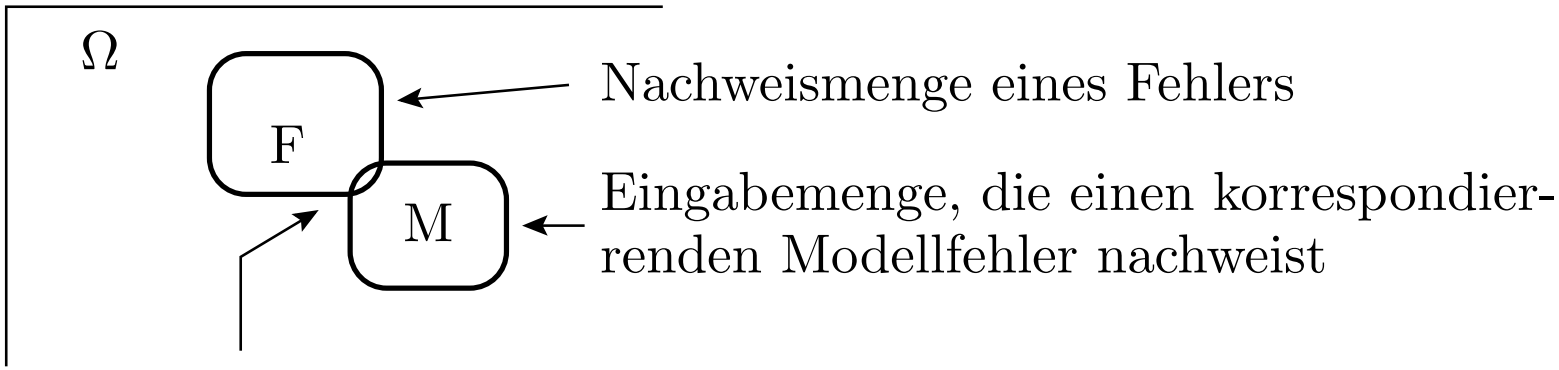
Identisch nachweisbare Modellfehler können zu einem Modellfehler zusammengefasst werden, im Beispiel M1 und M4,:

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

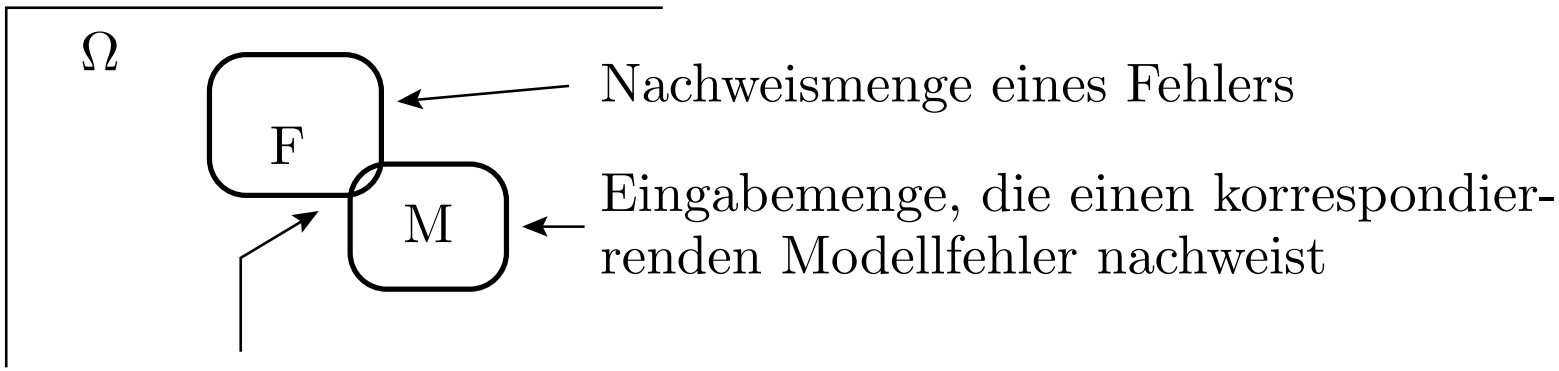
Ein Fehlermodell ist so zu wählen, dass die Modellfehlerüberdeckung, die sich bestimmen läßt, Rückschlüsse auf die tatsächliche Fehlerüberdeckung erlaubt. Der Zusammenhang zwischen tatsächlicher und Modellfehlerüberdeckung hängt von den Größenverhältnissen der Nachweismengen, deren Überschneidungen,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

Gezielte Testauswahl

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

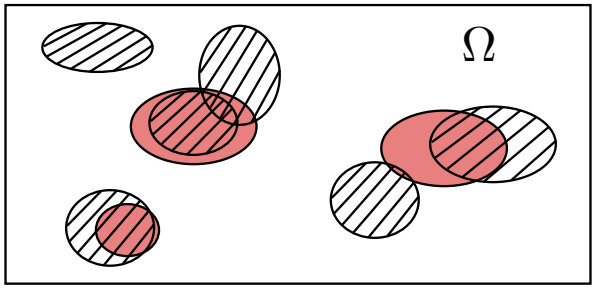


- Der Nachweis der unbekannten Fehler ist Zufall.
- Für alle Modellfehler, die sich Nachweisbedingungen mit einem Fehler teilen, impliziert jeder gefundene Test mit einer gewissen Wahrscheinlichkeit den Nachweis des Fehlers.



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

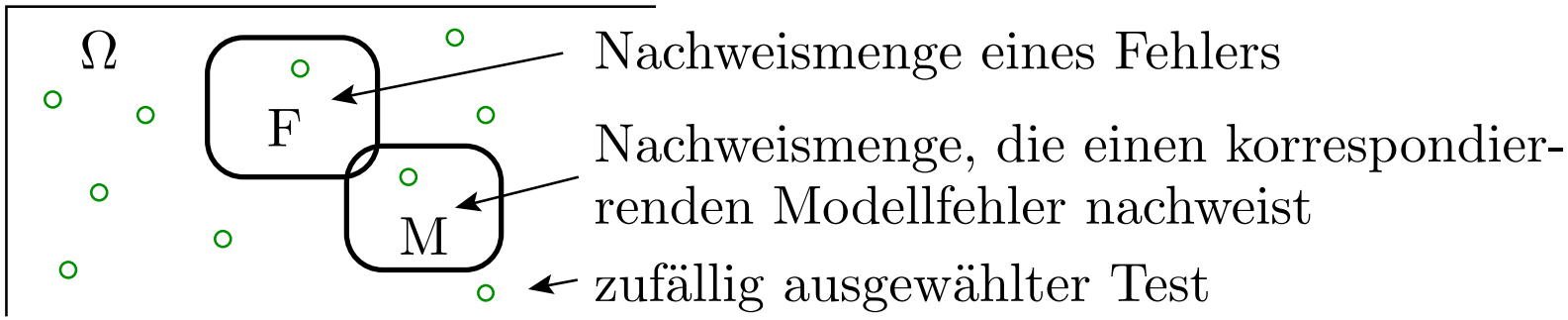
- Ein Fehlermodell erzeugt viele Modellfehler.
- Alle potenziellen Fehler sollten eine größere Nachweismengenüberscheidung mit mehreren Modellfehlern haben.
- Die Überschneidungen entstehen durch gleiche Anregungs- und Beobachtungsbedingungen.

Geziele Auswahl ist nur besser als eine zufällige Auswahl, wenn das Fehlermodell Modellfehler mit den Anregungs- und Beobachtungsbedingungen der zu erwartenden Fehler generiert.

Der verbreitete Ansatz, je Modellfehler genau einen Test zu suchen, ist nur zweckmäßig, wenn Modellfehler identisch mit tatsächlichen Fehler nachweisbar sind (praktisch nie der Fall).

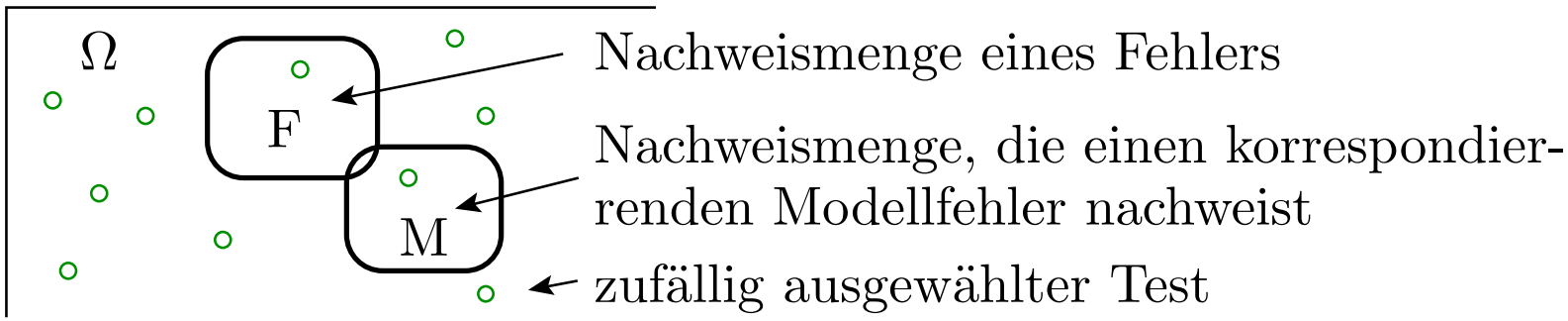
Zufällige Testauswahl

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



Gedankenmodell 1:

- Füllen des Eingaberaums so lange 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.

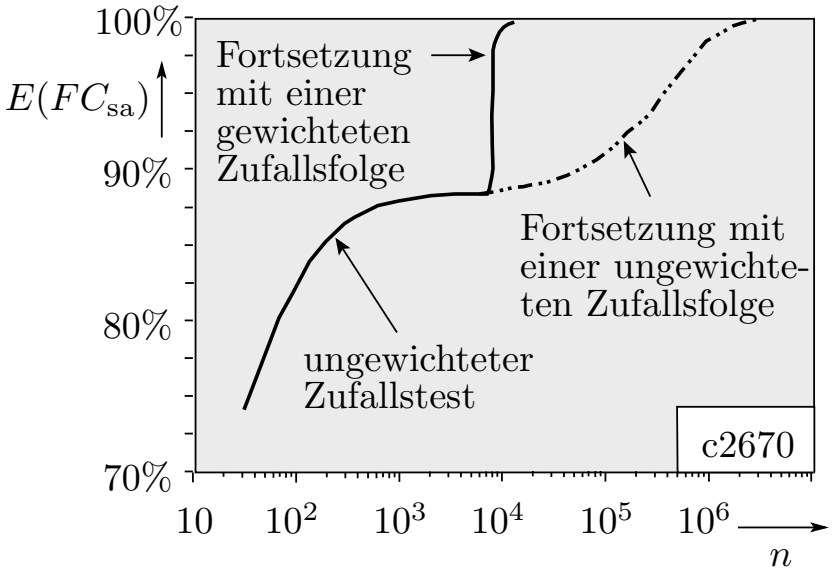
Mischformen

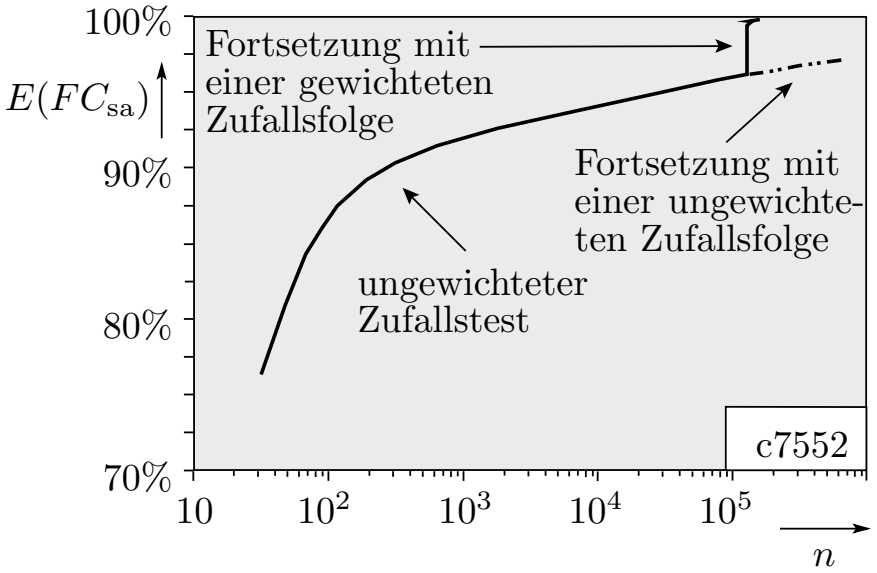
- Bevorzugung von Testeingaben, die auch Modellfehler nachweisen.
- Kombination eines Zufallstests, der die einfach nachweisbaren Modellfehler nachweist, mit gezielt ausgewählten Tests für die restlichen Modellfehler.
- Intuitive Auswahl basierend auf Erfahrungen über Fehler in anderen Objekten.
- ...

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

Experiment mit den Schaltungen c2670 und c7552[7]

- 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.





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

Verallgemeinerung des Verfahrens

- Zusammenstellen einer Modellfehlermenge
- Wiederhole für viele Zufallstests
  - Anhaken der nachweisbaren Fehler
- Wiederhole, bis genug Modellfehler abgehakt sind
  - Suche ein Operationsprofil, dass die restlichen Fehler besser nachweist
  - Wiederhole für viele Tests
    - Anhaken der nachweisbaren Fehler

Testsatzlänge ZT

Testsatzlänge von Zufallstests

Für Zufallstests beschränkt sich die Testauswahl auf die Festlegung der Testsatzlänge. Vorgaben:

1. Wertevorgabe, z.B. $n = 10^6$ Tests,
2. schwaches Überdeckungsziel, z.B. 100% Anweisungsüberdeckung,
3. Modellfehlerüberdeckung, z.B. 99% Haftfehlerüberdeckung,
4. tatsächliche Fehlerüberdeckung, z.B. $FC = 95\%$,
5. Zuverlässigkeit[8], z.B. $Z_\mathrm{T} = 1.000\,\mathrm{h}$.

2 – geringer Rechenaufwand; 3 – Fehlersimulation; 4,5 – wünschenswert, aber noch nicht Stand der Technik, Forschungsrichtung »FHNW-Funktion«.

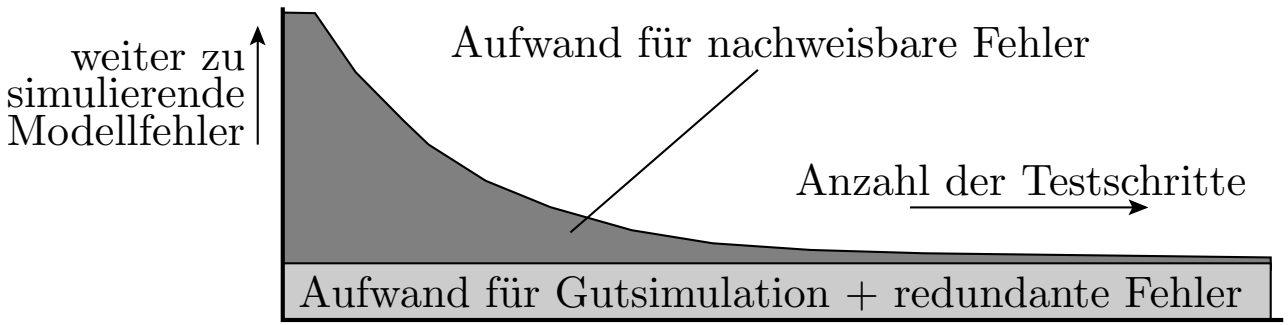
[8] Zuverlässigkeit im Sinne der mittleren Betriebsdauer oder abgearbeiteten Service-Anforderungen zwischen zwei Fehlfunktionen im Einsatz.

Fehlersimulation

Fehlersimulation für Systeme mit Gedächtnis:

```
Festlegung der zufälligen Testeingaben
Wiederhole für alle Testeingaben
    Bestimmung und Abspeicherung der Sollausgaben
Wiederhole für jeden Modellfehler
    Wiederhole für alle Testeingaben
        Bestimmung der Ausgabe und Soll/Ist-Vergleich
        Abbruch der Simulation des Modellfehlers bei Abweichung
```

Der Rechenaufwand wird vor allem durch die schlecht und nicht nachweisbaren (redundanten) Modellfehler bestimmt:



Für Systeme ohne Gedächtnis können auch die Fehler testschrittweise simuliert werden:

Festlegung der zufälligen Testeingaben
Wiederhole für alle Testeingaben
Bestimmung der Sollausgaben
Wiederhole für jeden nicht abgehakten Modellfehler
Bestimmung der Ausgabe und Soll/Ist-Vergleich
Abhaken des Modellfehlers bei Abweichung

Simulationsaufwand Überschlag[9]:

$$\text{Anz(Sim)} \approx \underbrace{n}_{\text{Gutsim.}} + \underbrace{n \cdot \varphi_{\text{mr}}}_{\text{red. F.}} + \underbrace{\int_{\frac{1}{n}}^{1} \frac{1}{p} \cdot H_{\text{M}}(p) \cdot dp}_{\text{nachweisbare Fehler}}$$

(n – Testsatzlänge; φ_{mr} – Anzahl der redundanten Fehler; $H_{\text{M}}(p)$ – FHNW-Funktion der Modelfehler). Mit einer Potenzfunktion als FHNW-Funktion:

$$k \cdot \varphi_{\text{m}} \cdot p^{k-1}$$

[9] Näherung, dass jeder Fehler genau $\frac{1}{p}$ mal simuliert wird.

$$k \cdot \varphi_{\mathrm{m}} \cdot p^{k-1}$$

φ_{m} – Anzahl der nachweisbaren Modellfehler; k – Exponent für die Abnahme der nichtnachweisbaren Fehler mit n:

$$\begin{aligned}
\mathrm{Anz(Sim)} &\approx n + n \cdot \varphi_{\mathrm{mr}} + k \cdot \varphi_{\mathrm{m}} \cdot \int_{\frac{1}{n}}^{1} p^{k-2} \cdot dp \\
&= n + n \cdot \varphi_{\mathrm{mr}} + \frac{k \cdot \varphi_{\mathrm{m}}}{k-2} \cdot \left(1 - \left(\frac{1}{n}\right)^{k-1}\right) \\
&\approx n \cdot (1 + \varphi_{\mathrm{mr}}) + \frac{k}{2-k} \cdot \varphi_{\mathrm{m}} \cdot n^{1-k}
\end{aligned}$$

Aufwandsabschätzung für eine Digitalschaltung aus 10^4Gattern. FHNW-Funktion: $k = 0{,}5$, $\varphi_{\mathrm{m}} = 10^4$. Anzahl der redundanten Fehler $\varphi_{\mathrm{mr}} = 10^2$. Anzahl der simulierten Tests: $n = 10^8$:

$$\mathrm{Anz(SimGatter)} \approx \underbrace{10^8 \cdot \left(1 + 10^2\right)}_{\text{redundante F.: } 10^{10}} + \underbrace{\frac{0{,}5}{2 - 0{,}5} \cdot 10^4 \cdot \left(10^8\right)^{1-0{,}5}}_{\text{nachweisbare Fehler: } 0{,}33 \cdot 10^8}$$

Erforderliche Modellfehleranzahl

- Die redundanten Modellfehler seien vorher aussortiert.
- Der Schätzer $\varphi_{\mathrm{NErk}} = \varphi_{\mathrm{NErk\_sim}}$ für nicht nachgewiesene Fehler hat etwa die Standardabweichung $\sqrt{\varphi_{\mathrm{NErk\_sim}}}$.
- Eine Obergrenze $\varphi_{\mathrm{NErk\_max}}$ verlangt als Richtwert:

$$\varphi_{\mathrm{NErk\_sim}} + 1 \ldots 3 \cdot \sqrt{\varphi_{\mathrm{NErk\_sim}}} < \varphi_{\mathrm{NErk\_max}}$$

- Simulationsabbruchkriterium $\varphi_{\mathrm{NErk\_sim}} = 10$ kann etwa für eine Obergrenze von $\varphi_{\mathrm{NErk\_max}} \leq 15 \ldots 20$ garantieren.
- Simulation mit 2000 Modellfehlern und $\varphi_{\mathrm{NErk\_sim}} = 10$ garantiert etwa für $FC \geq 99\%$.
- Von den ausgewählten Modellfehlern sollten zumindest die schlecht nachweisbaren Fehler unabhängig voneinander nachweisbar sein (siehe Foliensatz F2, Abschn. 3.5 Fehler und Fehlfunktionen. Nachweisabhängigkeiten). Forschungsbedarf.

Fehler- und Modellfehlerüberdeckung

Der Zusammenhang zwischen der Fehler- und der Modellfehlerüberdeckung basiert auf Ähnlichkeiten der FHNW-Funktionen. Idealerweise unterscheiden sich beide Funktionen nur in der Skalierung der Häufigkeits- und der Wahrscheinlichkeitsachse:

$$H_{\mathrm{M}}(p) \approx b \cdot H(c \cdot p)$$

($H(p)$ – FHNW-Funktion der Fehler; $H_{\mathrm{M}}(p)$ – der Modellfehler; b – Skalierung Häufigkeitsachse; c – Skalierung der Wahrscheinlichkeitsachse). Resultierende Beziehungen zwischen den zu erwartenden Fehlerüberdeckungen:

$$E(FC) \approx 1 - \frac{\int_0^1 H(p) \cdot e^{-np} \cdot dp}{\int_0^1 H(p) \cdot dp}$$

$$E(FC_{\mathrm{M}}) \approx 1 - \frac{\int_0^1 b \cdot H(c \cdot p) \cdot e^{-np} \cdot dp}{\int_0^1 b \cdot H(c \cdot p) \cdot dp}$$

b kürzt ich heraus. Substitution $q = c \cdot p$

$$E\left(FC_{\mathrm{M}}\right) \approx 1 - \frac{\int_0^{\frac{1}{c}} H\left(q\right) \cdot e^{-\frac{n}{c} \cdot q} \cdot \frac{dq}{c}}{\int_0^{\frac{1}{c}} H\left(q\right) \cdot \frac{dq}{c}}$$

Wenn für $p \gtrapprox \frac{1}{c}$ (einfach zu erkennende Fehler) $H\left(p\right) = 0$ gilt (sie sind im Vorfeld beseitigt bzw. werden bei der Simulation nicht berücksichtigt), ergibt sich:

$$E\left(FC_{\mathrm{M}}\right) \approx FC\left(\frac{n}{c}\right)$$

Fakt 1

Die Modellfehlerüberdeckung ist die Fehlerüberdeckung der c-fachen Testsatzlänge.

Potenz-FHNW-Funktion

FNHW-Funktionen, wenn die mit n_0 Zufallstests nachweisbaren Fehler beseitigt sind ($H(p) = 0$ für $p \gtrapprox \frac{1}{c}$):

$$\begin{aligned} H(p) &= \varphi_0 \cdot k \cdot p^{k-1} \cdot e^{-n_0 \cdot p}; \\ H_{\mathrm{M}}(p) &= b \cdot \varphi_0 \cdot k \cdot (c \cdot p)^{k-1} \cdot e^{-n_0 \cdot p} \end{aligned}$$

Fehler- und Modellfehlerüberdeckung:

$$E(FC) \approx 1 - \frac{\int_0^1 \varphi_0 \cdot k \cdot p^{k-1} \cdot e^{-(n+n_0) \cdot p} \cdot dp}{\int_0^1 \varphi_0 \cdot k \cdot p^{k-1} \cdot e^{-n_0 \cdot p} \cdot dp} \approx 1 - \left(\frac{n+n_0}{n_0} \right)^{-k}$$

$$E(FC_{\mathrm{M}}) \approx 1 - \frac{\int_0^1 \varphi_0 \cdot k \cdot (c \cdot p)^{k-1} \cdot e^{-(n+n_0) \cdot p} \cdot dp}{\int_0^1 \varphi_0 \cdot k \cdot (c \cdot p)^{k-1} \cdot e^{-n_0 \cdot p} \cdot dp} \approx 1 - \left(\frac{n+n_0}{n_0} \right)^{-k}$$

(0<k <1; Hinweise zur Herleitung siehe nächste Folie). In dem Verhältnis $\frac{n}{n_0}$ kürzt sich ein eventueller Skalierungsfaktor c heraus, also kein Widerspruch zur Folie zuvor.

Herleitung der Gleichungen auf der Folie zuvor

In

$$\int_0^1 \varphi_0 \cdot k \cdot p^{k-1} \cdot e^{-n \cdot p} \cdot dp$$

holt die Substitution $p = \frac{x}{n}$; $dp = \frac{dx}{n}$ die Testdauer n aus dem Integral:

$$\varphi_0 \cdot k \cdot n^{-k} \underbrace{\int_0^n x^{k-1} \cdot e^{-x} \cdot dx}_{\approx \Gamma(k) \approx k^{-1}} \approx \varphi_0 \cdot n^{-k}$$

Das Restintegral stebt für große n gegen die Gamma-Funktion $\Gamma(k)$ und diese für $0 < k \leq 1$ gegen $1/k$. In den Gleichungen der Folie zuvor ist n durch $n + n_0$ bzw n_0 zu ersetzen:

$$E(FC) \approx 1 - \frac{\int_0^1 \varphi_0 \cdot k \cdot p^{k-1} \cdot e^{-(n+n_0) \cdot p} \cdot dp}{\int_0^1 \varphi_0 \cdot k \cdot p^{k-1} \cdot e^{-n_0 \cdot p} \cdot dp} \approx 1 - \left(\frac{n + n_0}{n_0} \right)^{-k}$$

$$E(FC_{\mathrm{M}}) \approx 1 - \frac{\int_0^1 b \cdot \varphi_0 \cdot k \cdot (c \cdot p)^{k-1} \cdot e^{-(n+n_0) \cdot p} \cdot dp}{\int_0^1 b \cdot \varphi_0 \cdot k \cdot (c \cdot p)^{k-1} \cdot e^{-n_0 \cdot p} \cdot dp} \approx 1 - \left(\frac{n + n_0}{n_0} \right)$$

Zuverlässigkeit und Testzeit

Es sei angenommen, dass Betrieb und Test mit gleichem Operationsprofil erfolgen und ein Fehler a-mal beim Test beobachtbar sein muss, bevor er erfolgreich beseitigt wird[10]. Beseitigungswahrscheinlichkeit:

$$p_{i.\text{Beseit}}(n) = 1 - e^{-\frac{n \cdot p_{i.\text{nachw}}}{a}}$$

Abnahme der Wahrscheinlichkeit des Fehlervorhandenseins:

$$p_{i.\text{vorh}}(n) = p_{i.\text{vorh}} \cdot (1 - p_{i.\text{Beseit}}(n)) = p_{i.\text{vorh}} \cdot e^{-\frac{n \cdot p_{i.\text{nachw}}}{a}}$$

Wahrscheinlichkeit einer Fehlfunktion durch alle versteckten Fehler, wenn das Systeme mit n Schritten getestet und erkennbare Fehler im Mittel nach dem a-ten Auftreten beseitigt werden:

$$p_{\text{SF.Fehler}} = \sum_{i=1}^{\text{Anz(PF)}} \left(p_{i.\text{vorh}} \cdot e^{-\frac{n \cdot p_{i.\text{nachw}}}{a}} \cdot p_{i.\text{nachw}} \right)$$

[10] Tests beim Hersteller $a \approx 1$, Nutzerbetrieb als Test $a \gg 1$, abhängig wie häufig/gut der Feedback zum Hersteller ist.

Mit der FHNW-Funktion

$$p_{\text{SF.Fehler}}(n) = \int_0^1 p \cdot H(p) \cdot e^{-\frac{n \cdot p}{a}} \cdot dp$$

Mittlere Anzahl von Service-Leistungen zwischen zwei fehlerbedingten Fehlfunktionen:

$$Z_{\text{n.Fehler}}(n) = \frac{1}{p_{\text{SF.Fehler}}(n)} = \frac{1}{\int_0^1 p \cdot H(p) \cdot e^{-\frac{n \cdot p}{a}} \cdot dp}$$

Mit

$$H_{\text{M}}(p) \approx b \cdot H(c \cdot p)$$

$$Z_{\text{n.MFehler}}(n) = \frac{1}{b \cdot \int_0^1 p \cdot H(c \cdot p) \cdot e^{-\frac{n \cdot p}{a}} \cdot dp}$$

Substitution $q = c \cdot p$:

$$Z_{\text{n.MFehler}}(n) = \frac{1}{\frac{b}{c} \cdot \int_0^{\frac{1}{c}} p \cdot H(q) \cdot e^{-\frac{n \cdot q}{c \cdot a}} \cdot dq}$$

$$Z_{\text{n.MFehler}}(n) = \frac{1}{\frac{b}{c} \cdot \int_0^{\frac{1}{c}} p \cdot H(q) \cdot e^{-\frac{n \cdot q}{c \cdot a}} \cdot dq}$$

Für $H(p) = 0$ für $p \gtrapprox \frac{1}{c}$ (einfach zu erkennende Fehler werden im Vorfeld beseitigt bzw. bei der Simulation nicht berücksichtigt):

$$Z_{\text{n.MFehler}}(n) \approx \frac{c}{b} \cdot Z_{\text{n.Fehler}}\left(\frac{n}{c}\right)$$

Fakt 2

Die mit Modellfehlern schätzbare Zuverlässigkeit ist die $\frac{c}{b}$-fache Zuverlässigkeit der $\frac{1}{c}$– fachen Reifedauer.

Potenzfunktion als FNHW-Funktion

Bei einer Potenz-FNHW-Funktion und wenn alle mit n_0 Zufallstests nachweisbaren Fehler beseitigt sind:

$$\begin{aligned} H(p) &= \varphi_0 \cdot k \cdot p^{k-1} \cdot e^{-n_0 \cdot p} \\ H_{\mathrm{M}}(p) &= b \cdot \varphi_0 \cdot k \cdot (c \cdot p)^{k-1} \cdot e^{-n_0 \cdot p} \end{aligned}$$

nimmt die fehlerbezogenen Zuverlässigkeit sowohl für reale als auch für Modellfehler mit der $k+1$-ten Potenz der Testdauer zu:

$$\begin{aligned} Z_{\mathrm{n.Fehler}}(n) &= Z_{\mathrm{n.Fehler}}(n_0) \cdot \left(\frac{n}{n_0}\right)^{k+1} \\ Z_{\mathrm{n.MFehler}}(n) &= Z_{\mathrm{n.MFehler}}(n_0) \cdot \left(\frac{n}{n_0}\right)^{k+1} \end{aligned}$$

Die Skalierungsfaktoren a bis c kürzen sich dabei heraus.

Zunahme der Fehlerüberdeckung mit der Testsatzlänge bei einer Potenz-FNHW-Funktion

Der Anteil der nicht nachweisbaren Fehler veringert sich umgekehrt proportional zur k-ten Potenz der Testsatzlänge $(0 < k < 1)$:

$$E(FC) \approx 1 - \left(\frac{n + n_0}{n_0}\right)^{-k}$$

Eine Verringerung des zu erwartenden Anteils der nicht nachweisbaren Fehler 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.

Die Fehlerüberdeckung lässt sich über die Testsatzlänge nicht beliebig steigern.

Zunahme der fehlerbezogenen Zuverlässigkeit mit der Testsatzlänge bei einer Potenz-FNHW-Funktion

Die fehlerbezogene Zuverlässigkeit als die mittlere Anzahl von fehlerfrei abgearbeiteten Service-Leistungen zwischen zwei fehlerbedingten Fehlfunktionen nimmt mit der $k+1$-ten Potenz der Testdauer zu:

$$Z_{\text{n.Fehler}}(n) = Z_{\text{n.Fehler}}(n_0) \cdot \left(\frac{n}{n_0}\right)^{k+1}$$

(n_0 – Bezugstestsatzlänge). Die zehnfache Zuverlässigkeit verlangt

- bei $k = 1$ die 3,2-fache,
- bei $k = 0{,}5$ die 4,6-fache und
- bei $k = 0{,}333$ die 5,6-fache Testsatzlänge.

Zuverlässigkeitsanforderungen lassen sich einfacher als Anforderungen an die Fehlerüberdeckung befriedigen.

Abschlussdiskussion

Falls in Zukunft FHNW-Funktionen als Werkzeug für die Testsatzlängenabschätzung akzeptiert werden, ist eine Abschätzung der erforderlichen Testsatzlänge auf Basis einer Vorgabe

- einer Mindestfehlerüberdeckung oder
- einen Mindestwert für der fehlerbezogenen Teilzuverlässigkeit

möglich. Mit der Annahme einer Potenzfunktion als FHNW-Funktion ist das sogar recht einfach.

Inspektion (Review)

Inspektion (Review)

Kontrolltätigkeit, Sichtprüfung (von lat. inspicere = besichtigen, betrachten). Anwendbar auf:

- Dokumente (Spezifikationen, Nutzerdokumentationen, ...),
- Programmcode, Testausgaben,
- Schaltungsbeschreibungen,
- Sichtprüfung gefertigter Schaltungen.

Manuelle Inspektion von Entwurfsbeschreibungen:

- Statischer Test,
- zufälliger Fehlernachweis mit subjektiv geprägter Güte,
- auch für den Nachweis nicht funktionaler Fehler,
- auch für frühe Entwurfsphasen geeignet.

Kenngrößen einer Inspektion

Inspektionsfehlerüberdeckung:

$$IFC = \frac{\varphi_{\text{Erk}}}{\varphi}$$

(φ_{Erk} – Anzahl der nachweisbaren; φ – Anzahl aller (entstandenen) Fehler). Lässt sich insgesamt oder getrennt für funktionale und sonstige Fehler angeben.

Abschätzmöglichkeiten:

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

Weitere Bewertungsgrößen für Inspektionen nach [6]:

- Effizienz: Gefundene Abweichungen pro Mitarbeiterstunde.
- Effektivität: Gefundene Abweichungen je 1000 NLOC[11].

[11] NLOC: Anzahl der Nettocodezeilen (Codezeilen ohne Kommentar- und Leerzeilen).

Beispielbewertung einer Inspektion

- Programmgröße: 10.000 NLOC.
- Arbeitsaufwand: 200 Stunden.
- 228 gefundene Fehler, davon 156 funktionale.
- Geschätzte Gesamtfehleranzahl: 300, davon 200 funktionale.

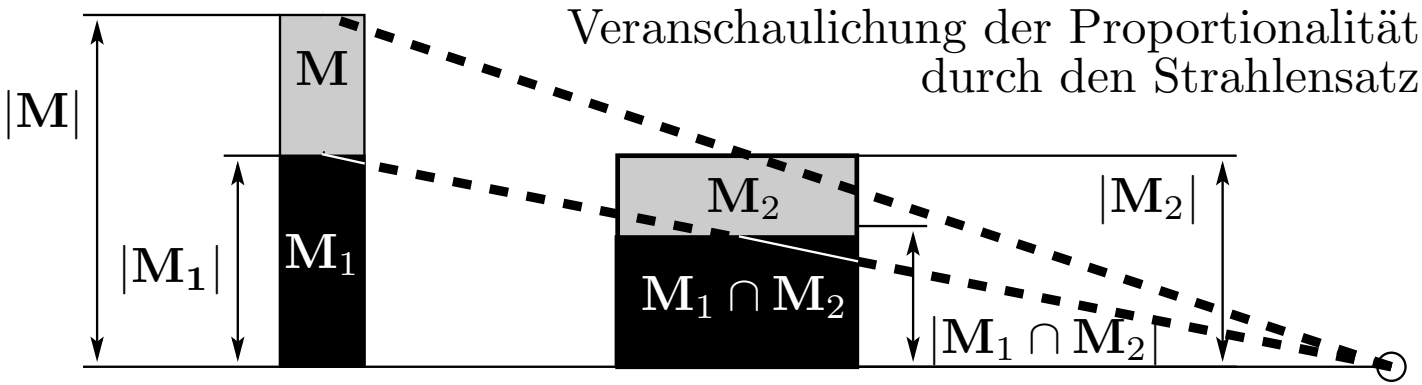
	gesamt	funktionale Fehler	sonstige Fehler
IFC	$\frac{228}{300}$	$\frac{156}{200}$	$\frac{72}{100}$
Effizienz	$\frac{228}{10.000}$	$\frac{156}{10.000}$	$\frac{72}{10.000}$
Effektivität	$\frac{228}{200\,\mathrm{h}}$	$\frac{156}{200\,\mathrm{h}}$	$\frac{72}{200\,\mathrm{h}}$

Effizienz und Effektivität ergeben sich direkt aus den Zählwerten. Die Inspektionsfehlerüberdeckung IFC verlangt eine Abschätzung der Anzahl der nicht gefundenen Fehler.

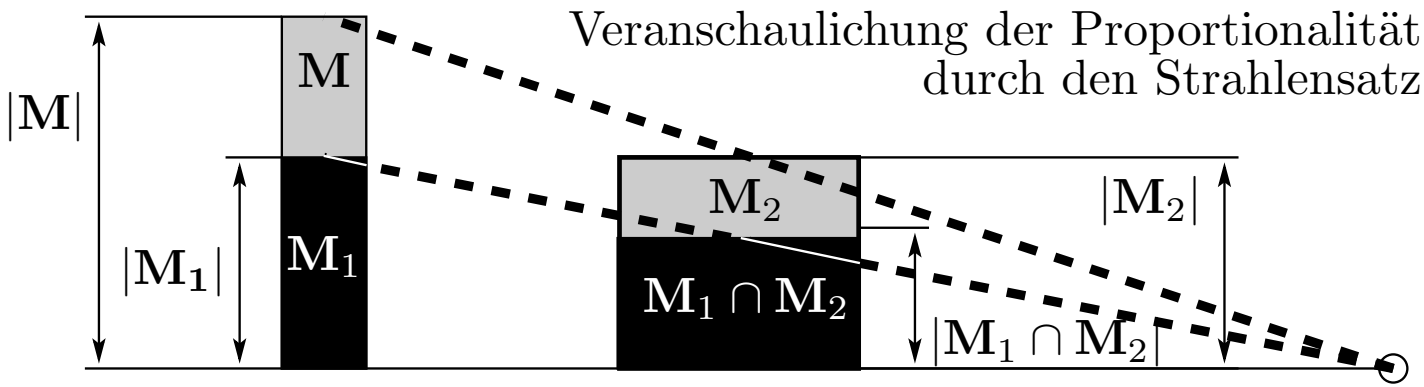
Capture-Recapture-Verfahren

Abgeleitet von einem Schätzer für die Größe von Tierpopulationen (z.B. Vögel in einem Gebiet) [3, 8, 7].

- Aus einer Menge $\mathbf{M}$ unbekannter Größe wird eine Menge $\mathbf{M}_1$ von Tieren eingefangen, gekennzeichnet und freigelassen.
- Nach Vermischung der Population eine zweite Menge $\mathbf{M}_2$ von Tieren einfangen und gekennzeichnete Tiere zählen.
- Bei tierunabhängiger Einfangwahrscheinlichkeit ist der Anteil der Tiere, die beim zweiten Einfangen gekennzeichnet sind...



(|...| – Größe der Menge.)



(beide Male eingefangen wurden) etwa gleich dem Anteil der gekennzeichneten Tiere:

$$\frac{|\mathbf{M}_1|}{|\mathbf{M}|} \approx \frac{|\mathbf{M}_1 \cap \mathbf{M}_2|}{|\mathbf{M}_2|}$$

($\mathbf{M}$ – Menge aller Tiere, $\mathbf{M}_1$, $\mathbf{M}_2$ – beim ersten bzw. zweiten mal eingefangene Tiere; $\mathbf{M}_1 \cap \mathbf{M}_2$ – Menge der beide Male eingefangenen Tiere). Geschätzte Größe der Tierpopulation:

$$|\mathbf{M}| \approx \frac{|\mathbf{M}_1| \cdot |\mathbf{M}_2|}{|\mathbf{M}_1 \cap \mathbf{M}_2|}$$

Fehler statt Tiere

Zwei Inspekteure i finden jeweils eine Menge von $\mathbf{M}_i$ Fehlern:

$$|\mathbf{M}| \approx \frac{|\mathbf{M}_1| \cdot |\mathbf{M}_2|}{|\mathbf{M}_1 \cap \mathbf{M}_2|}$$

($|\mathbf{M}_1 \cap \mathbf{M}_2|$ – Anzahl der von beiden Inspekteuren unabhängig voneinander gefundenen Fehler; $|\mathbf{M}|$ – geschätzte Anzahl der vorhandenen Fehler). Die geschätzte Fehlerüberdeckung ist das Verhältnis der Anzahl der insgesamt von beiden Inspekteure erkannten Fehler $|\mathbf{M}_1 \cup \mathbf{M}_2|$ zur geschätzten Geamtfehleranzahl $|\mathbf{M}|$:

$$IFC = \frac{\varphi_{\text{Erk}}}{\varphi} \approx \frac{|\mathbf{M}_1 \cup \mathbf{M}_2|}{|\mathbf{M}|} \approx \frac{|\mathbf{M}_1 \cap \mathbf{M}_2| \cdot |\mathbf{M}_1 \cup \mathbf{M}_2|}{|\mathbf{M}_1| \cdot |\mathbf{M}_2|}$$

Gebunden an die Annahmen:

- Inspekteure erkennen die Fehler unabhäng voneinander,
- Gleiche Erkennungswahrscheinlichkeiten aller Fehler.

Beispiel: Inspektionsergebnisse für ein Programm:

- Inspekteur 1: 228 gefundene Fehler, davon 156 funktionale.
- Inspekteur 2: 237 gefundene Fehler, davon 163 funktionale.
- Schnittmenge: 105 Fehler, davon 73 funktionale.

Geschätzte Anzahl der vorhandenen Fehler:

$$\varphi = |\mathbf{M}| = \frac{|\mathbf{M}_1| \cdot |\mathbf{M}_2|}{|\mathbf{M}_1 \cap \mathbf{M}_2|}$$

Inspektionsfehlerüberdeckung:

$$IFC \approx \frac{|\mathbf{M}_1 \cap \mathbf{M}_2| \cdot |\mathbf{M}_1 \cup \mathbf{M}_2|}{|\mathbf{M}_1| \cdot |\mathbf{M}_2|}$$

Fehler	$\lvert\mathbf{M}_1\rvert$	$\lvert\mathbf{M}_2\rvert$	$\lvert\mathbf{M}_1 \cup \mathbf{M}_2\rvert$	$\varphi = \lvert\mathbf{M}\rvert$	IFC
alle	228	237	105	515	70%
funktional	156	163	73	348	71%
sontige	72	74	32	166	68%

Vertrauenswürdigkeit der Schätzung

Der Schätzwert für die IFC ist umgekehrt proportional zur Anzahl der von beiden Inspekteuren gefundenen Fehler. Wenn beide Inspekteure genau dieselben Fehler finden, ergibt sich $IFC = 100\%$. Das zweimal genau dieselben Fehler gefunden werden, kann aber auch andere Ursachen haben:

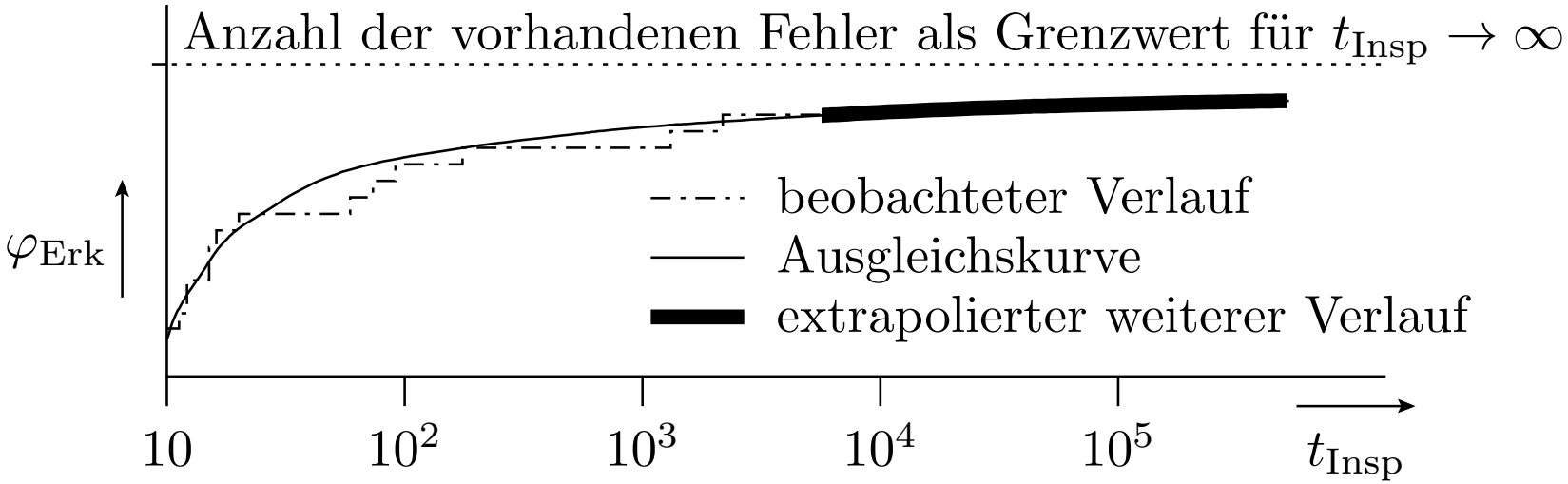
- Gegenseitig Information über die gefundenen Fehler.
- Die gefundenen Fehler waren viel leichter als die übrigen Fehler zu finden.
- Die gefundenen Fehler gehören zu »den üblichen Verdächtigen«, nach den beide Inspekteuren am gründlichsten gesucht haben, ...

Schätzwerte der IFC nach dem CR-Verfahren können mit erheblichen systematischen Fehlern behaftet sein.

Inspektion als Zufallstest

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

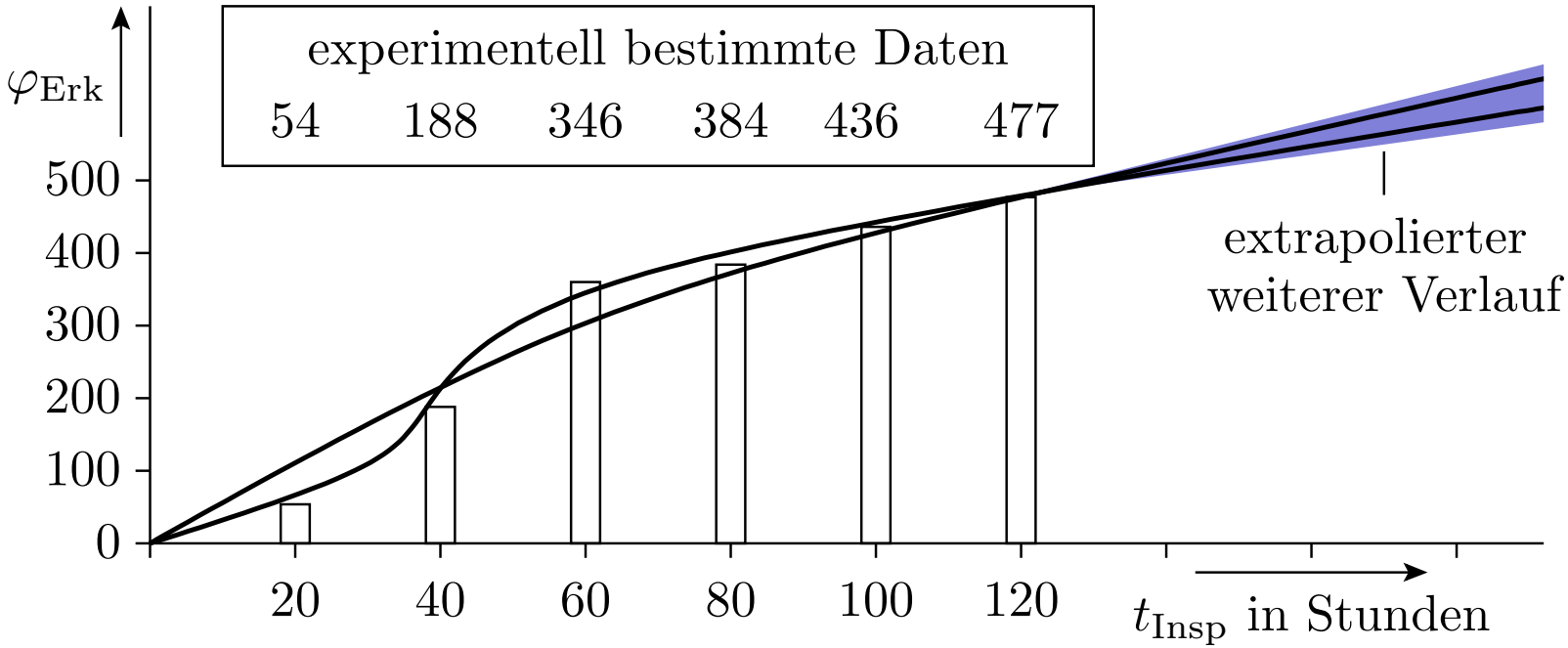
- Aufzeichnung der Anzahl der gefundenen Fehler in Abhängigkeit von der Inspektionsdauer.
- Abschätzen des weiteren Verlaufs.
- Gesamtfehleranzahl ist der Grenzwert für eine unendliche Inspektionsdauer:



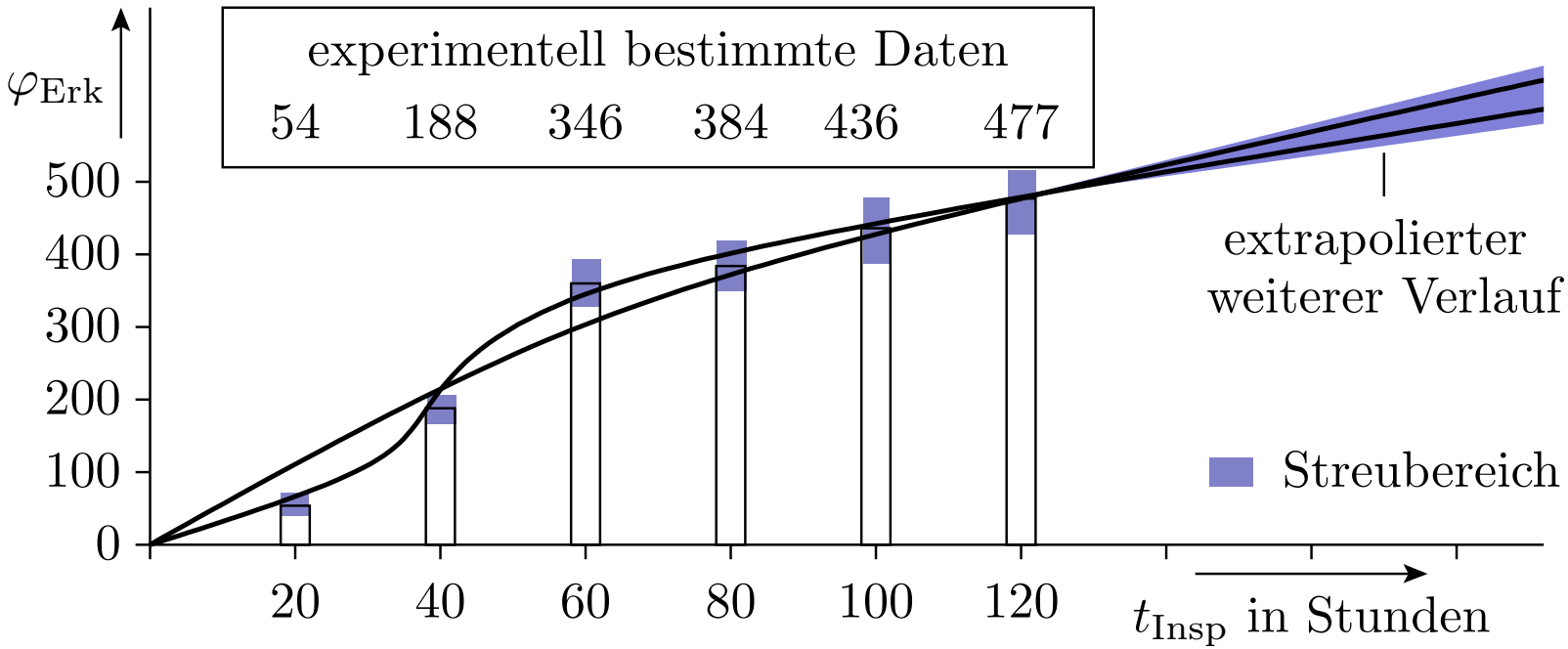
Experiment mit einem Inspekteur [12]

Inspektion des Buchmanuskripts [5] und der Beispielprogramme dazu

- Anzahl der gefunden Fehler in Abhängigkeit von der Inspektionsdauer.



[12] Bachelor-Arbeit von Yu Hong.



- Die experimentellen Daten sind Zufallsgrößen (Standardabw. abschätzungsweise[13] $\sqrt{\min\left(\varphi_{\text{Erk}}, \varphi_{\text{NErk}}\right)}$).
- Unterschiedliche Approximationsmöglichkeiten des Zeitverlaufs, z.B.

$$\varphi_{\text{NErk}}\left(t_{\text{Insp}}\right) = \varphi_{\text{NErk}}\left(t_0\right) \cdot \left(\frac{t_{\text{Insp}}}{t_0}\right)^k$$

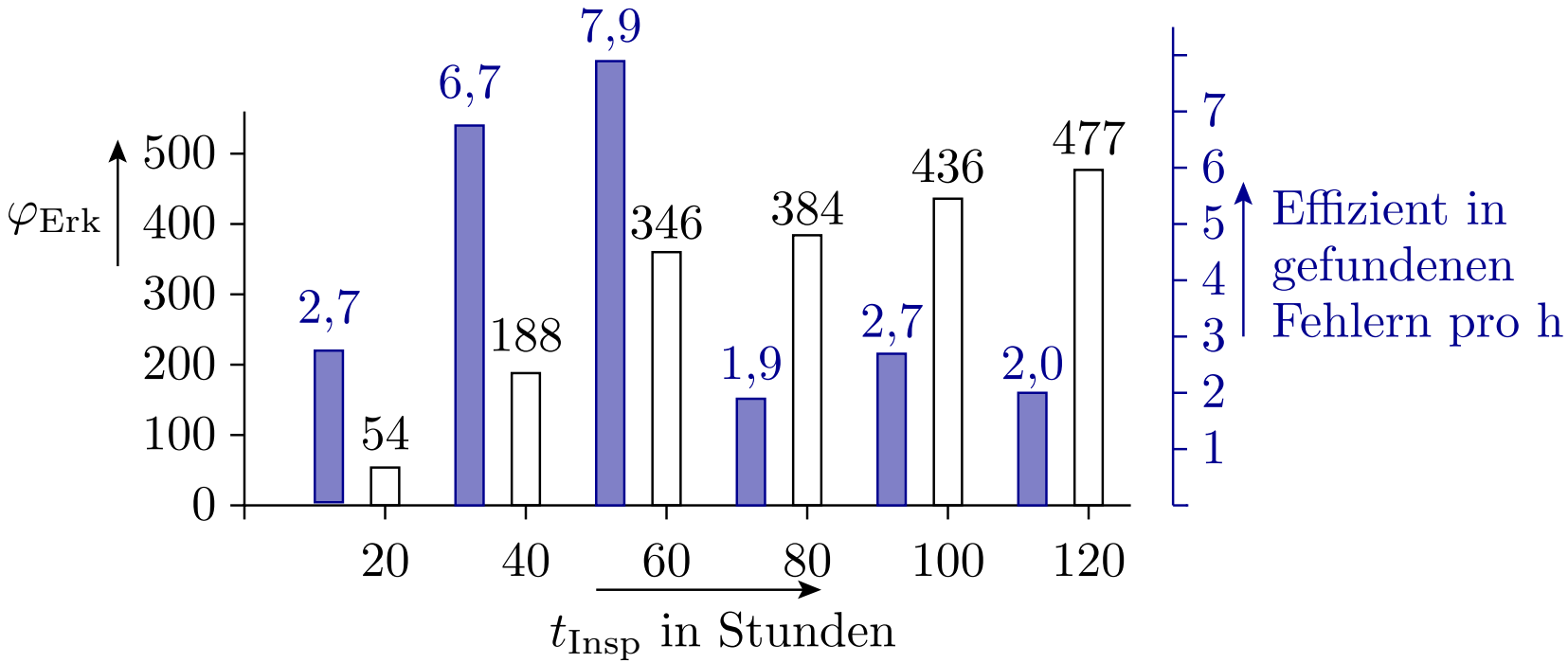
Vermutlich nicht vertrauenswürdiger als CR-Verfahren.

[13] Obergrenze Standardabweichung Binomialverteilung.

Unschiede zwischen Inspektion und Zufallstest

Menschen unterscheiden sich von Testmaschinen:

- Anlernphase: Zunahme der Effizienz mit t_{Insp}.



- Ermüdungsphase: Inspekteur wird »blind« für die verbleibenden Fehler.

- Bei mehrfachen Durchlesen nahm im Experiment nicht nur die Effizienz, sondern auch der dafür erforderliche Zeitaufwand deutlich ab.

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

- Nach den ersten drei Inspektionen des Datenmaterials war der Inspekteur offenbar »verbraucht« (blind für Fehler).

Organisation von Inspektionen

Für die effiziente Durchführung von Inspektionen sind auch psychologische Aspekte zu berücksichtigen:

- Arbeit »geschickt« auf mehrere Inspekteure verteilen.
- Einarbeitungs- und Ermüdungsphasen minimieren.
- Wenn Inspekteur ungleich Autor, zusätzliche Know-How-Weitergabe.
- »Vier Augen sehen mehr als zwei«. (Form der Diversität).

Inspektion ist eine aktuelle Forschungsrichtung:

- Formen der effiziente Durchführung und
- Gütebewertung

Publikationen mit dem Modellansatz »Zufallstest« bisher noch nicht gefunden. Interessant zu untersuchen wäre auch die Kombination CR-Verfahren und Zufallstest.

Aufgaben

Aufgabe 4.1: Statischer oder dynamischer Test?

Beispiel Programmieraufgabe in einem Praktikum:

1. Erstellen der Aufgabenstellung: Korrekturlesen durch den Lehrenden.
2. Beispielimplementierung durch den Lehrenden, Syntaxtest.
3. Beispiele ausprobieren.
4. Lösungssuche durch den Studierenden und Diskussion über die Lösung mit dem Praktikumspartner.
5. Programmieren und Syntaxtest,
6. Ausprobieren.
7. Praktikumsbetreuer schaut sich den Code an.
8. Praktikumsbetreuer lässt sich Beispiel vorführen.

Klassifizieren Sie, welche der Tests in den Ablaufschritten 1 bis 8 statisch und welche dynamisch sind.

Aufgabe 4.2: Entwicklung eines Testrahmens

Testobjekt sei das C-Programm für die Wurzelberechnung aus der Hausübung:

```
uint8_t wurzel(uint16_t x){
      uint8_t w=0;
      uint16_t sum=0;
      while (sum<x){sum += (w<<1)+1;
      w++;}
      return w;
     }
```

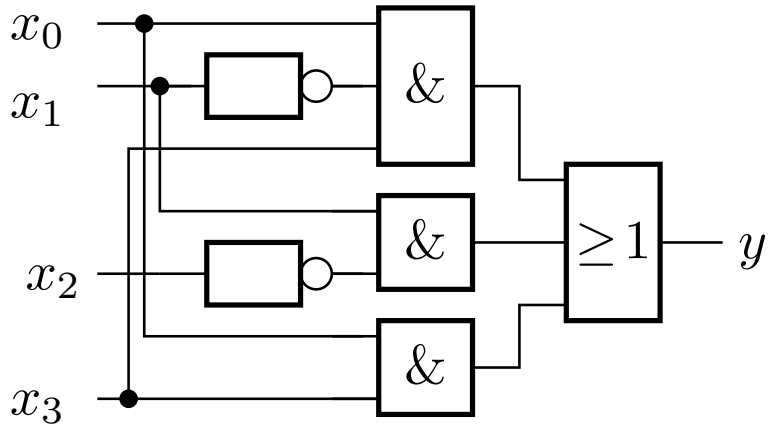
1. Scheiben Sie einen Testrahmen, der das Programm mit 1000 zufälligen Werten testet. Ergebniskontrolle mit der Probe:

$$y^2 \leq x < (y+1)^2$$

   Protokollierung aller x und y , die die Probe nicht bestehen.
2. Test Ihres korrigiertes Programm aus der Hausübung.

Aufgabe 4.3: Selbsttest

Testobjekt sei die nachfolgende Schaltung mit vier Eingängen und einem Ausgang aus der Hausübung:

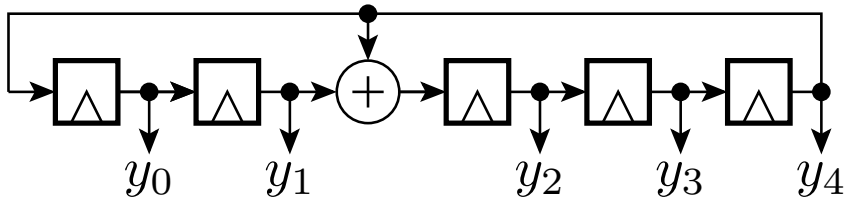


1. Ergänzen Sie am Eingang ein 4-Bit LSFR mit der Struktur von Folie 50
2. und am Ausgang ein 4-Bit-Signaturregister mit zentraler Rückführung und nur mit Eingang x_1 (Folie 54).

Rückführstelle für beide $g_1 = 1$. (Rückführpolynom $x^4 \oplus x \oplus 1$).

Aufgabe 4.4: LFSR-Zykluslänge

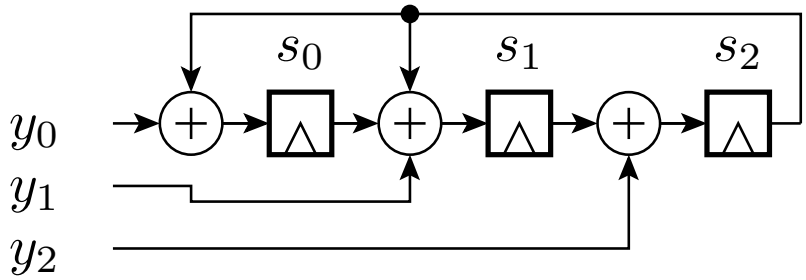
Untersuchen Sie für das nachfolgende 5-Bit linear rückgekoppelte Schieberegister die Zyklusstruktur.



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 Zustand als Testeingabe verwendet wird?

Aufgabe 4.5: Signaturregister

Gegeben ist das nachfolgende Sinaturregister, eine Sollausgabefolge von einem fehlerfreien Testobjekt und die verfälsche Ausgabefolge eines fehlerhaften Testobjekts:

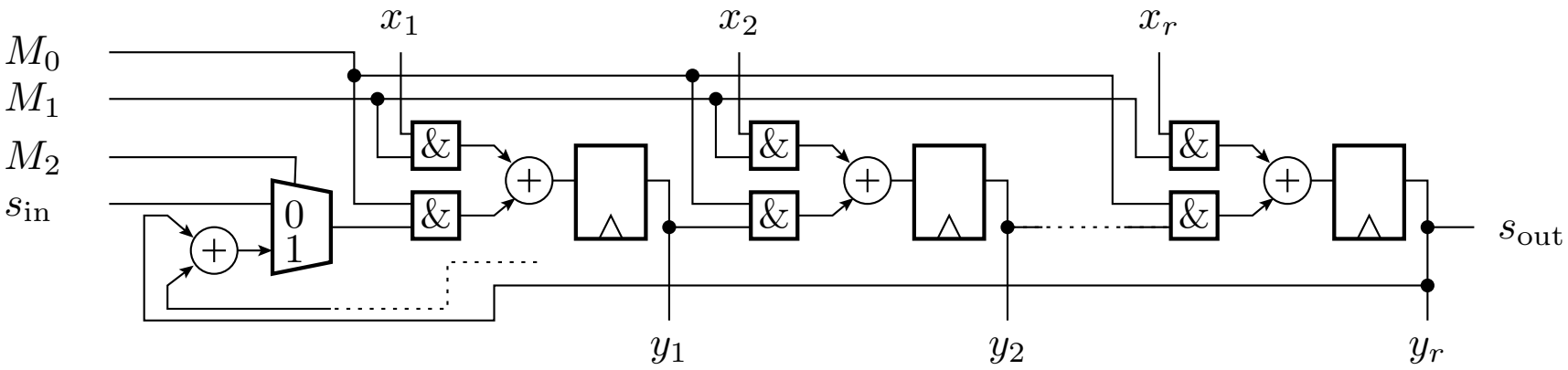


	korrekte Folge	Sollsignatur	fehlerhafte F.	Fehlersign.
Schritt	y_0 y_1 y_2	s_0 s_1 s_2	y_0 y_1 y_2	s_0 s_1 s_2
0	1 0 1	0 0 0	1 1 1	0 0 0
1	0 0 1		0 0 1	
2	1 1 0		0 1 0	
3	0 0 0		0 0 0	
4				

1. Wie groß ist die Wahrscheinlichkeit, dass das Signaturregister für eine verfälschte Datenfolge ein anderes Prüfkennzeichen als für korrekte Datenfolge berechnet?
2. Welche Signatur (Prüfkennzeichen) hat die Sollausgabefolge und welche die fehlerhafte Ausgabefolge?
3. Kontrollieren Sie am Beispiel, dass die Signatur der Differenzfolge gleich der Differenz der Signaturen ist (Überlagerungssatz).

Aufgabe 4.6: 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 4.7: Testsatzlänge

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 70% der Fälle erfolgreich behoben. Wie lang muss die Testzeit mindestens gewählt werden, damit der Fehler mit einer Wahrscheinlichkeit von >95% beseitigt wird?

Aufgabe 4.8: Exponenten der Fehlernachweisdichte

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?

Aufgabe 4.9: Inspektionsfehlerüberdeckung

Inspektionsergebnisse für ein Programm aus 1000 Codezeilen:

- Inspekteur 1: 28 gefundene Fehler
- Inspekteur 2: 32 gefundene Fehler
- Schnittmenge: 19 übereinstimmende gefundene Fehler.

Schätzen Sie nach dem Verfahren »Capture-Recapture« die Anzahl der nicht gefundenen Fehler und die Inspektionsfehlerüberdeckung.

Aufgabe 4.10: Effizienz und Effektivität

In der Aufgabe zuvor hat der erste Inspekteur zwei Stunden für das Aufspüren seiner 28 gefundenen Fehler und der zweite Inspekteur 2,5 Stunden für das Aufspüren seiner 32 Fehler benötigt. Wie groß waren Effizienz und Effektivität beider Inspekteure einzeln und wie groß waren Effizienz und Effektivität der gesamten Inspektion?

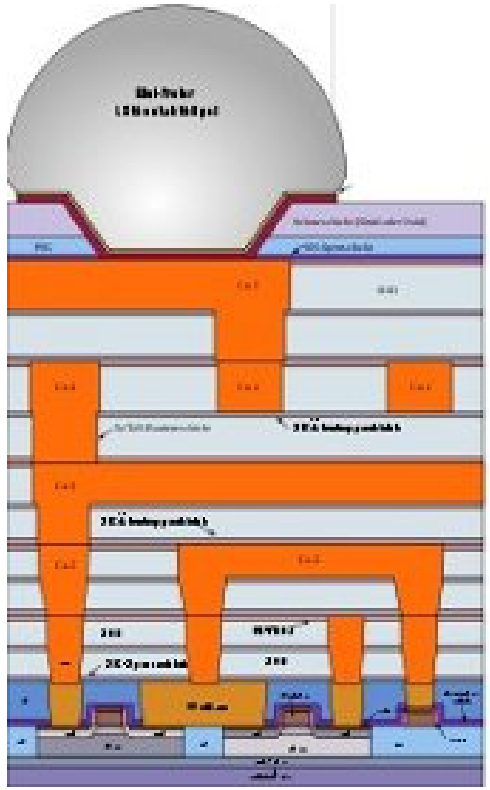
Schaltkreistest

Fertigungsfehler

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.

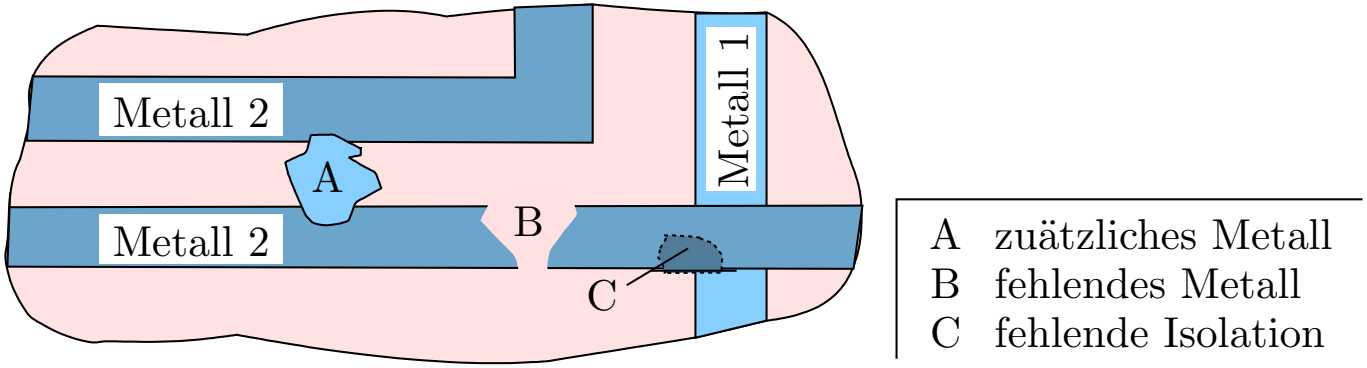


Typische Herstellungsfehler:

- 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.



- 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:

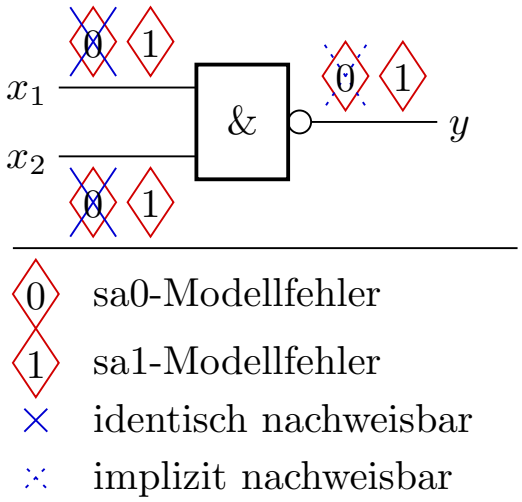
- Erhöhte Verzögerung und
- erhöhte Stromaufnahme

während der Tests.

Haftfehler

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)

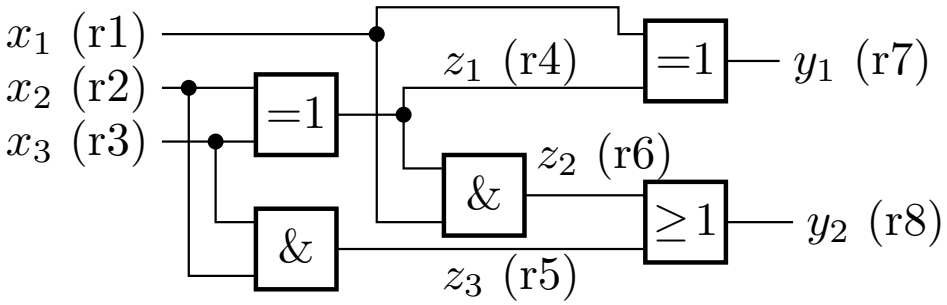
Nachweisimplikation

zugehörige Eingabe ist Element der Nachweismenge

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

Haftfehler sind einfach zu simulieren

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

- Jede zweistellige Logikoperation ist ein Maschinenbefehl.
- In jeder der 8, 16, 32 oder 64 Bits der Operanden kann ein anderer Testfall oder ein anderer Fehler simuliert werden.

Aufwandsabschätzung am Beispiel

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

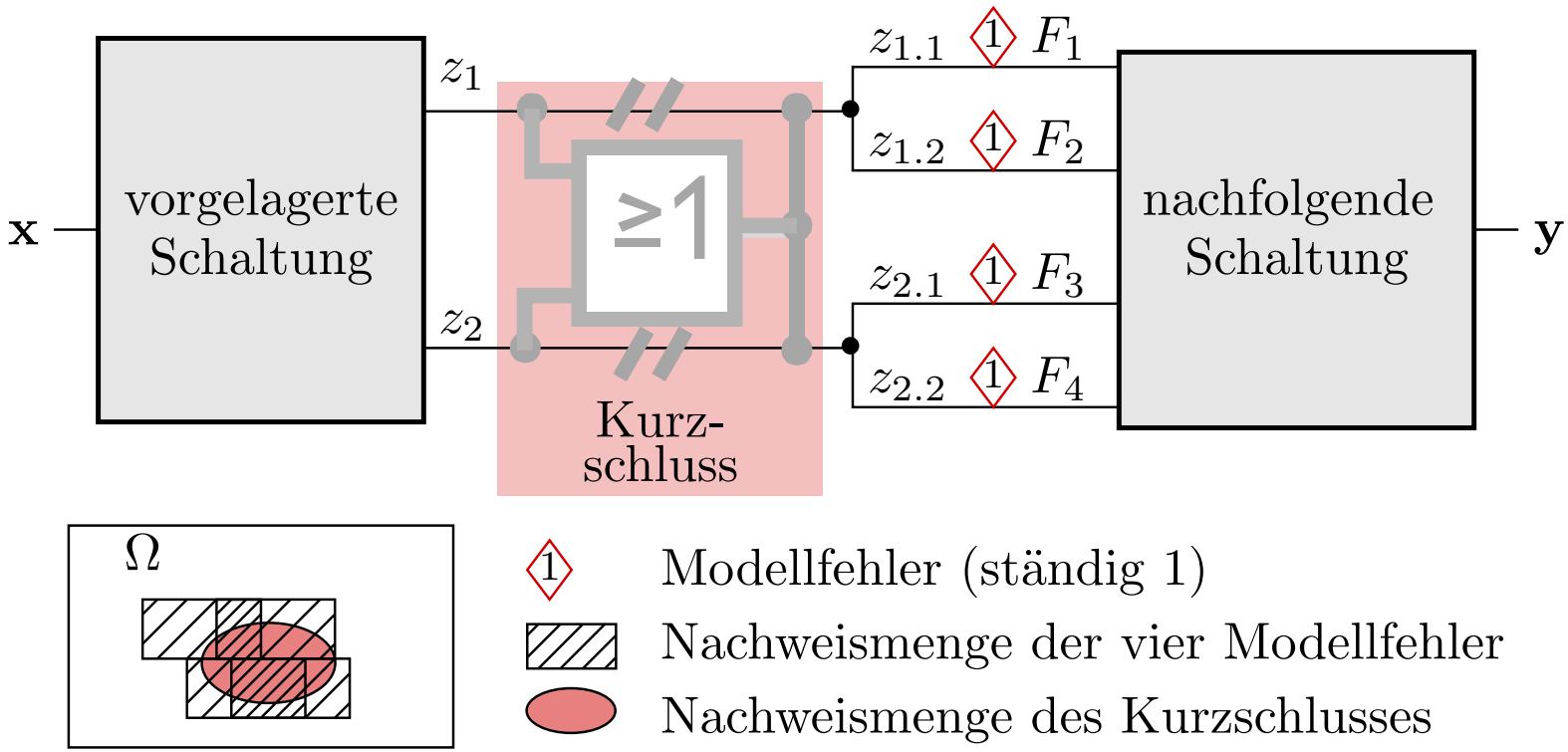
Rechenaufwand, wenn jeder Fehler mit allen Testeingaben simuliert wird und ohne bitparallele Simulation: 10^4 s, ca. 3 h.

Wenn mit jeder der 32 bzw. 64 Bit ein anderer Fehler simuliert wird, nur 6 bzw. 3 Minuten.

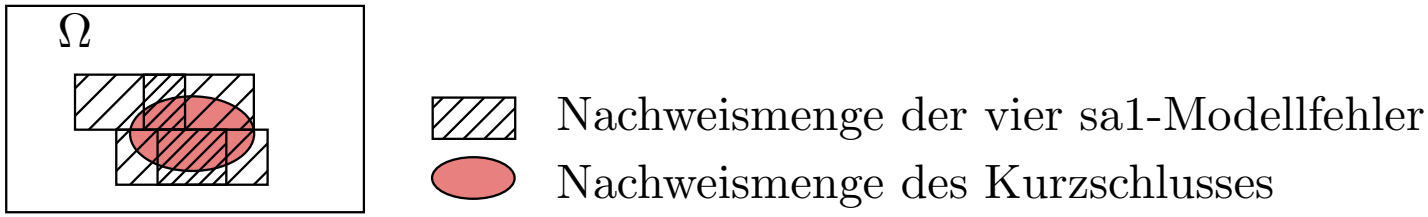
Kurzschlussnachweis mit einem Haftfehlertestsatz

- Zum Nachweis eines Kurzschlusses müssen auf den beteiligten Leitungen unterschiedliche Werte eingestellt und das dabei verfälschte Signal beobachtet werden.
- Die Anzahl der möglichen Kurzschlüsse nimmt im ungünstigsten Fall mit dem Quadrat der Leitungsanzahl zu. Viel größer als die Anzahl der Haftfehler.
- Die Fehlersimulation und die Testsatzberechnung aufwändiger als für Haftfehler, z.B. mögliches zusätzliches Speicherverhalten.
- Für Haftfehler ausgewählte Testsätze erkennen die meisten Kurzschlussmöglichkeiten, so das explizite Modellierung nicht zwingend ist.

Kurzschlussnachweis mit einem Haftfehlertestsatz



- Für jeden Haftfehler wird mindestens ein Test gesucht.
- Wie wahrscheinlich ist der Kurzschlussnachweis?



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:

- Gezielte Testsatzsuche: $FC_{\text{sa}} = 1$, $N \geq 4$ Versuchen mit Kurzschlussnachweiswahrscheinlichkeit 50%:

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

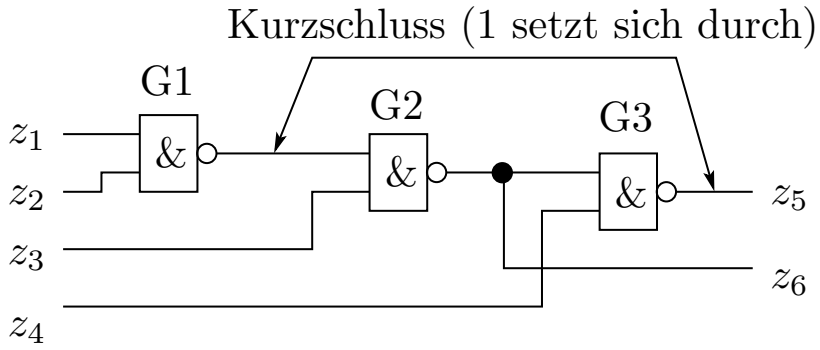
- Zufälliger Testauswahl: Tendentiell doppelt so große Nachweismenge wie die der vier ähnlich nachweisbaren Haftfehler:

$$H_{\text{Kurzschl}}(p) \sim H_{\text{sa}}(2 \cdot p)$$

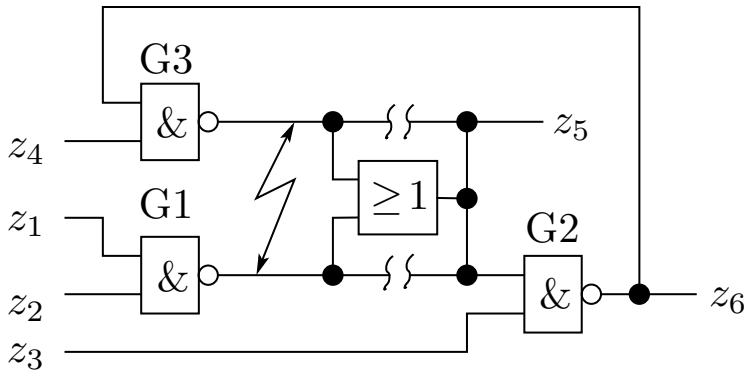
- Grobabschätzung: $FC_{\text{Kurzschl}}(n) \approx FC_{\text{sa}}\left(\frac{n}{2}\right)$.

Ein Kurzschluss kann auch ein zusätzliches Speicherverhalten verursachen.

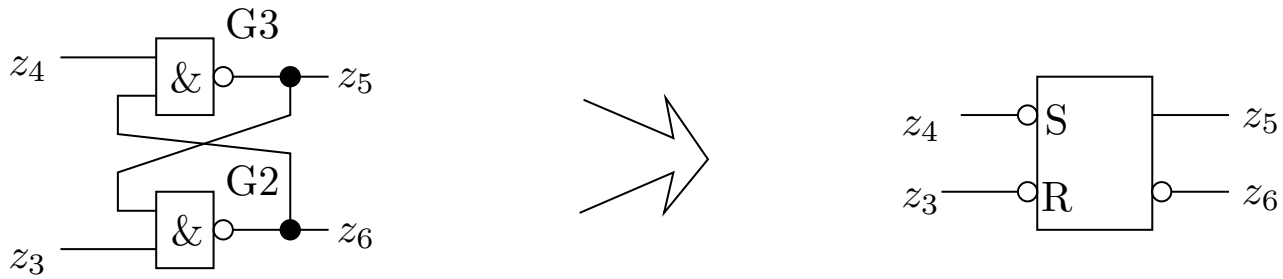
Schaltung mit Kurzschluss



Ersatz des Kurzschlusses durch ein ODER



Ersatzschaltung für $z_1 = z_2 = 1$

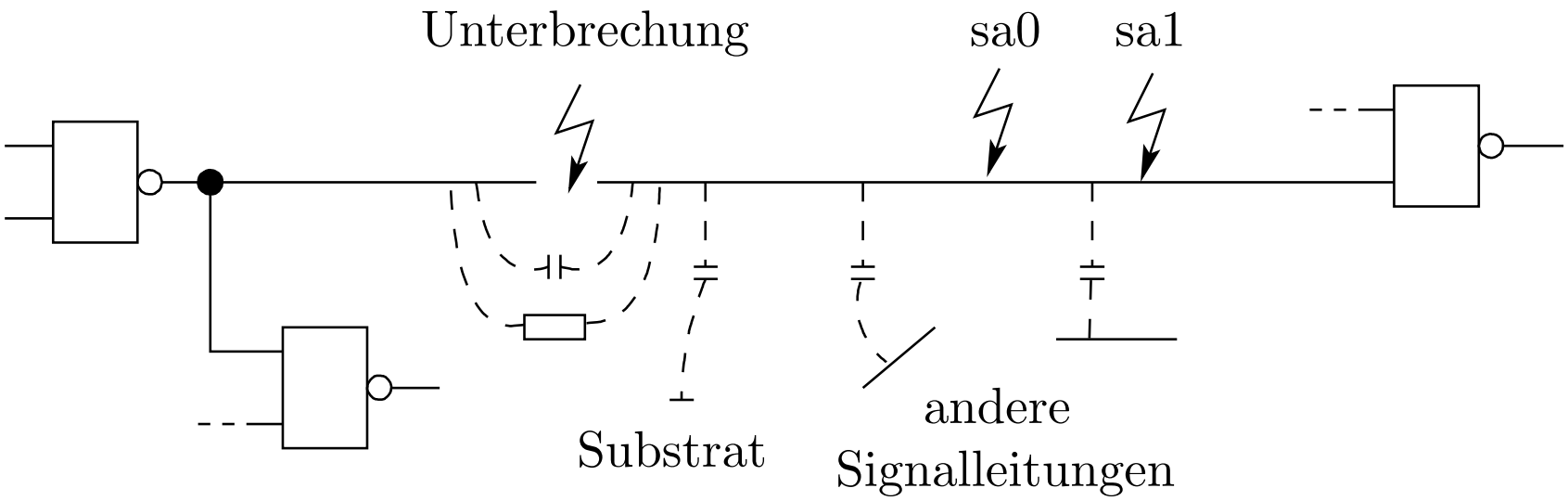


Für Fehler mit Speicherverhalten ist die Fehlersimulation und Testberechung deutlich aufwändiger/schwieriger als für Haftfehler.

Zusammenfassung, Trendabschätzungen

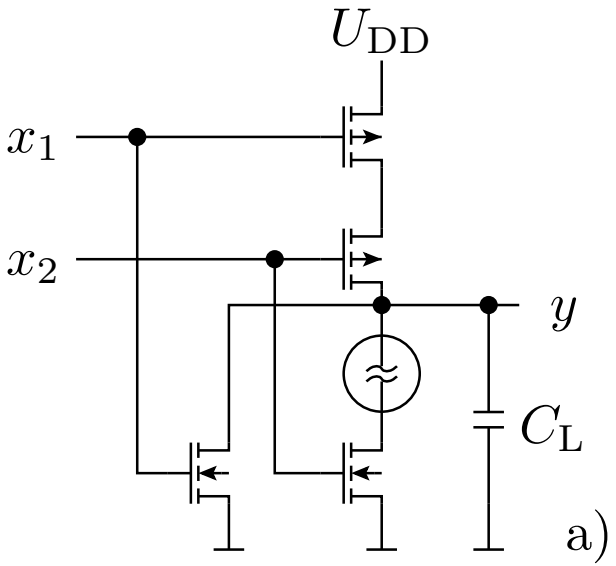
- Kurzschlüsse können sehr vielfältige Fehlerwirkungen haben.
- Berücksichtigung aller Möglichkeiten in der Regel weder notwendig, noch vom Aufwand her zu rechtfertigen.
- Gezielt berechnete Haftfehlertestsätze mit hoher Fehlerüberdeckung erkennen die meisten Kurzschlüsse.
- Die Kurzschlussüberdeckung 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 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)

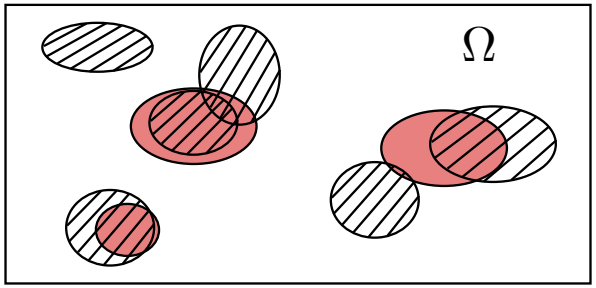
x_2	x_1	y
0	0	1
0	1	0
0	0	1
1	0	0

c)

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

- Überwiegender Nachweis mit 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 (z.B. null setzt sich bei Kurzschluss durch, ...)
- Jeder logisch nachweisbare Schaltkreisfehler teilt sich Anregungsbedingungen und Beobachtungspfade mit Haftfehlern. Trend $H(p) \sim H_{\mathrm{sa}}(c \cdot p)$ mit $c > 1$.
- Zufällige Testauswahl: $FC(n) \approx FC_{\mathrm{sa}}\left(\frac{n}{c}\right)$.
- Gezielte Testauswahl: großes FC_{sa} bewirkt großes FC, wobei FC erheblich davon abhängt, wie viele Tests für jeden Haftfehler gesucht werden. Im günstigen Fall mehrere.

Validierung mit Zahlen aus der Literatur

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.

Daraus folgt, dass die tatsächlichen Schaltkreisfehler im Mittel deutlich besser als Haftfehler nachweisbar zu sein scheinen.

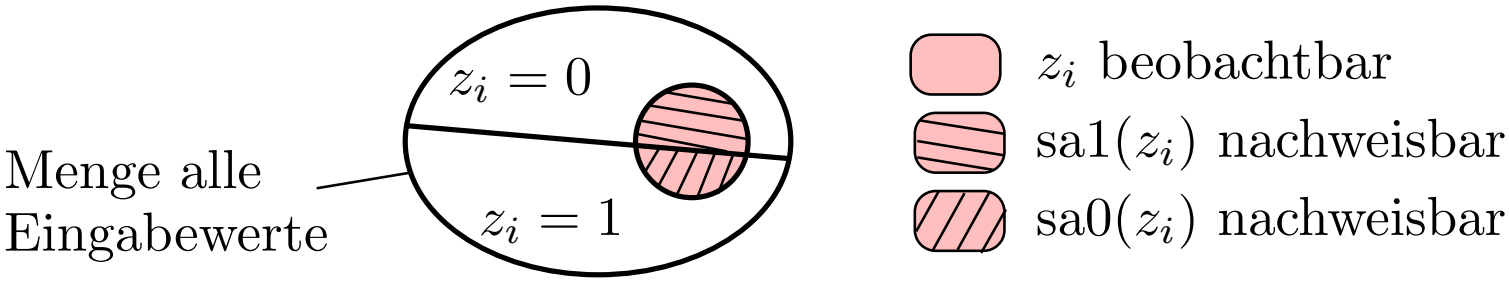
Andere Fehlermodelle

Weitere für Schaltkreise diskutierte Fehlermodelle

- Toggle Test: Der Testsatz muss jede Leitung mindestens einmal auf null und eins steuern.
- Zellenfehlermodell: Teilschaltung mit 1-Bit-Ausgabe funktioniert genau mit einer Eingabebedatung nicht.
- Gatterverzögerungsmodell: Für jedes Schaltelement werden die beiden Modellfehler verzögerter Anstieg (slow to rise) und verzögerter Abfall (slow to fall) unterstellt.
- Pfadverzögerungsfehler: Für jeden Schaltungspfad werden die beiden Modellfehler verzögerter Anstieg (slow to rise) und verzögerter Abfall (slow to fall) unterstellt.

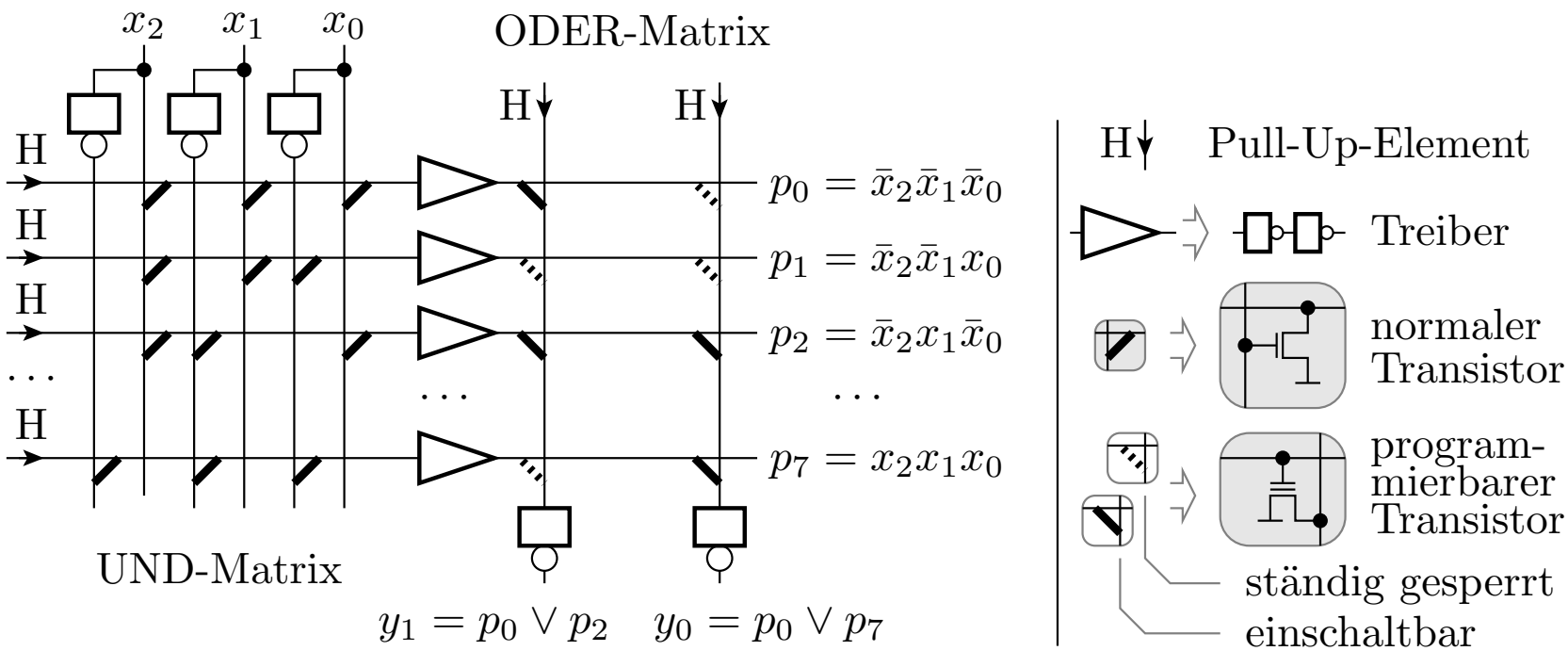
Toggle-Test

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 ($H(p) \sim H_{\text{Toggle}}(c \cdot p)$ mit $c \ll 1$).
- Zufallstest: Fehlerüberdeckung ist etwa die Toggle-Überdeckung der $\left(\frac{1}{c} \gg 1\right)$-fachen Testsatzlänge.
- Gezielter Testauswahl: Toggle-Überdeckung erlaubt kaum Aussagen über die Fehlerüberdeckung.
- Für HW Veraltet. Für SW werden noch vergleichbare Vollständigkeitsmaße genutzt.

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:

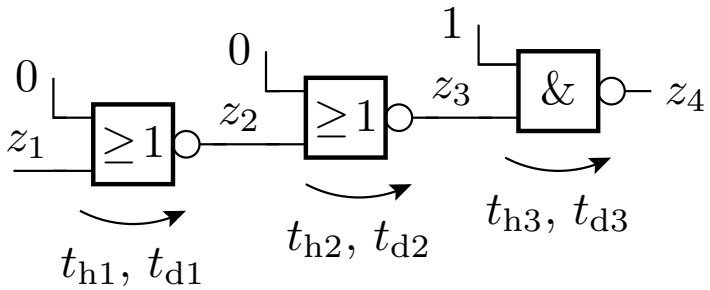
$$\varphi_{\mathrm{M}} = \mathrm{Anz(Ausg)} \cdot 2^{\mathrm{Anz(Eing)}}$$

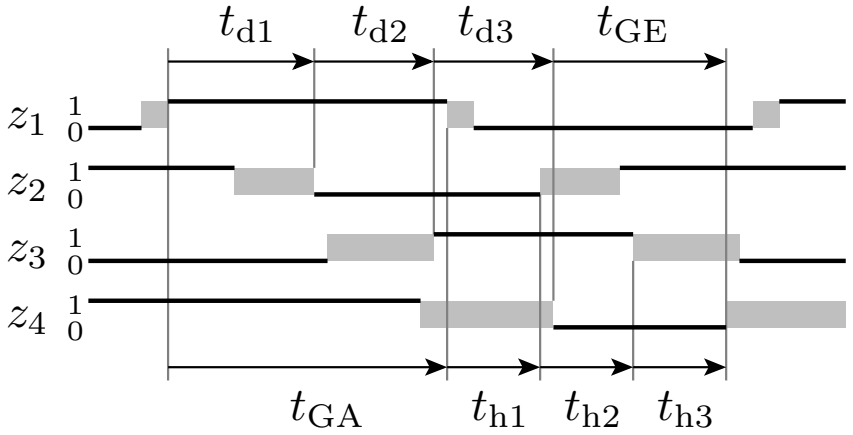
(Anz(Eing) – Anzahl der Eingänge; Anz(Ausg) – Anzahl der Ausgänge). Vergleich Zellenfehler und Haftfehler für freistrukturierte Schaltungen (Realisierung aus Gattern statt durch programmierte Speicher):

- 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



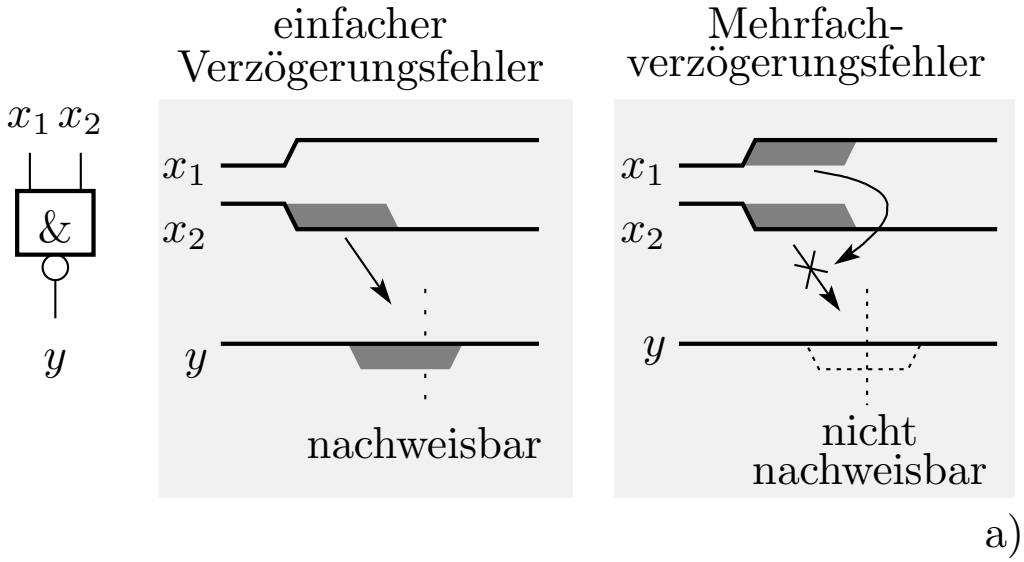


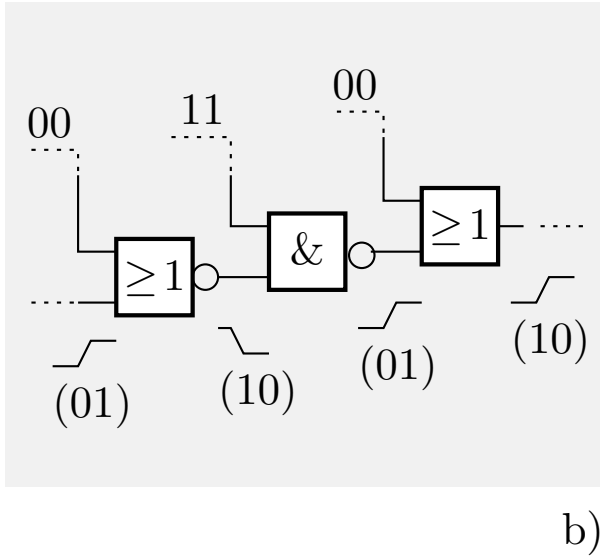
t_{GA}	Gültigkeitsdauer am Pfadanfang
t_{GE}	Gültigkeitsdauer am Pfadende

- 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). Bei zufälliger Initialisierung mit $g = 50\%$: $H_{GVerz}(p) \sim H_{sa}\left(\frac{p}{2}\right)$.
- Zufallstest: Gleiche Fehlerüberdeckung wie für Haftfehler erfordert etwa doppelte Testsatzlänge.
- Gezielte Auswahl: > 2 Tests je Modellfehler suchen.

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

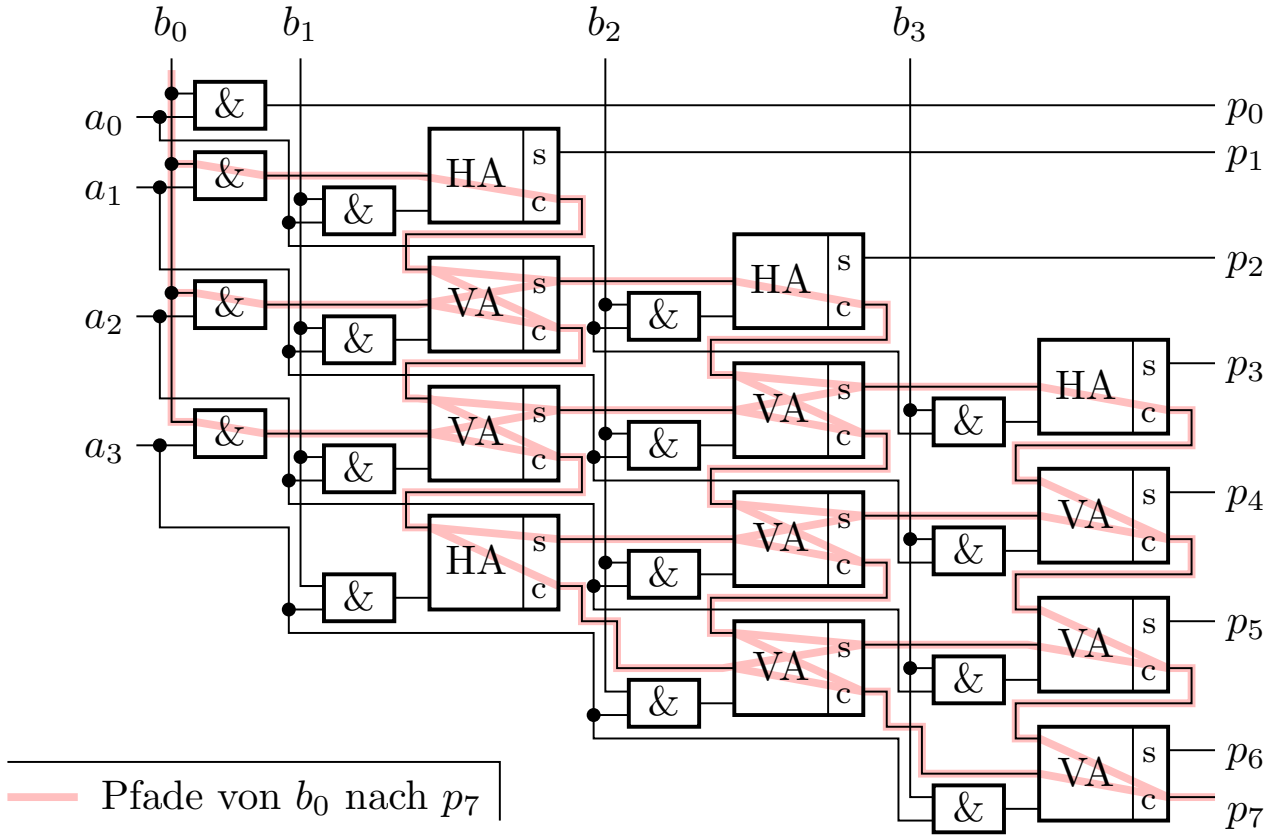


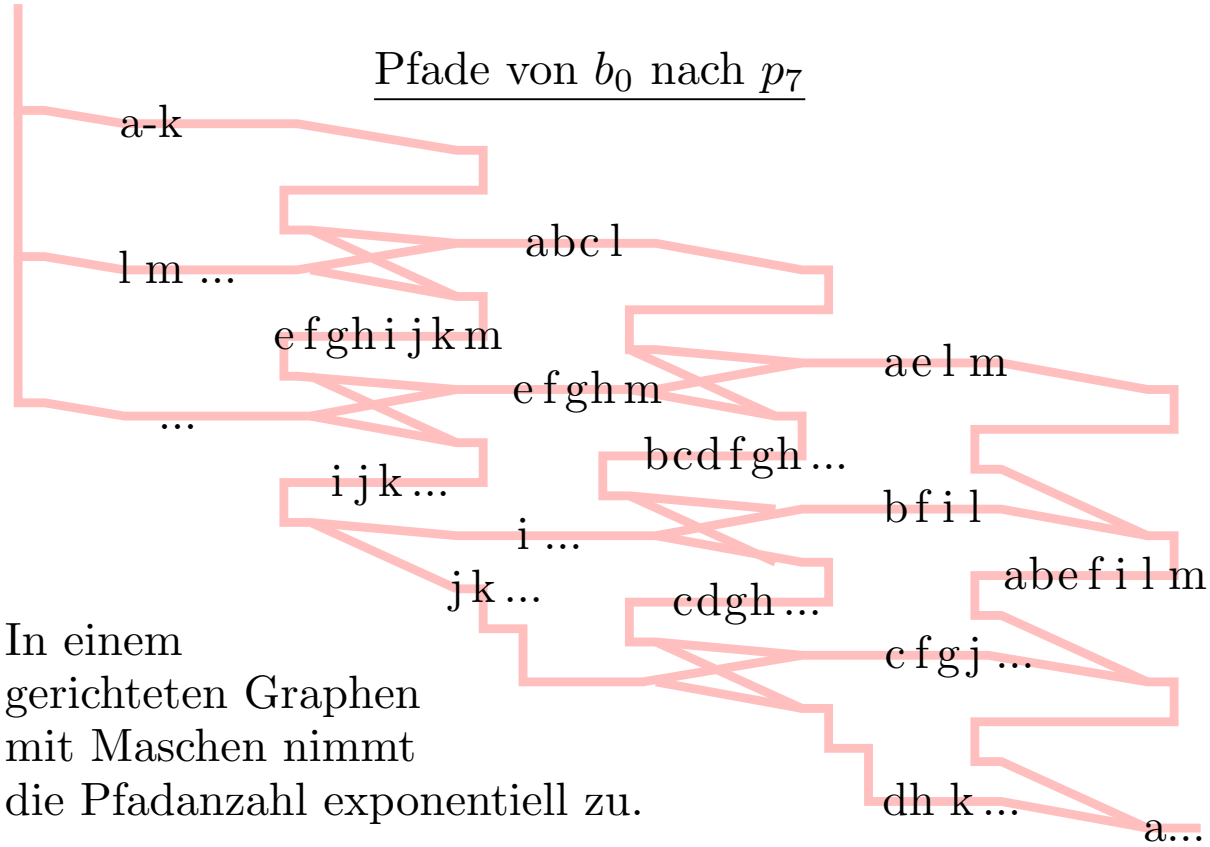


- Problem Pfadanzahl; Beispiel Matrixmultiplizierer

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

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

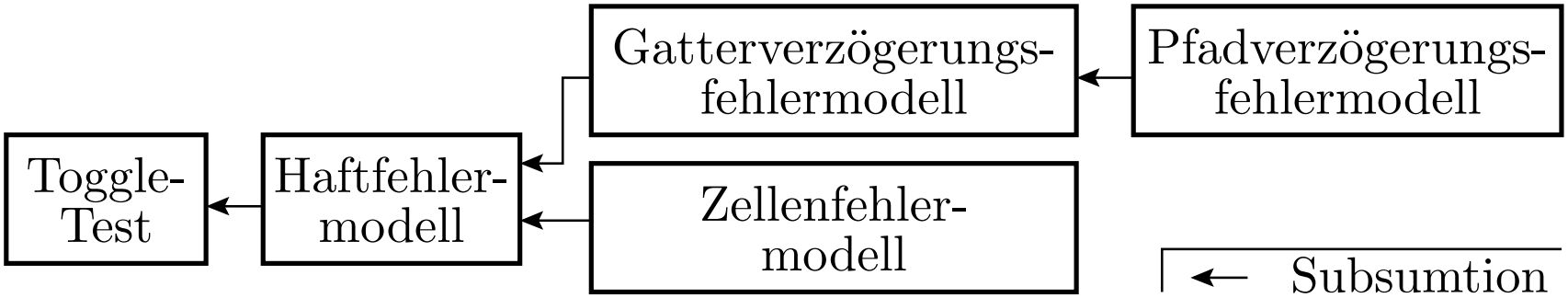




- Ausschluss verlängerter 01- und 10-Wechsel entlang aller Pfade schließt überhöhte Verzögerungen aus.
- In Schaltungen mit rekonvergenten Auffächerungen wächst die Pfadanzahl exponentiell mit der Schaltungsgröße.
- Exponenziell wachende Fehlermengen ⇒ nicht praktikabel.

Subsumtionshierarchie von Fehlermodellen

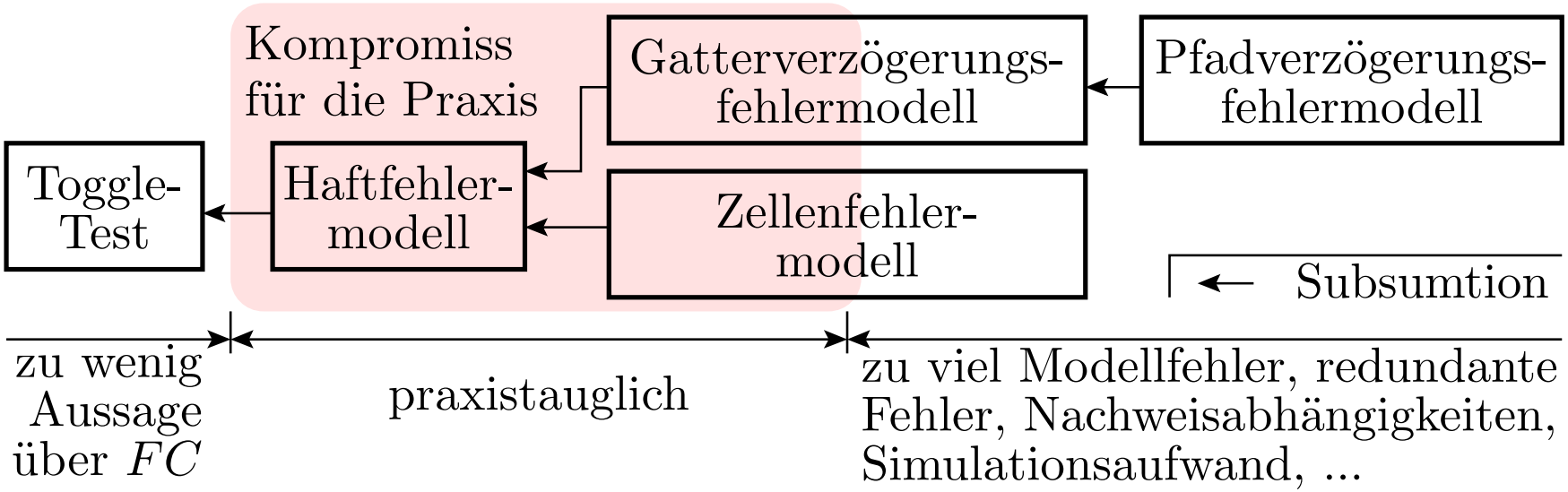
- Subsumption bedeutet in den Beschreibungslogiken, dass ein Konzept (eine eindeutig beschriebene Menge von Objekten) eine Teilmenge eines anderen Konzepts ist.
- Fehlermodell A subsumiert Fehlermodell B, wenn ein Testsatz, der alle Modellfehler von A nachweist, garantiert auch alle Modellfehler von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



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

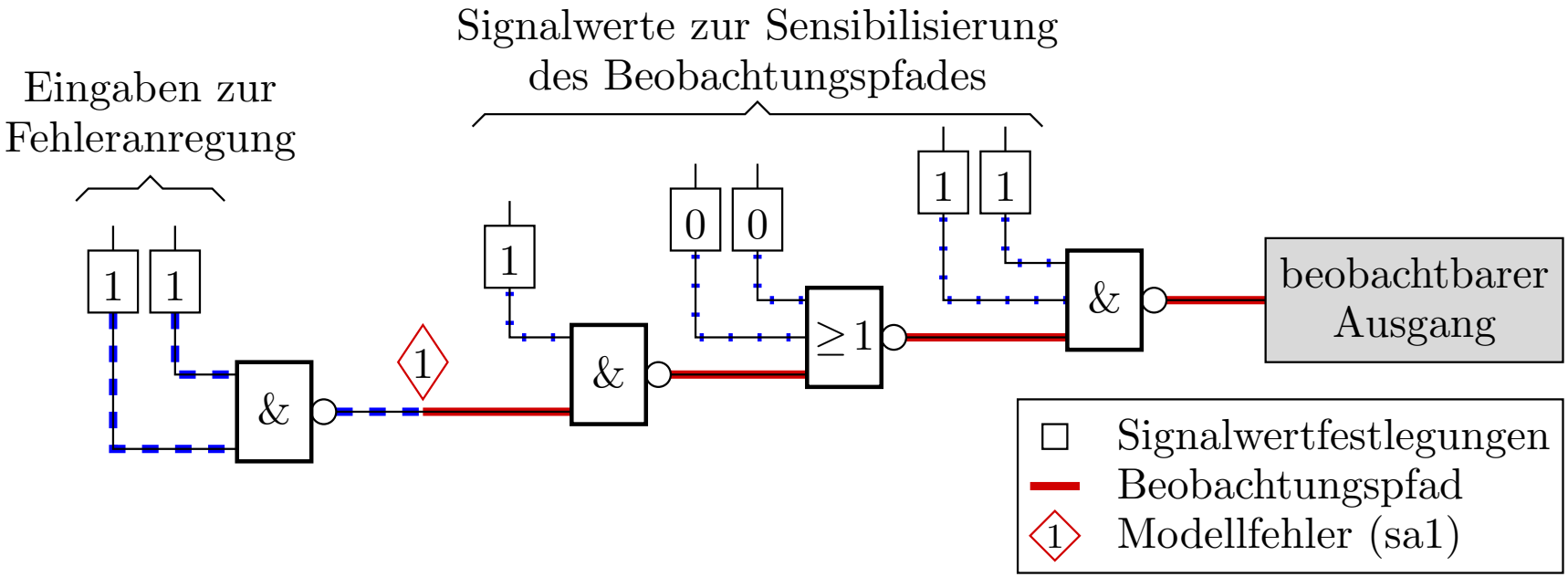
Je höher ein Fehlermodell in der Subsumtionshierarchie steht:

- 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.



Testberechnung (D-Alg.)

Pfadalgorithmen



Vom Fehlerort werden durch Festlegung von Signalwerten

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

Geeignet für Haftfehler (Pfadverzögerungsfehler, ...)

D-Algorithmus

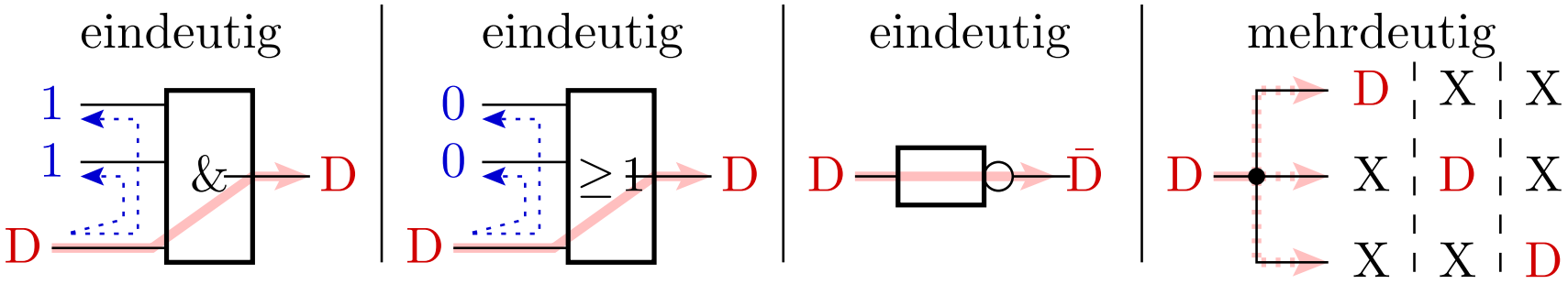
Erweiterung der Logikwerte um Pseudo-Werte:

D Signalwert ist gleich dem Logikwert am Fehlerort.

$\bar{D}$ Signalwert ist gleich dem inversen Signalwert am Fehlerort.

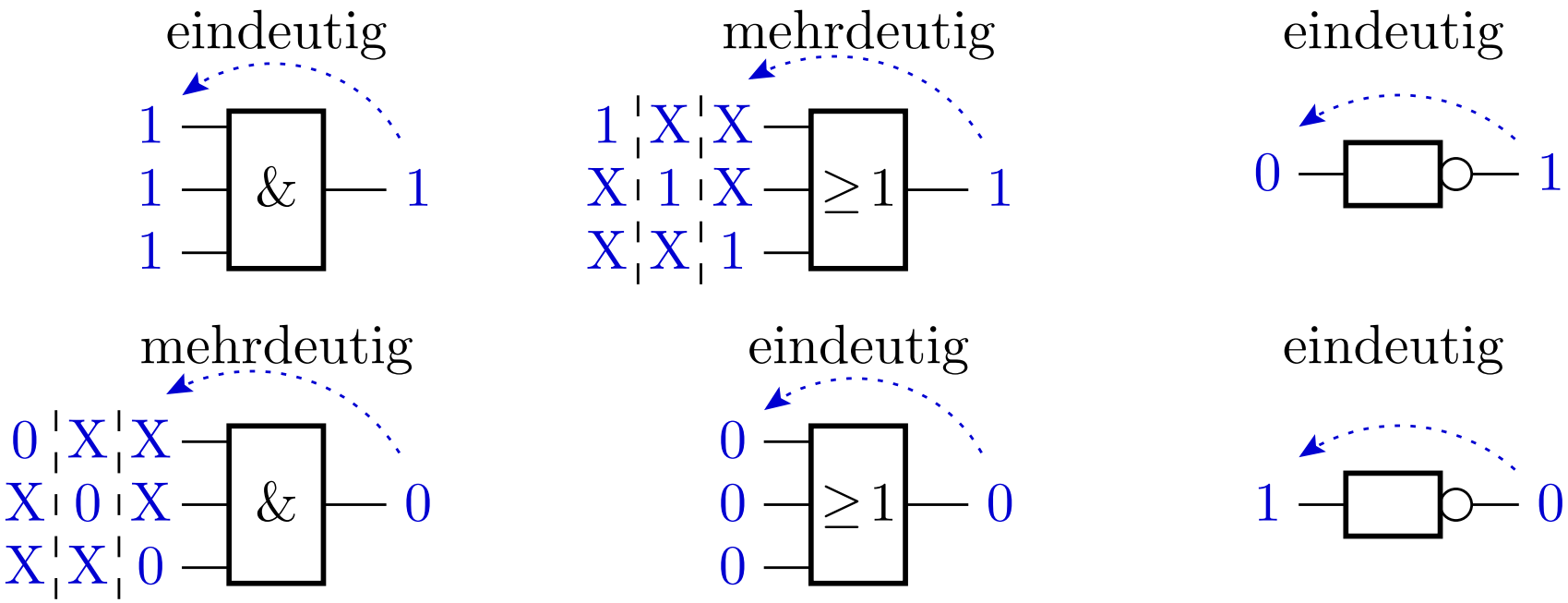
X Signalwert ist ungültig oder für den Fehlernachweis ohne Bedeutung.

Regeln für die Sensibilisierung eines Beobachtungspfades:

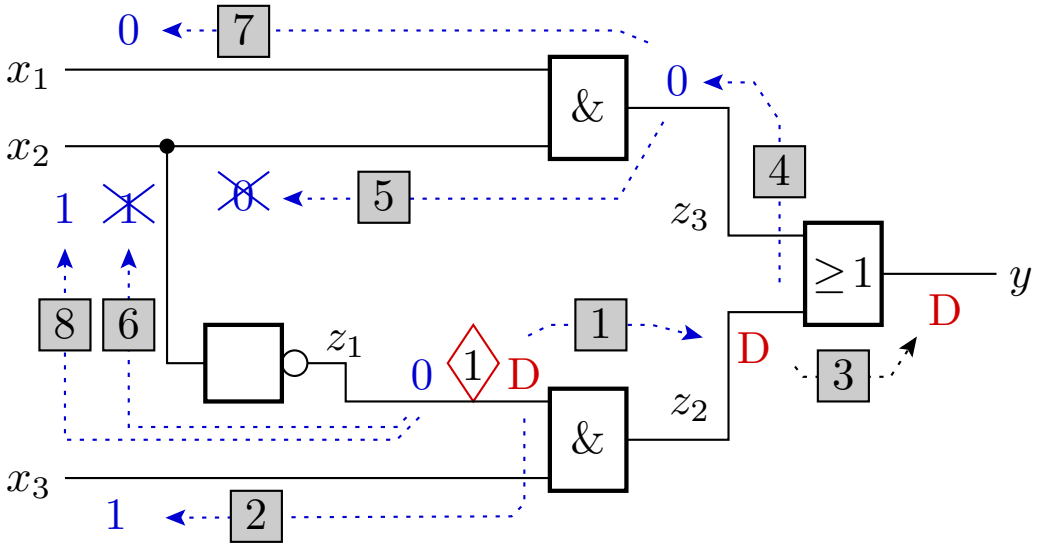


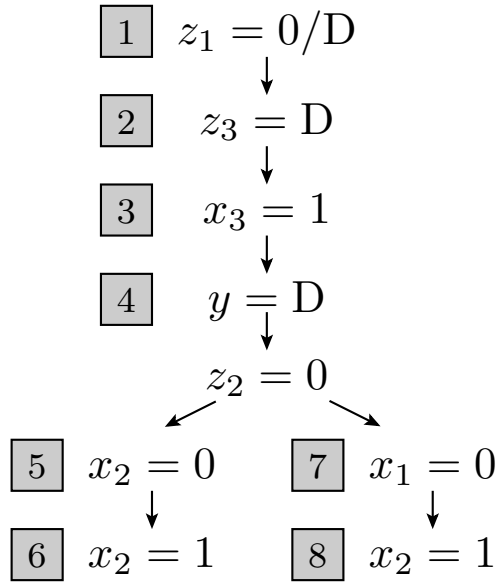
- Für $\bar{D}$ analog.

Einstellung von Steuerwerten



- Jede kombinatorische Schaltung kann durch 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	~~0~~1	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.

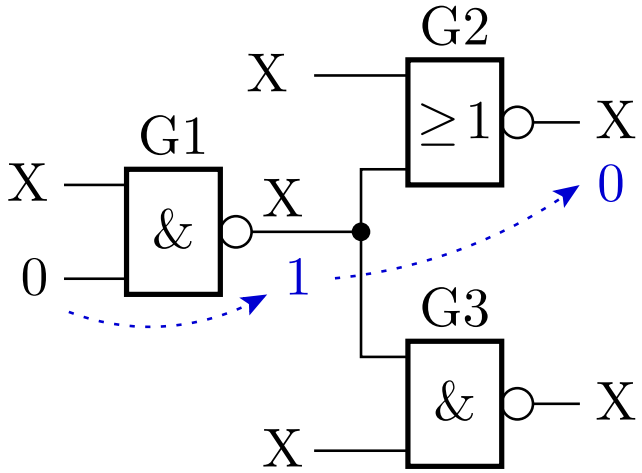
Heuristigen:

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

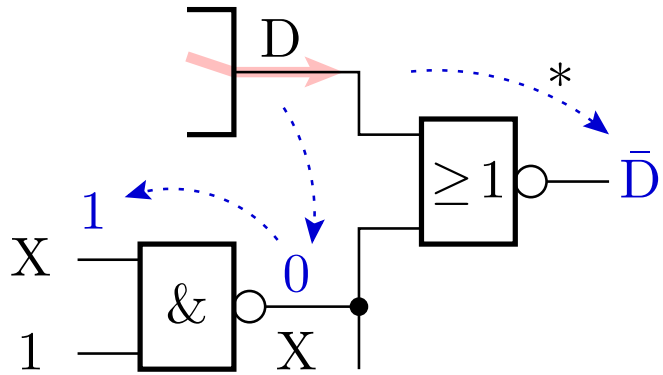
Implikationstest (Widerspruchsfrüherkennung)

- Aus den berechneten Wertefestlegungen alle eindeutig folgenden Werte berechnen.

Implikation in Signalflussrichtung



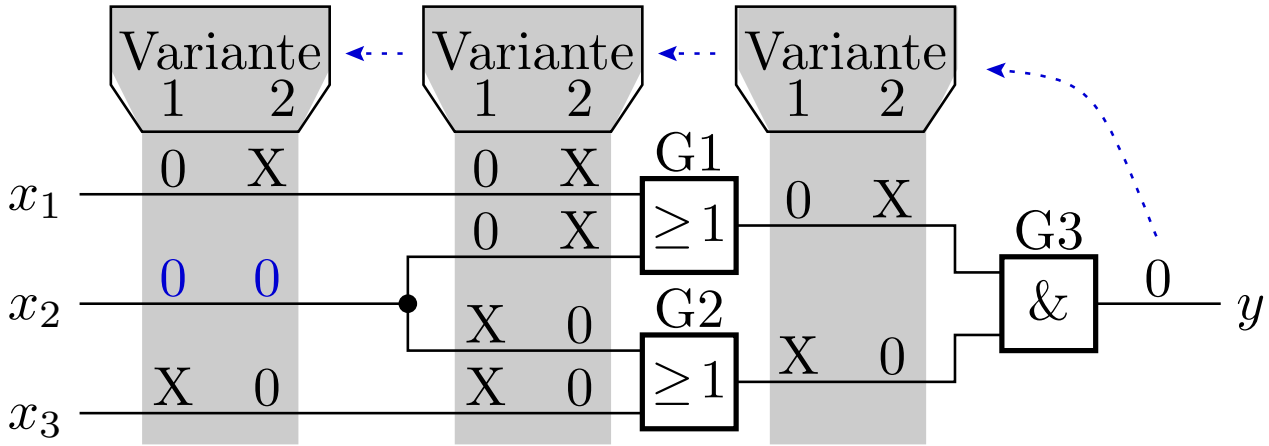
D-Pfad- und Rückwärtsimplikation



* eindeutige Weiterführung

- Mindert die Entscheidungsbaumtiefe.

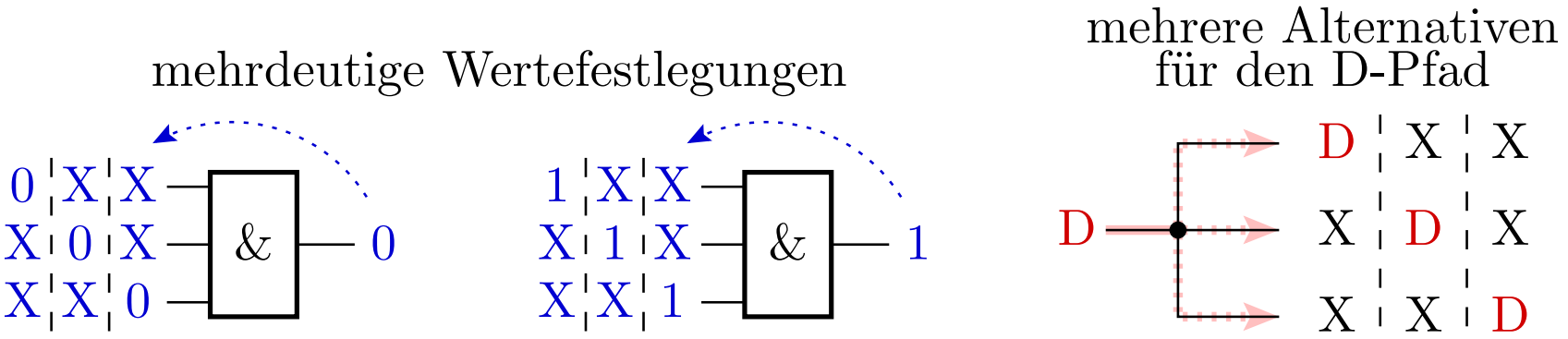
- Rückwärtsimplikation über mehrere Gatterebenen:



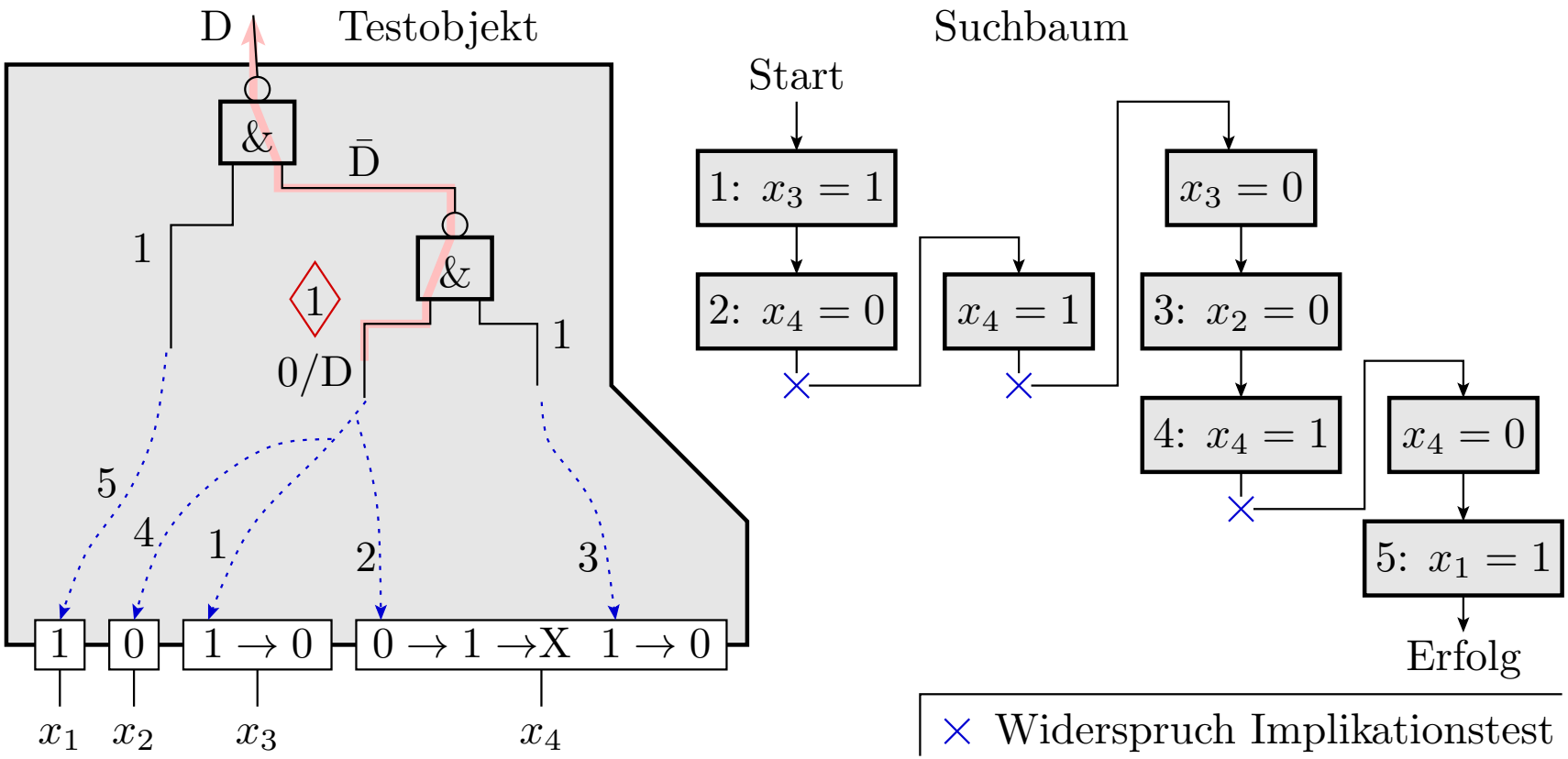
- Für $y = 1$ gibt es zwei Einstellmöglichkeiten.
- Für beide Möglichkeiten muss $x_2 = 1$ sein.
- Das Erkennen von Implikationen dieser Art mindert die Backtracking-Häufigkeit um bis zu 80 %.

Suchraumbegrenzung

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

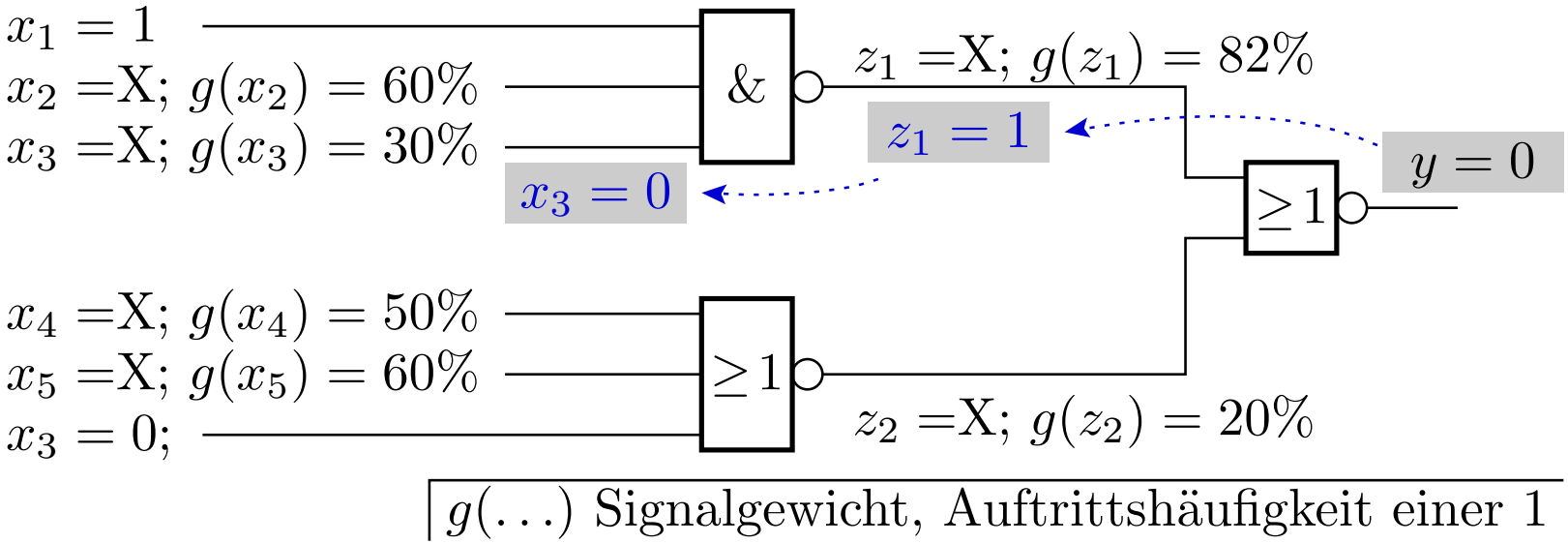


- Nur die Schaltungseingänge können unabhängig voneinander alle Wertevariationen annehmen.
- Es genügt, den Suchbaum mit den Eingabewertefestlegungen aufzubauen.
- Begrenzt Suchraum auf $2^{\text{Anz(Eing)}}$ (Anz(Eing) – Eingangsanzahl). Verringert Rechenaufwand um Zehnerpotenzen.



- 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

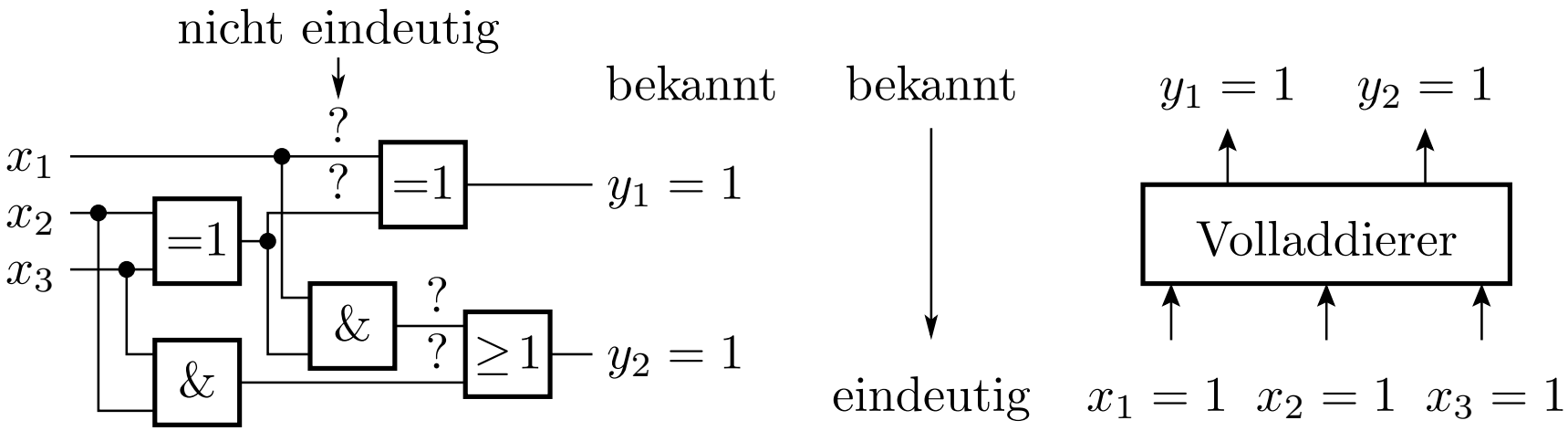
- Beschreibung durch Tabellenfunktion (Bsp. Volladdierer):

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

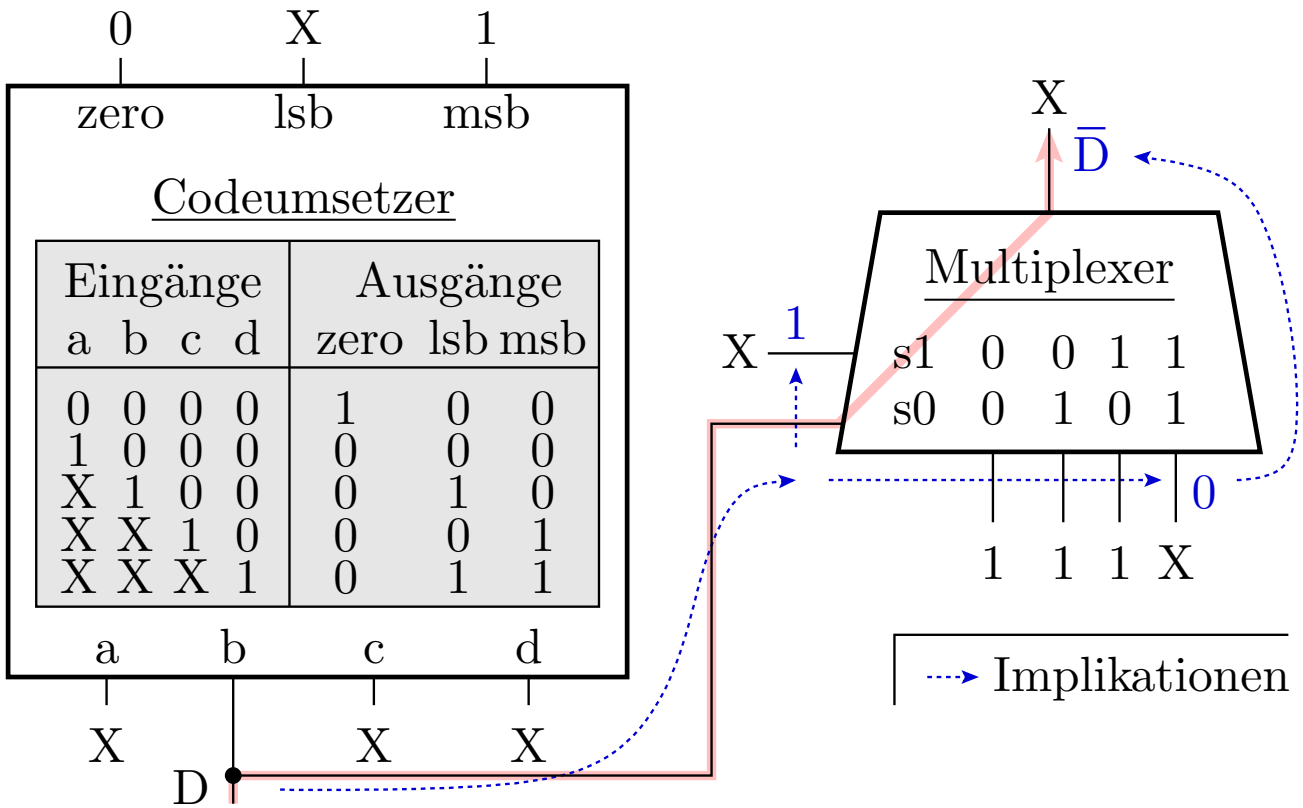
gegeben		Lösungsmenge
XXX00	$\Rightarrow$	00000
01DXX	$\Rightarrow$	01D$\bar{\text{D}}$D
1XXXD	$\Rightarrow$	10D$\bar{\text{D}}$D, 1D0$\bar{\text{D}}$D
11XX1	$\Rightarrow$	11111, 111001

- 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.



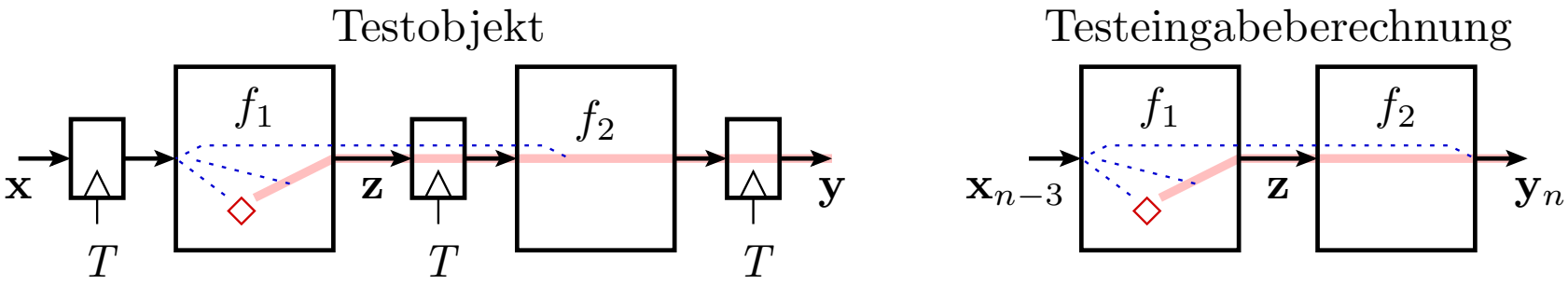
- »lsb« hängt bei »zero=0« und »msb=1« nicht von »b« ab. Eindeutiger D-Pfad über Multiplexer.
- Tabelleneingabewerte »X« (Eingang beeinflusst nicht die Ausgabe) führt zu Tabellen mit $\ll 2^{\text{Anz(Eing)}}$ Tabellenzeilen.

Sequentielle Schaltungen

Sequenzielle Schaltungen

Zurückführung auf kombinatorische Schaltungen:

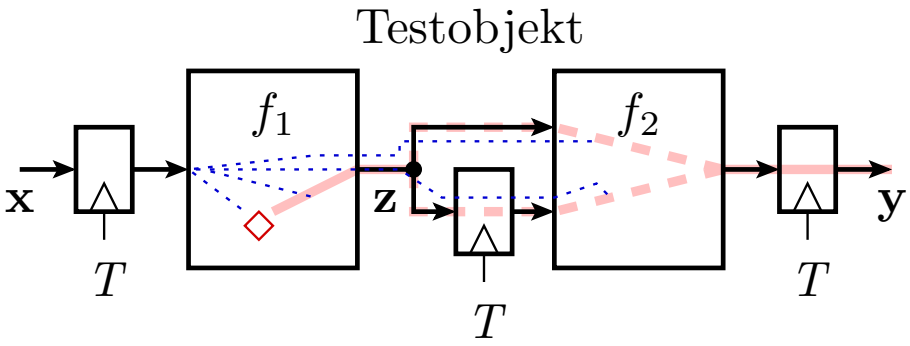
- nur Abtastung

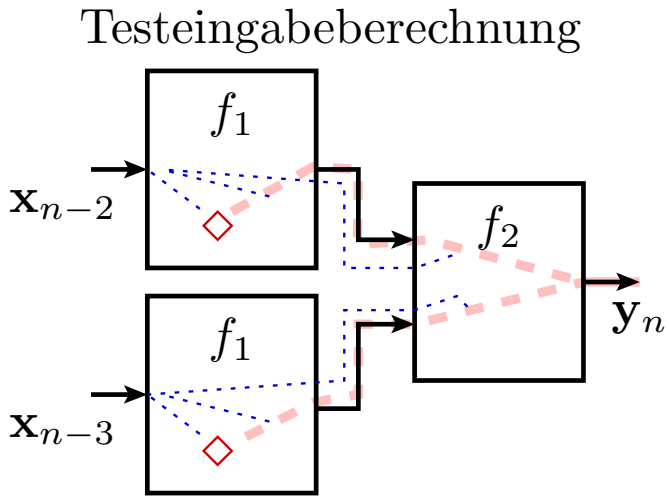


- Testberechnung wie für eine kombinatorische Schaltung
- Zeitversatz zwischen Ein- und Ausgabe berücksichtigen

```
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



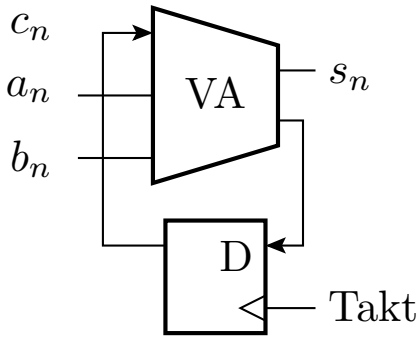


- Die kombinatorisch Ersatzschaltung enthält mehrere Kopien derselben Schaltung.
- Die Haftfehler sind in jeder Kopie.
- Eingaben mehrerer Zeitebenen / Mehr-Pattern-Test:

```
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 ...;
```

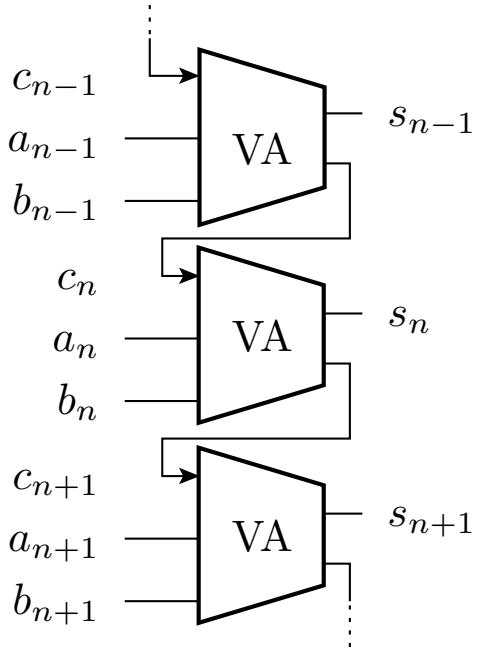
Schaltungen mit Rückführung

serieller Addierer



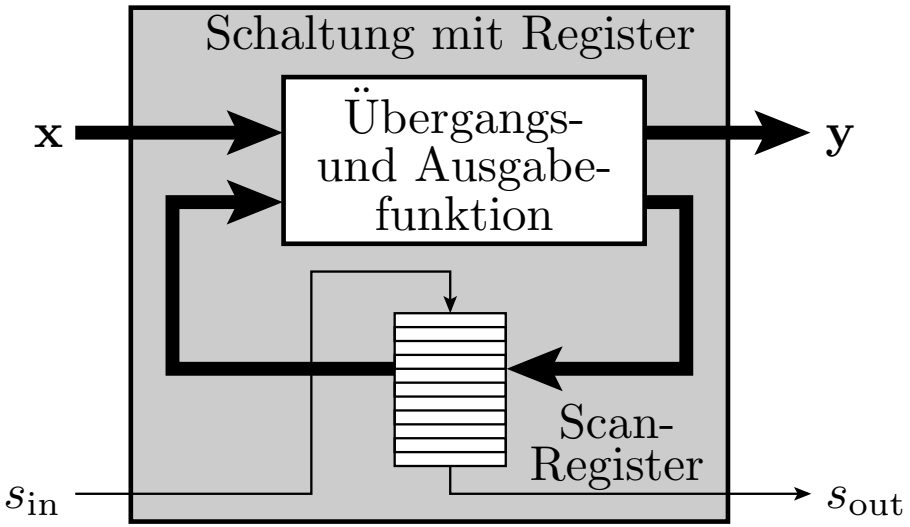
VA - Volladdierer

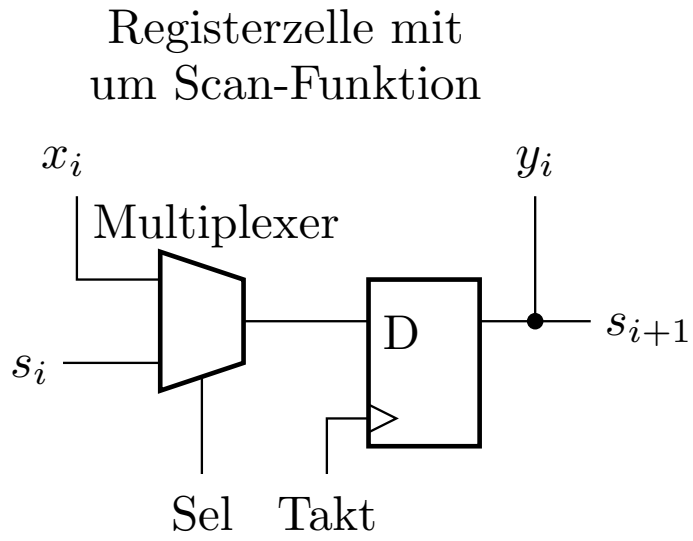
aufgerollter Addierer



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

Auftrennung der Rückführungen für den Test





- Erweiterung der Zustandsregister um eine Schiebefunktion. (Mindestaufwand ein Multiplexer je Speicherzelle.)
- Serielles Auslesen und Neuladen der Zustandsregister zwischen aufeinanderfolgenden Testschritten.

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 Speicherorganisation 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 $1 \rightarrow 0$ $0 \rightarrow 1$	 $W(i)1$, $R(i)$, $W(i)0$, $R(i)$ $W(i)0$, $R(i)$, $W(i)1$, $R(i)$
	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
$C(i)$ Inhalt Zelle i

$R(i)$ 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

Beispielalgorithmus 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	$R(i)1, W(i)0$		$R(i)0, W(i)1$		$R(i)1$
$\vdots$	$\vdots$		$\vdots$		$\vdots$
$N-1$	$R(i)1, W(i)0$		$R(i)0, W(i)1$		$R(i)1$

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

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	$R(i)1, W(i)0$		$R(i)0, W(i)1$		$R(i)1$
$\vdots$	$\vdots$		$\vdots$		$\vdots$
$N-1$	$R(i)1, W(i)0$		$R(i)0, W(i)1$		$R(i)1$

Lässt sich auch ohne großen Zusatzaufwand als Selbsttest implementieren.

Ausfälle

Ausfälle

Hardware unterliegt einem Verschleiß, der zu Ausfällen führen kann. Bei einem Ausfall entsteht ein Fehler. Im Gegensatz zu den nicht nachgewiesenen Herstellungsfehlern haben neue Fehler durch Ausfälle eine ähnliche FHNW-Funktion wie ungetestete Systeme, d.h. sie verursachen im Mittel weit häufiger Fehlfunktionen, aber nicht immer komplette Funktionsunfähigkeit.

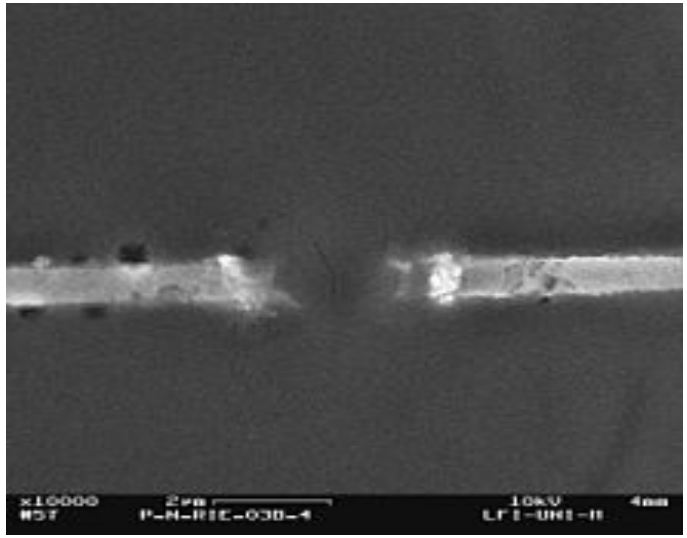
Eine Sonderstellung haben Frühausfälle. Ihre Ursache sind Beinahefehler[14], die den Verschleiß beschleunigen. Für sie haftet der Hersteller in Form von Garantieleistungen.

Maßnahmen für den Umgang mit Ausfällen sind:

- Überwachung und Fehlerbehandlung, die bis zur Fehlertoleranz gehen kann, und
- regelmäßige Wartung (z.B. KFZ-Inspektion).

[14] Materialrisse, kalte Lötstellen, ...

Verschleiß elektronischer Bauteile



Langsam ablaufende physikalische Vorgänge:

- Korrosion (Stecker, Schalter, Isolationen, Leiterbahnen, ...).
- Elektromigation: strombedingte Wanderung von Metalatomen bei hohen Stromdichten.
- Gateoxiddurchschlag: Hochschaukelnde Tunnelströme, Ladungseinlagerung bis zum lokalen Schmelzen des Oxid und Bildung von Verbindungen (Phänomen Zunahme des Stromverbrauchs über Monate bis zum Ausfall).
- Parameterdrift: Widerstandswerte, Kapazitäten, Schwellspannungen etc.

Verbesserung Fertigung, Material etc. $\Rightarrow$ weniger Ausfälle

Kenngrößen des Ausfallverhaltens

- Lebensdauer t_L: Zeit vom Beanspruchungsbeginn bis zum Ausfall. Zufallsgröße.
- Überlebenswahrscheinlichkeit: Wahrscheinlichkeit, dass ein System zu einem Zeitpunkt t noch »lebt«:

$$R(t) = P(t < t_\mathrm{L})$$

- Ausfallrate[15] λ: Relative Abnahme der Überlebenswahrscheinlichkeit mit der Zeit:

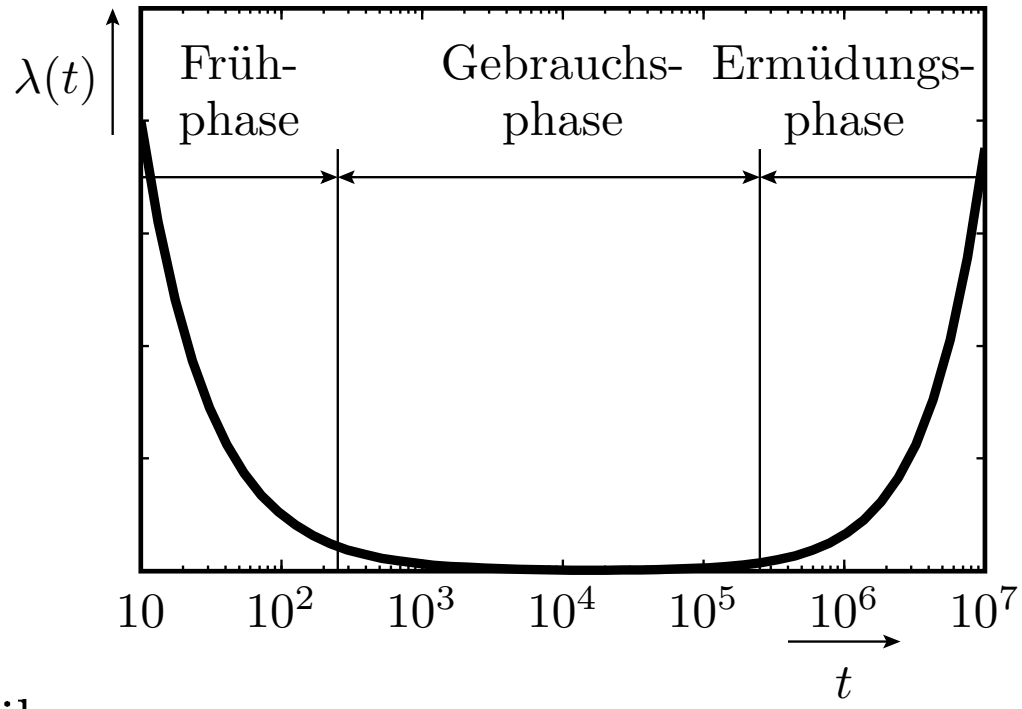
$$\lambda(t) = -\frac{1}{R(t)} \cdot \frac{dR(t)}{dt}$$

- Mittlere Lebensdauer:

$$E(t_\mathrm{L}) = \int_0^\infty R(t) \cdot dt$$

[15] wichtigste Vergleichsgröße [4, S.68]

Ausfallphasen



- Frühausfallphase (infant mortalities): Erhöhte Ausfallrate durch Schwachstellen (Materialrisse, lokal stark überhöhte Feldstärke oder Stromdichte, ...).
- Hauptnutzungsphase: Näherungsweise konstante Ausfallrate.
- Verschleißphase: Ausfall durch Materialermüdung[16].

Maßeinheit der Ausfallrate: fit (failure in time)

$$1\,\text{fit} = 1\,\text{Ausfall in } 10^9\,\text{Stunden}$$

[16] In IT-Konsumbereich heute oft »geplante Obsoleszenz«.

Ausfallraten in der Hauptnutzungsphase nach [4]

Bauteil	Ausfallrate in fit	Bauteil	Ausfallrate in fit
diskrete HBT	1 bis 100	Widerstände	1 bis 20
digitale IC	50 bis 200	Kondensatoren	1 bis 20
ROM	100 bis 300	Steckverbinder	1 bis 100
RAM	bis 500	Lötstellen	0,1 bis 1
analoge IC	20 bis 300		

(HBT – Halbleiterbauteile; IC – Schaltkreise)

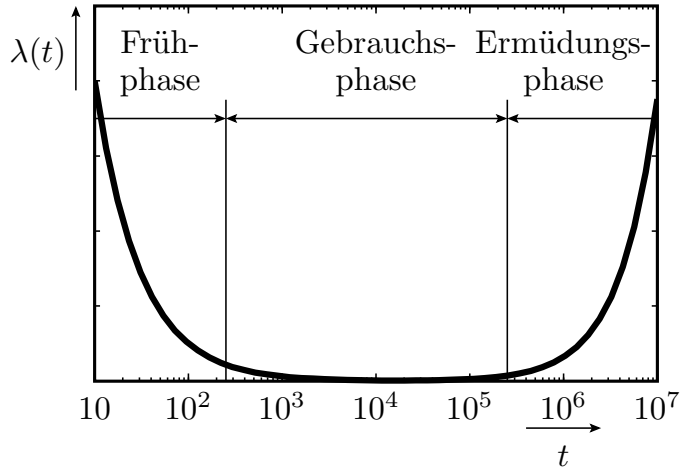
- Ausfallrate = Ausfallanzahl / Bauteilanzahl
- Bei mehreren Bauteilen und konstanten Ausfallraten addieren sich die Ausfallraten.

- Ausfallrate einer Baugruppe:

Bauteiltyp	Anzahl n	Ausfallrate λ	$n \cdot \lambda$
Schaltkreise	20	150 fit	3000 fit
diskrete BT	15	30 fit	450 fit
Kondensatoren	15	10 fit	250 fit
Widerstände	30	10 fit	300 fit
Lötstellen	2000	0,5 fit	1000 fit
Baugruppe			5000 fit

- Im Mittel 1 Ausfall in $2 \cdot 10^5$ Stunden (≈23 Jahre) Betriebsdauer.
- Von den heutigen PCs, Handys, ... fallen pro Jahr und hundert Stück nur wenige aus. Nach 2 ... 5 Jahren Ermüdungsausfälle, z.B. durch eingetrocknete Elkos.

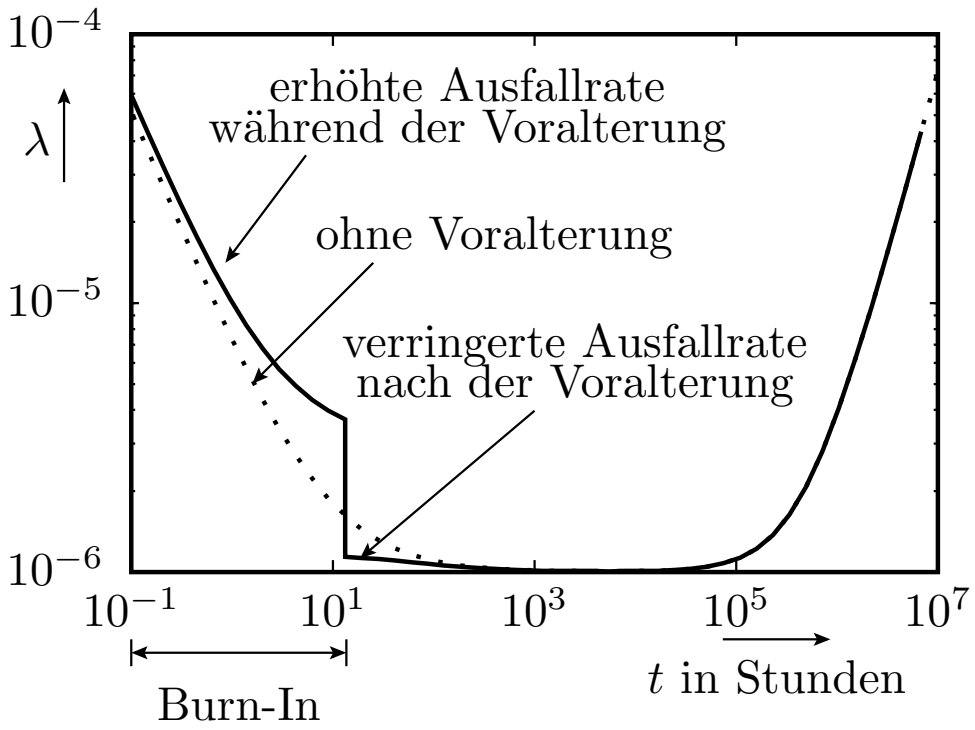
Frühausfälle



- Nach [2] kommen auf 100 richtige Fehler etwa ein Beinahefehler, der zu einem Frühausfall führt.
- Bei 50% fehlerfreien und 50% aussortierten Schaltkreisen
  - 50%/100 = 0,5% Beinahefehler.
  - Die Hälfte wird mit dem Ausschuss aussortiert
  - ≈ 0,25% (jeder 400ste) Schaltkreis verursacht ein Frühausfall
  - Bei 20 Schaltkreisen pro Gerät jedes zwanzigste Gerät.
  - Bei großen Systeme fast jedes System.
- Frühausfälle sind Garantiefälle.
- Garantieleistungen sind teuer (Reparatur, Ersatz, Auftragsabwicklung, Image-Verlust)

Was tun?

Voralterung (Burn-In)



- Beschleunigung der Alterung vor dem Einsatz durch »harte« Umgebungsbedingungen
  - überhöhte Spannung,
  - überhöhte Temperatur,
  - Stress.
- Einsatz erst nach der Frühphase (wenn die kränklichen Bauteile gestorben und ausgetauscht sind).

Künstliche Voralterung ist auch in anderen Bereichen, z.B. Maschinenbau gebräuchlich.

Kalte, warme und heiße Reserve

Systeme ohne Reparaturmöglichkeit, die lange verfügbar sein müssen (z.B. in einem Satelliten)

- erhalten Ersatzkomponenten und
- Funktionen zur automatische Rekonfiguration (Komponententausch) nach einem Ausfall.

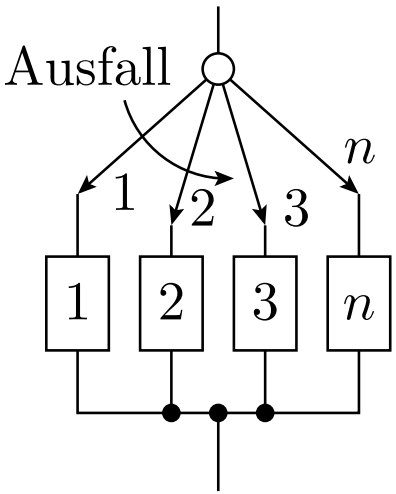
Arten der Reservekomponenten:

- Heiße Reserve: Reservekomponenten arbeiten parallel (z.B. Mehrversionssystem) und fallen mit derselben Wahrscheinlichkeit wie das aktive System aus.
- Kalte Reserve: Reservekomponenten werden geschont und funktionieren idealerweise noch alle zum Ausfallzeitpunkt der aktiven Komponente.
- Warme Reserve: Reserveeinheiten (z.B. das Reserverad im Auto) altern auch, wenn sie nicht genutzt werden, nur weniger.

Kalte Reserve

Für jede Komponente beginnt die Belastung erst nach Ausfall der vorherigen Komponente.

Phase	mittlere Dauer
1	$E(t_{\mathrm{L}.1})$
2	$E(t_{\mathrm{L}.2})$
3	$E(t_{\mathrm{L}.3})$
...	...
Summe:	$E(t_{\mathrm{L.ges}}) = \sum_{i=1}^{n} E(t_{\mathrm{L}.i})$



- Die Lebensdauern aller Komponenten addieren sich[17].

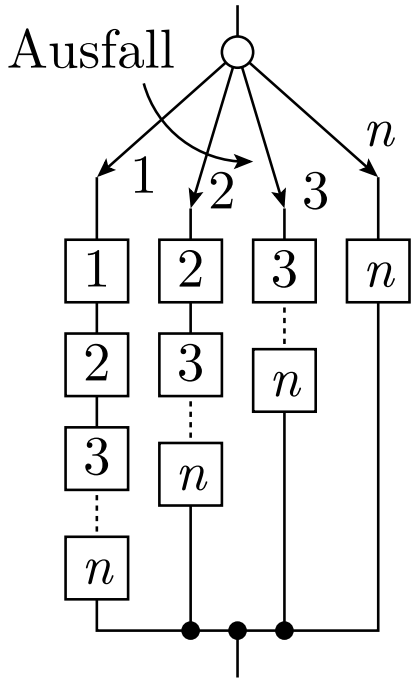
[17] Unter der Annahme, dass die Umschalter und die ungenutzten Reserveeinheiten Ausfallrate null haben.

Heiße Reserve

- Alle noch lebenden Komponenten können gleichmaßen ausfallen:

$$E\left(t_{\mathrm{L}.i}\right) = \frac{1}{\sum_{j=1}^{i} \lambda_j}$$

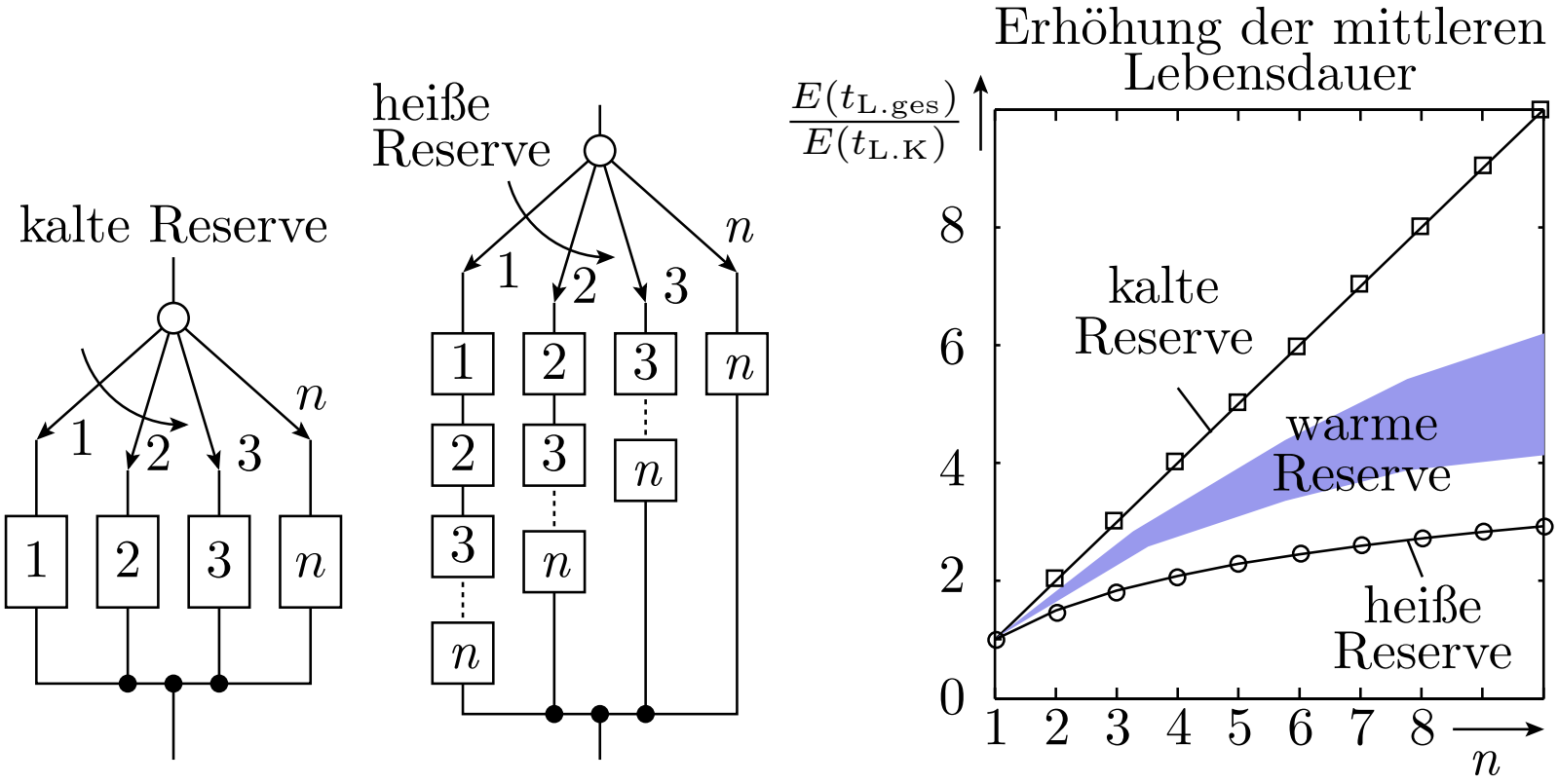
- Komponenten mit gleicher Ausfallrate λ_{K}:



Phase	mittlere Dauer
1	$\frac{1}{n \cdot \lambda_{\mathrm{K}}} = \frac{E(t_{\mathrm{L.K}})}{n}$
2	$\frac{1}{(n-1) \cdot \lambda_{\mathrm{K}}} = \frac{E(t_{\mathrm{L.K}})}{n-1}$
...	...
Summe:	$E\left(t_{\mathrm{L.ges}}\right) = E\left(t_{\mathrm{L.K}}\right) \cdot \sum_{i=1}^{n} \frac{1}{i}$

- Erste Reservekomponente erhöht die mittlere Lebensdauer nur um die Hälfte, die zweite nur um ein Drittel etc.

Warme Reserve



- Die Ausfallrate der »kalten« Ersatzkomponenten ist kleiner als im aktiven Zustand, aber größer null.
- »Warme« Reserveeinheiten verlängert die Lebensdauer mehr als »heiße« und weniger als »kalte«.

Verfügbarkeitsplan[18]

Ausfälle beeinträchtigen die Verfügbarkeit. Im Verfügbarkeitsplan bilden Elemente, die funktionieren müssen, damit das Gesamtsystem funktioniert, Reihenschaltungen, und redundante Elemente, die ausgefallene Elemente ersetzen können, Parallschaltungen. Abschätzung der Gesamtausfallrate aus den Ausfallraten der Komponenten:

- Reihenschaltung: Addition der Ausfallraten.
- Parallelschaltung (kalte Reserve): Addition der mittleren Lebensdauern.

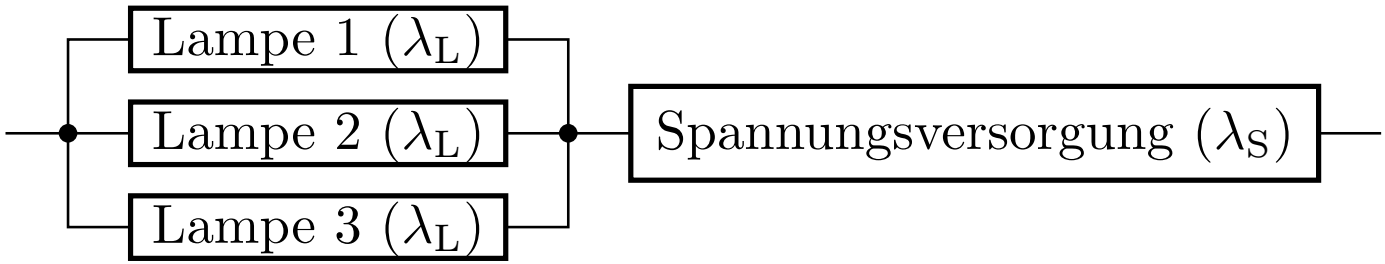
Bei heißer und warmer Reserve ist die Berechnung komplizierter, aber zumindest numerisch auch problemlos lösbar.

[18] In der Literatur »Zuverlässigkeitsstrukturen« [4, S.70]

Beispiel Flurbeleuchtung

Die Flurbeleuchtung sei verfügbar, wenn wenn mindestens eine von drei Lampen und die Spannungsversorgung funktioniert.

Verfügbarkeitsplan:



Die beiden nicht unbedingt erforderlichen Lampen bilden eine heiße Reserve:

$$\lambda_{Lges} \approx \frac{\lambda_L}{\frac{1}{3}+\frac{1}{2}+1} \approx 0{,}5 \cdot \lambda_L$$

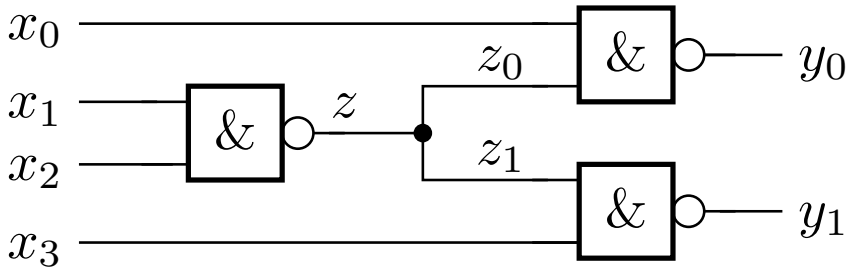
und halbieren die Ausfallrate der Lampeneinheit. Ausfallrate des Gesamtsystems:

$$\lambda_{ges} \approx \lambda_S + 0{,}5 \cdot \lambda_L$$

Aufgaben

Aufgabe 4.11: Haftfehlermenge

Stellen Sie für die nachfolgende Schaltung die Menge aller Haftfehler auf. Bei mehreren identisch nachweisbaren Haftfehlern ist jeweils nur einer in die Fehlermenge aufzunehmen.

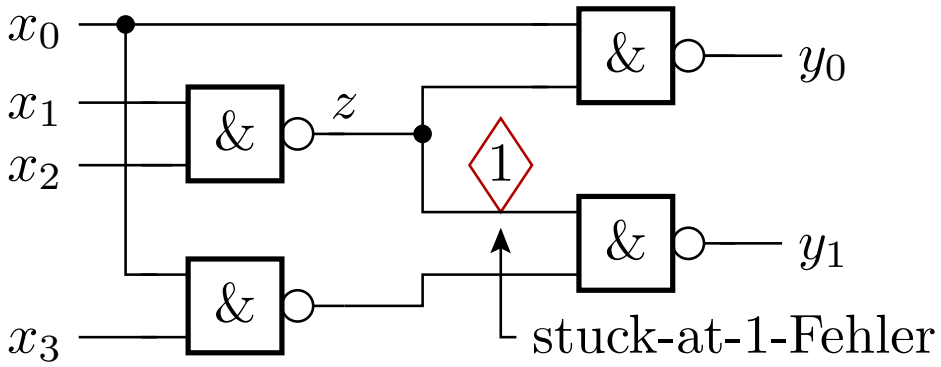


Hinweis: Notation der Haftfehler sa0(<Signalname>) bzw. sa1(<Signalname>).

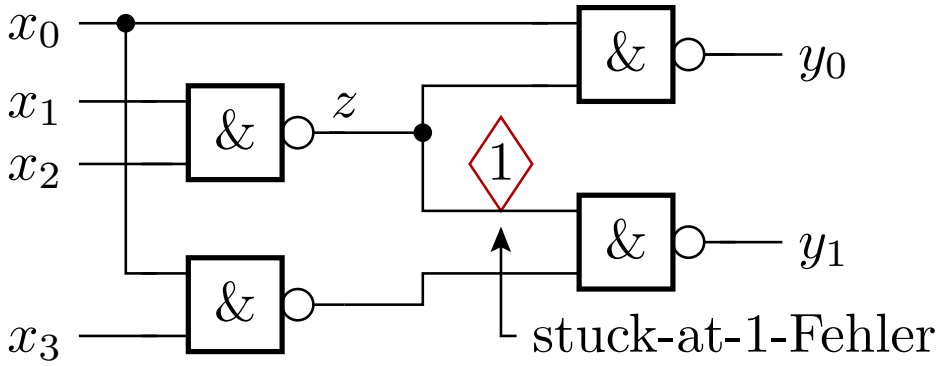
Aufgabe 4.12: Fehlernachweismengen

Bestimmen Sie für den eingezeichneten Haftfehler die Menge der 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.



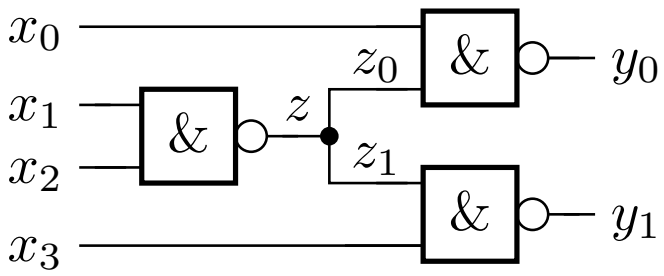
Hinweis: Notation der Eingabemengen als Kreuze in der Wertetabelle auf der nächsten Folie.



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 4.13: 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.

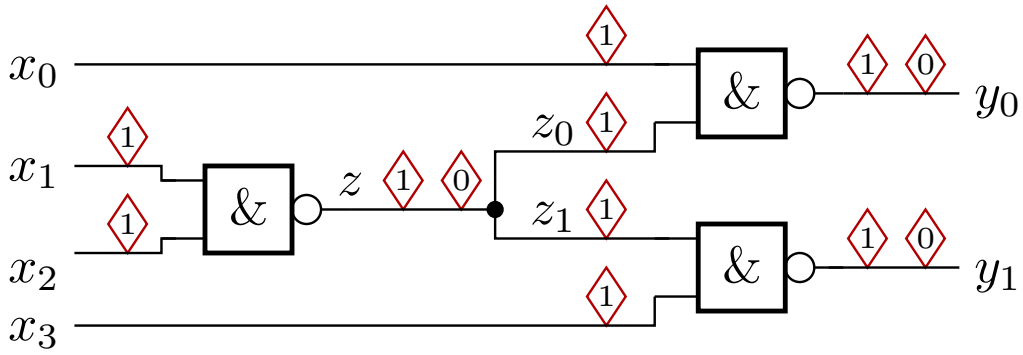


x_3	x_2	x_1	x_0	z	y_1	y_0
0	1	1	0			
1	1	0	0			
1	1	1	0			

Hinweis: Verwenden Sie als Hilfmittel die Wertetabelle rechts.

Aufgabe 4.14: Haftfehlerüberdeckung

1. Bestimmen Sie für die Eingaben des Toggle-Tests aus der Aufgabe zuvor, welche die eingezeichneten Haftfehler nachweisen.
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															

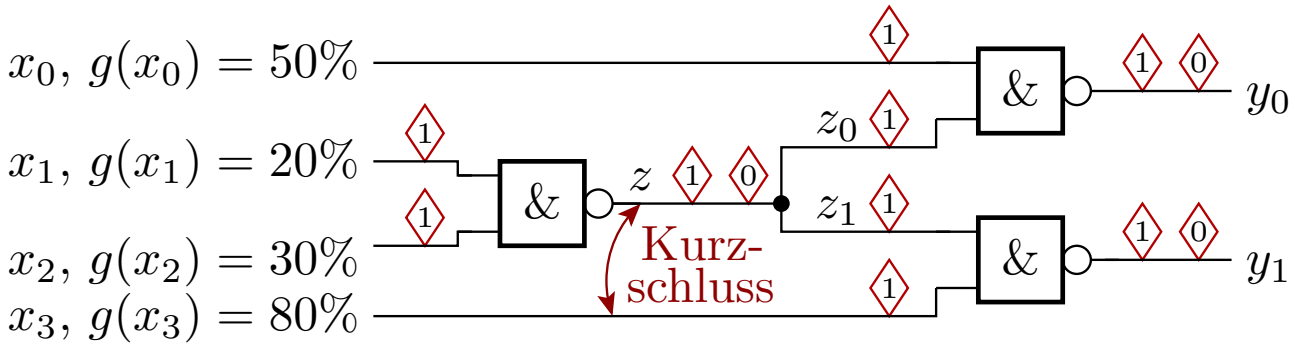
Aufgabe 4.15: Gatterverzögeungsfehler

1. Für welche der Haftfehler in der Aufgabe zuvor wird der korrespondierende Gatterverzögerungsfehler nachgewiesen.
2. Wie groß ist die Gatterverzögerungsfehlerüberdeckung des betrachteten 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 4.16: Kurzschlussnachweis

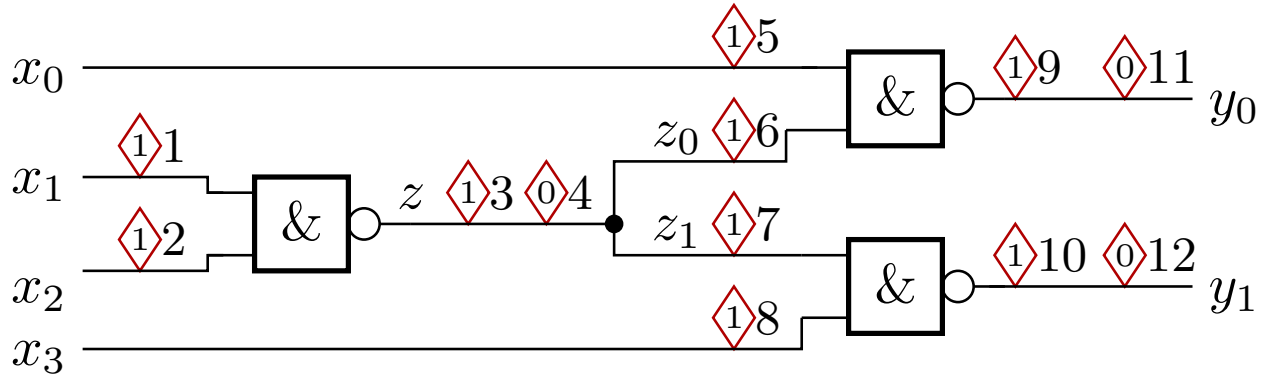
Bei dem eingezeichneten Kurzschluss soll sich null durchsetzen. Welche der eingezeichneten Haftfehler teilen sich Nachweisbedingungen mit dem Kurzschluss und wie groß ist für jeden dieser Haftfehler die Wahrscheinlichkeit, dass wenn ein Test ihn nachweist, auch der Kurzschluss nachgewiesen wird?



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

Aufgabe 4.17: Fehlersimulation

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



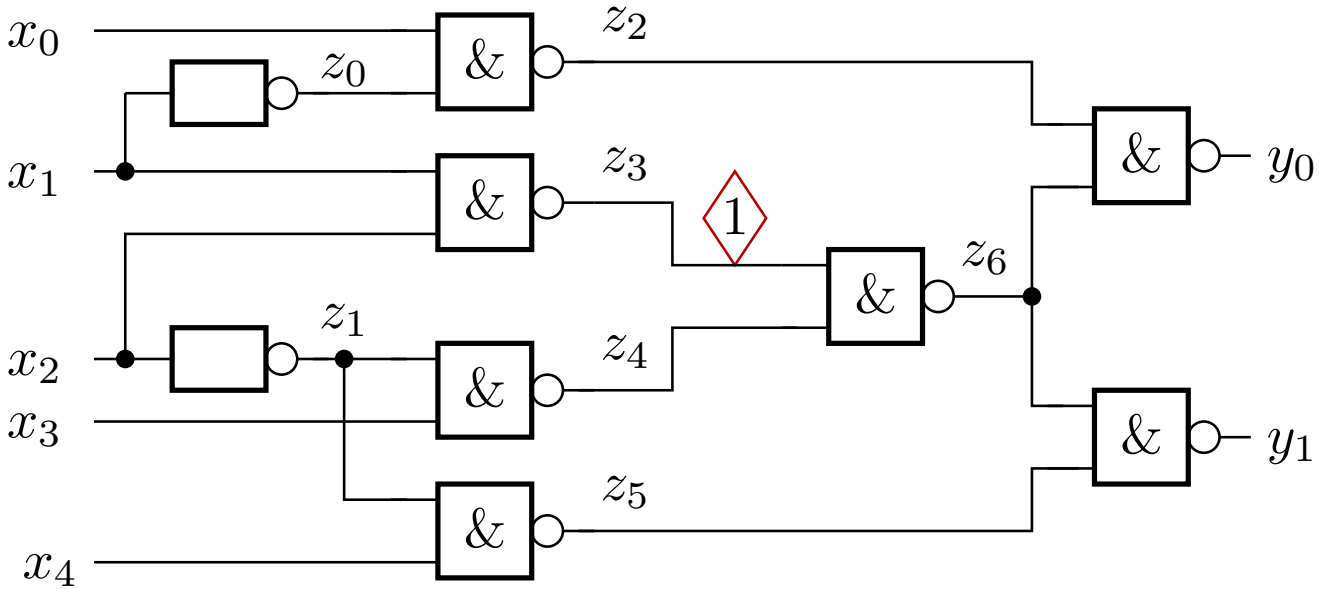

```
unit16_t x0, x1, x2, x3, z, z0, z1, y;
<wiederhole für jeden Testschritt>
 <lade Testeingaben in x0 bis x3, 0x0 oder 0xFF>
 <zu ergänzende Anweisungsfolge>
 <Protokollierung der erkannten Fehler>
```

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.

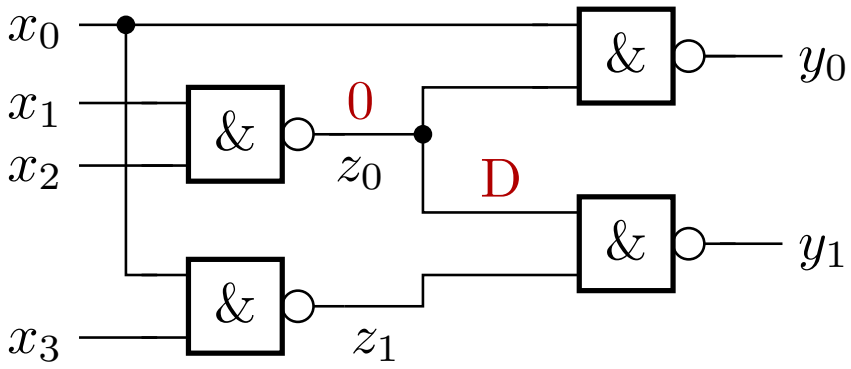
Aufgabe 4.18: D-Algorithmus

Berechnen Sie mit dem D-Algorithmus einen Eingabevektor für den Nachweis des eingezeichneten sa1-Fehler. Treiben Sie dazu zuerst einen Beobachtungspfad in Richtung y_0 und dann einen Steuerpfad in Richtung x_1.



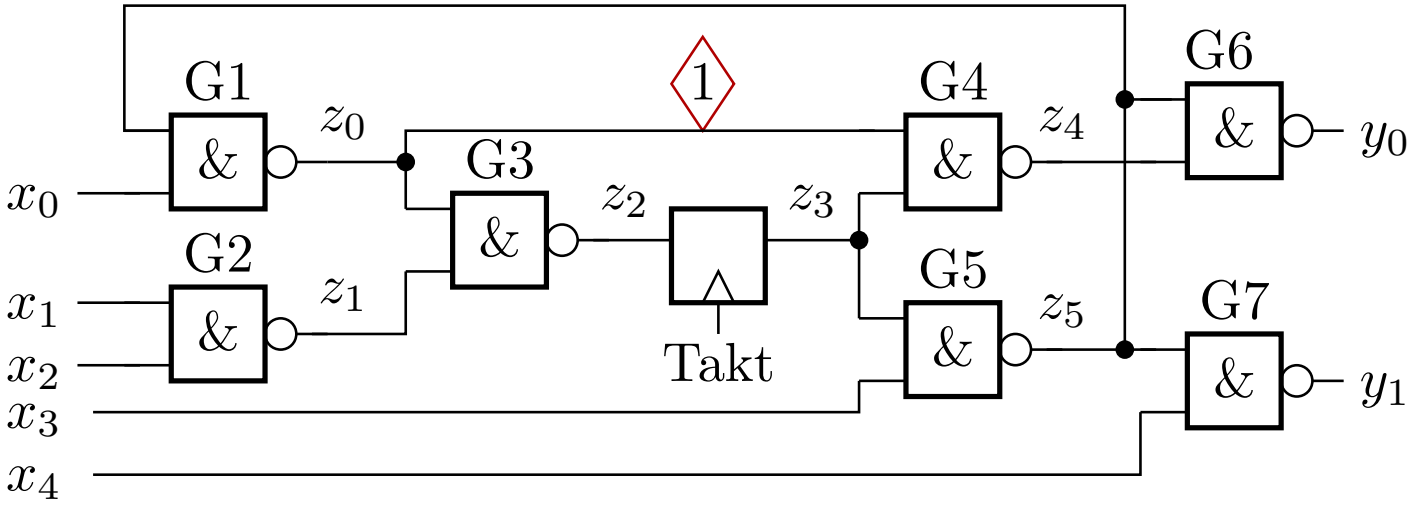
Aufgabe 4.19: Implikationstest

Bestimmen Sie für die nachfolgende Schaltung mit den beiden Signalfestlegungen (einmal »0« und einmal »D«) alle damit implizit festgelegten Signalwerte.



Aufgabe 4.20: Kombinatorische Ersatzschaltung

Rollen Sie die nachfolgende Schaltung zu einer kombinatorischen Ersatzschaltung für die Testberechnung des eingezeichneten Haftfehler mit einer Begrenzung der Länge der Steuer- und Beobachtungspfade auf max. drei Iterationen durch rückgekoppelte Schaltungsteile auf[19].



[19] Speicherzustände davor seinen für die Testberechnung unbestimmt und Speicherzustände danach nicht beobachtbar.

Aufgabe 4.21: Überlebenswahrscheinlichkeit

1. Wie groß ist die Überlebenswahrscheinlichkeit eines Systems mit einer über die Zeit konstanten Ausfallraten von $\lambda = 1000\,\text{fit}$ nach einer Nutzungsdauer von 100 Tagen?
2. Wie lang darf das Zeitinvall sein, in dem das System gewartet wird[20], damit die Überlebenswahrscheinlichkeit nicht kleiner als 99% wird?

[20] Wartung hier im Sinne von Test und Ersatz oder Repartur ausgefallener Systeme.

Aufgabe 4.22: Stressbetrieb

Ein Rechner wird eine Woche mit erhöhter Betriebsspannung übertaktet. Mindert oder erhöht das die Ausfallrate innerhalb der nachfolgenden ein bis zwei Jahre[21], wenn sich der Rechner

1. in der Frühphase,
2. in der Gebrauchsphase oder
3. in der Ermüdungsphase befindet?

[21] Die Ermügungsphase beginnt erst nach zwei Jahren, in der Regel mit dem Austrocknen der Elektrolytkondensatoren in den Netzteilen.

Aufgabe 4.23: Mittlere Lebensdauer eines Rechners

Wie groß ist die mittlere Lebensdauer eines Rechners aus 30 Schaltkreisen mit einer Ausfallrate von 150 fit, 100 diskreten Bauteilen mit einer Ausfallrate von 30 fit und 500 Lötstellen mit einer Ausfallrate von 0,5 fit?

Aufgabe 4.24: Dauerbetrieb oder Ausschalten?

Das Netzteil eines Rechners habe im normalen Betrieb eine Ausfallrate von 9000 fit. Im ausgeschalteten Zustand sei die Ausfallrate 0. Bei einem Einschaltvorgang werden die Bauteile des Netzteils stärker belastet, so dass das Netzteil mit einer Wahrscheinlichkeit von 0,01% ausfällt. Ab welcher Ausschaltdauer erhöht das Ausschalten die zu erwartende Lebensdauer des Rechners?

SW und HW-Entwürfe

Software und Hardware-Entwürfe

SW und HW-Entwürfe haben Gemeinsamkeiten:

- ähnlicher Entstehungsprozess und damit ähnliche zu erwartende Fehler,
- Fehlersimulation und Testauswahl mit Mutationen, statt mit Modellfehlern,
- einfache Modularisierung, Zugang zu Zwischenergebnissen, ...

Agile Entwürfe (nur SW und HW-Konfigurationen):

- Weiterentwicklung und Fehlerbeseitigung im Einsatz.
- Zyklische Herausgabe neuer Versionen mit erweiterten Eigenschaften, beseitigten alten und entstandenen neuen Fehlern.
- Nutzung der Anwender als Tester.

Zu erwartende Fehler

Entwurfsphasen:

- Zusammenstellen der Anforderungen
- Festlegung der Systemarchitektur
- Kodierung.

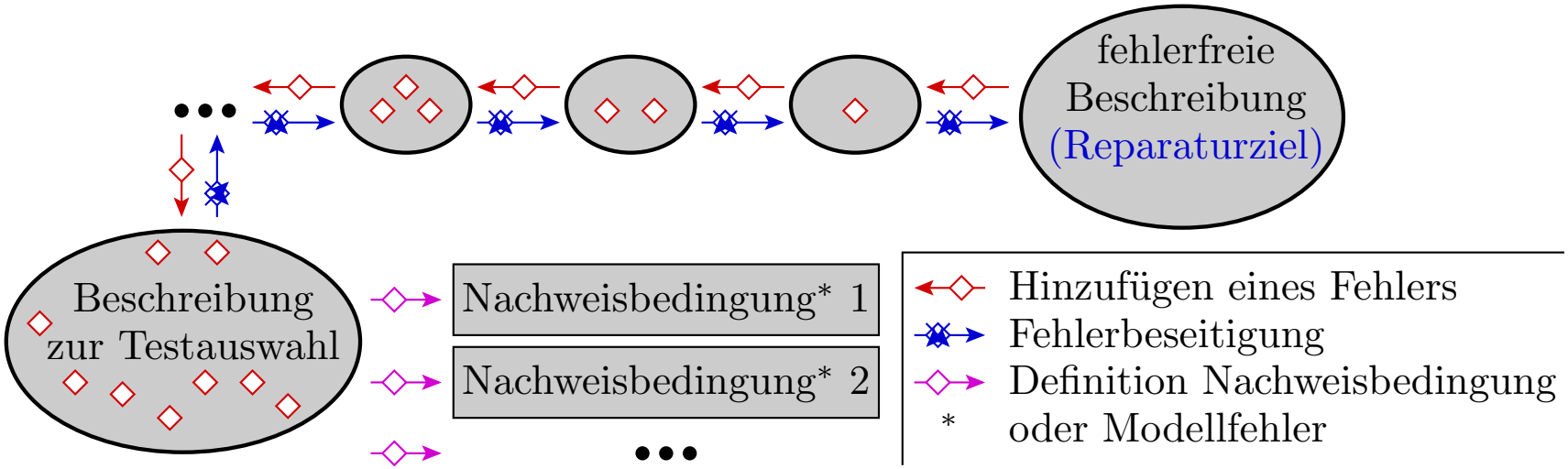
Fehler können in jeder Entwurfsphase entstehen:

- falsche, fehlende Anforderungen,
- falsche, fehlende Systemeigenschaften bei der Umsetzung,
- falsche, fehlende selbstverständliche Eigenschaften, z.B., dass ein Programm installierbar und bedienbar sein muss.
- Programmierfehler.

Tests:

- Reviews aller enstehenden Dokumente incl. Programme.
- statische Tests (Syntax, Programmierrichtlinien, ...)
- dynamische Tests, bei HW Simulation von Beispielen.

Mutationen



- Für Entwürfe gibt es keine korrekte Sollbeschreibung.
- Statt der Modellfehler lassen sich nur Mutationen einer potenziell fehlerhaften Beschreibung konstruieren.
- Die Konstruktion ähnlich nachweisbarer Mutationen setzt ein überwiegend korrekte Systembeschreibung voraus.

Zusammenstellen von Mutationen

Mutationen auf der Hochsprachenebene sind geringfügige Verfälschungen im Programmtext:

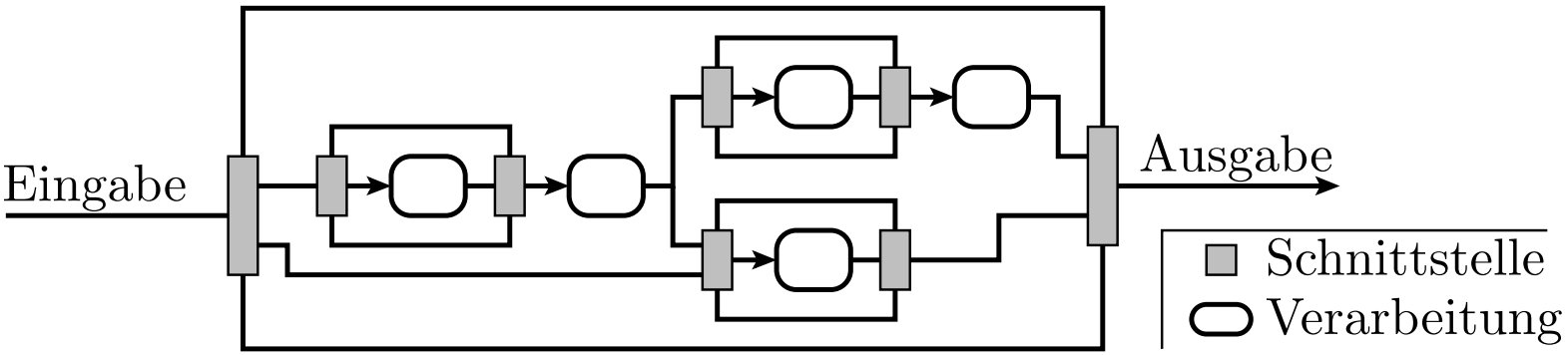
- Verfälschung arithmetischer Ausdrücke (x=a+b ⇒ x=a*b)
- Verfälschung boolescher Ausdrücke (if(a>b){} ⇒ if(a<b){})
- Verfälschung der Wertezuweisung (value=5 ⇒ value=50)
- Verfälschung der Adresszuweisung (ref=obj1 ⇒ ref=obj2)
- Entfernen von Schlüsselworten (static int x=5 ⇒ int x=5)

Mutationen der Anforderungen, des Systementwurfs, ...:

- Wie beschreibt man eine geringfügig falsche Anforderungen?
- Eine fehlende Anforderung lässt sich nicht mutieren.

- Ähnlich nachweisbare Mutationen lassen sich nur für einen Teil der Fehlermöglichkeiten aufstellen.
- Gezielte Testauswahl nur begrenzt sinnvoll.

Einfache Modularisierung



- IT-Systeme werden hierarchisch entworfen.
- Beim Entwurf wird jeder Teil-Service und jeder übergeordnete Service über seine Schnittstellen seperat getestet.
- Zur Kontrolle und Fehlersuche können beim Test (bei SW im Debugger, bei HW im Simulator) beliebige interne Variablen und Signale überwacht werden.

In kleinen Modulen mit zusätzlichen Beobachtungspunkten besitzen Fehler hohe Nachweiswahrscheinlichkeiten.

Testauswahl für Software und HW-Entwürfe

Auf Basis des Quellcodes:

- kontrollflussorientiert,
- datenflussorientiert.

Auf Basis der Zielfunktion: Entwurf einer alternativen (diversitären) Beschreibung spezieller Aspekte der Zielfunktion für die Testauswahl:

- Aufteilung des Eingaberaums in Äquivalenzklassen,
- Ursache-Wirkungs-Modell,
- Automatengraph.

Zufallstest:

- Modultest mit Pseudo-Zufallszahlen als Eingabe.
- Systemtest: Vorgabe von Testsenarien mit einer Mischung aus vorgegebenen und zufälligen Bedatungen.

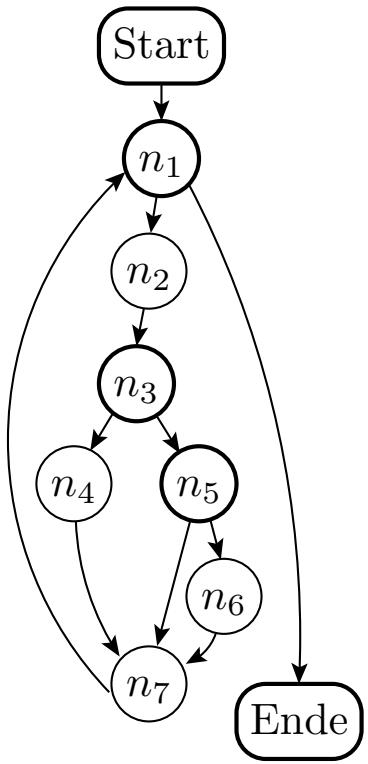
Kontrollflussorientiert

Ein Beispielprogramm und sein Kontrollflussgraph

```
type tCounter is record A, B, N: NATURAL; end record;

function ZZ(Ct_max: NATURAL) return tCounter is
 variable z: CHARACTER;
 variable Ct: tCounter:=(0,0,0);
begin
n1: for Ct.N<Ct_max loop
n2:  LeseZeichen(z);
n3:  if is_TypA(z) then
n4:   Ct.A:=Ct.A+1;
n5:  elsif is_TypB(z) then
n6:   Ct.B:=Ct.B+1;
     end if;
n7:  Ct.N := Ct.N +1;
    end loop;
 return Ct;
end function;
```

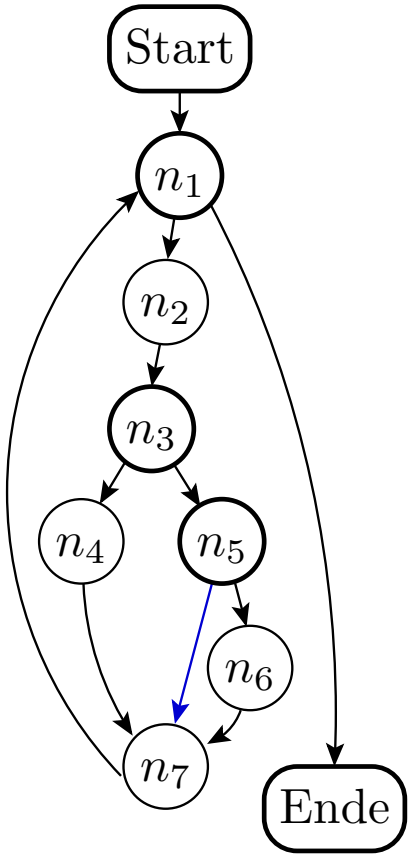
Testauswahlgrundlage
Kontrollflussgraph



Auswahlkriterien für Tests

Die in der Industrie verbreiteten Auswahlregeln auf Basis des Kontrollflussgraphen:

1. Jede Anweisung muss mindestens einmal ausgeführt werden. Beispiel: Start n_1, n_2, n_3, n_4, n_7, n_1, n_2, n_3, n_5, n_6, n_7, n_1, Ende
2. jede Kante muss mindestens einmal durchlaufen werden. Beispiel: Start n_1, n_2, n_3, n_4, n_7, n_1, n_2, n_3, n_5, n_6, n_7, n_1, n_2, n_3, n_5, n_7, n_1, Ende
3. Jede Entscheidung muss mindestens einmal von jeder Bedingung abhängen.



bei 100% Anweisungsüberdeckung möglicher weise ungetestete Kante

Bedingungsüberdeckung

```
n1: if (((A>10) and (A<20)) or ((B>1) and (B<100)))
        and (C=1) then
n2:  ...
    else
n3: ...
```

Kontrolle, dass die Entscheidung mindestens einmal von jeder Bedingung abhängt:

A	B	A>10	A<20	B>1	B<100
10	50	P(A>10)	–	–	–
11	50	N(A>10)	–	–	–
19	50	–	(A<20)	–	–
...	...	...	...	...	...

(P(...) – Fehler bei Vergrößerung um eins; N(...) – Fehler bei Verkleinerung um eins).

.

100% Überdeckungsforderung

- 100% Anweisungsüberdeckung: Gilt für nicht sicherheitskritische Systeme als ausreichend.
- 100% Zweigüberdeckung: Vom Standard RTCA DO-178 B gefordert ab Level C (Software, die bedeutende Ausfälle verursachen kann).
- 100% Bedingungsüberdeckung: Vom Standard RTCA DO-178 B fordert für Level A (flugkritische Software).

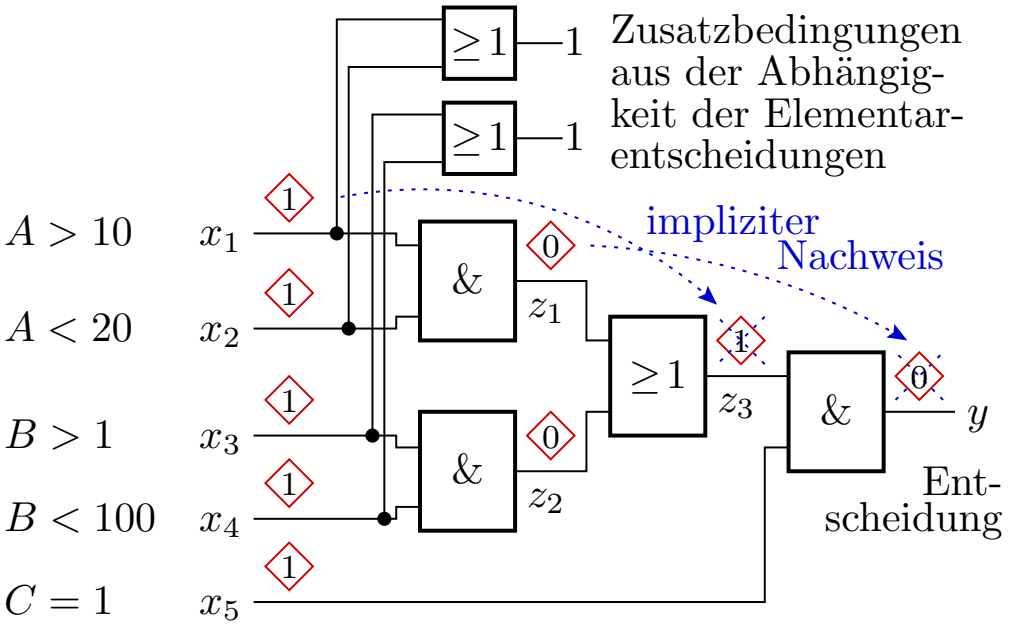
- Die Ausführung einer Anweisung/Verzeigung ist nur eine Anregungsbedingung für eine Mutation $H(p) \sim H_{\text{Mut}}(c \cdot p)$ $c \ll 1$ (vergl. Toggle-Test, Folie 141).
- Zufällige Auswahl: Testsatz um Faktor c verlängern.

Geziele Auswahl: mehr als c Tests für jede Bedingung.

Vergleich mit Haftfehlern

```
n1: if (((A>10) and (A<20)) or ((B>1) and (B<100)))
        and (C=1) then
n2:   ...
    else
n3:   ...
```

Kontrolle, dass die Entscheidung mindestens einmal von jeder Bedingung abhängt. Identisch mit dem Nachweis aller Haftfehler in der äquivalenten Schaltung.

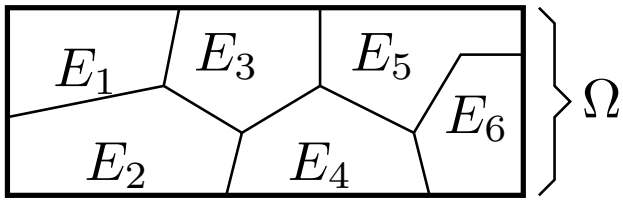


	x_5	x_4	x_3	x_2	x_1	y
sa1(x_1)	1	1	0	1	0	0
sa1(x_2)	1	0	1	0	1	0
sa1(x_3)	1	1	0	1	0	0
sa1(x_4)	1	0	1	0	1	0
sa1(x_5)	0	–	–	1	1	0
sa0(z_1)	1	1	1	1	0	1
sa0(z_2)	1	1	0	1	1	1

Äquivalenzklassen

Testauswahl mit Äquivalenzklassen

- Äquivalenzklasse: Eingabemenge identisch zu verarbeitender Daten.
- Ableitung aus der Aufgabenstellung und Beschreibung durch Fallunterscheidungen.



Ω Eingaberaum
E_i Äquivalenzklasse

- Wichtig sind die Fallunterscheidungen und ihre Diversität zur Implementierung.
- Für die für jede Eingabemenge auszuführende Funktion genügt eine Benennung oder informale Beschreibung.
- Testauswahl wie kontrollflussorientierte Auswahl[22]:
  - jede Funktion mit einer Datenstichprobe,
  - jeden Zweig mit einer Datenstichprobe,
  - jede Verzweigungsbedingung mit einer Datenstichprobe.

[22] Intuitiver Ansatz: Tests an den Bereichsgrenzen bevorzugen.

Modellierung des Testobjekts

- Beschreibung der Gesamtfunktion $f(\mathbf{x})$ durch Bedingungen $b_i(\mathbf{x})$ und Teilfunktionen $f_i(\mathbf{x})$:

  `wenn` $b_1(\mathbf{x})$ `dann` $y = f_1(\mathbf{x})$
  `wenn` $b_2(\mathbf{x})$ `dann` $y = f_2(\mathbf{x})$
  `wenn ...`

- ($\mathbf{x}$ – Vektor der Eingaben). Bedingungen bestehen aus Vergleichen und logischen Operationen, z.B.:

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

Zielfehler sind in der Implementierung fehlende oder falsch umgesetzte Anforderungen:

- Verlangt Diversität der Äquivalenzklassenbeschreibung zur Implementierung (unterschiedlicher Entwerfer, ...).
- Abweichungen sind meist an den Bereichsgrenzen sichtbar.

Testauswahl an den Bereichsgrenzen

Suche Eingaben, die Einzelbedingungen für kleine Bedingungs- oder Werteänderungen sensibilisieren (Bedingungsüberdeckung):

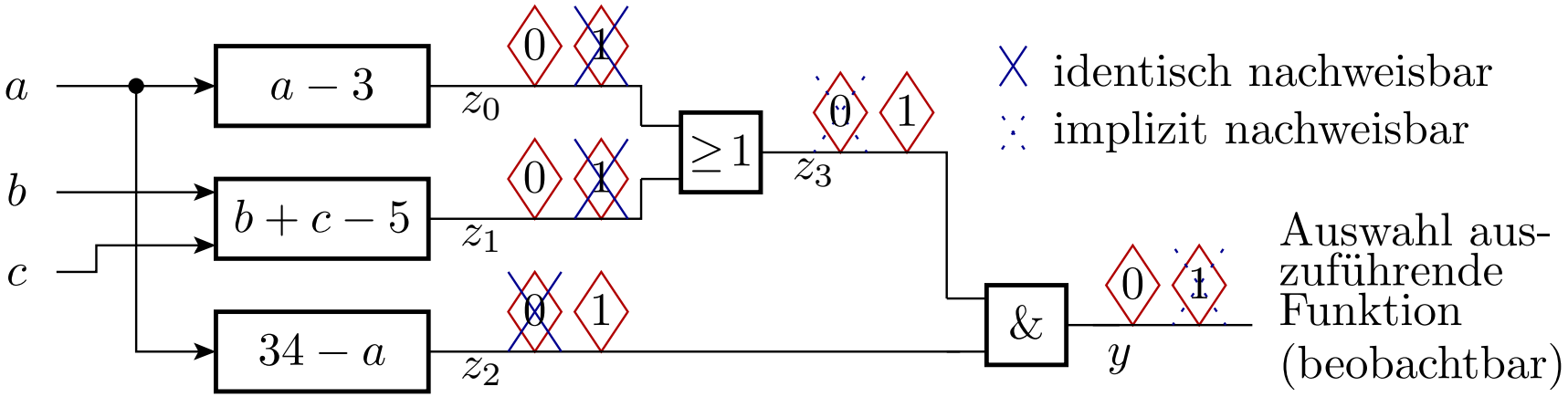
```
int a, b, c;
if ((a>3)|(b+c>5))&(a<34) <Ausführung Funktion A>;
```

a	b	c	Fehlernachweisbedingung
3	-0	-1	$(\underbrace{a>3\vert_{\text{sensib}}}_{a\in\{3,4\}}) \vee (\underbrace{b+c>5\vert_{\text{falsch}}}_{\text{z.B. } b=0,\, c=-1}) \wedge (\underbrace{a<34\vert_{\text{wahr}}}_{\text{erfüllt für a}\in\{3,4\}})$
34	7	9	$(\underbrace{a>3\vert_{\text{wahr}}}_{\text{erfüllt für } a\in\{33,34\}}) \vee (\underbrace{b+c>5\vert_{\text{wahr}}}_{\text{z.B. } b=7,\, c=9}) \wedge (\underbrace{a<34\vert_{\text{sensib.}}}_{\text{a}\in\{33,34\}})$

Zwei Modellfehlertypen für die sensibilisierte Bedingung:

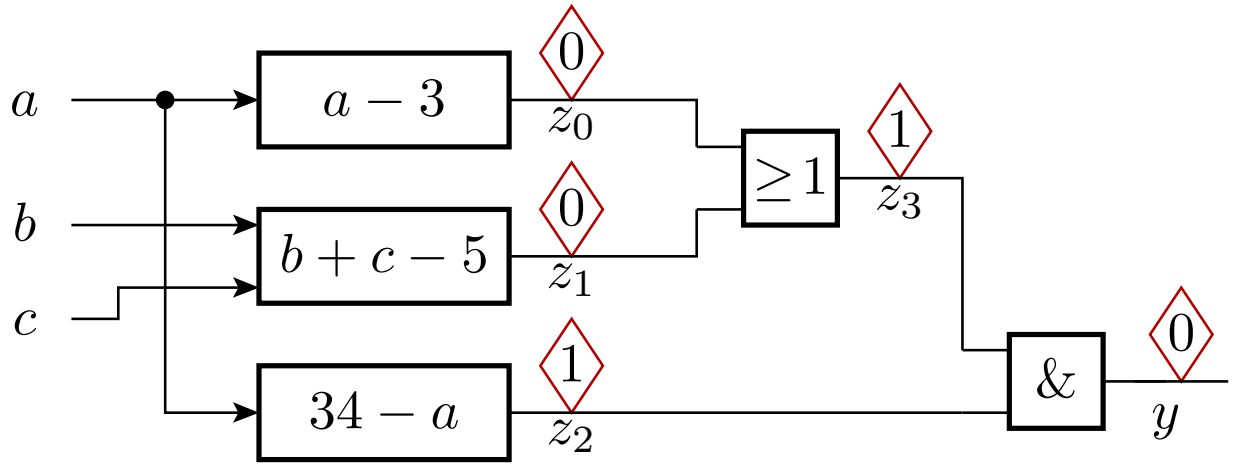
- Änderung von wahr nach falsch
- Änderung von falsch nach wahr

Vergleich mit Haftfehlermodellierung



- Zusammenstellung der Modellfehlermenge, zusammenfassen identischer Fehler, ...
- Sensibilisierung binärer Steuer- und Beobachtungspfade.
- Suche geeigneter Zahlenwert.
- Testauswahl wahlweise Zufallsauswahl + Fehlersimulation oder Testberechnung[23].

[23] Die logischen Verknüfungen werden meist so einfach sein, dass eine Zufallsauswahl genügt. Wegen $c < 1$ besser mehrere Tests je Modellfehler.

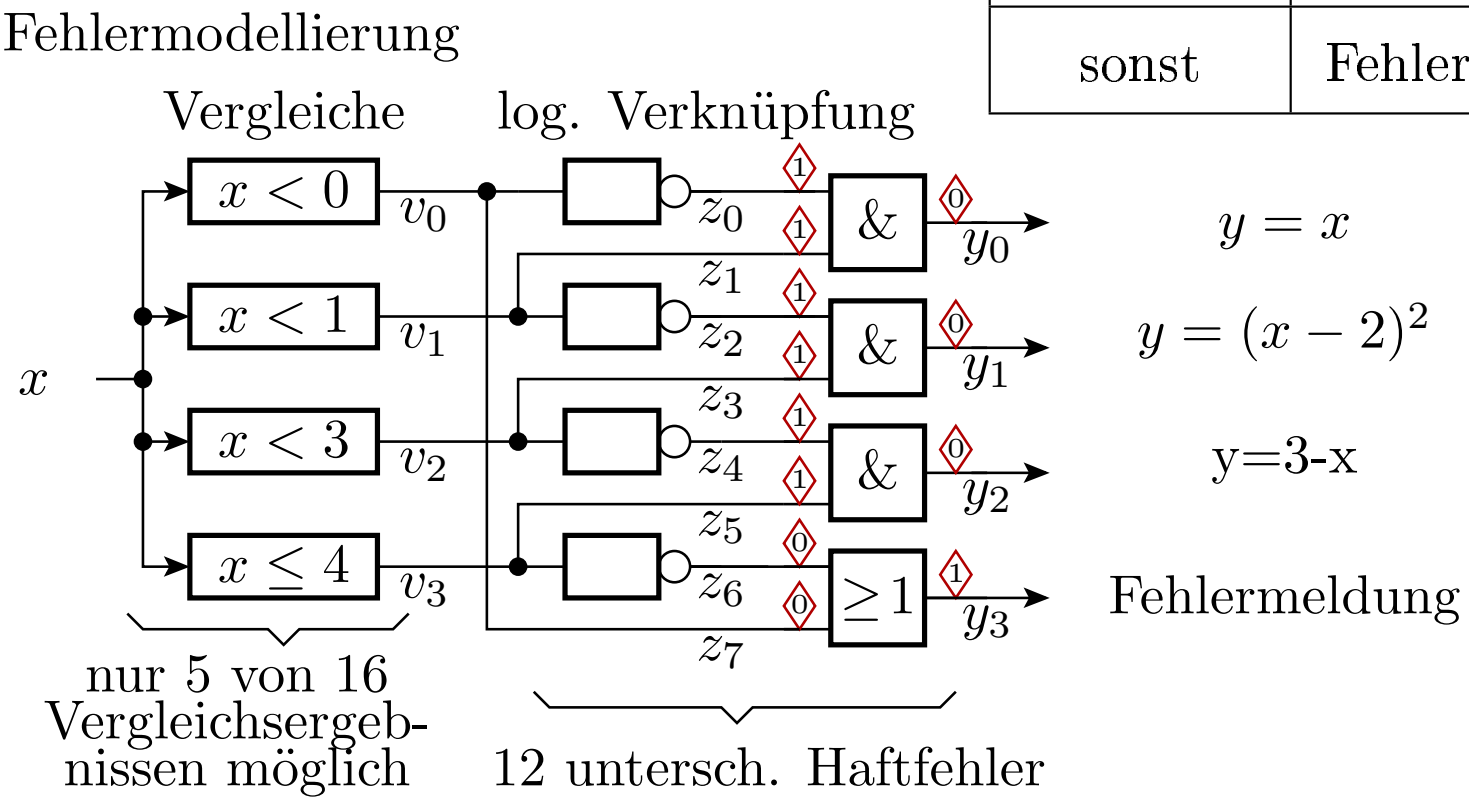


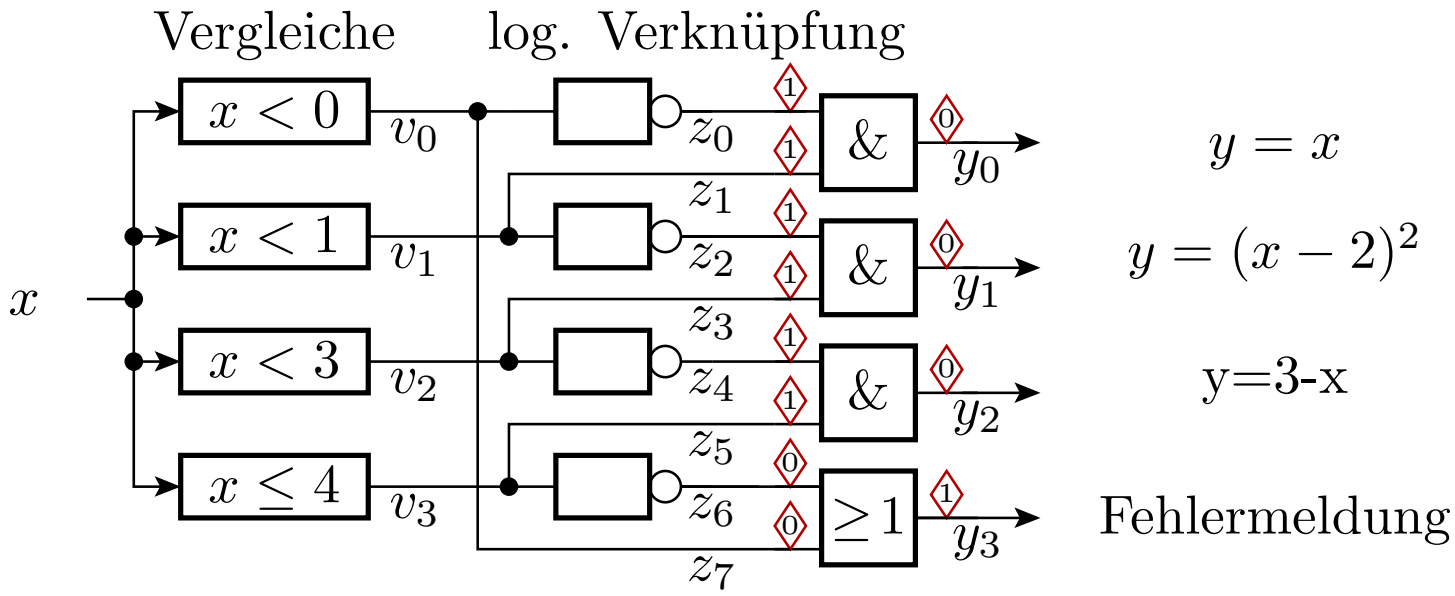
	$a < b$	$b + c > 5$	$a < 34$	a	b	c
sa0(z_0)						
sa0(z_1)						
sa1(z_2)						
sa1(z_3)						
sa0(y)						

D	sensibel falsch → wahr	0	falsch	X	beliebig
$\bar{D}$	sensibel falsch → wahr	1	wahr		

Komplette Abschnitts-weise definierte Funktion

x	y
$0 \leq x < 1$	$y := x$
$1 \leq x < 3$	$y := (x-2)^2$
$3 \leq x \leq 4$	$y := 3 - x$
sonst	Fehlermeldung





- Statt einer fehlerorientierte Testsuche bietet sich hier ein Test mit allen Variationendes Vergleichsergebnisse

v_0	1	$\bar{D}$	D	0	0	0	0	0	
v_1	1	1	1	1	$\bar{D}$	D	0	0	
v_2	1	1	1	1	1	1	1	$\bar{D}$	
v_3	1	1	1	1	1	1	1	1	
Beispiel für x	-23	$-\varepsilon$	0	0.17	1-ε	1	1,34	2-ε	

UW-Analyse

Ursache-Wirkungs-Analyse

Für die UW-Analyse ist aus der Zielfunktion eine Beschreibung abzuleiten aus:

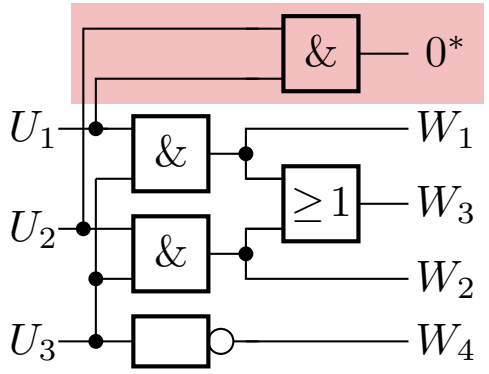
- Auslösern für Aktionen (Ursachen) und
- ausgelösten Aktionen (Wirkungen).

Auslöser (Ursachen) sind Eingabewertebereiche (ähnlich Äquivalenzklassen), die bestimmte Soll-Reaktionen zur Folge haben sollen.

Wirkungen sind einzeln spezifizierte Zielfunktionen, ergänzte selbstverständliche Funktionen und Fehlerbehandlungen. Jede Ursache und Wirkung wird durch eine binäre Variable (nicht eingetreten/eingetreten) beschrieben.

Zwischen den Ursachen und Wirkungen werden logische Bedingungen formuliert.

Beispiel: Zähle Zeichen



* Eingabezeichen kann nur Typ A oder B sein

- Wirkungen:
  W_1: Anzahl_TypA +1
  W_2: Anzahl_TypB +1
  W_3: Gesamtzahl +1
  W_4: Programm beenden
- Ursachen:
  U_1: Zeichen ist vom Typ A
  U_2: Zeichen ist vom Typ B
  U_3: Zeichenanzahl $<$ Maximalwert
- Sich ausschließende Ursachen:
  UND-Verknüpfung muss »0« sein.

Test mit allen einstellbaren Ursachen

U_1	0	1	0	1	0	1	0	1
U_2	0	0	1	1	0	0	1	1
U_3	0	0	0	0	1	1	1	1
W_1	0	0	0		0	1	0	
W_2	0	0	0		0	0	1	
W_3	0	0	0		0	1	1	
W_4	1	1	1		0	0	0	

- Eine Ursache-Wirkungs-Analyse deckt Mehrdeutigkeiten und Widersprüche in der Spezifikation auf

Testobjekt als VHDL- bzw. Ada-Funktion

```
type tCounter is record A, B, N: NATURAL; end record;

function ZZ(Ct_max: NATURAL) return tCounter is
 variable z: CHARACTER;
 variable Ct: tCounter:=(0, 0, 0);
begin
 for Ct.N<Ct_max loop
  LeseZeichen(z);
  ct.N := Ct.N +1;
  if   is_TypA(z) then Ct.A := Ct.A +1;
  elsif is_TypB(z) then Ct.B := Ct.B +1;
  end if;
  write(" z=" & str(z)    & " A=" & str(Ct.A)   (TA)
      & " B=" & str(Ct.B) & " N=" & str(Ct.N));(TA)
 end loop;
 write("Ende"); return Ct;
end function;
```

$^{(TA)}$ Testausgabe

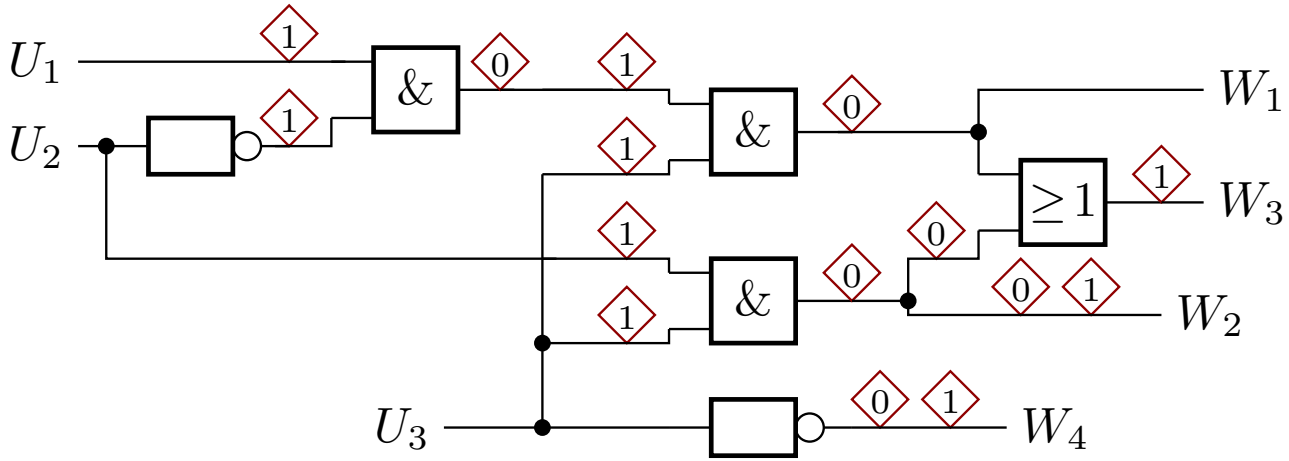
Funktionsaufrufe	Ausgaben	U_1	U_2	U_3	W_1	W_2	W_3	W_4
ZZ(3)	z=0 A=1 B=0 N=1	1	0	1	1	0	1	0
	z=A A=1 B=1 N=2	0	1	1	0	1	1	0
	z=x A=1 B=1 N=3	0	0	1	0	0	1	0
	Ende	–	–	0	0	0	0	1
ZZ(1)	z=1 A=1 B=0 N=1	1	0	1	1	0	1	0
	Ende	–	–	0	0	0	0	1
ZZ(1)	z=B A=0 B=1 N=1	0	1	1	0	1	1	0
	Ende	–	–	0	0	0	0	1
ZZ(0)	Ende	–	–	0	0	0	0	1

U_1 Zeichen ist eine Ziffer
U_2 Zeichen ist ein Großbuchstabe
U_3 max. Zählwert erreicht
– es wird kein Zeichen gelesen

W_1 A weiterzählen
W_2 B weiterzählen
W_3 N weiterzählen
W_4 Funktion beenden

Erkennbare Ungereimtheiten: Im UW-Graph gibt es keine Ursache für »anderes Zeichen gelesen«. Programm und Test berücksichtigen diesen Fall. Ist das so richtig?

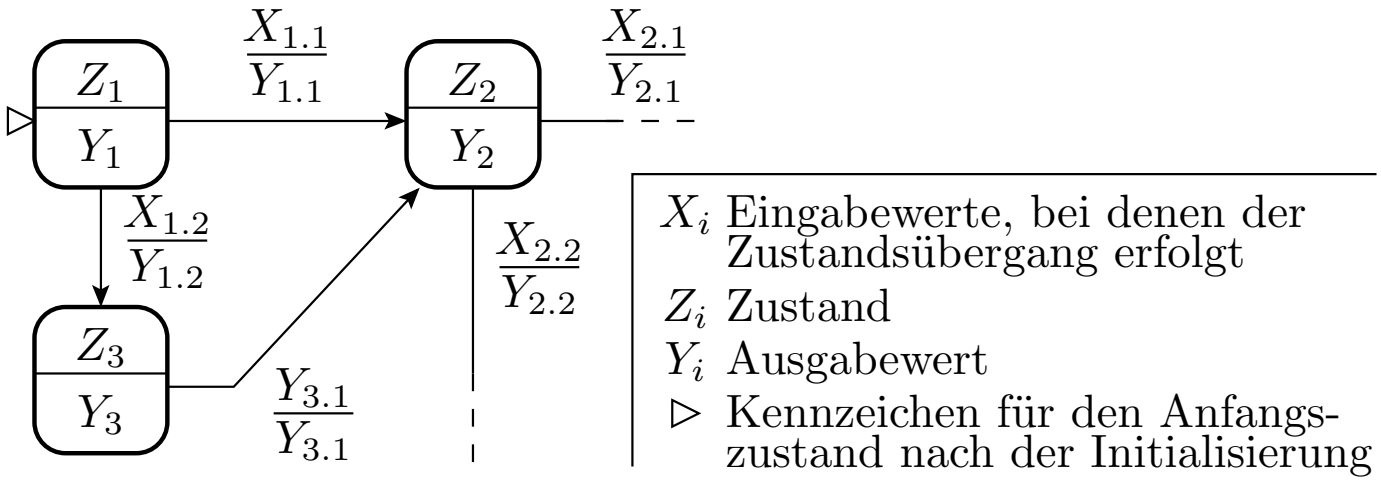
Zurückführung auf das Haftfehlermodell



Ein Ursache-Wirkungs-Graph kann wie eine logische Schaltung dargestellt werden. Für die Auswahl der Testfälle ist in der Regel zu fordern, dass bei jeder logischen Verknüpfung überprüft wird, dass jede Eingabeverfälschung in mindestens einem Test nachweisbar ist. Über das Haftfehlermodell gewonnene Tests sind gleichzeitig zu aktivierende Ursachen, die bestimme Wirkungskombinationen verursachen. Das sind nur Rahmenvorschriften für die Konstruktion der eigentlichen Testbeispiele.

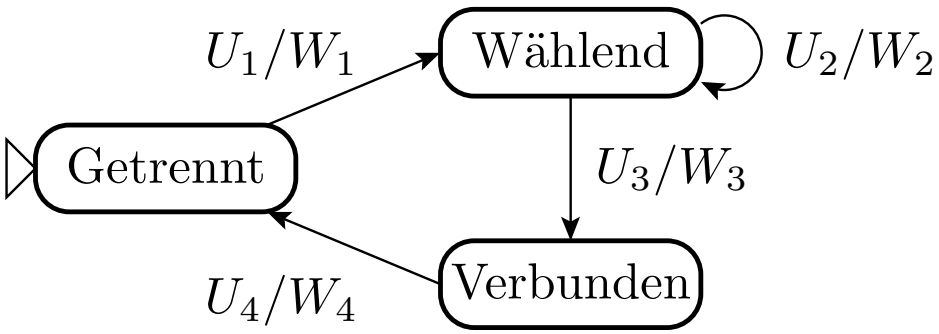
Automaten

Zielfunktion als Automat



Das Automatenmodell beschreibt die Zielfunktion eines Systems durch einen Menge von Eingaben, Ausgaben, Zuständen und Zustandsübergänge. Zustandsübergänge werden durch Eingaben ausgelöst. Bei den Übergängen und in den Zuständen werden Aktionen gesteuert. Wie im UW-Modell werden bei Automaten für die Testauswahl die Ursachen (Bedingungen für Zustandsübergänge) und die Wirkungen (gesteuerte Aktionen) binarisiert.

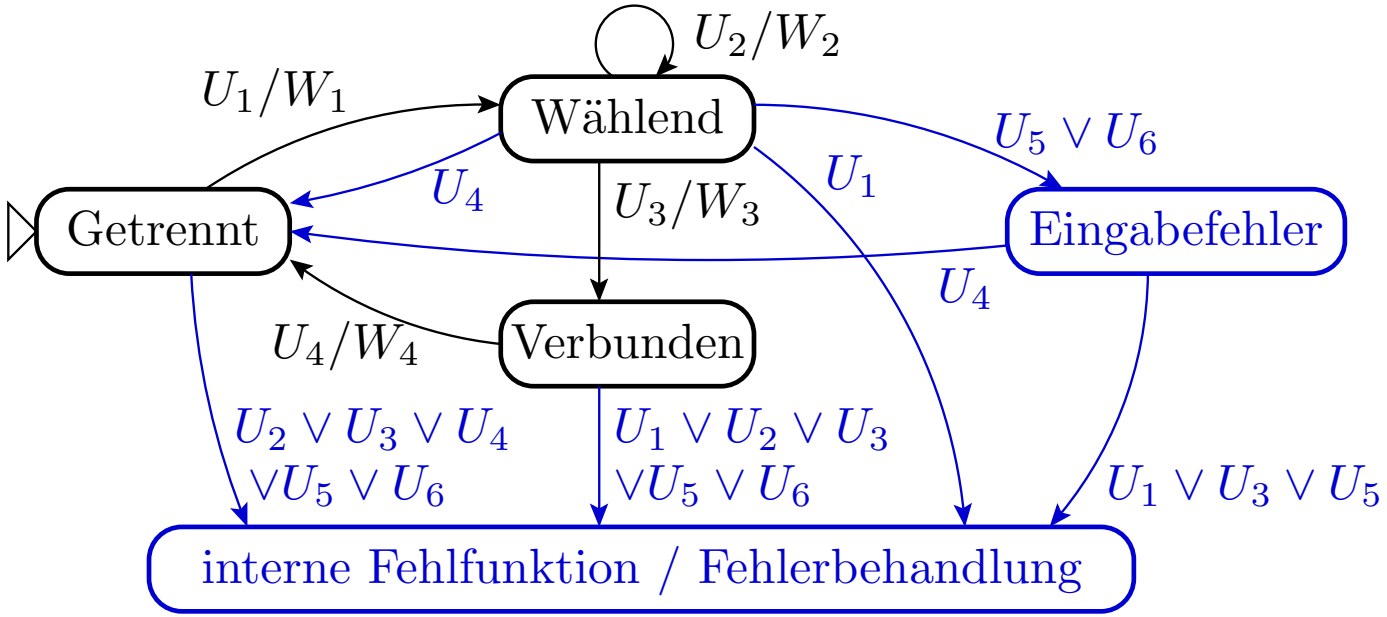
Verbindungsaufbau und -abbau beim Telefonieren



U_1	Abnehmen
U_2	Ziffer wählen
U_3	Rufnummer gültig
U_4	Auflegen
W_1	Rufnummer zurücksetzen
W_2	Ziffer zur Rufnummer hinzufügen
W_3	Verbindung aufbauen
W_4	Verbindung trennen

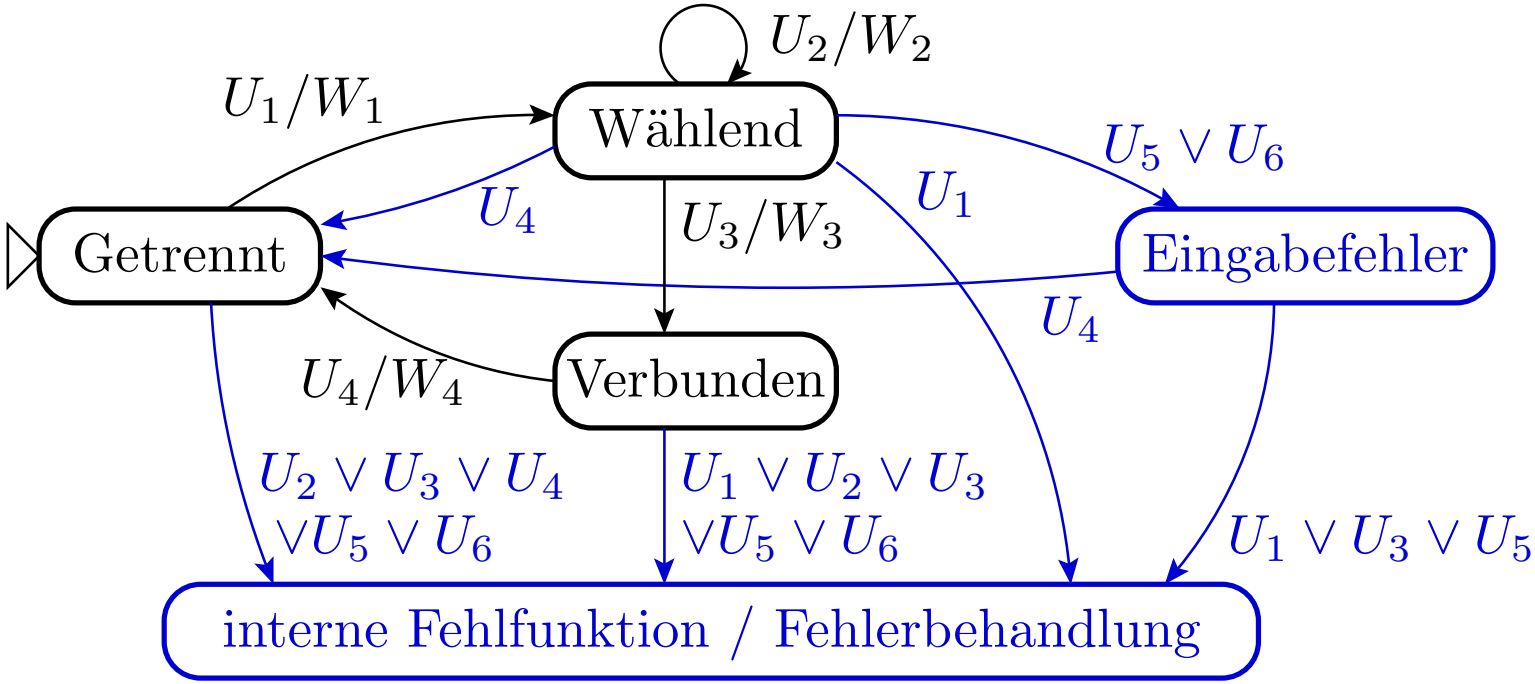
- Test der Soll-Funktion: $U_1 \rightarrow U_2 \rightarrow \ldots \rightarrow U_2 \rightarrow U_3 \rightarrow U_4$
- Verhalten für andere Eingabefolgen?
  - Abnehmen, Wählen, Auflegen ($U_1 \rightarrow U_2 \rightarrow U_4$)
  - Abnehmen, Wählen, Wählen, falsche Nummer)
  - ...

  $\Rightarrow$ Ablaufgraph ist noch unvollständig



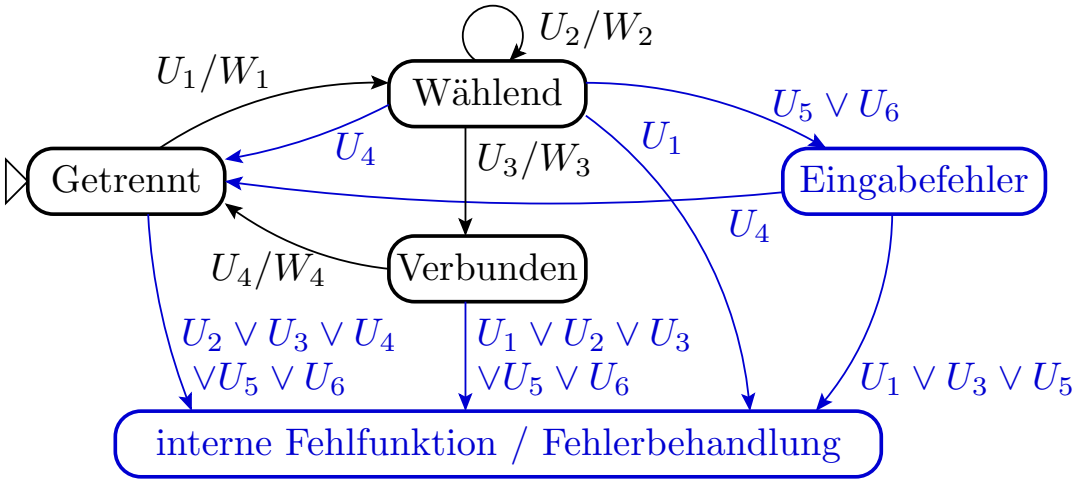
U_1	Abnehmen	W_1	Rufnummer zurücksetzen
U_2	Ziffer wählen	W_2	Ziffer zur Rufnummer hinzufügen
U_3	Rufnummer gültig	W_3	Verbindung aufbauen
U_4	Auflegen	W_4	Verbindung trennen
U_5	Rufnummer ungültig		
U_6	Timeout		

- Ergänzung um Knoten und Kanten für alle denkbaren Ursachen und Wirkungen. Präzisierung der Spezifikation.



Test aller Zustandsübergänge, Wirkungen, ...

- Abheben,Wählen, Wählen, Rufnummer gültig, Auflegen.
- Abheben, Wählen, Auflegen.
- Abheben, Wählen, Wählen, Timeout, Auflegen.
- Abheben, Wählen, Rufnummer ungültig, Auflegen.



Test der Reaktion auf interne Fehlfunktionen

- Initialisieren, Auflegen.
- Initialisieren, Rufnummer gültig, ...

Auswahlregeln sind wie bei der kontrollflussorientierten Auswahl:

- Ausprobieren aller Kanten (in Analogie zu 100% Zweigüberdeckung) oder
- jeder Übergang muss mindestens einmal von jeder Bedingung abhängen (Analogie Bedingungsüberdeckung, zurückführbar auf das Haftfehlermodell).

Aus einem Automatengraphen sind wie bei der UW-Analyse nur Rahmenvorschriften für die Konstruktion der eigentlichen Testbeispiele ableitbar, nämlich Folgen von auszulösenden Ursachen für die Kantenübergänge und erwartete Wirkungen in Form der den Kanten und Zuständen zugeordneten Aktionen.

Der zufällige Fehlernachweis für Automaten wird durch Markov-Kette beschrieben (vergl. Foliensatz F1).

Kontrollautomaten

Test einer Automatenansteuerung

Ein Automat, z.B. eine Maschine, Betriebssystem, ... , hat Regeln für die Ansteuerung. Beispielregeln für die Benutzung der Windows-API aus [1]:

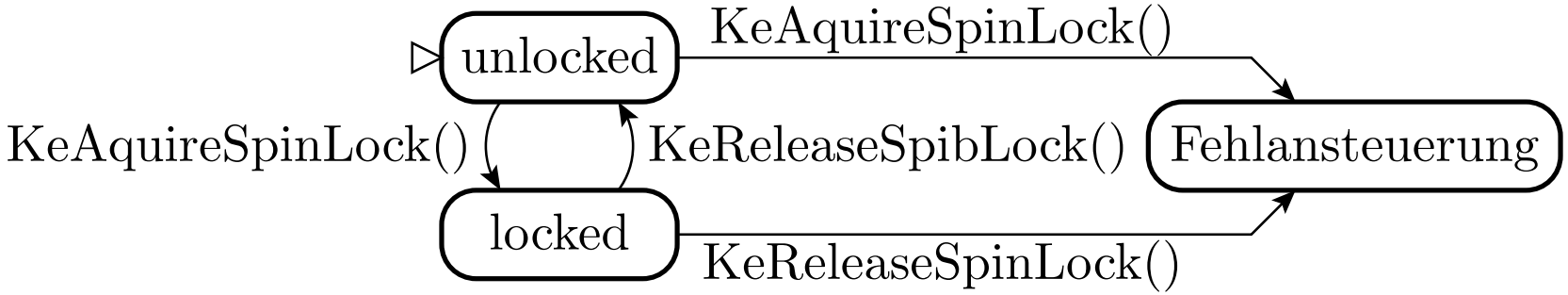
spinlock Spinlocks müssen alternierend reserviert und freigegeben werden.

spinlocksafe Vermeidung von Deadlocks mit Spinlocks.

criticalregions Problemvermeidung im Zusammenhang mit der Nutzung kritischer Regionen.

...

Kontrollautomat für Regel »spinlock«:



Eine zu testende Treiberfunktion

Eine Treiberfunktion ruft »KeAquire..« und »KeRelease...« u.U. mehrfach auf, in Fall-unterscheidungen, Schleifen, ... Für jeden Kontrollpfad muss der Spinlock alternierend bedient werden.

Fehlerausschluss erfordert Kontrolle für alle Pfade.

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

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

Idee in [1] erschöpfende statische Analyse unter Nutzung binärer Entscheidungsbäume.

Aufgaben

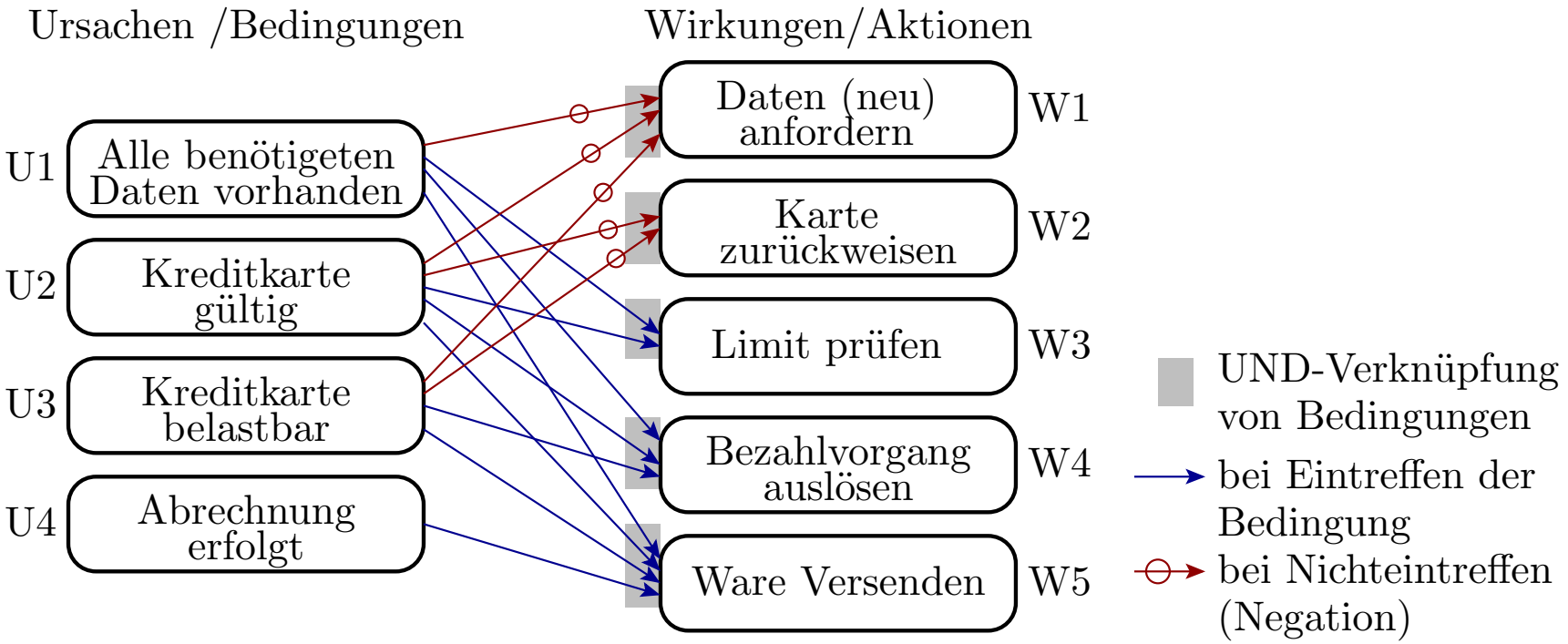
Aufgabe 4.25: Kontrollflussgraph

Stellen Sie für das nachfolgende Programm den Kontrollflussgraph und für diesen eine Zustandsfolge mit 100% Verzweigungsüberdeckung auf.

```
void Steuerschrittfunktion(uint16_t VAbst){
    SSF_Ct ++;
    if (SSF_Ct>=28){ // etwa alle 2 Sekunden
        SSF_Ct = 0;
        // Steuerzustand_A zirkular erhöhen: a,b,..,z,a,..
        zustand = getState(Steuerzustand_A);
        if (zustand<='z') zustand++;
        else zustand = 'a';
        setState(zustand, Steuerzustand_A);
        if (VAbst>2000) // Wenn Abstandsspannung größer 2V
        incErr(2);      // Fehlerzähler 2 erhöhen
    }
    startLCD();
}
```

Aufgabe 4.26: Ursache-Wirkungs-Analyse

Gegeben ist das Ergebnis einer Ursache-Wirkungs-Analyse in einer anderen Darstellung aus [http://test.silke-wingens.de/].

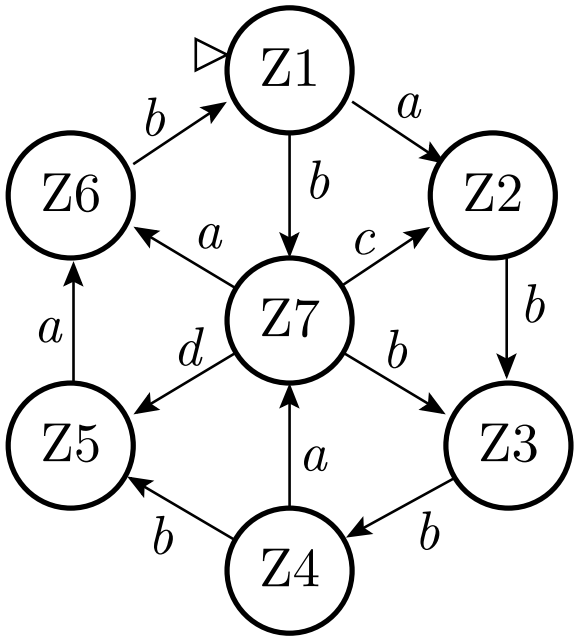


1. Stellen Sie die dargestellte Ursache-Wirkungs-Beziehung durch logische Verknüfungen dar.
2. Bestimmen Sie in dieser Darstellung eine Menge unterschiedlich nachweisbarer Haftfehler[24].
3. Bestimmen Sie für alle diese Haftfehler die Nachweiswahrscheinlichkeiten für nachfolgende Auftrittshäufigkeiten der Ursachen: $h(U1) = 20\%$, $h(U2) = 80\%$, $h(U3) = 40\%$ und $h(U4) = 30\%$.

[24] Ausgehend von der Anfangsfehlermenge mit je zwei Haftfehlern an jedem Gatteranschluss sollen redundante Fehler gesucht und ausgeschlossen und identisch nachweisbare Fehler zu einem Modellfehler zusammengefasst werden.

Aufgabe 4.27: Automatentest

Gegeben Sie der nachfolgende Automat mit symbolischen Zuständen und Eingaben:



Zustände: Z1, Z2, ...
Eingabewerte: a, b, c, d
Anfangszustand: Z1

Für jeden Zustand soll gelten, dass der Automat solange darin bleibt, bis er eine Eingabe zu seinem Verlassen bekommt.

1. Einwickeln Sie eine Ablauffolge, mit der alle Zustandsübergänge mindestens einmal ausprobiert werden.
2. Stellen Sie eine Markow-Kette in Matrix-Form auf, mit der sich die Wahrscheinlichkeit bestimmen lässt, das der mit d beschriftete Kantenübergang mindestens einmal getestet wurde. Zu Beginn sei der Automat im Anfangszustand $Z1$. Die Auftrittshäufigkeiten der symbolischen Eingabewerte seien $P(a) = 10\%$, $P(b) = 40\%$, $P(c) = 3\%$ und $P(d) = 5\%$.
3. Bestimmen Sie die Wahrscheinlichkeit, dass die Kante d mindestens einmal getestet wird als Funktion der Anzahl der Automatenschritte.

Literatur

4. Literatur

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

[2] Thomas S. Barnett and Adit D. Singh.
Relating yield models to burn-in fall-out in time.
pages 77–84, 2003.

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

[4] R. Kärger.
Diagnose von Computern.
Teubner, 1996.

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

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

[7] Frank Padberg, Thomas Ragg, and Ralf Schoknecht.
Using machine learning for estimating the defect content after an inspection.

IEEE Transactions on Software Engineering, 30(1):17–28, 2004.

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