Test und Verlässlichkeit Foliensatz 5: Dynamische Tests

Prof. G. Kemnitz

21. Juni 2018

Contents

Dynamische Tests

- Ausprobieren der Funktion mit einer Stichprobe von Eingaben.
- Testauswertung in der Regel Vergleich mit Sollwerten.
- Testdurchführung idealerweise automatisch.

Voraussetzung für die Durchführbarkeit:

- Entwurf / Fertigung abgeschlossen.
- Statisch nachweisbare Fehler beseitigt, insbesondere bei HW und physikalischer Anbindung mit Zerstörungsgefahr.

- Testbeispiele und Testumgebung (Prüftechnik, Testrahmen, ...).

Prüfgerechter Entwurf:

- Service-Struktur, Zugang zu den Anschlüssen, Zugang zu den internen Zuständen.
- Isolierte Testbarkeit von Teilsystemen, insbesondere solcher mit nicht deterministischen Sollverhalten, Schnittstellen zur Analogverarbeitung, ...

Testdurchführung und Testauswahl

- Das Testobjekt ist ein Service, der in zeitdiskreten Schritten Eingaben auf Ausgaben abbildet..
- Der Tester / Testrahmen stellt für jeden Testschritt die Eingaben im geforderten Format bereit und kontrolliert / protokolliert die Ausgaben.

Auswahl der Testbeispiele:

- »Alles« lässt sich nicht überprüfen.
- Gezielte Testauswahl für den Modellfehlernachweis.
- Zufällige Auswahl in Bezug auf die zu erwartenden Fehler.
- Gezielte Auswahl von Operationsprofilen ...

1 Gezielte Testauswahl

Vollständiger Test unmöglich

Für den Nachweis, dass ein Service für alle Eingabedaten korrekte Ergebnisse liefert, müsste er mindestens mit allen Eingaben ausprobiert werden. Bereits ab wenigen Eingabebits unmöglich:

	m	2^m	t^*
Gatter, 4 Eingänge	4	16	16 µs
ALU, 68 Eingänge	68	$3 \cdot 10^{20}$	10^7 Jahre
vier Eingabevariablen vom Typ int32_t	128	$3 \cdot 10^{38}$	10^{25} Jahre

(m – Anzahl der Eingabebits; 2^m – Anzahl der Eingabemöglichkeiten; t^* – Testdauer bei einer Service-Ausführungszeit von 1µs.)

- Die meisten Systeme verarbeiten mehr als 128 Eingabebits.
- Hinzu kommen oft tausende oder mehr gespeicherte Bits.
- Geschätzte Zeit seit dem Urknall $14 \cdot 10^9$ Jahre.

1.1 Testauswahlkriterien

Testauswahlkriterien

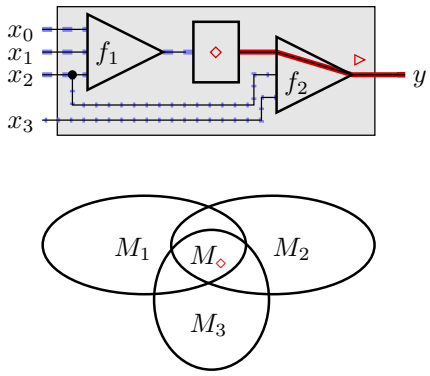
Tests erfolgen mit einer winzigen Stichprobe der Eingabemöglichkeiten.
Auswahlkriterien für die Testeingaben:

- Nachweis ausreichend vieler Fehler mit akzeptablem Aufwand,
- vor allem der Fehler, die die Mehrheit der FFs erzeugen und
- die erheblichen Schaden verursachen können.

Fehlernachweisbedingungen

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

- den Fehler anregen (lokale Datenverfälschung bewirkt) und
- einen Beobachtungspfad erzeugen, entlang dem sich die Verfälschung zu einem beobachtbaren Ausgang fortplanzt.

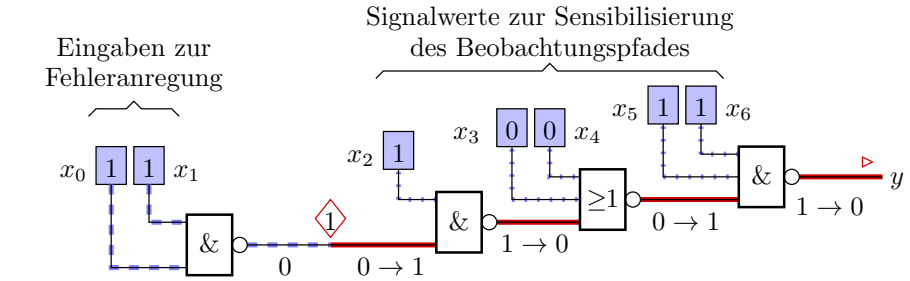


◇ Fehler
▷ Fehlfunktion (Datenverfälschung)
Eingaben zur Fehleranregung
Einstellen der Beobachtbarkeit
Beobachtungspfad

M_1 Eingabemenge, mit der der Fehler angeregt wird.
M_2 Eingabemenge, bei der der Fehlerort beobachtbar ist.
M_3 Eingabemenge für den lokalen Fehlernachweis.
$M_\diamond$ Nachweismenge des Fehlers

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

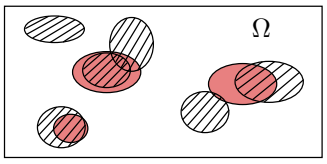
Nachweisbedingungen in einer Gatterschaltung



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

Tatsächliche Fehler und Modellfehler



Ω Menge der Eingabewerte / Teilfolgen die einen Fehler nachweisen können
Nachweismenge eines Modellfehlers
Nachweismenge eines tatsächlichen Fehlers

- Die tatsächlichen Fehler sind zum Zeitpunkt der Testauswahl unbekannt.
- Statt dessen wird mit einem Fehlermodell eine Modellfehlermenge generiert.
- Idealerweise teilt sich jeder tatsächliche Fehler mit mehreren Modellfehlern Nachweisbedingungen.
- Jeder für einen korrespondierenden Modellfehler gefundene Test weist den potentiellen Fehler mit einer Wahrscheinlichkeit nach, die von der Überlappung der Nachweismengen abhängt.

Beispiel

Wie hoch ist die zu erwartende Fehlerüberdeckung (mittlere Fehlernachweiswahrscheinlichkeit), wenn es für jeden tatsächlichen Fehler 3 korrespondierende Modellfehler gibt, bei deren Nachweis der tatsächliche Fehler mit Wahrscheinlichkeit von 40% nachgewiesen wird.

1. Wenn für jeden Modellfehler genau ein Test gesucht und für FC_{M} Modellfehler ein Test gefunden wird,
2. wenn für jeden Modellfehler zwei Test gesucht werden, für FC_{M} Modellfehler ein Test und $0{,}9 \cdot FC_{\mathrm{M}}$ zwei Tests gefunden werden?

Lösung

1. Im Mittel wird jeder tatsächliche Fehler von $3 \cdot FC_{\mathrm{M}}$ Tests mit einer Wahrscheinlichkeit von 40% nachgewiesen:
$$E\left(FC_{1\mathrm{T}}\right) \approx 1 - (1 - 40\%)^{3 \cdot FC_{\mathrm{M}}}$$
2. Im Mittel wird jeder tatsächliche Fehler von $3 \cdot FC_{\mathrm{M}} + 3 \cdot 0{,}9 \cdot FC_{\mathrm{M}}$ Tests mit einer Wahrsch. von 40% nachgewiesen:
$$E\left(FC_{2\mathrm{T}}\right) \approx 1 - (1 - 40\%)^{5{,}7 \cdot FC_{\mathrm{M}}}$$

FC_{M}	50%	75%	90%	99%	100%
$E\left(FC_{1\mathrm{T}}\right)$	53,5%	68,3%	75,8%	78,1%	78,4%
$E\left(FC_{2\mathrm{T}}\right)$	76,7%	88,7%	92,7%	94,4%	94,6%

Offenkundig hängt die zu erwartende Fehlerüberdeckung bei einer gezielten Testsuche weniger von der Modellfehlerüberdeckung, sondern mehr von der Anzahl der Tests je Modellfehler ab.

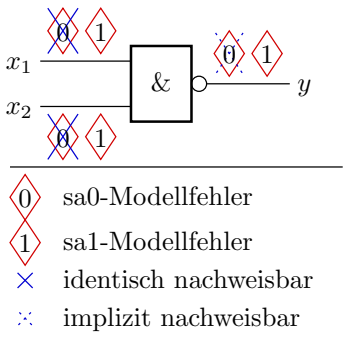
1.2 Fehlersimulation

Testsuche durch Fehlersimulation

Wiederhole, bis genügend Modellfehler nachgewiesen sind:

- Geziele, manuelle oder zufällige Auswahl weiterer Testbeispiele
- Fehlersimulation und Anhaken der nachweisbaren Modellfehler

Zuvor wird mit einem Fehlermodell eine Menge von Modellfehlern zusammgestellt, identisch nachweisbare Modellfehler zu einem zusammengefasst, redundante und optional implizit nachweisbare Fehler gestrichen.



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

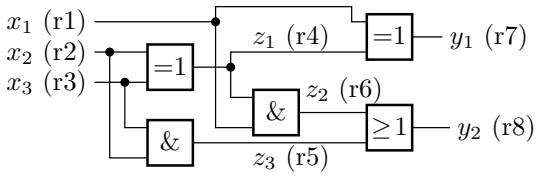
Nachweisidentität (gleiche Nachweismenge)

Nachweisimplikation

zugehörige Eingabe ist Element der Nachweismenge

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 jedem der 32 bzw. 64 Bits ein anderer Fehler simuliert wird, nur 6 bzw. 3 Minuten.

Wenn bereits nachgewiesene Modellfehler nicht weiter mit simuliert werden unter 1 Minute.

1.3 D-Algorithmus

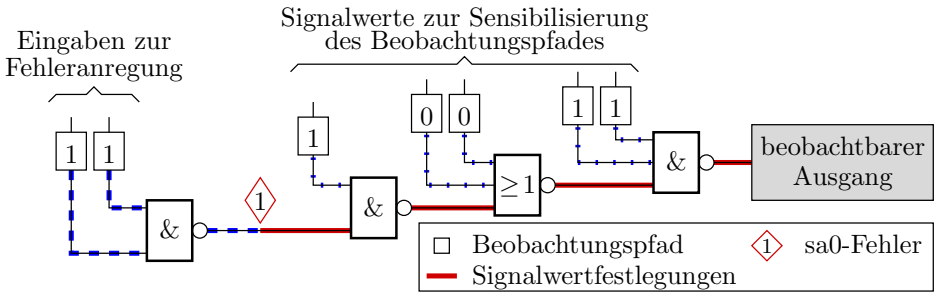
Testsuche für Haftfehler

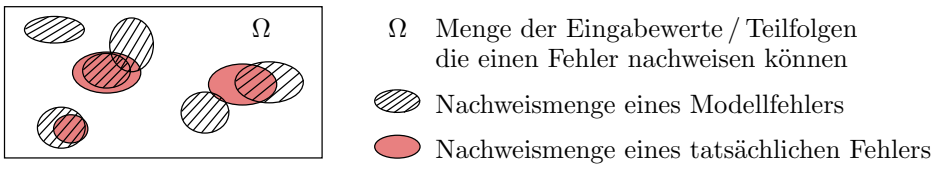
Ein Haftfehler unterstellt für den Fehlerort, dass der Wert

- entweder ständig 0 (sa0) oder
- ständig 1 ist (sa1) ist.

Ausgehend vom Fehlerort werden Eingaben gesucht,

- die den Wert am Fehlerort invertieren und
- bei denen die Invertierung am Fehlerort an einem Ausgang beobachtbar ist.





So gefundene Tests teilen sich mit allen lokalen Fehlern am selben Ort Anregungs- und Beobachtungsbedingungen. Wie später gezeigt, sind beim Schaltungstest zwar andere Fehler zu erwarten, aber das Haftfehlermodell erfüllt dennoch seinen Zweck.

Ansätze zur gezielten Testsuche für SW bevorzugen ähnliche binarisierte statt realistische Fehlerannahmen, da

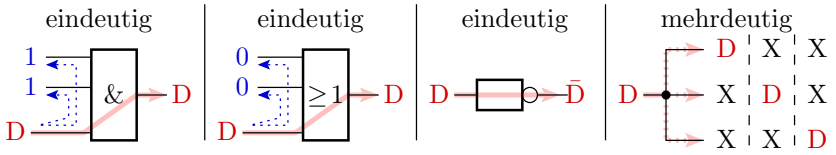
- geringerer Rechenaufwand für Fehlersimulation und Testberechnung und
- hinrichend akzeptabler Korrelation zwischen Fehler- und Modellfehlerüberdeckung.

D (Discrepancy)-Kalkül von Roth

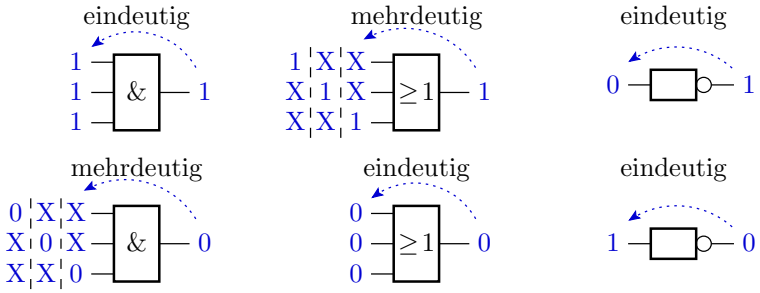
Erweiterung der Logikwerte um 3 Pseudo-Werte[1]:

D	0 wenn unverfälscht, 1 wenn verfälscht.
$\bar{D}$	1 wenn unverfälscht, 0 wenn verfälscht.
X	Signalwert ist ungültig oder für den Fehlernachweis ohne Bedeutung.

Regeln für die Sensibilisierung eines Beobachtungspfades:

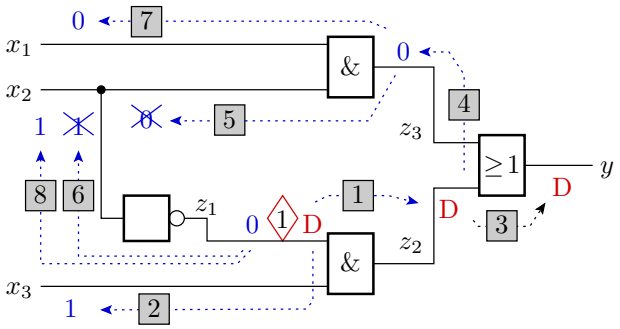


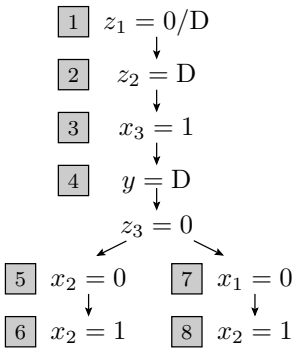
Einstellung von Steuerwerten



- Jede kombinatorische Schaltung kann durch eine Schaltung aus AND, OR, NOT nachgebildet werden.
- Später Verallgemeinerung auf LUT (**L**ook-**U**p **T**able, Tabellenfunktionen).

[1] W. Daehn: Testverfahren in der Mikroelektronik: Methoden und Werkzeuge. Springer 1997.





(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.

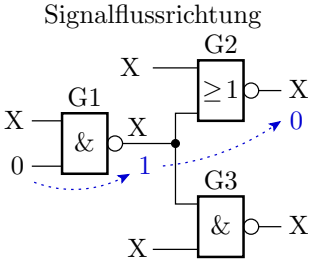
Heuristiken:

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

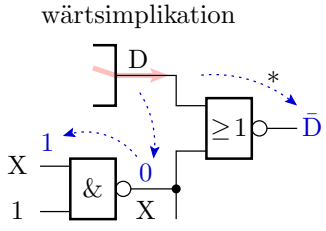
Implikationstest (Widerspruchsfrüherkennung)

- Aus den berechneten Wertefestlegungen alle eindeutig folgenden Werte berechnen.

Implikation in Signalflussrichtung



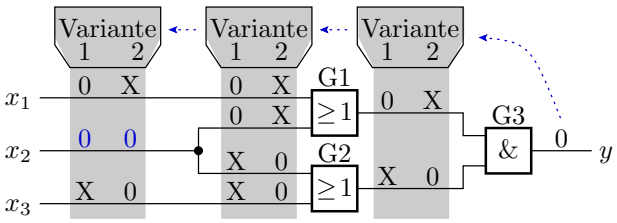
D-Pfad- und Rückwärtsimplikation



* eindeutige Weiterführung

- Mindert die Entscheidungsbaumtiefe.

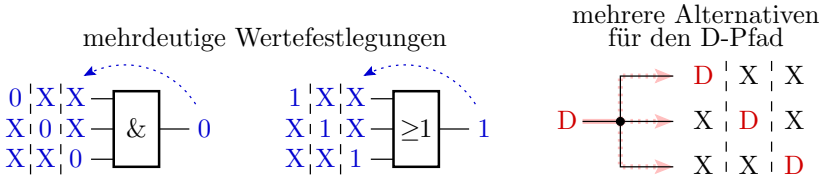
- Rückwärtsimplikation über mehrere Gatterebenen:



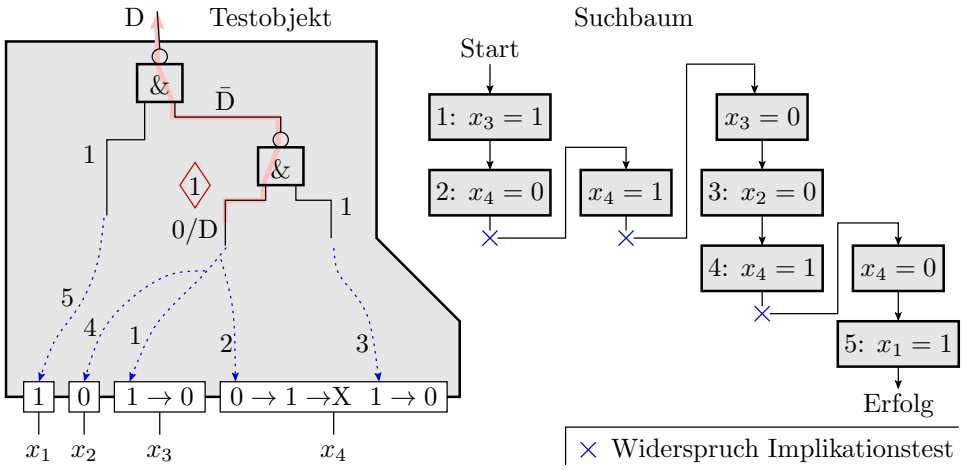
- Für $y = 0$ gibt es zwei Einstellmöglichkeiten.
- Für beide Möglichkeiten muss $x_2 = 0$ sein.
- Das Erkennen von Implikationen dieser Art mindert die Backtracking-Häufigkeit um bis zu 80 %.

Suchraumbegrenzung

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

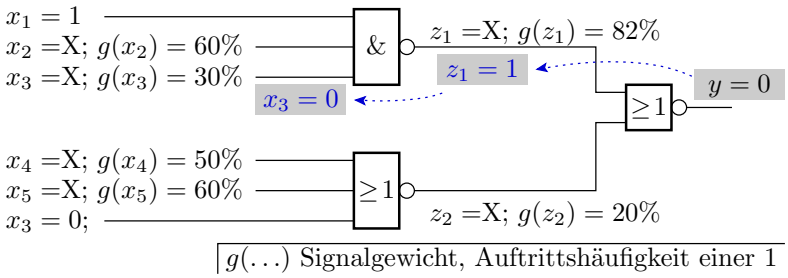


- Nur die Schaltungseingänge können unabhängig voneinander alle Wertevariationen annehmen.
- Es genügt, den Suchbaum mit den Eingabewertefestlegungen aufzubauen.
- Begrenzt Suchraum auf 2^{N_E} (2^{N_E} – Eingangsanzahl). Verringert Rechenaufwand um Zehnerpotenzen.



- 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 Signalwichtungen[2] über eine kurze Simulation mit Zufallswerten oder analytisch.
- Wahl der Steuerwerte / Beobachtungspfade, die mit größerer Wahrscheinlichkeit aktivierbar / sensibilisierbar sind.

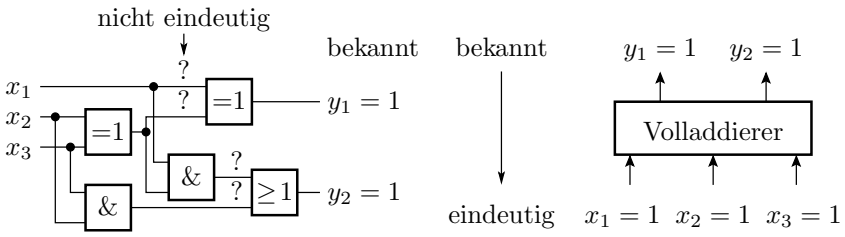
Komplexe Funktionsbausteine

- Beschreibung durch Tabellenfunktion (Bsp. Volladdierer):

x_2	x_1	x_0	s	c	gegeben	Lösungsmenge
0	0	0	0	0	XXX00	⇒ 00000
0	0	1	1	0		
0	1	0	1	0	01DXX	⇒ 01D$\bar{\text{D}}$D
0	1	1	0	1		
1	0	0	1	0		
1	0	1	0	1	1XXXD	⇒ 10D$\bar{\text{D}}$D, 1D0$\bar{\text{D}}$D
1	1	0	0	1	11XX1	⇒ 11111, 111001
1	1	1	1	1		

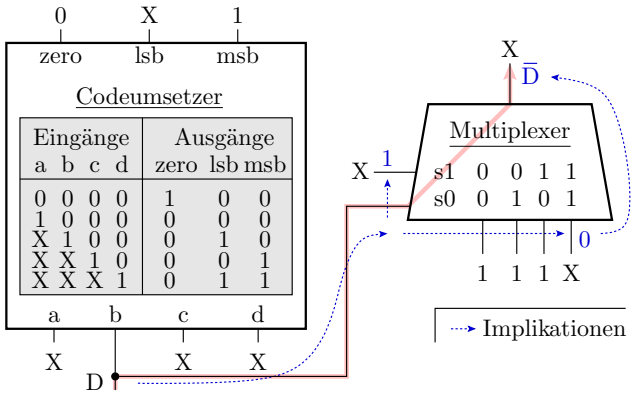
- Vervollständigung des Vektors der gegebenen Anschlusswerte durch Vergleich mit allen Tabellenzeilen:
  - »1« und »0« passen nur auf »1« und »0«.
  - »X« passt immer.
  - »D« muss für »D=0« und für »D=1« passen.

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.

[2] Die Wichtung eines Signals ist die Auftrittshäufigkeit einer »1«.

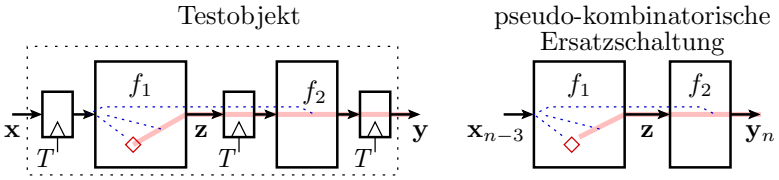


- »lsb« hängt bei »zero=0« und »msb=1« nicht von »b« ab. Eindeutiger D-Pfad über Multiplexer.
- Tabelleneingabewerte »X« (Eingang beeinflusst nicht die Ausgabe) führt zu Tabellen mit $\ll 2^{N_E}$ Tabellenzeilen (N_E – Anzahl der Eingänge).

1.4 Systeme mit Gedächtnis

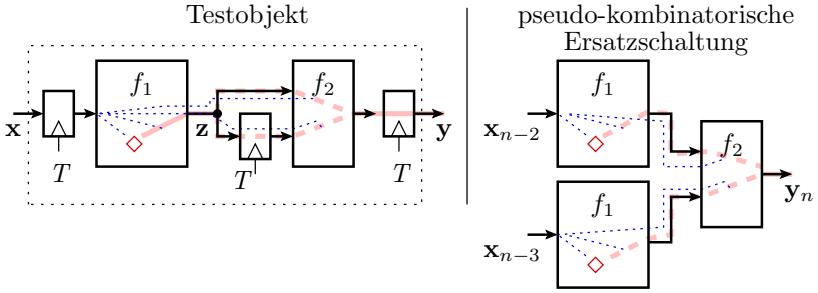
Pfadsensibilisierung für Systeme mit Gedächtnis

Schaltungen mit Speicherelementen werden für die Testsuche zu einer pseudo-kombinatorischen Ersatzschaltung aufgerollt. Abtastregister in einem geradlinigen Berechnungsfluss werden weggelassen:



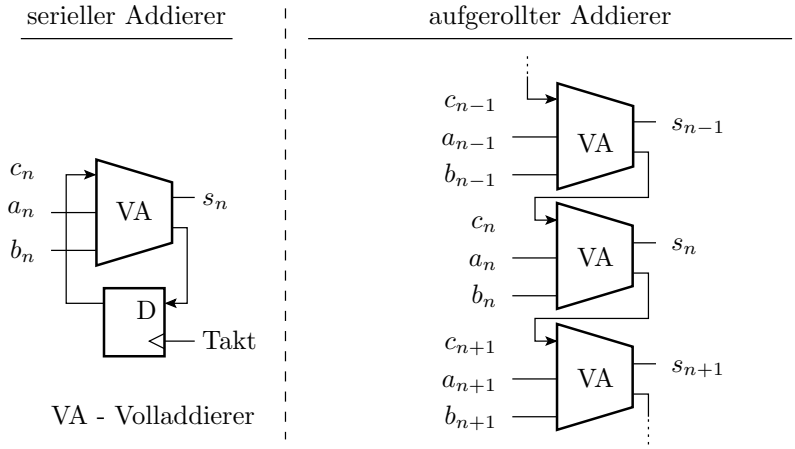
- Testberechnung wie für eine kombinatorische Schaltung.
- Die Verzögerung der Ausgabe gegenüber der Eingabe wird erst bei der Testdurchführung berücksichtigt.

Verarbeitung in mehreren Zeitebenen



- Mehrere Kopien gleicher Schaltungsteile in der pseudo-kombinatorischen Ersatzschaltung.
- Der eingebaute Haftfehler ist in jeder Kopie der Teilschaltung.
- Berechnet wird eine Folge von Testeingaben für mehrere Zeitschritte (Mehr-Pattern-Test).

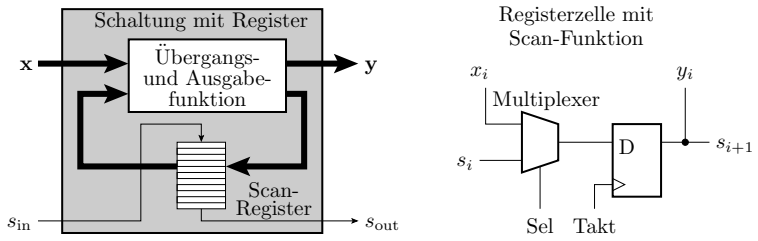
Schaltungen mit Rückführung



- Pseudo-kombinatorischen Ersatzschaltung mit endlos vielen Kopien der Übergangsfunktion.
- Längenbegrenzung der Steuer- und Beobachtungspfade.

Scan-Verfahren

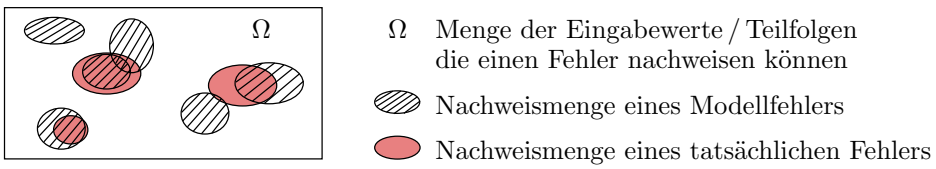
Auftrennung von Rückführungen durch Lese- und Schreibzugriff auf Zustandsspeicher. Scan-Verfahren für digitale Schaltungen:



- Erweiterung der Zustandsregister um eine Schiebefunktion. (Mindestaufwand ein Multiplexer je Speicherzelle.)
- Serielles Auslesen und Neuladen der Zustandsregister zwischen aufeinanderfolgenden Testschritten.
- Testsuche wie für ein System ohne Gedächtnis.

2 Zufallstest

Geziele vs. zufällige Testauswahl

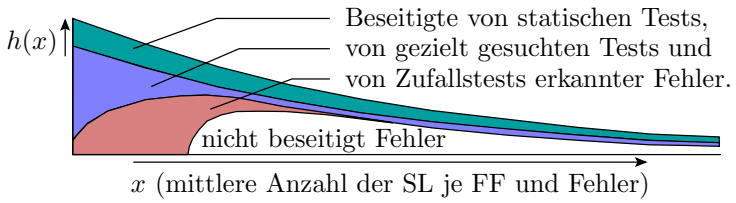


- Ein Fehlermodell erzeugt viele Modellfehler.
- Alle potentiellen Fehler und alle Modellfehler haben Nachweismengen, die sich mehr oder weniger überschneiden.
- Gezielte Testauswahl sucht für jeden Modellfehler nach einem oder mehreren Tests und hofft, mit diesen Tests auch die tatsächlichen Fehler zu finden.
- Ein Zufallstest wählt die Tests, ohne die Nachweismengen der Modellfehler zu bevorzugen. Bei gleicher Fehlerüberdeckung aufwändiger, aber größerer Zuververlässigkeitszuwachs.

2.1 Zuverlässigkeit & Test

Testauswahl und $h(x)$ (Wiederholung TV_F1)

Den Zusammenhang zwischen Fehleranzahl und Zuverlässigkeit beschreibt die QQ-Funktion $h(x)$ (Fehleranzahl in Abhängigkeit von der mittleren Anzahl der SL je FF). Für Systeme im Einsatz wird $h(x)$ maßgeblich durch die Art der Testauswahl für die Fehlerbeseitigungsiteration geformt:



- Statische Tests: Reduzierung von $h(x)$ unabhängig von x um eine schadensunabhängige Nichtbeseitigungswahrscheinlichkeit:
$$p_{\mathrm{NBes}} \neq f(x)$$
- Zufällige Testauswahl: Nichtbeseitigungswahrscheinlichkeit für 1 und n Testschritte:
$$p_{\mathrm{NBest}}(1) = 1 - \frac{p_{\mathrm{Bes}}}{x};\; p_{\mathrm{NBest}}(n) = e^{-\frac{p_{\mathrm{Bes}} \cdot n}{x}}$$
- Bei gezielter Testauswahl werden Fehler, die häufig FF verursachen mehr als bei statischen und weniger als bei Zufallstests bevorzugt.

Verbesserung der fehlerbezogene Teilzuverlässigkeit durch:

- Beseitigung von mit statischen Tests gefundenen Fehlern:
$$Z_{\mathrm{FB}} = \frac{Z_{\mathrm{F0}}}{p_{\mathrm{NBes}}}$$
- Beseitigung zufällig gefundener Fehler (Herleitung folgt):
$$Z_{\mathrm{FB}} = Z_{\mathrm{F}}(n) = Z_{\mathrm{F}}(n_0) \cdot \left(\frac{n}{n_0}\right)^{k+1} \tag{1}$$
- Beseitigung der mit gezielt berechneten Tests gefundenen Fehler:
$$\frac{Z_{\mathrm{F0}}}{p_{\mathrm{NBes}}} < Z_{\mathrm{FB}} < Z_{\mathrm{F}}(n_0) \cdot \left(\frac{n}{n_0}\right)^{k+1}$$

Wiederholung der Herleitung von Gl. 1

Ausgehend von TV_F1, Abschn. 4.1, Gl. 13:

$$\begin{aligned} p_{\mathrm{FFF}}(n) &= \int_0^{\infty} \frac{h(x,n,p_{\mathrm{Bes}})}{x} \cdot dx \\ &\quad \text{mit } h(x,n,p_{\mathrm{Bes}}) = c \cdot x^{-(k+1)} \cdot e^{-\frac{p_{\mathrm{Bes}} \cdot n}{x}} \\ &= \int_0^{\infty} c \cdot x^{-(k+2)} \cdot e^{-\frac{p_{\mathrm{Bes}} \cdot n}{x}} \cdot dx = \frac{c \cdot \Gamma(k+1)}{(p_{\mathrm{Bes}} \cdot n)^{k+1}} \\ p_{\mathrm{FFF}}(n) &= p_{\mathrm{FFF}}(n_0) \cdot \left(\frac{n}{n_0}\right)^{-(k+1)} \\ Z_{\mathrm{F}}(n) &= \frac{1}{p_{\mathrm{FFF}}(n)} = Z_{\mathrm{F}}(n_0) \cdot \left(\frac{n}{n_0}\right)^{k+1} \end{aligned}$$

(p_{Bes} – Beseitigungsw. je Testschritt; $p_{\mathrm{FFF}}(n)$ – Wahrsch. FF durch nicht beseitigte Fehler nach n Tests).

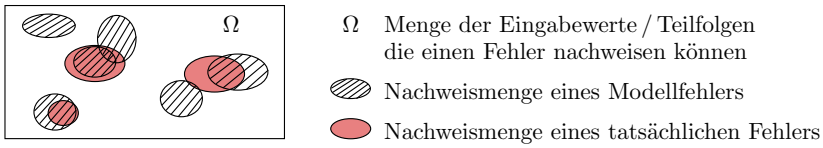
Prüftechnologie und Zufallstest

In einer typischen Prüftechnologie:

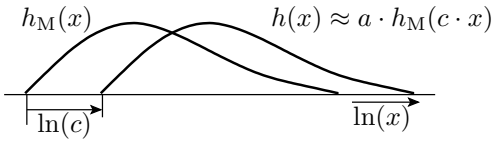
- werden »statisch erkennbare« Fehler, insbesondere solche mit Zerstörungsgefahr vor Inbetriebnahme mit statischen Tests gesucht und beseitigt.
- Dann folgen gezielt ausgewählte Tests, um die Mehrheit der restlichen Fehler zu finden und zu beseitigen.
- Sich anschließende Zufallstests, dazu gehört auch die Systemnutzung mit Fehlerbeseitigungsprozess, zielen auf Zuverlässigkeitsverbesserung durch »rekursive« Beseitigung der schadenstträchtigsten Restfehler.

2.2 Testzeitskalierung

Fehler- und Modellfehlerüberdeckung Zufallstest



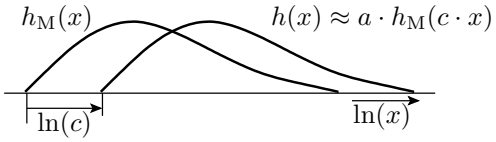
- Ein Fehlermodell erzeugt viele Modellfehler.
- Alle potentiellen Fehler und alle Modellfehler haben Nachweismengen, die sich mehr oder weniger überschneiden.
- Die Nachweiswahrscheinlichkeiten der Fehler unterscheiden sich tendenziell um einen Faktor c von denen der Modellfehler:



$$h\left(x\right) \sim h_{\mathrm{M}}\left(c \cdot x\right)$$

- $c > 1$: Modellfehler schlechter nachweisbar
- $c < 1$: Modellfehler besser nachweisbar.

Testsatzlängenskalierung

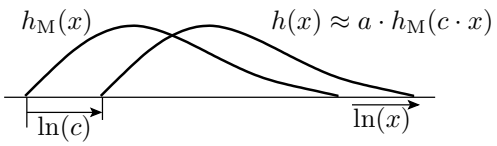


Zu erwartende Modellfehlerüberdeckung:

$$E\left(FC_{\mathrm{M}}\left(n\right)\right) \approx 1 - \frac{\int_0^\infty h_{\mathrm{M}}\left(x\right) \cdot e^{-\frac{n}{x}} \cdot dx}{\int_0^\infty h_{\mathrm{M}}\left(x\right) \cdot dx}$$

Zu erwartende Fehlerüberdeckung:

$$\begin{aligned} E\left(FC\left(n\right)\right) &\approx 1 - \frac{\int_0^\infty h\left(x\right) \cdot e^{-\frac{n}{x}} \cdot dx}{\int_0^\infty h\left(x\right) \cdot dx} \\ &= 1 - \frac{\int_0^\infty a \cdot h_{\mathrm{M}}\left(c \cdot x\right) \cdot e^{-\frac{n}{x}} \cdot dx}{\int_0^\infty a \cdot h_{\mathrm{M}}\left(c \cdot x\right) \cdot dx} \end{aligned}$$



Mit der Substitution $z = c \cdot x$

$$E\left(FC\left(n\right)\right) \approx 1 - \frac{\int_0^\infty a \cdot h_{\mathrm{M}}\left(z\right) \cdot e^{-\frac{c \cdot n}{z}} \cdot dz}{\int_0^\infty a \cdot h_{\mathrm{M}}\left(z\right) \cdot dz}$$

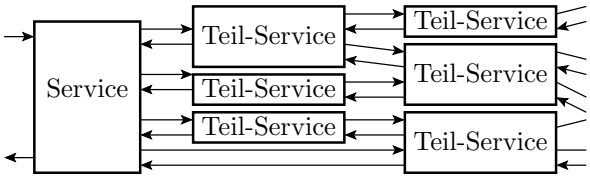
zeigt sich, dass die zu erwartende Fehlerüberdeckung etwa die zu erwartende Modellfehlerüberdeckung der c-fachen Testsatzlänge ist:

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

Tendentiell schlechtere oder besser Nachweisbarkeit der von einem Fehlermodell generierten Modellfehlern lässt sich bei einem Zufallstest durch eine Testlängenskalierung ausgleichen.

2.3 Isolierter Test von Teil-SL

Steuer- und Beobachtbarkeit

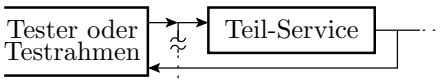


In einem hierarchischen System verursacht ein Fehler in einem Teil-Service nur dann ein Versagen der übergeordneten Service-Leistung, wenn

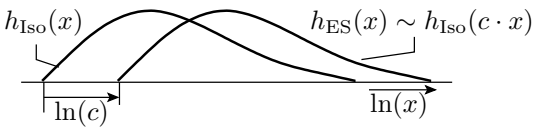
- die übergeordnete Service-Leistung den Teil-Service nutzt,
- der Fehler dabei angeregt wird / steuerbar ist,
- lokal nachweisbar ist und
- die lokale Verfälschung am Gesamtergebnis beobachtbar ist.

Der Nachweis von Fehlern in selten angeregten und/oder beobachtbaren SL verlangt unzumutbar lange Zufallstestsätze.

Isolierten Test von Teil-SL



Der Isolierte Test eines Teilsystems verbessert die Wahrsch. der Steuer- und Beobachtbarkeit um einen Faktor $c \ll 1$. Genau wie bei »besser nachweisbaren Modelfehlern« verschiebt sich $h\left(x\right)$:



Ein isolierter Test der Länge n weist so viele Fehler nach, wie ein $n \cdot c$ langer Test in der Systemumgebung, und verbessert im selben Maße die fehlerbezogene Teilzuverlässigkeit im betrachteten Teilsystem.

2.4 Gewichteter Zufallstest

Wichtung, Steuer- und Beobachtbarkeit

Die Wichtung eines binären Wertes ist die Auftrittswahrscheinlichkeit einer 1. In einem System, in dem logische Werte UND, ODER, ... verknüpft werden, sind die Wahrscheinlichkeiten der

- Steuerbarkeit,
- der Beobachtbarkeit und
- des Fehlernachweises

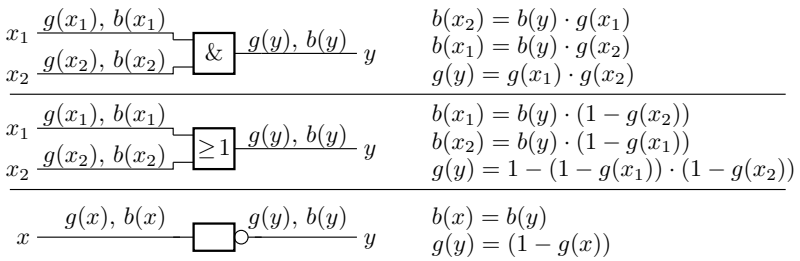
Funktionen der Wichtungen an den Eingängen und damit über Eingabewichtungen einstellbar.

Die Anregungswahrscheinlichkeit eines sa0-Fehlers ist die Wichtung $g(...)$ und die eines sa1-Fehler die Gegenwahrscheinlichkeit $1 - g(...)$ am Fehlerort.

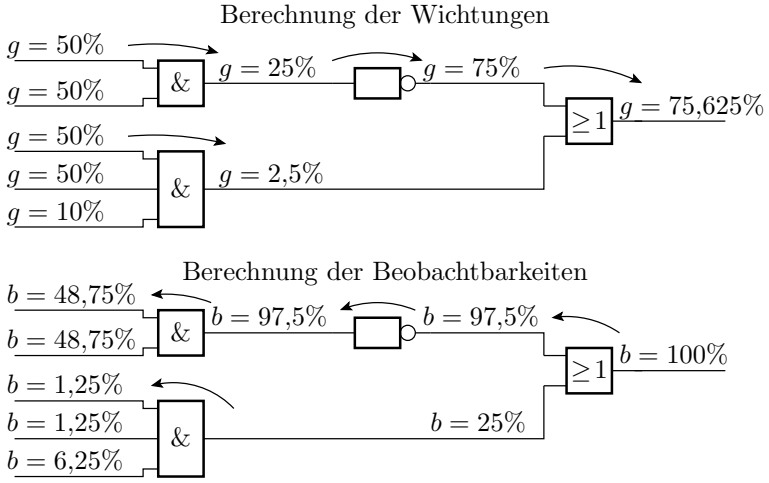
Die Nachweiswahrscheinlichkeit ist das Produkt aus Anregungs- und Beobachtungswahrscheinlichkeit.

Berechnung der Wichtungen und Beobachtbarkeiten

Die Wichtung einer UND-Verknüpfung ist das Produkt der Wichtungen der Operanden. ... Die Eingabe einer UND-Operation ist beobachtbar, wenn die andere Eingabe eins, bei einer ODER-Operation, wenn die andere Eingabe null ist. ...

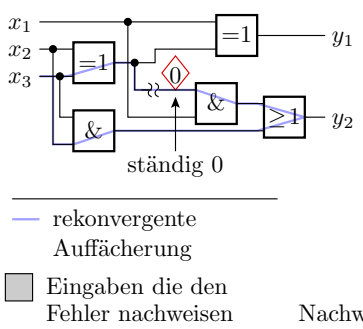


Wichtungen werden in Richtung und Beobachtbarkeiten entgegen der Richtung des Berechnungsflusses bestimmt.



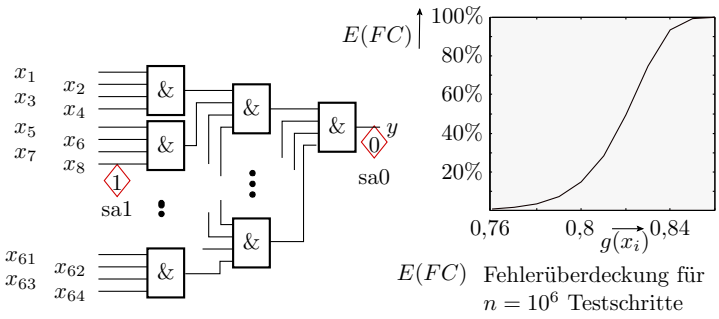
Rekonvergente Auffächerung

Bei rekonvergenter Auffächerung werden gleiche Zufallswerte nach unterschiedlicher Verknüpfung mit anderen Werte logisch verknüpft. Für verknüpfte Werte, die von derselben Zufallsgröße abhängen, gelten die einfachen Regeln auf Seite 15 nicht. Berechnung aus den Auftrittshäufigkeiten der Eingaben für den Fehlernachweis:



Eingabe x_3 x_2 x_1	Ausgabe y_2 y_1	Auftrittshäufigkeit		
0 0 0	0 0	0,125	0,1	0,1
0 0 1	0 1	0,125	0,05	0,1
0 1 0	0 1	0,125	0,15	0,2
0 1 1	1 0	0,125	0,2	0,05
1 0 0	0 1	0,125	0,05	0,2
1 0 1	1 0	0,125	0,2	0,05
1 1 0	1 0	0,125	0,05	0,2
1 1 1	1 1	0,125	0,2	0,1
Nachweiswahrscheinlichkeit:		0,25	0,4	0,1

Fehlerüberdeckung und Wichtung

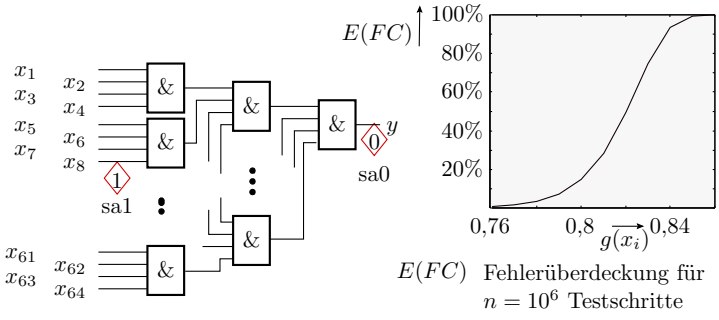


Angenommene Fehler: Für je einen der 64 Eingänge ständig 1:

$$p_{\text{sa1}} = g^{63} \cdot (1-g)$$

Für den Ausgang ständig 0:

$$p_{\text{sa0}} = g^{64}$$



Zu erwartende Fehlerüberdeckung als Mittelwert der Nachweiswahrscheinlichkeit:

$$E\left(FC\right) = \frac{64 \cdot g^{63} \cdot (1-g) + g^{64}}{65}$$

Eine Wichtung von $g = 86\%$ verringert die erforderliche Testsatzlänge für $E\left(FC\right) \gg 99\%$ von $n \gg 2^{64}$ auf $n \approx 10^6$.

Fehlerorientierte Wichtungsauswahl

Statt einheitlicher Wichtung aller Eingabesignale

- Festlegung einer individuellen Wichtung für jeden Eingang,
- Wichtungsumschaltung jeweils nach einem längeren Zufallstest.

Ein pragmatischer Ansatz dazu aus[3]:

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.

Auswahl der Wichtungswerte

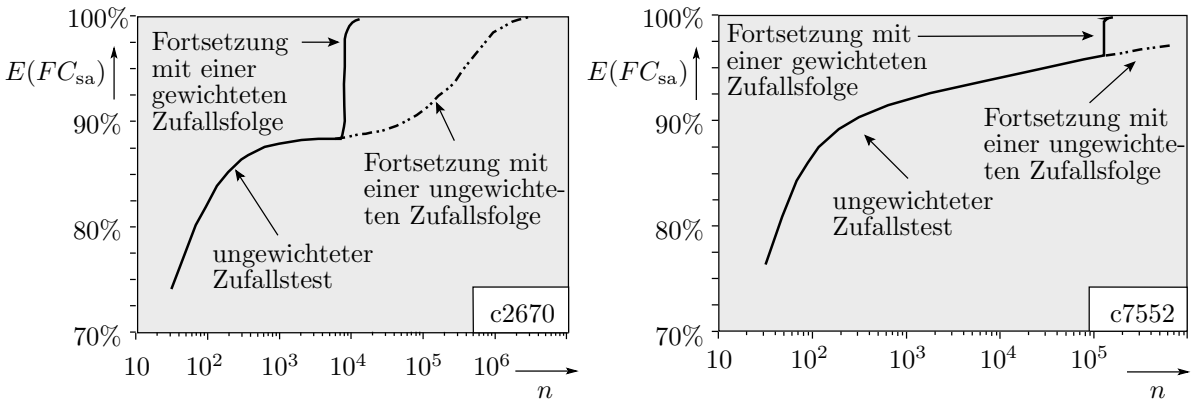
- Für alle Modellfehler, die der ungewichtete Zufallstest nicht nachweist, gezielte Berechnung einer Eingabe (D-Algorithmus) mit möglichst vielen »X« (Don't Care) Eingabewerten.
- Begrenzung der Wichtung auf 5 Werte:

$$g(x_i) = \begin{cases} 0 & x_i \in \{0, \mathrm{X}\} \\ 2^{-N_{\mathrm{EAND}}} & N_0 \gg N_1 \\ 0{,}5 & \text{sonst} \\ 1 - 2^{-N_{\mathrm{EAND}}} & N_0 \ll N_1 \\ 1 & x_i \in \{1, \mathrm{X}\} \end{cases}$$

  (N_0 – Anzahl der Nullen; N_1 – Anzahl der Einsen für Eingang x_i in den Testeingaben; $N_{\mathrm{EAND}} \in \{2, 3, 4, \ldots\}$ Anzahl der [N]AND-verknüpften Zufallsfolgen zur Wichtungserzeugung).
- Die zweite und weitere Wichtungsberechnungen berücksichtigen nur noch die Testeingaben bis dahin nicht nachgewieser Fehler.

Experiment mit den Schaltungen c2670 und c7552[4]

- 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.



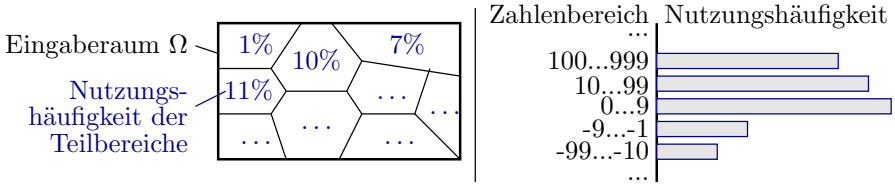
[3] J. Hartmann, G. Kemnitz: How to do weighted random testing for BIST? ICCAD 1993.

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

2.5 Operationsprofil

Operationsprofil

Der Eingaberaum für einen Service ist in der Regel in Teilbereiche unterteilt, die unterschiedlich häufig genutzt werden. Beispielsweise werden kleine Zahlenwerte häufiger als große und positive häufiger als negative genutzt.



Bei einem menügesteuerten Programm werden die einzelnen Menüeinträge unterschiedlich oft ausgewählt.

Operation	editieren	löschen	browse	drucken
Nutzungshäufigkeit	35%	12%	46%	7%

Fehlerorientierte Auswahl von Operationsprofilen

Die Nachweiswahrscheinlichkeiten für Fehler in den SL für die einzelnen Operationen hängen von der Nutzungshäufigkeit der Operationen ab. Pragmatischer Ansatz zur fehlerorientierten Auswahl von Operationsprofilen für den Test (Analog zum gewichteten Zufallstest mit Wichtungsumschaltung):

1. Festlegung einer größeren Menge von Modellfehlern.
2. Längerer ungewichteter Zufallstest und Abhaken aller damit nachweisbaren Modellfehler.
3. Suche für die restlichen Modellfehler ein Operationsprofil, dass deren Nachweiswahrscheinlichkeiten erheblich erhöht.
4. Längerer Zufallsttest mit diesem Operationsprofil und Abhaken aller damit nachweisbaren Modellfehler.
5. Wenn erforderlich, Wiederholung von Schritt 3 und 4.

Testfallbeschreibung durch Operationsprofile

- Ein Testfall beschreibt u.a., welche Zielfunktion er überprüft (vergl. TV_F3, Abschn. 5.4 »Schadenskosten«).
- Bei Überprüfung nur einer Beispieleingabe für eine Funktion ist die Wahrscheinlichkeit, dass dadurch zufällig ein vorhandener Fehler erkannt wird, meist nicht sehr groß.
- Besser für die Verlässlichkeit ist die Überprüfung zuzusichernder Eigenschaften mit einer größeren Eingabestichprobe.
- Statt einzelner Eingaben, Festlegung eines Operationsprofils für den Testfall und Test mit einer größeren Anzahl zufälliger Eingaben mit dem Operationsprofil.

Opertionsprofil »uneingearbeiteten Nutzer«:

- Uneingearbeitete Nutzer erleben andere FF als Profi-Nutzer.
- Die wahrgenommene Zuverlässigkeit dieser Nutzergruppe ist ausschlaggebend für die Akzeptanz eines neuen Produkts.

2.6 Pseudo-Zufallsgeneratoren

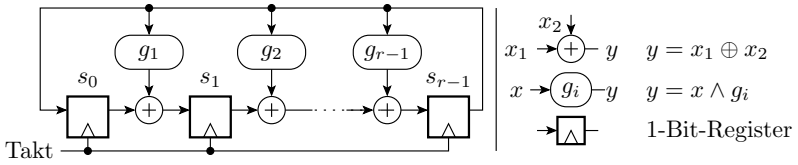
Erzeugung von Pseudo-Zufallszahlen

Pseudo-Zufallsfolgen sind wiederholbare Bitfolgen, mit denen als Testeingaben Fehler dieselben Nachweiswahrscheinlichkeiten wie mit echte Zufallsfolgen haben. Erzeugung mit autonomen linearen Automaten mit langen Zykluslängen. Beispiele für geeignete Übergangsfunktionen zyklischer linearer autonomer Automaten:

- fortgesetzte Multiplikation mit einer geeigneten Zahl. Abschneiden der führenden Stellen. ...
- fortgesetzte Bildung des Divisionsrests mit einer geeigneten Zahl[5] und Weiterverwendung des Divisionsrests.
- Linear rückgekoppelte Schieberegister[6],
- ...

Linear rückgekoppelte Schieberegister

Ein linear rückgekoppelte Schieberegister (LFSR **L**inear **F**eeback **S**hift **R**egister) in einer ersten Ausführung verschiebt seinen r-Bit-Zustand $\mathbf{s} = (s_{r-1}, s_{r-2}, \ldots, s_0)$ um eine Stelle nach links und addiert, wenn das herausgeschobene Bit s_{r-1} gleich »1« ist, eine Bitvektorkonstante $\mathbf{g} = (g_{\mathrm{r}-1}, g_{r-1}, \ldots, g_1, 1)$ zum Zustand $\mathbf{s}$:



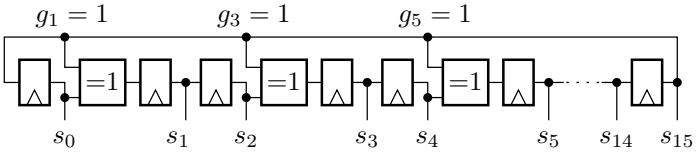
Für jede Bitanzahl r des Zustandsvektors gibt es Konstanten $\mathbf{g}$, sog. »primitive Polynome«, bei denen alle Zustände außer 000...0 ineinander übergehen. Nur solche Konstanten $\mathbf{g}$ werden verwendet.

Primitiven Polynome und die Konstante g

Mit dem Internet-Suchbegriff »Primitive Polynome« findet man z.B. für 16-Bit LFSR:

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

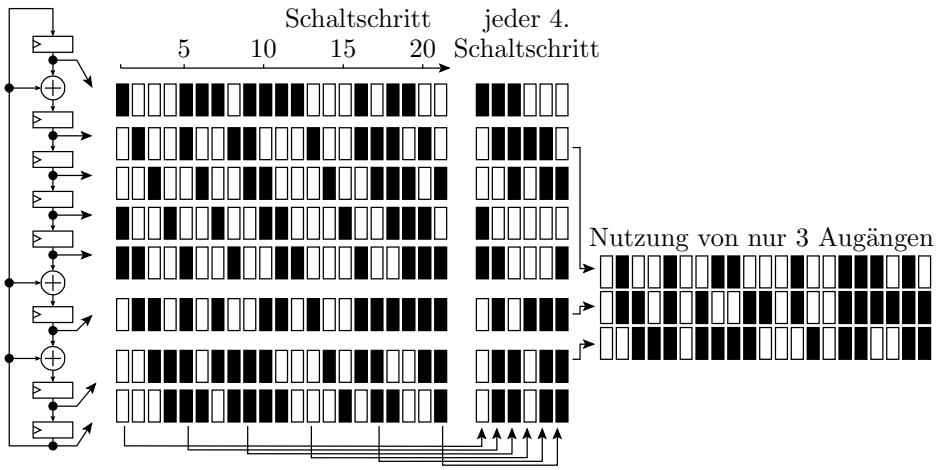
Das bedeutet $g_1 = g_3 = g_5 = 1$ und alle anderen $g_i|_{i \notin \{1,3,5\}} = 0$. In Realisierung als Digitalschaltung für $g_i = 1$ EXOR-Gatter einfügen und für $g_i = 0$ EXOR-Gatter weglassen.



Pseudo-Zufallsfolge eines 8-Bit-LFSR

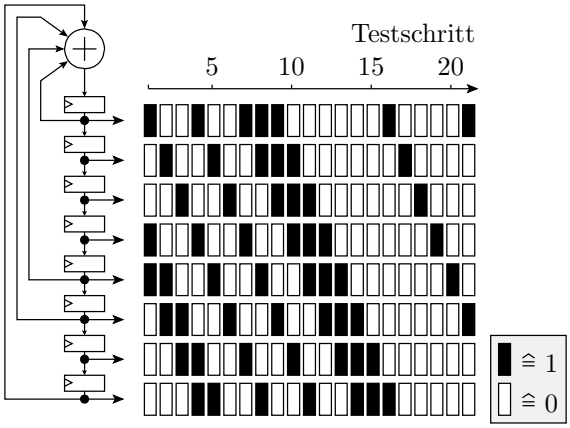
[5] Bevorzugt werden große Primzahlen.

[6] Zusammensetzung der Übergangsfunktion aus Verschiebeoperationen und bitweisem AND und bitweisem EXOR. Einfachste und für Testeingaben ausreichende Lösung.



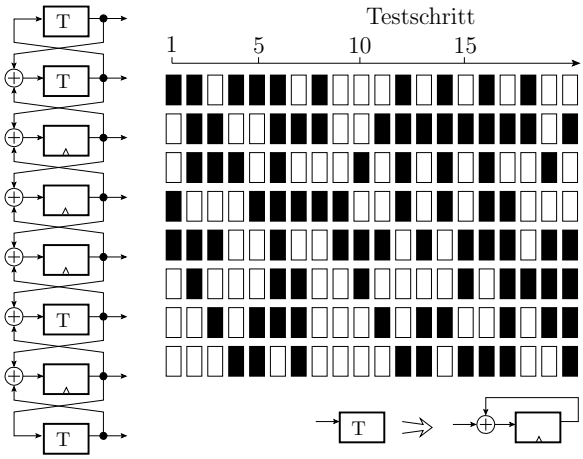
Falls die »Streifenmuster« durch die Schiebeoperationen stören, nur einen Teil der Ausgaben nutzen.

Bei »Umkehrung« der Signalflussrichtung wird aus den verteilten EXOR-Gattern ein zentrales EXOR-Netzwerk am Eingang.



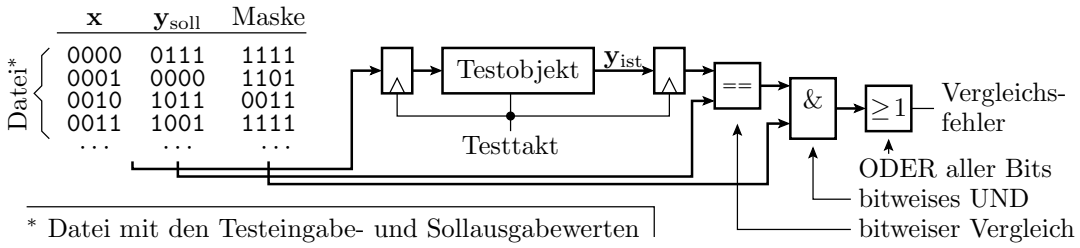
Gleiche Zyklusstruktur bei gleichen Rückführstellen. Bitfolgen mit Phasenverschiebung größer 1 auch durch EXOR mehrerer Bitströme.

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:



2.7 Selbsttest

Prinzip eines Digitaltesters



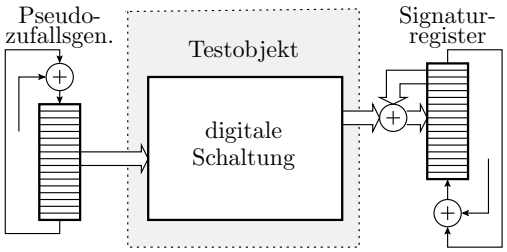
Vorm Digitaltester zum Selbsttest:

- Testeingabebereitstellung durch einen Pseudo-Zufallsgenerator, realisierbar als LFSR.
- Ersatz des Soll-/Ist-Vergleichs durch Prüfkennzeichenbildung, gleichfalls realisierbar als LFSR.

Zusatzforderungen an das Testobjekt (prüfgerechter Entwurf):

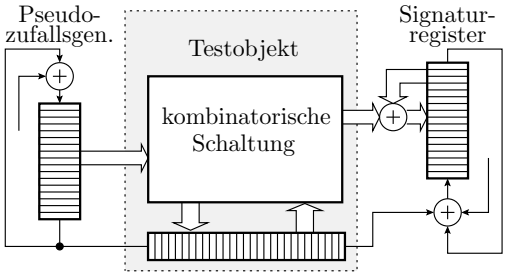
- mit einem Zufallstest ausreichend testbar,
- keine ungültigen auszumaskierenden Ergebnisbits, ...

Selbsttest (BIST Built-in Self Test)



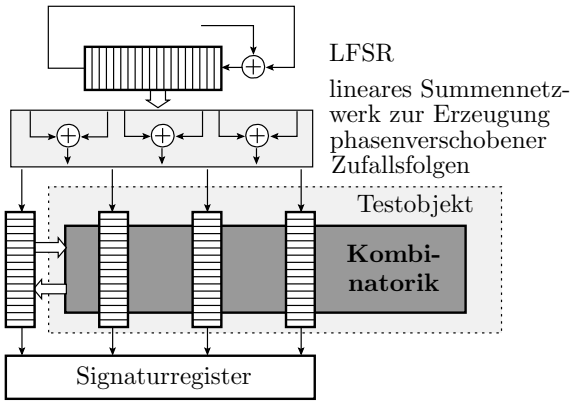
- Einrahmen der Schaltung mit Schieberegistern und Ergänzung einiger EXOR-Gatter.
- Wenn als Schieberegister vorhandene Ein- und Ausgaberegister verwendet werden, besonders niedriger Zusatzaufwand.
- Test mit voller Schaltungsgeschwindigkeit von Millonen bis Milliarden Test pro Sekunde.

BIST plus Scan



- Zwischen den Testschritten Zustandsregister seriell in das Signaturregister auslesen und neu beschreiben.
- Der isolierte Test der Übergangsfunktion reduziert in die Regel die erforderliche Testzeit viel mehr, als sie sich durch die zusätzlichen Schiebeschritte erhöht.

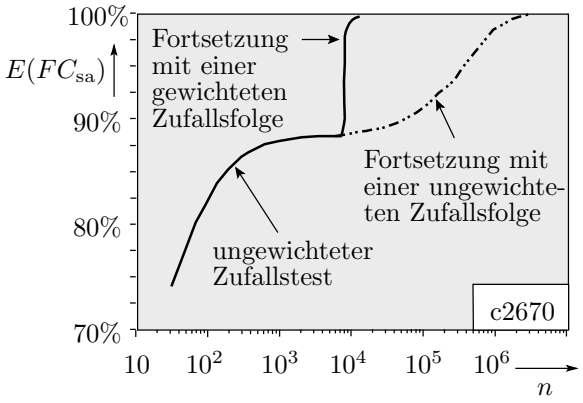
Für sehr große Systeme, z.B. Multi-Chip-Module mit mehreren Scan-Registern, die zwischen den Testschritten parallel gelesen und mit neuen Zufallswerten beschrieben werden.

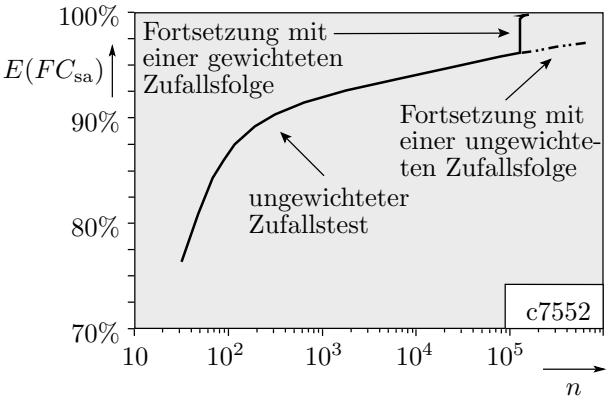


Weiterführende Literatur [G. Kemnitz: Test und Verlässlichkeit von Rechnern. Springer 2007.]

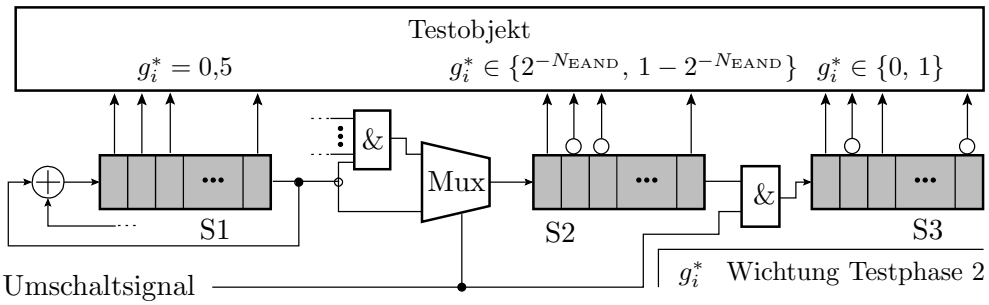
BIST mit Wichtungsumschaltung
Auf Seite 17 wurde eine Zufallstest mit umschaltbarer Wichtung für die Benchmark-Schaltungen c2670 und c7552 besprochen:

- Ein ungewichteten Zufallstest der Länge 10^4 bzw. 10^5 weist 90% bzw. 95% der Haftfehler nach.
- Danach Umschaltung auf eine aus berechneten Testeingaben für die restlichen Fehler abgeleiteten Wichtung zum Nachweis der restlichen Modellfehler.





Implementierung als Selbsttest



- Testphase 1: Erzeugung ungewichteter Pseudo-Zufallseingaben mit LFSR S1. Serielle Weitergabe an die Schiebereg. S2 und S3.
- Testphase 2: Verringerung der Wichtung in S2 durch UND-Verknüpfung von N_{EAND} Ausgabefolgen von S1 und für S2 durch »UND 0«. Erzeugung der Wichtungen $1-2^{-N_{\mathrm{EAND}}}$ und »1« durch Inverierung.

Nicht nennenswert aufwändiger als ohne Wichtung.

3 Schaltkreise

Test digitaler Schaltkreise

Besonderheiten:

- Tausende bis Millionen von Logikfunktionen je Testobjekt.
- Sehr eingeschränkte Beobachtbarkeit interner Signale.
- Hohe Anforderungen an Ausbeute und Test. Überschlag:
  - Ausbeute $Y \approx 50\%$,
  - angestrebter Fehleranteil nach dem Test $DL \approx 100\,\text{dpm}$,
  - zu fordernde Fehlerüberdeckung $FC \approx 99{,}9\%$.

Für digitale Schaltungen sind

- fehlerorientierte Testauswahl, Haftfehler,
- Zufallstests mit sehr vielen Testbeispielen und
- automatische Testsatzberechnung

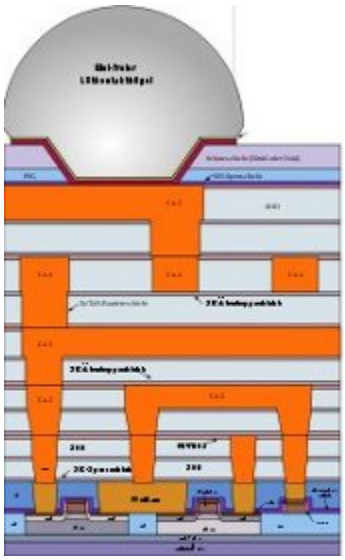
Seit Jahrzehnten Stand der Technik.

3.1 Fertigungsfehler und Haftfehler

Fertigungsfehler integrierter Schaltkreise

Schaltkreise werden schichtweise hergestellt:

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

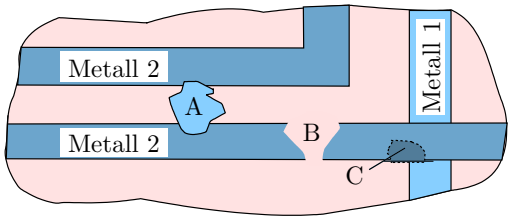


Typische Herstellungsfehler:

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

Leitungs- oder Isolationsmaterial in einer Schicht.

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



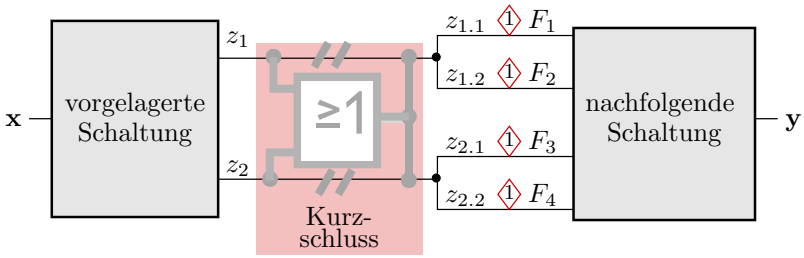
A zuätzliches Metall
B fehlendes Metall
C fehlende Isolation

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

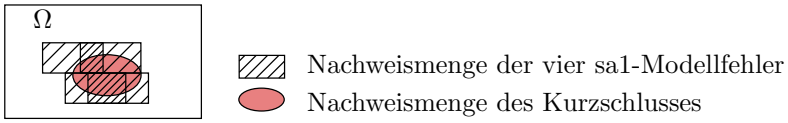
Notwendige Nachweisbedingungen:

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

Kurzschlussnachweis mit einem Haftfehlertestsatz



- Für den Beispielkurzschluss verfälscht eine »1« auf der einen Leitung eine »0« auf der anderen Leitung.
- Ein »0→1«-Verfälschung teilt sich mit jedem der zwei sa1-Fehler auf den Leitungszweigen Nachweisbedingungen.
- 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 für $g(z_1) = g(z_2) = g = 50\%$:

- Gezielte Suche von einem Test für F_1 bis F_4: Im Mittel $4 \cdot FC_{\text{sa}}$ Haftfehlertests weisen den Kurzschluss mit $g = 50\%$ nach:

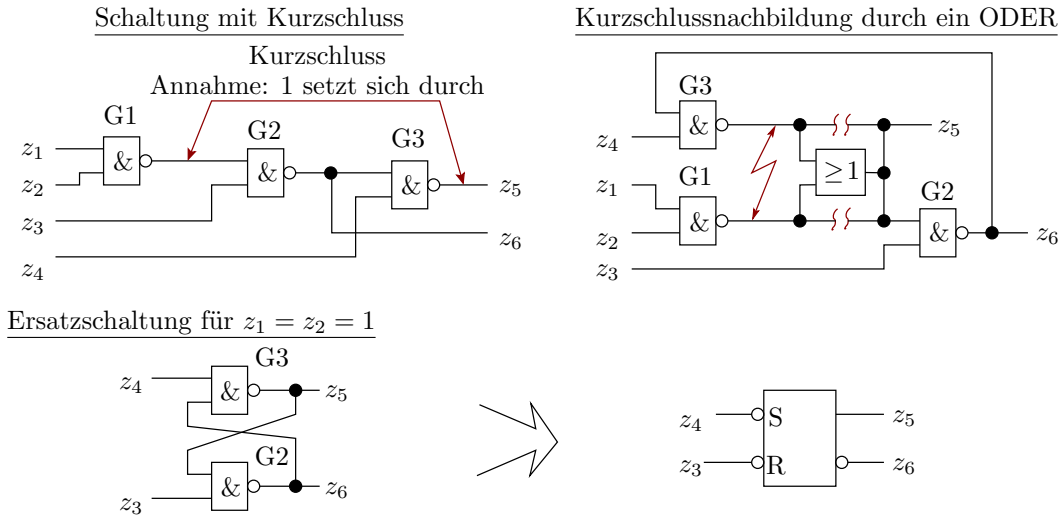
$$p_{\text{Kurzschl}} \approx 1 - 0{,}5^{4 \cdot FC_{\text{sa}}} \leq 94\%$$

- Zufällige Testauswahl: Tendenziell Nachweismenge doppelt so große wie die der vier ähnlich nachweisbaren Haftfehler:

$$\begin{aligned} h_{\text{Kurzschl}}(x) &\sim h_{\text{sa}}\left(\frac{x}{2}\right) \\ FC_{\text{Kurzschl}}(n) &\approx FC_{\text{sa}}\left(\frac{n}{2}\right) \end{aligned}$$

.

Kurzschlüsse können Gatterschaltungen zu Speicherzellen umbilden.

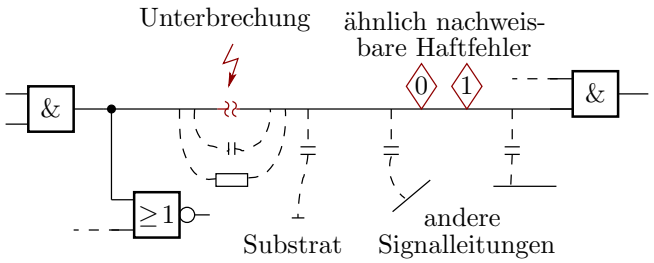


Für Fehler mit Speicherverhalten Fehlersimulation und Testberechung nach pseudo-kombinatorischem Iterationsmodell (siehe Seite 10).

Verallgemeinerung

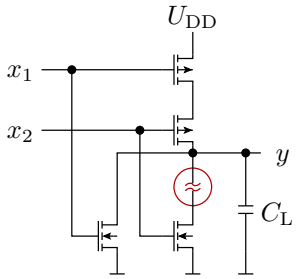
- Zunahme der Anzahl möglicher Kurzschlüsse im ungünstigsten Fall mit dem Quadrat der Leitungsanzahl. Haftfehler nur linear.
- Kurzschlüsse können sehr vielfältige Fehlerwirkungen haben (»0« setzt sich durch, ..., Speicherwirkung, nicht deterministisches Fehlverhalten. Berücksichtigung aller Möglichkeiten weder möglich noch notwendig.
- Gezielt berechnete Haftfehlertestsätze mit hoher Fehlerüberdeckung erkennen die meisten Kurzschlüsse. Die Kurzschlussüberdeckung hängt dabei weniger von der Haftfehlerüberdeckung und mehr von der Anzahl der Tests, die je Haftfehler gesucht werden, ab.
- Bei zufälliger Testauswahl ist die zu erwartende Kurzschlussüberdeckung die Haftfehlerüberdeckung für einen kürzeren Testsatz.

Fehlerwirkung von Unterbrechungen



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

Stuck-open-Fehler



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

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

Unterbrechung innerhalb eines Gatters, so dass der Gatterausgang für bestimmte Eingaben nicht auf- bzw. entladen wird.

- Überwiegender Nachweis mit Haftfehlertests.
- Sicherer Nachweis durch Kontrolle der Signalverzögerung.

Verallgemeinerung

Für Unterbrechungen gilt ähnliches wie für Kurzschlüsse:

- Für Haftfehler gesuchte Tests haben in der Regel eine bessere tatsächliche Fehlerüberdeckung, die erheblich von der Anzahl der gesuchten Tests je Haftfehler abhängt.
- Für Zufallstests ist die tatsächliche Fehlerüberdeckung die Haftfehlerüberdeckung eines längeren Testsatzes.

Alternativen/Ergänzungen zur Kontrolle des logischen Verhaltens:

- Kontrolle der Signalverzögerungen und
- Kontrolle der Stromaufnahme, insbesondere der Ruhestromaufnahme, die bei CMOS-Schaltungen sehr geringe Sollwerte hat während der Tests.

Validierung mit Zahlen aus der Literatur

Der typische Schaltkreistest hat eine Überdeckung für Haft- oder Verzögerungsfehler 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.

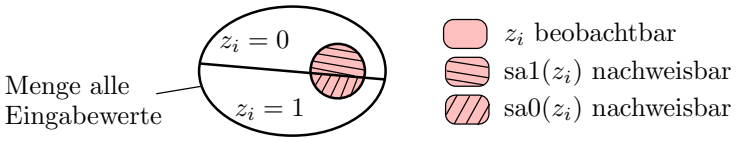
3.2 Andere Fehlermodelle

Weitere für Schaltkreise diskutierte Fehlermodelle

- Toggle Test: Der Testsatz muss jede Leitung mindestens einmal auf null und einmal auf eins steuern.
- Zellenfehlermodell: Teilschaltung mit 1-Bit-Ausgabe funktioniert genau mit einer Eingabe 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.

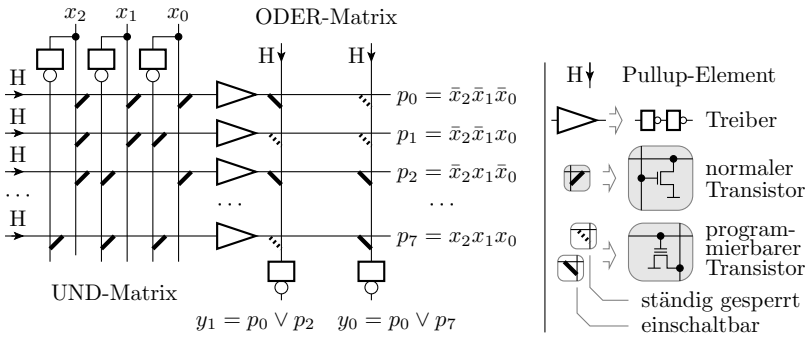
Toggle-Test

Auswahlkriterium Toggle Test: Jedes logische Signal bei Testabarbeitung mindestens einmal »0« und einmal »1«.



- Garantiert Steuerbarkeit für alle Haftfehler. Gleichzeitige Beobachtbarkeit ist Zufall ($h(x) \sim h_{\text{Toggle}}(c \cdot x)$ 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 Software ist ein vergleichbares Maß, die »Anweisungsüberdeckung« noch gebräuchlich (siehe später).

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:

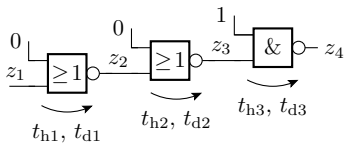
$$\varphi_{\text{M}} = N_{\text{A}} \cdot 2^{N_{\text{E}}}$$

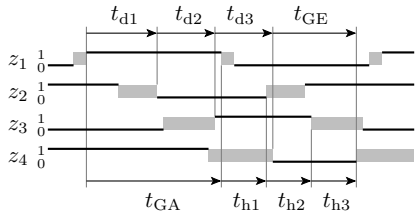
(N_{E} – Anzahl der Eingänge; N_{A} – Anzahl der Ausgänge). Vergleich Zellenfehler und Haftfehler für freistrukturierte Schaltungen (Realisierung aus Gattern statt durch programmierte Speicher):

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

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

Gatterverzögerungsfehler





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

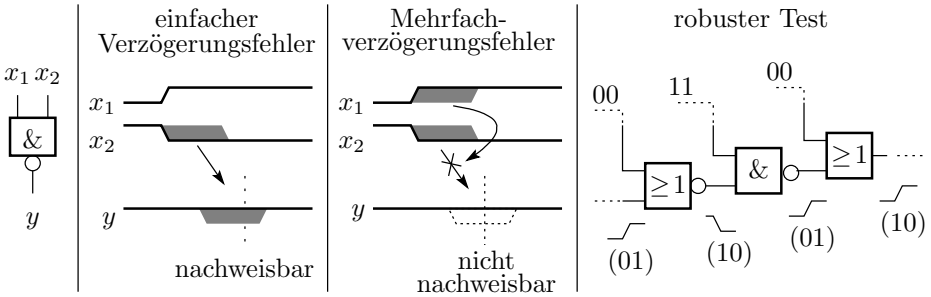
- Modellfehler: slow-to- rise-/ slow-to-fall-Fehler.
- 2-Pattern-Test: Initialisierungseingabe + Haftfehlertest.
- Unter Annahme $g = 50\%$:

$$h_{\mathrm{GVerz}}(x) \sim h_{\mathrm{sa}}\left(\frac{x}{2}\right)$$
$$FC_{\mathrm{GVerz}}(n) \approx FC_{\mathrm{sa}}\left(\frac{n}{2}\right)$$

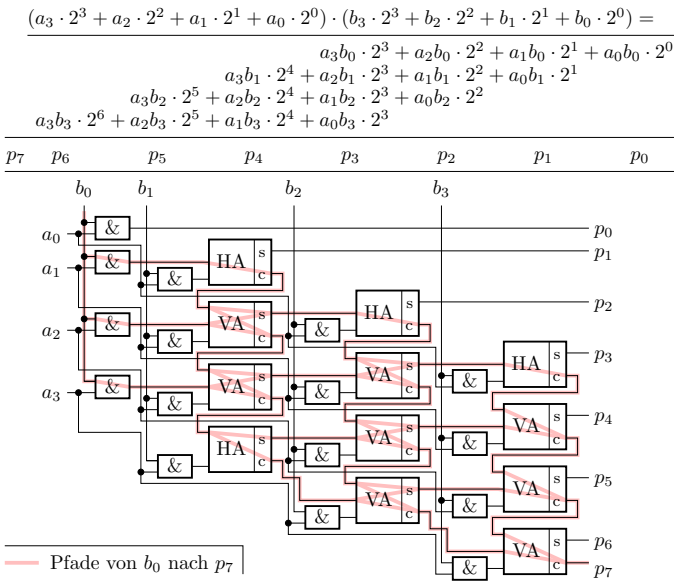
- Gezielte Auswahl: mindestens 2 Tests je Haftfehler suchen.

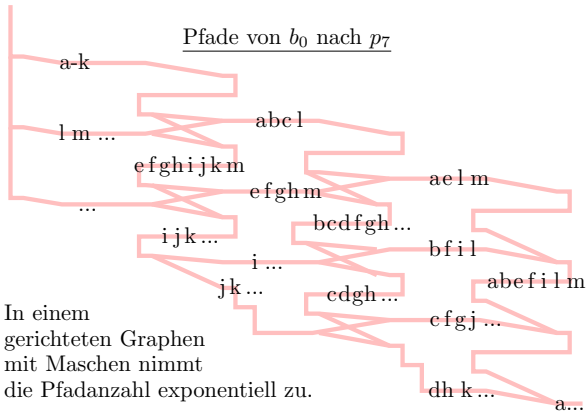
Pfadverzögerungsfehler

- Modellfehler: Für alle Pfade durch die Schaltung
  - slow-to-rise-Fehler (erhöhte 01-Verzögerung),
  - slow-to-fall-Fehler (erhöhte 10-Verzögerung).
- Robuster Test: Je Testschritt max. eine Signaländerung an den Eingängen jedes Gatters.



Exponent. Wachstum Modellfehleranzahl (Bsp. Multiplizierer)

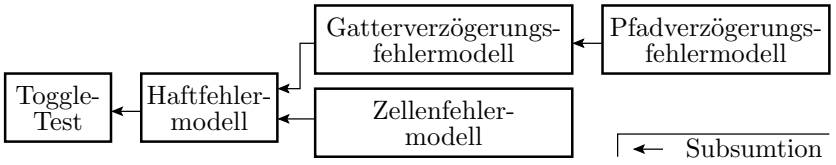




- Die Anzahl der Pfadverzögerungsfehler wächst exponentiell mit der Schaltungsgröße. Nicht praxistauglich.
- Bei mehreren Pfaden je Gatter reicht Kontrolle des Pfades mit der geringsten Laufzeitreserve (Gatterverzögerungsmodell mit Zusatzbedingung).

Subsumtionshierarchie für Fehlermodelle

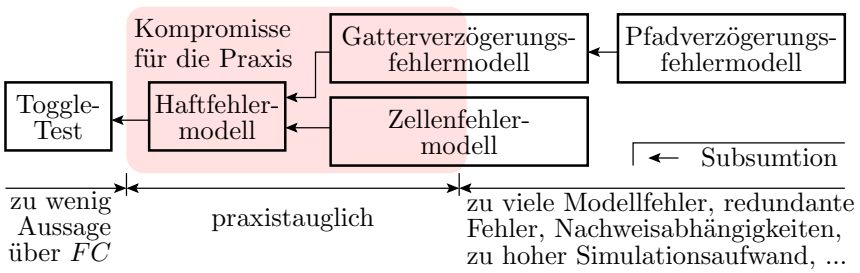
- Subsumption bedeutet in den Beschreibungslogiken, dass ein Konzept (eine eindeutig beschriebene Menge von Objekten) eine Teilmenge eines anderen Konzepts ist.
- Fehlermodell A subsumiert Fehlermodell B, wenn ein Testsatz, der alle Modellfehler von A nachweist, garantiert auch alle Modellfehler von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



3.3 Speichertest

Speichertest

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

- Haftfehler (Lesewert ist ständig null oder ständig eins).
- Übergangsfehler (Wert nur in einer Richtung änderbar).
- Stuck-open-Fehler (Ausgabe des zuletzt gelesenen Wertes).
- Zerstörendes Lesen (Löschen des Inhalts beim Lesen).
- Gegenseitige Beeinflussung unterschiedlicher Zellen, ...

Aufsetzend auf den Fehlerannahmen gibt es algorithmisch beschriebene Testabläufe mit der Speicherorganisation als Parameter, die alle unterstellten 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	$W(i)1$, $R(i)1$ $W(i)0$, $R(i)1$
	Übergangsfehler	Wert der Speicherzelle i ist nur in einer Richtung änderbar	kein Übergang $1 \rightarrow 0$ $0 \rightarrow 1$	 $W(i)1$, $R(i)1$, $W(i)0$, $R(i)0$ $W(i)0$, $R(i)0$, $W(i)1$, $R(i)1$
	Stuck-open-Fehler	kein Zugriff auf Speicherzelle i (Ausgabe des Wertes der vorherigen Leseoperation)		$W(i)0$, $R_1(j)$, $R(i)0$, $W(i)1$, $R_0(j)$, $R(i)1$
	zerstörendes Lesen	Inhalt von Speicherzelle i wird beim Lesen verändert	$R(i) \Rightarrow C(i) = \overline{C(i)}$	$W(i)0$, $R(i)0$, $R(i)0$ $W(i)1$, $R(i)1$, $R(i)1$
2	Kopplung Typ 1	Veränderung des Inhalts von Zelle i bestimmt Zustand in Zelle j	$W(i)0 \Rightarrow C(j) = 0$ $W(i)0 \Rightarrow C(j) = 1$ $W(i)1 \Rightarrow C(j) = 0$ $W(i)1 \Rightarrow C(j) = 1$	$W(j)0$, $W(i)0$, $R(j)0$, $W(i)1$, $R(j)0$ $W(j)1$, $W(i)0$, $R(j)1$, $W(i)1$, $R(j)1$
	Kopplung Typ 2	Veränderung des Inhalts von Zelle i bewirkt eine Änderung in Zelle j	$C(i) = \overline{C(i)} \Rightarrow$ $C(j) = \overline{C(j)}$	$W(j)0$, $W(i)0$, $R(j)0$, $W(i)1$, $R(j)0$, $W(i)0$, $R(j)0$ $W(j)1$, $W(i)0$, $R(j)1$, $W(i)1$, $R(j)1$, $W(i)0$, $R(j)1$

$W(i)0$ Schreibe in Zelle i eine 0
$W(i)1$ Schreibe in Zelle i eine 1
$R(j)$ Lese eine beliebige andere Zelle
$C(\ldots)$ Inhalt Zelle ...

$R(i)0$ Lese Inhalt Zelle i und vergleiche mit Sollwert 0
$R(i)1$ Lese Inhalt Zelle i und vergleiche mit Sollwert 1
$R_0(j)$ Lese eine andere Zelle, in der 0 steht
$R_1(j)$ Lese eine andere Zelle, in der 1 steht

Beispielalgorithmus Marching Test

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

$W(i)0$ Schreibe in Zelle i eine 0 $R(i)0$ Lese Zelle i und vergleiche mit 0
$W(i)1$ Schreibe in Zelle i eine 1 $R(i)1$ Lese Zelle i und vergleiche mit 1

Lässt sich auch ohne großen Zusatzaufwand als Selbsttest implementieren.

4 Software

4.1 Besonderheiten der Testauswahl

Besonderheiten von Software für die Testauswahl

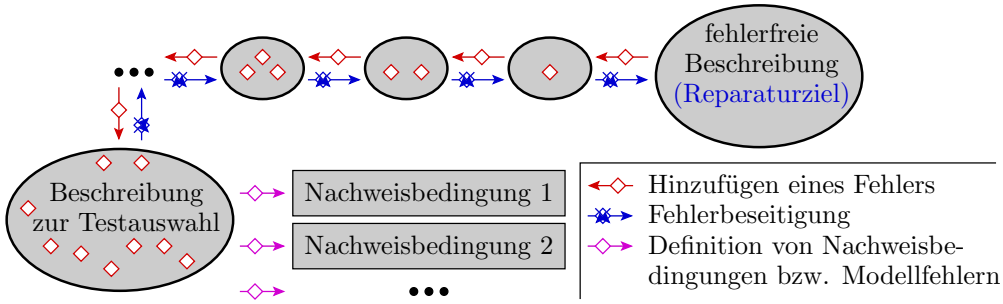
- Fehler entstehen im gesamten Entwurfsprozess (Spezifikation, Architekturentwurf, Programmierung) und auch bei der Fehlerbeseitigung.
- Es existiert in der Regel keine fehlerfreie Sollbeschreibung.
- Eine geziele Kontrolle und Veränderung beliebiger Zwischenergebnisse zur Fehlereingrenzung oder der Untersuchung einer hypothetischen Fehlerwirkung (außer bei Echtzeittests) unproblematisch.
- Agiler Entwurf: Fortlaufende Weiterentwicklung, Fehlerbeseitigung und Fehlerentstehung während der Nutzung.

Zu erwartende Fehler

- Anforderungsfehler: vergessene, überflüssige, falsch formulierte, nicht umsetzbare Anforderungen.
- Fehler bei der Architekturfestlegung: fehlende, überflüssige, falsch umgesetzte Anforderungen, Fehler im Algorithmus, unberücksichtigte selbstverständliche Anforderungen, z.B., dass ein Programm installierbar und bedienbar sein muss, nicht prüfgerecht, ...
- Codierung: fehlende, falsche überflüssige Anweisungen, Fallunterscheidungen, Ausnahmebehandlungen, ...

Es gibt keine korrekte Beschreibung zur Zusammstellung einer Menge potentieller bzw. Modellfehler.

Mutationen statt Modellfehler



- Testauswahl für eine fehlerhafte Entwurfsbeschreibung.
- Statt der Modellfehler lassen sich nur Mutationen einer *potentiell fehlerhaften Beschreibung* konstruieren.
- Für vergessene Aspekt lassen sich keine ähnlich nachweisbaren Mutationen ableiten.
- Gleichfalls nicht erzeugbar sind simulierbare Mutationen für Nicht-Code-Beschreibungen (Anforderungslisten, ...).

Mutationen für Programme

Mutationen auf der Hochsprachenebene sind geringfügige Verfälschungen im Programmtext:

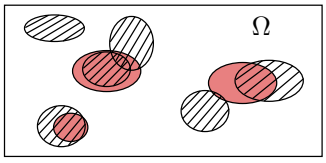
- Verfälschung arithmetischer Ausdrücke (x=a+b ⇒ x=a*b)
- Verfälschung boolescher Ausdrücke (if(a>b){} ⇒ if(a<b){})
- Verfälschung der Wertezuweisung (value=5 ⇒ value=50)
- Verfälschung der Adresszuweisung (ref=obj1 ⇒ ref=obj2)
- Entfernen von Schlüsselworten (static int x=5 ⇒ int x=5)

Bestimmung der Modellfehler- (Mutations-) überdeckung:

- Wiederhole für jede Fehlerannahme:
  - Erzeuge mutiertes Programm und übersetze.
  - Teste, bis zur ersten erkennbaren Ausgabeabweichung zwischen Mutation und Original oder bis Testsatz abgearbeitet.

Kostet viel Rechenzeit, ist aber prinzipiell durchführbar.

Gezielte Testauswahl



Ω Menge der Eingabewerte / Teilfolgen die einen Fehler nachweisen können

Nachweismenge eines Modellfehlers

Nachweismenge eines tatsächlichen Fehlers

Gezielte Testauswahl sucht für jede Mutation mindestens eine Eingabe aus deren Nachweismenge. Die Wahrscheinlichkeit für den Fehlernachweis hängt von den Überschneidungen der Nachweismengen ab.

Fehlende Anforderungen, Ausnahmebehandlungen, ... teilen sich keine Nachweisbedingungen mit mutierten Anweisungen und bleiben so bei der Testauwahl unberücksichtigt.

Für potentielle Fehler vom Typ »fehlt in der fehlerhaften Beschreibung« ist keine gezielte Testauswahl möglich.

Zufällige Testauswahl

- Alle potentiellen Fehler und alle Mutationen haben Nachweismengen, die sich mehr oder weniger überschneiden.
- Trotz des Fehlens ähnlich nachweisbarer Mutationen ist eine vom Fehlermodell abhängige Testzeitskalierung zu erwarten (vergl. Seite 13):

$$\begin{aligned} h(x) &\sim h_{\mathrm{Mut}}(c \cdot x) \\ E(FC(n)) &\approx E(FC_{\mathrm{Mut}}(c \cdot n)) \end{aligned}$$

Zufallstest ist auch für den potentiellen Fehlertyp »fehlt in der fehlerhaften Beschreibung« geeignet.

Software und Hardware-Entwürfen sollten immer einem ausreichend langen Zufallstest unterzogen werden.

Fehlende Sollfunktion

Das Sollergebnis für einen Test ergibt sich aus der Zielfunktion, nicht aus der Umsetzung.

Erfordert eine diversitäre Sollwertberechnung. Maskierung durch übereinstimmende Fehlfunktionen schwer ausschließbar.

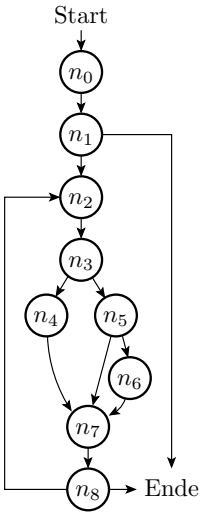
Konzeptionelle Ansätze:

- Erstellen der Testfälle und Sollwerte vor dem Entwurf, unabhängig vom Entwurf, durch getrennte Personen, ...
- Zusatzkontrollen der Testergebnisse: Format, Plausibilität der Werte, ...
- Entwicklung disversitärer Lösungsbeschreibungen zur Testauswahl und/oder Sollwertbestimmung.
- Review (aufgezeichneter) Istwerte.
- ...

4.2 Kontrollflussorientierte Testauswahl

Ein Beispielprogramm und sein Kontrollflussgraph

```
int Ct_A, Ct_B, Ct_N;
int ZZ(int Ct_max){
    char c;
n0: Ct_A=0; Ct_B=0; Ct_N=0;
n1: while (Ct_N<Ct_max){
n2:  c=getchar();
n3:  if (is_TypA(c))
n4:   Ct_A++;
n5:  else if (is_TypB(c))
n6:   Ct_B++;
n7:  Ct_N++;
n8: } //Test Abbruchbedingung
}
```



Auswahlkriterien für Tests

1. Anweisungsüberdeckung: Jede Anweisung muss mindestens einmal ausgeführt werden. Beispiel: Start, n_0, n_1, n_2, n_3, n_4, n_7, n_8, n_2, n_3, n_5, n_6, n_7, n_8, Ende

2. Kantenüberdeckung: Jede Kante muss mindestens einmal durchlaufen werden. Beispiel: Start, n_0, n_1, n_2, n_3, n_4, n_7, n_8, n_2, n_3, n_5, n_6, n_7, n_8, n_2, n_3, n_5, n_7, n_8, Ende

3. Entscheidungsüberdeckung: Jede Entscheidung muss mindestens einmal von jeder Bedingung abhängen.

Beobachtbarkeit verfälscher Anweisungsergebnisse nicht gefordert, d.h. Fehlerannahmen besser als zu erwartende Fehler nachweisbar.

Bestimmung der Überdeckung bei manueller/zufälliger Testauwahl:

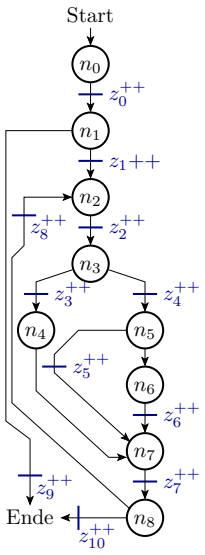
- Die Anweisungs- und Kantenüberdeckung lässt sich durch Einfügen von Zählern in das Programm vor der Compilierung bestimmen.
- Automatisierbar.

Kontrolle der Testergebnisse:

- einprogrammierte Überwachungsfunktionen,
- Trace-Aufzeichnung und Review aufgezeichneter Daten,
- Regressionstest.

Erweiterung der Kontrolle auf Anweisungsergebnisse möglich:

- Schrittbetrieb und manuelle Kontrolle der Zwischenergebnisse,
- Trace-Aufzeichnung und Inspektion aller Zwischenergebnisse.
- Einbeziehung der Beobachtbarkeit (siehe nächster Abschnitt).



Bestimmung der Kantenüberdeckung

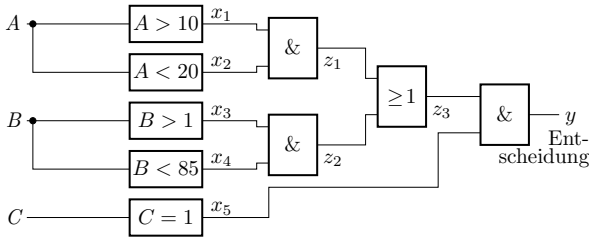
```
int z[11]={0,0,0,0, ...};
...
int ZZ(int Ct_max){char c;
n0: Ct_A=0;Ct_B=0;Ct_N=0;z(0)++;
n1: while (Ct_N<Ct_max){ z(1)++;
n2:  c=getchar(); z(2)++;
n3:  if (is_TypA(c)){
n4:   z(3)++; Ct_A++;}
      else {z(4)++;
n5:   if (is_TypB(c)){
n6:    Ct_B++; z(5)++;}
      else z(6)++;
     }
n7:  Ct_N++; z(7)++;
n8: ...}
```

[7] Zur Unterbringung aller Zähler Schleife in Maschinenbefehle auflösen.

Bedingungsüberdeckung
Ein logischer Ausdruck, z.B.

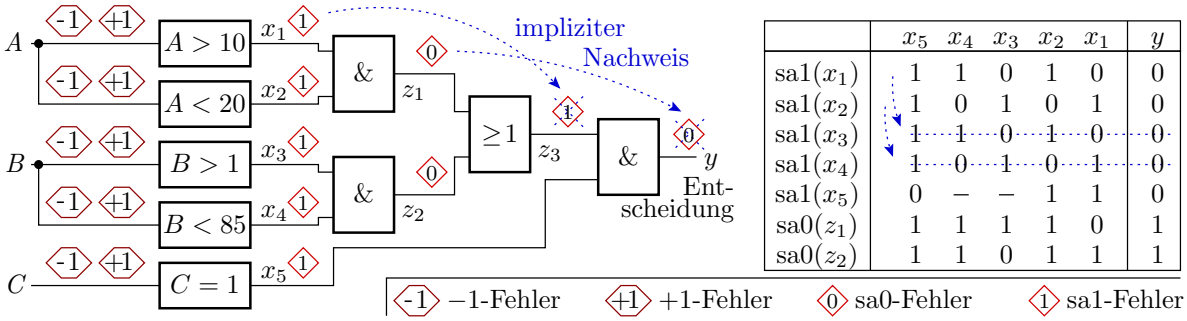
```
n1: if (((A>10) && (A<20)) || ((B>1) && (B<100))
        && (C==1)) {
n2:   ... }
    else {
n3: ...   }
```

ist nachbildbar durch einen Schaltplan aus Gattern und Vergleichern:



Berechnungsfluss mit eingezeichneten Fehlern

Die Bestimmung der Bedingungsüberdeckung lässt sich auf die Modellierung von Haftfehlern, (+1)-Fehler und (-1)-Fehlern zurückführen.



	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

Weiter wie bei Haftfehlern

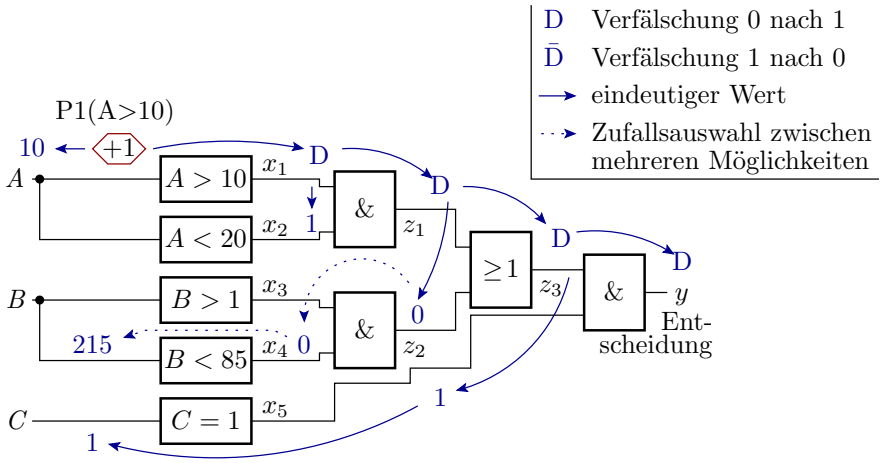
- Zusammenfassen identischer Fehler. Streichen redundanter und implizit nachweisbarer Fehler (vergl. Foliensatz F2, Abschn. 1.3 und dieser Foliensatz, Abschn. 2.2).
- Die $\mp$1-Fehler implizieren den Nachweis aller Haftfehler nach den Vergleichsoperatoren, ...
- Eventuelle redundante Fehler deuten auf Möglichkeiten zur Programmvereinfachung.

Fehlersimulation und Testberechnung für die so zusammengestellte Modellfehler- (Mutations-) Menge innerhalb der logischen Ausdrücke:

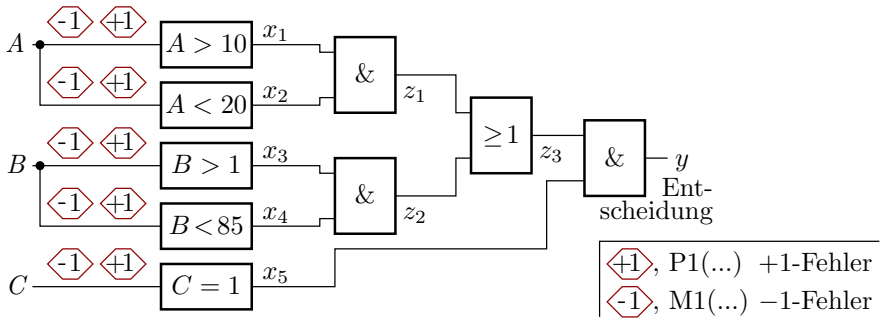
- könnte mit den für digitale Schaltungen etablierten Verfahren erfolgen.
- Das ist aber noch nicht Stand der Technik.

Die nächste Folie zeigt das Prinzip, wie für eine äquivalente Schaltung mit dem D-Algorithmus Tests berechnet werden.

Testsuche

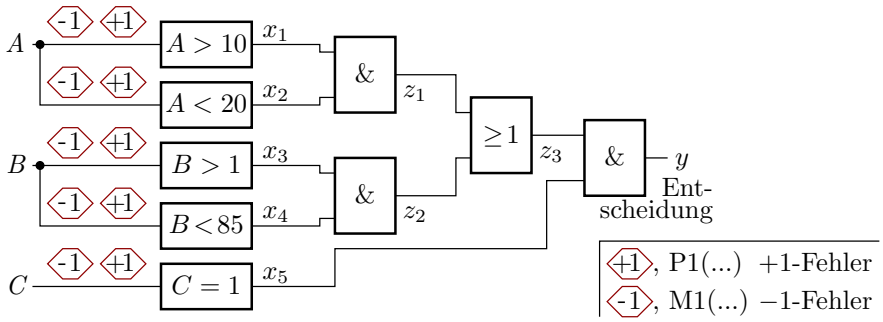


Für den »+1«-Fehler ist am Fehlerort ein Wert einzusetzen, bei dem sich Erhöhnung um eins das Vergleichsergebnis ändert. Vom Vergleichsergebnis wird ein D-Pfad zum Entscheidungsausgang durch zurücktreiben von Steuerpfaden zu Eingängen sensibilisiert.



	A	B	C	x_1	x_2	x_3	x_4	x_5	z_1	z_2	z_3	y
P1(A>10)	10	215	1	D	1	-	0	1	D	0	D	D
M1(A>10)												
P1(A<20)												
M1(A<20)												
P1(B>1)												
M1(B>1)												
P1(B<85)												
...												

Lösung



	A	B	C	x_1	x_2	x_3	x_4	x_5	z_1	z_2	z_3	y
P1(A>10)	10	215	1	D	1	-	0	1	D	0	D	D
M1(A>10)	11	0	1	$\bar{D}$	1	0	-	1	$\bar{D}$	0	$\bar{D}$	$\bar{D}$
P1(A<20)	19	86	1	1	$\bar{D}$	-	0	1	$\bar{D}$	0	$\bar{D}$	$\bar{D}$
M1(A<20)	20	1	1	1	D	0	-	1	D	0	D	D
P1(B>1)	11	1	1	-	0	D	1	1	0	D	D	D
M1(B>1)	23	2	1	-	0	$\bar{D}$	1	1	0	$\bar{D}$	$\bar{D}$	$\bar{D}$
P1(B<85)	19	84	1	0	-	1	$\bar{D}$	1	0	$\bar{D}$	$\bar{D}$	$\bar{D}$
...												

Testanforderungen für heutige Software

Nach Standard DO-178 B gilt als ausreichend[8]:

- 100% Anweisungsüberdeckung für nicht sicherheitskritische Systeme,
- 100% Zweigüberdeckung für Software, die bedeutende Ausfälle verursachen kann,
- 100% Bedingungsüberdeckung für flugkritische Software.

Nach Stand der Technik noch nicht gefordert:

- Beobachtbarkeit der Anweisungsergebnisse an Ausgängen.
- Ergebniskontrolle im Schrittbetrieb.
- Mehrfachausführung zur Erhöhung der Wahrscheinlichkeit der Fehleranregung und Beobachtbarkeit.

4.3 Def-Use-Ketten

Def-Use-Ketten

Def-Use-Tupel: Datenstruktur, die aufeinanderfolgende Paare von Schreib- und Lesezugriffen einer Variable beschreibt. Programmbeispiel »größter gemeinsamer Teiler[9] «:

```
    int ggt(int a, int b){
n0:   int c = a;
n1:   int d = b;
n2:   if(c == 0)
n3:     return d;
n4:   while(d != 0){
n5:     if(c > d)
n6:       c = c - d;
n7:     else
n8:       d = d - c;
      }
n9:   return c;
    }
```

Var	Def	Use
d	n1	n3
d	n1	n4
d	n1	n5
d	n1	n6
d	n1	n8
d	n8	n4
d	n8	n5
d	n8	n6
d	n8	n8
c	n0	n2
...	...	...

[8] Bei »ausreichendem« Test kann sich der Hersteller der Produkthaftung im Falle eines Unfalls durch einen nicht erkannten Fehler entziehen.

[9] Aus https://de.wikipedia.org/wiki/Def-Use-Kette vom 17.10.2015.

Berechnung und Verwendung von Def-Use-Ketten

Berechnung aller Def-Use-Tupel:

```
Wiederhole für alle Lesezugriffe aller Variablen:
    suche die Anweisungen, die den Wert geschrieben haben könnten
```

Verwendung als Testvollständigkeitskriterien:

- Für alle »Defs« mindestens ein »Use«.
- Für alle »Use« mindestens ein »Def«.
- Alle Def-Use-Tupel.
- Def-Use-Überdeckung als Testgüte (wenig populär).

Statische Code-Analyse:

- »Use« ohne »Def« ist ein Initialisierungsfehler.
- »Defs« ohne »Use« sind redundanter Code.

Fehlerlokalisierung:

- Rückverfolgung des Def-Use-Graphen zur Suche der Entstehungsursachen von Verfälschungen.

Beispiel: Rückverfolgung in »größter gemeinsamer Teiler«:

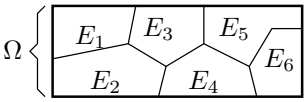
```
int ggt(int a, int b){
n0:   int c = a;
n1:   int d = b;
n2:   if(c == 0)
n3:     return d;
n4:   while(d != 0){
n5:     if(c > d)
n6:       c = c - d;
n7:     else
n8:       d = d - c;
      }
n9:   return c;
    }
```

Wenn »n9« FF, dann sind die möglichen »Defs«, an denen Unterbrechungspunkte beim nächsten Testdurchlauf zu setzen sind, »n0« und »n6«

4.4 Äquivalenzklassen

Testauswahl mit Äquivalenzklassen

- Äquivalenzklasse: Eingabemenge ähnlich zu verarbeitender Daten.
- Fehlerannahme A: Fehler in der Verarbeitung werden mit jedem Beispiel der Klasse mit hoher Wahrscheinlichkeit nachgewiesen.
- Fehlerannahme B: Spezifikations- und Implementierungsfehler sind oft falsch gesetzte Bereichsgrenzen.



Ω Eingaberaum
E_i Äquivalenzklasse

Testauswahl / Basis:

- Fertiges Programm: Testauswahl vergleichbar mit der für »Bedingungsüberdeckung« (siehe Seite 35).
- Spezifikation: Erstellen einer zum Testobjekt diversitären Fallbeschreibung. Weiter wie »Bedingungsüberdeckung«.

Spezifikationsbasierte Testauswahl

1. Zusammenstellung der Eingabedaten, Ausgabedaten, Berechnungsvorschriften und Bedingungen, die bei der Berechnung zu unterscheiden sind.

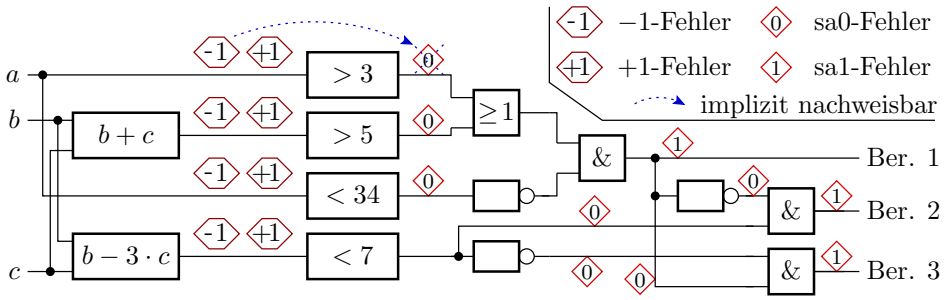
2. Bildung von Äquivalenzklassen durch Unterteilung der Eingabewertebereiche, beschreibbar durch ein Programm mit Fallunterscheidungen und Dummy-Funktionen für die Ausführung.

3. Konstruktion eines Tests mit 100% Anweisungs-, Zweig- oder Bedingungsüberdeckung für die so entstandene Programmbeschreibung.

4. (Manuelle) Sollwertbestimmung entsprechend Eingabebereich und zugeordneter Sollfunktion.

Beispiel einer aus der Spezifikation gewonnen Äquivalenzklassenbeschreibung

```
int fkt(int a, int b, int c){
 if((a>3)||(b+c>5))&& !(a<34))printf("Berechn. 1");
 else if(b-3*c<7)             printf("Berechn. 2");
 else                         printf("Berechn. 3");
}
```



Testauswahl / Überdeckungskontrolle weiter wie »Bedingungsüberdeckung« ab Seite 35.

4.5 UW-Analyse

Ursache-Wirkungs-Analyse

Bei der UW-Analyse wird wie bei dem spezifikationsbasierten Äquivalenzklassenverfahren aus der Zielfunktion eine zum Testobjekt diversitäre Beschreibung abgeleitet.

Der empirische Ansatz ist anders.

Statt nach Wertebereichen und diesen zugeordneten Verarbeitungsfunktionen wird die Zielfunktion sortiert nach:

- Auslösern für Aktionen (Ursachen) und

- ausgelösten Aktionen (Wirkungen).

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

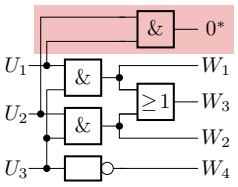
Wirkungen sind einzeln spezifizierte Zielfunktionen, ergänzte selbstverständliche Funktionen und Fehlerbehandlungen.

Jede Ursache und Wirkung wird durch eine binäre Variable (nicht eingetreten/eingetreten) beschrieben.

Zwischen den Ursachen und Wirkungen werden logische Verknüpfungen formuliert.

Die Testauswahl selbst ähnelt denen für »Bedingungsüberdeckung« und »Äquivalenzklassen« und lässt sich auch wieder auf die für Haftfehler zurückführen.

Beispiel »Zähle Zeichen«



* Eingabe kann nicht gleichzeitig Typ A und B sein

- Wirkungen:
  W_1: Anzahl_TypA +1[10]
  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

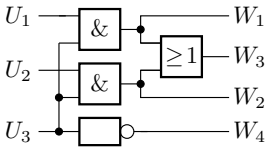
Test mit allen einstellbaren Ursachen

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	

- Sich ausschließende Ursachen: UND-Verknüpfung muss »0« sein.
- Eine Ursache-Wirkungs-Analyse deckt Mehrdeutigkeiten und Widersprüche in der Spezifikation auf.

Beispielimplementierung als C-Funktion

```
int Ct_A, Ct_B, Ct_N;
    int ZZ(int Ct_max){
    char c;
    Ct_A=0; Ct_B=0; Ct_N=0;
U3: while (Ct_N<Ct_max){
      c=getchar();
U1:  if (is_TypA(c))
W1:   Ct_A++;
U2:  else if (is_TypB(c))
W2:   Ct_B++;
W3:  Ct_N++;
W4:  }
    }
```



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	

Testbeispiel konkret /symbolisch

Funktions-aufruf	Eingabe	Sollzähl-werte	Ursachen U_1	U_2	U_3	Wirkungen 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 vom Typ A (Ziffer)	W_1	Ct_A++
U_2	Zeichen ist vom Typ B (Großbuchstabe)	W_2	Ct_B++
U_3	max. Zählwert nicht erreicht	W_3	Ct_N++
–	es wird kein Zeichen gelesen	W_4	Ende

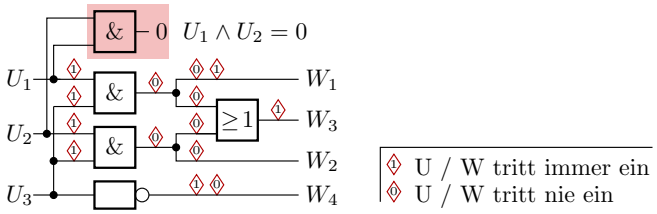
[10] Im Programmbeispiel wird Typ A Ziffer und Typ B Großbuchstabe sein.

Ungereimtheiten / Haftfehler

Erkennbare Ungereimtheiten:

- Im UW-Graph können bei »$U_3 = 0$« (max. Zählwert erreicht) Zeichen vom Type A oder B eingegeben werden, im Programm nicht. Wie lautet das gewünschte Sollverhalten?

Haftfehler im UW-Graph (identisch nachweisbare Fehler zusammengefasst):



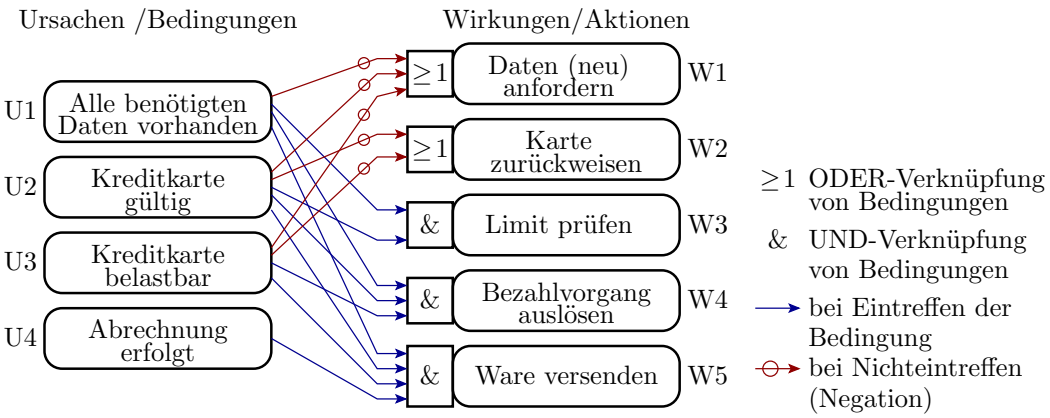
- Im Beispiel würde ein Test mit allen Kombinationen von Ursachen auch alle nachweisbaren Haftfehler erfassen.

- Für eine große Anzahl von Ursachen kann die Anzahl der Haftfehler auch wesentlich kleiner als die Anzahl der Ursachenkombinationen sein.

- Nach Berechnung der gleichzeitig zu (de-) aktivierenden Ursachen folgt die Suche geeigneter Eingaben und Kontrollen.

Beispielaufgabe

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 als logischen Signalflussplan dar.

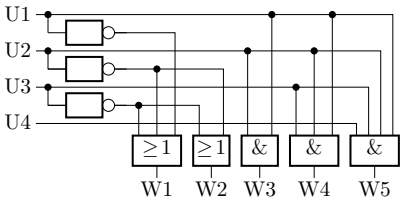
2. Bestimmen Sie in dieser Darstellung für Wirkung W3 die Menge der unterschiedlich nachweisbaren Haftfehler ohne redundante und implizit nachweisbare Fehler.

3. Suchen Sie für alle (drei) Haftfehler eine Menge von Ursachenkombinationen, mit denen sie anhand ihrer Wirkung nachweisbar sind.

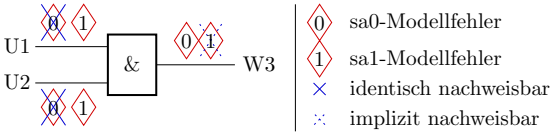
4. Bestimmen Sie für die (drei) Haftfehler die Nachweiswahrscheinlichkeiten für die Auftrittshäufigkeiten der Ursachen $h\,(\mathrm{U1}) = 30\%$, $h\,(\mathrm{U2}) = 70\%$, $h\,(U3) = 20\%$ und $h\,(U4) = 80\%$.

Lösung Aufgabenteil 1 und 2

1. Ursache-Wirkungs-Beziehung als logischen Signalflussplan:



2. Anfangsfehlermenge 6 Haftfehler. sa0(U1), sa0(U2) und sa0(W3) sind identisch und sa1(W3) implizit von sa1(U1) und sa1(U2) nachweisbar:



Lösung Aufgabenteil 3 und 4

3. Möglicher Testsatz:

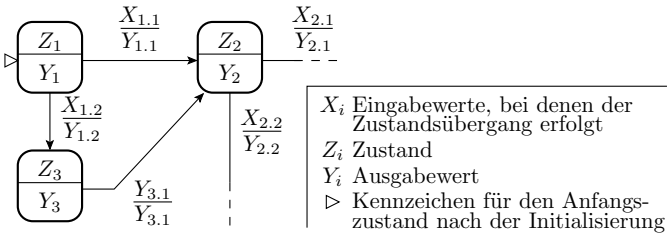
Fehler:	sa1(U1)	sa1(U2)	sa0(W2)
Test:	U1=0, U2=1	U1=1, U2=0	U1=1, U2=1

4. Nachweiswahrscheinlichkeit für $h\,(\mathrm{U1}) = 30\%$ und $h\,(\mathrm{U2}) = 70\%$:

U2	U1	Auftrittshäufigkeit	sa1(U1)	sa1(U2)	sa0(W2)
0	0	$30\% \cdot 70\% = 21\%$			
0	1	$30\% \cdot 30\% = 9\%$		x	
1	0	$70\% \cdot 70\% = 49\%$	x		
1	1	$70\% \cdot 30\% = 21\%$			x
Nachweiswahrscheinlichkeit:			49%	9%	21%

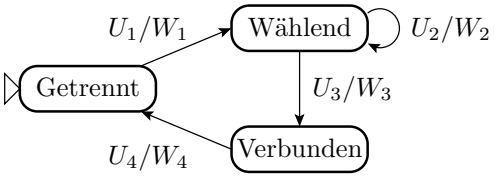
4.6 Automaten

Zielfunktion als Automat



Das Automatenmodell beschreibt die Zielfunktion eines Systems durch Mengen von Eingaben, Ausgaben, Zuständen und Zustandsübergängen. 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.

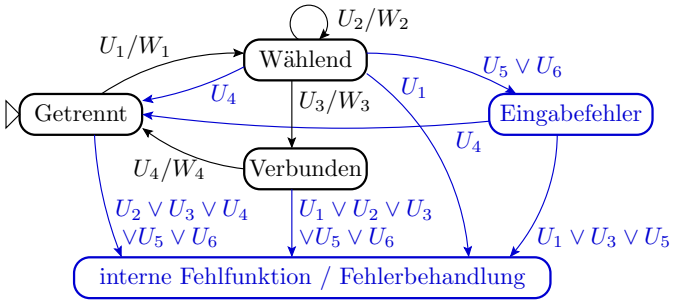
Verbindungsaufbau und -abbau beim Telefonieren



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 Sollfunktion: $U_1 \rightarrow U_2 \rightarrow \ldots \rightarrow U_2 \rightarrow U_3 \rightarrow U_4$
- 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-Ketten beschrieben (vergl. Foliensatz TV_F1).

5 Baugruppen

Baugruppentest

Inbetriebnahme von Prototypen:

- Sichtprüfung auf Bestückungs- und Lötfehler.
- Kontrolle der Verbindungen über Widerstandsmessungen.
- Anschluss der Spannung. Stromüberwachung. Kontrolle der Bauteile auf unnormale Erwärmung.
- Anschluss von Signalgeneratoren, Oszillographen, ...
- Manuelle Einstellung der Eingaben und Ausgabekontrolle.

Serienfertigung:

- Automatisierte statische Tests auf Verbindungs- und Bestückungsfehler (vergl. Foliensatz TV_F4, Abschn. 3).
- Auf das Ausprobieren von Beispielfunktionen wird zum Teil verzichtet[11].

Modularer Funktionstester

Der komplette Test von CP- Systemen verlangt die Bereitstellung von Testverläufen für die Sensorsignale und 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, ...

[11] Annahme: Schaltkreis- und andere Bauteilfehler werden fast alle von den Bauteiltests und die verbleibenden kaum im Verbund erkannt. Statische Tests erkennen (fast) alle Bestückungs- und Verbindungsfehler.

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.