Digitaltechnik

Übersicht zu den Lehrveranstaltungen für Zug 2B
(im Aufbau, Version vom 14.06.2010)

Hinweis: Das **Vorlesungsskript** von Prof. Dr.-Ing. H. P. Bauer enthält in sehr übersichtlicher und kompakter Form den gesamten fachlichen Stoff dieser Lehrveranstaltung und bildet daher deren Grundlage.

Zweck der folgenden Dokumentation ist die Protokollierung des jeweils bis zum angegebenen Zeitpunkt durchgearbeiteten Inhalts und die Ergänzung mit weiteren Hinweisen.

Inhalt

1 Definition Digitaltechnik (zum Skript Kapitel 1) 2
2 Logische Verknüpfungen (zum Skript Kapitel 2) 2
3 Schaltalgebra (zum Skript Kapitel 3) 6
4 Schaltungsanalyse (zum Skript Kapitel 4)....10
5 Schaltungssynthese (zum Skript Kapitel 5) 10
6 Zahlensysteme der Digitaltechnik (zum Skript Kapitel 6)....22
7 Codierung (zum Skript Kapitel 7) 30
8 Rechenschaltungen (zum Skript Kapitel 8) 37
9 Schaltwerke (Sequenzielle Logik, zum Skript Kapitel 9) 38
10 Zähler, Frequenzteiler und Schieberegister (zum Skript Kapitel 10)....46
11 Digitale Auswahlschaltungen (Skript Kapitel 11)....48
12 D/A- und A/D-Wandler (Skript Kapitel 12)....49

Veranstaltung am 24.03.2010

1 Definition Digitaltechnik (zum Skript Kapitel 1)

Die Gliederung und Unterteilung des Fachgebietes der Elektrotechnik ist – wie bei jedem anderen Fachgebiet auch - auf sehr unterschiedliche Weise möglich. Für einen Eindruck muss man nur in das Vorlesungsverzeichnis eines elektrotechnischen Hochschul-Studienganges schauen. Neben vielen anderen sind zwei elementare Unterteilungen

„Gleichstromtechnik – Wechselstromtechnik“

„Analogtechnik – Digitaltechnik“,

welche das Gesamtgebiet jeweils unter bestimmten Schwerpunkten betrachten.

Während die Gleichstromtechnik die Erzeugung von zeitlich konstanten „Gleichgrößen“ wie Gleichströmen und Gleichspannungen und ihre Wirkung in Bauelementen (Widerstände, Spulen, Kondensatoren) und technischen Strukturen (Schaltungen, geometrische Formen) behandelt, konzentriert sich die Wechselstromtechnik auf zeitlich veränderliche „Wechselgrößen“. Unabhängig von der Art der Unterteilung werden alle Zusammenhänge letztlich mit Hilfe der berühmten Maxwellschen Gleichungen in ihrer integralen oder differenziellen Form und den daraus ableitbaren Vereinfachungen und Sonderfällen beschrieben.

Andere Perspektiven bietet die Sicht auf stetig veränderbare (analoge) oder sich unstetig, also sprunghaft verhaltende (diskrete) Spannungen und Ströme. Da man die Werte diskreter Größen im Gegensatz zu den analogen Größen abzählen kann, bezeichnet man sie als digitale Größen (aus dem Griechischen für „Finger“ → zum Zählen, siehe auch Fingerhut → Digitalis). Im einfachsten und idealen Fall nimmt eine solche digitale Größe als Spannung oder Strom nur zwei Werte an, z. B. 0 Volt und 5 Volt, bzw. 0 Milliampere und 20 Milliampere. Diesen ordnet man mathematisch logische Zustände wie „0“ und „1“ zu. Technisch kann man solche mathematisch exakten Werte nur angenähert erreichen und begnügt sich mit Wertebereichen (Skript 1.4) wie etwa

0.0 – 0.4 Volt → „0“ → L (für „Low“)

2.4 – 5.0 Volt → „1“ → H (für „High“)

Eine – recht abstrakte - Definition für das hier behandelte Gebiet der elektrischen Digitaltechnik (Digitaltechnik gibt es u. a. auch in der Pneumatik) kann sein:

> *Die elektrische Digitaltechnik ist die Technik der Erzeugung, Verarbeitung und Anwendung digitaler elektrischer Signale und der dazu eingesetzten technischen Mittel.*

2 Logische Verknüpfungen (zum Skript Kapitel 2)

Digitale Signale lassen sich einsetzen, um logische Verknüpfungen darzustellen. Diese können gewünschten Zuständen eines technischen Systems (z. B. Ampelsteuerung, Funktionen eines Getränke-Automaten, Fertigungsablauf einer Fahrzeugproduktion, Unterprogramm zur Multiplikation für einen Mikroprozessor) zugeordnet werden. Es gibt Grundfunktionen und daraus zusammensetzbare Funktionen. Die Komplexität kann dabei in einem sehr weiten Bereich von „gering“ bis „extrem hoch“ reichen. Stichworte

- *Verknüpft (= mit einer Funktion verbunden) werden Eingangswerte zu Ausgangswerten.*
- *Da in einer binären Mathematik alle Größen die zwei Werte 0 und 1 annehmen können (daher auch zweiwertige Logik genannt), gilt das auch für die dort definierten Variablen. Eingangsvariable bezeichnet man üblicherweise mit Großbuchstaben vom Anfang des Alphabets (A, B, C, ...), Ausgangsvariable mit Großbuchstaben vom*

Ende des Alphabets (U, V, W, …, Z)

- *Elementare Verknüpfungen haben die Namen UND, ODER, NICHT bzw. AND, OR, NOT (Skript 2.1).*
- *Die Funktion NICHT ist eine **unäre** Funktion, sie wirkt nur auf einen zweiwertigen Operanden (Eingangswert) und gibt nur einen zweiwertigen Ausgangswert zurück. Fälle:*

  $A=0 \rightarrow Z=\overline{A}=1$

  $A=1 \rightarrow Z=\overline{A}=0$

- *UND und ODER sind Funktionen, die zwei oder mehr Operanden (A, B, C, ... als zweiwertige Eingangsgrößen) verknüpfen und **einen** zweiwertigen Ausgangswert Z zurückgeben*
- *Die Funktionen können als Wahrheitstabellen dargestellt werden, da es im Gegensatz zu den kontinuierlichen Funktionen wie $f(x) = x^2$ oder $f(x) = \sin(x)$ nur endlich viele Kombinationen der Eingangswerte gibt.*
- *UND-Funktion:* $Z=A \wedge B$

B	A	Z
0	0	0
0	1	0
1	0	0
1	1	1

- *ODER-Funktion:* $Z=A \vee B$

B	A	Z
0	0	0
0	1	1
1	0	1
1	1	1

- *UND und ODER können auch mehr als nur zwei Operanden verknüpfen, geben aber immer nur einen Ausgangswert zurück:*

  $Z=A \wedge B \wedge C \wedge D \wedge \ldots$ *(auch **Minterm** genannt, siehe weiter unten)*

  $Z=A \vee B \vee C \vee D \vee \ldots$ *(auch **Maxterm** genannt, siehe weiter unten)*

(Skript 2.2)

- *Zwei zweiwertige Operanden können in $2^2 = 4$ Permutationen auftreten. Jede Permutation erzeugt einen zweiwertigen Ausgangswert. Ordnet man den jeweils 4 gleichen Eingangspermutationen nacheinander alle Ausgangspermutationen zu, so erhält man $2^4 = 16$ verschiedene, von denen jede einzelne eine Funktion darstellt. Eine systematische Reihenfolge, welche die Ausgangspermutationen als aufsteigende bi-*

näre Zahlen (Spalte Z) interpretiert, ist

B	A	Z
0	0	0
0	1	0
1	0	0
1	1	0

$Z = 0$ (Nullfunktion)

B	A	Z
0	0	0
0	1	0
1	0	0
1	1	1

$Z = A \wedge B$ (UND-Funktion)

B	A	Z
0	0	0
0	1	0
1	0	1
1	1	1

$Z = B$

B	A	Z
0	0	0
0	1	1
1	0	0
1	1	0

$Z = A \wedge \overline{B}$

B	A	Z
0	0	0
0	1	1
1	0	0
1	1	1

$Z = A$

B	A	Z
0	0	0
0	1	1
1	0	1
1	1	0

$Z = A\overline{B} \vee \overline{A}B$ (XOR-Funktion)

usw.

- *Es gibt also außer UND und ODER noch 14 weitere Funktionen, die sich allerdings aus UND, ODER und NICHT zusammensetzen lassen. Nicht alle haben eine praktische Bedeutung, gebraucht werden aber die Folgenden. Dabei sind die Werte von Z den Wertekombinationen von B und A in der Reihenfolge 00, 01, 10, 11 wie bei UND oder ODER zugeordnet:*
  - *NAND* *Z: 1 1 1 0*
  - *NOR* *Z: 0 0 0 1*
  - *ÄQUIVALENZ* *Z: 1 0 0 1*
  - *ANTIVALENZ (XOR = EXCLUSIVE ODER)* *Z: 0 1 1 0*
  - *INHIBITION* *Z: 0 0 1 0*
  - *IMPLIKATION* *Z: 1 0 1 1*
- *Diese Funktionen werden von der Industrie bevorzugt als Logik-Bausteine angeboten.*

Fragen: Wie sehen die restlichen 8 Funktionen aus? (Sie haben nicht alle eigene Namen).

Welche Funktionen werden als käufliche Bausteine (= Gatter) angeboten? (Kataloge „wälzen“)

Welche Angaben sind in den Produktblättern enthalten?

In welchem Rahmen bewegen sich die Preise?

Wie wirkt die INHIBITION und die IMPLIKATION?

Warum kann die ANTIVALENZ als Modulo 2 -Addierer eingesetzt werden?

Aufgabe: Bauen Sie nur unter Verwendung von UND, ODER und NICHT diejenige Funktion auf, die die Ausgangswerte Z: 0 1 0 0 ausgibt.

Veranstaltung am 31.03.2010

(Skript 2.3)

Da die Verknüpfungen der Digitaltechnik durch spezielle elektrische Schaltungen (Gatter) realisiert werden, lassen sich bei bestimmten Ausführungstechnologien UND- und ODER-Verknüpfungen durch parallel geschaltete Ausgangssignale erzeugen. Man spricht dann von Wired-Verküpfungen = verdrahtete Verknüpfungen. Zwei Fälle:

- *Geringer Widerstand zur Speisespannung (z. B. 5 Volt) bei H-Pegel des Gatteraus-*

gangs: Wired OR

- *Merkhilfe: Bei Wired-OR „gewinnt“ der H-führende Ausgang in der Parallelschaltung. (Dank an den Kommilitonen für diese nette Idee!)*
- *Geringer Widerstand zur Masse (z. B. 0 Volt) bei L-Pegel des Gatterausgangs: Wired AND*
- *Merkhilfe: Bei Wired-AND „gewinnt“ der L-führende Ausgang in der Parallelschaltung (Dank, s. o.).*

Achtung: Ob eine Gatter-Familie für Wired-Verknüpfungen geeignet ist, muss man den Datenblättern entnehmen!

Aufgabe: Erklären Sie die elektrische Wirkungsweise der Wired-Verknüpfungen.

3 Schaltalgebra (zum Skript Kapitel 3)

Die zuvor angegebenen Erläuterungen sind im Wesentlichen Definitionen für Funktionen, die einfach zur Kenntnis genommen werden müssen. Sie bedürfen keiner weiteren Begründung. Ihre Berechtigung beziehen sie aus der praktischen Anwendbarkeit. Theoretisch mögliche, aber praktisch nicht verwendbare Funktionen, wie z. B. die Nullfunktion Z = 0 0 0 0, treten in den weiteren Betrachtungen nicht mehr auf.

Für die Analyse (= Aufbau einer gegebenen Schaltung aus einer Kombination von Grundfunktionen) und Synthese (= Aufbau einer Schaltung zu einer vorgegebenen Funktionstabelle aus Grundfunktionen) gibt es dazu einige Grundgesetze und Regeln, welche die Arbeit vereinfachen können und im Gebiet der Schaltalgebra (von C. E. Shannon aus der Bool'schen Algebra entwickelt) zusammen gefasst sind.

Zunächst unterscheidet man zwischen

- *Konstanten (logische Werte „0“ und „1“),*
- *Schaltvariablen (= Größen, welche die Werte „0“ und „1“ annehmen können). Sie werden oft mit den großen Anfangsbuchstaben A, B, C, … des Alphabets bezeichnet, wenn sie Eingangsgrößen darstellen, mit großen Endbuchstaben Z, X, Y, W, V, …, bei Ausgangsgrößen. Dies ist aber willkürlich.*
- *Schaltfunktionen, die sich aus beliebigen logischen Verknüpfungen von Eingsvariablen zu Ausgangsvariablen zusammensetzen, z. B.* $Z = A\overline{B} \vee \overline{A}B$.

Für Konstanten gelten die **Postulate** der Schaltalgebra.

Die **Theoreme** der Schaltalgebra sind Rechenregeln für die Verknüpfung

- *einer Variablen mit einer Konstanten, z. B.* $Z = A \wedge 1 = A$
- *einer Variablen mit sich selbst, z. B.* $Z = A \vee A = A$
- *einer Variablen mit ihrer Negation, z. B.* $Z = A \vee \overline{A} = 1$.

Gesetze der Schaltalgebra:

- ***Kommutativgesetz*** → *bei* ***gleichen*** *Operationen kann die Reihenfolge der Variablen beliebig vertauscht werden, z. B.*

  $Z = A \wedge B \wedge C = B \wedge A \wedge C = C \wedge B \wedge A$

  $Z = A \vee B \,o\, C = B \vee A \vee C = C \vee B \vee A$

- ***Assoziativgesetz*** → *bei* ***gleichen*** *Operationen kann die Reihenfolge der Operationen belie-*

big getauscht werden, z. B.

$$Z = A \wedge (B \wedge C) = (A \wedge B) \wedge C$$

$$Z = A \vee (B \vee C) = (A \vee B) \vee C$$

Die geklammerten Operationen werden dabei, wie in der „gewöhnlichen" Algebra, zuerst ausgeführt, siehe Vorrangregeln weiter unten.

Die beiden **Distributivgesetze** eignen sich besonders zur Veränderung und Vereinfachung schaltalgebraischer Ausdrücke:

- **Konjunktives Distributivgesetz** → die Konjunktion einer Variablen A mit der Disjunktion weiterer Variabler B, C, D, … lässt sich als Disjunktion aller paarweisen Konjunktionen mit A darstellen, z.B.

  $$Z = A \wedge (B \vee C \vee D) = (A \wedge B) \vee (A \wedge C) \vee (A \wedge D)$$

  Hinweis: Für eine bessere Übersichtlichkeit kann man das **UND**-Verknüpfungszeichen – und dann auch die Klammern - weglassen:

  $$Z = A(B \vee C \vee D) = AB \vee AC \vee AD$$

- **Disjunktives Distributivgesetz** → die Disjunktion einer Variablen A mit der Konjunktion weiterer Variabler B, C, D, … lässt sich als Konjunktion aller paarweisen Disjunktionen mit A darstellen, z. B.

  $$Z = A \vee (B \wedge C \wedge D) = (A \vee B) \wedge (A \vee C) \wedge (A \vee D)$$

  Als Umkehrung gilt die **Merkregel**: Ein gemeinsamer Operand kann ausgeklammert werden, z. B.

  $$Z = (F \vee A) \wedge (F \vee B) \wedge (F \vee C) = F \wedge (A \vee B \vee C)$$

- **Shannonsches Gesetz** → Die Negation eines logischen Ausdrucks ist identisch der Negation aller Variablen und Austauschen der Operationszeichen , z. B.

  $$Z = A \wedge B \wedge C \;\rightarrow\; \overline{Z} = \overline{A} \vee \overline{B} \vee \overline{C} \;,$$

  dabei Vorrang beachten (siehe weiter unten):

  $$Z = A \wedge B \wedge C \vee D = ABC \vee D \;\rightarrow\; \overline{Z} = \overline{A \wedge B \wedge C} \wedge \overline{D} = \overline{ABC} \wedge \overline{D}$$

- Die 2 **De Morgansche Gesetze** als Sonderfälle des Shannon-Gesetzes:

  $$Z = \overline{A \wedge B} = \overline{A} \vee \overline{B}$$

  $$Z = \overline{A \vee B} = \overline{A} \wedge \overline{B}$$

Veranstaltung am 07.04.2010

Vorrangregeln

Wie bereits zuvor beachtet, gilt für die Bearbeitungsreihenfolge logischer Funktionen die Regel

NICHT vor **UND** vor **ODER**

Dabei hilft es sehr, Klammern zu verwenden und **UND**-Verknüpfungen durch Weglassen des **UND**-Symbols optisch von ODER-Verknüpfungen abzusetzen, z. B.

$$Z = D \wedge E \wedge F \vee A \wedge C \vee B \wedge F = (D \wedge E \wedge F) \vee (A \wedge C) \vee (B \wedge F)$$

oder

$$Z = DEF \vee AC \vee BF \quad .$$

Achtung: Die Negation von Z gilt für die ganze rechte Seite und kann unter Verwendung des Shannon-Gesetzes stufenweise „nach innen“ gezogen werden:

$$\overline{Z} = \overline{DEF \vee AC \vee BF} = \overline{DEF} \wedge \overline{AC} \wedge \overline{BF} = \overline{DEF} \;\; \overline{AC} \;\; \overline{BF}$$

oder

$$\overline{Z} = (\overline{D} \vee \overline{E} \vee \overline{F}) \;\; (\overline{A} \vee \overline{C}) \;\; (\overline{B} \vee \overline{F}) \quad .$$

Absorptionsgesetze (auch Verschmelzungsgesetze genannt), (Skript 3.4)

Bei bestimmten Verknüpfungs-Konstellationen fällt ein Operand heraus. Zum Nachweis für

$$A \wedge (A \vee B) = A \quad (I)$$

verwendet man das Theorem

$$A \vee 0 = A$$

mit dem sich eine Erweiterungen bewirken lässt:

$$A \wedge (A \vee B) = (A \vee 0) \wedge (A \vee B) = A \vee (0 \wedge B) = A \vee 0 = A \quad .$$

Mit Hilfe des Distributivgesetzes:

$$(A \vee C) \wedge (A \vee B) = A \vee (C \wedge B) \quad .$$

kann A ausgeklammert werden und man erhält

$$A \wedge (A \vee B) = (A \vee 0) \wedge (A \vee B) = A \vee (0 \wedge B) = A \vee 0 = A \quad .$$

Im komplementären Fall

$$A \vee (A \wedge B) = A$$

negiert man zunächst doppelt

$$A \vee (A \wedge B) = \overline{\overline{A \vee (A \wedge B)}} \quad ,$$

wendet die De Morgan-Gesetze an und erhält die negierte Form von (I) mit negierten Operanden.

$$A \vee (A \wedge B) = \overline{\overline{A \vee (A \wedge B)}} = \overline{\overline{A} \wedge \overline{(A \wedge B)}} = \overline{\overline{A} \wedge (\overline{A} \vee \overline{B})} \quad .$$

Dann ist wegen

$$\overline{A} \vee 0 = \overline{A}$$

und des umgekehrten Distributivgesetzes

$$(\overline{A} \vee 0) \wedge (\overline{A} \vee \overline{B}) = \overline{A} \vee (0 \wedge \overline{B}) = \overline{A} \vee 0 = \overline{A}$$

auch

$$A \vee (A \wedge B) = \overline{\overline{A} \wedge (\overline{A} \vee \overline{B})} = \overline{\overline{A}} = A \quad .$$

Aufgabe: Dieses Ergebnis erhält man auch, wenn $A \wedge A = A$ genutzt wird. Weisen Sie es nach.

Außerdem gilt

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

Aufgabe: Weisen Sie beide Äquivalenzen unter Verwendung der Theoreme

$$A \vee \overline{A} = 1 \quad \text{bzw.} \quad A \wedge \overline{A} = 0 \quad \text{nach.}$$

Im weiteren Verlauf der Veranstaltung wurden einige Grundfunktionen des ALTERA-Programms MAX+plus II ausprobiert.

Zu erwähnen sind noch die beiden **Entwicklungssätze**:

$$Z(A,B,C,D,....) = \left[A \wedge Z(1,B,C,D,...)\right] \vee \left[\overline{A} \wedge Z(0,B,C,D,...)\right]$$

$$Z(A,B,C,D,....) = \left[A \vee Z(0,B,C,D,...)\right] \wedge \left[\overline{A} \vee Z(1,B,C,D,...)\right]$$

Die Eingangsvariablen können dabei links und rechts auch in negierter Form auftreten.

Aufgabe: Zeigen Sie für

$$Z(A,B,C) = (A \vee B) \wedge C \wedge D \quad ,$$

dass die beiden Sätze für diesen Fall erfüllt sind.

Veranstaltung am 14.04.2010

Aufbau von Logikschaltungen aus NOR- bzw. aus NAND-Gattern (Skript 3.4)

Wendet man die De Morganschen Gesetze an, so lassen sich die drei logischen Grundoperationen **NICHT, UND**, **ODER** ausschließlich über **NOR**–Funktionen (= **NICHT ODER**), bzw. ausschließlich über **NAND**-Funktionen (=**NICHT UND**) darstellen. Es gilt z. B.

$$A \wedge B = \overline{\overline{A \wedge B}} = \overline{\overline{A} \vee \overline{B}} \quad .$$

Hier wird die **NICHT ODER**-Funktion auf die negierten Operanden A, B angewendet. Die Negation der Operanden kann ebenfalls mit einer **NOR**-Funktion erfolgen:

$$A = \overline{\overline{A}} = \overline{\overline{A} \wedge \overline{A}} = \overline{A \vee A}$$

Man schaltet dann den Operanden A auf beide Eingänge. Damit wird die **UND**-Funktion aus drei **NOR**-Funktionen zusammengesetzt:

$$A \wedge B = \overline{\overline{A \wedge B}} = \overline{\overline{A} \vee \overline{B}} = \overline{\overline{A \vee A} \vee \overline{B \vee B}} \quad .$$

Die ODER-Funktion ist die negierte NOR-Funktion:

$$A \vee B = \overline{\overline{A \vee B}} \quad .$$

Der Ausgang der **NOR**-Funktion wird parallel auf die beiden Eingänge einer weiteren **NOR**-Funktion geschaltet, so dass man hier zwei **NOR**-Gatter benötigt.

Frage: Aus welchem Grund kann die Darstellung einer schaltalgebraischen Funktion durch **NOR**-Gatter trotz des Mehrbedarfes für die Realisierung einer Schaltung interessant sein?

Aufgabe: Weisen Sie nach, dass **NICHT**, **UND** sowie **ODER** auch ausschließlich durch **NAND**-Gatter darstellbar sind (aber nicht im Skript nachschauen ...).

Schaltungsanalyse und Schaltungssynthese

Ein wesentlicher Zweck der Digitaltechnik ist die Aufstellung schaltalgebraischer Ausdrücke für die Realisierung technisch-physikalischer Funktionsabläufe (z. B. Fertigungsprozess eines Werkstückes auf einer Drehbank, Waschmaschinen-Steuerung). Auch wenn diese zusätzlich oft noch mit kontinuierlichen Prozessen vermischt sind, müssen die diskreten digitalen Teile selbstverständlich auch für sich allein korrekt arbeiten.

Man muss also zunächst den gewünschten Ablauf exakt beschreiben, ihn widerspruchsfrei gestalten und dann in weiteren Schritten in eine Folge von Funktionen umsetzen, welche durch eine elektronische Schaltung abgebildet werden kann. Dies ist die Aufgabe der Schaltungssynthese, welche im Kapitel 5 des Skripts behandelt wird. Manchmal besteht die Aufgabe aber auch darin, zu einer gegebenen Schaltung die zugrunde liegende logische Funktion zu finden.

4 **Schaltungsanalyse** (zum Skript Kapitel 4)

Schaltfunktionen können in drei gleichwertigen Formen dargestellt werden:

- Funktionsplan mit Funktionssysmbolen (= Digitalschaltung)
- Wahrheitstabelle
- algebraischer Ausdruck (= Funktionsgleichung).

Jede Form lässt sich in jede andere überführen. Am einfachsten es, eine Digitalschaltung in eine Funktionsgleichung abzubilden und umgekehrt, siehe Vorlesungs-Skript Kapitel 4.1. Aus der Funktionsgleichung erhält man die Wahrheitstabelle.

Wenn für die zu analysierende (= die logischen Bestandteile und Zusammenhänge beschreibende) Funktion nur eine elektronische Schaltung (Platine) vorliegt, muss man zunächst durch Messung der elektrischen Ausgangssignale (= Logik-Pegel) für **alle** 2^n Permutationen der n Eingangssignale die Wahrheitstabelle bestimmen. Daraus kann dann mit den Hilfsmitteln der Schaltungssynthese die Funktionsgleichung und daraus wiederum die Digitalschaltung ermittelt werden.

5 **Schaltungssynthese** (zum Skript Kapitel 5)

Wenn man für Aufgabenstellung eine aufwandsarme Digitalschaltung sucht. Für das, was hier „Aufwand“ bedeutet, können unterschiedliche Kriterien gemeint sein, z. B.

- möglichst wenig Gatter
- möglichst wenig verschiedene Gatter (nur NOR oder nur NAND)
- möglichst kostengünstige Großserienfertigung der bestückten Platine
- möglichst hohe Störsicherheit
- möglichst geringer Leistungsverbrauch
- möglichst schnelle Verarbeitungsgeschwindigkeit

  usw.

Man ahnt, dass die Aufgabenstellung nicht nur die technische Funktion, sondern auch die weiteren Kriterien enthalten muss. Die Darstellung der technischen Funktion ist aber die Grundlage für alles Weitere, weshalb zunächst das dafür geeignete Verfahren behandelt wird. Wie im Skript Kapitel 5.1 beschrieben, muss dazu im ersten Schritt eine genaue Beschreibung der Funktion angefertigt werden (Punkt 1). Hierfür gibt es keine wirkliche Systematik, weshalb dies sehr sorgfältig durchzuführen ist. Nach dem Festlegen der Eingangs- und Ausgangsvariablen sowie der Zuweisung der Bedeutung ihrer logischen Zustände (Punkt 2) kann – nun systematisch - die Wahrheitstabelle aufgestellt werden (Punkt 3). Diese dient der Bestimmung der schaltalgebraischenFunktion (Punkt 4),

die im Allgemeinen noch vereinfacht oder umgeformt werden muss (Punkt 5). Hier fließen dann auch die Anforderungen durch die genannten Kriterien mit ein. Erst danach geht es an die technisch- physikalische Realisierung der Schaltung (Punkt 6).

Im Folgenden werden die Begriffe und Verfahren behandelt, die zu den Punkten 4 und 5 gehören.

Aus der Wahrheitstabelle muss zunächst die algebraische Funktionsgleichung ermittelt werden. Die unmittelbare Umsetzung durch Probieren ist zwar möglich, aber schon mit wenigen Eingangsgrößen (3) umständlich und vor allem fehleranfällig. Wesentlich eleganter und systematisch erhält man das Ergebnis, wenn man die Tabelle in eine der Normalformen bringt. Solche Normalformen haben einen klaren Aufbau.

Die beiden Hauptformen sind

- die **disjunktive Normalform** (DNF)
- die **konjunktive Normalform** (KNF).

Bei der DNF wird jede Zeile der Wahrheitstabelle als Konjunktion (= UND-Verknüpfung) der Eingangsvariablen aufgefasst, wobei jede Eingangsvariable in jeder Zeile vorkommt. Die Ausgangsvariable hat nur dann den Wert 1, wenn eine der Zeilen den Wert 1 aufweist. Die Ausgangsvariable ist dann die Disjunktion (= ODER-Verknüpfung) aller Zeilen, die eine 1 ergeben, daher disjunktive Normalform.

Beispiel:

Fall	A	B	C	Z
1	0	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	1	1	0
5	1	0	0	1
6	1	0	1	1
7	1	1	0	1
8	1	1	1	0

DNF: $$Z = A\overline{B}\,\overline{C} \vee A\overline{B}C \vee AB\overline{C}$$

Die drei konjunktiv verknüpften Terme mit den Eingangsvariablen sind die **Minterme**. Jeder Minterm enthält immer alle negierten oder nicht negierten Eingangsvariablen in beliebiger Permutation.

Als Vereinfachung für Z bieten sich hier z. B. die ersten beiden Terme an. Bei Anwendung des Distributivgesetzes lässt sich der gemeinsame Faktor $A\overline{B}$ ausklammern:

$$A\overline{B}\,\overline{C} \vee A\overline{B}C = A\overline{B} \wedge (\overline{C} \vee C) = A\overline{B} \wedge 1 = A\overline{B} \quad .$$

Im Übrigen kann man jede beliebige Wahrheitstabelle durch entsprechende Erweiterungen für die disjunktive Normalform einrichten.

Bei der KNF wird jede Zeile der Wahrheitstabelle als Disjunktion (= ODER-Verknüpfung) der Eingangsvariablen aufgefasst, wobei auch hier jede Eingangsvariable in jeder Zeile vorkommt. Die Ausgangsvariable ist die Konjunktion (= UND-Verknüpfung) aller Zeilen, die eine 0 ergeben, daher

konjunktive Normalform.

Beispiel:

Fall	A	B	C	Z
1	0	0	0	1
2	0	0	1	0
3	0	1	0	1
4	0	1	1	0
5	1	0	0	1
6	1	0	1	1
7	1	1	0	1
8	1	1	1	0

KNF: $$Z=(A\vee B\vee\overline{C})\wedge(A\vee\overline{B}\vee\overline{C})\wedge(\overline{A}\vee\overline{B}\vee\overline{C})$$

Die 3 disjunktiv verknüpften Terme mit den Eingangsgrößen sind die **Maxterme**. Jeder Maxterm enthält immer alle negierten oder nicht negierten Eingangsvariablen in beliebiger Permutation.

Hier kann man für eine Vereinfachung z. B. den Faktor $\overline{B}\vee\overline{C}$ aus dem zweiten und dritten Term ausklammern:

$$(A\vee\overline{B}\vee\overline{C})\wedge(\overline{A}\vee\overline{B}\vee\overline{C})=(A\,\overline{A})\vee(\overline{B}\vee\overline{C})=0\vee(\overline{B}\vee\overline{C})=\overline{B}\vee\overline{C}\quad .$$

Wiederum lässt sich jede Wahrheitstabelle durch Erweiterungen für die konjunktive Normalform einrichten.

DNF und KNF sind gleichwertig, jedoch ist das Aufstellen der DNF weniger aufwändig, wenn die Ausgangsfunktion weniger 1 als 0 enthält und umgekehrt. Außerdem wirkt sich die Form auch auf die Arbeit mit den nun zu betrachtenden Vereinfachungsverfahren aus.

Die DNF (oder KNF) kann im Allgemeinen weiter vereinfacht werden, Hierzu eignen sich

- die Anwendung der **Gesetze der Schaltalgebra**. Sie sind meistens nur für 2 bis 3 Eingangsvariable praktikabel.
- das grafische Verfahren nach Karnaugh-Veitch = **KV-Diagramm**. Es ist bis zu 4 Variablen sehr übersichtlich, kann auch für 5 und 6 Variable verwendet werden, wird dann aber schwerer zu überblicken.
- das algorithmische Verfahren nach **Quine-McClusky**. Es ist für eine beliebige Anzahl von Eingangsvariablen geeignet - im Vergleich mit dem KV-Diagramm allerdings eher ab 5 Eingangsvariablen - und erfordert dann die Bearbeitung mithilfe eines Rechenprogramms.

Die Anwendung der Schaltalgebra-Gesetze wurde und wird im Rahmen diese Veranstaltung weiterhin insbesondere für einfache Teilfunktionen komplexer Ausdrücken brauchbar bleiben und muss hier nicht gesondert erläutert werden.

Das KV-Diagramm ist ein effektiver Weg, um „per Hand“ logische Funktionen aufwandsarm darzustellen. Es verwendet die aus der Wahrheitstabelle immer ableitbare DNF und bestimmt systematisch unter Verwendung der **Distributiv-Gesetze** in den meisten Fällen die aufwandsärmste Version. Oft gibt es zu einer solchen Version jedoch andere, gleichwertige Varianten. Das KV-Diagramm kann man auch als Distributiv-Gesetz-Rechner ansehen.

Das KV-Diagramm ist eine andere Form der Wahrheitstabelle. Es besteht aus einer rechteckigen Anordnung von Kästchen, in die alle möglichen Minterme der Eingangsvariablen eingetragen werden. Bei 2 Variablen sind es 4, bei 3 sind es 8 bei 4 dann 16 usw. Allerdings ist eine ebene Anordnung für mehr als 4 Variable nicht mehr in einfacher Form möglich, so dass man hier auf das Anbauen weiterer „Variablen-Schichten“ in die Tiefe gehen muss. Dann eignet sie sich auch für die die Behandlung von 5 und 6 Variablen.

Beispiel für 4 Eingangsgrößen ABCD:

	A	A	$\overline{A}$	$\overline{A}$	
B					$\overline{D}$
B					D
$\overline{B}$					D
$\overline{B}$					$\overline{D}$
	$\overline{C}$	C	C	$\overline{C}$	

Jedem Kästchen wird einer der 8 möglichen Minterme als Kreuzungsfeld der in den Spalten- und Zeilenköpfen stehenden Variablen zugeordnet. Z. B. enthält die Funktion

$$Z=(A\,B\,\overline{C}\,\overline{D})\vee(A\,B\,\overline{C}\,D)\vee(A\,\overline{B}\,\overline{C}\,\overline{D})\vee(\overline{A}\,B\,C\,D)$$

als DNF 4 Minterme (= 4 NICHT-Gatter, 4 Vierfach-AND-Gatter, 1 Vierfach-OR-Gatter), die sich im KV-Diagramm durch den Eintrag einer 1 markieren lassen:

	A	A	$\overline{A}$	$\overline{A}$	
B	1				$\overline{D}$
B	1		1		D
$\overline{B}$					D
$\overline{B}$	1				$\overline{D}$
	$\overline{C}$	C	C	$\overline{C}$	

Der Minterm $(\overline{A}\,B\,C\,D)$ steht isoliert und bleibt unverändert:

	A	A	$\overline{A}$	$\overline{A}$	
B	1				$\overline{D}$
B	1		1		D
$\overline{B}$					D
$\overline{B}$	1				$\overline{D}$
	$\overline{C}$	C	C	$\overline{C}$	

Die beiden benachbarten Minterme $(A\,B\,\overline{C}\,\overline{D})$ und $(A\,B\,\overline{C}\,D)$ enthalten den gemeinsamen Term $(A\,B\,\overline{C})$:

	A	A	$\overline{A}$	$\overline{A}$	
B	1				$\overline{D}$
B	1		1		D
$\overline{B}$					D
$\overline{B}$	1				$\overline{D}$
	$\overline{C}$	C	C	$\overline{C}$	

Dieser steht wegen der DNF in einer ODER-Vernüpfung und kann unter Anwendung des disjunktiven Distributiv-Gesetzes ausgeklammert werden. Man erhält

$$(A\,B\,\overline{C}\,\overline{D})\vee(A\,B\,\overline{C}\,D)=A\,B\,\overline{C}\wedge(\overline{D}\vee D)=A\,B\,\overline{C}\quad .$$

Ebenso enthalten die beiden Minterme $(A\,B\,\overline{C}\,\overline{D})$ und $(A\,\overline{B}\,\overline{C}\,\overline{D})$ den gemeinsamen Faktor $(A\,\overline{C}\,\overline{D})$, der nach ebenfalls Ausklammern übrig bleibt:

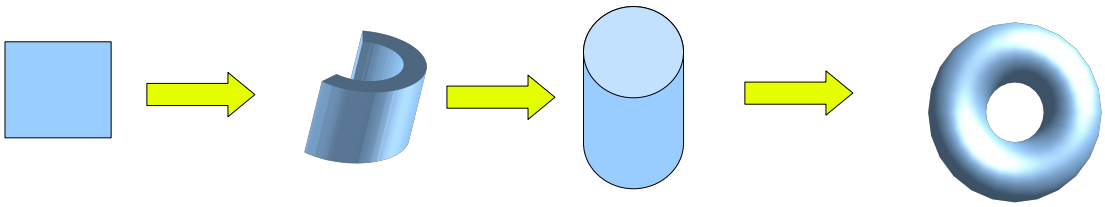
	A	A	$\overline{A}$	$\overline{A}$	
B	1				$\overline{D}$
B	1		1		D
$\overline{B}$					D
$\overline{B}$	1				$\overline{D}$
	$\overline{C}$	C	C	$\overline{C}$	

Die Funktion Z lässt sich daher auch in der aufwandsärmeren Form

$$Z=(A\,B\,\overline{C})\vee(A\,\overline{C}\,\overline{D})\vee(\overline{A}\,B\,C\,D)$$

mit 3 NICHT-Gattern, 2 Dreifach-AND-Gattern, 1 Vierfach-AND-Gatter und 1 Dreifach-ODER-Gatter schreiben (im Vergleich zur DNF: 4 NICHT-Gatter, 4 Vierfach-AND-Gatter, 1 Vierfach-OR-Gatter) .

Hinweis: Die beiden Minterme $(A\,B\,\overline{C}\,\overline{D})$ und $(A\,\overline{B}\,\overline{C}\,\overline{D})$ erscheinen als nicht benachbart, jedoch sind oberer und unterer sowie linker und rechter Rand topologisch verbunden. Rollt man in Gedanken das Rechteck zunächst horizontal zu einem Zylinder

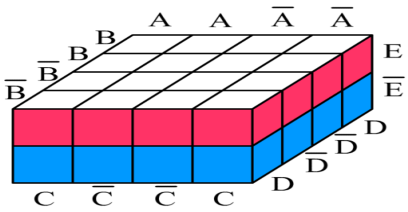


und „klebt“ die Ränder aneinander, so sieht man, dass linker und rechter Rand benachbart sind. Nun biegt man den Zylinder noch so, dass sich die Stirnflächen verbinden und ein Torus (= Hula-Hoop-Reifen) entsteht. Dann sind auch oberer und unterer Rand Nachbarn.

Beispiel für 5 Eingangsgrößen A, B, C, D, E:

Hier wird dem gesamten ebenen Rechteck für A, B, C und D als erste Schicht noch die Variable E

zugewiesen. Die negierte Variable $\overline{E}$ erhält als zweite Schicht in die Tiefe eine vollständige Kopie der ersten Schicht:



Davon abgesehen wird mit diesem Diagramm so gearbeitet wie mit dem für 4 Variable.

Veranstaltung am 21.04.2010

Weitere Vereinfachungen im Fall von „don't cares“:

Bei manchen Aufgabenstellungen treten bestimmte Permutationen der Eingangsgrößen nicht auf und erscheinen damit auch nicht in der Wahrheitstabelle. Ein **Beispiel**: In einem PKW leuchtet gewöhnlich eine Warnlampe auf, wenn der Motor läuft, aber der Sicherheitsgurt nicht angelegt wurde. Damit überprüft werden kann, ob die Lampe überhaupt noch funktioniert, soll sie – bei Ausstattung mit Otto-Motor - auch aufleuchten, wenn die Zündung eingeschaltet, aber der Motor noch nicht gestartet ist. Für die Zuordnung der drei Eingangsvariablen

A → Zündung an = 1/ aus =0

B → Motor an = 1/ aus = 0

C → Gurt angelegt = 1/ nicht angelegt = 0

und der Ausgangsvariablen

Z → Warnlampe leuchtet = 1/ leuchtet nicht = 0

stellt man zunächst die Wahrheitstabelle auf. Dabei können die Fälle 3 und 4 nicht auftreten, da der Motor bei ausgeschalteter Zündung aus physikalischen Gründen nicht läuft. Sie erhalten daher die Kennzeichnung mit „X“.

Fall	A	B	C	Z
1	0	0	0	0
2	0	0	1	0
3	0	1	0	X
4	0	1	1	X
5	1	0	0	1
6	1	0	1	1
7	1	1	0	1
8	1	1	1	0

Für die Aufstellung einer aufwandsarmen Ausgangsfunktion Z können sie in der DNF bei Bedarf als weitere Minterme aufgenommen werden. Im KV-Diagramm eröffnet das die Möglichkeit zusätz-

licher Vereinfachungen, die allerdings nicht zwangsweise damit verbunden sind. Außerdem muss sorgfältig sichergestellt sein, dass die entsprechenden Permutationen wirklich nicht auftreten, da es sonst zu Fehlfunktionen kommen würde.

Die DNF für das obige Beispiel besteht aus 3 Mintermen:

$$Z = (A\,B\,\overline{C}) \vee (A\,\overline{B}\,\overline{C}) \vee (A\,\overline{B}\,C)$$

Das KV-Diagramm hat die Form:

	A	A	$\overline{A}$	$\overline{A}$
B	1			
$\overline{B}$	1	1		
	$\overline{C}$	C	C	$\overline{C}$

Man kann zwei Zusammenfassungen durchführen:

	A	A	$\overline{A}$	$\overline{A}$
B	1			
$\overline{B}$	1	1		
	$\overline{C}$	C	C	$\overline{C}$

	A	A	$\overline{A}$	$\overline{A}$
B	1			
$\overline{B}$	1	1		
	$\overline{C}$	C	C	$\overline{C}$

Die Ausgangsfunktion ist dann

$$Z = (A\,\overline{C}) \vee (A\,\overline{B}) = A\,(\overline{B} \vee \overline{C}) \quad .$$

Bei Verwendung der „don't cares“ als Minterme erhält man

	A	A	$\overline{A}$	$\overline{A}$
B	1		X	X
$\overline{B}$	1	1		
	$\overline{C}$	C	C	$\overline{C}$

Damit könnten maximal zwei weitere Vereinfachungen möglich sein, was in diesem Fall aber nicht zutrifft, da sich die Funktionsgleichung nur aufbläht, wie es sich aus dem KV-Diagramm ergibt:

	A	A	$\overline{A}$	$\overline{A}$
B	1		X	X
$\overline{B}$	1	1		
	$\overline{C}$	C	C	$\overline{C}$

	A	A	$\overline{A}$	$\overline{A}$
B	1		X	X
$\overline{B}$	1	1		
	$\overline{C}$	C	C	$\overline{C}$

$$Z=(A\overline{C})\vee(A\overline{B})\vee(\overline{A}B)\vee(B\overline{C})=(A\overline{C})\vee(A\overline{B})\vee\overline{A}\vee 0=A(\overline{C}\vee\overline{B})\vee\overline{A}$$

Ein weiteres Beispiel ist die Umwandlungsfunktion von Binärzahlen in die 10 Dezimalziffern 0, 1, 2, …., 8, 9. Hierfür benötigt man von den 16 möglichen Binärzahlen mit 4 Stellen (= Tetraden) nur 10 Permutationen, die restlichen 6 werden als Pseudo-Tetraden bezeichnet und treten nicht auf, falls die Zahlen von einer reinen Dezimalziffern-Quelle geliefert worden waren. Die Wahrheitstabelle:

Fall	A	B	C	D	0	1	2	3	...
0	0	0	0	0	1	0	0	0	
1	0	0	0	1	0	1	0	0	
2	0	0	1	0	0	0	1	0	
3	0	0	1	1	0	0	0	1	
4	0	1	0	0	0	0	0	0	
5	0	1	0	1	0	0	0	0	
6	0	1	1	0	0	0	0	0	
7	0	1	1	1	0	0	0	0	
8	1	0	0	0	0	0	0	0	
9	1	0	0	1	0	0	0	0	
	1	0	1	0	X	X	X	X	
	1	0	1	1	X	X	X	X	
	1	1	0	0	X	X	X	X	
	1	1	0	1	X	X	X	X	
	1	1	1	0	X	X	X	X	
	1	1	1	1	X	X	X	X	

In dieser Form ist jede Dezimalziffern durch genau einen Minterm bestimmt, die zugehörige Funktion liegt also bereits in der Minimalform vor. Anders sieht es aus, wenn eine 7-Segment-LCD-Anzeige angesteuert werden soll.

Für jeden der 7 Leuchtbalken gibt es eine eigene Ausgangsfunktion:

T (für das untere Mittelsegment)

U (für das mittlere Mittelsegment)

V (für das obere Mittelsegment)

W (für das untere Linkssegment)

X (für das obere Linkssegment)

Y (für das untere Rechtssegment),

Z (für das obere Rechtsegment).

Das obere Mittelsegment V wird dann z. B. für die Dezimalziffern 0, 2, 3, 5, 6, 7, 8 und 9 benötigt, das obere Linkssegment für 0, 4, 5, 6, 8, 9 usw.

Aufgabe: Bestimmen Sie für das obere Mittelsegment und das untere Rechtssegment jeweils die Minimalform der zugeordneten Ansteuerfunktion unter Berücksichtigung der „don't cares“.

Lösungshinweise (aber erst selber versuchen!): Die Wahrheitstabelle für das obere Mittelsegment mit der Funktionsgleichung V= V(A,B,C,D) ist:

Fall	A	B	C	D	V
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
	1	0	1	0	X
	1	0	1	1	X
	1	1	0	0	X
	1	1	0	1	X
	1	1	1	0	X
	1	1	1	1	X

Da in der Ausgangsfunktion V nur 2 Nullen erscheinen, ist die Verwendung der KNF günstiger. Die „don't cares“ können dann nach Bedarf durch 0 ersetzt werden. Das KV-Diagramm ist

	A	A	$\overline{A}$	$\overline{A}$	
B		0		X	$\overline{D}$
B				X	D
$\overline{B}$		0	X	X	D
$\overline{B}$			X	X	$\overline{D}$
	$\overline{C}$	C	C	$\overline{C}$	

Die Ausgangsvariable V in KNF:

$$V=(A\vee B\vee C\vee\overline{D})\wedge(A\vee\overline{B}\vee C\vee D)$$

Dieser Ausdruck lässt sich nicht weiter vereinfachen. Man sieht es am KV-Diagramm, da die beiden Nullen nicht benachbart sind. Die „don't cares“ in der rechten Spalte bringen keine Vorteile, da sie ebenfalls keine Nachbarn zu den Nullen darstellen. Nur das eine „don't care“ in der dritten Spalte, zweite Zeile, trägt zur Vereinfachung bei:

$$\begin{aligned}V&=(A\vee B\vee C\vee\overline{D})\wedge(A\vee\overline{B}\vee C\vee D)\wedge(\overline{A}\vee\overline{B}\vee C\vee D)\\&=(A\vee B\vee C\vee\overline{D})\wedge\left[(A\wedge\overline{A})\vee(\overline{B}\vee C\vee D)\right]\\&=(A\vee B\vee C\vee\overline{D})\wedge\left[0\vee(\overline{B}\vee C\vee D)\right]\\&=(A\vee B\vee C\vee\overline{D})\wedge(\overline{B}\vee C\vee D)\end{aligned}$$

Aufgabe: Bestimmen Sie die aufwandsärmste Funktion Z der 4 Eingangsgrößen A, B , C, D, die der folgenden Wahrheitstabelle genügt und die „don't cares“ einbezieht:

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

(Lösung: $Z=(\overline{A}B)\vee(A\overline{B}\overline{C})\vee(ACD)$ *)*

Das Verfahren von Quine-McClusky:

Für mehr als 4 Variable lässt sich das KV-Diagramm zur Erstellung einer aufwandsarmen Ausgangsfunktion zwar auch noch verwenden, es wird aber zusehends mühsamer. Die Ingenieur-Arbeit hat ihren Schwerpunkt im Begreifen und Steuern physikalisch-technischer Zusammenhänge, im Erarbeiten und Verbessern praktikabler, kostengünstiger Lösungen und in der Überprüfung von Ergebnissen. Hierzu sind fast immer umfangreiche Berechnungen erforderlich, deren eigene Durchführung den Ingenieur aber an der falschen Stelle unnötig ablenkt und belastet. Daher soll das Ermitteln günstiger schaltalgebraischer Funktionen so weit wie möglich einem vertrauenswürdigen Rechenprogramm überlassen werden. Allerdings setzt die Vorbereitung der eigentlichen Rechenläufe und die Überprüfung der Ergebnisse das detaillierte Verständnis der Verfahren unbedingt voraus. Dieses muss man sich auch unter Zuhilfenahme kleiner Beispiele erarbeiten.

Das algorithmische Verfahren von Quine-McClusky eignet sich besonders für die programmierte Durchführung, basiert aber auch nur auf der systematischen Verwendung des Distributivgesetzes zum Reduzieren der DNF (oder KNF).

Ein Beispiel mit 4 Eingangsgrößen A, B, C, D: Aus einer Wahrheitstabelle wurde die DNF

$$Z=\overline{A}\,\overline{B}C\overline{D}\vee\overline{A}B\overline{C}\,\overline{D}\vee\overline{A}B\overline{C}D\vee\overline{A}BC\overline{D}\vee\overline{A}BCD\vee A\overline{B}\,\overline{C}D\vee AB\overline{C}D$$

aufgestellt. Sie besteht also aus 7 Mintermen. Um systematisch diejenigen Paare zu finden, die bis auf eine Variable übereinstimmen (welche dann jeweils eliminiert werden kann), werden sie für die Handrechnung als Zeilenköpfe in eine Tabelle eingetragen:

Durchlauf 1								Durchlauf 2			Durchlauf 3
$\overline{A}\,\overline{B}C\overline{D}$	+							$\overline{A}C\overline{D}$ **= P1**			
$\overline{A}B\overline{C}\,\overline{D}$		+	+					$\overline{A}B\overline{C}$	+		$\overline{A}B$ **= P4**
$\overline{A}B\overline{C}D$		+		+	+			$\overline{A}B\overline{D}$		+	$\overline{A}B$
$\overline{A}BC\overline{D}$	+		+			+		$\overline{A}BD$		+	
$\overline{A}BCD$				+		+		$B\overline{C}D$ **= P2**			
$A\overline{B}\,\overline{C}D$							+	$\overline{A}BC$	+		
$AB\overline{C}D$					+		+	$A\overline{C}D$ **= P3**			

Dann vergleicht man jeden Minterm mit jedem anderen (Durchlauf 1), markiert die bereits berücksichtigten Paare (hier mit einem „+“) , die einen gemeinsamen Anteil aus 3 Variablen haben und schreibt sie rechts hin. In diesem Beispiel kann es zu jedem Minterm höchsten 3 weitere solche Paare geben.

Im Durchlauf 2 werden nun die reduzierten Paare verglichen, passende Paare markiert und rechts die gemeinsamen Anteile hingeschrieben. Es folgt der Durchlauf 3, der hier aber keine weitere Reduzierung gestattet. Damit sind die Durchläufe abgeschlossen. Die übrig gebliebenen Terme nennt man **Primimplikanden** (= Primterme), hier sind es P1, P2, P3 und P4. Da sie durch Reduktion aus der DNF entstanden, bildet ihre Disjunktion (= ODER-Verknüpfung) eine aufwandsärmere Version der Ausgangsvariablen Z:

$$Z = \overline{A}C\overline{D} \vee B\overline{C}D \vee A\overline{C}D \vee \overline{A}B \quad .$$

P4 tritt doppelt auf, muss aber wegen

$$Z = \overline{A}C\overline{D} \vee B\overline{C}D \vee A\overline{C}D \vee \overline{A}B \vee \overline{A}B = \overline{A}C\overline{D} \vee B\overline{C}D \vee A\overline{C}D \vee \overline{A}B$$

nur einfach berücksichtigt werden.

Veranstaltung am 28.04.2010

Diese reduzierte Form ist allerdings noch nicht in jedem Fall die aufwandsärmste. Daher folgt ein weiterer Schritt, der sich der **Minterm-Primimplikanden-Tabelle** bedient. In den linken Zeilenköpfen stehen wie zuvor alle Minterme der DNF, in den oberen Spaltenköpfen alle gefundenen Primimplikanden. Nun werden die Minterme markiert, aus denen die Primimplikanden hervorgegangen sind. Dabei stehen bei den Primimplikanden mit 3 Variablen immer nur 2 Kreuze, beim Primimplikand P4 mit 2 Variablen aber 4, da er ja aus der Reduzierung von 4 Mintermen entstand:

	P1= $\overline{A}C\overline{D}$	P2= $B\overline{C}D$	P3= $A\overline{C}D$	P4= $\overline{A}B$
$\overline{A}\overline{B}C\overline{D}$	+			
$\overline{A}B\overline{C}\overline{D}$				+
$\overline{A}B\overline{C}D$		+		+
$\overline{A}BC\overline{D}$	+			+
$\overline{A}BCD$				+
$A\overline{B}\overline{C}D$			+	
$AB\overline{C}D$		+	+	

Im letzten Schritt werden diejenigen Primimplikanden ausgewählt, mit denen alle Minterme **einmal** markiert sind. Dabei fallen solche als Anteile für Z heraus, die bereits vollständig in den anderen berücksichtigt wurden. Im Beispiel trifft das auf P2 zu, da der Minterm der 3. Zeile schon durch P4 und der Minterm der 7. Zeile schon durch P3 erfasst ist.

Die aufwandsärmste Version von Z ist also

$$Z = \overline{A}C\overline{D} \vee A\overline{C}D \vee \overline{A}B \quad .$$

Aufgabe: Stellen sie mithilfe von Wahrheitstabellen fest, dass diese Version vom Ergebnis her identisch mit der DNF ist.

Hinweis auf eine Variante: Der Ablauf im Quine-McClusky-Verfahren lässt sich noch straffen, wenn man die Minterme nach Gruppen so sortiert, dass die erste (oberste) Gruppe alle Minterme enthält, bei denen alle Variablen negiert erscheinen, die zweite Gruppe alle Minterme mit nur einer nicht negierten Variablen usw., siehe Skript 5.3.3. Dann muss nicht mehr jeder Minterm explizit mit jedem anderen verglichen werden, sondern nur diejenigen benachbarter Gruppen, weil nur in diesen überhaupt Minterme auftreten, die sich in höchsten einer negierten und nicht negierten Variablen unterscheiden – und nur die sind interessant.

Im vorigen Beispiel enthielte die erste Gruppe dann die Minterme der Zeilen 1 und 2, die zweite Gruppe die der Zeilen 3, 4 und 6 und die dritte Gruppe die der Zeilen 5 und 7.

Durchlauf 1								Durchlauf 2			Durchlauf 3
$\overline{A}\overline{B}C\overline{D}$	+							$\overline{A}C\overline{D}$ **= P1**			$\overline{A}B$ **= P4**
$\overline{A}B\overline{C}\overline{D}$		+	+					$\overline{A}B\overline{C}$	+		$\overline{A}B$
$A\overline{B}\overline{C}D$				+				$\overline{A}B\overline{D}$		+	
$\overline{A}B\overline{C}D$		+			+	+		$A\overline{C}D$ **= P3**			
$\overline{A}BC\overline{D}$	+		+				+	$\overline{A}BD$		+	
$\overline{A}BCD$					+		+	$B\overline{C}D$ **= P2**			
$AB\overline{C}D$				+		+		$\overline{A}BC$	+		

Man erhält das gleiche Ergebnis, wegen der Vorgruppierung sind aber weniger Vergleiche erforderlich. **Frage:** Wie viele Vergleiche benötigt man für dieses Beispiel ohne und mit Gruppierung? (Antwort: 21 und 12)

Frage: Wie viele Vergleiche benötigt man für n Minterme ohne Gruppierung? (Antwort: $\binom{n}{2}$)

6 Zahlensysteme der Digitaltechnik (zum Skript Kapitel 6)

„Rechner rechnen“: Diese verkürzende Feststellung ist ungenau, wenn man die Herzen der Rechner, also die Prozessoren betrachtet. Diese können ausschließlich Boolsche Verknüpfungen bzw. schaltalgebraische Operationen und Schiebe-Operationen durchführen, wie es auch das Modell der **Turing-Maschine** beschreibt.

> **Hinweis:** Nur die XOR-Funktion stellt eine Ausnahme dar: Sie bildet die Addition zweier Zahlen im endlichen Zahlenkörper Z_2 mit seinen beiden Elementen 0 und 1 ab:
>
> $$z = (x + y) \ \mathrm{MOD}\, 2 \ .$$
>
> Hier gilt dann auch $+1 = -1$, so dass die Subtraktion gleich der Addition ist (warum?).

Mit dem vom Menschen vor allem gebrauchten Zahlenrechnen hat das nichts zu tun. Allerdings lassen sich Boolsche Verknüpfungen und Schiebe-Operationen sehr gut zur Darstellung von Zahlen und zum Ausführen der Grundoperation Addition verwenden. Subtraktionen können auf Additionen zurückgeführt, Multiplikationen aus Additionen und Divisionen aus Subtraktion zusammengesetzt werden. Damit sind die Boolschen Verknüpfungen und Schiebe-Operationen letztlich doch für das Zahlenrechnen verwendbar, wenn man die abstrakten Binärmuster aus 0 und 1 zweckmäßig interpretiert.

Zur Darstellung von Zahlen benutzen wir Menschen praktisch ausschließlich das Dezimalsystem, was sich aus verschiedenen Gründen so entwickelt hat. Vom mathematischen Standpunkt aus gibt es aber keinen Grund, dieses zu bevorzugen, im Gegenteil gibt es sogar beliebig viele und gleichwertige Zahlensysteme. Digitalrechner können am einfachsten mit Binärzahlen operieren, weshalb Dezimalzahlen in den meisten Fällen zunächst in Binärzahlen umgewandelt und Rechenergebnisse mit Binärzahlen wieder in Dezimalzahl-Darstellung zurück verwandelt werden.

In der Digitaltechnik benötigt man neben der Darstellung von Zahlen im Dezimal- und Binär-System häufig auch das dem Binärsystem verwandten Oktal- und Hexadezimal-System. Daher ist es

erforderlich, den Aufbau dieser Systeme und deren Umwandlungen zu kennen. Jedes Zahlensystem baut auf seiner Basis B auf:

B = 2 für das Binär-System

B = 8 für das Oktal-System

B = 10 für das Dezimal-System

B = 16 für das Hexadezimal-System.

Die hier verwendeten Zahlenangaben 2, 8, 10 und 16 für die Basis sind im Dezimalsystem dargestellt, weil dies alltagsüblich ist. Bei Gebrauch anderer Zahlensysteme muss man dies zur Eindeutigkeit aber angeben. Eine Möglichkeit besteht darin, der Zahl die Basis B tiefgestellt anzufügen, z. B.

$1101_2 = 13_8 = 11_{10} = B_{16}$

(„B“ ist hier nicht die Basis sondern die Ziffer B im 16-er System, s. weiter unten)

$11_8 = 1001_2 = 9_{10} = 9_{16}$

$101_{10} = 10101_2 = 145_8 = 65_{16}$

$110_{16} = 100010000_2 = 420_8 = 272_{10}$

Zahlen von Zahlensystemen haben die Eigenheit, als geordnete Folge bewerteter Ziffern zu erscheinen. Man nennt sie daher auch **polyadische Systeme**. Jede Zahl c ist dann ein durch die Ziffern a_n, a_{n-1}, … a_1, a_0 als Koeffizienten bewerteter Polynomausdruck der Zahlenbasis B:

$$c = c(B) = a_n \cdot B^n + a_{n-1} \cdot B^{n-1} + \ ... \ + a_1 \cdot B^1 + a_0 \quad .$$

Da Aufbau und die Zuordnung der Koeffizienten zu den Potenzen von B eindeutig sind, gilt diese systematische Darstellung für beliebige Zahlenbasen B und bleibt wegen des nur logarithmisch veränderlichen Umfangs auch für sehr große und kleine Zahlen übersichtlich. Als Kurzform schreibt man

$c = a_n \ a_{n-1} \ ... \ a_1 \ a_0$ (das ist die Reihenfolge der Koeffizienten = Ziffern, kein Produkt)

oder mit Kennzeichnung der Basis B

$c = a_n \ a_{n-1} \ ... \ a_1 \ a_{0\,(B)}$.

Die Ziffernsymbole a sind Elemente der Menge

$a \in \{0, \ 1, \ ..., \ B-1\}$.

B = 2:	$a \in \{0,1\}$
B = 3:	$a \in \{0,1,2\}$
B = 4:	$a \in \{0,1,2,3\}$
…	…
B = 8:	$a \in \{0,1,2,3,4,5,6,7,8\}$
…	...
B = 10:	$a \in \{0,1,2,3,4,5,6,7,8,9\}$
B = 11:	$a \in \{0,1,2,3,4,5,6,7,8,9,A\}$ mit $A_{11} = 10_{10}$
…	…

B = 16: $a \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

… ...

Frage: Wie schreibt man die die Binärzahl c = 10011 als polynomischen Ausdruck?

Antwort: Die Basis B ist als Dezimalzahl ausgedrückt $B = 2_{10}$, als Binärzahl $B = 10_2$. Will man c konsequent nur mit den beiden Elementen 0 und 1 des Binärsystems ausdrücken, wäre das

$$c_2 = 10011_2 = \left(1 \cdot 10^{100} + 0 \cdot 10^{11} + 0 \cdot 10^{10} + 1 \cdot 10^{1} + 1 \cdot 10^{0}\right)_2 \quad .$$

Diese Schreibweise ist sehr ungewohnt, weshalb man das Ergebnis 10011, die Basis B und die Exponenten stillschweigend im Dezimalsystem angibt, also

$$c = 19 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$ oder weiter vereinfacht und mit $2^0 = 1$

$$c = 19 = 1 \cdot 2^4 + 1 \cdot 2^1 + 1 \quad .$$

Die Alternative zu den polyadischen Zahlensystemen sind die **Additionssysteme** wie z. B. das Strichsystem, bei dem man zur Wahrung der Übersichtlichkeit immer Fünfergruppen bildet (Bierdeckel). Das Addieren geschieht bei diesen einfach durch Zusammenlegen. Die Subtraktion erfolgt durch Wegstreichen von Strichen. Leider sind Multiplikation und Divisionen hier sehr aufwändig, weshalb sich Additionssysteme, von einfachen Sonderfällen abgesehen, nicht für das allgemeine Zahlenrechen durchsetzen konnten.

Für die Arbeit mit Zahlensystemen ist die Kenntnis darüber notwendig,

- wie eine Zahl vom einen in ein anderes System umgewandelt wird
- wie die Grundoperation Addition durchgeführt wird.

Die restlichen 3 Grundoperationen Subtraktion, Multiplikation und Division lassen sich auf die Addition zurückführen. Funktionen wiederum werden aus den 4 Grundoperationen aufgebaut. Die beiden Grundaufgaben bei der Verwendung digitialer Schaltungen für das Rechnen sind also die

- Zahlendarstellung einschließlich ihrer Umwandlung zwischen den Zahlensystemen
- die Durchführung der Addition zweier Zahlen.

Alle weiteren Aufgaben des Zahlenrechnens bauen darauf auf.

Die einfachste Darstellung von Zahlen mit digitalen Schaltungen ist die Binärdarstellung, was wegen der beiden einzigen Zahlenelemente 0 und 1 und ihrer Zuordnung zu den beiden Logik-Pegeln 0 und 1 bzw. L und H naheliegt. Die Ein- oder Ausgangspegel einer Gruppe von Gattern lassen sich also unmittelbar als Binärzahlen interpretieren.

Die **Addition** zweier Zahlen ist eine vom Zahlensystem unabhängige Operation, weshalb wir den Ablauf von der Arbeit mit Dezimalzahlen übernehmen können: Ausgehend von der kleinsten Stelle rechts werden die Ziffern gleichen Gewichts addiert, falls im Ergebnis die Basiszahl B enthalten ist, wird diese herausgelöst und als Übertrag mit in die nächsthöhere (= linke) Stelle übernommen. Für Binärzahlen läuft das besonders einfach ab. Ein Beispiel:

```
  10011
 +11001
 ------
 101100
```

Die rechte Stelle ergibt bereits einen Übertrag, die zweite dann ebenfalls usw.

Aufgaben: Berechnen Sie die Summe der beiden Oktalzahlen 455_8 und 367_8 . (Ergebnis: 1044_8). Berechnen Sie die Summe der beiden Hexadezimal-Zahlen $A5D_{16}$ + $F71E_{16}$. (Ergebnis $1017B_{16}$).

Subtrahiert wird ebenfalls entsprechend dem vom Dezimalsystem bekannten Verfahren. Allerdings lässt sich die Subtraktion mithilfe des **Komplement-Verfahrens** so in Einzelschritte zerlegen, dass nur eine – insbesondere im Binärsystem sehr einfach durchzuführende - Subtraktion und sonst reine Additionen erforderlich sind. Der Realisierung mit Digitalschaltungen kommt das entgegen, da man sich auf Additionsschaltungen beschränken und diese zum Beispiel besonders schnell gestalten kann. Um das Verfahren zu erläutern, wird vereinbart::

a → Minuend (Beispiel: a = $1\,011\,101_2$)

b → Subtrahend (Beispiel: b = $11\,011_2$ = $0\,011\,011_2$)

Führende Nullen haben keine Bedeutung, dienen aber der besseren Übersichtlichkeit!

c = a-b → Differenz

n → Stellenzahl des Operanden mit den meisten Stellen (hier n = 7).

B → kleinste Zahl, die im gewählten Zahlensystem mit (n+1) Stellen dargestellt werden kann. (hier B = 10 000 000).

(B-1) → größte Zahl, die im gewählten Zahlensystem mit n Stellen dargestellt werden kann (hier (B-1) = 1 111 111), auch **(B-1)-Komplement** genannt.

Das Komplementverfahren geht von folgender Aufteilung der Subtraktion aus, die für jedes beliebige Zahlensystem gilt:

$$c = a-b = a+\left[(B-1)-b\right]-(B-1) \quad .$$

Im Allgemeinen ist damit nichts gewonnen, da – selbstverständlich – weiterhin Subtraktionen erforderlich sind, nun sogar eine mehr als zuvor. Im Binär-System aber lässt sich die Subtraktion

$$\left[(B-1)-b\right]$$

besonders einfach ausführen. Die Differenz ist identisch mit dem Komplement zu „b“, alle Stellen von „b“ müssen nur negiert werden. Der Grund liegt darin, dass (B-1) nur aus Einsen besteht und beim Subtrahieren von „b“ daher kein „Borgen“ aus der nächsthöheren Stelle nötig wird, was gewöhnlich aufwändig ist. Im Beispiel:

$\left[(B-1)-b\right]$ = 1 111 111

- 0 011 011

1 100 100 = (B-1)-Komplement

Das (B-1)-Komplement wird zum Minuenden „a“ addiert, vom Ergebnis ist aber (B-1) noch abzuziehen, also wieder eine Subtraktion, diesmal aber im Allgemeinen mit „Borgen“ und daher kein Vorteil. Addiert man jedoch zum (B-1)-Komplement eine Eins, so ist von diesem Ergebnis nicht (B-1), sondern (B-1) + 1 = B abzuziehen, was identisch mit dem Streichen der führenden Übertrags-“1“ ist. Die Subtraktionsgleichung sieht nun so aus:

$$c = a-b = a+\left[(B-1)-b+1\right]-(B-1)+1 = a+\left[(B-1)-b+1\right]-B \quad .$$

Hierin ist

$\left[(B-1)-b+1\right]$ = 1 111 111

- 0 011 011

1 100 100 = (B-1)-Komplement

+0 000 001

1 100 101

Addition zu a ergibt:

$a+\left[(B-1)-b+1\right]$ = 1 011 101

+ 1 100 101

11 000 010 → Übertrag in die Stelle (n+1) !

Nach Abzug von B (= Streichen der führenden „1“) erhält man die gesuchte Differenz:

11 000 010

-10 000 000 = B

01 000 010 = Ergebnis = Betrag von c = Differenz.

Leider trifft das in dieser einfachen Form nur zu, wenn mit a>b der Minuend größer als der Subtrahend ist (positives Ergebnis „c“), da nur dann im Zwischenergebnis $\left[(B-1)-b+1\right]$ ein Übertrag in die (n+1)-te Stelle auftritt (**warum?**). Bei a < b fehlt der Übertrag (wieder **warum?**), so dass die anschließende Subtraktion von B ein negative Differenz (= negative Zahl) ergibt. Um ihren Betrag zu ermitteln, wendet man die Definition für Beträge an:

$|c| = c \quad \text{für} \quad c \geq 0$

$|c| = -c \quad \text{für} \quad c < 0$.

Werden im obigen Beispiel die Werte von a und b vertauscht, also

a = 11 011

b = 1 011 101

so ändert sich im Ablauf nichts:

$\left[(B-1)-b+1\right]$ = 1 111 111

- 1 011 101

0 100 010 = (B-1)-Komplement

+0 000 001

0 100 011 = kein Übertrag !

Addition zu a ergibt:

$a+\left[(B-1)-b+1\right]$ = 0 011 011

+ 0 100 011

0 111 110 → kein Übertrag in die Stelle (n+1) !

Bei Abzug von B käme das erwartete negative Ergebnis heraus. Man würde - wie bei der Subtraktion üblich - die Operanden vertauschen, also nicht B vom Zwischenergebnis sondern umgekehrt das Zwischenergebnis von B abziehen und ein negatives Vorzeichen davor schreiben.

Macht man dies hier ebenfalls, so erhält man

$B-\left(a+\left[(B-1)-b+1\right]\right)$ = 10 000 000

-00 111 110

01 000 010 → „Borgen“ erforderlich,

was wieder einfacher mit der Komplement-Bildung über (B-1) und anschließender Addition von 1 zu machen ist:

$$(B-1)+1-\left(a+\left[(B-1)-b+1\right]\right)$$

$$= 1\ \ 111\ 111$$
$$-\ 0\ 111\ 110$$
$$1\ \ 000\ 001$$
$$+0\ \ 000\ 001$$
$$1\ 000\ 010$$

= **Betrag** des negativen Ergebnisses !

Nochmals: Die Subtraktion ist auch mit dem beschriebenen Verfahren in jedem Fall auszuführen, man kann sie aber im Binärsystem besonders einfach über die Komplementbildung mit (B-1) erledigen.

Als Ergebnis der vorangegangenen Erläuterungen bleiben die Regeln (siehe Skript 6.3.2.3):

(1) (B-1)-Komplement zum Subtrahenden b durch Negation der „0“ und „1“-Elemente bilden

(2) „1“ addieren

(3) Minuend a addieren

(4) Falls Übertrag in Stelle (n+1) auftritt: Übertrag streichen → positives Ergebnis

(5) Falls Übertrag in Stelle (n+1) nicht auftritt: (B-1)-Komplement bilden und 1 addieren → Betrag des negativen Ergebnisses

Die **Multiplikation** zweier Binärzahlen besteht aus einer Folge von Additionen. Für das Produkt gilt die Definition

Produkt = Multiplikand · Multiplikator

Am einfachsten aber uneffektivsten erhält man das Produkt dadurch, dass man den Multiplikanden so oft addiert, wie es dem Zahlenwert des Multiplikators entspricht. Bedeutend weniger Aufwand entsteht, wenn man die Stellenwertigkeit der Ziffern des Multiplikators berücksichtigt. Dann kann das Produkt durch eine Folge von Produkten des Multiplikanden, Rechtsverschiebungen und Additionen erfolgen. Verschiebungen sind in Digitalschaltungen über **Speicherketten** (= Register) einfach zu realisieren.

Besonders einfach wird die Multiplikation im Binärsystem, da als Produkte des Multiplikanden wegen der Ziffern 0 und 1 im Multiplikator nur die Teilergebnisse 0 oder „Multiplikand“ entstehen können. Ein Beispiel:

10 111 · 1 001 =

10 111 000	
+ 0 000 000	0 · Multiplikand und einmal rechts verschoben
10 111 000	
+ 000 000	0 · Multiplikand und zweimal rechts verschoben
10 111 000	
+ 10 111	1 · Multiplikand und dreimal rechts verschoben
11 001 111	= Produkt

Die **Divison** besteht aus einer Folge von Subtraktionen mit verschobenen Subtrahenden, wobei im Allgemeinen am Ende ein Rest kleiner als der Divisor übrig bleibt. Für die Division gilt die Definition:

Quotient = Dividend : Divisor

Vom Dividenden wird so oft der Divisor subtrahiert, wie es dem Zahlenwert des Divisors entspricht.

Effektiver ist es, die Stellenwertigkeiten der Ziffern des Dividenden einzubeziehen. Dabei dürfen nur positive Zwischenergebnisse entstehen, andernfalls hängt man rechts weitere Stellen aus dem Dividenden an. Ein Beispiel:

11 001 111 : 1001 = 10 111	
- 10 01	geht einmal in 1100, also subtrahieren ...
00 111	… und eine Stelle aus dem Dividenden „herunterziehen“
- 1 001	geht 0-mal in 0111 , daher nicht subtrahieren und die nächste Stelle aus dem Dividenden herunterziehen
111 1	
- 100 1	geht 1-mal in 1111, subtrahieren und die nächste Stelle aus dem Dividenden herunterziehen
011 01	
- 10 01	geht 1-mal in 1111, subtrahieren und die letzte Stelle aus dem Dividenden herunterziehen
01 001	
- 1 001	geht 1-mal, also subtrahieren
0 000	0 oder Rest < 1001 → Division beendet.

Die Subtraktionen lassen sich dabei immer über das vorher beschriebene Verfahren auf Komplementbildungen und reine Additionen zurückführen.

Die Verfahren für die Grundoperationen mit Binärzahlen liegen damit fest. Fließ- und Gleitkomma-Rechnungen werden hierauf zurückgeführt, was aber keine wesentlich neuen Gesichtspunkte mit sich bringt.

Die bisherigen Überlegungen gelten für Zahlen in Binärdarstellung. Meist liegen Zahlen aber im Dezimal-System vor. Und manchmal werden sie auch in Oktal- oder Hexadezimal-Darstellung benötigt. Dann muss man sie von einem ins andere System umwandeln. Es gilt:

Eine Zahl der Basis B wird über die Polynomdarstellung in eine Dezimalzahl umgewandelt

$$c_B = (a_n a_{n-1} ... a_1 a_0)_B = \left(a_n \cdot B^n + a_{n-1} \cdot B^{n-1} + \ ... \ + a_1 \cdot B^1 + a_0\right)_{10} = c_{10} \ .$$

Für die Koeffizienten (= Ziffern) a_i und die Basis B wird hier die Dezimaldarstellung gewählt. Ist B<10, sind die Ziffern der Basis B in der Ziffernmenge der Basis 10 enthalten und brauchen nicht umgewandelt zu werden. Bei B>10 ist zunächst die Darstellung im Dezimal-System durchzuführen.

Beispiele:

$$c_2 = 101101_2 = (1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 1)_{10} = c_{10} = 45_{10}$$

$$c_8 = 473_8 = (4 \cdot 8^2 + 7 \cdot 8^1 + 3)_{10} = c_{10} = 315_{10}$$

$$c_{16} = 14AE_{16} = (1\cdot16^3+4\cdot16^2+10\cdot16^1+14)_{16} = 5294_{10}$$

Die Umkehrung: Eine Dezimalzahl wird über das im Skript Kapitel 6.2.1 beschriebene Verfahren in eine Zahl der Basis B umgewandelt. Dabei läuft eine fortlaufende Division durch B ab, wobei die umgekehrte Folge der jeweilige Reste die Ziffern der Zielsystem-Zahl ergeben. **Beispiele:**

165_{10} → Binärzahl (Basis B =2)

165 : 2 = 82 Rest 1

82 : 2 = 41 Rest 0

41 : 2 = 20 Rest 1

20 : 2 = 10 Rest 0

10 : 2 = 5 Rest 0

5 : 2 = 2 Rest 1

2 :2 = 1 Rest 0

1 : 2 = 0 Rest 1

Die gesuchten Binärzahlstellen sind die Reste von unten nach oben geschrieben:

$165_{10} = 10100101_2$

Oder:

543_{10} → Oktalzahl (Basis B = 8)

543 : 8 = 67 Rest 7

67 : 8 = 8 Rest 3

8 : 8 = 1 Rest 3

1 : 8 = 0 Rest 1

Also $543_{10} = 1037_8$

Oder:

3099_{10} → Hexadezimalzahl

3099 : 16 = 193 Rest 11

193 : 16 = 12 Rest 1

12 : 16 = 0 Rest 12

Also $3099_{10} = C1B_{16}$

Die Umwandlung zwischen Binär- Oktal- und Hexadezimal-Zahlen funktioniert besonders einfach, da man

- je 3 Binärstellen zu einer Oktalstelle
- je 4 Binärstellen zu einer Hexadezimalstelle zusammenfassen kann.

Beispiele:

011_2 $= 3_8$ $= 3_{16}$

1011_2 $= 001\ 011_2$ $= 13_8$ $= B_{16}$

10011100_2 = $10\ 011\ 100_2$ = 234_8 = $9A_1$

Veranstaltung am 05.05.2010

7 **Codierung** (zum Skript Kapitel 7)

Da wir im Alltagsgebrauch zur Informationsdarstellung von Zahlen und Texten fast ausschließlich Schriftzeichen und Dezimalzahlen verwenden, besteht eine wesentliche Teilaufgabe der Digitaltechnik in der Darstellung dieser Elemente durch binäre Muster, um sie anschließend weiter verarbeiten zu können, sowie in deren Rückwandlung in die gewohnten Schriftzeichen und Dezimalzahlen. Diese Maßnahmen nennt man Codierung. Das Vorlesungsskript nennt die – recht abstrakte , aber offizielle – Definition nach DIN 4430:

> Ein Code ist eine Vorschrift für eine eindeutige Zuordnung der Zeichen eines Zeichenvorrats zu denjenigen eines anderen Zeichenvorrats.

Man könnte es auch etwas ausführlicher formulieren:

> Ein Code entsteht durch die Abbildung eines Quellenalphabets in ein Codealphabet, wobei das Codealphabet ganz bestimmte Eigenschaften aufweist, die das Quellenalphabet nicht hat. Die Abbildung selbst muss im Allgemeinen umkehrbar eindeutig sein, damit die durch die Codierung veränderte Form der Informationsdarstellung wieder die ursprüngliche Gestalt annehmen kann. Die Abbildungsvorschriften sind die **Codierungsverfahren**.

Einige Beispiele von sehr vielen:

Farb-Code	zur Kennzeichnung von ohmschen Widerstandswerten
Bar-Code zur	zur Kennzeichnung von Verkaufsartikeln
Pixel-Code	zur Zuordnung von Farb-und Grauwerten zur Bildpunkten
Fehlerkorrektur-Code	zur Beseitigung von Datenübertragungsfehlern
Geheim-Code	zur Verschlüsselung von Klartexten
Kompressions-Code	zur Volumenverringerung redundanter Informationen
Ton-Code	zur Übertragung von Wählziffern-Informationen
Binär-Code	zur Darstellung von Zahlen im Dual (=Binär)-System
BCD-Code	zur Darstellung von Dezimalziffern durch Binärzahlen.
ASCII-Code	zur Darstellung von Schriftzeichen

Man sieht, dass es sehr unterschiedliche Code-Typen gibt, insbesondere beschränken sich Codes nicht nur auf die Zuordnung von einer Zahlendarstellung zu einer anderen. Allerdings hat gerade diese Zuordnungsaufgabe durch den Einsatz von digitalen Schaltungen einen hohen Stellenwert erhalten, da hier ausschließlich mit **Bitmustern** gearbeitet werden kann. Um diese richtig verwenden zu können, ist es notwendig, zuvor eine Vereinbarung über deren **Interpretation** zu treffen.

Ein Bitmuster kann z. B. einem Bildpunkt, einem Schriftzeichen oder einer Zahl zugeordnet sein, ohne diese Festlegungen ist die weitere Verarbeitung sinnlos. Interpretiert man z. B. das Bitmuster eines ASCII-Zeichencodes als Zahl, so führt die Anwendung von Rechenoperationen hierauf im Allgemeinen nicht zu brauchbaren Ergebnissen. Andersherum wäre die Interpretation von Binärzahlen als Bitmuster für Schriftzeichen auch nicht zweckmäßig.

Wir betrachten hier insbesondere zwei Gruppen von Bitmustern:

Alphanumerische Codes zur Darstellung von Schriftzeichen

Numerische Codes zur Darstellung von Zahlen .

Zu den alphanumerischen Codes gehört der ASCII-Code. Die internationale Version ohne Umlaute und die deutsche mit Umlauten sind auf der nächsten und übernächsten Seite zu sehen, die mit Hilfe eines kleinen Programms erzeugt wurden, siehe auch Skript, Kapitel 7.1:

Es bedeuten:

- die jeweils linke Spalte die fortlaufende Nummerierung der 128 Zeichen
- die jeweils zweite Spalte das zugeordnete Binärmuster des in der letzten Spalte stehenden Zeichens
- Spalten 3, 4 und 5 die als Oktal-, Dezimal- und Hexadezimalen interpretierten Binärmuster.

So hat z. B. das Zeichen „m“ die Nummerierung 109 und das Binärmuster 0110 1101 oder als Zahl interpretiert, **was aber nicht zwingend wäre**,

- die Dualzahlen-Darstellung

$$01101101 = 0\cdot 2^7 + 1\cdot 2^6 + 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = 109_{10}$$

- die Oktalzahlen-Darstellung $155 = 1\cdot 8^2 + 5\cdot 8^1 + 5\cdot 8^0 = 109_{10}$
- die Dezimalzahlen-Darstellung $109 = 1\cdot 10^2 + 0\cdot 10^1 + 9\cdot 10^0 = 109_{10}$
- die Hexadezimal-Darstellung $6d = 6\cdot 16^1 + d(=13_{10})\cdot 16^0 = 109_{10}$.

Hinweis: *Der ASCII-Code ist ursprünglich eine 7-Bit-Code. Da Rechner aber als kleinste ausführbare Informationseinheit wenigstens das Byte (= 8 Bit) verwenden, werden die Zeichen meistens als 8-Bit-Binärmuster dargestellt. Deshalb ist das achte (linke) Bit hier als 0 hinzugefügt (nur insofern sind die beiden Tabellen nicht ganz korrekt). Erweiterte ASCII-Code-Varianten nutzen das achte Bit zur Verdoppelung auf 256 Zeichen, mit denen z. B. weitere Eigenheiten nationaler Zeichensätze ausgedrückt werden können.*

Aufgabe: Stellen Sie die Zeichenketten „h_da“ und „Süden“ als ASCII-Code nach DIN 66003 in Binär., Oktal-, Dezimal- und Hexadezimal-Form dar.

Aufgabe: Stellen Sie die die Zeichenkette „Loriot“ als ASCII-Code in Ternär-Form dar (Dreier-System)

Leckerbissen: Gegeben ist die Folge der Elemente 10, 11, 12, 13 14, 15, 16, 17, 20, 22, 24, 31, 121. Ergänzen Sie diese Folge um das nächste Element (Hinweis: Das nächste Element ist zugleich das letztmögliche).

Von den vielen numerischen Codes betrachten wir für unsere Zwecke diejenigen, bei denen Dezimalzahlen-Ziffern in unterschiedlichen Binärformaten dargestellt werden. Warum wird das überhaupt gebraucht? Da Zahlen in erster Linie ja dazu dienen, mit Rechenoperationen weiter verarbeitet zu werden, könnte man sich auf das Rechnen mit Dualzahlen beschränken, zumal dieses platzsparend und schnell funktioniert und als Grundlage für alle weiteren Operationen dient, wie wir vorher gesehen hatten. Die Umwandlung in andere Zahlenformate ist ohnehin auf einfache Weise möglich. Der Grund für die Verwendung unterschiedlicher Formate liegt einmal in der historischen Entwicklung seit der Erfindung der Digitalrechner, zum anderen in funktionellen Vorteilen.

ASCII–Tabelle ISO 646

Nr.	Bin	Okt	Dez	Hex	Zeichen	Nr.	Bin	Okt	Dez	Hex	Zeichen
0	0000 0000	000	000	00	Nul	64	0100 0000	100	064	40	@
1	0000 0001	001	001	01	SOH	65	0100 0001	101	065	41	A
2	0000 0010	002	002	02	STX	66	0100 0010	102	066	42	B
3	0000 0011	003	003	03	ETX	67	0100 0011	103	067	43	C
4	0000 0100	004	004	04	EOT	68	0100 0100	104	068	44	D
5	0000 0101	005	005	05	ENQ	69	0100 0101	105	069	45	E
6	0000 0110	006	006	06	ACK	70	0100 0110	106	070	46	F
7	0000 0111	007	007	07	BEL	71	0100 0111	107	071	47	G
8	0000 1000	010	008	08	BS	72	0100 1000	110	072	48	H
9	0000 1001	011	009	09	HAT	73	0100 1001	111	073	49	I
10	0000 1010	012	010	0a	LF	74	0100 1010	112	074	4a	J
11	0000 1011	013	011	0b	VT	75	0100 1011	113	075	4b	K
12	0000 1100	014	012	0c	FF	76	0100 1100	114	076	4c	L
13	0000 1101	015	013	0d	CR	77	0100 1101	115	077	4d	M
14	0000 1110	016	014	0e	SO	78	0100 1110	116	078	4e	N
15	0000 1111	017	015	0f	SIX	79	0100 1111	117	079	4f	O
16	0001 0000	020	016	10	DLE	80	0101 0000	120	080	50	P
17	0001 0001	021	017	11	DC1	81	0101 0001	121	081	51	Q
18	0001 0010	022	018	12	DC2	82	0101 0010	122	082	52	R
19	0001 0011	023	019	13	DC3	83	0101 0011	123	083	53	S
20	0001 0100	024	020	14	DC4	84	0101 0100	124	084	54	T
21	0001 0101	025	021	15	NAK	85	0101 0101	125	085	55	U
22	0001 0110	026	022	16	SYN	86	0101 0110	126	086	56	V
23	0001 0111	027	023	17	ETB	87	0101 0111	127	087	57	W
24	0001 1000	030	024	18	CAN	88	0101 1000	130	088	58	X
25	0001 1001	031	025	19	EM	89	0101 1001	131	089	59	Y
26	0001 1010	032	026	1a	SUB	90	0101 1010	132	090	5a	Z
27	0001 1011	033	027	1b	ESC	91	0101 1011	133	091	5b	[
28	0001 1100	034	028	1c	FS	92	0101 1100	134	092	5c	\
29	0001 1101	035	029	1d	GS	93	0101 1101	135	093	5d	]
30	0001 1110	036	030	1e	RS	94	0101 1110	136	094	5e	^
31	0001 1111	037	031	1f	US2	95	0101 1111	137	095	5f	_
32	0010 0000	040	032	20	SPACE	96	0110 0000	140	096	60	`
33	0010 0001	041	033	21	!	97	0110 0001	141	097	61	a
34	0010 0010	042	034	22	"	98	0110 0010	142	098	62	b
35	0010 0011	043	035	23	#	99	0110 0011	143	099	63	c
36	0010 0100	044	036	24	$	100	0110 0100	144	100	64	d
37	0010 0101	045	037	25	%	101	0110 0101	145	101	65	e
38	0010 0110	046	038	26	&	102	0110 0110	146	102	66	f
39	0010 0111	047	039	27	'	103	0110 0111	147	103	67	g
40	0010 1000	050	040	28	(	104	0110 1000	150	104	68	h
41	0010 1001	051	041	29	)	105	0110 1001	151	105	69	i
42	0010 1010	052	042	2a	*	106	0110 1010	152	106	6a	j
43	0010 1011	053	043	2b	+	107	0110 1011	153	107	6b	k
44	0010 1100	054	044	2c	,	108	0110 1100	154	108	6c	l
45	0010 1101	055	045	2d	−	109	0110 1101	155	109	6d	m
46	0010 1110	056	046	2e	.	110	0110 1110	156	110	6e	n
47	0010 1111	057	047	2f	/	111	0110 1111	157	111	6f	o
48	0011 0000	060	048	30	0	112	0111 0000	160	112	70	p
49	0011 0001	061	049	31	1	113	0111 0001	161	113	71	q
50	0011 0010	062	050	32	2	114	0111 0010	162	114	72	r
51	0011 0011	063	051	33	3	115	0111 0011	163	115	73	s
52	0011 0100	064	052	34	4	116	0111 0100	164	116	74	t
53	0011 0101	065	053	35	5	117	0111 0101	165	117	75	u
54	0011 0110	066	054	36	6	118	0111 0110	166	118	76	v
55	0011 0111	067	055	37	7	119	0111 0111	167	119	77	w
56	0011 1000	070	056	38	8	120	0111 1000	170	120	78	x
57	0011 1001	071	057	39	9	121	0111 1001	171	121	79	y
58	0011 1010	072	058	3a	:	122	0111 1010	172	122	7a	z
59	0011 1011	073	059	3b	;	123	0111 1011	173	123	7b	{
60	0011 1100	074	060	3c	<	124	0111 1100	174	124	7c	|
61	0011 1101	075	061	3d	=	125	0111 1101	175	125	7d	}
62	0011 1110	076	062	3e	>	126	0111 1110	176	126	7e	~
63	0011 1111	077	063	3f	?	127	0111 1111	177	127	7f	DEL

ASCII–Tabelle DIN 66003

Nr.	Bin	Okt	Dez	Hex	Zeichen	Nr.	Bin	Okt	Dez	Hex	Zeichen
0	0000 0000	000	000	00	Nul	64	0100 0000	100	064	40	@
1	0000 0001	001	001	01	SOH	65	0100 0001	101	065	41	A
2	0000 0010	002	002	02	STX	66	0100 0010	102	066	42	B
3	0000 0011	003	003	03	ETX	67	0100 0011	103	067	43	C
4	0000 0100	004	004	04	EOT	68	0100 0100	104	068	44	D
5	0000 0101	005	005	05	ENQ	69	0100 0101	105	069	45	E
6	0000 0110	006	006	06	ACK	70	0100 0110	106	070	46	F
7	0000 0111	007	007	07	BEL	71	0100 0111	107	071	47	G
8	0000 1000	010	008	08	BS	72	0100 1000	110	072	48	H
9	0000 1001	011	009	09	HAT	73	0100 1001	111	073	49	I
10	0000 1010	012	010	0a	LF	74	0100 1010	112	074	4a	J
11	0000 1011	013	011	0b	VT	75	0100 1011	113	075	4b	K
12	0000 1100	014	012	0c	FF	76	0100 1100	114	076	4c	L
13	0000 1101	015	013	0d	CR	77	0100 1101	115	077	4d	M
14	0000 1110	016	014	0e	SO	78	0100 1110	116	078	4e	N
15	0000 1111	017	015	0f	SIX	79	0100 1111	117	079	4f	O
16	0001 0000	020	016	10	DLE	80	0101 0000	120	080	50	P
17	0001 0001	021	017	11	DC1	81	0101 0001	121	081	51	Q
18	0001 0010	022	018	12	DC2	82	0101 0010	122	082	52	R
19	0001 0011	023	019	13	DC3	83	0101 0011	123	083	53	S
20	0001 0100	024	020	14	DC4	84	0101 0100	124	084	54	T
21	0001 0101	025	021	15	NAK	85	0101 0101	125	085	55	U
22	0001 0110	026	022	16	SYN	86	0101 0110	126	086	56	V
23	0001 0111	027	023	17	ETB	87	0101 0111	127	087	57	W
24	0001 1000	030	024	18	CAN	88	0101 1000	130	088	58	X
25	0001 1001	031	025	19	EM	89	0101 1001	131	089	59	Y
26	0001 1010	032	026	1a	SUB	90	0101 1010	132	090	5a	Z
27	0001 1011	033	027	1b	ESC	91	0101 1011	133	091	5b	Ä
28	0001 1100	034	028	1c	FS	92	0101 1100	134	092	5c	Ö
29	0001 1101	035	029	1d	GS	93	0101 1101	135	093	5d	Ü
30	0001 1110	036	030	1e	RS	94	0101 1110	136	094	5e	^
31	0001 1111	037	031	1f	US2	95	0101 1111	137	095	5f	_
32	0010 0000	040	032	20	SPACE	96	0110 0000	140	096	60	`
33	0010 0001	041	033	21	!	97	0110 0001	141	097	61	a
34	0010 0010	042	034	22	"	98	0110 0010	142	098	62	b
35	0010 0011	043	035	23	#	99	0110 0011	143	099	63	c
36	0010 0100	044	036	24	$	100	0110 0100	144	100	64	d
37	0010 0101	045	037	25	%	101	0110 0101	145	101	65	e
38	0010 0110	046	038	26	&	102	0110 0110	146	102	66	f
39	0010 0111	047	039	27	'	103	0110 0111	147	103	67	g
40	0010 1000	050	040	28	(	104	0110 1000	150	104	68	h
41	0010 1001	051	041	29	)	105	0110 1001	151	105	69	i
42	0010 1010	052	042	2a	*	106	0110 1010	152	106	6a	j
43	0010 1011	053	043	2b	+	107	0110 1011	153	107	6b	k
44	0010 1100	054	044	2c	,	108	0110 1100	154	108	6c	l
45	0010 1101	055	045	2d	–	109	0110 1101	155	109	6d	m
46	0010 1110	056	046	2e	.	110	0110 1110	156	110	6e	n
47	0010 1111	057	047	2f	/	111	0110 1111	157	111	6f	o
48	0011 0000	060	048	30	0	112	0111 0000	160	112	70	p
49	0011 0001	061	049	31	1	113	0111 0001	161	113	71	q
50	0011 0010	062	050	32	2	114	0111 0010	162	114	72	r
51	0011 0011	063	051	33	3	115	0111 0011	163	115	73	s
52	0011 0100	064	052	34	4	116	0111 0100	164	116	74	t
53	0011 0101	065	053	35	5	117	0111 0101	165	117	75	u
54	0011 0110	066	054	36	6	118	0111 0110	166	118	76	v
55	0011 0111	067	055	37	7	119	0111 0111	167	119	77	w
56	0011 1000	070	056	38	8	120	0111 1000	170	120	78	x
57	0011 1001	071	057	39	9	121	0111 1001	171	121	79	y
58	0011 1010	072	058	3a	:	122	0111 1010	172	122	7a	z
59	0011 1011	073	059	3b	;	123	0111 1011	173	123	7b	ä
60	0011 1100	074	060	3c	<	124	0111 1100	174	124	7c	ö
61	0011 1101	075	061	3d	=	125	0111 1101	175	125	7d	ü
62	0011 1110	076	062	3e	>	126	0111 1110	176	126	7e	ß
63	0011 1111	077	063	3f	?	127	0111 1111	177	127	7f	DEL

Im Skript, Kapitel 7.2.1, sind auf der Grundlage des Dualzahlenformats die 4 wesentlichen Dezimalzahlen-Codes beschrieben. Das allen gleiche Prinzip ist die Zuordnung der 10 Dezimalzahlen-Ziffern zu den 16 möglichen Bitmustern von 4 Bits, den **Tetraden**. Es werden also nur 10 Tetraden gebraucht, die restlichen 6 heißen **Pseudotetraden**. Im Einzelnen:

BCD-Code	Jeder Dezimalziffer wird ihr Wert als 4-stellige Dualzahl zugeordnet, **Beispiel:** $5_{10} = 0101_2$ Eine Pseudotetrade ist z. B. $1011_2 = 11_{10}$, da es die Ziffer „11“ im Dezimal-System nicht gibt.
Aiken-Code	Den Dezimalziffern 0, 1, 2 , 3, 4 werden die Dualzahlen wie beim BCD-Code zugeordnet, die Ziffern 5, 6, 7, 8, 9 erhalten $5_{10} = 1011$ (**Hinweis:** Nicht identisch mit $1011_2 = 11_{10}$) .. .. .. $9_{10} = 1100$ usw . Die Wertigkeit der Binärstellen ist nicht wie beim BCD-Code 8 4 2 1, sondern 2 4 2 1 (bitte nachprüfen) Diese Definition hat die Eigenheit, dass sich die zur Zahl 9 komplementären Dezimal-Ziffern im Binär-Format komplementär zu 1111 verhalten und dadurch ohne Rechnung leicht gebildet werden können. **Beispiel:** 3_{10} ist komplementär zu 6_{10} , da 3 + 6 = 9 0011 ist komplementär zu 1100 ($=6_{10}$) , da 0011 + 1100 = 1111. Pseudotetraden sind hier 0101, 0110 usw. Obwohl die Tetraden nicht alle den als Binärzahl ausgedrückten Wert der zugeordneten Dezimalzahl haben (**Beispiel:** Die Aiken-Code-Tetrade für 6_{10} ist 0110, diese ist aber ungleich zur Dualzahlen-Tetrade $0110_2 = 6_{10}$), werden sie wie Binärzahlen behandelt. Man kann mit Ihnen daher auch rechen, siehe weiter unten.
3-Excess-Code auch excess-3 -Code oder Stibitz-Code	Auch dieser Code hat die Komplementär-Eigenschaft wie der Aiken-Code, wird aber durch Addition von 0011 zur BCD-Form gebildet, zum Beispiel: $2_{10} = 0010 + 0011 = 0101$ $7_{10} = 0111 + 0011 = 1010$ 2+7 =0101 + 1010 = 1111 = 9 Die Binärstellen der Tetraden haben hier die Wertigkeiten $8\ 4\ -\overline{2}\ -\overline{1}$, d. h., dass das zweite Bit von links negiert und das doppelte Ergebnis abgezogen wird. Ähnlich wird mit der linken Stelle verfahren. **Beispiel:** $1010 = 1\cdot 8 + 0\cdot 4 - 0\cdot 2 - 1\cdot 1 = 9_{10}$.

Gray-Code (erweitert auf alle 16 Tetraden	Er hat die Eigenschaft, dass sich beim Übergang einer Dezimalziffer zur nächst kleineren oder nächst größeren jeweils nur eine Binärstelle in der Tetrade ändert. **Beispiel:** 3 auf 4 im DualCode: 0011_2 auf 0100_2 (es ändern sich 3 Stellen) 3 auf 4 im Gray-Code 0010 auf 0110 (es ändert sich 1 Stelle) 8 auf 7 im Dual-Code: 1000_2 auf 0111_2 (es ändern sich 4 Stellen) 8 auf 7 im Gray-Code: 1100 auf 0100 (es ändert sich 1 Stelle) Dies ist nützlich in der Messtechnik, um Fehler bei der automatischen Ablesung gering zu halten. Wenn z. B. bei einer Federwaage die Ablesung des Wiege-Ergebnisses dadurch erfolgt, dass eine Zeile aus vier Photodioden den Lichtdurchfall eines gemäß dem Gray-Code geschwärzten Tetradenfeldes registriert, können bei einer Zwischenstellung der Photodiodenzeile Bits aus beiden Tetraden erfasst werden, was bei Gray-Codierung aber nur den Fehler einer Dezimalstelle erzeugen würde. Bei Dualzahlen-Codierung wäre dagegen in vielen Fällen ein großer Fehler zu erwarten. **Beispiel:** Die Photodiodenzeile steht zwischen 7 und 8. Bei Gray-Codierung kann das Ergebnis 0100 1100 ‾‾‾‾ 1100 $\rightarrow 8_{10}$ erscheinen, die Abweichung wäre etwa eine halbe Dezimalstelle. Bei Dualzahlen-Codierung erhielte man möglicherweise 0111 1000 ‾‾‾‾ 1111 $\rightarrow 15_{10}$, also einen erheblichen Fehler. **Frage:** Warum sind BCD-, Aiken- und Excess-3-Code für die Erfassung von Messergebnissen prinzipiell ungeeignet? **Hinweis:** Man kann den Gray-Code für beliebig viele Binärstellen aufbauen, also nicht nur für Tetraden. Das Bildungsgesetz ist: Die höchste Stelle g_n stimmt mit der Stelle d_n der zugeordneten Dualzahl überein. Die anderen Stellen g_i sind 1, wenn die Stellen d_{i+1} und d_i der zugeordneten Dualzahl unterschiedlich sind, sonst 0. Dies ist die Antivalenz oder XOR-Funktion. **Frage:** Wie lauten die Gray-Codewörter für die Dezimalzahlen 15 und 16, wie von 23 und 24? (Antwort: 01000 und 11000, bzw. 11100 und 10100)

Addition und **Subtraktion** von BCD-codierten Dezimalzahlen werden ausführlich. im Skript Kapi-

tel 7.2.2 beschrieben.

Aufgabe: Wie stellt man 396_{10} als binäre Tetraden dar?

Aufgabe: $4_{10} + 3_{10} = ?$ im BCD-Code?

Aufgabe: $6_{10} - 8_{10} = ?$ im BCD-Code?

Aufgabe: $457_{10} – 869_{10} = ?$ im BCD-Code?

Weitere Codierungs-Aufgaben

Neben der für den Einsatz von Digitalschaltungen elementaren Aufgabe der Umwandlung von Informationen einer beliebigen Form in Binär-Codes (Codeumsetzer, Kapitel 7.3.3) und dem Rechnen mit binär dargestellten Zahlen gibt es weitere bedeutende Aufgaben, die mit speziellen Codes bearbeitet werden und ohne deren Lösung z. B. der heutige Internet-Betrieb weder praktikabel noch überhaupt durchführbar wäre. Einige davon sind als **fehlererkennende** und **fehlerkorrigierende** Codes im Skript Kapitel 7.2.3 erläutert. Dafür gibt es grundlegende Eigenschaften, die ein Code aufweisen muss. Sie werden über das Merkmal „**Hammingdistanz**“ bestimmt. Die **Distanz** oder der Abstand zwischen zwei beliebigen Wörtern eines Codes ist die Anzahl der unterschiedlichen Stellen. Die Hammingdistanz d ist dann die kleinste Distanz eines Codes, die bei Vergleich jedes Codewortes mit jedem anderen auftritt.

Frage: Welche Distanzen treten beim BCD-Code auf? Welche Hammingdistanz hat er?

Für Fehlererkennbarkeit und Fehlerkorrigierbarkeit eines Codes gibt es die einfachen Zusammenhänge zwischen Fehleranzahl und Hammingdistanz

$d = t_{erk} + 1$ mit t_{erk} = Anzahl erkennbarer, aber nicht korrigierbarer Fehler

$d = 2 \cdot t_{kor} + 1$ mit t_{kor} = Anzahl korrigierbarer Fehler.

Fragen: Welche Distanzen treten beim 7-Bit-ASCII-Code auf? Welche Hammingdistanz d hat er? Kann man bei ihm Fehler erkennen oder sogar korrigieren?

Frage: Welche Distanzen treten beim Walking-Code auf (Kapitel 7.2.3.2.2)? Welche Hammingdistanz d hat er? Können mit ihm Fehler behandelt werden?

> **Hinweis:** Für die Übertragung und Speicherung von Informationen sind fehlerkorrigierende Codes von zentraler Bedeutung. Kein Handybetrieb, keine Platten-oder CD/DVD-Speicherung wäre heute möglich, wenn die Informationen nicht über fehlerkorrigierende Codes abgesichert wären.
>
> Auch die **Geheimhaltung** von Informationen, der Nachweis von deren Vertrauenswürdigkeit und unverfälschtem Zustand wären ohne geeignete Codierung nicht realisierbar.
>
> Ähnliches trifft für die **Kompression** von Daten zu. Wer mehr hierzu erfahren will, kann aus der umfangreichen Literatur u. a. das Buch „**Codierung**“ von W. Dankmeier verwenden (Vieweg-Verlag, ISBN-10 3-528-25399-1, 3. Auflage, 2006), das die heute aktuellen Verfahren der **Fehlerkorrektur**, **Verschlüsselung** und **Datenkompression** anhand ausführlicher Beispiel erläutert,

In Skript, Kapitel 7.2.3.3, wird als Beispiel für den 1-Bit-Fehler korrigierenden Hammingcode derjenige erläutert, der 4 Dualzahlenbits und 3 Prüfbits enthält. Jedes Codewort besteht also aus 7 Bits.

Man kann diesen Code ganz allgemein zur fehlerkorrigierbaren Informationsübertragung verwenden, wenn man die 4 Dualzahlenbits als Bitmuster auffasst. Diese können dann Tetraden für Dezimal-Ziffern, Teile von Pixelblöcken eines Bildes, Teile von ASCII-codierten Zeichenketten, Teile von C-Programmcode oder etwas beliebig anderes sein. Man muss durch Blockung nur dafür sorgen, dass immer 4-Bit-Pakete entstehen. Da die Bitmuster in 2^4 verschiedenen Anordnungen erscheinen können, gibt es 16 Codewörter. Üblich ist auch die Darstellung als Codewort v

$$v=(u_1\ u_2\ u_3\ u_4\ y_1\ y_2\ y_3)$$

mit den Informationsbits u und den Prüfbits y (die Kontrollgruppen). Dieser Code ist wie die Version im Skript **systematisch**, da die Informations- und Prüfbits getrennt an festen Plätzen stehen. Das muss nicht sein, hält aber den Verarbeitungsaufwand gering. Die 3 Prüfbits lassen sich über folgende Gleichungen aus den Informationsbits bestimmen:

$$y_1=\left[u_1+u_2+u_4\right]\text{MOD}\ \ 2$$

→ MOD 2 bedeutet: wenn $u_1 + u_2 + u_4$ geradzahlig, $y_1 = 0$, wenn ungeradzahlig $y_1 = 1$.

$$y_2=\left[u_1+u_3+u_4\right]\text{MOD}\ \ 2$$

$$y_3=\left[u_2+u_3+u_4\right]\text{MOD}\ \ 2$$

Aufgabe: Stellen Sie die Wahrheitstabelle und die DNFs für die Prüfbits auf. Minimieren Sie die Verknüpfungen mit Hilfe der KV-Diagramme (soweit es geht). Skizzieren Sie die Schaltung.

> **Hinweis:** Der Hammingcode wird wegen seines einfachen Aufbaus und des geringen Bearbeitungsaufwands in fast allen Arbeitsspeicher-Controllern zur Absicherung von 1-Bit-Schreib- und Lesefehlern eingesetzt. Für 32-Bit orientierte Bausteine benötigt man dann zusätzlich zu den 32 Infobits insgesamt 7 Prüfbits. Mit diesen kann ein 1-Bit-Fehler korrigiert und zusätzlich ein 2-Bit-Fehler erkannt werden. Für die Datenübertragung auf Leitungen oder Funkkanälen reicht die Leistung des Hammingcodes aber nicht aus. Hier setzt man Codes für **Mehrbit-Fehler-Korrektur** wie den **BCH**-, **Reed-Solomon**-, **Turboprodukt**- oder **Faltungs-Code** ein (zur Funktionsweise siehe z. B. das oben genannte Buch „Codierung“).

Außer der Hammingdistanz spielen Begriffe wie **Redundanz** (absolute Redundanz und relative Redundanz) eine Rolle (Kapitel 7.2.3.1.1).

Frage: Welche absolute und relative Redundanz hat der 7-Bit-ASCII-Code, welche der 7-Bit-Hammingcode?

8 Rechenschaltungen (zum Skript Kapitel 8)

Werden Bitmuster als Zahlen interpretiert, lassen sich damit, wie in Kapitel 6 beschrieben, Rechenoperationen ausführen. Die grundlegendste Operation ist die Addition von Binärzahlen, da alle weiteren Operationen hierauf zurückgeführt werden können. Wie im vertrauten Additionsverfahren für Dezimalzahlen werden Binärzahlen von rechts nach links unter Berücksichtigung der Überträge addiert.

Die Summe aus den geringwertigsten rechten Stellen verarbeitet dabei nur die geringwertigsten Stellen der beiden Summanden, während die Summen der höherwertigen Stellen auch die Überträge einbeziehen. Der Addierschaltung für die niederwertigste Stelle kann daher etwas einfacher aufgebaut sein. Man bezeichnet sie auch als **Halbaddierer**, die restlichen als **Volladdierer**. Die Funktionen und Schaltungen sind ausführlich im Skript beschrieben.

Frage: Die Summenfunktion des Halbaddierers ist $S=\overline{A}B\vee A\overline{B}$ bzw. $S=(A\vee B)\overline{\ddot{U}}$. Wie kommt man zu dieser zweiten Form?

Antwort: Die Umkehrung des **Absorptionsgesetzes** $A\wedge(\overline{A}\vee B)=AB$ und Vertauschung von A mit $\overline{A}$ ergibt

$$\overline{A}\wedge(A\vee B)=\overline{A}B$$

und entsprechend

$$\overline{B}\wedge(A\vee B)=A\overline{B}\quad .$$

Daraus ergibt sich nach Anwendung des Distributivgesetzes die Summe

$$S=\overline{A}B\vee A\overline{B}=\overline{A}\wedge(A\vee B)\vee\overline{B}\wedge(A\vee B)=(A\vee B)\wedge(\overline{A}\vee\overline{B})$$

sowie mit Hilfe der doppelten Negation und der De Morganschen Gesetze auf den zweiten Term:

$$S=(A\vee B)\wedge\overline{\overline{(\overline{A}\vee\overline{B})}}=(A\vee B)\wedge\overline{(A\wedge B)}=(A\vee B)\overline{\ddot{U}}\quad .$$

Hinweis: Wenn man Summe und Übertrag des Halbaddierers nur unter Verwendung der beiden Summanden A und B berechnet, erhält man bei Realisierung mit NAND-Gattern eine aufwändigere Schaltung als bei zusätzlicher Nutzung des Übertrags Ü. Dies steht in scheinbarem Widerspruch zur Aussage, dass mit $S=\overline{A}B\vee A\overline{B}$ und $\ddot{U}=AB$ bereits die minimale Version vorliegt. **Frage:** Was ist der Grund für diesen scheinbaren Widerspruch?

Antwort: Die NAND-Realisierung ist nicht die minimale Gatterversion, sondern die Version mit der kleinsten Zahl unterschiedlicher Gattertypen. Dann lässt sich die Zahl der für den kompletten Halbaddierer erforderlichen NAND-Gatter aber durch Verwendung des ohnehin gebrauchten Übertrags weiter verkleinern → kein Widerspruch.

Frage: Kann die Addition zweier n-Bit langer Dualzahlen durch die Schaltung aus (n-1)-Volladdierer- und einem Halbaddierer-Baustein gleichzeitig in jeder Stelle erfolgen?

Antwort: Nein, aber warum nicht?

9 Schaltwerke (Sequenzielle Logik, zum Skript Kapitel 9)

Im Gegensatz zur bisher behandelten kombinatorischen Logik der **Schaltnetze** hängt bei **Schaltwerken** der augenblickliche Zustand nicht nur von den augenblicklichen Werten der Eingangssignale sondern zusätzlich vom vorausgegangenen Zustand ab, der bis zum augenblicklichen Zustand gespeichert wurde. Dabei liegt die Speicherwirkung nicht in den unvermeidlichen Gatterlaufzeiten, die ohnehin auch bei Schaltnetzen zu berücksichtigen sind, sondern in der Speicherfähigkeit gezielt dazu eingesetzter Bausteine. Man kann diese aus Standard-Gattern aufbauen, wenn man deren Ausgangssignale geeignet auf die Eingänge rückkoppelt und dadurch ein definiertes Speicherverhalten erhält. Eine damit versehene Digitalschaltung durchläuft dann im Allgemeinen eine bestimmte zeitliche Folge gleicher oder verschiedener Zustände, was sich insbesondere in den zeitlichen Verläufen der Ausgangssignale zeigt.

Hinweis: Auch Schaltnetze durchlaufen zeitliche Folgen der Ausgangssignale, wenn die Eingangssignale sich zeitlich ändern. In Schaltnetzen ist aber einem bestimmten Zustand der Eingangssignale immer der **gleiche** Zustand der Eingangssignale zugeordnet. Für Schaltwerke trifft das wegen der zusätzlichen Abhängigkeit von der Vergangenheit im All-

gemeinen nicht zu.

Solche Speicherbausteine sind zum Beispiel bistabile Kippstufen, wie sie im Skript, Kapitel 9, beschrieben werden. Ein Großteil von ihnen besteht aus dem **RS-Flipflop** als **Grundelement** (abgekürzt RS-FF), die übrigen stellen Erweiterungen hierzu dar. Gemäß der in 9.2.1 angegebenen Gliederung beinhalten sie folgende Merkmale:

	Ungetaktet
RS-Flipflop	Q kann durch S=1 und R=0 gesetzt, durch R=1und S=0 rückgesetzt werden. **Der Zustand S=1 und R=1 ist verboten,** da $Q=\overline{Q}$ wäre und dies der Definition widerspräche. Ein reales RS-Flipflop würde einen nicht vorhersagbaren Zustand einnehmen und eventuell unkontrolliert schwingen.

Das Verhalten aller bistabiler Kippstufen kann auf 5 verschiedene Arten dargestellt werden, jede Art dient dabei einem besonderen Zweck:

Nr.	Darstellungsart als ...	Zweck	Beispiel
1	… ausführliche Schaltfolgetabelle	Grundlage für 2, 3, 4, und 5	Skript Seite 9-4
2	… vereinfachte Schaltfolgetabelle	kompakte Form von (1), ist aber direkt keine Grundlage für (2), (3), (4) und (5), sondern muss dafür zunächst wieder in (1) zurückgeführt werden	Skript Seite 9-4
3	… Übertragungsfunktion	Verknüpft die Eingangsvariablen R und S sowie den Ausgangszustand Q_v (vorher) mit dem Ausgangszustand Q_n (nachher), Voraussetzung für (4)	Für RS-FF: $Q_n=S\vee\overline{R}\wedge Q_v$ $\overline{Q_n}=R\vee\overline{S}\wedge\overline{Q_v}$ Skript Seite 9-4
4	… Schaltfunktion	dient als Grundlage zum Aufbau eines Gatter-Schaltbildes, wird aus (3) durch Gleichsetzen von $Q_n = Q_v = X$ und $\overline{Q_n}=\overline{Q_v}=Y$ gebildet	Für RS-FF: $Q=X=S\vee\overline{Y}$ $\overline{Q}=y=R\vee\overline{X}$ Skript Seite 9-5
5	… Impulsdiagramm	zeitliche Darstellung der Variablen, kann aus (1) oder aus dem Schaltbild abgeleitet werden	Skript Seite 9-5

Hinweis zur Schaltfolgetabelle im Skript Seite 9-4: Die Angaben zu Q_n in der letzten Spalte sind die vorgegebenen Werte, welche die Schaltung als Ausgangszustände liefern soll. D. h., es soll sich der Wert von Q_n ergeben, wenn R, S und Q_v die in der gleichen Zeile angegebene Wertekombination hat. Dann muss für einen definierten, stabilen Ausgangszustand des FFs $\overline{Q_n}\neq Q$ erfüllt sein. Da der interne Schaltvorgang des FlipFlops zu diesem Zeitpunkt aber bereits abgeschlossen ist, dürfen zur Überprüfung dieses stabilen Ausgangszustandes nur die Variablen R, S, Q_n und

$\overline{Q_n}$ einbezogen werden. Der Zustand Q_v vor dem Auslösen des internen Schaltvorgangs liegt ja dann nicht mehr vor. Anhand der Zustands 5 (oder auch der anderen) kann man sich das klar machen:

Zustands-Nummer	Q_v	$\overline{Q_v}$	S	R	Q_n	$\overline{Q_n}$
4	1	0	0	0	1	0
5	1	0	0	1	0	1
6	1	0	1	0	1	0

Nur die Werte im gelb markierten Teil der Schaltfolgetabelle müssen verträglich sein (Sind sie es? Bitte nachprüfen!). Q_v hat auf die Verträglichkeit keinen Einfluss, da es **vor** dem Schaltvorgang wirkte. Q_n stellt sich aber **nach** dem Schaltvorgang ein.

Veranstaltung am 12.05.2010

Das RS-FF kann durch verschiedene Zusatzbeschaltungen in seinem Verhalten verändert werden. Insbesondere ist es dabei das Ziel,

- den „verbotenen“ Zustand R=S=1 zu vermeiden,
- mit Hilfe von Taktimpulsen begrenzter Dauer die Übernahme des Zustands von R und S nur zuzulassen, wenn diese Impulse einen bestimmten Wert haben, z. B. nur bei Taktpegel „1“ oder bei positiver Taktflanke,
- die Ausgabe des Zustands um eine Taktimpulslänge zu verzögern (Master-Slave-Technik),
- mit Hilfe zusätzlicher externer Signale den Ausgangszustand unabhängig von der Wirkung der Taktimpulse zu jeder Zeit setzen oder rücksetzen zu können (direkt wirkende Setz- und Rücksetz-Eingänge).

Daraus ergeben sich folgende gebräuchlichen FF-Typen, siehe Skript, Seiten 9-6 bis 9-13:

Ungetaktet	
RS-FF mit dominierendem Setzeingang	Durch eine Zusatzschaltung hinter dem S- und dem R-Eingang kann erzwungen werden, dass bei S=1 und R=1 der S-Eingang Vorrang erhält. Werden die „Tasten R und S gleichzeitig gedrückt“, erhält Q den Wert 1, $\overline{Q}$ den Wert 0
RS-FF mit dominierendem Rücksetzeingang	Wie zuvor! Werden die „Tasten R und S gleichzeitig gedrückt“, erhält aber Q den Wert 0, $\overline{Q}$ den Wert 1.
Statisch getaktet (taktzustandsgesteuert)	
RS-FF	Wie ungetaktetes RS-FF, jedoch werden durch zusätzliche UND-Gatter die Eingänger S und R nur durchgeschaltet, wenn ein Taktsignal C

	mit H-Pegel anliegt. Der Zustand S=1 und R=1 bleibt verboten, da $Q=\overline{Q}$ wäre, was der Definition widerspräche und zu nicht definierten Verhältnissen führen würde. Bei positiver Taktung werden die Eingangssignale durchgeschaltet, wenn C=1 ist, bei negativer Taktung, wenn C = 0 ist.
D-FF	Wie getaktetes RS-FF, aber der R-Eingang ist der negierte S-Eingang, daher tritt der verbotene Zustand nicht auf. Der S-Eingang heißt hier D (von „Data“ oder „Delay“). Immer wenn der Takt C anliegt, wird der Zustand von D in Q übernommen. Man kann auf diese Weise Datenbits zwischenspeichern, daher „D-FF“.
D-FF mit Vorbereitungseingang	Das Taktsignal C wird seinerseits durch eine UND-Verknüpfung mit dem Vorbereitungseingang V „enabled“. Solche Enable-Funktionen wendet man immer dann an, wenn eine Funktion vorübergehend abgeschaltet werden soll, ohne sie entfernen zu müssen. Beispiel: Interrupt-Eingang bei Mikroprozessoren.
RS-Master-Slave -FF (taktzustandsgesteuert)	Besteht aus zwei hintereinander geschalteten FFs. Der Master bildet während des Taktimpulses den neuen Ausgangszustand und gibt ihn um einen Taktimpuls verzögert an den Slave ab. Durch die entgegengesetzte Taktung von Master und Slave entsteht trotz der physikalischen Zustandsteuerung mit dem Takt C nach außen der Eindruck, als ob der neue Ausgangszustand an der negativen Taktflanke von C erscheint. Das RS-MS-FF arbeitet also **zweizustandsgesteuert**, verhält sich aber wie **einflankengetriggert**.
Dynamisch getaktet (taktflankengesteuert)	
RS-Master-Slave -FF (taktflankengesteuert)	Wie taktzustandsgesteuertes RS-MS-FF, der Zustand der Eingangsvariablen R und S wird aber • bei positiver Taktung nur während der positiven Taktflanke • bei negativer Taktung nur während der negativen Taktflanke verarbeitet.
JK-Master-Slave-FF	Ähnlich dem RS-MS-FF, durch kreuzweises Rückkoppeln der Ausgänge Q und $\overline{Q}$ auf die Eingänge R und S mit einer UND-Verknüpfung wird aber hier der verbotene Zustand R=S=1 verhindert. Die neuen Eingänge heißen J und K. Besonderheit: Bei J=1 und K=1 wechselt der Ausgangszustand.
T-FF	Ein JK-MS-FF mit fest verdrahteten Eingängen J=K=1. Der Ausgang schaltet also bei jedem Taktimpuls um, siehe oben. Das T-FF dient deshalb als 2:1-Frequenzteiler.
JK-FF und T-FF einflankengesteuert	„JK-MS-FF ohne Slave“, der Ausgangszustand wird unverzögert ausgegeben. Wegen der internen Rückkopplungen wie beim JK-MS-FF funktionieren diese Typen nur flankengetriggert. Bei Zustandssteue-

	rung würden sie unkontrolliert schwingen.
FlipFlops mit direkt wirkenden Setz- und Rücksetzeingängen	
Alle Typen	Zusätzliche, direkt wirkende Setz- und Rücksetz-Eingänge ermöglichen das gezielte Setzen und/oder Rücksetzen des Ausgangszustandes zu einem beliebigen Zeitpunkt innerhalb oder außerhalb des Taktimpuls-Zeitraums.

Veranstaltung am 19.05.2010

Weitere Schaltungen mit Kipp-Eigenschaften:

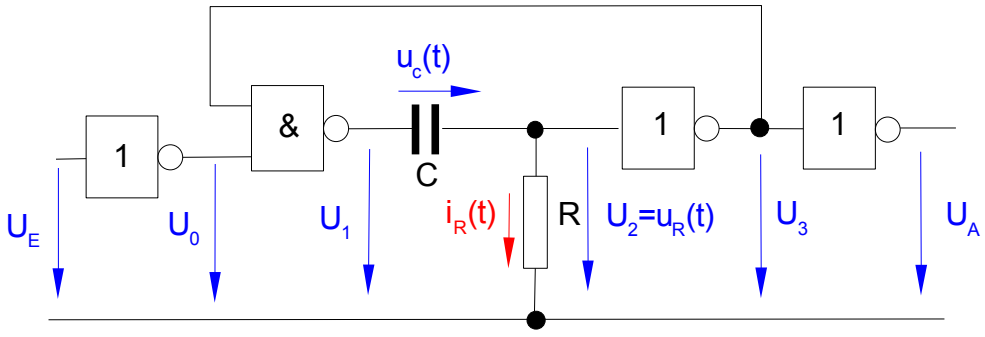
Monostabile Kippstufen (Monoflops) werden zur Erzeugung von Impulsen definierter Länge eingesetzt.

Die Funktionsweise eines aus Gattern aufgebauten Monoflops kann man sich alternativ zu der im Skript Kapitel 9.2.2, Seite 9-14, gegebenen Erläuterung auch anhand einer auf positiver TTL-Logik beruhenden Anordnung klar machen. Dabei wird vereinfachend Folgendes angenommen

- 5 Volt entsprechen dem H-Pegel (logische „1“).
- 0 Volt entsprechen dem L-Pegel (logische „0“).
- Die Gatter-Eingänge sind hochohmig, es fließen keine nennenswerten Ströme über die Gattereingänge.
- Die Gatter-Ausgänge sind niederohmig, sie wirken praktisch wie ideale Spannungsquellen.
- Die Spannung der Gattereingangspegel für die Umschaltung der Gatterausgänge von 5 Volt auf 0 Volt bzw. von 0 Volt auf 5 Volt beträgt 2.5 Volt.

Hinweis: Für das Verständnis der prinzipiellen Wirkungsweise reichen die genannten Annahmen aus, für eine genauere Betrachtung müssen aber die realen elektrischen Eigenschaften der Gatter einbezogen werden.

Aus dem logischen Gatter-Schaltbild gemäß Skript Seite 9-14 wird damit:



Für die Spannungssumme zwischen NAND- und NOT-Gatter gilt mit der Kirchhoffschen Maschenregel:

$$U_1 - u_c(t) - u_R(t) = 0 \quad \text{oder} \quad U_1 = u_c(t) + u_R(t)$$

Der Strom $i_c(t)$ durch den Kondensator C ist mit der Kondensatorspannung $u_c(t)$ über

$$i_c(t) = C \cdot \frac{u_c(t)}{dt}$$ verknüpft.

Da nach Annahme keine Ströme über den Eingang des NOT-Gatters fließen, ist $i_c(t) = i_R(t)$. Für den Maschenumlauf gilt daher die Differenzialgleichung

$$U_1 = u_c(t) + R \cdot C \cdot \frac{du_c(t)}{dt} \quad .$$

Bei Spannungsverläufen der Quelle $U_1(t)$, welche sich nicht durch eine elementare mathematische Funktion ausdrücken lassen, gibt es für $u_c(t)$ keine Lösung im Sinne eines geschlossenen Funktionsausdruckes. Allerdings kann man die Lösung so gut wie immer über numerische Verfahren ermitteln. Der vorliegende Fall liegt jedoch einfach, da die Spannung $U_1(t)$ wegen ihrer Eigenschaft als Ausgangsspannung des NAND-Gatters ausschließlich eine elementare Sprungfunktion von 0 auf 5 Volt oder von 5 auf 0 V darstellt. Je nach Ladespannung $u_c(t=t_{um})$ des Kondensators **vor** dem Umschaltzeitpunkt t_{um} ergibt sich die Lösung der Differenzialgleichung als

$$u_c(t) = u_c(-t_{um}) + \left(U_1 - u_c(-t_{um})\right) \cdot \left(1 - e^{-\frac{t - t_{um}}{R \cdot C}}\right), \quad t \geq t_{um} \quad .$$

Die Bezeichnung „$-t_{um}$“ soll hier andeuten, dass es sich um den Zeitpunkt „beliebig kurz“ vor dem Umschalten handelt.

Wenn $t_{um} = 0$ ist, geht die Lösung in die bekanntere Form

$$u_c(t) = u_c(-0) + \left(U_1 - u_c(-0)\right) \cdot \left(1 - e^{-\frac{t}{R \cdot C}}\right), \quad t \geq 0$$

über. Befindet sich die Schaltung seit längerer Zeit im stabilen Zustand, so liegen an der Schaltung folgende Spannungen an (bitte nachprüfen):

$U_E = 0$, $U_1 = 0$, $U_2 = 0$, $U_A = 0$ und $U_0 = 1$, $U_3 = 1$.

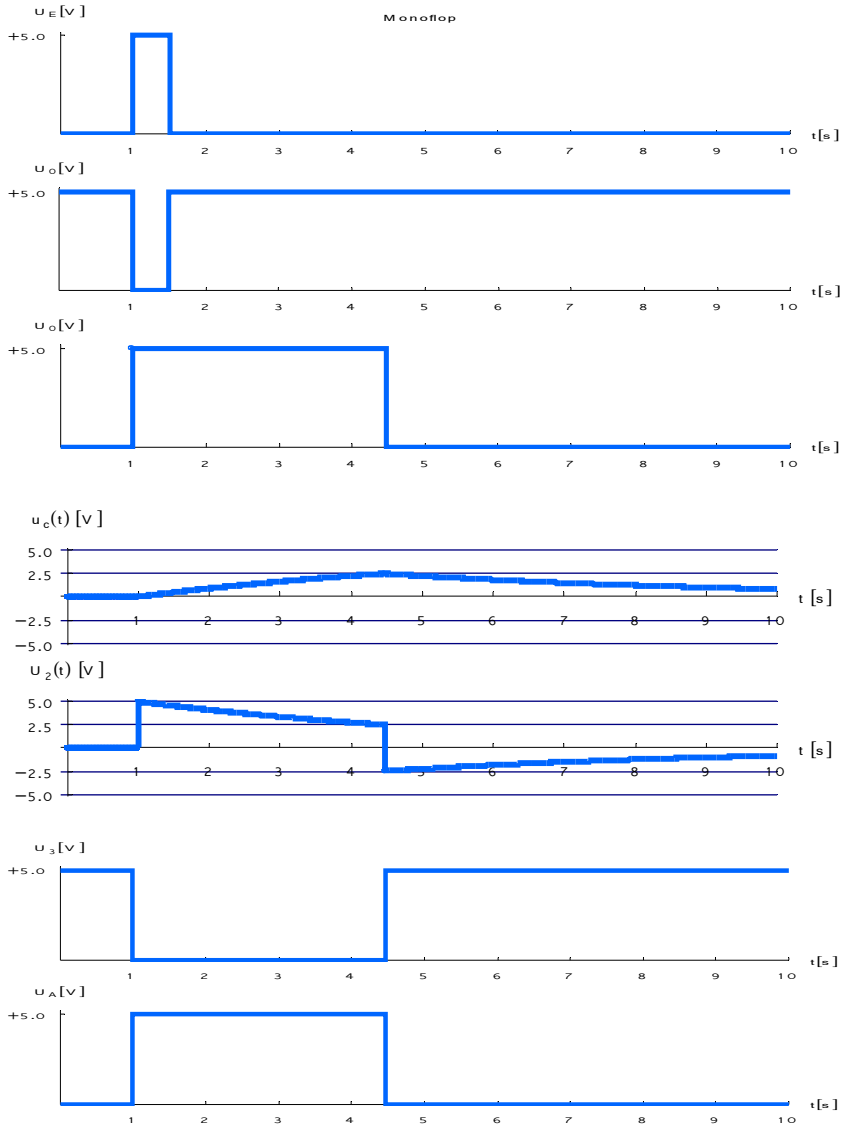
Wegen $U_1 = 0$ und $U_2 = 0$ ist auch $u_c(t) = 0$, der Kondensator hat sich entladen.

Das folgende Diagramm zeigt die zeitlichen Vorgänge für eine Zeitkonstante T_1 mit

$R = 100\,\Omega$ und $C = 0.05\,\text{Farad}$, also $T_1 = RC = 5$ [s].

Der Ablauf im Einzelnen:

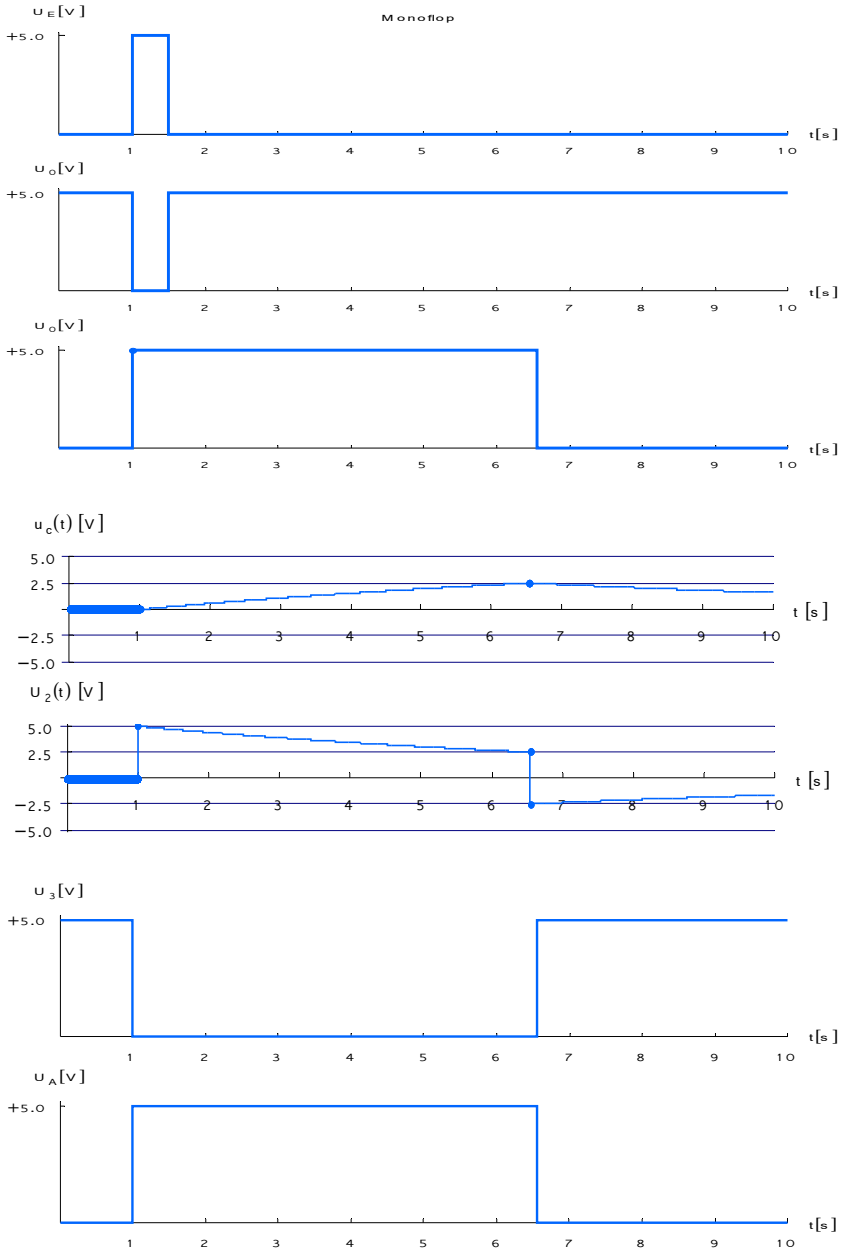
- Bei t = 1s liegt für die Dauer von 0.5 s ein positiver Spannungsimpuls U_E = 5 Volt (Trigger) am ersten NOT-Gatter.
- Dadurch springt der NAND-Ausgang U_1 von 0 auf 5 Volt.
- Da die Spannung $u_c(t)$ am Kondensator wegen der Stetigkeitsbedingung für die hierin gespeicherten Energie $W_c = \frac{1}{2} \cdot C \cdot u_c^2$ nicht springen kann, bleibt sie im ersten Moment auf $u_c = 0$ und steigt dann durch den Ladevorgang mit der Zeitkonstanten T_1 an.
- Die Eingangsspannung $u_2(t)$ des zweiten NOT-Gatters springt auf 5 Volt und klingt mit T_1 gegen 0 Volt ab, U_3 springt auf 0 Volt, U_A auf 5 Volt.
- Wenn die Spannung $u_2(t)$ am zweiten NOT-Gatter den Wert 2.5 Volt erreicht hat, springt U_3 wieder auf 5 Volt, U_A auf 0 Volt.
- Mit jedem jedem neuen Trigger U_E = 5 Volt, wiederholt sich dieser Vorgang (Hinweis: Bei den hier gemachten vereinfachenden Annahmen muss sich vor einem erneuten Trigger zunächst der Kondensator C wieder entladen haben. Das lässt sich durch eine andere Schaltungsanordnung beschleunigen).



Bei $R = 100\,\Omega$ und $C = 0.08\,\text{Farad}$, also T_1 = RC= 8 [s], wird der Impuls am Ausgang entsprechend länger wie das nächste Diagramm zeigt (aber nicht um den Faktor 8/5, warum nicht?). **Hinweis:** Betrachten Sie den funktionalen Zusammenhang zwischen der Zeitkonstanten T_1 und der Zeitspanne bis zum Erreichen der Umschaltschwelle).

Monoflops gibt es in der Ausführung als **nicht nachtriggerbar**, wobei weitere kurze Eingangsimpulse während des H-Pegels am Ausgang den Ausgangsimpuls nicht verlängern. In der **nachtriggerbaren** Ausführung verlängert sich die Dauer des H-Pegels am Ausgang jeweils um die Zeitdauer eines Ausgangsimpulses.

Eine von mehreren Anwendungen für Monoflops ist z. B. das Herstellen von Impulsfolgen mit Tastverhältnissen von 1:1 (oder beliebigen weitern) aus Impulsfolgen mit anderen Tastverhältnissen. So erzeugt ein Frequenzteiler 1:3 (siehe Kapitel 10, Seite 10-8) aus der Eingangsimpulsfolge T mit dem Tastverhältnis 1:1 eine Ausgangsimpulsfolge Q_2 mit dem Tastverhältnis 1:2. Durch Verlängerung der Ausgangsimpulse um 1/6 erhält man auch für Q_2 das gewünscht Tastverhältnis von 1:1.


Monoflop
u_E[V]
+5.0
t[s]
u_o[V]
+5.0
t[s]
u_o[V]
+5.0
t[s]
$u_c(t)$ [V]
5.0
2.5
−2.5
−5.0
t [s]
$u_2(t)$ [V]
5.0
2.5
−2.5
−5.0
t [s]
u_3[V]
+5.0
t[s]
u_A[V]
+5.0
t[s]
1 2 3 4 5 6 7 8 9 10

Astabile Kippstufen (Multivibratoren) erzeugen Taktimpulsfolgen mit definierter Frequenz und einstellbarem Tastverhältnis (Verhältnis zwischen Dauer des H- und L-Pegels), siehe Skript Kapitel 9.2.3. Sie dienen als Taktgeneratoren für Digitalschaltungen, z. B. Rechenschaltungen.

Aufgabe: Konstruieren Sie aus zwei Monoflops einen Multivibrator für die Frequenz 1 MegaHertz und stellen Sie durch entsprechende Dimensionierung der RC-Kombination das Tastverhältnis auf 2:1 ein, d. h., der H-Pegel soll doppelt so groß wie der L-Pegel sein und die Dauer für H-und L-Pegel zusammen soll 10^{-6} s betragen.

Schmitt-Trigger-Schaltungen werden zur Erzeugung von Impulsen mit steilen Flanken aus „verschliffenen“ Signalen eingesetzt.

Veranstaltung am 26.05.2010

10 Zähler, Frequenzteiler und Schieberegister (zum Skript Kapitel 10)

Asynchron arbeitende Zählschaltungen:

- Zählung im Dual-Code
- Zählung im BCD-Code
- Entwurfsverfahren für asynchron arbeitende Zählschaltungen
  - ohne direkt wirkende Rücksetzeingänge in 7 Teischritten, Beispiel auf den Seite 10-2, 10-3 und 10-4 des Skripts :
    - Anzahl erforderlicher FFs ermitteln
    - Vorschriftentabelle aufstellen (=Umkehrung der Schaltfolgetabelle)
    - Tabelle mit gewünschtem Zähler-Code und zugehöriger Schaltfolgetabelle aufstellen
    - Beschaltung der Takteingänge festlegen
    - Schaltfolgetabelle mit „don't cares“ vereinfachen
    - Logische Funktion der JK-Eingänge bestimmen
    - Schaltbild des Zählers skizzieren
  - mit direkt wirkenden Rücksetzeingängen (für Modulo N-Zähler)

Veranstaltung am 02.06.2010

Synchron arbeitende Zählschaltungen

Entwurf für Zählschaltungen ohne direkt wirkende Rücksetzeingänge:

Im Skript werden auf den Seiten 10-2 bis 10-4 am Beispiel eines BCD-Zählers mit 4 asynchron zu betreibenden FFs ausführlich die 7 Entwurfsschritte bis zur Aufstellung des Schaltbildes dargestellt.

Der Entwurf für synchron betriebene Zähler läuft gemäß Seite 10-6 genauso, nur entfallen die Schritte 4 und 5. Welches Ergebnis hat das zur Folge?

Die Schritte 1 und 2 sind identisch, aus dem Schritt 3 wird die Schaltfolgetabelle ohne die Vereinfachungen aus Schritt 5 übernommen:

Zählerstand	D	C	B	A	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
0	0	0	0	0	1	X	0	X	0	X	0	X
1	0	0	0	1	X	1	1	X	0	X	0	X
2	0	0	1	0	1	X	X	0	0	X	0	X
3	0	0	1	1	X	1	X	1	1	X	0	X
4	0	1	0	0	1	X	0	X	X	0	0	X
5	0	1	0	1	X	1	1	X	X	0	0	X
6	0	1	1	0	1	X	X	0	X	0	0	X
7	0	1	1	1	X	1	X	1	X	1	1	X
8	1	0	0	0	1	X	0	X	0	X	X	0
9	1	0	0	1	X	1	0	X	0	X	X	1
0	0	0	0	0								

Nun kann in Schritt 6 für jeden der 8 Eingänge die anzuwendende logische Funktion als Verknüpfungsergebnis der Ausgänge ermittelt werden. Die Eingänge J_A und K_A lassen sich bei Einbeziehung der „don't cares“ konstant auf „1“ legen, also $J_A = 1$, $K_A = 1$. Für die übrigen Eingänge trifft das im Gegensatz zu einem Teil bei der asynchronen Ausführung hier aber nicht zu.

Für J_B gilt mit der DNF: $J_B = \overline{D}\,\overline{C}\,\overline{B}\,A \vee \overline{D}\,C\,\overline{B}\,A$

Ohne die „don't cares“ (= X) lässt sich dieser Ausdruck sofort auf $J_B = \overline{D}\,\overline{B}\,A$ verinfachen (Distributivgesetz!). Bei Einbeziehung der „don't cares“ ist eine weitere Vereinfachung möglich. Hier hilft wieder das KV-Diagramm:

	A		$\overline{A}$		
B	X	X		X	$\overline{D}$
					D
$\overline{B}$					
	1	1	X		$\overline{D}$
	$\overline{C}$	C		$\overline{C}$	

Für eine Vereinfachung sind nur die beiden linken oberen „don't cares“ geeignet. Zusammen mit den beiden unteren „1“-Elementen wird C und B eliminiert (Distributivgesetz) und es bleibt

$$J_B = \overline{D}\,A \quad .$$

Die Funktionen für die restlichen 5 Eingangsvariablen werden entsprechend ermittelt. **Zur Erinnerung**: Die DNF lässt sich immer mit Vorteil verwenden, wenn die Anzahl der „1“-Elemente geringer oder höchstens gleich der Anzahl der „0“-Elemente ist, da dann die Anzahl der Minterme in der DNF klein bleibt. Im anderen Fall ist die KNF günstiger, oder man stellt die DNF für die negierte Variable auf.

Der Zählvorgang lässt sich bei asynchron arbeitenden Zähler durch Verwendung der negierten FF-Ausgänge als Takt für die Folge-FlipFlops auch **rückwärts** (= abwärts) gestalten. Durch Zusatzbeschaltungen kann die Zählrichtung gewählt werden, siehe Skript, Kapitel 10.2.3.3.

Frequenzteiler (Skript Kapitel 10.2)

Frequenzteiler teilen die Frequenz einer Eingangstaktimpuls-Folge in einem gewünschten ganzzahligen Verhältnis zur Ausgangsfolge herunter. Es gibt verschiedene Ausführungen:

- Die Frequenz des Ausgangssignals der höchstwertigen Stelle eines Modulo N-Dualzählers ist im Verhältnis 2^N:1 geteilt.
- Durch Rückkopplung der Ausgangssignale lassen sich Teiler mit ungeradzahligen Teilerverhältnissen aufbauen.
- Reihenschaltungen ergeben Teilerverhältnisse als Produkt der Einzel-Teilerverhältnisse, z. B. zur Erzielung sehr großer Teilungen mit Timern von Mikrocontrollern.
- Die Einbettung von k:1-Teilern in 3:1-Teiler ergibt ein Teilerverhältnis von (2k+1):1.
- Teiler mit direkt wirkenden Setzeingängen. Sie wirken wie Teiler auf Modulo-N-Dualzählerbasis (s. o.). Während dort jedoch die Rücksetzeingänge **aller** FFs beschaltet werden müssen, nutzt man hier die „1“-Zustände der bereits gesetzten FFs aus und muss dann nur noch diejenigen FFs setzen, die sich aktuell im „0“-Zustand befinden, siehe Beispiel im Skript, Kapitel 10.2.3.3. Der Vorteil liegt u. a. darin, dass in der Schaltung weniger Verbindungsleitungen ausgeführt werden müssen.

Register (Skript Kapitel 10.3)

Mit FFs lassen sich verschiedene Schaltungstypen zur Speicherung von Datenbits aufbauen:

- Auffangregister
- Schieberegister als
  - Serien-Parallel-Umsetzer
  - Parallel-Serien-Umsetzer

Veranstaltung am 09.06.2010

11 Digitale Auswahlschaltungen (Skript Kapitel 11)

Digitale Auswahlschaltungen gibt es als

- Multiplexer
- Demultiplexer
- Adressdecodierer
- Komparatoren

Multiplexer schalten adressierbar je eines von N digitalen Ausgangssignalen auf einen Eingangskanal, z. B. bei der Abfrage mehrerer Messstellen, deren Ergebnisse einem Anwendungsprogramm zur Weiterverarbeitung übergeben werden sollen.

Demultiplexer schalten ein Aussgangssignal adressierbar auf N Eingangskanäle, z. B. um die Ausgaben von N Messstellen über eine serielle Leitung zu übertragen und am Zielort wieder auf die N Kanäle aufzuspalten.

Adressdecodierer schalten von N Speicherregistern adressierbar genau eines zur Übergabe neu-

er oder zum Abruf bestehender Speicherinhalte frei – und sperren zugleich dies restlichen N-1.

Digitale Komparatoren vergleichen zwei Zahlen in Dual- oder anderer Darstellung und geben am Ausgang an, ob die eine Zahl größer, gleich oder kleiner ist als die andere. Ein Grundbaustein ist gemäß Skript Kapitel 11.4 der 1-Bit-Komparator, aus dem sich Komparatoren für mehrstelligen Zahlen zusammenfügen lassen.

Hinweis zum Beispiel im Skript auf Seite 11-3: Das Ausgangssignal Y für Gleichheit beider Eingangsbits hat die Funktion

$$Y = \overline{A}\,\overline{B} \vee AB \quad .$$

Erweitert man diese ODER-Verknüpfung durch die Terme $A\overline{A} = 0$ und $B\overline{B} = 0$, so erhält man

$$Y = A\overline{A} \vee \overline{A}\,\overline{B} \vee AB \vee B\overline{B} \quad .$$

Mit Hilfe des Distributivgesetzes kann man diesen Ausdruck wie bereits im Skript dargestellt auch in der Form

$$Y = (A \vee \overline{B}) \wedge (\overline{A} \vee B)$$

schreiben. Doppelte Negation und zweimalige Anwendung der DeMorganschen Gesetze ergibt

$$Y = \overline{\overline{(A \vee \overline{B}) \wedge (\overline{A} \vee B)}} = \overline{(A \vee \overline{B}) \vee \overline{\overline{A} \vee B}} = \overline{\overline{A}\,B \vee A\overline{B}} = \overline{Z \vee X} \quad .$$

Damit lässt sich die im Skript skizzierte, sehr kompakte Schaltung aufbauen.

Veranstaltung am 16.06.2010

12 D/A- und A/D-Wandler (Skript Kapitel 12)

Wirkungsweise

Abtasttheorem

(wird ergänzt)