Test und Verlässlichkeit Foliensatz 4: Statische Tests

Prof. G. Kemnitz

Institut für Informatik, TU Clausthal (TV_F4)

20. Juni 2017

Inhalt TV_F4: Statische Tests

Test allg.
1.1 Statische und dynamische Tests
1.2 Kosten, emotionale Barrieren
1.3 Produkthaftung und Standards

Statische SW-Tests
2.1 Inspektion
2.2 Syntax, Typ, WB
2.3 Statische Code-Analyse

Baugruppen und CPS
3.1 Inbetriebnahme
3.2 MDA
3.3 Optische Inspektion
3.4 In-Circuit-Test
3.5 Boundary-Scan

Statische Tests für Schaltkreise

Literatur

Test allg.

Statische und dynamische Tests

Statische und dynamische Tests

Definitionen

Statischer Test: Direkte Kontrolle auf Abwesenheit von Fehlern und Regelverstößen.
Dynamischer Test: Ausprobieren der Funktion mit einer Stichprobe von Eingaben.

Statische Tests:

- Kontrollen der Systembeschreibung und am System.
- Testfälle auf Fehler ausgerichtet z.B. Kontrolle auf Kurzschlüsse durch Widerstandsmessungen.
- Fehlerlokalisierung: Reparaturmaßnahme oft direkt aus dem Testergebnis ablesbar.
- Ausführbar, bevor das System ansatzweise funktioniert.
- Keine Garantie für irgend eine Funktion.

Dynamische Tests:

- Benötigen ein funktionierendes (simulierbares) Testobjekt.
- Fehlerlokalisierung oft aufwändig.
- Erfolgen meist nach Beseitigung der mit statischen Tests nachweisbaren Fehler.
- Demonstrieren Funktionalität, Vertrauensbildung, ...

Eine Prüftechnologie kombiniert statische und dynamische Tests.

Dieser Foliensatz behandelt statische Tests:

- Software und Hardware-Entwürfe: Inspektion, Syntaxtest, ...
- Baugruppen: Tests auf Kurzschlüsse, Unterbrechungen und Bestückungsfehler,
- Schaltkreise: Stichprobenkontrolle von Schichten, Kontrolle der Stromaufnahme, ...

Kontrollfrage

Nachfolgend ist eine vereinfachte Prüftechnologie beschrieben.

Welche der Tests sich statisch und welche dynamisch?

1. Kontrolle der Spezifikation durch den Auftraggeber.
2. Kontrolle der Lösungsidee durch einen Kollegen.
3. Syntaxtest des Programms.
4. Ausprobieren von Testbeispielen durch den Entwerfer.
5. Ausprobieren von Testbeispielen durch den Anwender.
6. Kontrolle der Dokumentationen.

Lösung:

- statisch: 1, 2, 3 und 6
- dynamisch: 4 und 5

Kosten, emotionale Barrieren

Testziele und Kosten

Testen dient nicht nur zum Auffinden von Fehlern, sondern auch

- zur Kontrolle der Wartbarkeit, Bedienbarkeit,
- Einhaltung von Standards, selbstverständlicher Anforderungen, z.B. dass eine Software installierbar sein muss,
- Kontrolle der Zuverlässigkeit, Funktionsfähigkeit unter vorgeschrieben Umgebungsbedingungen, ...
- Vertrauensbildung und Ausschluss einer Produkthaftung.

Probleme, die von den Tests übersehen werden, verursachen (oft erhebliche) Kosten.

Zehnerregel

IT-Systeme entstehen in Phasen und sind hierachisch aufgebaut.

- Bei jedem Phasenübergang (Anforderungsanalyse, Pflichenhefterstellung, ...) und
- auf jeder Hierarchieebene

wird getestet. Die Kosten, die ein Fehler verursacht, vervielfachen sich mit jedem Phasen- und Hierarchieübergang, bei dem er noch nicht beseitigt ist.

Die Zehnerregel postuliert eine Verzehnfachung:

- Kosten Bauteilfehler gleich erkannt: 1 Geldeinheit,
- erst auf der Baugruppe erkannt: 10 Geldeinheiten,
- erst im Gerät erkannt: 100 Geldeinheiten,
- erst in der Anwendung erkannt: 1.000 Geldeinheiten.

Emotionale Barrieren

Es ist noch nicht überall selbstverständlich, dass Fehlerentstehung unvermeidlich und an den Entstehungsaufwand gekoppelt ist.

Kontraproduktiv / schlechtes Management:

- »It is not a bug, it is a feature!« (Grün-Reden von Fehlern.)
- Fehler übersehen zur Konfliktvermeidung mit Verursacher.
- Weglassen von Tests, um Zeit- und Kosten zu sparen[1].

Anstrebenswert:

- Tester für weniger im Einsatz gefundene Fehler honorieren.
- Interesse der Entwickler an frühzeitiger Beseitigung ihrer Fehler. (Sabotage der Fehlerauffindung unvorteilhaft).

Die besten Tester: Maschinen, Pedanten, Authisten, ...

[1] Eingesparte Testkosten vervielfachen die fehlerbezogenen Kosten.

Produkthaftung und Standards

Produkthaftung

Fehler in IT-Systeme können erheblichen Schaden verursachen.

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« und muss im Schadensfall nachweisen, dass er nach Stand der Technik alles für die Schadensabwendung getan zu hat.

Wie lässt sich die Erfüllung der »Sorgfaltspflicht« bei einem vom Test übersehenen Fehler nachweisen?

Dazu dienen Standards, die in »abhakbarer« Weise beschreiben, was für Tests und andere verlässlichkeitssichernde Maßnahmen als ausreichend gelten und wie deren Erbringung zu dokumentieren ist.

Testfall nach [ANSI/IEEE-Standard 829]: Für einen Testfalls sind zusätzlich zu den Eingaben und Sollausgaben zu dokumentieren:

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

Weitere relevante Standards:

- ISO 9126/DIN 66272: Qualitätsmerkmale für Software.
- ANSI/IEEE Std 829-1998: Standard für Software Test Dokumentationen.
- ANSI/IEEE Std 1008-1993: Standard für Software Unit Test.
- ANSI/IEEE Std 1012-1998: Standard für Software Verfication and Validation Plans. ...

Erhebliche Testaufwandsanteile zur Vermeidung von Produkthaftung, Vertrauensbildung, ... statt zur Sicherung der Verlässlichkeit.

Statische SW-Tests

Statische Tests für SW und Entwurfsbeschreibungen

Jeder Systemtyp (SW, HW, SW+HW, CPS) hat testspezifische Besonderheiten. Software und Entwurfsbeschreibungen:

- Entstehung in Phasen: Spezifikation, Pflichtenheft, Architekturentwurf, ... zwischen denen zu testen ist.
- Einfache Fehlerbeseitigung durch Editieren der Dokumentationen, Programmtexte, ...
- Agile Entwürfe: Weiterentwicklung und Fehlerbeseitigung durch Updates während der gesamten Nutzungsdauer.

Statische Tests für Software und Hardware-Entwürfe:

- Inspektion (Review) von der Spezifikation über den Quellcode bis hin zu den Dokumentationen.
- Syntax- Typ- und Wertebereichskontrollen.
- Halb- und vollautomatisierte Code-Analysen.

Inspektion

Inspektion (Review)

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

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

Einordnung, Merkmale und Besonderheiten:

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

Automatisierung anstrebenswert.

Kenngrößen einer Inspektion

Inspektionsfehlerüberdeckung:

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

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

Abschätzmöglichkeiten:

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

Weitere Bewertungsgrößen für Inspektionen nach [4]:

- Effizienz: Gefundene Abweichungen pro Mitarbeiterstunde.
- Effektivität: Gefundene Abweichungen je 1000 NLOC[2].

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

Beispielaufgabe

Zur Bewertung einer Inspektion gesammelte Zählwerte und Zeiten:

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

Wie groß sind

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

der Inspektion?

Lösung

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

Gesucht: Inspektionsfehlerüberdeckung IFC, Effizienz (gefundene Fehler pro Mitarbeiterstunde). Effektivität (Gefundene Abweichungen je 1000 NLOC).

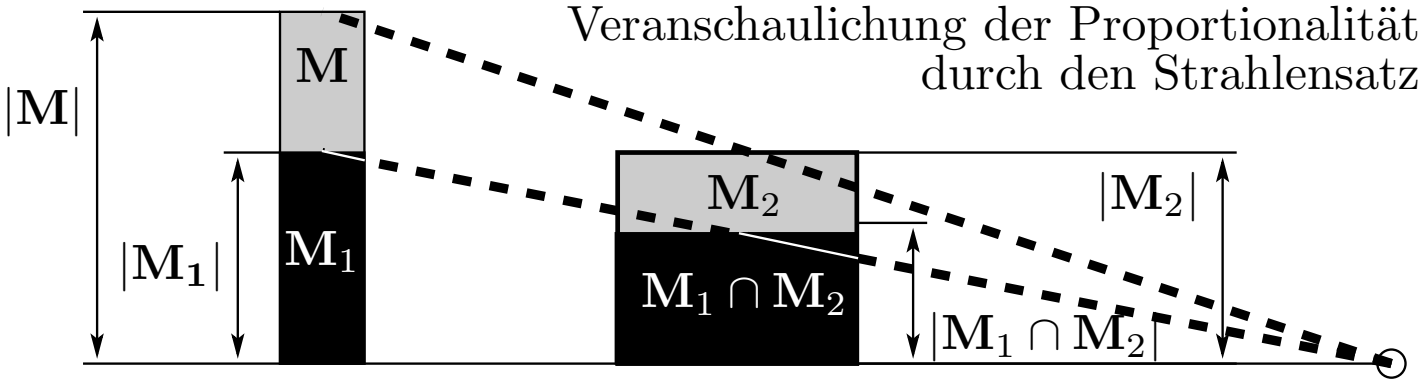
	gesamt	funktionale Fehler	sonstige Fehler
$IFC = \frac{\varphi_{\text{Erk}}}{\varphi}$	$\frac{228}{300}$	$\frac{156}{200}$	$\frac{72}{100}$
Effizienz	$\frac{228\text{ Fehler}}{200\text{ h}}$	$\frac{156\text{ Fehler}}{200\text{ h}}$	$\frac{72\text{ Fehler}}{200\text{ h}}$
Effektivität	$\frac{228\text{ Fehler}}{10.000\text{ NLOC}}$	$\frac{156\text{ Fehler}}{10.000\text{ NLOC}}$	$\frac{72\text{ Fehler}}{10.000\text{ NLOC}}$

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

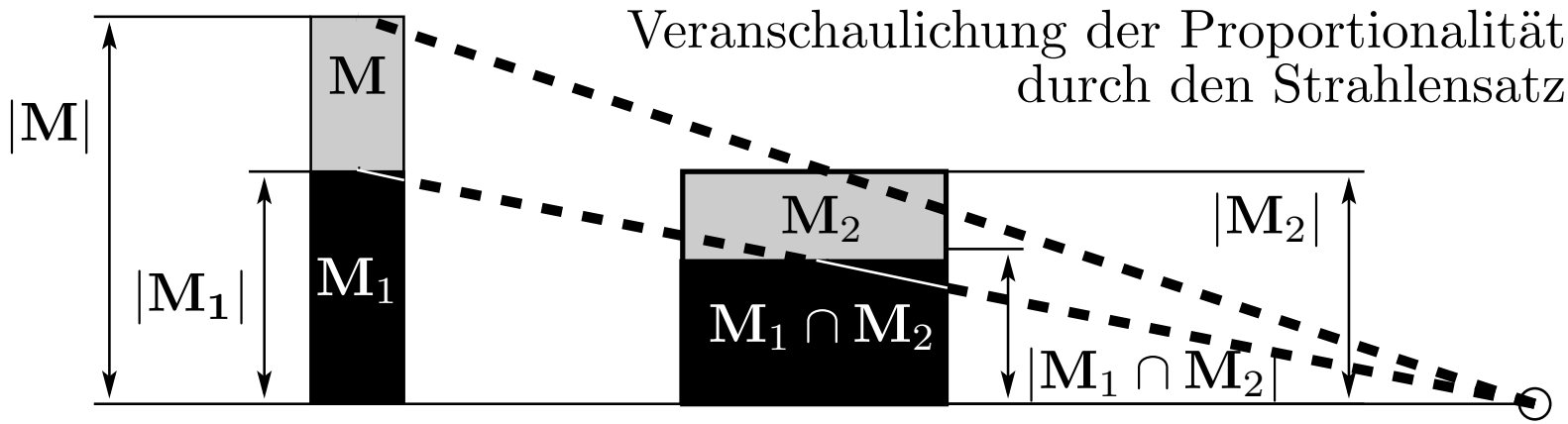
Capture-Recapture-Verfahren

Abgeleitet von einem Schätzer für die Größe von Tierpopulationen (z.B. von Vögeln in einem Gebiet) [2, 6, 5].

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



(|...| – Größe der Mengen).



(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 gleichen Fehler; $|\mathbf{M}|$ – geschätzte Anzahl der vorhandenen Fehler). Die geschätzte Fehlerüberdeckung ist das Verhältnis der Anzahl der insgesamt von beiden Inspekteuren gefundenen 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 unabhängig voneinander.
- Alle Fehler haben dieselbe Erkennungswahrscheinlichkeit.

Beispielaufgabe

Inspektionsergebnisse für ein Programm:

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

Welche Schätzwerte ergeben sich nach dem Capture-Recapture-Verfahren für

1. die Gesamtfehleranzahl,
2. die Inspektionsfehlerüberdeckung?

Lösung

1. Gesamtfehleranzahl:

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

2. Inspektionsfehlerüberdeckung:

$$IFC \approx \frac{|\mathbf{M}_1 \cap \mathbf{M}_2| \cdot |\mathbf{M}_1 \cup \mathbf{M}_2|}{|\mathbf{M}_1| \cdot |\mathbf{M}_2|} \quad (1)$$

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

Vertrauenswürdigkeit der Schätzung

Erforderliche Anzahl von Zählwerten $x_{\text{ist.min}}$ für $|\mathbf{M}_1|$, $|\mathbf{M}_2|$, ... zum Schätzen mittlerer Eintrittswahrscheinlichkeiten in Anlehnung an Foliensatz F2, Abschn. 3.4, Gl. 24:

$$x_{\text{ist.min}} \approx \frac{\kappa \cdot \left(\Phi^{-1}\left(1-\frac{\alpha}{2}\right)\right)^2}{\varepsilon_{\text{rel}}^2}$$

($\Phi^{-1}(\ldots)$ – inverse Normalverteilung; κ – Varianzerhöhung durch Nachweisabhängigkeiten; ε_{rel} – relativer Intervallradius; α – Irrtumswahrscheinlichkeit.

Zahlenbeispiel: $\alpha = 1\%$, $\kappa = 1$, $\varepsilon_{\text{rel}} = 10\%$:

$$x_{\text{ist.min}} \approx \frac{1 \cdot \left(\Phi^{-1}\left(1-\frac{0{,}01}{2}\right)\right)^2}{0{,}1^2} = \frac{2{,}57^2}{0{,}01} \approx 660$$

Erforderliche Größenordnung der Zählwerte ist 10^2 bis 10^3.

Bei Multiplikationen und Divisionen addieren sich die relativen numerischen Fehler. In Gl. 1 von vier Werten:

$$IFC \approx \frac{|\mathbf{M}_1 \cap \mathbf{M}_2| \cdot |\mathbf{M}_1 \cup \mathbf{M}_2|}{|\mathbf{M}_1| \cdot |\mathbf{M}_2|} \Rightarrow E\left(IFC\right) \in IFC \cdot (1 \mp 4 \cdot \varepsilon_{\text{rel}})$$

Vertrauenswürdige Schätzungen verlangen tausende gefundene Fehler.

Wenn beide Inspekteure genau dieselben Fehler finden $|\mathbf{M}_1| = |\mathbf{M}_2|$, ergibt sich $IFC = 100\%$. Ursachen für $|\mathbf{M}_1| = |\mathbf{M}_2|$ können aber auch sein:

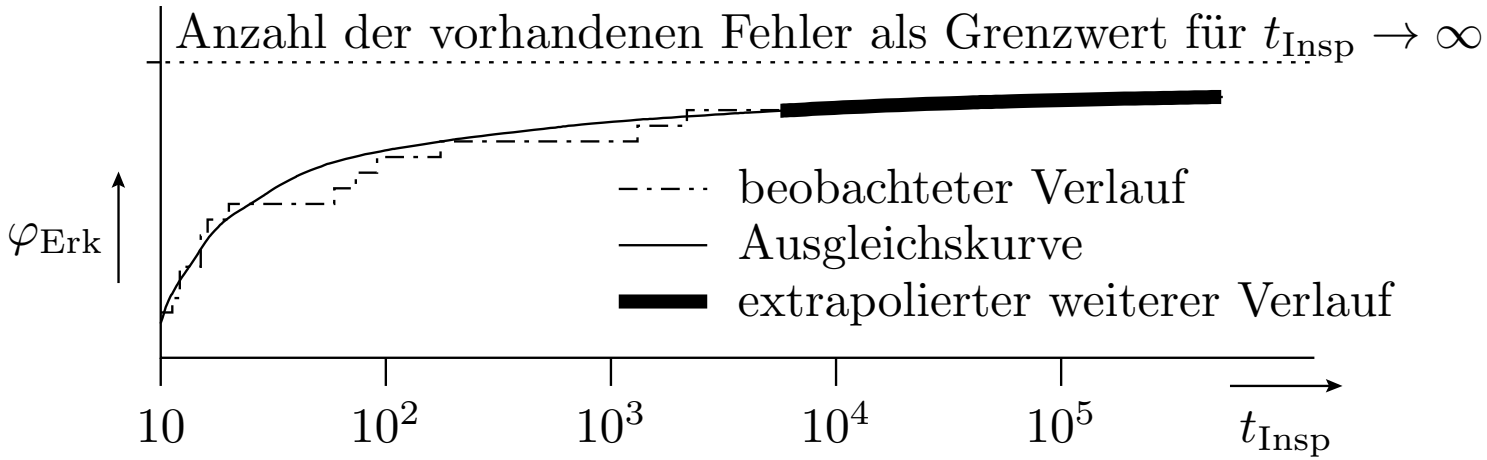
- Gegenseitig Information über die gefundenen Fehler.
- Die gefundenen Fehler waren viel leichter als die übrigen Fehler zu finden. ...

Schätzwerte der Anzahl der nicht gefundenen Fehler und der Inspektionsfehlerüberdeckung »nach Capture-Recapture« sind mit erheblichen zufälligen und systematischen Schätzfehlern behaftet.

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[3]:

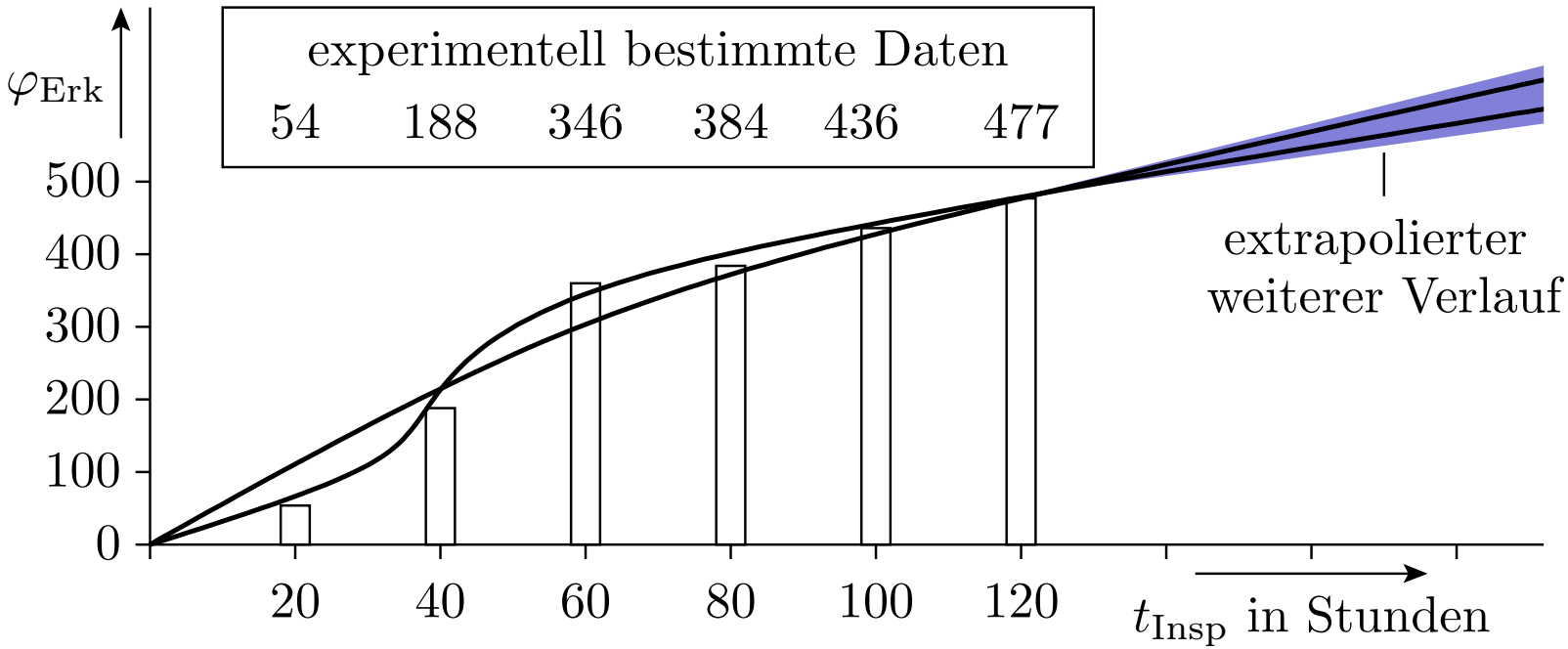


[3] Untersuchungen in dieser Richtung in der Literatur noch nicht gefunden.

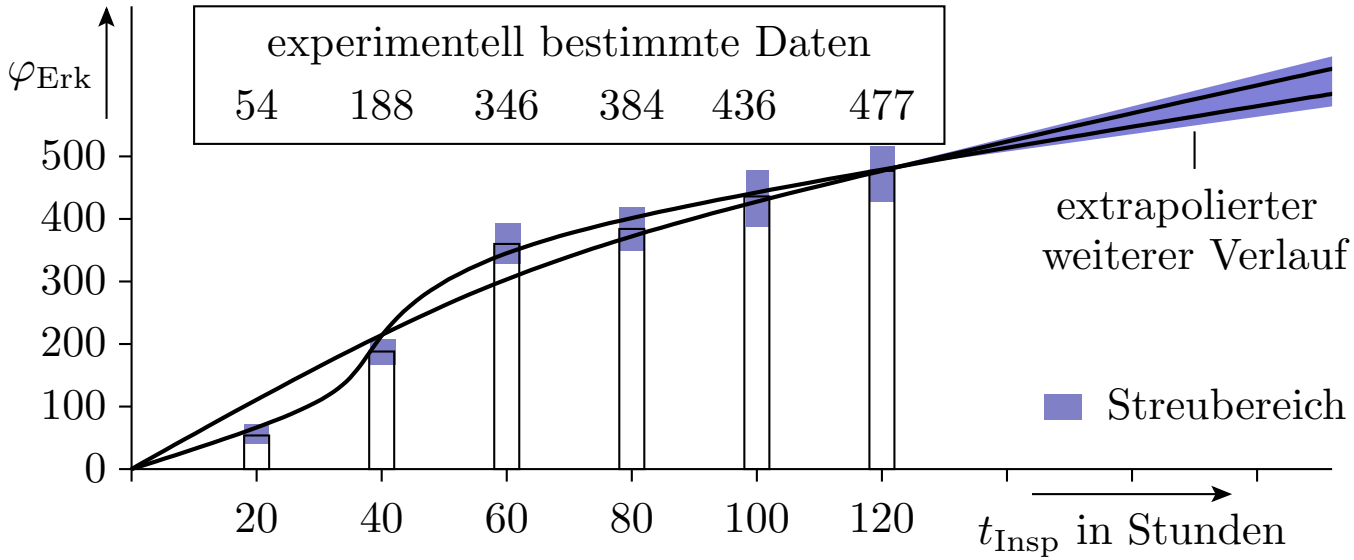
Experiment mit einem Inspekteur[4]

Inspektion des Buchmanuskripts [3] plus Beispielprogramme:

- Anzahl der gefunden Fehler in Abhängigkeit von der Inspektionsdauer.



[4] Bachelor-Arbeit von Yu Hong.



- Standardabweichung mindestens Wurzel aus Erwartungswert.
- Unterschiedliche Approximationsmöglichkeiten für die weitere Abnahme der zu erwartenden Anzahl der nicht gefundenen Fehler, z.B. wie beim Zufallstest mit einer FHSF[5]-Potenzfunktion:

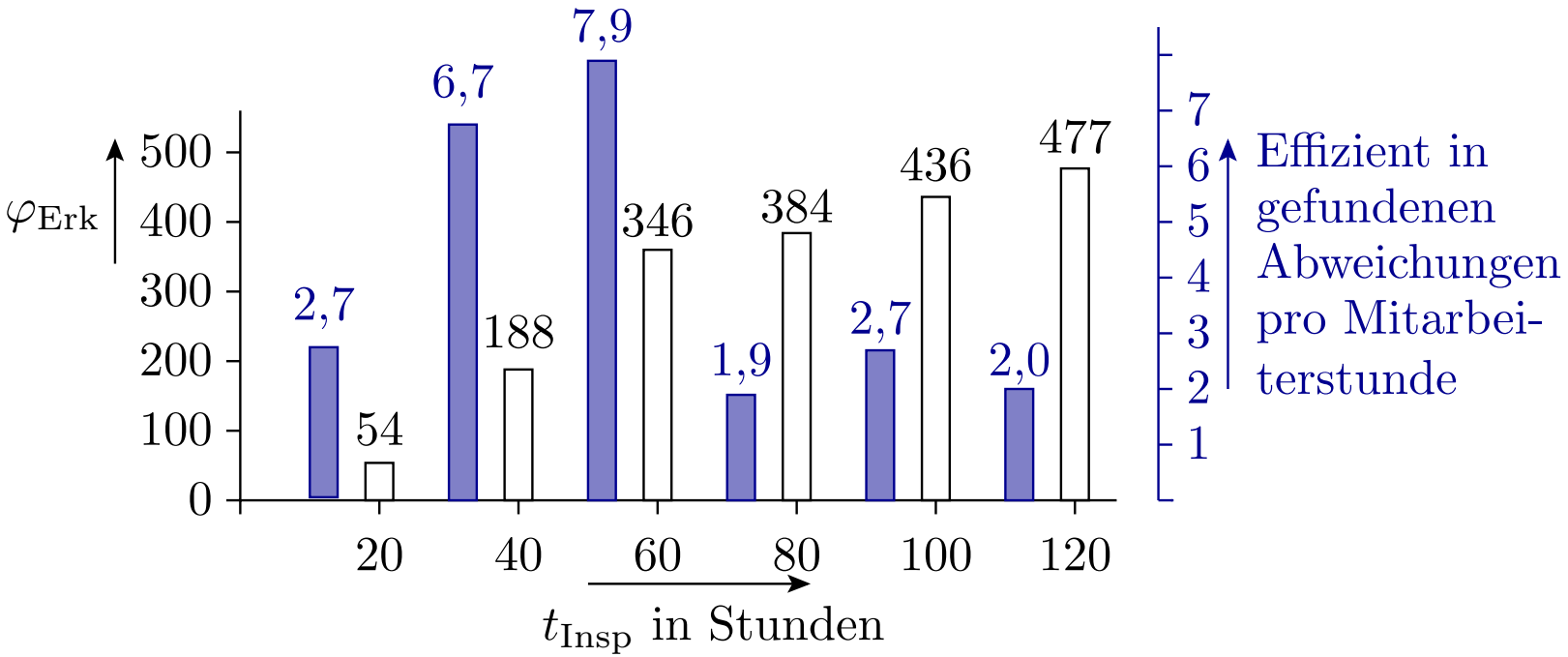
$$\varphi_{\text{NErk}}(t_{\text{Insp}}) = \varphi_{\text{NErk}}(t_0) \cdot \left(\frac{t_{\text{Insp}}}{t_0}\right)^{-k}$$

- Erhebliche systematische und zufällige Schätzfehler.

[5] FHSF – **F**ehlerauftritts**h**äufigkeit in Abhängigkeit von der mittleren Anzahl von **S**L je **F**F. Statt FF aber »Bemerken einer Unkorrektheit«.

Untschiede zwischen Inspektion und Zufallstest

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



Eine Inspektion hat eine »Anlernphase«, in der die Effizienz mit der Inspektionsdauer zunimmt.

- Beim dritten und vierten Lesen des Buchs und der Aufgabentexte nahm im Experiment nicht nur die Effizienz, sondern auch die dafür erforderliche Zeit deutlich ab, obwohl ein erheblicher Anteil (ca. 25%) der Fehler noch nicht gefunden war.

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

- Ein Mensch als Inspekteur ermüdet, wird blind für Fehler, ...

Ein gute Inspektionstechnologie vermeidet die uneffizienten Einarbeitungs- und Ermüdungsphasen.

Inspektionstechniken

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

Einteilung der Inspektionstechniken

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

Syntax, Typ, WB

Syntax-, Typ- und Wertebereichskontrollen

Die Übersetzung in eine rechnerinterne Darstellung eines Programms oder Simulationsmodells ist ein Service, der Formatkontrollen für die als Eingabe fungierende Entwurfsbeschreibung erlaubt:

- Syntaxtest.
- Typkontrolle für Ausdrücke, Zuweisungen und Zuordnungen.
- Wertekontrollen für konstante Ausdrücke.
- Wertebereichskontrollen (parzielle Korrektheit im Sinne, dass alle Zwischen- und Endergebnisse im darstellbaren Wertebereich liegen).

VHDL[6] – Sprache mit strenger Typkontrolle

VHDL definiert keine Datentypen, sondern Beschreibungsmittel, um Datentypen zu definieren. Beispiel seien die Zahlentypen.

- Ein Zahlentyp ist ein zusammenhängender Zahlenbereich:

  type *Zahlentyp* **is range** *Bereich*;

- Bereiche können auf- oder absteigend geordnet sein:

```
type tWochentag is range 1 to 7;
type tBitnummer is range 3 downto 0;
```

  $\text{Wochentag} \in \{1,2,3,4,5,6,7\}$; $\text{Bitnummer} \in \{3,2,1,0\}$

- ganzzahlige Bereichsgrenzen $\Rightarrow$ diskreter Zahlentyp
- Bereichsgrenzen mit Dezimalpunkt $\Rightarrow$ reeller Zahlentyp

```
type tWahrscheinlichkeit is range 0.0 to 1.0;
```

[6] Hardware-Beschreibungssprache mit für Kontrollen besonders gut geeignetem Typenkonzept.

Kontrollmöglichkeiten

- Kontrollen bei einer Variablenzuweisung

  Variablenname := *Ausdruck*;

  Typenübereinstimmung. Zulässiger Wert des Ausdrucks.

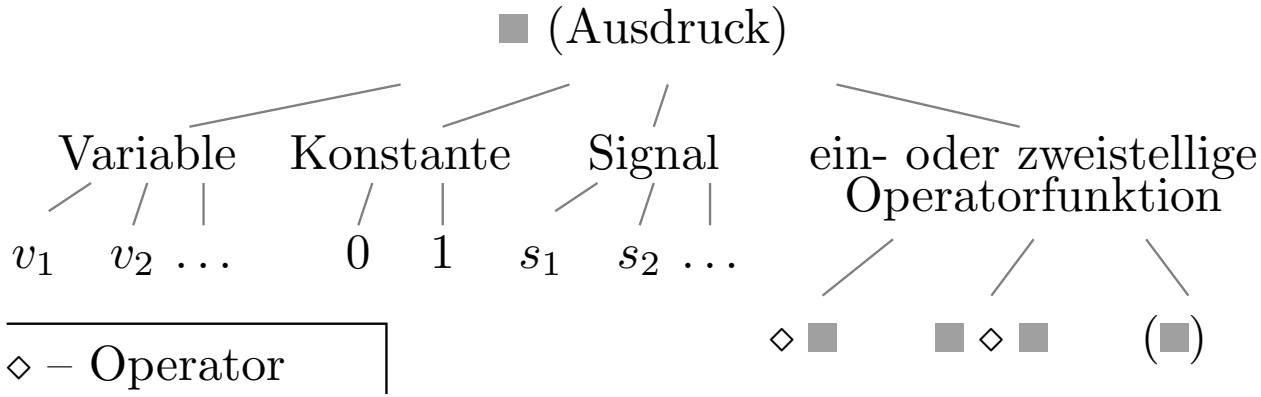
```
variable Wochentag: tWochentag;
variable Bitnummer: tBitnummer;
...
Wochentag := Bitnummer; -- Typ unzulaessig
Wochentag := 9;         -- Wert unzulaessig
```

- Kontrollen bei Signalzuweisungen

  Signalname <= W **[after** t_d**]{,** W **after** t_d**};**

  Gleicher Typ des Ausdruck W und des Signal. Ausdruck t_d muss Typ TIME haben; $t_\mathrm{d} \geq 0$.

Kontrollen in Ausdrücken: Ein Ausdruck ist eine rekursive Beschreibung einer Funktion mit ein- und zweistelligen Operatorfunktionen.



- Jeder Operator ist nur für bestimmte Operandentypen definiert.
- Jeder Kombination aus Operator und Operandentypen ist ein Ergebnistyp zugeordnet[7].
- Eine strenge Typenprüfung erkennt nicht nur, sondern vermeidet auch Fehler, indem sie den Programmierer zwingt, genauer über die beabsichtigte Zielfunktion nachzudenken.

[7] In VHDL sind die arithmetischen Operatoren (+, -, *, /) nur identische Operandentypen, z.B tWochentag, definiert.

Weitere Regeln in VHDL

- Logische Operatoren (and, or, ...) nur für identische Bit- oder Bitvektortypen. Ergebnis Operandentyp.
- Vergleichsoperatoren (>, =, ...): nur für identische Typen. Ergebnistyp boolean. ...

Regelabweichungen erfordern Typecast:

```
variable wt, r: t_Wochentag;
variable bn:    t_Bitnummer;
...
r = wt + bn;                    -- Typfehler
r = wt + t_Wochentag(bn);       -- mit Typcast zulässig
```

Kontrollen bei Zuordnungen (Schnittstellen, Aufrufparameter, ...)

BS => *BZ*{, *BS* => *BZ*} oder *ZB*{, *ZB*}

Typgleichheit zwischen zugeordneten Objekt *BZ* und den Objekten *BS*, denen etwas zugeordnet wird.

Beispielaufgaben für Zuweisungen

```
type tApfel is range 1 to 10;
type tBirne is range 1 to 10;
variable a1, a2: tApfel := 3;
variable b: tBirne := 2;
```

Zulässig, Typfehler, erkennbare Wertebereichsverletzung?

```
A1: a1 := a2 + 11;
A2: b  := a1 * a2;
A3: b  := b + a1;
```

Lösung

```
type tApfel is range 1 to 10;
type tBirne is range 1 to 10;
variable a1, a2: tApfel := 3;
variable b: tBirne := 2;
A1: a1 := a2 + 11;
A2: b  := a1 * a2;
A3: b  := b + a1;
```

Anweisung A1:

- »tApfel« + »universeller Zahlentyp« ⇒ »tApfel« √
- Ergebnistyp und Typ des Zuweisungsziels »tApfel« √
- Ergebnis ist 14. Wertebereichs des Ergebnisse $\{1, 2, \ldots, 10\}$!

Anweisung A2:

- »tApfel« * »tApfel« ⇒ »tApfel« √
- »tBirne« := »tApfel« ! (Typfehler)

Anweisung A3:

- »tBirne« * »tApfel« ! (Typfehler)

Beispielaufgaben für Ausdrücke

```
type tWahrsch is range 0.0 to 1.0;
variable w: tWahrsch;
variable r: real; -- vordefinierter Typ
```

Ausdrücke zulässig? Welchen Typ hat das Ergebnis?

```
... w * 2.0   -- Ergebnistyp tWahrsch
... w * 2     -- unzulässig
... 3 + 5     -- universeller ganzzahliger Typ
... 3 * 2.5   -- unzulässig
... w > 0.0   -- Ergebnistyp BOOLEAN
... w = 0     -- unzulässig
... w * r     -- unzulässig
```

Typecast zur Zulassung unerlaubter Typzuordnungen:

```
... w * tWahrsch(2)     -- Ergebnistyp tWahrsch
... 3 * interger(2.5)   -- Ergebnistyp INTEGER
... interger(w) = 0     -- Ergebnistyp BOOLEAN
```

Untertypen

Ableitung vom Basistyp durch Wertebereichsbeschränkung, z.B.:

```
subtype tWochenende is tWochentag range 6 to 7;
```

- Ein Untertyp wird bei der Typenprüfung wie sein Basistyp behandelt und
- erbt alle zulässigen Operationen, Funktionen etc.,
- kann aber nur Werte seines Wertebereichs annehmen.

```
variable wt: tWochentag;
variable we: tWochenende;
...
A1: we := wt;
A2: wt := ((we + 4) mod 7) + 1;
```

A1: Typenprüfung o.k.; Simulationsfehler bei $wt \notin \{6, 7\}$

A2: Typenprüfung o.k.; $we = 6 \mapsto 4$; $we = 7 \mapsto 5$ o.k.

Wertebereichskontrollen

Konstante Ausdrücke ersetzt ein Compiler durch ihren Wert und kontrolliert (idealerweise), dass der Wert mit dem Typ des Zuweisungsziels darstellbar ist.

Beispiel Konstantenzuweisung in C:

```
uint8_t a = (5<<7); // 10 1000 0000>0xFF(Fehler)
```

Für zu berechnende Werte

- ist die Einhaltung der Wertebereiche durch Programmanweisungen zu kontrollieren.
- In fast allen Sprachen incl. C manuell einzuprogrammieren.

Bessere Alternative:

- Wertebereichsverletzungen für alle zulässigen Ein- und Ausgaben ausschließen.
- Ein- und Ausgabebereichverletzungen mit »assert« abfangen.

Berechnung des Drain-Stroms für MOS-Transistoren

$$I_\mathrm{D} = K \cdot \begin{cases} 0 & U_\mathrm{GS} < U_\mathrm{th} \\ \frac{(U_\mathrm{GS}-U_\mathrm{th})^2}{2} & \text{sonst wenn } (U_\mathrm{GS} - U_\mathrm{DS} < U_\mathrm{th}) \\ (U_\mathrm{GS} - U_\mathrm{th}) \cdot U_\mathrm{DS} - \frac{U_\mathrm{DS}^2}{2} & \text{sonst} \end{cases}$$

Berechnung in VHDL mit Festkommazahlen. Vereinbarung der Datentypen und Variablen:

```
type tSpg    is -1E9 to 1E9; -- Spannung in 10nV
type tSteil  is -1E9 to 1E9; -- Steilheit in 10nA/V^2
type tStrom  is -1E9 to 1E9; -- Strom in 10nA

variable UGS, UDS, UTH: tSpg;
variable k: tSteil;
variable ID: tStrom;
```

Wertebereiche: Spannung: $\mp 10\,\mathrm{V}$, Steilheit: $\mp 10 \frac{\mathrm{A}}{\mathrm{V}^2}$, Strom: $\mp 10\,\mathrm{A}$

$$I_\mathrm{D} = K \cdot \begin{cases} 0 & U_\mathrm{GS} < U_\mathrm{th} \\ \frac{(U_\mathrm{GS}-U_\mathrm{th})^2}{2} & \text{sonst wenn } (U_\mathrm{GS} - U_\mathrm{DS} < U_\mathrm{th}) \\ (U_\mathrm{GS} - U_\mathrm{th}) \cdot U_\mathrm{DS} - \frac{U_\mathrm{DS}^2}{2} & \text{sonst} \end{cases}$$

Ohne Rücksicht auf Wertebereichs- und Typenfehler:

```
A1: if UGS<UTH then ID:=0;
A2: elsif UGS-UDS<UTH then
A3:   ID:=k/2*((UGS-UTH)**2/1E8)/1E8;
A4: else
A5:   ID:=k*((UGS-UTH)*UDS -(UDS**2)/2E8)/1E8;
A6: end if;
```

In A3 und A5:

- fehlen die Typkonvertierungen für die Multiplikation »tSteil« mit »tSpg« und vor der Zuweisung an »tStrom«.
- Wertebereichsüberlauf bei Produkten größer 10^9. Nach den Divisionen durch 10^8 max. eine gültige Dezimalstelle.

$$I_\mathrm{D} = K \cdot \begin{cases} 0 & U_\mathrm{GS} < U_\mathrm{th} \\ \frac{(U_\mathrm{GS}-U_\mathrm{th})^2}{2} & \text{sonst wenn } (U_\mathrm{GS} - U_\mathrm{DS} < U_\mathrm{th}) \\ (U_\mathrm{GS} - U_\mathrm{th}) \cdot U_\mathrm{DS} - \frac{U_\mathrm{DS}^2}{2} & \text{sonst} \end{cases}$$

Mit Typcasts und ausreichend großen WBs für Zwischenprodukte:

```
type tProd is -2E18 to 2E18;
variable tmp: tProd;
...
if UDS<UTH then ID:=0;
elsif UGS-UDS<UTH then
 tmp := ((tProd(UGS)-tProd(UTH))**2)/2E8;
else
 tmp := ((tProd(UGS)-tProd(UTH))*tProd(UDS)/2)/1E8;
 tmp := tmp - ((tProd(UDS))**2)/2E8;
end if;
ID := tStrom(tmp * tProd(k))/1E8);
```

Programmgeneratoren

- Mikrorechnersteuerungen für Motoren, Positionsreglungen von Robotern, ... werden in der Regel mit Gleitkommazahlen simuliert) und mit Festkommazahlen programmiert.
- Kontrolle auf / Ausschluss von Wertebereichsverletzungen für zulässige Ein- und Ausgaben für fertige Programme, insbesondere wenn sie Schleifen ohne feste Iterationsgrenzen enthalten aufwändig / nicht möglich.

Alternative:

- Entwicklung und Test des Algorithmus mit Gleitkommazahlen z.B. unter Matlab/Simulink.
- Automatische Generierung von korrektem Festkommacode statt manueller Programmierung und Kontrolle.

Statische Code-Analyse

Statische Code-Analyse

Untersuchung des Quellcodes auf Problemquellen:

- fehlerbegünstigend, Verständnis erschwerend,
- Uneindeutigkeiten,
- Programmkonstrukte, die Speicherlecks, Deadlocks, ... begünstigen,
- potentielle Sicherheitsrisiken,
- falsche Benutzung von Betriebssystemschnittstellen, ...

MISRA

Regeln für C-Programme:

- Bezeichnerlänge max. 31 Zeichen (längere Bezeichner werden von manchen Compilern nach 31 Zeichen abgeschnitten, Risiko, dass Compiler unterschiedliche Variablen zu einer zusammenfasst.
- Unterschiedliche Bezeichner für unterschiedliche Objekte:

```
int16_t i; {
 int16_t i; // Hier zwei Variablen i definiert.
            // Nach MISRA-Standard unzulässig.
 i = 3;     // Denn, welche ist hier gemeint?
}
```

- Jeder Variablen ist vor ihrer Nutzung ein Wert zuzuweisen, ...

Insgesamt über 100 Regeln, zum Teil verpflichtend, zum Teil Empfehlungen.

API-Benutzungsregeln

Beispielregeln für die Benutzung der Windows-API aus [1]:

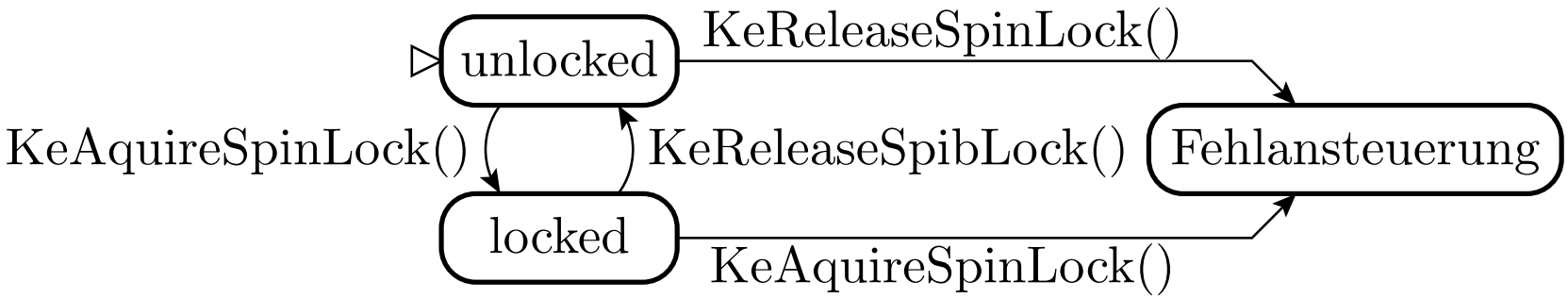
spinlock Spinlocks müssen alternierend reserviert und freigegeben werden.

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();
}
```

Sicherheitslücken

Die bekannteste Funktion, die Sicherheitslücken in C-Programmen verursacht, ist

```
char * strcpy(char *dest, char *src);
```

für Eingabezeichenketten ohne Längenkontrolle. Zu lange Zeichenketten überschreiben nachfolgende Variablen ...

Problemvermeidung durch statische Code-Analyse:

- Suche alle Aufrufe von strcpy (die Eingabedaten in Puffer kopieren).
- Ersatz durch

  ```
  char * strncpy(char *dest, char *src, int n);
  ```

  mit der zusätzlichen Übergabe der Puffergröße n.

Wie lässt sich durch Überschreiben von Daten hinter einem Zeichenkettenpuffer der Programmablauf manipulieren?

Baugruppen und CPS

Statische Tests für Baugruppen und CPS

Besonderheiten von Hardware und CPS (Cyper Physical Systems) gegenüber Software:

- Kontrolle physikalischer Größen.
- Beim Test defekter Systeme ist Schaden am System oder der Umwelt zu vermeiden.

Im weiteren behandelte statische Tests:

- MDA (Manufacturing Defect Analysis),
- Optische Inspektion,
- ICT (In Circuit Test) und
- dessen Nachbildung mit Boundary-Scan.

Inbetriebnahme

Inbetriebnahme elektronischer Baugruppen

Zur Vermeidung der Zerstörung von Bauteilen gelten bei uns im Labor und in den Praktika folgende Inbetriebnahmeregeln:

1. Sichtkontrolle im spannungsfreien Zustand.
2. Elektrische Verbindungskontrolle mit einem Durchgangsprüfer, Multimeter oder Tester ohne Betriebsspannung.
3. Rauchtest: Test mit Strombegrenzung und ständiger Kontrolle auf Erwärmung und Rauchentwicklung.
4. Funktionstests mit einem Labornetzteil mit eingestellter Strombegrenzung.

Während der Änderung / Fehlerbeseitigung an Schaltungen ist immer die Versorgungsspannung auszuschalten.

MDA

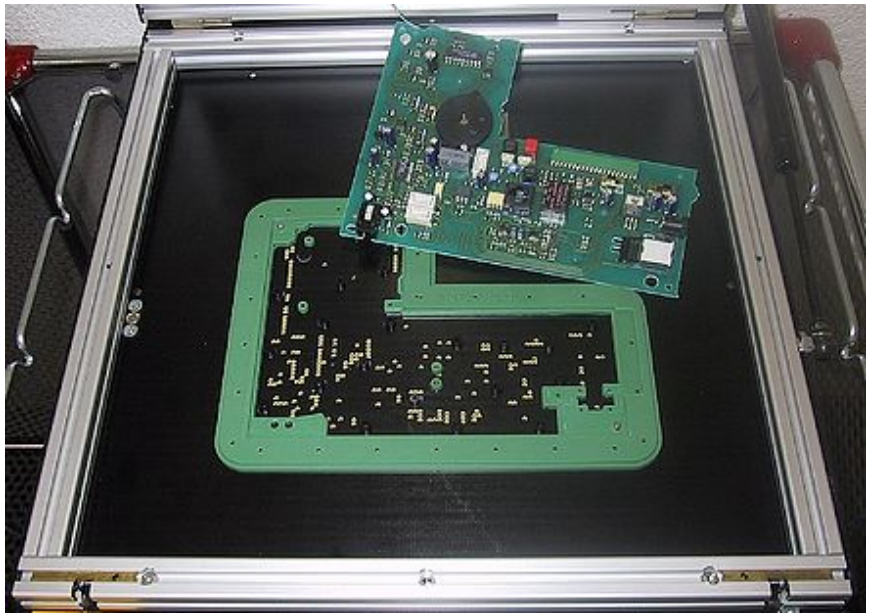
MDA (Manufacturing Defect Analyzer)

Prüfsysteme, die im spannungsfreien Zustand nach:

- Unterbrechungen,
- Kurzschlüssen und
- Fehlbestückungen

suchen.

Nadelbettadapter



In der Serienfertigung erfolgt die Kontaktierung für den Test auf Kurzschlüsse, Unterbrechungen und Bestückungsfehler mit einem mit Unterdruck angesaugten Nadeladapter.
Über die Nadeln werden Prüfgeräte angeschlossen.

Weitere Unterteilung der Prüfverfahren:

- MDA (Manufacturing Defect Analyzer),
- ICT (In-Circuit Test).

Weiterentwicklung im Zuge der zunehmenden Miniaturisierung:

- Boundary-Scan: »Silicon Nails« statt Nadeln.
- Selbsttest (BIST): Prüftechnikeinbau in die Schaltung.

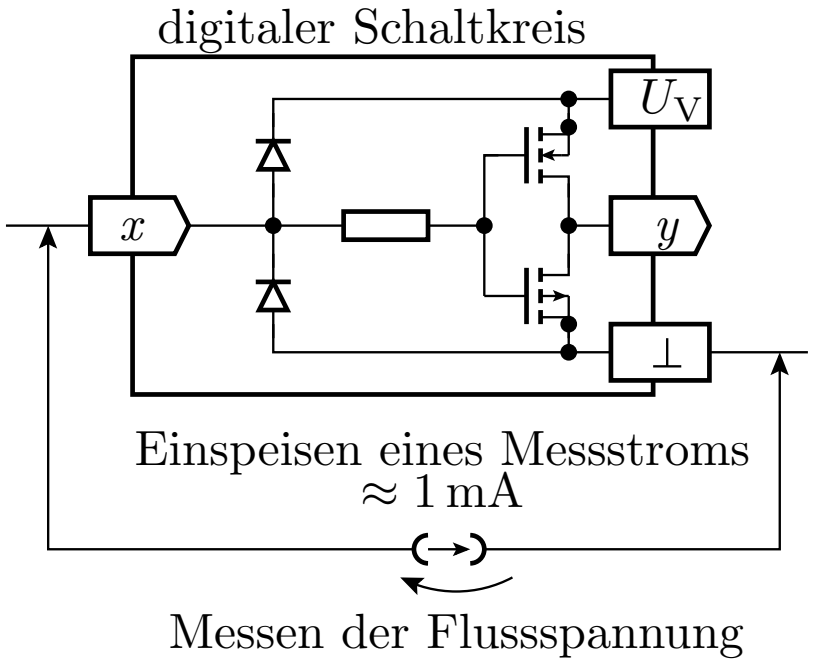
MDA

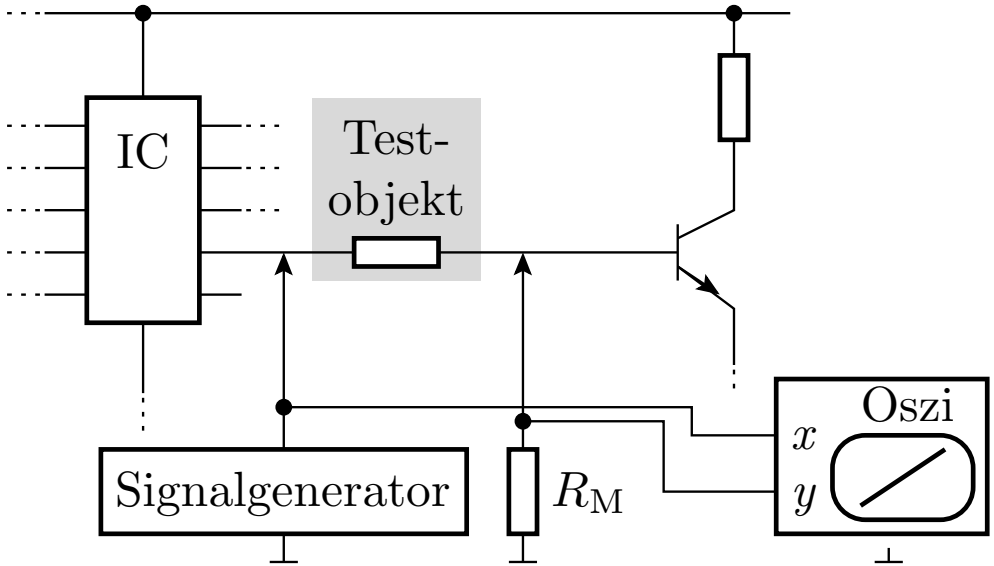
Suche potentieller Bestückungs- und Verdrahtungsfehler mit elektrischen Zweipunktmessungen:

- 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,
- ...



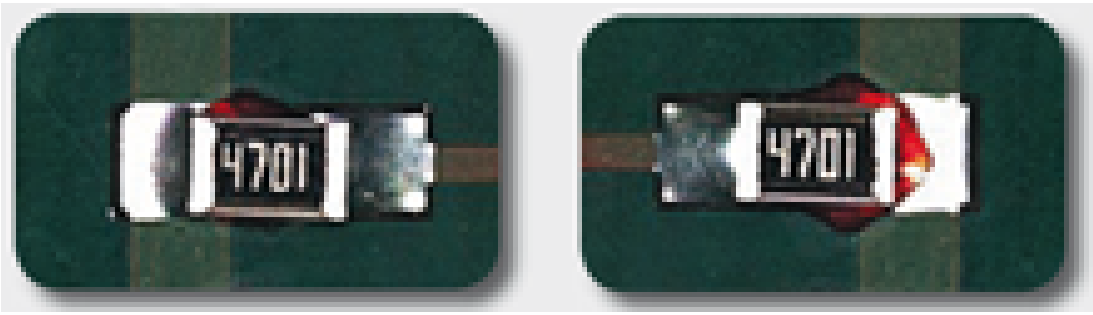


- Die Strom-Spannungs-Beziehung zwischen zwei Punkten in einer Schaltung hängt von der kompletten Schaltung, nicht nur einem einzelnen Bauteil ab.
- Bestimmbar durch Ausprobieren an einem »Golden Device«.
- Problematisch können sein
  - die Toleranzbereiche der Sollwerte unter Berücksichtigung der Bauteilstreuungen,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

Optische Inspektion

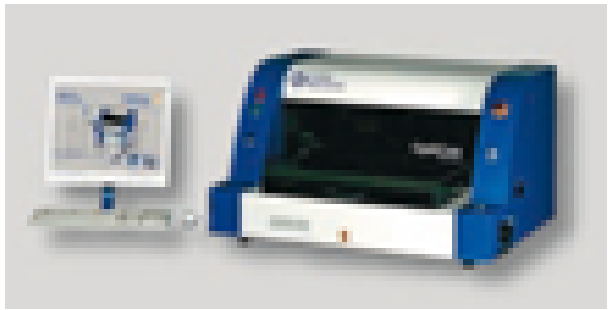
Optische Inspektion

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 (siehe Produkthaftung, Folie 13).

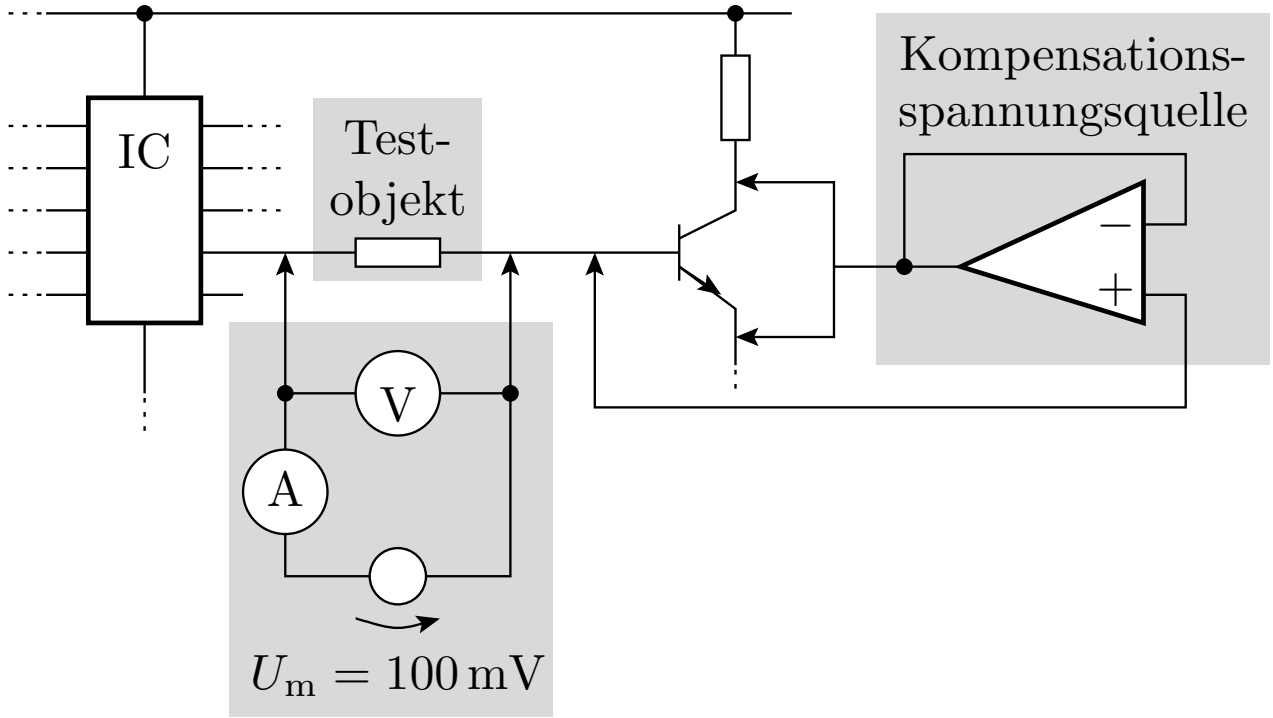
Automatisierte Baugruppeninspektion



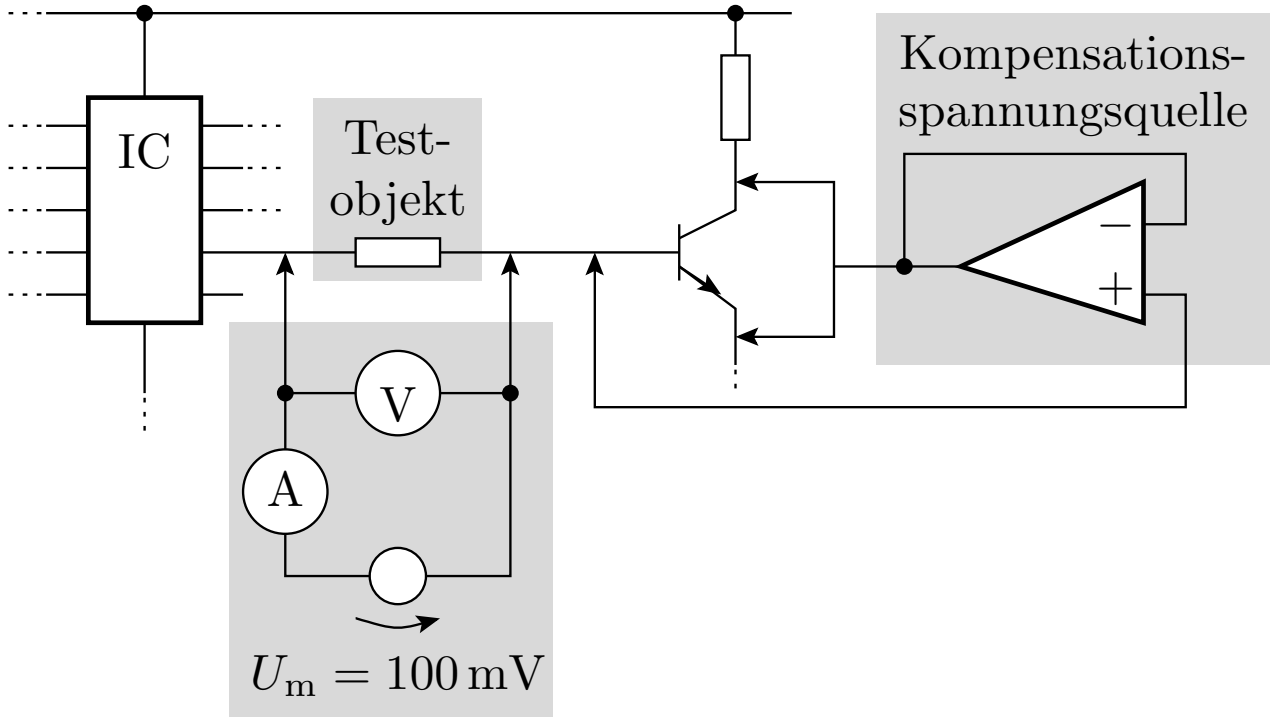
- Bildverarbeitungssystem mit Beleuchtung, Kamera, Verarbeitung, Monitor.
- Lernen von Bildern mit Fehlern und korrekten Bauteilen.
- Generierung des Prüfprogramms aus einer geometrischen Beschreibung und einer Bilddatenbank.
- Pflicht für sicherheitskritische Baugruppen (Automotive).

In-Circuit-Test

Analoger In-Circuit Test

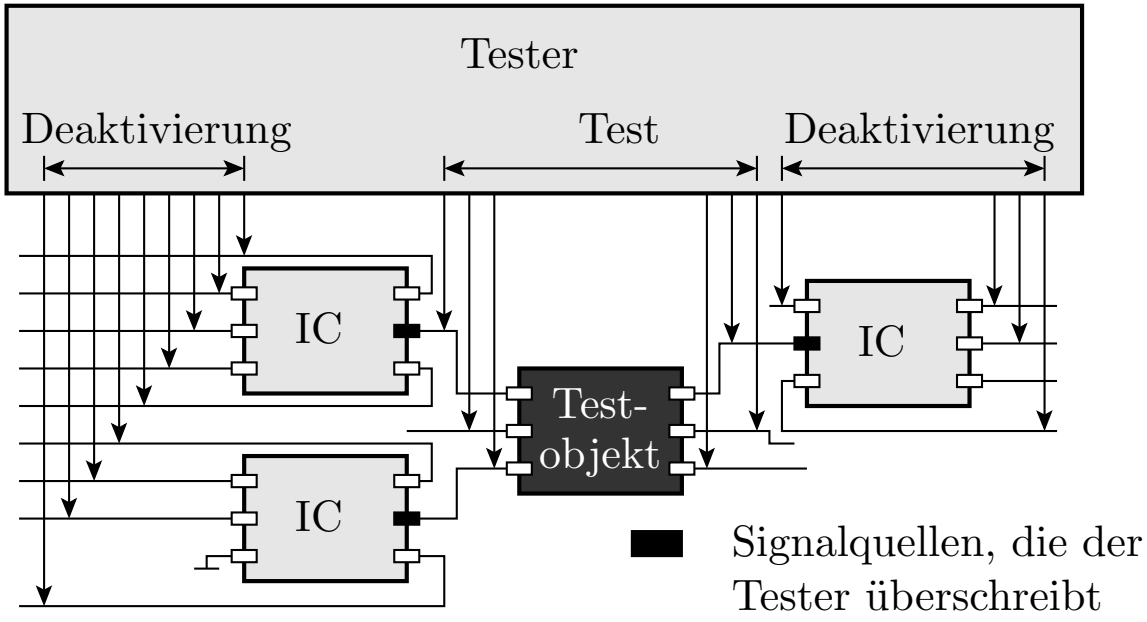


Unterdrückung von Parallelströmen zum Testobjekt durch Kompensation der Spannungsabfälle über den wegführenden Bauteilen auf einer Testobjektseite auf null. Erlaubt einen isolierten Zweipoltest.



- Vereinfacht die Testauswahl, Sollwertfestlegung, ...
- Mindert die Häufigkeit von Fehlklassifikationen.
- Für digitale Schaltkreise wenig geeignet.

Digitaler In-Circuit-Test



- Isolierter Test der Schaltkreise durch Überschreiben der digitalen Schaltkreiseeingaben mit stromstarken Treibern.
- Im Gegensatz zum analogen ICT Test unter Spannung.
- 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:

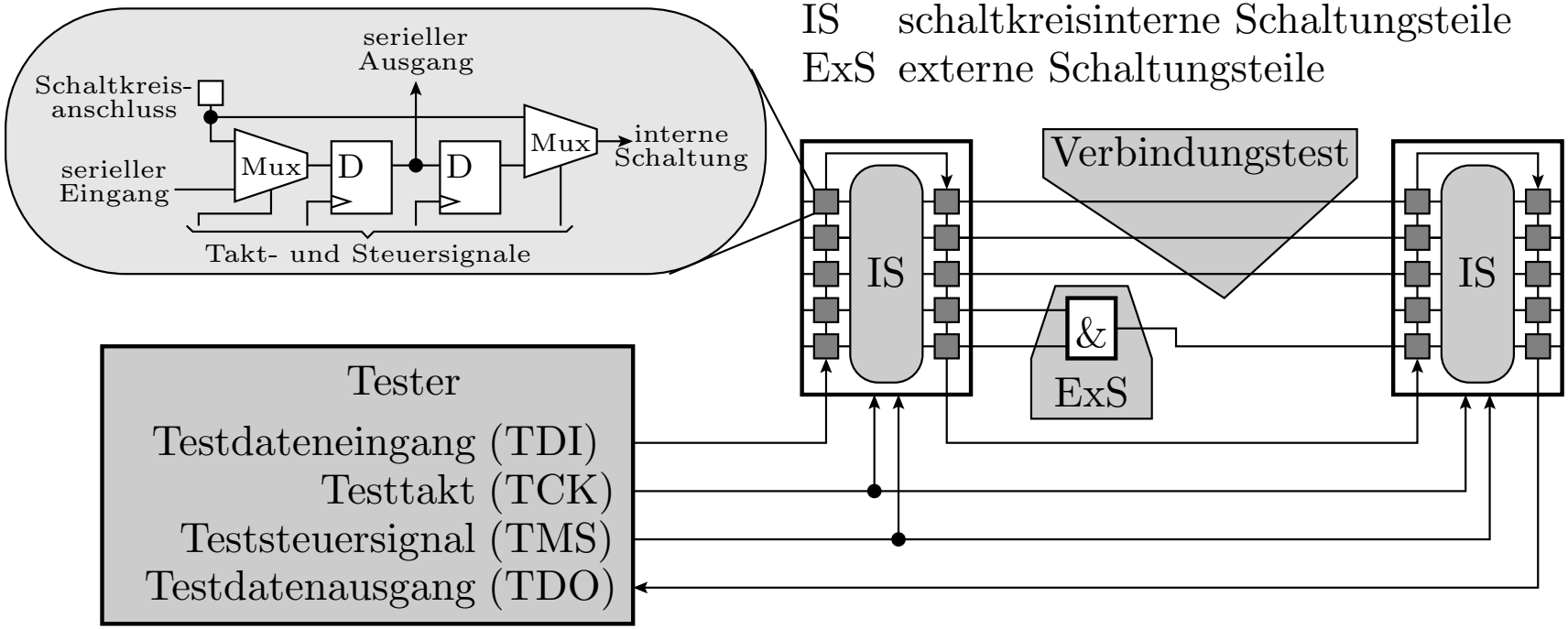
- Zusammensetzen aus Test- und Deaktivierungsvorschriften für alle Bauteile (gut gestelltes Problem).

Fehlererkennung und Lokalisierung:

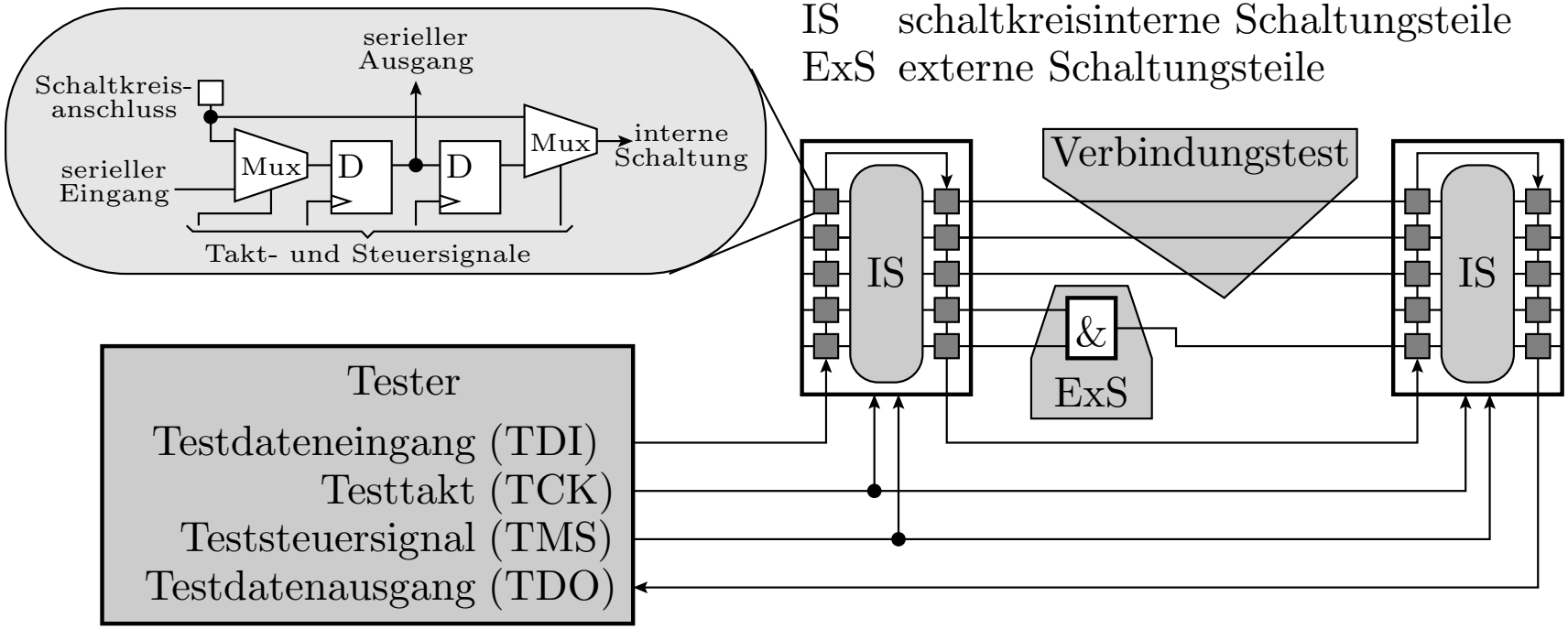
- Erkennt fast alle Kurzschlüsse, Unterbrechungen und Fehlbestückungen und gibt den genauen Fehlerort an.

Boundary-Scan

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:

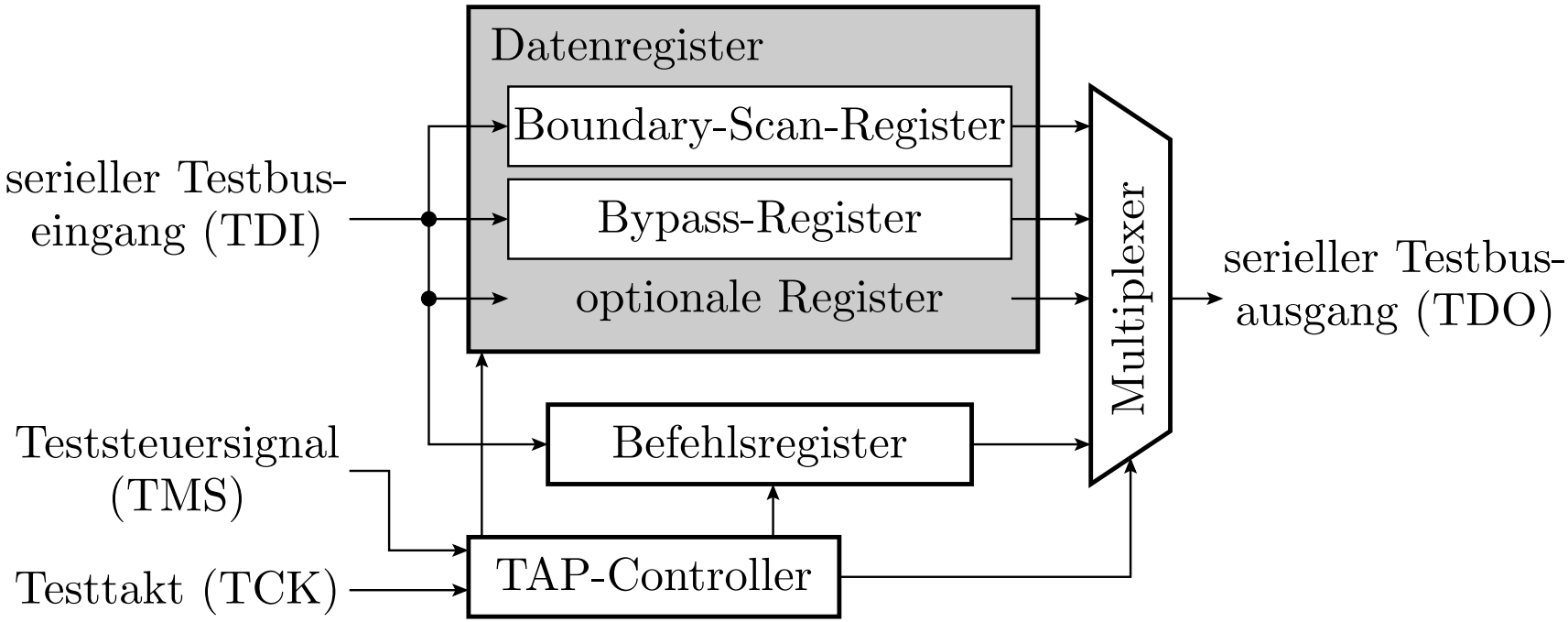
- BS-Register aller Schaltkreise auf der Baugruppe seriell beschreiben,
- einen Arbeitsschritt ausführen,
- die im Arbeitsschritt in den BS übernommenen Werte seriell an den Tester ausgeben und zeitgleich nächsten Eingabevektor laden.

- Ursprungsidee: Alternative zu den teuren, für jede Baugruppe speziell anzufertigenden Nadeladaptern.
- Als IEEE 1149.1 standardisierter serieller Testbus.

Nutzbar für weitere Test-, Diagnose- und Rekonfigurationsaufgaben, z.B. in den Laborübungen:

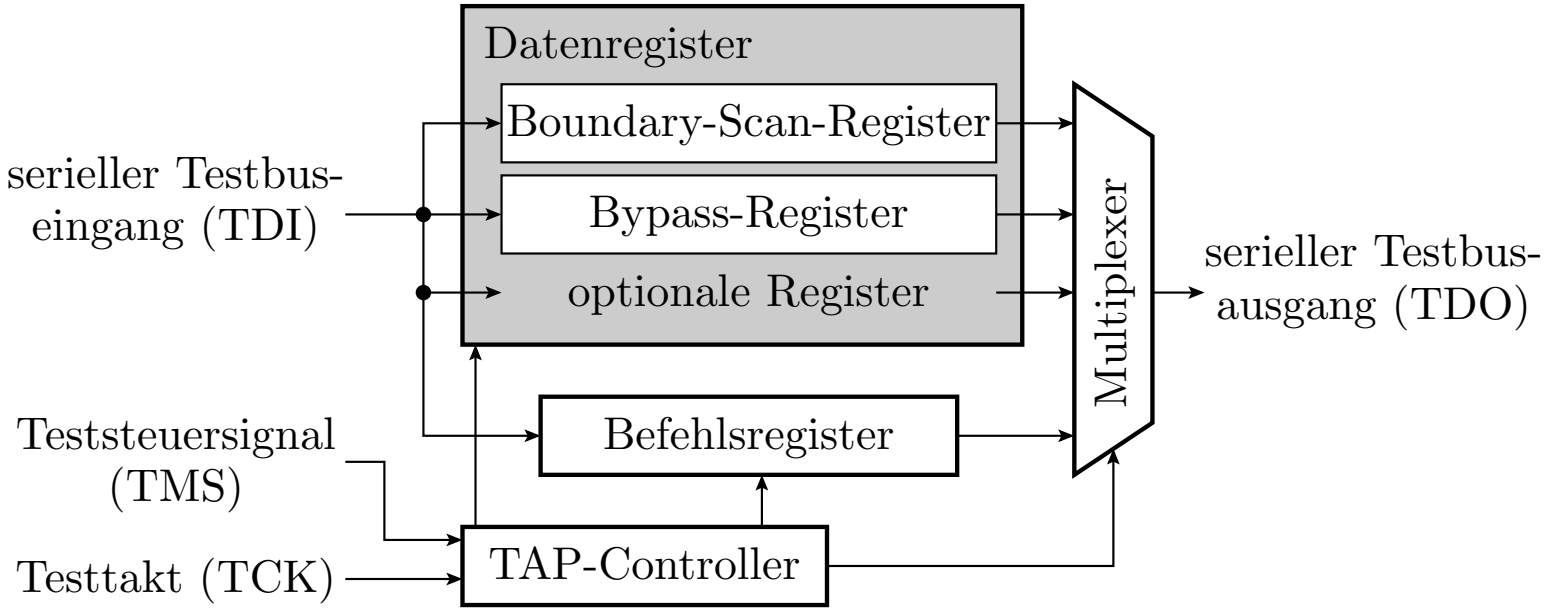
- Programmierung der Mikroprozessoren und FPGAs.
- Steuerung der In-Circuit-Debugger unter der AVR- und Microblaze-Entwicklungsumgebung.
- Steuerung des integrierten Logikanalysators Chip-Scope.

Die Testbusarchitektur der Schaltkreise



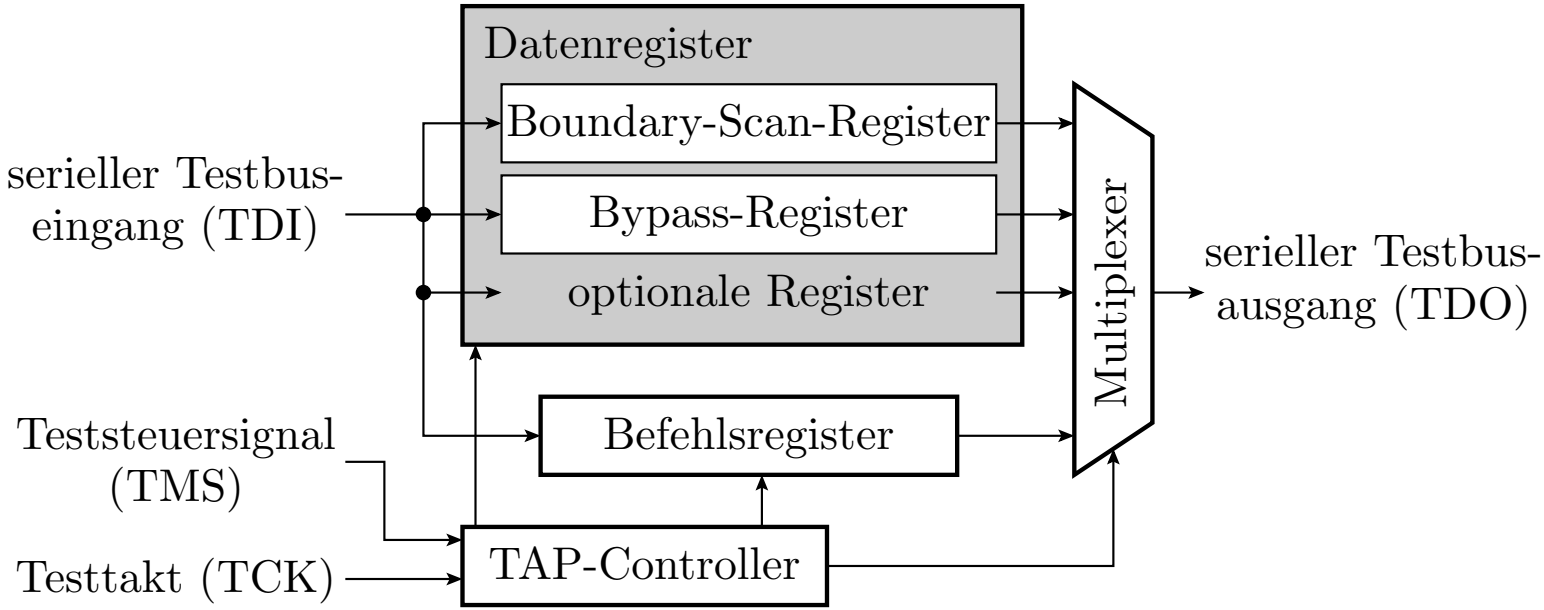
Eine Boundary-Scan-Implementierung umfasst:

- den TAP- (Test Access Port) Controller
- ein Befehlsregister
- mehrere Testdatenregister (mindestens das Boundary-Scan- und das Bypass-Register).



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:

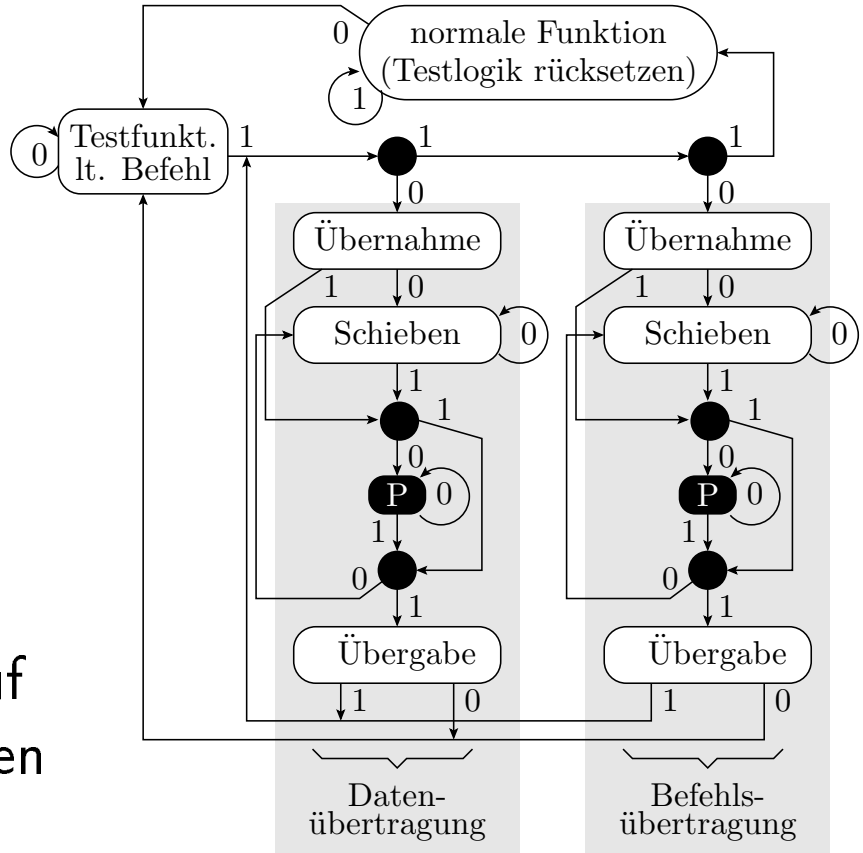
- Hersteller- und Bauteilidentifikationsregister
- weitere Test-, Programmier- oder Debug-Register

TAP-Controller, Busprotokoll

- Automat mit 16 Zuständen
- Kantenauswahl über TMS- (**T**est **M**ode **S**elect) Signal

Typischer Testablauf:

- Befehlsregister lesen (Bestückungskontrolle[8]),
- Bauteilnummern lesen (Bestückungskontrolle),
- Einen Teil der Schaltkreise auf Bypass setzen. Für die anderen Datenregister auswählen.
- Verbindungstest. ...



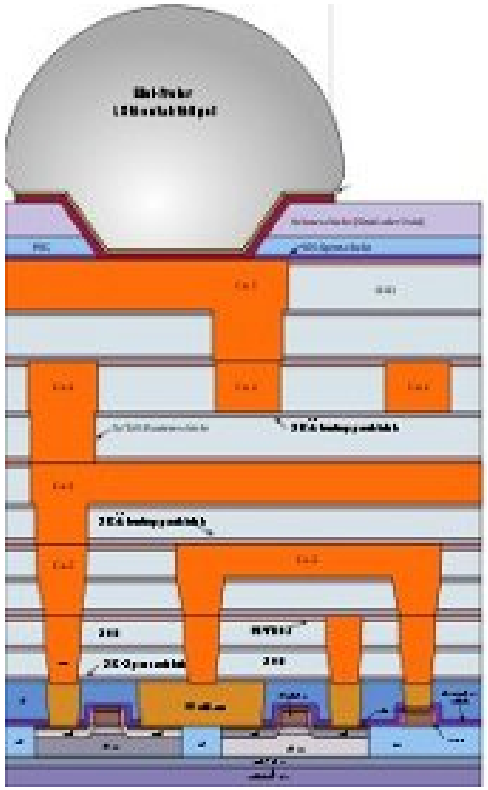
[8] Das Befehlsregister übergibt beim Lesen ein Muster zur Erkennung von Unterbrechungen der Schieberegisterkette auf der Baugruppe.

Statische Tests für Schaltkreise

Fertigungsfehler integrierter Schaltkreise

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.

Schaltkreisfehler und statische Tests

Einteilung der Schaltkreisfehler:

- Globale Fehler: Funktion großflächig beeinträchtigt. Prozesssteuerfehler, ...
- Lokale Fehler: defekte Einzelstrukturen.

Statische Schaltkreistests zielen vorrangig auf globale Fehler.

- Messen der Stomaufnahme (Versorgungsleitungen, Datenanschlüsse).
- Ausmessen spezieller hierzu gefertigter Teststrukturen.
- Stichprobenweises Messen von Schichteigenschaften (Dicke, Widerstand, ...).

Statische Tests erfolgen zum Teil zwischen den einzelnen Fertigungsschritten. Die herstellungsbegleitenden Tests erlauben:

- frühzeitiges Aussortieren fehlerhafter Zwischenprodukte,
- Erkennung / Lokalisierung Prozessfehler; Fehlervermeidung.

Literatur

5. Literatur

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

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

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

[4] Peter Liggesmeyer.
Software-Qualit"at: Testen, Analysieren und Verifizieren von Software.
Spectrum, 2002.

[5] 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.

[6] 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.