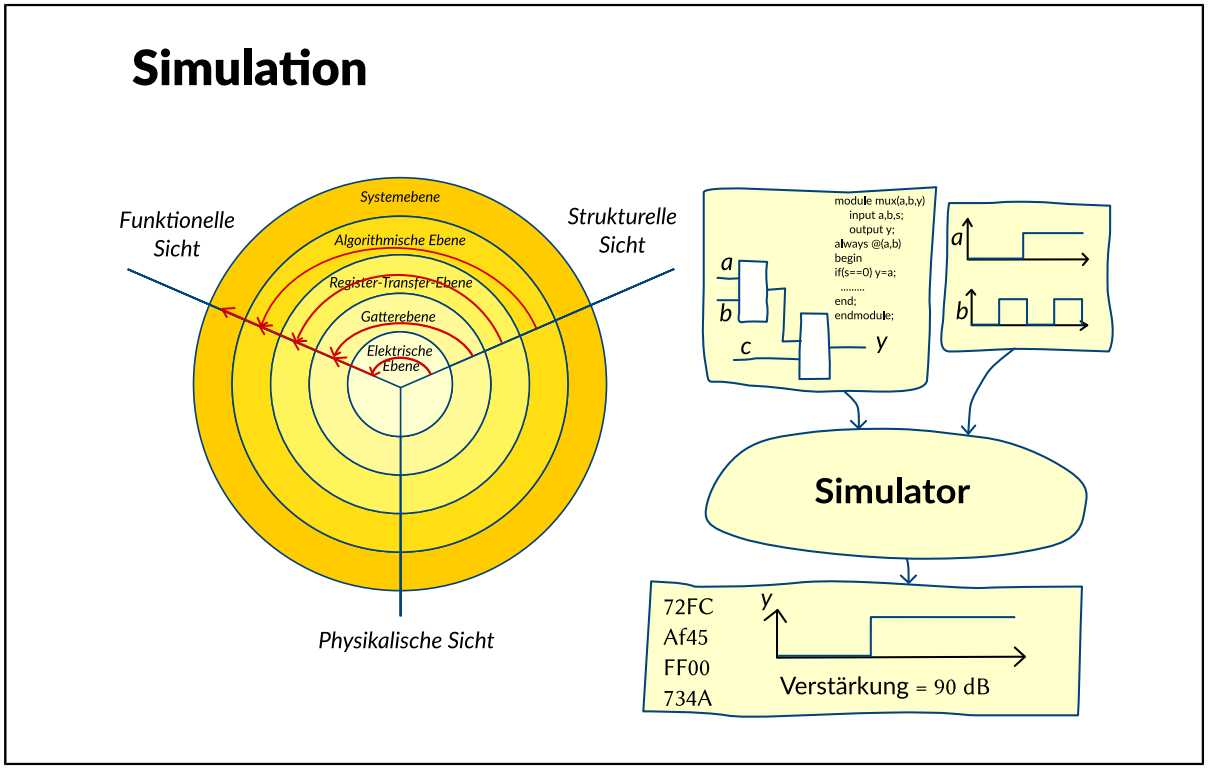
Electronic Design Automation (EDA)

Digitale Simulation

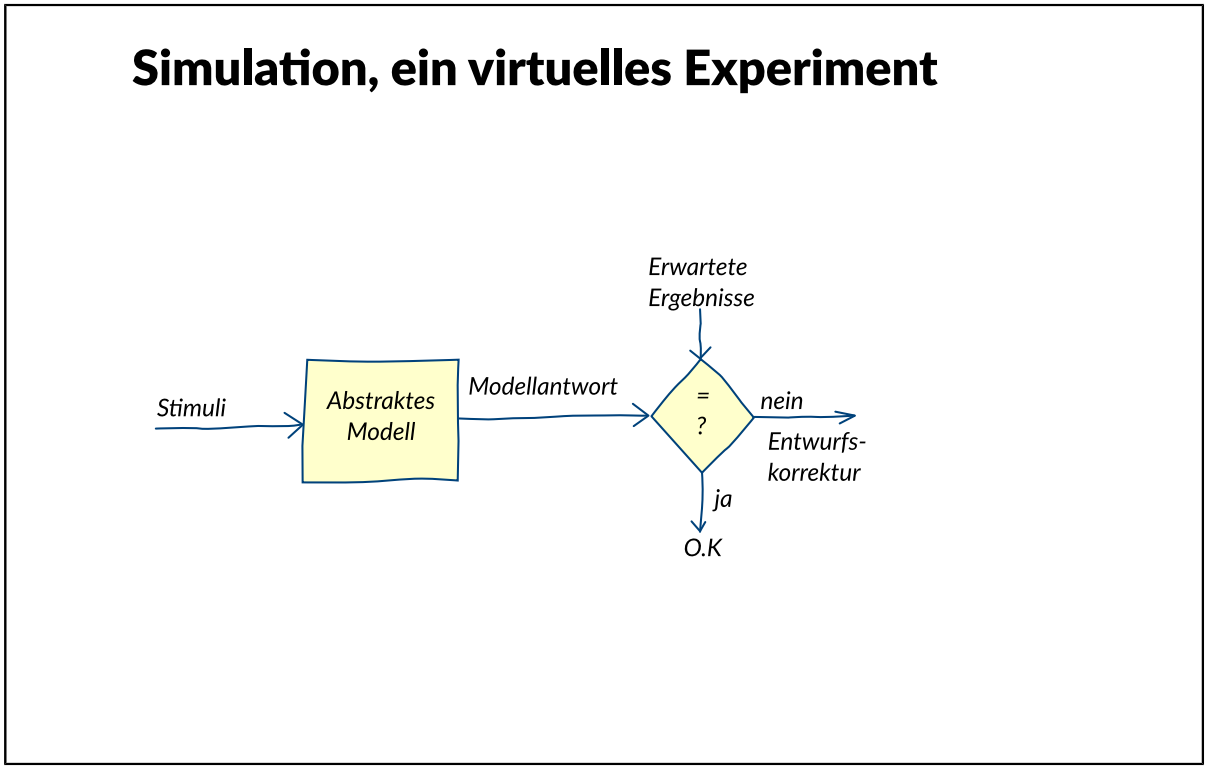
Simulation
Simulation, ein virtuelles Experiment
Simulationsebenen
Abstraktion
Wertdiskretisierung: Logikmodelle
Glitch
Delay-Modell I
Delay-Modell-II
Digitale Simulationsmethodik
Übersetzende Verfahren
Strukturierung
Compiled Code: Beispiel
Talbellengestützte Verfahren
Verkettete Datenstrukturen: Elemente
Verkettete Datenstrukturen: Beispiel
Äquitemporale Iteration
Ereignisse
Ablauf ereignisgesteuerter Simulation
Ereignisverwaltung
Ereignissteuerung: Beispiel
Multi-Level-Simulation

Digitale Simulation: Simulation



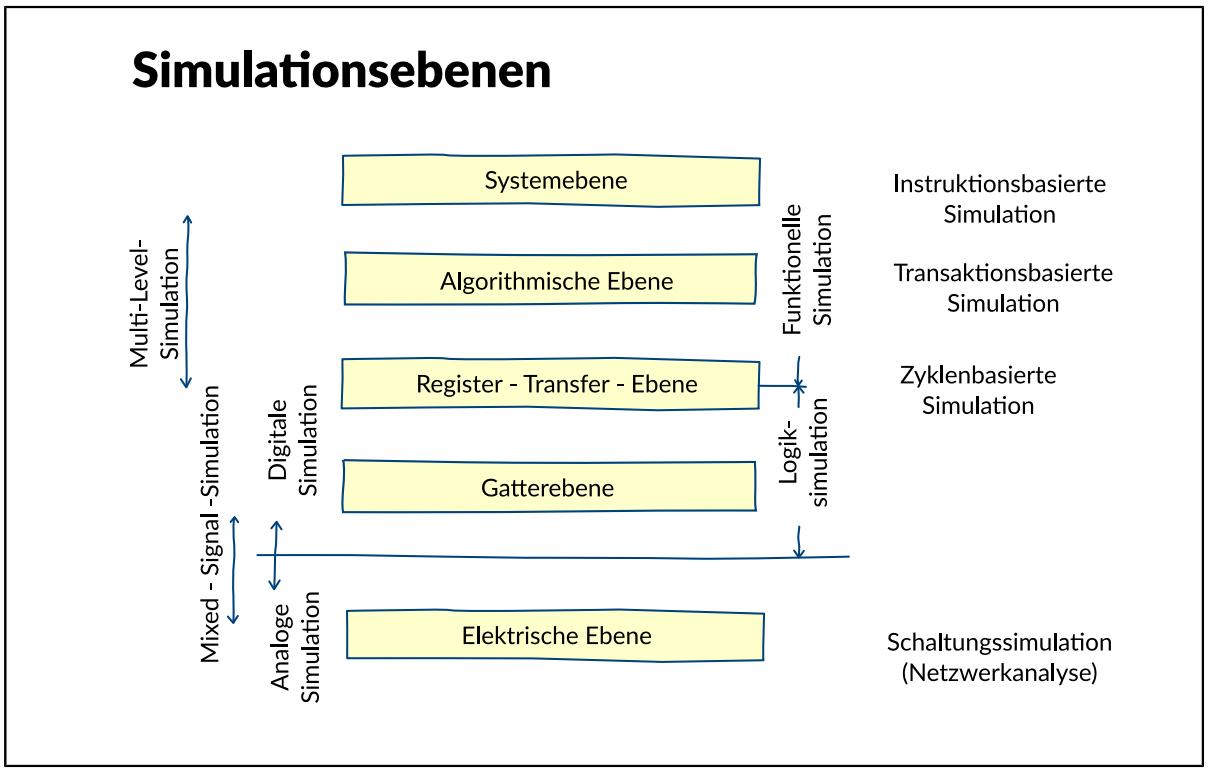
Simulation bedeutet, ein (Software-)Modell der Schaltung zu benutzen, um ihr Verhalten zu analysieren. Dazu werden die Eingänge des Schaltungsmodells mit Signalwerten (so genannten Stimuli) belegt und die berechneten Werte an den Ausgängen (oder auch an beliebigen Stellen innerhalb der Schaltung) beobachtet.

Digitale Simulation: Simulation, ein virtuelles Experiment



Simulation ist ein virtuelles Experiment. Zu seiner Durchführung versorgt der Entwickler das Modell der Schaltung mit sinnvollen Eingangswerten und überprüft durch die Simulation, ob am Ausgang der Schaltung die erwarteten Werte erscheinen. Es wird ein Testfall konstruiert und durch Simulation nachgewiesen, dass für diesen Testfall die Spezifikation erfüllt ist. Natürlich wird versucht, durch möglichst viele Testfälle alle Eventualitäten zu überprüfen. Die Menge aller möglichen Testfälle ist aber außer für sehr kleine Schaltungen viel zu groß, als dass sie jemals durch Simulation abgedeckt werden könnte. Ein 16-Bit-Addierer hat zum Beispiel 2^32 mögliche Eingangswerte. Ungefähr 4,3 Milliarden Testfälle müssen durchgerechnet werden, nur um einen einzigen 16-Bit-Addierer vollständig zu testen. Das im Begriff der Verifikation enthaltene Versprechen - der Beweis der Wahrheit - wird von Simulatoren nicht eingehalten. Simulation kann Fehler finden, aber nicht feststellen, ob eine Schaltung fehlerfrei ist.

Digitale Simulation: Simulationsebenen



Simulatoren sind auf allen Entwurfsebenen sehr wichtige EDA-Werkzeuge. Je nach Ebene sind die verwendeten Verfahren und Algorithmen jedoch sehr unterschiedlich. Auf der elektrischen Ebene löst ein **Schaltungssimulator** (historisch auch **Netzwerkanalyseprogramm** genannt) das eine elektrische Schaltung beschreibende System von nichtlinearen Differentialgleichungen. Dies ist für Analogschaltungen die sinnvolle Simulationsebene, auf der das Wert- und Zeitverhalten detailliert analysiert werden kann. Für größere - insbesondere digitale - Schaltungen sind Schaltungssimulatoren wegen ihrer hohen Rechenzeitanforderungen ungeeignet. Es muss daher unter Verzicht auf Genauigkeit eine höhere Abstraktionsebene gewählt werden.

In den 50er und 60er Jahren hat sich die **digitale Simulation** primär mit der Korrektheit der Funktion und kaum mit dem Zeitverhalten befasst. Die früheste Methode zur **Logiksimulation** ist die **"Compiled-Code"**-Methode, die später erläutert wird und heute vorwiegend für höhere Ebenen verwendet wird. Eine digitale Schaltung wird durch boolesche Gleichungen beschrieben und diese werden in jedem Taktschritt ausgewertet. Dieses Vorgehen ist sehr rechenzeitintensiv und nur für synchrone Schaltungen geeignet. Zudem wird das Zeitverhalten der Strukturelemente nicht berücksichtigt.

Mitte der 60er-Jahre wurde ein neues Verfahren, die **ereignisgesteuerte Logiksimulation**, vorgestellt, die sich schnell durchsetzte, da sie wesentlich schneller ist und die Simulation asynchroner Schaltungen sowie die Berücksichtigung von individuellen Verzögerungszeiten erlaubt. Sie vermeidet die Untersuchung solcher Schaltungsteile, die zu bestimmten Zeiten inaktiv sind. Digitale Schaltungen sind bis zu 90 % inaktiv.

Wegen der großen Komplexität der Schaltungen wird heute auf der Gatterebene wieder zwischen Funktion und Zeitverhalten getrennt. Es wird eine rein funktionelle Simulation durchgeführt und das Zeitverhalten mithilfe der statischen Timinganalyse analysiert. Diese Trennung von Funktion und Zeitverhalten hat sich auch auf den höheren Entwurfsebenen bewährt. Simulatoren auf diesen Ebenen sind weniger genau als Gatterebenensimulatoren, ihre Handhabung ist jedoch einfacher und sie sind erheblich schneller. Ihre Geschwindigkeit erreichen sie durch die Abstraktion des Zeitverhaltens, indem beispielsweise die Funktion nur zum Zeitpunkt einer Taktflanke analysiert wird (**zyklenbasierte Simulation**), sowie durch das Arbeiten mit Mehrbit- statt Einzelbitvariablen.

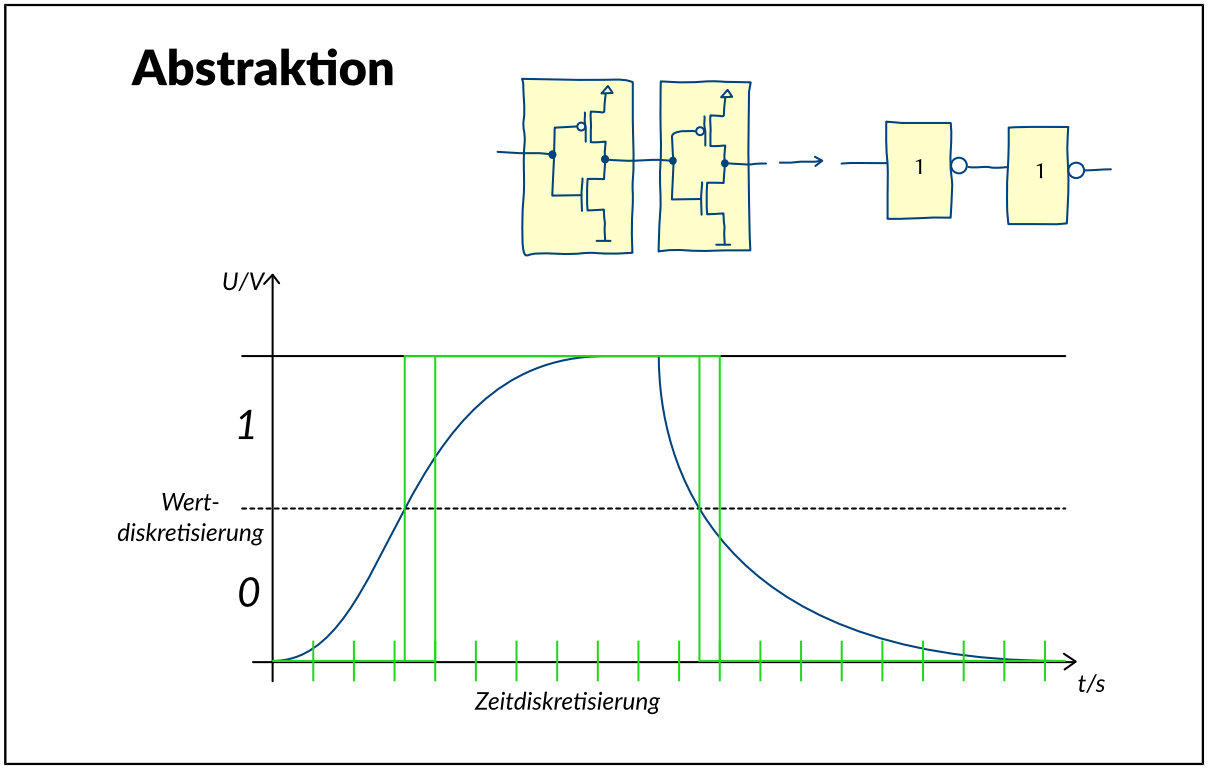
Zyklenbasierte Simulatoren sind der Register-Transfer-Ebene zuzuordnen. In der Ebene darüber nimmt die Abstraktion auch bei der Simulation weiter zu. So werden auf der algoritmischen Ebene nicht mehr die Werte einzelner Variablen betrachtet, sondern das Verhalten mehrerer Variablen in Form von sogenannten **Transaktionen**, wie z.B. das Einschreiben eines Datenwortes in einen Speicher, oder der Zugriff auf einen Bus mit dem zugehörigen Datentransfer.

Auf der Systemebene existieren schließlich noch sogenannte **Instruction Set**-Simulatoren, die den Instruktionssatz eines entworfenen Prozessors per Simulation nachbilden und damit in der Lage sind, grundsätzlich auf diesem "virtuellen" Prozessor auch Software ablaufen zu lassen. Dies ist aus Komplexitätsgründen aber nur für relativ kurze Programmteile möglich.

Da die Simulationszeit stets problematisch ist, entstanden für solche Fälle, in denen auf die Genauigkeit der Gatterebene nicht verzichtet werden konnte, so genannte **Hardwarebeschleuniger** und **Logikemulatoren**, bei denen der Simulationsalgorithmus in Hardware abgebildet wird. Es handelt sich also um spezielle Rechner, die mit großer Geschwindigkeit einen festverdrahteten bzw. anwenderprogrammierbaren Ablauf ausführen.

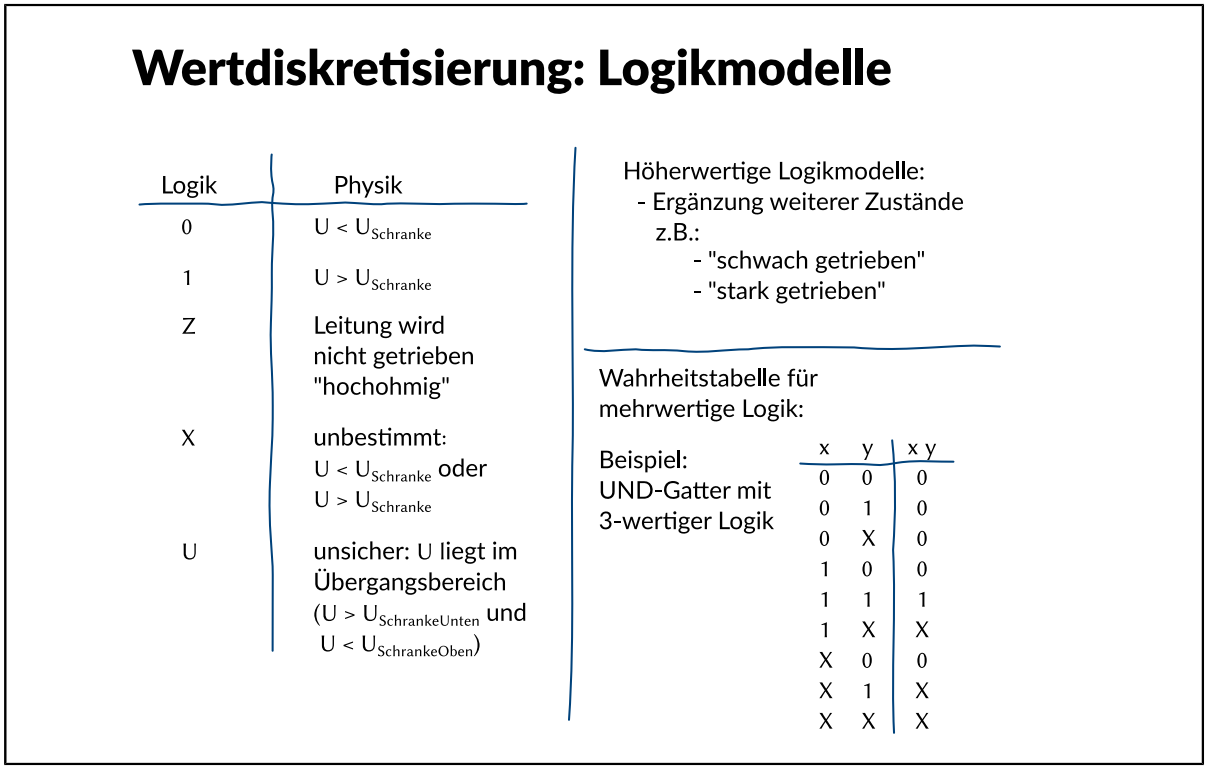
Simulation ist ein umfangreiches Gebiet und kann hier nicht vollständig behandelt werden. Die folgenden Betrachtungen werden sich deshalb auf die Gatterebene beschränken. Wegen ihrer grundlegenden Bedeutung soll dabei insbesondere die verzögerungszeitberücksichtigende Logiksimulation betrachtet werden. Alle weiter abstrahierende Simulatoren können ohne weiteres auf die hier beschriebenen zurückgeführt werden.

Digitale Simulation: Abstraktion



Für die Simulation auf Gatterebene muss die physikalische Realität mit kontinuierlichen Signalverläufen stark abstrahiert werden. Sowohl der Wertebereich als auch der Zeitliche Verlauf der Signale werden diskretisiert. In der einfachsten Form der digitalen Simulation kann ein Signal nur noch die Werte 1 und 0 annehmen. Der Simulator muss zu einem gegebenen diskreten Zeitpunkt alle Signalwechsel auswerten, sich unter Berücksichtigung der Gatter- und Leistungsverzögerungszeiten deren Auswirkung für zukünftige diskrrete Zeitpunkte merke und dann die Simulation am jeweils nächsten diskreten Zeitpunkt, der mit einer sogenannten Zeitschrittsteuerung gefunden wird, fortgesetzen.

Digitale Simulation: Wertdiskretisierung: Logikmodelle



Es liegt zunächst nahe, digitale Schaltungen mit zwei Werten (0 und 1) zu behandelen. Dies reicht jedoch für viele Anwendungen nicht aus, da beispielsweise auch hochohmige Zustände von Tri-State-Schaltungen behandelt werden müssen, oder Unbestimmtheit bzw. Unsicherheit modelliert werden soll.

Um elektrisch hochohmige Signale darstellen zu können, ergänzt man die Menge der Signalwerte um den Wert Z. Dieser Wert bedeutet ,dass das beterffende Signal nicht durch irgendwelche Schaltungsteile zur Annahme eines logischen Signalwechsel gezwungen wird.

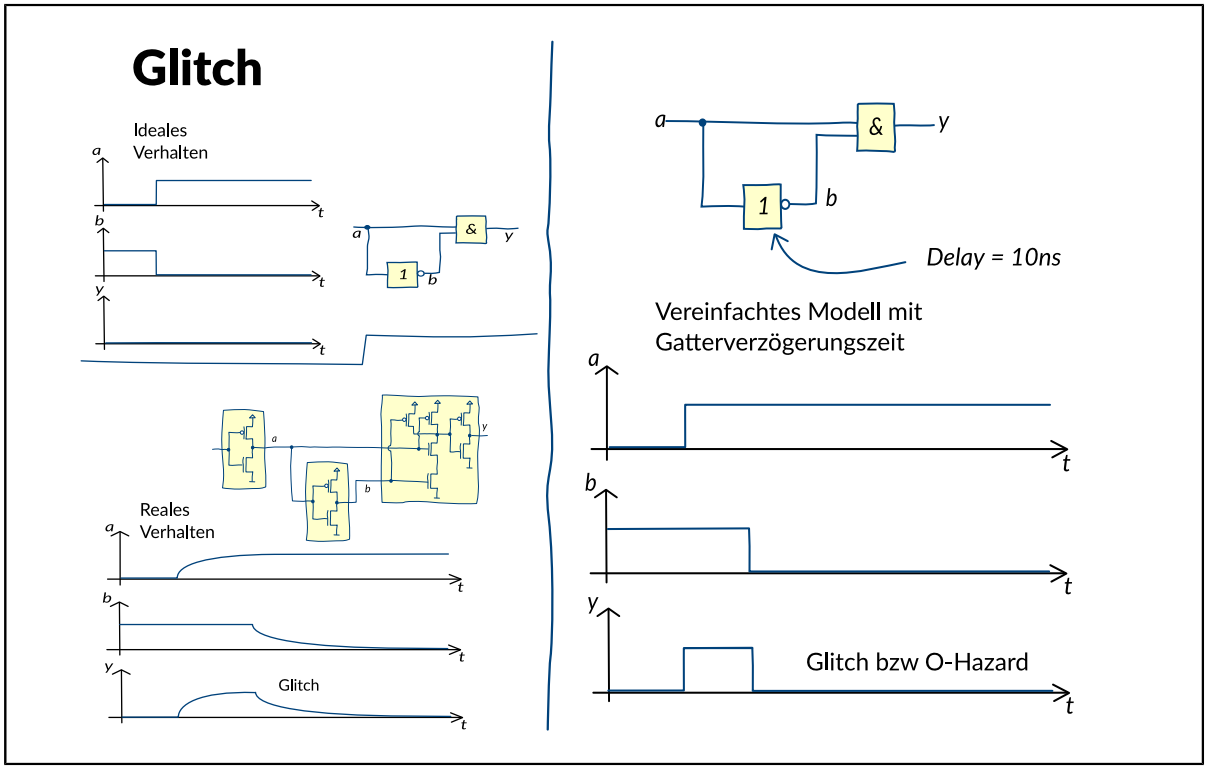
In den meisten Fällen wird die Menge {0,1,Z} um zwei weitere Werte erweitert:

- Ein Wert X mit der Interpretation "entweder 0 oder 1" d.h. es liegt Unbestimmtheit vor.
- Ein Wert U mit der Interpretation "elektrisch weder 0 noch 1", d.h. es liegt Unsicherheit in Form eines Zwischenwertes vor.

Auf diese Weise ist ein fünfwertiges Logikmodell mit der Wertemenge {0,1,Z,X,U} entstanden, für das die wntsprechendenVerknüpfungstabellen zu erstellen sind. Als vereinfachtes Beispiel zeigt das Bild dazu die Wahrheitstabelle fürein UND-Gatter in einer dreiwertigen Logik mit {0,1,X}.

Heutige Logiksimulatoren haben noch wesentlich größere Wertmengen. Damit können beispielsweise unterschiedliche Signalstärken (und damit in diskreter Form der Innenwiderstand des Signaltreibers) modelliert werden. Von Interesse ist auch die Erkennung steigender und fallender Flanken, kurzeitiger Pegeleinbrüche u.ä. über die DDarstellung von Signalwerten.

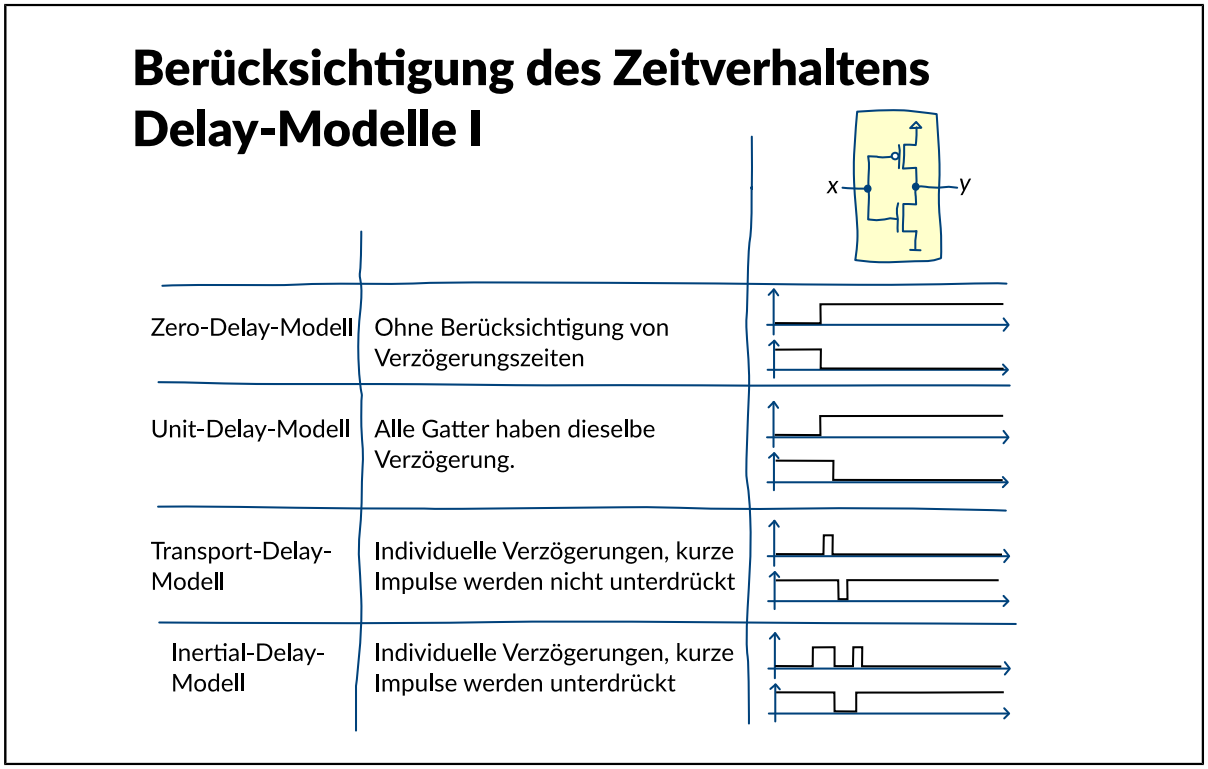
Digitale Simulation: Glitch



Ein Glitch oder ein so gennanter Hazard ist ein aufgrund unterschiedlicher Signallaufzeiten in den Pfaden einer Schaltung bedingte kurz Änderung des logischen Pegels. Ein 0-(1-)Hazard ist ein 1-(0-)Impuls auf einem Signal, welches aufgrund der logischen Funktion ständig 0(1) sein sollte.

Wir können Signale mit Mengen von Signalwertsequenztripeln der Form (a1,a2,a3) beschreiben. Die Signale a1 und a3 sollen stabile Signalpegel sein, d.h. aus {0,1}. Konstante Signale sind 0= {(0,0,0)} und 1={(1,1,1)}. Steigende (r) und fallende (f) Flanken können wir durch r={(0,0,1)(0,1,1)} bzw. f={(1,1,0)(1,0,0)} darstellen. Ein 0 Hazard liegt bei h={(0,1,0)} vor. Ein 1-Hazard ist entsprechend durch H={(1,0,1)} gegeben.

Digitale Simulation: Delay-Modell I



Zero-Delay

Bei synchronen Schaltungen ist man manchmal nicht an der Berechnung des genauen Zeitverlaufs von Signalwerten interessiert. Es reicht häufig zu wissen, welcher Wert sich an den Ausgängen der Schaltnetze nach "hinreichend" langem Warten einstellt. Dies entspricht der Tatsache, dass man die Taktrate entsprechend niedrig wählt.

Man braucht in diesem Fall nur noch die Funktion eines Gatters zu betrachten und kann daher die interessierenden Werte wesentlich schneller berechnen als in solchen Fällen, in denen auch das zeitliche Verhalten bestimmt werden muss. Die Verwendung eines Zero-Delay-Modells in einem Simulator bedeutet also, dass man das Verzögerungszeitverhalten ignoriert und eine rein funktionelle Simulation vornimmt.

Unit-Delay

Eine erste Berücksichtigung des Zeitverhaltens bietet die so genannte Unit-Delay-Simulation. Hierbei wird angenommen, dass alle Gatter ihre Ergebnisse gleich schnell berechnen. Man hat damit immer noch keine Information über den genauen zeitlichen Ablauf. Delays von komplexeren Gattern sind normalerweise größer als die von einfacheren Gattern. Diese Unterschiede können im Unit-Delay-Modell nicht berücksichtigt werden. Dadurch kann die Simulation einige der Pegeleinbrüche, die durch Delays erzeugt werden, nicht erkennen. Andere Pegeleinbrüche, die auftreten, wenn alle Gatter in etwa die gleiche Komplexität haben, können aber erkannt werden.

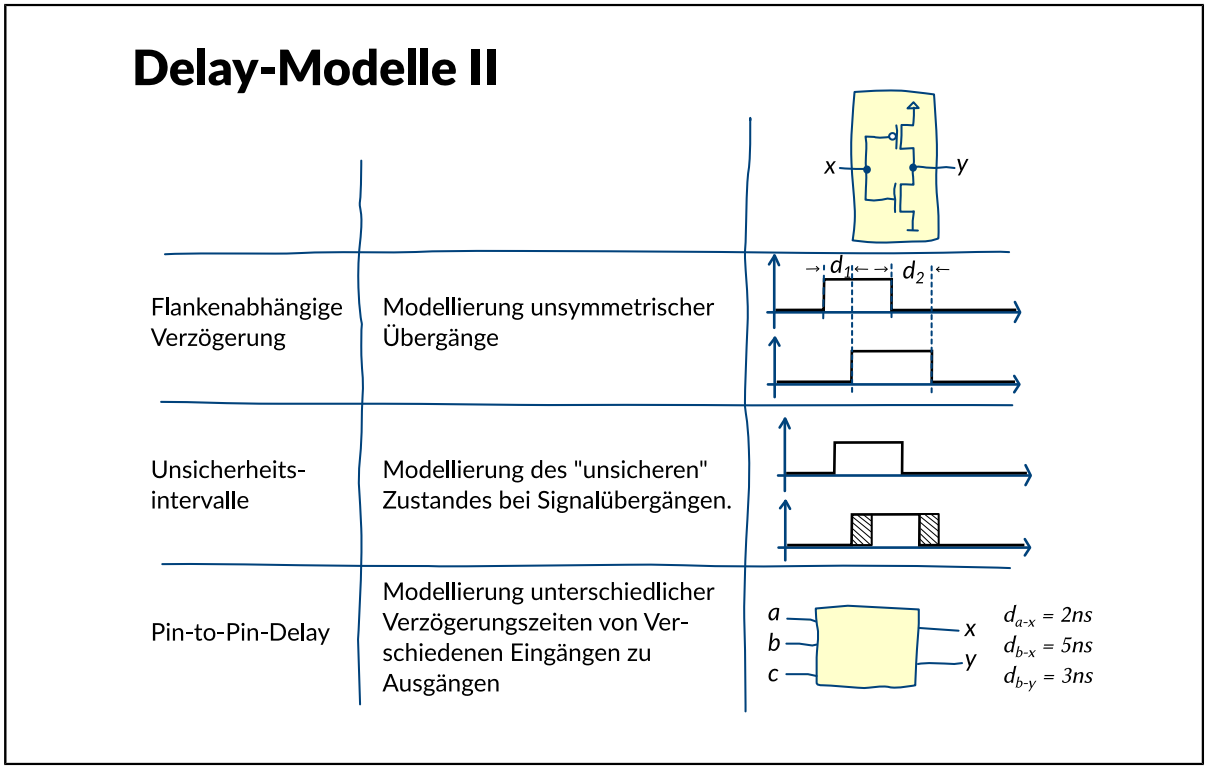
Transport-Delay

Die Transportverzögerung ermöglicht die Zuweisung individueller Verzögerungszeiten an die Gatter. Ändert sich also ein Eingangswert zum Zeitpunkt t, so ist die Wirkung am Ausgang zum Zeitpunkt t +d zu beobachten. Die Größe d wird Transport-Delay genannt. Dabei wird angenommen, dass auch beliebig kurze Impulse unverändert (lediglich verzögert) vom Eingang zum Ausgang übertragen werden.

Inertial-Delay

Aufgrund der vorhandenen Kapazitäten reagieren praktische Schaltungen nicht auf sehr kurze Eingangsimpulse, sie haben eine gewisse Trägheit (Inertia). Simulatoren bieten daher in der Regel die Möglichkeit, neben dem Transport-Delay eine untere Zeitschranke für relevante Eingangsimpulse anzugeben.

Digitale Simulation: Delay-Modell-II

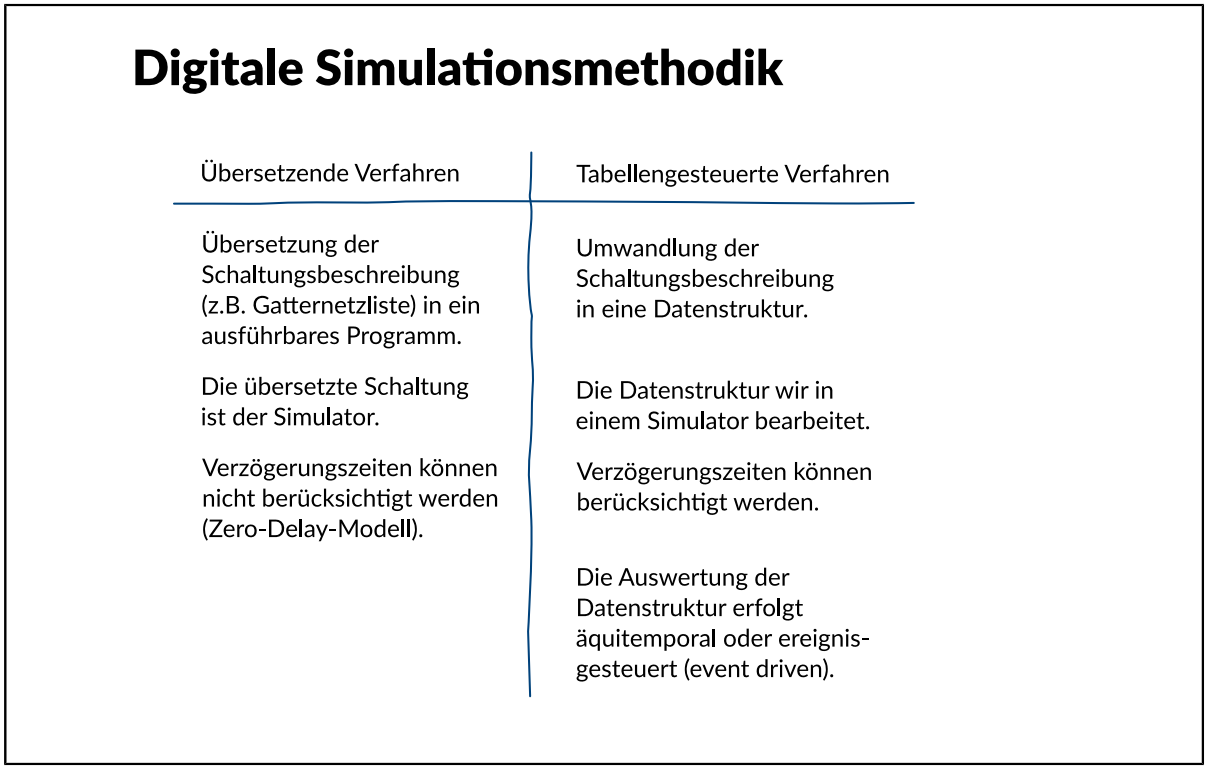


In einer weiteren Verfeinerung kann berücksichtigt werden, dass Signalwechsel auf ansteigende und abfallende Flanken häufig unterschiedlich viel Zeit benötigen.

Außerdem kann berücksichtigt werden, dass die genauen Verzögerungszeiten meist unbekannt sind oder beispielsweise Fertigungsschwankungen unterliegen und lediglich Bereiche (min,max) angegeben werden können.

Schließlich können die Verzögerungszeiten auf den Pfaden zwischen verschiedenen Ein- (und ggf. Aus-)gängen eines Gatters verschiedene Werte aufweisen. Dies wird durch so genannte Pin-to-Pin-Delays berücksichtigt.

Digitale Simulation: Digitale Simulationsmethodik

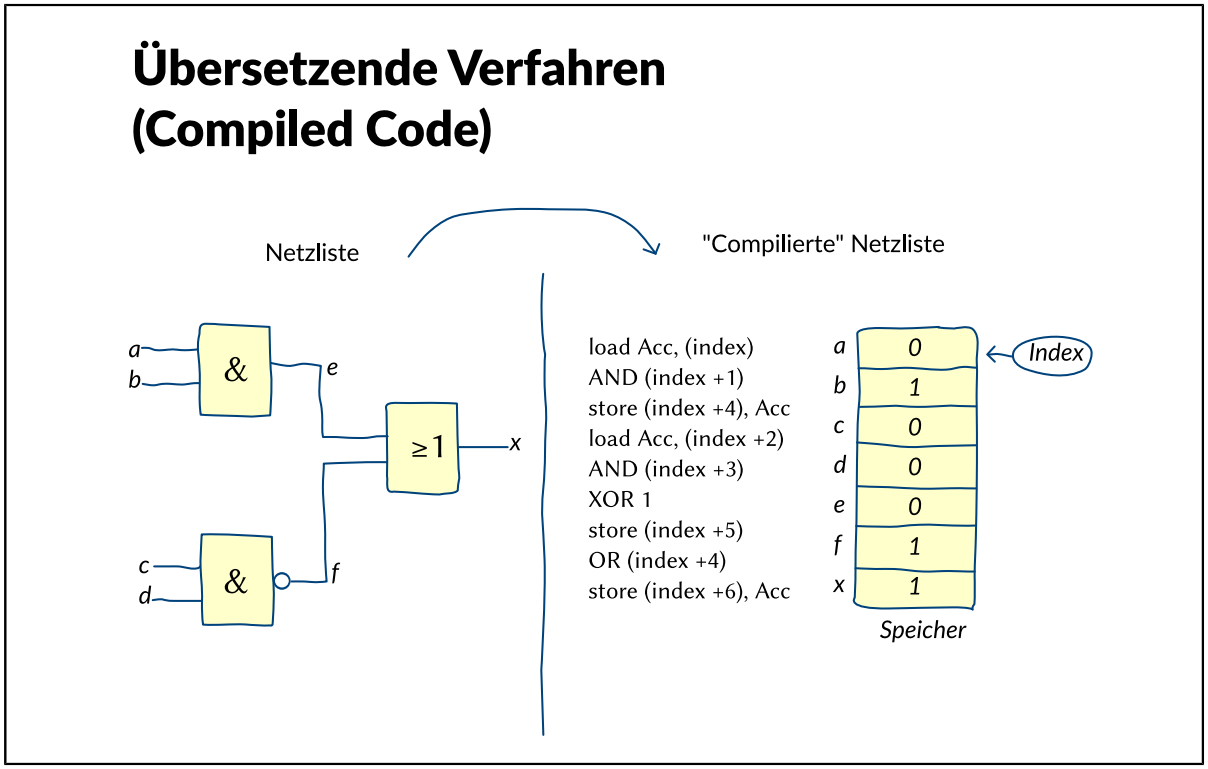


Die Aufgabe eines Simulationsalgorithmus besteht darin, ein Modellierungskonzept (Signalwerte, Zeit, logisches Verhalten) auf die Architektur eines Rechners zu übertragen. Die beste Lösung wird dabei erreicht, wenn die Architektur des Rechners genau mit dem Modellierungskonzept übereinstimmt (Algorithmus in Hardware) oder diesem zumindest ähnlich ist (Hardware-Beschleuniger). Nach diesem Grundsatz sind eine Reihe von speziellen Simulationsrechnern gebaut worden. Hardware-Beschleuniger, die durch besondere Rechnerarchitekturen, wie z.B. Pipelining, sequentielle Algorithmen um Faktoren von etwa 20-50 beschleunigen, haben auch kommerzielle Bedeutung erreicht.

Im Normalfall jedoch sind konventionelle von Neumann-Architekturen als Rechner zu benutzen. Hauptproblem dabei ist die Abbildung der im zu simulierenden Entwurf vorhandenen Parallelität auf die streng sequentielle Arbeitsweise des Rechners.

Wir unterscheiden dabei zwei grundsätzlich verschiedene Ansätze: übersetzende und tabellengesteuerte Verfahren. Bei den übersetzenden Verfahren wird die zu simulierende Schaltung durch ein (Maschinen- oder Hochsprachen-)Programm dargestellt. Hierfür sind naturgemäß Darstellungen in einer Hardware-Beschreibungssprache(HDL) besonders geeignet. Abgesehen von verschiedenen Steuerungsmechanismen stellt die Schaltungsbeschreibung selbst den Simulator dar. Dagegen wird bei tabellengesteuerten Verfahren die Schaltung in eine Datenstruktur umgesetzt. Diese Datenstruktur beschreibt alle logischen, topologischen und zeitlichen Eigenschaften der Schaltung und wird von einem schaltungsunabhängigen Programm, dem Simulator, bearbeitet.

Digitale Simulation: Übersetzende Verfahren



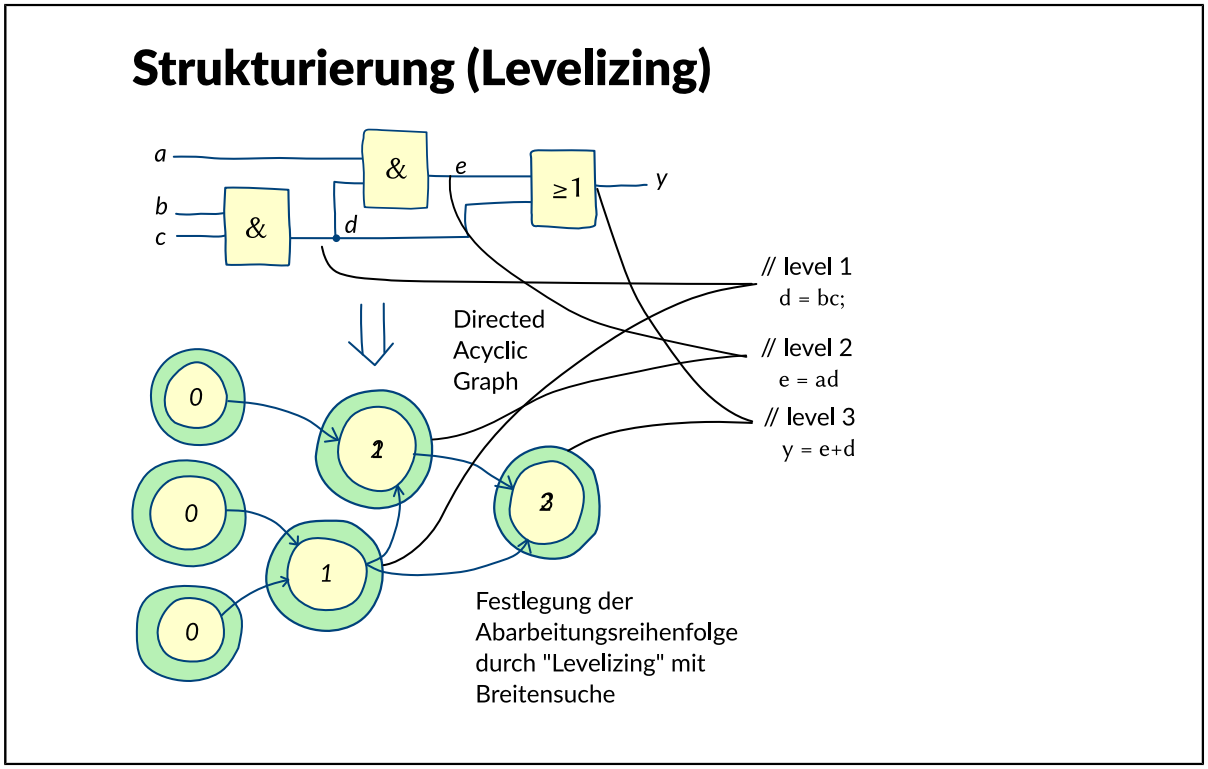
Zwar kann grundsätzlich jedes Modellierungskonzept, d.h. eine Schaltung auf beliebigen Ebenen des Y-Diagramms, in ausführbaren Programmcode umgesetzt werden, i.a. jedoch unterliegen übersetzende Verfahren (Compiled-Code-Simulation) insbesondere folgenden Einschränkungen:

- Es können nur kombinatorische oder strikt synchrone Schaltungen behandelt werden.
- Das genaue Zeitverhalten von Strukturelementen kann nicht direkt modelliert werden.

Ein Compiled-Code-Simulator übersetzt die Schaltungsbeschreibung in eine Folge von Maschinenbefehlen (oder Programmschnitten), die sowohl die Funktion als auch die Verbindungen der Schaltungselemente beschreibt. Für jedes dieser Elemente gibt es also eine oder mehrere Instruktionen sowie einen zugehörigen Eintrag im Speicher, der den momentanen Zustand des Elementes beschreibt.

Ein Beispiel für eine Compiled-Code-Simulation auf Gatterebene zeigt das Bild. Die Gatterfunktionen werden dabei direkt in Assembler-Befehle umgesetzt. Die zugrundeliegende Ein-Adress-Maschine führt die Operationen in einem Akkumulator aus. Eine direkte XOR-Operation mit "1" realisiert die Negation.

Digitale Simulation: Strukturierung



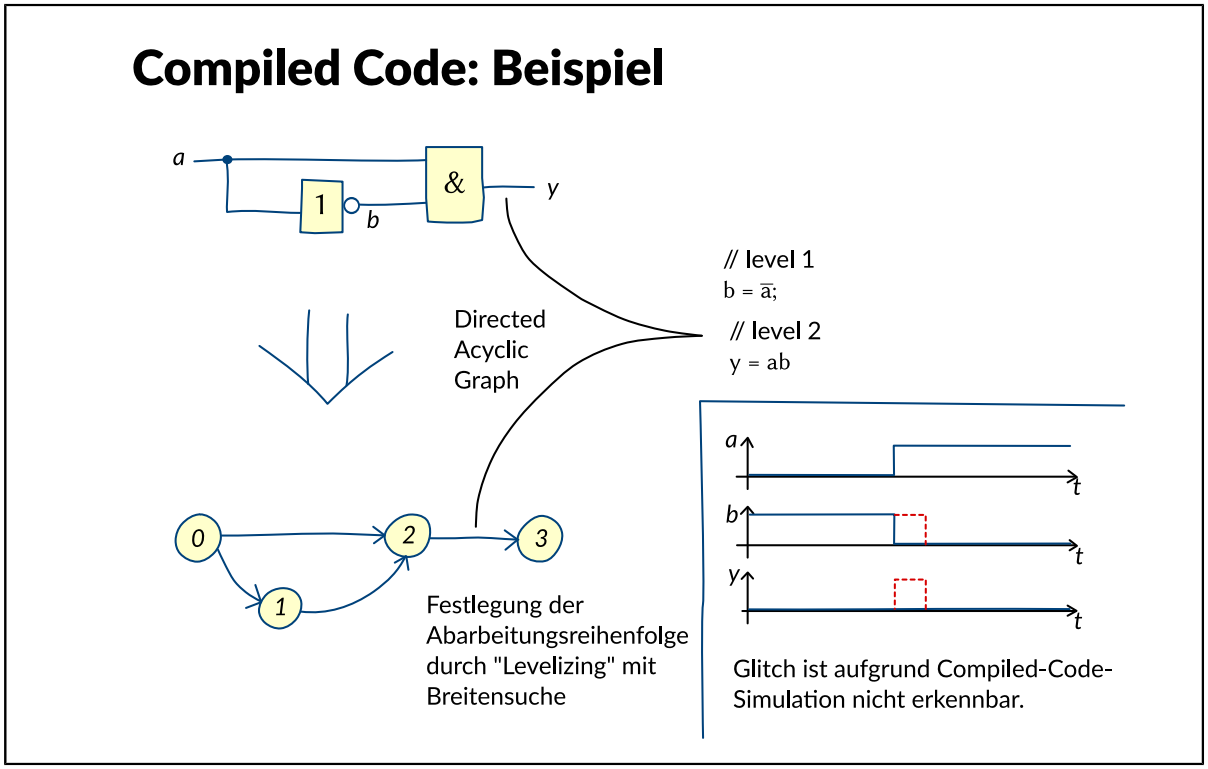
Um sicher zu stellen, dass kein logischer Wert eines Gatters verwendet wird, bevor er endgültig berechnet wurde, erfordert die Compiled-Code-Simulation eine Strukturierung der Schaltung in logische Ebenen (Levelizing). Auf diese Weise wird die (real parallel arbeitende) Schaltung in eine Sequenz von Rechnerschritten abgebildet.

Hierzu beschreibt man die Schaltung als gerichteten azyklischen Graphen (DAG), dessen Knoten die Gatter der Schaltung sowie die Ein- und Ausgänge repräsentieren. Jedem Knoten (K) wird eine nicht negative ganze Zahl level(K) zugewiesen, wobei gilt

- Für alle primären Eingänge Xi:level(Xi)=0
- Für alle anderen Knoten Ki:level(Ki)=1+max(level(Kj)), wobei Kj die Anfangsknoten aller bei Ki eingehenden Kanten bezeichnet.

Die sich nach dem Levelizing ergebende Sequenz, die in einer Programmiersprache beschrieben werden kann, lässt sich in einfacher Weise durch den Maschinencode eines beliebigen Prozessors ersetzen. Auf diese Weise erhält man einen schnellen Simulator, der jedoch auch rechnerabhängig ist. Im Falle der Hochsprache ist eine Übersetzung erforderlich. Dabei erzeugt ein Simulationsgenerator aus der Schaltungsbeschreibung den Simulator in einer Hochsprache. Ein gewöhnlicher Compiler übersetzt ihn in Maschinensprache. Erzeugt der Simulationsgenerator unmittelbar aus der Schaltungsbeschreibung Maschinencode, spricht man von einem "Native-Compiled"-Simulator.

Digitale Simulation: Compiled Code: Beispiel



Im ersten Beispiel wurde lediglich die Simulation eines Eingangsmusters beschrieben. Es ist jedoch leicht auf eine beliebige Folge von Eingangsmustern zu erweitern. Fügt man zusätzlich simulierte Register ein, können auch sequentielle Schaltungen simuliert werden.

Es soll nun gezeigt werden, dass ein solcher Simulator Timing-Probleme nicht erkennen kann. Man betrachtet dazu die im Bild dargestellte Schaltung. Aufgrund der logischen Gleichungen sollte ständig y=1 sein. Offenbar tritt aufgrund der Gatterverzögerung aber in der realen Schaltung ein statischer 0-Hazard nach Umschalten von x=0 auf x=1 auf, dessen Dauer der Verzögerungszeit des Inverters d entspricht.

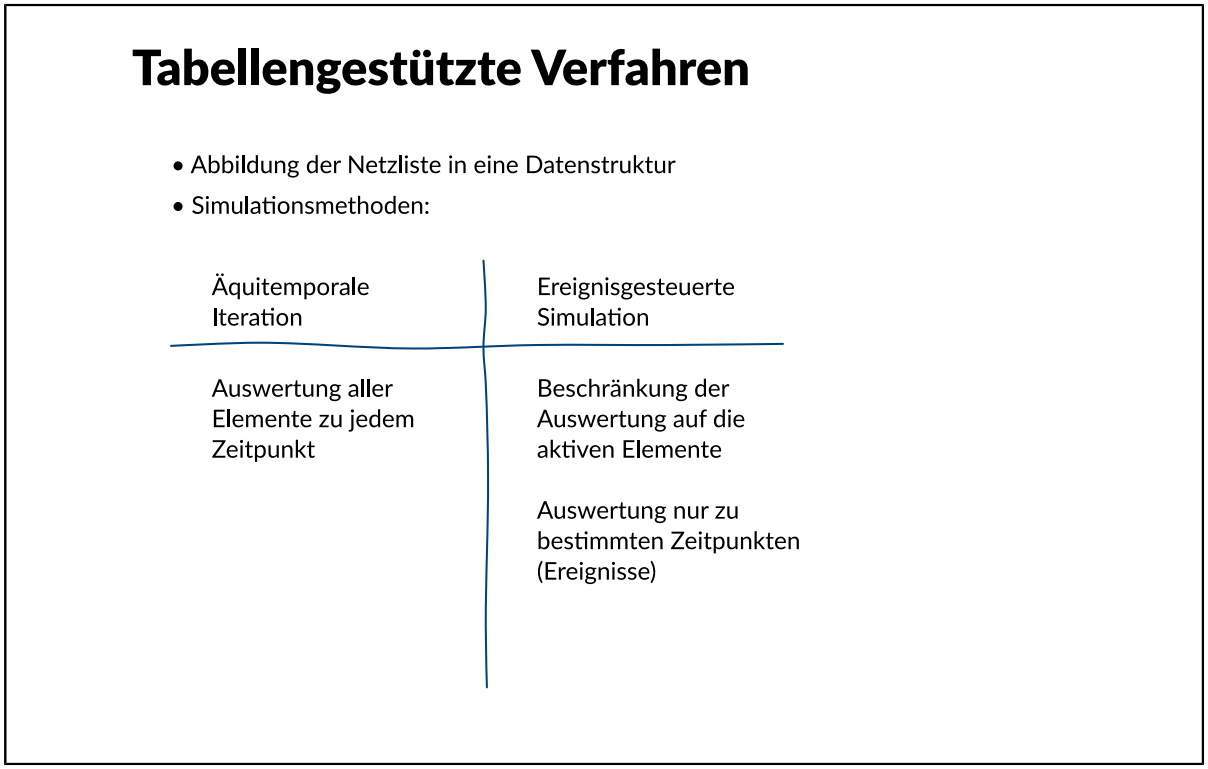
Simuliert man diese Schaltung mit dem obigen Verfahren, so wird als Reaktion auf einen neuen Eingabewert x zunächst a neu berechnet, bevor y aktualisiert wird. Damit wird der Pegeleinbruch vom Simulator aber nicht entdeckt. Der beschriebene Simulator simuliert lediglich das Verhalten einer verzögerungsfreien Schaltung. Er wird daher auch Zero-Delay-Simulator genannt. Für synchrone Schaltungen erzielt man mit einem solchen Simulator eine große Simulationsgeschwindigkeit. Für asynchrone Schaltungen kann man ihn nicht verwenden.

Compiled-Code-Simulation ist sehr effizient. Zwar wird ein zusätzlicher Aufwand für die Übersetzung benötigt, jedoch entsteht kein interpretativer Aufwand bei der eigentlichen Simulation. Compiled Code besitzt deshalb insbesondere für höhere Entwurfsebenen (z.B. RT-Level) große Bedeutung, da dort ein genaues Zeitmodell nicht erforderlich ist.

Die Simulationszeit von Compiled-Code-Simulatoren ist proportional zur Anzahl der Elemente. Sie kann durch verschiedene Maßnahmen verringert werden. Eine Methode besteht z.B. darin, unnötige Berechnungen zu vermeiden. So muss ein AND-Gatter nicht ausgewertet werden, wenn einer seiner Eingänge auf logisch 0 liegt. Die Länge des Codes steigt damit jedoch beträchtlich an.

Will man mehrwertige Logik benutzen oder stellen die Strukturelemente komplexere Funktionen dar, benutzt man i.a. Funktionstabellen oder Prozeduren für die einzelnen Elemente.

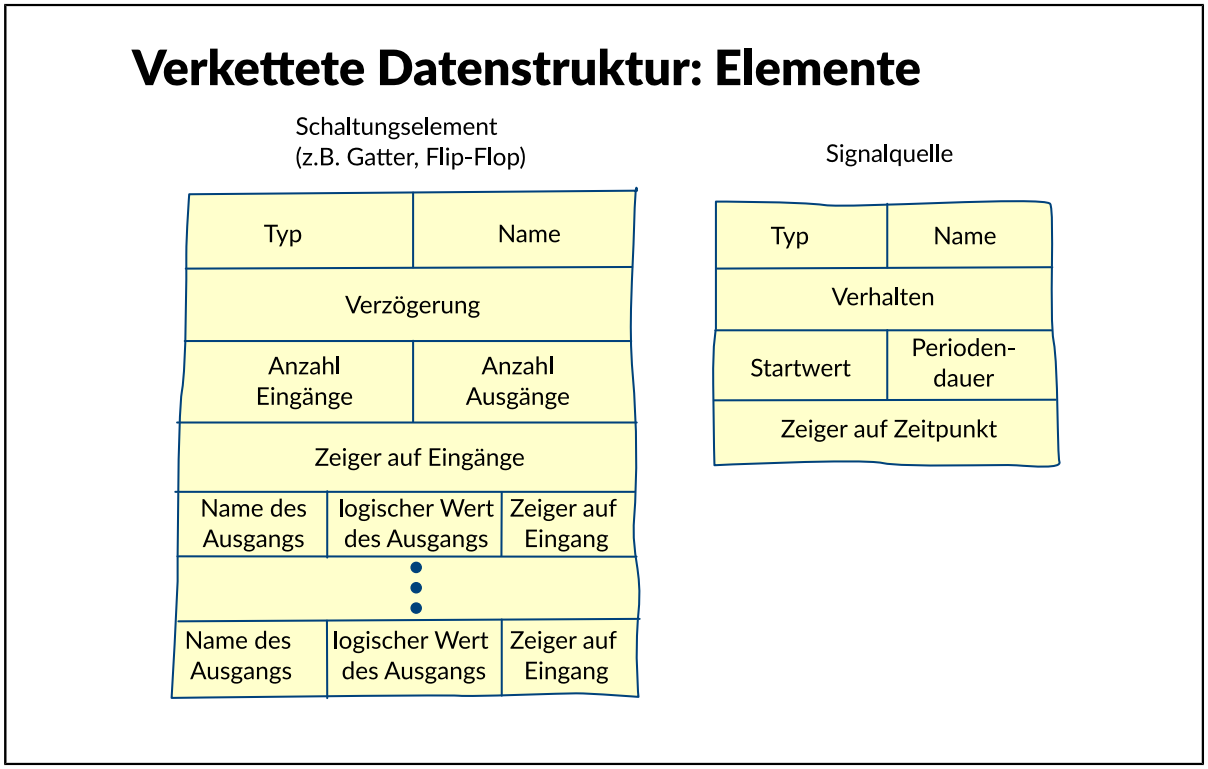
Digitale Simulation: Talbellengestützte Verfahren



Tabellengesteuerte Verfahren bilden die Schaltung, d.h. ihre Komponenten und Verbindungen in eine Datenstruktur ab. Dies können im Fall eines Unit-Delay-Simulators einfache Tabellen sein oder - für komplexe Delay-Modelle - spezielle Datenstrukturen, die vielfach mit Hilfe verketteter linearer Listen aufgebaut werden.

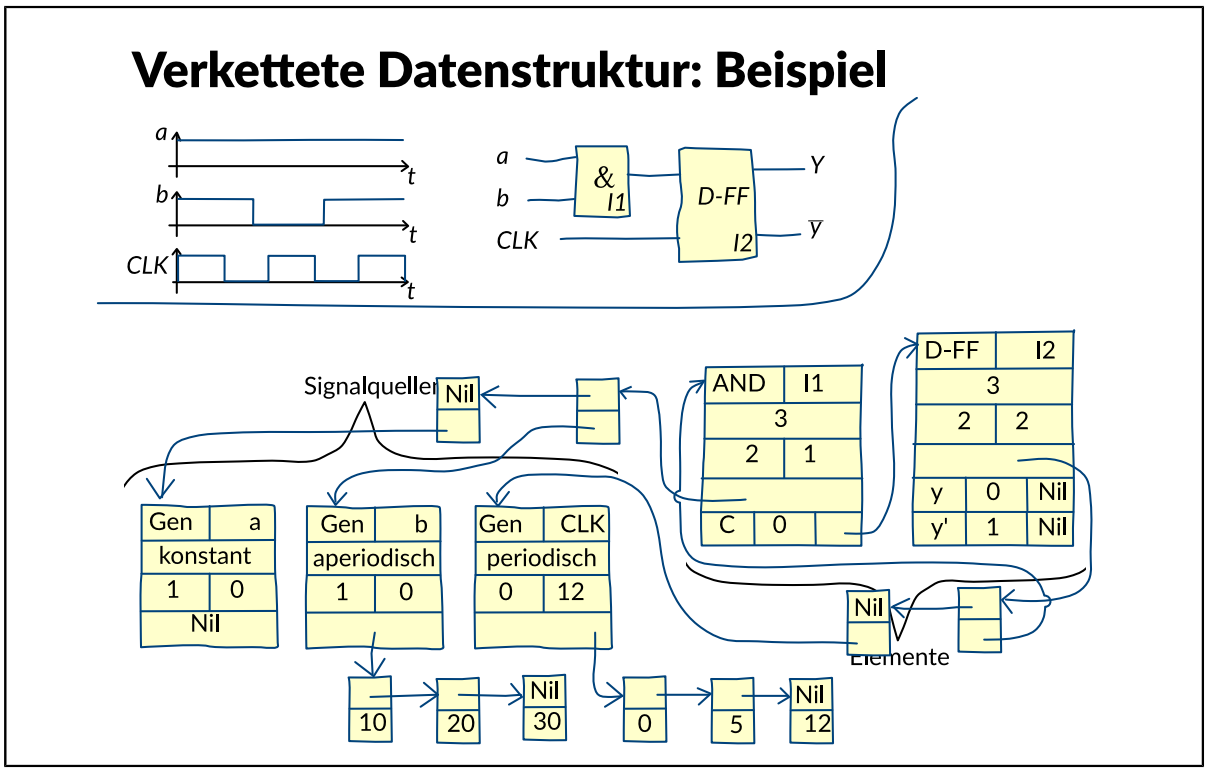
Bei der Bearbeitung dieser Datenstrukturen durch den Simulator wird zwischen der äquitemporalen Iteration und der so genannten ereignisgesteuerten Simulation unterschieden. Im ersten Fall macht der Simulator einen Zeitschritt und untersucht dann die logischen Zustände aller Schaltungselemente. Im zweiten Fall wird durch eine so genannte Ereignissteuerung dafür gesorgt, dass lediglich solche Elemente untersucht werden, an denen sich ein Eingangssignal verändert hat. Ereignissteuerung ist heute die Standardmethode bei der Simulation.

Digitale Simulation: Verkettete Datenstrukturen: Elemente



Elemente und Signale können durch geeignet aufgebaute Datensätze, wie hier beispielhaft gezeigt, beschrieben werden. Sie werden durch entsprechende Zeiger miteinander verkettet, so dass vollständige Schaltungen mitsamt der Stimuli beschrieben werden können.

Digitale Simulation: Verkettete Datenstrukturen: Beispiel

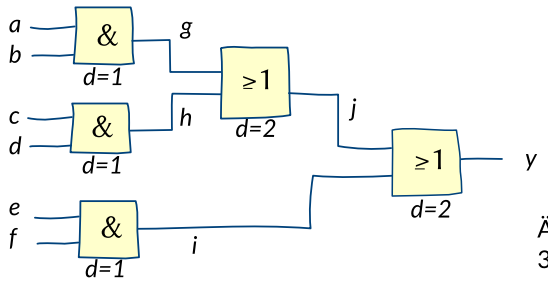


Das Bild zeigt beispielhaft eine logische Schaltung, die aus einem D-Flip-Flop und einem AND-Gatter aufgebaut ist und mit einer aperiodischen Folge von Nullen und Einsen beaufschlagt wird. Das Flip-Flop wird mit einem unsymmetrischen Takt betrieben, der eine Taktperiode von 12 Zeiteinheiten aufweist. Um eine effiziente Bearbeitung zu ermöglichen, ist die Beschreibung mehrfach verkettet aufgebaut und enthält Redundanz. Alle Eingänge eines Elements sind jeweils in einer verketteten Liste zusammengefaßt. Die Ausgänge dagegen sind als Tabelle innerhalb der Elementbeschreibung aufgeführt.

Mit Hilfe eines geeigneten Verfahrens - das den eigentlichen Simulator darstellt - ist diese Datenstruktur entsprechend dem zeitlichen Ablauf zu bearbeiten. Dazu gibt es zwei unterschiedliche Vorgehensweisen.

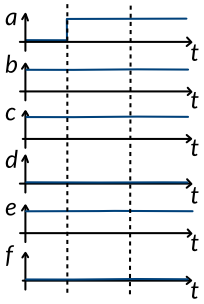
Digitale Simulation: Äquitemporale Iteration

Äquitemporale Iteration: Beispiel



Ereignissteuerung:
3 Gatterevaluierungen

Äquitemporale Iteration:
30 Gatterevaluierungen



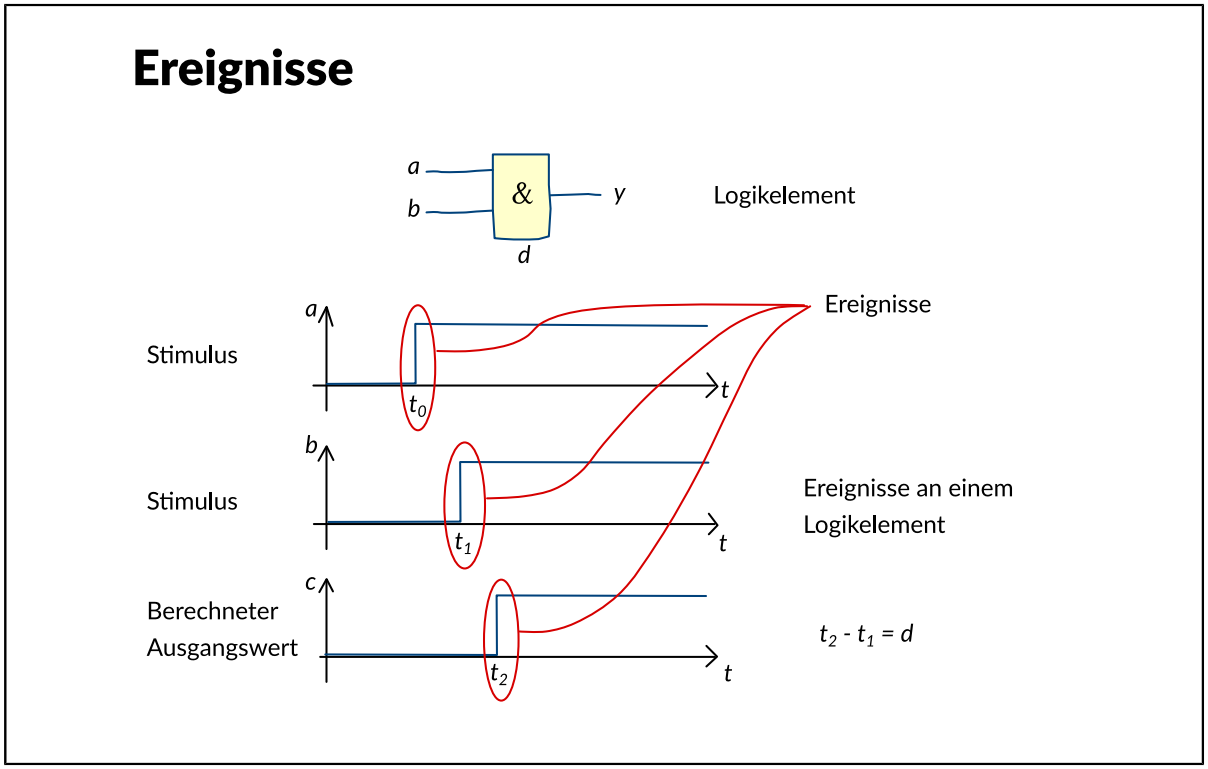
t	a	b	c	d	e	f	g	h	i	j	y
0	0	1	1	0	1	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0	0	0
2	1	1	1	0	1	0	1	0	0	0	0
3	1	1	1	0	1	0	1	0	0	0	0
4	1	1	1	0	1	0	1	0	0	1	0
5	1	1	1	0	1	0	1	0	0	1	0
6	1	1	1	0	1	0	1	0	0	1	1

Bei der äquitemporalen Iteration wird für jeden Zeitpunkt die gesamte Schaltung untersucht, d.h. jedes Element wird ausgewertet. Anschließend wird die Zeit inkrementiert. Die Schrittweite kann sich ggf. von Iteration zu Iteration ändern. Sie ist jedoch stets für alle Schaltungselemente dieselbe.

Ein solches Verfahren ist leicht zu implementieren. Es ist jedoch sehr wenig effizient, da in einem Schritt normalerweise sehr viele Elemente ihren Zustand gar nicht ändern, sie also eigentlich gar nicht untersucht werden müssten. Auf Gatterebene liegt die Wahrscheinlichkeit für eine Änderung lediglich bei bis zu 10%. Die Effizienz steigt, wenn die Schaltungsaktivität zunimmt oder die Signalauflösung erhöht wird. Äquitemporale Iteration wird deshalb im Regelfall nur auf der RT-Ebene (grobes Zeitraster) oder auf der elektrischen Ebene (kontinuierliche Signalwerte) angewendet.

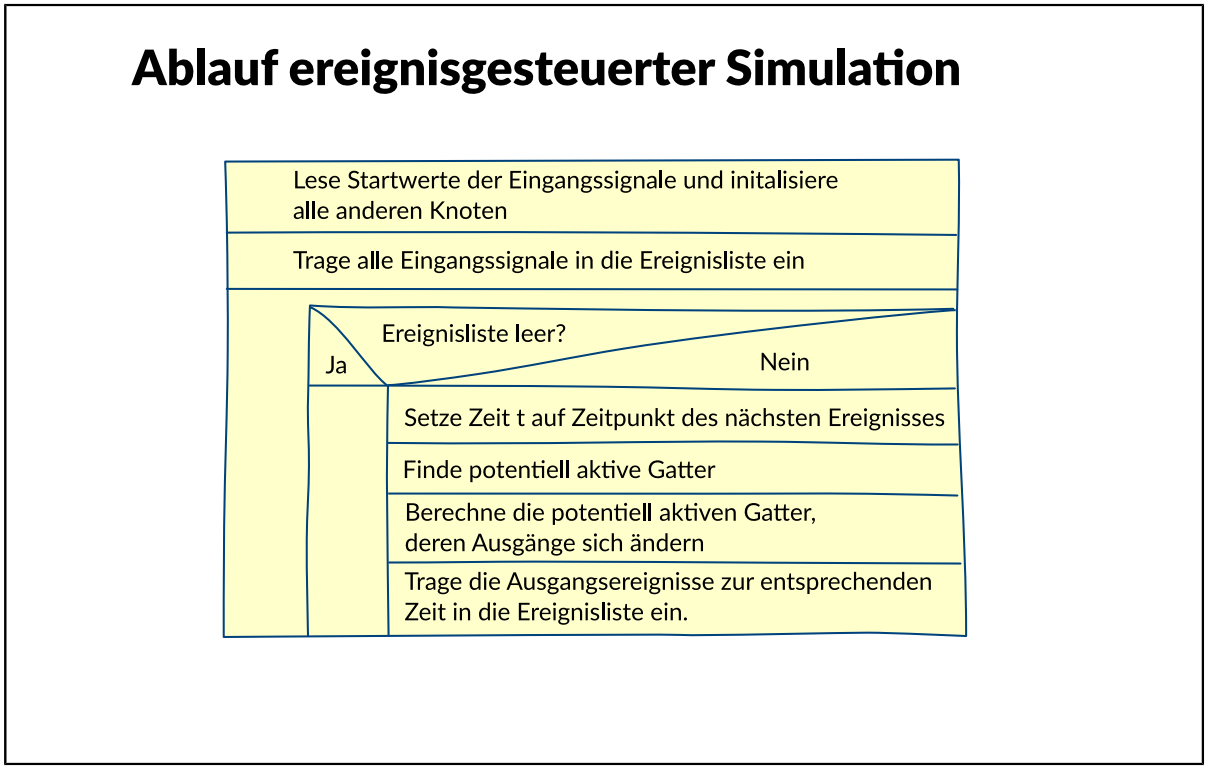
Das Beispiel zeigt eine Gatterschaltung, bei der sich zum Zeitpunkt t=1 ein Eingangssignal ändert. Bei den angenommenen Gatterverzögerungen sind dann 30 Gatterevaluierungen (5 Gatter in 6 Zeitschritten) erforderlich, um den Wert des Ausgangssignals zu bestimmen. Gelingt es, nur solche Gatter auszuwerten, bei denen sich das Eingangssignal tatsächlich geändert hat, kommt man mit 3 Gatterevaluierungen aus. Dies führt zum Konzept der Ereignissteuerung.

Digitale Simulation: Ereignisse



Ereignisgesteuerte Verfahren lösen das Effizienzproblem der äquitemporalen Iteration dadurch, dass sie die Berechnungen auf die notwendige Anzahl, d.h. die Berechnung der Gatter, bei denen sich mindenstens ein Eingangssignal ändert (Nichtidentität der Eingangssignale), beschränken. Immer, wenn sich Eingangssignale (Stimuli) oder der Ausgang eines Elements ändern, wird ein so genanntes Ereignis (Event) erzeugt. Nur Elemente, die mit diesem Netz verbunden sind, müssen bezüglich dieses Ereignisses untersucht werden. Diese Gatter heißen potentiell aktive Elemente. Auf diese Weise können - anders als beim Compiled-Code-Ansatz - kombinatorische, synchrone und asynchrone Schaltungen in gleicher Weise behandelt werden. Auch sind die verschiedensten Delay-Modelle implementierbar. Lediglich das Inertial-Delay-Modell führt zu Schwierigkeiten, da es erforderlich sein kann, ein aktiviertes Ereignis wieder zu deaktivieren.

Digitale Simulation: Ablauf ereignisgesteuerter Simulation



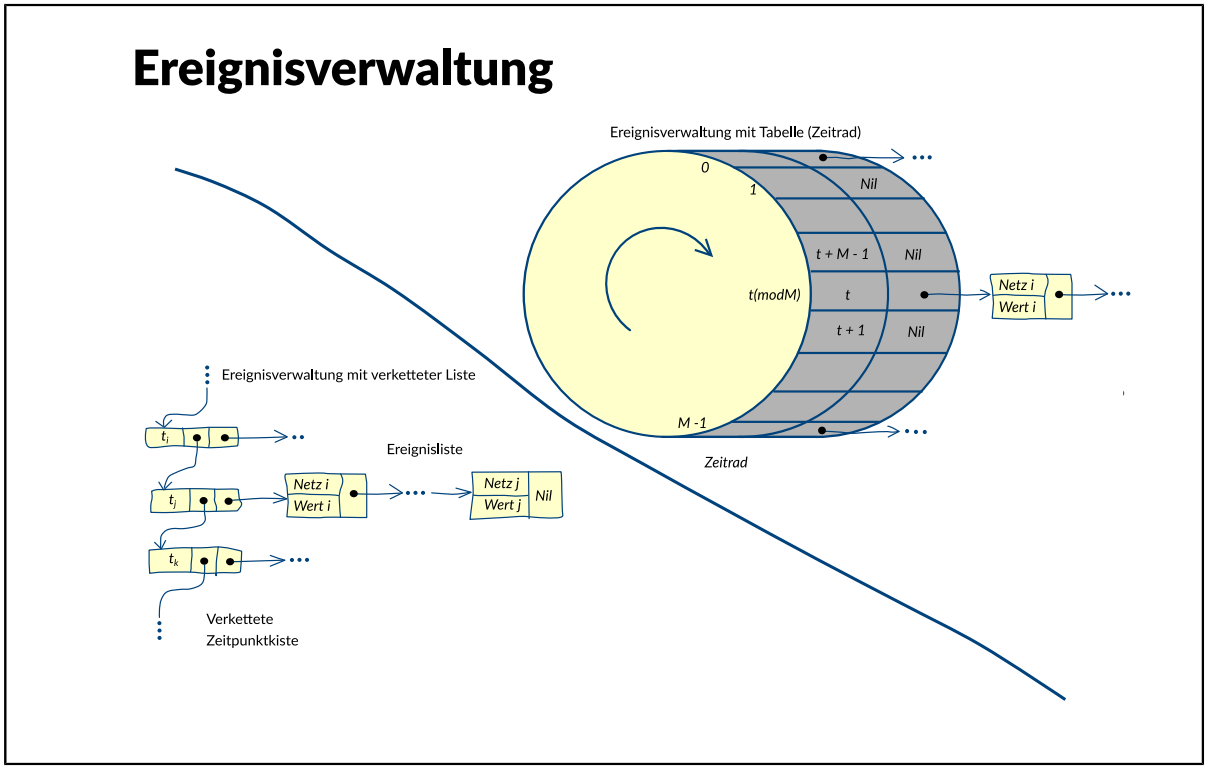
Mit den geschilderten Datenstrukturen ergibt sich der allgemeine Ablauf wie folgt: Zunächst werden alle Schaltungseingänge mit vorgegebenen Werten initialisiert. Alle anderen Netze werden auf X gesetzt. Das Zeitrad wird mit t0 initialisiert. Zu Beginn der Simulation sind alle mit den Schaltungseingängen verbundenen Elemente potentiell aktiv. Für jeden Zeitschritt des Zeitrads werden nun folgende Schritte ausgeführt:

Mit aktuellen Werten werden alle potentiell aktiven Elemente untersucht. Treten Änderungen an den Ausgängen auf, werden diese neuen Ereignisse in die Ereignisliste eingetragen. Die dort einzutragenden Zeitpunkte hängen von der aktuellen Zeit und den Verzögerungen der Elemente ab.

Diese Schleife wird bis zum Ende der Simulationszeit, das ist der Zeitpunkt, zu dem keine Ereignisse mehr vorhanden sind, durchlaufen.

Moderne Simulatoren nutzen die Geschwindigkeit der übersetzenden Verfahren und verbinden sie mit der Genauigkeit der Event-orientierten Verfahren. Deshalb werden zwar Event-Listen geführt, neue Zustandswerte aber durch übersetzten Code berechnet.

Digitale Simulation: Ereignisverwaltung



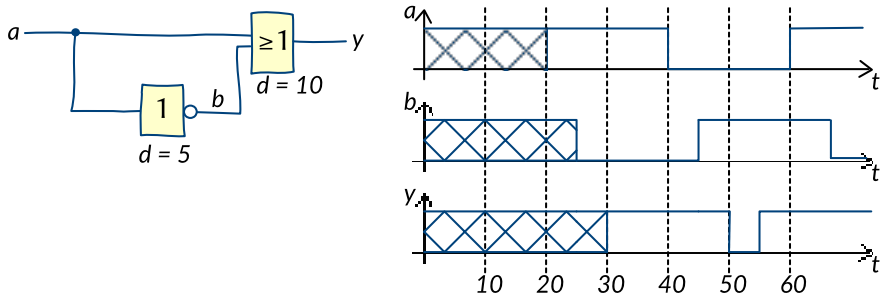
Mit der bereits beschriebenen Datenstruktur lassen sich in einfacher Weise die Eingangswerte für ein zu untersuchendes Element sowie die beim Auftreten eines Ereignisses potentiell aktiven Elemente finden. Eine weitere Datenstruktur wird zur Steuerung der Zeit und zur Verwaltung der Ereignisse benötigt. Wird ein Element K ausgewertet und festgestellt, dass sich sein Ausgangsnetz ändert, so muss die Änderung des Wertes dieses Netzes für einen Zeitpunkt t+d eingetragen werden, wobei t die aktuelle Simulationszeit und d die Verzögerung des betrachteten Elementes ist. Zur Verwaltung dieser Ereignisse lässt sich eine verkettete Liste verwenden.

In einer solchen Datenstruktur erfordert die Eintragung eines Ereignisses zum Zeitpunkt t das Durchsuchen der geordneten Zeitpunktliste, um die richtige Position für t zu finden. Die Suchzeit ist proportional zur Länge der Liste, die mit der Größe der Schaltung und ihrer Aktivität wächst. Häufig ist diese Liste sehr dicht belegt, d.h. dass die Differenzen zwischen zwei aufeinander folgenden Einträgen klein sind. Es liegt daher nahe, statt der Liste eine Tabelle (ein eindimensionales Feld) für die Zeitpunkte zu verwenden. Dann kann die Suche durch einen direkten Zugriff über den Index t ersetzt werden. Als Nachteil wirkt sich aus, dass Speicherplatz für Zeitpunkte ohne Ereignisse vergehalten werden muss. Der Speicherbedarf hierfür ist jedoch gering.

Dieses so genannte Zeitrad speichert Ereignisse zwischen der aktuellen Simulationszeit t und t+M-1, wobei M die Feldgröße darstellt. Alle Zeiten sind deshalb als t mod M zu deuten. Jedes Ereigniss für einen Zeitpunkt t+d mit t < M kann ohne Suche in das Zeitrad eingefügt werden. Ereignisse für Zeitpunkte, die über den im Zeitrad enthaltenen Bereich hinausgehen (t>(M-1)) werden in einer "Overflow"-Ereignisliste gespeichert. Einfügungen in diese Liste erfordern einen Suchvorgang, i.a. werden jedoch die meisten Ereignisse direkt im Zeitrad enthalten sein. Sobald der Zeitpunkt eines Ereignisses in den Bereich des Zeitrades fällt, sollte es deshalb dorthin übertragen werden.

Digitale Simulation: Ereignissteuerung: Beispiel

Ereignissteuerung: Beispiel



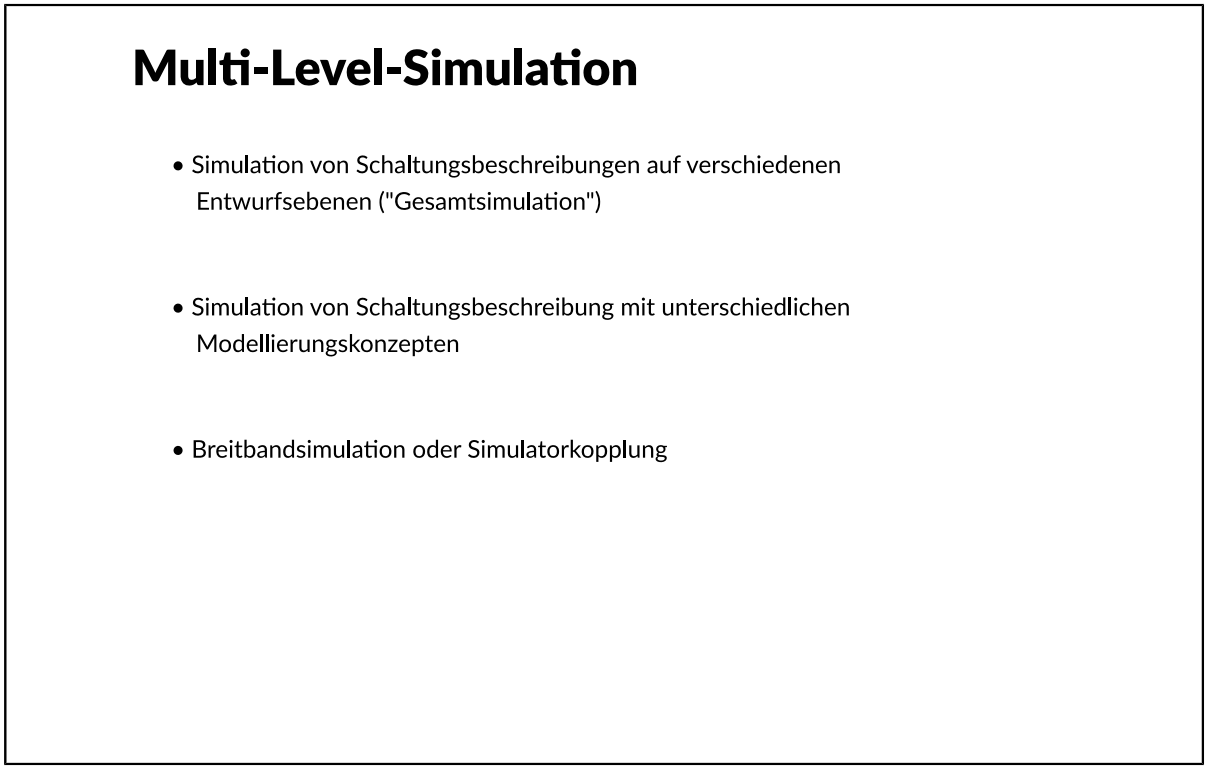
Zeit								
t	*0*	*20*	*25*	*30*	*40*	*45*	*50*	*55*
Signale								
a	*x*	*1*	*1*	*1*	*0*	*0*	*0*	*0*
b	*x*	*x*	*0*	*0*	*0*	*1*	*1*	*1*
y	*x*	*x*	*x*	*1*	*1*	*1*	*0*	*1*
Ereignisse								
1	{20, a=1}	{25, b=0}	{30, y=1}	{40, a=0}	{45, b=1}	{50, y=0}	{55, y=1}	.....
2	{40, a=0}	{30, y=1}	{40, a=0}	{60, a=1}	{50, y=0}	{55, y=1}	{60, a=1}	
3	{60, a=1}	{40, a=0} ...	{60, a=1}		{60, a=1}	{60, a=1}		

Das Bild zeigt den groben Ablauf einer ereignisgesteuerten Simulation für das bereits bei der Compiled-Code-Simulation betrachtete Schaltungsbeispiel. Im Zeitdiagramm ist ein Pegeleinbruch (Glitch) bei t=50 zu erkennen. Mit dem verwendeten Simulator können also - anders als beim einfachen Compiled-Code-Verfahren - mit Hilfe eines Transport-Delay-Zeitmodells Timing-Probleme solcher Art erkannt werden.

Ereignisgesteuerte Verfahren sind grundsätzlich für die meisten Verzögerungszeit- und Zustandsmodelle anwendbar. Für die Auswertung der Elemente sind Unterprogramm- oder Tabellentechniken (Wahrheitstabellen) gebräuchlich. Die Zeit für die Auswertung von Elementen bestimmt sehr wesentlich die Simulationsdauer, die Effizienz der Auswertung ist deshalb sehr wichtig.

Gegenüber der Compiled-Code-Simulation hat das ereignisgesteuerte Verfahren den Vorteil, dass in jedem Zeitschritt nur die potentiell aktiven Elemente ausgewertet werden. Der Nachteil liegt im entsprechenden Zeitbedarf für das Finden dieser Elemente und die Verwaltung der Ereignisse. Ereignisgesteuerte Simulatoren sind deshalb nur dann wesentlich schneller, wenn die Schaltungsaktivität gering ist.

Digitale Simulation: Multi-Level-Simulation



Wie bereits in der Einleitung zur Simulation besprochen, werden häufig Simulatoren benötigt, die in der Lage sind, mit Schaltungsbeschreibungen auf verschiedenen Entwurfsebenen umzugehen, insbesondere um eine Gesamtsimulation einer Schaltung durchführen zu können, wenn die Teile der Schaltung in unterschiedlicher Abstraktion bzw. mit unterschiedlichen Modellierungskonzepten beschrieben sind. Klassische Simulatoren dagegen erwarten eine Schaltungsbeschreibung auf genau einer Entwurfsebene.

Für eine Multi-Level-Simulation (wenn diese in einem Simulationslauf unterschiedlich beschriebene Teilschaltungen zulässt, spricht man auch von Mixed-Level-Simulation) können zwei unterschiedliche Ansätze verfolgt werden:

- Breitband-Simulator
- Simulatorkopplung

Für Schaltungen, die mit einer Hardwarebeschreibungssprache wie VHDL beschrieben sind, scheint ein Breitband-Simulator, d.h. ein Algorithmus, der alle Ebenen umspannt, der natürliche Ansatz zu sein. Solche Simulatoren werden auf der Basis ereignisgesteuerter Algorithmen realisiert. Da sehr unterschiedliche Modellierungskonzepte unterstützt werden müssen, sind Breitband-Simulatoren sehr komplex und deshalb für viele Anwendungen wenig effizient.