Lehrbuch Digitaltechnik

Eine Einführung mit VHDL

von
Prof. Dr. Jürgen Reichardt

2., überarbeitete Auflage

Oldenbourg Verlag München

Prof. Dr. Jürgen Reichardt hat seit 1995 die Professur für Informationstechnik an der Hochschule für Angewandte Wissenschaften Hamburg inne. Vorher war Herr Reichardt – nach Studium und Hochschultätigkeit in Hamburg und Stuttgart – von 1984 bis 1993 Gruppenleiter Chip-Entwicklung bei Philips Semiconductors Hamburg. Seine akademischen Arbeitsschwerpunkte sind: Digitaler Systementwurf, Digitale Signalprozessoren, HW-SW-Codesign, System-on-Chip(SoC)-Entwurf.

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
www.oldenbourg-verlag.de

Lektorat: Angelika Sperlich
Herstellung: Constanze Müller
Einbandgestaltung: hauser lacour
Gesamtherstellung: Grafik + Druck, München

Dieses Papier ist alterungsbeständig nach DIN/ISO 9706.

ISBN 978-3-486-70680-2

Vorwort zur 2. Auflage

Der große Erfolg, die zahlreiche positive Resonanz der Leser und das studentische Feedback aus meinen Lehrveranstaltungen haben das didaktische Konzept des Lehrbuchs Digitaltechnik bestätigt. Die Verknüpfung von Grundlagenwissen mit praktischen Beispielen zur VHDL Implementierung erscheinen mir für eine moderne Lehre des Faches Digitaltechnik an einer Hochschule weiterhin sehr gut geeignet.

In der nun vorliegenden zweiten Auflage sind neben der Beseitigung von Unklarheiten und Fehlern, die sich in vielen Gesprächen insbesondere mit Studierenden heraus kristallisiert hatten, alle Beispiele zur VHDL Arithmetik aktualisiert worden. Anstatt der proprietären Synopsys Bibliotheken wird nun die Bibliothek ieee.`numeric_std` mit den Datentypen `signed` bzw. `unsigned` verwendet. Damit wird der zunehmenden Verbreitung des vom IEEE definierten Arithmetikstandards Rechnung getragen.

Den vielen Studierenden und allen Kollegen, sei für ihr kritisch konstruktives Feedback und ihrer Toleranz bei der Aufklärung von Fehlern in der ersten Auflage herzlich gedankt.

Hamburg, März 2011 J. Reichardt

Inhaltsverzeichnis

1 Einleitung

Warum schon wieder ein neues Lehrbuch zur Digitaltechnik? Diese Frage wurde mir zu Beginn des Buchprojektes einige Male gestellt. Die daraus resultierenden Gespräche mit Kollegen haben deutlich zu Tage gebracht, dass vielfältige Gründe für ein neues Buchkonzept vorliegen, die ich im Folgenden skizzieren werde.

Obwohl die Grundlagen der Digitaltechnik im letzten Jahrzehnt weitgehend unverändert blieben, so haben sich die Vorlesungs- und Praktikumsinhalte in diesem Fach an den Hochschulen in den letzten Jahren doch erheblich verändert. Dies liegt an neuen Methoden, die sich beim Entwurf digitaler Schaltungen und Systeme weitgehend durchgesetzt haben. Eine digitale Schaltung wird nur noch im Ausnahmefall mit diskreten Logikbausteinen aufgebaut und deren Funktion auf einer Platine nachträglich überprüft. Die stark gestiegenen Anforderungen an die Komplexität digitaler Systeme, zusammen mit den Anforderungen eines geringen Platzbedarfs und niedriger Stromaufnahme haben dazu geführt, dass in umfassendem Maße reprogrammierbare Bausteine zur Implementierung digitaler Logikfunktionen eingesetzt werden.

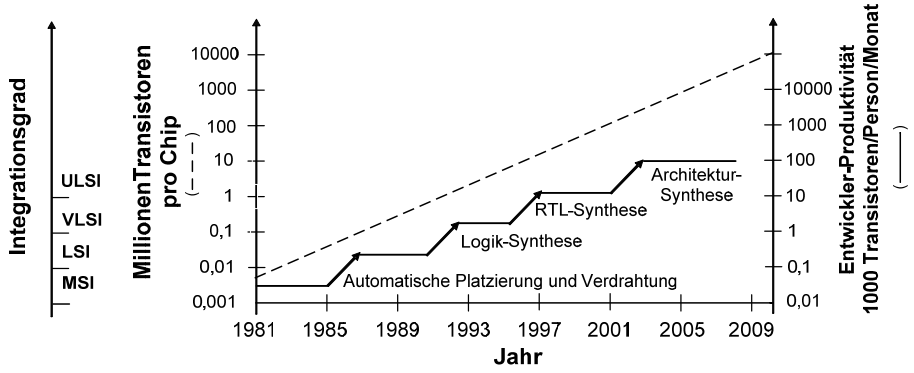


Bild 1.1: Entwicklung des Integrationsgrads digitaler Schaltungen (gestrichelte Linie) und Entwurfsproduktivität der Schaltungsentwickler (durchgezogene Linie). Wesentliche Veränderungen der Entwurfsmethoden sind dargestellt (nach [28])

Eine wesentliche Kraft zur Veränderung der Entwurfsverfahren war und ist bis heute das unter dem Begriff „Design Productivity Gap“ zusammengefasste Problem, dass die Produktivität der Schaltungsentwickler nicht mehr mit den technologischen Möglichkeiten bei der

Halbleiterfertigung Schritt halten kann (vgl. Bild 1.1). Während durch umfangreiche technologische Innovationen der Integrationsgrad von der Medium Scale Integration MSI mit einigen 1000 Transistoren pro Chip Anfang der 1980er-Jahre über die Very Large Scale Integration VLSI auf die heutige Ultra Large Scale Integration ULSI mit einigen 100 Millionen Transistoren pro Chip angestiegen ist, konnte die Entwicklerproduktivität nicht in diesem Maße gesteigert werden. Obwohl damals bereits von jedem Entwickler pro Monat etwa hundert Transistoren entworfen und verdrahtet wurden und dies heute im Mittel mehr als Hunderttausend sind, reicht die Entwicklerproduktivität heute bei Weitem nicht aus, um den technologischen Fortschritt ausschöpfen zu können. Wie Bild 1.1 zeigt, ist die etwa 1000-fache Verbesserung der Entwicklerproduktivität auf die Einführung neuer Methoden des Digitalentwurfs zurückzuführen. Dabei übernehmen Softwarewerkzeuge nicht nur die Platzierung und Verdrahtung sowie die Optimierung der Logik, sondern sie werden auch eingesetzt, um aus „Programmierdateien“, die das gewünschte Verhalten einer Schaltung beschreiben, weitgehend automatisch eine Schaltungsimplementierung zu generieren. Bild 1.1 zeigt auch, dass in Zukunft dringend weitere Innovationen bei den Entwurfsverfahren benötigt werden, wenn der Abstand zum technologischen Fortschritt der Halbleitertechnologie nicht noch größer werden soll.

Nun wird von Studierenden, die eine Grundlagenvorlesung in Digitaltechnik absolviert haben, sicher nicht erwartet, dass sie in der Lage sind, Digitalschaltungen mit einer derartigen Komplexität zu entwerfen, allerdings werden heute durchaus verbreitet im Rahmen von Diplom- oder Masterarbeiten Digitale Systeme mit mehr als 100.000 Transistoren entworfen.

Entsprechend hat dieses Lehrbuch den Anspruch, neben den unabdingbaren Grundlagen der Digitaltechnik auch eine Einführung in die Verhaltensbeschreibung digitaler Schaltungen mit der Hardwarebeschreibungssprache VHDL (Very High Speed Integrated Circuit Hardware Description Language) auf Register-Transfer-Abstraktionsniveau (Register-Transfer-Level, RTL) zu geben. Eine Einführung in die Architektursynthese (High-Level-Synthese) muss dagegen weiterführenden Lehrveranstaltungen und entsprechender Literatur zum Entwurf Digitaler Systeme überlassen bleiben [7], [15].

Für dieses, in meiner Lehre bereits seit vielen Jahren eingesetzte Konzept existieren nach meiner Überzeugung bisher keine geeigneten deutschsprachigen Lehrbücher. Die bisher auf dem Buchmarkt erhältlichen Lehrbücher behandeln die Einführung in VHDL in der Regel überwiegend in ergänzenden Kapiteln und nicht als integralen Bestandteil bei der Vermittlung der digitaltechnischen Konzepte.

Das nun vorliegende Lehrbuch, welches sich aus der Lehrveranstaltung Digitaltechnik speist, die von mir seit vielen Jahren an der Hochschule für Angewandte Wissenschaften Hamburg gehalten wird, soll diese Lücke füllen: Grundkonzepte der Hardwarebeschreibungssprache VHDL werden in nur zwei speziellen Kapiteln eingeführt, weitere syntaktische und semantische Erweiterungen dieser Sprache werden jeweils direkt im Zusammenhang mit dem Grundwissen über die digitalen Grundbausteine vermittelt. Entsprechend kann das Lehrbuch auch nicht den Anspruch auf eine vollständige VHDL-Darstellung erfüllen. Dazu gehört auch, dass die VHDL-Konstrukte überwiegend mit Beispielen und nicht durch vollständige Syntaxbeschreibungen vorgestellt werden.

Dieses Lehrbuch kann somit als Kompromiss zwischen einem Standardlehrbuch zur Digitaltechnik und einem Lehrbuch zur Einführung von VHDL aufgefasst werden. Dabei musste selbstverständlich auf einige Vertiefungen, wie sie sich z.B. in [5] und [6] finden, verzichtet werden. Ich bin jedoch davon überzeugt, dass die gewählte Stoffauswahl einen guten Kompromiss darstellt.

Die im Buch gewählten Anwendungsbeispiele sind durch das Symbol B herausgehoben, sie wurden unter dem Aspekt ausgewählt, dass sie einen praktischen Bezug zu wichtigen Grundschaltungen der Digitaltechnik haben. Dabei konnte leider nicht in allen Fällen vermieden werden, dass ein Vorgriff auf Lehrinhalte erfolgte, die erst an späterer Stelle des Lehrbuchs umfassend erläutert werden. In jedem Fall wurde jedoch versucht, die notwendigen Informationen bereits an dieser Stelle zu vermitteln. An einigen Stellen des Textes sind durch das Symbol A markierte Aufgaben eingebettet, mit denen das gerade Erlernte vertieft werden soll. Durch das Symbol T sind Tipps markiert, die den Lehrstoff in einem erweiterten Zusammenhang erscheinen lassen. Das Ende derartiger Einschübe ist durch das Symbol ■ gekennzeichnet.

1.1 Die Hardwarebeschreibungssprache VHDL

Als Hardwarebeschreibungssprachen weit verbreitet sind VHDL und Verilog [23]. Letztere ist integraler Bestandteil der umfassenden und entsprechend teuren Entwicklungssoftware, die in international agierenden Unternehmen eingesetzt wird. Im Gegensatz dazu gibt es im Bereich der kleinen und mittelständischen Unternehmen sowie der europäischen Hochschulen eine deutliche Präferenz für VHDL.

Bei der Darstellung von VHDL in diesem Lehrbuch werden vorwiegend solche Sprachkonstrukte vorgestellt, die auch synthesefähig sind, für die also durch Hardwarecompiler eine Datei für reprogrammierbare Digitalhardware erstellt werden kann. Trotz der als „Programmiersprache“ erscheinenden VHDL-Syntax sollte sich der Hardwareentwickler jedoch immer vor Augen führen, dass VHDL eine Sprache ist, die es erlaubt, die Nebenläufigkeit der verschiedenen Funktionen eines Designs zu repräsentieren, also die gleichzeitige Ausführung unterschiedlicher Hardwareaktionen. Diese Nebenläufigkeit bereitet dem Neuling, der in der Regel bereits Erfahrung mit prozeduralen oder objektorientierten Programmiersprachen wie z.B. C, C++ oder Java besitzt, erfahrungsgemäß anfangs einige Schwierigkeiten. Da es für die verwendeten Hardwarecompiler spezielle VHDL-Codierungsempfehlungen gibt, die durch die IEEE-Norm 1076.3 [12] definiert sind, und in denen nicht der vollständige VHDL-Sprachumfang genutzt werden kann, empfiehlt es sich, die in diesem Lehrbuch vorgestellten VHDL-Beispiele als Entwurfsmuster (Templates) für die Standardbauelemente der Digitaltechnik aufzufassen: Der fortgeschrittene Hardwaredesigner sollte weiter in Standardkomponenten wie einfachen Boole’schen Logikbausteinen (Logikgattern), komplexeren De-, Multiplexern etc. sowie diversen Ausführungsformen von Flipflops (Registern), Zählern und Zustandsautomaten denken, er sollte aber auch die Templates verinnerlicht haben, mit denen sich diese Funktionen in VHDL synthesegerecht modellieren lassen. Die langjährige Praxis mit Studierenden sowie das Feedback der Praktiker in der Industrie haben gezeigt, dass die

Einhaltung einiger wesentlicher Codierungsrichtlinien einen VHDL-Code mit übereinstimmender Simulations- und Synthesesemantik garantiert und somit einen Entwurfsablauf gewährleistet, der frei von Überraschungen ist.

1.2 Digitale und Analoge Signale

Die Digital- hat ebenso wie die Analogtechnik die Aufgabe, Signale zu verarbeiten. Dabei dienen die Signale der Erfasssung, Speicherung und Verarbeitung von Nachrichten. Die durch Sensoren erfassten Signale beschreiben physikalische Größen wie Spannung, Strom, Kraft, Druck, Frequenz usw. und erzeugen in der Regel ein analoges Ausgangssignal, welches durch Analog/Digital-Umsetzer digitalisiert wird. Die digital verarbeiteten Signale werden an Anzeigen weitergeleitet oder durch Digital/Analog-Umsetzer rückgewandelt, so dass analog operierende Aktoren daraus wieder eine physikalische Größe machen können.

Charakteristisch für analoge Signale ist der kontinuierliche Signalwertbereich zwischen zwei sensorbedingten Grenzwerten. Im Gegensatz dazu besitzen digitale Signale nur eine endliche Zahl diskreter Werte. Voraussetzung für die digitaltechnische Verarbeitung physikalischer Signale ist also eine Wertdiskretisierung, die im Analog/Digital-Umsetzer mit gestufter Übertragungscharakteristik vorgenommen wird.

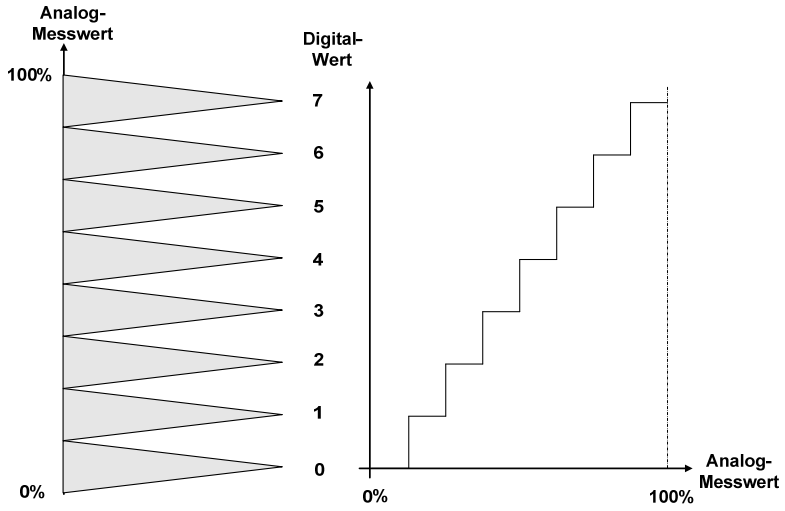


Bild 1.2: Diskretisierung eines wertkontinuierlichen Analogsignals durch eine Kennlinie mit schrittweiser Übertragungscharakteristik

Wie Bild 1.2 zeigt, wird immer ein ganzer Bereich von Analogwerten auf einen Digitalwert abgebildet. Dies bedeutet einen Informationsverlust, der als Quantisierungsfehler bezeichnet

wird, denn nach der Wertdiskretisierung ist unklar, auf welchen Analogwert innerhalb eines Quantisierungsintervalls das Digitalsignal ursächlich zurückzuführen war. Dieser prinzipielle Nachteil der digitalen Signalverarbeitung wird aber dadurch gemildert, dass die Anzahl der verwendeten diskreten Digitalwerte vergrößert wird.

Im Vergleich zur Analogtechnik besitzt die Digitaltechnik den wesentlichen Vorzug, dass in den allermeisten Fällen letztlich nur die zwei Signalzustände 0 und 1 verwendet werden. Dies hat Vorteile für:

- Eine sichere Reproduzierbarkeit und garantierte Rauschfreiheit,
- hohe Langzeitstabilität,
- Reprogrammierbarkeit,
- sichere Datenübertragung,
- eine hohe Datenkompressionsrate.

Neben der Wertdiskretisierung ist in der Digitaltechnik auch die Zeitdiskretisierung bedeutsam. Bei einem zeitdiskreten digitalen Signal ist garantiert, dass der wertdiskrete Signalwert für eine gewisse Zeit konstant bleibt. Dies erfordert die Vorgabe eines Arbeitstaktes bei der Verarbeitung der digitalen Signale. Digitale Systeme, die einen, meist mit „Clock“ bezeichneten Arbeitstakt besitzen, werden als synchrone Systeme bezeichnet und der Takt, mit dem der Analog/Digital-Umsetzer arbeitet, wird als Abtasttakt bezeichnet.

1.3 Digitale Systeme

Die Bedeutung digitaler Systeme hat in den letzten Jahren im Vergleich zu analogen Systemen erheblich zugenommen. Dafür sind drei Gründe ausschlaggebend:

- Technologische Fortschritte bei der Technologie von Analog/Digital-Umsetzern, die es erlauben, mit höheren Abtastfrequenzen zu arbeiten.
- Eine zunehmende Strukturverkleinerung bei gleichzeitigem Preisverfall in der Halbleiterindustrie.
- Technologische Fortschritte bei Verfahren zur effizienten Kompression digitaler Daten.

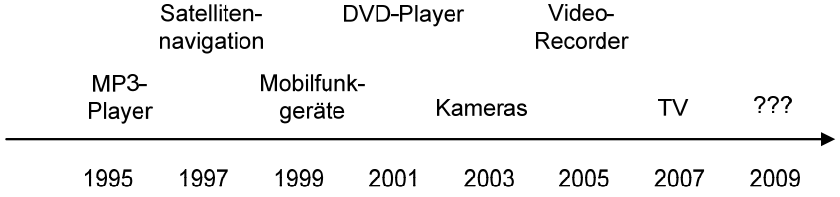


Bild 1.3: Zeitleiste zur Einführung digitaler Systemlösungen für vormals analoge Produkte

Daher wurden in den letzten Jahren viele vormals analog operierende Anwendungen auf Digitaltechnologie umgestellt (vgl. Bild 1.3).

In dem vorliegenden Buch zur Digitaltechnik sollen die Grundlagen zum Entwurf digitaler Systeme vermittelt werden. Dabei geht es nicht nur darum, grundlegende Konzepte zu vermitteln, vielmehr sollte sich der Leser beim Durcharbeiten dieses Lehrbuchs eine Wissensbibliothek mit den verschiedenen Grundbausteinen der Digitaltechnik anlegen. Für jeden dieser Bausteine sollten nach Durcharbeiten dieses Lehrbuchs die folgenden Informationen hinterlegt sein:

- Die Funktion des Bausteins
- Das zeitliche Ansteuerverhalten des Bausteins: Welche Signale müssen zu welchem Zeitpunkt am Eingang anliegen, welche Signale erscheinen zu welchem Zeitpunkt am Ausgang?
- Ein synthesefähiges Entwurfsmuster für den VHDL-Code

Exemplarisch soll nachfolgend an einem Beispiel der digitalen Bildbearbeitung erläutert werden, um welche Klassen digitaler Hardware es sich dabei handelt:

Ein reprogrammierbarer digitaler Hardwarebaustein (Field Programmable Gate Array, FPGA) hat die Aufgabe, den Kontrast der Bilder einer Kamera unter Echtzeitbedingungen so zu verstärken, dass ein über einen PCI-Bus angeschlossener PC diese weiterverarbeiten kann (vgl. Bild 1.4).

Bild 1.4: Kontrastarmes Kamerasignal a) und Ergebnis der Kontrastverstärkung b)

Da die Kamera die einzelnen Bildpunkte (Pixel) als Grauwertsignal zwischen 0 (schwarz) und 255 (weiß) nicht über den ganzen Wertebereich liefert, besteht die Aufgabe, die schmale Grauwertverteilung des Quellbilds a) so zu strecken, dass der gesamte Grauwertbereich genutzt wird. Bild 1.5 zeigt die Grauwertverteilung der Pixel aus Bild 1.4a sowie die entsprechende Verteilung nach der Bearbeitung durch den FPGA-Baustein. Die in dem FPGA verwendete Hardwarearchitektur zur Grauwert-Äqualisation zeigt Bild 1.6.

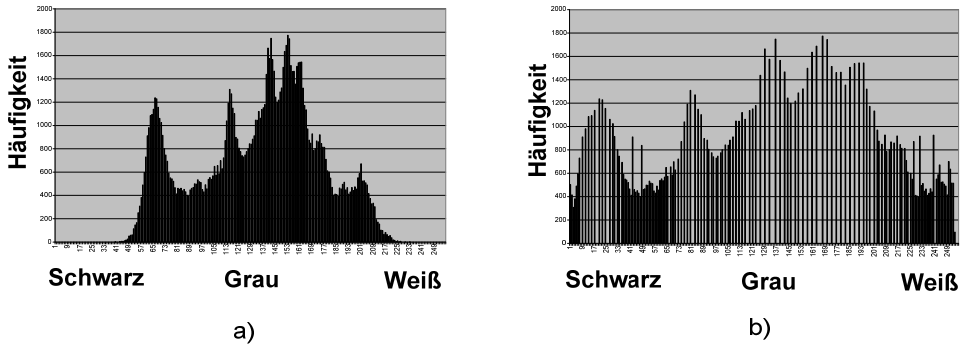


Bild 1.5: Häufigkeitsverteilung der Grauwerte im Quellbild a) und nach der Kontrastverstärkung b)

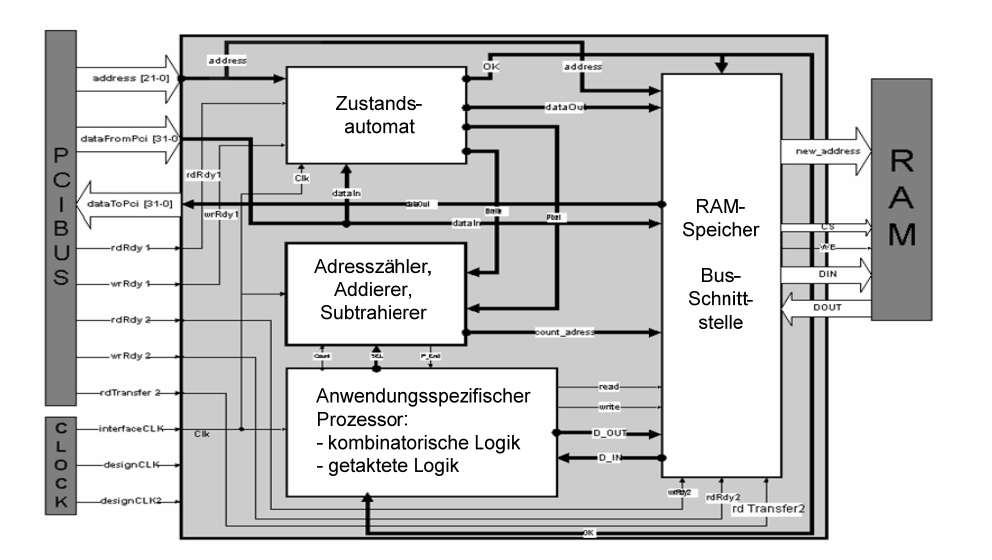


Bild 1.6: Architektur zur Kontrastverstärkung eines Grauwertsignals

Ohne hier auf Details eingehen zu wollen, sind dennoch typische Elemente eines digitalen Systems erkennbar, deren Eigenschaften in den nachfolgenden Kapiteln erläutert werden:

- Kombinatorische Logik
- Addierer und Subtrahierer
- Speicher mit bidirektionalen Busschnittstellen
- Zähler und Schieberegister
- Zustandsautomaten

1.4 Gliederung des Buches

Im nachfolgenden Kap. 2 werden zunächst einige Grundbegriffe zum Verständnis der Entwurfsmethodik für digitale Systeme erläutert. Dazu gehören insbesondere Modellierungskonzepte, ohne die heute kein komplexes digitales System mehr auskommt.

In vier Kapiteln werden die wesentlichen Grundlagen zum Entwurf kombinatorischer Schaltungen gelegt, die in keinem Lehrbuch der Digitaltechnik fehlen dürfen. Dazu gehören die Gesetze und Methoden beim Entwurf mit einfachen Logikgattern (Boole'sche Algebra, Kap. 3), die Beschreibung der in der Digital- und Mikroprozessortechnik verwendeten Zahlensysteme (Kap. 5) sowie die Beschreibung von Methoden zum Entwurf von kombinatorischen Digitalschaltungen mit minimalem Hardwareaufwand (Kap. 6). Die Definition von Code-Kenngrößen sowie die Erläuterung einiger häufig verwendeter Codes der Digitaltechnik in Kap. 8 zählen ebenfalls zu den Grundlagen der kombinatorischen Schaltungstechnik.

Quasi „zwischendurch", nämlich in den Kap. 4 und 7 erfolgt eine grundlegende Einführung in die Konzepte der Hardwarebeschreibungssprache VHDL, wobei zunächst der Schwerpunkt auf die Hardwaremodellierung mit nebenläufigen Anweisungen gelegt wird und in Kap. 7 das allgemein gültige Konzept der Modellierung mit Prozessen erläutert wird. In den nachfolgenden Kapiteln werden an Hand konkreter Beispiele immer mehr VHDL-Syntaxkonstrukte eingeführt, wobei das Ziel darin besteht, typische Entwurfsmuster für die in der Digitaltechnik verwendeten Hardwarekomponenten zu erlernen.

Das Kap. 9 beschreibt die physikalische Implementierung und Beschaltung von CMOS-Logikgattern. Ausgehend vom Aufbau und den Kennlinien der MOS-Transistoren wird zunächst das Umschaltverhalten des CMOS-Inverters und nachfolgend die Struktur einfacher Logikgatter erläutert. Neben den Pegelbereichen aktueller Logikfamilien werden drei verschiedene Ausgangsschaltungen von CMOS-Gattern vorgestellt und diese auch in VHDL modelliert.

Mit dem Kap. 10 wird die Ebene einfacher Logikgatter verlassen und es erfolgt eine Beschreibung komplexerer kombinatorischer Komponenten, die sich üblicherweise im „Datenpfad" eines digitalen Systems befinden. Dazu gehört neben Multiplexern, Demultiplexern und Prioritätsencodern auch Arithmetikhardware wie Komparatoren, Addierer und Multiplizierer. Diese Komponenten werden unter Einführung neuer VHDL-Syntaxkonstrukte durch geeignete Entwurfsmuster modelliert.

Die weiteren Kap. 11 bis 14 sind der Beschreibung bzw. Modellierung getakteter Hardware gewidmet: Beginnend bei taktzustandsgesteuerten Latches werden in Kap. 11 zunächst die flankengesteuerten Flipflops eingeführt. Diese dienen nicht nur als Grundlage für den Entwurf endlicher Zustandsautomaten, dessen Konzepte in Kap. 12 vermittelt werden, sondern auch für den Entwurf von Synchronzählern (Kap. 13) und Schieberegistern (Kap. 14). Bei den in diesen Kapiteln vorgestellten VHDL-Konzepten steht eine synthesegerechte Modellierung im Vordergrund, die insbesondere das Taktflankenverhalten korrekt abbildet.

In den letzten zwei Kapiteln wird der innere Aufbau und das Verhalten digitaler Halbleiterspeicher (Kap. 15) und programmierbarer Logik (Kap. 16) vorgestellt. Auf diese Bausteine

wird an mehreren Stellen des Buches verwiesen. Daher wird dem Leser empfohlen, sich während des Durcharbeitens der anderen Kapitel vertiefende Informationen hier zu holen. Als Beispiel für die laufende Weiterentwicklung der VHDL-Synthesestandards enthält Kap. 15 auch synthesegerechte VHDL-Modelle zur Implementierung von RAM- und ROM-Speichern auf FPGAs.

Im Anhang findet der Leser neben einer Einführung in die Nutzung des VHDL-Simulators ModelSim eine Zusammenfassung der wesentlichen Codierungsrichtlinien, die einen synthesegerechten VHDL-Entwurf digitaler Schaltungen sicherstellen.

1.5 Vertiefende Aufgaben

A Aufgabe 1.1

Beantworten Sie die folgenden Verständnisfragen:

a) Was versteht man unter dem Begriff „Design Productivity Gap“? Welche Maßnahmen wurden in der Vergangenheit aus dem damit beschriebenen Konflikt abgeleitet?
b) Worin unterscheiden sich digitale und analoge Signale?
c) Welche Vorteile bieten digitaltechnische Systemlösungen im Vergleich zu analogen?
d) Wodurch ist ein synchrones System gekennzeichnet? ■

2 Modellierung digitaler Schaltungen

In diesem einführenden Kapitel sollen grundlegende Konzepte des Entwurfs digitaler Schaltungen vorgestellt werden, zu denen insbesondere die Modellierung mit Hardwarebeschreibungssprachen zählt. Dabei werden Begriffe erläutert, die zum Verständnis eines modernen digitalen Systementwurfs erforderlich sind.

2.1 Lernziele

Nach Durcharbeiten dieses Kapitels sollen Sie

- das Y-Modell des Entwurfsablaufs für digitale Systeme kennen und wissen, dass bei einem Entwurf verschiedene Sichten und unterschiedliche Abstraktionsebenen betrachtet werden müssen;
- die Bedeutung der Modellierung durch Hardwarebeschreibungssprachen verstanden haben und die unterschiedlichen Modellierungsstile kennen;
- die grundlegenden Eigenschaften kombinatorischer und getakteter Logik kennen;
- das Konzept der Modellierung auf Register-Transfer-Ebene verstanden haben;
- die wesentlichen Schritte zum Entwurf programmierbarer Digitalschaltungen kennen.

2.2 Entwurfssichten und Abstraktionsebenen

Der Entwurf einer digitalen Schaltung bzw. eines digitalen Systems bedeutet die Umsetzung einer Produktidee in eine produktionsfähige Beschreibung. Da die Herstellung weitgehend rechnergestützt erfolgt, wird dies in der Regel eine während des Entwicklungsprozesses erstellte Implementierungsdatei sein, die während der Fertigung von einer geeigneten Software interpretiert wird. Diese Datei kann z.B. enthalten:

- Informationen zur Herstellung eines Platinenlayouts,
- Daten zur Programmierung von programmierbaren Logikbausteinen wie CPLDs und FPGAs (vgl. Kap. 16),
- Informationen zur Maskenherstellung beim Entwurf einer applikationsspezifischen integrierten Schaltung (Application Specific Integrated Circuit, ASIC).

Selbstverständlich unterliegt auch der Entwurfsprozess einem erheblichen Konkurrenz- und Kostendruck. Entwurfsfehler, die durch eine geänderte Implementierungsdatei (Redesign) behoben werden müssen, sind meist sehr kostenintensiv und sollten daher unbedingt vermieden werden.

Die anzustrebende Vorgehensweise setzt daher einen umfassenden Einsatz von Simulationswerkzeugen voraus. Diese interpretieren Modelle der zu implementierenden Schaltung bzw. des Systems, die entweder in einer Programmiersprache wie z.B. C oder C++, oder aber in einer geeigneten Hardwarebeschreibungssprache (Hardware Description Language, HDL) erstellt wurden. Selbstverständlich ist dabei zu berücksichtigen, dass ein Modell immer nur Teilaspekte des Schaltungsverhaltens modellieren kann. Dabei müssen während des Entwurfsprozesses unterschiedliche Sichten betrachtet werden:

- das Verhalten der Schaltung bzw. des Systems,
- die Strukturierung des Systems in Teilaufgaben,
- die geometrisch-physikalische Anordnung der zur Lösung von Teilaufgaben verwendeten Komponenten.

In dem von D.D. Gajski [22] eingeführten Y-Diagramm des Entwurfsprozesses werden diese Sichten grafisch durch drei Achsen dargestellt (vgl. Bild 2.1).

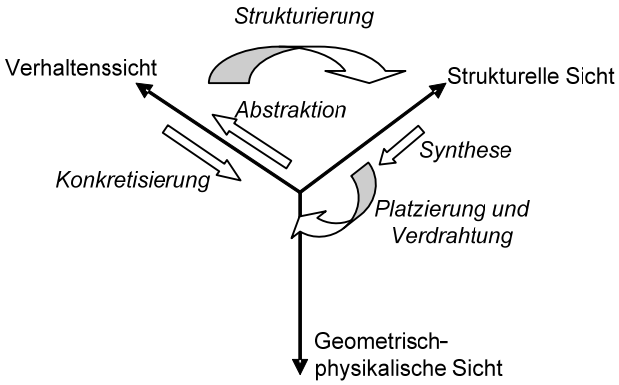


Bild 2.1: Darstellung der Entwurfsschritte im Y-Diagramm

Digitalschaltungen sind wesentlicher Bestandteil nicht nur von Mikroprozessoren, sondern auch von Geräten für Kommunikations- bzw. Konsumeranwendungen, wie z.B. Mobiltelefonen, Videospielkonsolen, portablen Media-Playern oder Satellitennavigationsgeräten. Üblicherweise besitzen diese Anwendungen einen oder mehrere Prozessoren, von denen einer oder mehrere Software-programmierbar sind. Die Entwurfsaufgabe besteht also aus einem Software- wie einem Hardwareentwurf (Hardware-Software-Codesign). Die Komplexität derartiger Anwendungen erlaubt es mit den heute zur Verfügung stehenden Digitalsimulatoren nicht, alle Detaileigenschaften des Systems mit einem einzelnen Simulationsmodell

nachzubilden, vielmehr ist es insbesondere zu Beginn der Systementwicklung erforderlich, das Gesamtsystem auf einer höheren und damit abstrakteren Ebene zu modellieren, welche wesentliche Detailfragen bewusst offen lässt. Eine derartige Systemsimulation ist die Grundlage für Entscheidungen über Entwurfsalternativen und erlaubt den Übergang auf eine niedrigere Abstraktionsebene. Auf dieser Ebene wird das Systemmodell häufig durch einen strukturierenden Konkretisierungsschritt in Teilmodelle aufgespalten, die Teilkomponenten des Systems repräsentieren. Die Modellierung dieser Komponenten erfolgt mit höherem Detaillierungsgrad und, sofern es sich um Hardwarekomponenten handelt, mit einer Hardwarebeschreibungssprache. In einem weiteren Transformationsschritt, bei dem auf die geometrisch-physikalische Sicht gewechselt wird, erfolgt nun die Entscheidung, wie die einzelnen Hardwarekomponenten relativ zueinander angeordnet werden.

Den Übergang von einer höheren zu einer niedrigeren Abstraktionsebene bezeichnet man als Synthese und die konkrete Anordnung der Komponenten erfordert eine Platzierung sowie eine Verdrahtung (engl. place and route) (vgl. Bild 2.1).

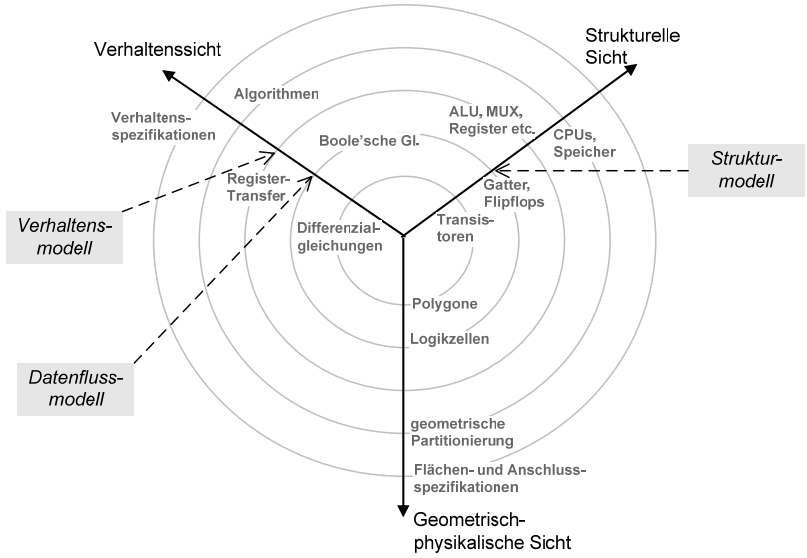


Bild 2.2: Das Entwurfsdiagramm nach Gajski [22] mit typischen HDL-Modellierungsebenen (vgl. Kap. 2.3)

In dem in Bild 2.2 dargestellten, nach Gajski benannten Diagramm werden für die verschiedenen Sichten konkrete Abstraktionsebenen vorgeschlagen, in denen die Entwurfsschritte durch Transformationen zwischen einerseits den Entwurfssichten und andererseits den Entwurfsebenen aufgefasst werden [22]. Ziel ist es dabei, von der äußeren Spezifikationsebene durch konzentrische Transformationen zum Kreismittelpunkt zu gelangen, in dem der höchste Detaillierungsgrad vorliegt:

- Alle Logikgatter wurden in Transistoren aufgelöst,
- deren Verhalten wird durch Differenzialgleichungen beschrieben und
- deren Hardwareimplementierung erfolgt durch photolithografisch genutzte Masken, die durch Polygone beschrieben werden.

In diesem einführenden Lehrbuch zur Digitaltechnik können nicht alle Teilaspekte aus Bild 2.2 behandelt werden, sondern wir beschränken uns auf:

- Die Verhaltensmodellierung von einfachen Logikgattern und Flipflops durch Boole'sche Gleichungen (vgl. Kap. 3), sowie die
- Modellierung des Zusammenspiels von komplexeren kombinatorischen Schaltungskomponenten wie z.B. Multiplexern (MUX) und arithmetisch logischen Recheneinheiten (Arithmetical Logical Unit, ALU) (vgl. Kap. 10) etc. mit Registern (vgl. Kap. 11) bzw. Speichern (vgl. Kap. 15) auf der Register-Transfer-Ebene (vgl. Kap. 2.4.3).

2.3 Modellierung mit Hardwarebeschreibungssprachen

Zur Modellierung digitaler Hardware werden heute überwiegend Beschreibungssprachen der zweiten Generation eingesetzt: Verilog [23] und VHDL (Very High Speed Integrated Circuit Hardware Description Language) [9].[1] Obwohl beide Sprachen die Simulation und rechnergestützte Synthese digitaler Hardware erlauben, erfolgt die Modellierung in diesem Lehrbuch ausschließlich mit VHDL. Dabei wird besonders darauf geachtet, einen Entwurfsstil zu verwenden, der eine automatische Synthese garantiert. Nachfolgend soll gezeigt werden, dass die gleiche Problemstellung gemäß Bild 2.2 auf verschiedene Weise modelliert werden kann. Die hier vorgestellten VHDL-Codebeispiele sollen dazu dienen, dem Leser einen ersten Einblick in die Vielfalt der Modellierungsstile zu geben, ohne dass auf syntaktische Details geachtet werden muss.

Exemplarisch soll ein 2-zu-1-Multiplexer (MUX) betrachtet werden, dessen Aufgabe darin besteht, abhängig von einem Steuersignal S eines der beiden Eingangssignale IA oder IB auf den Ausgang Y zu schalten (vgl. Bild 2.3).

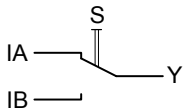


Bild 2.3: Schaltermodell eines 2-zu-1-Multiplexers

[1] Als Hardwarebeschreibungssprachen der dritten Generation werden z.B. SystemC [24] und System-Verilog [25] bezeichnet. Insbesondere SystemC bietet im Gegensatz zu den Sprachen der zweiten Generation auch eine Hardware-Software-Cosimulation, da SystemC als C++-Klassenbibliothek mit integriertem Simulationskern implementiert wird.

2.3.1 Datenflussmodelle

In einem Datenflussmodell wird das Verhalten der zu entwerfenden Schaltung durch eine relativ geringe Zahl logischer Grundfunktionen realisiert. In Beispiel 3.14 wird gezeigt, dass die Funktion des Multiplexers durch zwei UND-Gatter, einen Inverter und ein ODER-Gatter dargestellt werden kann (vgl. Bild 3.14). Diese Boole'schen Funktionen werden durch die im Listing 2.1 verwendeten VHDL-Operatoren `and`, `not` und `or` implementiert. Im Gajski-Diagramm findet sich diese Art der Modellierung auf der, von innen gesehen, zweiten Abstraktionsebene der Verhaltensachse.

Listing 2.1: VHDL-Datenflussbeschreibung für einen 2-zu-1-Multiplexer

```
architecture DATENFLUSS of MUX is
signal NODE1, NODE2, NODE3: bit; -- lokale Koppelsignale
begin
   NODE1 <= IB and S;
   NODE2 <= not S;
   NODE3 <= IA and NODE2;
   Y <= NODE1 or NODE3;
end DATENFLUSS;
```

2.3.2 Strukturmodelle

In Strukturmodellen werden die Ein- und Ausgänge von Komponenten, die in einer Bibliothek abgelegt sind, durch lokale Signale miteinander verbunden. Auf diese Weise wird eine so genannte Netzliste aufgebaut. Im Gegensatz zur Datenflussbeschreibung, bei der die Anzahl der unterschiedlichen logischen Verknüpfungen recht gering ist, können alle Komponentenmodelle verwendet werden, die zuvor in einer Bibliothek abgelegt wurden. In VHDL stehen alle Elemente der Bibliothek mit dem Namen „work" automatisch zur Verfügung. Die in Listing 2.2 dargestellte Netzliste des 2-zu-1-Multiplexers verwendet die zuvor in der Bibliothek abgelegten Komponenten `UND`, `INVERTER` und `ODER`, deren Ein- und Ausgänge mit den Eingangs- und Ausgangssignalen bzw. den lokalen Signalen des Multiplexers verbunden werden (vgl. Bild 3.14).

Da alle Komponenten selbst aus anderen Komponenten bestehen können, lassen sich mit Strukturmodellen auch tiefere Entwurfshierarchien beschreiben. Im Gajski-Diagramm findet sich diese Art der Modellierung ebenfalls auf der zweiten Abstraktionsebene, allerdings auf der Strukturachse.

Listing 2.2: VHDL-Strukturmodell des 2-zu-1-Multiplexers

```
architecture STRUKTUR of MUX is
signal NODE1, NODE2, NODE3 : bit;
begin
   U1: UND port map(IB, S, NODE1);
   U2: INVERTER port map(S, NODE2);
   U3: UND port map(NODE2, IA, NODE3);
   U4: ODER port map(NODE1, NODE3, Y);
end STRUKTUR;
```

2.3.3 Verhaltensmodelle

Bei dieser Art der Modellierung soll das Verhalten der Schaltung durch Sprachelemente ähnlich wie in einer prozeduralen Programmiersprache dargestellt werden. Abgeschlossene Hardware-Funktionsblöcke werden durch Prozesse abgebildet. Innerhalb von Prozessen sind u.a. Schleifen und bedingte Verzweigungen erlaubt. Diese Art der Modellierung ist sehr flexibel und erfordert zunächst kein vertieftes Verständnis über die zugrunde liegende Hardware. Entsprechend finden sich Verhaltensmodelle, die im Gajski-Diagramm als Register-Transfer-Modell bezeichnet sind, auf einer höheren Abstraktionsebene als die bisher vorgestellten Modellierungsstile. Ein entsprechendes Multiplexermodell zeigt Listing 2.3.

Listing 2.3: Verhaltensmodell des 2-zu-1-Multiplexers

```
architecture VERHALTEN of MUX is
begin
P1: process(IA, IB, S)
begin
    if S = '1' then
       Y <= IB;
    else
       Y <= IA;
   end if;
end process P1;
end VERHALTEN;
```

2.4 Kombinatorische und getaktete Logik

Voraussetzung für die Anwendung rechnergestützter Synthesewerkzeuge auf Verhaltensebene ist ein spezieller Entwurfsstil. Bei der Register-Transfer-Modellierung (Register-Transfer-Level, RTL) werden zwei unterschiedliche Arten digitaler Hardware sorgfältig voneinander getrennt und in unterschiedlichen Entwurfseinheiten modelliert. Dabei handelt es sich um kombinatorische und getaktete Logik. In den nachfolgenden Kapiteln werden Sie diverse Hardwarefunktionsblöcke und deren VHDL-Modelle kennen lernen, sich also eine „Datenbank“ aufbauen, auf die Sie zurückgreifen können, wenn es darum geht, komplexere Systeme zu konzipieren. Dabei werden Sie in aller Regel zunächst prüfen müssen ob für die gewünschte Funktion ein kombinatorischer oder ein getakteter Block geeignet ist.

In Kap. 3 wird gezeigt, dass kombinatorische Logik prinzipiell auch durch ein geeignetes Netzwerk von Schaltern dargestellt werden kann. Wir sprechen daher auch von Schalt*netzen*. Mit getakteter Logik kann hingegen ein bestimmter logischer Schaltungszustand gespeichert werden (vgl. Kap. 12), man spricht von einem Schalt*werk*. Die Speicherung des Logikzustands erfolgt in einem Register, weswegen getaktete Logik häufig auch als Registerlogik bezeichnet wird.

2.4.1 Eigenschaften kombinatorischer Logik

Charakteristisch für kombinatorische Logikbausteine ist die nahezu sofortige Reaktion des Ausgangssignals auf Änderungen der Eingangssignale.

Jede Änderung eines der Eingangssignale kann zu einer Änderung des Ausgangssignals führen. Bild 2.4 zeigt dieses Verhalten am Beispiel eines ODER-Gatters (vgl. Kap. 3.3.3) mit den Eingängen A und B sowie dem Ausgang Y. Dargestellt ist das mit dem VHDL-Simulator ModelSim-XE (vgl. Kap. 17.1) ermittelte Impulsdiagramm, in dem die Boole'schen Signalwerte 0 und 1 entlang der Zeitachse dargestellt sind. Der Verlauf der Eingangssignale A und B wurde dem Simulator vorgegeben und das Ausgangssignal Y beschreibt das Verhalten des ODER-Gatters. Die Pfeile markieren die Wirkung der Eingangssignale auf den Ausgang. Die z.B. zum markierten Zeitpunkt t = 100 ns beobachtbare Verzögerung des Ausgangssignals Y resultiert aus der Signallaufzeit durch das Gatter.

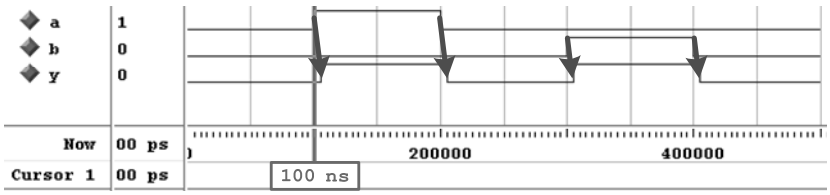


Bild 2.4: Kombinatorisches Schaltverhalten eines ODER-Gatters. Die Pfeile demonstrieren die unmittelbare Wirkung der Eingänge A und B auf den Ausgang Y.

2.4.2 Eigenschaften getakteter Logik

Charakteristisch für getaktete Logikbausteine ist die Tatsache, dass sich Ausgangssignale nur nach einer Pegeländerung (Flanke) eines vorgegebenen Taktsignals ändern können. Dabei kann ausgewählt werden, ob dies eine ansteigende ($0 \rightarrow 1$) oder eine abfallende ($1 \rightarrow 0$) Flanke sein soll.

Das Schaltsymbol und -verhalten eines getakteten D-Flipflops, welches seinen Signalwert D am Dateneingang nur bei einem Wechsel des Taktsignals CLK an den Ausgang Q weiterreicht (vgl. Kap. 11) zeigt Bild 2.5. Im Gegensatz zur kombinatorischen Logik reicht eine Änderung des Dateneingangssignals also nicht aus, damit sich ein getakteter Signalausgang ändert (z.B. bei t = 50 ns in Bild 2.5). Vielmehr wird das Eingangssignal D = 1 erst bei der ansteigenden Flanke von CLK zum Zeitpunkt t = 100 ns übernommen. Ähnlich wie bei der kombinatorischen Logik erscheint das Ausgangssignal Q wegen der Signallaufzeit durch das Flipflop bezogen auf die Flanke leicht verzögert. Bild 2.5 macht im Zeitbereich zwischen

t = 250 und 275 ns auch deutlich, dass eine Änderung des Dateneingangs D nicht immer an den Flipflopausgang weitergereicht wird, also auch „überlesen“ werden kann.

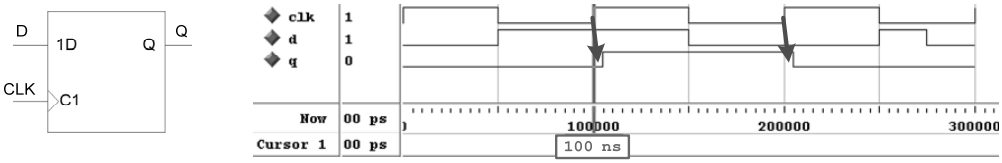


Bild 2.5: Schaltsymbol und Verhalten eines getakteten Logikbausteins D-Flipflop. Die Pfeile zeigen die Wirkung des Taktsignals CLK auf den Ausgang Q.

2.4.3 Modellierung auf Register-Transfer-Ebene

Die Vehaltensmodellierung auf Register-Transfer-Ebene (engl. Register-Transfer-Level, RTL) erfordert, dass die kombinatorische Hardware von der getakteten Hardware getrennt, also in unterschiedlichen Prozessen (vgl. Kap. 7) modelliert wird.

Ursache dieser Restriktion sind die Grenzen der VHDL- bzw. Verilog-RTL-Synthesewerkzeuge. Das in Bild 2.6 dargestellte Verhalten einer gemischt kombinatorischen und getakteten Schaltung lässt sich in VHDL für die Simulation zwar in einem Prozess modellieren, allerdings wird dieser nicht synthesefähig sein. Vielmehr gilt es festzuhalten, dass der VHDL-Sprachumfang für die Synthese auf eine Teilmenge einzuschränken ist, die durch die IEEE-Norm 1076.3 definiert ist [12]. Für die in Bild 2.6 dargestellte Schaltung bedeutet dies, dass vier Prozesse erforderlich sind:

- je ein Prozess für die drei kombinatorischen Schaltungsteile,
- ein Prozess für beide getakteten Schaltungsteile[2].

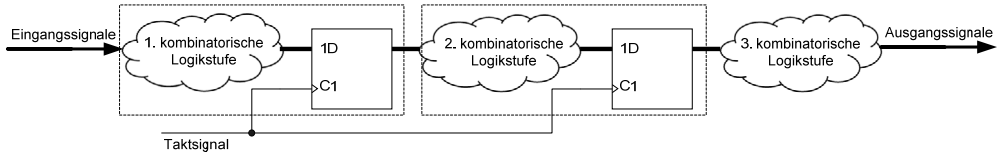


Bild 2.6: Register-Transfer-Modell für digitale Schaltungen

[2] Tatsächlich kann getaktete Hardware nur dann in einem Prozess zusammengefasst werden, wenn sie mit der gleichen Flanke betrieben wird. In Kap. 13 wird ein einfacheres RTL-Modellierungskonzept vorgestellt, bei dem kombinatorische Logik an den Eingängen getakteter Logik gemeinsam in einem Prozess modelliert werden kann (vgl. die gestrichelten Rahmenlinien in Bild 2.6).

2.5 Entwurfsmethodik für programmierbare digitale Schaltungen

Ein typischer Entwurfsablauf für eine Digitalschaltung, die in einem programmierbaren Baustein implementiert werden soll, ist in Bild 2.7 dargestellt.

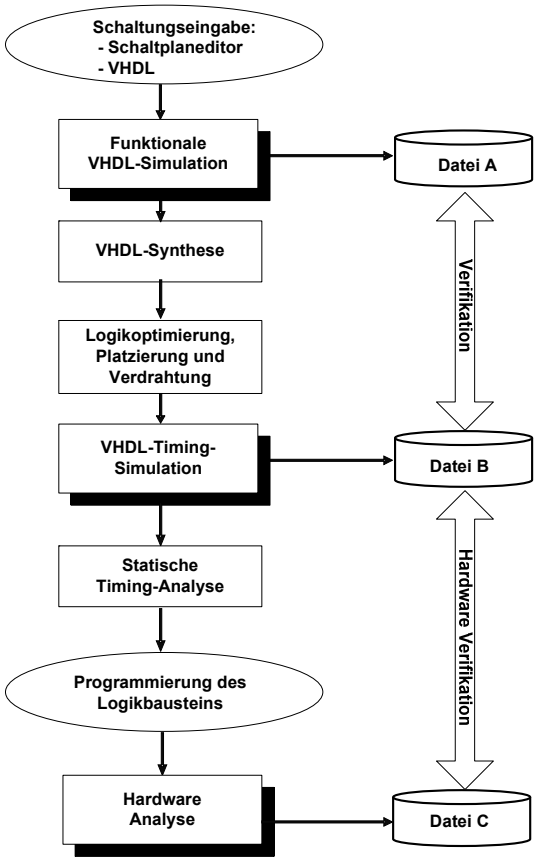


Bild 2.7: Entwurfsablauf für Digitalschaltungen

Dieser lässt sich wie folgt beschreiben:

- Die Entwurfseingabe erfolgte früher mit einem grafischen Schaltplaneditor, bei Verwendung einer Hardwarebeschreibungssprache jedoch mit einem Texteditor.
- Anschließend wird eine funktionale VHDL-Simulation mit dem Ziel durchgeführt, zu verifizieren, ob das VHDL-Modell eine geeignete Lösung für die Entwurfsidee darstellt. Häufig werden die Ergebnisse für einen späteren Vergleich auch abgespeichert (Datei A).

- Der verifizierte VHDL-Code wird durch ein RTL-Synthesewerkzeug in eine generische Netzliste umgewandelt. Dabei bleibt zunächst noch unberücksichtigt, in welcher Art von programmierbarem Baustein der Entwurf später implementiert werden soll.
- Anschließend erfolgt die Hardware-Implementierungsphase, während der die generische Netzliste unter Berücksichtigung der in dem vorgesehenen Baustein tatsächlich zur Verfügung stehenden Hardwareressourcen optimiert wird. Außerdem erfolgt deren Platzierung und Verdrahtung (engl. place and route).
- Nachfolgend ist zu verifizieren, dass die während der automatischen Synthese- und Implementierungsschritte durchgeführten Transformationen korrekt waren, die implementierte Schaltung also das gleiche Verhalten hat, wie das RTL-Modell. Dazu wird von der Entwicklungsumgebung der VHDL-Code eines strukturellen Implementierungsmodells so erstellt, dass die tatsächlichen Signallaufzeiten durch die verwendeten Gatter sowie die Verdrahtung berücksichtigt werden. Mit diesem Modell erfolgt eine VHDL-Timing-Simulation, deren Ergebnis abgespeichert wird (Datei B).
- Für einen taktsynchronen Entwurf muss die maximale Taktfrequenz bestimmt werden, mit der die Schaltung betrieben werden kann. In dem RTL-Modell von Bild 2.6 bedeutet dies, dass die kombinatorische Logik bestimmt werden muss, die die längste Laufzeit zwischen zwei Registern besitzt. Dieser kritische Pfad wird in den meisten Entwicklungswerkzeugen durch eine automatisch durchgeführte statische Timinganalyse bestimmt.
- Nach Programmierung des Bausteins kann abschließend die Verifikation der Entwurfsidee in der Zielhardware erfolgen.

Bei dem in Bild 2.7 dargestellten Entwurfsablauf wurde auf eine technologieunabhängige funktionale Simulation nach der VHDL-Synthese verzichtet. Dies erscheint immer dann gerechtfertigt, wenn die Schaltung nicht zu komplex ist, der synthesegerecht modellierte VHDL-Code also gut überschaubar ist.

2.6 Vertiefende Aufgaben

A Aufgabe 2.1

Beantworten Sie die folgenden Verständnisfragen:

a) Was versteht man unter dem Begriff „Synthese“?
b) Worin unterscheiden sich Datenfluss-, Struktur- und Verhaltensmodelle?
c) Welche prinzipiellen Unterschiede bestehen zwischen kombinatorischer und getakteter Logik?
d) Welche Aussagen über die Entwurfsschritte für Digitale Systeme trifft das Gajski-Diagramm?
e) Worauf haben Sie zu achten, wenn Sie ein VHDL-Modell auf Register-Transfer-Ebene erstellen wollen? ■

3 Boole’sche Algebra

In diesem Kapitel werden die Grundfunktionen der Digitaltechnik eingeführt, deren theoretischer Hintergrund relativ einfach, aber dennoch sehr mächtig ist. Ein algebraisches Modell wurde 1854 von dem Mathematiker George Boole entwickelt. Die erste technische Anwendung dieser zweiwertigen Boole’schen Algebra wurde 1938 durch Claude E. Shannon gezeigt, indem er die Algebra auf Serien- und Parallelschaltungen von Schaltern und Relais anwendete. Auch die heute in der Digitaltechnik verwendeten elektronischen Schalter (Transistoren etc.) werden durch die Boole’sche Algebra modelliert. Aus diesem Grunde wird auch häufig der Begriff Schaltalgebra verwendet. Das Kapitel betrachtet die Schaltalgebra zunächst idealisiert, d.h. ohne die später einzuführenden Schaltverzögerungen der Bausteine.

3.1 Lernziele

Nach Durcharbeiten dieses Kapitels sollen Sie

- die Gesetze der Boole’schen Algebra kennen und diese zur Vereinfachung von Schaltfunktionen einsetzen können;
- Aufgabenstellungen aus dem Bereich der kombinatorischen Logik in Wahrheitstabellen abbilden können;
- in der Lage sein, die Inhalte einer Wahrheitstabelle als disjunktive und als konjunktive Normalform abbilden zu können;
- Logikbausteine wie XOR und XNOR kennen und vorteilhaft einsetzen können;
- die Wahrheitstabellen sowie die DIN- und US-Schaltsymbole der gebräuchlichen Gatter kennen und in der Lage sein, Logikschaltpläne zu erstellen.

3.2 Schaltvariable und Schaltfunktionen, Signale

In der Schaltalgebra werden die Konstanten 0 und 1 sowie Variable und Funktionen verwendet. Schaltvariable, die üblicherweise mit einem Namen (z.B. Y, B1 oder X1EN) bezeichnet werden, können ebenfalls nur die binären Werte 0 und 1 annehmen. Der Informationsgehalt der Schaltvariablen beträgt 1 Bit (Binary Digit). Das Verhalten kann man sich als geöffneten, bzw. geschlossenen Schalter vorstellen.

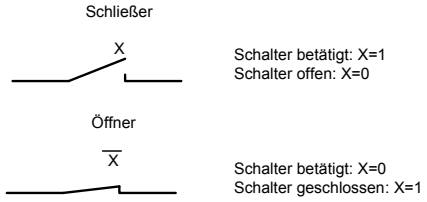


Bild 3.1: Modellierung von Schaltfunktionen durch schließende und öffnende Schalter

Mit Schaltfunktionen können Schaltvariable durch spezielle Transformationen, die auch als Gatterfunktionen bezeichnet werden, ineinander überführt werden.

$$Y = f(X_1, X_2, X_3, \ldots X_n) \quad \text{mit} \quad X_i, Y \in \{0, 1\}$$

Das Ergebnis einer Schaltfunktion ist 1 wenn durch die Schalteranordnung ein Strom fließen kann. Die physikalische Repräsentation einer Schaltvariablen wird als Signal bezeichnet. Häufig werden beide Begriffe auch synonym eingesetzt. Wegen der großen Bedeutung des Begriffs „Signal" im Unterschied zu einer „Variablen" bei der VHDL-Modellierung werde ich nachfolgend den Signalbegriff verwenden.

3.3 Elementare Schaltfunktionen

Schon eine geringe Anzahl elementarer Schaltfunktionen reicht aus, um ein Grundgerüst der Digitaltechnik aufzubauen. Ein möglicher Satz solcher Elementarfunktionen sind NICHT, UND und ODER. In diesem Abschnitt werden diese Gatterfunktionen mit einem und zwei Eingangssignalen vorgestellt. In der Praxis werden die UND- und ODER-Schaltfunktionen jedoch auch mit mehreren Eingangssignalen verwendet.

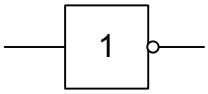
Es sei bereits hier darauf hingewiesen, dass die in diesem Abschnitt zusammengestellten Regeln nur für den idealisierten Fall gelten, dass die Ausgänge der Schaltfunktionen sofort auf den Eingang reagieren. Schaltzeiten bleiben also zunächst unberücksichtigt.

Üblicherweise wird kombinatorische Logik, die durch Schaltfunktionen repräsentiert wird, in Form einer Wahrheitstabelle dargestellt.

In Wahrheitstabellen werden auf der linken Seite alle möglichen Wertekombinationen der Eingangssignale aufgelistet. Auf der rechten Seite findet sich der Boole'sche Wert der Schaltfunktion. Dabei werden die Zeilen der Wahrheitstabelle so angeordnet, dass sie aufsteigenden Binärzahlen (vgl. Kap. 5) entsprechen.

3.3.1 Die NICHT-Schaltfunktion (Inversion)

Die NICHT-Schaltfunktion, häufig auch als Inversion, Komplement oder Negation bezeichnet, kann man sich als öffnenden Schalter vorstellen (vgl. Bild 3.1). Die Betätigung des Schalters (Eingangssignal A = 1) ergibt den Wert Y = 0 am Ausgang. Die dazu gehörige Wahrheitstabelle ist neben dem Schaltsymbol in Bild 3.2 dargestellt.



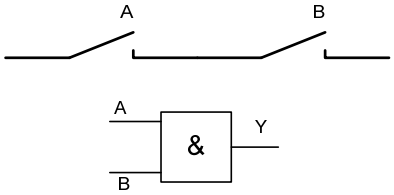
A	Y
0	1
1	0

Bild 3.2: Schaltsymbol und Wahrheitstabelle eines Inverters

Das in den logischen Gleichungen für die Inversion verwendete Symbol ist „¬“ bzw. ein Querstrich über dem Eingangssignal (Inversionsstrich). Also gilt $Y = \neg A = \overline{A}$ (Sprechweise: *nicht A*). Insbesondere gilt natürlich auch, dass eine doppelte Inversion wieder das Ausgangssignal ergibt: $A = \overline{\overline{A}}$. Wenn ein kompletter Logikausdruck zu invertieren ist, so muss der Inversionsstrich über den gesamten Ausdruck gezeichnet werden.

3.3.2 Die UND-Schaltfunktion (Konjunktion)

Die Schaltfunktion UND für zwei Eingänge A und B ist in Bild 3.3 als Reihenschaltung zweier Schließer dargestellt.



B	A	Y
0	0	0
0	1	0
1	0	0
1	1	1

Bild 3.3: Schaltfunktion, Schaltsymbol und Wahrheitstabelle eines UND-Gatters mit zwei Eingängen A und B

Die Wahrheitstabelle zeigt, dass das Ausgangssignal nur dann 1 wird, wenn beide Eingangssignale 1 sind, die Schalter also beide geschlossen sind.

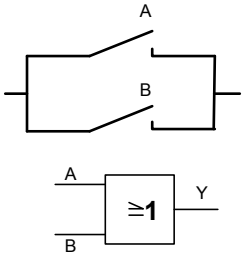
Das in der Digitaltechnik verwendete Symbol ist „∧“. Die Wahrheitstabelle beschreibt also $Y = A \wedge B$ (Sprechweise: *A und B*).

Wegen einer gewissen algebraischen Ähnlichkeit der UND-Verknüpfung mit der Multiplikation werden derartige Ausdrücke auch als logisches Produkt bezeichnet. Insbesondere in der anglo-amerikanischen Literatur wird daher auch der „*"-Operator verwendet bzw. dieser auch einfach weggelassen: $A \wedge B = A * B = AB$.

3.3.3 Die ODER-Schaltfunktion (Disjunktion)

Die Funktion ODER mit den beiden Eingängen A und B wird in der Schaltalgebra als Parallelschaltung zweier Schließer dargestellt (vgl. Bild 3.4).

Das in der Digitaltechnik verwendete Symbol ist „$\vee$". Die Wahrheitstabelle beschreibt also $Y = A \vee B$ (Sprechweise: *A oder B*). Hier existiert eine algebraische Ähnlichkeit zur Addition, sodass eine ODER-Verknüpfung auch als logische Summe bezeichnet wird und die anglo-amerikanische Literatur häufig den Operator „+" verwendet: $A \vee B = A + B$.



B	A	Y
0	0	0
0	1	1
1	0	1
1	1	1

Bild 3.4: Schaltfunktion, Schaltsymbol und Wahrheitstabelle eines ODER-Gatters mit zwei Eingängen A und B

3.3.4 Boole'sche Funktionen mit mehreren Eingängen

Prinzipiell lassen sich die Boole'schen Schaltfunktionen auch mit mehreren Eingängen darstellen. Die Wahrheitstabelle einer Funktion mit n Eingangssignalen besitzt dann 2^n Zeilen. Charakteristisch für die UND-Verknüpfung ist in jedem Fall, dass die Ergebnisspalte nur eine einzige 1 enthält und zwar dort, wo alle Eingangssignale den Wert 1 haben. Die Wahrheitstabelle des UND-Gatters besitzt also eine minimale Anzahl von Einsen. Aus diesem Grund wird das Ergebnis der UND-Verknüpfung auch als Minterm bezeichnet.

Ein UND-verknüpfter Ausdruck von Signalen wird als Minterm bezeichnet. Häufig verwendet man auch den Begriff „logisches Produkt" oder Produktterm.

Die Wahrheitstabelle eines ODER-Gatters mit mehreren Eingängen besitzt genau an einer Stelle eine 0 und zwar dort, wo alle Eingangssignale 0 sind. Damit besitzt die Wahrheitstabelle die maximale Anzahl möglicher Einsen. Entsprechend wird ein ODER-Ausdruck häufig auch als Maxterm bezeichnet.

Ein ODER-verknüpfter Ausdruck von Signalen wird als Maxterm bezeichnet. Häufig verwendet man auch den Begriff „logische Summe“ oder Summenterm.

Damit können nun die aus dem anglo-amerikanischen Sprachgebrauch stammenden Begriffe „sum of products“ und „product of sums“ definiert werden:

Als Summe von Produkten (engl.: sum of products, SOP) werden Terme mit UND-Verknüpfungen bezeichnet, die untereinander ODER-verknüpft sind.

Als product of sums (POS) werden ODER-verknüpfte Ausdrücke bezeichnet, die untereinander UND-verknüpft sind.

B Beispiel 3.1: Darstellung logischer Ausdrücke in SOP- bzw. POS-Darstellung

SOP: $Y = (A \wedge B) \vee (C \wedge D \wedge E) \vee F$
POS: $Y = (A \vee B \vee C \vee D) \wedge (E \vee F) \wedge G$ ■

3.4 Rechenregeln der Schaltalgebra

Für die Boole’schen Operatoren UND, ODER und NICHT gelten Rechenregeln, die z.B. mit Hilfe von Wahrheitstabellen bewiesen werden können.

3.4.1 Theoreme

Die nachfolgenden Theoreme lassen sich an Hand der Schaltermodelle leicht überprüfen:

- Die Verknüpfung eines Signals A mit den Konstanten 0 und 1 ist in Bild 3.5 dargestellt.

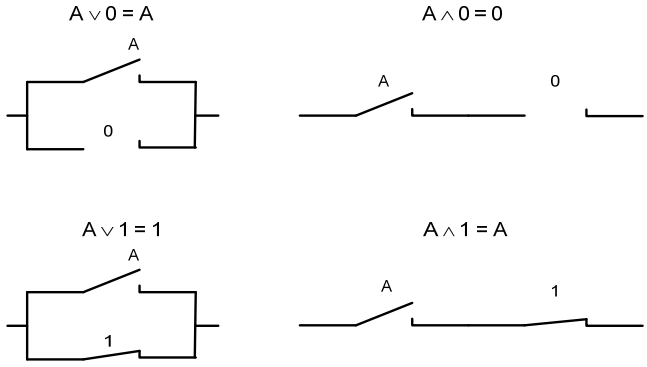


Bild 3.5: Darstellung der Null- und Eins-Theoreme mit Schaltfunktionen

- Die Idempotenz (vgl. Bild 3.6) beschreibt die logische Verknüpfung eines Signals mit sich selbst.

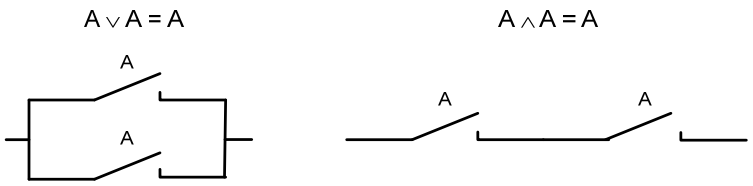


Bild 3.6: Schaltermodell der Idempotenz

- Die UND- bzw. ODER-Verknüpfung eines Signals mit seinem Komplement zeigt Bild 3.7.

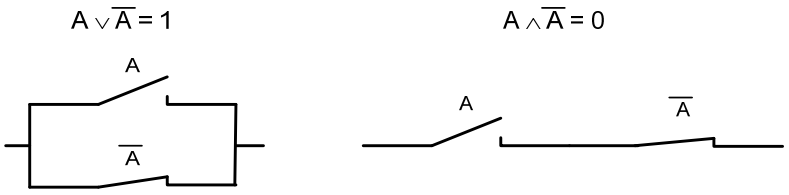


Bild 3.7: UND- und ODER-Verknüpfung eines Signals mit seinem Komplement

3.4.2 Kommutativgesetze

Die Kommutativgesetze regeln die Vertauschbarkeit der Operanden bei der UND- bzw. bei der ODER-Verknüpfung.

$$A \wedge B = B \wedge A \quad \text{und} \quad A \vee B = B \vee A \tag{3.1}$$

Die Operanden der UND-Verknüpfung lassen sich tauschen. Ebenso lassen sich die Operanden der ODER-Verknüpfung tauschen.

3.4.3 Assoziativgesetze

Die Assoziativgesetze regeln die Reihenfolge der Ausführung ausschließlicher UND- bzw. ODER-Operationen.

$$(A \wedge B) \wedge C = A \wedge (B \wedge C) = A \wedge B \wedge C \quad \text{und}$$
$$(A \vee B) \vee C = A \vee (B \vee C) = A \vee B \vee C \tag{3.2}$$

Bei einer reinen UND-Verknüpfung mehrerer Operanden ist die Reihenfolge der paarweisen Ausführung der UND-Operation egal. Klammern können auch weggelassen werden. Dieses gilt auch bei einer reinen ODER-Verknüpfung mehrerer Operanden.

3.4.4 Distributivgesetze

Die Distributivgesetze beschreiben das „Ausklammern“ eines gemeinsamen Operanden (hier des Signals A) bei gemischten UND- und ODER-Verknüpfungen.

$$(A \wedge B) \vee (A \wedge C) = A \wedge (B \vee C) \quad \text{und} \quad (A \vee B) \wedge (A \vee C) = A \vee (B \wedge C) \tag{3.3}$$

In UND-verknüpften Termen, die miteinander ODER-verknüpft sind, und die einen gemeinsamen Operanden besitzen, lässt sich dieser UND-verknüpft vor die Klammer ziehen, in der sich ein ODER-verknüpfter Ausdruck befindet.

In ODER-verknüpften Termen, die miteinander UND-verknüpft sind, und die einen gemeinsamen Operanden besitzen, lässt sich dieser ODER-verknüpft vor die Klammer ziehen, in der sich ein UND-verknüpfter Ausdruck befindet.

A Aufgabe 3.1

Zeigen Sie die Gültigkeit des Distributivgesetzes $(A \wedge B) \vee (A \wedge C) = A \wedge (B \vee C)$ durch Ausfüllen der Wahrheitstabelle und Vergleich der letzten beiden Spalten.

C	B	A	$A \wedge B$	$A \wedge C$	$B \vee C$	$(A \wedge B) \vee (A \wedge C)$	$A \wedge (B \vee C)$
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

■

3.4.5 De Morgan'sche Gesetze

Die de Morgan'schen Gesetze beschreiben, wie eine UND- bzw. eine ODER-Verknüpfung umgeformt werden kann, wenn der ganze Ausdruck invertiert wird.

$$\neg (A \wedge B \wedge C \wedge \ldots \wedge N) = \overline{A} \vee \overline{B} \vee \overline{C} \vee \ldots \vee \overline{N} \tag{3.4}$$

$$\neg (A \vee B \vee C \vee \ldots \vee N) = \overline{A} \wedge \overline{B} \wedge \overline{C} \wedge \ldots \wedge \overline{N} \tag{3.5}$$

Wenn ein ausschließlich UND-verknüpfter Ausdruck mit beliebig vielen Eingangssignalen invertiert wird, so entspricht dies einer ODER-Verknüpfung der invertierten Eingangssignale.

Wenn ein ausschließlich ODER-verknüpfter Ausdruck mit beliebig vielen Eingangssignalen invertiert wird, so entspricht dies einer UND-Verknüpfung der invertierten Eingangssignale.

3.4.6 Vereinfachungsregeln

Zur Vereinfachung logischer Ausdrücke können die nachfolgenden Kürzungsregeln verwendet werden, die sich z.B. über Wahrheitstabellen herleiten lassen:

- Die Adsorptionsgesetze:

$$A \wedge (\overline{A} \vee B) = A \wedge B \quad \text{und} \quad A \vee (\overline{A} \wedge B) = A \vee B \tag{3.6}$$

- Die Absorptionsgesetze:

$$A \wedge (A \vee B) = A \quad \text{und} \quad A \vee (A \wedge B) = A \tag{3.7}$$

- Die Nachbarschaftsgesetze:

$$(A \wedge B) \vee (\overline{A} \wedge B) = B \quad \text{und} \quad (A \vee B) \wedge (\overline{A} \vee B) = B \tag{3.8}$$

Insbesondere die Nachbarschaftsgesetze stellen ein sehr mächtiges Instrument zur Vereinfachung logischer Ausdrücke dar, welches wir in Kap. 6 dazu verwenden werden, um eine Vereinfachungssystematik einzuführen (KV-Minimierung).

Die Nachbarschaftsgesetze besagen, dass ein Signal, welches in zwei Klammern in direkter und in invertierter Form mit dem gleichen Operanden (hier B) und dem gleichen Operator verknüpft wird, weggelassen werden kann.

In den nachfolgenden Beispielen werden die Boole'schen Gesetze angewendet, um einige der Vereinfachungsregeln zu beweisen.