# Chiptest

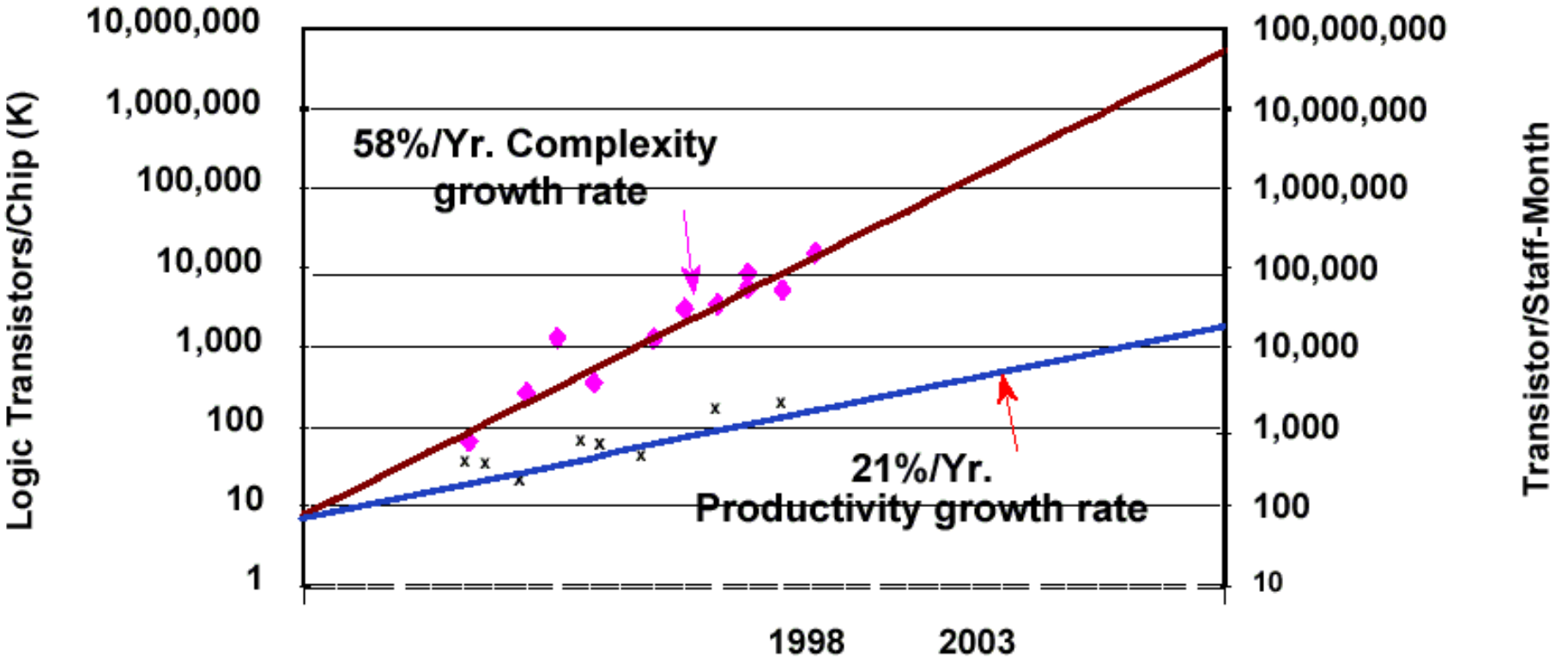
Tobias Ruf

10. Juli 2002

# Chiptest

- Motivation
- Fehler in digitalen Schaltungen
- Tests digitaler Schaltungen
- (Automatische) Testmustergenerierung
- Design for Testability
- Beispiele

# Motivation




**Design Complexity and Designer Productivity**

# Motivation

| | 1997 | 1998 | 1999 | 2002 |
|---|---|---|---|---|
| Process Technology | 0.35 micron | 0.25 micron | 0.18 micron | 0.13 micron |
| Cost of Fabrication | $1.5 - 2.0 billion | $2.0 - 3.0 billion | $3.0 - 4.0 billion | $4.0 ---- billion |
| Design Cycle | 18-12 months | 12-10 months | 10-8 months | 8-6 months |
| Derivative Cycle | 8-6 months | 6-4 months | 4-2 months | 3-2 months |
| Silicon Complexity | 200-500k gates | 1-2M gates | 4-6M gates | 10-25M gates |
| Applications | Cellular, PDA, DVD | Set-top boxes wireless PDA | Internet appliances | Ubiquitous computing |
| Primary IP Sources | Intragroup | Intergroup | Intercompany | Intercompany Interindustry |

# Kosten für Testequipment

- Anschaffungskosten, 0.5-1.0 GHz, 1024 Pins

  = 1,2M $ + 1024 * 3000 $ = 4,272M $

- Nutzungskosten pro Jahr (5 Jahre Laufzeit)

  = Verschleiß + Wartung + Bedienung

  = 0,854M $ + 0,085M $ + 0,5M $ = 1,439M $

- Nutzungskosten

  = 1,439M $ / (365 * 24 * 60 * 60) s = 4.5 cents / s

# Chiptest

- Motivation
- Fehler in digitalen Schaltungen
- Tests digitaler Schaltungen
- (Automatische) Testmustergenerierung
- Design for Testability
- Beispiele

# Fehler

- Eine Schaltung oder ein System ist fehlerhaft, wenn sie von ihrem spezifizierten Verhalten abweicht
- Ein Hardware-Fehler ist ein physikalischer Defekt, der ein solches Versagen verursachen kann
- Ein Test ist ein Experiment an der Schaltung zur Suche eventuell vorhandener Fehler

# Fehlerursachen

- Physikalische Fehler können durch Designfehler oder Fertigungsfehler verursacht werden
- Designfehler treten durch unvollständige oder inkonsistente Spezifikationen oder durch Verletzung von Designregeln auf
  - Übersprechen zwischen gegenseitig nicht abgeschirmten Leiterbahnen
- Fertigungsfehler
  - Unterbrechung oder Kurzschluß von Leiterbahnen
  - Fehler durch defekte Bauteile

# Fehlerursachen

  - falsche Bestückung
  - Lötfehler

- Nach der Fertigung können weitere Fehler auftreten
  - elektrische Entladungen beschädigen CMOS-Bauteile
  - thermische oder elektrische Überlastung
  - Steckverbindungen mit mangelhaftem Kontakt

# Charakterisierung von Fehlern

- Eigenschaft
  - Logisch
  - Nichtlogisch, z.B. fehlerhaftes Taktsignal
- Spannungswert
  - feste Logikwerte
  - variierende / undefinierte Logikwerte
- Ausdehnung
  - lokal, Fehler betrifft nur eine Variable
  - verbreitet, Fehler betrifft mehrer Variablen, z.B. durch fehlerhaftes Taktsignal

# Charakterisierung von Fehlern

- Dauer
  - permanent
  - temporär, z.B. durch Wackelkontakte oder Versorgungsstörungen
- Ort des Auftretens
  - in einem Bauteil
  - auf der Platine
- statische, dynamische Fehler
  - statisch
  - dynamisch, z.B. zu flache Signalflanken, Reflexionen am Leitungsende, unterschiedliche Signallaufzeiten

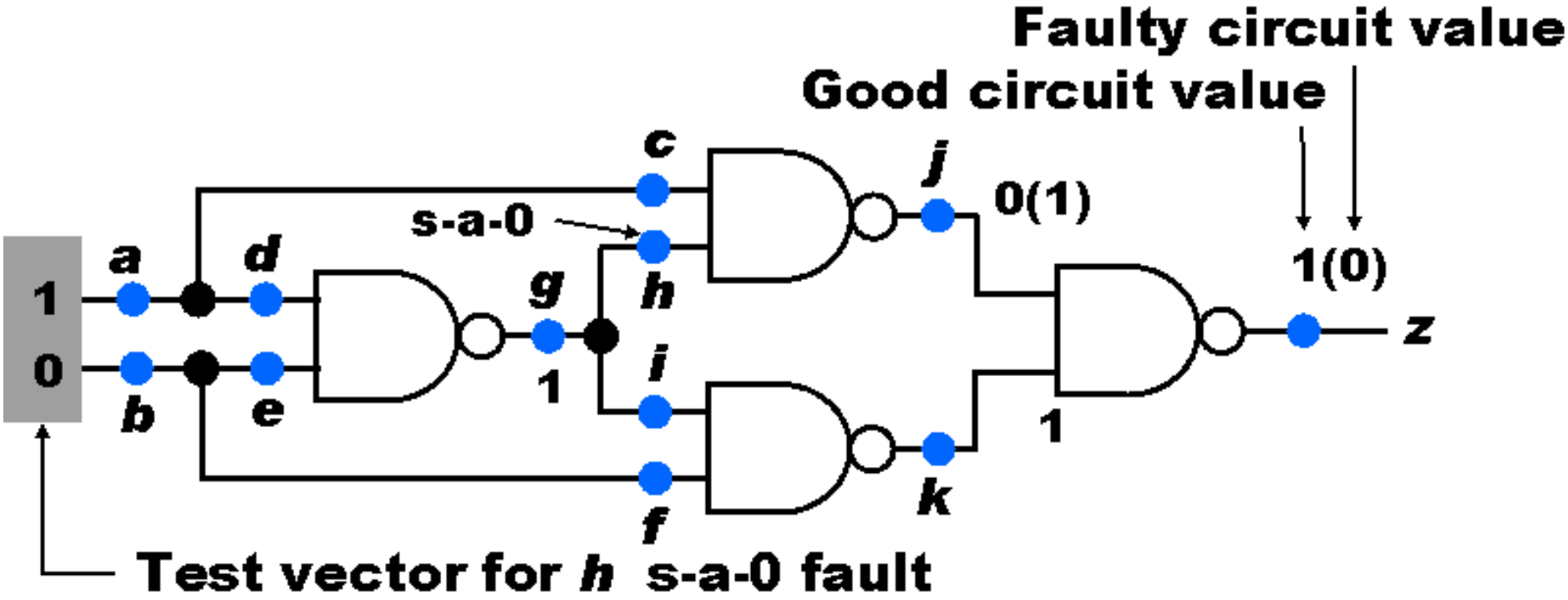
# Fehlermodelle

- Fehlermodelle beschreiben den logischen Fehler der durch physikalische Fehler verursacht wird
  - Vereinfachung komplexer physikalischer Fehler
  - verschiedene physikalische Ursachen durch ein Modell beschreibbar
  - Modelle können technologieunabhängig sein

# Stuck-At-Fehlermodell

- allgemeinstes Fehlermodell
- Knoten der Schaltung auf Logikwert 0 (stuck-at-0) oder 1 (stuck-at-1) fixiert
- respräsentiert beispielweise Kurzschlüße nach Plus oder Masse in vielen Technologien
- Vorteil der Einfachheit da jeder Knoten maximal 2 Fehler
- Verhalten einer fehlerhaften Schaltung ist logisch, kann durch veränderte Bool'sche Gleichung beschrieben werden

# Stuck-At-Fehlermodell

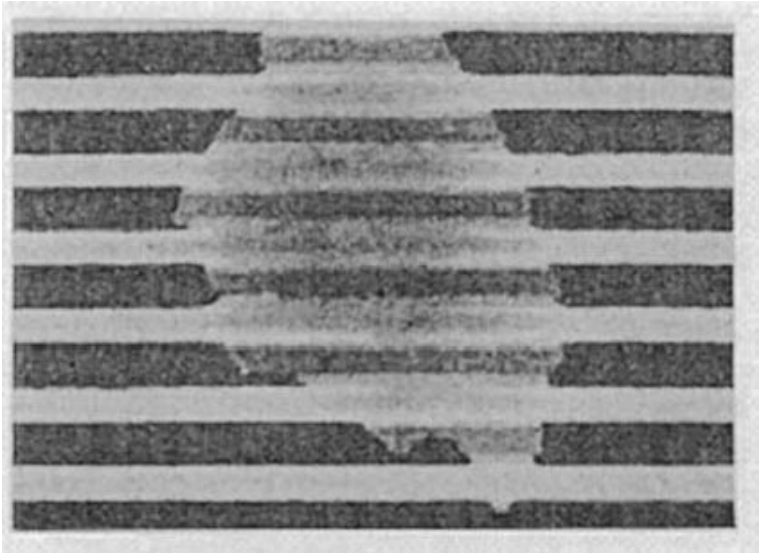


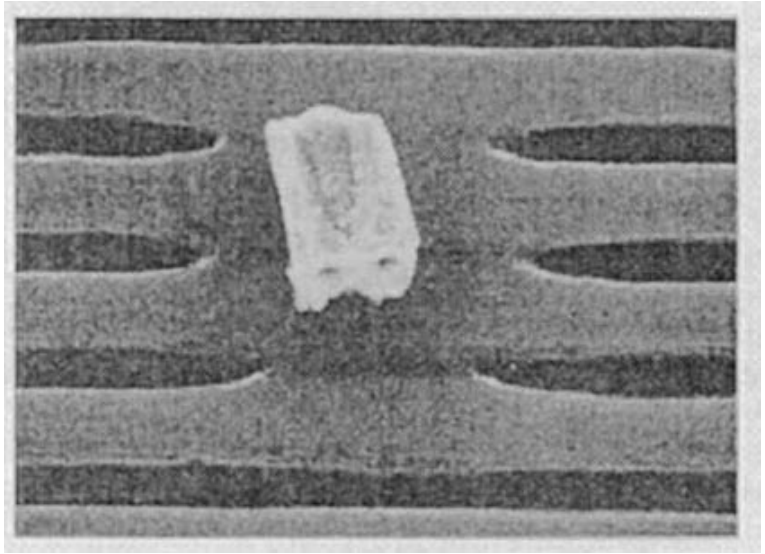
# Einzelfehlerannahme

- Um Fehlermodelle, z.B. Stuck-At, einfacher mathematisch behandelbar zu machen, geht man von nur einem gleichzeitig auftretenden Fehler aus (engl. Single Stuck-At-Fault)

# Kurzschluss-Fehlermodell (engl. Bridging Fault)

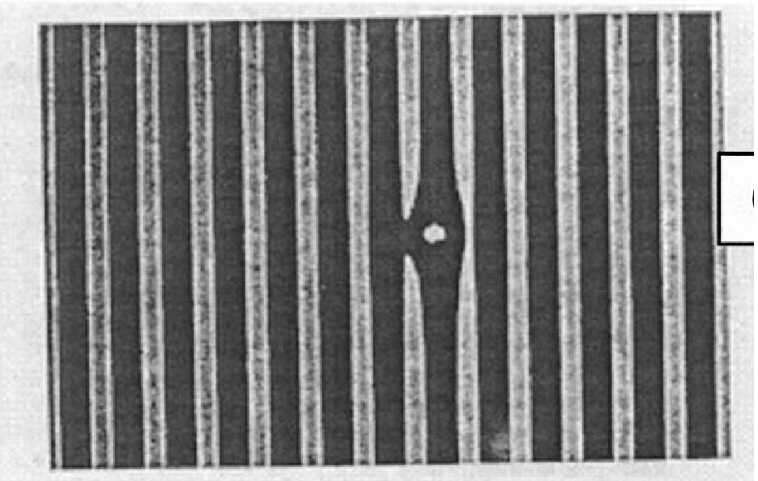
- Erweiterung des Stuck-At-Modells
- Berücksichtigt zusätzlich Kurzschlüsse zur Spannungsversorgung oder zwischen Signalleitungen

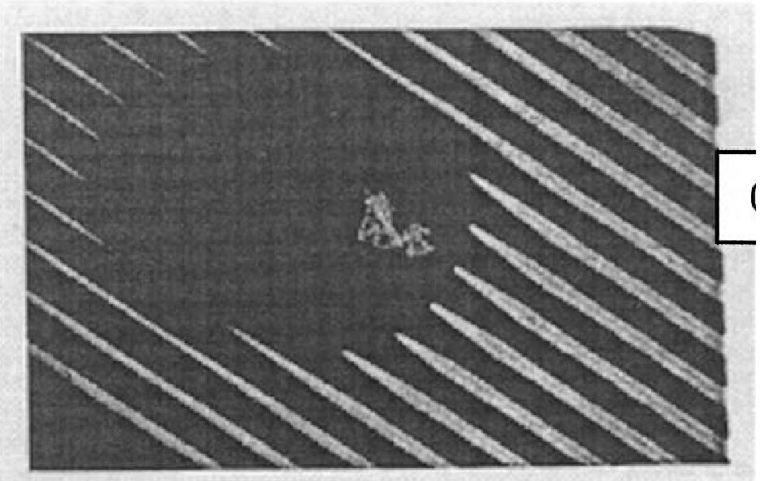




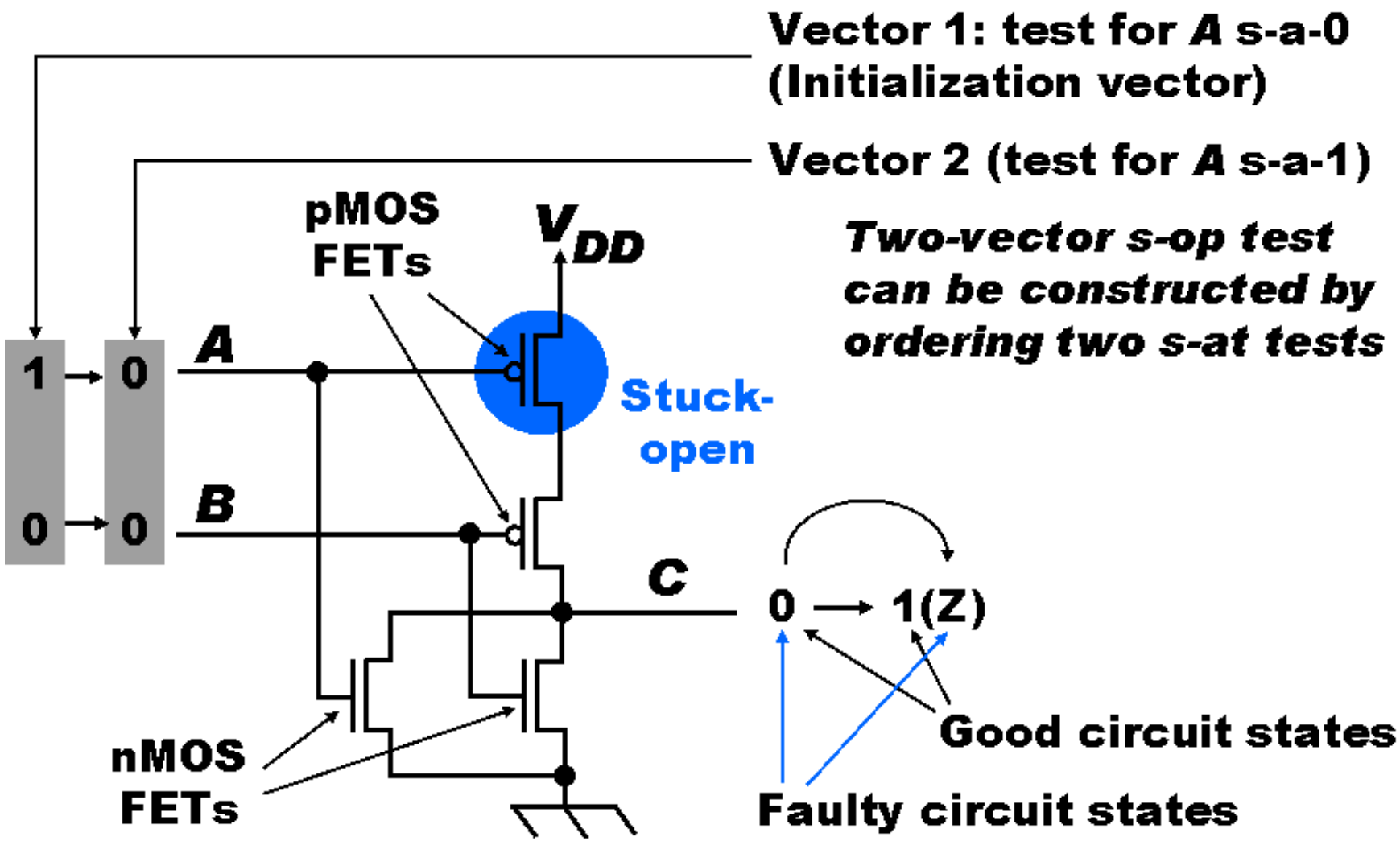
# Unterbrechungen (engl. open)

- beinhaltet Brüche, fehlende oder nur teilweise vorhandene Verbindungen oder Verbindungen mit größerem Widerstand
- kann zu Delay-Fehlern oder Stuck-At-Fehlern führen

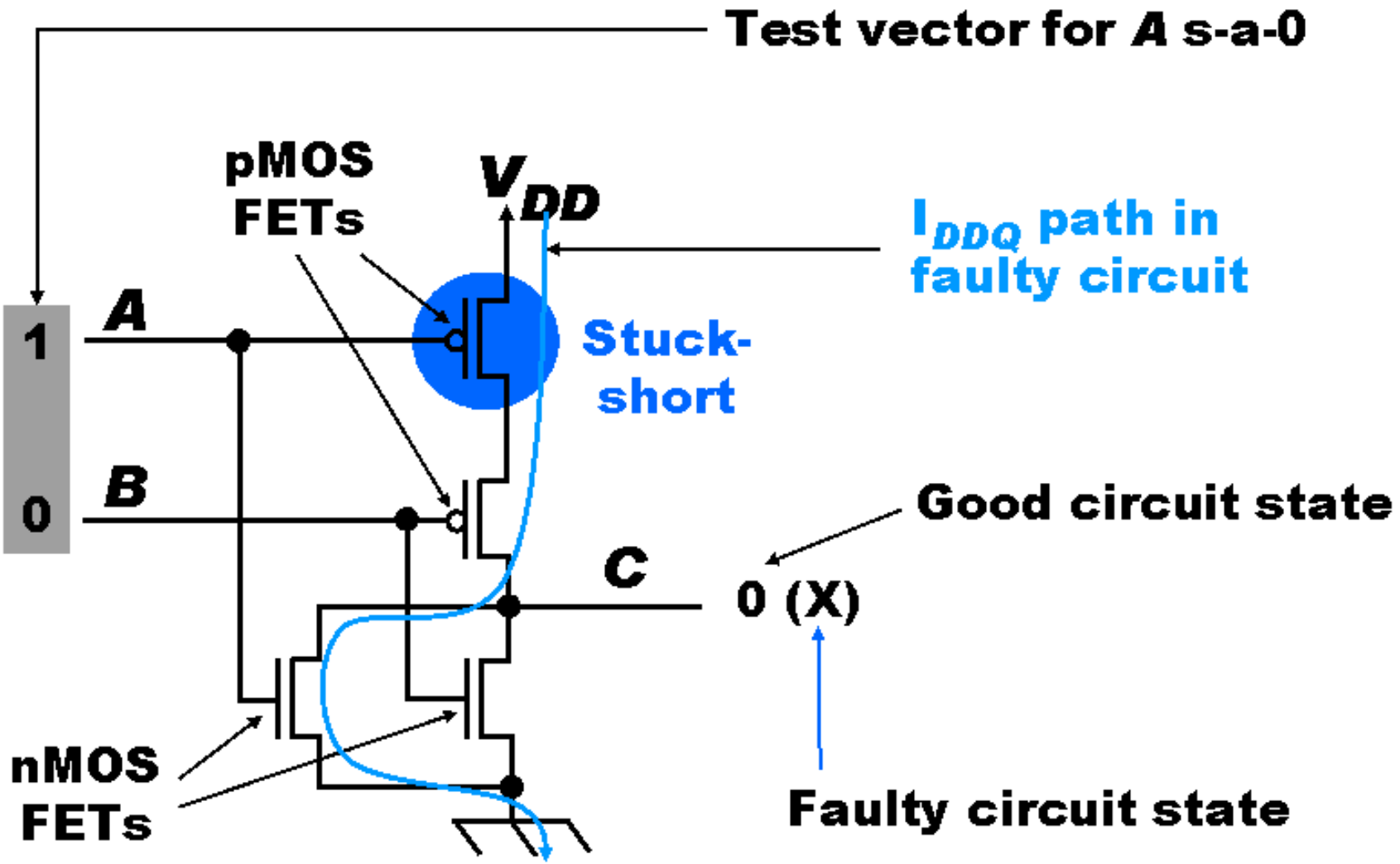




# nichtleitende Transistoren (engl. Stuck-Open)



# leitende Transistoren (engl. Stuck-Short)



# Delay-Fehler

- Verändertes Zeitverhalten der Schaltung
- Logikverhalten korrekt, Wechsel zwischen Logiksignalen verzögert

# Chiptest

- Motivation
- Fehler in digitalen Schaltungen
- Tests digitaler Schaltungen
- (Automatische) Testmustergenerierung
- Design for Testability
- Beispiele

# Tests

- Der Test digitaler Schaltungen dient der Detektion und Lokalisierung von Fehlern
- Ein Test besteht aus einer Reihe von Testsignalen und Messung der resultierenden Ausgangssignale, diese werden mit Ergebnissen der Fehlersimulation verglichen
- Unterschiedliche Anforderungen
  - Prototypentest
  - Produktionstest
  - Wartungstest

# Kosten eines Tests

- zu den Kosten zählen die Erzeugung von Testsignalen und die Durchführung des Tests
- Ziel ist maximale Fehlerabdeckung mit möglichst wenig Testmustern
- für 50 Eingänge bei 50 MHz erfordert das Anlegen aller $2^{50}$ Kombinationen ca. 260 Tage
- Kosten eines Tests müssen in Relation zu den Kosten eines nicht gefundenen Fehlers gesetzt werden

# Kontrollierbarkeit und Observierbarkeit

- ein Knoten ist kontrollierbar, wenn durch Anlegen bestimmter Eingangsmuster sein Logikwert überprüft werden kann
- ein Knoten ist observierbar, wenn aus dem Ausgangsmuster der Schaltung eindeutig auf seinen Logikpegel geschlossen werden kann
- Kontrollierbarkeit und Observierbarkeit können durch Einfügen von Kontrollleitungen erhöht werden ($\rightarrow$ Design for Testability)

# Kontrollierbarkeit und Observierbarkeit

- redundante Logik und sequentielle Schaltungen führen zu geringerer Observierbarkeit

# Qualität eines Tests

- Simulation aller möglichen Fehler im Kontext eines Fehlermodells mit Testmustersequenz
- Fehlerabdeckung $= \frac{\text{durch Sequenz entdeckbare Fehler}}{\text{potentielle Fehler im Fehlermodell}}$
- Fehlerabdeckung kann durch Erweiterung der Testmustersequenz auf Kosten einer längeren Testzeit erhöht werden

# Generierung von Tests

- Strukturelle Tests
  - von Struktur der Schaltung ausgehend
  - Beschreibung beispielsweise in Form von Funktionsgleichungen
- Funktionstests
  - basierend auf funktionellem Schaltungsmodell, unabhängig von Implementierung
  - dadurch auch Design-Verifikation

# Generierung von Tests

- exhaustive oder pseudoexhaustive Testmuster
  - exhaustiver Test eines Schaltnetz mit n Eingängen durch Anlegen aller möglichen $2^n$ Testmuster
  - pseudoexhaustiver Test durch Unterteilung der Schaltung in Unterschaltungen, z.B. Steuer- und Dateneingänge, und exhaustiver Test dieser

- Tests mit Pseudo-Zufallszahlen
  - erfordert längere Testmustersequenz als deterministisch generierte Tests
  - manche Eingänge (Reset-Signal) müssen ausgenommen werden
  - typischerweise am Anfang des Testprozesses verwendet, um einfach zu findende Fehler aus der Fehlerliste zu streichen

# Generierung von Tests

- Automatische Testgenerierung (ATG)
  - Erzeugung von Tests für komplexe Designs kann sich auf mehrere Monate erstrecken
  - unter dem Begriff „Automatische Testgenerierung“ versteht man Werkzeuge zur Entschärfung dieses Engpasses
  - eines dieser Werkzeuge ist die “Automatische Testmustergenerierung“
  - Fehlersimulation, automatisierte Prozesse zum Überprüfen und Verbessern des Designs sind andere Beispiele

# Chiptest

- Motivation
- Fehler in digitalen Schaltungen
- Tests digitaler Schaltungen
- (Automatische) Testmustergenerierung
- Design for Testability
- Beispiele

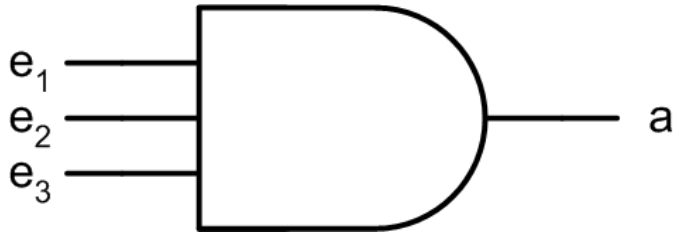
# Varianten

- Aufbau von Fehlermatrizen
  - für jeden Fehler Zuordnung zwischen allen Eingangssignalen und den zugehörigen Ausgangssignalen
  - Suche einer minimalen Testmenge
- Auswertung der Funktionsgleichung einer Schaltung
  - Abhängigkeiten von Ausgängen und Eingängen werden gesucht
  - Fehlersuche durch Variation der entsprechenden Eingänge
  - Bool'sche Differenz

# Varianten

- Pfadsensibilisierung
  - Suche nach Pfaden durch die Schaltung, um die Knoten auf diesen zu testen
  - D-Algorithmus
  - PODEM-Algorithmus

# Fehlermatrix




| Nr. | $e_1$ | $e_2$ | $e_3$ | $a$ | $s0_{e_1}$ | $s0_{e_2}$ | $s0_{e_3}$ | $s1_{e_1}$ | $s1_{e_2}$ | $s1_{e_3}$ | $s0_a$ | $s1_a$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

# Fehlermatrix

| Nr. | $e_1$ | $e_2$ | $e_3$ | $a$ | $s0_{e_1}$ | $s0_{e_2}$ | $s0_{e_3}$ | $s1_{e_1}$ | $s1_{e_2}$ | $s1_{e_3}$ | $s0_a$ | $s1_a$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

| Nr. | $e_1$ | $e_2$ | $e_3$ | $a$ | $s0_{e_1}$ | $s1_{e_1}$ | $s1_{e_2}$ | $s1_{e_3}$ | $s1_a$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

# Fehlermatrix

| Nr. | $e_1$ | $e_2$ | $e_3$ | $a$ | $s0_{e_1}$ | $s1_{e_1}$ | $s1_{e_2}$ | $s1_{e_3}$ | $s1_a$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

| Nr. | $e_1$ | $e_2$ | $e_3$ | $a$ | $s0_{e_1}$ | $s1_{e_1}$ | $s1_{e_2}$ | $s1_{e_3}$ | $s1_a$ |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

minimale Testmustermenge {{0,1,1},{1,0,1},{1,1,0},{1,1,1}}

# Bool'sche Differenz

- Versuch Änderungen eines Eingangssignals an einem Ausgang sichtbar zu macheen
- Basierend auf Bool'schen Funktionsgleichungen für die Ausgänge
- $f = \overline{b} + a * c$

  $f_a = f_{|a=0} \oplus f_{|a=1} = (\overline{b}) \oplus (\overline{b} + c) = b * c$

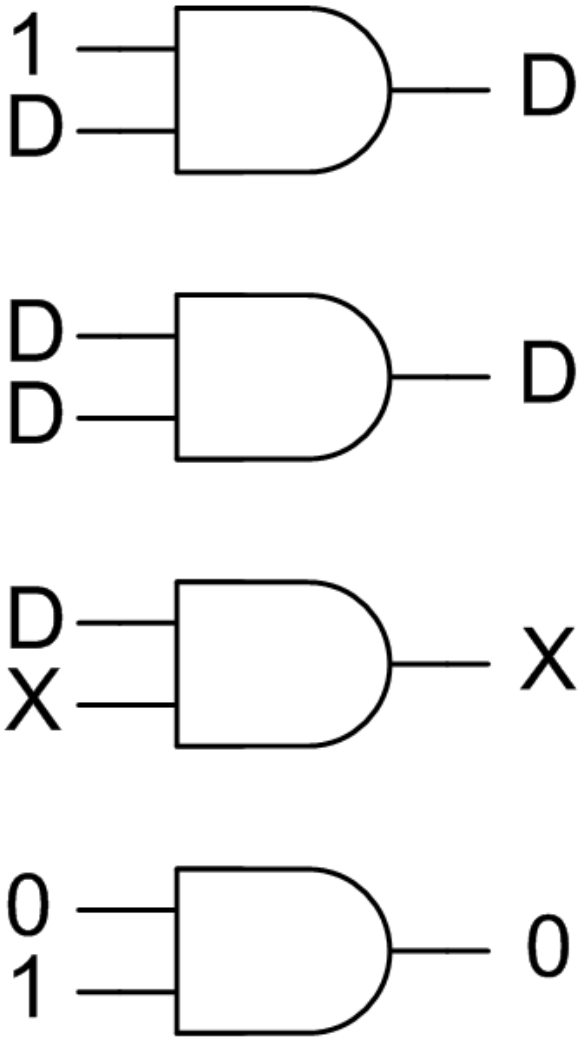
  $f_b = f_{|b=0} \oplus f_{|b=1} = (1 + a * c) \oplus (a * c) = \overline{a} + \overline{c}$

  $f_c = f_{|c=0} \oplus f_{|c=1} = (\overline{b}) \oplus (\overline{b} + a) = a * b$

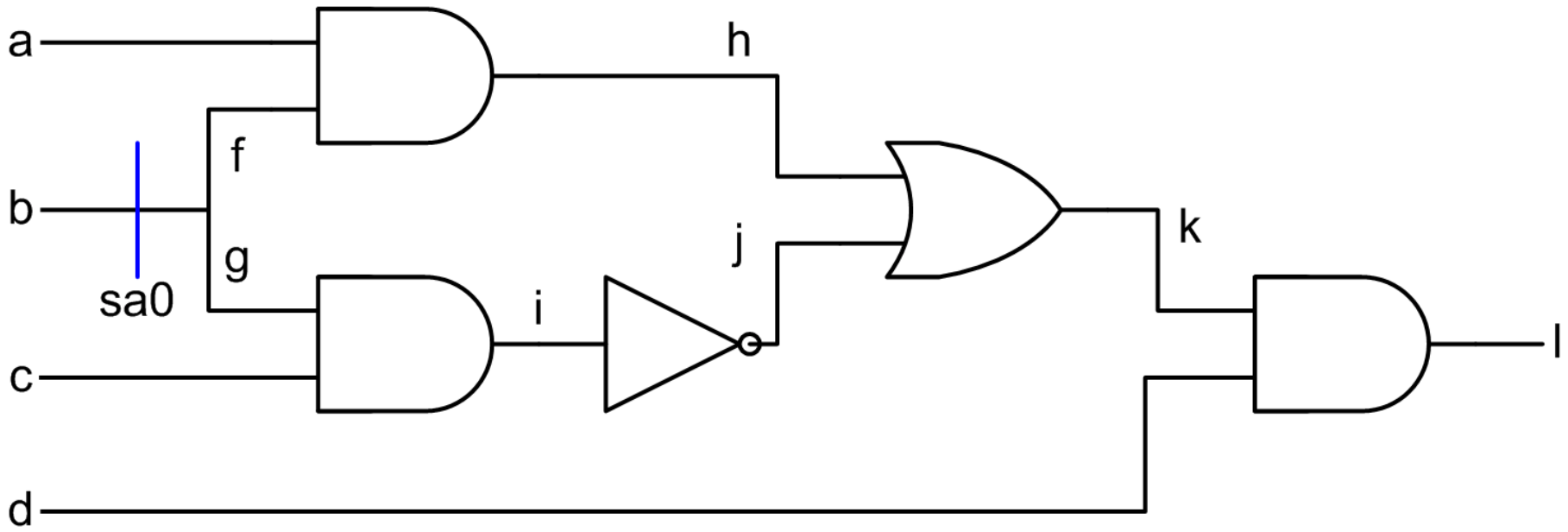
# 5-wertige D-Algebra nach Roth

| Wert | Wert ohne Fehler | Wert bei Fehler |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| $D$ | 1 | 0 |
| $\overline{D}$ | 0 | 1 |
| X | ? | ? |

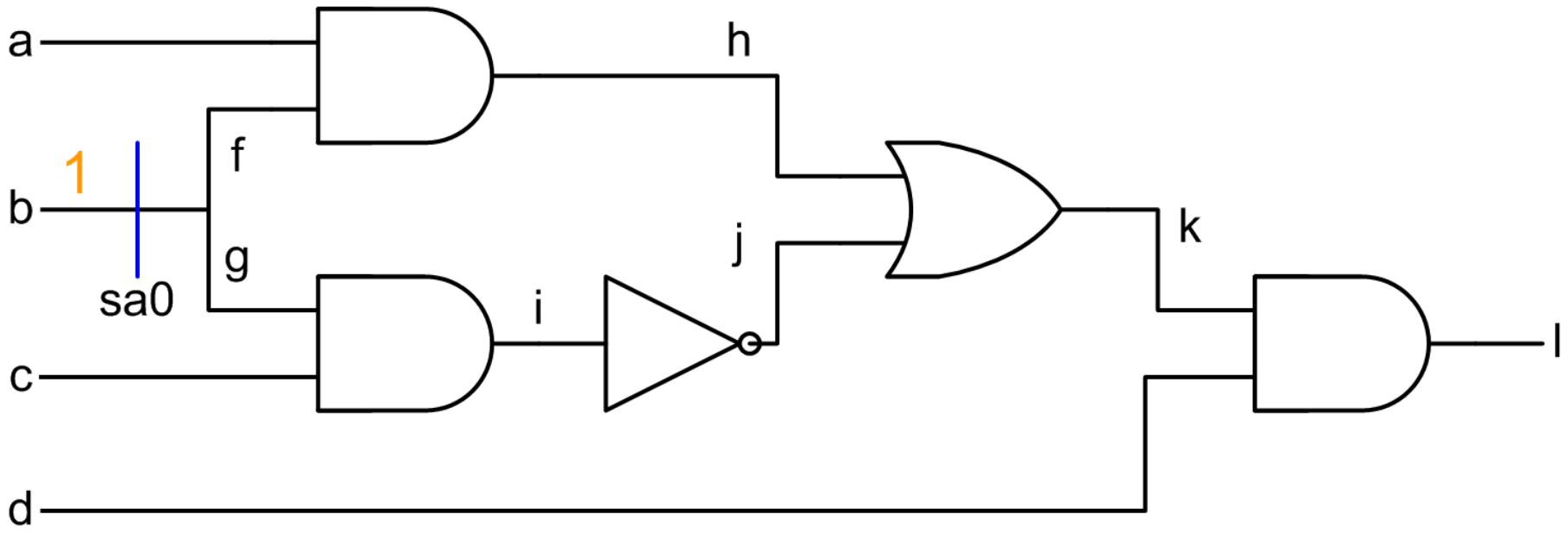
# 5-wertige D-Algebra nach Roth



# Pfadsensibilisierung

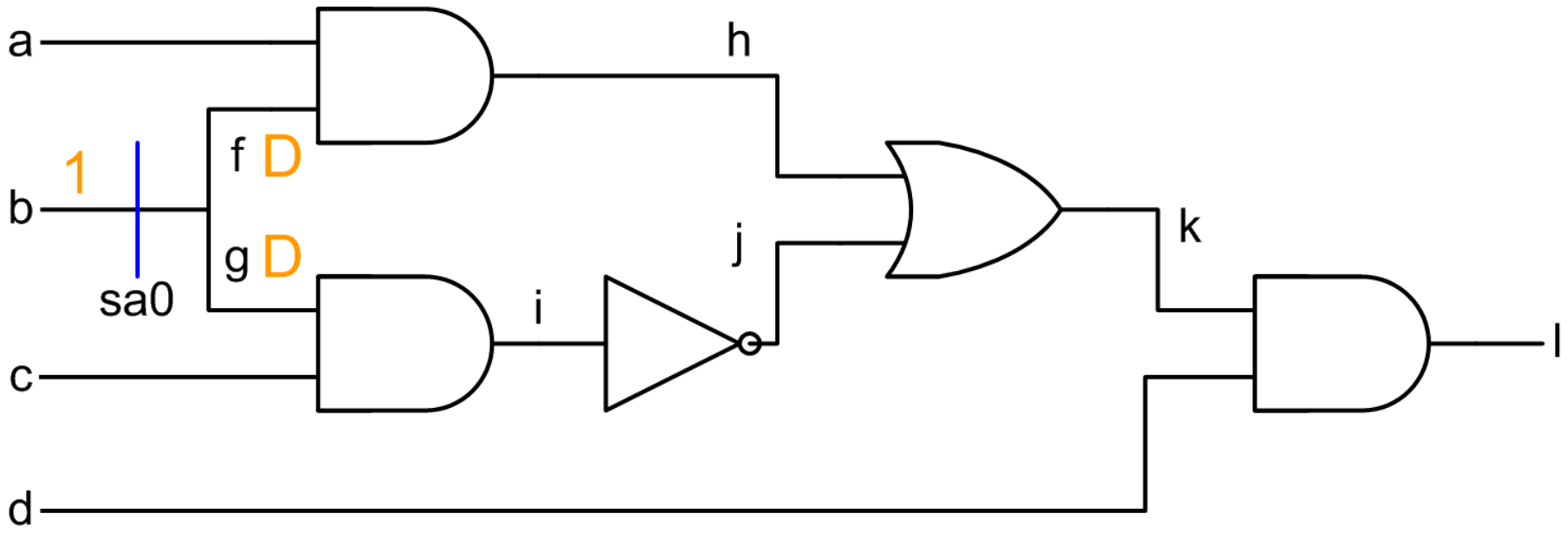


# Pfadsensibilisierung



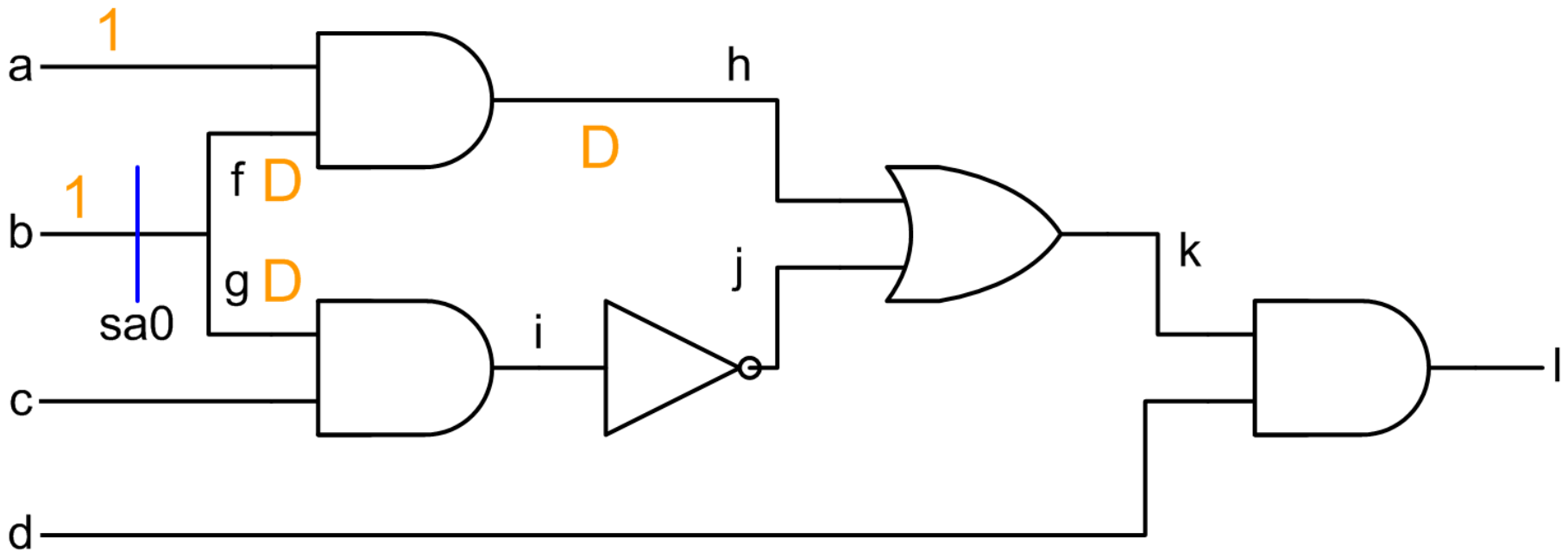

Um Stuck-At-0-Fehler sichtbar zu machen, muß Knoten b mit logischem Wert 1 belegt werden (fehlerinjizierende Belegung, engl. fault sensitization)

# Pfadsensibilisierung



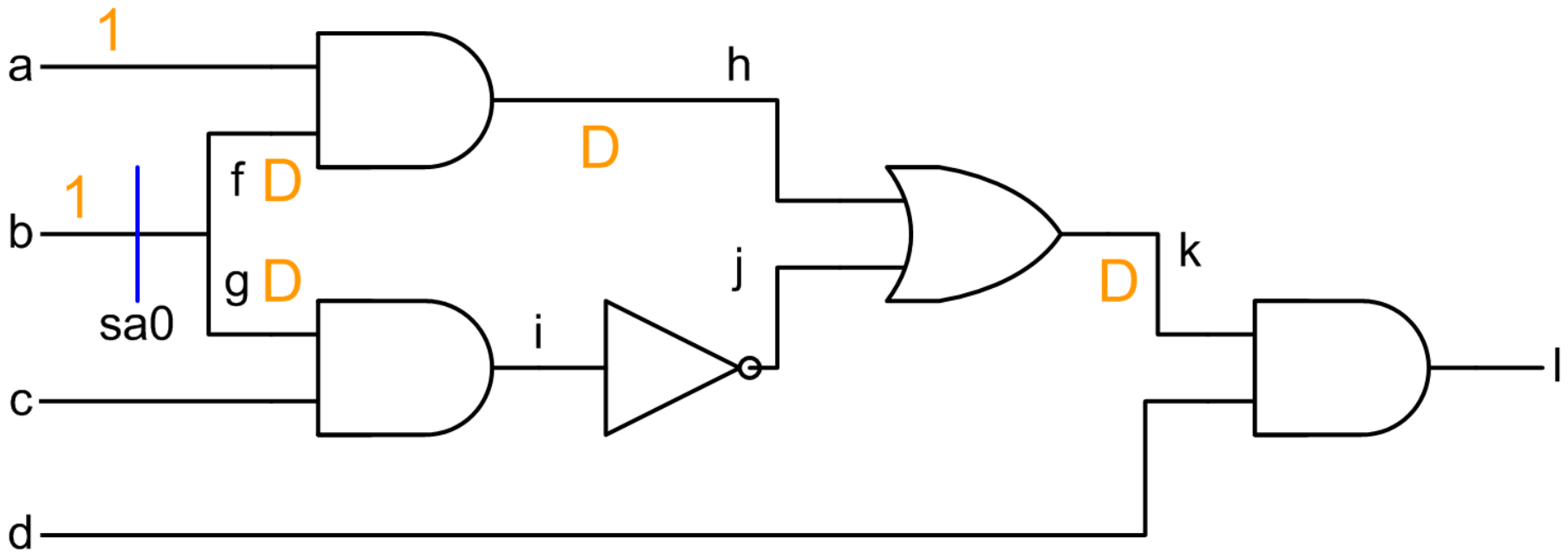

Ablesbarkeit des Fehlers an Knoten f und g

# Pfadsensibilisierung



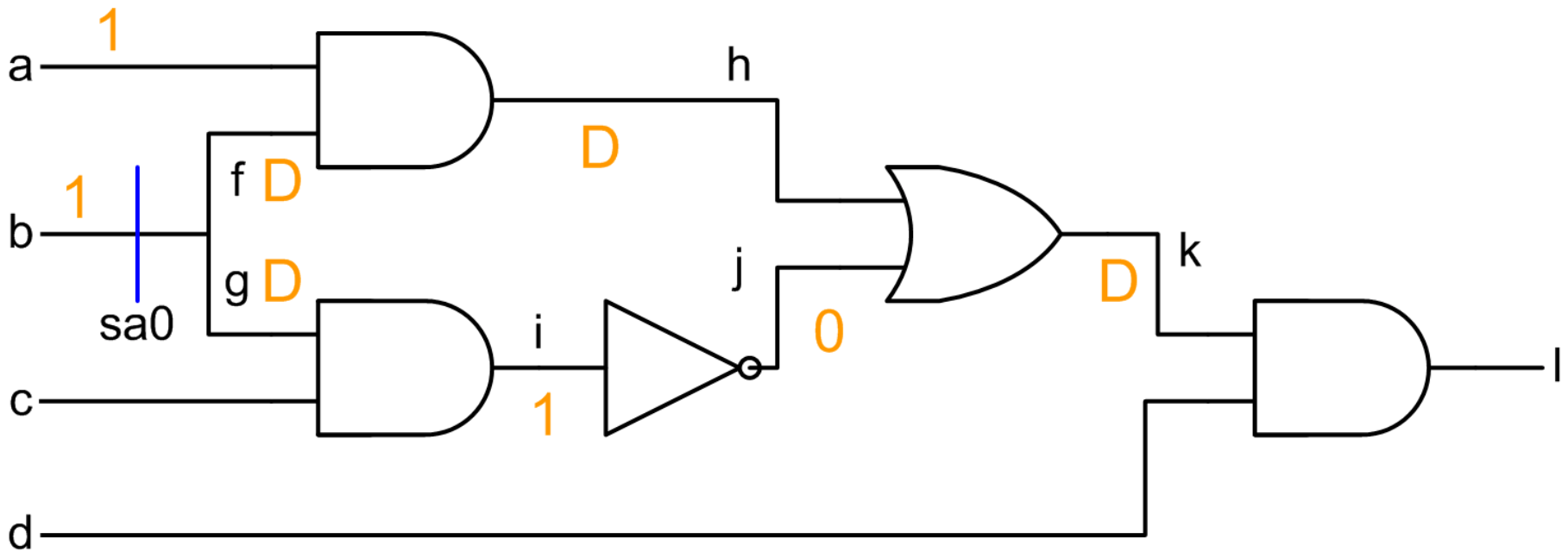

Propagieren des Signals über den Pfad f-h-k-l, Rückwärtsimplikation zum Knoten a (engl. line justification)

# Pfadsensibilisierung



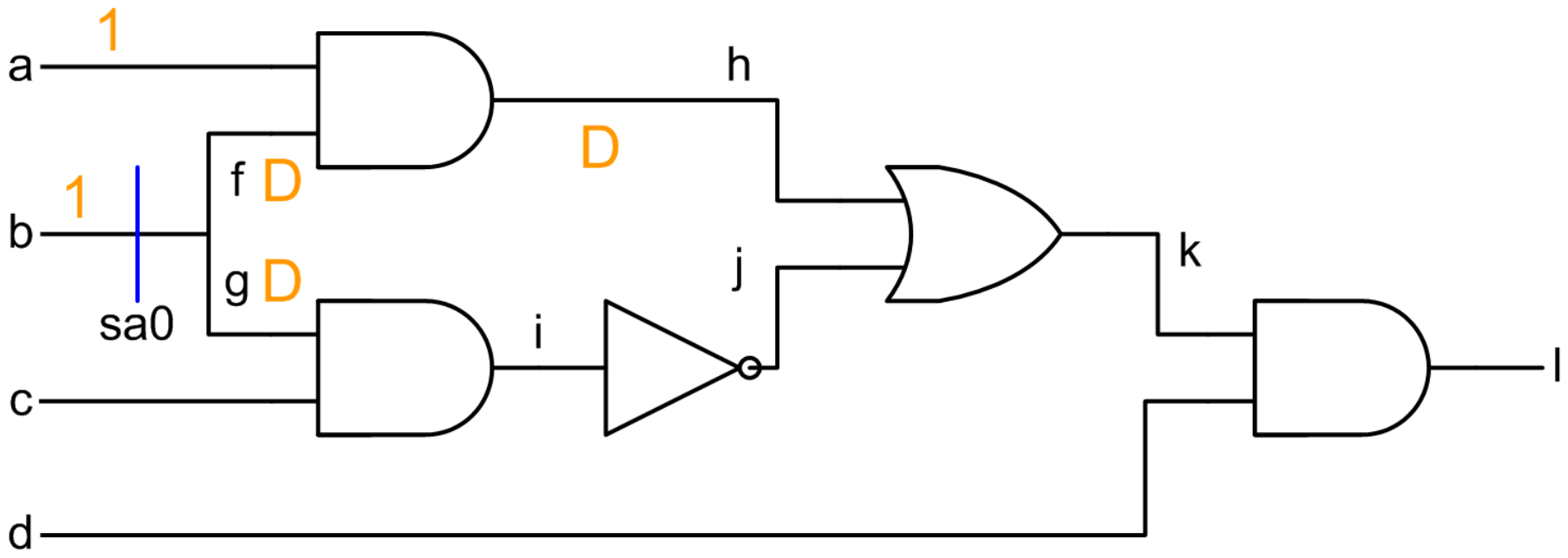

Propagieren des Signals über den Pfad f-h-k-l

# Pfadsensibilisierung



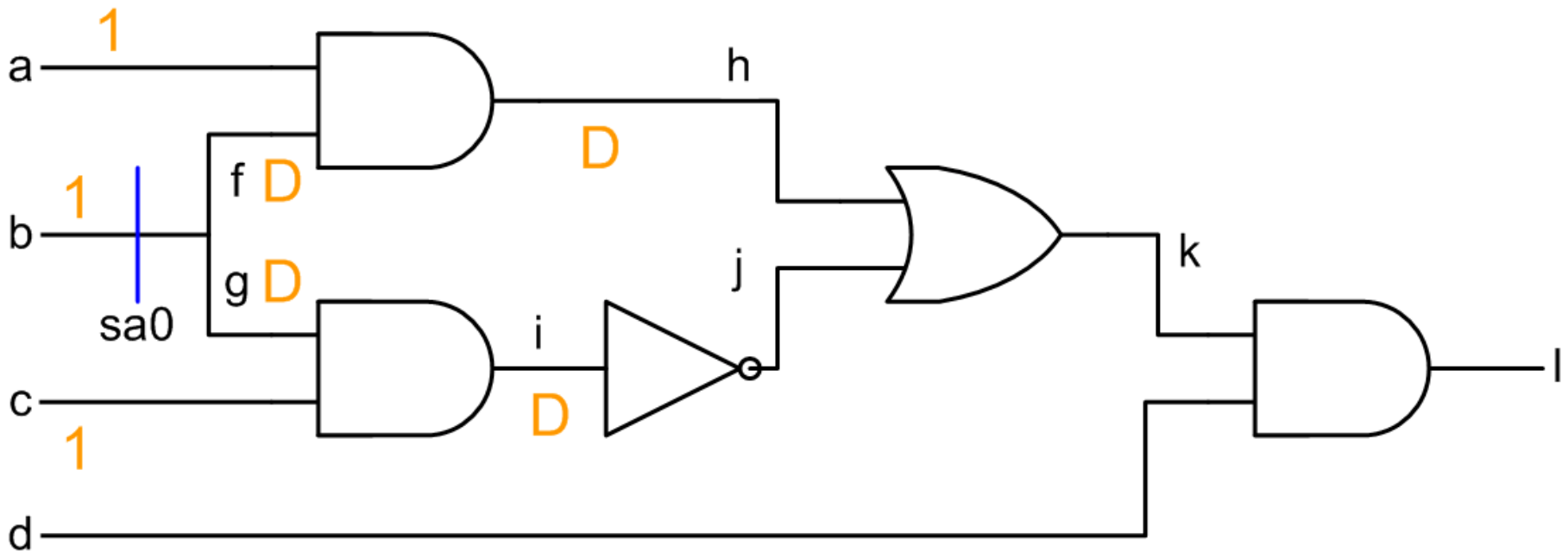

Propagieren des Signals über den Pfad f-h-k-l, Rückwärtsimplikation zum Knoten j, i, Inkonsistenz

# Pfadsensibilisierung



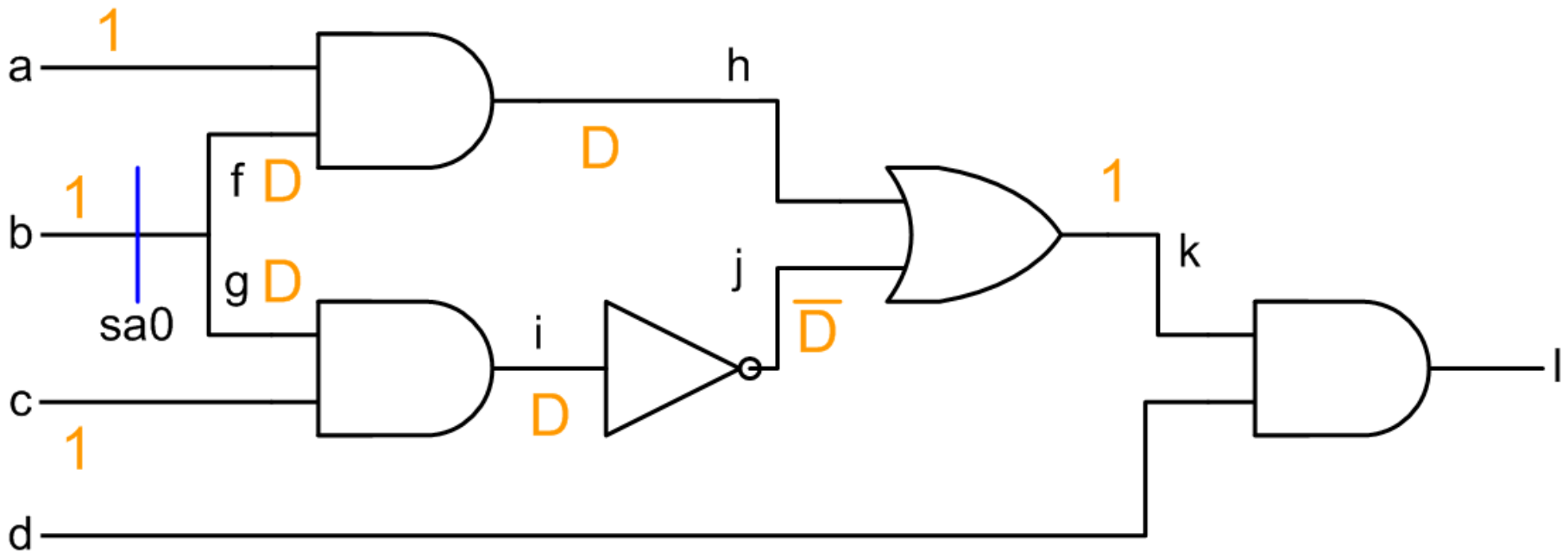

Backtracking bis zur letzten Entscheidung

# Pfadsensibilisierung



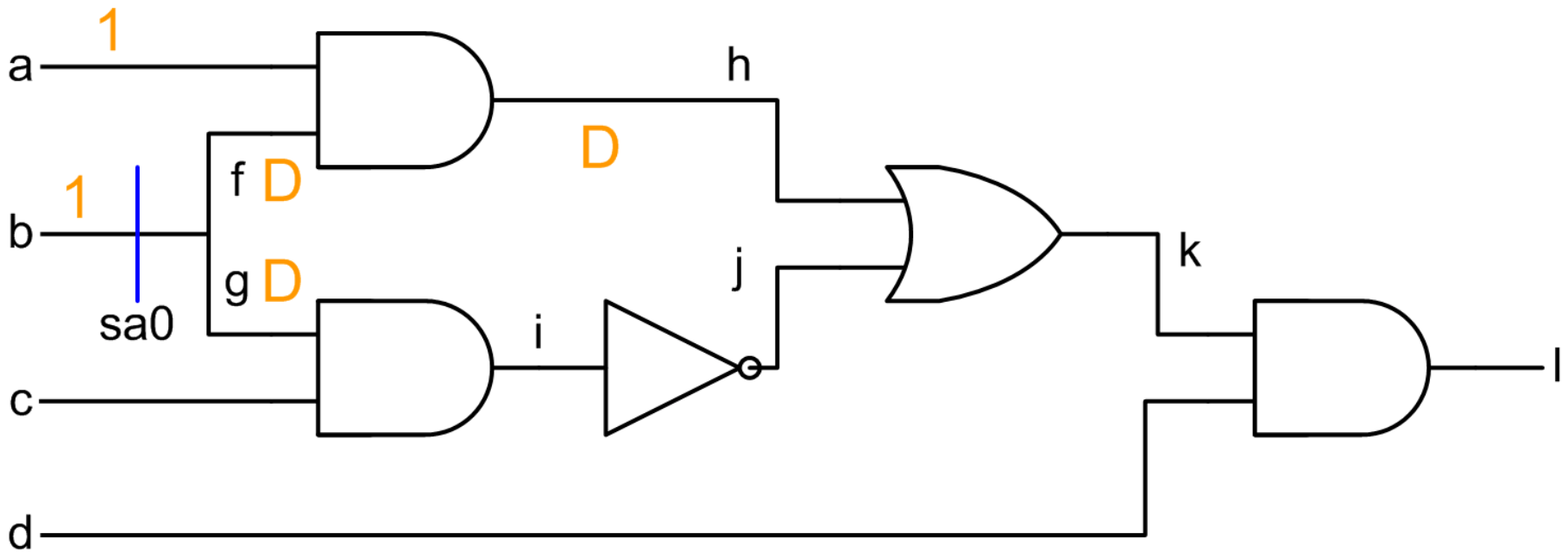

Propagation über i zusätzlich zu h

# Pfadsensibilisierung



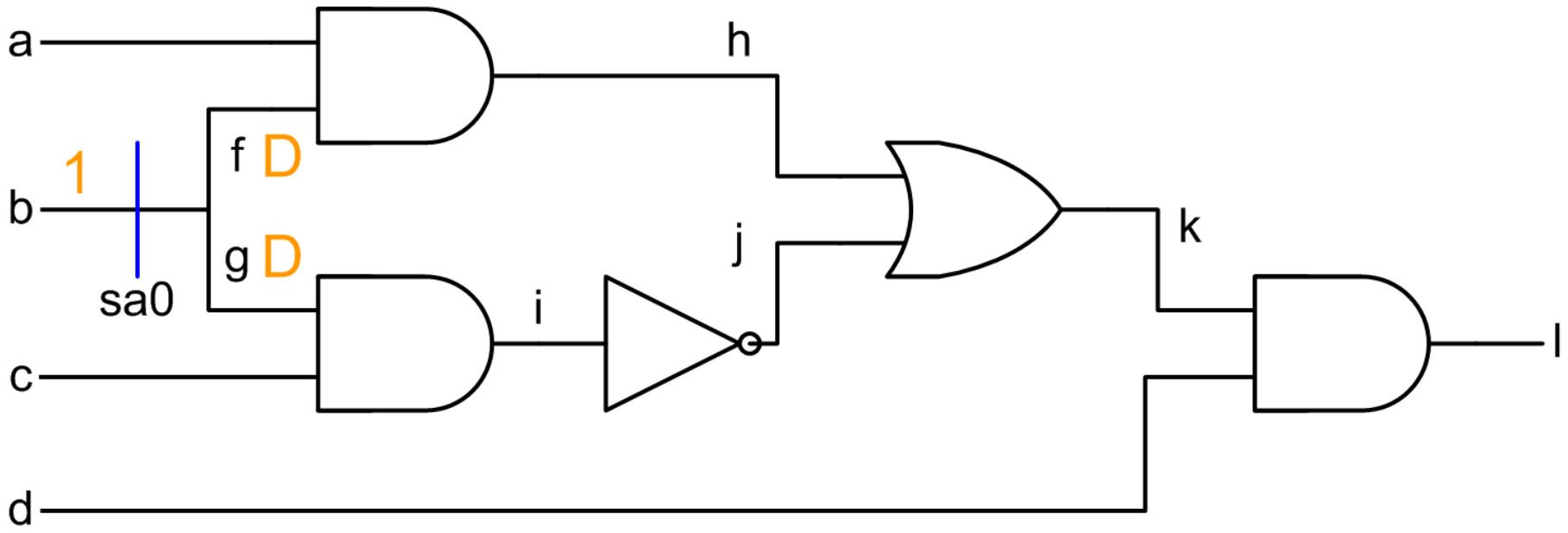

Blockiert bei k

# Pfadsensibilisierung



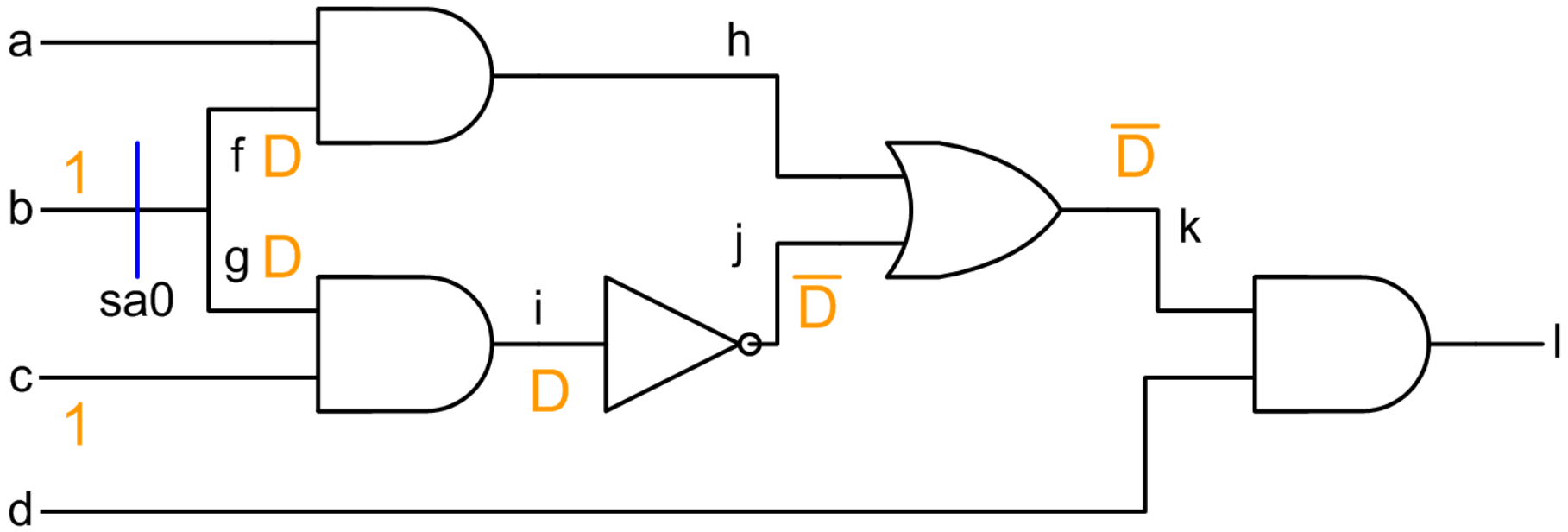

Backtracking bis zur letzten Entscheidung

# Pfadsensibilisierung



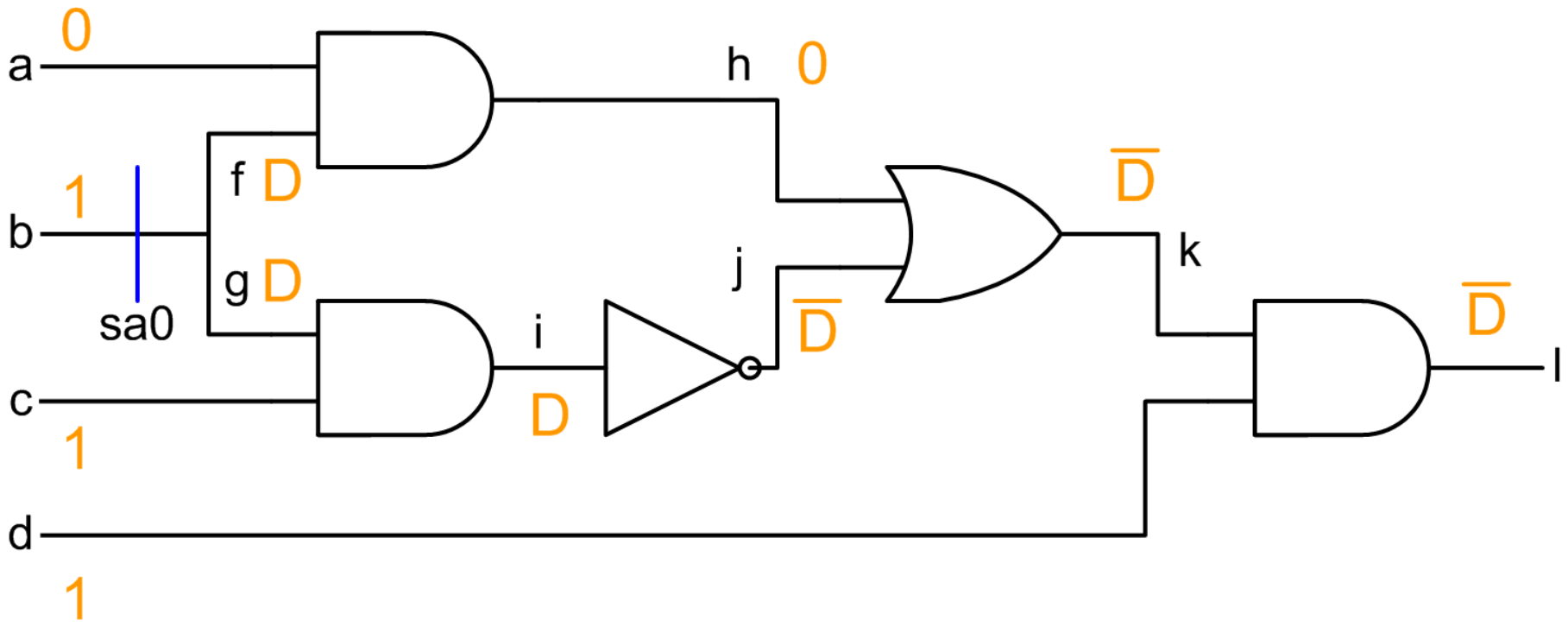

weiteres Backtracking, neuer Versuch mit Pfad g-i-j-k-l

# Pfadsensibilisierung




Propagieren des Signals über den Pfad g-i-j-k-l, Rückwärtsimplikation zum Knoten c

# Pfadsensibilisierung




Testpfad b-g-i-j-k-l gefunden !

# D-Algorithmus

- erster Algorithmus der für jeden Fehler auch einen Test findet, falls existent
- 1966 bei IBM von Roth entwickelt
- führte die D-Notation ein

# Funktionsweise D-Algorithmus

- auf betrachteter Leitung mit Stuck-At-Fehler wird dem Fehler entgegengesetzter Wert erzeugt
- Rückwärtsimplikation zu den Eingängen hin durchführen (Implikation = eindeutig)
- alle Gatter, die Fehler in Richtung Ausgänge fortpflanzen können in sogenannte D-Front aufnehmen
- (Loop) Auswahl eines Gatters g aus der D-Front (plus Entfernen)
  - Eingangsbelegung von g wird zur Fehlerpropagierung ergänzt

# Funktionsweise D-Algorithmus

  - Vorwärts- / Rückwärtsimplikationen werden ausgeführt, bei Inkonsistenz mit nächstem Gatter fortfahren (Backtracking)

- noch undefinierte Belegungen werden bestimmt

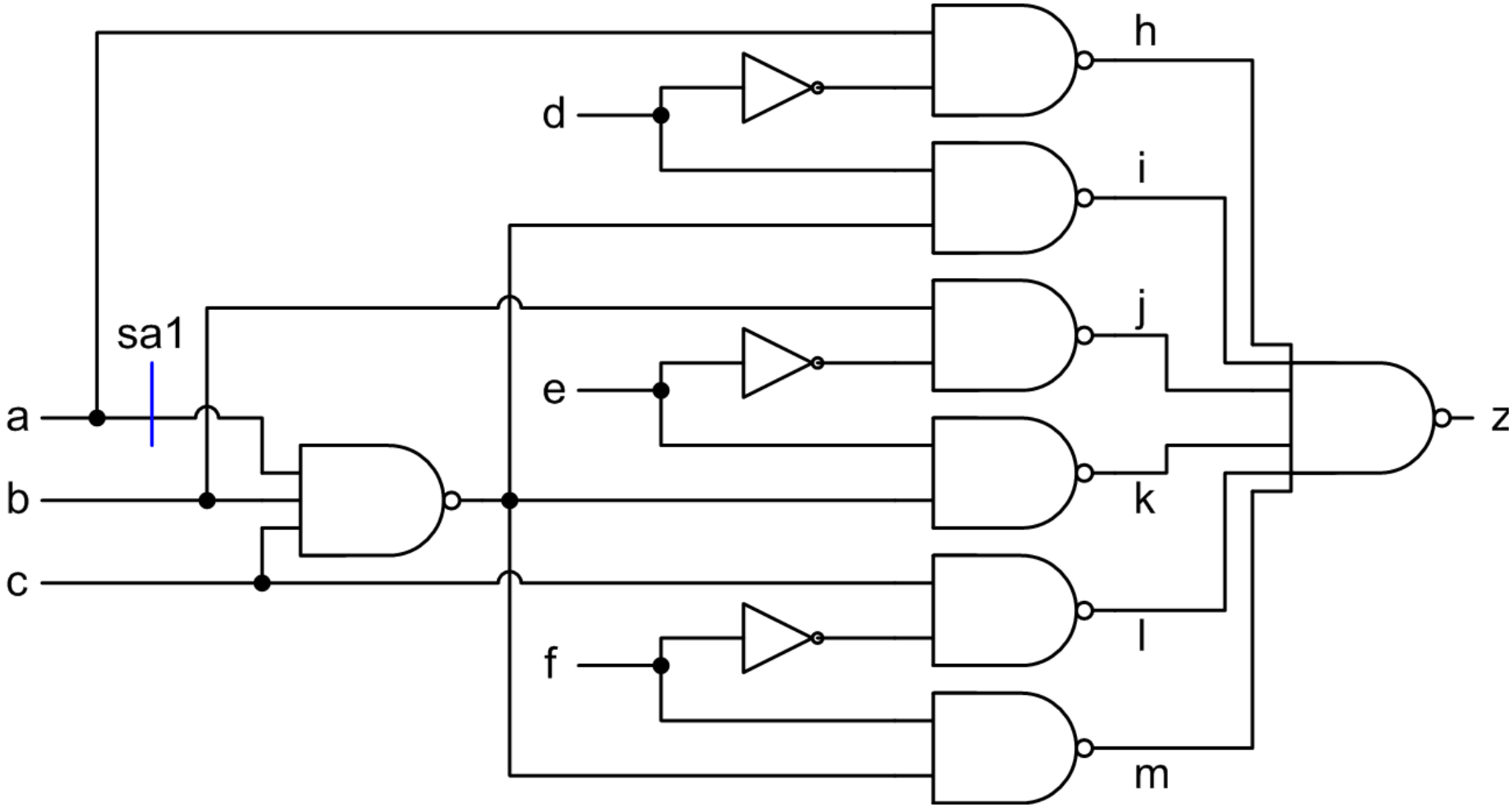
# Verbesserungen

| Algorithmus | geschätzte Beschleunigung | Jahr |
|---|---|---|
| D-Algorithmus | 1 | 1966 |
| PODEM | 7 | 1981 |
| FAN | 23 | 1983 |
| TOPS | 292 | 1987 |
| SOCRATES | 1574 | 1988 |
| Waicukauski | 2189 | 1990 |
| EST | 8765 | 1991 |
| TRAN | 3005 | 1993 |
| Rekursives Lernen | 485 | 1995 |
| Tafertshofer | 25057 | 1997 |

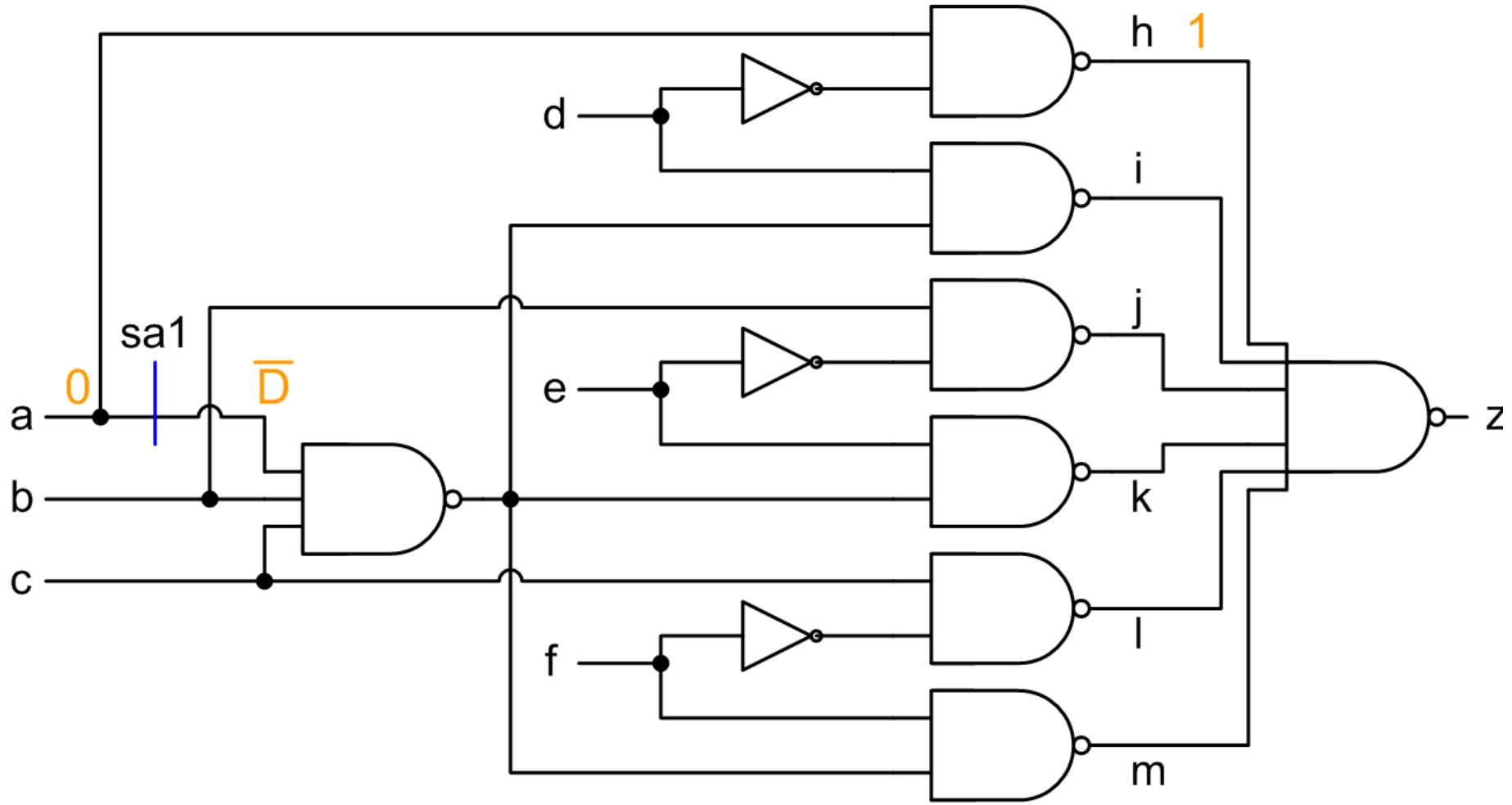
# PODEM (Path Oriented Decision Making)

- D-Algorithmus erlaubt Wertzuweisungen an beliebige Knoten
- exponentielle Laufzeit zur Anzahl der Knoten
- XOR-Knoten lassen den D-Algorithmus die exponentielle Laufzeit erreichen
- PODEM beschränkt die Wertzuweisung auf primäre Eingänge
- Backtracking bei Inkonsistenzen auf diese beschränkt
- exponentielle Laufzeit zur Anzahl der Eingänge

# PODEM $\longleftrightarrow$ D-Algorithmus

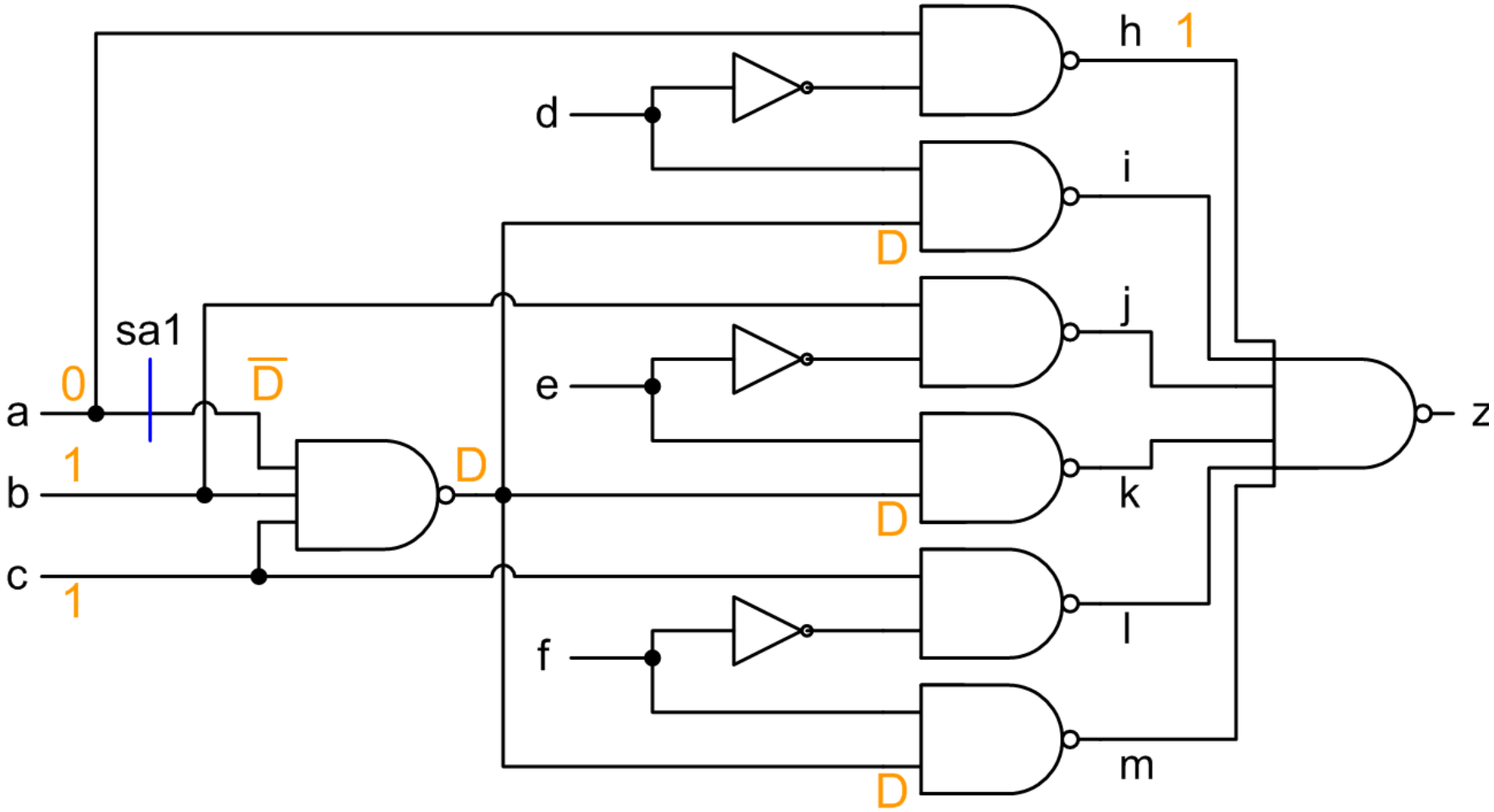


# D-Algorithmus



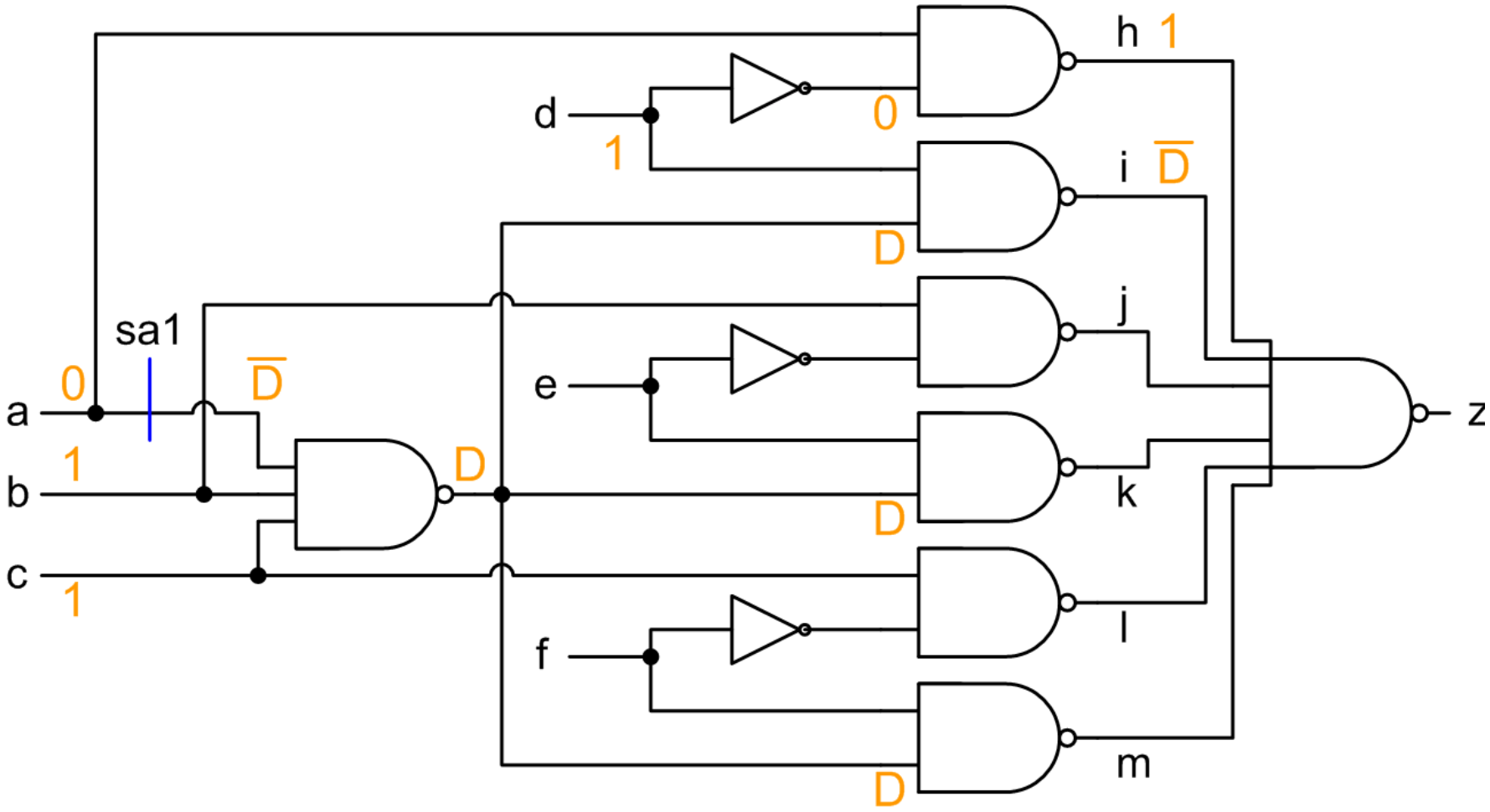

Fehlerinjektion durch a=0

# D-Algorithmus



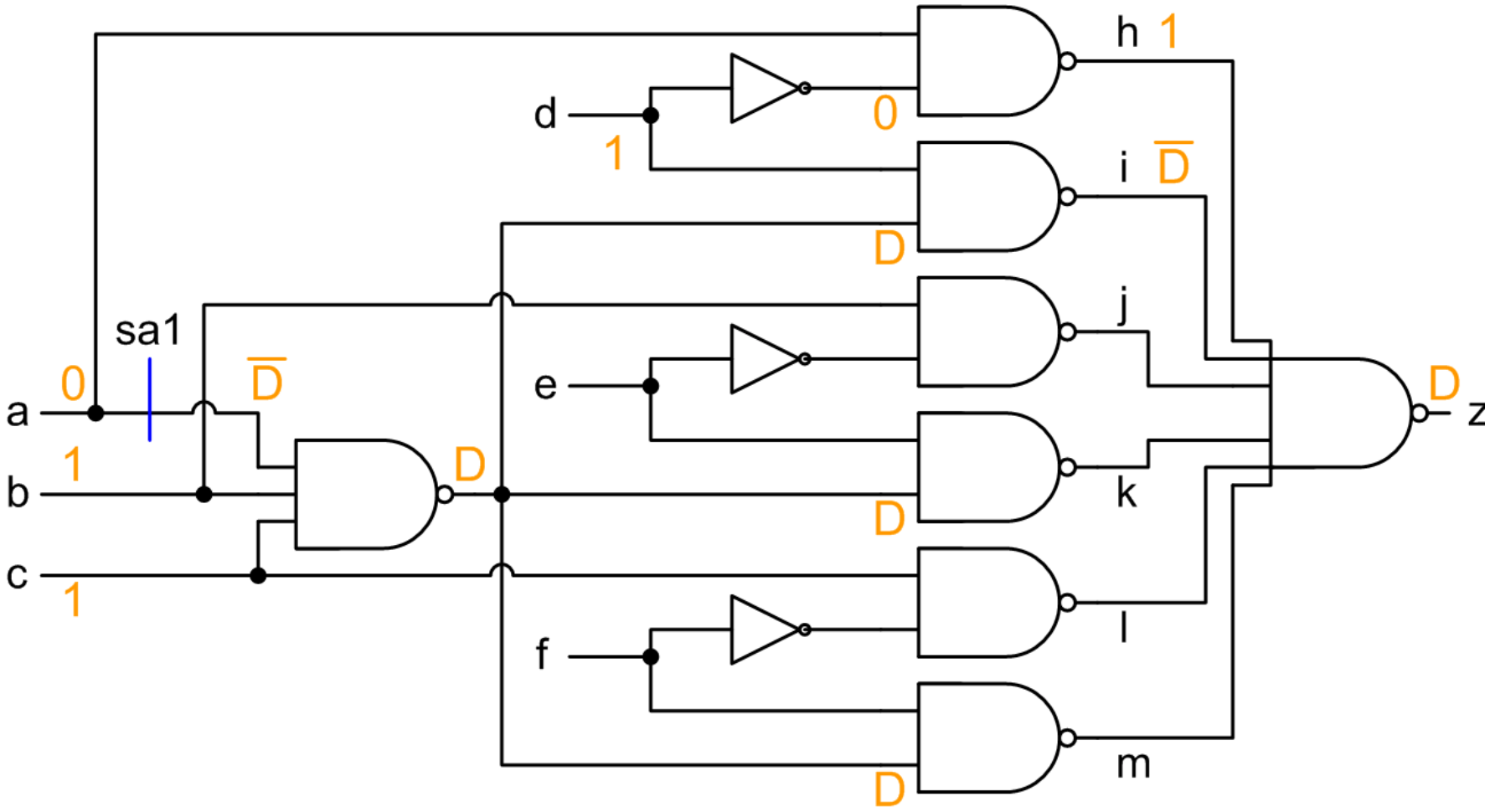

Propagieren durch NAND-Gatter mittels b=1 und c=1

# D-Algorithmus



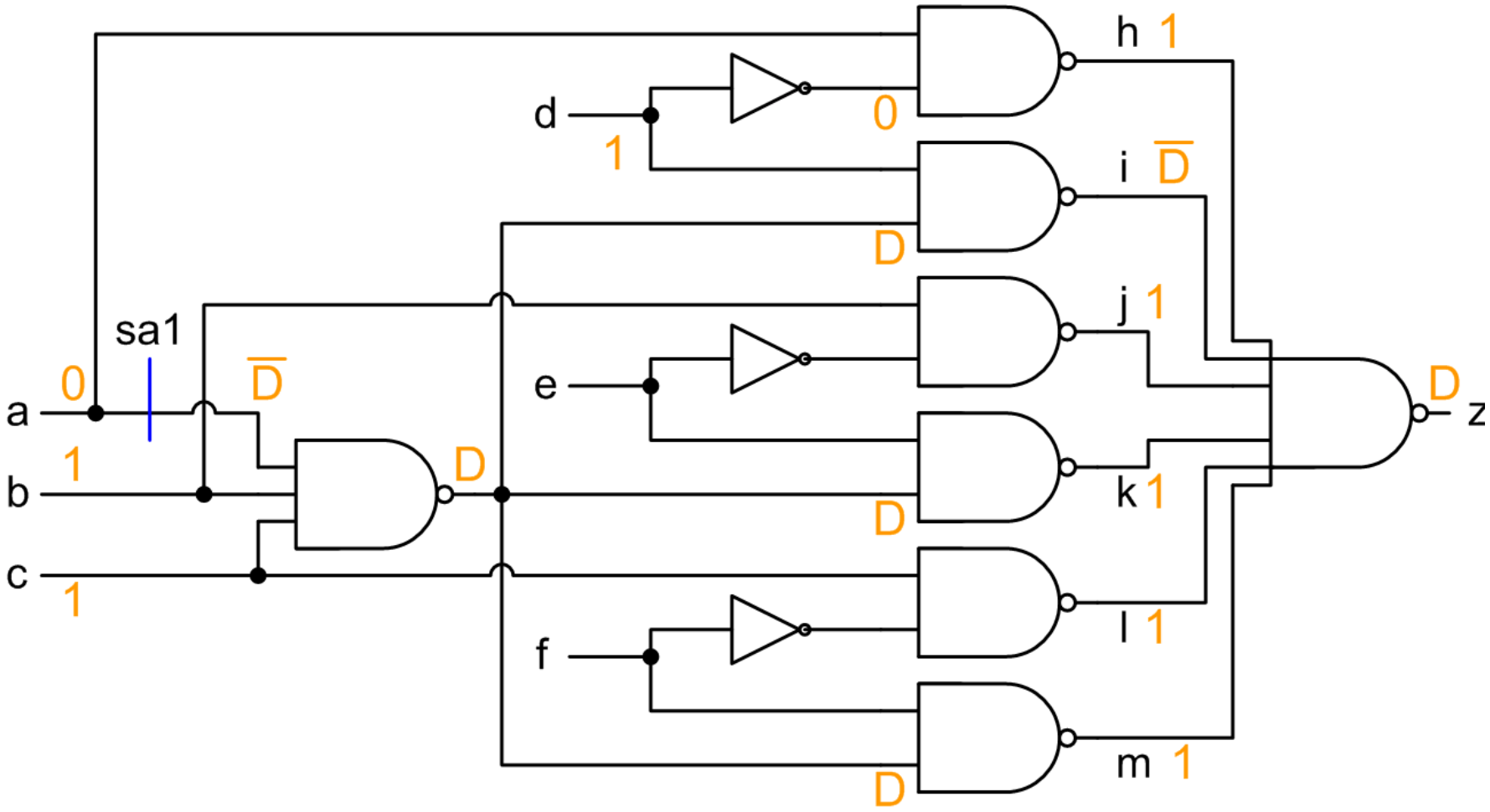

Propagieren über Pfad i (Alternativen wären k und m)

# D-Algorithmus



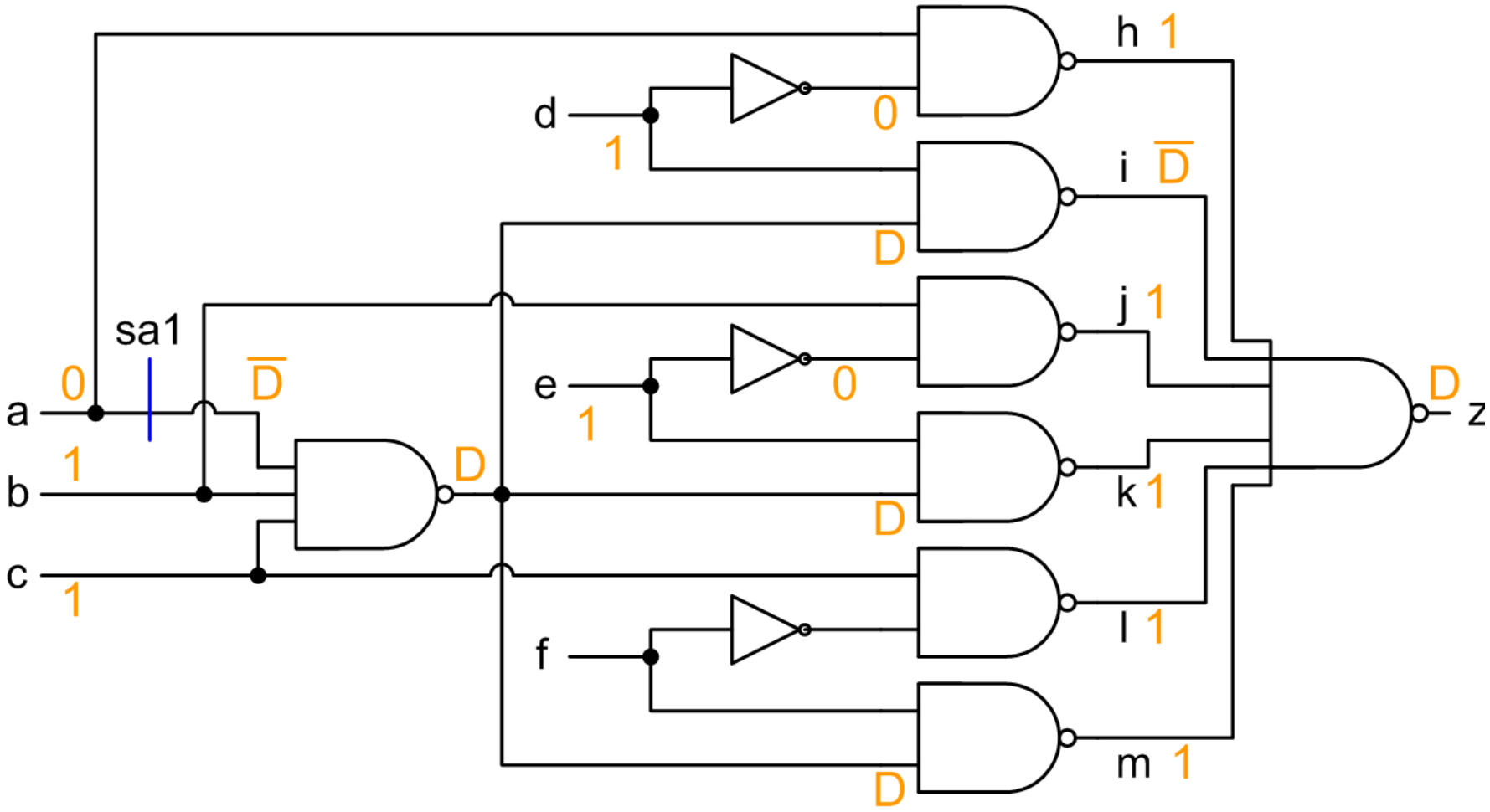

Propagieren zum Ausgang z (Alternativen k und m)

# D-Algorithmus



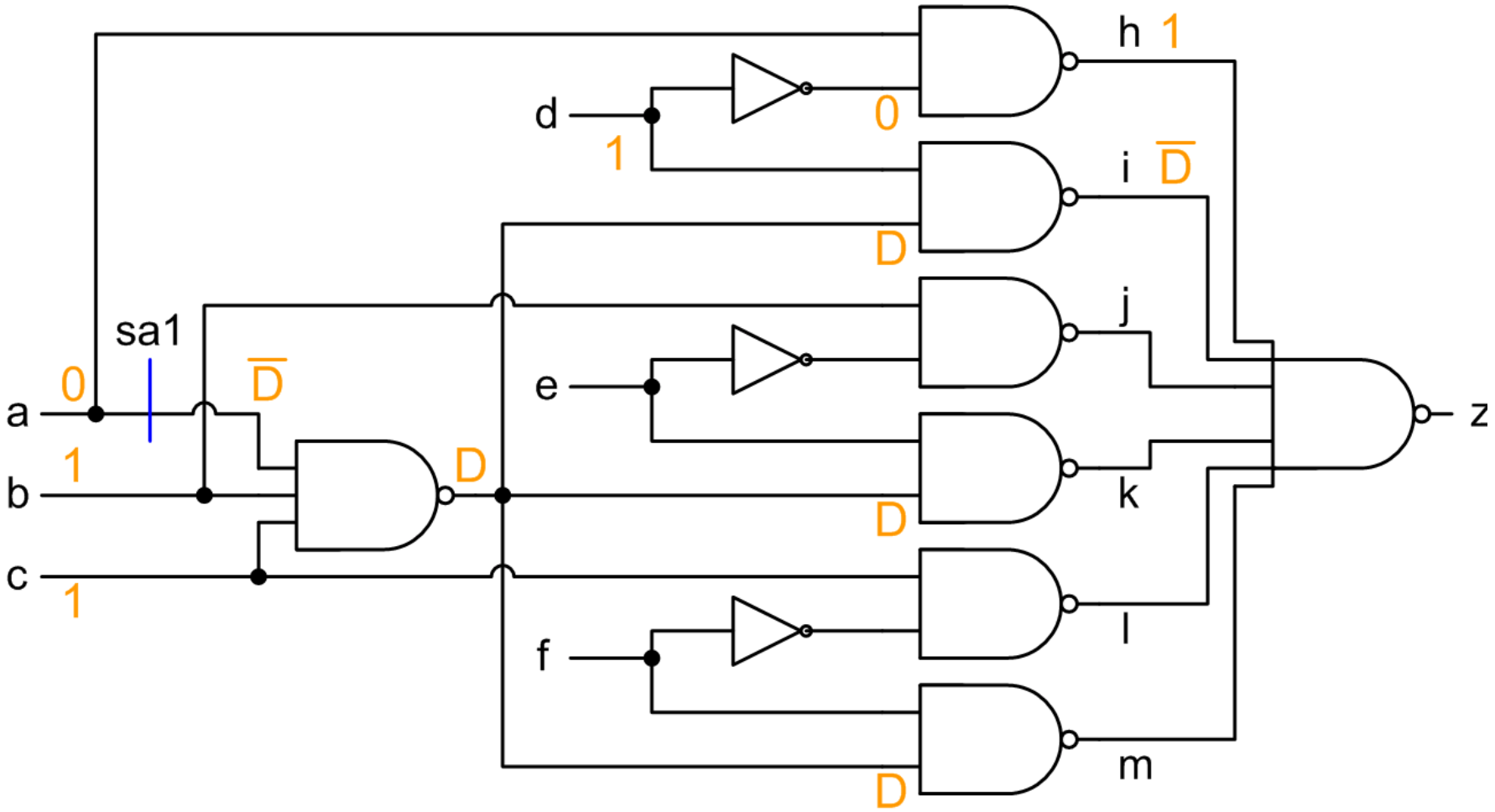

Propagieren zum Ausgang z (Alternativen k und m)

# D-Algorithmus



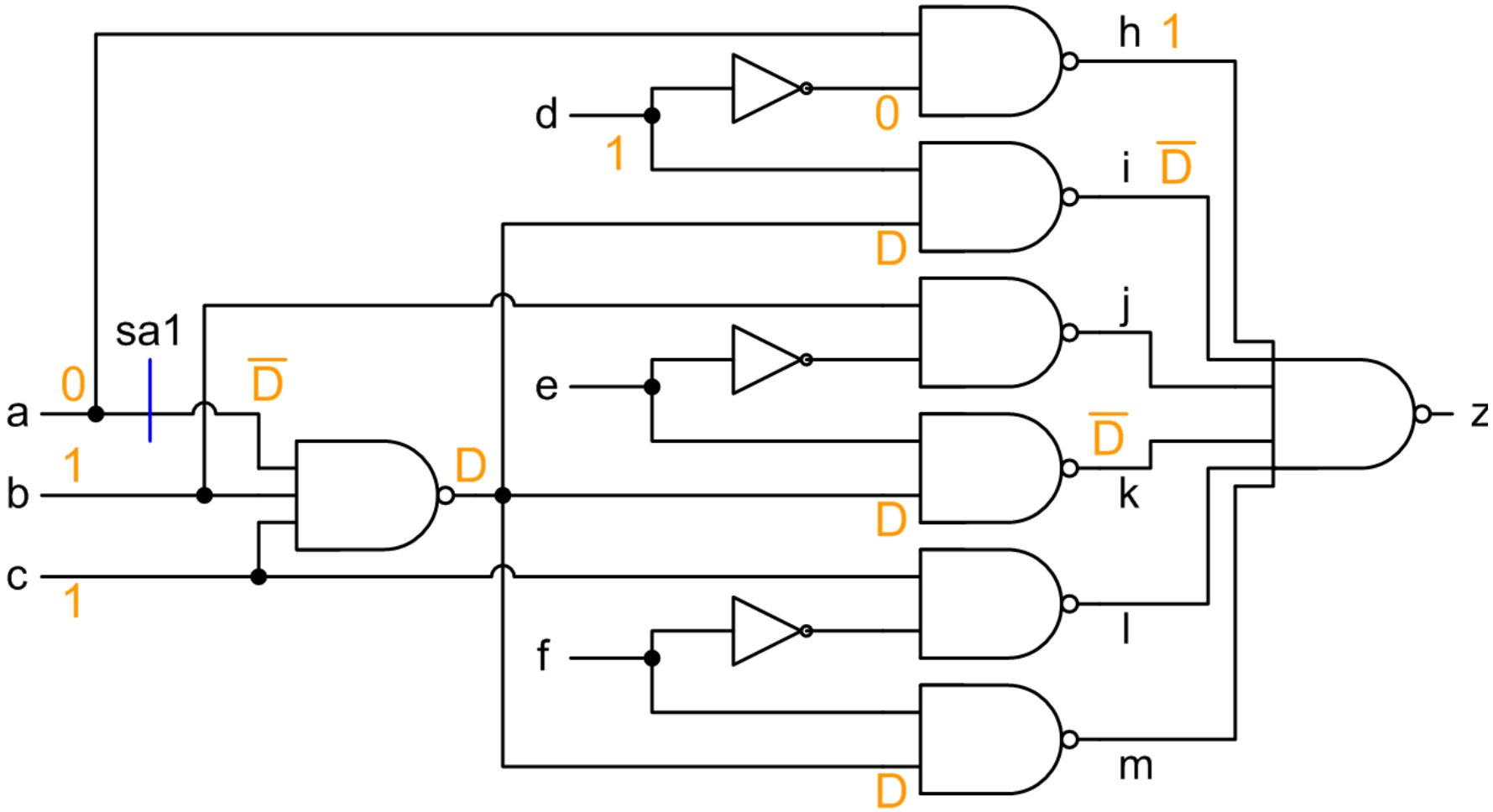

e=1 kann nicht j=1 und k=1 erfüllen → Inkonsistenz

# D-Algorithmus



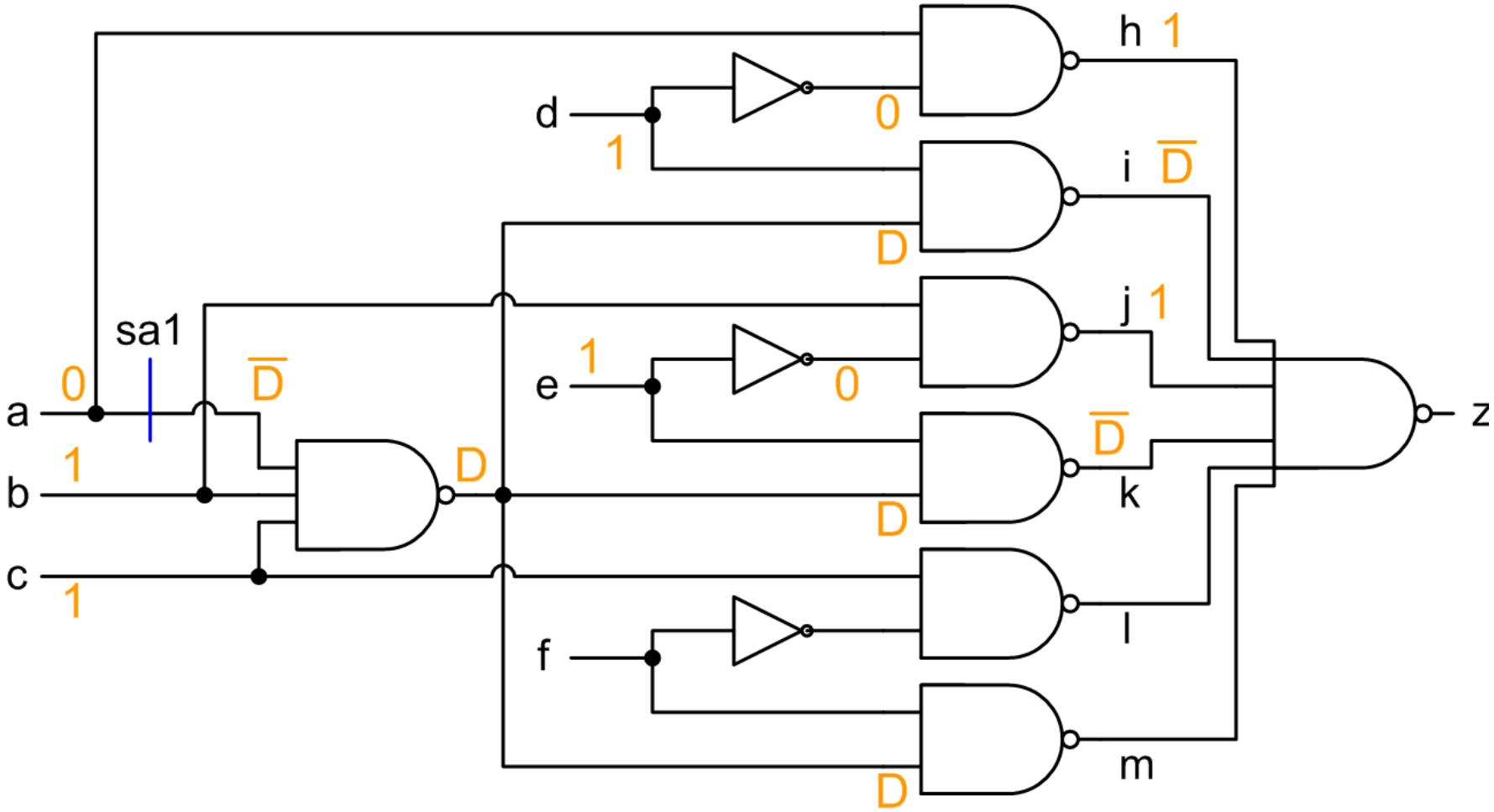

Backtracking zum letzten Auswahlpunkt

# D-Algorithmus



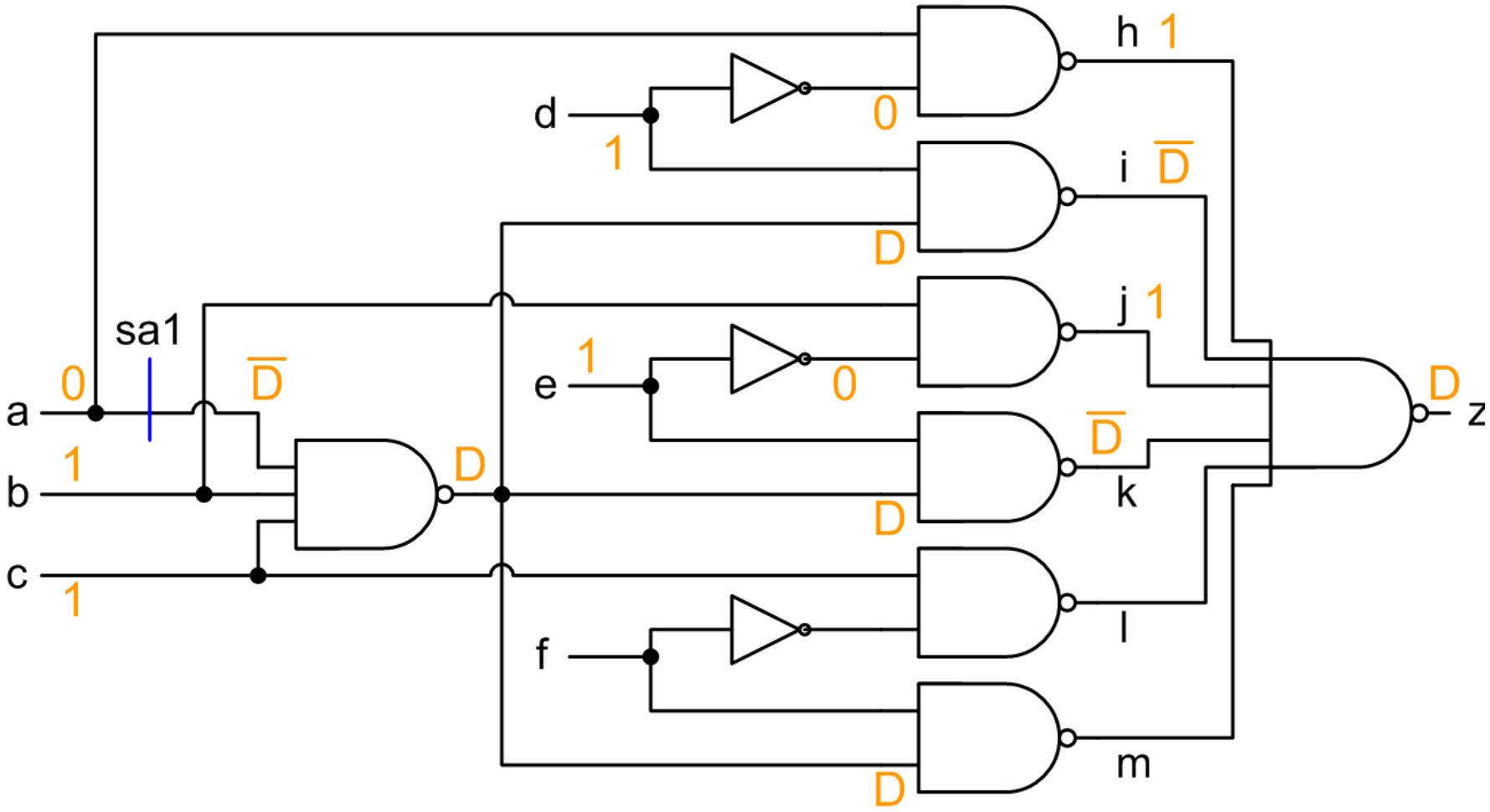

Propagieren über k (Alternative m)

# D-Algorithmus



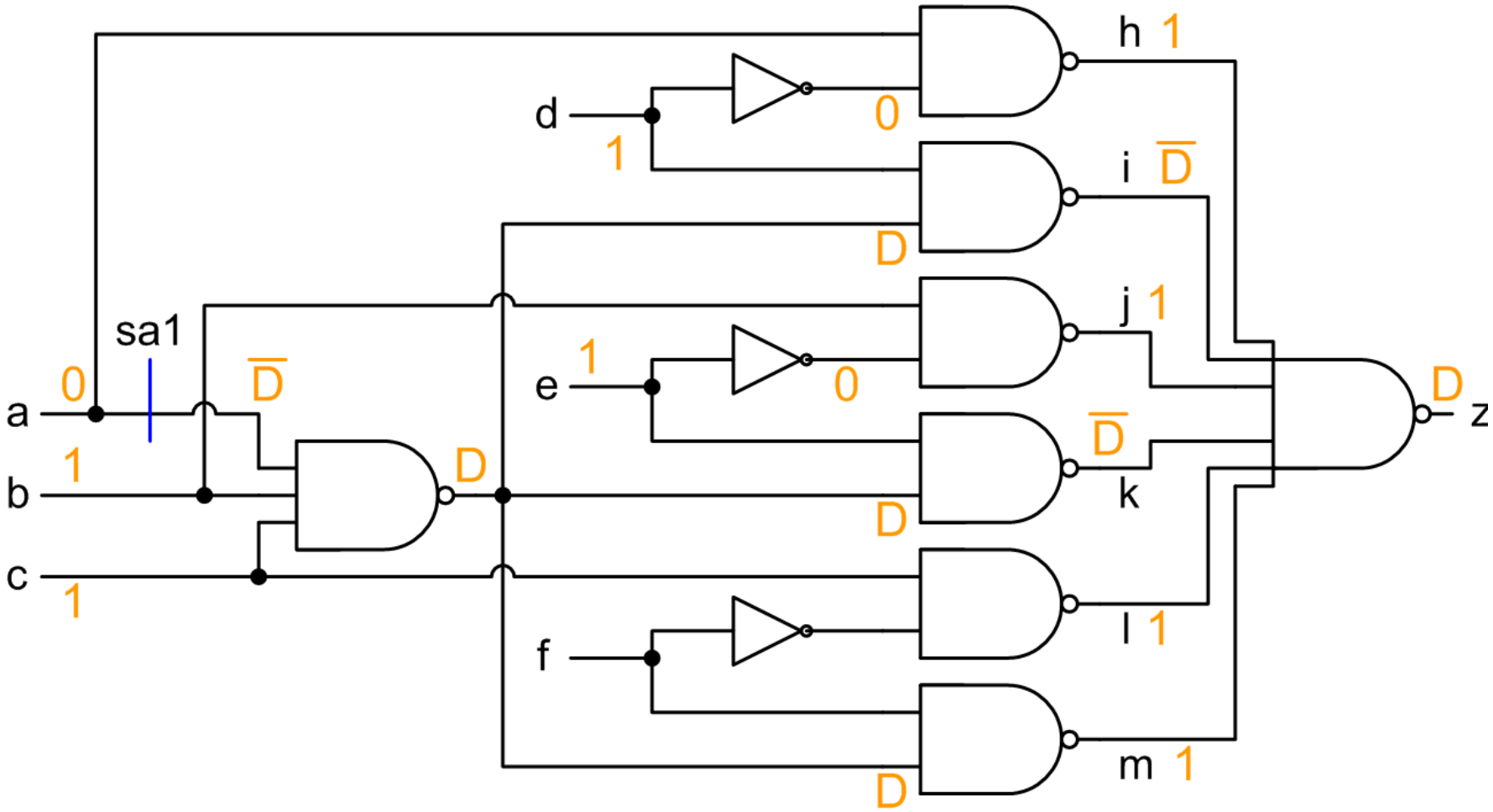

mit e=1 funktioniert Propagieren

# D-Algorithmus



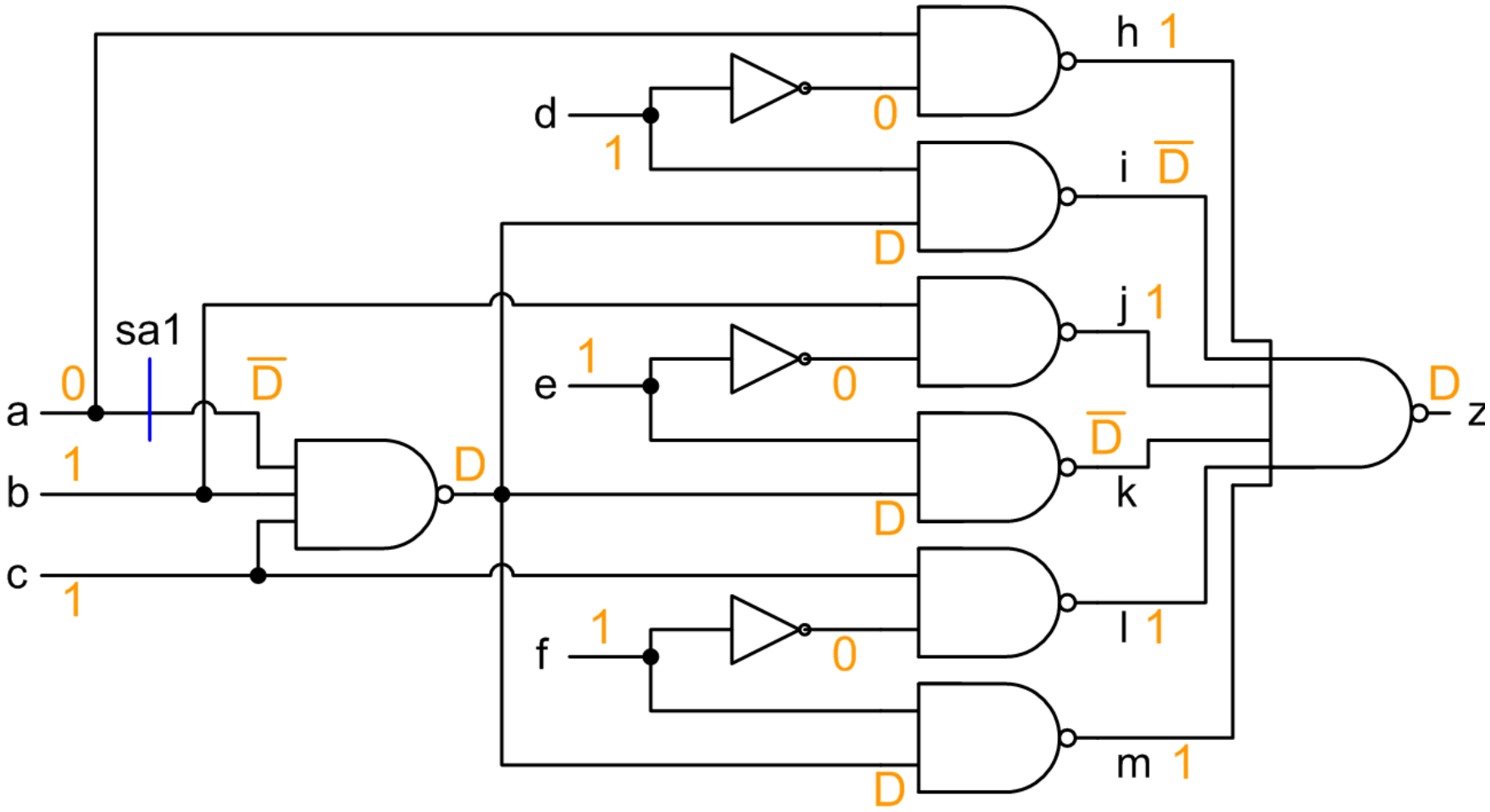

Versuch Propagieren zum Ausgang z (Alternative m)

# D-Algorithmus



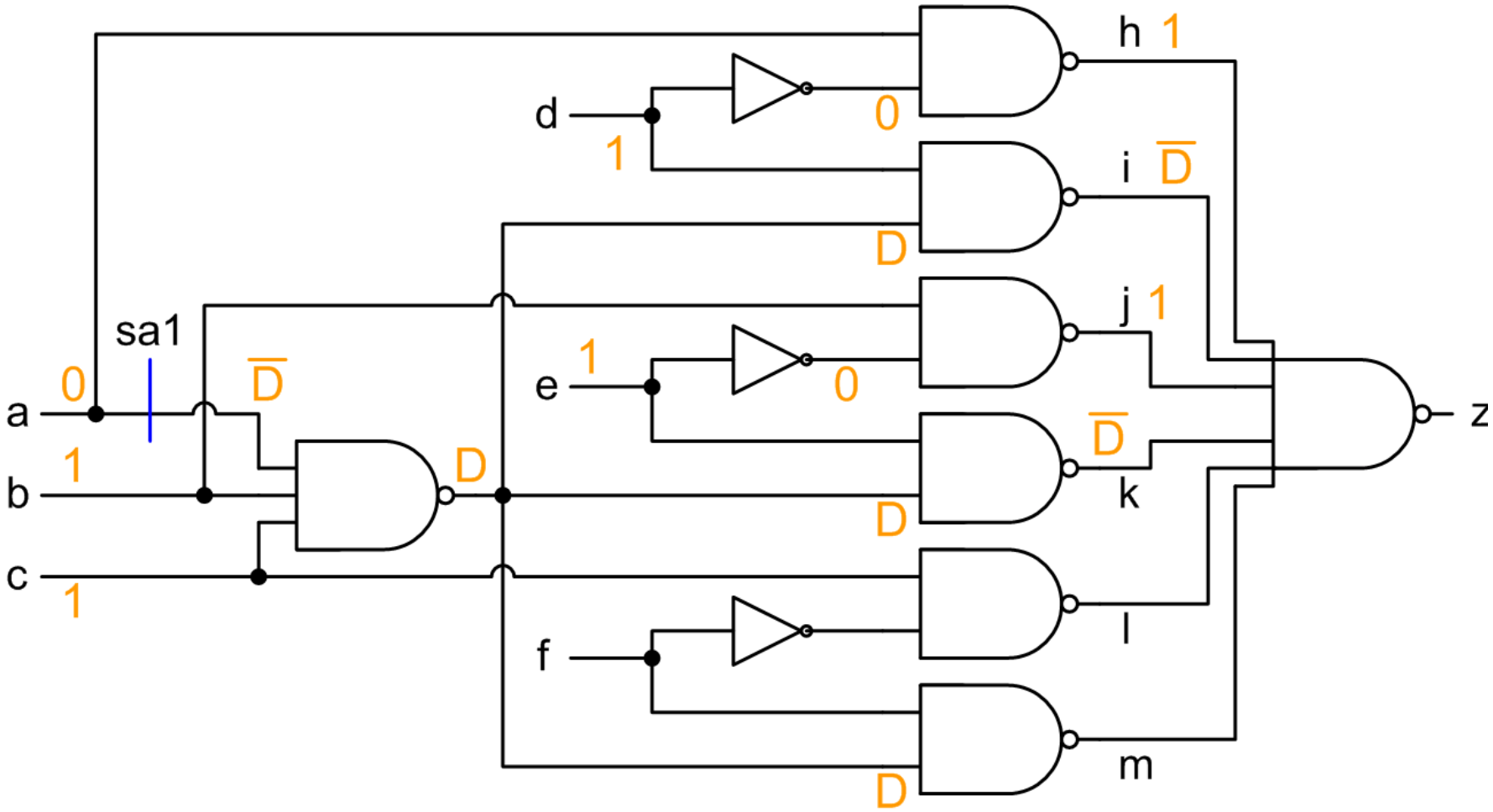

Versuch Propagierens zum Ausgang z (Alternative m)

# D-Algorithmus



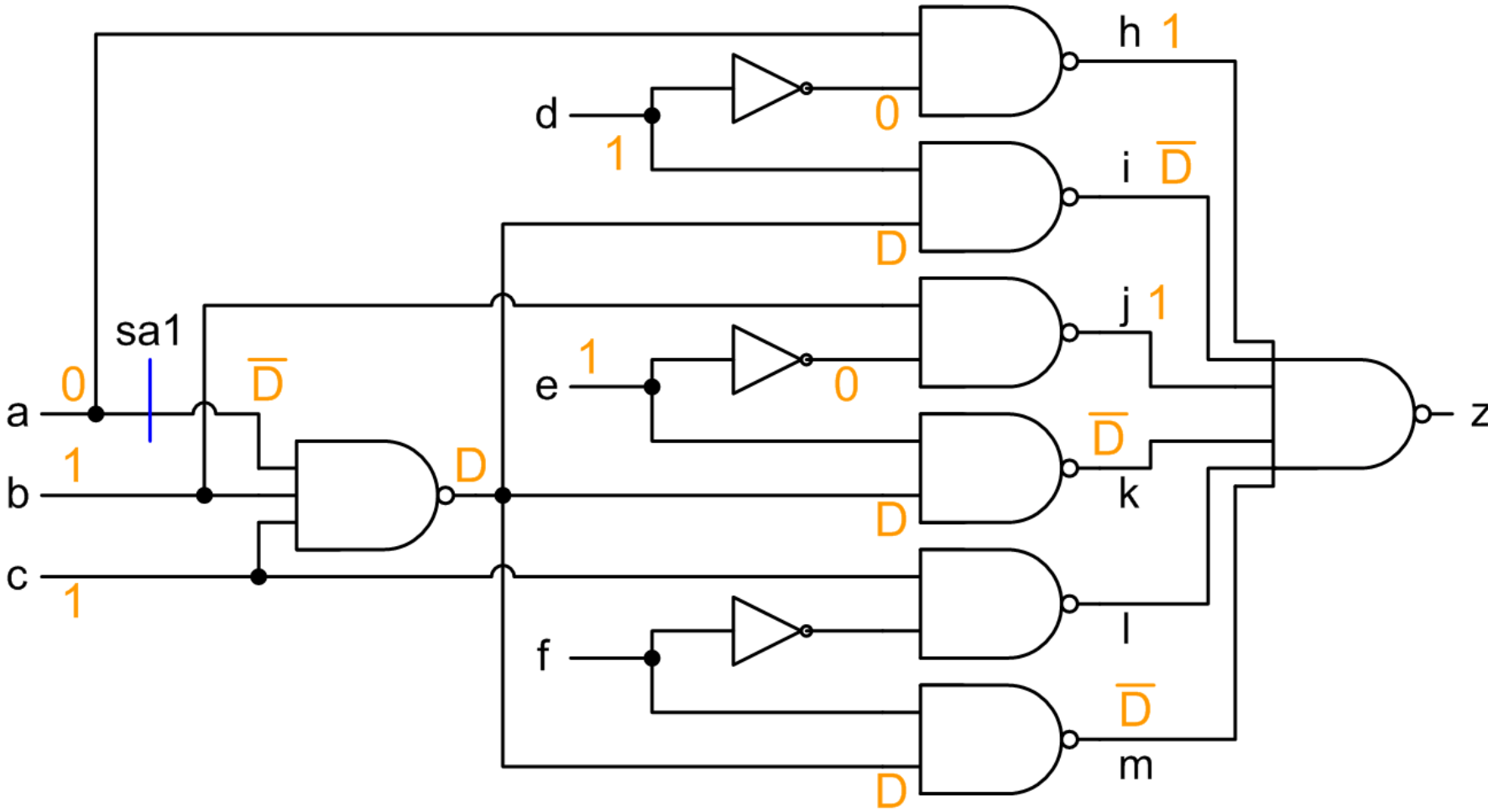

f=1 kann nicht l=1 und m=1 erfüllen → Inkonsistenz

# D-Algorithmus



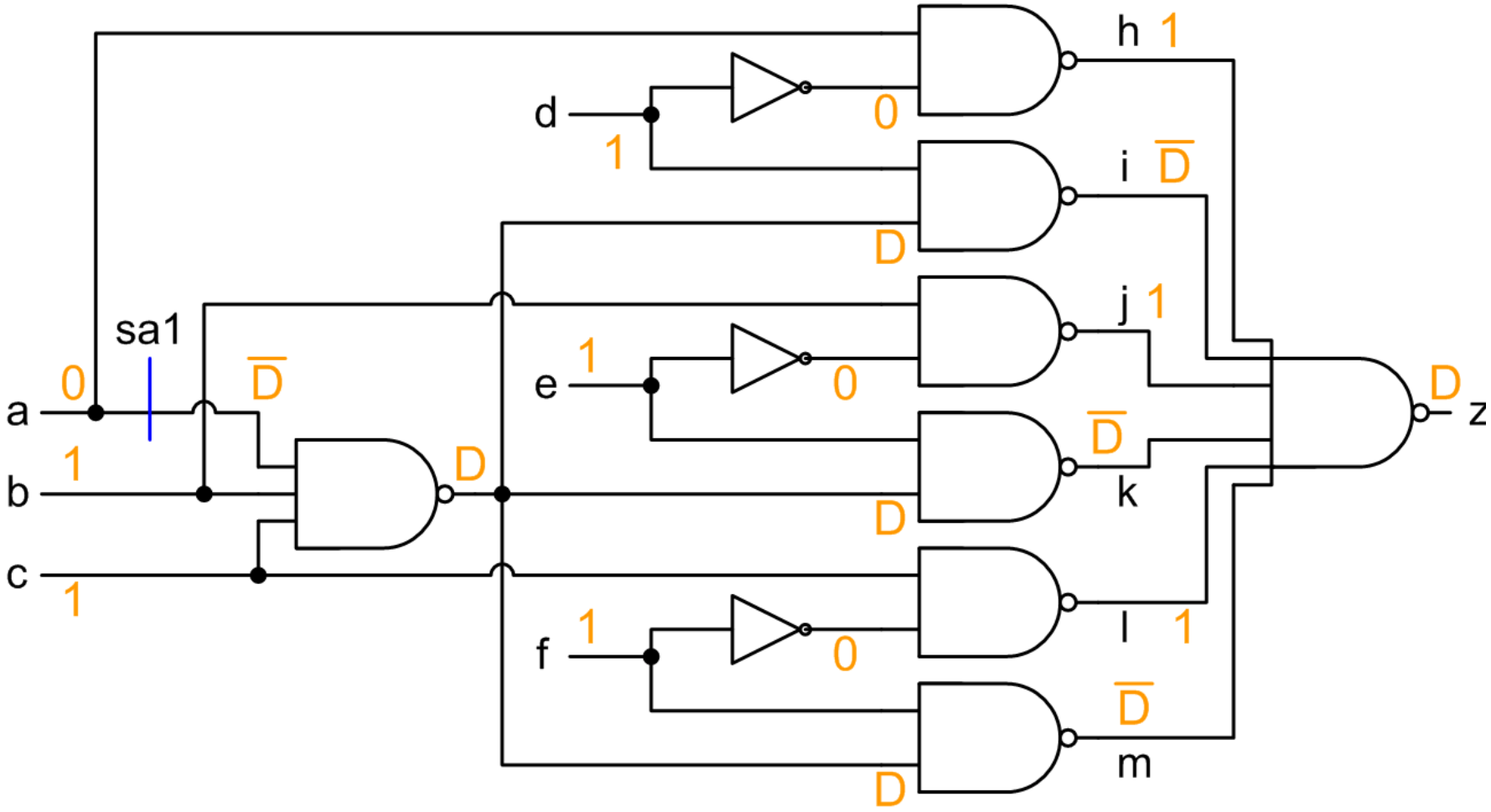

Backtracking zum letzten Auswahlpunkt

# D-Algorithmus



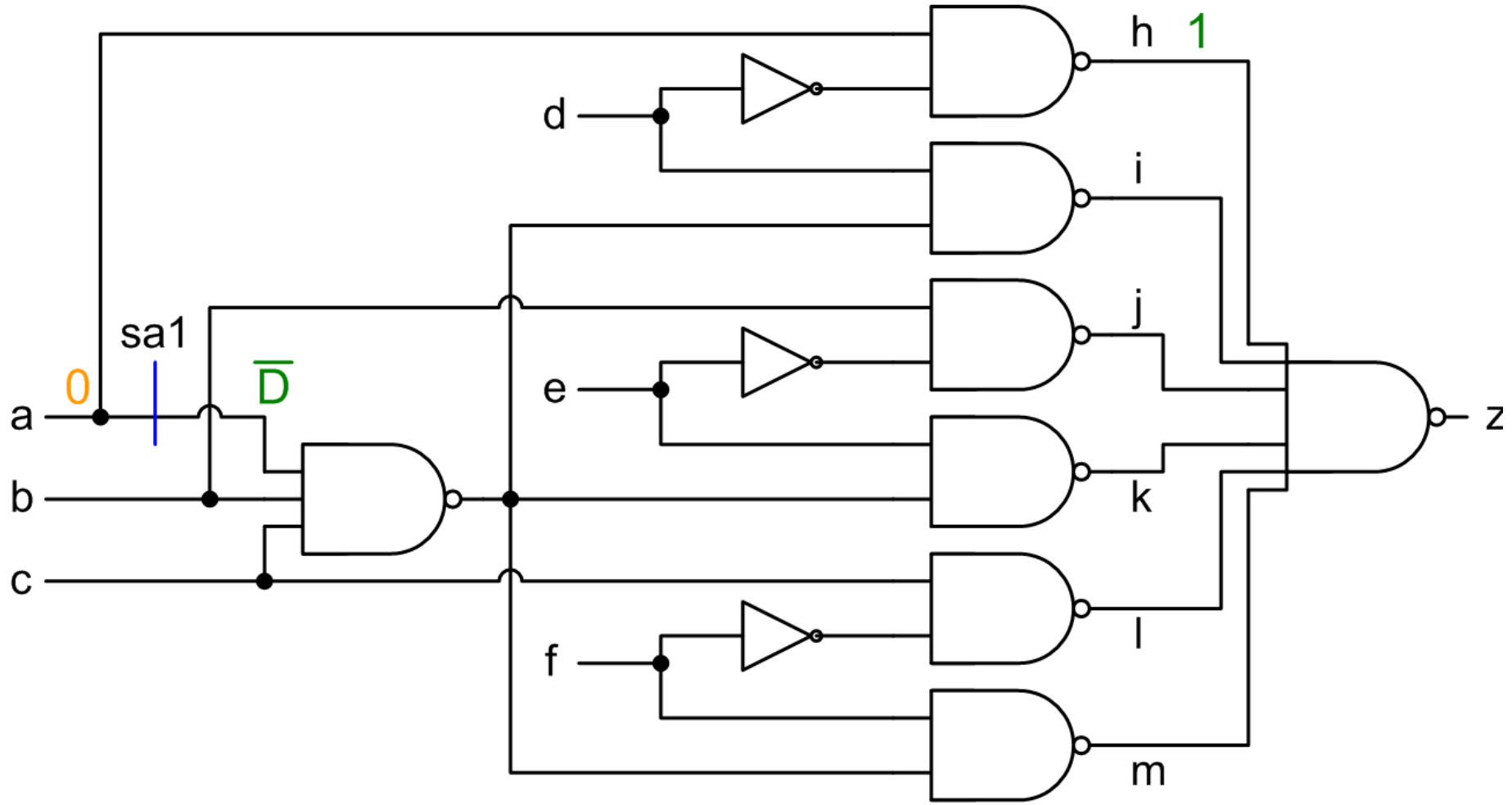

Propagieren über m

# D-Algorithmus



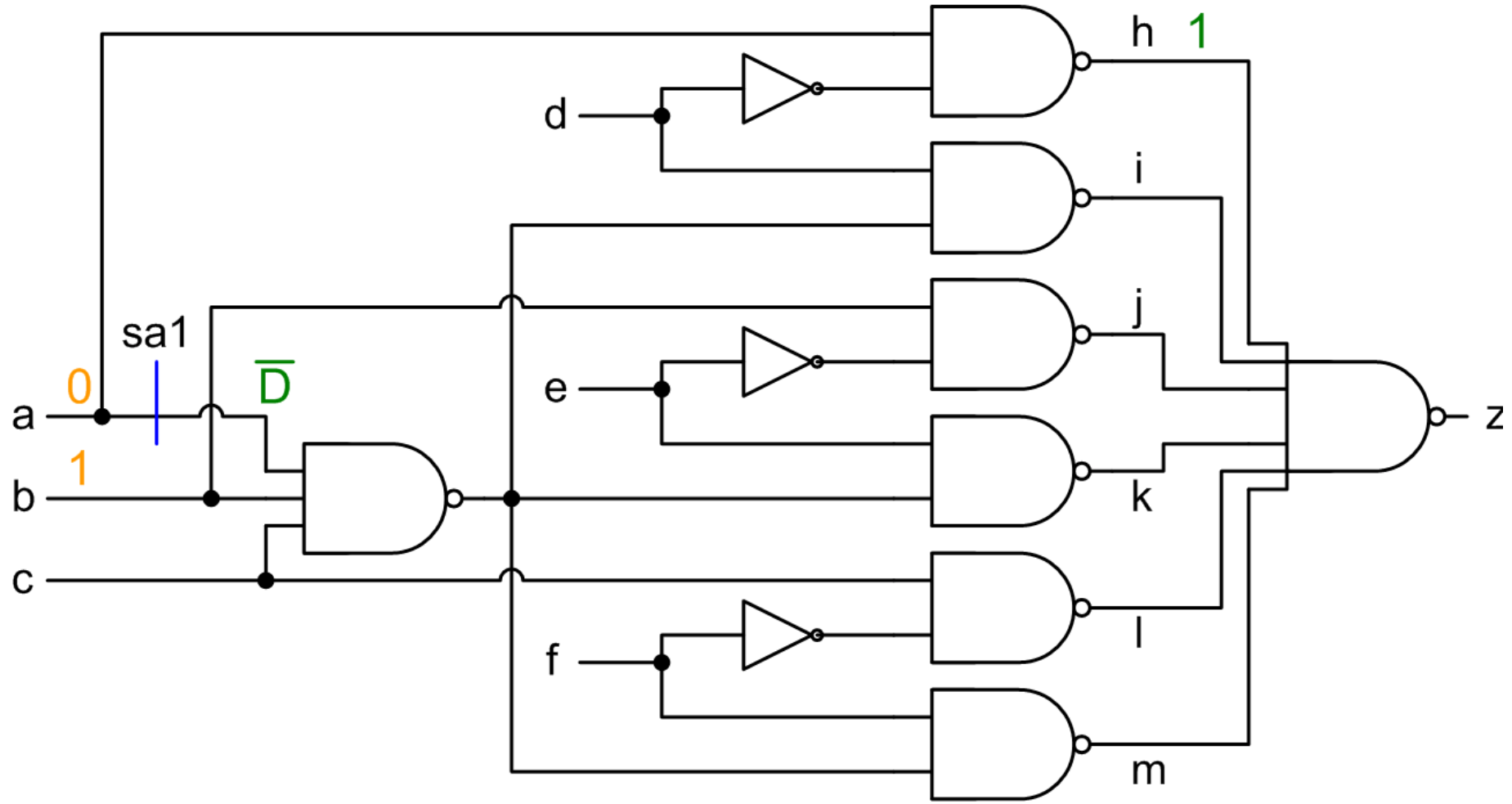

f=1 erfüllt dies und impliziert z=D → Testbelegung

# PODEM



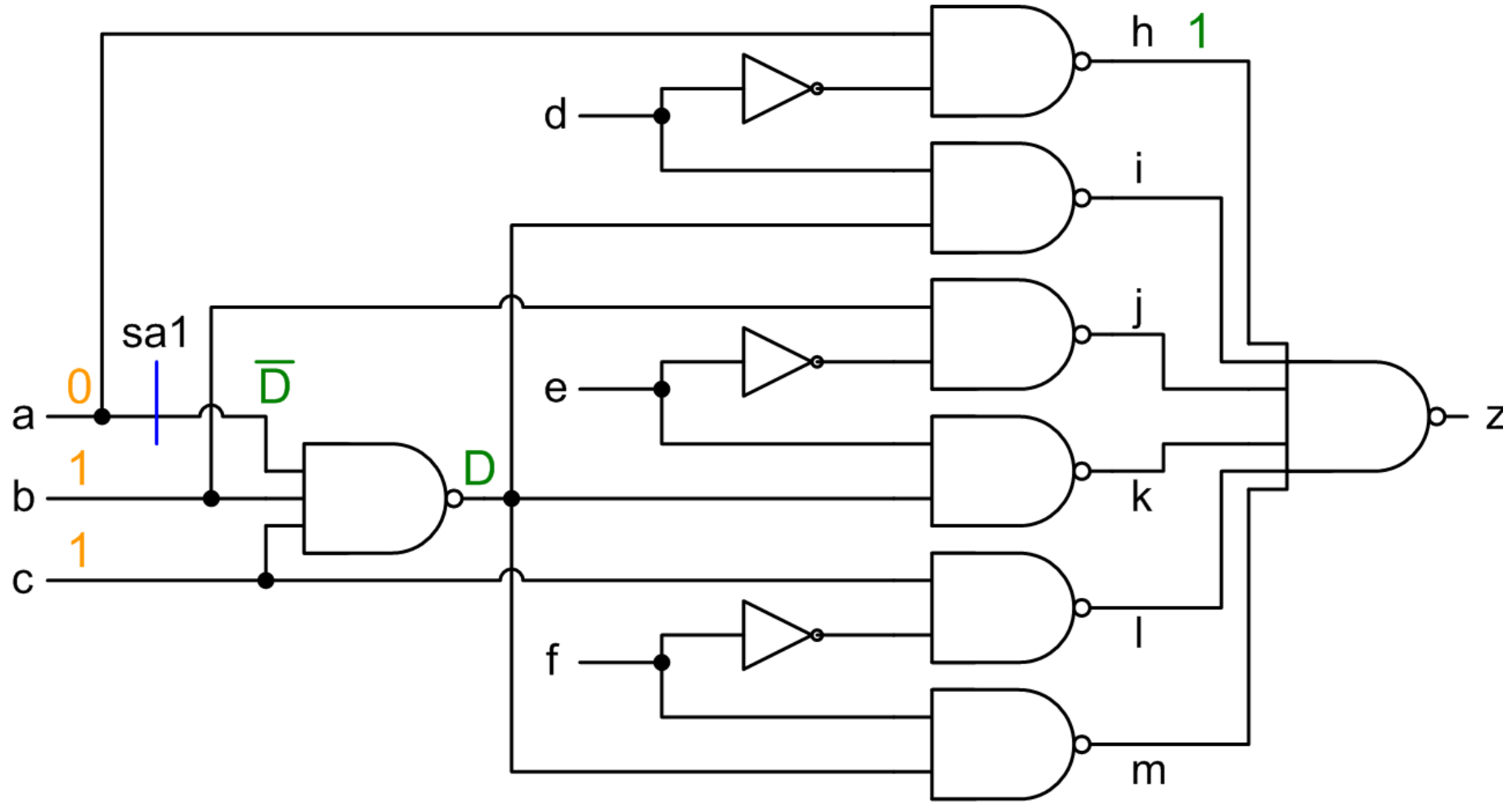

Fehlerinjektion durch a=0

# PODEM



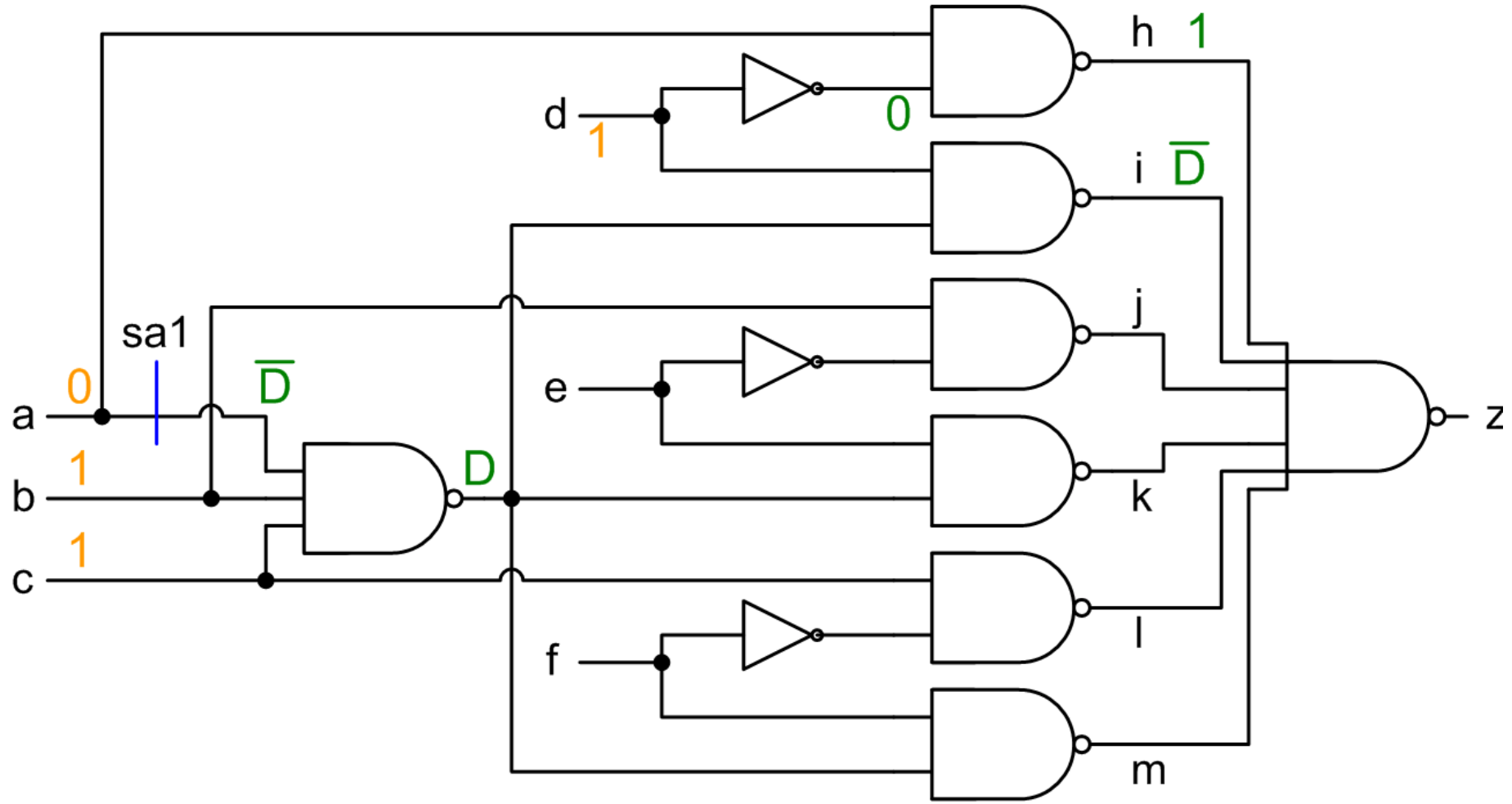

b=1

# PODEM



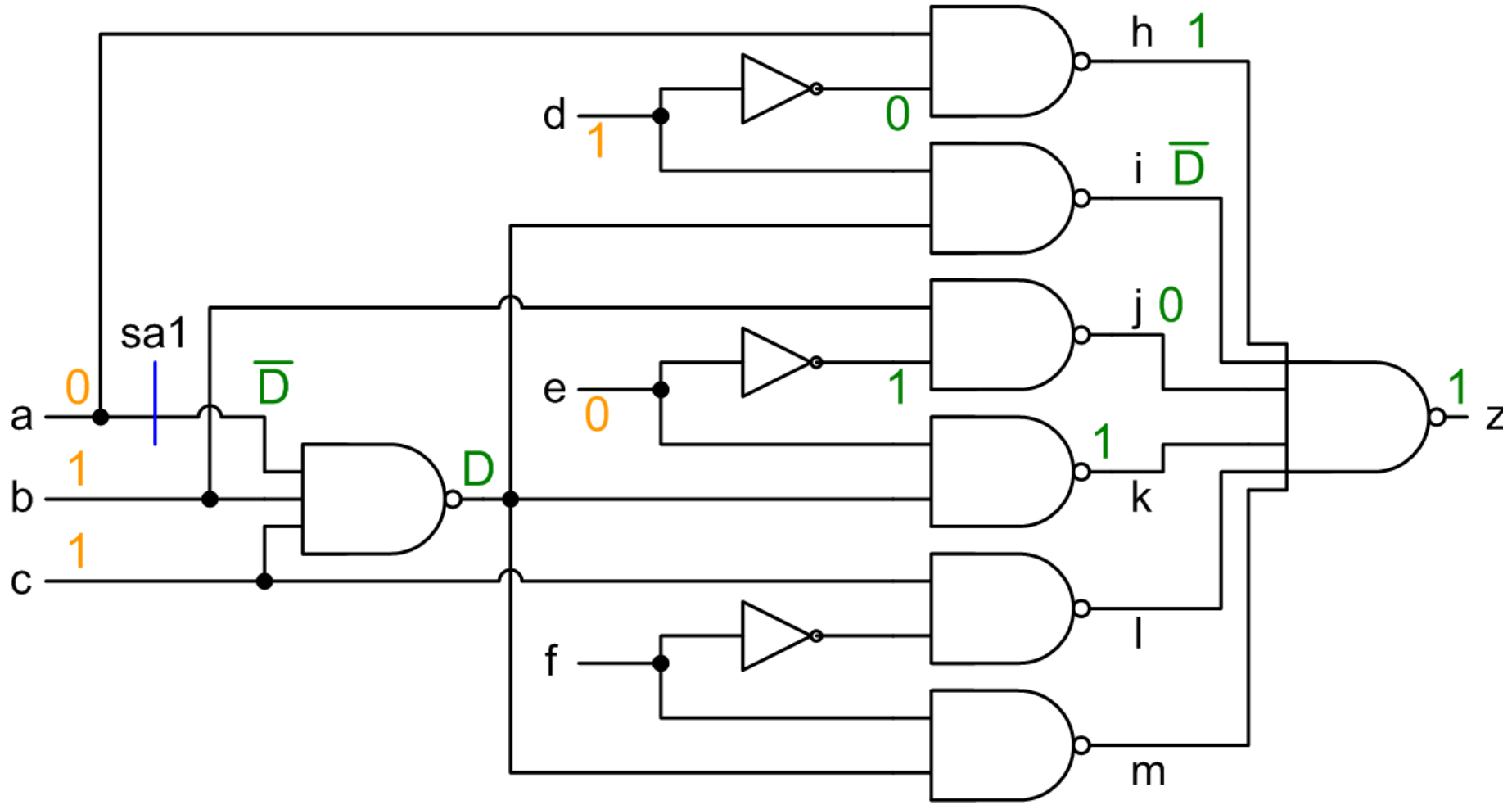

c=1

# PODEM



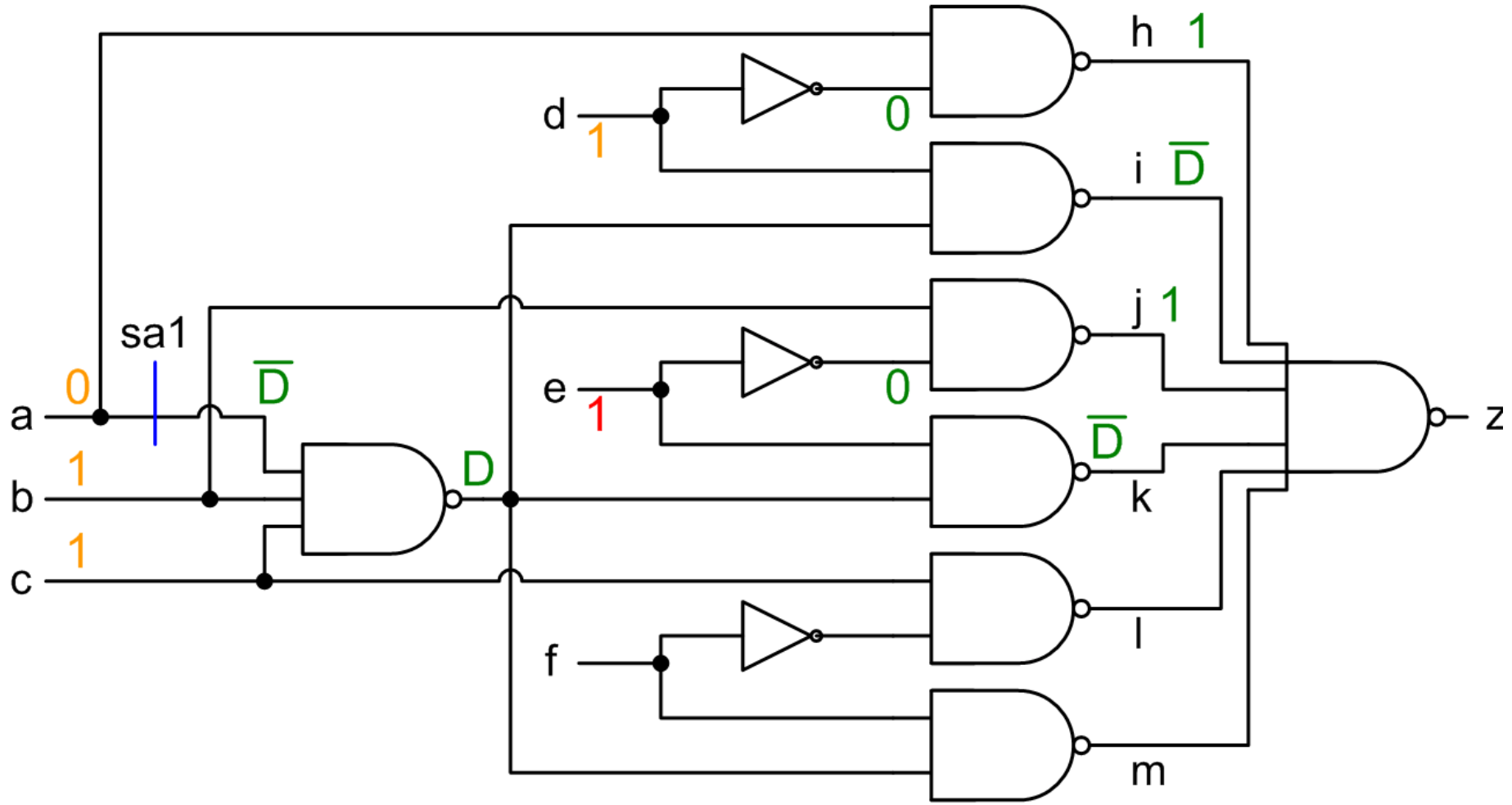

d=1 zur Propagation

# PODEM



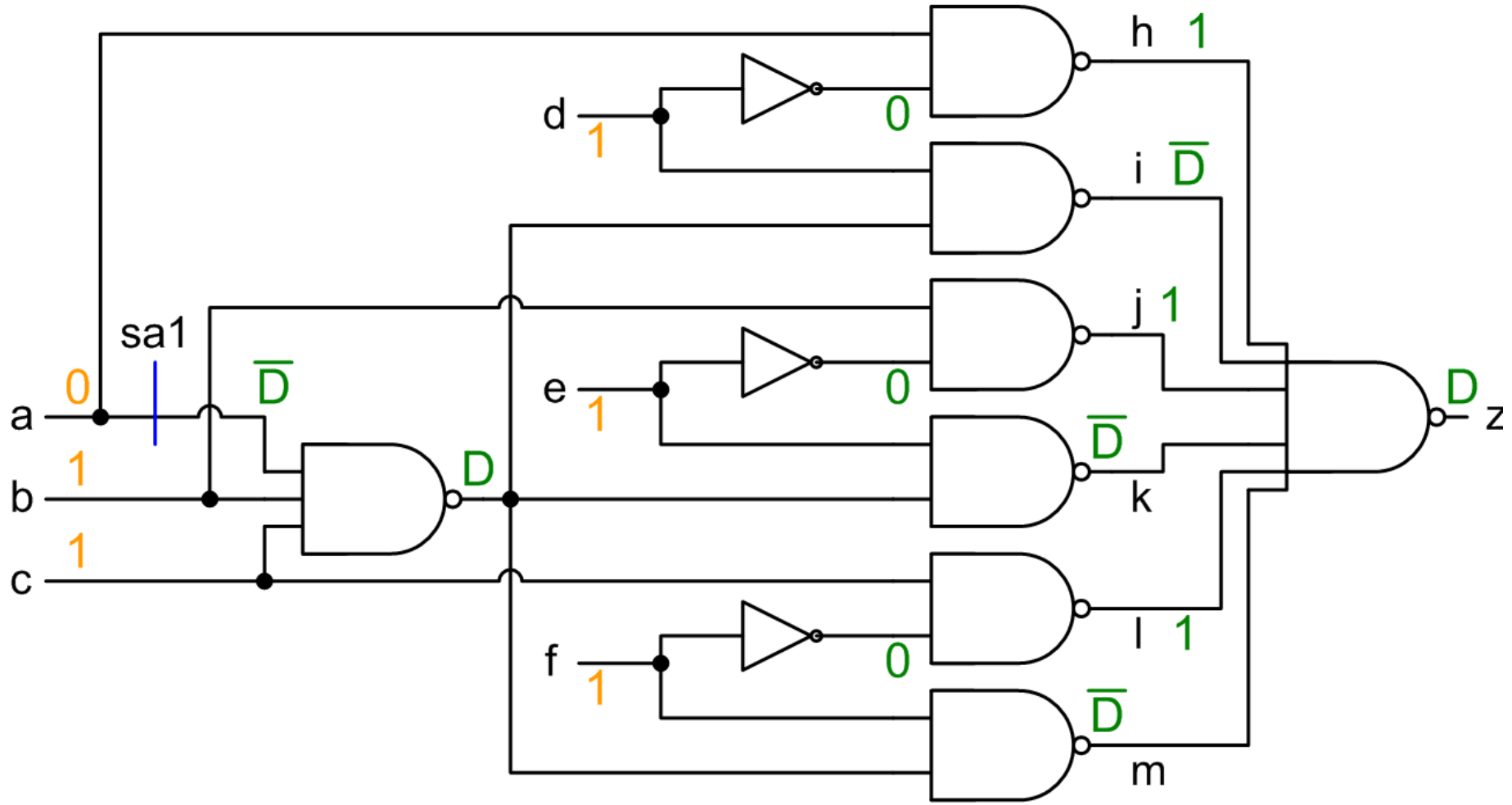

e=0 um j, k auf logisch 1 zu setzen → Implikation z=1

# PODEM




Backtracking, Neubelegung e=1

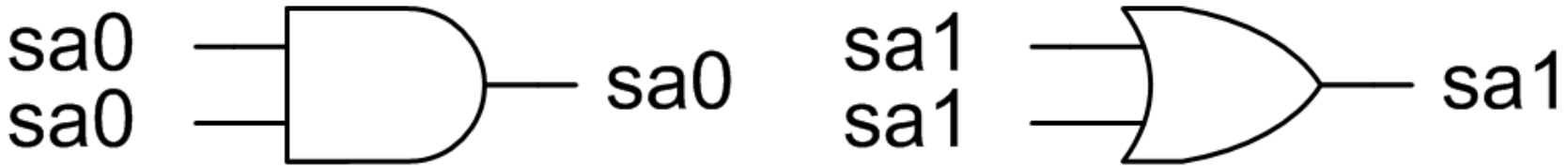
# PODEM




Versuch f=1 $\rightarrow$ z=D $\rightarrow$ Belegung gefunden

# Beschleunigung

- Beschleunigung der Berechnung durch Reduktion der Fehleranzahl
- Zusammenfassen von Fehlern mit gleicher Wirkung




- Beschleunigung der Ausführung durch Zusammenlegen von Testmustern,
  {X,0,1,1,X} und {1,X,X,1,0} können zu Testmuster {1,0,1,1,0} zusammengefaßt werden (statische Kompaktierung)

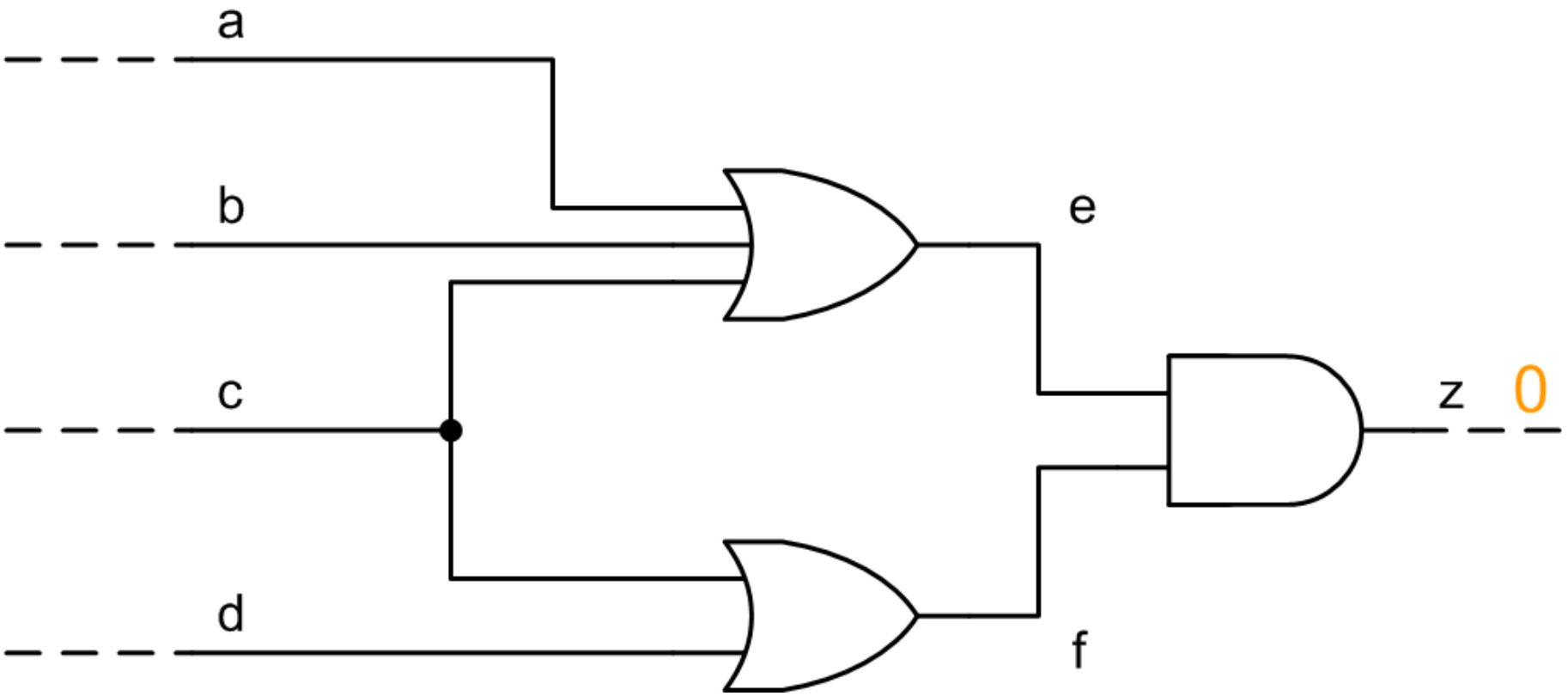
# Beschleunigung

- Schritt der Zusammenlegung kann auch beim Erzeugen von Testmustern direkt geschehen (dynamische Kompaktierung)

# Dynamic Decision Ordering

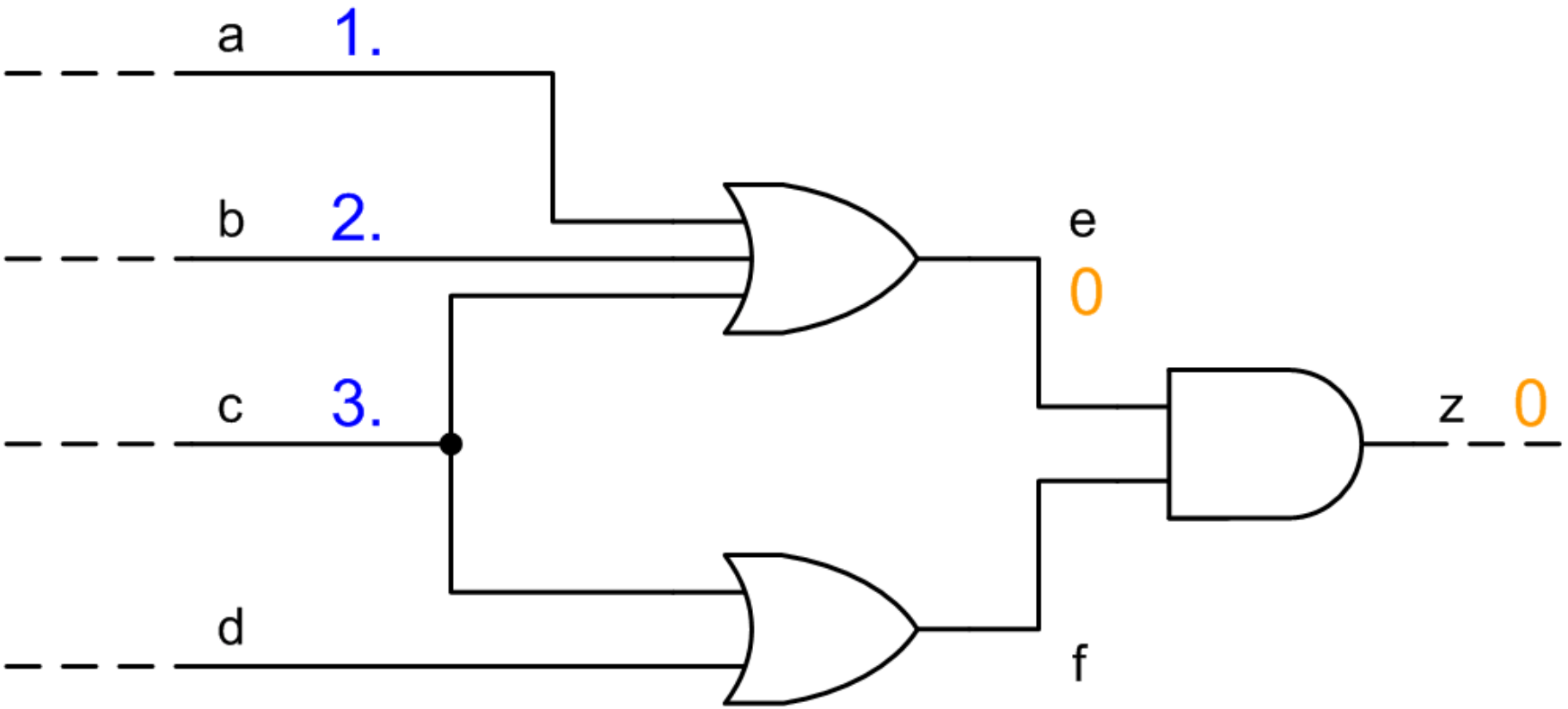
- kann einem Knoten während der Testmustergenerierung ein benötigter Wert nicht zugewiesen werden, so wird dieser Knoten für die Zukunft markiert
- müssen im Folgenden mehreren Knoten Werte zugewiesen werden, werden zuerst die Knoten untersucht die markiert sind

# Dynamic Decision Ordering



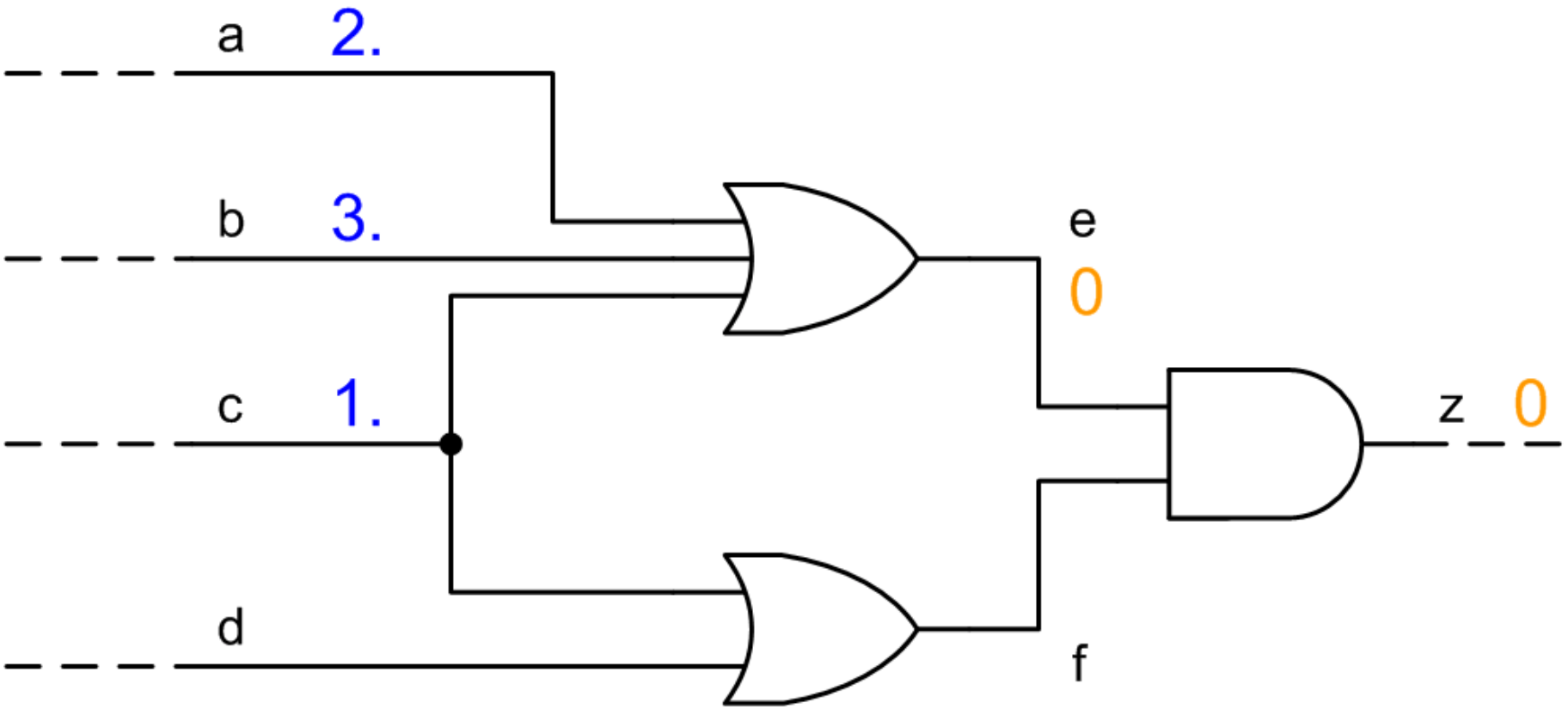

Festlegung z=0 bietet 2 Möglichkeiten: setze e=0 oder f=0

# Dynamic Decision Ordering



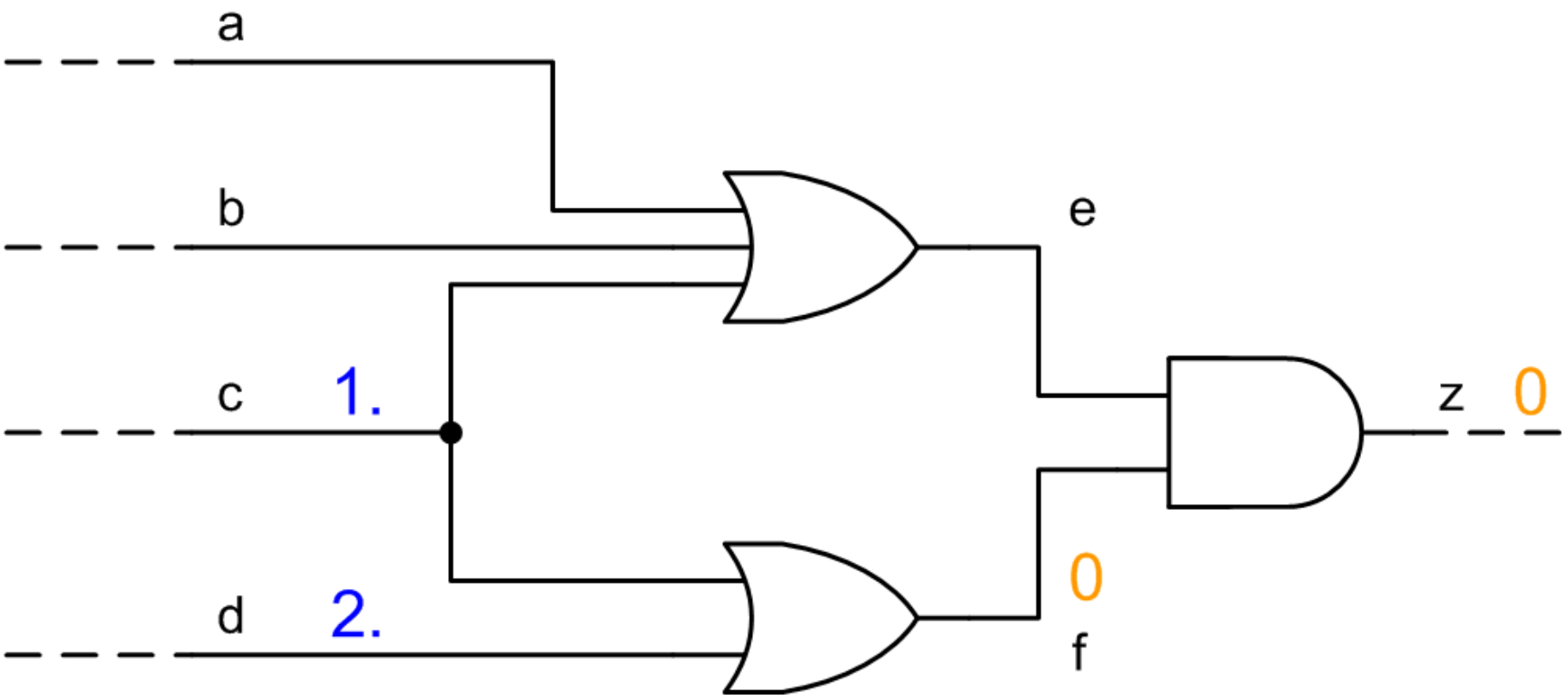

Festlegung e=0 erfordert a=0, b=0, c=0

# Dynamic Decision Ordering



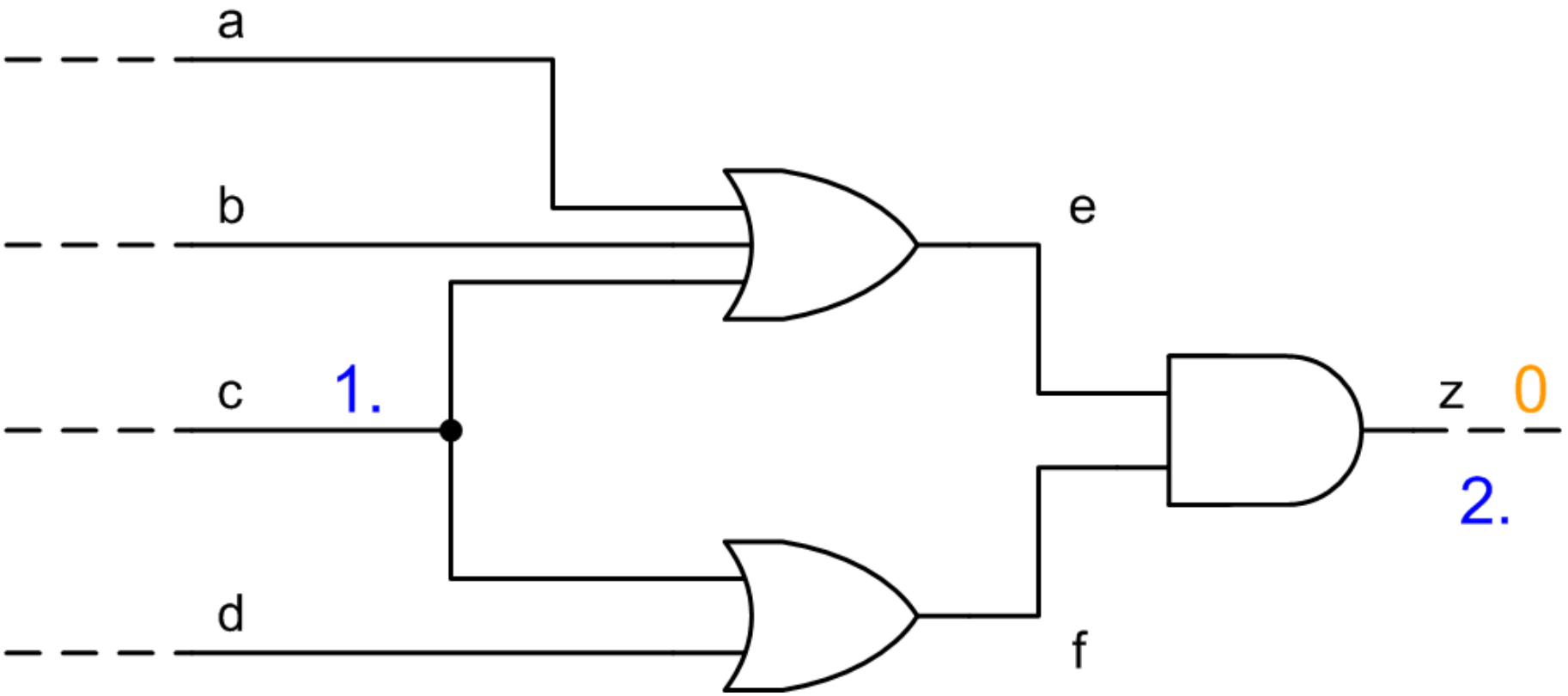

tritt bei c=0 ein Konflikt auf, wird die Bearbeitungsreihenfolge verändert und neu gestartet

# Dynamic Decision Ordering




der Knoten c wird markiert und bei folgenden Schritten, z. B. f=0, vor allen unmarkierten Knoten bearbeitet

# Conflict Driven Recursive Learning

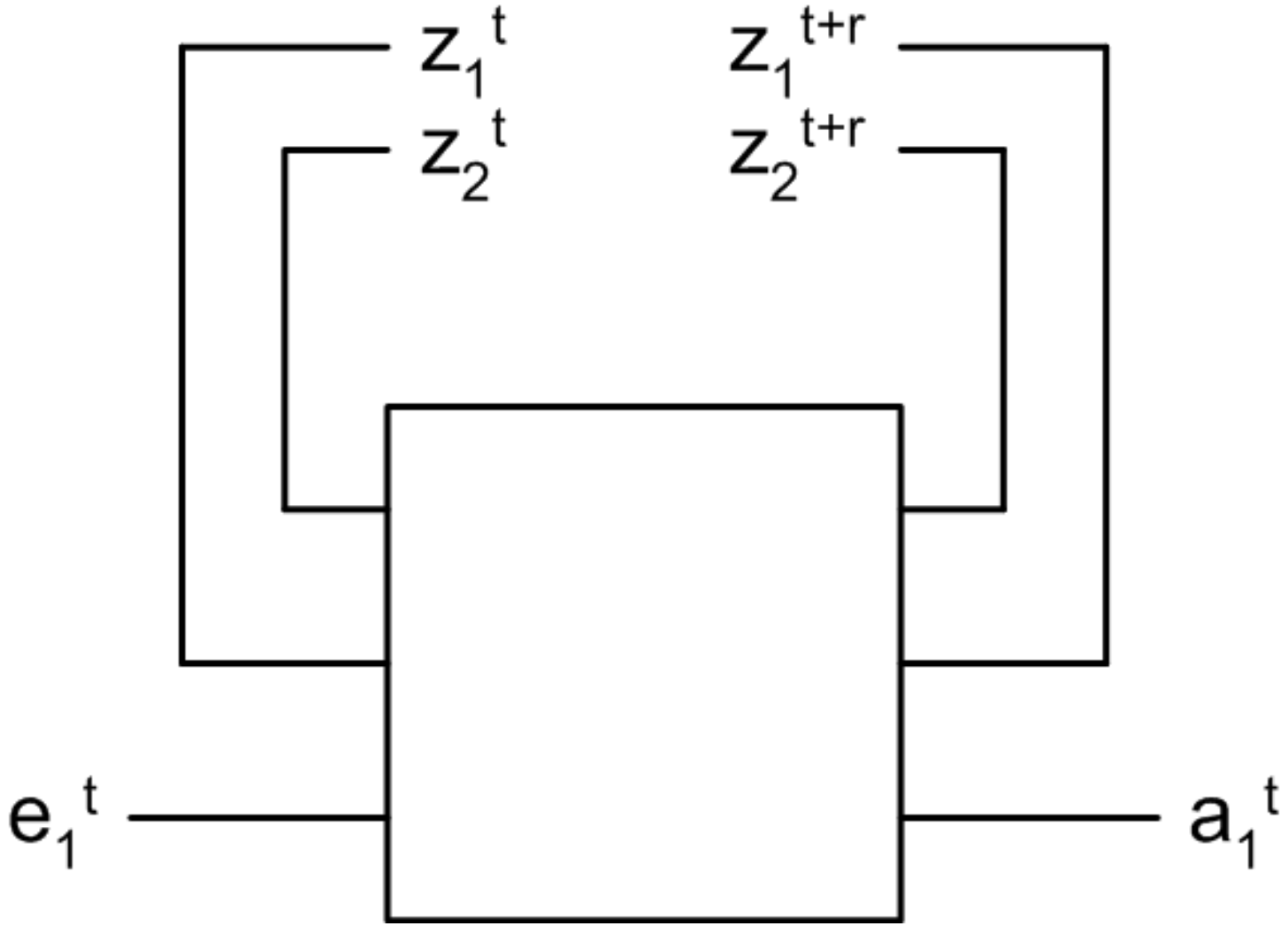



wird später erneut versucht z=0 zu setzen, wird zuerst die (für alle Alternativen zwingende) Bedingung c=0 bearbeitet

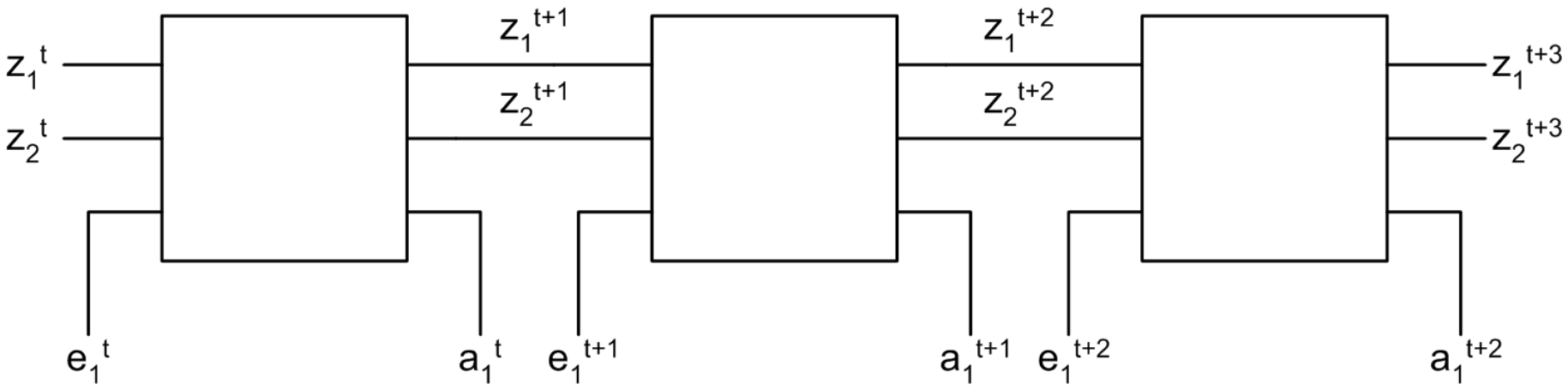
# Behandlung sequentieller Schaltungen

- Ausgangssignale können auch von zurückliegenden Eingangssignalen abhängen
- Einwirkung vorangegangener Eingangssignale spiegelt sich in Zuständen wieder
- Ziel ist die Modellierung sequentieller Schaltungen durch pseudokombinatorische
- Abbildung der zeitlichen Abfolge von Schaltungszuständen auf eine räumliche Anordnung

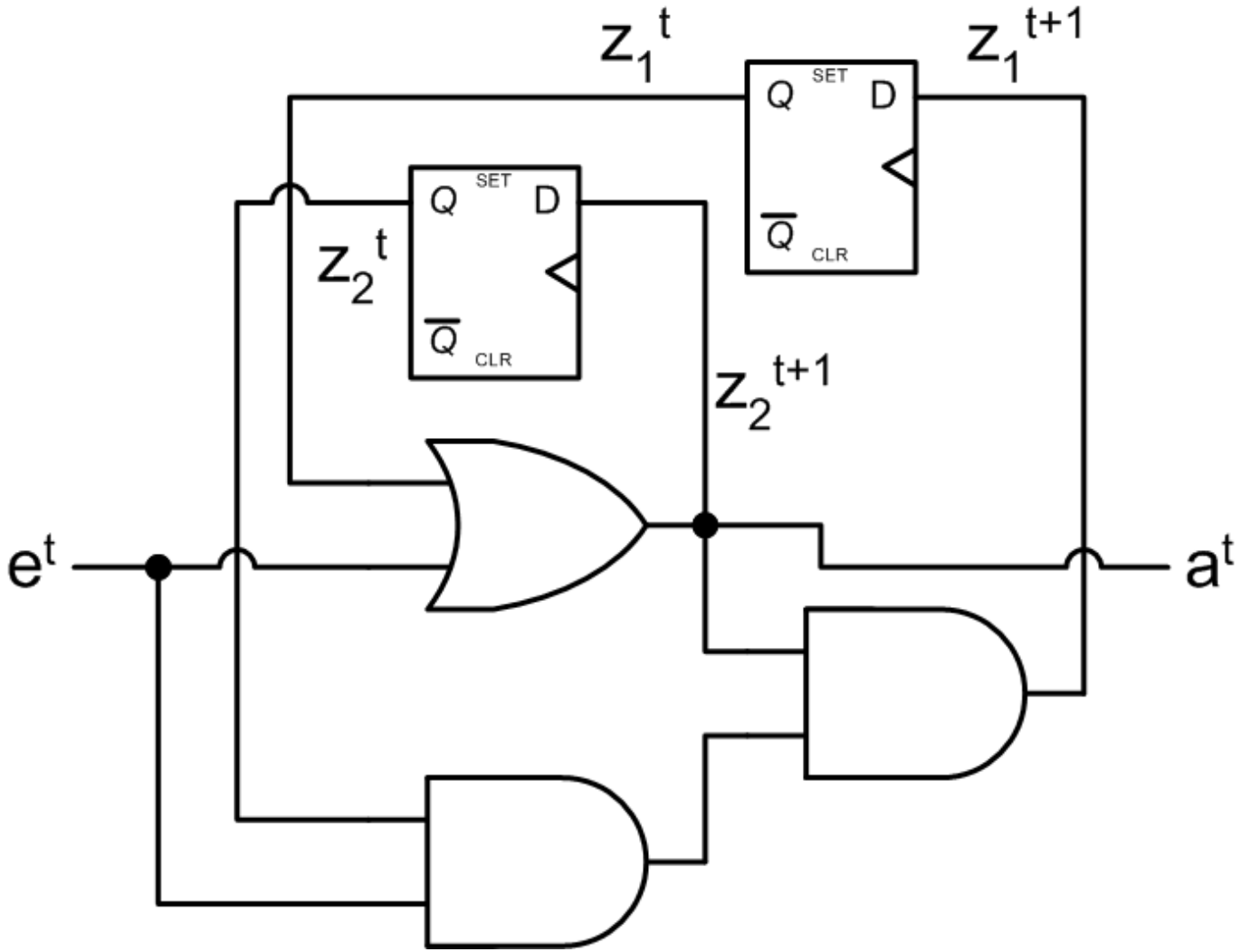
# Behandlung sequentieller Schaltungen



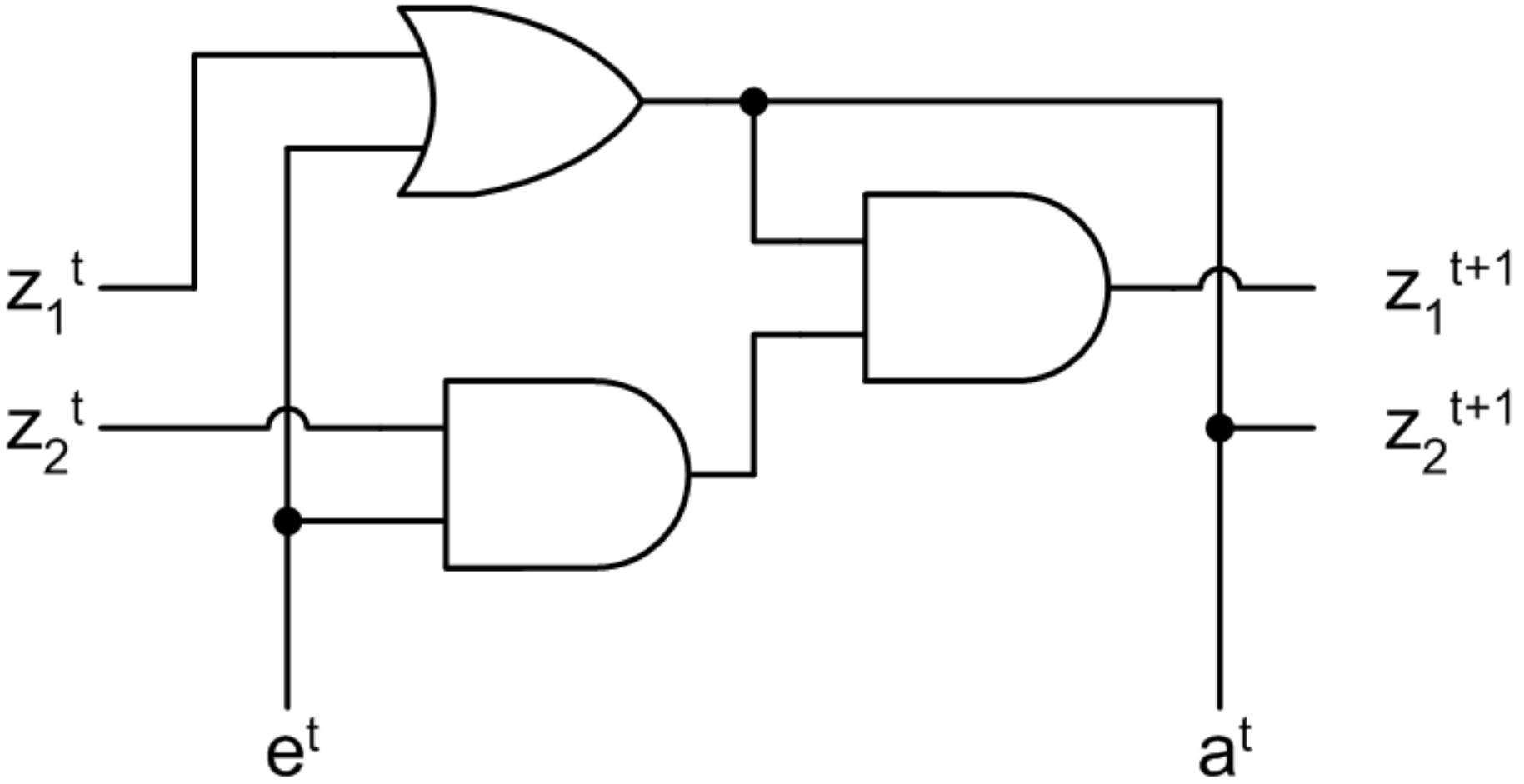
# Behandlung sequentieller Schaltungen



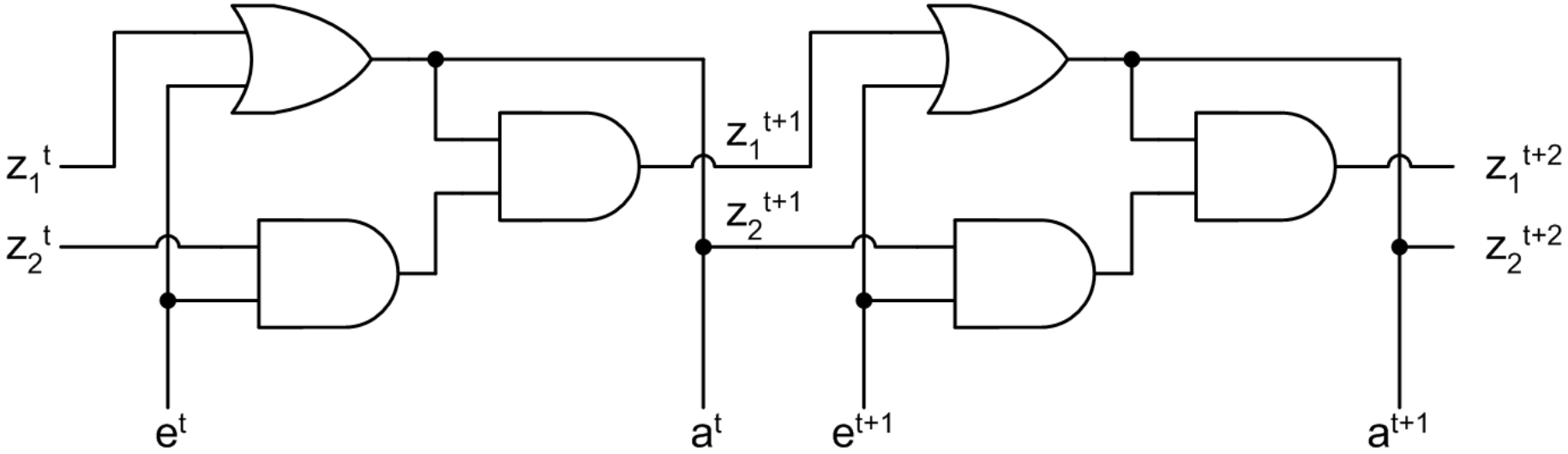
# Behandlung sequentieller Schaltungen



# Behandlung sequentieller Schaltungen



# Behandlung sequentieller Schaltungen



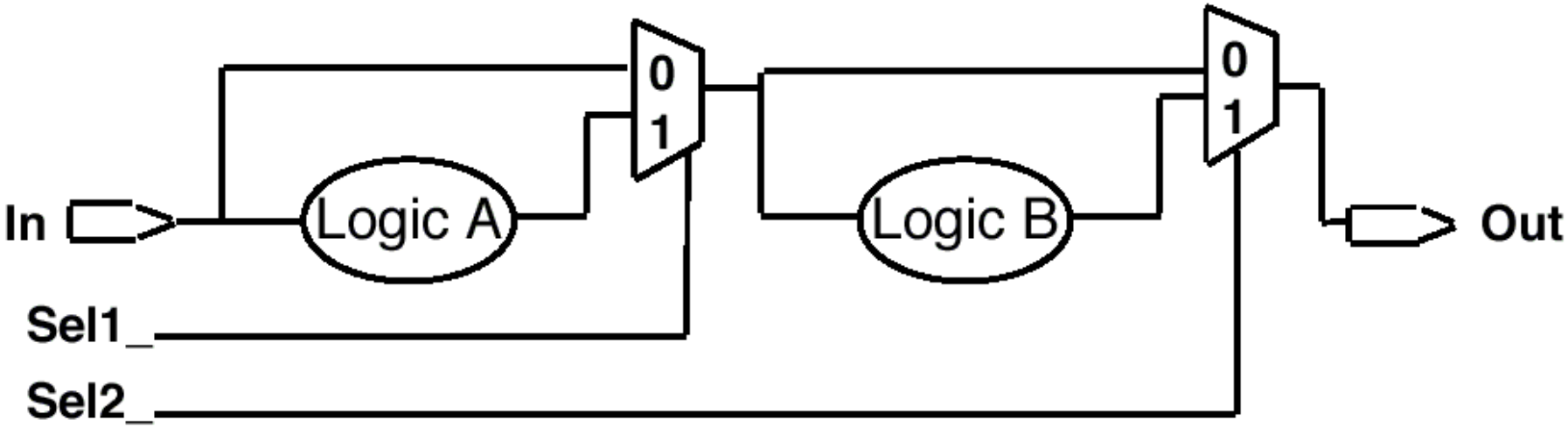
# Chiptest

- Motivation
- Fehler in digitalen Schaltungen
- Tests digitaler Schaltungen
- (Automatische) Testmustergenerierung
- Design for Testability
- Beispiele

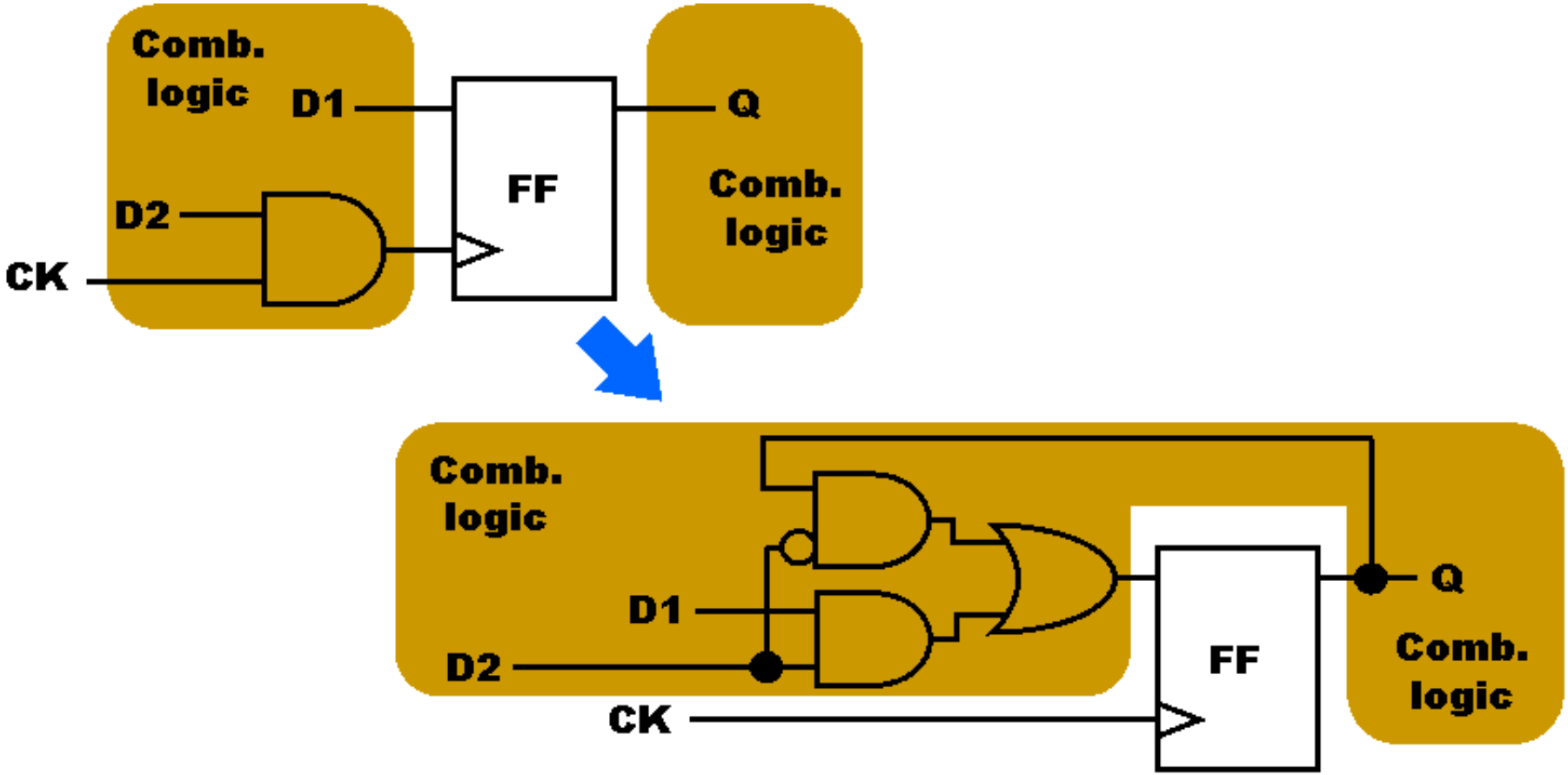
# Design for Testability

- Verbesserung der Testbarkeit zur Designphase
  - Partitionierung
  - Initialisierungsleitungen
  - Einfügen von Testpunkten
  - Scan-Technik
  - Build-in-Self-Test

# Partitionierung



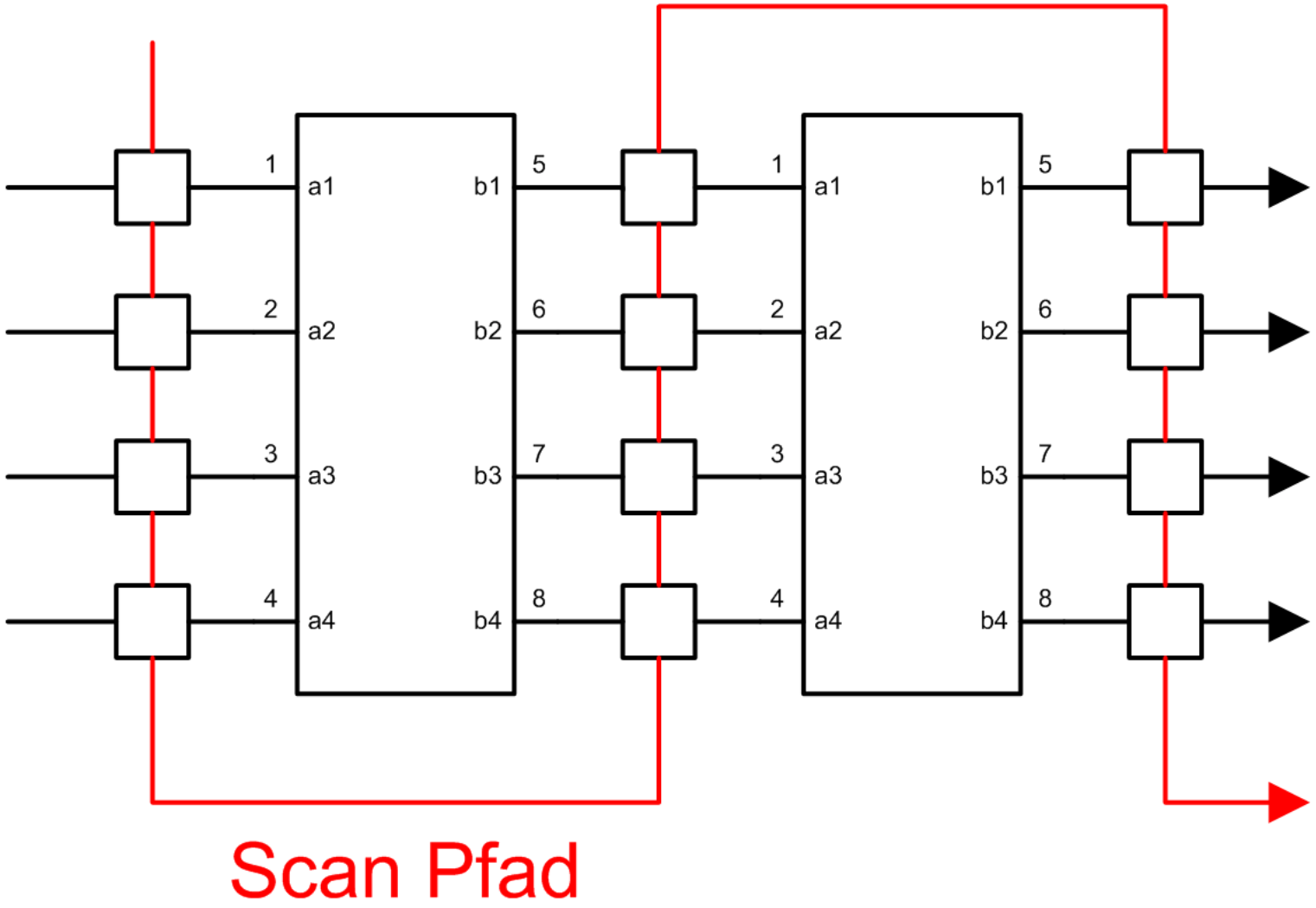
# Clock-Signale



# Scan-Technik

- Einbau zusätzlicher „Scan-Zellen“ (Register) in eine Schaltung
- Scan-Zellen können logischen Wert eines Knotens speichern oder einen Wert an seinen Ausgang legen
- Scan-Zellen sind seriell zu einem Schieberegister, dem „Scan-Pfad“, verbunden

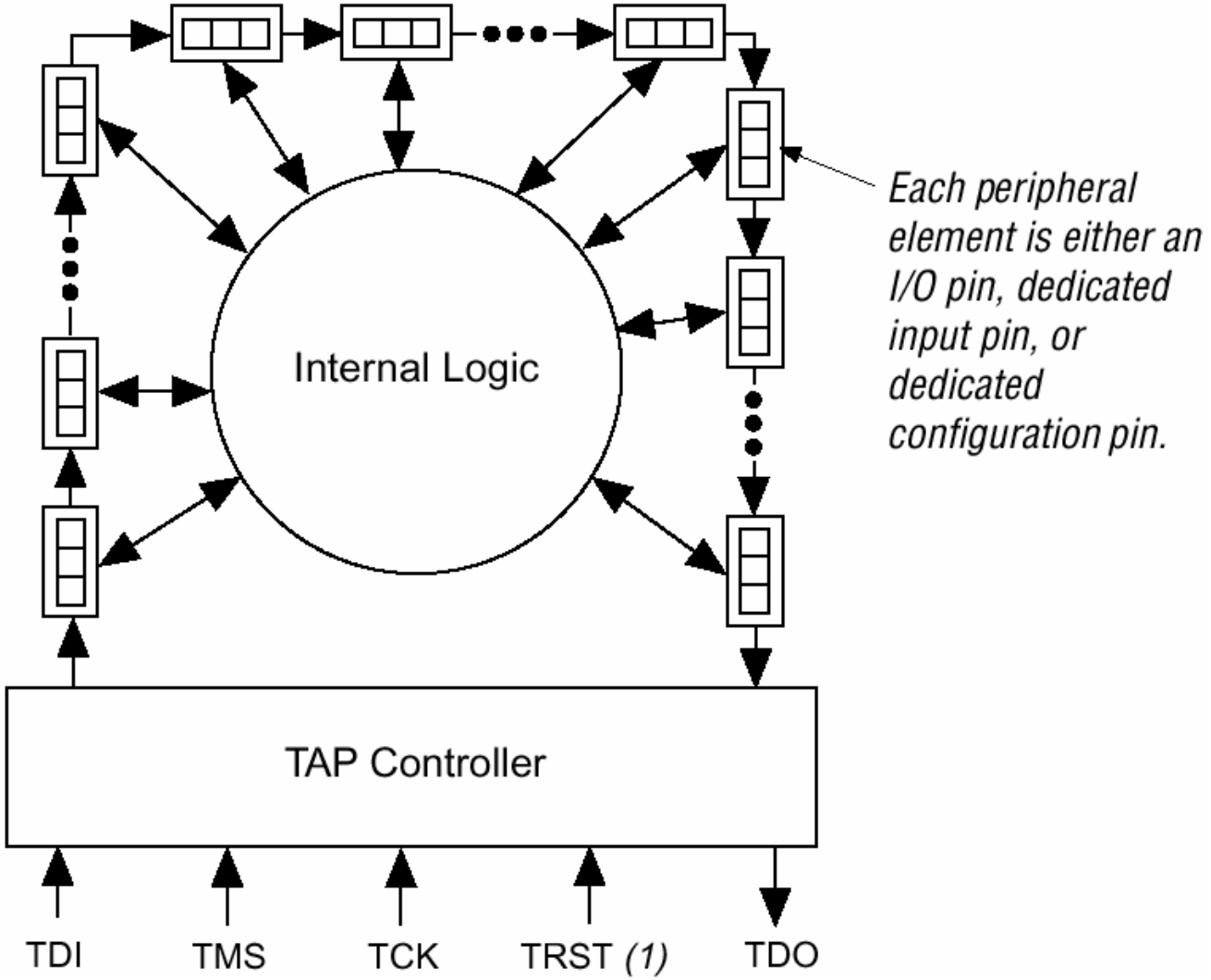
# Scan-Pfad



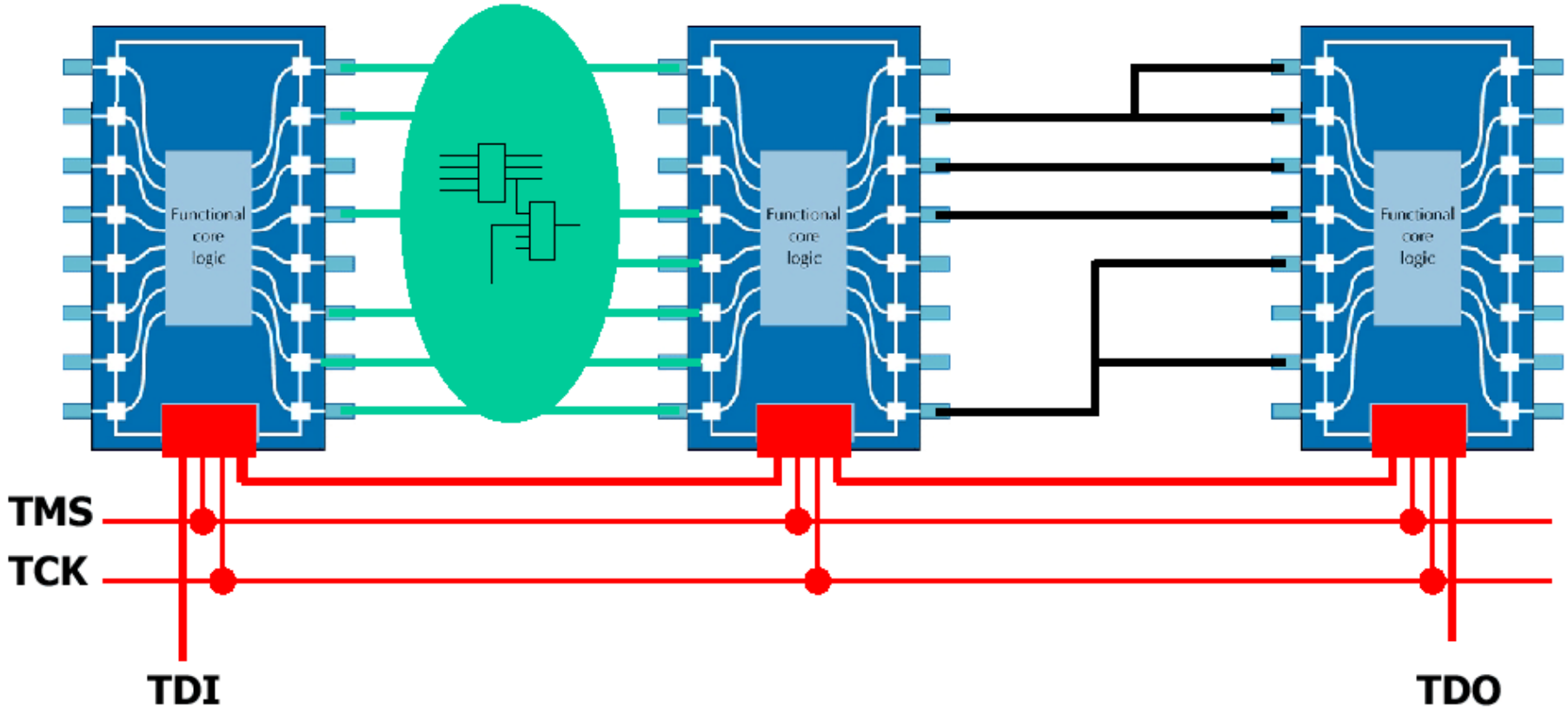
# Boundary-Scan-Standard

- Standardisierung der Scan-Technik (IEEE 1149.1)
- Ermöglicht die Kombination von ICs verschiedener Hersteller
- Standardisierte Testmethoden und Testausrüstungen

# Boundary-Scan-Standard



# Boundary-Scan-Standard



# Build-in-Self-Test (BIST)

- Ermöglicht der Schaltung sich selbst zu Testen
- Vorteilhaft, wenn am Einsatzort der Schaltung keine Testwerkzeuge zur Verfügung stehen
- Selbsttests werden in einem eigenen Vortrag behandelt

# Chiptest

- Motivation
- Fehler in digitalen Schaltungen
- Tests digitaler Schaltungen
- (Automatische) Testmustergenerierung
- Design for Testability
- Beispiele

# UltraSparc

- Boundary Scan, Full Scan (alle Flip-Flops in Scan-Pfad), ein Scan-Pfad, 22.000 Elemente
- BIST über Scan-Pfad erreichbar
- dadurch 3,5% zusätzlicher Flächenbedarf
- Verilog-Dateien (auf Gate-Level) als Input zur Testmustererstellung
- Anforderung min 95% Fehlerabdeckung
- Stuck-At-Fehlermodell, Einzelfehlerannahme

# Intel Pentium Pro

- Boundary Scan
- DFT führte zu 4% Flächenzunahme für die CPU, 6% für den Level 2-Cache
- Stuck-At-, Kurzschluss-, Delay-Fehlermodell
- 96% Fehlerabdeckung für Stuck-At-Fehlermodell
- 8 Monate für Debugging, Test-Entwicklung und Produktionsvorbereitung

# AMD K6

- Boundary Scan, Full Scan, 4 Scan-Pfade, 37.000 Elemente
- BIST, Flächenzunahme 3-4%
- Stuck-At-, Kurzschluss-Fehlermodell
- zusätzlich IDDQ-Tests

# Quellen

- Christian Hähnel

  Test und Simulation der Track-Finding-Unit des HERA-B-First-Level-Triggers

  leonardo.statistik.uni-mannheim.de/dr/dr.html

- Hans Wojtkowiak

  Test und Testbarkeit digitaler Schaltungen

- Kewal K. Saluja

  Testing and Testable Design of Digital Systems

  www.cae.wisc.edu/~saluja/ECE553/INFO.html

# Quellen

- E. M. Rudnick

  Microprocessor Verification, Test, and Design

  www.crhc.uiuc.edu/ECE371EMR/

- Bob Klenke

  Test Technology Overview

  www.cedcc.psu.edu/ee497f/rassp_43/index.htm

- Vishwani D. Agrawal

  VLSI Testing

  www.ece.wisc.edu/~va/COURSE/lectures.html

# Quellen

- Nicola Nicolici

  SOC Design and Test

  www.ece.mcmaster.ca/~nicola/ece744/

- Rochit Rajsuman

  Iddq Testing for CMOS VLSI

  www-micrel.deis.unibo.it/AFFI/rajsuman2000.pdf