

Digitaltechnik

Grundlagen

HTT-Biel/Bienne

(Version v2.1d)

Marcel Jacomet

Inhaltsverzeichnis

0.1	Danksagung	1
1	Einführung	2
1.1	Ziel und Aufbau	2
1.2	Aufbau	2
1.3	Literatur und Links	3
1.4	Was ist Digitaltechnik?	4
1.5	Analog versus Digital	4
1.6	Vorteile der Digitaltechnik	4
2	Zahlen	6
2.1	Zahlensysteme	6
2.2	Umwandlung von und in Dezimalzahlen	7
2.3	Fest- und Gleitkommadarstellung von Zahlen	10
2.4	Darstellung positiver und negativer Zahlen	10
2.5	Dualarithmetik	11
3	Codierungen	13
3.1	Lernziele	13
3.2	Einführung	13
3.3	Binär-Dezimal Codes	14
3.4	Einschritt-Codes	16
3.5	Fehlererkennung	18
3.6	Fehlerkorrigierbare Codes	20
3.7	Alphanumerische Codes	23
4	Boolesche Gleichungen	25
4.1	Lernziele	25
4.2	Boolesche Konstanten und Variablen	26
4.3	Grundfunktionen der Booleschen Algebra	26

4.4	Spezielle Funktionen der Booleschen Algebra	28
4.5	Analyse logischer Schaltungen	28
4.6	Disjunktive und Konjunktive Normalform	30
4.7	Rechenregeln der Booleschen Algebra	30
4.8	Vereinfachung und Optimierung von Funktionen	32
4.8.1	Vereinfachung mit dem Karnaugh-Diagramm	33
4.8.2	Unvollständig definierte Funktionen	35
4.8.3	Vereinfachung nach Quine und McCluskey	35

5 Kipperschaltungen 40

5.1	Lernziele	40
5.2	Einführung	41
5.3	Elementare bistabile Kippstufen (Latch)	41
5.3.1	Latch mit asynchronen Eingängen (Set-Reset Latch)	41
5.3.2	Latch mit synchronen Eingängen	43
5.4	Zeitverhalten von Latches mit synchronen Eingängen	44
5.5	Pulsteuerung und Flankensteuerung	45
5.6	Flip-Flop's (bistabile Kippstufen)	46
5.6.1	SR-Flip-Flop	46
5.6.2	D-Flip-Flop	48
5.6.3	JK-Flip-Flop	49
5.7	Asynchrone Eingänge von Flip-Flop's	50
5.8	Monoflops (monostabile Kippstufen)	52
5.9	Multivibrator (astabile Kippstufe)	53
5.10	Hazards	55
5.10.1	Entstehung von Hazards	55
5.10.2	Vermidung von Hazards	57

6 Zustandsmaschinen 59

6.1	Lernziele	59
6.2	Grundlegende Sequenzer Strukturen	60
6.3	Beschreibung von Zustandsmaschinen	61
6.3.1	Übergangs- und Ausgangstabelle	62
6.3.2	Zustandsdiagramm	63
6.4	Analyse von Sequenzerschaltungen	64
6.5	Synthese von Sequenzerschaltungen	66
6.5.1	Vollständigkeit und Konsistenz	67
6.5.2	Parasitäre Zustände	68
6.5.3	Hazards in Zustandsmaschinen	70
6.5.4	Einsynchronisation	70

6.5.5 Entwurfsbeispiel: Lichtsignalsteuerung	71
--	----

Abbildungsverzeichnis

3.1	Abstastfehler bei gewöhnlicher binärer Codierung (kein Einschrittcodierung).....	17
3.2	Einschritt-Code.....	17
3.3	Paritätsbit für Fehlererkennung.....	19
4.1	Konjunktion: Logisches AND Gatter.....	27
4.2	Disjunktion: Logisches OR Gatter.....	28
4.3	Komplement: Logisches NOT Gatter.....	28
4.4	Äquivalenz: Identität bei mehreren Eingängen, (logisches exclusive NOR Gatter bei zwei Eingängen).....	29
4.5	Antivalenz: Logisches exclusive OR Gatter.....	29
4.6	Beispiel einer logischen Schaltung.....	30
4.7	Disjunktive Normalform (links) und konjunktive Normal- form (rechts).....	31
4.8	Zweistufige Logik.....	31
4.9	DeMorgan's Gesetze.....	32
4.10	Beispiel zur Anwendung von DeMorgan's Gesetze.....	33
4.11	Karnaugh-Tabelle.....	34
4.12	Vereinfachung bei zwei Eingangsvariablen.....	34
4.13	Vereinfachung bei drei Eingangsvariablen.....	35
4.14	Vereinfachung bei vier Eingangsvariablen.....	35
4.15	Vereinfachung bei fünf Eingangsvariablen.....	36
4.16	Schema zum Beispiel von fünf (vier) Eingangsvariablen.....	36
4.17	Schachbrettartiges Muster im Karnaugh-Diagramm führt zu EXOR Verknüpfung.....	37
4.18	Vereinfachung am Beispiel mit don't cares.....	37
5.1	Speicher für 1 Bit.....	41
5.2	Wahrheitstabelle des SR-Latches.....	42

5.3	Schema eines NAND und NNOR SR-Latch's.	42
5.4	Zeitverhalten des NOR-NOR SR-Latch's.	43
5.5	Anwendungsbeispiel prellfreier Schalter.	43
5.6	SR-Latch mit Kontroll-Eingang.	44
5.7	D-Latch.	44
5.8	Charakteristisches Zeitverhalten eines Latch's.	44
5.9	Gegenüberstellung von Pulssteuerung und Flankensteuerung.	46
5.10	Prinzipschema und Symbol des SR-Master-Slave Flip-Flop's.	47
5.11	Impulsdiagramm des SR-Master-Slave Flip-Flop's.	47
5.12	Master-Slave Implementation des flankengesteuerten D-Flip-Flop's.	48
5.13	Einspeicher Implementation des Flip-Flop's.	48
5.14	Herleitung der Bedingung für den S Eingang.	50
5.15	JK-Flip-Flop aufgebaut aus SR-Flip-Flop.	50
5.16	Symbol eines D-Flip-Flop's mit asynchronen Setz- und Rücksetzeingängen.	51
5.17	Implementation eines SN7474 Flip-Flop's.	51
5.18	Symbole einer flankengertigerten monostabilen Kippstufe.	52
5.19	Zeidiagramm von nachtriggerebaren und nicht nachtriggerebaren Monoflops.	53
5.20	Symbol eines Multivibrators.	53
5.21	Spannungsverlauf des Schmitt-Triggers.	54
5.22	Zeidiagramm eines einfachen Multivibrators.	54
5.23	Reales logisches Gatter mit Verzögerung.	55
5.24	Gegenüberstellung statischer und dynamischer Hazards.	56
5.25	Schaltung mit statischem Hazard an Signal s	56
5.26	Entstehung eines Hazards.	57
6.1	Struktur des Mealy-Automaten.	60
6.2	Struktur des Moore-Automaten.	61
6.3	Der Medwedjew-Automat ist ein Spezialfall des Moore-Automaten.	62
6.4	Beispiel eines Zustandsdiagramms.	64
6.5	Zustandsmaschine mit D-Flip-Flop's.	65
6.6	Zustandsdiagramm des Automaten in Abbildung 6.5.	66
6.7	Inkonsistentes Zustandsdiagramm.	67
6.8	Unvollständiges Zustandsdiagramm.	68

6.9 Zustandsdiagramm mit Schatten-Zustandsdiagramm aus parasitären Zuständen.	68
6.10 Die parasitären Zustände aus Abbildung 6.9 sind ins Zu- standsdiagramm eingebettet.	69
6.11 Einsynchronisation in Moore-Automaten.	71
6.12 Lichtsignalsteuerung mit Haupt- und Nebenstrasse.	71
6.13 Zustandsdiagramm der Lichtsignalsteuerung.	75
6.14 Schema der Lichtsignalsteuerung.	76

Tabellenverzeichnis

2.1	Darstellung positiver und negativer Zahlen	11
3.1	Beispiel für Codes: duale, oktale, dezimale und hexade- zimale Darstellung	15
3.2	Gebäuchliche 4-stelligen BCD Codes	16
3.3	Gray-Code	18
3.4	Paritätsbit für gerade Parität	19
3.5	2 aus 5 Code	20
3.6	Code mit Fehler. Wo ist der Fehler?	21
3.7	Hamming Code	22
3.8	7 Bit ASCII Code	24
4.1	Wahrheitstabelle	26
4.2	Rechenregeln der Booleschen Algebra	32
4.3	Vereinfachungen nach Quine und McCluskey.	38
4.4	Minterm-Printern Tabelle.	38
5.1	Wahrheitstabelle des JK-Flip-Flop's.	49
6.1	Beispiel einer Übergangstabelle.	63
6.2	Übergangstabelle der Zustandsmaschine aus Abbildung 6.5.	66
6.3	Übergangstabelle für die Lichtsignalsteuerung.	72
6.4	Ausgangstabelle der Lichtsignalsteuerung.	73

0.1 Danksagung

Der Digitaltechnik Kurs wurde an der HTI-Biel von Marcel Jacomet aufgebaut und erstmals 1993 erteilt. Zwischenzeitlich haben Michael Höckel und Michael Filioi den Kurs verwendet und die Übersetzung ins französische vorgenommen, wofür ich mich bei ihnen bedanken möchte. Ebenso bedanken möchte ich mich bei den Studenten und Herrn Ernst Schwab für die zahlreichen Korrekturhinweise.

Kapitel 1

Einführung

1.1 Ziel und Aufbau

Dieser Kurs vermittelt eine Einführung in die Grundlagen der Digitaltechnik ausgehend von kombinatorischen Schaltungen bis zu endlichen Automaten (Sequenzzer oder auch Zustandsmaschinen genannt). Die Kursbesucher sollen in der Lage sein, einfache Digitalschaltungen selbständig analysieren und systematisch synthetisieren zu können.

Der Kurs bildet die Grundlage für weitere Vorlesungen wie, Digitaltechnik Labor, Mikroelektronik Kurs (VLSI-Design), RISC Prozessor Design, Digitale Signalverarbeitung (DSP), etc.

Im Selbststudium werden vorgegebene Stoffteilgebiete anhand von Studienunterlagen und Fachliteratur erarbeitet und Übungsaufgaben selbständig gelöst.

1.2 Aufbau

Folgende Kapitel werden in dieser Vorlesung behandelt:

1. Einführung

2. Zahlen: Zahlensysteme, Umwandlung von und in Dezimalzahlen, Dualarithmetik, Addition, Subtraktion, Division, Multiplikation.
3. Codes: Binär-Decimal Codes, Einschnitt-Codes, Codes für Fehlererkennung, Codes für Fehlerkorrektur (Hamming), Alphanumerische Codes.
4. Boolesche Algebra und Logische Gatter: Boolesche Konstanten und Variablen, Grundfunktionen, Spezielle Funktionen, Analyse logischer Schaltungen, Disjunktive und Konjunktive Normalform, Vereinfachung und Optimierung von Funktionen nach Karnaugh und nach Quine/McCluskey Methode.
5. Kipperschaltungen: Elementare bistabile Kippstufen, Latch mit synchron und asynchronen Eingängen, Zeitverhalten, Pulststeuerung und Flankensteuerung, Flip-Flop's (bistabile Kippstufen), Monoflops (monostabile Kippstufen), Multivibrator (astabile Kippstufen), Entstehung und Vermeidung von Hazards.
6. Zustandsmaschinen: Grundlegende Sequenzer Strukturen (Mealy, Moore, Medwedjew Automaten), Übergangstabelle, Ausgangstabelle, Zustandsdiagramm, Analyse von Sequenzerschaltungen, Synthese von Sequenzerschaltungen, Vollständigkeit und Konsistenz, parasitäre Zustände, Hazards in Sequenzern, Einsynchronisation.
7. Fuzzy-Logik (optional): Geschichtlicher Hintergrund, Philosophie und Eigenschaften von Fuzzy-Logik Produkten, Fuzzy-Set Theorie, Unscharfe Mengen, Linguistische Variablen, Fuzzy-Operatoren, Arbeitsweise eines Fuzzy-Logik Systems, Fuzzyifizierung, Inferenzmaschine und Wissensbasis, Defuzzyifizierung, Anwendungsbeispiele aus der Schweizer Industrie, Einführungsstrategie in Industrie und Zukunftsperspektiven.

1.3 Literatur und Links

Auf der Web Seite des MicroLab (Mikroelektronik Labor) sind sämtliche Kursunterlagen zu finden: www.microlab.ch. Die zur Digitaltechnik empfohlene Literatur ist am Schluss des Scripts aufgeführt .

1.4 Was ist Digitaltechnik?

Digitaltechnik ist diejenige Technik, die sich mit der ziffermässigen Darstellung irgendwelcher Grössen befasst.

Begriffe:

- *digit*: Ziffer, Stelle
- *numérique*: mit Zahlen oder Ziffern

Anwendungsgebiete:

- Informationsverarbeitung (Informatik), Datenverarbeitung
- Digitale Signalverarbeitung
- Digitale Steuerungs- und Regelungstechnik
- Digitale Messtechnik
- Digitale Übertragungstechnik

1.5 Analog versus Digital

Analoge Schaltungen und Systeme verarbeiten zeitvarierende Signale, die jeden Wert in einem kontinuierlichen Bereich von Spannung, Strom oder einer anderen Grösse annehmen können. Digitale Systeme kennen zu jeder Zeit nur zwei diskrete Zustände die wir logisch 1 und 0 nennen (oder LOW und HIGH, respektive FALSE und TRUE).

1.6 Vorteile der Digitaltechnik

- Reproduzierbarkeit: Analoge Schaltungen hängen von inneren und äusseren Störeinflüssen wie Temperatur, Alterung, Rauschen etc. ab.

- Einfaches Design: Schaltungen lassen sich einfach aufteilen, wiederverwenden und skalieren (IC-Design).
- Einfache Speicherung und Weiterverarbeitung ohne Genauigkeitsverlust.
- Leistungselektronik: Als Verstärker reicht ein gesteuerter Schalter (einfacher Aufbau, kleine Verluste).
- Flexibilität und Funktionalität:
 - Zum Verknüpfen, Vergleichen und Sortieren verwendet man einfache logische Schaltungen”.
 - Mathematische Probleme werden numerisch mit den vier Grundrechenarten, vornehmlich durch Addition und Multiplikation gelöst.
 - Bestimmte Probleme in der Signal- und Informationsverarbeitung können nur mit digitalen Ansätzen gelöst werden.
- Speicher für lange Speicherzeiten und schnellen Zugriff sind einfach zu realisieren.
- Programmierbarkeit: Probleme können mit der gleichen Schaltungsanordnung gelöst werden, welche nur anders aufbereitet, respektive programmiert werden.
- Die Genauigkeit des Systems kann durch Erhöhen der Stellenzahl beliebig erweitert werden.
- Einfachere Übertragung auf neue (Chip-) Technologien.

Kapitel 2

Zahlen

2.1 Zahlensysteme

- Strichlisten (Basis 1): für Weiterverarbeitung nicht sinnvoll, da sehr umständlich bei grossen Zahlen.
- Römische Zahlen: CCCIC = 399, CD = 400. Hier haben die Stellen einer Zahl keine feste Wertigkeit, benachbarte Zahlen können sich in der Stellenzahl stark unterscheiden.

- Dezimalzahlen (Stellenwertsysteme oder polyaedrische Systeme):

$$\begin{aligned} 3970.25 &= 3 \cdot 1000 + 9 \cdot 100 + 7 \cdot 10 + 2 \cdot 0.1 + 5 \cdot 0.01 \\ &= 3 \cdot 10^3 + 9 \cdot 10^2 + 7 \cdot 10^1 + 0 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2} \end{aligned}$$

Summe aus Produkten von Koeffizienten zwischen 0 und 9 und Zehnerpotenzen (eventuell in abgekürzter Schreibweise).

Allgemeine Form von polyaedrischen Zahlensystemen zur Basis B , wobei jeder Stelle ein fester Wert zugeordnet ist:

$$Z = c_{n-1} \cdot B^{n-1} + c_{n-2} \cdot B^{n-2} + \dots + c_1 \cdot B^1 + c_0 \cdot B^0 + \dots + c_{-1} \cdot B^{-1} + c_{-(m-1)} \cdot B^{-(m-1)} + c_{-m} \cdot B^{-m}$$

$$Z = \sum_{i=-m}^{n-1} c_i B^i$$

Es gilt: $0 \leq c_i \leq B - 1$ mit n als Stellenzahl links vor dem Komma, B als Basis des Zahlensystems, c_i als Koeffizient und der Ordnungszahl i .

In der digitalen Verarbeitungstechnik wird das duale Zahlensystem mit der Basis 2 sowie den Ziffern 0 und 1 verwendet. Schaltungen zur Verarbeitung von Dualzahlen lassen sich leicht entwerfen, wie später gezeigt wird.

2.2 Umwandlung von und in Dezimalzahlen

Eine polyaedische Zahl wird üblicherweise in einer gekürzten Schreibweise dargestellt:

$$\begin{aligned} [c_{n-1}c_{n-2} \cdots c_1c_0c_{-1} \cdots c_{-(m-1)}c_{-m}] = \\ [(((c_{n-1}B + c_{n-2})B + \cdots)B + c_0)] + \\ [((((c_{-m}/B + c_{-(m-1)})/B + \cdots)/B + c_{-1})/B)] \end{aligned}$$

Umwandlung des ganzzahligen Anteils von Zahlen beliebiger Basis in Dezimalzahlen:

c_{n-1}	$=$	S_1
$S_1B + c_{n-2}$	$=$	S_2
$S_2B + c_{n-3}$	$=$	S_3
$\vdots$		
$S_{n-2}B + c_1$	$=$	S_{n-1}
$S_{n-1}B + c_0$	$=$	S_n

Umwandlung des gebrochenen Anteils von Zahlen beliebiger Basis in Dezimalzahlen:

c_{-m}	$=$	s_1
$s_1/B + c_{-(m-1)}$	$=$	s_2
$s_2/B + c_{-(m-2)}$	$=$	s_3
$\vdots$		
$s_{m-1}/B + c_{-1}$	$=$	s_m
s_m/B	$=$	s_{m+1}

Die resultierende Dezimalzahl entspricht der Summe von S_i und s_i .

Beispiele zur Umwandlung in Dezimalzahlen:

- Die Dualzahl 111000.011_{bin} ist in eine Dezimalzahl umzuwandeln.
Lösung: 53.375_{dec}
- Die Oktalzahl (Basis 8) 111000.011_{oct} ist in eine Dezimalzahl umzuwandeln.
Lösung: 37376.0175781_{dec}

Für die Umwandlung aus Dezimalzahlen kann man ähnliche Regeln herleiten. Für den ganzzahligen Anteil (links vor dem Komma) aus:

$$Z = c_{n-1} \cdot B^{n-1} + c_{n-2} \cdot B^{n-2} + \dots + c_1 \cdot B^1 + c_0 \cdot B^0$$

ergibt sich aus Division durch B:

$$Z/B = c_{n-1} \cdot B^{n-2} + c_{n-2} \cdot B^{n-3} + \dots + c_1 \cdot B^0 + \text{Rest } c_0$$

Nach n Divisionen findet man den Koeffizienten c_{n-1} . Für den gebrochenen Anteil (rechts vor dem Komma) aus:

$$z = c_{-1} \cdot B^{-1} + c_{-2} \cdot B^{-2} + \dots + c_{-(m-1)} \cdot B^{-(m-1)} + c_{-m} \cdot B^{-m}$$

ergibt sich durch Multiplikation mit B :

$$zB = c_{-1} + c_{-2} \cdot B^{-1} + \dots + c_{-(m-1)} \cdot B^{-(m-2)} + c_{-m} \cdot B^{-(m-1)}$$

Nach n Multiplikationen ist der Koeffizient c_{-m} isoliert.

Umwandlung ganzer Dezimalzahlen in Zahlen beliebiger Basis B :

Z/B	$=$	S_1	Rest	c_0
S_1/B	$=$	S_2	Rest	c_1
S_2/B	$=$	S_3	Rest	c_2
$\vdots$				
S_{n-2}/B	$=$	S_{n-1}	Rest	c_{n-2}
S_{n-1}/B	$=$	0	Rest	c_{n-1}

Umwandlung echter Dezimalbrüche in Zahlen beliebiger Basis B :

$Z \cdot B$	$=$	$s_1 + c_{-1}$
$s_1 \cdot B$	$=$	$s_2 + c_{-2}$
$s_2 \cdot B$	$=$	$s_3 + c_{-3}$
$\vdots$		
$s_{m-1} \cdot B$	$=$	$s_m + c_{-(m-1)}$
$s_m \cdot B$	$=$	$s_{m+1} + c_{-m}$

Beispiel zur Umwandlung aus einer Dezimalzahl. Die Dezimalzahl 109.78125_{dec} ist:

- a) in eine Dualzahl umzuwandeln:
Lösung: 1101101.11001_{bin}
- b) in eine Oktalzahl umzuwandeln:
Lösung: 155.62_{oct}

2.3 Fest- und Gleitkommadarstellung von Zahlen

Bei der Festkommadarstellung steht das Komma an einer genau festgelegten Stelle, beispielsweise im Rechnungswesen mit Franken und Rappen 246.23 sFr. Bei der Gleitkommadarstellung werden Zahlen durch Mantisse und Exponenten dargestellt (grosse Zahlenbereiche / wissenschaftliche Berechnungen). Überschlüssige Stellen werden bei niedrigen Wertigkeiten unterdrückt. Beispiel für Fortran/Basic-Darstellung mit 8 Mantissenstellen:

$$\begin{aligned} 43.5 &= 0.435 \cdot 10^2 \\ &= 0.43500000E+2 \\ 156.378 &= \\ 0.00032703 &= \\ 237438965.17 &= \end{aligned}$$

2.4 Darstellung positiver und negativer Zahlen

Vorzeichen-Betrags-Darstellung: In der Digitaltechnik wird das + Zeichen durch eine binäre 0, das – Zeichen durch binär 1 dargestellt. Das Vorzeichenbit steht in der Regel links von den Betragsbits.

Komplement-Darstellung: Einer-Komplement durch Inversion der einzelnen Bits. Zweier-Komplement = Einer-Komplement plus Addition einer 1 in der niedrigsten Stelle.

Offset-binäre Darstellung: Wird häufig verwendet in Analog-Digital und Digital-Analog Wandler. Negativste Zahl wird durch das niedrigste Bitmuster (00...00) dargestellt, positive Zahlen beginnen allgemein mit 1, negative mit 0. Dies entspricht bis auf das Vorzeichenbit dem Zweier-Komplement.

Dezimalzahl	Vorzeichen Betrags- Darstellung	Einer- Komplement Darstellung	Zweier Komplement Darstellung	Offset- Binär Darstellung
+7	0111	0111	0111	1111
+6	0110	0110	0110	1110
+5	0101	0101	0101	1101
+4	0100	0100	0100	1100
+3	0011	0011	0011	1011
+2	0010	0010	0010	1010
+1	0001	0001	0001	1001
+0	0000	0000	0000	1000
-0	1000	1111	0000	1000
-1	1001	1110	1111	0111
-2	1010	1101	1110	0110
-3	1011	1100	1101	0101
-4	1100	1011	1100	0100
-5	1101	1010	1011	0011
-6	1110	1001	1010	0010
-7	1111	1000	1001	0001
-8			1000	0000

Tabelle 2.1: Darstellung positiver und negativer Zahlen

2.5 Dualarithmetik

Addition: Wie im Dezimalsystem !?: $1 + 1 = 10$

Subtraktion: Indirekt: $D = M - S = M + (B^n - S - B^n) = M + K - B^n$ Die Differenz D erhält man, indem man zum Minuenden M das Komplement $K = B^n - S$ des Subtrahenden S zu B^n addiert und anschließend B^n wieder subtrahiert (mit n gleich der maximalen Stellenzahl links vom Komma). Das Komplement K erhält man, indem man B^n um eine Wertigkeit verringert, die nun unproblematische Subtraktion durchführt und anschließend den Wert wieder addiert (Komplementieren). Für die Subtraktion von B^n muss man zwei Fälle unterscheiden:

- $M > S$, so muss man lediglich die 1 an der Position $n + 1$ weglassen.
- $M < S$ ist die Differenz negativ (an der Stelle $n + 1$ ist eine

0), so wird $M + K - B^n$ durch $-(B^n - M - K) = -(B^n - (M + K))$ ersetzt. Dies erfordert ein zweites Komplementieren (Rückkomplementieren).

Beispiele:

- Die Differenz $123 - 11$ ist im Dezimal- und im Dualsystem über die Addition des Komplements zu berechnen.
Lösung: $112_{dec}; 111000_{bin}$
- Die Differenz $52 - 417$ ist im Dezimal- und im Dualsystem über die Addition des Komplements zu berechnen.
Lösung: $-365_{dec}; -101101101_{bin}$

Multiplikation: Wie im Dezimalsystem nur im Dualen viel einfacher ($1 \cdot 1 = 1$).

Beispiel: Das Produkt $217 \cdot 35$ ist im Dualen zu berechnen:

Lösung: 1110110101011_{bin}

Division: Fortlaufende Subtraktion wird durch die Addition des Komplements ersetzt. Der jeweilige Divisor wird dabei immer zur nächsthöheren Basispotenz ergänzt. Ergibt eine Addition des Komplements keinen Übertrag, so ist das entsprechende Teilergebnis 0 und es muss danach mit dem alten Minuenden ergänzt, durch die nächste Stelle des Dividenden, weitergerechnet werden!

Beispiel: Der Quotient $667/29$ ist im dualen zu berechnen. Es gilt $667_{dec} = 1010011011_{bin}$ und $29_{dec} = 11101_{bin}$.

Lösung: Als erstes wird das Komplement des Divisors zur nächsthöheren Basispotenz gebildet. Damit erhält man weiter:
 $1010011011/11101 = 010111 = 10111_{bin} = 23_{dec}$

Kapitel 3

Codierungen

3.1 Lernziele

Nach dem Studium des vorliegenden Kapitels sind die folgenden Aufgaben zu lösen oder die folgenden Fragestellungen präzise zu beantworten:

- Definition des Codes ist bekannt.
- Einsatzmöglichkeiten des Einschnitt-Codes ist bekannt.
- Fehlererkennung und Fehlerkorrektur mittels Paritätsbit ist geläufig.
- Hamming-Code kann auf beliebige Wortbreite hergeleitet werden.

3.2 Einführung

Codes werden dazu verwendet, um Zahlen, Buchstaben und Zeichen in anderen Darstellungsformen zu verwenden. So repräsentieren unterschiedliche Codes die verschiedenen Darstellungsformen. Codes werden nach unterschiedlichen Gesichtspunkten entwickelt:

- geringer Speicherbedarf
- schnelle, einfache Verarbeitung
- Vermeidung von Abtastfehlern
- dc-free (Ohne Gleichstrom)
- clock recovery (Takt Rekonstruktion)
- etc...

Aufgrund der unterschiedlichen Zielsetzungen, existieren Ummengen von Codes.

Definition: Ein Code ist eine Vorschrift für die eindeutige Zuordnung zwischen zwei Mengen.

3.3 Binär-Dezimal Codes

Binär-Dezimal Codes sind Codes, welche den Dezimal-Ziffern einen Code im binären Zahlensystem zuweisen. Um die Ziffern 0 bis 9 binär zu codieren, werden 4 Bits benötigt. Hingegen können mit 4 Bits $2^4 = 16$ Zeichen dargestellt werden, was 6 mehr als benötigt ist. Um 10 Ziffern mit 4 Bits darzustellen, existieren prinzipiell die folgende Anzahl von Zuordnungsmöglichkeiten und damit auch unterschiedliche Code:

$$2^4/6! = 2.9 \cdot 10^{10}$$

Die grosse Auswahlmöglichkeit erlaubt Codes nach verschiedenen Gesichtspunkten aufzustellen, wie:

- **Bewertbarkeit:** jeder Stelle des Binärzeichens ist eine feste Wertigkeit zugeordnet.
- **Abtasticherheit:** Durch Fehler darf kein falscher Wert abgetastet werden.

Dual 2 ⁴ 2 ³ 2 ² 2 ¹ 2 ⁰				Oktal 8 ¹ 8 ⁰		Dezimal 10 ¹ 10 ⁰		Hexadezimal 16 ¹ 16 ⁰	
00000					00	00		00	
00001					01	01		01	
00010					02	02		02	
00011					03	03		03	
00100					04	04		04	
00101					05	05		05	
00110					06	06		06	
00111					07	07		07	
01000					10	08		08	
01001					11	09		09	
01010					12	10		0A	
01011					13	11		0B	
01100					14	12		0C	
01101					15	13		0D	
01110					16	14		0E	
01111					17	15		0F	
10000					20	16		10	
10001					21	17		11	
10010					22	18		12	
10011					23	19		13	
10100					24	20		14	
10101					25	21		15	
⋮					⋮	⋮		⋮	
11110					36	30		1E	
11111					37	31		1F	

Tabelle 3.1: *Beispiel für Codes: duale, oktale, dezimale und heradezimale Darstellung*

- Uebertragungssicherheit: Durch Fehler darf kein neues Zeichen entstehen.
- Rechenfähigkeit: Spezielle Codierungen erlauben gewisse Operationen rascher auszuführen.

In Code-Tabellen finden sich diverse Codes. Die nachfolgenden vier Code-Typen sind exemplarisch aufgeführt:

- Aiken, Stibitz Code: Diese Codes sind negationssymmetrisch (9er Komplement).
- 4-2-2-1 Code: Code verwendbar für schnelle Zähler.
- White Code: Dieser Code hat keine duale Wertigkeit.
- Gray, Glixon, O'Brien, reflektierter Excess3: Dies sind Einschnitt-Codes.

Name	8-4-2-1 Code	Aiken Code	unsym- metri- scher 2-4-2-1 Code	Stibitz Code Excess- 3-Code	4-2-2-1 Code	White Code	Glixon Code	O'Brien Code	reflek- tierter Excess- 3-Code
Stellen- wert	8-4-2-1	2-4-2-1	2-4-2-1		4-2-2-1	5-2-1-1			
0000	0	0	0	-	0	0	0	-	-
0001	1	1	2	-	1	1	1	0	0
0010	2	2	3	-	2	2	2	2	-
0011	3	3	4	0	3	3	3	1	-
0100	4	4	5	1	-	2	7	4	4
0101	5	-	6	2	-	3	6	-	3
0110	6	-	7	3	4	4	5	3	1
0111	7	-	8	4	5	5	4	-	2
1000	8	-	9	5	-	6	9	-	-
1001	9	-	-	6	-	7	-	9	-
1010	-	-	-	7	6	-	-	7	9
1011	-	5	-	8	7	7	-	8	-
1100	-	6	-	9	-	-	8	5	5
1101	-	7	-	-	8	8	-	6	6
1110	-	8	-	-	9	9	-	-	7
1111	-	9	9	-	-	-	-	-	8
0	0000	0000	0000	0011	0000	0000	0000	0001	0010
1	0001	0001	0001	0100	0001	0001	0001	0011	0110
2	0010	0010	0010	0101	0010	0011	0011	0010	0111
3	0011	0011	0011	0110	0011	0101	0010	0110	0101
4	0100	0100	0100	0111	0110	0110	0110	0100	0100
5	0101	1011	0101	1000	0111	1000	0111	1100	1100
6	0110	1100	0110	1001	1010	1001	0101	1110	1101
7	0111	1101	0111	1010	1011	1011	0100	1010	1111
8	1000	1110	1110	1011	1101	1101	1000	1011	1110
9	1001	1111	1111	1100	1111	1111	1000	1001	1010

Tabelle 3.2: *Gebühliche 4-stelligen BCD Codes*

3.4 Einschnitt-Codes

Mit Einschnitt-Codes lassen sich Abtastfehler vermeiden. Sollen Strecken oder Winkel codiert werden, so benutzt man dazu Code-Stäbe oder Code-Scheiben. Beim Abtasten dieser Codierrichtungen kann es bei den Übergängen zu Fehlern kommen (siehe Abbildung 3.1).

In Abbildung 3.2 ist eine Code-Scheibe dargestellt, bei welcher mit Hilfe eines Einschnitt-Codes (Gray Code) Abtastfehler vermieden werden können.

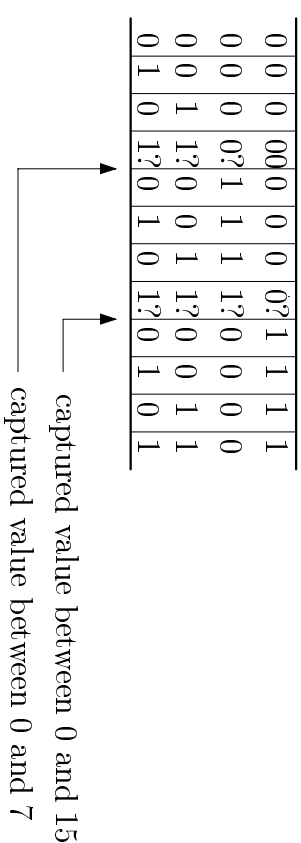


Abbildung 3.1: Abrastfehler bei gewöhnlicher binärer Codierung (kein Einschnittcode).

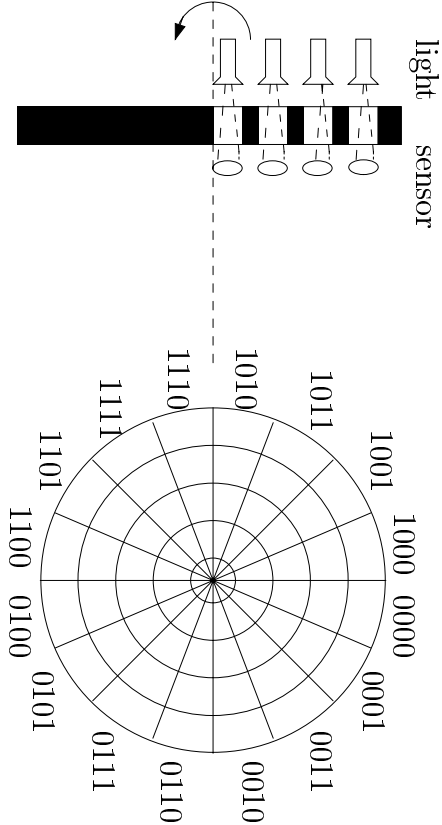


Abbildung 3.2: Einschnitt-Code

Der Gray-Code ist für binär-dezimal Codierung über mehr als eine Dezimalstelle nicht geeignet, da sich die Codes für die Zeichen 0 und 9 in drei Stellen unterscheiden.

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

Tabelle 3.3: *Gray-Code*

3.5 Fehlererkennung

Eine weiteres Kriterium für die Erzeugung eines Codes ist die Übertragungsicherheit, welche mit Redundanz erkaufte wird. Redundanz im Code ermöglicht die Fehlererkennung oder sogar die Fehlerkorrektur. Fehlererkennbare Codes sind Codes, bei welchen immer ein *einfacher Fehler*, d.h. eine Verfälschung einer Bitposition von 0 auf 1 oder umgekehrt erkennbar ist.

Paritätskontrolle: Die Paritätskontrolle ist die einfachste Art der Codesicherung. Die 1 eines Zeichens werden durch eine Zusatzstelle, dem sogenannten Paritätsbit, auf eine gerade (even) oder ungerade (odd) Abzahl 1 ergänzt. Wird ein solches Paritätsbit einem Code-Wort zugefügt, so können bei diesem Verfahren Einzelfehler erkannt werden. Beim Paritätsbit können im betrachteten Fall die folgenden Fehler erkannt werden:

- Einzelfehler werden erkannt
- Doppelfehler werden nicht erkannt

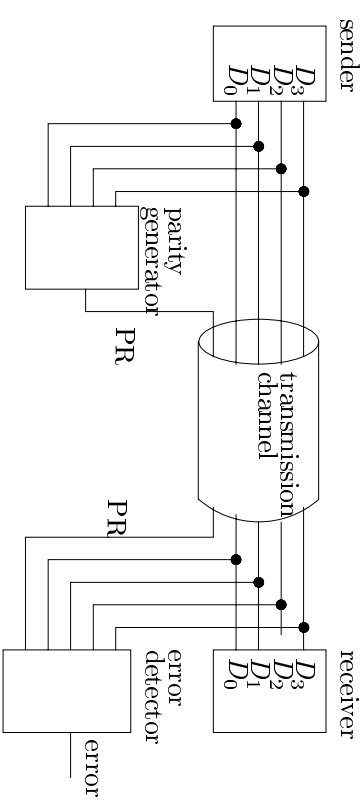


Abbildung 3.3: *Paritätsbit für Fehlererkennung*

- Dreifachfehler werden erkannt, etc

Anwendungsbeispiel für 4 bit Codierung:

Dezimalzahl	2 ³	2 ²	2 ¹	2 ⁰	PB
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	1	0	0	0	1
5	1	0	0	1	0
6	1	1	0	0	0
7	1	1	1	1	1

Tabelle 3.4: *Paritätsbit für gerade Parität*

m aus n Code: Es stellt sich die Frage, inwiefern das Paritätsbit optimal für die Fehlererkennung bei einem BCD Code ist. Es wird deshalb eine neuer Code, der *m aus n* Code definiert. Code *m aus n* führt zu *N* verschiedene Zahlen:

$$N = \binom{n}{m} = \frac{n!}{n!(n-m)} \tag{3.1}$$

Bei diesem Code sind *m* von *n* Bits immer 1 gesetzt. Die Kontrolle erfolgt sehr einfach über die Quersumme der 1, welche beim nach-

folgend aufgestellten 2 aus 5 Code inner 2 sein muss (Achtung: bei 0 stimmt der Stellenwert nicht).

Name Stellenwert	2 aus 5 Code 74210
0	11000
1	00011
2	00101
3	00110
4	01001
5	01010
6	01100
7	10001
8	10010
9	10100

Tabelle 3.5: 2 aus 5 Code

Bei diesem Code werden die folgenden Fehler erkannt:

- Einzelfehler werden erkannt
- Doppelfehler werden erkannt, falls sie den gleichen Fehlertyp aufweisen.
- Dreifachfehler werden erkannt.

Wie beim Vergleich der beiden Codierungen Paritätsbit und 2 aus 5 Code ersichtlich ist, lassen sich mit dem letzteren noch mehr Übertragungsfehler erkennen.

3.6 Fehlerkorrigierbare Codes

Einführung: Für die Korrektur von Bit-Fehlern gibt es verschiedene Möglichkeiten:

- Rückfrageverfahren, falls die Quersummenprüfung einen Fehler ergeben hat.
- Blockverfahren: Zusätzlich zum Paritätsbit verwendet man ein zusätzliches Prüfwort nach einem Zahlenblock. Beispiel Aiken-Code a) ein Block ohne Fehler b) mit einem Fehler

c) mit zwei Fehlern in einer einzigen Zeile d) mit zwei Fehlern in verschiedenen Zeilen und Spalten (zwei gleichzeitige Fehler können nicht mehr korrigiert, wohl aber erkannt werden). Hamming Code (oder andere Fehlerkorrektur-Codes, wie Reed Solomon).

Hamming Code:

- In jedem einzelnen Zeichen wird ein Fehler erkannt und korrigiert.
- Will man bei einem einzelnen Informationsbit einen Fehler korrigieren können, muss man das Paritätsbit durch ein zweites Paritätsbit überprüfen (2 aus 3). Bsp: 111 (Informationsbit, 2. Prüfbit, 1. Prüfbit); Wo liegt jeweils der Fehler: 110, 101, 011

Dies bedeutet, wenn Fehler korrigiert werden sollen, muss die Redundanz verglichen mit der Fehlererkennung erhöht werden.

Anhand eines 2-Bit breiten Informationswortes soll die Fehlerkorrektur mittels Paritätsbits ermittelt werden. Da ein einzelnes Paritätsbit dazu nicht ausreicht, ist ein Versuch mit 2 Paritätsbits vorzunehmen:

m_1		m_0	k_1	k_0
x	x	x	x	x
x	x	x	x	x
0	0	0	0	0
0	1	1	1	1
1	0	1	1	0
1	1	1	0	1
1	1	1	1	1

Tabelle 3.6: Code mit Fehler. Wo ist der Fehler?

Wie aus dem obigen Beispiel ersichtlich ist, braucht es mehr als 2 Paritätsbits um nur 2 Informationsbits zu überprüfen. Aus dem Hamming Code ist ersichtlich, wie viele Paritätsbits für eine bestimmte Anzahl Informationsbits notwendig sind. In der nachfolgenden Abbildung ist gezeigt, wie 3 Paritätsbits bei 4 Informationsbits Einfachfehler erkennen, lokalisieren und korrigieren können.

n_7	n_6	n_5	n_4	n_3	n_2	n_1
m_4	m_3	m_2	k₃	m_1	k₂	k₁
x	x	x	x			
x	x			x	x	
x		x		x		x

F_3	F_2	F_1	
r	r	r	kein Fehler
r	r	f	k_1, n_1 ist falsch
r	f	r	k_2, n_2 ist falsch
r	f	f	m_1, n_3 ist falsch
f	r	r	k_3, n_4 ist falsch
f	r	f	m_2, n_5 ist falsch
f	f	r	m_3, n_6 ist falsch
f	f	f	m_4, n_7 ist falsch

Tabelle 3.7: *Hamming Code*

Die Fehlertabelle 3.7 der Korrekturbits (rechts) gibt die fehlerbehaftete Stelle an. Die Korrekturbits müssen an den Stellen 1, 2 und 4 positioniert sein, damit die Auswertung der k -bits die Position des Fehlerbits ergibt. Aus der Tabelle ist abzuleiten:

- k1 ergänzt m1, m2, m4
- k2 ergänzt m1, m3, m4
- k3 ergänzt m2, m3, m4

Folglich können mit 3 Korrekturbits (8 Werte) maximal 4 Informationsbit korrigiert werden. Um Einfachfehler korrigieren zu können, sind die folgenden beiden Bedingungen beim Hamming-Code notwendig:

- Jede Kolonne muss einen Eintrag haben.
- Kolonnen müssen unterscheidbar sein.

Mit den Variablen k , für Anzahl Paritätsbits und der Variablen m , der Anzahl Informationsbits lässt sich der folgende Zusammenhang finden:

$$2^k - 1 \leq m + k = n$$

Somit lassen sich für eine beliebige Anzahl Informationsbits die entsprechende notwendige Anzahl Paritätsbits berechnen, damit eine Fehlerkorrektur von Einfachfehlern vorgenommen werden kann.

Bei den folgende fehlerbehafteten Hamming Codes (4 Informationsbits und 3 Paritätsbits, gerade Parität) sind die Einfachfehler zu lokalisieren und die korrekten Informationsworte zu erzeugen.

- a) 0010100
- b) 1000100
- c) 0001100

3.7 Alphanumerische Codes

Alphanumerische Codes sind Codes für Buchstaben, Ziffern und Sonderzeichen. Ein Beispiel dafür ist der ASCII Code (american standard code of information interchange), welcher speziell für den Datenaustausch zwischen Computern verwendet wird. Der ASCII Code weist 7 Bits auf. Das 8. Bit wird für länderspezifische Sonderzeichen oder als Paritätsbit verwendet.

	n_7	0	0	0	0	0	1	1	1
	n_6	0	0	1	1	1	0	1	1
	n_5	0	1	0	1	1	0	1	1
n_4	n_3	n_2	n_1	NUL	DLE	SP	@	P	p
0	0	0	0	SOH	DC1	!	A	Q	q
0	0	0	1	STX	DC2	"	B	R	r
0	0	1	0	ETX	DC3	#	C	S	s
0	0	1	1	EOT	DC4	\$	D	T	t
0	1	0	0	ENQ	NAK	%	E	U	u
0	1	0	1	ACK	SYN	&	F	V	v
0	1	1	0	BEL	ETB	,	G	W	w
0	1	1	1	BEL	ETB	(	H	X	x
1	0	0	0	BS	CAN	)	I	Y	y
1	0	0	1	HT	EM		J	Z	z
1	0	1	0	LF	SUB	+	K	[	{
1	0	1	1	VT	ESC	++	L	\	
1	1	0	0	FF	FS	,	M	^	}
1	1	0	1	CR	GS	-	N	_	;
1	1	1	0	SD	RS	.	O	`	,
1	1	1	1	SI	US	/		-	DEL

Tabelle 3.8: 7 Bit ASCII Code

Kapitel 4

Boolesche Gleichungen und Logische Gatter

4.1 Lernziele

Nach dem Studium des vorliegenden Kapitels sind die folgenden Aufgaben zu lösen oder die folgenden Fragestellungen präzise zu beantworten:

- Unterschiede zwischen Boolescher- und klassischer Algebra.
- Welches sind Grund- und Spezial-Funktionen der Booleschen Algebra.
- Rechenregeln der Booleschen Algebra können angewendet werden.
- Beliebige kombinatorische Schaltungen können analysiert werden.
- Ausgehend von Wahrheitstabellen können die disjunktiven oder konjunktiven Normalformen von Schaltungen hergeleitet werden.
- Schaltungen können mittels graphischer und algorithmischer Methode vereinfacht werden.

4.2 Boolesche Konstanten und Variablen

In der Digitaltechnik werden Bauelemente eingesetzt, welche ein binäres Verhalten aufweisen. Die Ausgangssignale dieser digitalen Bauelemente weisen somit nur zwei unterschiedliche Signalwerte auf. Der Mathematiker George Boole entwickelte 1847 auf der Basis dieser Zwewertigkeit die nach ihm benannte Theorie der Booleschen Algebra. Später wurde diese Algebra von Shannon weiterentwickelt und zur Berechnung von logischen Schaltungen verwendet. Die Boolesche Algebra unterscheidet sich somit prinzipiell von der gewöhnlichen Algebra durch ihre Konstanten und Variablen, welche nur zwei Werte annehmen können: 0 und 1.

4.3 Grundfunktionen der Booleschen Algebra

Zusammenhänge zwischen den Variablen oder Konstanten bezeichnet man in der Booleschen Algebra analog zur gewöhnlichen Algebra ebenfalls als Funktionen. Allerdings existieren in der Booleschen Algebra keine kontinuierliche Funktionen, sondern es werden nur diskreten Wertepaaren Funktionswerte zugeordnet. Eine solche Zuordnung lässt sich einfach in einer Funktionstabelle oder Wahrheitstabelle definieren. In einer Wahrheitstabelle sind sämtliche Variablenkombinationen aufgeführt und jeder Kombination ist der entsprechende Funktionswert zugeordnet

E_1	E_1	Y
0	0	1
0	1	1
1	0	0
1	1	1

Tabelle 4.1: Wahrheitstabelle

In Tabelle 4.1 ist eine Wahrheitstabelle für zwei Eingangsvariablen E_1 und E_0 und einer Ausgangsvariablen Y dargestellt. Die Wahrheitstabelle ist ein mögliches Mittel, eine Funktion zu definieren. Eine weitere Variante ist die sogenannte Boolesche Gleichung, welche ähnlich wie die algebraische Gleichung für Berechnungen herangezogen werden kann.

Wie oben gezeigt, lassen sich mit Wahrheitstabellen und Booleschen Gleichungen Funktionen definieren. Es stellt sich nun die Frage was für Funktionen überhaupt in der Booleschen Logik existieren. Es kann einfach gezeigt werden, dass mit einer Variablen genau $2^2 = 4$, mit zwei Variablen $2(2^2) = 16$ Funktionen definiert werden können. Trotz dieser Vielfalt kann man mit nur wenigen Grundfunktionen sämtliche Funktionen definieren. Man unterscheidet zwischen drei wesentlichen Elementarfunktionen:

- Konjunktion: logische Multiplikation (logisches AND), Symbole: $\cdot$ oder $\wedge$
- Disjunktion: logische Addition (logisches OR), Symbole: $+$ oder $\vee$
- Komplement: logische Inversion (logisches NOT), Symbole: $\neg$ oder $_$

Prinzipiell reichen Komplement und Disjunktion oder Komplement und Konjunktion aus, um jede beliebige Boolesche Funktion zu beschreiben. Im nachfolgenden sind die drei Grundfunktionen in Form von Booleschen Gleichungen, Wahrheitstabellen, Schaltern und ihren Symbolen beschrieben.

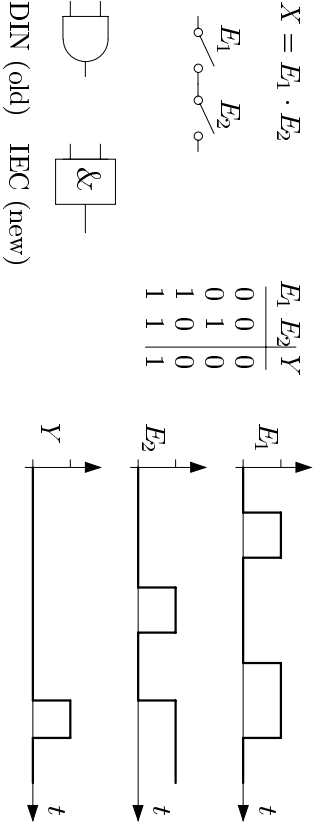
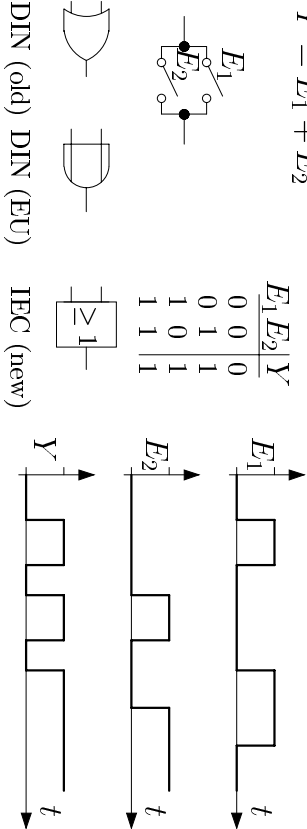


Abbildung 4.1: Konjunktion: Logisches AND Gatter

$Y = E_1 + E_2$



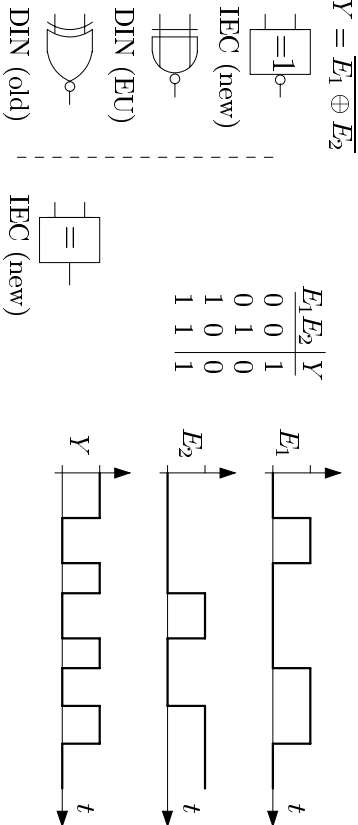


Abbildung 4.4: Äquivalenz: Identität bei mehreren Eingängen, (logisches exklusive NOR Gatter bei zwei Eingängen).

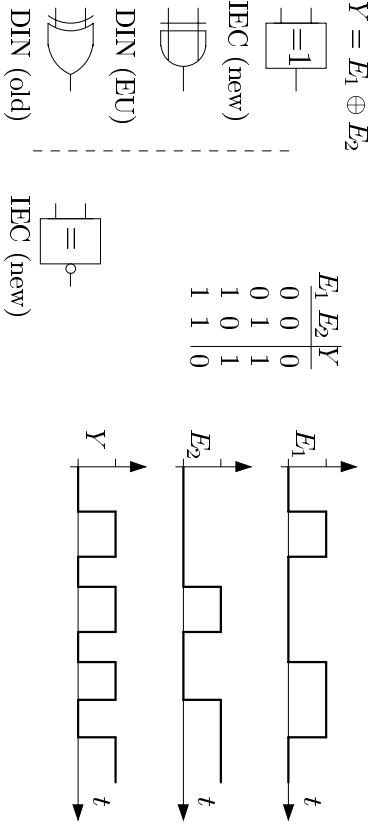


Abbildung 4.5: Antivalenz: Logisches exklusive OR Gatter.

der Eingänge bestimmen. Als Beispiel sei das Schema der Schaltung in Abbildung 4.6 zu analysieren. Für einen gewählten Eingangsvektor $(A, B, C, D) = (0, 1, 1, 0)$ ist der Ausgangswert zu bestimmen. Ebenso lässt sich sehr einfach die Funktion der Schaltung in Form einer Booleschen Gleichung oder in Form einer Wahrheitstabelle finden.

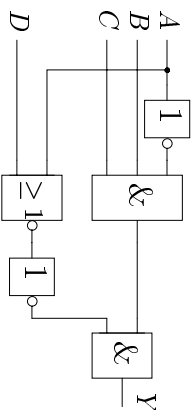


Abbildung 4.6: Beispiel einer logischen Schaltung.

4.6 Disjunktive und Konjunktive Normalform

Aus der Algebra ist bekannt, dass Funktionen in unterschiedlichen Gleichungen dargestellt werden können. Gleichungen können erweitert, durch Klammerausdrücke verschachtelt werden etc. ähnliches kann auch mit den Booleschen Gleichungen angestellt werden. Eine besonders nützliche und somit wichtige Darstellungsform der Booleschen Gleichungen sind die sogenannten disjunktive und konjunktive Normalformen. Hat man eine disjunktive Verknüpfung zwischen konjunktiven Termen, so spricht man von einer disjunktiven Normalform. Bei konjunktiver Verknüpfung zwischen disjunktiven Termen erhält man eine Konjunktive Normalform.

Ausgehend von der Wahrheitstabelle sollen anhand zweier unterschiedlicher Beispiele die beiden Normalformen der Booleschen Gleichungen systematisch hergeleitet werden.

Die so erhaltenen disjunktiven und konjunktiven Normalformen einer logischen Gleichung spielen eine besondere Rolle für das systematische vereinfachen von Verknüpfungsgleichungen. Belässt man die Booleschen Gleichungen in diesen Normalformen, so erhält man daraus immer eine zweistufige Logik (siehe Abbildung 4.8).

4.7 Rechenregeln der Booleschen Algebra

Die wichtigsten Rechenregeln der Booleschen Algebra seien ohne Kommentar in tabellarischer Form zusammengestellt (siehe Tabelle 4.2:

A	B	C	X	A	B	C	Y
0	0	0	0	0	0	0	1
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	1	1	0	0
1	1	1	1	1	1	1	1

$X = ABC \vee \bar{A}\bar{B}C$

$Y = (\bar{A} \vee B \vee C)(\bar{A} \vee \bar{B} \vee C)$

Abbildung 4.7: Disjunktive Normalform (links) und konjunktive Normalform (rechts).

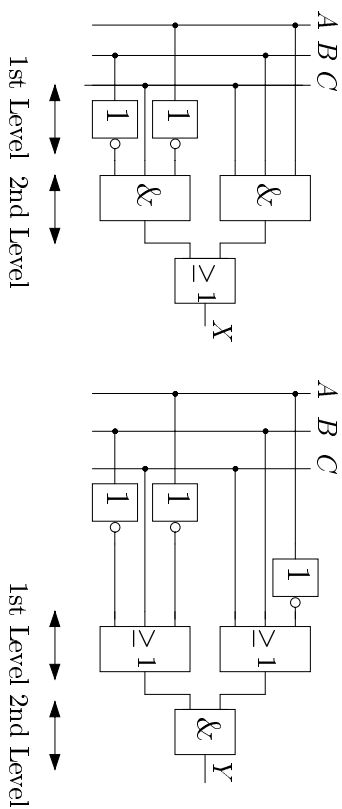


Abbildung 4.8: Zweistufige Logik

Sehr häufig wird das Gesetz von DeMorgan verwendet, weil damit AND und OR Gatter ineinander umgewandelt werden können.

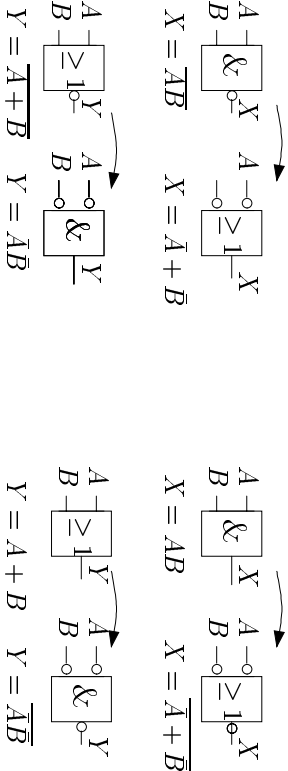
DeMorgan's Gesetze sind im Prinzip ein Spezialfall vom Shannonschen Theorem:

$$\overline{f(e_m, \overline{e_n}, \vee, \wedge, \oplus, \bar{\oplus}, \dots)} = f(\overline{e_m}, e_n, \wedge, \vee, \bar{\oplus}, \oplus, \dots) \quad (4.1)$$

Beispiel: Die folgende Logikschaltung ist so zu modifizieren, dass nur NAND Gatter benötigt werden:

Grundgesetz	(1)	$A \wedge A = A$
	(2)	$A \vee A = A$
Konstante	(3)	$A \wedge 1 = A$
	(4)	$A \wedge 0 = 0$
	(5)	$A \vee 1 = 1$
	(6)	$A \vee 0 = A$
Komplemente	(7)	$A \wedge \bar{A} = 0$
	(8)	$A \vee \bar{A} = 1$
	(9)	$\bar{\bar{A}} = A$
De Morgan	(10)	$\overline{A \wedge B} = \bar{A} \vee \bar{B}$
	(11)	$\overline{A \vee B} = \bar{A} \wedge \bar{B}$
Kommutativ	(12)	$A \wedge B = B \wedge A$
	(13)	$A \vee B = B \vee A$
Distributiv	(14)	$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
	(15)	$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
Assoziativ	(16)	$A \wedge (B \wedge C) = (A \wedge B) \wedge C$
	(17)	$A \vee (B \vee C) = (A \vee B) \vee C$
Absorption	(18)	$A \wedge (A \vee B) = A$
	(19)	$A \vee (A \wedge B) = A$

Tabelle 4.2: Rechenregeln der Booleschen Algebra



4.8 Vereinfachung und Optimierung von Funktionen

Die aus einer Wahrheitstabelle gewonnene disjunktive oder konjunktive Normalform einer booleschen Funktion ist meist nicht die kürzeste und

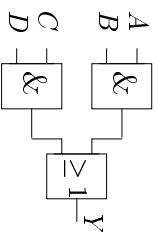


Abbildung 4.10: Beispiel zur Anwendung von DeMorgan's Gesetzze.

einfachste Form. Oft lassen sich Funktionen vereinfachen und zum Teil auch Variablen eliminieren. Die technische Realisierung der vereinfachten Funktion erfordert dann einen geringeren Aufwand als zum Beispiel eine der Normalformen. Für die systematische Vereinfachung muss die Boolesche Funktion in disjunktiver oder konjunktiver Normalform vorliegen. Es gibt verschiedene Vereinfachungsmethoden, von denen die folgenden beiden hier behandelt werden sollen:

- grafisches Verfahren nach Karnaugh und Veitch
- rechnerisches Verfahren nach Quine und McCluskey

In beiden Verfahren können nur Funktionen mit einer Ausgangsvariablen vereinfacht werden. Verfahren, mit denen Funktionen mit mehreren Ausgangsvariablen optimiert werden können, basieren meist auf den obigen Verfahren.

4.8.1 Vereinfachung mit dem Karnaugh-Diagramm

Um einfache Funktionen bis zu 5 oder maximal 6 Eingangsvariablen zu vereinfachen eignet sich die graphische Methode nach Karnaugh und Veitch sehr gut. Jeder Zeile der Wahrheitstabelle wird eine Zelle im Karnaugh-Diagramm zugewiesen (siehe Abbildung 4.11).

Nebeneinander liegende Minsterne können in Gruppen von 2^n zusammengefasst werden. Es kommt das Gesetz $AB \vee AB = A$ zur Anwendung.

Als Beispiel diene die folgende Boolesche Funktion mit zwei Eingangsvariablen (siehe Abbildung 4.12).

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

Abbildung 4.11: Karnaugh-Tabelle

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

$$Y = \bar{A}\bar{B} + \bar{A}B + AB$$

Abbildung 4.12: Vereinfachung bei zwei Eingangsvariablen

Bei der Vereinfachung mit dem Karnaugh-Diagramm sind die folgenden Regeln zu befolgen:

- Mit möglichst wenig Schleifen sind alle 1 (oder 0) zu erfassen.
- Es sind möglichst viele Zellen (2^n) mit einer Schleife zu erfassen.

Im folgenden sind Beispiele für Boolesche Funktionen mit mehreren Variablen zur Vereinfachung aufgeführt.

Bei diesem Beispiel mit den fünf Eingangsvariablen sei dazu noch die elektronische Schaltung dargestellt (siehe Abbildung 4.16).

Bei der Zusammenfassung von logischen 1 wurden immer möglichst grosse Schleifen gesucht. Vielfach treten aber auch Funktionen auf, bei denen die logischen 1 wie ein Schachbrett-Muster verteilt sind. In diesen Fällen findet man praktisch keine Vereinfachungsmöglichkeiten. Nutzt man bei den Vereinfachungen nicht mehr die Gleichung $AB \vee A\bar{B} = A$, sondern die Beziehung $A\bar{B} \vee \bar{A}B = A \oplus B$ aus, lassen sich genau diese Schachbrettmuster zusammenfassen, wie dies am Beispiel in Abbildung 4.17 illustriert ist.

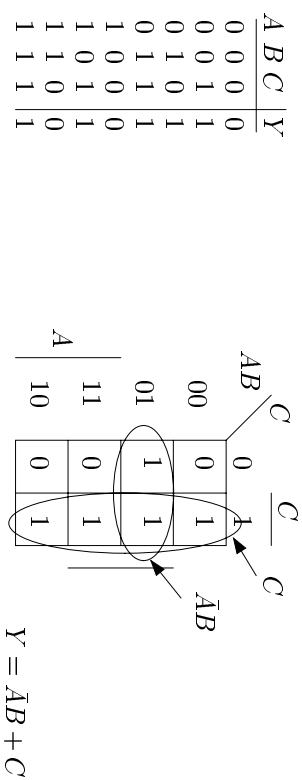


Abbildung 4.13: Vereinfachung bei drei Eingangsvariablen

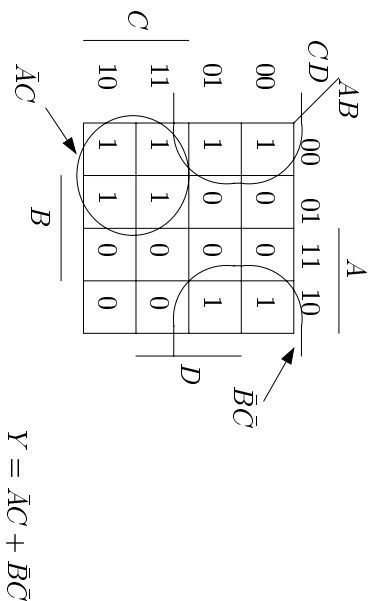


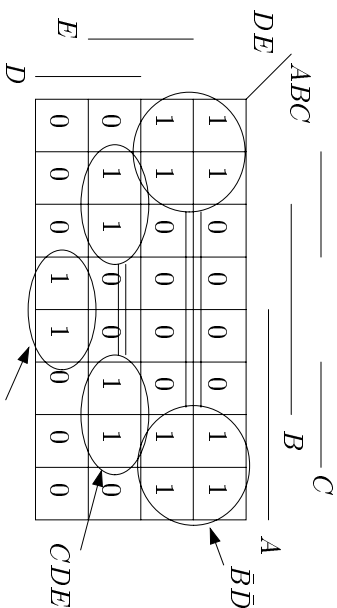
Abbildung 4.14: Vereinfachung bei vier Eingangsvariablen

4.8.2 Unvollständig definierte Funktionen

Sind in einer Wahrheitstabelle nicht immer alle Wertekombinationen der Variablen definiert (don't care), so spricht man von einer unvollständig definierten Funktion. Sind solche *don't cares* ("X") vorhanden, so lassen sie sich beliebig interpretieren, was bei den Vereinfachungen mehr Möglichkeiten zur Optimierung offen lässt (siehe Abbildung 4.18).

4.8.3 Vereinfachung nach Quine und McCluskey

Das Vereinfachungsverfahren nach Quine und McCluskey ist ein rechnerisches Verfahren zur Vereinfachung von Booleschen Gleichungen mit ei-



$$Y = \bar{B}\bar{D} + CDE + B\bar{C}D\bar{E}$$

Abbildung 4.15: Vereinfachung bei fünf Eingangsvariablen

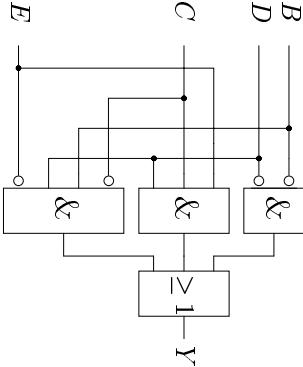


Abbildung 4.16: Schema zum Beispiel von fünf (vier) Eingangsvariablen.

nen Ausgang. Bei diesem Verfahren werden die Minterme der disjunktiven Normalform systematisch miteinander verglichen, um Variablen der Beziehung $A \vee \bar{A} = 1$ zu eliminieren. Zwei Minterme können nur dann vereinfacht werden, wenn sie sich in nur einer Variablen unterscheiden. Um solche Beziehungen zu finden, werden die Minterme in Gruppen zusammengefasst, welche nach Anzahl nichtinvertierter Variablen geordnet werden. Um Vereinfachungen zu finden, sind dann aufeinanderfolgende Gruppen miteinander zu vergleichen. Die folgende Boolesche Gleichung diene als Beispiel:

<u>A</u>	<u>B</u>	<u>Y</u>
0	0	0
0	1	1
1	0	1
1	1	0

<u>A</u>	<u>B</u>	<u>Y</u>
0	0	1
0	1	0
1	1	0
1	0	1



$Y = A \oplus B$

Abbildung 4.17: Schachbrettartiges Muster im Karnaugh-Diagramm führt zu EXOR Verknüpfung.

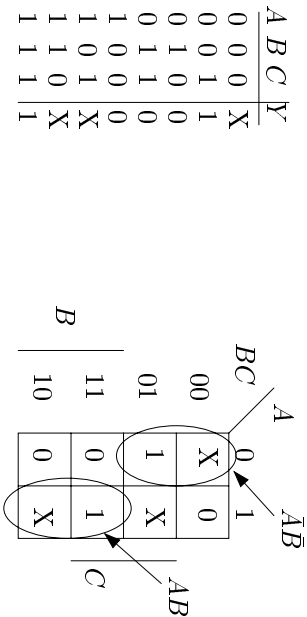


Abbildung 4.18: Vereinfachung am Beispiel mit don't cares

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} \vee \bar{A}\bar{B}C\bar{D} \vee \bar{A}B\bar{C}\bar{D} \vee \bar{A}BC\bar{D} \\ \vee A\bar{B}\bar{C}\bar{D} \vee A\bar{B}C\bar{D} \vee ABC\bar{D} \vee ABCD$$

Die beiden Minterme der "Gruppe I" sind mit denjenigen der "Gruppe 2" zu vergleichen (vgl. Tabelle 4.3). Sind zwei Minterme vorhanden, welche sich in nur einer Variablen unterscheiden, so sind die entsprechenden Minterme der "Gruppe I" und "Gruppe 2" zuzuhaken und der vereinfachte Term in die neue Rubrik "1. Vereinfachung" einzutragen. Dieses Verfahren wird nun solange fortgeführt, bis keine Vereinfachungen mehr möglich sind. Nicht abgehakte Minterme werden Printerm genannt (vgl. Tabelle 4.4).

Kapitel 5

Kippschaltungen

5.1 Lernziele

Nach dem Studium des vorliegenden Kapitels sind die folgenden Aufgaben zu lösen oder die folgenden Fragestellungen präzise zu beantworten:

- Verhalten der unterschiedlichen Implementationen von SR-Latches sind bekannt.
- Unterschiede zwischen synchronen und asynchronen Eingängen, Puls- und Flankensteuerung,
- Latch und Flip-Flops sind bekannt.
- Charakteristisches Zeitverhalten von Speicherelementen, mit setup-, hold, Verzögerungszeit und Kontrollpuls-Breite sind bekannt.
- Unterschiede zwischen bistabilen, monostabilen und astabilen Kippstufen sind bekannt.
- Verhalten von SR-, D-, JK-, T-, etc Speichern ist bekannt.
- Master-Slave und Einspeicher Implementation des Flip-Flops können analysiert werden.

- Wie werden die Zeiten bei den Monoflops bestimmt und welche Monoflop Typen existieren.
- Wie entstehen Hazards, wie lassen sie sich vermeiden und wo müssen sie zwingend vermieden werden.

5.2 Einführung

Bei den bisher behandelten logischen Bauelementen waren die Ausgangssignale nur von den jeweils herrschenden Eingangssignalen abhängig. Im folgenden werden erstmals sogenannte Kippstufen behandelt, deren Ausgangssignale nicht nur von den aktuellen Eingangswerten abhängig sind, sondern auch von früher herrschenden Zustand abhängen. Kippstufen weisen somit ein Gedächtnis (Speicher) auf. Man unterscheidet prinzipiell drei Arten, bistabile, monostabile und astabile Kippstufen.

5.3 Elementare bistabile Kippstufen (Latch)

Zwei Inverter ergeben zusammen ein Gebilde, welches von einem einmal eingenommenen logischen Zustand nicht mehr abweichen kann. Diese Anordnung kann 1 Bit speichern.

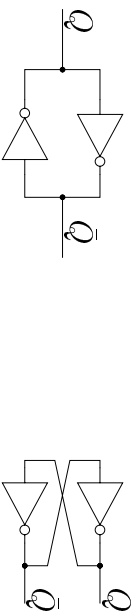


Abbildung 5.1: Speicher für 1 Bit.

5.3.1 Latch mit asynchronen Eingängen (Set-Reset Latch)

Ersetzt man die beiden Inverter durch NAND oder NOR Gatter, so erhält man ein steuerbares Speicherelement, das sogenannte Set-Reset Latch.

Die Funktion eines solchen Set-Reset Latches lässt sich aber auch aus der Wahrheitstabelle mittels Karnaugh-Diagramm herleiten.

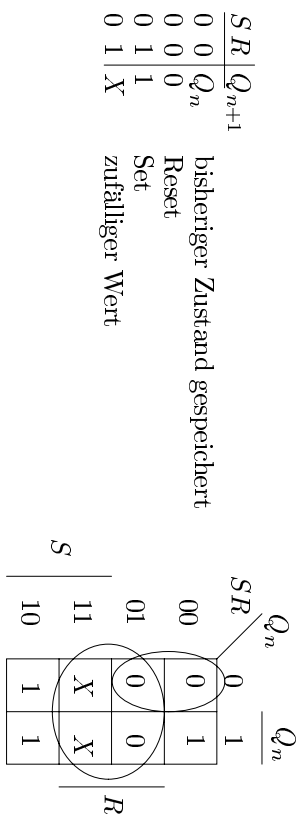


Abbildung 5.2: Wahrheitstabelle des SR-Latches.

Die vereinfachte Boolesche Funktion für den invertierten Ausgang lautet:

$$\overline{Q_{n+1}} = R + \bar{S} \overline{Q_n} \tag{5.1}$$

Dadurch lässt sich das SR-Latch mit zwei NOR Gatter realisieren.

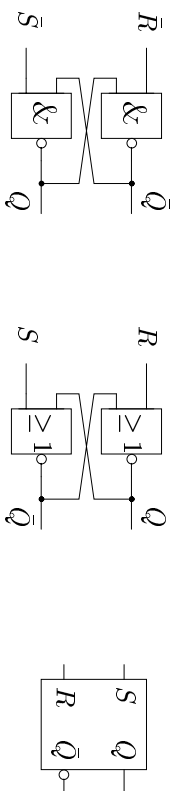


Abbildung 5.3: Schema eines NAND und NNOR SR-Latch's.

Typisch für ein Latch ist, dass sich Änderungen am Eingang sofort am Ausgang bemerkbar machen (transparentes Latch). Die Eingänge dieses Latch's werden deshalb auch als transparent bezeichnet. Das zeitliche Verhalten des SR-Latch's, aufgebaut aus NOR Gattern, ist in der folgenden Abbildung 5.4 dargestellt. Dabei ist die direkte Abhängigkeit der Ausgänge von den Eingängen zu sehen und das Auftreten eines zufälligen Ausgangswertes, wenn beide Eingänge gleichzeitig von 1 nach 0 wechseln.

SR-Latches sind die Grundbausteine für komplexere Speicherbausteine. Anwendungsbeispiel: mechanische Kontakte haben die unschöne Ge-

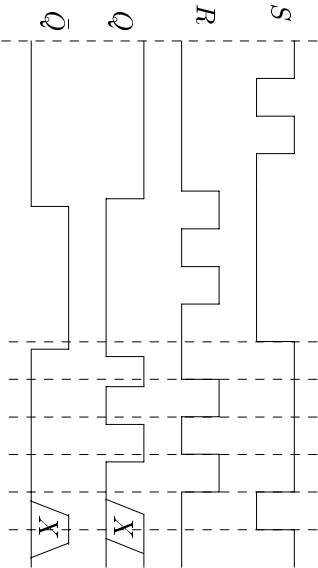


Abbildung 5.4: Zeiterhalten des NOR-NOR SR-Latch's.

wohnheit, beim Schliessen zu prellen. Mit einer geeigneten Beschaltung des SR-Latchs lässt sich dies verhindern.

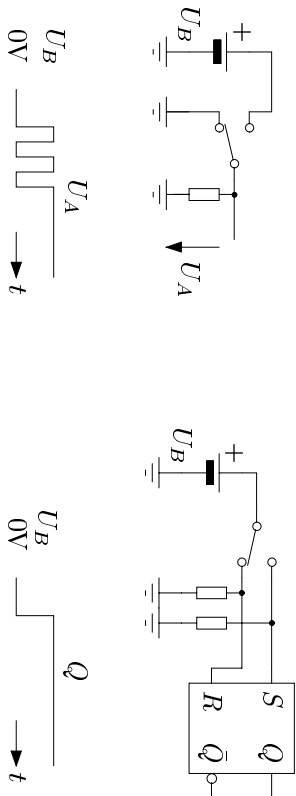


Abbildung 5.5: Anwendungsbeispiel prellfreier Schalter.

Ein weiteres Latch mit einem synchronen Eingang ist das sogenannte D-Latch. Die am D Eingang anliegenden Daten erscheinen an den Ausgängen, wenn das Kontrollsignal $C = 1$ ist. Wird C logisch 0, so wird der letzte Wert gespeichert.

5.3.2 Latch mit synchronen Eingängen

Es ist häufig von Nutzen mit Hilfe eines Aktivierungssignales (Gate, Strobe, Enable, Control Signal) den Zugang der Eingangssignale zum Latch steuern zu können. Dazu ist ein einfaches Netzwerk vor das SR-

Latch zu schalten, welches die Set und Reset Eingänge Kontrolliert. Ist das Signal $C = 0$, so vermögen die beiden Eingänge S und R den Zustand des Latch's nicht zu verändern.

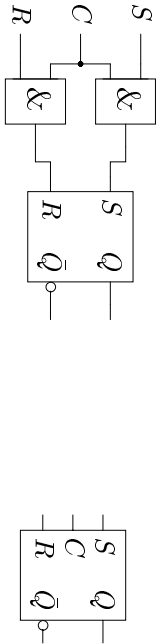


Abbildung 5.6: *SR-Latch mit Kontroll-Eingang.*

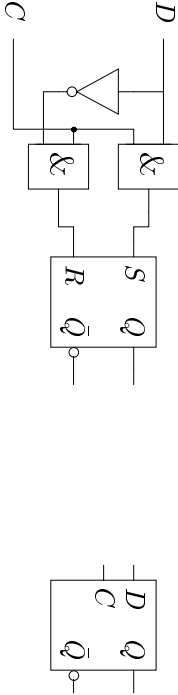


Abbildung 5.7: *D-Latch.*

5.4 Zeitverhalten von Latches mit synchro-
nen Eingängen

Um ein korrektes Funktionieren eines synchronen Latch's zu gewährleisten, müssen gewisse zeitlichen Bedingungen eingehalten werden:

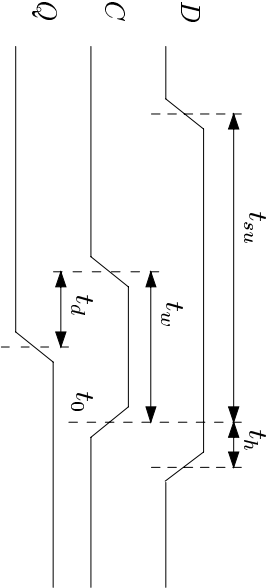


Abbildung 5.8: *Charakteristisches Zeitverhalten eines Latch's.*

Setup-Zeit (t_{su}): Minimale Zeit, welche die Eingangsdaten vor dem Speichervorgang (t_0) anlegen müssen, damit sie korrekt erkannt werden.

Hold-Zeit (t_h): Minimale Zeit, welche die Eingangsdaten nach dem Speichervorgang (t_0) noch vorhanden sein müssen.

Kontrollpuls-Breite (t_w): Minimale Zeit, welche das Kontrollsignal aktiv sein muss.

Verzögerungszeit (t_d): Verzögerungszeit des Ausgangssignales.

Die Setup-Zeit ist nötig, um sicher zu sein, dass jede Änderung der Eingangsdaten die Schaltung durchlaufen hat, bevor das Kontrollsignal inaktiv wird. Die Setup-Zeit und die Kontrollpuls Breite sind dafür verantwortlich, dass die internen Signalwerte am Ausgang erscheinen, bevor die Eingangssignale verschwinden.

5.5 Pulssteuerung und Flankensteuerung

Bei den Speicherelementen existieren zwei grundsätzlich verschiedene Arten der Taktsteuerung.

Pulssteuerung (Pulstriggerung): Bei der Pulssteuerung wirken die Informationseingänge solange auf den Speicherinhalt ein, wie der Pegel des Steuerungseinganges aktiv bleibt.

Flankensteuerung (Flankentriggerung): Bei der Flankensteuerung wirken die Informationseingänge nur während der aktiven Flanke des Steuerungseinganges auf den Speicherinhalt.

In Abbildung 5.9 ist der Unterschied dieser beiden Ansteuerungstypen in einem Zeitdiagramm dargestellt. Dabei ist deutlich zu sehen, dass bei der Pulssteuerung ein aktiver Pegel, bei der Flankensteuerung eine Pegeländerung den Speicherinhalt zu verändern vermag.

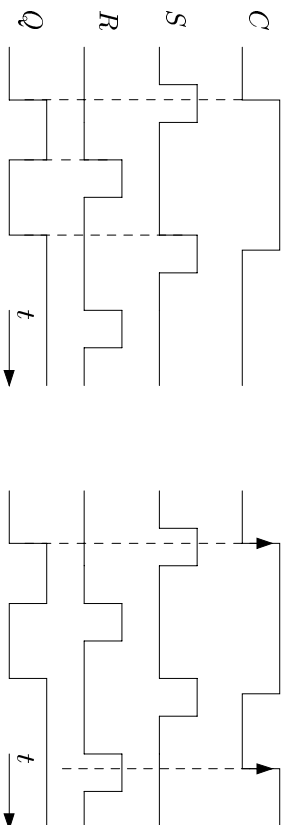


Abbildung 5.9: Gegenüberstellung von Pulssteuerung und Flankensteuerung.

5.6 Flip-Flop's (bistabile Kippstufen)

Bei synchronen Digitalsystemen werden mit einem Steuersignal verschiedene Speicherelemente gleichzeitig angesteuert. Die Speicherelemente können dabei auch in Serie geschaltet werden, so dass die Ausgänge einer ersten Speicherstufe die Eingänge einer zweiten Speicherstufe bilden. Mit den früher behandelten Latches lässt sich eine solche Schaltung nicht realisieren, da die Latch's im transparenten Zustand die Eingangsdaten durch die ganze Latch-Kette hindurch schieben würden. Es muss also ein neues Speicherelement eingeführt werden, das sogenannte Flip-Flop.

Unter einem Flip-Flop versteht man ein gesteuertes bistabiles Speicherelement, welches den transparenten Modus, wie es das Latch kennt, nicht besitzt. Das bedeutet, dass eine Änderung eines Flip-Flop Ausgangs nie direkt das Resultat einer Änderung eines synchronen Einganges sein kann. Die Konsequenz davon ist, dass bei einem Flip-Flop eine Synchronisierung möglich ist, ohne dass dabei Daten verloren gehen, was bei einem Latch nicht möglich ist.

5.6.1 SR-Flip-Flop

Aus Abbildung 5.11 ist ersichtlich, dass sich ein Master-Slave Flip-Flop prinzipiell aus zwei einfachen gesteuerten Latch's aufbaut. Verwendet

man dazu zwei gesteuerte SR-Latch's, so erhält man das entsprechende SR-Master-Slave Flip-Flop.

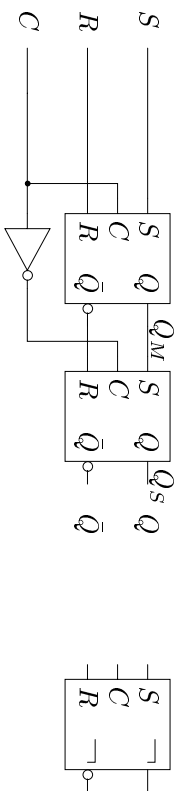


Abbildung 5.10: *Prinzipschema und Symbol des SR-Master-Slave Flip-Flop's.*

Im Inputdiagramm ist das zeitliche Verhalten dieses Master-Slave Flip-Flop Types dargestellt. Bei einem hohen C -Signal werden die Daten in das erste Latch, dem Master, eingelesen. Die Ausgänge des Flip-Flop's behalten den alten Zustand gespeichert. Erst wenn das C -Signal logisch 0 ist, werden die Daten vom zweiten Latch, dem Slave, übernommen und zum Ausgang des Flip-Flop's gebracht. Das Master-Latch ist in dieser Phase im gespeicherten Zustand und lässt keine neuen Daten passieren. Das Signal Q_M ist der Q Ausgang des Masters, das Signal Q_S der Q Ausgang des Slaves und damit des gesamten Flip-Flop's.

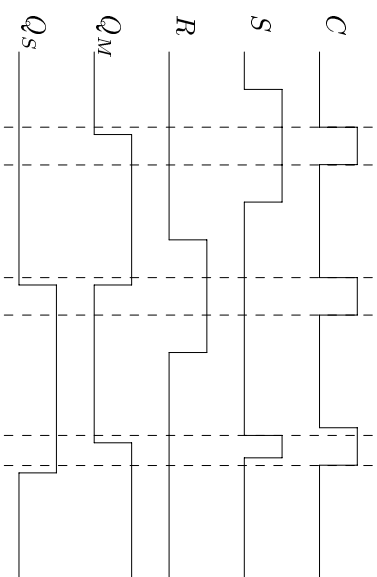


Abbildung 5.11: *Inputdiagramm des SR-Master-Slave Flip-Flop's.*

Der zweite 1 Puls des Set Einganges fällt innerhalb der 1 Phase des C -Takteinganges. Da diese Eingangsänderung vom Master-Latch übernommen wird und bei tiefem C -Eingang auf die Flip-Flop Ausgänge

übertragen wird, spricht man in diesem Fall von einem pulsetriggerten Flip-Flop. Das Ausgangssignal des Flip-Flop's reagiert also nicht auf eine Flanke des C -Eingangs sondern nur auf dessen Pegel.

5.6.2 D-Flip-Flop

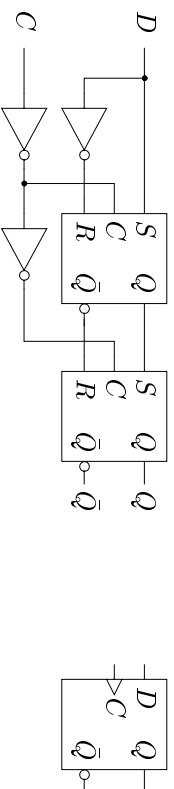


Abbildung 5.12: Master-Slave Implementation des flankengesteuerten D-Flip-Flop's.

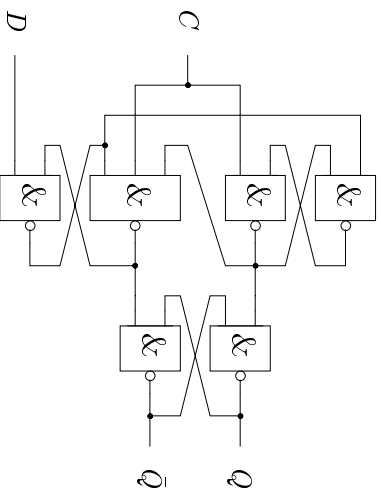


Abbildung 5.13: Einspeicher Implementation des Flip-Flop's.

Ähnlich den verschiedenen Varianten des einfachen Latch's, lassen sich nun ebenfalls mehrere Flip-Flop Varianten realisieren. Verwendet man das Master-Slave Prinzip auf das D-Latch an, so erhält man die ältere Implementationsvariante des D-Flip-Flop's (Abbildung 5.12). Ändert sich der D Eingang während einem tiefen C Signal vorübergehend, so übernimmt das Master-Latch diese Änderung ebenfalls nur vorübergehend. Für die Flip-Flop Ausgangssignale ist aber nur der aktuelle Eingangswert bei steigender Taktflanke von Bedeutung, weshalb man hier von

einem flankengertigten D-Flip-Flop spricht. Eine andere, aktuellere Implementationsvariante des flankengertigten D-Flip-Flop's stellt das Schema in Abbildung 5.13 dar. Hier handelt es sich nicht mehr um eine Master-Slave Struktur, sondern um einen Einfachspeicher mit vorgeschalteter Logik. Analysiert man die Funktionsweise dieser Struktur etwas genauer, so stellt man fest, dass bei tiefem Taktsignal C die erste Kippstufe nicht wie beim Master-Slave Flip-Flop als Speicher fungiert, da die Ausgnge dieser Kippstufe unabhngig vom Eingang D logisch 1 sind. Erst wenn das Taktsignal von 0 auf 1 wechselt, ibernimmt zuerst die erste, dann die zweite Kippstufe den entsprechenden Wert des Eingangs.

5.6.3 JK-Flip-Flop

Neben einer grossen Funktionsvielfalt verschiedener Flip-Flop's wird vor allem das JK-Flip-Flop sehr hufig eingesetzt. hnlich dem SR-Flip-Flop weist es ebenfalls zwei Dateneingnge auf, welche das Flip-Flop setzen, beziehungsweise rckstellen knnen. Der einzige, aber wesentliche Unterschied zum SR-Flip-Flop besteht aber darin, dass beim JK-Flip-Flop beide Dateneingnge gleichzeitig logisch 1 sein drfen. Die Bedeutung der beiden J und K Eingnge ist in der Wahrheitstabelle in Tabelle 5.1 dargestellt:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

Tabelle 5.1: Wahrheitstabelle des JK-Flip-Flop's.

Erweitert man diese Wahrheitstabelle um den Eingang Q_n , so findet man eine Beziehung fr den S Eingang eines SR-Flip-Flop's (Abbildung 5.14). hnlich kann fr den R Eingang vorgegangen werden.

In Abbildung 5.15 ist das Schema des JK-Flip-Flop's dargestellt. Daraus ist ersichtlich, wie aus dem SR- ein JK-Flip-Flop hergeleitet werden kann.

$\begin{array}{c c} JK & Q_n \\ \hline \end{array}$			$\begin{array}{c c} Q_n & \\ \hline \end{array}$	
0	0	0	0	0
0	0	1	X	
0	1	0	0	
0	1	1	0	
1	0	0	1	
1	0	1	X	
1	1	0	1	
1	1	1	0	

$S = J\bar{Q}$

JK		Q_n	
J	K	0	1
		0	X
0	0	0	X
	1	0	
1	0	1	0
1	1	1	0
1	1	1	X

Abbildung 5.14: Herleitung der Bedingung für den S Eingang.

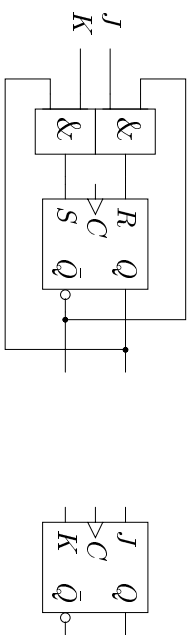


Abbildung 5.15: JK-Flip-Flop aufgebaut aus SR-Flip-Flop.

5.7 Asynchrone Eingänge von Flip-Flop's

Neben den synchronen Dateneingängen können die im vorausgegan-
nen Abschnitt besprochenen Flip-Flop's noch asynchrone Eingänge auf-
weisen. Die zusätzlichen asynchrone Eingänge werden normalerweise
für das Setzen oder Rückstellen der Flip-Flop's benötigt:

- asynchroner Preset Eingang (PR)
- asynchroner Clear Eingang (CR)

Der Preset Eingang setzt den Ausgang Q des Flip-Flop's, der Cle-
ar Eingang löscht den Ausgang Q . In Abbildung 5.16 ist das Symbol
eines flankengetriggerten D-Flip-Flops mit asynchrone $\overline{PR}$ und $\overline{CR}$
Eingängen dargestellt. Die Implementation, wie sie beim SN7474 Schalt-
kreis dafür verwendet wird ist in Abbildung 5.17 dargestellt. Die beiden
Signale sind aktiv tief, was bedeutet, dass sie nicht gleichzeitig logisch
0 sein dürfen, weil sonst der Ausgangswert Q undefiniert ist.

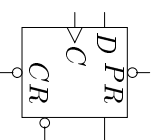


Abbildung 5.16: Symbol eines D-Flip-Flop's mit asynchronen Setz- und Rücksetzeingängen.

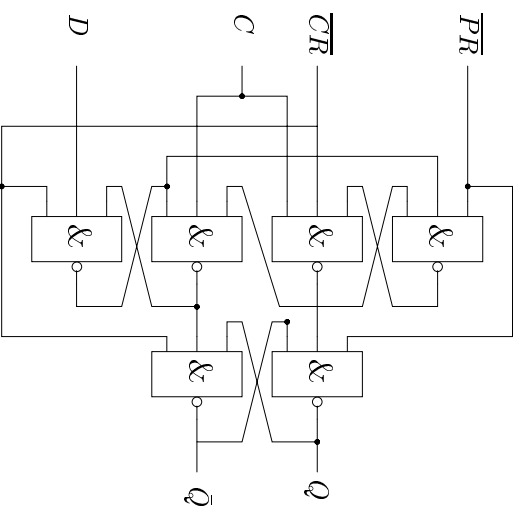


Abbildung 5.17: Implementation eines SN7474 Flip-Flop's.

Beim Entwurf von Digitalschaltungen sollte darauf geachtet werden, dass die darin verwendeten Flip-Flop's zumindest beim Systemstart in einen definierten Zustand gebracht werden können. Dies bedingt, dass nur Flip-Flop's verwendet werden sollen, welche einen asynchronen Setz- oder Rücksetz-Eingang aufweisen. Aus Testüberlegungen ist es äusserst nützlich, wenn ein definierter Zustand zu jedem Zeitpunkt erreicht werden kann, was nach einem generellen Initialisierungssignal verlangt (meist RESET).

5.8 Monoflops (monostabile Kippstufen)

Ein Monoflop ist eine rückgekoppelte binäre Schaltung, die wie ein Flip-Flop zwei verschiedene innere Zustände annehmen kann. Der Unterschied zum Flip-Flop besteht darin, dass beim Monoflop nur einer der beiden inneren Zustände stabil ist. Der andere Zustand behält seinen Wert nur für eine bestimmte Zeit und kippt danach wieder in seinen stabilen Zustand zurück.

Monoflops lassen sich aus SR-Latch's oder Flip-Flop's mit vorgeordneten Differenzier- oder Verzögerungsgliedern aufbauen. Um die Zeitkonstante einstellen zu können, wird ein externes RC-Glied verwendet.

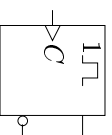


Abbildung 5.18: *Symbole einer flankengetriggerten monostabilen Kippstufe.*

In Abbildung 5.18 ist das Symbol einer monostabiler Kippschaltungen dargestellt. Bei aktiver Triggerung erzeugt es einen positiven Ausgangspuls definierter Länge. Bei den Monoflops unterscheidet man zwei verschiedene Typen:

- nachtriggerbare Monoflops
- nicht nachtriggerbare Monoflops

Der Unterschied dieser beiden Typen ist im Zeitdiagramm in Abbildung 5.19 illustriert. Beim nicht nachtriggerbaren Monoflop haben aktive Triggersignale währenddem gerade ein Puls am Ausgang erzeugt wird keinen Einfluss auf das Ausgangssignal. Ein Vertreter eines Monoflops ist zum Beispiel der Baustein SN74123, welcher je nach Beschaltung auf die steigende oder fallende Flanke reagiert und nachtriggerbar oder nichtnachtriggerbar arbeiten kann.

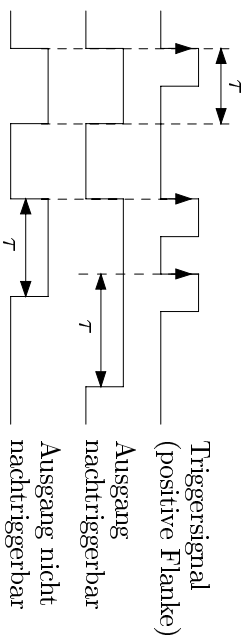


Abbildung 5.19: Zeitdiagramm von nachtrIGGERbaren und nicht nachtrIGGERbaren Monoflops.

5.9 Multivibrator (astabile Kippstufe)

Ein Multivibrator ist eine rückgekoppelte binäre Schaltung, die zwei verschiedene innere Zustände hat, zwischen denen sie dauernd hin und her kippt. Die Schaltung wird deshalb auch astabile Kippstufe oder Rechteckgenerator genannt. Das astabile Verhalten wird wie beim Monoflop durch eine zeitabhängige Rückkopplung mit Differenzier- oder Verzögerungsgliedern erreicht.

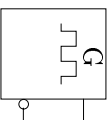


Abbildung 5.20: Symbol eines Multivibrators.

Eine sehr einfache Schaltung eines Multivibrators besteht aus einem Schmitt-Trigger und einem RC-Verzögerungsglied, wie es in Abbildung 5.22 dargestellt ist. Ein Schmitt-Trigger verhält sich ähnlich einem Inverter, mit dem Unterschied, dass am Eingang nicht eine, sondern je eine tiefe und eine hohe Schwellspannung existiert. Er wird vielfach dazu verwendet, um Störungen besser unterdrücken zu können.

Die Spannung U_c über dem Kondensator wird in der Aufladephase von U_{su} gegen U_{Qmax} aufgeladen. Sobald der Wert der oberen Schwellspannung U_{so} erreicht wird, wird der Aufladevorgang durch einen Entladevorgang ersetzt. Die Spannung U_c wird dabei von U_{so} gegen U_{Qmin} entladen, bis U_{su} erreicht wird. In der Zeitphase von 0 bis t_1 gilt die

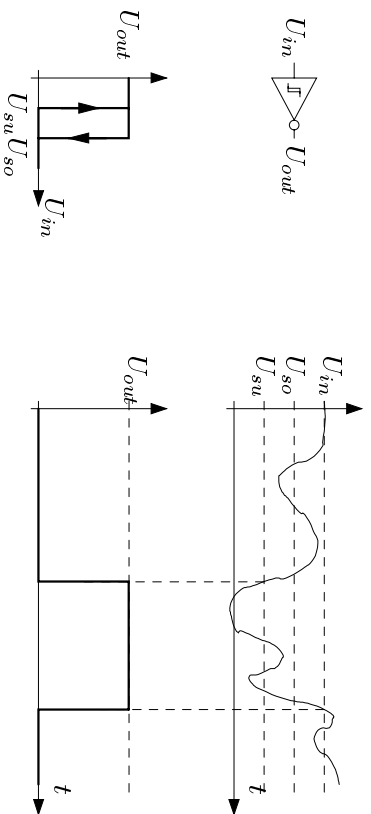


Abbildung 5.21: Spannungsverlauf des Schmitt-Triggers.

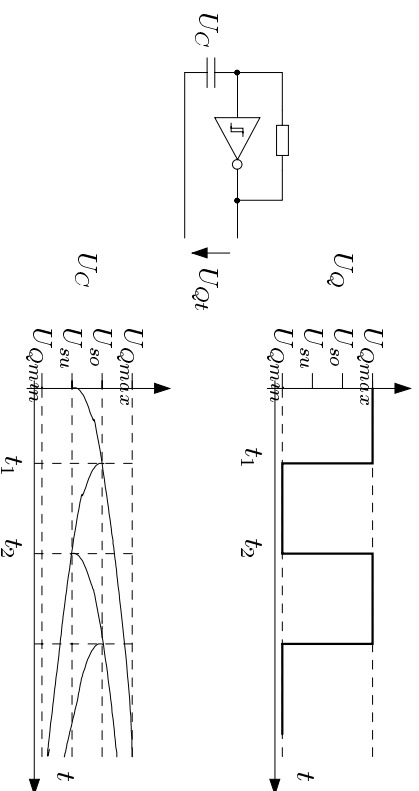


Abbildung 5.22: Zeitdiagramm eines einfachen Multivibrators.

folgende Beziehung:

$$U_c = (U_{Qmax} - U_{su})(1 - e^{-\frac{T}{RC}}) + U_{su} \quad (5.2)$$

Zum Zeitpunkt $t = t_1$ ist die Kondensatorspannung gleich der oberen Schwellspannung U_{so} . Damit erhält man für die halbe Periode T die folgende Beziehung:

$$T = t_1 = RC \ln \frac{U_{Qmax} - U_{su}}{U_{Qmax} - U_{so}} \quad (5.3)$$

Aus dieser Beziehung ist ersichtlich, wie die Periode mit einem RC-Glied festgelegt werden kann.

5.10 Hazards

Bei den bisherigen Betrachtungen wurde immer angenommen, dass ein logisches Gatter keine interne Verzögerungszeit aufweist. Dadurch war auch die Problematik von Einschwingvorgängen beiseite geschoben. Bringt man bei logischen Elementen nicht nur ihre logische Funktion ins Spiel, sondern berücksichtigt auch ihr zeitliches Verhalten, so wird man ungewollte Phänomene beobachten, welche die korrekte Funktionsweise einer Schaltung zunichte machen können.

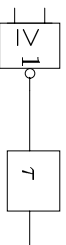


Abbildung 5.23: *Reales logisches Gatter mit Verzögerung.*

In Abbildung 5.23 ist ein reales logisches Gatter dargestellt, welches aus einer logischen Booleschen Funktion und einem Verzögerungselement besteht. Berücksichtigt man solche endliche Gatterlaufzeiten, so können unliebsame Einschwingvorgänge beobachtet werden.

5.10.1 Entstehung von Hazards

Hängt das Ausgangssignal einer kombinatorischen Schaltung neben den Eingangssignalen auch von Verzögerungszeiten ab, dann weist die Schaltung einen Hazard auf. Bei einer kombinatorischen Schaltung können Einschwingvorgänge von Hazards betroffen werden. Hazards lassen sich prinzipiell in zwei Klassen unterteilen:

- statische Hazards
- dynamische Hazards

Es ist ein statischer Hazard anwesend, wenn nur eine vorübergehende Änderung des Ausgangssignales hervorgerufen wird und der statische

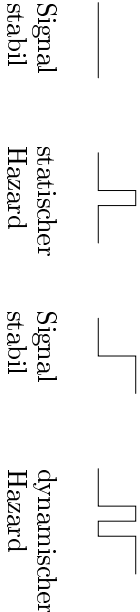


Abbildung 5.24: *Gegenüberstellung statischer und dynamischer Hazards.*

Ausgangswert nicht tangiert wird. Bei einem dynamischen Hazard ändert sich das Ausgangssignal mehrmals, bevor es auf seinen stabilen Wert wechselt.

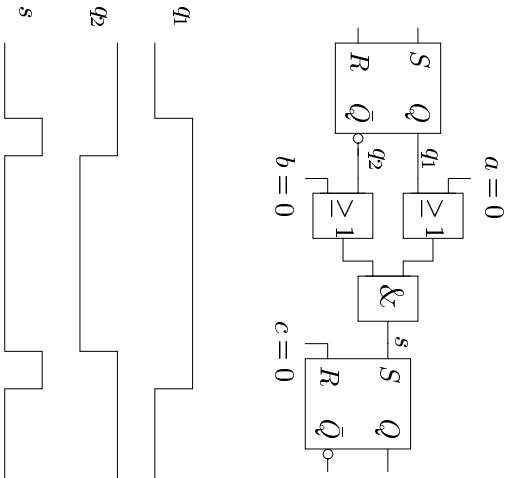


Abbildung 5.25: *Schaltung mit statischem Hazard an Signal s .*

In Abbildung 5.25 ist ein Schaltungsbeispiel wiedergegeben, welches aufgrund eines Hazards nicht funktionstüchtig ist. Das erste SR-Latch weist am Q Ausgang den logischen Wert 0 auf. Wird es danach gesetzt, so wird zuerst das Signal q_1 auf 1 wechseln, bevor q_2 auf 0 fällt. Da somit kurzzeitig beide Eingänge des AND-Gatters 1 sind, wird am Signal s ein statischer Hazard auftreten. Dieser Hazard setzt nun fälschlicherweise das zweite Latch.

		AB		a		A	
		00	01	11	10		
CD	00	1	0	0	1		
	01	1	0	0	1		
C	11	1	1	0	0		
	10	1	1	0	0		
						D	
						B	

Abbildung 5.26: Entstehung eines Hazards.

Hazards können nur dann auftreten, wenn sich mehr als eines der Eingangs- oder der internen Signale fast gleichzeitig wechseln. Ob in bestimmten Situationen ein Hazard auftreten kann, lässt sich durch das Betrachten des Karnaugh-Diagrammes (Abbildung 5.26) feststellen. Die Signale $ABCD$ wechseln von 0000 nach 0101. Da die beiden Signale B und D in der Realität nicht exakt gleichzeitig von 0 nach 1 wechseln können, wird das eine Signal zuerst den Wechsel vornehmen. Wie aus dem Pfeil a ersichtlich ist, erstrebt ein statischer Hazard, wenn zuerst das Signal B von 0 nach 1 und erst danach D wechselt. Ist die Reihenfolge umgekehrt, so führt der Weg von Pfeil b nur über logische 1, das heißt das Ausgangssignal bleibt stabil und weist keine Hazards auf.

5.10.2 Vermeidung von Hazards

Hazards lassen sich leider in kombinatorischen Schaltungen nicht immer unterdrücken. Vermeiden lassen sich gegebenenfalls Situationen, welche Hazards erzeugen können. Es ist deshalb wichtig, besonders kritische Signale auf die Möglichkeit von Hazards zu untersuchen und entsprechende Massnahmen zu ergreifen. Generell darf man feststellen, dass Clock, Setz- und Rücksetzsignale von Flip-Flop's kritische Signale sind. Ein statischer Hazard auf eine solche Leitung ruft immer ein Fehlverhalten der Schaltung hervor. Befolgt man zum Beispiel die folgende Entwurfsregel, so lassen sich schon viele kritische Hazards eliminieren:

- Keine kombinatorische Verknüpfungen zur Erzeugung von Clock-signalen und Setz- beziehungsweise Rücksetzsignalen verwenden.
- Hazardempfindliche Signale korrekt einsynchronisieren.

Es darf generell festgestellt werden, dass asynchrone Schaltung sehr kritisch bezüglich dem Entstehen von Hazards sind. Dies ist ein Grund, weshalb soweit wie möglich synchrone Schaltungen entworfen werden sollen. Die Sicherheit, welche man sich mit synchronen Schaltungen erkaufte, geht auf Kosten des Synchronisierungs-Mechanismus, was erhöhte Verzögerungszeiten bedeutet.

Kapitel 6

Zustandsmaschinen

6.1 Lernziele

Nach dem Studium des vorliegenden Kapitels sind die folgenden Aufgaben zu lösen oder die folgenden Fragestellungen präzise zu beantworten:

- Wodurch unterscheiden sich Mealy-, Moore und Medwedjew Automaten. Welches sind die Vor-, respektive Nachteile derselben.
- Ausgehend von der Beschreibung eines Algorithmus kann das entsprechende Zustandsdiagramm aufgestellt werden.
- Vollständigkeit und Konsistenz werden beim Aufstellen des Zustandsdiagrammes berücksichtigt.
- Ausgehend vom Zustandsdiagramm lassen sich Übergangstabelle und Ausgangstabelle generieren und daraus die minimierte Form des Automaten synthetisieren.
- Die Problematik der parasitären Zustände und die Initialisierung des Automaten ist bekannt.
- Die Problematik der Hazards in Sequenzern und bei der Ansteuerung externer Datenpfade ist bekannt.

- Die Problematik der Einsynchronisation asynchroner Signale in Sequenzern ist bekannt.

6.2 Grundlegende Sequenzer Strukturen

Zustandsmaschinen sind Schaltungen, welche aus einer gegebenen Sequenz von Eingangssignalen eine vordefinierte Sequenz von Ausgangssignalen erzeugen. Zustandsmaschinen, auch endliche Automaten, Folgeschaltungen oder Sequenzer genannt (engl. finite state machine), sind in der Digitaltechnik und vor allem in VLSI Schaltungen sehr häufig anzutreffen. Die Gründe dafür sind nahelegend. Ablaufsteuerungen lassen sich einfach und verständlich in Form von Zustandsdiagrammen beschreiben. Die Syntheseierung von Schaltungen aus den Zustandsdiagrammen erfolgt völlig automatisch und damit zeiteffizient und fehlerfrei, was für den Entwurf von Digital- oder VLSI Schaltungen höchst willkommen ist.

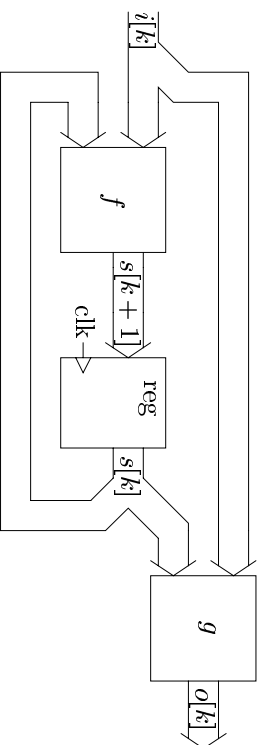


Abbildung 6.1: *Struktur des Mealy-Automaten.*

Zwei bekannte Implementationsvarianten endlicher Automaten sind der Mealy- und der Moore-Automat. Beide Automaten sind je aus zwei kombinatorischen Blöcken (mit den Funktionen f und g) und einem Register für die Zustandsspeicherung aufgebaut. In Abbildung 6.1 ist die Struktur eines Mealy-Automaten dargestellt. Daraus ist ersichtlich, dass die Ausgänge o sowohl von den internen Zuständen s , als auch direkt von den Eingängen i abhängig sind. Der Mealy-Automat erfüllt die folgenden Funktionen:

$$o[k] = g(i[k], s[k])$$

$$s[k+1] = f(i[k], s[k])$$

Leicht abgeändert ist die Struktur des Moore-Automaten (siehe Abbildung 6.2), bei welchem keine direkte Abhängigkeit von Ein- und Ausgangssignalen mehr besteht. Diese Tatsache widerspiegelt sich direkt in seinen charakteristischen Gleichungen:

$$\begin{aligned} o[k] &= g(s[k]) \\ s[k+1] &= f(i[k], s[k]) \end{aligned}$$

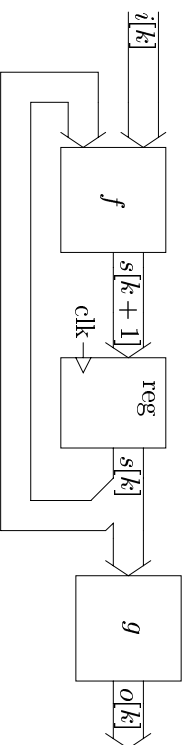


Abbildung 6.2: *Struktur des Moore-Automaten.*

Ein recht häufig geschehener Spezialfall des Moore-Automaten ist der sogenannte Medwedjew-Automat, bei welchem die Zustandsvariablen direkt als Ausgänge dienen (siehe Abbildung 6.3). Durch den Verzicht auf einen kombinatorischen Block nach den Zustandsvariablen ist gewährleistet, dass die Ausgänge hazardfrei sind. Die charakteristischen Gleichungen des Medwedjew-Automaten vereinfachen sich dadurch wie folgt:

$$\begin{aligned} o[k] &= s[k] \\ s[k+1] &= f(i[k], s[k]) \end{aligned}$$

Bei allen drei Automaten kann der kombinatorische Block f in Form einer PLA (AND-OR) Struktur aufgebaut werden.

6.3 Beschreibung von Zustandsmaschinen

Für die Beschreibung des Verhaltens eines Zustandsmaschinen stehen prinzipiell zwei Mittel zur Verfügung:

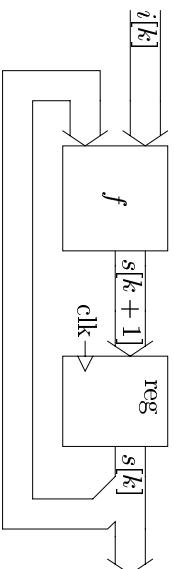


Abbildung 6.3: Der Medwedjew-Automat ist ein Spezialfall des Moore-Automaten.

- Übergangstabelle und Ausgangstabelle
- Zustandsdiagramm

6.3.1 Übergangs- und Ausgangstabelle

Durch die Übergangstabelle, zusammen mit der Ausgangstabelle wird ein endlicher Automat oder Sequenzer vollständig beschrieben. Diese Beschreibungsart lehnt sich sehr stark an die bereits bekannten Wahrheitstabellen an. Die Übergangstabelle, auch Zustandstabelle genannt, definiert den Übergang von einem alten in einen neuen Zustand, weshalb diese Tabelle häufig auch Übergangstabelle genannt wird. Im Gegensatz zu den Wahrheitstabellen aus der rein kombinatorischen Logik, bei welchen die Ausgänge nur in Abhängigkeit der Eingänge beschrieben wurden, sind in der Übergangstabelle die Ausgänge in Abhängigkeit der Eingänge und der alten Zustände zu beschreiben. Betrachtet man die in den Abbildungen 6.1, 6.2 und 6.3 dargestellten Zustandsmaschin-Strukturen mit den dazugehörigen Gleichungen, so ist ersichtlich, dass jeweils die Funktion $f(i[k], s[k])$ diese Übergangsbedingungen beschreibt. Die Übergangstabelle stellt somit die Funktion $f(i[k], s[k])$ tabellarisch dar.

Ein Beispiel einer einfachen Übergangstabelle ist in Tabelle 6.1 dargestellt. Ist der Automat im *Zustand-1*, so springt er in der nächsten Taktphase in den *Zustand-2*, sofern der Eingang *a* logisch 1 ist, andernfalls springt er in den *Zustand-4*. Auf diese einfache Art lässt sich jede Zeile als eine Übergangsbedingung lesen. Bei grösseren Übergangstabellen bleibt die Interpretation zwar eindeutig, die Übersichtlichkeit geht aber schnell verloren. Beim Moore und beim Mealy-Automaten

alte Zustände		Eingänge		neue Zustände	
Name	$s[k]$	$i[k] = (a, b)$	Name	$s[k + 1]$	
Zustand-1	00	1X	Zustand-2	01	
Zustand-1	00	0X	Zustand-4	11	
Zustand-2	01	0X	Zustand-2	01	
Zustand-2	01	1X	Zustand-3	10	
Zustand-3	10	X1	Zustand-1	00	
Zustand-3	10	00	Zustand-3	10	
Zustand-3	10	10	Zustand-4	01	
Zustand-4	11	XX	Zustand-1	00	

Tabelle 6.1: Beispiel einer Übergangstabelle.

(siehe Abbildung 6.1 und 6.2) erzeugen erst die Funktionen $g(s[k])$ respektiv $g(i[k], s[k])$ die eigentlichen Ausgänge. Bei diesen beiden Automaten Typen ist somit eine weitere Tabelle, die Ausgangstabelle für eine vollständige Beschreibung der Zustandsmaschine notwendig. Diese Ausgangstabellen stellen aber nichts anderes als eine einfache kombinatorische Verknüpfung der betrachteten Eingänge dar und kann somit durch eine gewöhnliche Wahrheitstabelle dargestellt werden.

6.3.2 Zustandsdiagramm

Das Zustandsdiagramm erlaubt die Funktion einer Zustandsmaschine auf eine graphische Art und Weise zu beschreiben. Die Funktionalität kann damit sehr übersichtlich dargestellt werden. Durch das Einführen von Namen und Begriffen, welche Zustände und Übergangsbedingungen beschreiben, kann auf einer höheren Abstraktionsebene gearbeitet werden. Bei sehr grossen und komplexen Zustandsmaschinen wird durch eine hierarchische Verschachtelung der Zustandsdiagramme die Übersichtlichkeit im einzelnen Diagramm gewahrt:

In Abbildung 6.4 ist dasselbe Beispiel, wie es in der Übergangstabelle von Tabelle 6.1 beschrieben ist, als Zustandsdiagramm dargestellt. In diesem Diagramm wird ein Zustand durch einen Knoten (Kreis) dargestellt. Die Pfeile stellen die Übergänge von einem Zustand (Knoten) in einen anderen dar. Unbeschriftete Pfeile weisen auf einen unbedingten Zustandswechsel hin, beschriftete Pfeile stellen bedingte Übergänge dar.

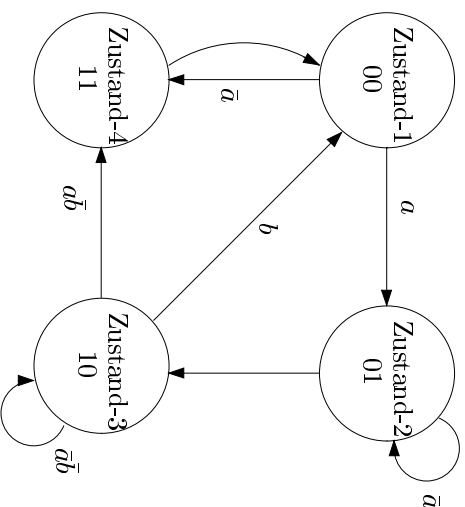


Abbildung 6.4: Beispiel eines Zustandsdiagramms.

Der Übergang von *Zustand-3* nach *Zustand-4* erfolgt in diesem Beispiel nur, wenn die beiden Eingangsvariablen a und b die Bedingung $ab = 1$ erfüllen.

6.4 Analyse von Sequenzerschaltungen

Bei der Analyse eines Sequenzers sind aus dem Schema die Zustands- und eventuell die Ausgangstabelle herzuleiten. Mithilfe der erarbeiteten Tabellen lässt sich das Zustandsdiagramm aufstellen, welches eine übersichtlichere Darstellung des Verhaltens vermittelt.

$$\begin{aligned}
 Q_2^{n+1} &= \overline{(Q_1^n \oplus e)} \cdot \overline{Q_0^n} + Q_0^n \cdot Q_2^n \\
 Q_1^{n+1} &= (Q_2^n \oplus e) \cdot Q_0^n + \overline{Q_0^n} \cdot Q_1^n \\
 Q_0^{n+1} &= (Q_1^n \oplus Q_2^n) \oplus e
 \end{aligned}$$

Die Übergangstabelle und das Zustandsdiagramm sind aus der Schaltung in Abbildung 6.5 herzuleiten. Die Frage nach der Funktion der

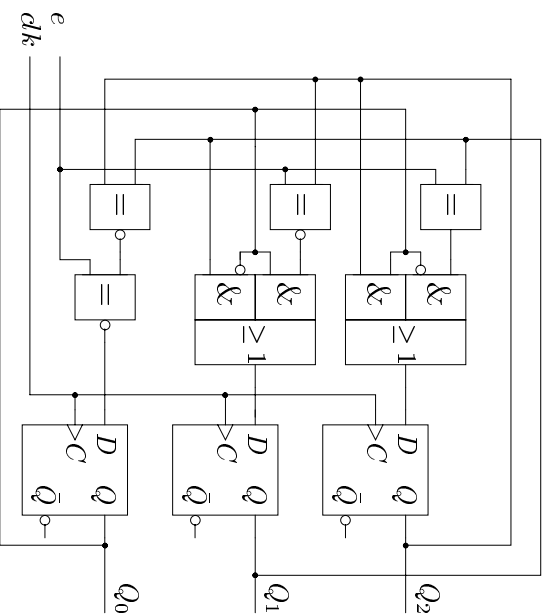


Abbildung 6.5: Zustandsmaschine mit D-Flip-Flop's

Schaltung kann danach einfach beantwortet werden. Dazu werden zuerst die algebraischen Gleichungen aufgestellt, nach welchen die Flip-Flop Eingänge gehorchen müssen.

Anhand dieser Gleichungen lässt sich die Übergangstabelle aufstellen, welche aus den logischen Werten des Eingangs e und den alten Zuständen der Speicherelemente Q^n die neuen Zustände Q^{n+1} für die Speicher definiert (siehe Tabelle 6.2).

Mit den Informationen der Übergangstabelle kann das Zustandsdiagramm gezeichnet werden. In der Tabelle sind acht verschiedene Zustände codiert, was zu den entsprechenden acht Zuständen (Knoten) im Zustandsdiagramm führt. Jede Zeile der Tabelle entspricht einer Übergangsbedingung, welche im Diagramm mit einem Pfeildargestellt wird. Sind nun alle Zeilen der Tabelle im Diagramm übertragen, so ist ein vollständige Funktionsbeschreibung der Zustandsmaschine in einer graphischen Form dargestellt (siehe Abbildung 6.6). Aus dem Diagramm ist nun gut ersichtlich, dass für den Eingang $e = 1$ die Zustandsmaschine im Uhrzeigersinn in einem zyklischen Code vorwärts zählt, bei $e = 0$ im selben zyklischen Code rückwärts zählt. Da bei der Schaltung in Abbil-

e	Q_2^n	Q_1^n	Q_0^n	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	0	1	1
0	0	1	1	0	0	1
0	1	0	0	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	1	0
0	1	1	1	1	1	1
1	0	1	1	1	1	1
1	0	0	0	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	1	0
1	0	1	1	0	1	0
1	1	0	0	0	0	0
1	1	0	1	1	0	0
1	1	1	0	1	1	1
1	1	1	1	1	0	1

Tabelle 6.2: *Übergangstabelle der Zustandsmaschine aus Abbildung 6.5.*

Abbildung 6.6 die Speicherzustände direkt als Ausgänge verwendet werden, spricht man von einem Medwedjew Automaten.

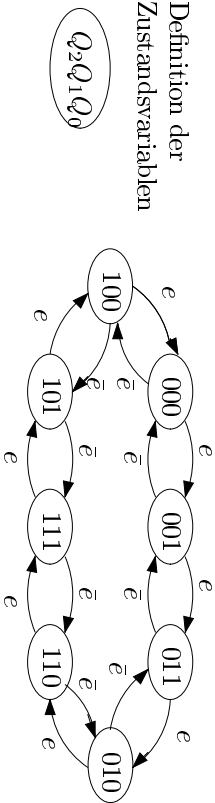


Abbildung 6.6: *Zustandsdiagramm des Automaten in Abbildung 6.5.*

6.5 **Synthese von Sequenzerschaltungen**

Der Entwurf einer Zustandsmaschine erfolgt über das Zustandsdiagramm, wie es in Abbildung 6.4 (siehe auch Abbildung 6.6) dargestellt ist. Den

Zuständen, namentlich *Zustand-1* bis *Zustand-4*, werden Werte zu ihrer Identifikation zugewiesen. Bei den Mealy- und Moore-Automaten können durch geschickte Zuweisungen von Werten zu den einzelnen Zustandsvariablen optimierte kombinatorische Blöcke entstehen. Den Ablauf im Zustandsdiagramm und damit die Ausgangswerte werden durch die Eingangsvariablen a und b bestimmt, welche in diesem Beispiel gerade mit den Ausgängen identisch sein sollen (Medwelow-Automat). Um vom Zustandsdiagramm zu einer Implementation des Automaten zu gelangen, ist die Übergangstabelle herzuleiten (siehe Tabelle 6.1). Jede Zeile der Tabelle repräsentiert einen Übergang (Pfeil) eines alten in einen neuen Zustand unter der erfüllten Übergangsbedingung.

6.5.1 Vollständigkeit und Konsistenz

Beim Aufstellen des Zustandsdiagramms sind die folgenden Regeln strikt einzuhalten:

- In keinem der Zustände dürfen gleichzeitig zwei Übergangsbedingungen erfüllt werden können. Dies bedeutet, dass die UND-Verknüpfung (Schnittmenge) von zwei beliebigen Übergangsbedingungen (Pfeile), welche einen Zustand (Knoten) verlassen, immer 0 sein muss.
- Ebenso dürfen keine Eingangskombinationen auftreten, welche nicht einem Übergang zugewiesen sind. Dies bedeutet, dass die ODER-Verknüpfung aller übergangsbedingungen (wegführende Pfeile) immer 1 sein muss.

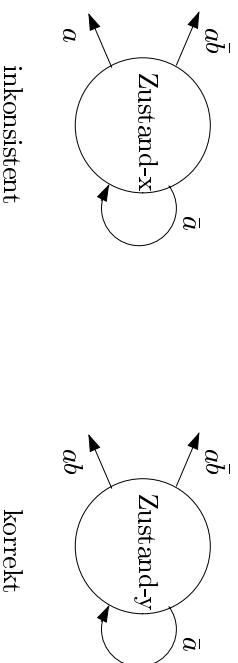


Abbildung 6.7: Inkonsistentes Zustandsdiagramm.

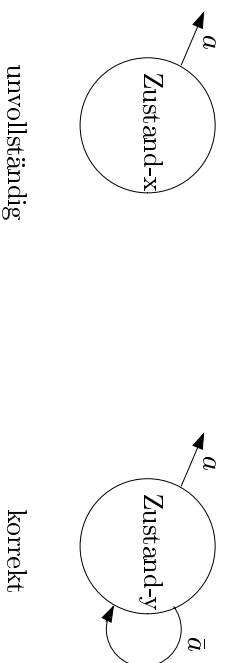


Abbildung 6.8: Unvollständiges Zustandsdiagramm.

Ist eine dieser Bedingungen nicht erfüllt, so ist das Zustandsdiagramm inkonsistent oder unvollständig und somit nicht realisierbar.

6.5.2 Parasitäre Zustände

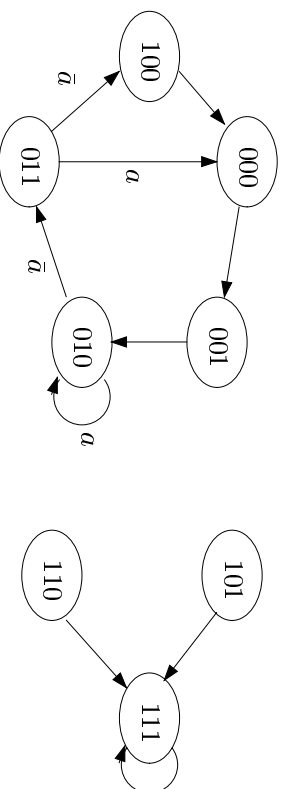


Abbildung 6.9: Zustandsdiagramm mit Schatten-Zustandsdiagramm aus parasitären Zuständen.

Vorsicht ist mit den sogenannten parasitären Zuständen geboten (siehe Abbildung 6.9). Dies sind Zustände, welche nicht in die Ablaufsteuerung eingebunden sind. Sie treten auf, wenn nicht alle der möglichen 2^n Zustandskodierungen (bei n Zustandsvariablen) verwendet werden. Es ist dafür zu sorgen, dass man nicht in einem parasitären Zustand gefangen bleibt und das auch keine Schatten-Zustandsdiagramme ausgeführt werden, welche aus parasitären Zuständen bestehen. Ziel jeder Entwicklung ist es, eine möglichst robuste Schaltung zu erhalten. Die Auswirkung von temporären Fehlern sind auf einen möglichst kurzen Zeitintervall zu begrenzen. Bei einem Sequenzer gibt es Regeln, wie

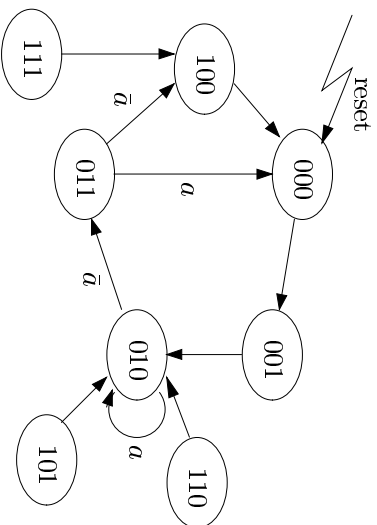


Abbildung 6.10: Die parasitären Zustände aus Abbildung 6.9 sind ins Zustandsdiagramm eingebettet.

mit den parasitären Zuständen umgegangen werden soll:

- So weit es möglich ist sind parasitäre Zustände zur Deckung mit den echten Zuständen zu bringen (Beispielsweise 110 mit 011 mit 000).
- Alle parasitären Zustände sind auf kürzestem Weg ins Zustandsdiagramm zu führen (Beispielsweise 110 mit 011).
- Aus jedem Zustand kann auf Wunsch in einen Start-Zustand gesprungen werden.

In Abbildung 6.10 sind zwei der drei parasitären Zustände aus Abbildung 6.9 mit den echten Zuständen zur Deckung gebracht worden. Der dritte parasitäre Zustand führt direkt ins Zustandsdiagramm. Zusätzlich kann durch ein Reset-Signal jederzeit in den Start-Zustand gesprungen werden. Grosse Zustandsmaschinen benötigen meistens eine unverhältnismässig grosse AND-OR-Ebene. Ein Unterteilen in mehrere kleinere Zustandsmaschinen reduziert nicht nur die benötigte Siliziumfläche, sondern reduziert auch gleichzeitig die Verzögerungszeiten im kombinatorischen Block.

Es ist nicht immer ratsam, den kombinatorischen Teil der Zustandsmaschinen als AND-OR-Ebene aufzubauen. Moderne Synthese-Werkzeug, welche die vielen Freiheitsgrade der Logiksynthese (AND, OR,

EXOR, komplexe Gatter) nutzen, sind häufig instande kompaktere oder schnellere Sequenzer zu erzeugen, als dies mit den klassischen PLA-basierten Sequenzerstrukturen der Fall ist.

6.5.3 Hazards in Zustandsmaschinen

Kombinatorische Schaltungen sind potentielle Quellen für die Produktion von Hazards. Da in einer Zustandsmaschinen kombinatorische Schaltungsblöcke vorhanden sind, besteht auch hier die Gefahr von unerwünschten Hazards. Betrachtet man die in den Abbildungen 6.1 bis 6.3 skizzierten Zustandsmaschinen-Strukturen, so stellt man fest, dass nur der kombinatorische Block g Hazards am Ausgang zu produzieren vermag. Nur bei Medwedjew-Automaten entstehen keine Hazards. Bei den anderen beiden Automaten-Typen können mit einem zusätzlichen nachgeschalteten Register die Hazards unterdrückt werden, wobei jedoch eine zusätzliche Verzögerung um eine Taktperiode in Kauf genommen werden muss.

6.5.4 Einsynchronisation

Betrachtet man die Implementationsstrukturen eines endlichen Automaten (Abbildungen 6.1 bis 6.3), so ist ersichtlich, dass durch den kombinatorischen Block g sich die Setup-Zeiten der Eingänge erhöhen. Solange sich die Eingänge des Automaten synchron zum Takt ändern ergeben sich daraus keine Setup-Zeit Verletzungen. Probleme können dann entstehen, wenn sich die Eingänge völlig asynchron zum Takt ändern können. Durch die unterschiedlichen Laufzeiten durch den kombinatorischen Block, können an den Speichereingängen kurzzeitig unerwartete Eingangskombinationen auftreten, welche zu einem falschen Zustand (eventuell parasitären Zustand) führen können.

Eine Möglichkeit, die Problematik asynchroner Eingänge bei Zustandsmaschinen etwas zu entschärfen, ist die Einsynchronisation solcher Signale. In Abbildung 6.11 ist gezeigt, wie bei einem Mealy-Automaten durch ein vorgeschaltetes Register eine Einsynchronisation vorgenommen werden kann. Die erhöhte Sicherheit erkauft man sich aber mit einer größeren Verzögerungszeit.

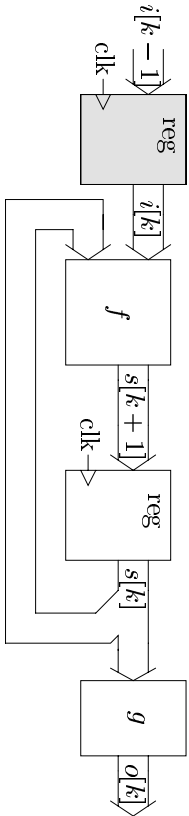


Abbildung 6.11: *Einsynchronisation in Moore-Automaten.*

6.5.5 Entwurfsbeispiel: Lichtsignalsteuerung

Anhand einer einfachen Lichtsignalsteuerung soll ein Moore Automat synthetisiert werden. Die Funktionsweise des Lichtsignals sei wie folgt beschrieben.

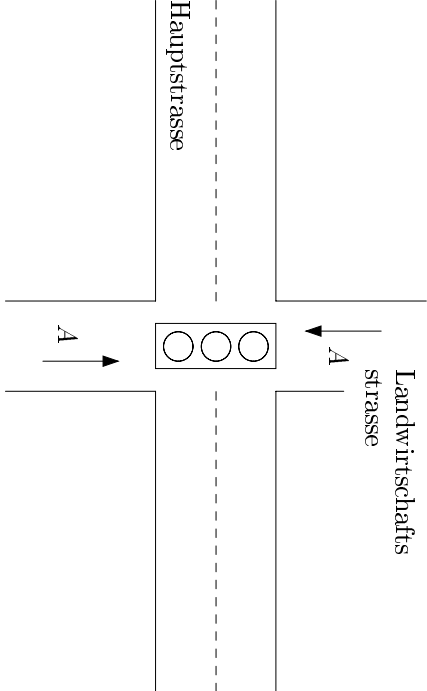


Abbildung 6.12: *Lichtsignalsteuerung mit Haupt- und Nebenstrasse.*

Problemstellung: Eine stark befahrene Hauptstrasse weist eine Kreuzung mit einer selten befahrenen Landwirtschaftsstrasse auf. Die Landwirtschaftsstrasse weist vor der Kreuzung einen Sensor auf, welcher die Anwesenheit von Landwirtschaftsfahrzeugen detektieren kann ($A = 1$, Fahrzeug vorhanden). Warten keine Fahrzeuge auf der Landwirtschaftsstrasse, so soll das Signal auf der Hauptstrasse grün ($H_G = 1$) und dasjenige der Landwirtschaftsstrasse rot ($L_R = 1$) sein. Wartet ein

Fahrzeug auf der Landwirtschaftsstrasse, so wechselt das Signal auf der Hauptstrasse zuerst auf orange ($H_O = 1$) und eine Zeiteinheit später auf rot ($H_R = 1$). Gleichzeitig wird das Signal der Landwirtschaftsstrasse auf grün ($L_G = 1$) gestellt. Nach zwei Zeiteinheiten wechselt das Signal auf der Landwirtschaftsstrasse auf orange ($L_O = 1$) und danach auf rot, sofern keine weiteren Landwirtschaftsfahrzeuge erscheinen. Andernfalls bleibt das Signal der Landwirtschaftsstrasse im gesamten maximal drei Zeiteinheiten auf grün geschaltet. Das Signal der Landwirtschaftsstrasse kann frühestens nach vier rot-Zyklen wieder auf grün geschaltet werden.

In einem ersten Schritt zur Entwicklung eines Automaten ist das Zustandsdiagramm zu erstellen. Gemäss den Regeln aus der Problemstellung werden die Zustände und deren Übergänge graphisch dargestellt. Da in diesem Beispiel nur acht verschiedene Zustände auftreten, wurden sie mit drei Bits codiert, welche die Ausgänge der Speicher darstellen werden. Die geforderten Signale für die beiden Lichtsignale mit den drei Lampen müssen somit aus den Zuständen erzeugt werden. Für die Realisierung dieser Schaltung wird somit ein Moore-Automat eingesetzt werden. Aus dem Zustandsdiagramm lässt sich nun sehr einfach die Übergangstabelle erstellen, indem für jede Übergangsbedingung eine Zeile in der Tabelle aufgestellt wird. In Tabelle 6.3 ist die entsprechende Übergangstabelle dargestellt.

A	Q_2^n	Q_1^n	Q_0^n	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}
X	0	0	0	0	0	1
X	0	1	0	1	1	0
X	1	0	0	0	0	0
X	1	0	1	1	0	0
X	1	1	0	1	1	1
X	0	0	1	0	1	1
0	0	1	0	0	1	1
1	0	1	1	0	1	0
0	1	1	0	1	0	0
1	1	1	0	1	0	1

Tabelle 6.3: *Übergangstabelle für die Lichtsignalsteuerung.*

Aus den codierten Zustandsbits sind nun noch die Ansteuerungen für die beiden Lichtsignalanlagen mit den drei Signalfarben zu generieren. Dazu ist eine sogenannte Ausgangstabelle zu erstellen. In Tabelle 6.4 ist eine solche Ausgangstabelle dargestellt. Die drei Zustandsbits Q_i bilden

Q_2	Q_1	Q_0	H_G	H_O	H_R	L_G	L_O	L_R
0	0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	0	1
0	1	1	1	0	0	0	0	1
0	1	0	0	1	0	0	0	1
1	1	0	0	0	1	1	0	0
1	1	1	0	0	1	1	0	0
1	0	1	0	0	1	1	0	0
1	0	0	0	0	1	0	1	0

Tabelle 6.4: Ausgangstabelle der Lichtsignalsteuerung.

die Eingänge in der Wahrheitstabelle, die Stellgrößen der Lichtsignale die Ausgänge.

Mit der Ausgangstabelle und dem Zustandsdiagramm oder der Übergangstabelle ist die Funktion des Moore-Automaten eindeutig bestimmt. Durch Vereinfachungen mittels Karnaugh-Diagrammen der beiden Tabellen lässt sich nun eine logische Schaltung für die Lichtsignalanlage aufbauen. Aus der Übergangstabelle erhält man die Beziehungen:

$$\begin{aligned} Q_2^{n+1} &= \overline{Q_0^n} Q_1^n + Q_0^n Q_2^n \\ Q_1^{n+1} &= \overline{Q_0^n} Q_1^n + Q_0^n \overline{Q_2^n} \\ Q_0^{n+1} &= \overline{Q_1^n} Q_2^n + \overline{A} Q_0^n \overline{Q_2^n} \overline{Q_0^n} Q_1^n Q_2^n + A Q_1^n Q_2^n \end{aligned}$$

Aus der Ausgangstabelle erhält man die folgenden einfachen booleschen Gleichungen:

$$\begin{aligned} H_G &= \overline{Q_0} \overline{Q_2} + \overline{Q_1} \overline{Q_2} & L_G &= \overline{Q_0} Q_2 + Q_1 Q_2 \\ H_O &= \overline{Q_0} Q_1 \overline{Q_2} & L_O &= \overline{Q_0} \overline{Q_1} Q_2 \\ H_R &= Q_2 & L_R &= \overline{Q_2} \end{aligned}$$

Aus diesen vereinfachten Gleichungen lässt sich nun direkt das Logikschema der Lichtsignalsteuerung in Form eines Moore-Automaten erstellen, wie es in Abbildung 6.14 abgebildet ist.

Endliche Automaten können für verschiedene Aufgaben herangezogen werden. Mit ihnen können Zähler verschiedenster Art aufgebaut werden, wie dies in Abbildung 6.6 angedeutet wurde. Das Hauptanwendungsgebiet für endliche Automaten sind vor allem Ablaufsteuerungen, wie sie fast ausnahmslos in jeder komplexeren Digitalschaltung anzutreffen sind. Dabei wird der Ablauf und das Zusammenspiel von unterschiedlichen Hardwareeinheiten über Steuersignale, welche der Automat generiert, kontrolliert. Ein etwas simples Beispiel dazu stellt die behandelte Lichtsignalsteuerung dar, da hier nur das Kommando für die Signallampen erzeugt wurde.

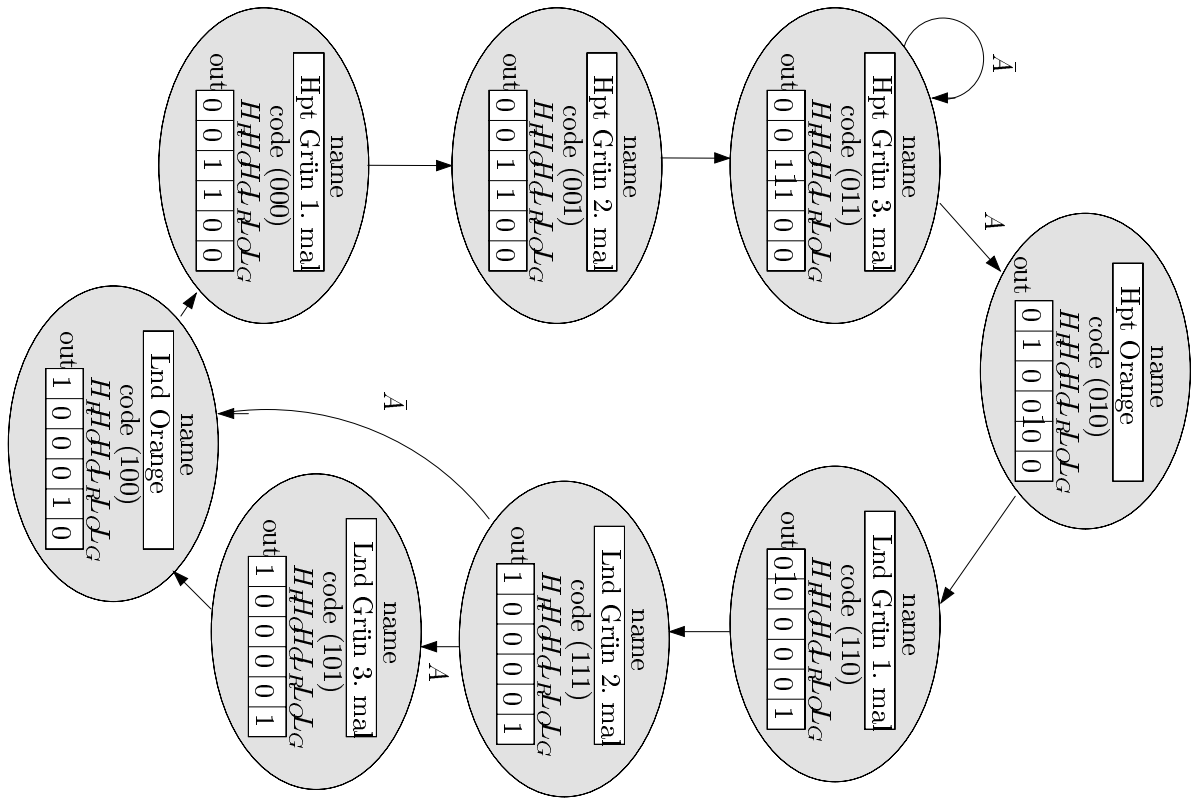


Abbildung 6.13: Zustandsdiagramm der Lichtsignalsteuerung.

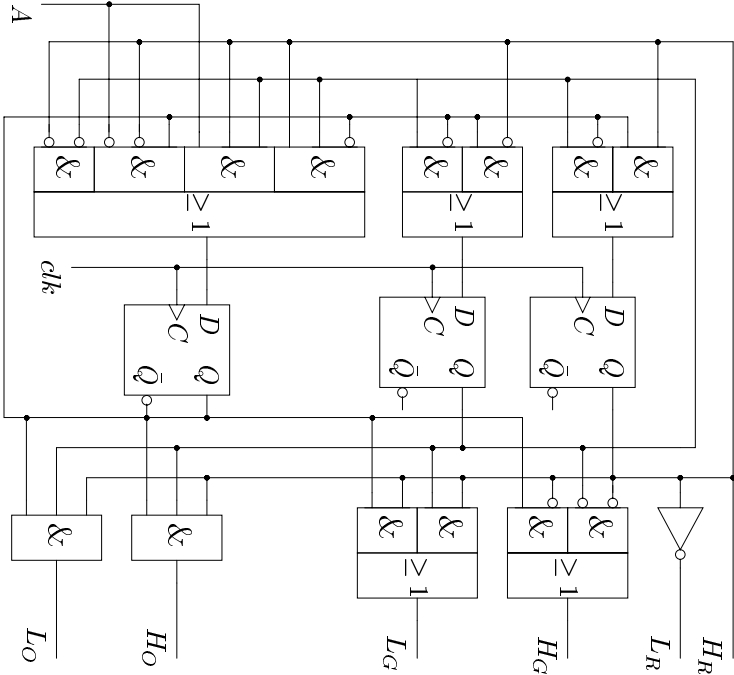


Abbildung 6.14: Schema der Lichtsignalsteuerung.

Literaturverzeichnis

- [Beu90] Klaus Beuth. *Digitaltechnik*. Vogel Fachbuch / Elektronik 4, 7. auflage edition, 1990.
- [Bor89] L. Borucki. *Digitaltechnik*. Verlag B. G. Teubner Stuttgart, 1989.
- [Hay93] John P. Hayes. *Introduction to Digital Logic Design*. Addison-Wesley Publishing Company, 1993.
- [Man] Mange. *Technique Numerique*. ETH Lausanne.
- [Wak] John F. Wakerly. *Digital Design, Principles and Practices*. Prentice Hall, 2nd edition edition.