Repetitorium zur Vorlesung **DIGITALTECHNIK**

Karlsruher Institut für Technologie (KIT)
Wintersemester 2010/11
(1. Semester)

Dozenten:

PROF. DR.-ING. J. BECKER
DIPL.-ING. J. HEIßWOLF

Stand: 12. Februar 2012
Version 3.0

von
Lukas Kölsch
lukaskoelsch@gmx.de | lukas.koelsch@student.kit.edu

Hinweis:

Dieses Dokument ist eine Art Kurz-Zusammenfassung der Vorlesungsinhalte zum Kernfach „Digitaltechnik“ im ersten Semester und beinhaltet *weder* Herleitungen dieser Inhalte *noch* eine etwaige Einschränkung/ Einordnung des Gezeigten. Es eignet sich also weniger zum Verständnis des Stoffes (dafür ist die Vorlesung samt Vorlesungsfolien oder gleichnamigem Buch zuständig), sondern eher zum schnellen Überblick vor und während der Klausurlernphase, dahingehend ob alle Inhalte verstanden wurden, nimmt also bestenfalls eine Art Zwischenstellung zwischen Formelsammlung und Lehrbuch ein.

Obwohl ich das Dokument ursprünglich nur zu meinem Privatgebrauch erstellt habe, nehme ich sämtliche Hinweise auf Fehler oder unvollständige Inhalte natürlich jederzeit gerne entgegen (Emailadresse siehe Titelseite).

Da dieses Repetitorium nicht von „offizieller Seite“ geprüft/ verifiziert wurde, kann ich für die vollständige Überdeckung aller Vorlesungsinhalte leider nicht garantieren. Es würde mich natürlich trotzdem freuen, wenn es euch einen Mehrwert - beispielsweise bei der Vorbereitung auf die Klausur - böte.

Inhaltsverzeichnis

Kapitel 1

1 Informationsübertragung

1.1 Funktion und Struktur

- mehrstufige Aufteilung von Systemen, um die Komplexität zu reduzieren
- zwei Transformationsrichtungen

  Top-Down-Spezifikation = Synthese-Schritt: Implementierung zunehmend, mehr Detailreichtum

  Bottom-Up-Spezifikation = Analyse-Schritt: Spezifikation zunehmend, mehr Abstraktion
- Entwurf heute weitgehend gemäß Top-Down-Prinzip, einzelne Komponenten werden vorher im Bottom-Up-Verfahren technisch optimal gestaltet und in den Top-Down-Weg einbezogen
- Abstraktionsebenen

  Systemebene: Gesamte Anwendung (z.B. Rechensystem)

  Modulebene: Elementare Digitalfunktionen (z.B. Addierer)

  Logikebene: Elementare Logikgatter (z.B. UND-Gatter)

  Transistorebene: z.B. Inverter, Negationsglied

  Physikalische Ebene: Transistorintegration mit Halbleitermaterialien (z.B. Silicium)

1.2 Nachricht und Signal

- *Formalisierung* von Information, um Informationsspielraum zu minimieren
- *Signal* = Darstellung von Nachrichten oder Daten mit physikalischen Mitteln

  kontinuierliches Signal: physikalisch und messtechnisch *nicht* realisierbar

  zeitdiskretes, wertkontinuierliches Signal: diskrete Abtastzeitpunkte $\Longrightarrow$ Messungenauigkeiten

  zeitkontinuierliches, wertdiskretes Signal: z.B. Quanteneffekte

  zeitdiskretes, wertdiskretes Signal =:*digitales* Signal
- *Problem:* exakte Diskrimination an Intervallgrenzen $\Longrightarrow$ *besser:* weiche Diskrimination durch willkürliche Digitalwertzuordnung mittels *undefinierter Bereiche*

- Signaldarstellung mittels *Binärsignalen:*

$$\text{Binärsignal:} \quad x \in \{L, H\} \tag{1.1}$$
$$\text{Binärvektoren:} \quad X = (x_n, \ldots, x_1) \tag{1.2}$$
$$\text{Kombinationen pro Binärstelle:} \quad 2^n \tag{1.3}$$
$$\text{inverse Funktion:} \quad n = \log_2 N = \mathrm{ld} N \tag{1.4}$$
$$\text{Darstellung von } N \text{ Werten:} \quad n = \lceil \mathrm{ld} N \rceil \tag{1.5}$$

1.3 Informationsgehalt

- *Annahme:* ein Zeichen trägt umso mehr Information, je seltener es beim Empfänger eintrifft: Informationsgehalt H_e eines Zeichens steigt $\Longleftrightarrow$ Auftrittswahrscheinlichkeit p dieses Zeichens nimmt ab

$$\text{Informationsgehalt eines Zeichens } e: \qquad H_e = \mathrm{ld}\left(\frac{1}{p}\right) \qquad [e] = 1 \text{ Bit} \tag{1.6}$$

- Also: Einheit 1 Bit entspricht der Elementaren Entscheidung zwischen zwei gleichwahrscheinlichen Möglichkeiten ($p = 0,5$)
- Betrachtung nicht gleichwahrscheinlicher Zeichen. Alphabet mit N Zeichen

$$\text{es gilt:} \qquad \sum_{i=1}^{N} p(i) = 1 \tag{1.7}$$

- In einer Folge mit mit L Zeichen ist die zu erwartende Auftrittshäufigkeit eines speziellen Zeichens i:

$$L \cdot p(i) \tag{1.8}$$

- Alle beobachteten Zeichen i liefern den gesamten *Informationsbeitrag:*

$$L \cdot p(i) \cdot H_{ei} = L \cdot p(i) \cdot \mathrm{ld}\frac{1}{p(i)} \tag{1.9}$$

- durchschnittlicher Informationsgehalt pro Zeichen:

$$\text{Entropie der Quelle:} \qquad H = \sum_{i=1}^{N} \left(p(i) \cdot \mathrm{ld}\left(\frac{1}{p(i)}\right)\right) \tag{1.10}$$

- Bei Gleichverteilung gilt $H = H_e$
- Die SHANNONsche Funktion $H = p \cdot \mathrm{ld}\left(\frac{1}{p}\right) + (1-p) \cdot \mathrm{ld}\left(\frac{1}{1-p}\right)$ ist für $p = 0.5$ maximal

Kapitel 2

2 Codierung

2.1 Codierung und Fehlerkorrektur

- *Code* = Vorschrift für eindeutige Zuordnung (*Codierung*), muss nicht bijektiv sein
- Intention bei der Codierung:
  1. Übertragungssicherheit
  2. gekipptes Bit muss erkannt/ korrigiert werden können
- *Codewörter sind elementare Einheiten zur Darstellung von Information*
- Maximale Anzahl N von m-stelligen Codewörtern:

$$N \leq 2^m \tag{2.1}$$

- Anzahl strukturierter m-stelliger Codewörter mit k Einsen:

$$\left(\begin{array}{c} m \\ k \end{array} \right) = \frac{m!}{k!\,(m-k)!} \tag{2.2}$$

- Problemstellen bei der *Codewandlung:* Übergänge, bei denen sich mehr als eine Binärstelle ändert. Beim Wechel müssen nicht alle Binärstellen gleichzeitig wechseln (dadurch fehlerhafte Übergänge in andere Codewörter möglich)
- *Definition:* Hamming-Distanz HD_{ij}

  Seien die Codewörter CW_i, $CW_j \in \{0,1\}^n$,
  HD_{ij} = Anzahl der Stellen, an denen sich CW_i und CW_j unterscheiden.
  Dann heißt HD_{ij} die *Hamming-Distanz* von CW_i und CW_j
- *Definition:* Minimale Hamming-Distanz HD_{min}

  Sei $X \subseteq \{0,1\}^n$ beliebig und
  $HD_{min} := \min \{CW_i, CW_j \in X \wedge CW_i \neq CW_j\}$
  Dann heißt HD_{min} *minimale Hamming-Distanz* von X
- *Gray-Code:* einschrittiger Code zur A/D-Wandlung. Konstruktion:
  1. Gray-Code mit $m = x$ Binärstellen liegt vor (vgl. Tabelle 2.5 auf Seite 12)
  2. in umgekehrter Reihenfolge an gegebenen Code anhängen (Spiegelung an der Horizontalen)

LSB	**MSB**							
Binär	000	001	010	011	100	101	110	111
	Steuerzeichen				Großbuchstaben		Kleinbuchstaben	
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	„	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Tabelle 2.1: ASCII-Code

3. zusätzliche Binärstelle anfügen (zuerst $\frac{2^{x+1}}{2}$ Nullen/Einsen, dann $\frac{2^{x+1}}{2}$ Einsen/Nullen). Dies lässt sich beliebig fortsetzen

- *ASCII-Code:* Kodierung von 128 Zeichen durch 7 Bit:
- Problem in der Praxis: Störeinflüsse verfälschen den Wert der zur Darstellung verwendeten physikalischen Größe
- einzelne Bitfehler sind erkennbar bei Codes mit $HD_{min} = 2$
- bei $HD_{min} = 3$: Zweifachfehler erkennbar oder Einfachfehler korrigierbar
- Allgemein gilt: Geeignete Codes können Fehler erennen und sogar korrigieren. Notwendig hierzu: Hinzufügen *redundanter* Informationen
- weitere Möglichkeit: Hinzufügen eines *Paritätsbits* - auch zweidimensional anwendbar (Blocksicherung)
- *also:* m Informationsbits werden k *Fehlerschutzbits* als Redundanz angehängt und übertragen. Damit Vergrößerung des Datenstroms auf die Länge $n = m + k$

$$\text{Relative Redundanz:} \qquad r = \frac{2^n - \#gCW}{2^n} \tag{2.3}$$

2.2 Nachrichtenübertragung und optimale Codes

- *Zeichenverflechtung/Scrambling*: Binärstellen eines Binärworts werden nicht nacheinander übertragen $\Longrightarrow$ länger anhaltender Störeinfluss (*Bündelstörung*) bestimmter Länge verfälscht nur Binärstellen von unterschiedlichen Codewörtern
- *optimale Codes* versuchen die mittlere Codewortlänge $\overline{m}$ zu minimieren. Der minimal erreichbare Idealwert ist dabei die Entropie der Sendequelle

$$\text{mittlere Codewortlänge:} \qquad \overline{m} = \sum_{i=1}^{n} p\left(x_i\right) \cdot m\left(x_i\right) \tag{2.4}$$

- *Shannon-Fanø-Code*
  1. Zeichenvorrat nach aufsteigender Wahrscheinlichkeit linear sortieren
  2. Zwei Teilmengen so konstruieren, dass die Summenwahrscheinlichkeiten der beiden Teilmengen möglichst gleich groß sind
  3. Die erste Teilmenge erhält das Codierungszeichen „0“ und die zweite Teilmenge eine „1“
  4. mit den jeweils resultierenden Teilmengen rekursiv in gleicher Weise fortfahren bis nur noch ein Zeichen in resultierenden Teilmengen enthalten ist
- *Huffman-Code*
  1. Sortierte Liste Q: Jeweils die beiden Zeichen in Q mit den niedrigsten Aufenthaltswahrscheinlichkeiten finden
  2. Zwei gefundene Zeichen werden zu neuem Element z verschmolzen, dessen AWS die Summe der beiden Einzelwahrscheinlichkeiten zugeordnet wird
  3. Einsortierung von z in Q entsprechend seiner AWS
  4. Abbruch des Algorithmus, sobald nur noch ein Objekt in Q übrig ist (*Codierungsbaum*)
- Sowohl Shannon-Fanø- als auch Huffman-Codes sind *präfixfreie* Codes
- es gilt $\overline{m}_{\text{Huffman}} \leq \overline{m}_{\text{Fanø}}$

2.3 Hamming-Codes

- Sei $X \subseteq \{0,1\}^n$ ein Code mit $HD_{min} =: d$. Dann sind bis zu $d-1$ Fehler erkennbar
- Sei $HD_{min} =: d = 2e+1$. Dann sind bis zu $e = \frac{d-1}{2}$ Fehler korrigierbar
- gegeben: m Nutzbits und k Prüfbits $\Longrightarrow 2^m$ gültige Codewörter
- *Beispiel:* Forderung: 1-F-Korrigierbarkeit

- pro gültigem CW: n ugCWs mit $HD = 1$
- *also:* $n + 1$ Tatbestände pro gültigem CW, in 2^k Zuständen codierbar

$$(n + 1) \leq 2^k \tag{2.5}$$

- *also*:

$$(n + 1) \leq 2^k \Longleftrightarrow (m + k + 1) \quad \leq 2^k \Longleftrightarrow k \geq \mathrm{ld}(m + k + 1) \tag{2.6}$$

k=minimale Anzahl von Prüfbits, um eine 1-F-Korrigierbarkeit umzusetzen

$$\Longleftrightarrow k = \lceil \mathrm{ld}(m + k + 1) \rceil \tag{2.7}$$

- analog gilt für die 2-F-Korrigierbarkeit:

$$2^k \geq \overbrace{1}^{\text{fehlerfrei}} + \overbrace{(m + k)}^{\text{1 Fehler}} + \overbrace{\binom{m+k}{2}}^{\text{2 Fehler}} \tag{2.8}$$

- *Hamming-Code:* Prüfstelle y_i überprüft alle Binärstellen (dies können sowohl Informationsstellen m als auch Prüfstellen k sein), die in der i-ten Stelle (im i-tem Bit) der Dualzahlendarstellungen der entsprechenden Binärstellenpositionen (*duale Kennzahlen*) eine „1“ aufweisen
- *also:* jedes Nutzbit wird durch mindestens 2 Prüfbits überprüft

2.4 Zahlensysteme

- Stellenwerte einer Zahl entsprechen den Potenzen der Basis des jeweiligen polyadischen Zahlensystems

$$\text{Aufbau einer Zahl } N\text{:} \quad N = d_n \cdot R^n + \ldots + d_1 \cdot R^1 + d_0 \cdots R^0 = \sum_{i=0}^{n} d^i \cdot R^i \tag{2.9}$$

N :Zahl im Zahlensystem
R :Basis (Radix) mit $R \geq 2$
R^i :Wertigkeit der i-ten Stelle
d_i :Ziffer der Stelle i
Z :Menge der Ziffern: $d_i \in Z = \{0, 1, 2, \ldots, R - 1\}$

- Wenn ein Ziffernvorrat in einer Stelle erschöpft ist, erfolgt ein Übertrag in nächsthöhere Stelle
- Wandlung in beliebige Formel

1. Wandlung ins Dezimalsystem gemäß Formel (2.9)
2. Wandlung aus dem Dezimalsystem: Rekursion (N_D wird durch R geteilt (Ergebnis: Stelle d_0, anschließend entstehende Quotienten rekursiv nach gleichem Schema verarbeiten)

- Rechenoperationen im Dualsystem
  - Addition: $0+0=0$, $0+1=1+0=1$, $1+1=10$, $1+1+1=11$
  - Subtraktion:

$$a-b=\underbrace{\left(a+\underbrace{(11111111_B-b+1)}_{c}\right)}_{d}-100000000_B \tag{2.10}$$

  - $(d-1000000_B)$ berechnet man wie folgt: wenn Bit 9 vor d gesetzt ist ($\Longrightarrow$ Ergebnis positiv): Bit 9 abschneiden. Ansonsten ($\Longrightarrow$ Ergebnis negativ): Bildung des Betrags mittels Zweierkomplement
  - in K_2-Darstellung erkennt man eine negative Zahl immer an der führenden Eins

- Binary Coded Decimal-Code (BCD-Code)
  - jede Dezimalziffer wird durch eine 4-Bit-Dualzahl (*Tetrade*) dargestellt
  - Addition: falls *Pseudotetraden* oder *Überträge* auftreten: Addition von $6_D = 0110_B$
  - jede Tetrade darf maximal *ein mal* korrigiert werden

- Stibitz-Code/Exzess-3-Code
  - alle Tetraden sind im Vergleich zum BCD-Code um $3_D = 0011_B$ größer $\Longrightarrow$ symmetrische Anordnung der Ziffern:

Dezimal	0	1	2	3	4
Stibitz	0011	0100	0101	0110	0111
Stibitz	1000	1001	1010	1011	1100
Dezimal	5	6	7	8	9

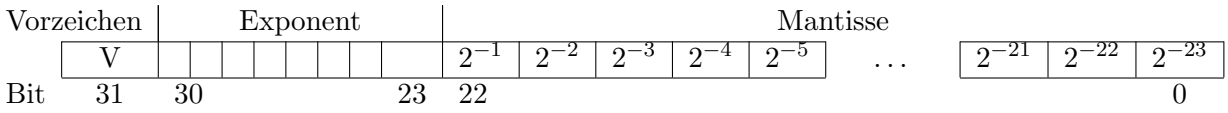
Tabelle 2.2: Stibitz-Code

  - *Addition:* nach Zwischenergebnis muss, wenn kein Übertrag auftritt, 0011_B addiert, ansonsten 0011_B subtrahiert/ 1101_B addiert werden

- Aiken-Code/ 2-4-2-1-Code: ebenfalls symmetrische Anordnung, um eine einfache Komplementbildung zu ermöglichen

Dezimal	0	1	2	3	4
AIKEN	0000	0001	0010	0011	0100
AIKEN	1111	1110	1101	1100	1011
Dezimal	9	8	7	6	5

Tabelle 2.3: AIKEN-Code



Exponent E	Mantisse M	Wert
255	$\neq 0$	ungültig(NAN)
255	0	$-1V \cdot \infty$ ($\pm$unendlich)
$\boxed{0 < E < 255}$	M	$\boxed{-1^V \cdot 2^{E-127} \cdot (1, M)}$
0	$\neq 0$	$-1^V \cdot 2^{-126} \cdot (0, M)$
0	0	$-1^V \cdot 0$

Tabelle 2.4: Gleitkommazahlen gemäß IEEE-754

- *Gleitkommazahlen* zur Repräsentation reeller Zahlen $N_{GK} \in \mathbb{R}$:
- Darstellbarer Zahlenbereich bei 32Bit: $-2^{127} \cdot \text{Mant} \leq N_{GK} \leq 2^{127} \cdot \text{Mant}$
- Addition von Gleitkommazahlen
  1. Exponenten normalisieren: GK-Zahl mit kleinerem Exponenten an die GK-Zahl mit dem größeren Exponenten anpassen ($\Longrightarrow$ Mantisse M_B um $(E_A - E_B)$ Stellen nach rechts schieben
  2. Mantisse wie Festkommazahlen nach den bekannten Regeln addieren
  3. ggf. Normierung der Ergebnismantisse (bei Überträgen die höchstwertige Stelle, anschließend Mantisse M_Y normieren: $1 \leq \|M_Y\| \leq 2$, und Exponent anpassen:

$$\begin{aligned} Y = A + B &= M_A \cdot 2^{E_A} + M_B \cdot 2^{E_B} = M_A \cdot 2^{E_A} + M_B \cdot 2^{-(E_A - E_B) < 2^{E_A}} \\ &= M_A \cdot 2^{E_A} + M'_B \cdot 2^{E_A} = \left(M_A + M'_B\right) \cdot 2E_A \\ &= M_Y \cdot 2^{E_Y} \end{aligned} \tag{2.11}$$

2.5 Codewandlung/Umschaltung

- *Beispiele* für Codewandlung: BCD $\Longrightarrow$ 7-Segment (Anzeige auf Displays), ASCII $\Longrightarrow$ BCD für Dezimalzahlen $0 \ldots 9$, GRAY-Code nach Dualzahl:

GRAY-Code	000	001	011	010	110	111	101	100
Dualzahl	000	001	010	011	100	101	110	111

Tabelle 2.5: GRAY-Code nach Dualzahl

- *Codeumschaltung:* Codewörter werden mehrfach belegt, ihre Bedeutung ist dannn durch spezielle *Umschaltzeichen* änderbar. Es gibt drei Arten von Codewörtern:
  1. Codewörter, deren Bedeutung umgeschaltet wird
  2. spezielle Codewörter, die eine Umschaltung der Zeichengruppen bewirken
  3. Codewörter, die *unabhängig* von der Umschaltung immer gleich sind
- die notwendige Anzahl der speziellen Umschalt-Codewörter entspricht der Anzahl der Gruppen von Zeichen. Es gilt:

$$N' = 2^m - i - j \tag{2.12}$$

$$N'' = J \cdot N' + i \tag{2.13}$$

N :Zahl der Codewörter

m :Anzahl der Binärstellen der Codewörter

i :Zahl der allen Gruppen gemeinsamen Zeichen

j :Zahl der Gruppen (und damit Anzahl der Umschaltzeichen)

N' :Zahl der nutzbaren Codewörter je Gruppe

N'' :Zahl der mit Umschaltung darstellbaren Zeichen

Kapitel 3

3 Mathematische Grundlagen

3.1 Mengen

- *Menge* = Zusammenfassung endlich oder unendlich vieler bestimmter, wohldefinierter, wohlunterschiedener Objekte
- Angabe von Mengen
  - Angabe der einzelnen Elemente: $M_1 = \{x_1, x_2, x_3\}$
  - platzsparende Angabe für einfache Folgen: $M_2 = \{1, 2, 3, \ldots, 100\}$
  - Konstruktionsvorschrift: $M_3 = \{x | x^2 + 3x + 1 = 0\}$
- *Kardinalität* = Anzahl der Elemente einer Menge: $|M_1| = 3$
- *Leere Menge:* enthält keine Elemente: $\emptyset = \{\}$, $|\emptyset| = 0$
- *Untermenge:* $(x \in M \Longrightarrow x \in N) \Longleftrightarrow M \subseteq N$
- *echte Untermenge:* $(x \in M \Longrightarrow x \in N) \wedge (M \neq N) \Longleftrightarrow M \subset N$
- *Potenzmenge* = Menge der Untermengen: $\mathcal{P}(X) := \{U | U \subseteq X\}$; Sonderfälle: $\mathcal{P}(\emptyset) = \{\emptyset\}$, $|\mathcal{P}(\emptyset)| = 1$
- *Vereinigung* und *Durchschnitt*:

$$V = S \cup T := \{x | x \in S \text{ oder } x \in T\} \tag{3.1}$$

$$V = S \cap T := \{x | x \in S \text{ und } x \in T\} \tag{3.2}$$

- für *disjunkte* Mengen gilt: $S \cap T = \emptyset$, $|S \cap T| = 0$, $|S \cup T| = |S| + |T|$
- *Komplementmenge* $C_M(S)$ $(S \subseteq M)$ = Menge aller Elemente von M, die nicht zu S gehören
- n-faches *kartesisches Produkt* S^n der Menge S mit sich selbst ist die Menge aller *geordneten* n-Tupel $(r, s, \ldots, z)$ mit $r \in S$, $s \in S$, ..., $z \in S$

$$S^n = \underbrace{S \times S \times \ldots \times S}_{n\text{-mal}} \qquad |S^n| = |S|^n \tag{3.3}$$

Typ	refl.	symm.	antisymm.	trans.	Beispiel
Ordnungsrelation	X		X	X	$=, \leq$
strenge Ordnungsrelation			X	X	$<, >$
Äquivalenzrelation	X	X		X	$=, \equiv$
Verträglichkeitsrelation	X	X			$\sim$

Tabelle 3.1: Typen von Relationen

Überdeckende Mengen $\in \tau$	Überdeckte Größen $\in M$			
	f_1	f_2	...	f_j
G_1				
G_2				
$\vdots$				
G_j				ist $f_j \in G_j$?

Tabelle 3.2: Struktur der Überdeckungstabelle

3.2 Relationen

- zweistellige Relation α zwischen zwei Mengen X und Y setzt fest, ob $x \in X$ in Beziehung zu $y \in Y$ steht: $x\alpha y$ oder $x\bar{\alpha}y$
- Eigenschaften von Relationen

  Reflexivität: für beliebige $x \in X$ gilt $x\alpha x$

  Symmetrie: aus $x\alpha y$ folgt $y\alpha x$ (Beispiel: „=“-Relation)

  Antisymmetrie: aus $x\alpha y$ und $y\alpha x$ folgt $x = y$

  Transitivität: aus $x\alpha y$ und $y\alpha z$ folgt $x\alpha z$
- Typen von Relationen: *(siehe Tabelle 3.1)*
- *Überdeckungsrelation:* bestimmtes Element f_j wird von einer Teilmenge $G_j \subseteq M$ überdeckt $\longrightarrow$ eine Relation zwischen der Menge der Elemente aus M und der Menge der Teilmengen τ, Darstellung der Überdeckungsrelation als $\tau \times M$-Matrix (siehe Tabelle 3.2)

3.3 Graphentheorie

- *Graph* = Repräsentation von Beziehungen durch Kanten und Knoten
- *abstrakter Graph* $G(V, E, \Phi)$ mit V = Menge der Knoten, E = Menge der Kanten, Inzidenzabbildung Φ
  - *ungerichteter* Graph: $\Phi : E \longrightarrow \{V, V\}$, $e \in E \longmapsto \{v_1, v_2\} \in V$

  - *gerichteter* Graph/ Digraph: $\Phi : E \longrightarrow V \times V$, $e \in E \longmapsto (v_1, v_2) \in V$
  - sobald eine Kante gerichtet ist, ist der komplette Graph gerichtet

$$\text{Inzidenzmatrix:} \qquad a_{ij} := \begin{cases} 0, \text{falls } e_j \textit{ nicht} \text{ inzident zu } V_i \\ 1, \text{falls } e_j \text{ inzident zu } V_i \end{cases} \tag{3.4}$$

$$\text{Adjazenzmatrix:} \qquad a_{ij} := \begin{cases} 1, \text{falls } \{v_i, v_j\} \in E \\ 0, \text{sonst} \end{cases} \tag{3.5}$$

$$\text{allgemein gilt:} \qquad \forall_j : \sum_i a_{ij} = 2 \tag{3.6}$$

- *endlicher* Graph: V und E sind endliche Mengen
- *entarteter* Graph: $E = \emptyset$, Graph besteht ausschließlich aus isolierten Knoten
- *einfacher* Graph: zu je zwei verschiedenen Knoten existiert höchstens eine Kante
- *Isomorphie* zweier Graphen: es existiert eine eineindeutige Abbildung zwischen Kanten und Knoten zweier Graphen, sodass die Inzidenzabbildungen erhalten bleiben
  - *Frage:* $G_1 \overset{!}{=} G_2$. *Gesucht:* Eineindeutige Abbildung $\lambda : \Phi_1 \to \Phi_2$
  - Kardinalität überprüfen: $|V_1 \overset{!}{=} V_2|$ und $|E_1 \overset{!}{=} E_2|$
- *Clique/ vollständiger* Graph: je zwei verschiedene Knoten sind durch eine Kante verbunden
  - vollständiger Subgraph: $(V', E'), V' \in V, E' \in E$
  - es gilt: $\forall_{v_1 \in V'} \exists_{e_1 \in E} : (v_i, v_j) \forall_{v_j} \in V' \backslash \{v_i\}$
- *zusammenhängender* Graph: von jedem beliebigen Knoten des Graphen kann man zu jedem anderen Knoten des Graphen gelangen
- *streng zusammenhängender* Graph: von jedem beliebigen Knoten eines gerichteten Graphen gibt es zu jedem anderen Knoten eine gerichtete Folge von Kanten
- *Schlinge:* Kante, die einen Knoten mit sich selbst verbindet: $\Phi(e) = (g, g)$
- *Mehrfachkanten:*
  - *parallele* Kanten: $\Phi(e) = \Phi(f) = \{g, h\}$ bzw. (g, h) mit $e \neq f$
  - *antiparallele* Kanten: $\Phi(e) = (g, h)$ und $\Phi(f) = (h, g)$
- *Grad* $d(g)$ eines Knotens: Anzahl der damit adjazenten Kanten. Bei gerichteten Graphen:
  - *Ausgangsgrad* $d^+(g)$: Anzahl der abgehenden Kanten
  - *Eingangsgrad* $d^-(g)$: Anzahl von ankommenden Kanten

Ungerichteter Graph		**Gerichteter Graph**	
$g^1 \neq g^{n+1}$	$g^1 = g^{n+1}$	$g^1 \neq g^{n+1}$	$g^1 = g^{n+1}$
offene Kanten-progression	geschlossene Kanten-progression	offene Kanten-progression	geschlossene Kanten-progression
sind alle Kanten einer Progression verschieden:			
Ketten-progression	geschlossene Kantenzugprogression	Weg-progression	Zyklus-progression
berücksichtigt man die Kanten einer Progression ohne Ordnung:			
Kette	geschlossener Kantenzug	Weg	Zyklus

Tabelle 3.3: Spezielle Kantenfolgen in Graphen

- *Unmittelbare Nachbarschaft* zweier Knoten: Knoten sind durch eine Kante verbunden ($V'(g)$=Menge der unmittelbar benachbarten Knoten). Bei gerichteten Graphen:
  - $V_1'(g)$: Menge der unmittelbaren Vorgänger
  - $V_2'(g)$: Menge der unmittelbaren Nachfolger

$$|V_1'(g)| = d^-(g) \tag{3.7}$$
$$|V_2'(g)| = d^+(g) \tag{3.8}$$
$$V' = V_1' \cup V_2' \tag{3.9}$$
$$|V'(g)| \leq d(g) \tag{3.10}$$

- *Mittelbare Nachbarschaft* zweier Knoten: Knoten sind durch eine Folge von Kanten verbunden
- *Artikulation* eines Graphen: Knoten, durch dessen Ausfall der Graph in zwei Subgraphen zerfallen würde ($\longrightarrow$ kritische Systemstelle)
- *Kantenprogression* der Länge n: endliche Folge von n (nicht notwendigerweise verschiedenen) Kanten e^i, die $n+1$ (nicht notwendigerweise verschiedene) Knoten g^i verbinden:

$$\Phi\left(e^i\right) = \left(g^i, g^{i+1}\right) \qquad \text{für } i = 1, 2, \ldots n \tag{3.11}$$

  - *einfache* Kantenprogression: alle Knoten in einer Kantenprogression sind verschieden
  - Euler-*Pfad*: Pfad (Folge von Kanten), der jede Kante des Graphen genau ein mal benutzt
  - Euler-*Tour*: Euler-Pfad, wobei Startknoten = Endknoten

- *zyklischer [azyklischer]* Graph: es existiert mindestens eine [keine] geschlossene Kantenzugprogression (jede Schleife zählt als Zyklus)
- *Baum:* zusammenhängender, zyklenfreier Graph
  - *gerichteter* Baum: es darf *kein* Zyklus existieren, der zugehörige ungerichtete Graph muss zusammenhängend sein
  - *Wurzel:* Knoten, der keine Vorgänger hat: $d^- = \emptyset$
  - *Blatt:* Knoten, der keine Nachfolger hat: $d^+ = \emptyset$
  - *Binärbaum:* jeder Knoten des Baumes (außer den Blättern) hat genau zwei Nachfolger: $d^+(g) = 2$
  - *symmetrischer* Binärbaum: Abstand von der Wurzel zu den Blättern ist für alle Blätter identisch
  - *Tiefe* des Baumes: Abstand von der Wurzel zu den Blättern *(nur bei Binärbäumen)*
  - *ausgeglichener* Baum: Die Abstände von der Wurzel zu den Blättern unterscheiden sich um maximal eins
  - *minimal aufspannender* Baum: Auflösung aller Zyklen durch gezieltes Entfernen von Kanten, sodass der Graph gerade nicht mehr in zwei Teilgraphen zerfällt
- *bipartiter* Graph: Knoten eines Graphen sind in zwei Teilmengen aufteilbar, sodass keine Kante zwei Knoten in derselben Teilmenge verbindet:
  1. $V = V_1 \cup V_2$, wobei $V_1 \cap V_2 = \emptyset$
  2. $\Phi(e) = (g, h)$ oder $\Phi(e) = (h, g)$, wobei $e \in E$, $g \in V_1$, $h \in V_2$
- *geometrischer* Graph: jedem Knoten wird eine geometrische Position im $n-$dimensionalen Raum $\mathbb{R}^n$ zugeteilt, Kanten dürfen keinen Punkt gemeinsam haben. Zu jedem Graphen existiert ein isomorpher geometrischer Graph in $\mathbb{R}^3$
- *planarer* Graph: Graph, zu dem ein isomorpher, zweidimensionaler Graph existiert. Sie müssen nicht kreuzungsfrei dargestellt sein
- *dualer* Graph: bei einem planaren Graphen werden äußere Umgebungen (*Gebiete*) in Knoten und Grenzen zwischen zwei Gebieten in Kanten umgewandelt. Weiterhin wird jedem eigenständigen zusammenhängenden Graphen ein Umgebungsknoten U zugeordnet
- *gewichteter* Graph: jeder Kante werden zusätzliche Attribute zugeordnet

3.4 Petrinetze

- *Petrinetz* = bipartiter, gerichteter Graph $P = (S, T, F, M)$, mit

  - S : Menge der *Stellen* (= Bedingungen, Situationen)
  - T : Menge der *Transitionen* (= Aktionen), $S \cap T = \emptyset$, $S \cup T \neq \emptyset$
  - F : *Kantenmenge*, mit $F \subseteq S \times T \cup T \times S$ ($s \longmapsto t$: Transition schaltet, wenn s erfüllt, s ist Vorstelle von t; $t \longmapsto s$: Zustand s wird erreicht, sobald die Transition geschaltet hat, s ist Nachstelle von t)
  - M : *Anfangsmarkierung:* $M : S \longrightarrow N$ (N= Anzahl Token pro Stelle)
  - *Eckenmenge* $E = S \cup T$

- Transition ist schaltbereit, sobald alle Eingangsstellen eine Markierung tragen und alle Ausgangsstellen leer sind (sofern sie nicht gleichzeitig Eingangsstellen sind)

  ↪ *Schaltbedingung:* alle Eingangsstellen besitzen mindestens einen Token

- besitzt eine Transition t mehrere Vorstellen, müssen alle Bedingungen erfüllt sein, damit t schalten kann. Besitzt eine Transition t mehrere Nachstellen, so werden durch das Schalten von t alle Nachstellen gleichzeitig erreicht

  ↪ *Schaltwirkung:* von allen Eingangsstellen einen Token entfernen, allen Ausgangsstellen einen Token zuordnen

- *Vorbereich* eines Netzelements x: $\bullet x = \{y | (y, x) \in F\}$
- *Nachbereich* eines Netzelements x: $x\bullet = \{y | (x, y) \in F\}$
- *Verzweigungen:* ein Knoten heißt
  - *vorwärtsverzweigt*, falls $|x \bullet| > 1$
  - *rückwärtsverzweigt*, falls $|\bullet x| > 1$

3.5 Algebraische Strukturen/ Boolesche Algebra

- Regelsystem der Mengenalgebra $MA = [\mathcal{P}(M), \cap, \cup, C_M, \emptyset, M]$:

  Abgeschlossenheit der Operationen *Durchschnitt* und *Vereinigung*:

$$\text{M1:} \qquad S \cap T \subseteq M \qquad S \cup T \subseteq M \tag{3.12}$$
$$S \cap T \in \mathcal{P}(M) \qquad S \cup T \in \mathcal{P}(M) \tag{3.13}$$

  Kommutativität:

$$\text{M2:} \qquad S \cap T = T \cap S \qquad S \cup T = T \cup S \tag{3.14}$$

  Distributivität:

$$\text{M3a:} \quad R \cap (S \cup T) = (R \cap S) \cup (R \cap T) \tag{3.15}$$
$$\text{M3b:} \quad R \cup (S \cap T) = (R \cup S) \cap (R \cup T) \tag{3.16}$$

Existenz neutraler Operationen $\cap$, $\cup$ bezüglich der Bezugsmenge M bzw. der leeren Menge $\emptyset$:

$$\text{M4:} \qquad S \cap M = S \qquad S \cup \emptyset = S \tag{3.17}$$

Existenz komplementärer Elemente S und $C_M(S)$ bezüglich der Operationen $\cap$, $\cup$:

$$\text{M5:} \qquad S \cap C_M(S) = \emptyset \qquad S \cup C_M(S) = M \tag{3.18}$$

- Regelsystem der Booleschen Algebra $BA = [K, \top, \bot, \neg, O, I]$
  - ist $\mathcal{P}(M)$ die Potenzmenge einer beliebigen Menge M, so ist das Verknüpfungsgebilde $[\mathcal{P}(M), \cap, \cup, C_M, \emptyset, M]$ stets eine Boolesche Algebra
  - Bei jeder Booleschen Algebra gilt für die Menge K: $|K| = 2^n$ $(n = 1, 2, \ldots)$
  - HUNTINGTONsche Axiome $(a, b, c, I, O \in K)$:

$$\text{H1:} \qquad a \top b \in K \qquad a \bot b \in K \tag{3.19}$$

$$\text{H2:} \qquad a \top b = b \top a \qquad a \bot b = b \bot a \tag{3.20}$$

$$\text{H3:} \qquad (a \bot b) \top c = (a \top c) \bot (b \top c) \tag{3.21}$$

$$(a \top b) \bot c = (a \bot c) \top (b \bot c) \tag{3.22}$$

$$\text{H4:} \qquad I \top a = a \qquad O \bot a = a \tag{3.23}$$

$$\text{H5:} \qquad a \top k = O \qquad a \bot k = I \tag{3.24}$$

  - Dualitätsprinzip: Vertauscht man in einem Satz unter alleiniger Verwendung der Operatoren $\top$, $\bot$ und $\neg$ alle Operatoren $\top$ durch $\bot$ und umgekehrt, so erhält man wieder einen Satz (enthält der Satz die neutralen Elemente O und I, so müssen auch diese vertauscht werden)

- Beispiele für Interpretationen der HUNTINGTONschen Axiome:
  - Boolesche Algebra mit zwei Werten: $BA_2 = [\{0, 1\}, \top, \bot, \neg, 0, 1]$
  - Aussagenlogik: $AL = [\{\text{wahr, falsch}\}, \text{UND, ODER, NICHT}, \text{stets falsch, stets wahr}]$
  - Mengenalgebra mit zwei Elementen: $MA_2 = [\{\emptyset, M\}, \cap, \cup, C_M, \emptyset, M]$
  - Mengenalgebra allgemein: $[\mathcal{P}(M), \cap, \cup, C_M, \emptyset, M]$
- Schaltalgebraische Interpretationen der HUNTINGTONschen Axiome: $SA = [\{0, 1\}, \&, \vee, \neg, 0, 1]$

Regeln für 0 und 1:

$$\text{R1a:} \quad \overline{0} = 1 \qquad \text{R1b:} \quad \overline{1} = 0 \tag{3.25}$$

$$\text{R2a:} \quad 0 \vee 0 = 0 \qquad \text{R2b:} \quad 1 \& 1 = 1 \tag{3.26}$$

$$\text{R3a:} \quad 1 \vee 1 = 1 \qquad \text{R3b:} \quad 0 \& 0 = 0 \tag{3.27}$$

$$\text{R4a:} \quad 0 \vee 1 = 1 \qquad \text{R4b:} \quad 1 \& 0 = 0 \tag{3.28}$$

$\mathbf{MA_2}$	**HA**	**SA**
$\{\emptyset, M\}$	K	$\{0, 1\}$
$\cap$	$\top$	$\&$
$\cup$	$\bot$	$\vee$
C_M	$\neg$	$\neg$
$\emptyset$	O	0
M	I	1

Tabelle 3.4: Mögliche Interpretationen der HUNTINGTONschen Axiome

Regeln für einen Operanden a:

$$\text{R5a: } a \vee 0 = a \qquad \text{R5b: } a\&1 = a \tag{3.29}$$
$$\text{R6a: } a \vee 1 = 1 \qquad \text{R1b: } a\&0 = 0 \tag{3.30}$$
$$\text{R7a: } a \vee a = a \qquad \text{R7b: } a\&a = a \tag{3.31}$$
$$\text{R8a: } a \vee \overline{a} = 1 \qquad \text{R8b: } a\&\overline{a} = 0 \tag{3.32}$$
$$\text{R9: } \overline{(\overline{a})} = \overline{\overline{a}} = a \tag{3.33}$$

Assoziative Gesetze:

$$\text{R10a: } a \vee (b \vee c) = (a \vee b) \vee c = a \vee b \vee c \tag{3.34}$$
$$\text{R10b: } a\&(b\&c) = (a\&b)\&c = a\&b\&c \tag{3.35}$$

Absorptionsgesetze:

$$\text{R11a: } a \vee (a\&b) = a \qquad \text{R11b: } a\&(a \vee b) = a \tag{3.36}$$

DE MORGANsche Regeln:

$$\text{R12a: } \overline{a \vee b} = \overline{a}\&\overline{b} \qquad \text{R12b: } \overline{a\&b} = \overline{a} \vee \overline{b} \tag{3.37}$$

Kapitel 4

4 Schaltalgebra

4.1 Digitale Schaltfunktionen und Normalformtheoreme

- *Funktion* als Spezialfall der Relation, Definition durch Angabe aller Paare (s, t) oder durch Kurznotation $x \longmapsto f(x)$
- *Digitale Schaltfunktion*: $S = \{0,1\}^n$ und $T = \{0,1\} \Longrightarrow X = (x_n, \ldots, x_2, x_1)$, $X_i \in \{0,1\}$, $y \in \{0,1\}$
- Schaltfunktion lässt sich schreiben als $y = f(X) = f(x_n, \ldots, x_2, x_1)$, mit $x_n, \ldots, x_1$ als *unabhängigen* Variablen und y als *abhängiger* Variablen
- drei Teilmengen von Belegungen
  1. *Nullstellenmenge:* $N = \{X_j | X_j \longmapsto 0\}$
  2. *Einsstellenmenge:* $E = \{X_j | X_j \longmapsto 1\}$
  3. *Redundanzmenge:* $R = \{X_j | X_j \longmapsto -\}$ („don't care")
- *[un]vollständig definierte* Schaltfunktion: ordnet [nicht] allen Belegungen X_j einen Funktionswert aus $f_j \in \{0,1\}$ zu: $|N| + |E| = 2^n$ $[|N| + |E| + |R| = 2^n]$
- Anzahl möglicher Schaltfunktionen: $MF = \{0,1\}^{2^n}$
- Darstellung einer Schaltfunktion mittels
  1. Symmetriediagramm (vgl. Tabelle 4.1, Seite 22). Für ein neues Literal X_i gilt: $j_{\text{neu}} = j_{\text{alt}} + 2^{i-1}$
  2. *Binary Decision Diagram* (BDD) kanonische Darstellung von Funktionen $\Longrightarrow$ Isomorphie leicht überprüfbar
  3. *Funktionstabelle* (vgl. Tabelle 4.2): Spaltenzahl wächst mit n, Zeilenzahl wächst mit $2^n \Longrightarrow$ Darstellungstechnik nicht effizient
- mögliche Basissysteme der Schaltalgebra:

$$\text{Konjunktion } (\&): \qquad y = f(x_1, x_2) = \begin{cases} 1 & \text{für } x_1 = x_2 = 1 \\ 0 & \text{sonst} \end{cases} \tag{4.1}$$

$$\text{Disjunktion } (\vee): \qquad y = f(x_1, x_2) = \begin{cases} 0 & \text{für } x_1 = x_2 = 0 \\ 1 & \text{sonst} \end{cases} \tag{4.2}$$

				x_5				
		x_1				x_1		
	0	1	5	4	24	25	21	20
x_2	2	3	7	6	26	27	23	22
	12	13	17	16	36	37	33	32
	10	11	15	14	34	35	31	30
			x_3					

x_4

Tabelle 4.1: Symmetriediagramm für $n = 5$

x_2	x_1	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}	y_{17}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
		—	neg. Disjunktion $\triangledown$(NOR)	(neg. Implikation)	—	neg. Implikation	—	Antivalenz $\not\equiv$(XOR)	neg. Konjunktion $\overline{\&}$(NAND)	Konjunktion & (AND)	Äquivalenz $\equiv$	—	Implikation $\Longrightarrow$	—	(Implikation) $\Longleftarrow$	Disjunktion $\vee$ (OR)	—

Tabelle 4.2: Schaltfunktionen bei $n = 2$

- Numerische Interpolation nach LAGRANGE: (*geg:* (x_0, y_0), (x_1, y_1), (x_2, y_2) als *Stützstellen* von y)

$$P_L(x) = y_0 \cdot L_0(x) + y_1 \cdot L_1(x) + y_2 \cdot L_2(x) \tag{4.3}$$

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)},\ L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)},\ L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

für $i \neq j$ gilt: $L_i(x) \cdot L_j(x) = 0$ (orthogonale Basisfunktionen)

Grundsätzliche Konstruktionsvorschrift für Schaltfunktionen:

$$A_0 \cdot b_0(x) + A_1 \cdot b_1 + \ldots + A_{2^{n-1}} \cdot b_{2^{n-1}}(x) = \sum_{k=0}^{2^{n-1}} A_k \cdot b_k(x) \tag{4.4}$$

- allgemein gilt für beliebige n:
  - *Minterm:* für genau eine Belegung wird der Wert „1“ angenommen, Projektion von Einsstellen auf beliebige Positionen im Symmetriediagramm
  - *Maxterm:* für genau eine Belegung wird der Wert „0“ angenommen, Projektion von Nullstellen auf beliebige Positionen im Symmetriediagramm

$$\text{Trema-Funktion:} \quad \ddot{x} := \begin{cases} x_i & \text{wenn 1 an Position } x_i \text{ in der Belegung} \\ \overline{x_i} & \text{wenn 0 an Position } x_i \text{ in der Belegung} \end{cases} \tag{4.5}$$

$$\text{Minterm:} \quad m_j = \ddot{x}_n \& \ddot{x}_{n-1} \& \ldots \& \ddot{x}_2 \& \ddot{x}_1 \tag{4.6}$$

$$\text{Maxterm:} \quad M_j = \overline{\ddot{x}_n} \vee \overline{\ddot{x}_{n-1}} \vee \ldots \vee \overline{\ddot{x}_2} \vee \overline{\ddot{x}_1} \tag{4.7}$$

$$m_j \& m_k = 0 \qquad M_j \vee M_K = 1 \qquad j \neq k\ ,\ 0 \leq j\ ,\ k \leq n^2 - 1$$
$$\overline{m_j} = M_j \qquad \overline{M_j} = m_j$$
$$m_j \& M_j = 0 \qquad m_j \vee M_j = 1$$

- *gesucht:* Grundsätzliche Konstruktionsvorschrift für Schaltfunktionen unter Verwendung orthogonaler Minterm- und Maxtermbasisfunktionen:
  - Disjunktive Normalform (DNF):

$$\begin{aligned} y &= (f_{2^n-1} \& m_{2^n-1}) \vee (f_{2^n-2} \& m_{2^n-2}) \vee \ldots \vee (f_1 \& m_1) \vee (f_{2^n-1} \& m_{2^n-1}) \\ &= \bigvee_{j=0}^{2^n-1} (f_j \& m_j) \end{aligned} \tag{4.8}$$

– Konjunktive Normalform (KNF):

$$\begin{aligned} y &= (f_{2^n-1} \vee m_{2^n-1}) \,\&\, (f_{2^n-2} \vee m_{2^n-2}) \,\&\, \ldots \,\&\, (f_1 \vee m_1) \,\&\, (f_{2^n-1} \vee m_{2^n-1}) \\ &= \bigwedge_{j=0}^{2^n-1} (f_j \vee m_j) \end{aligned} \tag{4.9}$$

- *Hauptsatz der Schaltalgebra:* „Jede beliebige Schaltfunktion $y = f(x_n, \ldots, x_1)$ lässt sich als *Disjunktion* von *Mintermen* / *Konjunktion* von *Maxtermen* eindeutig darstellen. In der Disjunktion/ Konjunktion treten genau diejenigen Minterme/ Maxterme auf, die zu den Einsstellen/ Nullstellen der Schalftunktion gehören“

- aus dem Basissystem $[\&, \vee, \neg]$ lassen sich weitere Basissysteme mit nur zwei oder nur einem Operatoren herleiten:

  Basissystem NAND $\left[\overline{\&}\right]$: $\overline{x}$ wird mittels $\overline{x\&1}$ dargestellt

$$y = \bigvee_{j=0}^{2^n-1} (f_j \& m_j) = \overline{\overline{\bigvee_{j=0}^{2^n-1} (f_j \& m_j)}} = \overline{\bigwedge_{j=0}^{2^n-1} \overline{f_j \& m_j}} = \overline{\bigwedge_{j=0}^{2^n-1} \left(\overline{f_j} \vee \overline{m_j}\right)} = \overline{\bigwedge_{j=0}^{2^n-1} \overline{m_j}}\Bigg|_{f_j=1} \tag{4.10}$$

  Basissystem NOR $[\overline{\vee}]$: $\overline{x}$ wird mittels $\overline{x \vee 0}$ dargestellt

$$y = \bigwedge_{j=0}^{2^n-1} (f_j \vee M_j) = \overline{\overline{\bigwedge_{j=0}^{2^n-1} (f_j \vee M_j)}} = \overline{\bigvee_{j=0}^{2^n-1} \overline{f_j \vee M_j}} = \overline{\bigvee_{j=0}^{2^n-1} \left(\overline{f_j} \& \overline{M_j}\right)} = \overline{\bigvee_{j=0}^{2^n-1} \overline{M_j}}\Bigg|_{f_j=0} \tag{4.11}$$

4.2 Entwicklungssatz der Schaltalgebra

- Formelle Methode zur gezielten Umwandlung jedes beliebigen schaltalgebraischen Ausdrucks in eine der beiden Normalformen

  – zur Bildung der DNF:

$$\begin{aligned} f(x_n, \ldots, x_i, \ldots, x_1) &= \overbrace{[x_i \& f(x_n, \ldots, 1, \ldots, x_1)]}^{x_i=1} \vee \overbrace{[x_i \& f(x_n, \ldots, 0, \ldots, x_1)]}^{x_i=0} \\ &= \left[x_i \&\, f|_{x_i=1}\right] \vee \left[\overline{x_i} \&\, f|_{x_i=0}\right] \end{aligned} \tag{4.12}$$

  – zur Bildung der KNF:

$$\begin{aligned} f(x_n, \ldots, x_i, \ldots, x_1) &= \overbrace{[x_i \vee f(x_n, \ldots, 0, \ldots, x_1)]}^{x_i=1} \,\&\, \overbrace{[x_i \vee f(x_n, \ldots, 1, \ldots, x_1)]}^{x_i=0} \\ &= \left[x_i \vee f|_{x_i=0}\right] \,\&\, \left[\overline{x_i} \vee f|_{x_i=1}\right] \end{aligned} \tag{4.13}$$

- Die *Restfunktionen/ Kofaktoren* $f|_{x_i=1}$ und $f|_{x_i=0}$ lassen sich nach den weiteren Variablen entwickeln
- Beweis der beiden Fälle:

$$f(x_n, \ldots, x_i = 1, \ldots, x_1) = \left[1\& \; f|_{x_i=1}\right] \vee \left[\overline{1}\& \; f|_{x_i=0}\right] = f|_{x_i=1} \vee 0 = f|_{x_i=1} \tag{4.14}$$

$$f(x_n, \ldots, x_i = 0, \ldots, x_1) = \left[0\& \; f|_{x_i=1}\right] \vee \left[\overline{0}\& \; f|_{x_i=0}\right] = 0 \vee f|_{x_i=0} = f|_{x_i=0} \tag{4.15}$$

4.3 Minimierung am Symmetriediagramm

- Durch gezieltes Weglassen von Literalen in einem Minterm [Maxterm] kann dieser Term mehrere Einsstellen [Nullstellen] repräsentieren:

$$E = \left\{ \begin{array}{l} (0,0,0,0) \\ (0,0,1,0) \\ (0,1,0,0) \\ (0,1,1,0) \end{array} \right\} \xrightarrow{\text{1. Schritt}} \left\{ \begin{array}{l} (0,0,—,0) \\ (0,1,—,0) \end{array} \right\} \xrightarrow{\text{2. Schritt}} \{(0, \overbrace{—,—}^{\text{ungeb.}}, 0)\} \tag{4.16}$$

- *gesucht: Primterme* (Terme mit minimaler Anzahl von Literalen, die trotzdem nur Einsstellen [Nullstellen] der zu beschreibenden Funktion erfassen $\longrightarrow$ (kosten)günstigste Auswahl von Primtermen gesucht, die *alle* Einsstellen [Nullstellen] überdecken (*Minimierungsproblem*)

  $\longrightarrow$ *gesucht:* Minimierung von $L(y)$= Anzahl der Literale der Primterme zzgl. Zahl der verwendeten Terme

- *Block* B = Menge von Belegungen $X \in \{0,1\}^n$, die in den *gebundenen* Variablen übereinstimmen und gleichen Funktionswert haben (Bei r freien Literalen: 2^r Belegungen zusammengefasst)

- Minimierung am Symmetriediagramm
  1. Finden aller Primblöcke/ Primterme, Spiegelung von kleineren Blöcken an allen möglichen Symmetrielinien $\Longrightarrow$ Bilden von Primeinsblöcken (DMF: Primimplikanten, KMF: Primimplikate)
  2. Auswahl der *Kerne* (Primeinsblöcke, die allein eine Einsstelle [Nullstelle] überdecken
  3. Auswahl weiterer Primeinsblöcke (Strategie und Bewertung notwendig!)

4.4 Algebraische Minimierung

- Grafische Methode wird durch einen Algorithmus ersetzt ($\Longrightarrow$ kann durch einen Computer ausgeführt werden)

- NELSON-Verfahren: Bestimmung der Menge aller Primimplikanten [Primimplikate] zur Bildung einer DMF [KMF]
  1. alle Freistellen werden zu Einsstellen verfügt: Einsstellenergänzung f^E. Anschließend Bildung einer *Nullblocküberdeckung* für f^E der gegebenen (unvollständigen) Schaltfunktion $\tau_0 = \{B_{01}, B_{02}, \ldots, B_{0r}\}$
  2. Aufstellen eines schaltalgebraischen Ausdrucks (KNF bzw. KMF) für f^E: $f^E = W_{01}\&W_{02}\&\ldots\&W_{0r}$
  3. Ausdistribuieren (schrittweise) des Ausdrucks aus *2.* und Umformen und Streichen überflüssiger Termanteile bzw. Terme (Anwendung von Distributiv- und Absorptionsgesetzen $\Longrightarrow$ aus der konjunktiven Form wird die gewünschte disjunktive Form)
  4. Streichen aller in Schritt *3.* gefundenen Terme, die *nur Freistellen* überdecken
- Grafisches Hilfsmittel: *Überdeckungstabelle*. Spalten: Überdeckte Größen (Einsstellen oder Nullstellen), Zeilen: Überdeckende Größen (Primblöcke)
  - zur Bewertung wird eine Spalte mit Kosten c_k angefügt
  - *Kern:* eine Einsstelle wird nur durch einen einzigen Primterm abgedeckt. Kerne müssen auf jeden Fall in die Überdeckungslösung aufgenommen werden
  - Spalten von Einsstellen, die von Kernimplikanten abgedeckt werden, können gestrichen werden
  - *Spaltendominanzregel:* Wenn Vektor i_1 den Vektor i_2 dominiert ($i_1 \geq i_2$) kann i_1 (*dominierende Spalte)* gestrichen werden (denn dominierende Spalten werden sowieso überdeckt)

$$PA = (p_1 \vee p_2 \vee p_3 \vee p_4)\&(p_1 \vee p_3) = p_1 \vee p_3 \tag{4.17}$$

  - *Zeilendominanzregel:* Wenn Primterm p_1 den Primterm p_2 dominiert ($p_2 \leq p_1$) und es weiterhin gilt $c_1 \leq c_2$, kann die *dominierte Zeile* p_2 gestrichen werden
- Allgemeine Vorgehensweise bei Überdeckungstabellen:
  1. Kerne bestimmen und Streichen aller überdeckten Spalten (leergewordene Zeilen können auch gestrichen werden)
  2. Spaltendominanzen finden und dominierende Spalten streichen
  3. Zeilendominanzen finden und dominierte Zeilen streichen (falls das Kostenkriterium c_i erfüllt ist)
  4. Schritte *1.* bis *3.* wiederholen, bis ÜT nicht mehr reduzierbar (keine Kerne und Dominanzen mehr, ggf. noch zyklische Resttabelle auflösen)
- PETRICK-Verfahren: Bestimmung der kostenminimalen Auswahl von Primimplikanten zur Einsstellenüberdeckung [Primimplikate zur Nullstellenüberdeckung]. PETRICK-Ausdruck= algebraische Beschreibung der Überdeckungsbedingungen, disjunktiv verknüpfte Präsenzvariablen

- Präsenzvariable p_k:

$$p_k = \begin{cases} 0 & \text{Primterm } k \text{ ist nicht in Lösung enthalten} \\ 1 & \text{Primterm } k \text{ ist Bestandteil der Lösung} \end{cases} \tag{4.18}$$

- zur Überdeckung jeder Einsstelle muss mindestens ein Primterm zur Lösung gehören, der diese Stelle überdeckt
- für jede Einsstelle der Tabelle enthölt der PA einen Term, die Terme werden *konjunktiv* zum Petrick-Ausdruck verküpft $\Longrightarrow$ der Petrick-Ausdruck muss immer eins liefern.
  Beispiel:

$$\begin{aligned} PA &= (p_5 \vee p_6)\&(p_2 \vee p_6)\&(p_5 \vee p_7)\&p_4\&(p_1 \vee p_2)\&(p_3 \vee p_7)\&(p_1 \vee p_3) \stackrel{!}{=} 1 \\ &= \ldots [\text{Absorptionsregeln, etc.}] \ldots \\ &= p_1p_4p_6p_7 \vee p_2p_3p_4p_5 \vee p_1p_3p_4p_5p_6 \vee p_2p_3p_4p_6p_7 \vee p_1p_2p_4p_5p_7 \stackrel{!}{=} 1 \end{aligned} \tag{4.19}$$

- jeder konjunktive Teilterm repräsentiert genau eine *mögliche Lösung* des Auswahlproblems $\Longrightarrow$ diejenige Überdeckung mit minimaler Kostensumme K kann ermittelt werden

- Allgemeines Vorgehen bei der algebraischen Minimierung
  1. Bestimmung aller Primimplikanten mittels NELSON-Verfahren
  2. Erstellen der Überdeckungstabelle (incl. Kosten der PIs)
  3. Bestimmen der Kerne (Streichen überdeckter Einsen ($\stackrel{\wedge}{=}$Spalten) und Streichen leergewordener Zeilen)
  4. Spaltendominanzregel anwenden, dominierende Spalten streichen
  5. Zeilendominanzregel anwenden, dominierte Zeilen streichen (+Kosten!!), danach ggf. zurück zu Schritt *(3.)*
  6. *Prüfe:* Tabelle vollständig abgearbeitet?
     - ja $\longrightarrow$ Erstelle DMF (=Disjunktion der ermittelten Kerne)
     - nein $\longrightarrow$ Zyklische Resttabelle $\Longrightarrow$ Petrick-Ausdruck, minimieren

- *Ergebnis:* kostenminimaler algebraischer Ausdruck der zu realisierenden digitalen Schaltfunktion

Kapitel 5

5 Bausteine der Digitaltechnik

5.1 Grundlegende Schaltnetze

- aus logischen Ausdrücken kann unmittelbar eine *Konstruktionsvorschrift* (= Strukturausdruck) für Digitalschaltungen erstellt werden
- *Stufenanzahl/ Tiefe der Schaltung* = maximale Anzahl von Schaltgliedern von den Eingängen zum Ausgang (Inverter werden hierbei nicht gezählt)
- DIN IEC 748, Teil 2: *„Ein Schaltnetz ist eine Digitalschaltung, in der es für jede mögliche Kombination von digitalen Signalen eine – und nur eine – Kombination von digitalen Signalen an den Ausgängen gibt“*
- *Halbaddierer*: summieren die beiden Eingangsbits a_i und b_i und legen die Summe auf den Ausgang s_i, zusätzlich wird ein Übertragsbit c_{i+1} erzeugt

$$s_i = a_i \not\equiv b_i \tag{5.1}$$

$$c_{i+1} = a_i \& b_i \tag{5.2}$$

- *Volladdierer*: besitzen zusätzlich einen Übertragseingang (vorherige Stellen werden miteinbezogen)

$$\begin{aligned} s_n &= (\overline{a_n} \& b_n \& \overline{c_{n-1}}) \vee (a_n \& \overline{b_n} \& \overline{c_{n-1}}) \vee (a_n \& b_n \& c_{n-1}) \vee (\overline{a_n} \& \overline{b_n} \& c_{n-1}) \\ &= \big(\overline{c_{n-1}} \& (\overline{a_n} \& b_n \vee a_n \& \overline{b_n})\big) \vee \big(c_{n-1} \& (a_n \& b_n \vee \overline{a_n} \& \overline{b_n})\big) \\ &= (\overline{c_{n-1}} \& (a_n \not\equiv b_n)) \vee \big((c_{n-1} \& (\overline{a_n \not\equiv b_n})\big) = c_{n-1} \not\equiv (a_n \not\equiv b_n) \end{aligned} \tag{5.3}$$

$$\begin{aligned} c_n &= c_{n-1} \& b_n \vee c_{n-1} \& a_n \vee b_n \& a_n = (c_{n-1} \& (a_n \vee a_n)) \vee b_n \& a_n \\ &= (c_{n-1} \& (b_n \not\equiv a_n)) \vee a_n \& a_n \end{aligned} \tag{5.4}$$

a_i	b_i	s_i	c_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabelle 5.1: Charakteristik des Halbaddierers

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabelle 5.2: Charakteristik des Volladdierers

- 4-Bit Ripple-Carry-Addierer
  - erster Baustein ist ein HA, anschließend folgen VAs. Nachteil: sehr langer kritischer Pfad
  - die Laufzeit beträgt $2 \cdot n_{Gatter}$ für die Berechnung von c_n
- Carry-Look-Ahead-Addierer
  - die Berechnung der Überträge geschieht parallel

$$p_i := a_i \not\equiv b_i \text{ (Propagate)} \tag{5.5}$$
$$g_i := a_i \& b_i \text{ (Generate)} \tag{5.6}$$
$$s_i = a_i \not\equiv b_i \not\equiv c_i = p_i \not\equiv c_i \tag{5.7}$$
$$\begin{aligned} c_{i+1} &= a_i \& b_i \vee a_i \& c_i \vee b_i \& c_i = a_i \& b_i \vee c_i \& (a_i \vee b_i) \\ &= a_i \& b_i \vee c_i \& (a_i \not\equiv b_i) = g_i \vee c_i \& p_i \end{aligned} \tag{5.8}$$
$$\Longrightarrow c_1 = g_0 \vee c_0 \& p_0 \tag{5.9}$$
$$c_2 = g_1 \vee c_1 \& p_1 = g_1 \vee g_0 \& p_1 \vee c_0 \& p_0 \& p_1 \tag{5.10}$$
$$c_3 = g_2 \vee c_2 \& p_2 = g_2 \vee g_1 \& p_2 \vee c_0 \& p_1 \& p_2 \vee c_0 \& p_0 \& p_1 \& p_2 \tag{5.11}$$
$$\begin{aligned} c_n &= g_{n-1} \vee c_{n-1} \& p_{n-1} = g_{n-1} \& g_{n-2} \vee g_{n-3} \& p_{n-2} \& p_{n-1} \vee \ldots \\ &\quad \vee g_0 \& p_1 \cdots p_{n-1} \vee c_0 \& p_0 \cdots p_{n-1} \end{aligned} \tag{5.12}$$

- kombinierte Additions- und Subtraktionsbausteine: Negation bitweise mittels XOR

5.2 Hardwaretechnische Realisierung von Schaltnetzen

- universelle Realisierung von Minterm-/Maxterm-orientierten oder von blockorientierten Schaltnetzen durch eine Matrixstruktur
- *Universal Logic Array* (ULA)
  - besteht aus 2^n UND-Schaltgliedern und aus einem oder mehreren ODER-Schaltgliedern $\longrightarrow$ Minterm-orientiert

  - *Personalisierung:* konjunktive Verknüpfung der Minterme mit 0 oder 1

- *Read Only Memory* (ROM)
  - besteht aus 2^n UND-Schaltgliedern und aus einem oder mehreren ODER-Schaltgliedern $\longrightarrow$ Minterm-orientiert
  - *Personalisierung:* Weglassen/ Hinzufügen von Mintermen in der 2. Stufe

- *Programmable Array Logic* (PAL)
  - besteht im Allgemeinen aus weniger als 2^n UND-Schaltgliedern und einem oder mehreren ODER-Schaltgliedern $\longrightarrow$ Primterm-orientiert
  - *Personalisierung:* Festlegung der Primterme in der 1. Stufe

- *Programmable Array Logic* (PLA)
  - Aufbau ähnlich zu PAL, allerding flexibler in der 2. Stufe
  - *Personalisierung:* beide Matrizen sind programmierbar $\longrightarrow$ mehrfache Ausnutzung von Primtermen möglich

5.3 Entwurf von Schaltnetzen

1. Umsetzung der verbalen Aufgabenstellung in eine formale Form (z.B. Funktionstabelle)
2. Bildung einer Normalform, Bildung einer vollständigen Blocküberdeckung
3. Bildung einer Minimalform (z.B. nach S-Diagramm oder Nelson/Petrick)
4. Umformung in das gewählte Basissystem
5. Umformung in den Strukturausdruck
6. Umsetzen in das entsprechende Schaltnetz

5.4 Automaten und Binärspeicher

- *Problem* bei Schaltnetzen: alle Eingabegrößen müssen gleichzeitig eingegeben werden, für hochkomplexe Funktionen technisch nicht realisierbar
- *Abhilfe:* Verlagerung in die Dimension Zeit $\longrightarrow$ *Schaltwerke*
- Eingabealphabet E und Ausgabealphabet A bilden die zeitlich abhängige Einheit AT:

$$AT : A_h^\nu = f_{AT}\left(E_g^\nu, E_g^{\nu-1}, E_g^{\nu-2}, \ldots, E_g^{\nu-\alpha}\right) \tag{5.13}$$

- insgesamt werden $\alpha + 1$ Elemente des Eingabealphabets beobachtet (α =zeitliche Tiefe)
- die Folge $E_g^{\nu-1}, \ldots, E_g^{\nu-\alpha}$ wird auf das *Zustandsalphabet* $S = \{S_1, \ldots, S_k, \ldots, S_w\}$, ($E_g^{\nu-1}, \ldots, E_g^{\nu-\alpha} \longrightarrow S$ mit $|E|^\alpha >> S$) abgebildet

$$\text{Ausgabefunktion:} \quad A_h^\nu = \lambda(E_g^\nu, S_k^\nu) \tag{5.14}$$

$$\text{Überführungsfunktion:} \quad S_k^{\nu+1} = \delta(E_g^\nu, S_k^\nu) \tag{5.15}$$

$$\text{Kennzeichnung von AT als Quintupel:} \quad AT = (E, A, S, \delta, \nu) \tag{5.16}$$

- wichtig: der neue Zustand $S_k^{\nu+1}$ muss zum richtigen Zeitpunkt übernommen werden (Speicherbaustein notwendig)
- Unterscheidung bzgl. $\lambda(E_g^\nu, S_k^\nu)$:

$$\text{Mealy-Automat:} \quad A_h^\nu = \lambda(E_g^\nu, S_k^\nu) \tag{5.17}$$

$$\text{Moore-Automat:} \quad A_h^\nu = \lambda(S_k^\nu) \tag{5.18}$$

$$\text{Medwedew-Automat:} \quad A_h^\nu = S_k^\nu \tag{5.19}$$

- *Automatengraph:* Knoten repräsentieren Zustände, Kanten repräsentieren Zustandsübergänge
- *Automatentafel:* Kartesisches Produkt aus Eingabe- und Zustandsmenge (vgl. Tabelle 5.3)

	$s^{\nu+1}/A^\nu$		
S^ν	E_1^ν	$\ldots$	E_n^ν
$\ldots$		$\ldots$	
S^ν	$\ldots$	$S_k^{\nu+1}/A_h^{\nu+1}$	$\ldots$
$\vdots$		$\vdots$	

	$s^{\nu+1}$			
S^ν	E_1^ν	$\ldots$	E_n^ν	A^ν
$\ldots$		$\ldots$		
S^ν	$\ldots$	$S_k^{\nu+1}$	$\ldots$	A_h^ν
$\vdots$		$\vdots$		

Tabelle 5.3: Automatentafel für Mealy- und Moore-Automaten

- λ und δ werden als Schaltnetze abgebildet ($\Longrightarrow$ Eingaben, Zustände und Ausgaben müssen in binäre Größen kodiert werden)
- Speicherbausteine: *FlipFlops*. Mit r FlipFlops können 2^r unterschiedliche Zustände gespeichert werden
- *Ablaufdiagramm:* platzsparendere und effizientere Beschreibungsform, Zustand= Rechteck mit Doppelstrich, Abfrage= Sechseck, Ausgabe= Rechteck
- *Ablauftabelle:* Teiltabellen für jeden Ablaufschritt
- Entwurf von Schaltnetzen:

Q^ν	$X_R(3)$				$Q^{\nu+1}$	$Y(3)$
	x_6	x_4	x_3	x_1		
3	0	-	0	1	I_1	Y_1
	0	1	0	0	I_2	
	0	1	1	1	I_2	
	1	1	0	1	I_2	
	1	0	-	-	I_3	
	-	0	1	1	I_3	
	1	1	1	0	I_4	

Q^ν :momentaner Zustand
$X_r(k)$:Eingabeblöcke der jeweils *relevanten* Eingabevariablen
$Q^{\nu+1}$:Folgezustand
$Y(k)$:Ausgabeblöcke

Tabelle 5.4: Beispiel einer Ablauftabelle (MOORE)

1. Definition von Ein- und Ausgabevariablen
2. Wahl des Schaltwerktyps und Erstellen des Ablaufdiagramms bzw. der Ablauftabelle gemäß Aufgabenstellung
3. Zustandskodierung
4. Wahl des FlipFlop-Typs und Aufstellen der Ansteuerfunktionen
5. Entwurf des Schaltnetzes für die Überführungsfunktion auf der Basis der Ansteuerfunktionen
6. Entwurf des Schaltnetzes für die Ausgabefunktion
7. Evtl. Umformung der logischen Ausdrücke in geeignete *Strukturausdrücke*
8. Umsetzen in das Schaltbild des Schaltwerks

5.5 Digitale Speicherbausteine (FlipFlop-Schaltungen)

- Unterscheidung zwischen *flankengesteuerten* und *pegelgesteuerten* Speicherelementen (*Latches*)
- RS-FlipFlop (RS-Latch)

$Q(t)$	S	R	$Q(t+1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	undefiniert
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	undefiniert

S	R	$Q(t+1)$	
0	0	$Q(t)$	kein Wechsel
0	1	0	Reset
1	0	1	Set
1	1	undefiniert	

$q(t)$	$q(t+1)$	R	S
0	0	-	0
0	1	0	1
1	0	1	0
1	1	0	-

Tabelle 5.5: Charakteristische Tabelle und Ansteuertabelle des RS-Latch

  - Komplementäre Ausgänge Q und Q'
  - 2 Steuereingänge R („Reset“) und S („Set“)
  - *Nachteil:* es existiert eine ungültige Eingangskombination
  - charakteristische Gleichung: $q_k^{\nu+1} = S^\nu \vee (q_k \& \overline{R^\nu})$
  - getaktetes RS-Latch: Ausgang verändert sich nur, wenn T =HIGH ist

- D-FlipFlop (D-Latch)

W	D	$Q(t+1)$	Zustand
1	0	0	Reset
1	1	1	Set
0	-	$Q(t)$	kein Wechsel

$q(t)$	$q(t+1)$	D	W
0	0	-	0
0	1	1	1
1	0	0	1
1	1	-	0

Tabelle 5.6: Charakteristische Tabelle und Ansteuertabelle des D-Latch

  - S wird invertiert an R zugeführt $\longrightarrow$ *Vorteil:* die ungültige Eingangskombination wird eliminiert
  - charakteristische Funktion: $q_k^{\nu+1} = (D^\nu \& W^\nu) \vee (q_k \& \overline{W^\nu})$

- Flankengesteuerte FlipFlops

RS-FlipFlop

S	R	C	$Q(t+1)$	Zustand
0	0	X	$Q(t)$	kein Wechs.
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	?	ungültig

X :don't care
↑: Takt-übergang low→high

D-FlipFlop

D	C	$Q(t+1)$	Zustand
1	↑	1	Set
0	↑	0	Reset

Tabelle 5.7: Flankengesteuerte FlipFlops (*rising edge*)

  - FlipFlops sind *bistabile* Speicherelemente. Geringere Rückkopplungsgefahr durch Flankensteuerung
  - Anliegender Wert wird in einem kleineren Zeitfenster vom Eingang auf den Ausgang geschaltet

- JK-FlipFlop
  - Q und Q': rückgekoppelt auf impulsgesteuerte NAND-Gatter
  - keine ungültigen Zustände, beinhaltet einen *Wechselzustand*
  - charakteristische Funktion: $q^{\nu+1} = (\overline{K} \& q^\nu) \vee (J \& \overline{q^\nu})$

- Toggle-FlipFlop (T-Flipflop)

J	K	Takt	$Q(t+1)$	Zustand
0	0	$\uparrow$	$Q(t)$	halten
0	1	$\uparrow$	0	Reset
1	0	$\uparrow$	1	Set
1	1	$\uparrow$	$Q(t)'$	wechseln

Tabelle 5.8: JK-FlipFlop

T	CLK	$Q(t+1)$	Zustand
0	$\uparrow$	$Q(t+1)$	halten
1	$\uparrow$	$Q(t+1)'$	wechseln

Q	T	$Q(t+1)$
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 5.9: Toggle-FlipFlop

  - gebildet durch Zusammenschalten der J- und K-Eingänge
  - Applikation: *Frequenzteiler*
  - charakteristische Funktion: $q^{\nu+1} = (T \& \overline{q^{\nu}}) \vee (\overline{T} \& q^{\nu})$

- Master-Slave-FlipFlop

J	K	CLK	$Q(t+1)$	Zustand
0	0	1	$Q(t)$	halten
0	1	$\uparrow\downarrow$	0	Reset
1	0	$\uparrow\downarrow$	1	Set
1	1	$\uparrow\downarrow$	$Q(t)'$	invertieren

Tabelle 5.10: Master-Slave-JK-FlipFlop

  - *Master* ist aktiviert bei steigender Flanke, *Slave* ist aktiviert bei fallender Flanke $\longrightarrow$ vollständige Entkopplung des Ausgangs vom Eingang

5.6 Binäre Schalter und Gatter

- Übergang von Relais zu *Transistoren* (elektronische Schalter): Über eine Steuerleitung wird die Leitfähigkeit zwischen zwei Elektroden eingestellt. Logische Operationen lassen sich so über Parallel- und Reihenschaltungen realisieren.

- Inverter (Einschalterprinzip): realisiert mit einem NMOS-Transistor am GND. Nachteil: Im Schaltungszustand $U_x = H$ fließt konstant ein Strom

- Inverter (Zweischalterprinzip): NMOS am GND, PMOS an $U_B \Longrightarrow$ in keinem Schaltungszustand fließt konstant ein Strom
- NOR [NAND] (Einschalterprinzip): zwei NMOS in Parallelschaltung [Serienschaltung]
- NOR (Zweischalterprinzip): zwei PMOS in Reihenschaltung an U_B, zwei NMOS in Parallelschaltung an GND
- NAND (Zweischalterprinzip): zwei PMOS in Parallelschaltung an U_B, zwei NMOS in Reihenschaltung an GND

5.7 CMOS-Technologie

- *CMOS (Complementary-MOS)*: NMOS- und PMOS-Transistoren werden in jeweils komplementären Netzen eingesetzt
  - Ausgang der realisierten Funktion gleich 1 $\longrightarrow$ Pull-Up-Netz schaltet den Ausgang F auf V_{DD}
  - Ausgang der realisierten Funktion gleich 0 $\longrightarrow$ Pull-Down-Netz schaltet den Ausgang F auf V_{SS}
  - PUN wird mit PMOS realisiert, PDN wird mit NMOS realisiert
- NMOS-Schalter schließt, wenn das Gate auf „high“ („1“) ist
- PMOS-Schalter schließt, wenn das Gate auf „low“ („0“) ist
- PUN muss zu PDN dual sein (kann mittels De Morgans Theorem gezeigt werden)
- *Vorgehensweise*: PDN bestimmen, indem die Schaltfunktion der NMOS-Schaltungen abgelesen und anschließend der komplette Ausdruck invertiert wird
- es muss gelten:

$$\text{Fehlerfreiheit:} \quad F \& G = 0 \qquad \text{(kein Kurzschluss)} \tag{5.20}$$

$$\text{Wohldefiniertheit:} \quad F \vee G = 1 \qquad \text{(niemals hochohmig auf dem Ausgang)} \tag{5.21}$$

5.8 Funktionseinheiten der Digitaltechnik

- Multiplizierer
  - die Multiplikation einer Dualzahl mit einer Zweierpotenz 2^n bedeutet eine *Linksverschiebung* um n Stellen und Nachziehen von n Nullen
  - der Multiplikand muss mit jeder Stelle des Multiplikators multipliziert werden (bitweise konjunktiv verknüpft werden)

a_{x2}	a_{x1}	y_a		a_{y2}	a_{y1}	y_{e1}	y_{e2}	y_{e3}	y_{e4}
0	0	x_{a1}		0	0	x_e	0	0	0
0	1	x_{a2}	$\Longleftrightarrow$	0	1	0	x_e	0	0
1	0	x_{a3}		1	0	0	0	x_e	0
1	1	x_{a4}		1	1	0	0	0	x_e

Tabelle 5.11: 4:1-Multiplexer und 1:4-Demultiplexer

  - das Ergebnis muss also um die entsprechende Stellenzahl nach links verschoben und zu den übrigen Ergebnissen aufaddiert werden
  - *Problem:* sehr lange Laufzeiten

- Multiplexer
  - bestehen aus Eingängen X_{an}, Ausgang Y_a und Steuergrößen A_n und erlauben *serielle Übertragung*
  - *Funktionsweise*: die Minterme der Steuervektoren A_n geben immer genau ein UND-Gatter frei
  - Strukturausdrücke nach dem Entwicklungssatz sind direkt umsetzbar. Bei der Realisierung einer Funktion mit n Literalen über Multiplexer muss nach $n-1$ Variablen entwickelt werden

- Demultiplexer
  - können zur Ansteuerung von Steuerleitungen von Speicherzellen verwendet werden
  - Liegt der Wert „1“ am Eingang des Demultiplexers, repräsentieren die Ausgänge sämtliche Minterme des Adressvektors

- Zähler
  - dienen dazu, auf einer speziellen Signalleitung beobachtete Ereignisse zu zählen
  - weitere Eingriffsmöglichkeiten: Überlaufmeldung, Nullstellungsmeldung, Rückstellmöglichkeit auf 0
  - zu Tabelle 23: $K_1 = 1$, $K_2 = N \vee q_1^\nu$, $K_3 = N \vee q_2^\nu q_3^\nu$, $J_1 = \overline{N}$ $J_2 = \overline{N} q_1^\nu$, $J_3 = \overline{N} q_2^\nu q_1^\nu$
  - um bei Zählern von Realisierungsdetails zu abstrahieren, führt man *Blocksymbole* ein. Es besteht aus einem Daten- und einem Steuerteil

- Schieberegister
  - Verbund aus mehreren FlipFlops zur Darstellung von Informationen
  - Alle Speicherinhalte können um eine Zelle weitergeschoben werden (Rechtsschieben oder Linksschieben)

Q^ν			X^ν	$Q^{\nu+1}$			Y^ν						
q_3	q_2	q_1	N	q_3	q_2	q_1	$\ddot{U}$	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	0	1	0	0	-	0	-	1	-
0	0	0	1	0	0	0	0	0	-	0	-	0	-
0	0	1	0	0	1	0	0	0	-	1	-	-	1
0	0	1	1	0	0	0	0	0	-	0	-	-	1
0	1	0	0	0	1	1	0	0	-	-	0	1	-
0	1	0	1	0	0	0	0	0	-	-	1	0	-
0	1	1	0	1	0	0	0	1	-	-	1	-	1
0	1	1	1	0	0	0	0	0	-	-	1	-	1
1	0	0	0	1	0	1	0	-	0	0	-	1	-
1	0	0	1	0	0	0	0	-	1	0	-	0	-
1	0	1	0	1	1	0	0	-	0	1	-	-	1
1	0	1	1	0	0	0	0	-	1	0	-	-	1
1	1	0	0	1	1	1	0	-	0	-	0	1	-
1	1	0	1	0	0	0	0	-	1	-	1	0	-
1	1	1	0	0	0	0	1	-	1	-	1	-	1
1	1	1	1	0	0	0	1	-	1	-	1	-	1

Tabelle 5.12: Ansteuerfunktion eines 3-bit-Dualzählers

  - beim Schieben geht jeweils das in Schieberichtung voranstehende Datenbit verloren
  - das letzte Datenbit wird mit einem Inhalt gefüllt, der von den Randbedingungen dieser Zelle abhängt
  - *zirkuläre Rotation* möglich
  - Schieberegister werden auch verwendet, um Datenbitgruppen von räumlichen Folgen in eine zeitliche Folge umzuwandeln:
    * *Parallel-Serienwandlung*: die Daten $(D_n, \ldots, D_1, D_0)$ werden parallel in das Register geladen. Danach können sie seriell in der zeitlichen Reihenfolge ausgegeben werden
    * *Serien-Parallelwandlung*: die Daten werden in zeitlicher Folge in das Register geschoben. Danach können sie gebündelt ausgelesen werden

- Pufferspeicher (First-In-First-Out, FIFO)
  - meist *wortorganisierte* Speicher mit konstanter Wortbreite
  - dienen häufig zur Kopplung unterschiedlich schneller Baugruppen
  - auf Eingangsseite eingeschriebenes Wort wird solange zum Ausgang vorgeschoen, bis dieser erreicht wird oder ein belegter Platz erreicht wird
  - wird ein Wort ausgelesen, rücken alle dahinter gelegenen Wörter um eine Position vor

- Stapelspeicher (Last-In-First-Out, LIFO, Kellerspeicher, Stack)
  - eignet sich zu Berechnungen mit häufigem Zwischenspeichern von Ergebnissen und gleichzeitiger Einhaltung einer Ordnung der Ergebnisse
  - das zuletzt eingespeicherte Datenwort (PUSH) wird zuerst auf dem LIFO ausgelesen (POP)
  - *anschaulicher Vergleich*: ein Tellerstapel
- Allgemeine Speicher, Arbeitsspeicher
  - quadratische Form der Speichermatrix erstrebenswert $\Longrightarrow$ es bedarf eines zweiten Decoders (*Spalten-Decoder*), um das entsprechende Datenwort im Speicherwort zu identifizieren
  - sei N die Länge des Speicherworts und n die Länge des Datenworts. Dann erhält man die Anzahl r der Adressbits des Spalten-Decoders:

$$r = \left\lceil \mathrm{ld}\left(\frac{N}{n}\right) \right\rceil \tag{5.22}$$

  - die Adressbits des Zeilendekoders können somit um r Bits gekürzt werden
- Sonderfunktionen (Bustechnologien)
  - *Open-Collector-Schaltung*: mehrere schreibende Ausgänge sind zusammengeschaltet (*wired*-Verbindungen). *Problem*: unsymmetrische Lastverhältnisse, dadurch sehr langsam
  - *Tri-State-Schaltung*: zwei Schalter pro Variable sowie jeweils ein weiteres Signal OE (*Output enable*), durch Decoder realisiert, das bei Nichtauswahl der Einheit die Ausgangsstufe vom Bus trennt. Dadurch Vermeidung von Kurzschlüssen. *Problem*: Evtl. Zugriffskonflikte, falls in Anordnungen die Benutzung des Busses durch mehrere Komponenten möglich ist. *Abhilfe*: Busverwaltung (*Arbiter*)

Tabellenverzeichnis