

Studienskript

DIGITAL-UND INFORMATIONSTECHNIK

DLBAETDIT01

iu

INTERNATIONALE
HOCHSCHULE

DIGITAL- UND

INFORMATIONSTECHNIK

IMPRESSUM

Herausgeber:
IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

Postanschrift:
Albert-Proeller-Straße 15-19
D-86675 Buchdorf
media@iu.org
www.iu.de

DLBAETDIT01
Versionsnr.: 001-2023-1020
N.N.

© 2023 IU Internationale Hochschule GmbH
Dieses Lernskript ist urheberrechtlich geschützt. Alle Rechte vorbehalten.
Dieses Lernskript darf in jeglicher Form ohne vorherige schriftliche Genehmigung der
IU Internationale Hochschule GmbH (im Folgenden „IU“) nicht reproduziert und/oder
unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet wer-
den.

Die Autor:innen/Herausgeber:innen haben sich nach bestem Wissen und Gewissen
bemüht, die Urheber:innen und Quellen der verwendeten Abbildungen zu bestimmen.
Sollte es dennoch zu irrtümlichen Angaben gekommen sein, bitten wir um eine dement-
sprechende Nachricht.

INHALTSVERZEICHNIS

DIGITAL- UND INFORMATIONSTECHNIK

Einleitung

Wegweiser durch das Studienskript	6
Basisliteratur	7
Weiterführende Literatur	8
Übergeordnete Lernziele	10

Lektion 1

Mathematische Grundlagen der Digitaltechnik	11
1.1 Überblick über digitale Systeme	12
1.2 Zahlensysteme und ihre Verhältnisse	14
1.3 Grundrechenarten in Zahlensystemen	20
1.4 Digitalcodes und ihre Anwendungen	27

Lektion 2

Darstellung, Synthese und Analyse Boolescher Funktionen	35
2.1 Boolesche Algebra und Funktionen	36
2.2 Disjunktive und konjunktive Normalform	43
2.3 Karnaugh-Veitch-Diagramm	46
2.4 Substitution durch NOR-/NAND-Gatter	53

Lektion 3

Schaltnetze (Kombinatorische Logik)	57
3.1 Logikgatter	58
3.2 Verknüpfung von Gattern	64
3.3 Decoder, Encoder und Multiplexer	72

Lektion 4

Rechenschaltungen	83
4.1 Addierschaltungen	84
4.2 Subtrahierschaltungen	88
4.3 Multiplikationsschaltungen	91

Lektion 5	
Schaltwerke (Sequenzielle Logik)	95
5.1 Grundlagen	96
5.2 Kippschaltungen: Das Latch	98
5.3 Kippschaltungen: Das Flip-Flop	103
5.4 Analyse getakteter Schaltwerke	108
5.5 Entwurf getakteter Schaltwerke	113
Lektion 6	
Register, Zähler und Automaten	119
6.1 Register	120
6.2 Zähler	126
6.3 Automaten	134
Lektion 7	
Einführung in programmierbare Logik	139
7.1 Programmierbare Logikbausteine	140
7.2 Complex Programmable Logic Devices (CPLD)	148
7.3 FPGA	150
7.4 Einführung in HDL	153
Verzeichnisse	
Literaturverzeichnis	160
Abbildungsverzeichnis	161

EINLEITUNG

HERZLICH WILLKOMMEN

WEGWEISER DURCH DAS STUDIENSKRIPT

Dieses Studienskript bildet die Grundlage Ihres Kurses. Ergänzend zum Studienskript stehen Ihnen weitere Medien aus unserer Online-Bibliothek sowie Videos zur Verfügung, mit deren Hilfe Sie sich Ihren individuellen Lern-Mix zusammenstellen können. Auf diese Weise können Sie sich den Stoff in Ihrem eigenen Tempo aneignen und dabei auf lernspezifische Anforderungen Rücksicht nehmen.

Die Inhalte sind nach didaktischen Kriterien in Lektionen aufgeteilt, wobei jede Lektion aus mehreren Lernzyklen besteht. Jeder Lernzyklus enthält jeweils nur einen neuen inhaltlichen Schwerpunkt. So können Sie neuen Lernstoff schnell und effektiv zu Ihrem bereits vorhandenen Wissen hinzufügen.

In der IU Learn App befinden sich am Ende eines jeden Lernzyklus die Interactive Quizzes. Mithilfe dieser Fragen können Sie eigenständig und ohne jeden Druck überprüfen, ob Sie die neuen Inhalte schon verinnerlicht haben.

Sobald Sie eine Lektion komplett bearbeitet haben, können Sie Ihr Wissen auf der Lernplattform unter Beweis stellen. Über automatisch auswertbare Fragen erhalten Sie ein direktes Feedback zu Ihren Lernfortschritten. Die Wissenskontrolle gilt als bestanden, wenn Sie mindestens 80 % der Fragen richtig beantwortet haben. Sollte das einmal nicht auf Anhieb klappen, können Sie die Tests beliebig oft wiederholen.

Wenn Sie die Wissenskontrolle für sämtliche Lektionen gemeistert haben, führen Sie bitte die abschließende Evaluierung des Kurses durch.

Die IU Internationale Hochschule ist bestrebt, in ihren Skripten eine gendersensible und inklusive Sprache zu verwenden. Wir möchten jedoch hervorheben, dass auch in den Skripten, in denen das generische Maskulinum verwendet wird, immer Frauen und Männer, Inter- und Trans-Personen gemeint sind sowie auch jene, die sich keinem Geschlecht zuordnen wollen oder können.

BASISLITERATUR

Fricke, K. (2018). *Digitaltechnik. Lehr- und Übungsbuch für Elektrotechniker und Informatiker*. (8. Aufl.), Springer Vieweg. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.37732&site=eds-live&scope=site>

Urbanski, K., Gehrke, W. & Weitowitz, R. (2012). *Digitaltechnik. Ein Lehr- und Übungsbuch*. Springer. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsbas&AN=edsbas.1DADFD0B&site=eds-live&scope=site>

WEITERFÜHRENDE LITERATUR

LEKTION 1

Bättig, R. (2020). *Angewandte Mathematik 1 mit MATLAB und Julia. Ein anwendungs- und beispielorientierter Einstieg für technische Studiengänge*, (1. Aufl., S. 1–31), Springer Vieweg. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edshbz&AN=edshbz.DE.605.HBZ01.036617929&site=eds-live&scope=site>

LEKTION 2

Ledin, J. (2021). *Architecting High-Performance Embedded Systems*. (S. 16–19). Packt Publishing. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsknv&AN=edsknv.kt012W1ME3&site=eds-live&scope=site>

LEKTION 3

Bernstein, H. (2019). *Formelsammlung. Elektrotechnik, Elektronik, Messtechnik, analoge und digitale Elektronik*, (2. Aufl., S. 309–310). Springer Vieweg. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edshbz&AN=edshbz.DE.605.HBZ01.022758067&site=eds-live&scope=site>

LEKTION 4

Uelzen, T. (2018). 5. Elementare Schaltnetze, In Ostfalia Hochschule für angewandte Wissenschaft (Hrsg.), *Digitaltechnik Grundlagen* (S. 1–26). (Im Internet verfügbar).

LEKTION 5

Bhunia, S. & Tehranipoor, M. (2019). *Hardware Security – A Hands-On Learning Approach*, (S. 29–31). <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edshbz&AN=edshbz.DE.605.HBZ01.037339706&site=eds-live&scope=site>

LEKTION 6

König, A. (2022). Digitaltechnik. In Technische Universität Chemnitz (Hrsg.), *Digitaltechnik. Sequentielle Schaltwerke* (Kap. 7). (Im Internet verfügbar).

LEKTION 7

Gessler, R. (2020). *Entwicklung Eingebetteter Systeme. Vergleich von Entwicklungsprozessen für FPGA- und Mikroprozessor-Systeme Entwurf auf Systemebene*, (2. Aufl., S. 253–270), Springer Vieweg. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edshbz&AN=edshbz.DE.605.HBZ01.036754996&site=eds-live&scope=site>

ÜBERGEORDNETE

LERNZIELE

In dem Kurs **Digital- und Informationstechnik** werden die Grundlagen der digitalen Logik und der Informationstechnik vorgestellt. Zu Beginn erfolgt neben der Theorie der Booleschen Algebra eine ausführliche Erklärung der notwendigen Mathematik. Danach liegt der Schwerpunkt auf dem Entwurf und der Analyse der wichtigsten Arten digitaler Systeme, wie der kombinatorischen und sequenziellen Logik. Abschließend enthält dieser Lehrbrief eine Einführung in praktische Realisierungstechniken digitaler Schaltungen, indem ein Überblick über Hardwarebeschreibungssprachen vermittelt wird.

Der Kurs dient als Grundlage für alle weiterführenden Kurse über digitale Elektronik und digitalen Systementwurf. Die Theorien und Techniken bilden die Basis, auf der weiterführende Analyse- und Entwurfsmethoden aufgebaut werden und gelten sowohl für einfache als auch für komplexe digitale Systeme.

LEKTION 1

MATHEMATISCHE GRUNDLAGEN DER DIGITALTECHNIK

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die Hauptmerkmale der Digitaltechnik zu benennen.
- verschiedene Zahlensysteme zu erkennen.
- die Konvertierung zwischen Zahlensystemen durchzuführen.
- diverse Codierverfahren zu benennen.

1. MATHEMATISCHE GRUNDLAGEN DER DIGITALTECHNIK

Einführung

Ein digitaler Computer speichert Daten in Form von Ziffern (Zahlen) und geht in diskreten Schritten von einem Zustand zum nächsten über. Die Zustände eines Digitalrechners bestehen in der Regel aus binären Ziffern, die sich in Form von magnetischen Markierungen auf einem Speichermedium, Ein- und Ausschaltern oder Relais äußern können. In Digitalrechnern werden sogar Buchstaben, Wörter und ganze Texte digital dargestellt. Es ist daher unerlässlich, die verschiedenen Zahlensysteme und die mathematischen Beziehungen zwischen ihnen zu verstehen, um sich mit digitalen Systemen vertraut zu machen.

1.1 Überblick über digitale Systeme

Das gegenwärtige technologische Zeitalter wird als „digitales Zeitalter“ bezeichnet, da die digitalen Technologien im täglichen Leben eine große Rolle spielen. Kommunikation, Wirtschaftstransaktionen, Verkehrsmanagement, Raumfahrtnavigation, medizinische Versorgung, Wetterüberwachung, das Internet und zahlreiche andere kommerzielle, industrielle und wissenschaftliche Unternehmungen sind auf **digitale Systeme** angewiesen. Neben digitalen Computern existieren auch digitale Telefone, digitale Fernsehgeräte, Digital Versatile Discs (DVDs), digitale Kameras und tragbare Gadgets. Musik wird gerne auf tragbaren Media Playern und anderen mobilen Geräten mit hochwertigen Bildschirmen gehört. Diese Geräte verfügen über grafische Benutzeroberflächen (GUIs), die es ihnen ermöglichen, Aufgaben auszuführen, die für die Benutzenden zwar einfach erscheinen mögen, in Wirklichkeit aber die sorgfältige Ausführung einer Reihe komplizierter interner Anweisungen erfordern.

Die meisten dieser Geräte – wenn auch nicht alle – verfügen über eingebaute digitale Spezialcomputer. Die Universalität des Digitalrechners ist sein bemerkenswertestes Merkmal. Er ist in der Lage, auf vorher festgelegte Daten zu reagieren, indem er eine Reihe von Anweisungen befolgt, die als Programm bezeichnet werden. Den Benutzenden ist es möglich, die Software oder die Daten an die eigenen Bedürfnisse anzupassen und zu verändern. Dank dieser Vielseitigkeit können Allzweck-Digitalcomputer eine Vielzahl von Informationsverarbeitungsaufgaben in einem breiten Spektrum von Anwendungen bewältigen. Die Fähigkeit digitaler Systeme, einzelne Informationen darzustellen und mit ihnen zu arbeiten, ist eines ihrer Merkmale.

Diskrete Informationen sind in jeder Menge mit einer begrenzten Anzahl von Ziffern vorhanden. Die 26 Buchstaben des Alphabets, die 52 Spielkarten, die zehn Dezimalziffern und die 64 Felder auf einem Schachbrett sind alles Beispiele für diskrete Mengen. Numerische Berechnungen wurden auf frühen digitalen Computern durchgeführt. Die Ziffern waren in diesem Fall die diskreten Elemente. Der Name „Digitalcomputer“ geht auf diese Verwen-

Das digitale System
Es verarbeitet diskrete Informationen, die als binäre Ziffern dargestellt sind.

zung zurück. In einem digitalen System werden diskrete Informationen durch physikalische Größen, sogenannte Signale, dargestellt. Die typischsten Arten von elektrischen Signalen sind Spannungen und Ströme. Die Schaltkreise, die diese Signale umsetzen, bestehen meist aus elektronischen Bauteilen, nämlich den Transistoren.

Die meisten modernen elektronischen Digitalsysteme verwenden binäre Signale, die lediglich zwei diskrete Werte enthalten. Die beiden Werte einer binären Ziffer oder eines Bits sind 0 und 1. Binärcodes sind Sammlungen von Bits, die zur Darstellung diskreter Informationen verwendet werden. Ein digitales System verwendet zum Beispiel einen Vier-Bit-Code, um die Dezimalzahlen 0 bis 9 darzustellen (z. B. wird die Zahl 7 durch 0111 dargestellt). Das Codesystem, in dem ein Muster von Bits existiert, bestimmt, wie dieses Muster in eine Zahl übersetzt wird.

Gruppen von Bits können durch eine Vielzahl von Methoden erzeugt werden, um bestimmte Symbole – nicht immer Zahlen – darzustellen, die dann verwendet werden, um das System in einem digitalen Format zu konstruieren. Ein System, das diskrete Informationen verarbeitet, die intern in binärer Form dargestellt werden, wird als digitales System bezeichnet. Binäre Systeme sind in der modernen Technologie am weitesten verbreitet, da sie, wie wir sehen werden, mit elektronischen Komponenten implementiert werden können.

Je nach Art der zu verarbeitenden Daten können diskrete Informationsmengen entstehen oder als Ergebnis eines kontinuierlichen Prozesses quantisiert werden. Ein Gerät, das eine analoge (kontinuierliche) Größe in eine digitale (diskrete) Darstellung umwandelt, wird als Analog-Digital-Wandler bezeichnet und kann häufig automatisch die Quantisierung eines Prozesses durchführen.

Das bekannteste Beispiel für ein digitales System ist der Allzweckcomputer. Die Hauptbestandteile eines Computers sind:

- **die Speichereinheit:** hier werden Programme sowie Eingabe-, Ausgabe- und Zwischendaten gespeichert,
- **die zentrale Proessoreinheit:** diese führt mathematische Berechnungen und andere Datenverarbeitungsaufgaben durch, die von der Software vorgegeben werden, und
- **die Eingabe-/Ausgabeeinheiten:** über ein Eingabegerät, z. B. eine Tastatur, werden vorbereitete Programme und Daten in den Speicher übertragen. Die Ergebnisse der Berechnungen werden an ein Ausgabegerät, z. B. einen Drucker, übermittelt, der sie ausdruckt und dem Benutzenden anzeigt. Mit einem Digitalcomputer können viele verschiedene Eingabe- und Ausgabegeräte verwendet werden.

Ein Digitalcomputer ist ein leistungsfähiges Werkzeug, das neben arithmetischen Berechnungen auch für die Durchführung logischer Prozesse einsetzbar ist. Er kann auch so konstruiert sein, dass er Entscheidungen in Abhängigkeit von internen und externen Faktoren trifft. Die meisten digitalen Geräte sind programmierfähig, ähnlich wie ein Computer. Dieselbe zugrundeliegende Hardware eines programmierbaren Geräts ist durch Änderung des Programms auf eine Vielzahl von Anwendungen übertragbar, wodurch die Entwicklungskosten auf einen größeren Kundenstamm verteilt werden können.

Aufgrund von Fortschritten in der digitalen integrierten Schaltungstechnik sind die Kosten für digitale Geräte drastisch gesunken. Digitale Geräte werden immer erschwinglicher, da die Zahl der Transistoren, die auf einem Stück Silizium untergebracht werden können, um komplizierte Operationen zu ermöglichen, steigt, und die Kosten pro Einheit sinken. Die Verarbeitungsgeschwindigkeit von Geräten, die mit digitalen integrierten Schaltkreisen hergestellt werden, kann Milliarden von Operationen pro Sekunde erreichen. Mithilfe von Fehlerkorrekturcodes lassen sich äußerst zuverlässige digitale Systeme schaffen. Ein Beispiel dafür ist die Digital Versatile Disc (DVD), worauf digitale Informationen in Form von Video-, Audio- und anderen Daten aufgezeichnet werden, ohne dass ein einziges Stück verloren geht. Etwaige Fehler in den digitalen Informationen auf einem digitalen Speichermedium können automatisch, mittels einer Prüfung der Codierung in jedem digitalen Sample, erkannt und behoben werden, bevor es erneut abgespielt wird.

1.2 Zahlensysteme und ihre Verhältnisse

Ein allgemeines Zahlensystem mit der Basis (auch Radix genannt) A ist ein Zahlensystem, in dem jede Zahl n nach Potenzen von A zerlegt wird. Der Wert einer ganzen positiven $(N+1)$ -stelligen Zahl beträgt:

$$n = \sum_{i=0}^N a_i A^i = a_N A^N + a_{N-1} A^{(N-1)} + \dots + a_2 A^2 + a_1 A + a_0 \quad (1.1)$$

wobei

a_i = Stellenwert der Stelle i ; $0 \leq a_i \leq A - 1$

A = Basis des Zahlensystems

A^i = Stellenfaktor der Stelle i

Um die Schreibweise zu vereinfachen wird folgende Form benutzt:

$$n = (b_N b_{N-1} b_{N-2} \dots b_2 b_1 b_0) \quad (1.2)$$

wo führende Nullen weggelassen sind.

BEISPIEL ZAHLENSYSTEME

Eine Dezimalzahl wie 4293 ist äquivalent zu 4 Tausender, plus 2 Hunderter, plus 9 Zehner, plus 3 Einer. Die Tausender, Hunderter usw. sind Potenzen von 10, die sich aus der Position der Koeffizienten (Ziffern) in der Zahl ergeben. Um genau zu sein, ist 7392 eine Kurzschreibweise für das, was geschrieben werden sollte als $4 \cdot 10^3 + 2 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0$. Es ist jedoch üblich, nur die numerischen Koeffizienten aufzuschreiben und aus ihrer Position die erforderlichen 10er-Potenzen abzuleiten, wobei die Potenzen von rechts nach links zunehmen. Weitere Beispiele sind:

- eine Basis-5-Zahl:

$$(4021,2)_5 = 4 \cdot 5^3 + 0 \cdot 5^2 + 2 \cdot 5^1 + 1 \cdot 5^0 + 2 \cdot 5^{-1} \\ = (511,4)_{10}$$

- eine Basis-8-Zahl (Oktalsystem):

$$(127,4)_8 = 1 \cdot 8^2 + 2 \cdot 8^1 + 7 \cdot 8^0 + 4 \cdot 8^{-1} = (87,5)_{10}$$

- eine Basis-16-Zahl (Hexadezimalsystem):

$$(B65F)_{16} = B \cdot 5^3 + 0 \cdot 5^2 + 2 \cdot 5^1 + 1 \cdot 5^0 + 2 \cdot 5^{-1} \\ = (511,4)_{10}$$

wobei Hexadezimalzahlen oft durch ein tiefgestelltes H gekennzeichnet sind,
z. B. B65F_H

Das Dualsystem, auch Zweiersystem oder Binärsystem genannt, mit der Basis 2 ist für Computersysteme von größter Bedeutung, da es die Grundlage für die digitale Logik bildet. In diesem System wird jede Zahl nur mit den Zeichen 0 und 1 dargestellt, z. B.

$$(101011)_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 \\ + 1 \cdot 2^0 = (43)_{10}$$

Das Dualsystem
Dies ist ein Zahlensystem mit der Basis 2.

Im Zusammenhang mit der Computertechnik werden die folgenden Ausdrücke zur Darstellung von Binärzahlen verwendet

- **das Bit:** steht für Binärziffer (engl. Binary-digit) und repräsentiert eine einzelne Binärziffer, entweder eine 0 oder eine 1,
- **das Wort** (engl. Word): ein Behälter, der aus vier Bits besteht, und
- **das Byte:** ein Behälter, der aus acht Bits oder zwei Wörtern besteht und die bei weitem gebräuchlichste Dateneinheit beim Rechnen ist.

Größere Datenmengen können auf unterschiedliche Weise ausgedrückt werden, z. B. können 16 Bits als zwei Bytes oder als 16-Bit-Wort beschrieben werden (nicht zu verwechseln mit einem 4-Bit-Wort).

Tabelle 1 zeigt die Zahlen von null bis 15 in den Dezimal-, Dual-, Oktal- und Hexadezimalsystemen.

Tabelle 1: Zahlen von 0–15 für Basis 10, 2, 8 und 16

Dezimal	Dual	Oktal	Hexadezimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Quelle: Moustafa Nawito, 2022.

Umrechnung zwischen Zahlensystemen: die Restmethode

Als Beispiel dient die Dezimalzahl 4321, die in die Dezimalschreibweise umgewandelt werden soll. Um die letzte Ziffer zu erhalten, verschieben Sie den Dezimalpunkt um eine Stelle nach links, d. h., Sie dividieren die gegebene Zahl durch ihre Basis 10.

$$4321/10 = 432 + 1/10$$

Der Rest von eins ist die letzte Ziffer. Um die vorletzte Ziffer zu ermitteln, verschieben Sie den Dezimalpunkt wieder um eine Stelle nach links und sehen, was herausfällt.

$$432/10 = 43 + 2/10$$

Der Rest von zwei ist die vorletzte Ziffer. Man wiederholt diesen Vorgang, bis nichts mehr übrig ist, danach ist die Umwandlung vollständig. Die Umwandlung ist in Abbildung 1 zusammengefasst.

Abbildung 1: Die Zahl 1781 im Dezimalsystem

4321/10 =	432	1	-----	+
432/10 =	43	2	-----	
43/10 =	4	3	-----	
4/10 =	0	4	--+	(Stoppen wenn Quotient 0 ist)
			4 3 2 1	(Basis 10)

Quelle: Moustafa Nawito, 2022.

Nun folgt ein nichttriviales Beispiel. Die Dezimalzahl 1781 soll in binärer Schreibweise ausgedrückt werden. Da die gewünschte Basis zwei ist, dividieren Sie die angegebene Dezimalzahl wiederholt durch zwei.

Abbildung 2: Umwandlung der Dezimalzahl 1781 zum Dualsystem

1781/2 =	890	1	-----	+
890/2 =	445	0	-----	+
445/2 =	222	1	-----	+
222/2 =	111	0	-----	+
111/2 =	55	1	-----	+
55/2 =	27	1	-----	+
27/2 =	13	1	-----	+
13/2 =	6	1	-----	+
6/2 =	3	0	-----	+
3/2 =	1	1	-----	+
1/2 =	0	1	--+	
				1 1 0 1 1 1 1 0 1 0 1

Quelle: Moustafa Nawito, 2022.

In der folgenden Abbildung ist die Darstellung derselben Dezimalzahl 1781 im Oktalsystem veranschaulicht.

Abbildung 3: Umwandlung der Dezimalzahl 1781 zum Oktalsystem

1781/8 =	222	5	-----	+
222/8 =	27	6	-----	+
27/8 =	3	3	-----	+
3/8 =	0	3	--+	
			3 3 6 5	

Quelle: Moustafa Nawito, 2022.

In der folgenden Abbildung ist die Darstellung derselben Dezimalzahl 1781 im Hexadezimalsystem aufgezeigt.

Umrechnung von einer beliebigen Basis in eine Dezimalzahl: die Polynommethode

Es ist bekannt, dass der Wert jeder Dezimalzahl durch Multiplikation der Zahlenkoeffizienten oder Ziffern mit der entsprechenden Potenz von 10 berechnet werden kann, also zum Beispiel $4567 = 4 \cdot 10^3 + 5 \cdot 10^2 + 6 \cdot 10^1 + 7 \cdot 10^0$, wobei sieben die Ziffer mit dem niedrigsten Wert ist (engl. Least Significant Digit – LSD) und vier die Ziffer mit dem höchsten Wert ist (engl. Most Significant Digit – MSD). So hat jede Ziffer einen Wert: $10^0 = 1$ für die LSD, ansteigend auf $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$ usw.

Ebenso hat die niederwertigste Ziffer einer Hexadezimalzahl einen Wert von $16^0 = 1$ für die niederwertigste Ziffer, ansteigend auf $16^1 = 16$ für die nächste Ziffer, $16^2 = 256$ für die nächste, $16^3 = 4096$ für die nächste und so weiter. 2345_{16} bedeutet also, dass es vier Ziffern gibt; und es gibt fünf Einsen in der äußersten rechten Ziffer, vier Sechzehner in der nächsten Ziffer, drei 256ern im nächsten und zwei 4096er in der äußersten linken Ziffer (MSD). Die Gesamtsumme ist:

$$2 \cdot 4096 + 3 \cdot 256 + 4 \cdot 16 + 5 \cdot 1 = 9029_{10}$$

Zusammenfassend lässt sich die Umrechnung von einer beliebigen Basis in die Basis 10 mit der folgenden Formel bewerkstelligen.

$$x_{10} = \sum_{i=-m}^{n-1} d_i b^i \quad (1.3)$$

Dabei ist b die Basis, d_i die Ziffer an der Position i , m die Anzahl der Nachkommastellen, n die Anzahl der Ziffern des ganzzahligen Teils und X_{10} die erhaltene Zahl in Dezimalzahlen. Dies ist die Grundlage der Polynom-Methode für die Umwandlung von Zahlen von einer beliebigen Basis in Dezimalzahlen.

KONVERTIEREN VON OKTALEN ZAHLEN IN DEZIMALZAHLEN

Konvertieren Sie $432,14_8$ in Dezimalschreibweise.

Lösung:

$$\begin{aligned} 4 \cdot 8^2 + 3 \cdot 8^1 + 2 \cdot 8^0 + 1 \cdot 8^{-1} + 4 \cdot 8^{-2} &= 4 \cdot 64 + 3 \cdot 8 \\ + 2 \cdot 1 + 1/8 + 4/64 &= 282,1875_{10} \end{aligned}$$

Umrechnung zwischen Binär-Oktal und Binär-Hexadezimalzahlen

Wie Tabelle in 1 gezeigt, gibt es eine direkte Beziehung zwischen dem Binärsystem und dem Oktalsystem, nämlich dass drei binäre Ziffern einer oktalen Ziffer entsprechen. Ebenso lassen sich vier binäre Ziffern direkt in eine hexadezimale Ziffer umwandeln.

Um eine Binärzahl in eine Oktalzahl umzuwandeln, zerteilen wir die Basis-2-Zahl in Dreiergruppen, beginnend mit dem Radixpunkt (Dezimalzeichen), und füllen die äußersten Gruppen nach Bedarf mit Nullen auf, um Dreiergruppen oder Tripel zu bilden. Dann wird jedes Tripel in das oktale Äquivalent umgewandelt. Für die Umrechnung von der Basis zwei in die Basis 16 werden Vierergruppen verwendet. Nehmen wir die Konvertierung von 11001_2 zur Basis 8: $11001_2 = 011_2 \ 001_2 = 3_8 \ 1_8 = 31_8$. Hier muss man beachten, dass bei jedem Tripel die beiden Bits ganz links mit einer Null aufgefüllt werden, um ein vollständiges Tripel zu erhalten.

Betrachten wir nun die Umrechnung von 10110110_2 zur Basis 16: $10110110_2 = 1011_2 \ 0110_2 = B_{16} \ 6_{16} = B6_{16}$ (Beachten Sie, dass „B“ eine Ziffer zur Basis 16 ist, die 1011 entspricht. B ist keine Variable.)

Die Konvertierungsmethoden können verwendet werden, um eine Zahl von einer beliebigen Basis in eine andere Basis zu konvertieren, aber es ist vielleicht nicht sehr intuitiv, etwas wie $894,03_H$ in die Basis 9 zu konvertieren. Als Hilfe bei der Durchführung einer unnatürlichen Konvertierung können wir in einem Zwischenschritt in die vertrautere Form zur Basis 10 konvertieren und dann die Konvertierung von der Basis 10 in die Zielbasis fortsetzen. In der Regel wird bei der Umrechnung in die Basis 10 die Polynommethode und bei der Umrechnung aus der Basis 10 die Rest- und Multiplikationsmethode verwendet.

1.3 Grundrechenarten in Zahlensystemen

Numerische Komplemente

Komplemente werden von digitalen Computern verwendet, um die Logik zu verändern und den Subtraktionsprozess zu vereinfachen. Durch die Verwendung einfacherer Schaltungen zur Erfüllung der Aufgaben können Prozesse erleichtert werden. Für jedes Basis- r -System gibt es zwei verschiedene Arten von Komplementen: das Basiskomplement und das verminderte Basiskomplement, wobei das zweite als $(r - 1)$ -Komplement bezeichnet wird, ist das erste als r -Komplement bekannt. Wenn der Wert der Basis r im Namen ersetzt wird, werden die beiden Arten als 2er-Komplement und 1er-Komplement für Binärzahlen bzw. als 10er-Komplement und 9er-Komplement für Dezimalzahlen bezeichnet.

Vermindertes Basiskomplement

Bei einer Zahl N zur Basis r mit n Ziffern ist das $(r - 1)$ -Komplement von N , d. h. das verminderte Basiskomplement, definiert als $(r^n - 1) - N$. Für Dezimalzahlen gilt, dass $r = 10$ und $r - 1 = 9$, also ist das 9er-Komplement von einer Zahl N definiert als $(10^n - 1) - N$.

In diesem Fall ist 10^n eine Zahl, die aus einer einzigen 1 gefolgt von n Nullen besteht und $10^n - 1$ ist eine Zahl, die durch n Neuner dargestellt wird. Zum Beispiel, für $n = 5$, haben wir $10^5 = 100000$ und $10^5 - 1 = 99999$. Daraus folgt, dass das 9er-Komplement von einer Dezimalzahl durch Subtraktion jeder Ziffer von 9 erhalten wird, z. B. das 9er-Komplement von 345600 ist $999999 - 345600 = 654399$.

Bei Binärzahlen ist $r = 2$ und $r - 1 = 1$, also ist das 1er-Komplement von N $(2^n - 1) - N$. Auch hier wird 2^n durch eine Binärzahl repräsentiert, die aus einer 1 gefolgt von n Nullen besteht. $2^n - 1$ ist eine Binärzahl, die aus n Einsen besteht. Zum Beispiel, für $n = 5$, haben wir $2^5 = (100000)_2$ und $2^5 - 1 = (11111)_2$. Das 1er-Komplement von einer Binärzahl erhält man also durch Subtraktion jeder Ziffer von 1. Wenn man jedoch die Binärziffern von 1 subtrahiert, kann man entweder $1 - 0 = 1$ oder $1 - 1 = 0$ haben, was dazu führt, dass alle Bits der Zahl von 0 auf 1 bzw. 1 von 1 auf 0 ändern. Man kann also sagen, dass das 1er-Komplement einer Binärzahl gebildet wird, indem Einsen in Nullen und Nullen in Einsen umgewandelt werden, zum Beispiel:

- das 1er-Komplement von 0100111 ist 1011000 und
- das 1er-Komplement von 1010010 ist 0101101.

Das $(r - 1)$ -Komplement von Oktal- oder Hexadezimalzahlen erhält man durch Subtraktion jeder Ziffer 1 von 7 bzw. F (dezimal 15).

Basiskomplement

Bei einer Zahl N zur Basis r mit n Ziffern ist das r -Komplement definiert als $r^n - N$ für $N \neq 0$ und als 0 für $N = 0$. Im Vergleich mit dem $(r - 1)$ -Komplement stellen wir fest, dass aus dem r -Komplement durch Addition von 1 das $(r - 1)$ -Komplement entsteht, da $r^n - N = [(r^n - 1) - N] + 1$.

So ist das 10er-Komplement der Zahl $1234\ 8765 + 1 = 8766$ und wird durch Addition von 1 zum 9er-Komplementwert. Das 2er-Komplement der Binärzahl 010011 ist $101100 + 1 = 101101$ und ergibt sich aus der Addition von 1 zum 1er-Komplement-Wert.

Eine andere Methode zur Berechnung des 2er-Komplements besteht, indem man alle Nullen mit dem niedrigsten Wert (die LSDs) und die erste 1 unverändert lässt und in allen anderen höherwertigen Ziffern die Einsen durch Nullen und die Nullen durch Einsen ersetzt, zum Beispiel

- das 2er-Komplement von 1101100 ist 0010100 und
- das 2er-Komplement von 0110111 ist 1001001.

In den vorherigen Definitionen wurde davon ausgegangen, dass die Zahlen keinen Radixpunkt haben. Falls die ursprüngliche Zahl N einen Radixpunkt enthält, dann soll dieser vorübergehend entfernt werden, um das r - oder $(r - 1)$ -Komplement zu formen. Der Radixpunkt wird dann in der komplementierten Zahl an der gleichen relativen Position wiederhergestellt. Es ist auch erwähnenswert, dass das Komplement des Komplements die Zahl auf ihren ursprünglichen Wert

zurücksetzt. Zur Veranschaulichung dieser Beziehung ist zu beachten, dass das r -Komplement von N gleich $r^n - N$ ist, sodass das Komplement des Komplements $r^n - (r^n - N) = N$ ist und der ursprünglichen Zahl entspricht.

Subtraktion mit Komplementen

Bei der üblichen Subtraktion nehmen wir eine eins von einer höherwertigen Stelle, wenn die Minuendziffer kleiner ist als die Subtrahendziffer. Diese Methode funktioniert gut, wenn man die Subtraktion schriftlich (per Hand) durchführt. Insofern die Subtraktion jedoch mit digitaler Hardware durchgeführt wird, ist dieser Weg weniger effizient als die Methode, die Komplemente verwendet.

Die folgenden Schritte können angewandt werden, um zwei n -stellige vorzeichenlose Werte M von N in der Basis r zu subtrahieren:

1. Das Komplement des Subtrahenden N wird um den Minuenden M erhöht wie folgt:

$$M + (r^n - N) = M - N + r^n$$

2. Wenn $M \geq N$ ist, ergibt die Summe einen Endübertrag r^n , der ignoriert werden kann; das Ergebnis ist $M - N$.
3. Wenn $M < N$ ist, ergibt die Summe $r^n - (N - M)$, das ist das r -Komplement von $(N - M)$ und führt nicht zu einem Endübertrag. Man nimmt das r -Komplement der Summe und fügt ein negatives Vorzeichen hinzu, um die Antwort in der bekannten Form zu erhalten.

BEISPIEL BINÄRE SUBTRAKTION

Für die zwei Binärzahlen $M = 1101011$ und $N = 1011010$ berechnen Sie

- a) $M - N$ und
- b) $N - M$

Lösung

- a)

M =	1101011
Zweierkomplement von N =	+0100110
Summe =	10010001
Endübertrag 2^7 ignorieren	-10000000

	Endergebnis M - N =	10001
b)		
	N =	1011010
	Zweierkomplement von M =	+0010101
	Summe =	11101111
	Zweierkomplement =	00010001
	Endergebnis N - M =	- 10001

Vorzeichenbehaftete Binärzahlen

Vorzeichenlose Zahlen können zur Darstellung positiver ganzer Zahlen, einschließlich 0, verwendet werden. Wir benötigen jedoch eine Notation für negative Werte, um negative ganze Zahlen darzustellen. In der Standardmathematik wird eine negative Zahl durch ein Minuszeichen und eine positive Zahl durch ein Pluszeichen dargestellt. Computer können aufgrund von Hardwarebeschränkungen nur binäre Ziffern speichern. Üblicherweise wird das Vorzeichen durch Hinzufügen eines Bits zur äußersten linken Ziffer der Zahl angegeben. So sollte das Vorzeichenbit auf 0 für positive und 1 für negative Zahlen gesetzt werden.

Es ist wichtig zu verstehen, dass sowohl vorzeichenbehaftete als auch nicht vorzeichenbehaftete binäre Ganzzahlen bei der Darstellung in einem Computer aus einer Reihe von Bits bestehen. Ob eine Zahl vorzeichenbehaftet ist oder nicht, hängt vom Benutzenden ab. Im Falle einer vorzeichenbehafteten Binärzahl steht das Bit ganz links für das Vorzeichen, während die übrigen Bits den Wert bezeichnen. Das ganz linke Bit der Binärzahl ist das MSD, wenn die Binärzahl vorzeichenlos sein soll.

Zum Beispiel ist der Wert von z. B. 0101 immer 5 bzw. +5. Bei z. B. 1101 ist der Wert entweder -7 oder 13, abhängig von der Bedeutung des ganz linken Bits. Diese Methode oder Konvention zur Darstellung vorzeichenbehafteter Binärzahlen ist als Konvention für vorzeichenbehaftete Beträge bezeichnet. In diesem Format wird das Vorzeichen der Zahl durch ein Symbol (+ oder -) oder durch ein Bit (0 oder 1) angegeben. So sind vorzeichenbehaftete Zahlen in der Alltagsmathematik dargestellt.

Bei der Durchführung von Rechenoperationen auf einem Computer ist es praktischer, negative Zahlen mit einer separaten Methode, dem sogenannten Vorzeichen-Komplement-System, darzustellen. Eine negative Zahl wird in diesem System durch ihr Komplement dargestellt. Das vorzeichenbehaftete Komplementsystem negiert eine Zahl, indem es ihr Komplement nimmt, im Gegensatz zum vorzeichenbehafteten Betragssystem, das das Vorzeichen einer Zahl ändert. Das Komplement beginnt immer mit einer 1, was eine negative Zahl bedeutet, denn Komplemente positiver Zahlen beginnen immer mit einer 0 (plus) an der äußersten linken Position. Im vorzeichenbehafteten Komplementsystem kann das 1er- oder das 2er-Komplement verwendet werden, wobei das 2er-Komplement häufiger angewandt wird.

Als Beispiel dient die binäre Darstellung der Zahl 9, die acht Bits hat. 00001001 ist die Binärdarstellung von +9, die durch ein Vorzeichenbit von 0 an der äußersten linken Position, gefolgt vom Binärwert von 9, dargestellt wird. Da jedes der acht Bits einen Wert haben muss, werden nach dem Vorzeichenbit Nullen bis zur ersten 1 angehängt. Es gibt drei Möglichkeiten, -9 mit acht Bits auszudrücken, während es nur eine Möglichkeit gibt, +9 darzustellen:

- Vorzeichendarstellung des Betrags: 10001001,
- Darstellung eines vorzeichenbehafteten Einerkomplements: 11110110 und
- Darstellung eines vorzeichenbehafteten Zweierkomplements: 11110111.

Tabelle 2 zeigt wie vorzeichenbehaftete Binärzahlen im Bereich -8 bis +7 mit verschiedenen Methoden dargestellt werden können.

Tabelle 2: Vorzeichenbehaftete Binärzahlen

Dezimalzahl	vorzeichenbehaftetes 2er-Komplement	vorzeichenbehaftetes 1er-Komplement	vorzeichenbehafteter Betrag
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	—	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011

Dezimalzahl	vorzeichenbehaftetes 2er-Komplement	vorzeichenbehaftetes 1er-Komplement	vorzeichenbehafteter Betrag
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	—	—

Quelle: Moustafa Nawito, 2022.

Addition

Im vorzeichenbehafteten Betragssystem erfolgt die Addition zweier Zahlen nach den üblichen arithmetischen Verfahren. Wenn die Vorzeichen identisch sind, addiert man die beiden Größen und weist der Summe das gemeinsame Vorzeichen zu. Unterscheiden sich die Vorzeichen, wird der kleinere Betrag vom größeren subtrahiert und das Ergebnis erhält das Vorzeichen des größeren Betrags. Um zum Beispiel $(+35) + (-47) = -(47 - 35) = -12$ zu erhalten, muss man zunächst den größeren Betrag, 47, vom kleineren Betrag, 35, subtrahieren und dann das Vorzeichen von 47 zum Ergebnis hinzufügen. Bei dieser Methode werden die Beträge und Vorzeichen verglichen und anschließend entweder addiert oder subtrahiert. Binäre Zahlen, die durch vorzeichenbehaftete Beträge dargestellt werden, folgen demselben Verfahren. Im Gegensatz dazu erfordert die Regel des vorzeichenbehafteten Komplementsystems für die Addition ganzer Zahlen nur eine Addition und keinen Vergleich oder eine Subtraktion. Bei Binärzahlen ist das Verfahren recht einfach und kann wie folgt beschrieben werden: Die Kombination der beiden ganzen Zahlen, einschließlich ihrer Vorzeichenbits, ergibt die Addition zweier vorzeichenbehafteter Binärzahlen, wobei negative Zahlen in vorzeichenbehafteter 2er-Komplementform dargestellt werden. Die Vorzeichen-Bit-Position wird dabei nicht berücksichtigt. Hier sind einige numerische Beispiele für die Addition:

$$\begin{array}{r}
 + 7 \ 00000111 \quad - 7 \ 1111100 \\
 + 12 \ 00001100 \quad + 12 \ 00001100 \\
 = + 19 \ 00010011 = + 5 \ 00000101 \\
 + 7 \ 00000111 \quad - 7 \ 11111001 \\
 - 12 \ 11110100 \quad - 12 \ 11110100 \\
 = - 5 \ 11111011 = - 19 \ 11101101
 \end{array}$$

Wenn die nach der Addition entstandene Summe negativ ist, liegt sie im 2er-Komplement vor. Negative Zahlen müssen zunächst in 2er-Komplementform ausgedrückt werden. Zum Beispiel wird das 2er-Komplement von +7, also -7, als 11111001 dargestellt. Die in jedem der vier Szenarien durchgeführte Operation ist die Addition mit dem Vorzeichenbit. Negative Ergebnisse liegen immer in Form des 2er-Komplements vor und jeder Übertrag aus der Vorzeichenbitposition wird automatisch eliminiert. Wir müssen sicherstellen, dass das Ergebnis genügend Bits hat, um die Summe zu halten, zur Gewährleistung der korrekten Antwort. Wenn die Summe von zwei n -Bit-Werten mehr als $n + 1$ Bits belegt, liegt ein

Überlauf vor. Da wir beim Addieren mit Papier und Bleistift nicht durch die Breite der Seite eingeschränkt sind, gibt es kein Problem mit einem Überlauf. Um eine Zahl $n + 1$ Bits lang zu machen, fügen wir einfach eine weitere 0 zu einer positiven Zahl oder eine weitere 1 zu einer negativen Zahl an der höchstwertigen Stelle hinzu, bevor wir die Addition durchführen. Da es in einem Computer nur eine endliche Anzahl von Bits gibt, kommt es zu einem Überlauf, wenn das Ergebnis um ein Bit größer ist als die Anzahl der verfügbaren Bits. Diejenigen, die an das System der vorzeichenbehafteten Größen gewöhnt sind, sind mit der Komplementdarstellung negativer ganzer Zahlen nicht vertraut. Es ist erforderlich, eine negative Zahl in eine positive Zahl umzuwandeln, um sie verständlicher darzustellen, bevor ihr Wert im vorzeichenbehafteten 2er-Komplement berechnet wird. Da zum Beispiel das ganz linke Bit 1 ist, ist die vorzeichenbehaftete Binärzahl 11111001 negativ. Die binäre Darstellung von +7 ist 0000111, das ist das 2er-Komplement. Daraus ergibt sich, dass die anfängliche negative ganze Zahl gleich -7 ist.

Multiplikation und Division

Bei einer binären Multiplikation von vorzeichenlosen Zahlen wird der Multiplikator mit jeder einzelnen Ziffer des Multiplikanden multipliziert, um die Multiplikation durchzuführen. Das Ergebnis dieser ziffernweisen Multiplikation ist null oder gleich dem Multiplikator, da die Ziffern des Multiplikanden nur die Werte 0 bzw. 1 annehmen können. Das Ergebnis ist das Produkt der beiden Operanden, wenn die einzelnen Produkte entsprechend dem Stellengewicht des verwendeten Multiplikandenbits untereinander geschrieben und die so entstandenen Produkte dann addiert werden. Um mögliche Überläufe bei der Multiplikation zu vermeiden, wird die Wortbreite des Produkts in vielen Fällen gleich der Summe der Wortbreiten von Multiplikand und Multiplikator gewählt. Als Beispiel kann die Multiplikation von 1011 und 0101 wie folgt durchgeführt werden (Gehrke et al., 2016, S. 29).

$$\begin{array}{r}
 1011 \cdot 0101 \\
 + \quad 1011 \\
 + \quad 0000 \\
 + \quad 1011 \\
 + \quad 0000 \\
 = \quad 0110111
 \end{array}$$

Bei der binären Division von vorzeichenlosen Zahlen wird der Dividend testweise vom Divisor abgezogen. Das Ergebnis der Subtraktion wird für weitere Berechnungen verwendet, wenn es bei der Subtraktion keinen Überlauf und ein Quotientenbit mit dem Wert 1 gibt. Das berechnete Quotientenbit wird jedoch auf 0 gesetzt und das Subtraktionsergebnis verworfen, wenn ein Überlauf stattgefunden hat. Der Minuend wird zur Berechnung hinzugezählt. Die berechnete Differenz (kein Überlauf) oder das Minimum wird vor der folgenden Subtraktion addiert, um das nächste Quotientenbit zu bestimmen (falls ein Überlauf aufgetreten ist). Der volle Dividendenbetrag wird auf diese Weise der Reihe nach durchlaufen. Der Rest der Division wird durch das Ergebnis der vorherigen Subtraktion bestimmt. Es ist zu beachten, dass die führenden Nullen des Divisors nicht berücksichtigt werden. Für einen Dividend mit dem Wert 10101010 und einen Divisor mit der Zahl 1011 sind die Schritte für eine binäre Division wie folgt:

$$\begin{array}{r}
10101010 : 1011 = 01111 \\
- 1011 \\
\hline
10101 \\
- 1011 \\
\hline
10100 \\
- 1011 \\
\hline
10011 \\
- 1011 \\
\hline
10000 \\
- 1011 \\
\hline
0101
\end{array}$$

Das Endergebnis ist 01111 mit dem Rest 0101.

1.4 Digitalcodes und ihre Anwendungen

Signale mit zwei eindeutigen Werten und Schaltungskomponenten mit zwei stabilen Zuständen werden in digitalen Systemen verwendet. Es besteht eine direkte Entsprechung zwischen digitalen Signalen, digitalen Schaltungselementen und Binärziffern. So können beispielsweise n digitale Schaltkreiselemente verwendet werden, um eine Binärzahl mit n Ziffern darzustellen. Diese Komponenten haben jeweils ein Ausgangssignal, das gleich 0 oder 1 ist. Binäre Ziffern sind nur eines von vielen verschiedenen Informationselementen, die von digitalen Systemen dargestellt und verarbeitet werden können.

Ein Binärcode (d. h. eine Folge von 0 und 1) kann verwendet werden, um eine beliebige diskrete Information darzustellen, die aus einer Sammlung von Zahlen herausragt. Da derzeit nur Schaltkreise, die Muster von 0 und 1 codieren und manipulieren, kostengünstig für den Einsatz in Computern hergestellt werden können, müssen die Codes binär sein. Man muss sich jedoch darüber im Klaren sein, dass Binärcores nur die Symbole verändern, nicht aber die Bedeutung der Informationskomponenten, für die sie stehen.

Ein zufälliger Blick auf die Bits eines Computers zeigt, dass sie in der Regel eine Form von codierter Information und keine binären Zahlen enthalten. Der Begriff „ n -Bit-Binärcode“ bezieht sich auf eine Menge von n Bits, die bis zu 2^n verschiedene Kombinationen von 1 und 0 annehmen können, wobei jede Kombination für ein anderes Element der Menge steht, die codiert wird. Zwei Bits können verwendet werden, um eine Menge von vier Elementen zu codieren, und jedem Element kann eine der folgenden Bitkombinationen zugeordnet werden: 00, 01, 10, 11. Für eine Menge von acht Elementen wird ein Drei-Bit-Code benötigt, während für eine Menge von 16 Elementen ein Vier-Bit-Code erforderlich ist. Zur Berechnung der Bitkombination eines n -Bit-Codes wird die Binärzahl von 0 bis $2^n - 1$ verwendet. Um Mehrdeutigkeit bei der Codezuweisung zu vermeiden, muss jedem Element eine eindeutige binäre Bitkombination zugewiesen werden, keine zwei Elemente können denselben Wert haben. Eine maximale Anzahl von Bits, die in einem Binärcode verwendet werden können, existiert nicht, obwohl n die niedrigste Anzahl von Bits ist, die benötigt

wird, um 2^n eindeutige Größen zu codieren. Die zehn Dezimalziffern können beispielsweise mit zehn Bits codiert werden, und jeder Ziffer kann eine Bitkombination aus neun Nullen und einer Eins zugeordnet werden. Die Bitkombination 0010000000 ist in diesem speziellen Binärcode der Zahl sieben zugeordnet.

BCD-Code

Obwohl das binäre Zahlensystem das natürlichste System für einen Computer ist, weil es in der heutigen elektronischen Technologie leicht darstellbar ist, sind die meisten Menschen mehr an das Dezimalsystem gewöhnt. Die Umwandlung von Dezimalzahlen in Binärzahlen, die Durchführung aller arithmetischen Operationen im Binärsystem und die anschließende Konvertierung der Binärausgaben zurück in das Dezimalsystem ist eine Methode zur Behebung dieser Diskrepanz. Um mit diesem Verfahren ganze Dezimalzahlen in Binärzahlen umzuwandeln, müssen sie zunächst im Computer gespeichert werden. Die Dezimalzahlen müssen mit 1 und 0 codiert werden, um sie darstellen zu können, da der Computer nur binäre Werte verarbeiten kann. Insofern Dezimalzahlen in codierter Form im Computer gespeichert sind, können auch direkt arithmetische Operationen mit ihnen durchgeführt werden.

Wenn die Gesamtzahl der Glieder der Menge nicht ein Vielfaches von zwei ist, gibt es in einem Binärcode einige nicht zugeordnete Bitkombinationen. Eine solche Menge umfasst die zehn Dezimalziffern. Die Mindestanzahl an Bits, die für einen Binärcode erforderlich ist, um zwischen zehn verschiedenen Elementen zu unterscheiden, beträgt vier, aber sechs der möglichen 16 Optionen sind immer noch nicht zugewiesen. Vier Bits können auf zehn verschiedene Arten angeordnet werden, um verschiedene Binärcodes zu erzeugen. Die in der folgenden Tabelle erwähnte direkte binäre Zuordnung ist diejenige, die am häufigsten für Dezimalziffern verwendet wird. Der Name dieses Systems ist Binary Coded Decimal, kurz BCD.

Tabelle 3: Binary Coded Decimal (BCD)

Dezimalziffer	BCD-Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000

Dezimalziffer	BCD-Code
9	1001

Quelle: Moustafa Nawito, 2022.

Die Vier-Bit-Codierung für eine Dezimalziffer ist in Tabelle 2 dargestellt. In BCD benötigt eine Zahl mit k Dezimalziffern $4k$ Bits. Die 12-Bit-BCD-Darstellung der Dezimalzahl 546 lautet 0101 0100 0110, wobei jeder Satz von vier Bits für eine Dezimalstelle steht. Nur wenn eine Dezimalzahl zwischen null und neun liegt, ist sie mit ihrem binären Gegencode in BCD vergleichbar. Obwohl sowohl eine BCD-Zahl als auch die ihr entsprechende Binärzahl 1 und 0 enthalten, sieht eine BCD-Zahl größer als zehn anders aus. Außerdem werden bei BCD die Binärkombinationen 1010 bis 1111 nicht verwendet oder interpretiert, da sie bedeutungslos sind. Nehmen wir die Dezimalzahl 815 und ihre Darstellung in Binär- und BCD-Code:

$$815_{10} = (1000\ 0001\ 0101)_{\text{BCD}} = 1100101111_2$$

Die entsprechende Binärzahl benötigt nur zehn Bits, aber der BCD-Wert erfordert zwölf Bits, um die Zeichen des Dezimalwerts zu codieren. Es sollte klar sein, dass eine BCD-Zahl mehr Bits benötigt, um sie darzustellen, als der entsprechende Binärwert. Die Verwendung von Dezimalzahlen hat jedoch einen Vorteil, da Menschen, die das Dezimalsystem verwenden, Computer-Eingabe- und -Ausgabedaten produzieren.

Es ist wichtig zu verstehen, dass BCD-Zahlen trotz der Verwendung von Bits in ihrer Darstellung Dezimalzahlen und keine Binärzahlen sind.



MERKE

Der Hauptunterschied zwischen einer Dezimalzahl und einer BCD-Zahl besteht darin, dass letztere den Binärcode 0000, 0001, 0010, bis 1001 verwendet, während erstere die Symbole 0, 1, 2 und 9 verwendet. Der Dezimalwert ist derselbe. BCD verwendet acht Bits, um die Dezimalzahl 11 als 0001 0001 und die Dezimalzahl 25 als 0010 0101 darzustellen.

Weitere Dezimalcodes

In Binär-codes für Dezimalzahlen müssen mindestens vier Bits pro Ziffer vorhanden sein. Durch die Kombination von vier Bits in zehn verschiedenen Kombinationen können viele verschiedene Codes erstellt werden. Die folgende Tabelle veranschaulicht BCD zusammen mit drei weiteren typischen Codes. Von den 16 Bit-Permutationen, die mit vier Bits erstellt werden können, verwendet jeder Code nur zehn. Die restlichen sechs unbrauchbaren Kombinationen sind bedeutungslos und sollten vermieden werden.

- Zu den gewichteten Codes gehören der 2421-Code und BCD. Jede Bitposition in einem gewichteten Code erhält einen Gewichtungsfaktor, sodass jede Ziffer durch Summierung der Gewichte aller Einsen in der codierten Kombination bewertet werden kann. Die Zweierpotenzen jedes Bits werden durch die Gewichte des BCD-Codes repräsentiert, die 8, 4, 2 und 1 betragen. Da $8 \cdot 0 + 4 \cdot 1 + 2 \cdot 0 + 1 \cdot 1 = 5$ ist, verstehen die Gewichte beispielsweise die Bitzuordnung 0101 als die Dezimalzahl 5. Wenn die entsprechenden Ziffern 2421 mit der Bitkombination 1001 gewichtet werden, ergibt sich $2 \cdot 1 + 4 \cdot 0 + 2 \cdot 0 + 1 \cdot 1 = 3$, was der Dezimalzahl entspricht. Es ist zu beachten, dass der Code 2421 die Codierung einiger Ziffern auf zwei verschiedene Arten ermöglicht. So kann beispielsweise der Bitkombination 0100 oder 1010 die Dezimalzahl vier zugeordnet werden, da beide Kombinationen ein Gewicht von vier haben. BCD-Addierer addieren BCD-Werte direkt, Ziffer für Ziffer, ohne die Ganzzahlen vorher in Binärzahlen umzuwandeln. Das Ergebnis muss jedoch um sechs erhöht werden, wenn es größer als neun ist. BCD-Addierer sind weitaus hardwareintensiver und übertreffen herkömmliche Binäraddierer in Bezug auf die Geschwindigkeit nicht mehr.
- Beispiele für selbstkomplementierende Codes sind 2421 und der Exzess-3 (auch Stibitz-Code und Überschuss-3-Code). Mit diesen Codes kann das 9er-Komplement einer dezimalen Ganzzahl erzeugt werden, indem einfach Einsen in Nullen und Nullen in Einsen geändert werden (d. h. durch Komplementierung jedes Bits im Muster). Die Dezimalzahl 395 wird zum Beispiel mit dem zusätzlichen 3er-Code als 0110 1100 1000 codiert. Durch einfaches Komplementieren jedes Bits des Codes, wie beim 1er-Komplement von binären Ganzzahlen, wird das 9er-Komplement von 604 als 1001 0011 0111 dargestellt. Aufgrund seiner Fähigkeit, sich selbst zu komplementieren, wurde der Excess-3-Code in einigen älteren Systemen verwendet. Ein ungewichteter Code namens Excess-3 erhält jede codierte Kombination aus dem passenden Binärwert + 3. Beachten Sie, dass der BCD-Code nicht selbstkomplementär ist.
- Ein Beispiel dafür, wie man einen Dezimalcode sowohl mit positiven als auch mit negativen Gewichten versehen kann, ist der Code 8, 4, -2, -1. Die Bitkombination 0110 wird in diesem Fall in die Dezimalzahl zwei übersetzt und wie folgt berechnet: $8 \cdot 0 + 4 \cdot 1 + (-2) \cdot 1 + (-1) \cdot 0 = 2$.

Tabelle 4: BCD, 2421, Exzess-3 und 8,4,-2,-1 Dezimalcodes

Dezimalziffer	BCD	2421	Exzess-3	8,4,-2-1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001

Dezimalziffer	BCD	2421	Exzess-3	8,4,-2-1
8	1000	1110	1011	1000
9	1001	1111	1100	1111

Quelle: Moustafa Nawito, 2022

Gray-Code

Zahlreiche physikalische Systeme erzeugen Daten in Form von kontinuierlichen Größen. Bevor diese Daten in einem digitalen System verwendet werden können, müssen sie in digitale Form umgewandelt werden mittels Analog-Digital-Wandler. Gelegentlich ist es praktisch, digitale Daten, die aus analogen Daten umgewandelt wurden, mit dem in folgender Tabelle dargestellten Gray-Code (vgl. Gray, 1927) darzustellen. Der **Gray-Code** hat einen Vorteil gegenüber einer reinen binären Zahlenfolge: Nur ein Bit in der Codegruppe ändert sich, wenn sich eine Zahl ändert. Zum Beispiel wandelt sich der Gray-Code von 0010 auf 01100, wenn man von drei auf vier wechselt.

Gray-Code

Dies ist ein Binärcode, bei dem nur ein Bit in der Codegruppe sich ändert, wenn sich eine Zahl ändert.

Tabelle 5: Gray-Code

Dezimalziffer	Gray-Code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Quelle: Moustafa Nawito, 2022.

Der Gray-Code wird in Anwendungen eingesetzt, bei denen der regelmäßige Fluss der von der Hardware erzeugten Binärzahlen zu einem Fehler oder einer Mehrdeutigkeit beim Übergang von einer Zahl zur nächsten führen könnte. Bei der Verwendung von Binärzahlen könnte eine Änderung von z. B. 0111 auf 1000 zu der zwischenzeitlich falschen Zahl 1001 führen, wenn das ganz rechte Bit länger braucht, um sich zu ändern, als die Werte der anderen drei Bits. Für den Computer, der die Daten verwendet, kann dies fatale Folgen haben. Da beim Übergang zwischen zwei Zahlen nur ein Bit seinen Wert ändert, löst der Gray-Code dieses Problem.

ASCII-Code

Nur Zeichen, wie Buchstaben, Ziffern und Sonderzeichen, werden mit dem ASCII-Code (American Standard Code for Information Interchange) codiert. Zur Darstellung jedes Zeichens werden sieben Bits verwendet. Der 7-Bit-Code nach DIN 66003, der im Gegensatz zum ASCII-Code u. a. auch die deutschen Umlaute umfasst, entspricht fast genau dem ASCII-Code. Die folgende Abbildung veranschaulicht die Zeichencodierung nach dem ASCII-Code. In dieser Abbildung werden die Bits a0, a1, a2 und a3 für die Zeilenauswahl verwendet, während die Bits a4, a5 und a6 für die Spaltenauswahl genutzt werden. Für die Übertragung eines ASCII-Zeichens wird normalerweise ein Byte (acht Bits) angewandt. Das achte Bit wird in der Datentechnik häufig zur Vergrößerung des Zeichensatzes verwendet. Dadurch kann die Anzahl der codierten Zeichen verdoppelt werden.

Der sogenannte Unicode findet in vielen Computersystemen auch zur Codierung von Zeichen Anwendung, da der ASCII-Code nur einen begrenzten Zeichensatz von 128 oder 256 verschiedenen Zeichen bietet. Das Ziel von Unicode ist es, jedes derzeit existierende Zeichen codieren zu können. Aus diesem Grund werden mit Unicode, der bis zu 65535 Zeichen unterstützt, Stufen (auch als Ebenen bezeichnet) festgelegt. Der Nachteil, eine wesentlich höhere Anzahl von Bits pro Zeichen angeben zu müssen, wird jedoch durch den Vorteil ausgeglichen, alle gängigen Zeichen codieren zu können. Daher wird bei einfachen Anwendungen (z. B. Status- und Fehlermeldungen eines digitalen Systems) in der Regel auf die Verwendung von Unicode zugunsten des einfacheren ASCII-Codes verzichtet (Gehrke, 2016).

Abbildung 6: Siebenstelliger ASCII-Code

ASCII Table		Decimal		Hexadecimal		Binary		Octal		Char		
Decimal	Hexadecimal	Decimal	Hexadecimal	Binary	Octal	Char	Char	Decimal	Hexadecimal	Binary	Octal	Char
000	00	000	00	0000000	000	(NULL)	0	000	00	0110000	060	
001	01	0000001	001	(START OF HEADING)	001		1	001	01	0110001	061	a
002	02	0000010	002	(START OF TEXT)	002		2	002	02	0110010	062	b
003	03	0000011	003	(END OF TEXT)	003		3	003	03	0110011	063	c
004	04	0000100	004	(END OF TRANSMISSION)	004		4	004	04	0110100	064	d
005	05	0000101	005	(ENQUIRY)	005		5	005	05	0110101	065	e
006	06	0000110	006	(ACKNOWLEDGE)	006		6	006	06	0110110	066	f
007	07	0000111	007	(BELL)	007		7	007	07	0110111	067	g
008	08	0001000	010	(BACKSPACE)	008		8	008	08	0111000	070	h
009	09	0001001	011	(HORIZONTAL TAB)	009		9	009	09	0111001	071	i
010	0A	0001010	012	(LINE FEED)	010		A	006	6A	01101010	072	j
011	0B	0001011	013	(VERTICAL TAB)	011		B	005	9A	0111011	073	k
012	0C	0001100	014	(FORM FEED)	012		C	006	3C	0111100	074	l
013	0D	0001101	015	(CARRIAGE RETURN)	013		D	006	3D	0111101	075	m
014	0E	0001110	016	(SHIFT OUT)	014		E	006	3E	0000001	076	=
015	0F	0001111	017	(SHIFT IN)	015		F	006	3F	0111111	077	>
016	10	0010000	020	(DATA LINK ESCAPE)	016		@	006	40	0000000	100	@
017	11	0010001	021	(DEVICE CONTROL 1)	017		A	005	41	1000001	101	A
018	12	0010010	022	(DEVICE CONTROL 2)	018		B	006	42	1000010	102	B
019	13	0010011	023	(DEVICE CONTROL 3)	019		C	007	43	1000011	103	C
020	14	0010100	024	(DEVICE CONTROL 4)	020		D	008	44	1000100	104	D
021	15	0010101	025	(NEGATIVE ACKNOWLEDGE)	021		E	009	45	1000101	105	E
022	16	0010110	026	(SYNCHRONOUS IDLE)	022		F	009	46	1000110	106	F
023	17	0010111	027	(ENG OF TRANS., BLOCK)	023		G	011	47	1000111	107	G
024	18	0011000	030	(CANCEL)	024		H	011	48	1001000	110	H
025	19	0011001	031	(END OF MEDIUM)	025		I	011	49	1001001	111	I
026	1A	0011010	032	(SUBSTITUTE)	026		J	011	4A	1001010	112	J
027	1B	0011011	033	(ESCAPE)	027		K	011	4B	1001011	113	K
028	1C	0011100	034	(FILE SEPARATOR)	028		L	011	4C	1001100	114	L
029	1D	0011101	035	(GROUP SEPARATOR)	029		M	011	4D	1001101	115	M
030	1E	0011110	036	(RECORD SEPARATOR)	030		N	011	4E	1001110	116	N
031	1F	0011111	037	(UNIT SEPARATOR)	031		O	011	4F	1001111	117	O
032	20	0100000	040	(SPACE)	032		P	000	50	1010000	120	P
033	21	0100001	041	!	033		Q	001	51	1010001	121	Q
034	22	0100010	042	#	034		R	001	52	1010010	122	R
035	23	0100011	043	\$	035		S	001	53	1010011	123	S
036	24	0100100	044	%	036		T	001	54	1010100	124	T
037	25	0100101	045	&	037		U	001	55	1010101	125	U
038	26	0100110	046	8	038		V	001	56	1010110	126	V
039	27	0100111	047	'	039		W	001	57	1010111	127	W
040	28	0101000	050	(040		X	001	58	1011000	130	X
041	29	0101001	051)	041		Y	001	59	1011001	131	Y
042	2A	0101010	052	*	042		Z	001	5A	1011010	132	Z
043	2B	0101011	053	+	043		[001	5B	1011011	133	[
044	2C	0101100	054	,	044]	001	5C	1011100	134]
045	2D	0101101	055	-	045		^	001	5D	1011101	135	^
046	2E	0101110	056	.	046		_	001	5E	1011110	136	_
047	2F	0101111	057	/	047		-	001	5F	1011111	137	-

Quelle: Psychpsy, Wikimedia Commons, [CC 1.0], 2022.



ZUSAMMENFASSUNG

Das moderne Zeitalter ist gekennzeichnet durch den weit verbreiteten Einsatz digitaler Systeme in fast allen Bereichen des täglichen Lebens. Diese Systeme verarbeiten und speichern Daten sowie Informationen in Form diskreter Ziffern.

Um die Funktionsweise digitaler Systeme nachzuvollziehen, muss ein Verständnis für die verschiedenen Zahlensysteme entwickelt werden, insbesondere für das Binärsystem. Es ist möglich, zwischen verschiedenen Binärsystemen zu konvertieren, entweder direkt oder durch eine Umwandlung in das Dezimalsystem als Zwischenschritt.

Ähnlich wie im Dezimalsystem können auch im Binärsystem arithmetische Operationen, wie Addition, Subtraktion, Multiplikation und Division, durchgeführt werden. Bei der Subtraktion reduziert die Verwendung von Komplementen die Komplexität der Operation erheblich und ermöglicht eine effiziente Hardware-Realisierung.

Für die Darstellung digitaler Daten stehen verschiedene Codes mit unterschiedlichen Anwendungsszenarien zur Verfügung. BCD und andere dezimale Codes stellen binäre Daten in einer für den Menschen verständlicheren Weise dar. Der Gray-Code findet vor allem bei der Fehlererkennung Anwendung. Der ASCII-Code und seine Erweiterungen werden verwendet, um einen umfangreichen Satz von Zeichen, Buchstaben und Symbolen darzustellen.

LEKTION 2

DARSTELLUNG, SYNTHESE UND ANALYSE BOOLESCHER FUNKTIONEN

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die grundlegenden Axiome und Theorien der Booleschen Algebra anzuwenden.
- Boolesche Funktionen in den Normalformen darzustellen.
- Karnaugh-Veitch-Diagramme für die Vereinfachung von logischen Ausdrücken zu verwenden.
- Boolesche Funktionen mittels NOR- oder NAND-Gattern vollständig zu realisieren.

2. DARSTELLUNG, SYNTHESE UND ANALYSE BOOLESCHER FUNKTIONEN

Einführung

Die digitale Logik befasst sich mit der Entwicklung digitaler Systeme, die logische und arithmetische Operationen durchführen können. Die Grundlagen des digitalen Logikentwurfs liegen in den Theorien und Axiomen der Booleschen Algebra. Diese werden in der folgenden Lektion neben dem Konzept der logischen Funktionen detailliert vorgestellt. Der Schwerpunkt liegt auch auf Methoden und Techniken zur Vereinfachung logischer Ausdrücke, um die Effizienz der Hardware-Realisierung zu erreichen.

2.1 Boolesche Algebra und Funktionen

Wie jedes andere deduktive mathematische System kann auch die Boolesche Algebra durch ihre Elemente, Operatoren und unbewiesenen Axiome oder Postulate definiert werden. Jede Gruppe von Objekten, die typischerweise eine Eigenschaft gemeinsam haben, wird als Menge von Elementen bezeichnet.

- Wenn x und y bestimmte Objekte sind und S eine Menge ist, bedeutet $x \in S$, dass x ein Mitglied von S ist, und $y \notin S$ bedeutet, dass y kein Element von S ist.
- $A = \{1,2,3,4\}$ bedeutet, dass die Zahlen 1, 2, 3 und 4 die Elemente der Menge A sind.
- Ein binärer Operator ist eine Regel, die jedem Paar von Elementen in der Menge S ein anderes Element aus der Menge S zuordnet, z. B. $a * b = c$. Man sagt, dass \cdot ein binärer Operator ist, wenn es eine Methode gibt, um c aus dem Paar (a, b) zu erhalten und wenn a, b und $c \in S$ sind. Allerdings ist $*$ kein binärer Operator, wenn $a, b \in S$ und $c \notin S$ sind.

George Boole schuf 1854 den mathematischen Rahmen, der heute als Boolesche Algebra bekannt ist. Eine von Claude E. Shannon 1938 entwickelte zweiwertige Boolesche Algebra, die sogenannte Schaltalgebra, stellt die Eigenschaften von bistabilen elektrischen Schaltkreisen dar. Für die formale Formulierung der Booleschen Algebra werden hier die von E. V. Huntington 1904 entwickelten Postulate verwendet. Die Boolesche Algebra ist eine algebraische Struktur, die unter bestimmten, von Huntington aufgestellten Axiomen durch eine Menge von Elementen, B , sowie die binären Operatoren $+$ und \cdot definiert ist. Diese Postulate lauten:

1. Ausgehend von der allgemeinen Definition des algebraischen Abschlusses, nämlich „Eine Menge S ist in Bezug auf einen binären Operator abgeschlossen, wenn der binäre Operator für jedes Paar von Elementen von S eine Regel angibt, um ein einziges Element von S zu erhalten“, gilt Folgendes
 - a) Die Struktur ist geschlossen in Bezug auf den Operator $+$.
 - b) Die Struktur ist geschlossen in Bezug auf den Operator \cdot .
- 2.

- a) Das Element 0 ist ein Identitätselement in Bezug auf $+$, d. h. $x + 0 = 0 + x = x$.
 - b) Das Element 1 ist ein Identitätselement in Bezug auf \cdot ; das heißt, $x \cdot 1 = 1 \cdot x = x$.
- 3.
- a) Die Struktur ist kommutativ in Bezug auf $+$, d. h. $x + y = y + x$.
 - b) Die Struktur ist kommutativ in Bezug auf \cdot ; d. h. $x \cdot y = y \cdot x$.
- 4.
- a) Der Operator \cdot ist distributiv über $+$, d. h. $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
 - b) Der Operator $+$ ist distributiv über \cdot , d. h. $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. Für jedes Element $x \in B$ gibt es ein Element $x' \in B$, das auch als $\bar{x} \in B$ geschrieben werden kann (Komplement von x genannt), sodass
- a) $x + x' = 1$ und
 - b) $x \cdot x' = 0$.
6. Es gibt mindestens zwei Elemente $x, y \in B$, sodass $x \neq y$ ist.

Es ist wichtig, an dieser Stelle zu erwähnen, dass mehrere Arten der Booleschen Algebra auf der Grundlage der gewählten B-Elemente und der Funktionsprinzipien erstellt werden können. Die Behandlung dieses Themas wird sich auf eine zweiwertige Boolesche Algebra beschränken, da unser Interesse in diesem Kurs auf der Anwendung der Booleschen Algebra auf gatterartige Schaltungen liegt, die häufig in digitalen Geräten und Computern verwendet werden (d. h. eine Boolesche Algebra mit nur zwei Elementen). Die zweiwertige Boolesche Algebra wird von Ingenieur:innen auch als „Schaltalgebra“ bezeichnet. Um die Ähnlichkeiten zwischen der zweiwertigen Booleschen Algebra und anderen binären Systemen zu betonen, wird in diesem Lehrbrief auch der Begriff „binäre Logik“ für die zweiwertige Boolesche Algebra verwendet.

Binäre Logik

Die binäre Logik befasst sich mit Operationen, die eine logische Bedeutung haben und mit Variablen, die zwei diskrete Werte besitzen. Obwohl wahr und falsch, ja und nein und andere Namen für die beiden Werte, die die Variablen annehmen, verwendet werden, ist es bequemer, sie in Form von Bits zu konzipieren und für unsere Zwecke die Werte 1 und 0 zu verwenden. Die Einführung in die binäre Logik in diesem Lernzyklus ist analog zu dem Zweig der Algebra, der als Boolesche Algebra bekannt ist.

Die binäre Logik besteht aus einer Reihe von logischen Operationen und binären Variablen. Jede Variable hat nur zwei mögliche Werte: 1 und 0. Die Variablen werden durch Buchstaben des Alphabets gekennzeichnet, z. B. A, B, C, x, y und z . Drei logische Operationen sind grundlegend: AND, OR und NOT. Jede Operation hat ein binäres Ergebnis, das durch den Buchstaben z gekennzeichnet ist.

1. AND (UND): Ein Punkt oder das Fehlen eines Operators dient als Darstellung für diese Operation. Zum Beispiel: „ x AND y ist gleich z “ wird als „ $x \cdot y = z$ “ oder „ $xy = z$ “ gelesen. Nach der Interpretation der logischen Operation AND ist $z = 1$, wenn und nur wenn $x = 1$ und $y = 1$ ist; andernfalls ist $z = 0$ (zur Erinnerung: x, y und z sind binäre Variablen und können nur den Wert 1 oder 0 annehmen; andere Werte sind ausgeschlossen). Z ist das Ergebnis der Operation xy . Die AND-Operation ist auch als

Konjunktion

Eine logische Operation, wobei alle Eingangsvariablen wahr sein müssen, damit der Ausgang auch wahr ist, nennt man Konjunktion.

Diskonjunktion

Eine logische Operation, wobei nur eine Eingangsvariable wahr sein muss, damit der Ausgang auch wahr ist, nennt man Disjunktion.

Konjunktion bezeichnet. Beim Schreiben logischer Ausdrücke wird auch das Symbol \wedge verwendet, um die AND-Verknüpfung auszudrücken (sieht aus wie der Großbuchstabe A, aber ohne den Strich).

- OR (ODER): Diese Operation wird mit einem Pluszeichen gekennzeichnet. Wenn z. B. $x + y = z$ als „ x OR y ist gleich z “ geschrieben wird, bedeutet dies, dass $z = 1$ ist, wenn entweder x oder y oder sowohl x als auch $y = 1$ sind. $Z = 0$, wenn x und y beide gleich 0 sind. Beim Schreiben von logischen Ausdrücken wird auch das Symbol \vee verwendet, um die OR-Verknüpfung auszudrücken (nicht zu verwechseln mit dem Großbuchstaben V). Die OR-Operation ist auch als **Diskonjunktion** bezeichnet.
- NOT (NICHT): Diese Operation kann durch eine Primzahl oder durch einen Überstrich dargestellt werden. $x' = z$ (oder $\bar{x} = z$) bedeutet zum Beispiel „nicht x ist gleich z “, was bedeutet, dass z das ist, was x nicht ist. Wenn also $x = 1$ ist, ist $z = 0$, wenn aber $x = 0$ ist, folgt $z = 1$. Da die NOT-Operation eine 1 in eine 0 und eine 0 in eine 1 umwandelt, ergibt die Komplementierung einer 1 eine 0 und umgekehrt. Beim Schreiben von logischen Ausdrücken wird auch das Symbol \neg verwendet, um die NOT-Verknüpfung auszudrücken (Negationssymbol genannt).

Die Operationen AND und OR haben Parallelen zu den Operationen Multiplikation bzw. Addition, und die binäre Logik ähnelt der binären Arithmetik. Die Symbole für AND und OR sind in Wirklichkeit die gleichen wie die Symbole für Multiplikation und Addition. Die binäre Arithmetik sollte jedoch nicht mit der binären Logik verwechselt werden. Es ist wichtig zu verstehen, dass eine arithmetische Variable eine Zahl bezeichnet, die mehrere Ziffern haben kann. Eine logische Variable kann immer nur entweder 1 oder 0 sein. $1 + 1 = 10$ in der binären Arithmetik bedeutet zum Beispiel „eins plus eins ist gleich zwei“, während $1 + 1 = 1$ in der binären Logik „eins OR eins ist gleich eins“ bedeutet.

Die Definition der logischen Operation gibt für jede Kombination der Werte von x und y einen Wert von z an. Für die Auflistung der Definitionen logischer Operationen kann ein übersichtliches Format verwendet werden, das als Wahrheitstabelle bekannt ist. Eine Wahrheitstabelle ist eine Liste aller denkbaren Kombinationen der Variablen, die die Beziehung zwischen den verfügbaren Werten für die Variablen und dem Ergebnis der Operation aufzeigt. Um die Wahrheitstabellen für die AND- und OR-Verknüpfungen unter Verwendung der Variablen x und y zu erstellen, muss man alle möglichen Werte, die die Variablen haben könnten, wenn sie paarweise kombiniert werden, auflisten.

Die folgende Tabelle enthält die Wahrheitstabellen für NOT, AND und OR. Diese Tabellen machen die Definitionen der Operationen deutlich.

Tabelle 6: Wahrheitstabellen für NOT, AND, und OR-Operationen

NOT		AND			OR		
A	A'	A	B	A • B	A	B	A + B
0	1	0	0	0	0	0	0
1	0	0	1	0	0	1	1

	1	0	0	1	0	1
	1	1	1	1	1	1

Quelle: Moustafa Nawito, 2022.

Ausgehend von den allgemeinen Postulaten der Booleschen Algebra sind die wichtigsten Theoreme und Postulate der binären Logik in folgender Tabelle zusammengefasst.

Tabelle 7: Theoreme und Postulate der binären Logik

Identitätspostulat	$x + 0 = x$	$x \cdot 1 = x$
Kommutativpostulat	$x + y = y + x$	$xy = yx$
Distributivpostulat	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
Umkehrpostulat	$x + x' = 1$	$x \cdot x' = 0$
Assoziativtheorie	$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$
DeMorgansche Theorie	$(x + y)' = x'y'$	$(xy)' = x' + y'$
Theorie 1	$x + x = x$	$x \cdot x = x$
Theorie 2	$x + 1 = 1$	$x \cdot 0 = 0$
Theorie 3	$(x')' = x$	
Theorie 5	$x + x \cdot y = x$	$x(x + y) = x$

Quelle: Moustafa Nawito, 2022 in Anlehnung an Mano, 2013, S. 43.

Bei der Analyse Boolescher Ausdrücke gilt die Rangfolge

1. Klammerung,
2. Not,
3. AND und
4. OR.

Mit anderen Worten: Ausdrücke innerhalb von Klammern müssen vor allen anderen Operationen ausgewertet werden. Das Komplement ist die nächste Operation in der Reihenfolge der Priorität, gefolgt von AND und dann OR.

Das **Dualitätsprinzip**, ein entscheidendes Merkmal der Booleschen Algebra, besagt, dass jeder algebraische Ausdruck, der sich aus den Postulaten der Booleschen Algebra ableiten lässt, gültig bleibt, wenn die Operatoren und die Identitätselemente vertauscht werden. Die Elemente der Identität und der Menge B in einer zweiwertigen Booleschen Algebra sind 1 bzw. 0. Es gibt zahlreiche Anwendungen für das Dualitätsprinzip. Die OR- und AND-Operatoren werden einfach vertauscht, und Einsen und Nullen werden durch Nullen und Einsen ersetzt, um das Dual einer algebraischen Aussage zu erhalten.

Dualitätsprinzip
Es besagt, dass in der Booleschen Algebra, jeder Ausdruck, der sich aus den Postulaten der Booleschen Algebra ableiten lässt, gültig bleibt, wenn die Operatoren und die Identitätselemente vertauscht werden.

Mithilfe der oben genannten Gesetze können komplizierte logische Ausdrücke vereinfacht werden.

Boolesche Funktionen

Ein algebraischer Ausdruck für eine Boolesche Funktion besteht aus binären Variablen, den Konstanten 0 und 1 und den Symbolen für die logischen Operationen. Die Funktion hat zwei mögliche Werte für einen gegebenen Wert der binären Variablen: 1 oder 0. Zur Veranschaulichung dient die Boolesche Funktion:

$$F1 = (x + y)z'$$

Die Funktion ist als algebraischer Ausdruck dargestellt. In diesem Kontext sind die Variablen x , y und z und ihre Komplemente auch als Literale zu bezeichnen. Der binäre Wert des Ausdrucks wird zur Bewertung einer Booleschen Funktion verwendet, die die logische Beziehung zwischen binären Variablen für alle denkbaren Werte der Variablen ausdrückt. Eine Wahrheitstabelle kann eine Boolesche Funktion veranschaulichen. Die Wahrheitstabelle hat 2^n Zeilen, wobei n die Anzahl der Variablen in der Funktion ist. Durch Zählen der Binärzahlen von 0 bis $2^n - 1$ werden die Binärkombinationen der Wahrheitstabelle gefunden. Die Wahrheitstabelle für die Funktion F1 ist in folgender Tabelle dargestellt. Die drei Variablen x , y und z können jeweils in einer von acht Binärkombinationen mit einem anderen Satz von Bits versehen werden. Für jede dieser Kombinationen enthält die Spalte F1 entweder 0 oder 1.

Tabelle 8: Wahrheitstabelle für die Funktion $F1 = (x + y)z'$

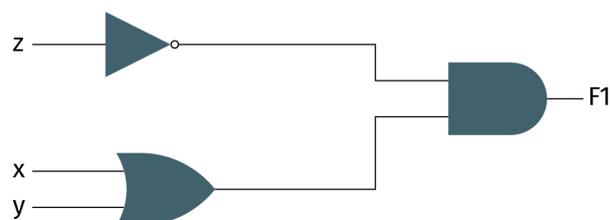
x	y	z	F1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Quelle: Moustafa Nawito, 2022.

Die folgende Abbildung zeigt den Entwurf einer logischen Schaltung, auch Schaltplan genannt, für F1. Für den Eingang x existiert ein Inverter, der sein Komplement erzeugt. Für den Ausdruck yz gibt es ein AND-Gatter. Für die Kombination von x' und yz besteht ein OR-Gatter. In logischen Schaltplänen werden die Variablen der Funktion als Eingänge des Schaltkreises und die binäre Variable F1 als Ausgang des Diagramms verwendet. Die

Beziehungen zwischen Ausgang und Eingang der Schaltung sind im Schaltplan dargestellt. Es wird gezeigt, wie die logischen Werte der einzelnen Ausgänge aus den logischen Werten der Eingänge berechnet werden, anstatt jede mögliche Kombination von Eingängen und Ausgängen detailliert darzustellen.

Abbildung 7: Schaltplan für die Funktion $F1 = (x + y)z'$



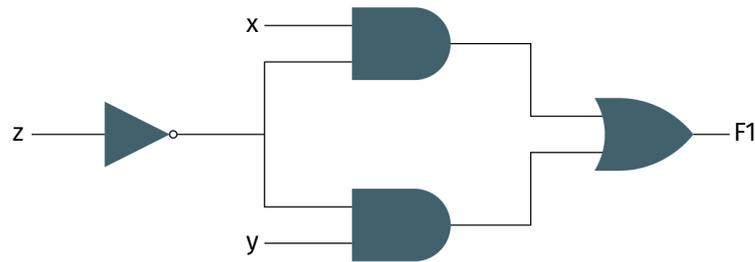
Quelle: Moustafa Nawito, 2022.

Eine Boolesche Funktion kann nur auf eine einzige Weise in einer Wahrheitstabelle ausgedrückt werden. Die Darstellung der Funktion ist auf verschiedene Arten möglich, die jedoch alle eine gleichwertige Logik haben, wenn die Funktion in algebraischer Form vorliegt. Die Art und Weise, wie die Gatter im Entwurf der Logikschaltung verbunden werden, hängt von dem spezifischen Ausdruck ab, der zur Darstellung der Funktion verwendet wird. Andererseits wird der logische Ausdruck dadurch bestimmt, wie die Gatter verbunden sind. Ein wichtiges Merkmal, das unsere Verwendung der Booleschen Algebra bestimmt, ist Folgendes: Gelegentlich ist es möglich, einen einfacheren Ausdruck für dieselbe Funktion zu erstellen, indem ein Boolescher Ausdruck gemäß den Prinzipien der Booleschen Algebra geändert wird, wodurch die Anzahl der Gatter in der Schaltung und die Anzahl der Eingänge zum Gatter verringert wird. Da die Kosten einer Schaltung dadurch drastisch gesenkt werden können, werden die Entwickler ermutigt, die Komplexität und die Anzahl der Gatter zu reduzieren. Als Beispiel kann Funktion F1 wie folgt ausgedrückt werden.

$$F1 = xz' + yz'$$

Wie in folgender Abbildung zu sehen ist, führt dies zu einem Schaltplan mit einem zusätzlichen AND-Gatter, auch wenn der Ausdruck der Funktion in obiger Abbildung völlig gleichwertig ist.

Abbildung 8: Schaltplan für die Funktion $F1 = xz' + yz'$



Quelle: Moustafa Nawito, 2022.

Das Komplement einer Funktion F , nämlich F' , kann mithilfe des in den „Wahrheitstabellen für NOT, AND, und OR-Operationen“ dargestellten DeMorgan-Theorems ermittelt werden. Ihre allgemeine Form, die für mehr als zwei Variablen gilt, lautet:

$$(A + B + C + D + \dots + Z)' = A'B'C'D' \dots Z' \quad (2.1)$$

$$(ABCD \dots Z)' = (A' + B' + C' + D' + \dots + Z') \quad (2.2)$$

Mit anderen Worten, die verallgemeinerte Form von DeMorgans Theoremen besagt, dass das Komplement einer Funktion realisiert wird, indem man die Operatoren AND und OR und jedes Literal vertauscht.

Algebraische Manipulationen von Booleschen Funktionen

Beim Entwurf einer Hardware-Realisierung ist es häufig möglich und (immer erwünscht), eine einfachere Schaltung zu erhalten, indem die Anzahl der Terme, die Anzahl der Literale oder beides in einem booleschen Ausdruck verringert wird. Bei der Manipulation der Booleschen Algebra geht es hauptsächlich darum, eine Gleichung zu verkleinern, um eine einfachere Schaltung zu erstellen. Entwickler digitaler Schaltungen verwenden Computer-Minimierungswerkzeuge, die ideale Schaltungen mit Millionen von Logikgattern für komplexe Boolesche Funktionen und zahlreichen Ausgängen erstellen können. Die grundlegende Theorie, worauf solche Computertools basieren, wurde in diesem Lernzyklus bereits präsentiert (siehe „Wahrheitstabellen für NOT, AND, und OR-Operationen“).

Für manuelle Manipulation von Booleschen Ausdrücken ist der einzige manuelle Weg ein Cut-and-Try-Verfahren, bei dem fundamentale Beziehungen und andere Manipulationstechniken verwendet werden, die mit der Zeit zur zweiten Natur werden, aber immer noch anfällig für menschliche Fehler sind.

BEISPIEL ALGEBRAISCHE MANIPULATION BOOLESCHER AUSDRÜCKE

Die folgenden Beispiele zeigen, wie die Boolesche Algebra algebraisch manipuliert werden kann:

- $x(x' + y)' = xx' + xy' = 0 + xy' = xy'$
- $(x + y)(x' + y) = xx' + xy + yx + yy = 0 + y(x + x) + y = yx + y = y$
 $(x + 1) = y$
- $xy + y'z + yz = xy + z(y + y') = xy + z$

2.2 Disjunktive und konjunktive Normalform

Durch die ausschließliche Verwendung von AND, OR und NOT kann jede binäre Funktion dargestellt werden. Das Beispiel der Funktion F in folgender Tabelle zeigt, wie dies auf methodische Art und Weise geschehen kann. Es gibt zwei Möglichkeiten, wie man vorgehen kann. Zunächst wird die Kanonische Disjunktive Normalform (KDNF) präsentiert.

Die Kanonische Disjunktive Normalform (KDNF)

Tabelle 9: Wahrheitstabelle der Funktion F2 für die Darstellung von KDNF und KKNF

x	y	z	F2
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Quelle: Moustafa Nawito, 2022.

Zunächst werden die Eingangsvektoren betrachtet, für die die Funktion $F2 = f(x, y, z)$ 1 ergibt. Dies sind in unserem Fall die Vektoren 0, 1, 3, 5 und 6. Nun wird für jeden dieser Eingangsvektoren eine Konjunktion der Elemente x_i gebildet, die für genau diesen Eingangsvektor den Wert 1 annimmt. Dies wäre dann der Vektor 5:

$$m_5 = x \neg y z$$

Minterm
Ein algebraischer Ausdruck, wobei alle Eingangsvariablen einer logischen Funktion sich in Konjunktion befinden.

Man nennt m_5 auch **Minterm**. Minterme werden auch als vollständige Konjunktionen bezeichnet, da sie immer alle Eingangsvariablen einschließen. In einem Minterm können, je nachdem, ob die betreffende Eingangsvariable 1 oder 0 ist, alle Eingangsvariablen invertiert oder nicht invertiert erscheinen. Die anderen Minterme für das Beispiel lauten:

$$\begin{aligned} m_0 &= x \cdot y \cdot z \\ m_1 &= \neg x \cdot \neg y \cdot z \\ m_3 &= \neg x \cdot y \cdot z \\ m_6 &= x \cdot y \cdot \neg z \end{aligned}$$

Folglich hat ein Minterm den Wert 1 für nur eine bestimmte Instanz der Eingangsvariablen. Da die Funktion 1 ausgeben soll, wenn eines der Minterme Gleichheit erreicht, muss die Disjunktion der Minterme verwendet werden, um die gesamte Funktion auszudrücken. Die kanonische disjunktive Normalform ist die Bezeichnung für diese Darstellungstechnik (KDNF). Da jedes Minterm jede Variable enthält, wird es als kanonisch bezeichnet. In unserem Beispiel ist die Funktion durch folgende Gleichung dargestellt:

$$F_2 = m_0 + m_1 + m_3 + m_5 + m_6 = \neg x \neg y \neg z + \neg x \neg y z + \neg x y z + x \neg y z + x y \neg z$$

Die DKNF kann auch als „Summe von Mintermen“ bezeichnet werden.

Die Kanonische Konjunktive Normalform (KKNF)

Eine Alternative ist die Darstellung der Funktion durch die Eingangsvektoren, für die die Funktion 0 zurückgibt. Das bedeutet, $f(x, y, z) = 0$ ist also wahr. Diese drei Eingangsvektoren sind 1, 4 und 7 für die Funktion in der oberen Tabelle.

Maxterm
Ein algebraischer Ausdruck, wobei alle Eingangsvariablen einer logischen Funktion sich in Disjunktionen befinden, nennt sich Maxterm.

Es entstehen die sogenannten **Maxterme**. Dies sind die Disjunktionen. Wenn der passende Eingangsvektor vorhanden ist, sind sie genau 0.

$$\begin{aligned} M_2 &= x + \neg y + z \\ M_4 &= \neg x + y + z \\ M_7 &= \neg x + \neg y + \neg z \end{aligned}$$

Daraus folgt, dass die Eingangsvariablen, die gleich 1 sind, im Maxterm invertiert vorkommen müssen. Der Maxterm ist nicht invertiert und enthält die Eingangsvariablen, die im Eingangsvektor den Wert 0 haben. Da der Funktionswert nur dann 0 sein kann, wenn mindestens einer der Maxterme gleich 0 ist, kann die gesamte Funktion nun durch die Konjunktion der Maxterme dargestellt werden. Für unser Beispiel werden wir die kanonische konjunktive Normalform (KKNF) verwenden:

$$F_2 = M \cdot M_4 \cdot M_7 = (x + \neg y + z) \cdot (\neg x + y + z) \cdot (\neg x + \neg y + \neg z)$$

Die KKNF ist auch als „Produkt von Maxtermen“ zu bezeichnen.

Mithilfe der KKNF und DKNF kann für jede beliebige Boolesche Funktion ein Boolescher Ausdruck und ein Schaltplan aus der Wahrheitstabelle erstellt werden.

BEISPIEL KDNF UND KKNF EINER BOOLESCHEN FUNKTION

Die Wahrheitstabelle für die Funktion V ist in folgender Tabelle gegeben. Bitte schreiben Sie die KDNF und die KKNF der Funktion. Welche Form eignet sich am besten für eine Hardware-Implementierung und wie sieht der entsprechende Schaltplan aus?

Tabelle 10: Wahrheitstabelle der Funktion V

x	y	z	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Quelle: Moustafa Nawito, 2022.

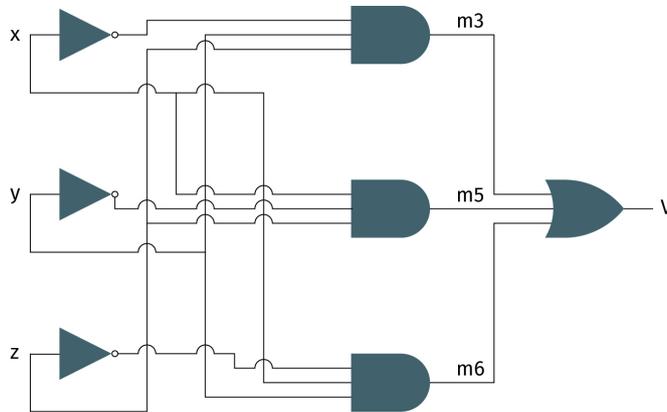
Lösung:

$$\text{DKNF: } m_3 + m_5 + m_6 = x'y z + x y' z + x y z'$$

$$\text{KKNF: } M_0 M_1 M_2 M_4 M_7 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)(x' + y' + z')$$

Die DKNF eignet sich am besten für eine Gatter-Implementierung, da sie deutlich weniger Terme hat und damit weniger Hardware benötigt.

Abbildung 9: Schaltplan der Funktion V



Quelle: Moustafa Nawito, 2022.

2.3 Karnaugh-Veitch-Diagramm

Obwohl die DKNF und die KKNF die Darstellung beliebiger Boolescher Funktionen als algebraische Ausdrücke und damit als Schemata von Logikgattern erlauben, ist das Ergebnis nicht optimiert. Mit anderen Worten, die Funktion kann weiter reduziert und vereinfacht werden, um eine möglichst praktikable Form zu erhalten, die eine kosteneffiziente Hardware-Realisierung gewährleistet. Eine sehr wichtige Technik zur Vereinfachung Boolescher Funktionen basiert auf den sogenannten Karnaugh-Veitch-Diagrammen (im Englischen oft als „K-maps“ bezeichnet). Sie wurden 1953 von Maurice Karnaugh als Verbesserung des Veitch-Diagramms von Edward W. Veitch aus dem Jahr 1952 eingeführt.

Jedes Quadrat auf einer K-Map, einer aus Quadraten bestehenden Grafik, stellt ein Minterm der zu reduzierenden Funktion dar. Da jede Boolesche Funktion als Summe ihrer Minterme geschrieben werden kann, lässt sich eine Boolesche Funktion auf ein Diagramm visuell durch den Bereich identifizieren, der von den Quadraten begrenzt wird, deren Minterme die Funktion bilden. Tatsächlich zeigt das Diagramm eine visuelle Darstellung jedes denkbaren Standardformausdrucks einer Funktion. Man kann verschiedene algebraische Ausdrücke für dieselbe Funktion erzeugen, indem man verschiedene Muster identifiziert; der einfachste kann dann ausgewählt werden. Die von dem Diagramm erzeugten vereinfachten Ausdrücke können nur eine von zwei konventionellen Formen annehmen:

- die Summe von Produkten (vereinfachte KDNF) oder
- das Produkt von Summen (vereinfachte KKNF).

Es wird davon ausgegangen, dass ein algebraischer Ausdruck mit den wenigsten Literalen, die in jedem Term möglich sind, der einfachste algebraische Ausdruck ist. Diese Formel erzeugt einen Schaltplan mit der geringsten Anzahl von Gattern und Eingängen in jedes Gatter. Der einfachste Ausdruck ist nicht ausschließlich einmalig zu finden: Manchmal ist es möglich, zwei oder mehr Ausdrücke zu erhalten, die die Reduktionsanforderungen erfüllen. In diesem Fall ist jede Antwort angemessen.

Die folgende Abbildung zeigt das Diagramm für eine Funktion mit vier binären Variablen (w, x, y und z). Die Minterme und die Verbindung zwischen den Quadraten und den vier Variablen sind hervorzuheben. In der Gray-Code-Sequenz, die zur Nummerierung der Zeilen und Spalten verwendet wird, ändert sich der Wert zwischen zwei aufeinanderfolgenden Zeilen oder Spalten nur um eine Ziffer. Das Mintermeintervall für jedes Quadrat lässt sich durch Verkettung der Zeilennummer und der Spaltennummer ermitteln.

Abbildung 10: KV-Diagramm für eine Funktion mit vier Variablen

	C'		C		
A'	A'B'C'D'	A'B'C'D	A'B'CD	A'B'CD'	B'
	m0	m1	m3	m2	
A	A'BC'D'	A'BC'D	A'BCD	A'BCD'	B
	m4	m5	m7	m6	
A	ABC'D'	ABC'D	A'B'C'D'	ABCD'	
	m12	m13	m15	m14	
A	AB'C'D'	AB'C'D	AB'CD	AB'CD'	B'
	m8	m9	m11	m10	
	D'	D	D'		

Quelle: Moustafa Nawito, 2022.

Die Verkettung der Zahlen der zweiten Zeile (01) und der zweiten Spalte (01) ergibt beispielsweise die Binärzahl 0101, die die binäre Darstellung der Zahl fünf ist. Folglich steht das Quadrat in der zweiten Spalte und dritten Zeile für das Intervall m5.

Nebeneinander liegende Quadrate werden als benachbarte Quadrate bezeichnet. Außerdem wird davon ausgegangen, dass das Diagramm auf einer Fläche liegt, deren benachbarte Quadrate durch die obere, untere, rechte und linke Kante gebildet werden, die sich gegenseitig berühren. So bilden beispielsweise m0 und m2 sowie m3 und m11 benachbarte Quadrate. Eine Untersuchung des Diagramms mit vier Variablen macht es einfach, die Anordnung der benachbarten Quadrate zu erkennen, die während des gesamten Vereinfachungsprozesses hilfreich ist:

- Ein Term mit vier Literalen wird durch einen Begriff mit einem Quadrat pro Minterm dargestellt.
- Zwei benachbarte Quadrate offerieren einen Term mit drei Literalen.

- Ein Term mit zwei Literalen wird durch vier benachbarte Quadrate dargestellt.
- Acht benachbarte Quadrate repräsentieren einen Term mit einem Literal.
- Ein Term, der immer gleich 1 ist, wird durch 16 benachbarte Quadrate dargestellt.

Keine andere Anordnung der Quadrate kann die Funktion vereinfachen.

BEISPIEL VEREINFACHUNG BOOLESCHER FUNKTIONEN MITTELS KV-DIAGRAMMEN

Eine DKNF der Funktion Y ist gegeben als

$$Y(A, B, C, D) = \sum(m_0, m_1, m_2, m_4, m_5, m_6, m_8, m_9, m_{12}, m_{13}, m_{14})$$

Vereinfachen Sie die Funktion mithilfe des KV-Diagramms.

Lösung:

Folgende Tabelle zeigt die Wahrheitstabelle der Funktion Y und die folgende Abbildung stellt das entsprechende KV-Diagramm dar.

Tabelle 11: Wahrheitstabelle der Funktion

$$Y(A, B, C, D) = \sum(m_0, m_1, m_2, m_4, m_5, m_6, m_8, m_9, m_{12}, m_{13}, m_{14})$$

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1

A	B	C	D	Y
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Quelle: Moustafa Nawito, 2022.

Abbildung 11: KV-Diagramm der Funktion

$$Y(A, B, C, D) = \sum(m_0, m_1, m_2, m_4, m_5, m_6, m_8, m_9, m_{12}, m_{13}, m_{14})$$

	\bar{C}		C		
\bar{A}	1	1	0	1	\bar{B}
	1	1	0	1	B
A	1	1	0	1	
	1	1	0	0	\bar{B}
	\bar{D}	D	\bar{D}		

Quelle: Moustafa Nawito, 2022.

Es ist notwendig, ein Diagramm mit vier Variablen zu verwenden, da die Funktion vier Variablen enthält. Das Diagramm zeigt die Positionen der Minterme mit Einsen. Das Literal C' kann durch Kombination der acht benachbarten Quadrate, die jeweils mit einer 1 markiert sind, gebildet werden. Es ist notwendig, die verbleibenden drei Einsen auf der rechten Seite als zwei oder vier benachbarte Quadrate zu verbinden, anstatt sie zu kombinieren, um einen reduzierten Term zu bilden. Die Anzahl der Literale im Term nimmt mit zunehmender Anzahl der zusammengefügtten Quadrate ab. Der Term $A'D'$ ergibt sich in diesem Beispiel aus der Addition der beiden obersten 1en auf der rechten Seite und der beiden obersten Einsen auf der linken Seite. Es ist wichtig zu beachten, dass dasselbe Quadrat mehrmals verwendet werden kann. Das Quadrat in der dritten Reihe und vierten Spalte, das links liegt, wird mit der Zahl 1 bezeichnet (Quadrat 1110). Wir kombinieren dieses Quadrat mit den zuvor verwendeten Quadraten, um eine Fläche aus vier benachbarten Quadraten zu bilden, anstatt es allein zu nehmen, was zu einem Term mit vier Literalen führen würde. Der Term BD' setzt sich aus diesen Quadraten zusammen, die auch die beiden mittleren Zeilen und die beiden Endspalten bilden. Die vereinfachte Funktion kann wie folgt ausgedrückt werden: $Y = C' + A'D' + BD'$.

Folgende Abbildung zeigt eine Zusammenfassung der KV-Diagramme für Funktionen mit zwei bis vier Variablen.

EXKURS

An dieser Stelle ist es wichtig zu erwähnen, dass die KV-Diagramm-Methode für die manuelle Vereinfachung von Funktionen gedacht ist. Daher findet sie für Funktionen mit mehr als vier Variablen nur sehr selten eine praktische Anwen-

dung. Für größere Funktionen werden spezielle Computeralgorithmen eingesetzt, um die Komplexität zu reduzieren und einen vereinfachten Ausdruck zu erhalten.

Ein solcher Algorithmus basiert auf dem Verfahren nach Quine und McCluskey. Dieses Thema sprengt den Rahmen dieses Textes und wird nicht weiter behandelt. Interessierte Leser:innen können mehr über das Thema in Fricke (2021, S. 60–63) erfahren.

Abbildung 12: KV-Diagramme für Funktionen mit zwei bis vier Variablen

	B'	B
A'	0	1
A	2	3

	B'		B	
A'	0	1	3	2
A	4	5	7	6
	C'	C	C'	

	C'		C		
A'	0	1	3	2	B'
	4	5	7	6	B
A	12	13	15	14	
	8	9	11	10	B'
	D'	D	D'		

Quelle: Moustafa Nawito, 2022.

Primimplikanten

Bei der Auswahl benachbarter Quadrate für ein Diagramm ist darauf zu achten, dass

1. alle Minterme der Funktion abgedeckt sind, wenn die Quadrate kombiniert werden,
2. die Anzahl der Terme des Ausdrucks so gering wie möglich gehalten wird und
3. es keine redundanten Terme gibt (d. h. Minterme, die bereits von anderen Termen abgedeckt werden).

Gelegentlich gibt es zwei oder mehr Ausdrücke, die die Bedingungen für eine Vereinfachung erfüllen. Um den Prozess der Zusammenführung von Quadraten im Diagramm systematischer zu gestalten, führen wir das Konzept der Primimplikanten ein.

- Ein **Primimplikant** ist ein Produktausdruck, der durch die Verbindung der größten Anzahl von zusammenhängenden Diagrammquadraten entsteht.
- Ein Primimplikant gilt als wesentlich, wenn er der einzige Primimplikant ist, der alle Minterme des Quadrats erfüllen kann.

Primimplikant

Ein Produktausdruck, der durch die Verbindung der größten Anzahl von zusammenhängenden KV-Diagrammquadraten entsteht, nennt man Primimplikant.

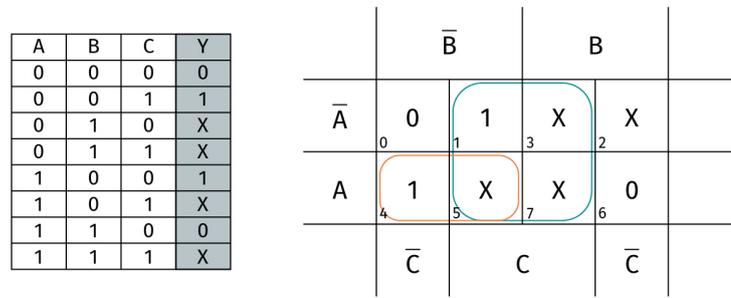
Kombiniert man alle möglichen Höchstzahlen von Quadraten, erhält man die Primimplikanten einer Funktion aus dem Diagramm. Dies bedeutet, dass eine einzelne 1, die nicht an andere Einsen im Diagramm angrenzt, eine Primimplikante darstellt. Liegen keine vier aufeinanderfolgenden Quadrate dazwischen, bilden zwei benachbarte Einsen eine Pri-

mimplikante. Wenn sie nicht in einer Gruppe von acht benachbarten Quadraten enthalten sind, bilden vier benachbarte Einsen eine Primimplikante usw. Die Anzahl der Primimplikanten, die jedes mit einer 1 bezeichnete Quadrat abdecken, kann zur Bestimmung der erforderlichen Primimplikanten verwendet werden. Wenn der Primimplikant der einzige Primimplikant ist, der das Minimum abdeckt, dann ist er entscheidend.

Don't Care-Terme

Es gibt Situationen, in denen die Angaben zu einer Funktion unvollständig sind. Dann können einige Funktionswerte nach dem Zufallsprinzip ausgewählt werden. Im KV-Diagramm sind sie mit einem d oder x (don't care – der Wert ist egal) gekennzeichnet, und man kann die Funktion immer noch minimieren, indem man „Don't Care“-Terme verwendet. Als Beispiel betrachten wir die Funktion in folgender Abbildung.

Abbildung 13: Unvollständige Funktion mit Don't Care Bedingungen



Quelle: Moustafa Nawito, 2022.

Da für bestimmte Eingänge die Ausgänge entweder 0 oder 1 sein können oder, mit anderen Worten, dies für den Betrieb dieser Schaltung nicht relevant ist (d. h. es uns „egal“ ist – „don't care“), können diese Minterme als 1 betrachtet und bei der Minimierung der Funktion verwendet werden. Die Funktion ist wie folgt auszuführen:

$$Y(A, B, C) = \sum(m1, m2) + d(m3, m4, m5, m7)$$

wobei die Don't Care Minterme im Term d (m3,m4,m5,m7) dargestellt sind.

Der resultierende Ausdruck ist $Y = AB' + C$

Vereinfachung als Produkt von Summen

Im vorherigen Beispiel wurde die Form der Summe der Produkte verwendet, um die minimierten Booleschen Funktionen auszudrücken, die aus dem KV-Diagramm gewonnen wurden. Die Summenproduktform kann mit einer kleinen Anpassung hergestellt werden. Die Minterme der Funktion sind durch die Einsen dargestellt, die in die Quadrate der Karte gesetzt wurden. Das Komplement der Funktion wird durch die Minterme dargestellt, die in der typischen Summenproduktform einer Funktion nicht vorhanden sind. Diese Beobach-

tung zeigt, dass die Quadrate auf dem Diagramm, die nicht mit Einsen markiert sind, das Komplement einer Funktion repräsentieren. Das Komplement der Funktion F (d. h. F') kann einfacher durch eine Produktsummenformel ausgedrückt werden, wenn die leeren Quadrate mit Nullen markiert und zu gültigen Nachbarquadraten zusammengefasst werden. Als Ergebnis des Satzes von DeMorgan erhalten wir die Funktion F in Form des Summenprodukts als Komplement von F' .

Als Beispiel betrachten wir die Funktion $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$, dessen KV-Diagramm in folgender Abbildung dargestellt ist. Aus dem Diagramm kann die Funktion minimiert und als $F = B'D' + B'C' + A'C'D$ ausgedrückt werden. Kombiniert man die mit 0 gekennzeichneten Quadrate, wie im Diagramm dargestellt, so erhält man die vereinfachte komplementäre Funktion: $F' = AB + CD + BD'$. Wendet man den Satz von DeMorgan an (indem man jedes Literal ergänzt), so erhält man die vereinfachte Funktion in Form des Produkts in der Summenform: $F = (A' + B')(C' + D')(B' + D)$

Abbildung 14: KV-Diagramm der Funktion $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$

	\bar{C}	C	
\bar{A}	1	1	0
A	1	1	0
	1	1	0
	1	1	0

\bar{B}
 B
 \bar{B}

\bar{D}
 D
 \bar{D}

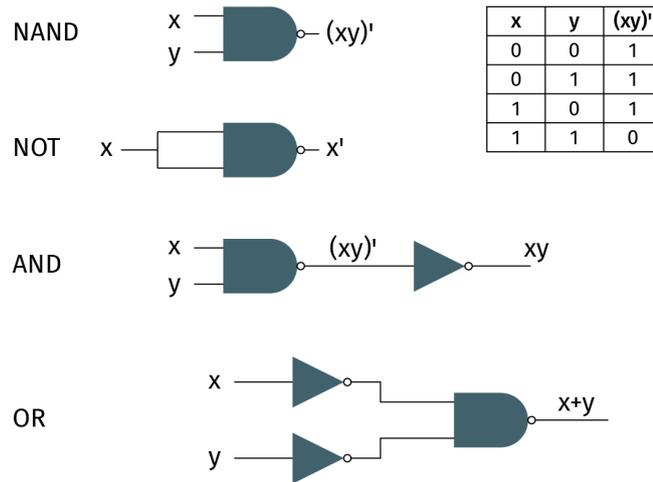
Quelle: Moustafa Nawito, 2022.

2.4 Substitution durch NOR-/NAND-Gatter

Die NAND-Verknüpfung ist definiert als AND gefolgt von NOT oder als „NOT eines AND“ und die NOR-Verknüpfung ist definiert als OR gefolgt von einem NOT oder als „NOT eines OR“. Anstelle von AND- und OR-Gattern werden in der Regel NAND- oder NOR-Gatter zum Aufbau digitaler Schaltungen verwendet. NAND- und NOR-Gatter sind die grundlegenden Gatter, die in allen IC-Digitallogikfamilien verwendet werden und lassen sich einfacher aus elektronischen Bauteilen zusammensetzen. Aufgrund der Bedeutung von NAND- und NOR-Gattern für die Entwicklung digitaler Schaltungen wurden Regeln und Verfahren entwickelt, um Boolesche Funktionen, die durch AND, OR und NOT ausgedrückt werden, in vergleichbare NAND- und NOR-Logikdiagramme umzuwandeln.

Die folgende Abbildung veranschaulicht das Schaltsymbol und die Wahrheitstabelle des NAND-Gatters. Wenn man die Theoreme und Postulate der Booleschen Algebra und die Wahrheitstabelle des NAND verwendet, kann man die Funktionen NOT, AND und OR nur mit NAND realisieren. Da alle logischen Ausdrücke, egal wie kompliziert sie sind, durch diese drei Operationen dargestellt werden können, folgt daraus, dass jede digitale Schaltung nur mit NAND realisiert werden kann.

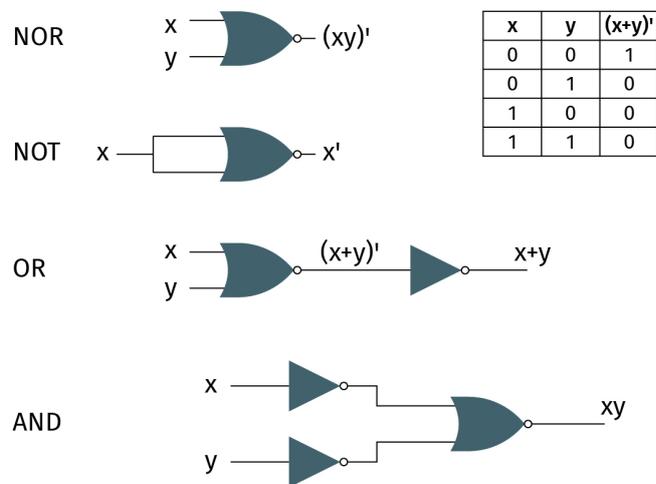
Abbildung 15: Realisierung von NOT, AND und OR mittels NAND



Quelle: Moustafa Nawito, 2022.

Analog dazu sind das Symbol und die Wahrheitstabelle von NOR in der nachfolgenden Abbildung dargestellt. Wie man sieht, ist es auch möglich, NOT, AND und OR nur mit NOR umzusetzen. Folglich kann jede digitale Schaltung nur mit NOR-Gattern realisiert werden.

Abbildung 16: Realisierung von NOT, AND und OR mittels NOR



Quelle: Moustafa Nawito, 2022.



ZUSAMMENFASSUNG

Die Boolesche Algebra bildet die mathematische und formale Grundlage für die Analyse und den Entwurf digitaler Systeme. Die binäre Logik ist eine Form der Booleschen Algebra, die sich mit der Verarbeitung von Variablen befasst, die nur zwei Werte annehmen können, daher auch der Name zweiwertige Algebra.

Um logische Beziehungen und Operationen auszudrücken, werden Boolesche Funktionen definiert, die auf Booleschen Variablen oder Literalen operieren und als algebraische Ausdrücke von Termen ausgedrückt werden, die aus diesen Literalen bestehen.

Um eine effiziente und kostengünstige Hardware-Realisierung von Booleschen Funktionen zu erreichen, ist es wünschenswert, diese Ausdrücke so weit wie möglich zu minimieren oder zu vereinfachen. Dies kann mithilfe von Try-and-Error-Manipulationen geschehen, die auf den grundlegenden Theorien und Postulaten der binären Logik basieren. Eine effizientere manuelle Methode ist die Verwendung von KV-Diagrammen und der Ausdruck von Funktionen in vereinfachten KDNF und KKNF.

Für eine tatsächliche Hardware-Realisierung werden NAND- und NOR-Gatter verwendet, da sie einfach herzustellen sind und NOT-, AND- und OR-Verknüpfungen und damit jede logische Funktion realisieren können.

LEKTION 3

SCHALTNETZE (KOMBINATORISCHE LOGIK)

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die verschiedenen Arten von Logikgattern zu erkennen und anzuwenden.
- die Merkmale kombinatorischer Logik zu benennen.
- die Analyse und das Design von Schaltnetzen durchzuführen.
- die Eigenschaften wichtiger kombinatorischer Schaltungen zu erkennen.

3. SCHALTNETZE (KOMBINATORISCHE LOGIK)

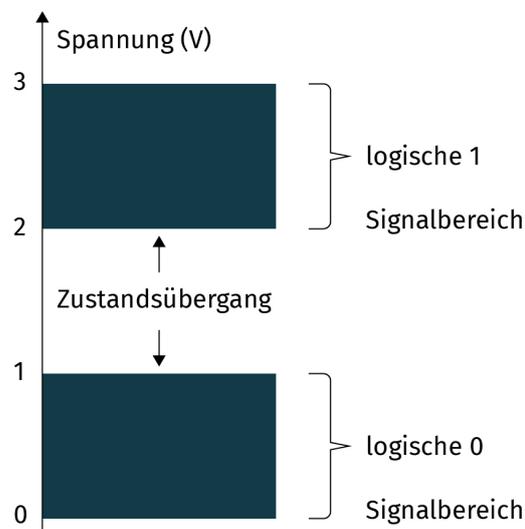
Einführung

Schaltnetze oder kombinatorische Logikschaltungen entstehen durch die direkte Verbindung von Logikgattern und sind eine der beiden Hauptarten von logischen Systemen, die andere ist die sequenzielle Logikschaltung. Die Rolle der kombinatorischen Schaltungen kann gar nicht hoch genug eingeschätzt werden, da sie in jeder einzelnen digitalen Schaltung vorhanden sind. In dieser Lektion werden die Grundlagen dieser wichtigen Schaltungs-kategorie vorgestellt, beginnend mit den Eigenschaften von digitalen Logikgattern und der Erörterung häufig verwendeter kombinatorischer Logikblöcke. Ein wichtiges Ziel dieser Lektion ist es, methodische Analyse- und Entwurfstechniken für kombinatorische Schaltungen zu vermitteln.

3.1 Logikgatter

Elektronische Schaltungen, sogenannte Logikgatter, verarbeiten ein oder mehrere Eingangssignale, um ein Ausgangssignal zu erzeugen. Elektrische Signale, wie z. B. Spannungen oder Ströme, können in einem digitalen System als einer von zwei identifizierbaren Werten interpretiert werden: '0' oder '1' (Logikpegel werden oft als '0' und '1' geschrieben, um sie von den Zahlen 0 und 1 zu unterscheiden.). Analoge elektrische Signale, z. B. Spannungen oder Ströme, existieren und haben Werte in einem bestimmten kontinuierlichen Bereich, z. B. 0 bis 3 V. Spannungsgesteuerte Logikschaltungen reagieren auf zwei verschiedene Spannungspegel, von denen jeder eine binäre Variable darstellt, die entweder einer logischen 1 oder einer logischen 0 entspricht. Zur Veranschaulichung könnte ein spezifisches digitales System logisch 0 als ein Signal von 0 V und logisch 1 als ein Signal von 3 V definieren. Jeder Spannungspegel hat einen praktischen Bereich, wie in folgender Abbildung dargestellt. Die Eingangsanschlüsse digitaler Schaltungen nehmen Binärsignale an, die in den zulässigen Bereich fallen, und antworten an den Ausgangsanschlüssen mit Binärsignalen, die in denselben Bereich fallen. Nur bei einem Zustandsübergang wird der mittlere Bereich zwischen den erlaubten Bereichen durchquert. Durch das Senden von Binärsignalen über verschiedene Anordnungen von Logikgattern, wobei jedes Signal eine bestimmte Binärvariable bezeichnet, kann jede gewünschte Information für Berechnungen oder Steuerungen verarbeitet werden. Das physikalische Signal wird entweder als '0' oder als '1' interpretiert, wenn es innerhalb eines bestimmten Bereichs liegt.

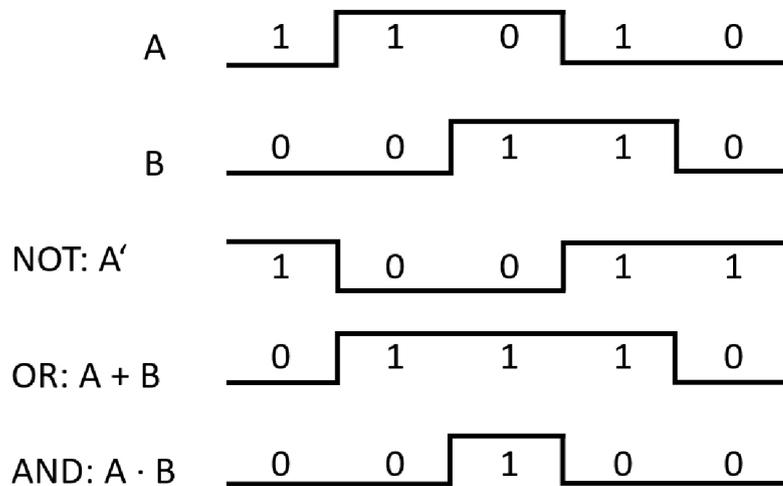
Abbildung 17: Beispiel für Spannungs- und Logikpegel für 3 V Logikgatter



Quelle: Moustafa Nawito, 2022.

Gatter sind Hardwarekomponenten, die Ausgangssignale erzeugen, die einer logischen 1 oder einer logischen 0 entsprechen. Es gibt vier mögliche Zustände für die Eingangssignale x und y an den AND- und OR-Gattern: 00, 10, 11 oder 01. Die nachfolgende Abbildung zeigt diese Eingangssignale zusammen mit den zugehörigen Ausgangssignalen der einzelnen Gatter. Die Zeitdiagramme repräsentieren die hypothetische Reaktion der einzelnen Gatter auf die vier Eingangssignalkombinationen. Die vertikale Achse des Zeitdiagramms stellt das Signal dar, wie es zwischen den beiden möglichen Spannungspegeln wechselt, während die horizontale Achse die Zeit angibt. Die Übergänge zwischen den logischen Werten erfolgen zwar schnell, aber nicht sofort. Die logische 0 wird durch den niedrigen Pegel repräsentiert, die logische 1 durch den hohen Pegel. Wenn beide Eingangssignale logisch 1 sind, reagiert das AND-Gatter mit einem Ausgangssignal von logisch 1. Wenn eines der Eingangssignale eine logische 1 ist, reagiert das OR-Gatter mit einem Ausgangssignal von logisch 1. Der Begriff „Inverter“ wird häufig zur Beschreibung des NOT-Gatters verwendet. Die Signalantwort des Zeitdiagramms, die zeigt, wie das Ausgangssignal den logischen Sinn des Eingangssignals invertiert, macht deutlich, warum dieses Signal diesen Namen trägt.

Abbildung 18: Eingangs- und Ausgangssignale für NOT-, OR- und AND-Gatter



Quelle: Moustafa Nawito, 2022.

Wie bereits erwähnt, ist die Anzahl der möglichen logischen Operationen, die für n binäre Variablen durchgeführt werden können, gleich 2^n . Das bedeutet, dass es für zwei Variablen x, y 16 mögliche Operationen gibt, die ausgeführt werden können. In folgender Abbildung sind alle möglichen Operationen für zwei Variablen aufgeführt, wobei die Wahrheitstabelle, der algebraische Ausdruck und das entsprechende Logikgattersymbol sowohl in DIN- als auch in US-Form dargestellt sind. Wie man sieht, werden nicht alle Operationen durch logische Gatter realisiert oder dargestellt. Zum Beispiel ergeben die Operationen F0 und F15 beide eine Konstante, eine logische 0 bzw. eine logische 1. In einer realen Schaltung würde dies einfach bedeuten, dass der Ausgang F0 mit der niedrigsten Spannung (in der Regel die Masse) und der Ausgang F15 mit der höchsten Spannung (in der Regel die Versorgungsspannung) verbunden wird.

Abbildung 19: Alle Booleschen Funktionen für zwei Variablen

Wahrheitstabelle					Boolesche Funktion	Operationssymbol	Name	Bedeutung	Schaltsymbol (DIN 40900)	Schaltsymbol (US - international)
x	y	0	1	1						
F0	0	0	0	0	$F0 = 0$		Null	Binäre Konstante 0		
F1	0	0	0	1	$F1 = xy$	$x \cdot y$	AND	x und y		
F2	0	0	1	0	$F2 = xy'$	x/y	Inhibition	x aber nicht y		
F3	0	0	1	1	$F3 = x$		Identität	x		
F4	0	1	0	0	$F4 = x'y$	y/x	Inhibition	y aber nicht x		
F5	0	1	0	1	$F5 = y$		Identität	y aber nicht x		
F6	0	1	1	0	$F6 = xy' + x'y$	$x \oplus y$	EXOR	x oder y, aber nicht beide		
F7	0	1	1	1	$F7 = x + y$	$x + y$	OR	x oder y		
F8	1	0	0	0	$F8 = (x + y)'$	$x \downarrow y$	NOR	NOT - OR		
F9	1	0	0	1	$F9 = xy + x'y'$	$(x \oplus y)'$	EXNOR (Äquivalenz)	x ist gleich y		
F10	1	0	1	0	$F10 = y'$	y'	Komplement	nicht y		
F11	1	0	1	1	$F11 = x + y'$	$y \supset x$	Implikation	falls y, dann x		
F12	1	1	0	0	$F12 = x'$	x'	Komplement	nicht x		
F13	1	1	0	1	$F13 = x' + y$	$x \supset y$	Implikation	falls y, dann y		
F14	1	1	1	0	$F14 = (xy)'$	$x \uparrow y$	NAND	NOT - AND		
F15	1	1	1	1	$F15 = 1$		Eins	Binäre Konstante 1		

Quelle: Moustafa Nawito, 2022.

Gatter mit mehr als zwei Eingängen

Alle Gatter in obiger Abbildung können erweitert werden, um mehr als zwei Eingänge zu verarbeiten. Wenn die binäre Operation, die ein Gatter darstellt, sowohl kommutativ als auch assoziativ ist, kann das Gatter so erweitert werden, dass es einfach mehr Eingänge enthält. Diese beiden Eigenschaften gelten für die Operationen AND und OR, wie sie in der Booleschen Algebra definiert sind. Die Gleichungen $x + y = y + x$ (kommutative Eigenschaft) und $(x + y) + z = x + (y + z) = x + y + z$ (assoziative Eigenschaft) für die OR-Funktion zeigen, dass die Gattereingänge vertauscht werden können und dass die OR-Funktion auf drei oder mehr Variablen erweiterbar ist. Die NAND- und NOR-Funktionen sind kommutativ. Wenn die Definition der Operation leicht verändert wird, können ihre Gatter so erweitert werden, dass sie mehr als zwei Eingänge haben. Die fehlende Assoziativität der NAND- und NOR-Operatoren ist das Problem.

Um dieses Problem zu umgehen, definieren wir das Mehrfach-NOR- (oder NAND-)Gatter als ein komplementäres OR- (oder AND-)Gatter. In Anbetracht dieser Tatsache haben wir

$$x \downarrow y \downarrow y = (x + y + z)' \tag{3.1}$$

$$x \uparrow y \uparrow y = x \cdot y \cdot z \tag{3.2}$$

Beim Schreiben von kaskadierten NOR- und NAND-Operationen muss die rechte Klammer verwendet werden, um die richtige Reihenfolge der Gatter zu kennzeichnen.

Ungerade Funktion
Eine logische Funktion, deren Ausgang nur dann wahr ist, wenn eine ungerade Anzahl von Eingängen wahr ist, wird ungerade Funktion genannt.

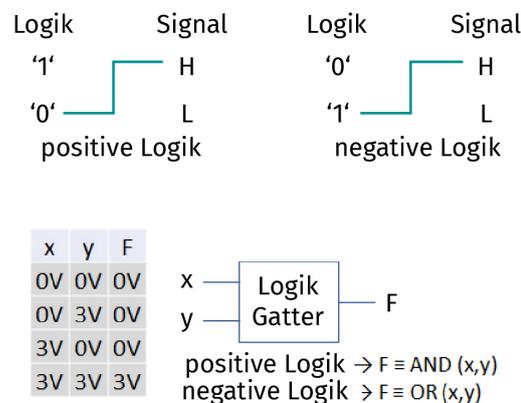
Die Exklusiv-OR- und Äquivalenzgatter können auf mehr als zwei Eingänge erweitert werden und sind sowohl kommutativ als auch assoziativ. In der Hardware sind Exklusiv-OR-Gatter mit mehreren Eingängen jedoch nicht üblich. Vielmehr werden häufig verschiedene Arten von Gattern verwendet, um sogar Funktionen mit zwei Eingängen zu erstellen. Als **ungerade Funktion** ergibt das Exklusiv-OR den Wert 1, wenn die Eingangsvariablen eine ungerade Anzahl von Einsen enthalten. Diese Eigenschaft wird auch als „Parität“ (engl. parity) bezeichnet und wird häufig bei der Fehlererkennung verwendet.

Positive und negative Logik

Das binäre Signal eines Gatters hat an seinen Eingängen und Ausgängen einen von zwei Werten, mit Ausnahme des Übergangs. Die logische 1 wird durch einen Signalwert repräsentiert, die logische 0 durch den anderen. Es gibt zwei alternative Zuordnungen von Signalpegel zu Logikwert, da zwei Signalwerte zwei Logikwerten zugeordnet sind, wie in folgender Abbildung veranschaulicht. H und L stehen für höhere (engl. high) bzw. niedrigere (engl. low) Signalpegel. Ein positives logisches System kann definiert werden, indem der hohe Pegel H für die logische 1 gewählt wird. Ein negatives Logiksystem ist definiert, insofern die logische 1 durch den niedrigen Pegel L dargestellt wird. Da beide Signale positiv oder negativ sein können, sind die Begriffe positiv und negativ verwirrend. Die Art der Logik wird durch die Zuordnung von Logikwerten zu den relativen Amplituden der beiden Signalpegel bestimmt und nicht durch die tatsächlichen Werte der Signale selbst. Signalwerte, wie H und L , werden verwendet, um digitale Hardware-Gatter zu definieren. Die Wahl der positiven oder negativen logischen Polarität bleibt dem Benutzenden überlassen.

Betrachten wir zum Beispiel die Wahrheitstabelle des in der unteren Abbildung dargestellten allgemeinen digitalen Gatters. Hier werden die logischen Pegel in Form von 3 V (H) und 0 V (L) impliziert. Dies stellt die Hardware-Konstruktion des Gatters dar, jedoch hängt die Definition des Gattertyps von der Wahl der logischen Bedeutung von 3 V und 0 V ab. Für den (normalen) positiven Logikfall, d. h. $H = '1'$ und $L = '0'$, stellt dieses Gatter eine AND-Funktion dar. Im negativen logischen Fall, d. h. $H = '0'$ und $L = '1'$, führt dieses Gatter eine OR-Funktion aus.

Abbildung 20: Positive und negative Logik



Quelle: Moustafa Nawito 2022.

Logikfamilien

Heutzutage werden digitale Schaltungen in Hardware als integrierte Schaltungen realisiert. Die Umsetzung digitaler Hardware mit diskreten Bauelementen ist sehr selten, ja sogar fast unbekannt, weil die schiere Anzahl der Logikgatter, die in digitalen Anwendungen verwendet werden, eine sehr hohe Anzahl von Gattern erfordert. Diese integrierten Schaltungen können nach ihrer Komplexität und ihrem Integrationsgrad klassifiziert werden:

- Bauelemente, die Small-Scale-Integration (SSI) verwenden, enthalten mehrere unabhängige Gates in einem einzigen Gehäuse. Die Ein- und Ausgänge der Gates sind direkt mit den Pins des Gehäuses verdrahtet. Die Anzahl der verfügbaren Pins im IC begrenzt die Anzahl der Gates, die oft weniger als zehn beträgt.
- Bauelemente mit Medium-Scale-Integration (MSI) besitzen zwischen zehn und 1.000 komplexe Gatter in einem einzigen Gehäuse. Sie führen in der Regel bestimmte einfache digitale Aufgaben aus. Decodierer, Addierer und Multiplexer sowie Register und Zähler sind MSI-Funktionen.
- Bei Bauelementen, die Large-Scale-Integration (LSI) verwenden, sind Tausende von Gattern in einem einzigen Gehäuse untergebracht. Dazu gehören digitale Systeme wie CPUs, Speicherchips und programmierbare Logikbausteine.
- Bauelemente, die die Very Large Scale Integration (VLSI), bzw. Ultra Large Scale Integration (ULSI) verwenden, besitzen Millionen bzw. Milliarden von Gattern in einem einzigen Gehäuse. Große Speicherarrays und hochentwickelte Mikrocomputerschaltungen sind zwei Beispiele dafür. VLSI-Bauteile haben die Technologie für die Entwicklung von Computersystemen verändert, da sie sehr klein und kostengünstig sind und es dem Entwickler ermöglichen, Strukturen zu bauen, die vorher nicht möglich waren.

Digitale integrierte Schaltungen werden nicht nur nach ihrer Komplexität oder logischen Funktionsweise eingeteilt, sondern auch nach der jeweiligen Schaltungstechnologie, zu der sie gehören. Eine digitale Logikfamilie ist die Bezeichnung für eine solche Schaltungs-

technologie. Jede Logikfamilie verfügt über eine einzigartige elektronische Grundschialtung, auf der immer kompliziertere digitale Schaltungen und Komponenten aufgebaut werden. Ein NAND-, NOR- oder Inverter-Gatter dient als Grundschialtung in jeder Technologie. Die Technik wird in der Regel nach den elektronischen Bauteilen benannt, die zum Aufbau der Grundschialtung verwendet werden. Digitale integrierte Schaltungen sind in einer Vielzahl von Logikfamilien auf den Markt gekommen. Die zumeist angewendeten sind:

- **TTL** – Transistor-Transistor Logic,
- **ECL** – Emitter-Coupled Logic,
- **MOS** – Metal-Oxide Semiconductor sowie
- **CMOS** – Complementary Metal-Oxide Semiconductor.

TTL gibt es seit 50 Jahren und wird als Standard angesehen. Bei Systemen, die einen Hochgeschwindigkeitsbetrieb erfordern, bietet ECL einen Vorteil. Während CMOS in Systemen bevorzugt wird, die einen niedrigen Stromverbrauch benötigen, wie z. B. Digitalkameras, Personal Media Player und andere mobile, tragbare Geräte, ist MOS für Schaltungen geeignet, die eine hohe Bauteildichte erfordern. Da ein niedriger Stromverbrauch für das VLSI-Design entscheidend ist, hat CMOS TTL und ECL als vorherrschende Logikfamilie verdrängt. Im Folgenden werden die wichtigsten Merkmale aufgeführt, die die Logikfamilien unterscheiden.

- „Fan-out“ beschreibt die maximale Anzahl von Standardlasten, die der Ausgang eines konventionellen Gatters ohne Beeinträchtigung der normalen Leistung unterstützen kann. Die Strommenge, die von einem Eingang eines anderen ähnlichen Gatters derselben Familie benötigt wird, wird üblicherweise zur Definition einer Standardlast verwendet.
- „Fan-in“ ist die Anzahl der Eingänge, die ein Gatter akzeptieren kann.
- Die vom Gatter verbrauchte Leistung, die von der Stromquelle stammt, wird als Verlustleistung bezeichnet.
- Die durchschnittliche Übergangszeit, die ein Signal benötigt, um vom Eingang zum Ausgang zu gelangen, ist als Ausbreitungsverzögerung deklariert.
- Die maximal hinzugefügte externe Rauschspannung wird als Rauschspanne definiert.

3.2 Verknüpfung von Gattern

Kombinatorische Schaltung

Eine logische Schaltung, deren Ausgang nur von aktuell vorhandenen Eingangswerten abhängt, wird kombinatorische Schaltung genannt.

Eine **kombinatorische Schaltung** oder ein Schaltnetz (beide Ausdrücke sind äquivalent) besteht aus logischen Gattern, deren Ausgänge immer nur durch die Kombination der Eingänge bestimmt werden, die gerade vorhanden sind. Sie setzt sich somit aus mehreren miteinander verbundenen Logikgattern zusammen. Kombinatorische Logikgatter wandeln binäre Informationen aus den bereitgestellten Eingangsdaten in die erforderlichen Ausgangsdaten um, indem sie auf die Werte der Signale an ihren Eingängen reagieren und den Wert des Ausgangssignals erzeugen. Ein Satz Boolescher Funktionen kann eine Operation logisch spezifizieren, die die Ausführung einer kombinatorischen Schaltung ermöglichen.

Die folgende Abbildung ist ein Blockschaltbild einer kombinatorischen Schaltung. Die m Ausgangsvariablen werden von der internen kombinatorischen Logikschaltung erzeugt und wandern an einen externen Ort, während die n binären Eingangsvariablen von einer externen Quelle stammen. Physikalisch gesehen ist jede Eingangs- und Ausgangsvariable ein analoges Signal, dessen Werte als binäre Signale decodiert werden, die eine logische 1 und eine logische 0 darstellen (Anmerkung: Logiksimulatoren zeigen nicht die tatsächlichen analogen Signale an; sie zeigen nur Nullen und Einsen an).

Abbildung 21: Blockschaltbild eines Schaltnetzes



Quelle: Moustafa Nawito, 2022.

Es gibt 2^n potenzielle binäre Eingabemöglichkeiten für n Eingangsvariablen. Jede Ausgangsvariable kann einen möglichen Wert für jede mögliche Kombination von Eingängen haben. Folglich kann eine Wahrheitstabelle, die die Ausgangswerte für jede Kombination von Eingangsvariablen auflistet, zur Spezifikation einer kombinatorischen Schaltung verwendet werden. Die Anwendung Boolescher Funktionen ist auch möglich, um kombinierte Schaltungen darzustellen, eine für jede Ausgangsvariable. Die n Eingangsvariablen werden genutzt, um jede Ausgangsfunktion zu implizieren.

Analyseverfahren

Um kombinatorische Schaltungen zu analysieren, muss die Funktion identifiziert werden, die die Schaltung implementiert. Ein mitgeliefertes Logikdiagramm dient als Ausgangspunkt für diese Aufgabe. Mithilfe einer Computersimulationssoftware, durch manuelles Auffinden der Wahrheitstabelle oder der Booleschen Funktionen kann die Analyse durchgeführt werden. Der erste Schritt bei der Analyse ist die Überprüfung, ob es sich bei der gelieferten Schaltung um eine kombinierte und nicht um eine sequenzielle Schaltung handelt. Das Diagramm einer kombinatorischen Schaltung besteht aus logischen Gattern ohne Rückkopplungskanäle oder Speicherkomponenten. Ein Rückkopplungspfad verbindet den Ausgang eines Gatters mit seinem Eingang, wobei der Ausgang des zweiten Gatters als ein Teil des Eingangs des ersten Gatters dient. Nachdem gewährleistet ist, dass das Logikdiagramm eine kombinatorische Schaltung darstellt, kann man zur Ermittlung der Wahrheitstabelle oder der Booleschen Ausgangsfunktionen übergehen. Wenn die Funktion der Schaltung untersucht werden soll, ist es erforderlich, das Verhalten der Schaltung anhand der resultierenden Booleschen Funktionen oder der Wahrheitstabelle zu verstehen. Dabei wird wie folgt vorgegangen, um die Booleschen Ausgangsfunktionen aus einem Logikdiagramm zu erhalten:

1. Man verwendet beliebige Symbole mit sinnvollen Namen, um alle Gatterausgänge zu identifizieren, die Funktionen von Eingangsvariablen sind. Die Booleschen Operationen für jeden Gatterausgang sollten bekannt sein.
2. Man gibt den zuvor beschrifteten Gattern und den Gattern, die eine Funktion der Eingangsvariablen sind, weitere beliebige Symbole. Für diese Gatter sollen die Booleschen Funktionen bestimmt werden.
3. Man fährt mit dem in Schritt zwei beschriebenen Verfahren fort, bis die Ausgänge der Schaltung erhalten sind.
4. Um die Booleschen Ausgangsfunktionen in Bezug auf die Eingangsvariablen zu erhalten, werden die zuvor festgelegten Funktionen wiederholt ersetzt.

Die vorgeschlagene Methode wird durch die Untersuchung der kombinierten Schaltung in der folgenden Abbildung veranschaulicht. Die Schaltung hat zwei binäre Ausgänge, $F1$ und $F2$, und drei binäre Eingänge, A , B und C . Verschiedene Gatter haben Zwischensymbole, um ihre Ausgänge zu identifizieren. $V1$ und $V2$ sind die Ausgänge von Gattern, die einfach von ihren Eingangsvariablen abhängen. Die Eingangsvariablen können direkt zur Berechnung des Ausgangs $F2$ verwendet werden. Die jeweiligen Booleschen Funktionen dieser drei Ausgänge lauten wie folgt:

$$F1 = A' + B' + C$$

$$V1 = (ABC)'$$

$$V2 = A + B + C$$

Anschließend werden die Ausgänge von Gattern betrachtet, die auf zuvor definierte Symbole wirken:

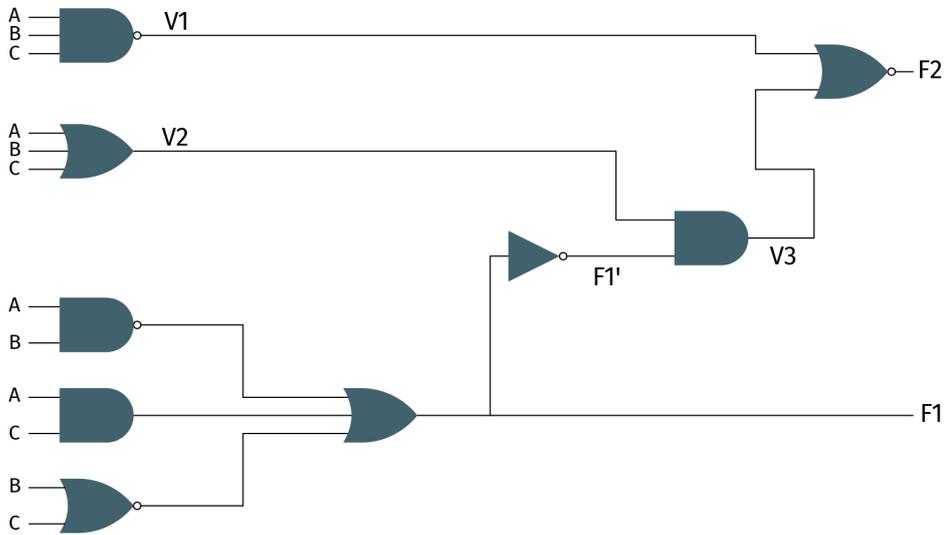
$$V3 = F1'V2$$

$$F2 = (V3 + V1)$$

Durch konsekutives Substitutionsverfahren kann gezeigt werden, dass $F2$ den folgenden Ausdruck aufweist:

$$F2 = ABC$$

Abbildung 22: Beispiel zur Analyse von Schaltnetzen



Quelle: Moustafa Nawito, 2022.

Ohne die Ableitungen der Booleschen Funktionen durchzugehen, kann die Wahrheitstabelle direkt aus dem Logikdiagramm abgeleitet werden, indem die folgenden Schritte befolgt werden:

1. Herausfinden, wie viele Eingangsvariablen in der Schaltung vorhanden sind: Erstellen Sie die 2^n Kombinationen von Eingängen, die für n Eingänge möglich sind und listen Sie dann die binären Ganzzahlen von 0 bis $(2^n - 1)$ in einer Tabelle auf,
2. Benutzen von beliebigen Symbolen, um die Ausgänge der gewählten Gatter zu identifizieren,
3. Ermitteln der Wahrheitstabelle für die Ausgänge der Gatter, die nur von ihren Eingangsvariablen abhängig sind sowie
4. Fortfahren, die Wahrheitstabelle für die Ausgänge der Gatter zu ermitteln, die von den zuvor festgelegten Werten abhängen, bis alle Spalten der Ausgänge identifiziert sind.

Die folgende Tabelle zeigt die Wahrheitstabelle für $F1$ und $F2$.

Tabelle 12: Wahrheitstabelle für die Funktionen $F1$ und $F2$

A	B	C	F1	F2
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0

A	B	C	F1	F2
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Quelle: Moustafa Nawito, 2022.

Entwurfsverfahren

Der Entwurf einer kombinatorischen Schaltung beginnt mit der Deklaration des Entwurfsziels und endet mit einem logischen Schaltplan oder einer Reihe von Booleschen Funktionen, die zur Erstellung des logischen Diagramms verwendet werden können. Die einzelnen Schritte des Prozesses sind wie folgt.

1. Bestimmen Sie die erforderliche Anzahl von Ein- und Ausgängen anhand der Schaltungsspezifikationen und weisen Sie jedem ein Symbol zu.
2. Erstellen Sie die Wahrheitstabelle, die die notwendige Verbindung zwischen Eingängen und Ausgängen beschreibt.
3. Bestimmen Sie in Abhängigkeit von den Eingangsvariablen die kondensierten Booleschen Funktionen für jeden Ausgang.
4. Erstellen Sie ein Logikdiagramm und stellen Sie sicher, dass der Entwurf korrekt ist (manuell oder durch Simulation).

Eingangs- und Ausgangsspalten bilden die Wahrheitstabelle einer kombinatorischen Schaltung. Die 2^n Binärzahlen für die n Eingangsvariablen werden verwendet, um die Eingangsspalten zu erstellen. Die Anwendung der angegebenen Spezifikationen erfolgt, um die binären Werte für die Ausgänge zu bestimmen. Die Ausgangsfunktionen der Wahrheitstabelle liefern eine genaue Spezifikation der kombinatorischen Schaltung. Da verbale Spezifikationen häufig vage sind, ist es entscheidend, dass sie in der Wahrheitstabelle genau übersetzt werden. Andernfalls könnte die Wahrheitstabelle falsch erstellt werden.

Zur Vereinfachung der in der Wahrheitstabelle angegebenen binären Ausgangsfunktionen wird jede verfügbare Technik eingesetzt, einschließlich algebraischer Manipulation, der KV-Diagramm-Methode oder einer computergestützten Vereinfachungssoftware. Häufig gibt es mehrere verschiedene vereinfachte Ausdrücke.

Bei der Auswahl einer Implementierung für eine bestimmte Anwendung werden eine Reihe von Kriterien als Leitfaden verwendet. Ein praktischer Entwurf muss eine Vielzahl von Charakteristiken berücksichtigen, die beim Entwurf integrierter Schaltungen zu beachten sind, darunter die Anzahl der Gatter sowie der Eingänge zu einem Gatter, die Laufzeit des Signals durch die Gatter, die Anzahl der Verbindungen, die Ansteuerungsfähigkeit jedes Gatters, die Anzahl der Gatter, mit denen der Ausgang der Schaltung verbunden werden kann und viele andere.

Es ist schwierig zu definieren, was eine akzeptable Lösung im Allgemeinen ist, da die Bedeutung jeder Einschränkung von der spezifischen Anwendung abhängt. In den meisten Fällen beginnt der Vereinfachungsprozess mit der Erreichung eines einfachen Ziels, z. B. der Erstellung der vereinfachten Booleschen Funktionen in einer Standardform. Die Vereinfachung wird dann in weiteren Schritten fortgesetzt, um andere Leistungsanforderungen zu erfüllen.

Beispiel

Entwerfen Sie einen BCD zu 2421 Codewandler.

Lösung:

Da sowohl der BCD-Code als auch der 2421-Code vier Bits besitzen, definieren Sie vier Eingänge A_3, A_2, A_1, A_0 zur Darstellung des BCD-Codes und vier Ausgänge F_3, F_2, F_1, F_0 zur Repräsentation des 2421-Ausgangscodes. Folglich wird die Wahrheitstabelle aufgestellt, in der für jeden BCD-Code, der durch die Eingänge dargestellt wird, der entsprechende 2421-Code, der durch den Ausgang dargestellt wird, danebensteht, wie in folgender Tabelle zu sehen ist.

Tabelle 13: Wahrheitstabelle zum BCD zu 2421 Wandler

Wert	A3	A2	A1	A0	F3	F2	F1	F0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	1	0	0
7	0	1	1	1	1	1	0	1
8	1	0	0	0	1	1	1	0
9	1	0	0	1	1	1	1	1
10	1	0	1	0	X	X	X	X
11	1	0	1	1	X	X	X	X
12	1	1	0	0	X	X	X	X
13	1	1	0	1	X	X	X	X
14	1	1	1	0	X	X	X	X

Wert	A3	A2	A1	A0	F3	F2	F1	F0
15	1	1	1	1	X	X	X	X

Quelle: Moustafa Nawito, 2022.

An diesem Punkt kann das Problem als Entwurf von vier verschiedenen Schaltungen zur Erzeugung von vier Ausgängen betrachtet werden. Zuerst schreibt man die DKNF für jeden Ausgang mithilfe der Wahrheitstabelle:

$$F3 = \sum(m5, m6, m7, m8, m9) + d(m10, m11, m12, m13, m14, m15)$$

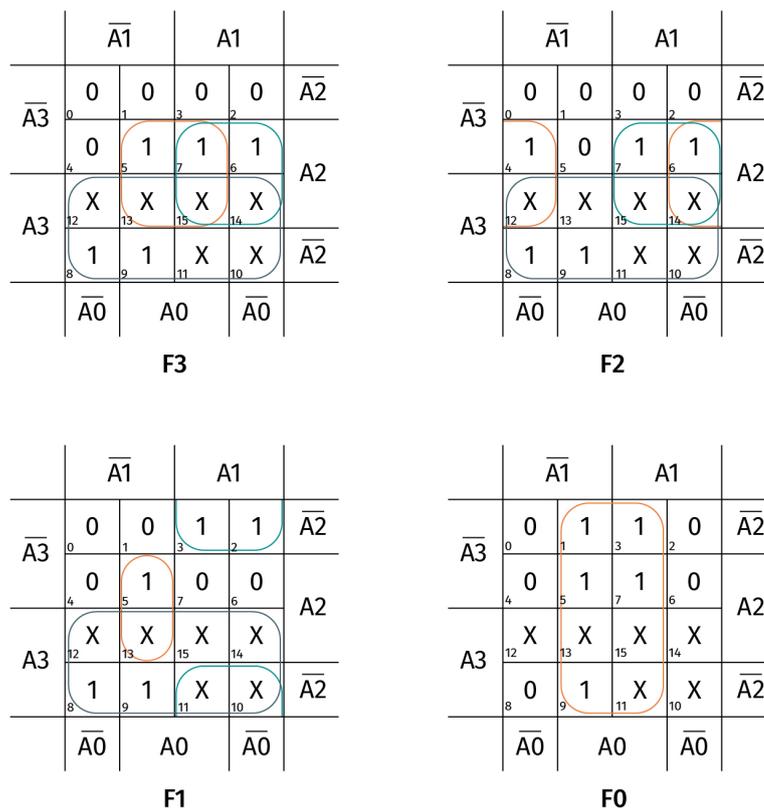
$$F2 = \sum(m4, m6, m7, m8, m9) + d(m10, m11, m12, m13, m14, m15)$$

$$F1 = \sum(m2, m3, m5, m8, m9) + d(m10, m11, m12, m13, m14, m15)$$

$$F0 = \sum(m1, m3, m5, m7, m9) + d(m10, m11, m12, m13, m14, m15)$$

Darauf basierend erstellt man vier KV-Diagramme, wie in folgender Abbildung dargestellt.

Abbildung 23: Die vier KV-Diagramme zum BCD zu 2421 Wandler



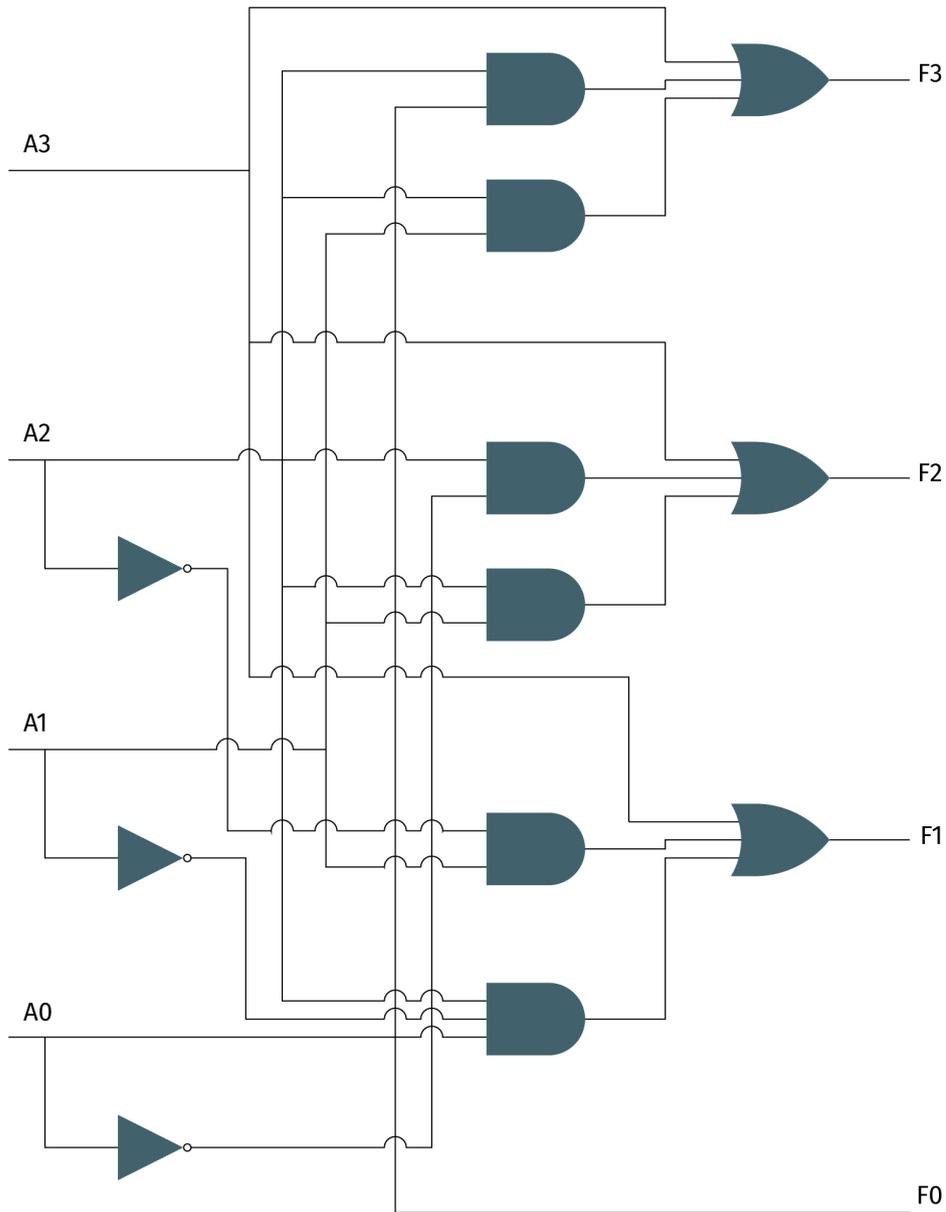
Quelle: Moustafa Nawito, 2022.

Die minimierten Ausdrücke der vier Ausgänge sind dann:

$$\begin{aligned}
 F_3 &= A_0A_2 + A_1A_2 + A_3 \\
 F_2 &= A_0'A_2 + A_1A_1 + A_3 \\
 F_1 &= A_0A_1'A_2 + A_1A_2' + A_3 \\
 F_0 &= A_0
 \end{aligned}$$

Der resultierende Schaltplan für den Wandler ist in folgender Abbildung dargestellt.

Abbildung 24: Schaltplan des BCD zu 2421 Wandlers



Quelle: Moustafa Nawito, 2022.

3.3 Decoder, Encoder und Multiplexer

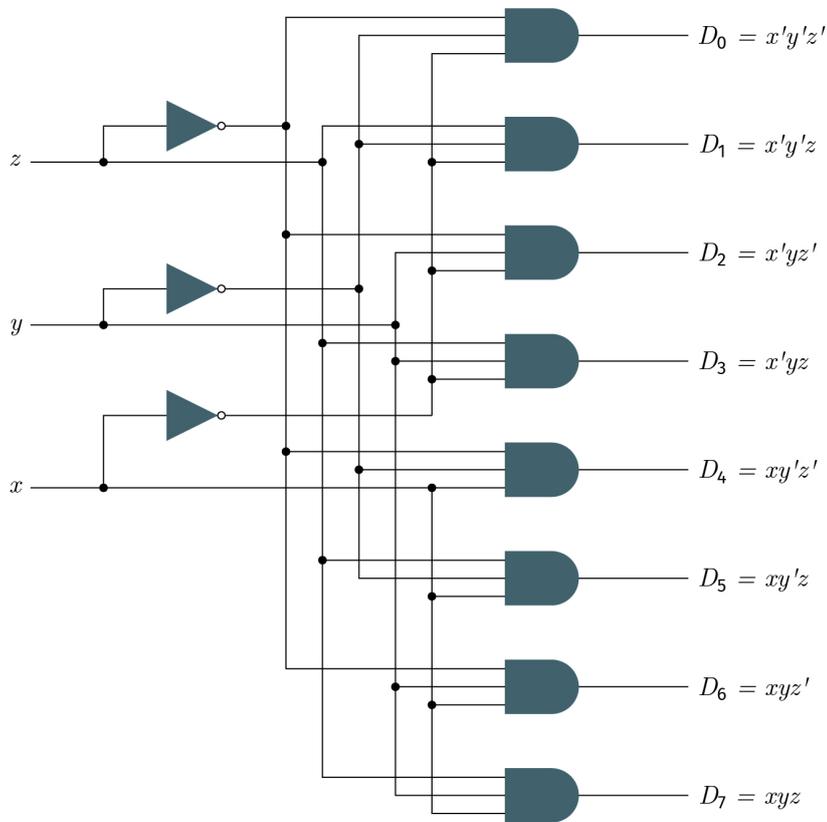
Decoder

Decoder
Eine kombinatorische Schaltung, die n Eingangsleitungen in maximal 2^n Ausgangsleitungen umwandelt, wird Decoder genannt.

Binärcodes werden in digitalen Systemen zur Darstellung diskreter Informationsmengen verwendet. Eine kombinatorische Schaltung, die als **Decoder** bezeichnet wird, wandelt binäre Daten von n Eingangsleitungen in maximal 2^n verschiedene Ausgangsleitungen um. Der Decoder kann weniger als 2^n Ausgänge haben, wenn die Informationen, die mit n Bits codiert sind, unbenutzte Kombinationen enthalten. Diese Decoder werden als n -zu- m -Zeilen-Decoder bezeichnet, wobei $m \leq 2^n$. Die 2^n (oder weniger) Minterme der n Eingangsvariablen sind das erwünschte Ergebnis, das die Eingangsvariablen erzeugen sollen. Jeder Satz von Eingängen führt zu einer anderen Ausgabe. Andere Codeumwandler, wie z. B. ein BCD-Sieben-Segment-Decoder, werden auch als Decoder bezeichnet.

Betrachten wir zur Veranschaulichung die Drei-zu-Acht-Linien-Decoder (3x8 Decoder) Schaltung in folgender Abbildung. Aus den drei Eingängen werden acht Ausgänge erzeugt, von denen jeder einem der drei Minterme der Eingangsvariablen entspricht. Das Komplement der Eingänge wird von den drei Invertern bereitgestellt und eines der Minterme wird von jedem der acht AND-Gatter erzeugt. Eine Anwendung dieses Decoders ist die Umwandlung von Binär- in Oktalzahlen. Die Ausgänge stellen die acht Ziffern einer Oktalzahl dar, während die Eingangsvariablen eine Binärzahl repräsentieren. Jeder Drei-Bit-Code kann jedoch decodiert werden, um acht Ausgänge zu erhalten – einen für jedes Codeelement, indem ein Drei-zu-Acht-Zeilen-Decoder verwendet wird.

Abbildung 25: 3x8-Decoder



Quelle: Moustafa Nawito, 2022, in Anlehnung an Mano, 2013, S. 151.

Die folgende Wahrheitstabelle kann helfen, die Funktionsweise des Decoders zu verdeutlichen. Es gibt sieben Ausgänge, die für jede mögliche Kombination von Eingängen gleich 0 sind und nur einen Ausgang, der gleich 1 ist. Die nun in den Eingangszeilen vorhandene Binärzahl wird durch den Ausgang dargestellt, dessen Wert 1 ist, was dem Minterm entspricht.

Tabelle 14: Wahrheitstabelle des 3x8-Decoders

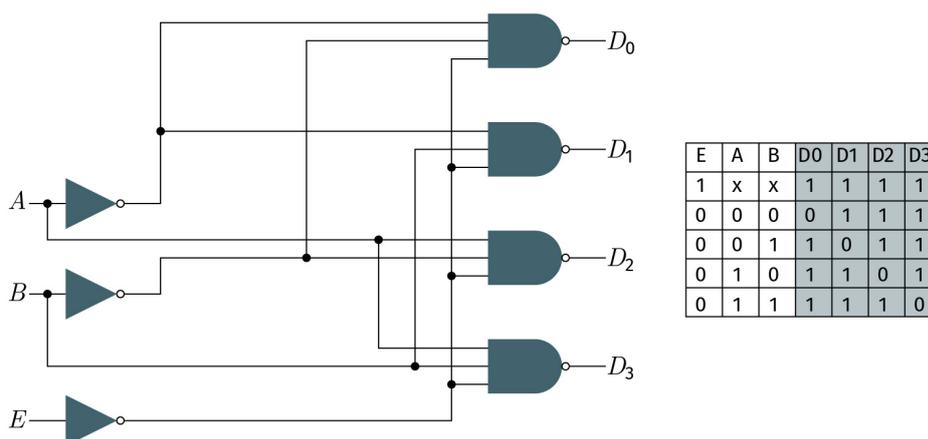
Eingänge			Ausgänge							
x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0

Eingänge			Ausgänge								
1	0	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	0	1

Quelle: Moustafa Nawito, 2022.

NAND-Gatter werden bei der Konstruktion einiger Decoder verwendet. Es ist kostengünstiger, die Decoder-Minterme in ihrer komplementären Form herzustellen, da ein NAND-Gatter die AND-Verknüpfung mit einem invertierten Ausgang erzeugt. Außerdem verfügen Decoder über einen oder mehrere Freigabeeingänge (enable), die die Funktionsweise der Schaltung regeln. Die folgende Abbildung zeigt einen 2x4-Decoder mit NAND-Gattern, der einen Freigabeeingang hat. Der Komplement-Freigabeeingang und die komplementären Ausgänge der Schaltung ermöglichen deren Funktion. Wenn E gleich 0 ist, ist der Decoder aktiviert (d. h. Aktiv-Low-Freigabe). Die Wahrheitstabelle zeigt, dass nur ein Ausgang gleich 0 sein kann und dass jeder andere Ausgang gleich 1 ist. Der von den Eingängen A und B gewählte Minterm wird durch den Ausgang repräsentiert, dessen Wert gleich 0 ist. Unabhängig von den Werten der beiden anderen Eingänge wird die Schaltung abgeschaltet, wenn E gleich 1 ist. Keiner der Ausgänge ist gleich 0 und keiner der minimalen Terme wird gewählt, wenn die Schaltung ausgeschaltet ist. Ein Decoder ist häufig in der Lage, mit komplementären oder unkomplementären Ausgängen zu arbeiten. Ein 0- oder 1-Signal kann zur Aktivierung des Freigabeeingangs verwendet werden. Einige Decoder verfügen über zwei oder mehr Freigabeeingänge, die jeweils eine bestimmte logische Anforderung erfüllen müssen, um die Schaltung zu aktivieren.

Abbildung 26: 2x4-Decoder mit enable



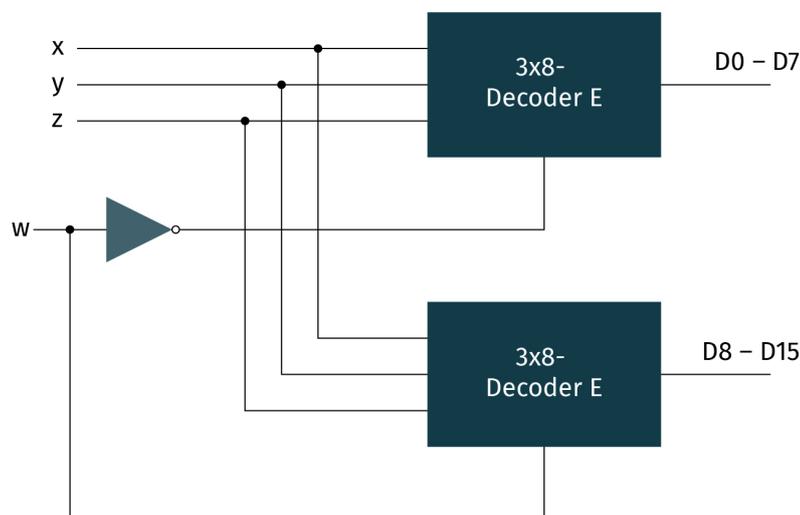
Quelle: Moustafa Nawito, 2022.

Ein Decoder mit einem Freigabeeingang kann als **Demultiplexer** dienen, d. h. eine Schaltung, die Daten von einer einzigen Leitung nimmt und sie an eine von 2^n verschiedenen Ausgangsleitungen sendet. Die Bitkonfiguration der n Auswahlleitungen bestimmt, welcher Ausgang gewählt wird. Wenn E als Dateneingangsleitung und A und B als Auswahlleitungen betrachtet werden, kann der in obiger Abbildung gezeigte Decoder als Demultiplexer mit einer bis vier Leitungen arbeiten. Die binäre Kombination der beiden Auswahlleitungen A und B legt fest, dass die Eingangsdaten nur an eine der Ausgangsleitungen gesendet werden, obwohl die einzige Eingangsvariable E einen Pfad zu allen vier Ausgängen hat. Die Wahrheitstabelle der Schaltung kann zur Bestätigung dieser Eigenschaft herangezogen werden. Zum Beispiel wird der Ausgang D_2 gleich dem Eingangswert E sein, wenn die Auswahlleitungen $AB = 10$ sind, aber alle anderen Ausgänge bleiben auf 1. Ein Decoder mit einem Freigabeeingang wird als Decoder-Demultiplexer bezeichnet, da Decoder- und Demultiplexer-Operationen von derselben Schaltung ausgehen. Eine größere Decoder-Schaltung kann durch die Kombination von Decodern mit Freigabeeingängen erstellt werden. In der unteren Abbildung werden zwei 3-zu-8-Leitungsdecoder mit Freigabeeingängen kombiniert, um einen 4-zu-16-Leitungsdecoder zu erzeugen. $W = 0$ schaltet den oberen Decoder frei und sperrt den anderen. Alle unteren Ausgänge des Decoders sind Nullen, während die oberen acht Ausgänge die Werte 0000 bis 0111 erzeugen. Wenn $w = 1$ ist, sind die Freigabeebedingungen umgekehrt: Die Ausgänge des oberen Decoders sind alle Nullen, während die Ausgänge des unteren Decoders den Bereich 1000–1111 liefern.

Demultiplexer

Eine kombinatorische Schaltung, die Daten von einer einzigen Leitung nimmt und sie an eine von 2^n verschiedene Ausgangsleitungen sendet, nennt sich Demultiplexer.

Abbildung 27: 4x16-Decoder aus zwei 3x8 mit enable



Quelle: Moustafa Nawito, 2022.

Implementation von Schaltnetzen mit Decodern

Die 2^n Minterme von n Eingangsvariablen werden durch einen Decoder gegeben. Ein bestimmter Satz von Eingangsbitmustern wird mit jedem bestätigten Ausgang des Decoders verbunden. Jede Boolesche Funktion ist in Form einer Zwischensumme darstellbar.

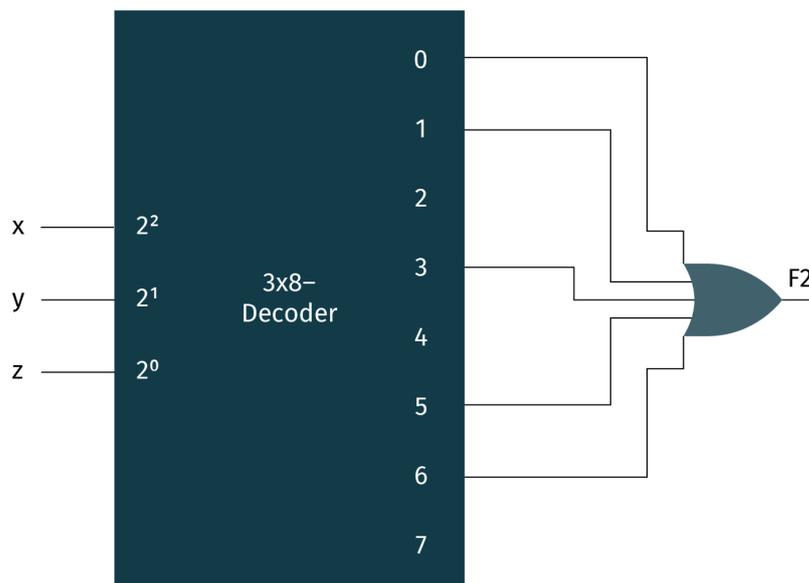
Daher kann eine Hardware-Implementierung der Funktion durch einen Decoder, der die Zwischensummen der Funktion erzeugt, und ein externes ODER-Gatter, das die logische Summe der Zwischensummen bildet, generiert werden. Mit einem n -zu- 2^n -Zeilen-Decoder und m ODER-Gattern lässt sich eine beliebige kombinatorische Schaltung mit n Eingängen und m Ausgängen erstellen. Die Boolesche Funktion der Schaltung muss als Summe von Mintermen geschrieben werden, um eine kombinatorische Schaltung mit einem ODER-Gatter und einem Decoder zu generieren. Der nächste Schritt besteht darin, einen Decoder auszuwählen, der alle Minimalterme der Eingangsvariablen liefern kann. Entsprechend der Liste der minimalen Terme jeder Funktion werden die Eingänge jedes ODER-Gatters aus den Ausgängen der Decoder ausgewählt.

Als Beispiel betrachten wir die Funktion F_2 , dessen KDNF wie folgt gegeben ist

$$= \neg x \neg y \neg z + \neg x \neg y z + \neg x y \neg z + x \neg y \neg z + x y \neg z$$

Die zugehörige Implementation mittels eines 3x8-Decoders und in OR-Gatter mit fünf Eingängen ist in folgender Abbildung dargestellt.

Abbildung 28: Implementation der Funktion $F_2 = \Sigma(m_0, m_1, m_3, m_5, m_6)$ mittels 3x8-Decoders und OR-Gatter



Quelle: Moustafa Nawito, 2022.

Encoder

Eine digitale Schaltung, die als Encoder bezeichnet wird, führt die entgegengesetzte Funktion eines Decoders aus. Ein Encoder hat n Ausgangsleitungen und 2^n (oder weniger) Eingangsleitungen. Insgesamt erzeugen die Ausgangsleitungen den Binärcode, der dem Ein-

gangswert entspricht. Der Oktal-zu-Binär-Encoder, dessen Wahrheitstabelle in folgender Abbildung dargestellt ist, ist ein Beispiel für einen Encoder. Er hat drei Ausgänge, die die entsprechende Binärzahl erzeugen, und acht Eingänge (einen für jede der Oktalziffern).

Es wird davon ausgegangen, dass es immer nur einen Eingang mit dem Wert 1 gibt. Die Wahrheitstabelle kann verwendet werden, um die Eingänge von OR-Gattern zu identifizieren, die zur Erstellung des Encoders verwendet werden können. Wenn die eingegebene Oktalziffer eins, drei, fünf oder sieben ist, ist der Ausgang z gleich 1. Für die Oktalziffern zwei, drei, sechs oder sieben ist der Ausgang y gleich 1, während für die Oktalziffern vier, fünf, sechs oder sieben der Ausgang x gleich 1 ist. Die folgenden Booleschen Ausgangsfunktionen können zur Beschreibung dieser Bedingungen verwendet werden:

$$z = D1 + D3 + D5 + D7$$

$$y = D2 + D3 + D6 + D7$$

$$x = D4 + D5 + D6 + D7$$

Zur Realisierung des Encoders können drei OR-Gatter verwendet werden.

Tabelle 15: Wahrheitstabelle eines 8x3-Encoders

Eingänge								Ausgänge		
D0	D1	D2	D3	D4	D5	D6	D7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Quelle: Moustafa Nawito, 2022.

Bei dem in obiger Abbildung beschriebenen Encoder kann immer nur ein Eingang aktiv sein. Insofern zwei Eingänge gleichzeitig aktiv sind, erzeugt der Ausgang eine unlogische Kombination. Wenn zum Beispiel $D3$ und $D6$ beide 1 sind, ist der Ausgang des Encoders 111, da alle drei Ausgänge gleich 1 sind. Die binäre drei oder die binäre sechs werden nicht durch den Ausgang 111 repräsentiert. Um dieses Problem zu beheben, müssen Codierschaltungen eine Eingangspriorität festlegen, um zu gewährleisten, dass nur ein Eingang codiert wird. Wenn wir Eingängen mit höheren tiefgestellten Zahlen eine höhere Priorität geben, und wenn $D3$ und $D6$ beide 1 sind, wird die Ausgabe 110 sein, da $D6$ eine höhere Priorität hat als $D3$. Eine weitere Unklarheit im Oktal-Binär-Codierer ist die Erzeugung einer Ausgabe, die ausschließlich aus Nullen besteht, wenn alle Eingänge null sind,

obwohl diese Ausgabe identisch mit der ist, die erzeugt wird, wenn D_0 gleich 1 ist. Durch die Bereitstellung eines weiteren Ausgangs, der anzeigt, ob mindestens ein Eingang gleich 1 ist, kann die Diskrepanz beseitigt werden.

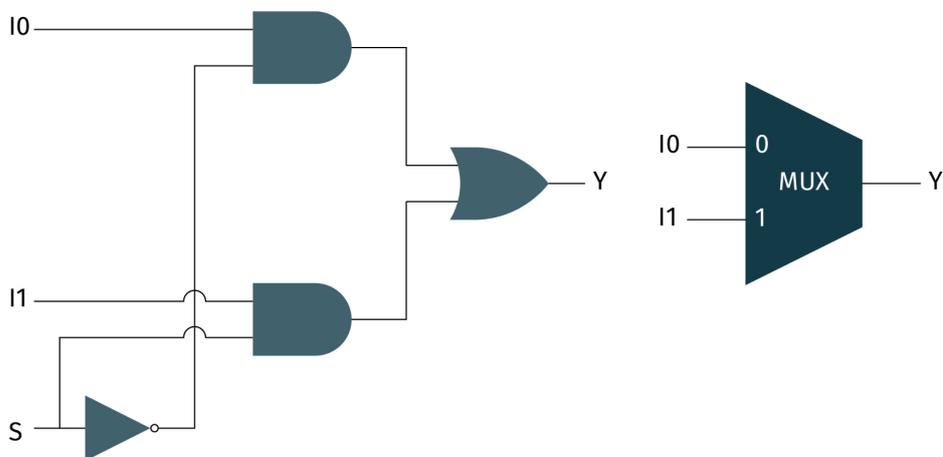
Multiplexer

Mithilfe einer kombinatorischen Schaltung, die als **Multiplexer** bezeichnet wird, werden Binärdaten aus einer oder mehreren Eingangsleitungen ausgewählt und an eine einzige Ausgangsleitung gesendet. Eine Gruppe von Auswahlleitungen bestimmt, welche Eingangsleitung gewählt wird. Normalerweise gibt es 2^n Eingangsleitungen und n Auswahlleitungen, und die Bitkombinationen auf den Auswahlleitungen bestimmen, welcher Eingang gewählt wird.

Wie in folgender Abbildung zu sehen, verbindet ein 2-zu-1-Leitung-Multiplexer eine von zwei 1-Bit-Quellen mit einem gemeinsamen Ziel. In der Schaltung sind zwei Dateneingangsleitungen, eine Ausgangsleitung und eine Auswahlleitung S (engl. selection) vorhanden. I_0 hat einen Pfad zum Ausgang, wenn $S = 0$ und das obere AND-Gatter aktiviert ist. I_1 hat einen Pfad zum Ausgang, wenn $S = 1$ und das untere AND-Gatter aktiviert ist. Der Multiplexer, der wie ein elektrischer Schalter funktioniert, wählt eine der beiden Quellen aus.

Manchmal wird das Blockdiagramm eines Multiplexers durch ein keilförmiges Symbol dargestellt. Es veranschaulicht visuell, wie eine ausgewählte Datenquelle aus einer Vielzahl von Quellen an eine bestimmte Stelle geleitet wird. In Blockschaltbildern wird der Multiplexer häufig als „MUX“ bezeichnet. In der nachfolgenden Abbildung (Schaltplan und Wahrheitstabelle vom 4x1-Multiplexer) ist ein 4-zu-1-Multiplexer abgebildet. I_0 bis I_3 sind die vier Eingänge, die an einen Eingang eines AND-Gatters angelegt werden. Die Auswahl eines bestimmten AND-Gatters erfolgt durch Decodierung der Auswahlleitungen S_1 und S_0 . Der einzeilige Ausgang wird durch Anlegen der Ausgänge der AND-Gatter an ein einziges OR-Gatter erzeugt. Jede Kombination der Ein- und Ausgänge der binären Auswahlwerte ist in der Funktionstabelle aufgeführt. Da er einen von mehreren Eingängen auswählt und die Binärdaten an die Ausgangsleitung weiterleitet, wird ein Multiplexer auch als Datenselektor bezeichnet.

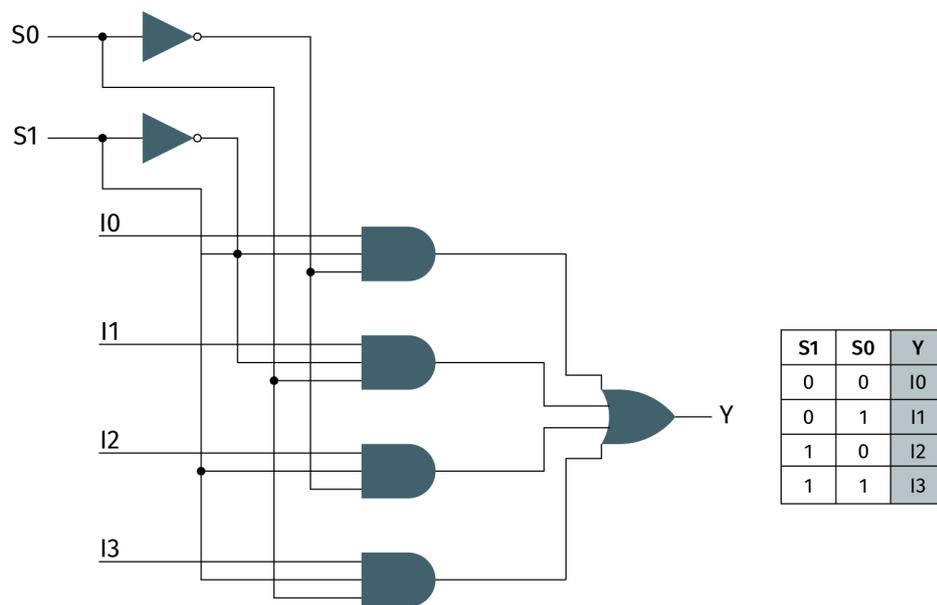
Abbildung 29: Schaltplan und Symbol vom 2x1-Multiplexer



Quelle: Moustafa Nawito, 2022.

Die AND-Gatter und Inverter des Multiplexers ähneln einer Decoderschaltung und decodieren tatsächlich die Auswahl der Eingangsleitungen. Ein n -zu- 2^n -Decoder wird in der Regel in einen 2^n -zu-1-Leitungsmultiplexer umgewandelt, indem man ihm 2^n Eingangsleitungen hinzufügt, eine für jedes AND-Gatter. Ein einziges OR-Gatter empfängt die Ausgänge aller AND-Gatter. Die Größe eines Multiplexers wird durch die Anzahl 2^n seiner Dateneingangsleitungen und durch seine einzige Ausgangsleitung bestimmt. Die 2^n Datenleitungen sind gleichbedeutend mit den n Auswahlleitungen. Ähnlich wie Decoder können Multiplexer einen Freigabeeingang enthalten, der die Funktionsweise des Geräts steuert. Die Ausgänge sind deaktiviert, wenn der Freigabeeingang inaktiv ist, und der Schaltkreis arbeitet wie ein Standardmultiplexer, wenn er aktiv ist.

Abbildung 30: Schaltplan und Wahrheitstabelle vom 4x1-Multiplexer



Quelle: Moustafa Nawito, 2022.

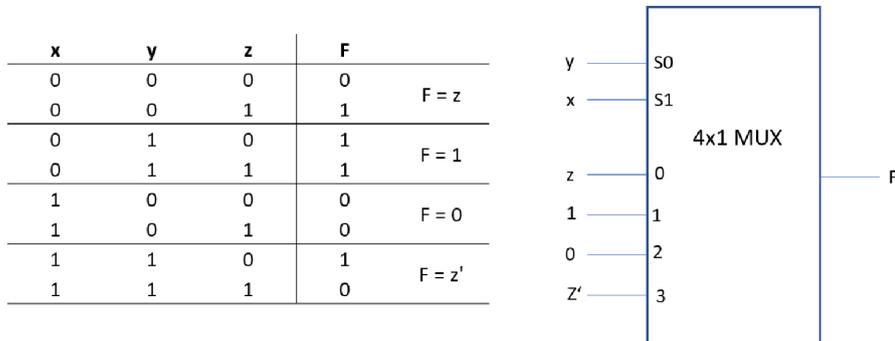
Implementation von Schaltnetzen mit Multiplexer

Ähnlich wie Decoder können Multiplexer zur Implementierung Boolescher Funktionen genutzt werden, allerdings auf effizientere Weise. Das typische Verfahren zur Verwendung eines Multiplexers mit $n - 1$ Auswahlengängen und $2^n - 1$ Dateneingängen zur Implementierung einer beliebigen Booleschen Funktion mit n Variablen sieht wie folgt aus:

- Zunächst wird eine Wahrheitstabelle erstellt, die die Boolesche Funktion enthält.
- Die Auswahlengänge des Multiplexers werden dann mit den ersten $n - 1$ Variablen der Tabelle belegt.
- Für jede Kombination der Auswahlvariablen wird die Ausgabe als Funktion der Endvariablen bewertet. Diese Funktion kann eine der Folgenden sein: 0, 1, die Variable oder das Komplement der Variablen.
- Die richtige Reihenfolge dieser Werte wird dann auf die Dateneingaben angewendet.

Als Beispiel betrachten wir die Funktion $F = \sum (m_1, m_2, m_3, m_6)$. Die zugehörige Wahrheitstabelle ist modifiziert anhand der oben beschriebenen Schritte, wie in folgender Abbildung dargestellt. Auf diese Art kann die Funktion mit einem 4x1-Multiplexer realisiert werden, wie gezeigt.

Abbildung 31: Realisierung der Funktion $F = \sum(m_1, m_2, m_3, m_6)$ mit einem 4x1-Multiplexer



Quelle: Moustafa Nawito, 2022.



ZUSAMMENFASSUNG

Kombinatorische Logik ist für alle Arten von digitalen Systemen unerlässlich. Schaltnetze erzeugen Ausgänge, die nur von den aktuell an ihren Anschlüssen anliegenden Eingängen abhängig sind, ohne Rückkopplungs- oder Speichereffekte. Bei der Erstellung von Schaltungen werden zahlreiche Arten von Logikgattern verwendet, die für die konkrete Umsetzung in verschiedene Logikfamilien unterteilt werden. Jede Logikfamilie wird in bestimmten Anwendungen eingesetzt.

Die korrekte Analyse von kombinatorischen Schaltungen hängt in hohem Maße davon ab, dass die Eingänge, Ausgänge und Zwischensignale organisiert und mit eindeutigen Namen und Bezeichnungen versehen werden. Andererseits kann der Entwurf einer kombinatorischen Logik auf der Grundlage der Wahrheitstabelle, des Booleschen Ausdrucks oder anderer Beschreibungsformen erfolgen, solange eine detaillierte und vollständige Spezifikation der Schaltung vorliegt.

Datenauswahlblöcke sind eine wichtige Klasse von kombinatorischen Schaltungen. Eine ihrer wichtigsten Anwendungen ist die effiziente Realisierung von kombinatorischen Schaltungen direkt aus der Wahrheitstabelle oder aus der DKNF der Funktion.

LEKTION 4

RECHENSCHALTUNGEN

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die verschiedenen Arten von Rechenschaltungen zu benennen.
- die Funktionsweise von Addierschaltungen zu erkennen.
- die Unterschiede zwischen Subtraktions- und Additionsschaltungen zu benennen.
- die Funktionsweise von Multiplikationsschaltungen zu erkennen.

4. RECHENSCHALTUNGEN

Einführung

Mit Digitalrechnern werden verschiedene Aufgaben der Informationsverarbeitung durchgeführt. Dazu gehören auch die zahlreichen Rechenoperationen. In dieser Lektion werden die Grundtypen arithmetischer Schaltungen vorgestellt, die auf kombinatorischer Logik beruhen. Der Schwerpunkt liegt auf den Gemeinsamkeiten und Unterschieden zwischen diesen Schaltungen und darauf, wie größere Schaltungen aus den Grundbausteinen erstellt werden können.

4.1 Addierschaltungen

Die Addition von zwei Binärziffern ist die grundlegendste Rechenoperation. Für diese einfache Addition gibt es vier alternative Grundoperationen: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$ und $1 + 1 = 10$. Als Übertrag bezeichnet man das höherwertige Bit dieses Ergebnisses. Der Übertrag, der durch die Addition von zwei Bits entsteht, wird zum folgenden höherwertigen Bitpaar addiert, wenn die Summanden-Zahlen mehr signifikante Ziffern besitzen. Ein sogenannter **Halbaddierer** ist eine kombinatorische Schaltung, die die Addition von zwei Bits durchführt. Ein Volladdierer (VA) ist ein Schaltkreis, der drei Bits addiert, einschließlich zweier Hauptbits und eines vorangehenden Übertrags. Die Namen der Schaltungen leiten sich von der Vorstellung ab, dass ein Volladdierer mit zwei Halbaddierern realisiert werden kann. Ein binärer Addierer für zwei n -Bit-Zahlen wird durch Kaskadierung von n vollständigen Addierern erzeugt.

Halbaddierer

Eine kombinatorische Schaltung, die die Addition von zwei Bits durchführt, nennt sich Halbaddierer.

Halbaddierer

Ein Halbaddierer benötigt zwei binäre Eingänge und zwei binäre Ausgänge. Die Summanden-Bits werden durch die Eingangsvariablen bezeichnet, während Summe und Übertrag durch die Ausgangsvariablen erzeugt werden. Wir geben den beiden Eingängen die Symbole x und y und den Ausgängen die Symbole S (für Summe) und C (für Übertrag, engl. carry). Die folgende Tabelle enthält die Wahrheitstabelle des Halbaddierers. Nur wenn beide Eingänge 1 sind, ist der Ausgang C gleich 1. Das niederwertigste Bit der Summe wird durch den S -Ausgang dargestellt. Die Wahrheitstabelle kann verwendet werden, um die minimierten Booleschen Funktionen für die beiden Ausgänge direkt zu extrahieren.

$$\begin{aligned} S &= xy' + x'y \\ C &= xy \end{aligned}$$

sind die vereinfachten Ausdrücke für die Summe der Produkte.

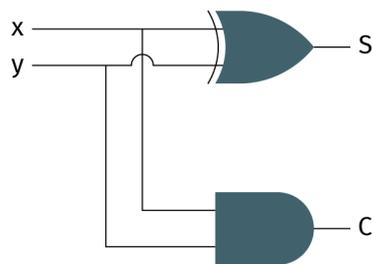
Tabelle 16: Wahrheitstabelle des Halbaddierers

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Quelle: Moustafa Nawito, 2022.

Wie in folgender Abbildung zu sehen ist, kann die Summe mit einem Exklusiv-OR und der Übertrag mit einem AND-Gatter gebildet werden. Anhand dieser Form wird verdeutlicht, wie zwei Halbaddierer zu einem Volladdierer kombiniert werden können.

Abbildung 32: Schaltplan des Halbaddierers



Quelle: Moustafa Nawito, 2022.

Ein Volladdierer muss verwendet werden, um n -Bit-Binärwerte zu addieren:

- Der Additionsprozess arbeitet Bit für Bit, von rechts nach links, beginnend mit dem niederwertigsten Bit.
- Nach dem niederwertigsten Bit muss bei der Addition an jeder Stelle zusätzlich zu den entsprechenden Bits der Wörter ein mögliches Übertrags-Bit aus der Addition an der vorherigen Stelle berücksichtigt werden.

Der Volladdierer bildet die arithmetische Summe von drei Bits. Er hat zwei Ausgänge und drei Eingänge. Die beiden zu addierenden signifikanten Bits werden durch zwei der Eingangsvariablen, x und y , dargestellt. Der Übertrag von der vorhergehenden niederwertigen Position wird durch den dritten Eingang, z , angezeigt. Da die arithmetische Summe von drei binären Ziffern zwischen null und drei liegen kann und die binäre Darstellung von zwei oder drei zwei Bits erfordert, sind zwei Ausgänge notwendig. Die Buchstaben S für Summe und C für Übertrag stehen für die beiden Ausgänge. Das LSD-Bit der Summe wird durch die Binärvariable S dargestellt. Der Ausgangsübertrag, der durch Addition des Eingangsübertrags und der Summenbits entsteht, wird durch die Binärvariable C dargestellt. Alle möglichen Kombinationen der drei Variablen werden durch die acht Zeilen unter den

Eingangsvariablen repräsentiert. Die arithmetische Summe der Eingangsbits wird verwendet, um die Ausgangsvariablen zu bestimmen. Der Ausgang ist 0, insofern alle Eingangsbits 0 sind. Wenn entweder einer oder alle drei Eingänge gleich 1 sind, ist der S -Ausgang ebenfalls 1. Insofern zwei oder drei der Eingänge gleich 1 sind, hat der C -Ausgang einen Übertrag von 1. Die Eingangs- und Ausgangsbits der kombinatorischen Schaltung können auf verschiedene Weise interpretiert werden, je nachdem, wo das Problem im Prozess liegt. Physikalisch gesehen werden die binären Eingangssignale als binäre Ziffern betrachtet, die arithmetisch verbunden werden können, um am Ausgang eine zweistellige Summe zu ergeben. Bei der Aufzeichnung in einer Wahrheitstabelle oder beim Aufbau der Schaltung mithilfe von Logikgattern werden die identischen Binärwerte dagegen als Variablen Boolescher Funktionen berücksichtigt.

Die folgende Tabelle zeigt die Wahrheitstabelle des Volladdierers. Für den Entwurf wird ein Standardverfahren mit einem KV-Diagramm mit drei Variablen verwendet. Die resultierenden Ausdrücke für S und C sind:

$$S = xy'z' + x'yz' + xyz + x'y'z = z \oplus (x \oplus y)$$

$$C = xy + xz + yz = (x \oplus y)z + xy$$

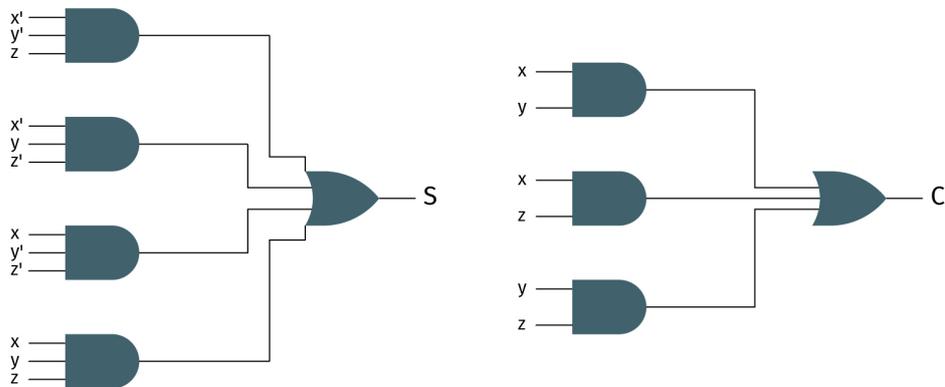
Diese Ausdrücke zeigen, dass der Volladdierer mit Und- und Oder-Gattern in der Standardform der Summe der Produkte oder als Kaskadierung von zwei Halbaddierern in Reihe realisiert werden kann, wie in der unteren Abbildung (Schaltplan des Volladdierers) verdeutlicht.

Tabelle 17: Wahrheitstabelle des Volladdierers

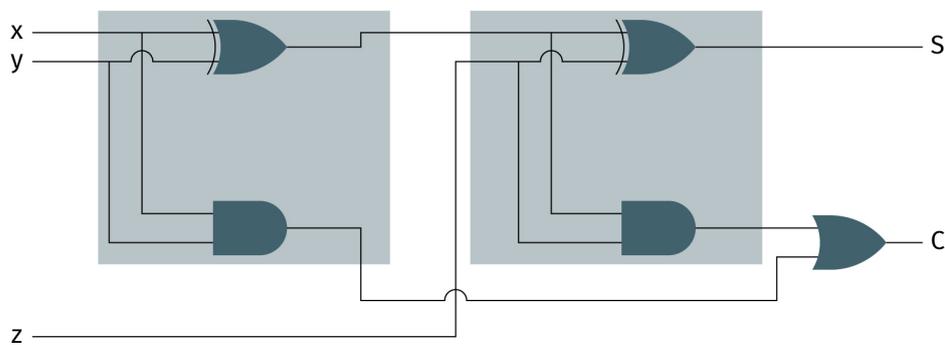
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Quelle: Moustafa Nawito, 2022.

Abbildung 33: Schaltplan des Volladdierers



Volladdierer als Summe von Produkten



Volladdierer als zwei Halbaddierer in Reihe

Quelle: Moustafa Nawito, 2022.

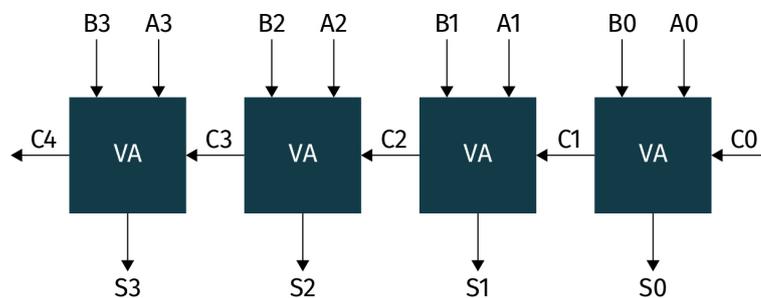
Binäraddierer

Eine digitale Schaltung, die als Binäraddierer bezeichnet ist, erzeugt die arithmetische Summe von zwei binären ganzen Zahlen. Er kann mit Volladdierern aufgebaut werden, die in Kaskade geschaltet sind, wobei der Eingangsübertrag eines Volladdierers in den Ausgangsübertrag des folgenden Volladdierers einfließt. Eine Kette von n Volladdierern oder eine Kette von einem Halbaddierer und $n - 1$ Volladdierern wird benötigt, um n -Bit-Werte zu addieren. Der Eingangsübertrag an der niederwertigsten Stelle ist im ersten Szenario auf 0 festgelegt. In folgender Abbildung sind vier Volladdierer miteinander verbunden, um einen binären Vier-Bit-Ripple-Carry-Addierer zu bilden (Ripple-Carry bedeutet Welle-Übertrag, da sich der Übertrag wie eine Welle durch die Flip-Flops ausbreitet.). Das niederwertigste Bit wird durch den Index 0 gekennzeichnet, während die Summand-Bits von A und B durch Zahlen von rechts nach links gekennzeichnet sind. Die gesamten Volladdierer bilden eine Kette, die den Übertrag verbindet. Der Eingangsübertrag des Addierers ist C_0 , und er durchläuft alle Volladdierer, um den Ausgangsübertrag C_4 zu erzeugen. Die erforder-

derlichen Summenbits werden von den S -Ausgängen erzeugt. Jeder Ausgangsübertrag der n Volladdierer, die für einen n -Bit-Addierer benötigt werden, muss mit einem Eingangsübertrag des n -ten Volladdierers höherer Ordnung verbunden werden.

Betrachten wir zur Veranschaulichung die beiden Binärwerte $A = 1011$ und $B = 0011$. Der 4-Bit-Addierer bildet seine Summe $S = 1110$. Dieser Addierer ist ein Beispiel für eine typische Standardkomponente. Er kann in einer Vielzahl von arithmetischen Anwendungen eingesetzt werden. Beachten Sie, dass diese Schaltung neun Eingänge hat, sodass die klassische Methode eine Wahrheitstabelle mit $2^9 = 512$ Einträgen erfordern würde. Eine einfache und unkomplizierte Implementierung kann durch eine iterative Kaskadierung einer Standardfunktion erreicht werden.

Abbildung 34: 4-Bit-Binäraddierer



Quelle: Moustafa Nawito, 2022.

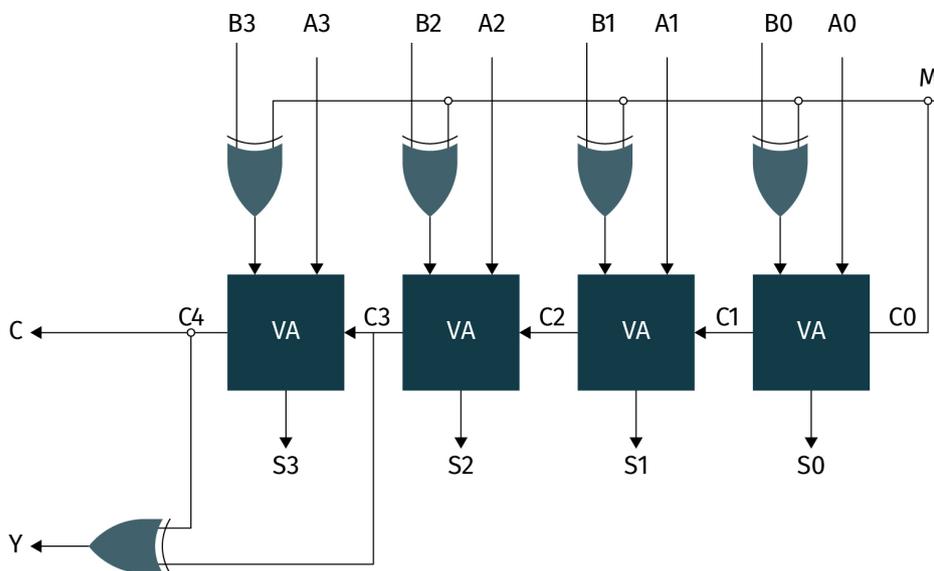
4.2 Subtrahierschaltungen

Eine kombinatorische Schaltung, die als Binäraddierer-Subtrahierer bekannt ist, führt die Addition und Subtraktion von Binärzahlen durch. Diese Schaltung wird mithilfe einer hierarchischen Architektur erstellt. Eine Komplementärschaltung enthält die Subtraktionsschaltung.

Komplemente sind die praktischste Methode zur Subtraktion von Binärwerten ohne Vorzeichen. Damit man A von B entfernen kann, wird das 2er-Komplement von B zu A addiert. Das 2er-Komplement erhält man, indem eine 1 zum LSD des 1er-Komplements addiert wird. Mithilfe von Invertern lässt sich das 1er-Komplement realisieren, und der Eingangsübertrag kann verwendet werden, um die Summe um 1 zu erhöhen. Ein Addierer mit Invertern zwischen jedem Dateneingang B und dem entsprechenden Eingang des gesamten Addierers bildet die Schaltung für die Subtraktion von A von B . Bei der Subtraktion muss der Eingangsübertrag C_0 gleich 1 sein. Das Ergebnis dieses Verfahrens ist $A + B$'s Komplement mal 1, plus 1. Dies ist gleichbedeutend mit A plus B 's 2er-Komplement. Das Ergebnis ist $A - B$ für vorzeichenlose Zahlen, wenn $A \geq B$ ist, oder das 2er-Komplement von $(B - A)$, wenn $A < B$ ist. Insofern es keinen Überlauf gibt, ist das Ergebnis für vorzeichenbehaftete Zahlen $A - B$.

Durch den Einbau eines Exklusiv-OR-Gatters in jeden Volladdierer können die Additions- und Subtraktionsoperationen in einer Schaltung mit einem einzigen gemeinsamen Binäraddierer zusammengeführt werden. Die folgende Abbildung zeigt eine 4-Bit-Addierer-Subtrahierer-Schaltung. Die Schaltung arbeitet als Addierer, wenn $M = 0$ ist, und als Subtrahierer, wenn $M = 1$ ist. Beide Eingänge M und einer der Eingänge von B werden in jedes Exklusiv-OR-Gatter eingespeist. $B \oplus 0 = B$ ist das Ergebnis, wenn $M = 0$. Der Eingangsübertrag ist 0, die Volladdierer erhalten den Wert von B , und die Schaltung addiert A und B . $B \oplus 1 = \bar{B}$ und $C_0 = 1$ sind wahr für $M = 1$. Die Eingänge von B werden alle komplementiert und eine 1 wird durch den Eingangsübertrag addiert. Die Schaltung führt die Operation A plus das 2er-Komplement von B aus. (Das Exklusiv-OR mit Ausgang Y dient zur Erkennung eines Überlaufs).

Abbildung 35: 4-Bit Addierer-Subtrahierer mit Überlauferkennung



Quelle: Moustafa Nawito, 2022.

Es ist wichtig zu beachten, dass die grundlegenden Additions- und Subtraktionsregeln für ganze Zahlen ohne Vorzeichen und Binärzahlen im System des vorzeichenbehafteten Komplements gleich sind. Computer können daher beide Arten der Arithmetik mit einer einzigen Hardware-Schaltung verarbeiten. Je nachdem, ob die Zahlen als vorzeichenbehaftet oder vorzeichenlos angenommen werden, muss der Benutzende oder Programmierende die Ergebnisse einer solchen Addition oder Subtraktion entsprechend interpretieren.

Überlauf bei digitalem Subtrahieren und Addieren

Ein Überlauf tritt auf, wenn zwei ganze Zahlen mit jeweils n Stellen addiert werden und das Ergebnis $n + 1$ Stellen einnimmt. Dies gilt für binäre oder dezimale Ganzzahlen mit oder ohne Vorzeichen. Da es beim Addieren mit Papier und Bleistift keine Beschränkung

gibt, wie viel auf die Seite geschrieben werden kann, ist ein Überlauf kein Problem. Dies jedoch stellt eine Hürde in digitalen Computern dar, weil in diesem Kontext eine endliche Anzahl von Bits existieren. So stellt sich ein Überlauf problematisch dar, da ein Ergebnis mit $n + 1$ Bits nicht in ein n -Bit-Wort passt. Aus diesem Grund wird in vielen Systemen bei einem Überlauf ein entsprechendes Speicherelement gesetzt, sodass der Benutzende den Überlauf nachträglich überprüfen kann. Ob zwei Binärwerte als vorzeichenbehaftet oder vorzeichenlos gelten, wirkt sich darauf aus, ob nach der Addition der beiden Zahlen ein Überlauf erkannt wird. Ein Überlauf wird identifiziert, wenn zwei vorzeichenlose Zahlen vom Endübertrag der höchstwertigen Stelle aus addiert werden. Zwei Fakten über vorzeichenbehaftete Zahlen sind entscheidend:

- negative Zahlen haben immer die Form des 2er-Komplements und
- das Bit ganz links spiegelt immer das Vorzeichen wider.

Das Vorzeichenbit wird als Teil der Zahl behandelt, wenn zwei vorzeichenbehaftete Zahlen addiert werden. Der Endübertrag signalisiert keinen Überlauf.

Wenn eine Zahl positiv und die andere negativ ist, kann die Addition nicht zu einem Überlauf führen, da das Ergebnis der Addition einer positiven Zahl zu einer negativen Zahl einen kleineren Betrag hat als der größere der beiden Ausgangswerte. Wenn die beiden addierten Zahlen beide positiv oder beide negativ sind, kann es zu einem Überlauf kommen. Anhand des folgenden Beispiels können Sie sehen, wie dies geschehen kann: Zwei 8-Bit-Register enthalten zwei binäre Ganzzahlen mit Vorzeichen, +60 und +90. Jedes Register kann Zahlen im Bereich von binär +127 bis binär -128 enthalten. Der kombinierte Wert der beiden Zahlen von +150 bedeutet, dass sie größer sind als ein 8-Bit-Register speichern kann. Das Gleiche gilt für -70 und -80. Als nächstes werden die letzten beiden Überträge und die binären Darstellungen der beiden Additionen gezeigt: Das 8-Bit-Ergebnis, das negativ sein sollte, hat ein positives Vorzeichenbit, und das 8-Bit-Ergebnis, das positiv sein sollte, hat ein negatives Vorzeichenbit (d. h. das achte Bit). Die so erzeugte 9-Bit-Antwort ist jedoch genau, wenn der Übertrag der Vorzeichen-Bit-Position als das Vorzeichen-Bit des Ergebnisses behandelt wird. Da die Antwort jedoch nicht in acht Bits passt, erklären wir, dass ein Überlauf stattgefunden hat.

Sowohl der Übertrag in die Vorzeichenbitposition als auch der Übertrag aus der Vorzeichenbitposition können verwendet werden, um ein Überlaufproblem zu erkennen. Ein Überlauf ist aufgetreten, wenn das Verhältnis dieser beiden Überträge nicht gleich ist. Die Beispiele, in denen die beiden Überträge explizit angezeigt werden, dienen als Hinweis darauf. Insofern der Ausgang des Gatters gleich 1 ist, wird ein Überlauf festgestellt, wenn die beiden Überträge auf ein Exklusiv-OR-Gatter angewendet werden. Dieser Ansatz erfordert, dass das 2er-Komplement einer negativen Zahl durch Addition von 1 zum 1er-Komplement der negativen ganzen Zahl berechnet wird. Dadurch wird das Problem gelöst, das auftritt, wenn die größte negative Zahl komplementiert wird.

Die Abbildung oberhalb zeigt den binären Addierer-Subtrahierer-Schaltkreis mit den Ausgängen C und V . Das C -Bit erkennt einen Übertrag nach der Addition oder einen Kredit nach der Subtraktion, wenn die beiden binären Ganzzahlen als vorzeichenlos betrachtet werden. Das V -Bit identifiziert einen Überlauf, wenn die Zahlen als vorzeichenbehaftet angesehen werden. Insofern nach einer Addition oder Subtraktion $V = 0$ ist, gab es kei-

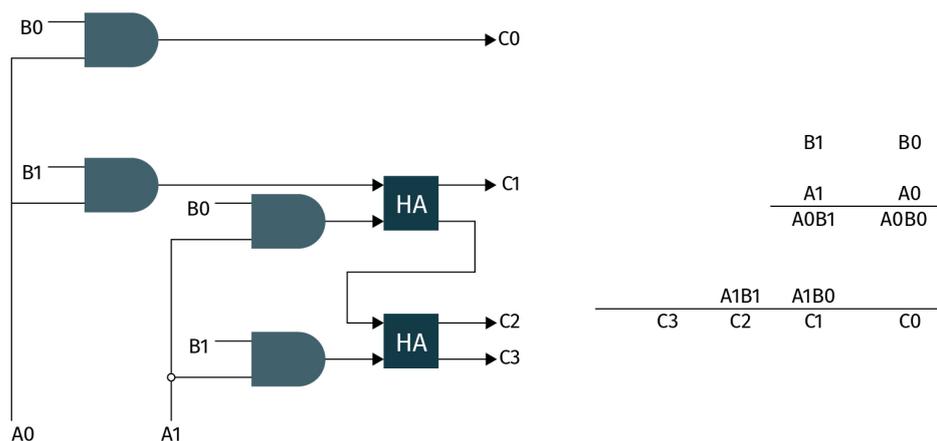
nen Überlauf und das n -Bit-Ergebnis war korrekt. Ist $V = 1$, ergibt das Verfahren ein Ergebnis mit $n + 1$ Bits, aber nur die ganz rechten n Bits der Zahl passen in den zugewiesenen Platz, was zu einem Überlauf führt. Das eigentliche Vorzeichen, das verschoben wurde, ist im $(n + 1)$ -ten Bit enthalten.

4.3 Multiplikationsschaltungen

Das Verfahren zur Multiplikation von Binärzahlen ist das gleiche wie bei der Multiplikation von Dezimalwerten. Beginnend mit dem niederwertigsten Bit wird der Multiplikand mit jedem Bit des Multiplikators multipliziert. Jede dieser Multiplikationen erzeugt ein Teilprodukt. Die Position jedes nachfolgenden Teilprodukts wird um eins nach links verschoben. Die Summe der Teilprodukte ergibt das Endprodukt.

Betrachten wir die Multiplikation zweier 2-Bit-Zahlen, wie in folgender Abbildung dargestellt, um zu sehen, wie ein binärer Multiplizierer mit einer kombinatorischen Schaltung implementiert werden kann. $B1$ und $B0$ sind die Multiplikandenbits, $A1$ und $A0$ sind die Multiplikatorenbits und $C3C2C1C0$ ist das Ergebnis. Durch Multiplikation von $B1B0$ mit $A0$ wird das erste Teilprodukt gebildet. Die Multiplikation von zwei Bits, z. B. $A0$ und $B0$, ergibt eine 1, wenn beide Bits 1 sind, andernfalls eine 0, ähnlich wie bei einer AND-Verknüpfung. Folglich können AND-Gatter verwendet werden, um das partielle Produkt, wie im Diagramm dargestellt, zu realisieren. Durch Multiplikation von $B1B0$ mit $A1$ und Verschiebung um eine Stelle nach links entsteht das zweite Teilprodukt. Zwei Halbaddierer (HA) werden verwendet, um die beiden Teilprodukte zu kombinieren. Die Teilprodukte enthalten in der Regel mehr Bits, sodass die Summe der Teilprodukte mit Volladdierern realisiert werden muss. Da die Summe aus dem Ausgang des ersten AND-Gatters gebildet wird, muss das niederwertigste Bit des Produkts keinen Addierer durchlaufen.

Abbildung 36: 2-Bit mal 2-Bit binäre Multiplikationsschaltung

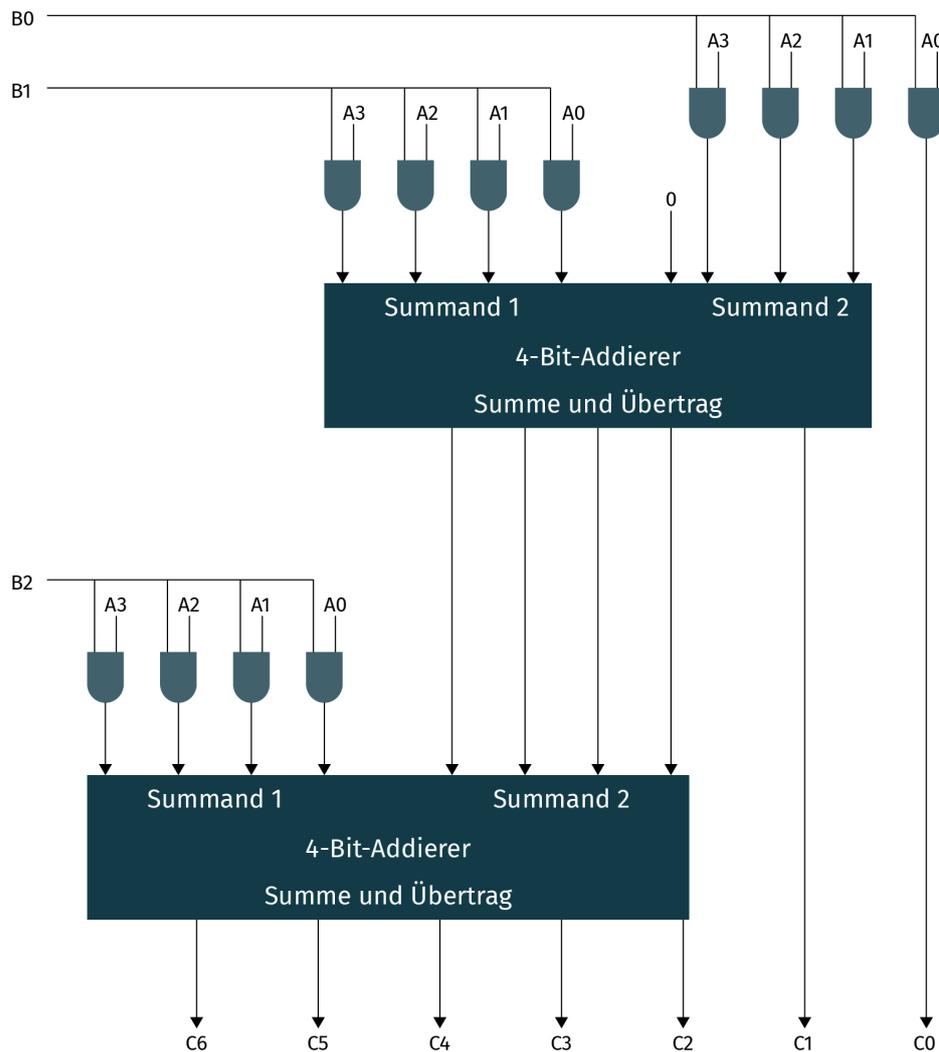


Quelle: Moustafa Nawito, 2022.

Auf ähnliche Weise kann ein binärer Multiplizierer mit Schaltnetzen mit mehr Bits aufgebaut werden. Auf so vielen Ebenen, wie Bits im Multiplikator vorhanden sind, wird ein Bit des Multiplikators mit jedem Bit des Multiplikanden AND-verknüpft. Der Binärausgang eines AND-Gatters wird auf jeder Stufe mit dem Teilprodukt der vorhergehenden Stufe kombiniert, um ein neues Teilprodukt zu generieren. Die Bildung des Produkts erfolgt auf der letzten Stufe. Um ein Produkt aus $(J + K)$ -Bits zu erzeugen, benötigt man $(J \times K)$ AND-Gatter und $(J - 1)$ - K -Bit-Addierer für J -Multiplikatorbits und K -Multiplikandenbits.

Betrachten wir als zweites Beispiel eine Multiplikatorschaltung, die eine Binärzahl mit vier Bits mit einer Zahl mit drei Bits multipliziert. $A3A2A1A0$ und $B2B1B0$ sollen als Multiplikand bzw. Multiplikator dienen. Da $K = 4$ und $J = 3$, erfordert ein Produkt aus sieben Bits den Einsatz von zwölf AND-Gattern und zwei 4-Bit-Addierern. Die folgende Abbildung veranschaulicht das Logikdiagramm des Multiplizierers.

Abbildung 37: 3-Bit mal 4-Bit binäre Multiplikationsschaltung



Quelle: Moustafa Nawito, 2022.



ZUSAMMENFASSUNG

Mithilfe der kombinatorischen Logik können Schaltungen erstellt werden, die die arithmetischen Operationen Addition, Subtraktion und Multiplikation ausführen. Das Kernelement der binären Addition ist das EXOR-Gatter, da es zur Erstellung des Halbaddierers verwendet wird. Dieses wiederum kann als Baustein für die Realisierung eines Volladdierers verwendet werden, indem zwei davon in Reihe geschaltet werden. Ein n -Bit-Binäraddierer besteht aus einer Reihenschaltung von n Vollad-

dieren. Jeder Volladdierer berechnet eine Ziffer der Ausgangssumme, angefangen von der LSD bis zur MSD. Der Übertrag breitet sich in der gesamten Addiererkette aus.

Die Hardware-Realisierung der binären Subtraktion basiert, wie die manuelle algebraische Subtraktion, auf der Komplement-Operation. Die Addierer-Subtrahierer-Schaltung wird mit Volladdierern und EXOR-Gattern realisiert. Besondere Vorsicht ist bei der Berücksichtigung von Überlaufbedingungen geboten, um nicht ein falsches Ergebnis zu erhalten. Die Interpretation des Ergebnisses als vorzeichenbehaftet oder vorzeichenlos hängt vom Benutzenden ab.

Ein binärer Multiplizierer ist eine gatterintensive Schaltung und wird mit Volladdierern und EXOR-Gattern realisiert.

LEKTION 5

SCHALTWERKE (SEQUENZIELLE LOGIK)

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die Grundlagen von Schaltwerken sequenzieller Logik zu definieren.
- die Merkmale sequenzieller Logik aufzulisten.
- die Analyse und das Design von Schaltwerken durchzuführen.
- die Eigenschaften wichtiger sequenzieller Schaltungen zu benennen.

5. SCHALTWERKE (SEQUENZIELLE LOGIK)

Einführung

Die Fähigkeit, Informationen im Binärformat zu speichern, abzurufen und zu verarbeiten, ist eine Funktion, die in fast allen elektronischen Konsumgütern enthalten ist. Elektronische Teile, die Daten speichern können oder einen Speicher besitzen, sind für die Technologie, die diese Geräte ermöglicht und unterstützt, unerlässlich. Diese Lektion befasst sich mit dem Betrieb und der Steuerung dieser elektronischen Bauelemente sowie mit ihrer Verwendung in Schaltungen.

5.1 Grundlagen

Die bisher besprochenen digitalen Schaltungen sind kombinatorisch, d. h., ihr Ausgang hängt einfach und unmittelbar von ihren Eingängen ab. Sie haben keinen Speicher und sind nicht von früheren Eingangswerten abhängig. **Schaltwerke** oder sequenzielle Schaltungen hingegen besitzen einen Speicher und dienen als Speicherkomponenten. Sie sind in der Lage, Informationen zu speichern, aufzubewahren und dann bei Bedarf in der Zukunft abzurufen. Die folgende Abbildung zeigt ein Blockdiagramm einer sequenziellen Schaltung. In diesem System wird ein Speicherelement-Rückkopplungspfad geschaffen, indem Speicherelemente mit einem Schaltnetz verbunden werden. Die Bauteile, aus denen die Speicherteile bestehen, können binäre Daten speichern. Der Zustand (engl. state) der sequenziellen Schaltung zu einem bestimmten Zeitpunkt wird durch die binären Informationen bestimmt, die zu diesem Zeitpunkt in diesen Elementen gespeichert sind. Der Binärwert der Ausgänge wird durch den Empfang von Binärdaten von externen Eingängen und dem aktuellen Zustand der Speicherelemente durch die Folgeschaltung bestimmt. Die Voraussetzung für die Änderung des Zustands in den Speicherelementen wird ebenfalls durch diese externen Eingänge geschaffen. Das Blockschaltbild zeigt, dass die Ausgänge einer sequenziellen Schaltung sowohl von den Eingängen als auch vom aktuellen Zustand der Speicherelemente abhängen. Sowohl der aktuelle Zustand als auch die externen Eingänge beeinflussen den späteren Zustand der Speicherelemente. Eingänge, Ausgänge und interne Zustände werden somit in einer zeitlichen Abfolge angegeben, die eine sequenzielle Schaltung definiert. Im Gegensatz dazu sind die Ausgänge der kombinatorischen Logik ausschließlich von den aktuellen Werten der Eingänge abhängig.

Schaltwerke

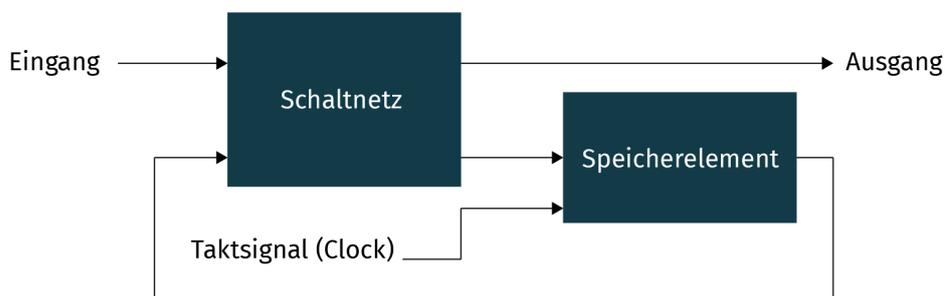
Logische Schaltungen, die Schaltnetze und Speicherelemente beinhalten, werden Schaltwerke genannt. Ihre Ausgänge hängen von gegenwärtigen sowie von früheren Eingangswerten ab.

Synchrone und asynchrone Schaltwerke

Sequenzielle Schaltungen lassen sich auf der Grundlage der zeitlichen Abfolge ihrer Signale in zwei Hauptkategorien unterteilen. Ein System, dessen Verhalten sich aus der Kenntnis seiner Signale zu bestimmten Zeitpunkten vorhersagen lässt, wird als synchrone sequenzielle Schaltung bezeichnet. Das Verhalten einer asynchronen sequenziellen Schaltung wird durch die Eingangssignale bestimmt, die sie zu einem bestimmten Zeitpunkt empfängt, und durch die Reihenfolge, in der sich diese Eingangssignale ändern.

Eine synchrone sequenzielle Schaltung verwendet Signale, die sich nur zu bestimmten Zeitpunkten auf die Speicherkomponenten auswirken. Ein Taktgeber, der ein Taktsignal in Form einer periodischen Folge von Taktimpulsen erzeugt, synchronisiert die Geräte. Zur Bezeichnung des Taktsignals werden häufig die Begriffe „clock“ und „clk“ verwendet. Einfluss auf die Speicherelemente üben nur die einzeln eintreffenden Taktimpulse aus, da die Taktimpulse auf diese Weise über das gesamte System verteilt werden. Das Timing der Rechenaktivitäten innerhalb der Schaltung wird tatsächlich durch die Taktimpulse gesteuert, während Änderungen an den Speicherkomponenten und Ausgängen durch andere Signale (sowohl externe Eingänge als auch andere Signale) gesteuert werden. Eine Schaltung, die zum Beispiel zwei binäre Ganzzahlen addieren und speichern soll, würde deren Summe aus den Werten der Zahlen berechnen und das Ergebnis speichern, wenn ein Taktimpuls auftritt. Getaktete sequenzielle Schaltungen sind eine Art von synchronen sequenziellen Schaltungen, die in der Praxis am häufigsten verwendet werden und Taktimpulse zur Steuerung von Speicherelementen verwenden. Da die Aktivität innerhalb der Schaltung und die daraus resultierende Aktualisierung der gespeicherten Werte mit dem Auftreten von Taktimpulsen übereinstimmen, werden sie als synchrone Schaltungen bezeichnet. Da synchrone Schaltungen selten Instabilitätsprobleme aufweisen und sich leicht in diskrete, unabhängige Prozesse aufteilen lassen, die jeweils separat berücksichtigt werden können, ist der Entwurf synchroner Schaltungen praktisch. Die Schaltung in der unteren Abbildung ist synchron.

Abbildung 38: Blockdiagramm eines synchronen Schaltwerks



Quelle: Moustafa Nawito, 2022.

Kippschaltungen

Kippschaltungen sind die Bezeichnung für Speicherkomponenten, die in zeitgesteuerten sequenziellen Schaltungen eingesetzt werden. In einer Kippschaltung kann ein Bit an Informationen gespeichert werden. Der Ausgang der Kippschaltung ist entweder 0 oder 1, wenn es sich in einem stabilen Zustand befindet. Um die erforderliche Anzahl von Bits zu speichern, kann eine sequenzielle Schaltung mehrere Kippschaltungen verwenden.

Die allgemeine Funktionsweise von Kippschaltungen ist wie folgt zu beschreiben.

- Die Eingänge der Schaltung oder die in der Kippschaltung gespeicherten Werte werden logisch kombiniert, um die Ausgänge zu erzeugen. Darüber hinaus steuern die Eingänge der Schaltung oder die jetzt in der Kippschaltung gespeicherten Werte den Wert, der in einem Speicherelement gespeichert wird, wenn der Taktimpuls auftritt (oder beides).
- Wenn ein Taktimpuls auftritt, wird der neue Wert gespeichert (Aktualisierung der Kippschaltung). Die kombinatorische Logik, die den nächsten Wert erzeugt, muss vor dem Auftreten des Taktimpulses einen stabilen Wert erreicht haben. Daher ist die Arbeitsgeschwindigkeit der kombinatorischen Logikschaltungen von entscheidender Bedeutung. Die kombinatorische Logik muss auf eine Zustandsänderung der Kippschaltung rechtzeitig reagieren, sodass sie vor dem Eintreffen des nächsten Impulses aktualisiert wird, wenn die Taktimpulse (Synchronisierungsimpulse) in regelmäßigen Abständen eintreffen. Die Mindestzeit zwischen den Taktimpulsen, die für eine ordnungsgemäße Funktion der Schaltung erforderlich ist, wird zu einem großen Teil durch die Ausbreitungsverzögerungen bestimmt.
- Nur ein Übergang im Wert der Taktimpulse, z. B. von 0 auf 1, kann eine Zustandsänderung der Kippschaltung bewirken.
- Die Rückkopplungsschleife zwischen dem am Eingang erzeugten Wert und dem darin gespeicherten Wert ist effektiv unterbrochen, wenn es keinen Takt gibt, weil die Ausgänge unempfindlich gegenüber Änderungen an den Ausgängen der kombinatorischen Schaltungen sind, die ihre Eingänge steuern. Infolgedessen ändern sich die Zustände nur zu bestimmten, durch die Taktimpulse angezeigten Zeiten von einem zum nächsten.

Die Anzahl der Eingänge, die jede Art von Speicherelement hat, und die Art und Weise, wie diese Eingänge den binären Zustand beeinflussen, sind die Hauptunterschiede zwischen den Kippschaltungen:

- **Latches** sind Speicherkomponenten, die mit Signalpegeln und nicht mit Signalübergängen arbeiten. Sie sind pegelempfindlich.
- **Flip-Flops** sind Speicherkomponenten, die durch einen Taktübergang gesteuert werden. Sie sind flankenempfindlich.

5.2 Kippschaltungen: Das Latch

Latches sind die grundlegenden Schaltungen, aus denen alle Kippschaltungen aufgebaut sind. Sie sind vorteilhaft für den Entwurf asynchroner sequenzieller Schaltungen und die Speicherung binärer Daten, aber sie sind unpraktisch für die Verwendung als Speicherelemente in synchronen sequenziellen Schaltungen. Der Grund dafür wird im Folgenden erläutert.

SR-Latch

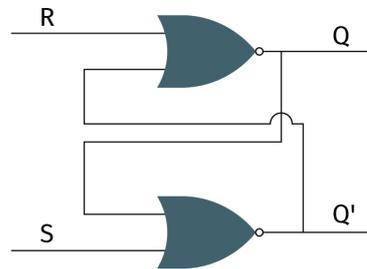
Die SR-Latch-Schaltung besteht aus zwei kreuzgekoppelten NOR- oder NAND-Gattern sowie zwei Eingängen, die mit *S* für set (setzen) und *R* für reset (rücksetzen) gekennzeichnet sind. Die folgende Abbildung zeigt (a) das SR-Latch, das aus zwei kreuzgekoppelten NOR-Gattern besteht und die zugehörige Wahrheitstabelle (bei Latches auch Funktionstabelle genannt). Es gibt zwei nützliche Zustände für das Latch. Das Latch befindet sich im

gesetzten Set-Zustand, wenn der Ausgang $Q = 1$ und $Q' = 0$ ist. Der Reset-Zustand liegt vor, wenn $Q = 0$ und $Q' = 1$ ist. Typischerweise sind Q und Q' Komplemente. Anstatt sich gegenseitig zu ergänzen, sind beide Ausgänge allerdings gleich 0, wenn beide Eingänge gleichzeitig gleich 1 sind. Zusätzlich erreicht das Latch einen metastabilen oder unerwarteten Zustand, wenn beide Eingänge gleichzeitig auf 1 gestellt werden. Daher ist es in realen Anwendungen nicht zulässig, beide Eingänge auf 1 zu setzen.

Unter normalen Bedingungen bleiben beide Eingänge des Latches auf 0, es sei denn, der Zustand muss geändert werden. Das Latch geht in den gesetzten Zustand über, wenn am S -Eingang kurzzeitig eine 1 angelegt wird. Um die Möglichkeit eines undefinierten Folgezustands zu verhindern, der sich aus der verbotenen Eingangsbedingung ergibt, muss der S -Eingang auf 0 zurückkehren, bevor weitere Änderungen vorgenommen werden. Die Schaltung geht in den gesetzten Zustand über, wenn zwei Eingangsbedingungen erfüllt sind, wie in der Funktionstabelle zu sehen ist. Der Eingang S muss der ersten Bedingung entsprechen ($S = 1, R = 0$), damit die Schaltung in den gesetzten Zustand übergeht. Durch kurzzeitiges Anlegen einer 1 an den Eingang R , nachdem beide Eingänge den Wert 0 erreicht haben, ist es möglich, in den Reset-Zustand überzugehen. Die Schaltung bleibt im Reset-Zustand, nachdem die 1 von R entfernt wurde. Die Latch-Schaltung kann sich also entweder im gesetzten oder im Reset-Zustand befinden, wenn beide Eingänge S und R gleich 0 sind, je nachdem, welcher Eingang zuletzt eine 1 enthielt.

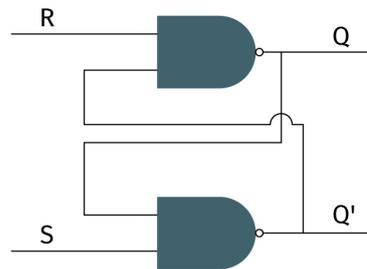
Beide Ausgänge des Latches werden auf 0 gesetzt, wenn eine 1 sowohl an den S - als auch an den R -Eingang angelegt wird. Da der Zustand, der sich aus den Eingangsübergängen ergibt, von der Reihenfolge abhängt, in der sie zu 0 zurückkehren, erzeugt diese Aktion einen undefinierbaren zukünftigen Zustand. Außerdem verstößt sie gegen die Regel, dass Ausgänge komplementär zueinander sein müssen. Indem man verhindert, dass während des normalen Betriebs an beiden Eingängen gleichzeitig Einsen anliegen, wird dieser Umstand vermieden.

Abbildung 39: (a) SR-Latch aus NOR-Gattern, (b) SR-Latch aus NAND-Gattern



S	R	Q	Q'	Kommentar
1	0	1	0	
0	0	1	0	nach S = 1, R = 0
0	1	0	1	
0	0	0	1	nach S = 0, R = 1
1	1	0	0	verboten

(a)



S	R	Q	Q'	Kommentar
1	0	0	1	
1	1	0	1	nach S = 1, R = 0
0	1	1	0	
1	1	1	0	nach S = 0, R = 1
0	0	1	1	verboten

(b)

Quelle: Moustafa Nawito, 2022.

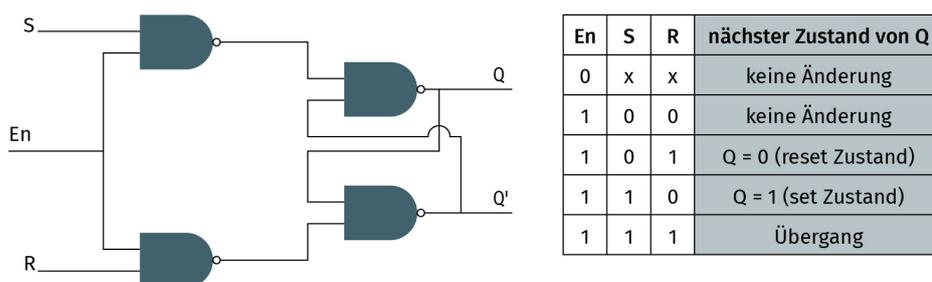
Die obere Abbildung (b) zeigt das SR-Latch mit zwei kreuzweise gekoppelten NAND-Gattern. Solange der Status des Latches nicht geändert werden muss, funktionieren beide Eingänge mit dem Wert 1. Sobald der Eingang S auf 0 gesetzt wird, wechselt der Ausgang Q auf 1 und das Latch geht in den gesetzten Zustand über. Die Schaltung bleibt im gesetzten Zustand, wenn der Eingang S wieder auf 1 gesetzt wird. Durch Eingabe einer 0 in den Eingang R , bevor beide Eingänge wieder auf 1 gesetzt werden, können wir den Zustand des Latches ändern. Die Schaltung geht daraufhin in den Reset-Zustand über und verbleibt dort auch dann, wenn beide Eingänge wieder auf 1 zurückgegangen sind.

Die Eingangskombination, die vermieden werden sollte, ist, wenn beide Eingänge gleich 0 sind, was das NAND-Latch verbietet. Man beachte, dass die Eingangssignale für die NAND-Latch-Schaltungen das Komplement der Werte erfordern, die für die NOR-Latch benötigt werden, wenn man sie mit der NOR-Latch vergleicht. Das NAND-Latch wird gelegentlich auch als $S'R'$ -Latch bezeichnet, da es ein 0-Signal benötigt, um seinen Zustand zu ändern.

Durch Hinzufügen eines zusätzlichen Eingangssignals, das bestimmt (steuert), wann der Zustand der Latch-Schaltung geändert werden kann, indem es festlegt, ob S und R (oder S' und R') die Schaltung beeinflussen können, kann das Verhalten der grundlegenden SR-Latch geändert werden. Die folgende Abbildung zeigt ein SR-Latch mit einem Steuereingang En (engl. enable). Es besteht aus zwei NAND-Gattern zusätzlich zu dem Standard-SR-Latch. Die beiden anderen Eingänge werden durch den Steuereingang En aktiviert. Solange das Freigabesignal auf 0 liegt, bleiben die Ausgänge der NAND-Gatter auf dem Logik-1-Pegel. So verhält sich das SR-Latch, wenn es sich im Ruhezustand befindet. Informationen von den S - oder R -Eingängen können das Latch verändern, wenn der Freigabe-

eingang auf 1 gesetzt ist. Wenn $S = 1$, $R = 0$ und $En = 1$ ist, wird der eingestellte Zustand erreicht (aktiv-hoch Freigabe). Um den Reset-Zustand zu erlangen, müssen die Eingänge $S = 0$, $R = 1$ und $En = 1$ sein. In beiden Fällen behält der Schaltkreis seinen aktuellen Zustand bei, wenn En auf 0 zurückgesetzt wird. Der Steuereingang schaltet den Schaltkreis aus, indem er 0 an En anlegt, wodurch sichergestellt wird, dass der Zustand des Ausgangs unabhängig von den Werten von S und R konstant bleibt. Außerdem bleibt der Zustand der Schaltung unverändert, wenn $En = 1$ ist und die Werte der Eingänge S und R beide 0 sind. Die in der unteren Abbildung beigefügte Funktionstabelle enthält eine Liste dieser Umstände. Wenn alle drei Eingänge gleich sind, ist der Zustand unbestimmt. Das SR-Basis-Latch wird durch diese Bedingung in den undefinierten Zustand übertragen, wodurch an beiden Eingängen 0 gesetzt wird. Da der nächste Zustand davon abhängt, ob der S - oder der R -Eingang zuerst null erreicht, kann nicht mit Sicherheit vorhergesagt werden, was passiert, wenn der Freigabeeingang auf 0 zurückkehrt. Diese Schaltung wird in der Praxis nur selten verwendet, da es schwierig ist, ihren unbestimmten Zustand zu verwalten. Das SR-Latch ist jedoch eine wichtige Schaltung, da sie als Grundlage für andere nützliche Latches und Flip-Flops dient.

Abbildung 40: SR-Latch mit enable

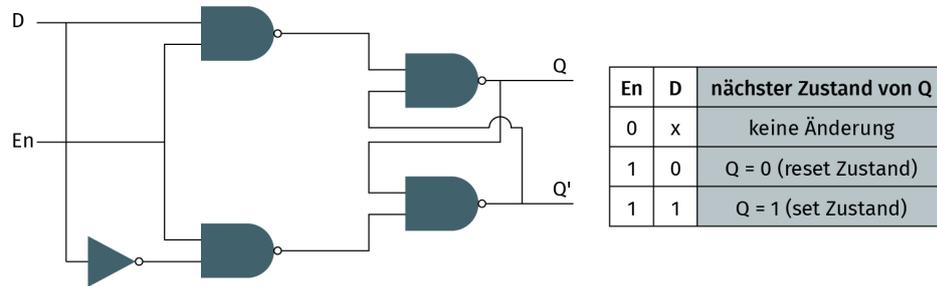


Quelle: Moustafa Nawito, 2022.

D-Latch

Eine Technik, um den unerwünschten Zustand des unbestimmten Zustands im SR-Latch zu beseitigen, besteht darin, dafür zu sorgen, dass die Eingänge S und R niemals gleichzeitig gleich 1 sind. Dies wird im D-Latch in folgender Abbildung umgesetzt. D (Daten) und En sind die einzigen beiden Eingänge für dieses Latch. Unmittelbar nach seinem Komplement am R -Eingang wird der D -Eingang an den S -Eingang gelegt. Beim kreuzgekoppelten SR-Latch liegen beide Eingänge auf dem Pegel 1, während der Freigabeeingang auf 0 liegt. Der Schaltkreis kann seinen Zustand nicht ändern, solange D einen beliebigen Wert hat. Wenn $En = 1$ ist, wird der Eingang D abgetastet. Der Schaltkreis befindet sich im gesetzten Zustand, wenn $D = 1$ ist; in diesem Fall wird der Ausgang Q auf 1 gesetzt. Wenn D gleich 0 ist, wird der Ausgang Q zu 0 und die Schaltung geht in den Reset-Zustand über.

Abbildung 41: Schaltdiagramm und Funktionstabelle vom D-Latch

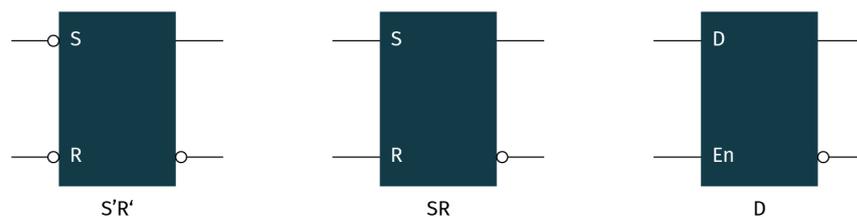


Quelle: Moustafa Nawito, 2022.

Das *D*-Latch wird so genannt, weil es über einen internen Speicher verfügt, der Daten aufnehmen kann. Er eignet sich für die Verwendung als kurzfristiger binärer Datenspeicher zwischen einer Einheit und ihrer Umgebung. Wenn der Freigabeeingang aktiviert ist, werden die am Dateneingang des *D*-Latches anliegenden Binärdaten an den *Q*-Ausgang geliefert. Solange der Freigabeeingang aktiviert ist, wird der Ausgang genauso aktualisiert wie der Dateneingang. Wegen des durch diesen Umstand ermöglichten Kanals vom Eingang *D* zum Ausgang ist die Schaltung häufig als transparentes Latch bezeichnet. Die binären Daten, die zum Zeitpunkt des Übergangs am Dateneingang anlagen, werden am *Q*-Ausgang gehalten (d. h. gespeichert), wenn das Freigabe-Eingangssignal deaktiviert ist, bis der Freigabe-Eingang erneut aktiviert wird. Dabei ist zu beachten, dass der Freigabeeingang mit einem Inverter verbunden sein kann. Danach hat das externe Freigabesignal je nach der physikalischen Schaltung den Wert 0 (aktiv niedrig) oder 1 (aktiv hoch).

Die untere Abbildung stellt die Schaltsymbole vom *SR*-, *S'R'*- und *D*-Latch dar.

Abbildung 42: Schaltsymbole vom *SR*-, *S'R'*- und *D*-Latch



Quelle: Moustafa Nawito, 2022.

5.3 Kippschaltungen: Das Flip-Flop

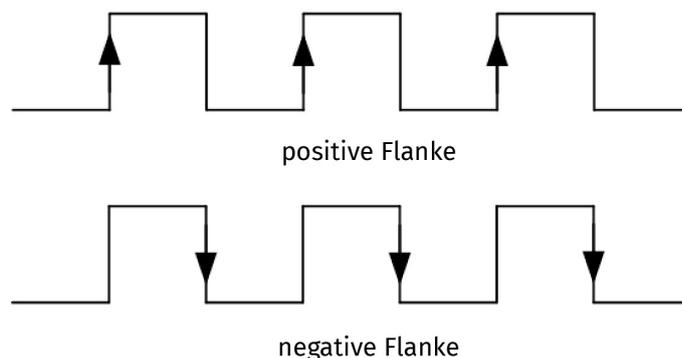
Flip-Flop-Schaltungen sind so konzipiert, dass sie als Bestandteil einer sequenziellen Schaltung, die einen gemeinsamen Taktgeber verwendet, gut funktionieren können. Der Zustand eines Flip-Flops wird durch eine Änderung des Steuereingangs umgeschaltet. Diese temporäre Änderung bezeichnet sich als Trigger. Der dadurch verursachte Übergang löst das Flip-Flop aus.

Die Reaktion eines Latches auf eine Pegeländerung des Taktes ist sein Nachteil. Eine positive Pegelantwort am Freigabeeingang eines *D*-Latches ermöglicht Änderungen am Ausgang, wenn sich der *D*-Eingang ändert, während der Takt bei 1 bleibt. Die Grundlage der korrekten Funktionsweise eines Flip-Flops besteht darin, es nur während einer Signalübertragung auszulösen. Dies kann erreicht werden durch die Beseitigung des Rückkopplungspaths, der für den Betrieb der sequenziellen Schaltung erforderlich ist.

Ein Takt hat zwei Übergänge: Der erste ist der von 0 nach 1 und der zweite der von 1 nach 0. Der positive Übergang wird als positive Flanke und der negative Übergang als negative Flanke bezeichnet, wie in folgender Abbildung zu sehen. Ein Latch kann auf zwei Arten in ein Flip-Flop umgewandelt werden:

- Eine Methode besteht darin, zwei Latches in einer bestimmten Anordnung zu verwenden, die das Flip-Output-Flop isoliert und dafür sorgt, dass es nicht beeinträchtigt wird, während sich der Eingang ändert.
- Eine andere Methode besteht darin, ein Flip-Flop zu erstellen, das nur während der Signalübergänge des Synchronisationssignals (Takt) aktiviert wird (von 0 auf 1 oder von 1 auf 0) und für den Rest des Taktsignals inaktiv bleibt. Im Folgenden wird gezeigt, wie beide Formen von Flip-Flops implementiert werden.

Abbildung 43: Positive Flanke und negative Flanke

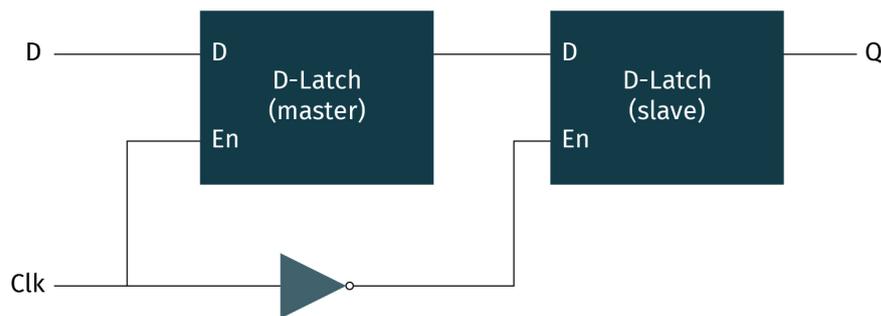


Quelle: Moustafa Nawito, 2022.

Flankengesteuertes D-Flip-Flop

Die untere Abbildung veranschaulicht, wie ein D -Flip-Flop mit zwei D -Latches, einem Inverter und einem D -Flip-Flop aufgebaut ist. Der Master-Latch ist der erste, der Slave-Latch der zweite. Die Schaltung ändert ihren Ausgang Q nur bei der negativen Flanke des Synchronisations- oder Regeltakts und tastet den D -Eingang (bezeichnet als Clk) ab. Der Ausgang des Inverters ist 1, während der Takt auf 0 steht. Der aktivierte Ausgang Q des Slave-Latches ist der gleiche wie der Y des Master-Latches. $Clk = 0$, wodurch das Master-Latch deaktiviert wird. Die Daten vom externen D -Eingang werden an den Master übertragen, wenn der Eingangsimpuls auf den Logik-1-Pegel wechselt. Da jedoch der Freigabe-eingang des Slaves auf 0 gesetzt ist, ist er deaktiviert, wenn der Takt auf Pegel 1 liegt. Der Slave-Ausgang kann durch Änderungen am Eingang nicht beeinflusst werden, wohl aber der Master-Ausgang an Y . Der Master wird ausgeschaltet und vom D -Eingang getrennt, wenn der Taktimpuls null erreicht. Der Wert von Y wird gleichzeitig mit der Freigabe des Slaves an das Flip-Flop am Ausgang von Q gesendet. Daher kann eine Änderung im Flip-Output-Flop nur vor und während des Übergangs des Takts von 1 auf 0 erfolgen.

Abbildung 44: Master-Slave-D-Flip-Flop



Quelle: Moustafa Nawito, 2022.

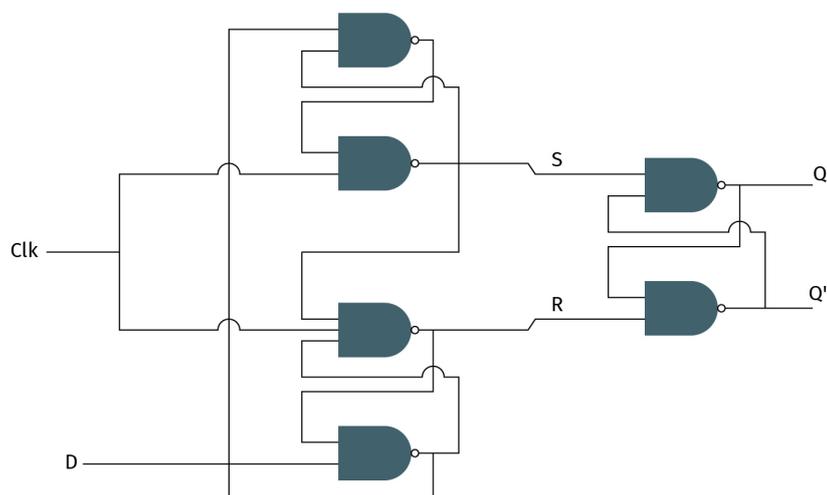
Das Verhalten des Master-Slave-Flip-Flops schreibt vor, dass

- sich der Ausgang nur einmal ändern darf,
- eine Änderung des Ausgangs durch die negative Flanke des Taktgebers ausgelöst wird und
- die Änderung nur während des negativen Pegels des Taktgebers erfolgen darf.

Der Wert, der am Ausgang des Flip-Flops erzeugt wird, ist der Wert, der in der Master-Stufe unmittelbar vor dem Auftreten der negativen Flanke gespeichert wurde. Es ist auch möglich, die Schaltung so zu gestalten, dass sich der Flip-Flop-Ausgang bei der positiven Flanke des Takts ändert. Dies geschieht in einem Flip-Flop, das einen zusätzlichen Inverter zwischen der Clk-Klemme und der Verbindung zwischen dem anderen Inverter und dem Eingang En des Master-Latches hat. Ein solches Flip-Flop wird mit einem negativen Impuls getriggert, sodass die negative Flanke des Taktes auf den Master und die positive Flanke auf den Slave und die Ausgangsklemme wirkt.

Wie in der folgenden Abbildung zu sehen ist, verwendet ein anderes flankengetriggertes D -Flip-Flop-Design drei SR-Latches. Die externen Eingänge D (Daten) und Clk (Takt) werden von zwei Latches verarbeitet. Die Flip-Outputs des Flops werden vom dritten Latch bereitgestellt. Wenn $Clk = 0$ ist, werden die S - und R -Eingänge des Ausgangslatches auf dem Logik-1-Pegel gehalten. Der Ausgang bleibt also in seinem aktuellen Zustand. Der Eingang D kann entweder als 0 oder 1 eingegeben werden. Wenn Clk auf 1 ansteigt, wird R zu 0, wenn $D = 0$ ist. Da das Flip-Flop dadurch in den Reset-Zustand übergeht, wird Q zu 0. Die Klemme R bleibt auf 0, wenn sich der Eingang D ändert, wenn $Clk = 1$ ist, da $Q = 0$ ist. Dadurch ist das Flip-Flop gesperrt und reagiert nicht mehr auf Eingangsänderungen. Die Ausgangsverriegelung wird in den Ruhezustand versetzt, ohne den Ausgang zu beeinflussen, wenn der Takt auf 0 zurückgesetzt wird, was dazu führt, dass R auf 1 geht. Das Gleiche gilt, wenn $D = 1$ ist, wenn Clk von 0 auf 1 geht und $S = 0$ wird. Infolgedessen geht die Schaltung in den gesetzten Zustand über und Q ist gleich 1. Solange $Clk = 1$ ist, hat jede Änderung von D keinen Einfluss auf das Ergebnis.

Abbildung 45: D-Flip-Flop mit positiver Flankenauflösung



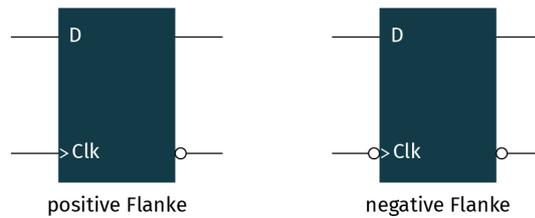
Quelle: Moustafa Nawito, 2022.

Beim Einsatz von flankengetriggerten Flip-Flops ist das Timing der Flip-Flops auf die Eingangsdaten und den Takt zu berücksichtigen. Vor dem Auftreten des Taktübergangs muss der D -Eingang für eine Mindestzeit, die sogenannte Setup-Zeit, auf einem konstanten Wert gehalten werden. Analog dazu muss der D -Eingang nach dem positiven Taktübergang mindestens eine Clock-Periode lang unverändert bleiben, die sogenannte Haltezeit. Die Dauer zwischen der Triggerflanke und der Stabilisierung des Ausgangs in einem neuen Zustand wird als Verzögerungszeit des Flip-Propagation-Flops bezeichnet. Die Datenbücher der Hersteller für bestimmte Logikfamilien enthalten Informationen zu diesen und anderen Faktoren.

Die folgende Abbildung zeigt die visuelle Darstellung des flankengetriggerten D -Flip-Flops. Mit Ausnahme des pfeilartigen Symbols vor dem Buchstaben Clk , das einen dynamischen Eingang kennzeichnet, ist es dem Symbol für das D -Latch sehr ähnlich. Das Flip-

Flop reagiert auf den Flankenwechsel des Taktgebers, was durch den dynamischen Indikator (>) angezeigt wird. Eine negative Flanke zur Auslösung des Schaltkreises ist durch eine Blase außerhalb des Blocks und neben dem dynamischen Indikator darstellbar. Eine Reaktion mit einer positiven Flanke wird durch das Fehlen einer Blase angezeigt.

Abbildung 46: Schaltsymbol des D-Flip-Flops



Quelle: Moustafa Nawito, 2022.

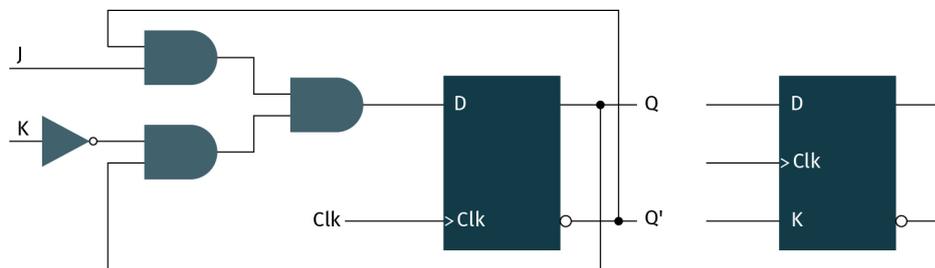
JK- und T-Flip-Flop

Die folgende Abbildung veranschaulicht das Schaltbild eines JK -Flip-Flops, das aus Gattern und einem D -Flip-Flop besteht, sowie das Schaltsymbol. Wenn beide Eingänge aktiviert sind, ist der Flip-Flop-Ausgang komplementär. Der J -Eingang setzt das Flip-Flop auf 1 und der K -Eingang setzt es auf 0 zurück. Die Untersuchung der an den D -Eingang angeschlossenen Schaltung ermöglicht es, dies zu bestätigen:

$$D = JQ' + K'Q \quad (5.1)$$

Die folgende Taktflanke setzt den Ausgang auf 1, wenn $J = 1$ und $K = 0$, d. h., wenn $D = Q' + Q = 1$ ist. Insofern gleichzeitig $D = 0$ und $J = 0$ sind, wird der Ausgang durch die folgende Taktflanke auf 0 zurückgesetzt. Sobald $J = K = 1$ und $D = Q'$ ist, wird der Ausgang mit der folgenden Taktflanke abgeschlossen. Wenn $D = Q$ und $J = K = 0$ ist, verändert die Taktflanke den Ausgang nicht.

Abbildung 47: JK-Flip-Flop



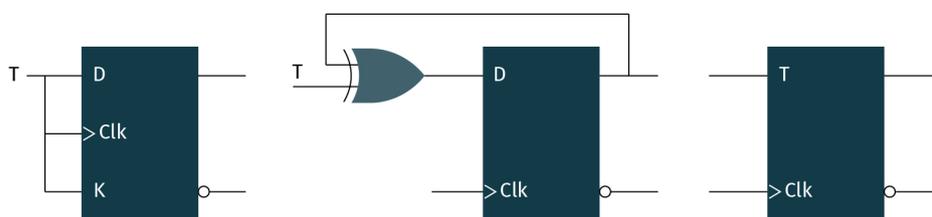
Quelle: Moustafa Nawito, 2022.

Wenn die Eingänge J und K in einem JK -Flip-Flop miteinander verbunden sind, ist der Ausgang das komplementäre Flip-Flop, das als T (engl. toggle, umschalten) bekannt ist. Die folgende Abbildung veranschaulicht dies. Der Ausgang wird von einer Taktflanke nicht beeinflusst, wenn $T = 0$ ($J = K = 0$). Der Ausgang ist die Taktflanke bei $T = 1$ ($J = K = 1$). Binäre Zähler können mit dem Komplementär-Flip-Flop entworfen werden. Ein D -Flip-Flop und ein Exklusiv-OR-Gatter können verwendet werden, um ein T -Flip-Flop zu erstellen. Der Ausdruck für den D -Eingang ist wie folgt:

$$D = T \oplus Q' = TQ' + KT'Q \quad (5.2)$$

$D = Q$ und der Ausgang bleibt unverändert, wenn $T = 0$. $D = Q'$ und der Ausgang ist komplementär, wenn $T = 1$. Ein T -Symbol erscheint im Eingang des grafischen Symbols für diese Kippschaltung.

Abbildung 48: T-Flip-Flop



Quelle: Moustafa Nawito, 2022.

An dieser Stelle sei darauf hingewiesen, dass das Flip-Flop, das in aktuellen VLSI-Schaltungen am effizientesten und wirtschaftlichsten aufgebaut ist, das D -Flip-Flop ist, da es die geringste Anzahl von Logikgattern zur Realisierung benötigt. Das T -Flip-Flop und das JK -Flip-Flop sind weniger weit verbreitet, weshalb wir uns im Folgenden auf die Behandlung von sequenziellen Schaltungen beschränken, die auf dem D -Flip-Flop basieren. Interessierte Leser:innen können mehr über Schaltungen, die auf anderen Typen basieren, in Referenzen, wie Fricke (2021) und Gehrke et al. (2016), erfahren.

Charakteristische Tabellen und Gleichungen von Flip-Flops

Die logischen Eigenschaften eines Flip-Flops und seine Funktionsweise werden in der sogenannten charakteristischen Tabelle dargestellt. Diese Tabelle definiert den nächsten Zustand $Q(n+1)$ (manchmal auch mit der Zeitvariable t geschrieben als $Q(t+1)$), der sich aus dem Übergang des Taktgebers ergibt, in Abhängigkeit von den Eingängen und dem aktuellen Zustand $Q(n)$ (auch $Q(t)$). Der aktuelle Zustand ist der Zustand, der vor dem Anlegen der Taktflanke eingetreten ist, und der nächste Zustand ist der Zustand, der eine Taktperiode später eintritt. In englischsprachigen Textbüchern werden $Q(n)$ und $Q(n+1)$ jeweils als „current state“ und „next state“ bezeichnet.

Die folgende Tabelle zeigt die Eigenschaften eines D -Flip-Flops. Der Zustand eines D -Flip-Flops ist unabhängig vom aktuellen Zustand und hängt ausschließlich vom D -Eingang ab. Zur Darstellung kann $Q(n+1) = D$ verwendet werden. Es bedeutet, dass der Wert des

nächsten Zustands gleich dem Wert von D ist. Es ist anzumerken, dass das D -Flip-Flop keine „keine-Änderung“-Bedingung aufweist. Ein solcher Zustand kann erreicht werden, indem man entweder die Uhr ausschaltet oder die Uhr laufen lässt, indem man die Flip-Output-Flops mit dem D -Eingang verbindet. Wenn die Flip-Flop-Zustände konstant bleiben müssen, wird der Ausgang in beiden Fällen erfolgreich umgewandelt.

Tabelle 18: Charakteristische Tabelle des D-Flip-Flops

D	$Q(n+1)$
1	0 Reset
1	1 Set

Quelle: Moustafa Nawito, 2022.

Eine charakteristische Gleichung kann verwendet werden, um die Merkmale der Kippstufe, wie sie in der charakteristischen Tabelle aufgeführt sind, algebraisch zu erklären. Für die D -Flip-Flop-Schaltung haben wir die charakteristische Gleichung

$$Q(n+1) = D \quad (5.3)$$

Sie besagt, dass der Wert des Eingangs D im aktuellen Zustand der Wert des Ausgangs im folgenden Zustand sein wird.

5.4 Analyse getakteter Schaltwerke

Eine Analyse erklärt, was eine bestimmte Schaltung unter verschiedenen Betriebsbedingungen leisten wird. Die Eingänge, Ausgänge und der Flip-Flop-Zustand einer zeitgesteuerten sequenziellen Schaltung wirken sich alle auf ihr Verhalten aus. Die Eingänge und der aktuelle Zustand bestimmen sowohl die Ausgänge als auch den nachfolgenden Zustand. Die Erstellung einer Tabelle oder eines Diagramms für die zeitliche Abfolge der Eingänge, Ausgänge und internen Zustände ist der erste Schritt bei der Analyse einer sequenziellen Schaltung. Zusätzlich können Boolesche Ausdrücke verwendet werden, um auszudrücken, wie sich die sequenzielle Schaltung verhält. Diese Ausdrücke müssen entweder explizit oder implizit die erforderliche zeitliche Abfolge enthalten. Im Folgenden werden diese verschiedenen Analysewerkzeuge für sequenzielle Schaltungen vorgestellt.

Die Zustandsgleichung

Zustandsgleichungen können verwendet werden, um das Verhalten einer getakteten sequenziellen Schaltung algebraisch zu erklären. Als Funktion des aktuellen Zustands und der Eingänge gibt eine Zustandsgleichung – auch bekannt als Übergangsgleichung – den nachfolgenden Zustand an. Nehmen wir die sequenzielle Schaltung in nachfolgender Abbildung als Beispiel. Da der D -Eingang eines Flip-Flops den Wert des nächsten Zustands bestimmt (d. h. den Zustand, der nach dem Taktübergang erreicht wird), ist es möglich, eine Reihe von Zustandsgleichungen für die Schaltung zu schreiben:

$$A(n+1) = A(n) \cdot B(n) + x$$

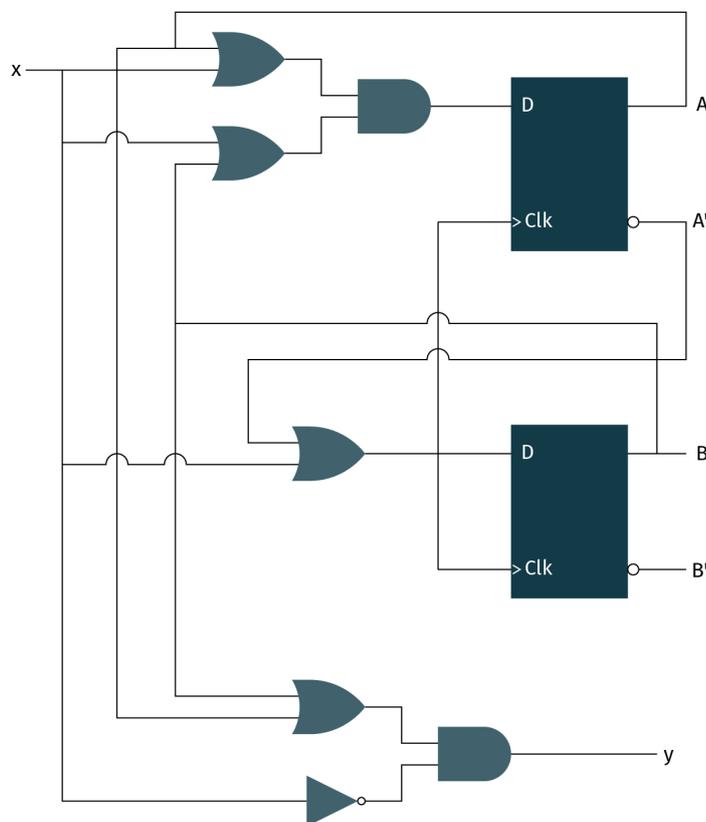
$$B(n+1) = A'(n) + x$$

Die linke Seite zeigt die nächsten Zustände für die Flip-Flops A und B (daher die Bezeichnung $n+1$). Die rechte Seite veranschaulicht die aktuellen Zustände und die Eingänge (in diesem Fall gibt es nur einen Eingang x). Da klar ist, dass die rechte Seite den Zustand des Stroms repräsentiert, können die Gleichungen der Einfachheit halber wie folgt geschrieben werden.

$$A(n+1) = AB + x$$

$$B(n+1) = A' + x$$

Abbildung 49: Sequenzielle Schaltung zur Analyse



Quelle: Moustafa Nawito, 2022.

In ähnlicher Weise kann die Zustandsgleichung für den Ausgang y wie folgt geschrieben werden.

$$y(n) = (A(n) \cdot x') + (B(n) \cdot x')$$

oder in vereinfachter Form als

$$y = Ax' + Bx'$$

Die Zustandstabelle

Eine Zustandstabelle kann das Timing der Eingänge, Ausgänge und Flip-Flop-Zustände auflisten (manchmal auch als Übergangstabelle bezeichnet). Folgende Tabelle zeigt die Zustandstabelle für die Schaltung in obiger Abbildung. Die Tabelle ist in vier Abschnitte unterteilt: Ausgang, Eingang, nächster Zustand und aktueller Zustand. Die aktuellen Zustände der Flip-Flops A und B werden im Abschnitt „aktueller Zustand“ angezeigt. Jeder potenzielle aktuelle Zustand wird durch einen x -Wert im Eingangsbereich dargestellt. Die Zustände der Flip-Flops zum Zeitpunkt $n + 1$ sind einen Taktzyklus später, im Abschnitt „nächster Zustand“ repräsentiert. Der Wert von y zum Zeitpunkt n für jeden aktuellen Zustand und jede Eingangsbedingung ist dem Ausgangsteil zu entnehmen. Für die Erstellung einer Zustandstabelle ist die Auflistung aller möglichen binären Kombinationen der aktuellen Zustände und Eingänge erforderlich. In diesem Fall haben wir acht binäre Möglichkeiten, die von 000 bis 111 reichen. Die nächsten Zustandswerte werden dann aus dem Logikdiagramm oder aus den Zustandsgleichungen ermittelt.

Tabelle 19: Zustandstabelle der Schaltung zur Analyse

aktueller Zustand		Eingang	nächster Zustand		Ausgang
$A(n)$	$B(n)$	x	$A(n + 1)$	$B(n + 1)$	Y
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0

Quelle: Moustafa Nawito, 2022.

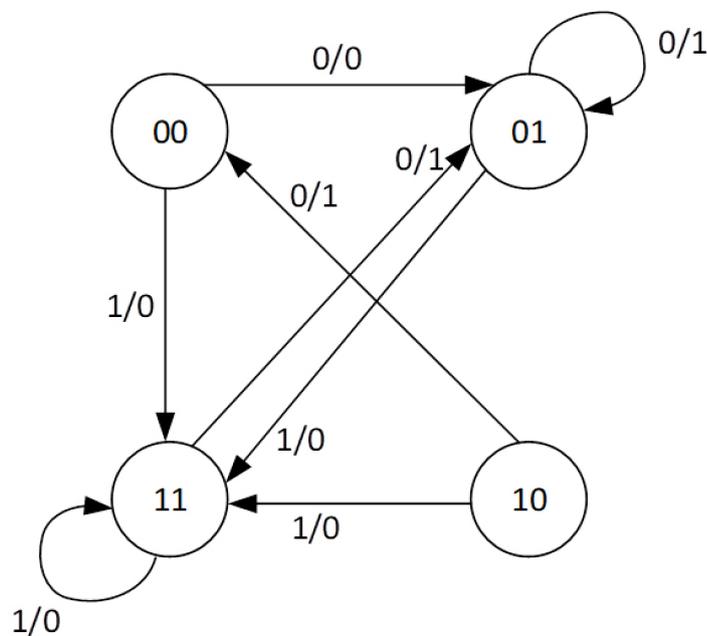
Im Allgemeinen benötigt die Zustandstabelle 2^{m+n} Zeilen für eine sequenzielle Schaltung mit m Flip-Flops und n Eingängen. Unter den Spalten „aktueller Zustand“ und „Eingang“ wird eine Liste von Binärzahlen von 0 bis 2^{m+n-1} angezeigt. Der Teil „nächster Zustand“ enthält m Spalten, eine für jedes Flip-Flop. Die Zustandsgleichungen dienen als direkte Quelle der Binärwerte für den nächsten Zustand. Im Ausgangsteil gibt es genauso viele Spalten wie Ausgangsvariablen vorhanden sind. Ähnlich wie bei einer Wahrheitstabelle wird der Binärwert aus der Schaltung oder aus der Booleschen Funktion abgeleitet.

Das Zustandsdiagramm

Eines der wichtigsten und am weitesten verbreiteten Hilfsmittel für die Analyse und Visualisierung von sequenziellen Schaltungen ist das Zustandsdiagramm (engl. state diagram). Es enthält im Prinzip die gleichen Informationen wie die Zustandstabelle, allerdings in einer bildlichen Form. Dies erleichtert die visuelle Analyse der Schaltung für den Menschen. Das Zustandsdiagramm wird in den folgenden Schritten aufgebaut.

- Für jeden Zustand wird ein Kreis gezeichnet, in den der Name oder der Code des Zustands geschrieben wird. Wie in folgender Abbildung 53 zu sehen ist, hat das Zustandsdiagramm für die in obiger Abbildung (Sequenzielle Schaltung zur Analyse) dargestellte Schaltung vier Zustände, nämlich 00, 01, 10 und 11, wie in der Zustandstabelle angegeben.
- Um einen Übergang von einem Zustand zu einem anderen anzuzeigen, z. B. von Zustand 00 zu Zustand 01, wird ein Pfeil zwischen zwei Zuständen gezeichnet, wobei der Pfeil auf den Zielzustand weist. Dies gilt auch für den Fall, dass keine Änderung vorliegt, d. h., dass ein Zustand in sich selbst übergeht, wie bei Zustand 01.
- Auf den Pfeilen, die Zustandsübergänge anzeigen, wird der Wert des Eingangs (x), der diesen Übergang verursacht hat, gefolgt von einem Schrägstrich und dem Wert des Ausgangs (y), der an diesem Eingang in diesem Zustand aufgetreten ist, geschrieben (d. h. in der Form x/y). Wenn also z. B. für den Zustand 01 der Eingang 0 ist, ist der Ausgang 1 und der Übergang erfolgt wieder zu sich selbst, daher wird in dieser Zeile 0/1 geschrieben, während der Übergang von Zustand 01 zu Zustand 11 mit 1/0 bezeichnet wird.

Abbildung 50: Zustandsdiagramm



Quelle: Moustafa Nawito, 2022.

Flip-Flop Eingangsgleichungen

Flip-Flops und Gatter generieren das Logikdiagramm einer sequenziellen Schaltung. Die Verbindungen zwischen den Gattern bilden eine kombinatorische Schaltung, die algebraisch mit Booleschen Ausdrücken definiert werden kann. Die für die Erstellung des Logikdiagramms der sequenziellen Schaltung erforderlichen Informationen sind die Kenntnis der Art der Flip-Flops und eine Liste der Booleschen Ausdrücke der kombinatorischen Schaltung. Eine Gruppe von Booleschen Funktionen, die sogenannten Ausgangsgleichungen, werden verwendet, um den Teil der kombinatorischen Schaltung, der externe Ausgänge erzeugt, algebraisch zu charakterisieren. Eine weitere Gruppierung Boolescher Funktionen, die sogenannten „Flip-Flop-Eingangsgleichungen“ (engl. excitation equations), liefern eine algebraische Beschreibung des Teils der Schaltung, der die Eingänge für Flip-Flops erzeugt. Es ist üblich, die Variable der Eingangsgleichung mit dem Symbol für den Flip-Flop-Eingang und den Namen des Flip-Flop-Ausgangs mit einem tiefgestellten Index zu bezeichnen. Die folgende Eingangsgleichung spezifiziert beispielsweise ein OR-Gatter mit den Eingängen x und y , das mit dem D -Eingang eines Flip-Flops mit einem Q -bezeichneten Ausgang verbunden ist:

$$DQ = x + y$$

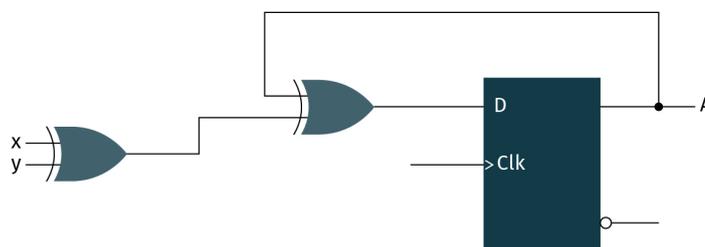
Für die in obiger Abbildung (Sequenzielle Schaltung zur Analyse) dargestellte Schaltung lassen sich die Eingangsgleichungen demnach wie folgt formulieren.

$$DA = (A + x)(B + x)$$

$$DB = A' + x$$

Umgekehrt kann mithilfe der Eingangsgleichungen eines Flip-Flops aus dem Booleschen Ausdruck ein schematisches Diagramm extrahiert werden. Zum Beispiel besagt die Eingangsgleichung $DA = A \oplus x \oplus y$ einfach, dass der D -Eingang eines D -Flip-Flops mit der Bezeichnung A durch die Anwendung von XOR auf x und y und dann XOR auf das Ergebnis mit A versorgt wird, wie in der unteren Abbildung dargestellt.

Abbildung 51: Schaltdiagramm basierend auf Eingangsgleichungen



Quelle: Moustafa Nawito, 2022.

5.5 Entwurf getakteter Schaltwerke

Entwurfstechniken oder -prozesse legen die Hardware fest, die ein gewünschtes Verhalten ausführen soll. Der Entwurf kleinerer Schaltungen kann manuelle Arbeit erfordern, aber die Industrie verlässt sich auf automatisierte Synthesetechnologien, um große integrierte Schaltungen zu erstellen. Das D -Flip-Flop ist der sequenzielle Baustein, der von Synthesetools verwendet wird. In Verbindung mit zusätzlichen Schaltungen kann es die Funktionen von JK - und T -Flip-Flops emulieren. Die Art des Flip-Flops ist für Designer:innen eigentlich kaum von Belang; stattdessen konzentrieren sie sich auf die genaue Spezifikation der sequenziellen Funktionalität, die vom Synthesewerkzeug implementiert werden soll. Hier werden wir D -Flip-Flops verwenden, um den manuellen Weg zu demonstrieren.

Eine Liste von Anforderungen ist der Ausgangspunkt für die Konstruktion einer zeitgesteuerten sequenziellen Schaltung, die entweder zu einem Logikdiagramm oder einer Liste von Booleschen Funktionen führt, aus denen das Logikdiagramm abgeleitet werden kann. Eine sequenzielle Schaltung erfordert eine Zustandstabelle für ihre Spezifikation, im Gegensatz zu einer kombinatorischen Schaltung, die vollständig durch eine Wahrheitstabelle angegeben werden kann. Eine Zustandstabelle oder eine analoge Darstellung, wie ein Zustandsdiagramm, ist der erste Schritt beim Entwurf sequenzieller Schaltungen.

Flip-Flops und kombinatorische Gatter bilden eine synchrone sequenzielle Schaltung. Die Schaltung wird entworfen, indem man zunächst die Flip-Flops auswählt und dann eine kombinatorische Gatterstruktur ermittelt, die in Kombination mit den Flip-Flops zu einer Schaltung führt, die die angegebenen Anforderungen erfüllt. Die Anzahl der in der Schaltung benötigten Zustände und die Auswahl der Zustandszuweisungscodes bestimmen die Anzahl der Flip-Flops. Durch den Vergleich der Eingangs- und Ausgangsgleichungen der Flip-Flops wird die kombinatorische Schaltung aus der Zustandstabelle generiert. Nachdem die Art und die Anzahl der Flip-Flops festgelegt sind, etabliert sich der Entwurfsprozess von einem sequenziellen zu einem kombinatorischen Schaltungsproblem. Die Methoden des kombinatorischen Schaltungsentwurfs können auf diese Weise genutzt werden.

Die empfohlenen Schritte zum Aufbau synchroner sequenzieller Schaltungen lassen sich wie folgt zusammenfassen:

1. Erstellen Sie ein Zustandsdiagramm für die Schaltung unter Verwendung der Wortbeschreibung und der Spezifikationen der gewünschten Operation.
2. Falls erforderlich, sollte die Anzahl der Zustände reduziert werden.
3. Geben Sie den Zuständen Binärwerte.
4. Ermitteln Sie die Zustandstabelle in binärer Form.
5. Entscheiden Sie sich für die Art der Flip-Flops, die Sie verwenden wollen.
6. Bestimmen Sie die kondensierten Eingangs- und Ausgangsgleichungen für Flip-Flops.
7. Erstellen Sie das logische Flussdiagramm.

Reduzierung der Zustände

Das Eingangs- und Ausgangsverhalten von zwei sequenziellen Schaltungen mag identisch sein, wobei ihre Zustandsdiagramme jedoch eine unterschiedliche Anzahl interner Zustände aufweisen können. Bestimmte Merkmale sequenzieller Schaltungen können

einen Entwurf vereinfachen, indem die Anzahl der verwendeten Gatter und Flip-Flops verringert wird. Im Allgemeinen sinken die Kosten einer Schaltung mit zunehmender Anzahl von Flip-Flops. Algorithmen zur Verringerung der Anzahl der Zustände in einer Zustandstabelle bei gleichzeitiger Beibehaltung der externen Eingabe-/Ausgabeanforderungen werden als Zustandsreduktionsalgorithmen bezeichnet. Eine Verringerung der Anzahl der Zustände kann (oder auch nicht) zu einer Verringerung der Anzahl der Flip-Flops führen, da m Flip-Flops 2^m Zustände erzeugen. Die identische Schaltung (mit weniger Flip-Flops) benötigt gelegentlich zusätzliche kombinatorische Gatter, um ihren nächsten Zustand und ihre Ausgangslogik zu realisieren, was ein unvorhergesehenes Ergebnis der Flip-Flop-Reduzierung darstellt.

Zwei Zustände gelten als äquivalent, wenn sie für jedes Mitglied der Eingangsgruppe genau dieselbe Ausgabe erzeugen und die Schaltung entweder in denselben oder in einen äquivalenten Zustand versetzen, entsprechend des Algorithmus' für die Zustandsreduktion einer vollständig angegebenen Zustandstabelle. Wenn zwei Zustände gleichwertig sind, kann einer von ihnen eliminiert werden, ohne dass sich die Art und Weise ändert, wie Eingang und Ausgang miteinander verbunden sind.

BEISPIEL: ZUSTANDSREDUZIERUNG

Als Beispiel betrachten wir die sequenzielle Schaltung, die durch die Zustandstabelle in folgender Tabelle beschrieben wird. Wie angegeben, sind die Zustände 00 und 01 identisch, da sie sowohl bei $x = 0$ als auch bei $x = 1$ in denselben Zustand übergehen und denselben Ausgang y erzeugen. In diesem Fall kann ein Zustand entfernt werden, ohne die Funktionalität der sequenziellen Schaltung zu beeinträchtigen.

Tabelle 20: Reduzierbares Zustandsdiagramm

aktueller Zustand		Eingang	nächster Zustand		Ausgang
$A(n)$	$B(n)$	x	$A(n+1)$	$B(n+1)$	Y
0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	0	0	1

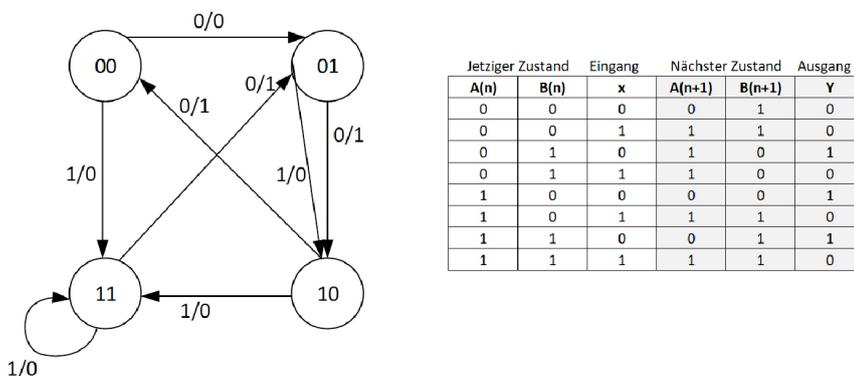
aktueller Zustand		Eingang	nächster Zustand		Ausgang
$A(n)$	$B(n)$	x	$A(n+1)$	$B(n+1)$	Y
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0

Quelle: Moustafa Nawito, 2022.

Entwurfsbeispiel

Wir betrachten die durch das Zustandsdiagramm und die Zustandstabelle in folgender Abbildung beschriebene sequenzielle Schaltung.

Abbildung 52: Entwurfsbeispiel



Quelle: Moustafa Nawito, 2022.

Die vier Zustände werden durch zwei D -Flip-Flops dargestellt, deren Ausgänge mit A und B bezeichnet werden. Es gibt einen Ausgang y und einen Eingang x . Die Werte des nächsten Zustands in der Zustandstabelle beschreiben die D -Eingangsbedingungen für das Flip-Flop, da die charakteristische Gleichung des D -Flip-Flops $Q(n+1) = D_Q$ ist. Die Spalten der nächsten Zustände von A und B können verwendet werden, um die Eingangsgleichungen des Flip-Flops zu extrahieren, die dann wie folgt in Form einer Zwischensumme angegeben werden können.

$$A(n+1) = D_A(A, B, x) = \sum(1, 2, 3, 5, 7)$$

$$B(n+1) = D_B(A, B, x) = \sum(0, 1, 5, 6, 7)$$

$$y(A, B, x) = \sum(2, 4, 6)$$

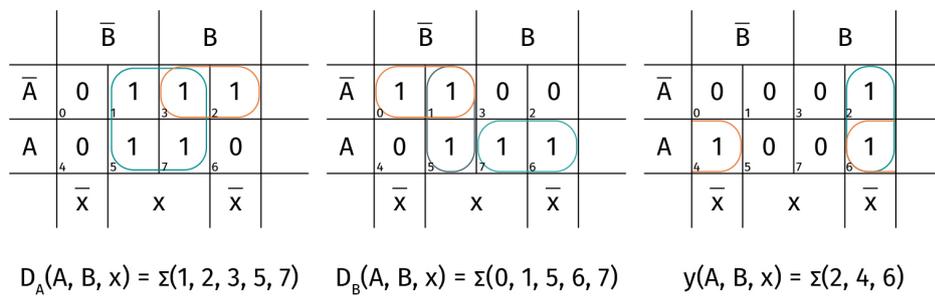
Die Booleschen Gleichungen werden mithilfe der in folgender Abbildung dargestellten KV-Diagramme vereinfacht. Die vereinfachten Gleichungen lauten:

$$DA = A'B + x$$

$$DB = A'B' + AB + B'x$$

$$y = Ax' + Bx'$$

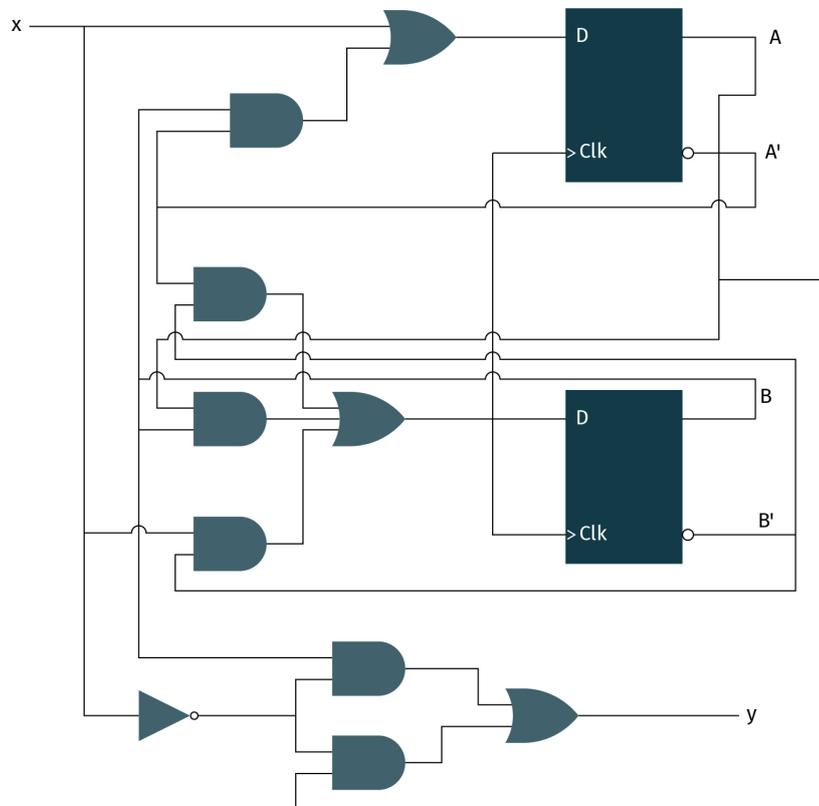
Abbildung 53: KV-Diagramme für D_A , D_B und y



Quelle: Moustafa Nawito, 2022.

Schließlich kann das schematische Diagramm der entworfenen sequenziellen Schaltung, wie in folgender Abbildung gezeigt, konstruiert werden.

Abbildung 54: Schaltplan des Entwurfsbeispiels



Quelle: Moustafa Nawito, 2022.



ZUSAMMENFASSUNG

Sequenzielle Schaltungen zeichnen sich dadurch aus, dass sie einen kombinatorischen Teil und ein Speicherelement in einer Rückkopplungsschleife besitzen. Dies macht die Ausgabe von aktuellen und vorherigen Zuständen abhängig.

Das Latch ist das einfachste Speicherelement und kann ein einzelnes Bit speichern. Es gibt verschiedene Arten von Latches, die jedoch alle den gleichen Nachteil aufweisen, nämlich dass sie pegelabhängig sind. Daher sind sie eher für asynchrone Anwendungen geeignet.

Flip-Flops basieren auf Latches, sind aber so konstruiert, dass sie nur flankenempfindlich sind. Dadurch sind sie zuverlässiger und für synchrone sequenzielle Schaltungen geeignet. Das D-Flip-Flop ist der am weitesten verbreitete Typ von allen.

Die Analyse sequenzieller Schaltungen hängt nicht nur von der Kenntnis der Eingangswerte ab, wie dies bei der kombinatorischen Logik der Fall ist, sondern vor allem vom Zeitablauf und vom Übergang zwischen verschiedenen Zuständen. Aus diesem Grund wurden verschiedene Werkzeuge für die korrekte Analyse solcher Schaltungen entwickelt, darunter das Zustandsdiagramm.

Der Entwurf sequenzieller Schaltungen ist von der Extraktion der richtigen Gleichungen und der Erstellung der Zustandstabelle abhängig. Nach der Reduzierung der Anzahl der Zustände wird der Entwurf ähnlich wie bei kombinierten Schaltungen fortgesetzt, wobei KV-Diagramme verwendet und die vereinfachten Gleichungen erstellt werden, die die Grundlage für die Konstruktion des Schaltplans bilden.

LEKTION 6

REGISTER, ZÄHLER UND AUTOMATEN

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die Funktionsweise und Struktur von verschiedenen Registerarten zu erkennen.
- die Verbindung zwischen Zähler und Register wiederzugeben.
- die Eigenschaften verschiedener Zähler zu differenzieren.
- die Theorie der Automaten zu verstehen.
- die Modellierung von sequenziellen Schaltungen mittels Automaten durchzuführen.

6. REGISTER, ZÄHLER UND AUTOMATEN

Einführung

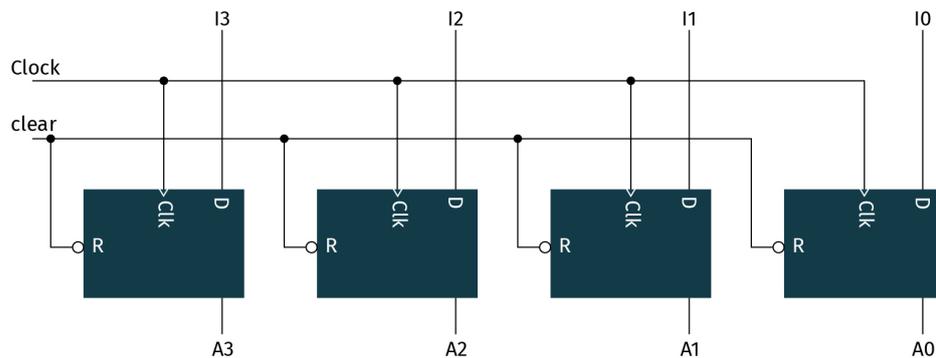
Die Fähigkeit, digitale Daten zu speichern, ist eine wesentliche Funktion digitaler Systeme und kann nur durch den Einsatz sequenzieller Schaltungen erreicht werden. Obwohl das Flip-Flop das grundlegende Speicherelement ist, ist es für den praktischen Einsatz allein nicht geeignet. Register sind sequenzielle Schaltungen, die auf Flip-Flops basieren. Da sie die Realisierung praktischer Speichermodule ermöglichen, sind sie ein wichtiger Baustein in allen digitalen Systemen. Darüber hinaus sind sie die Basis, auf der nützliche Schaltungen wie Zähler realisiert werden können. In dieser Lektion befassen wir uns mit Registern und Zählern und stellen ihren Aufbau und ihre verschiedenen Funktionsweisen vor. Außerdem wird das Konzept der Automaten eingeführt, welches ein wichtiges Werkzeug zur Modellierung jeder Art von sequenziellen Schaltungen darstellt.

6.1 Register

Ein Register besteht aus einer Gruppe von Flip-Flops, von denen jedes einen Taktgeber hat und ein Bit an Informationen speichern kann. Ein n -Bit-Register ist ein Satz von n Flip-Flops, die n Bits binärer Daten speichern können. Neben den Flip-Flops kann ein Register auch kombinatorische Gatter enthalten, die bestimmte Datenverarbeitungsaufgaben übernehmen. Ein Register besteht im weitesten Sinne aus einer Reihe von Flip-Flops und den Gattern, die sie steuern. Flip-Flops speichern die Binärdaten, während die Gatter regeln, wie sie in das Register gesendet werden.

Register gibt es in einer Vielzahl von Typen. Die einfachste Art von Registern hat nur Flip-Flops und keine Gatter. Die folgende Abbildung veranschaulicht den Aufbau eines Vier-Bit-Datenspeicherregisters dieses Typs mit vier D -Flip-Flops. Die Flip-Flops werden alle auf der positiven Flanke jedes Impulses durch den gemeinsamen Takteingang aktiviert, wodurch auch die Binärdaten an den vier Eingängen in das Register übertragen werden. Auf der Grundlage des Wertes von (I_3, I_2, I_1, I_0) unmittelbar vor der Taktflanke wird der Wert von (A_3, A_2, A_1, A_0) nach der Taktflanke berechnet. Durch das Abtasten der vier Ausgänge kann jederzeit auf die im Register gespeicherten Binärdaten zugegriffen werden. Alle vier Flip-Flops erhalten den Eingang Clear an ihren Active-Low-Eingängen R (Reset). Wenn dieser Eingang gleich 0 ist, werden alle Flip-Flops asynchron zurückgesetzt. Der Clear-Eingang kann verwendet werden, um das Register zu löschen (alle Flip-Flops in Reset, alle Ausgänge sind 0), bevor das Register seine zeitgesteuerte Operation ausführt. Während des gesamten Verlaufs einer typischen zeitgesteuerten Operation müssen die R -Eingänge auf logisch 1 bleiben.

Abbildung 55: 4-Bit-Register



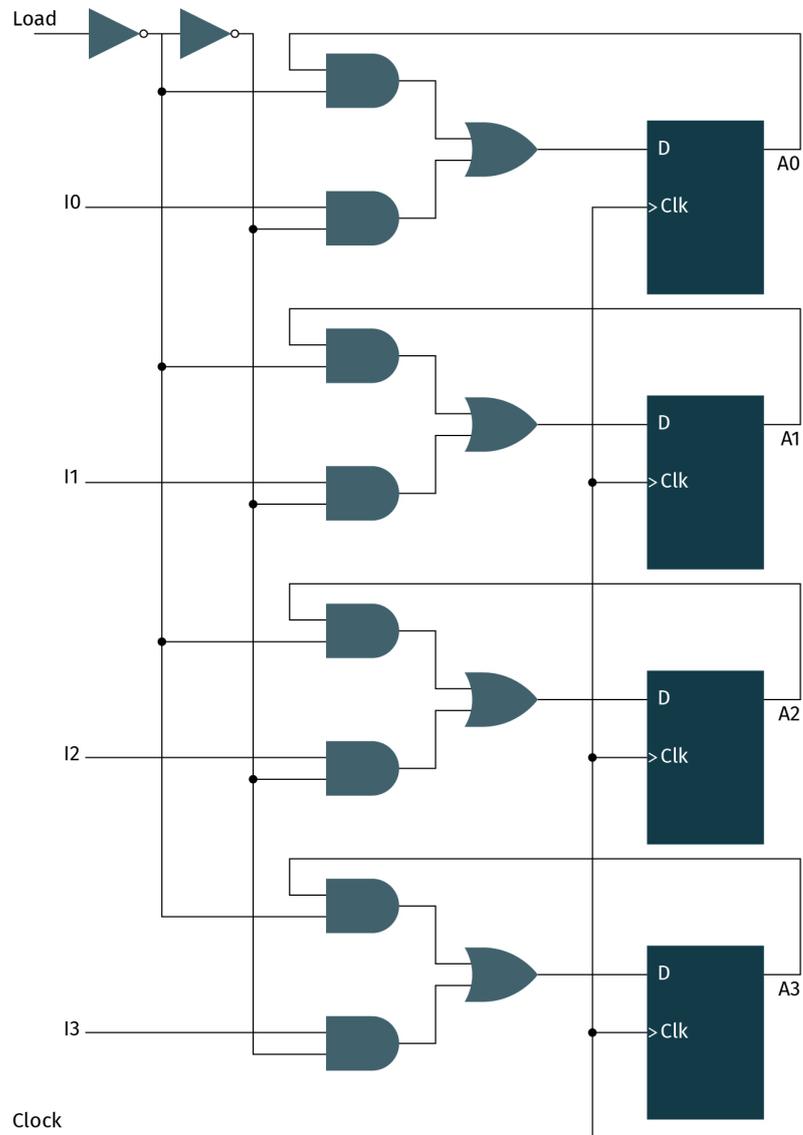
Quelle: Moustafa Nawito, 2022.

Parallele Datenladung

Der wesentliche Baustein digitaler Systeme ist ein Register mit paralleler Datenladung. In synchronen Digitalsystemen wird ein Haupttaktgenerator verwendet, der einen kontinuierlichen Strom von Taktimpulsen erzeugt. Alle Flip-Flops und Register des Systems empfangen die Impulse. Die Auswahl des Registerbetriebs für jeden Takt wird durch ein separates Steuersignal kontrolliert. Ein Register wird als geladen oder aktualisiert bezeichnet, wenn neue Daten in das Register übertragen werden. Wenn alle Bits des Registers gleichzeitig mit einem einzigen Takt geladen werden, wird dies als paralleles Laden bezeichnet. Das Register in folgender Abbildung lädt alle vier Eingänge gleichzeitig, wenn eine Taktflanke an die *Clk*-Eingänge des Registers angelegt wird. Wenn der Inhalt des Registers bei dieser Einstellung intakt bleiben soll, müssen entweder die Eingänge konstant gehalten werden oder der Takt ist in der Schaltung zu deaktivieren. Der Datenbus, der das Register im ersten Szenario mit Strom versorgt, wäre nicht für anderen Verkehr verfügbar. Im zweiten Szenario kann das Takteingangssignal durch ein Freigabegatter gesteuert werden, um zu verhindern, dass es das Register erreicht, indem z. B. eine AND-Verknüpfung des Eingangstakts mit einem anderen Steuersignal durchgeführt wird. Wenn das Steuersignal 1 ist, wird der Takt an das Register weitergeleitet, wenn das Steuersignal 0 ist, bleibt der Takt niedrig.

Das Einfügen von Gattern in den Taktpfad (In diesem Fall sagen wir, dass der Takt „gated“ ist, d. h., durch ein logisches Gatter läuft.) ist jedoch nicht empfehlenswert, da dies dazu führt, dass die Logik mit Taktimpulsen ausgeführt wird. Wenn Logikgatter hinzugefügt werden, werden die Ausbreitungsverzögerungen zwischen den Flip-Flop-Eingängen und dem Haupttakt ungleichmäßig. Es muss sichergestellt werden, dass alle Taktimpulse im gesamten System zur gleichen Zeit ankommen, damit alle Flip-Flops zur gleichen Zeit aktiviert werden, um das System vollständig zu synchronisieren. Aus der Implementierung von Logik mit Taktimpulsen resultieren unterschiedliche Verzögerungen und dies könnte dazu führen, dass das System nicht mehr synchron ist. Aus diesem Grund ist es vorzuziehen, den Betrieb des Registers mit den *D*-Eingängen zu steuern, anstatt mit den *C*-Eingängen der Flip-Flops, die den Takt steuern. Auf diese Weise wird ein durch ein Logikgatter gesteuerter Takt simuliert, ohne dass der Taktweg der Schaltung geändert wird.

Abbildung 56: 4-Bit-Register mit paralleler Datenladung



Quelle: Moustafa Nawito, 2022.

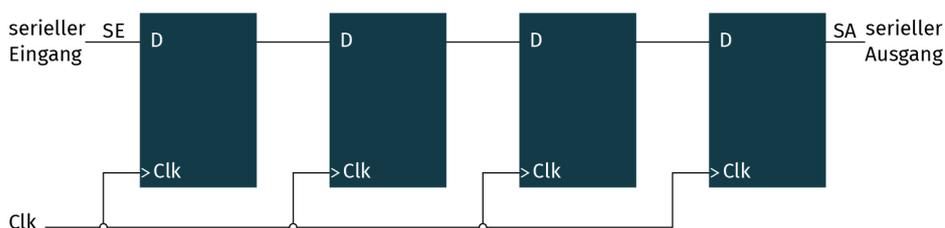
Die obere Abbildung zeigt ein 4-Bit-Register mit einem Datenladungssteuereingang, der durch Gatter in die D -Eingänge der Flip-Flops geleitet wird. Die zusätzlichen Gatter implementieren einen zweikanaligen MUX, dessen Ausgang den Eingang des Registers entweder über den Datenbus oder den Ausgang des Registers steuert. Der Datenladungseingang des Registers gibt vor, was mit jedem Taktimpuls geschehen soll. Wenn der Datenladungseingang 1 ist, bewirkt die nächste positive Flanke des Taktes, dass die Daten an den vier externen Eingängen in das Register übertragen werden. Die Ausgänge der Flip-Flops sind mit den entsprechenden Eingängen verbunden, wenn der Datenladungeingang auf 0 gesetzt ist. Einem D -Flip-Flop fehlt eine „no change“-Bedingung, sodass die Rückkopplungsver-

bindung vom Ausgang zum Eingang erforderlich ist. Der *D*-Eingang bestimmt den nachfolgenden Zustand des Registers mit jeder Taktflanke. Damit der Ausgang unverändert bleibt (d. h., der Ausgang wird bei jedem Taktimpuls an den Eingang weitergeleitet) muss der *D*-Eingang dem aktuellen Wert des Ausgangs entsprechen. Die Taktimpulse werden kontinuierlich an die *Clk*-Eingänge angelegt. Der Ladeeingang steuert, ob der folgende Impuls neue Daten übernimmt oder die bereits im Register vorhandenen Daten erhält. Als Reaktion auf eine Taktflanke werden die Daten gleichzeitig mit allen vier Bits von den Dateneingängen und -ausgängen des Registers übertragen.

Schieberegister

Ein Schieberegister ist ein Register, das die in jeder Zelle gespeicherten Binärdaten in die nächste Zelle in einer bestimmten Richtung verschieben kann. Der Ausgang eines Flip-Flops ist mit dem Eingang des nächsten Flip-Flops in einer Kette verbunden, die die logische Konfiguration des Schieberegisters bildet. Gemeinsame Taktimpulse werden an alle Flip-Flops gesendet und veranlassen sie, Daten von einer Stufe zur nächsten zu verschieben. Die folgende Abbildung zeigt das einfachste vorstellbare Schieberegister, das ausschließlich aus Flip-Flops besteht. Der *D*-Eingang des Flip-Flops rechts neben dem vorgesehenen Flip-Flop ist mit dem Ausgang dieses Flip-Flops verbunden. Hier ist ein unidirektionales (von links nach rechts) Schieberegister dargestellt. Der Inhalt des Registers wird mit jedem Taktimpuls um eine Bitposition nach rechts verschoben. Mit dieser Einstellung wird keine Linksverschiebung unterstützt. Was während der Verschiebung in das ganz linke Flip-Flop übergeht, wird durch den seriellen Eingang bestimmt. Die Verwendung des Flip-Output-Flops ganz rechts ermöglicht die Erzeugung der seriellen Ausgabe.

Abbildung 57: 4-Bit-Schieberegister



Quelle: Moustafa Nawito, 2022.

Manchmal ist es wichtig, die Verschiebung so zu steuern, dass sie nur bei bestimmten Impulsen erfolgt und bei anderen nicht. Das Taktsignal kann unterdrückt werden, indem man das Taktsignal wie beim Datenregister ausblendet, um die Verschiebung des Registers zu verhindern. In Hochgeschwindigkeitsschaltungen ist es vorzuziehen, die Taktwirkung zu unterdrücken, anstatt das Taktsignal zu „gaten“, indem der Taktpfad beibehalten wird, während der Ausgang jeder Registerzelle über einen zweikanaligen MUX geleitet wird, dessen Ausgang mit dem Eingang der Zelle gekoppelt ist.

Stehen die Flip-Flop-Ausgänge eines Schieberegisters zur Verfügung, können seriell eingegebene Daten durch Verschieben parallel aus den Flip-Flop-Ausgängen extrahiert werden. Parallel eingegebene Daten können in serieller Form durch Verschieben der im Register gespeicherten Daten extrahiert werden, wenn das Schieberegister mit einer parallelen Ladefunktion ausgestattet ist.

Einige Schieberegister verfügen über die erforderlichen Ein- und Ausgangsanschlüsse für die parallele Übertragung. Sie können auch in der Lage sein, sowohl nach rechts als auch nach links zu schieben. Das allgemeinste Schieberegister hat die folgenden Fähigkeiten:

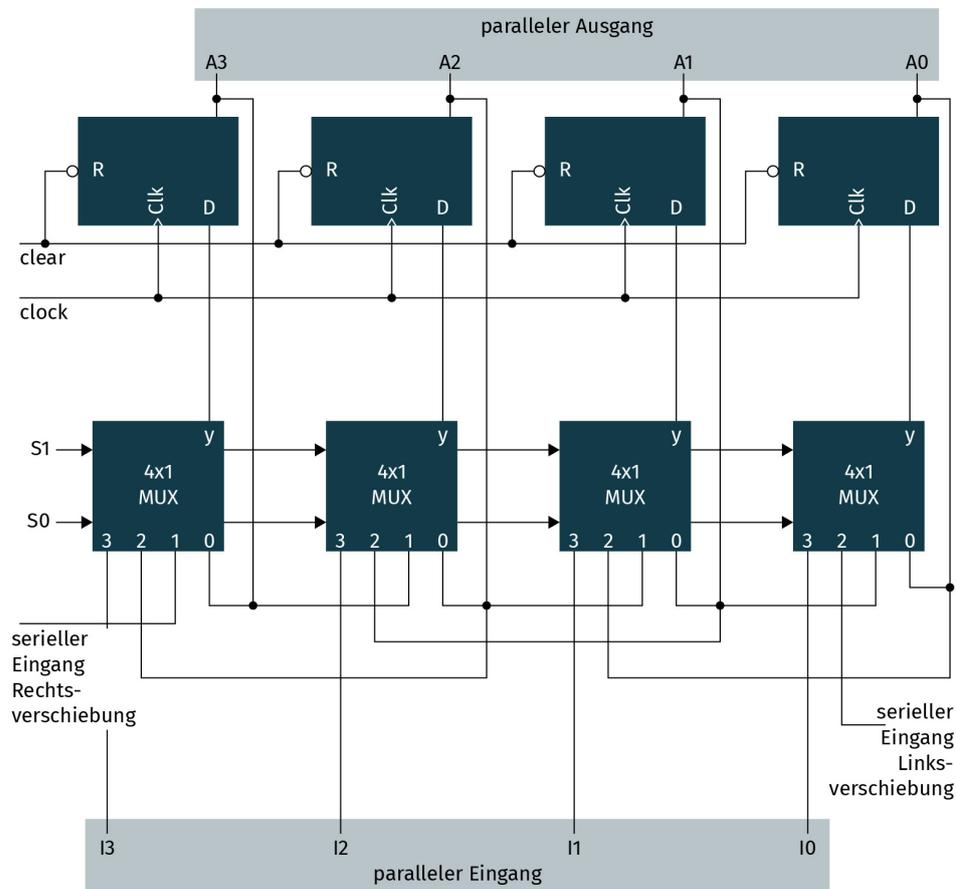
- eine Löschststeuerung (clear), um das Register auf 0 zu setzen,
- einen Takteingang (clock) zur Synchronisierung der Operationen,
- eine Rechtsschiebe-Steuerung zur Aktivierung der Rechtsschiebe-Operation und der mit der Rechtsschiebe-Operation verbundenen seriellen Eingangs- und Ausgangsleitungen,
- eine Linksverschiebungs-Steuerung zur Aktivierung der Linksverschiebungs-Operation und der seriellen Eingangs- und Ausgangsleitungen, die mit der Linksverschiebung verbunden sind,
- ein Parallel-Load-Steuerelement, um eine parallele Übertragung zu ermöglichen, und die n Eingangsleitungen, die mit der parallelen Übertragung verbunden sind,
- n parallele Ausgangsleitungen sowie
- ein Steuerzustand, der die Informationen im Register in Reaktion auf den Takt unverändert lässt.

Ein unidirektionales Schieberegister ist ein Register, das nur in eine Richtung schieben kann. Ein bidirektionales Schieberegister ist demgegenüber ein Register, das in beide Richtungen schalten kann. Ein Register wird als **universelles Schieberegister** bezeichnet, wenn es neben dem Schieben auch das parallele Laden unterstützt. Die folgende Abbildung zeigt den Schaltplan eines universellen 4-Bit-Schieberegisters mit allen oben genannten Merkmalen. Die Schaltung besteht aus vier D -Flip-Flops und vier Multiplexern. $S1$ und $S0$ sind zwei gemeinsame Auswahleingänge, die von den vier Multiplexern genutzt werden. Jeder Multiplexer wählt den Eingang 0, wenn $S1S0 = 00$, den Eingang 1, wenn $S1S0 = 01$, und die beiden anderen Eingänge, wenn $S1S0 = 02$. Entsprechend den Funktionseinträgen in folgender Tabelle steuern die Auswahleingänge die Betriebsart des Registers. Der aktuelle Wert des Registers wird an die D -Eingänge der Flip-Flops angelegt, wenn $S1S0 = 00$ ist. Dieser Umstand schafft einen Pfad von jedem Flip-Output-Flop zu seinem Eingang, sodass der Ausgang während dieser Betriebsart zum Eingang zurückkehrt. Es findet keine Zustandsänderung statt, da jedes Flip-Flop bei der nächsten Taktflanke den Binärwert erhält, den es zuvor hatte.

Universelles Schieberegister

Ein bidirektionales Schieberegister, mit der Möglichkeit, Daten parallel zu laden, nennt sich universelles Schieberegister.

Abbildung 58: 4-Bit universelles Schieberegister



Quelle: Moustafa Nawito, 2022.

Tabelle 21: Funktionstabelle des universellen Schieberegisters

S1	S0	Operation
0	0	Keine Änderung
0	1	Rechtsverschiebung
1	0	Linksverschiebung
1	1	Parallele Datenladung

Quelle: Moustafa Nawito, 2022.

Wenn $S_1S_0 = 01$ ist, hat der Anschluss 1 des Multiplexers einen Pfad zu den D -Eingängen der Flip-Flops. Aufgrund der daraus resultierenden Rechtsverschiebung erhält das Flip-Flop A_3 den seriellen Eingang. Wenn $S_1S_0 = 10$ ist, findet eine Linksverschiebung statt, und das Flip-Flop A_0 erhält den anderen seriellen Eingang. Die Binärdaten auf den paral-

lenen Eingangsleitungen werden dann bei der folgenden Taktflanke, wenn $S1S0 = 11$ ist, gleichzeitig in das Register übertragen. Dabei ist zu beachten, dass die Daten bei einer Verschiebung nach rechts als MSB und bei einer Verschiebung nach links als LSB in das Register eingehen. Ein aktives Low-Signal namens Clear b löscht alle Flip-Flops.

Digitale Systeme, die weit voneinander entfernt sind, werden häufig mit Schieberegistern verbunden. Stellen Sie sich die Situation vor, dass n Bits zwischen zwei Orten übertragen werden müssen. Es ist kostspielig, n Bits parallel über n Leitungen zu senden, wenn die Entfernung groß ist. Kostengünstiger ist es, eine einzige Leitung zu verwenden und die Daten seriell, ein Bit nach dem anderen, zu senden. Der Sender überträgt die n -Bit-Daten seriell über die gemeinsame Leitung, nachdem er sie parallel in ein Schieberegister übernommen hat. Die Daten werden vom Empfänger seriell übernommen und in ein Schieberegister eingetragen. Wenn alle n Bits empfangen wurden, können sie gleichzeitig aus den Ausgängen des Registers extrahiert werden. Folglich führt der Empfänger eine Seriell-Parallel-Wandlung der Daten durch, während der Sender eine Parallel-Seriell-Wandlung vornimmt.

6.2 Zähler

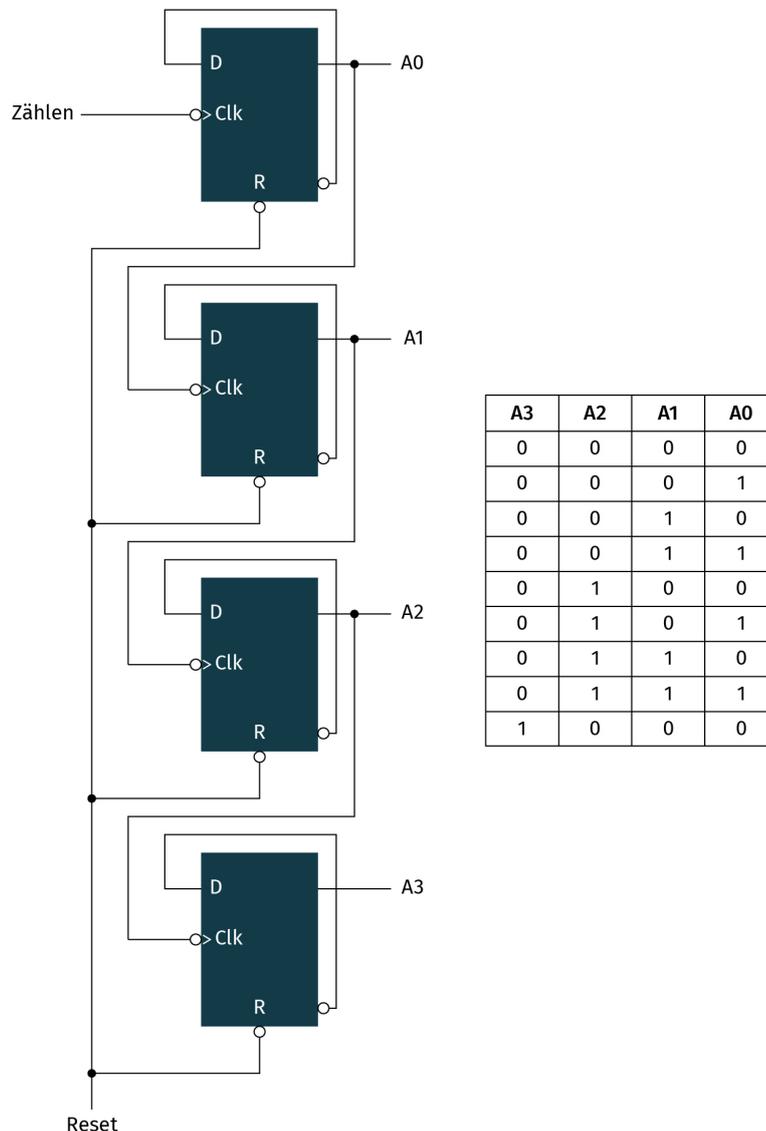
Ein Zähler ist ein Register, das beim Anlegen von Eingangsimpulsen eine bestimmte Reihe von Zuständen durchläuft. Die Eingangsimpulse können von einer externen Quelle kommen oder Taktimpulse sein, und sie können zufällig oder in vorbestimmten Abständen auftreten. Die binäre Zahlenfolge oder eine beliebige andere Folge von Zuständen kann sich an die Zustandsfolge angliedern. Ein Binärzähler ist ein Zähler, der das binäre Zahlensystem verwendet. Ein n -Bit-Binärzähler kann im Binärsystem von 0 bis 2^{n-1} zählen und besteht aus n Flip-Flops. Es gibt zwei Arten von Zählern: synchrone Zähler und asynchrone Zähler. Ein Flip-Flop-Ausgangsübergang dient als Quelle für die Aktivierung anderer Flip-Flops in einem asynchronen Zähler. Mit anderen Worten: Die *Clk*-Eingänge einiger oder aller Flip-Flops werden durch die Übergänge in den Ausgängen anderer Flip-Flops und nicht durch die gemeinsamen Taktimpulse aktiviert. Er wird als Ripple (engl. Welle) Zähler bezeichnet, was bedeutet, dass sich das Signal wie eine Welle durch die Flip-Flops ausbreitet. In einem synchronen Zähler hingegen wird ein gemeinsamer Takt von den *Clk*-Eingängen aller Flip-Flops empfangen.

Asynchrone Zähler

Eine komplementäre Folge von Flip-Flops wird zu einem binären Ripplezähler verbunden, wobei der Ausgang jedes Flip-Flops mit dem *Clk*-Eingang des nachfolgenden Flip-Flops höherer Ordnung verbunden ist. Die eingehenden Zählimpulse werden von dem Flip-Flop empfangen, das das niederwertigste Bit hält. Indem man den Komplement-Ausgang eines *D*-Flip-Flops mit dem *D*-Eingang verbindet, kann man ein Komplement-Flip-Flop erzeugen. Auf diese Weise ergänzt das Flip-Flop den aktuellen Zustand am *D*-Eingang, der immer das Komplement des aktuellen Zustands ist. Die folgende Abbildung zeigt das Logikdiagramm und die Funktionstabelle eines 4-Bit-Binär-Ripplezählers. Jedes Flip-Output-Flop ist mit dem *Clk*-Eingang des in der Kette nachfolgenden Flops verbunden. Die eingehenden Zählimpulse werden von dem Flip-Flop empfangen, das das niederwertigste

Bit hält. Die Flip-Flops reagieren auf den Übergang der negativen Flanke des Eingangs, wie durch die Blase vor dem dynamischen Anzeigezeichen neben *Clk* angezeigt wird. Wenn der Ausgang des Flip-Flops, mit dem *Clk* gekoppelt ist, von 1 auf 0 wechselt, findet der negative Übergang statt.

Abbildung 59: 4-Bit-Binär-Asynchrone Zähler



Quelle: Moustafa Nawito, 2022.

Anhand der ersten neun in der oberen Abbildung genannten Binärwerte wird die Funktionsweise des 4-Bit-Binär-Ripplezählers erläutert. Mit jedem Zählimpulseingang erhöht sich der Zählwert binär von 0 auf 1. Der Zähler wird auf 0 zurückgesetzt, um nach Erreichen von 15 den nächsten Zählvorgang zu beginnen. Jeder Zählimpulseingang vervoll-

ständig das niederwertigste Bit, A_0 . Wenn A_0 von 1 auf 0 übergeht, wird A_1 vervollständigt. Wenn A_1 von 1 auf 0 wechselt, vervollständigt es jedes Mal A_2 . A_2 komplettiert A_3 immer dann, wenn es von 1 auf 0 wechselt, und so weiter für alle anderen höherwertigen Bits in einem Rundlaufzähler. Nehmen wir den Wechsel von Zählerstand 0011 zu 0100 als Beispiel. Mit A_0 wird der Zählimpuls addiert. A_0 ergänzt A_1 und aktiviert es, da es von 1 auf 0 wechselt. Infolgedessen konvertiert A_1 von 1 auf 0 und ergänzt A_2 , wodurch es von 0 auf 1 wechselt. Da A_2 einen positiven Übergang verursacht und das Flip-Flop nur auf negative Übergänge reagiert, löst A_2 nicht A_3 aus. Indem man jedes Bit einzeln ändert, kann man die Zahl 0100 erreichen, insofern man von 0011 zu 0010, 0000 und dann 0100 zählt. Das Signal durchläuft den Zähler in einem welligen Muster von einer Stufe zur nächsten, da sich die Flip-Flops der Reihe nach ändern.

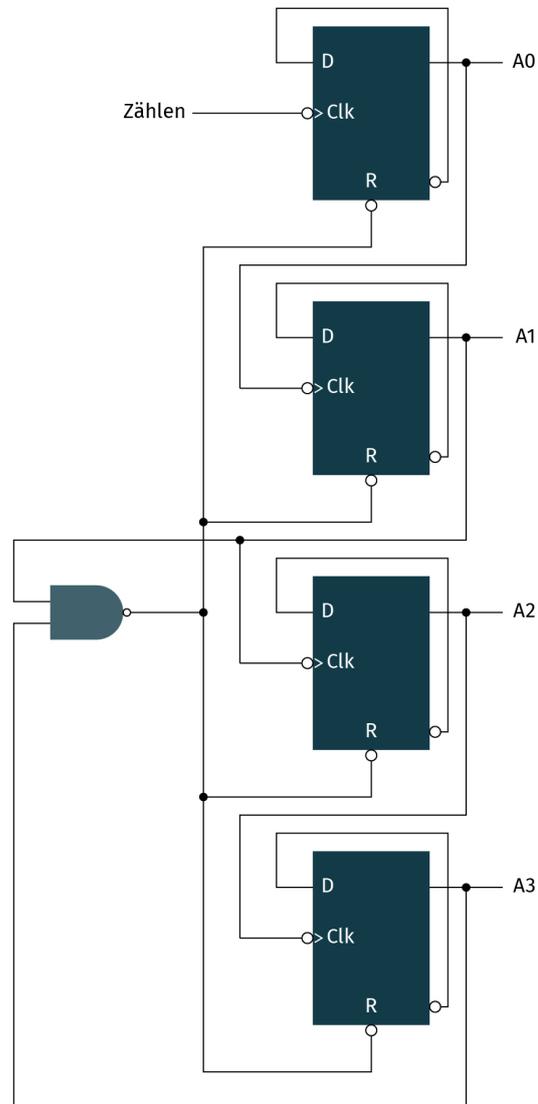
Binäre Countdownzähler sind binäre Zähler, die rückwärts zählen (sog. Rückwärtszähler). Bei jedem Eingangszählerimpuls in einem Countdownzähler wird der Binärwert um eins verringert. Ein 4-Bit-Countdownzähler beginnt bei Binärwert 15 und durchläuft die Binärwerte 14, 13, 12, ..., 0, bevor er zu Binärwert 15 zurückkehrt. Das niederwertigste Bit wird bei jedem Zählimpuls komplementiert, wie aus einer Liste der Zählfolge eines binären Countdownzählers ersichtlich ist. Wenn das Bit vor ihm in der Sequenz von 0 auf 1 wechselt, ergänzt es alle anderen Bits in der Sequenz.

BCD-Ripplezähler

Ein Dezimalzähler durchläuft eine Reihe von zehn Zuständen und setzt sich nach dem Zählen bis 9 auf 0 zurück. Da eine Dezimalziffer einen Binärcode mit mindestens vier Bits erfordert, muss ein solcher Zähler mindestens vier Flip-Flops zur Darstellung jeder Ziffer haben. Die binäre Codierung, die zur Darstellung einer Dezimalziffer verwendet wird, bestimmt die Reihenfolge der Zustände in einem Dezimalzähler. Ein Binärzähler und ein Dezimalzähler sind ähnlich, aber der Zustand nach 1001 (der Code für die Dezimalziffer 9) in einem Binärzähler ist 0000 (der Code für die Dezimalziffer 0).

Das Logikdiagramm eines 4-Bit-BCD-Ripplezählers ist in folgender Abbildung dargestellt. Wie man sieht, ist es dem eines binären Rundsteuerzählers sehr ähnlich, mit der Ausnahme, dass eine logische Bedingung mithilfe eines zusätzlichen Gatters geschaffen wird, um den Zähler wieder auf 0000 zurückzusetzen, wenn der Zähler kurz davor ist, den unerwünschten Zustand 1010, d. h. dezimal 10, zu erreichen. Konkret werden die beiden Ausgangsbits A_1 und A_3 , die im Falle von 1010 beide den Wert einer logischen 1 haben, in ein NAND-Gatter eingegeben, dessen Ausgang mit dem gemeinsamen Reset aller vier Flip-Flops verbunden ist. Wenn A_1 und A_3 den Wert 1 haben, erzeugt das NAND-Gatter eine logische 0, wodurch der Reset aktiviert wird und der Zähler bei 0000 wieder von vorne beginnt.

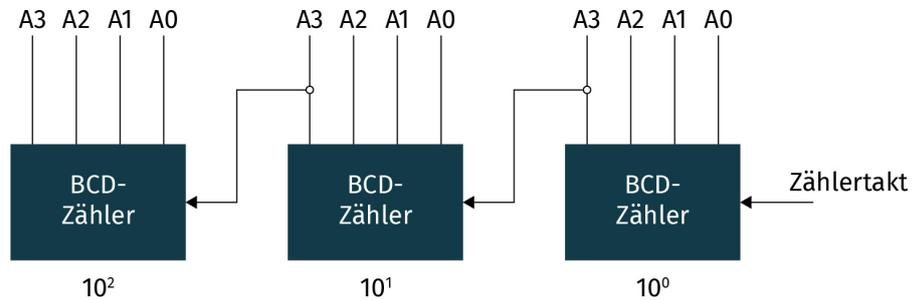
Abbildung 60: BCD-Ripplezähler



Quelle: Moustafa Nawito, 2022.

Da er von 0 bis 9 zählt, ist der BCD-Zähler in obiger Abbildung ein Dekadenzähler. Wir brauchen einen 2-Dekadenzähler, damit wir dezimal von null bis 99 zählen können, und einen 3-Dekadenzähler, damit wir von null bis 999 zählen können. Indem man BCD-Zähler kaskadenförmig miteinander verbindet, einen für jede Dekade, können viele Dekadenzähler erstellt werden. In folgender Abbildung ist ein 3-Dekadenzähler dargestellt. Die Eingänge der zweiten und dritten Dekade stammen aus dem achten Quartal der vorherigen Dekade. Die nächsthöhere Dekade beginnt zu zählen, wenn Q8 in einer Dekade von 1 auf 0 wechselt, während ihre eigene Dekade von 9 auf 0 wechselt.

Abbildung 61: 3-Dekadenzähler

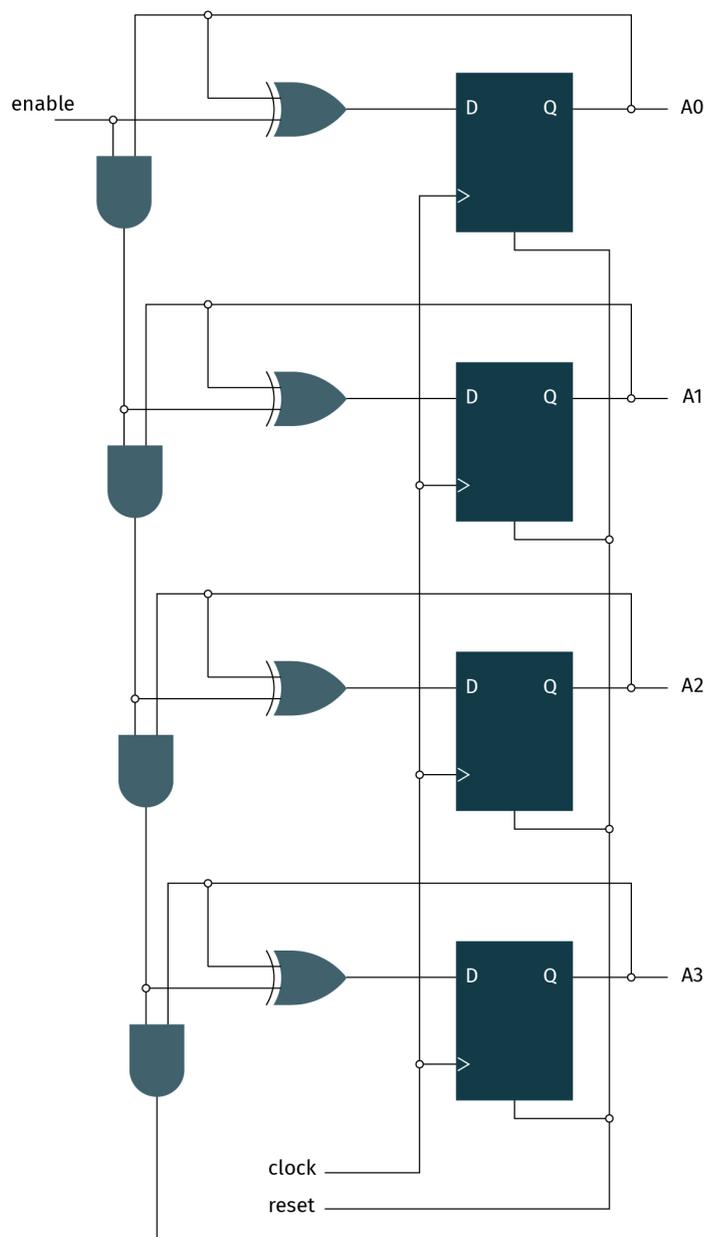


Quelle: Moustafa Nawito, 2022.

Synchronzähler

Im Gegensatz zu den Asynchronzählern werden bei den Synchronzählern alle Flip-Flop-Eingänge mit Taktimpulsen versorgt. Anstatt wie bei einem Rundsteuerzähler ein Flip-Flop nach dem anderen zu aktivieren, werden alle Flip-Flops gleichzeitig durch einen gemeinsamen Takt aktiviert. Jeder Impuls in einem synchronen Binärzähler ergänzt das Flip-Flop an der niederwertigsten Stelle. Wenn alle Bits in den niederwertigen Stellen gleich 1 sind, wird ein Flip-Flop in einer beliebigen anderen Position abgeschlossen. Insofern beispielsweise der aktuelle Zustand eines 4-Bit-Zählers $A_3A_2A_1A_0 = 0011$ ist, lautet der nachfolgende Zählerstand 0100. A_0 wird durchgängig mit Komplementen versehen. Da sich A_0 derzeit im Zustand der Komplementierung von 1 befindet, ist $A_1 = 1$. Aufgrund des aktuellen Zustands von $A_1A_0 = 11$ ist A_2 komplementiert. A_3 wird jedoch nicht komplementiert, da der aktuelle Zustand von $A_2A_1A_0 = 011$ nicht zu einer All-Eins-Bedingung führt. Synchroner Binärzähler können mit komplementären Flip-Flops und Gattern aufgebaut werden und haben ein vorhersehbares Muster. Der in folgender Abbildung dargestellte 4-Bit-Zähler zeigt das regelmäßige Muster. Alle Flip-Flops haben einen gemeinsamen Takt an ihren C -Eingängen. Die Zählerfreigabe schaltet den Zähler ein.

Abbildung 62: 4-Bit-Synchronzähler

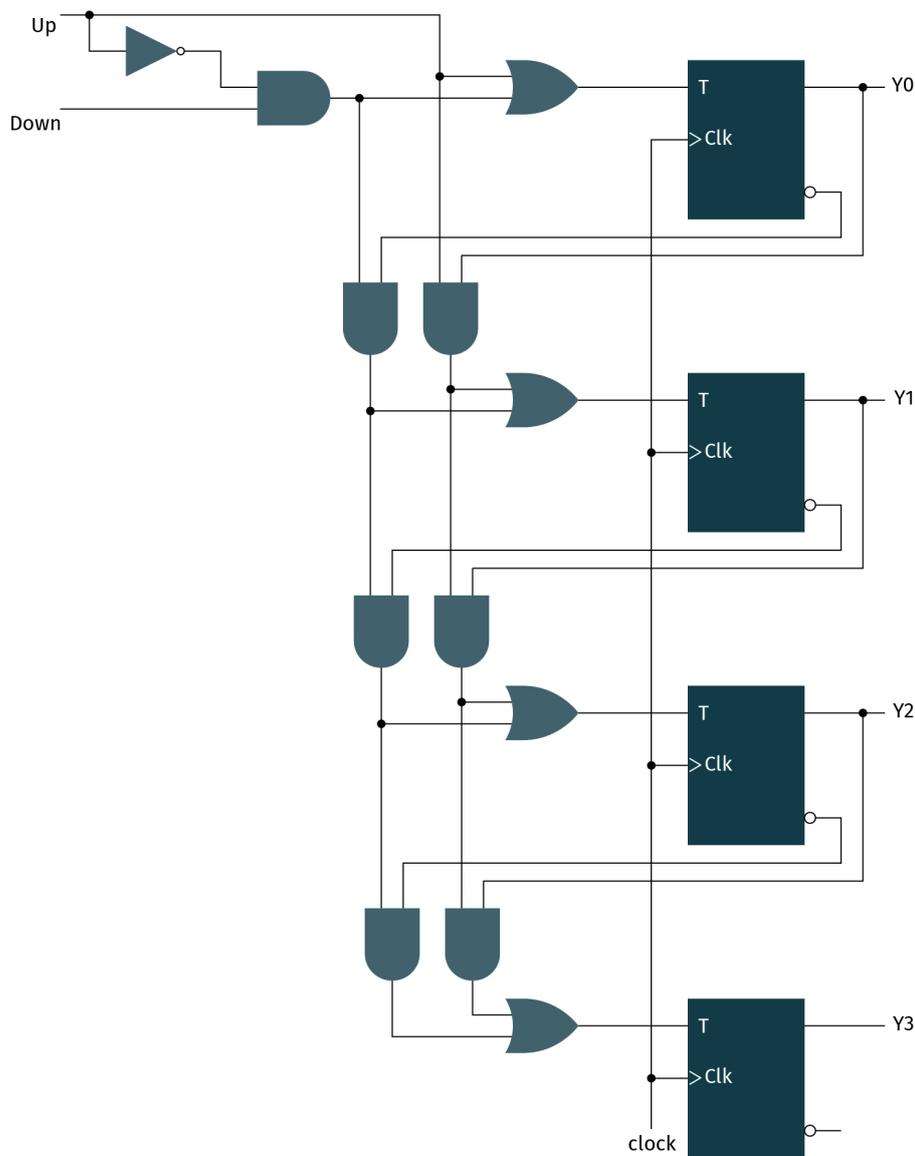


Quelle: Moustafa Nawito, 2022.

Ein binärer Countdownzähler kann, wie in obiger Abbildung dargestellt, aufgebaut werden, mit dem Unterschied, dass die Eingänge zu den AND-Gattern von den komplementären Ausgängen statt von den normalen Ausgängen der vorherigen Flip-Flops stammen müssen. Die beiden Operationen können in einer Schaltung kombiniert werden, um einen Zähler zu bilden, der entweder aufwärts oder abwärts zählt. Die Schaltung eines Vorwärts-Abwärts-Binärzählers mit *T*-Flip-Flops ist in folgender Abbildung dargestellt. Er hat einen

Aufwärts- und einen Abwärts-Steuereingang. Wenn der Aufwärtssteuereingang 1 ist, zählt die Schaltung aufwärts, da die T -Eingänge ihre Signale von den Werten der vorherigen normalen Ausgänge der Flip-Flops erhalten. Insofern der Abwärtseingang 1 und der Aufwärtseingang 0 ist, zählt die Schaltung abwärts, da die komplementären Ausgänge der vorherigen Flip-Flops an die D -Eingänge angelegt werden. Wenn die Eingänge Up und Down beide 0 sind, ändert der Schaltkreis seinen Zustand nicht und bleibt in der gleichen Zählung. Insofern die Eingänge Up und Down beide 1 sind, zählt die Schaltung aufwärts. Durch diese Bedingungen wird sichergestellt, dass zu jedem Zeitpunkt nur eine Operation ausgeführt wird.

Abbildung 63: Synchroner Up-Down-Zähler



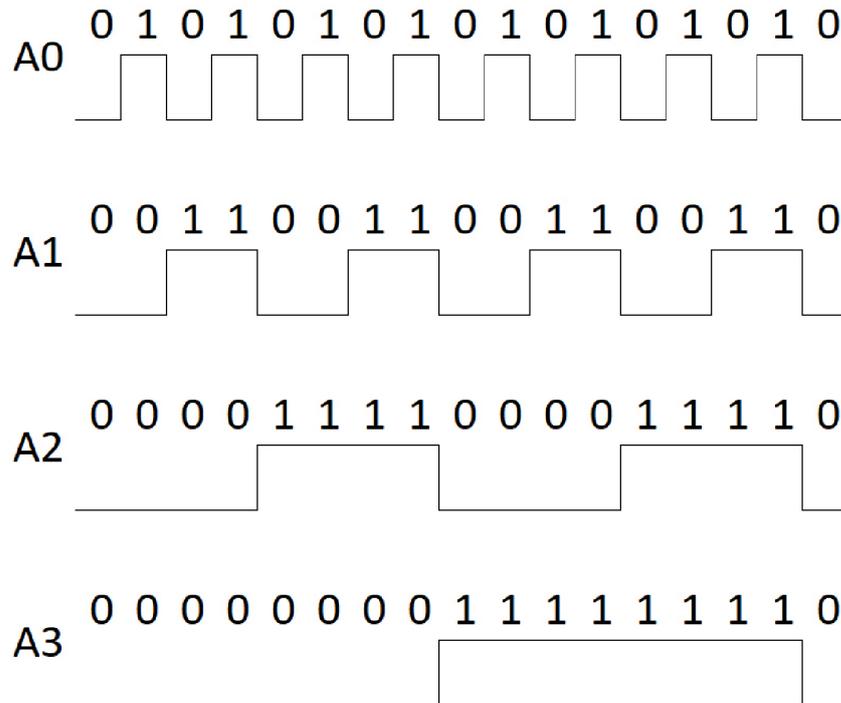
Quelle: Moustafa Nawito, 2022, in Anlehnung an Mano, 2013, S. 274.

Zähler als Frequenzteiler

Eine genauere Betrachtung des Ausgangs eines Binärzählers (Abbildung „4-Bit-Binär-Asynchronzähler“) zeigt eine interessante Eigenschaft: Die LSD A_0 wechselt doppelt so schnell von 0 auf 1 wie A_1 , und A_1 wechselt doppelt so schnell von 0 auf 1 wie A_2 , und dasselbe gilt für A_2 und A_3 . Dies wird in folgender Abbildung veranschaulicht. Man sieht, dass Zähler als Frequenzteiler funktionieren können, wobei jeder Ausgang die Frequenz des vorherigen Ausgangs durch zwei teilt. Dies gilt sowohl für synchrone als auch für asynchrone Zähler. In der Praxis werden jedoch synchrone Zähler als Frequenzteiler verwendet, um

eine höhere Genauigkeit zu erreichen. Es ist wichtig zu beachten, dass die Taktfrequenz, die in der Abbildung nicht dargestellt ist, doppelt so hoch ist wie die des Ausgangssignals an A0.

Abbildung 64: Frequenzteiler

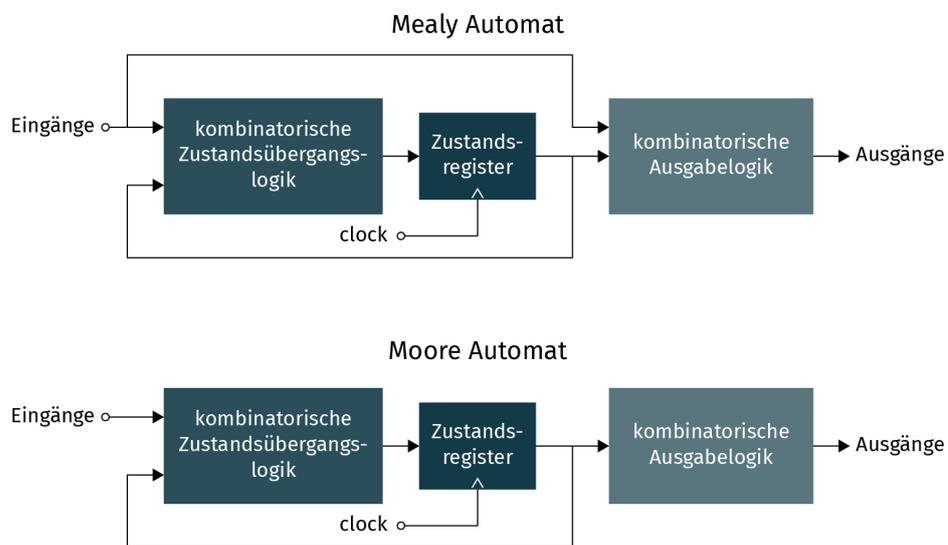


Quelle: Moustafa Nawito, 2022.

6.3 Automaten

Das allgemeinste Modell einer sequenziellen Schaltung umfasst Eingänge, Ausgänge sowie interne Zustände und wird als Automat (auch Zustandsautomat) bezeichnet. Das Mealy-Modell und das Moore-Modell sind die beiden Modelle für Automaten, die normalerweise differenziert werden. Beide sind in folgender Abbildung dargestellt. Sie unterscheiden sich lediglich in der Art und Weise, wie das Ergebnis erzeugt wird. Die Ausgabe des Mealy-Modells hängt sowohl von der Eingabe als auch vom aktuellen Zustand ab. Die Ausgabe nach dem Moore-Modell wird nur vom aktuellen Zustand beeinflusst. Beide Arten von Ausgaben sind in einem Schaltkreis möglich. Endliche Zustandsautomaten, oft als FSMs (engl. Finite State Machine) bezeichnet, sind die beiden am häufigsten verwendeten Modelle einer sequenziellen Schaltung. Ein Mealy-FSM oder eine Mealy-Maschine ist die Bezeichnung für das Mealy-Automat einer sequenziellen Schaltung. Ein Moore-FSM oder eine Moore-Maschine ist eine andere Bezeichnung für das Moore-Automat.

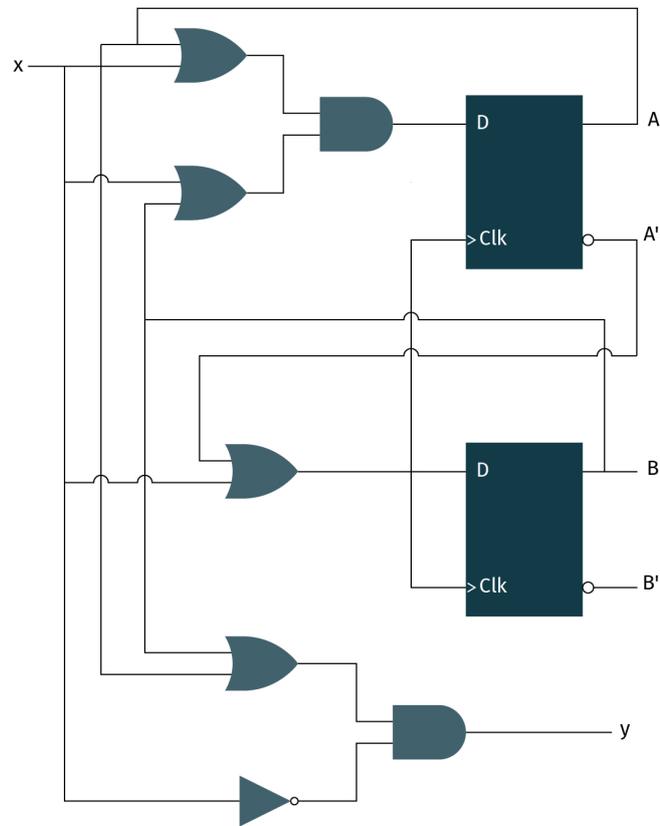
Abbildung 65: Mealy- und Moore-Automat



Quelle: Moustafa Nawito, 2022.

Die in folgender Abbildung gezeigte Schaltung ist ein Beispiel für eine Mealy-Maschine. Sowohl der aktuelle Zustand von A und B als auch der Eingang x sind Funktionen des Ausgangs y .

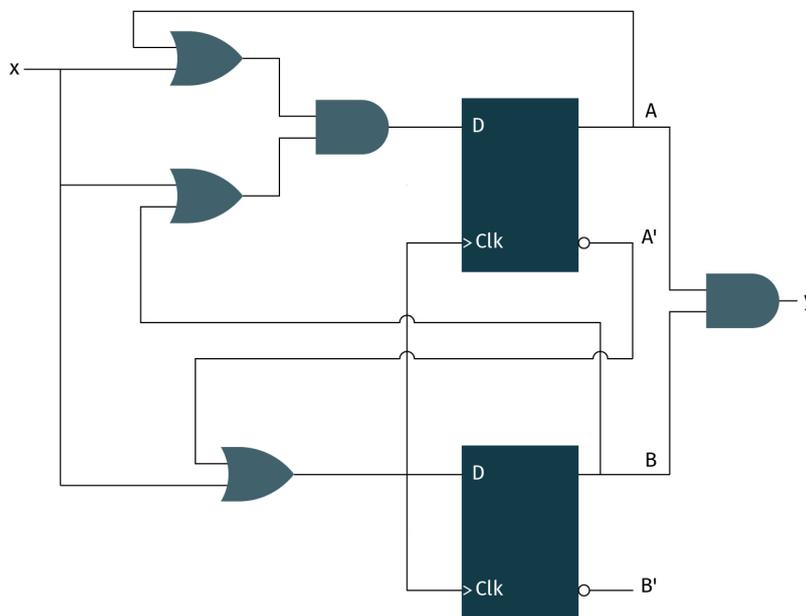
Abbildung 66: Beispiel für einen Mealy-Automat



Quelle: Moustafa Nawito, 2022.

In folgender Abbildung ist ein Moore-Modell dargestellt. In diesem Fall ist die Ausgabe lediglich eine Funktion des aktuellen Zustands. Nur die entlang der gerichteten Linien hervorgehobenen Eingänge sind in dem entsprechenden Zustandsdiagramm dargestellt. Die in den Kreisen angegebenen Flip-Flop-Zustände sind die Ausgänge. Da sie ausschließlich von Flip-Flop-Ausgängen abhängen, die ebenfalls mit dem Takt synchronisiert werden, sind die Ausgänge sequenzieller Schaltungen in einem Moore-Modell mit dem Takt synchronisiert.

Abbildung 67: Beispiel für einen Moore-Automat



Quelle: Moustafa Nawito, 2022.

Wenn sich die Eingänge eines Mealy-Modells ändern, während der Taktzyklus läuft, können auch die Ausgänge wechseln. Die Verzögerung zwischen dem Zeitpunkt, zu dem sich die Eingänge ändern, und dem Zeitpunkt, zu dem sich die Flip-Flop-Ausgänge ändern, kann auch dazu führen, dass die Ausgänge vorübergehend falsche Werte aufweisen. Bei einer Schaltung vom Typ Mealy müssen die Eingänge mit dem Takt synchronisiert und die Ausgänge kurz vor der Taktflanke abgetastet werden, um synchronisiert zu sein. Um sicherzustellen, dass die Eingänge der Flip-Flops vor der aktiven Flanke des Taktes zur Ruhe kommen, werden die Eingänge bei der inaktiven Flanke des Taktes verändert. Die Mealy-Maschine gibt den Wert aus, der kurz vor der aktiven Flanke des Taktes vorhanden war.



ZUSAMMENFASSUNG

Register sind die grundlegendste Art praktischer Speicherelemente. Sie bestehen aus einer Reihe von Flip-Flops, die jeweils ein einzelnes Bit speichern. Ein wichtiges Merkmal von Registern ist die Fähigkeit, Daten parallel zu speichern, was in aktuellen Schaltungen Verarbeitungszeit spart. Eine besondere Art von Registern ist das Schieberegister, das in der seriellen Kommunikation und Datenverarbeitung Anwendung findet.

Zähler können als Register betrachtet werden, die einen vorbestimmten Ausgang aufweisen. Synchrone Zähler haben einen Haupttakt, der mit allen Flip-Flops verbunden ist, und zählen nur bei der Flanke des Taktes. Welligkeitszähler sind asynchrone Zähler, die so aufgebaut sind, dass jedes Flip-Flop dem nachfolgenden das Signal gibt, mit der Aktualisierung zu beginnen und somit den Zählerstand zu erhöhen. Eine sehr wichtige Anwendung für Zähler ist die Verwendung als Frequenzteiler.

Alle sequenziellen Schaltungen können als Zustandsautomaten modelliert werden. Die gebräuchlichsten Typen sind die Mealy-Maschinen, deren Ausgang von den Eingängen und den Zuständen abhängt, und die Moore-Maschine, deren Ausgang nur von den Zuständen der sequenziellen Schaltung abhängt.

LEKTION 7

EINFÜHRUNG IN PROGRAMMIERBARE LOGIK

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die Merkmale von programmierbaren Logikbausteinen zu erkennen.
- die Struktur und Funktionsweise verschiedener programmierbarer Logikbausteine zu benennen.
- die Realisierung Boolescher Funktionen mittels programmierbarer Logikbausteine durchzuführen.
- die Struktur von FPGAs aufzuzeigen.
- die einführenden Grundlagen von Hardwarebeschreibungsfragen zu benennen.

7. EINFÜHRUNG IN PROGRAMMIERBARE LOGIK

Einführung

Die Realisierung praktischer digitaler Systeme erfordert die Möglichkeit, die logische Funktion auf einer bereits vorhandenen Hardware-Plattform zu implementieren. In dieser Lektion werden wir die gängigste Hardware und Methoden zur Implementierung programmierbarer Logik untersuchen. Darüber hinaus wird eine kurze Einführung in Hardwarebeschreibungssprachen gegeben.

7.1 Programmierbare Logikbausteine

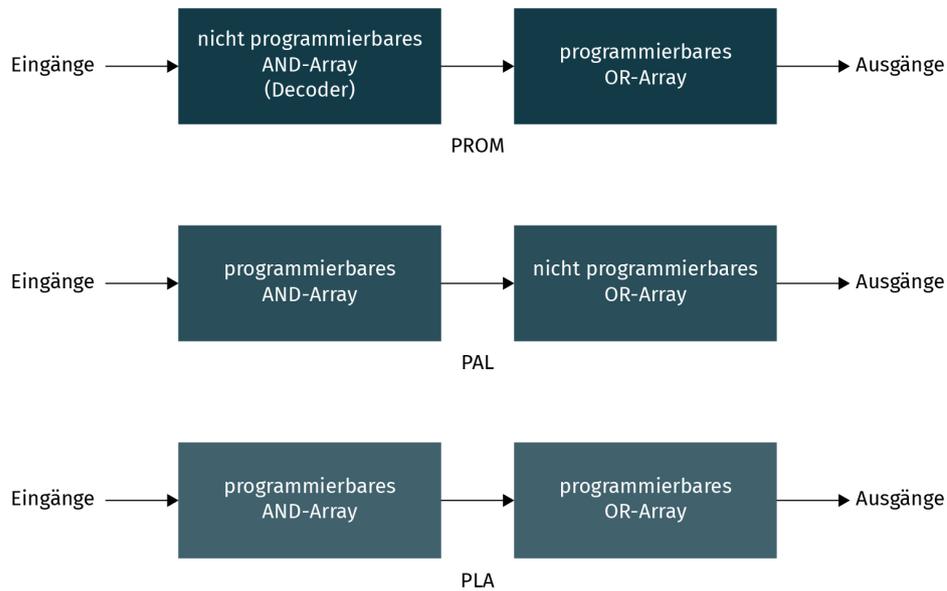
Ein integrierter Schaltkreis mit programmierbaren Gattern, die in ein AND-Array (AND-Anordnung oder AND-Feld) und ein OR-Array unterteilt sind, um eine AND-OR-Produktsumme Implementierung zu ermöglichen, wird als programmierbarer Logikbaustein (engl. Programmable Logic Device, kurz **PLD**) bezeichnet. Kombinatorische PLDs können in drei Hauptkategorien eingeteilt werden, je nachdem, wo die programmierbaren Verbindungen im AND-OR-Array angeordnet sind. Die Konfiguration der drei PLDs ist in folgender Abbildung dargestellt:

PLD

Eine programmierbare integrierte Schaltung, die es ermöglicht, logische Funktionen mittels AND- und OR-Gatterarrays als Produktsumme zu realisieren, nennt man PLD.

- Ein nicht programmierbares (festes) AND-Array, das als Decoder aufgebaut ist, und ein programmierbares OR-Array sind beide im programmierbaren Festspeicher (engl. Programmable Read-Only Memory – PROM) vorhanden. Die Booleschen Funktionen werden von den programmierbaren OR-Gattern in Form einer Summe von Mintermen implementiert.
- Die programmierbare Zellenlogik (engl. Programmable Array Logic – PAL) verfügt über ein festes OR-Array und ein konfigurierbares AND-Array. Die Booleschen Funktionen werden in jedem OR-Gatter logisch addiert, und die AND-Gatter sind so programmiert, dass sie die Produktterme liefern.
- Die programmierbare logische Anordnung (engl. Programmable Logic Array – PLA), die die Programmierung sowohl von AND- als auch von OR-Arrays ermöglicht, ist der anpassungsfähigste PLD. Um die erforderliche Produktsummenimplementierung zu ermöglichen, kann jedes OR-Gatter die Produktterme im AND-Array teilen.

Abbildung 68: PROM, PAL und PLA



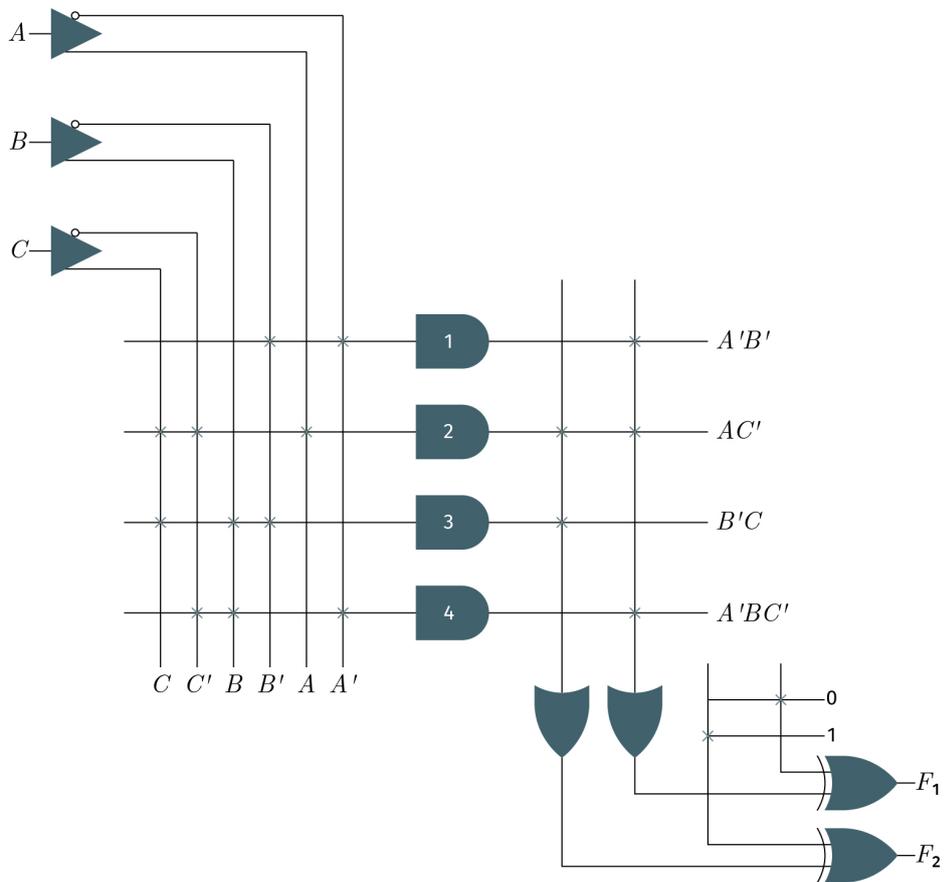
Quelle: Moustafa Nawito, 2022.

In diesem Skript werden wir uns nur auf PLA- und PAL-Geräte konzentrieren.

PLA

Obwohl der PLA konzeptionell mit dem PROM vergleichbar ist, erzeugt er nicht alle Minsterme und bietet keine vollständige Decodierung der Variablen. Ein Array von AND-Gattern, die so programmiert werden können, dass sie jeden Produktterm der Eingangsvariablen erzeugen, tritt an die Stelle des Decoders. Die Summe der Produkte wird dann für die erforderlichen Booleschen Funktionen bereitgestellt, indem die Produktterme mit OR-Gattern verbunden werden. Die folgende Abbildung zeigt die interne Logik einer PLA mit drei Eingängen und zwei Ausgängen. Obwohl diese Schaltung zu klein ist, um in einer kommerziellen Umgebung effektiv zu sein, wird hier veranschaulicht, wie die konventionelle Logik einer PLA aufgebaut ist.

Abbildung 69: Interne Struktur eines PLAs



Quelle: Moustafa Nawito, 2022.

In obiger Abbildung gibt es 34 Verbindungen, die programmierbar sind. Eine programmierbare Verbindung zwischen zwei Leitungen entspricht logisch einem Schalter, der entweder geschlossen (d. h., die beiden Leitungen sind verbunden) oder offen (d. h., die beiden Leitungen sind nicht verbunden) sein kann. Der programmierbare Schnittpunkt zwischen zwei Leitungen wird manchmal als Koppelpunkt bezeichnet. Zur Realisierung von Koppelschaltern werden verschiedene physikalische Vorrichtungen verwendet. Eine der einfachsten Technologien nutzt eine Sicherung (engl. fuse), die normalerweise die beiden Punkte verbindet, aber, durch das Anlegen eines Hochspannungsimpulses an die Sicherung, geöffnet oder „durchgebrannt“ wird. Ein x bezeichnet eine intakte Sicherung (Verbindung).

Wie zu sehen ist, wird der Eingang über eine Puffer-Wandler-Kombination geleitet, die sowohl den wahren als auch den komplementären Ausgang hat. Die Schnittpunkte der vertikalen und horizontalen Linien zeigen, dass jeder Eingang und sein Komplement mit den Eingängen der einzelnen AND-Gatter verbunden sind. Die Eingänge jedes OR-Gatters sind mit den Ausgängen der AND-Gatter verknüpft. Das XOR-Gatter erhält den Ausgang des OR-Gatters und verknüpft ihn mit einem Signal, das entweder als logische 1 oder als logi-

sche 0 ausgelegt werden kann. Wenn der XOR-Eingang mit 1 verbunden ist, wird der Ausgang invertiert. Der Ausgang bleibt unverändert, wenn der XOR-Eingang mit 0 verbunden ist. Die spezifischen Booleschen Operationen, die von der PLA in obiger Abbildung verwendet werden, sind

$$F1 = A'B' + AC' + A'BC'$$

$$F2 = (AC' + B'C)'$$

Der Ausgang jedes AND-Gatters ist im Diagramm zusammen mit den Produkttermen aufgeführt, die es erzeugt. Die Eingänge mit den verbundenen und mit x gekennzeichneten Koppelpunkten dienen als Grundlage für die Berechnung des Produktterms. Die logische Summe der gewählten Produktterme wird über ein OR-Gatter ausgegeben. Je nach realisierter Logik kann der Ausgang erweitert oder belassen werden. Die sogenannte Fuse-Map (Programmirtabelle) einer PLA kann in tabellarischer Form angegeben werden, wie in folgender Tabelle dargestellt ist. Die PLA-Programmirtabelle ist in drei Teile unterteilt:

- Im ersten Teil sind die Produktterme numerisch aufgelistet.
- Die notwendigen Verbindungen zwischen Eingängen und AND-Gattern werden im zweiten Abschnitt beschrieben.
- Im dritten Teil werden die Wege zwischen den AND- und OR-Gattern aufgezeigt.

Um das XOR-Gatter zu programmieren, können wir für jede Ausgangsvariable ein W (wahr) oder K (für komplementär) haben. Die Produktterme auf der linken Seite dienen nur zu Referenzzwecken und sind nicht Teil der Tabelle. Die Eingänge werden durch die Symbole 1, 0 oder – für jeden Produktterm gekennzeichnet. Eine 1 wird neben die Eingangsvariable gesetzt, die einer Variablen im Produktbegriff entspricht, die in der Form erscheint, die angibt, dass sie wahr ist. Die übereinstimmende Eingangsvariable wird mit einer 0 gekennzeichnet, wenn sie komplementär zu sein scheint. Insofern die Variable im Produktausdruck nicht vorhanden ist, wird sie mit einem Bindestrich gekennzeichnet. Die Verbindungen zwischen den Eingängen und den AND-Gattern werden in der Spalte „Eingänge“ der Programmirtabelle beschrieben. Der Wert 1 in der Spalte „Eingang“ zeigt eine Verknüpfung der Eingangsvariablen zum AND-Gatter an. Eine Verknüpfung vom Komplement der Variablen zum Eingang des AND-Gatters wird durch eine 0 in der Eingangsspalte angezeigt. Ein Strich verdeutlicht, dass sowohl die Eingangsvariable als auch ihr Komplement durchgebrannt sind. Eine offene Klemme am Eingang eines AND-Gatters funktioniert ähnlich wie eine 1. Die Spaltenüberschrift „Ausgänge“ enthält Informationen über die Verbindungswege zwischen den AND- und OR-Gattern. Bei den Produktbegriffen, die in der Funktion enthalten sind, sind die Ausgangsvariablen mit 1 gekennzeichnet. Jeder Produktbegriff mit einer 1 in der Ausgangsspalte erfordert eine Verbindung vom Ausgang des AND-Gatters zum Eingang des OR-Gatters. Bei den mit einem Strich versehenen Variablen wird eine durchgebrannte Sicherung angezeigt. Ein offener Anschluss am Eingang eines OR-Gatters verhält sich ähnlich wie eine 0. Der andere Eingang des entsprechenden XOR-Gatters muss mit 0 verbunden werden, um ein W auszugeben, und mit 1, um ein C auszugeben.

Tabelle 22: Programmier­tabelle des PLAs

	Produkt- term	Eingänge			Ausgänge	
		A	B	C	(W)	(K)
					F1	F2
A'B'	1	0	0	-	1	-
AC'	2	1	-	0	1	1
B'C	3	-	0	1	-	1
A'BC'	4	0	1	0	1	-

Quelle: Moustafa Nawito, 2022.

Die Anzahl der Eingänge, die Anzahl der Produktterme und die Anzahl der Ausgänge bestimmen die Größe einer PLA. 48 Produktterme, acht Ausgänge und 16 Eingänge sind für PLAs in integrierten Schaltungen üblich. Die Kernlogik der PLA besteht aus n Puffer-Inverter-Gattern, k AND-Gattern, m OR-Gattern und m XOR-Gattern für n Eingänge, k Produktterme und m Ausgänge. Die Eingänge des AND-Arrays sind über $2n \times k$ Verbindungen, die OR- und AND-Arrays über $k \times m$ Verbindungen und die XOR-Gatter über m Verbindungen mit ihm verbunden.

Beim Aufbau eines digitalen Systems mit einer PLA ist es nicht erforderlich, die internen Verbindungen der Einheit darzustellen. Es genügt über eine PLA-Programmier­tabelle, mit der die PLA angewiesen werden kann, die erforderliche Logik zu liefern. Die PLA kann feld- oder maskenprogrammierbar sein, genau wie ein PROM. Kunden schicken dem Hersteller eine PLA-Programmtabelle für die Maskenprogrammierung. Der Hersteller verwendet diese Tabelle, um im Auftrag des Kunden eine personalisierte PLA mit der vorgesehenen internen Logik zu erstellen. Das feldprogrammierbare Logik-Array (FPLA) ist eine zweite Art von PLA, die für die Benutzer zugänglich ist und von ihm mit einem handelsüblichen Hardware-Programmiergerät programmiert werden kann.

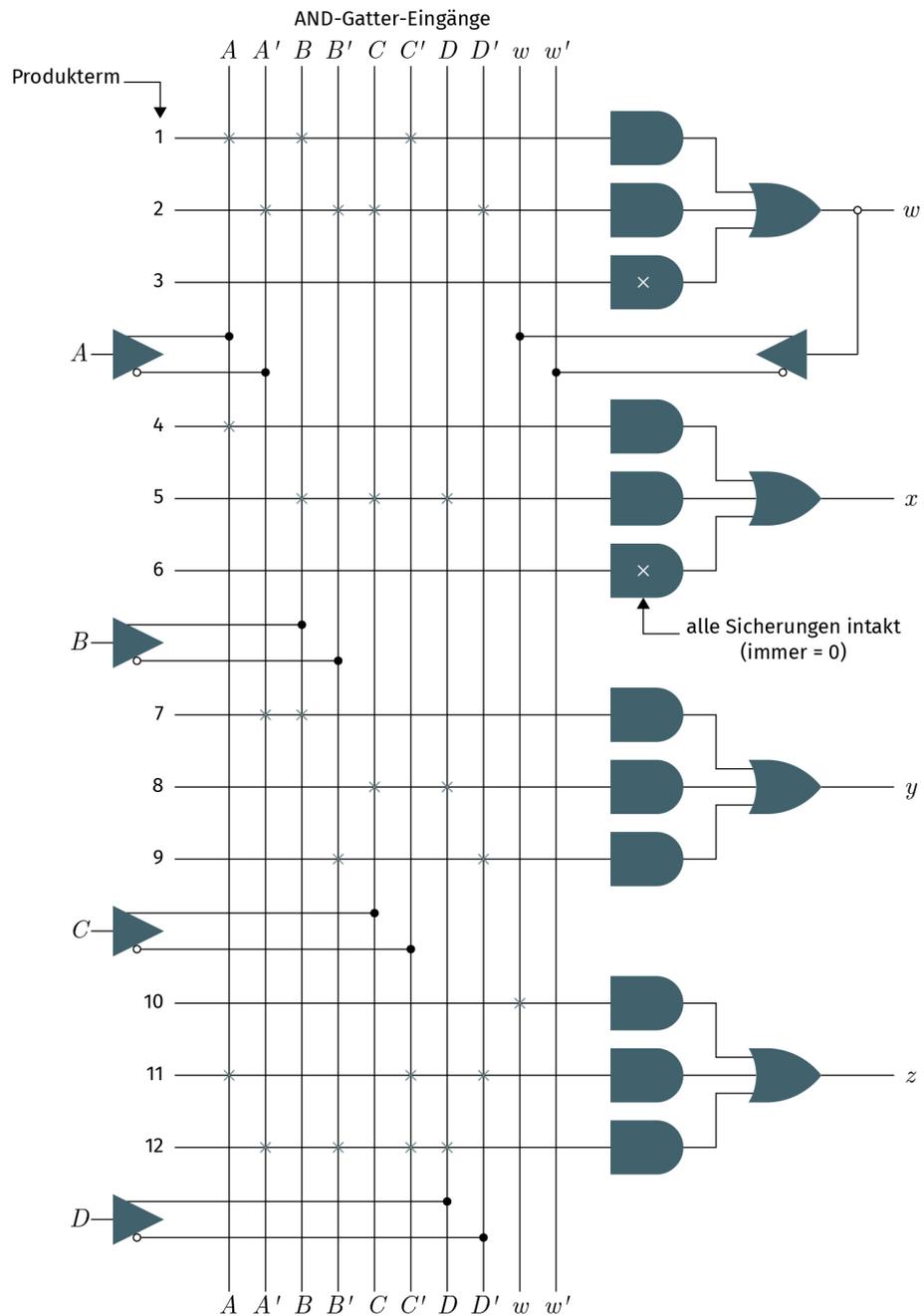
Da ein PLA nur eine begrenzte Anzahl von AND-Gattern hat, muss viel Forschungsarbeit geleistet werden, um die Anzahl der unterschiedlichen Produktterme beim Aufbau einer kombinatorischen Schaltung mit einem PLA zu reduzieren. Dies kann erreicht werden, indem jede Boolesche Funktion auf die kleinstmöglichen Terme reduziert wird. Unter der Voraussetzung, dass alle Eingangsvariablen verfügbar sind, ist es unerheblich, wie viele Literale in einer Phrase enthalten sind. Der wahre Wert und das Komplement jeder Funktion sollten kondensiert werden, um festzustellen, welche Funktion einfacher geschrieben werden kann und welche Funktion Produktwörter bietet, die von anderen Funktionen geteilt werden.

PAL

Der PAL ist einfacher zu programmieren als der PLA, aber weniger flexibel, da nur die AND-Gatter konfigurierbar sind. Der logische Aufbau eines typischen PAL mit vier Eingängen und vier Ausgängen ist in folgender Abbildung dargestellt. Die Abbildung zeigt auch, dass die PLA bereits so programmiert ist, dass sie vier Boolesche Funktionen erzeugt, die in Kürze erläutert werden.

Jeder Ausgang wird von einem festen OR-Gatter erzeugt, und jeder Eingang enthält ein Puffer-Inverter-Gatter. Die Einheit ist in vier Abschnitte unterteilt, von denen jeder aus einem AND-OR-Array besteht. Jeder Abschnitt enthält drei programmierbare AND-Gatter und ein festes OR-Gatter. Zehn vertikale Linien, die jede horizontale Linie im Diagramm kreuzen, stellen die zehn programmierbaren Eingangsverbindungen für jedes AND-Gatter dar. Die Mehrfacheingangskonfiguration des AND-Gatters wird durch die horizontale Linie dargestellt. Ein Puffer-Inverter-Gatter empfängt einen der Ausgänge und speist ihn in zwei Eingänge eines AND-Gatters zurück.

Abbildung 70: Struktur eines programmierten PALs



Quelle: Moustafa Nawito, 2002, in Anlehnung an Mano, 2013, S. 328.

Die Booleschen Funktionen müssen beim Entwurf mit einem PAL vereinfacht werden, damit sie in jeden Bereich passen. Ein Produktterm kann nicht zwischen zwei oder mehr OR-Gattern geteilt werden, anders als bei einem PLA. Folglich kann jede Funktion für sich allein vereinfacht werden, ohne Berücksichtigung der gemeinsamen Produktterme. Die

Anzahl der Produktterme in jedem Abschnitt ist festgelegt. Wenn die Anzahl der Terme in der Funktion zu groß ist, kann es notwendig sein, zwei Abschnitte zu verwenden, um eine Boolesche Funktion zu implementieren.

Ein Beispiel für die Verwendung eines PAL beim Entwurf einer kombinierten Schaltung sind die folgenden Booleschen Funktionen:

$$\begin{aligned}w(A, B, C, D) &= \sum(2, 12, 13) \\x(A, B, C, D) &= \sum(7, 8, 9, 10, 11, 12, 13, 14, 15) \\y(A, B, C, D) &= \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15) \\z(A, B, C, D) &= \sum(1, 2, 8, 12, 13)\end{aligned}$$

Vereinfacht man die vier Funktionen auf eine minimale Anzahl von Termen, so erhält man die folgenden Booleschen Funktionen:

$$\begin{aligned}w &= ABC' + A'B'CD' \\x &= A + BCD \\y &= A'B + CD + B'D' \\z &= ABC' + A'B'CD' + AC'D' + A'B'C'D = w + AC'D' + A'B'C'D\end{aligned}$$

Man beachte, dass die Funktion für z vier Produktterme aufweist. Die logische Summe von zwei dieser Terme ist gleich w . Durch die Verwendung von w ist es möglich, die Anzahl der Terme für z von vier auf drei zu reduzieren. Die PAL-Programmiertabelle ist ähnlich wie die für die PLA, nur dass lediglich die Eingänge der AND-Gatter programmiert werden müssen.

Die folgende Tabelle enthält die PAL-Programmiertabelle für die vier Booleschen Funktionen. Die Tabelle ist in vier Abschnitte mit jeweils drei Produkttermen unterteilt, um der PAL von obiger Abbildung zu entsprechen. Die ersten beiden Abschnitte benötigen nur zwei Produktterme, um die Boolesche Funktion zu implementieren. Der letzte Abschnitt, für den Ausgang z , benötigt vier Produktterme. Mithilfe der Ausgabe von w können wir die Funktion auf drei Terme reduzieren. Die Fuse-Map für den PAL, wie sie in der Programmier-tabelle angegeben ist, ist in der oberen Abbildung dargestellt. Für jede 1 oder 0 in der Tabelle markieren wir den entsprechenden Schnittpunkt im Diagramm mit dem Symbol für eine intakte Sicherung. Für jeden Strich markieren wir im Diagramm durchgebrannte Sicherungen sowohl im wahren als auch im Komplement-Eingang. Wenn das AND-Gatter nicht verwendet wird, lassen wir alle seine Eingangssicherungen intakt. Da der entsprechende Eingang sowohl den wahren Wert als auch das Komplement jeder Eingangsvariablen empfängt, haben wir $AA' = 0$ und der Ausgang des AND-Gatters ist immer 0.

Wie bei allen PLDs wird der Entwurf mit PALs durch den Einsatz von CAD-Techniken vereinfacht. Das Auslösen der internen Sicherungen ist ein Hardware-Verfahren, das mithilfe spezieller elektronischer Instrumente durchgeführt wird.

Tabelle 23: Programmiertabelle des PALs

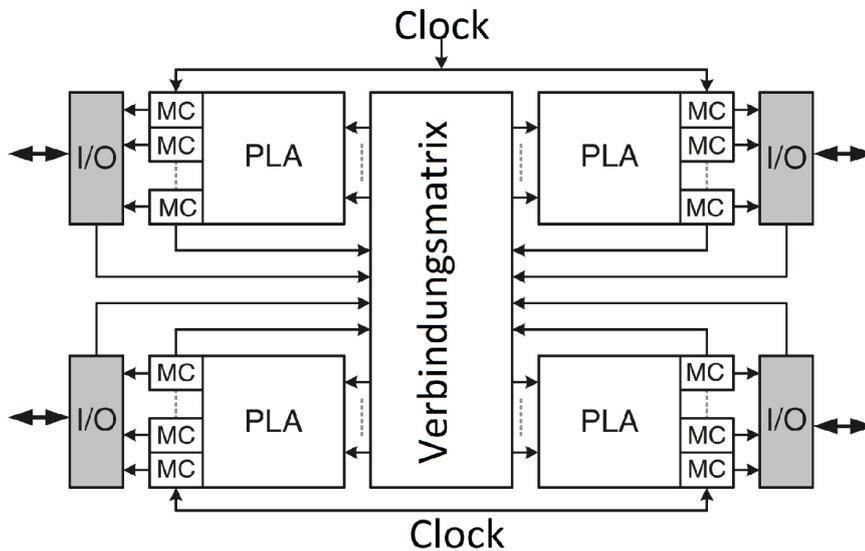
Produkt-term	AND- Eingänge					Ausgänge
	A	B	C	D	w	
1	1	1	0	-	-	$w = ABC' + A'B'CD'$
2	0	0	1	0	-	
3	-	-	-	-	-	$x = A + BCD$
4	1	-	-	-	-	
5	-	1	1	1	-	
6	-	-	-	-	-	$y = A'B + CD + B'D'$
7	0	1	-	-	-	
8	-	-	1	1	-	
9	-	0	-	0	-	$z = w + AC'D' + A'B'C'D$
10	-	-	-	-	1	
11	1	-	0	0	-	
12	0	0	0	1	-	

Quelle: Moustafa Nawito, 2022 in Anlehnung an Mano, 2013, S. 327.

7.2 Complex Programmable Logic Devices (CPLD)

Die sogenannten Complex Programmable Logic Devices (CPLDs) (komplexe programmierbare Logikbausteine) sind eine Erweiterung des PLA-Konzepts. Zahlreiche dieser Bausteine nutzen das PLA-Prinzip und kombinieren mehrere PLA-Blöcke mit einer programmierbaren Verbindungsmatrix, die die Verbindung der Eingänge eines PLA-Blocks mit den Ausgängen eines anderen Blocks ermöglicht. Bei dieser Methode kann nur die disjunkte Form zur Umsetzung der logischen Funktion verwendet werden. Eine weitere Möglichkeit sind Kaskaden von disjunkten Stufen. Dies kann zu einer vorteilhafteren Implementierung führen, insbesondere bei komplizierteren Funktionen. Folgende Abbildung zeigt den grundlegenden Aufbau eines CPLD. Die programmierbaren AND/OR-Strukturen (PLA) und die sogenannten Makrozellen sind jeweils Merkmale von CPLDs (micro-cell MC). Die Register Ebenen der einfachen PLA-Bausteine können als Erweiterung der Makrozellen betrachtet werden.

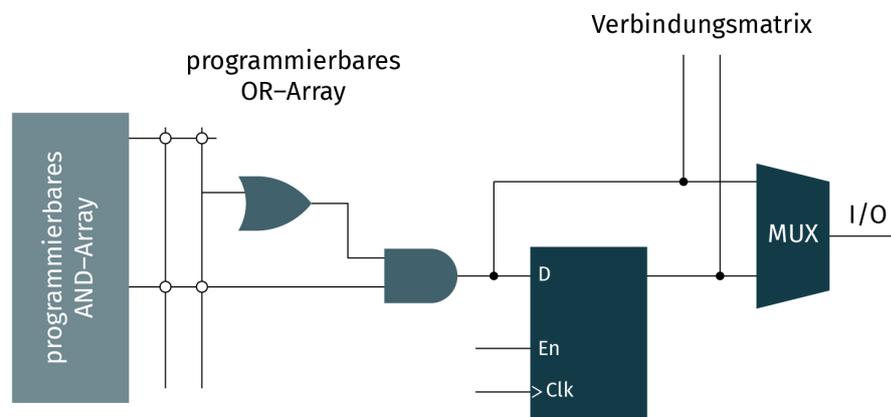
Abbildung 71: Struktur eines CPLDs auf PLA-Basis



Quelle: Moustafa Nawito, 2022.

Die folgende Abbildung zeigt das konzeptionelle Layout der Makrozelle eines CPLD der Coolrunner II-Serie der Firma Xilinx. Als zentrales Bauteil der Makrozelle fungiert das D-Flip-Flop mit dem über die PLA-Struktur verbundenen D -Eingang. Es kann gewählt werden, ob der von der AND/OR-Struktur berechnete Term nicht-invertiert oder mithilfe eines Exklusiv-Oder-Gatters invertiert an das D -Flip-Flop geschickt wird. Der Term kann entweder synchron oder asynchron (Abgriff vor dem Flip-Flop) an die Verbindungsmatrix zurückgegeben werden (Abgriff nach dem Flip-Flop). Ebenso ist es möglich, zu wählen, ob die Ausgabe eines Wertes synchron oder asynchron erfolgen soll. Das Flip-Flop der Makrozelle in der Abbildung hat Eingänge für Enable, Set und Reset, die ebenfalls von der PLA-Struktur verwaltet werden. In der Praxis stellt sich die Frage, welcher CPLD-Baustein am besten geeignet ist, eine bestimmte Aufgabe zu lösen (Gehrke et al., 2016, S. 71).

Abbildung 72: Schematischer Aufbau einer Makrozelle



Quelle: Moustafa Nawito, 2022 in Anlehnung an Gehrke et al., 2016, S. 272.

Es ist von entscheidender Bedeutung zu berücksichtigen, wie viele Gatter ein bestimmter CPLD zusätzlich zur erforderlichen Anzahl von Ein- und Ausgängen ersetzen kann. Oft kann die Architektur eines CPLD allein keine Antwort auf diese Frage geben. Die Realisierung wird ineffizient, wenn die zu implementierende Funktion schlecht in die Struktur des CPLD-Bausteins passt, sodass viele der verfügbaren CPLD-Ressourcen nicht genutzt werden können. Darüber hinaus hat auch die Effektivität der Syntheseprogramme, die zur Implementierung der erwünschten Funktion verwendet werden, einen großen Einfluss auf das Ergebnis. Um den Ressourcenverbrauch für verschiedene Funktionsblöcke vorherzusagen zu können, werden daher in der Praxis vor der endgültigen Auswahl eines CPLD-Funktionsblocks vielfache Syntheseläufe durchgeführt, es sei denn, es lassen sich Erfahrungen aus vergleichbaren Fällen ableiten.

Es gibt zahlreiche Hersteller, die CPLDs anbieten. Xilinx, Altera, Lattice, MicroSemi und Atmel sind die wichtigsten. Die Hersteller geben gelegentlich an, wie viele Makrozellen einem CPLD entsprechen, um einen Vergleich mit PLA-basierten CPLDs zu ermöglichen (Gehrke et al., 2016, S. 272).

7.3 FPGA

Der Begriff „Gate-Array“ bezog sich ursprünglich auf Geräte mit einer großen Anzahl eingebauter Logikgatter. Kunden konnten die Verdrahtung der Gatter und anschließend die zu implementierende Logikfunktion angeben. Die Verdrahtung der Gatter wurde dann im Auftrag des Kunden in einer Halbleiterfabrik vorgenommen. Field Programmable Gate Arrays (FPGA) bieten die Möglichkeit, beim Kunden oder „vor Ort“ – engl. „in field“ programmiert oder angepasst zu werden, d. h., sie müssen nicht in der Halbleiterfabrik programmiert werden. Der Benutzer kann auch den Zweck der Field Programmable Gate Arrays angeben, die heutzutage die am häufigsten verwendete Art von Gate Arrays sind. Durch die elektrische Programmierung entfallen die aufwändigen Produktionsverfahren in

einer Halbleiterfabrik; stattdessen kann die gewünschte Logikfunktion mithilfe eines Programmiergeräts in Sekundenschnelle in ein FPGA geladen werden. FPGAs haben sich weitgehend durchgesetzt, da sie zu erschwinglichen Preisen erhältlich sind. Die grundlegenden Ideen hinter diesen Bausteinen werden folgend näher erläutert.

Ein wichtiges Merkmal von FPGAs ist die Möglichkeit, wesentlich anspruchsvollere Funktionalitäten umzusetzen, als dies mit PALs oder CPLDs denkbar wäre. FPGAs unterscheiden sich von vielen CPLDs auch darin, wie die „Programmierbarkeit“ technisch realisiert wird. FPGAs sind auf einer tabellenbasierten Architektur aufgebaut, während einfache Logikbausteine (einschließlich CPLDs) im Kern eine zweistufige AND/OR-Struktur haben. Das grundlegende Konzept eines FPGA ist recht einfach: Man entwirft einen Baustein mit zahlreichen winzigen Logikblöcken, die programmierbare Lookup-Tabellen (Nachschlagetabellen LUTs) enthalten. Eine LUT kann zum Beispiel vier Eingänge und einen Ausgang haben. Die logische Funktion, mit der der Ausgangswert aus den Eingängen berechnet werden soll, wird durch die anschließende Programmierung der LUTs bestimmt.

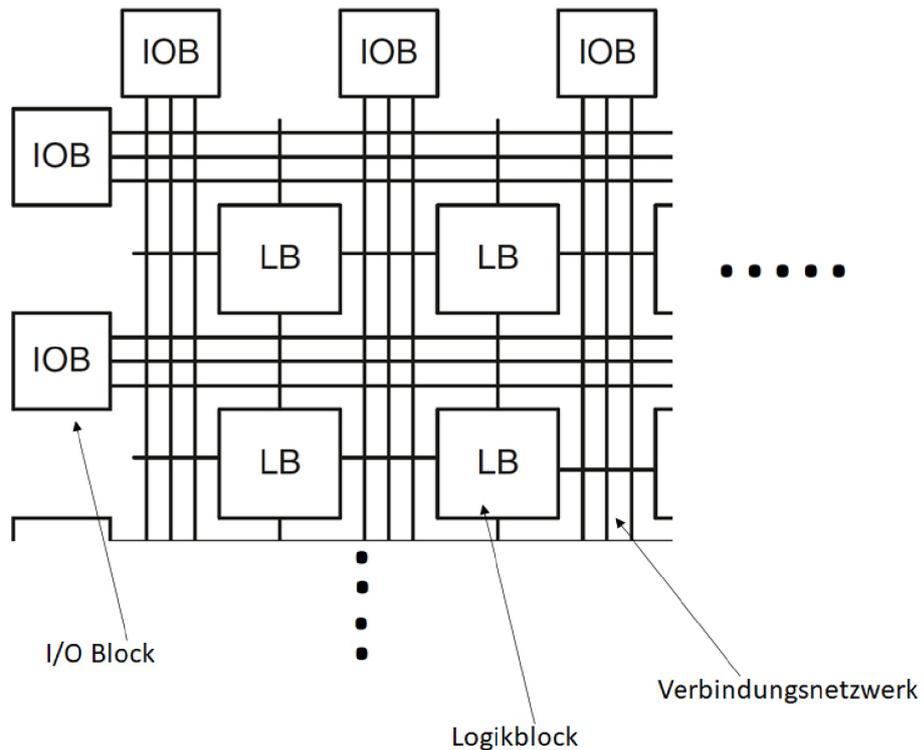
Zu den Logikblöcken gehören auch Flip-Flops, weil sie für die Entwicklung eines digitalen Systems benötigt werden. Da dies den tatsächlichen Schaltungen entspricht, sind LUTs und Flip-Flops in der Regel in gleicher Anzahl vorhanden. Jede LUT wird in diesem Fall mit einem Flip-Flop versehen, damit ihr Ausgangswert auch in einem Logikblock gespeichert werden kann. Die Verbindungen zwischen den Logikblöcken werden über ein Verbindungsnetzwerk hergestellt. Durch programmierbare Schalter kann das Verbindungsnetzwerk individuell gestaltet werden (Schaltmatrix). Wenn Daten von einem Logikblock zu einem bestimmten anderen Logikblock übertragen werden müssen, sind die „Schalter“ innerhalb des Netzes so ausgelegt, dass sie eine elektrische Verbindung zwischen den beiden Logikblöcken herstellen. Dies ist vergleichbar mit der Funktionsweise von Eisenbahnschienen. Im Gegensatz zu einer Eisenbahnverbindung werden die Schalter nicht dynamisch während des Betriebs ausgetauscht, sondern nach dem Einschalten einmalig für die entsprechende Logikfunktion konfiguriert (Gehrke et al., 2016, S. 273).

Mehrere LUTs können kombiniert werden, um komplexe logische Funktionen zu erreichen, da die Logikblöcke und das Verbindungsnetzwerk programmierbar sind. Die Anzahl der verfügbaren Logikblöcke setzt natürlich eine Grenze für die maximale Komplexität der Gesamtfunktion. Wenn außerdem eine extrem komplizierte Signalverdrahtung erforderlich ist, ist es natürlich möglich, dass das Verdrahtungsnetzwerk der begrenzende Faktor für die Implementierung eines FPGA-basierten Systems ist.

FPGAs enthalten neben Logikblöcken und einem Verbindungsnetzwerk auch Input/Output-Blöcke (IO-Blöcke oder kurz IOBs). Diese Blöcke können unter anderem dazu verwendet werden, Logikpegel zu verändern. Mit den IOBs lassen sich die Pegel interner Signale so verändern, dass sie beispielsweise an Geräte mit einer Versorgungsspannung von 3,3 V gesendet werden können, wenn ein FPGA mit einer internen Versorgungsspannung von 1,8 V arbeitet. Die IOBs haben auch Eigenschaften, die für eine besonders schnelle Ein- und Ausgabe nützlich sein können. Ein Beispiel dafür sind IOB-interne Parallel-Seriell-Wandler auf Basis von Schieberegistern (Serializer). Die Logikblöcke versorgen die IOBs gleichzeitig mit Ausgangsdaten (z. B. 4 oder 8 Bit). Die Daten innerhalb der IOBs werden „serialisiert“, d. h. Bit für Bit an den äußeren Anschluss des FPGAs ausgegeben. Obwohl die logische Funktion, die im FPGA implementiert ist, eher langsam ist, kann mit dieser

Methode eine hohe Datenrate am Ausgang des FPGAs erreicht werden (Gehrke et al., 2016, S. 274). Für den Eingang können De-Serialisierer verwendet werden, die die Daten seriell lesen und in paralleler Form an die interne Logik des FPGAs weitergeben. Die untere Abbildung zeigt den grundsätzlichen Aufbau eines FPGAs, der aus Logikblöcken, IO-Blöcken und einem Verbindungsnetzwerk besteht.

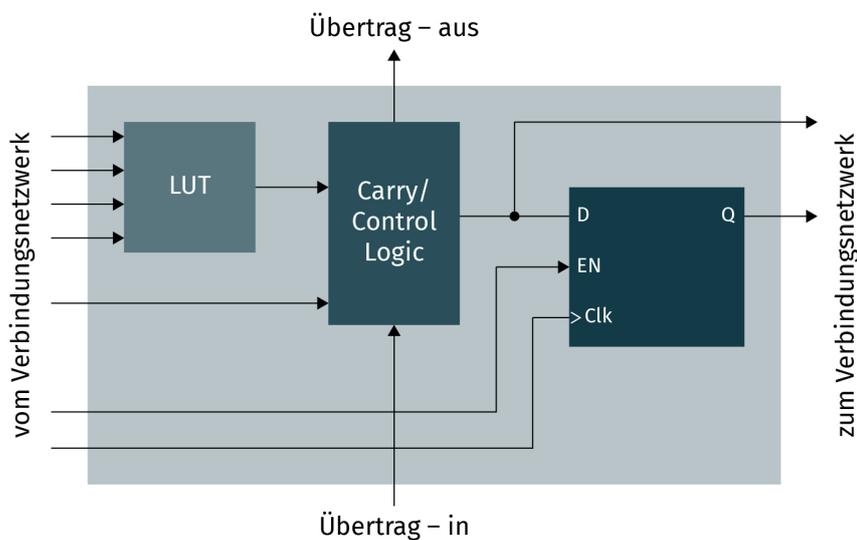
Abbildung 73: Prinzipieller Aufbau eines FPGAs



Quelle: Moustafa Nawito, 2022.

In folgender Abbildung ist die Struktur eines grundlegenden Logikblocks dargestellt. Das Verbindungsnetzwerk ist mit der Mehrzahl der Ein- und Ausgänge verbunden. Der Carry/Control-Logic-Block ist zusätzlich mit den Anschlüssen CIN und COUT verbunden. Mithilfe dieser Anschlüsse und der zugehörigen Logik wird ein sehr schneller Durchlauf des Übertragsbits eines Addierers ermöglicht. Die Verzögerungszeit der Addition kann mithilfe unserer einzigartigen Übertragslogik stark verkürzt werden.

Abbildung 74: Aufbau eines Logikblocks



Quelle: Moustafa Nawito, 2022, in Anlehnung an Gehrke et al., 2016, S. 275.

Um eine Schaltung auf einem FPGA zu realisieren, wird eine Beschreibung der Schaltung mit einer Hardwarebeschreibungssprache erstellt. Diese Beschreibung kann dann mit speziellen CAD-Programmen auf den FPGA übertragen werden, um die gewünschte Hardware zu implementieren.

7.4 Einführung in HDL

Nur wenn die Schaltung winzig klein ist, kann man Logikschaltungen manuell erstellen. Für alles andere, z. B. für eine praktische Schaltung, setzen Entwicklungsteams computer-gestützte Entwurfswerkzeuge ein. Computergestützte Entwurfswerkzeuge helfen den Designer:innen, ihre Phantasie und Arbeit optimal zu nutzen, und verringern gleichzeitig die Wahrscheinlichkeit, dass sie einen fehlerhaften Entwurf erstellen. Alle modernen Entwurfswerkzeuge stützen sich auf eine Hardware-Beschreibungssprache (Hardware Description Language [HDL]), um eine Schaltung in Software zu definieren, zu erstellen und zu testen, bevor sie überhaupt hergestellt wird, da die Erstellung von Prototypen integrierter Schaltungen zu kostspielig und zeitaufwändig ist.

Eine textbasierte Computersprache, die Hardwarebeschreibungssprache, wird zur textuellen Beschreibung der Hardware digitaler Systeme verwendet. Sie ähnelt einer Standard-Computerprogrammiersprache wie C, konzentriert sich aber auf die Definition der Hardware-Architektur und der Funktionsweise von Logikschaltungen. Sie kann zur Darstellung von Booleschen Ausdrücken, Wahrheitstabellen, Logikdiagrammen und komplexen Abstraktionen des Verhaltens eines digitalen Systems verwendet werden. Die Beziehung zwischen den Signalen, die die Eingänge einer Schaltung bilden, und den Signalen, die die Ausgänge der Schaltung darstellen, ist ein Merkmal einer HDL-Sprache. Eine HDL-

Beschreibung eines AND-Gatters erklärt zum Beispiel, wie sich die logischen Werte der Eingänge des Gatters auf den logischen Wert des Ausgangs des Gatters auswirken. Die folgende Tabelle fasst die wichtigsten Unterschiede zwischen einer HDL und einer Programmiersprache zusammen.

Tabelle 24: Unterschiede zwischen HDL und Software-Programmiersprache

HDL	Software-Programmiersprache
Die Struktur und das Verhalten elektrischer Schaltungen, vor allem digitaler Logikschaltungen, werden durch HDL definiert.	Um eine CPU in die Lage zu versetzen, eine bestimmte Funktion auszuführen, erstellt die Software-Sprache eine Reihe von Anweisungen.
Sie beschreibt, wie sich digitale Schaltungen verhalten.	Sie hilft bei der Entwicklung von zahlreichen Anwendungen.
Die Arbeit mit ihr ist schwieriger.	Die Arbeit mit ihr ist weniger kompliziert.
Die Entwicklung und Anwendung von textbasierten Beschreibungen von Schaltungen bildet die Grundlage dieses Designs.	Es wird mit dem Ziel, ausführbare Softwareprogramme zu erstellen, die auf einem geeigneten Prozessor laufen können, verwendet.
Sie ist eine Sprache, die die räumliche Organisation und das zeitliche Verhalten von Hardware syntaktisch und semantisch unterstützt.	Sie ist eine Sprache, die Computeranweisungen interpretieren und ausführen kann.

Quelle: Moustafa Nawito, 2022.

Eine HDL wird als Dokumentationssprache verwendet, um digitale Systeme in einer Weise darzustellen und zu beschreiben, die sowohl von Menschen als auch von Computern verstanden werden kann und sich als Kommunikationssprache zwischen Entwicklern eignet. Das Sprachmaterial kann effektiv von Computersoftware verarbeitet werden und lässt sich bequem speichern, abrufen, bearbeiten und versenden.

HDLs werden in einer Reihe wichtiger Phasen des Entwurfsablaufs einer integrierten Schaltung eingesetzt:

- Bei der Entwurfserfassung wird eine HDL-basierte Beschreibung der Funktionalität erstellt, die in Hardware implementiert wird. Je nach HDL kann die Beschreibung in Form von Booleschen Logikgleichungen, Wahrheitstabellen, einer Netzliste mit verbundenen Gattern oder einem abstrakten Verhaltensmodell erfolgen. Die Aufteilung einer größeren Schaltung in kleinere, gekoppelte und voneinander abhängige Funktionseinheiten kann ebenfalls durch das HDL-Modell dargestellt werden. Im Allgemeinen ist die Schaltung auf zwei Arten beschreibbar:
 - eine strukturelle Beschreibung (structural model), bei der die tatsächlichen Komponenten und Verbindungen aufgelistet werden, ähnlich der SPICE-Syntax, und
 - eine verhaltensbasierte Beschreibung (behavioral model), bei der nur eine Beschreibung der beabsichtigten Funktion auf oberster Ebene gegeben wird.
- Die Logiksimulation demonstriert mithilfe eines Computers das Verhalten eines digitalen Systems. Ein Simulator versteht die HDL-Beschreibung und erzeugt entweder eine verständliche Ausgabe, z. B. eine Folge von Eingangs- und Ausgangssignalwerten, die zeitlich geordnet sind, oder er zeigt die Wellenformen der Signale an. Bevor die Hard-

ware hergestellt wird, kann man mithilfe der Schaltungssimulation vorhersagen, wie sie funktionieren wird. Ohne die Schaltung physisch zu bauen und zu benutzen, kann die Simulation Funktionsfehler in einem Entwurf finden. Die entsprechenden HDL-Anweisungen sind veränderlich, um Fehler zu beheben, die während einer Simulation gefunden werden. Die Stimuli (d. h. die logischen Werte der Eingänge der Schaltung), die zum Testen der Funktionalität des Entwurfs verwendet werden, werden als Prüfstand bezeichnet. Um ein digitales System nachzubilden, muss der Entwurf also zunächst in einem HDL spezifiziert und dann mit einer Testschaltung (testbench) getestet werden, die ebenfalls in einem HDL geschrieben wurde. Eine andere und kompliziertere Methode verwendet formale mathematische Techniken, um die richtige Funktionalität einer Schaltung zu demonstrieren. Wir werden hier nur über die Simulation sprechen.

- Der Prozess der Erstellung einer Netzliste – einer Liste physischer Komponenten und ihrer Verbindungen – aus dem in einer HDL gegebenen Modell eines digitalen Systems wird als Logiksynthese bezeichnet. Die Hardware-Äquivalente der Gatter in der Liste können verwendet werden, um einen integrierten Schaltkreis zu konstruieren oder eine Leiterplatte mithilfe der Netzliste zu entwerfen. Die Logiksynthese ist vergleichbar mit der Konstruktion eines Programms in einer traditionellen Hochsprache. Der Unterschied besteht darin, dass bei der Logiksynthese eine Datenbank erstellt wird, in der die Komponenten und die Organisation einer Schaltung detailliert beschrieben sind, und nicht ein Objektcode. Die Datenbank enthält Anweisungen, wie eine physische integrierte Schaltung zu erstellen ist, die die in einer HDL-Deklaration angegebene Funktionalität in Silizium umsetzt. Die Logiksynthese konzentriert sich auf die Elemente eines digitalen Entwurfs, die mithilfe von Computersoftware automatisiert werden können, und basiert auf formalen, präzisen Techniken zur Konstruktion digitaler Schaltungen. Dank der Software für die Logiksynthese können heute riesige, anspruchsvolle Schaltungen entworfen werden.
- Die Verifizierung des Zeitverhaltens (timing verification) bestätigt, dass die hergestellte integrierte Schaltung mit einer spezifizierten Geschwindigkeit arbeitet. Ein Signalübergang am Eingang einer Schaltung kann nicht sofort zu einer Änderung des logischen Werts am Ausgang einer Schaltung führen, da jedes logische Gatter in einer Schaltung eine Ausbreitungsverzögerung hat. Die Ausbreitungsverzögerung bestimmt letztlich die maximale Betriebsgeschwindigkeit einer Schaltung. Die Timing-Verifizierung stellt sicher, dass kein Signalpfad durch die Ausbreitungsverzögerung beeinträchtigt wird. Dieser Prozess wird durchgeführt, bevor die Schaltung für die Produktion zur Verfügung gestellt wird und nachdem die Logiksynthese die genauen Komponenten der Schaltung bestimmt hat.
- Bei der Fehlersimulation im VLSI-Schaltungsentwurf wird das Verhalten einer idealen Schaltung mit dem einer Schaltung mit einer prozessbedingten Unvollkommenheit verglichen. Eine Schaltung kann mit einem Fehler hergestellt werden, wenn sich Staub oder andere Verunreinigungen in der Luft des Reinraums befinden. Ein Schaltkreis mit einem Problem wird nicht die gleiche Leistung erbringen wie ein Schaltkreis ohne Problem. Über die Fehlersimulation werden Eingangsreize gefunden, mit denen sich die fehlerhafte Schaltung von der fehlerfreien Schaltung unterscheiden lässt. Um sicherzustellen, dass nur qualitativ hochwertige Geräte an den Kunden geliefert werden, werden diese Testmuster zur Prüfung der produzierten Produkte verwendet. Die Testerstellung und die Fehlersimulation können in verschiedenen Entwurfsphasen erfolgen, werden aber immer vor der Produktion abgeschlossen, um die Katastrophe zu vermeiden, dass eine Schaltung entsteht, deren interne Logik nicht getestet werden kann.

VHDL und Verilog

Die Entwickler integrierter Schaltungen verwenden sowohl private als auch öffentliche HDLs. Die IEEE unterstützt VHDL und Verilog als die beiden Standard-HDLs im öffentlichen Bereich. Das amerikanische Verteidigungsministerium hat die Verwendung von VHDL vorgeschrieben. Der erste Buchstabe der Abkürzung für einen sehr schnellen integrierten Schaltkreis (Very High Speed Integrated Circuit) VHSIC, wird durch das V in VHDL impliziert. Verilog war ursprünglich eine private HDL von Cadence Design Systems, aber als Schritt zur Genehmigung als IEEE-Standard übergab Cadence das Eigentum an Verilog an eine Gruppe von Unternehmen und akademischen Einrichtungen, die als Open Verilog International (OVI) bekannt ist. Im Prinzip ist VHDL schwieriger als Verilog zu lernen, wobei beide Sprachen wichtig und relevant sind. Verilog-HDL wurde zum ersten Mal 1995 als Standard-HDL akzeptiert; danach wurde es 2001 und 2005 überarbeitet und verbessert.

Es gehört nicht zum Umfang dieses Textes, auf die Details von VHDL und Verilog einzugehen. Interessierte Lesende finden mehr Informationen zu diesem Thema bei Gehrke et al. (2016). Um die einleitende Diskussion über HDL zu vervollständigen, stellen wir jedoch ein kleines Beispiel vor.

Die folgende Abbildung zeigt die Implementierung der einfachen kombinatorischen Funktion $F = A'B'C' + AB'C' + AB'C$ in beiden Sprachen. Wie man sieht:

- Ein Verilog-Modul beginnt mit dem Modulnamen und einer Auflistung der Ein- und Ausgänge. Die Anweisung `assign` beschreibt die kombinierte Logik. `~` steht für NOT, `&` für AND und `|` für OR. Verilog-Signale wie die Eingänge und Ausgänge sind Boolesche Variablen (0 oder 1). Sie können auch fließende und undefinierte Werte haben.
- VHDL-Code besteht aus drei Teilen: der `library-use`-Klausel, der `entity`-Deklaration und dem `architecture`-Körper. Die `library-use`-Klausel ist erforderlich. Die `entity`-Deklaration listet den Modulnamen sowie seine Ein- und Ausgänge auf. Der `architecture`-Körper definiert, was das Modul tut. VHDL-Signale, wie Eingänge und Ausgänge, müssen eine Typdeklaration haben. Digitale Signale sollten als Typ `STD_LOGIC` deklariert werden. `STD_LOGIC`-Signale können einen Wert von '0' oder '1', aber auch fließende und undefinierte Werte aufweisen. Der `STD_LOGIC`-Typ ist in der Bibliothek `IEEE.STD_LOGIC_1164` definiert, weshalb diese Bibliothek verwendet werden muss. In VHDL gibt es keine gute Standardreihenfolge für Operationen, daher sollten Boolesche Gleichungen in Klammern gesetzt werden.

Abbildung 75: VHDL- und Verilog-Beispiele

Verilog	VHDL
<pre>module simplefunction (input a, b, c, output y); assign y = ~a & ~b & ~c a & ~b & ~c a & ~b & c; endmodule</pre>	<pre>library IEEE; use IEEE.STD_LOGIC_1164.all; entity simplefunction is port (a, b, c: in STD_LOGIC; y: out STD_LOGIC); end; architecture synth of simplefunction is begin y <= ((not a) and (not b) and (not c)) or (a and (not b) and (not c)) or (a and (not b) and c); end;</pre>

Quelle: Moustafa Nawito, 2022.



ZUSAMMENFASSUNG

Die drei Haupttypen von programmierbaren Logikbausteinen sind PROM, PLA und PAL. Allen gemeinsam ist, dass eine logische Funktion in Hardware als Summe von Produkten implementiert wird. Bei einem PLA sind sowohl die AND- als auch die OR-Reihen programmierbar, was mehr Flexibilität bietet, aber die Komplexität des Entwurfs erhöht. Bei einem PAL hingegen ist das OR-Array fest, was die Flexibilität verringert und die Programmierbarkeit des Entwurfs erleichtert.

Durch die Kombination vieler PAL-Einheiten kann ein CPLD aufgebaut werden, um kompliziertere Schaltungen zu realisieren. Intern enthält der CPLD viele Makrozellen, die hauptsächlich aus einem D-Flip-Flop bestehen.

Für die Realisierung von komplizierteren Schaltungen werden FPGAs eingesetzt. Die interne Struktur besteht aus einem regelmäßigen Array von Logikblöcken, die über Input-Output-Blöcke mit der Außenwelt verbunden sind.

HDLs sind sehr leistungsfähige Werkzeuge für den digitalen Entwurf. Sie werden in allen Phasen des Entwurfsprozesses eingesetzt und gewährleisten eine genaue und relativ schnelle Implementierung digitaler Systeme, insbesondere im Vergleich zu manuellen Methoden oder zu traditioneller analoger Entwurfstechnik.

ANHANG

LITERATURVERZEICHNIS

- Fricke, K. (2018). *Digitaltechnik. Lehr- und Übungsbuch für Elektrotechniker und Informatiker*. (8. Aufl.), Springer Vieweg. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.37732&site=eds-live&scope=site>
- Gehrke, W., Winzker, M., Urbanski, K. & Weitowitz, R. (2016). *Digitaltechnik. Grundlagen, VHDL, FPGAs, Mikrocontroller*. Springer. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsbas&AN=edsbas.C8E5E9E7&site=eds-live&scope=site>
- Mano, M. & Ciletti, M., (2013). *Digital Design with an Introduction to Verilog HDL*. (5. Auflage), Pearson.
- Liebig, H. & Thome, S. (2006). *Logischer Entwurf digitaler Systeme*. (4. Auflage), Springer.
- Psychpsy (2022). Datei:ASCII-Table-wide.svg, Wikimedia Commons, [CC0 1.0], <https://de.wikipedia.org/wiki/Datei:ASCII-Table-wide.svg>

ABBILDUNGSVERZEICHNIS

Tabelle 1: Zahlen von 0–15 für Basis 10, 2, 8 und 16	16
Abbildung 1: Die Zahl 1781 im Dezimalsystem	17
Abbildung 2: Umwandlung der Dezimalzahl 1781 zum Dualsystem	17
Abbildung 3: Umwandlung der Dezimalzahl 1781 zum Oktalsystem	17
Abbildung 4: Umwandlung der Dezimalzahl 1781 zum Hexadezimalsystem	18
Abbildung 5: Umwandlung der Dezimalzahl 2451,3 zum Hexadezimalsystem	18
Tabelle 2: Vorzeichenbehaftete Binärzahlen	24
Tabelle 3: Binary Coded Decimal (BCD)	28
Tabelle 4: BCD, 2421, Exzess–3 und 8,4,–2,–1 Dezimalcodes	30
Tabelle 5: Gray-Code	31
Abbildung 6: Siebenstelliger ASCII-Code	33
Tabelle 6: Wahrheitstabellen für NOT, AND, und OR-Operationen	38
Tabelle 7: Theoreme und Postulate der binären Logik	39
Tabelle 8: Wahrheitstabelle für die Funktion $F1 = (x + y)z'$	40
Abbildung 7: Schaltplan für die Funktion $F1 = (x + y)z'$	41
Abbildung 8: Schaltplan für die Funktion $F1 = xz' + yz'$	42
Tabelle 9: Wahrheitstabelle der Funktion F2 für die Darstellung von KDNF und KKNF	43
Tabelle 10: Wahrheitstabelle der Funktion V	45
Abbildung 9: Schaltplan der Funktion V	46
Abbildung 10: KV-Diagramm für eine Funktion mit vier Variablen	47

Tabelle 11: Wahrheitstabelle der Funktion $Y(A, B, C, D) = \sum(m_0, m_1, m_2, m_4, m_5, m_6, m_8, m_9, m_{12}, m_{13}, m_{14})$	48
Abbildung 11: KV-Diagramm der Funktion $Y(A, B, C, D) = \sum(m_0, m_1, m_2, m_4, m_5, m_6, m_8, m_9, m_{12}, m_{13}, m_{14})$	50
Abbildung 12: KV-Diagramme für Funktionen mit zwei bis vier Variablen	51
Abbildung 13: Unvollständige Funktion mit Don't Care Bedingungen	52
Abbildung 14: KV-Diagramm der Funktion $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$	53
Abbildung 15: Realisierung von NOT, AND und OR mittels NAND	54
Abbildung 16: Realisierung von NOT, AND und OR mittels NOR	55
Abbildung 17: Beispiel für Spannungs- und Logikpegel für 3 V Logikgatter	59
Abbildung 18: Eingangs- und Ausgangssingale für NOT-, OR- und AND-Gatter	60
Abbildung 19: Alle Booleschen Funktionen für zwei Variablen	61
Abbildung 20: Positive und negative Logik	63
Abbildung 21: Blockschaltbild eines Schaltnetzes	65
Abbildung 22: Beispiel zur Analyse von Schaltnetzen	67
Tabelle 12: Wahrheitstabelle für die Funktionen F1 und F2	67
Tabelle 13: Wahrheitstabelle zum BCD zu 2421 Wandler	69
Abbildung 23: Die vier KV-Diagramme zum BCD zu 2421 Wandler	70
Abbildung 24: Schaltplan des BCD zu 2421 Wandlers	71
Abbildung 25: 3x8-Decoder	73
Tabelle 14: Wahrheitstabelle des 3x8-Decoders	73
Abbildung 26: 2x4-Decoder mit enable	74
Abbildung 27: 4x16-Decoder aus zwei 3x8 mit enable	75

Abbildung 28: Implementation der Funktion $F_2 = \sum(m_0, m_1, m_3, m_5, m_6)$ mittels 3x8-Decoders und OR-Gatter	76
Tabelle 15: Wahrheitstabelle eines 8x3-Encoders	77
Abbildung 29: Schaltplan und Symbol vom 2x1-Multiplexer	79
Abbildung 30: Schaltplan und Wahrheitstabelle vom 4x1-Multiplexer	80
Abbildung 31: Realisierung der Funktion $F = \sum(m_1, m_2, m_3, m_6)$ mit einem 4x1-Multiplexer	81
Tabelle 16: Wahrheitstabelle des Halbaddierers	85
Abbildung 32: Schaltplan des Halbaddierers	85
Tabelle 17: Wahrheitstabelle des Volladdierers	86
Abbildung 33: Schaltplan des Volladdierers	87
Abbildung 34: 4-Bit-Binäraddierer	88
Abbildung 35: 4-Bit Addierer-Subtrahierer mit Überlauferkennung	89
Abbildung 36: 2-Bit mal 2-Bit binäre Multiplikationsschaltung	91
Abbildung 37: 3-Bit mal 4-Bit binäre Multiplikationsschaltung	93
Abbildung 38: Blockdiagramm eines synchronen Schaltwerks	97
Abbildung 39: (a) SR-Latch aus NOR-Gattern, (b) SR-Latch aus NAND-Gattern	100
Abbildung 40: SR-Latch mit enable	101
Abbildung 41: Schaltdiagramm und Funktionstabelle vom D-Latch	102
Abbildung 42: Schaltsymbole vom SR-, S'R'- und D-Latch	102
Abbildung 43: Positive Flanke und negative Flanke	103
Abbildung 44: Master-Slave-D-Flip-Flop	104
Abbildung 45: D-Flip-Flop mit positiver Flankenauslösung	105
Abbildung 46: Schaltsymbol des D-Flip-Flops	106

Abbildung 47: JK-Flip-Flop	106
Abbildung 48: T-Flip-Flop	107
Tabelle 18: Charakteristische Tabelle des D-Flip-Flops	108
Abbildung 49: Sequenzielle Schaltung zur Analyse	109
Tabelle 19: Zustandstabelle der Schaltung zur Analyse	110
Abbildung 50: Zustandsdiagramm	111
Abbildung 51: Schaltdiagramm basierend auf Eingangsgleichungen	112
Tabelle 20: Reduzierbares Zustandsdiagramm	114
Abbildung 52: Entwurfsbeispiel	115
Abbildung 53: KV-Diagramme für D_A , D_B und y	116
Abbildung 54: Schaltplan des Entwurfsbeispiels	117
Abbildung 55: 4-Bit-Register	121
Abbildung 56: 4-Bit-Register mit paralleler Datenladung	122
Abbildung 57: 4-Bit-Schieberegister	123
Abbildung 58: 4-Bit universelles Schieberegister	125
Tabelle 21: Funktionstabelle des universellen Schieberegisters	125
Abbildung 59: 4-Bit-Binär-Asynchronzähler	127
Abbildung 60: BCD-Ripplezähler	129
Abbildung 61: 3-Dekadenzähler	130
Abbildung 62: 4-Bit-Synchronzähler	131
Abbildung 63: Synchroner Up-Down-Zähler	133
Abbildung 64: Frequenzteiler	134
Abbildung 65: Mealy- und Moore-Automat	135

Abbildung 66: Beispiel für einen Mealy-Automat	136
Abbildung 67: Beispiel für einen Moore-Automat	137
Abbildung 68: PROM, PAL und PLA	141
Abbildung 69: Interne Struktur eines PLAs	142
Tabelle 22: Programmiertabelle des PLAs	144
Abbildung 70: Struktur eines programmierten PALS	146
Tabelle 23: Programmiertabelle des PALS	148
Abbildung 71: Struktur eines CPLDs auf PLA-Basis	149
Abbildung 72: Schematischer Aufbau einer Makrozelle	150
Abbildung 73: Prinzipieller Aufbau eines FPGAs	152
Abbildung 74: Aufbau eines Logikblocks	153
Tabelle 24: Unterschiede zwischen HDL und Software-Programmiersprache	154
Abbildung 75: VHDL- und Verilog-Beispiele	157



IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt



Postanschrift
Albert-Proeller-Straße 15-19
D-86675 Buchdorf



media@iu.org
www.iu.org



Hilfe & Kontakt (FAQ)
Antworten rund um Dein Studium findest
Du jederzeit auf myCampus.