Inhaltsverzeichnis

1 Schaltungstechnik I **3**

1.1 Boolesche Algebra / Schaltalgebra . . . 3

1.1.1 Axiomensystem . . . 3

1.1.2 Boolesche Funktionen . . . 4

1.2 Gatterlogik . . . 5

1.2.1 Implementierung . . . 5

1.2.2 Multiplexer (Auswahlschaltung) . . . 5

1.2.3 2-Bit-Dekodierer . . . 6

1.3 Implementierung von Schaltnetzen . . . 6

1.3.1 Elektronische Grundlagen . . . 6

2 Schaltungstechnik II **8**

2.1 Minimierung Boolescher Formeln . . . 8

2.1.1 Äquivalenzumformungen . . . 8

2.1.2 Systematische Vereinfachungsverfahren . . . 8

2.1.3 Petricks Algorithmus . . . 12

2.2 Programmierbare Logikarrays . . . 13

2.2.1 Allgemeine Struktur . . . 14

2.2.2 Hardware-Implementation . . . 14

2.3 Hardware-Beschreibungssprache . . . 15

2.3.1 Aufbau . . . 15

3 Binärarithmetik & Ihre Implementierung **16**

3.1 Erinnerung: Arithmetik . . . 16

3.1.1 Zahlensysteme . . . 16

3.1.2 Addition und Subtraktion . . . 16

3.1.3 Multiplikation und Division . . . 17

3.2 Binäre Arithmetik und Implementierung durch Schaltkreise . . . 17

3.3 Addition . . . 17

3.3.1 Halbaddierer . . . 17

3.3.2 Volladdierer . . . 18

3.3.3 n-Bit-Addierer . . . 19

3.3.4 Darstellung negativer Zahlen . . . 20

3.4 Multiplikation . . . 20

3.4.1 Multiplikation mit Potenzen der Basis . . . 21

3.4.2 Multiplikation mit Zweierpotenzen: Bit-Schieben . . . 21

3.4.3 Allgemeine Binäre Multiplikation . . . 21

3.4.4 Negative Zahlen . . . 21

3.4.5 Standardalgorithmusa 21
3.4.6 Booths Algorithmus 21
3.5 Die Arithmetisch-Logische-Einheit (Arithmetic Logical Unit. ALU) . 21
3.5.1 Die ALU in der Hack-Architektur 21
3.5.2 Einbindung der ALU in den Prozessor der Hack-Architektur . 21

4 Sequentielle Logik 22
4.1 Sequentielle Logik 22
4.1.1 Logikschaltungen: Kombinatorische & Sequentielle Logik . . . 22
4.1.2 Prinzip der Rückkopplung 23
4.1.3 Asynchrone und synchrone Schaltwerke 23
4.1.4 Taktsignal (Clock Signal) 24
4.2 Bistabile Kippstufen 24
4.3 Register, Zähler und Speicher 24
4.4 Hardware-Simulation der sequentiellen Logik 24

Kapitel 1

Schaltungstechnik I

Boolesche Algebra / Schaltalgebra

DEFINITION: Boolesche Algebra

Eine *Boolesche Algebra* ist eine Menge B mit dem Nullelement 0 und dem Einselement 1 (d.h., $0, 1 \in B$), auf der die Operationen **Konjunktion** $\wedge$ und **Disjunktion** $\vee$ sowie **Negation** $\neg$ definiert sind.

DEFINITION: Schaltalgebra

Eine *Schaltalgebra* ist eine Boolesche Algebra mit der Trägermenge $B = \{0, 1\}$.

1.1.1 Axiomensystem

	George Boole (1847)	
Kommutativität	$a \wedge b = b \wedge a$	$a \vee b = b \vee a$
Assoziativität	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$	$(a \vee b) \vee c = a \vee (b \vee c)$
Idempotenz	$a \wedge a = a$	$a \vee a = a$
Distributivität	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
Neutralität	$a \wedge 1 = a$	$a \vee 0 = a$
Extremalität	$a \wedge 0 = 0$	$a \vee 1 = 1$
Involution	$\neg\neg a = a$	
	De Morgan (1860)	
De Morgan	$\neg(a \wedge b) = \neg a \vee \neg b$	$\neg(a \vee b) = \neg a \wedge \neg b$
Komplementarität	$a \wedge \neg a = 0$	$a \vee \neg a = 1$
Dualität	$\neg 0 = 1$	$\neg 1 = 0$
Absorption	$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$

1.1.2 Boolesche Funktionen

DEFINITION: Boolesche Funktionen

Eine Boolesche Funktion ist eine Funktion $f : \{0,1\}n \to \{0,1\}, n \geq 0$. Die Anzahl n der Argumente der Funktion f heißt ihre Stelligkeit (arity).
Für kleine Stelligkeiten gibt es Spezielle Ausdrücke:

- $n = 1$: **unär** (unary)
- $n = 2$: **binär** (binary)
- $n = 3$: **ternär** (ternary)

Boolesche Funktionen werden durch **Schaltnetze** implementiert.

Jede Boolesche Funktion kann durch **Wahrheitstafeln** und **Formeln der Schaltalgebra** dargestellt werden.

Darstellung durch Wahrheitstafeln

- eine Spalte pro Funktionsargument,
- eine Zeile pro mögliche Wertekombination der Funktionsargumente,
- zusätzliche Spalte für den Funktionswert.

Abbildung 1.1: Wahrheitstafel einer ternären Booleschen Funktion

x_1	x_2	x_3	y
0	0	0	0
1	0	0	1
0	1	0	1
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

Darstellung durch Formeln der Schaltalgebra

Disjunktive Normalform
→ Darstellung der Minterme
→ Sum of Products (SOP)
Bilde *Disjunktion der Konjunktionen* aus Literalen aus jeder Zeile in der der Funktionswert 0 ist

Konjunktive Normalform
→ Darstellung der Maxterme
→ Product of Sums (POS)
Bilde *Konjunktion der Disjunktionen* aus Literalen aus jeder Zeile in der der Funktionswert 1 ist

Konjunktive und Disjunktive Normalform im Vergleich: Bevorzuge die disjunktive bei weniger Einsen, die konjunktive bei weniger Nullen in den Funktionswerten.

Vollständige Operationenmengen

DEFINITION: Vollständige Operationenmengen

Eine vollständige Operationenmenge ist eine Menge von Booleschen Operationen, die ausreicht, um alle Booleschen Funktionen darzustellen

Da sowohl die disjunktive als auch die konjunktive Normalform nur die Operationen Konjunktion, Disjunktion und Negation benutzt, ist die Menge $O = \{\vee, \wedge, \neg\}$ eine vollständige Operationenmenge.

Von einer anderen Operation O' kann man zeigen, dass sie vollständig ist, indem man die drei Operationen von O nur mit Hilfe der Operationen aus dieser Menge O' darstellt.

Gatterlogik

DEFINITION: (Logik-)Gatter

Ein Logikgatter ist eine Anordnung zur Realisierung einer booleschen Funktion, die binäre Eingangssignale zu einem binären Ausgangssignal verarbeitet. Die Eingangssignale weerden durch Implementierung logischer Operatoren zu einem einzigen logischen Ergebnis umgewandelt.

1.2.1 Implementierung

Boolesche Funktionen werden mit Hilfe der Gatterlogik implementiert. Das heißt, mehrere elementare Gatter werden zusammengeschaltet, um komplexe Boolesche Funktionen zu berechnen:

Vollständige Operatorenmengen

Datenflußsteuerung

Wir können den Fluss von Daten mit sogenannten Steuerleitungen kontrollieren: Wir nehmen an:

- Datensignal d läuft auf einer *Datenleitung* D
- Steuersignal c läuft auf einer *Steuerleitung* C

Nur wenn auf Steuerleitung C eine logische 1 anliegt, soll das Datensignal weitergeleitet werden. Das ist mit einem AND-Gatter einfach zu implementieren: Wir benutzen ein AND-Gatter und verbinden D und C zu einem gemeinsamen Ausgangssignal, an dem dann eine logische 1 anliegt, wenn an Datenleitung D eine 1 anliegt und Steuerleitung C ebenfalls auf 1 steht

1.2.2 Multiplexer (Auswahlschaltung)

01 - 31ff

1.2.3 2-Bit-Dekodierer

Implementierung von Schaltnetzen

DEFINITION:

Durch elektronische Bauteile, speziell (Feldeffekt-)Transistoren, werden spannungsgesteuerte Schalter dargestellt. aus mehreren Schaltern werden dann Gatter zusammengesetzt, die einfache Boolesche Funktionen darstellen. Die Gatter bilden jeweils eine vollständige Operationenmenge.

1.3.1 Elektronische Grundlagen

Schalter

Um Gatter zu implementieren werden Schalter benötigt, die automatisch betätigt werden können. Hierzu gibt es nun mehrere Ansätze:

- Erste Lösung: *Relais*: (elektromechanische/magnetische Schalter)
  Relativ groß, hoher Stromverbrauch, Mechanisch und deshalb Störungsanfällig
- Bessere Lösung: *Vakuumröhren*: (Verstärkerröhren)
  Immernoch groß aber geringerer Stromverbrauch
- Entscheidente Verbesserung: *Transistoren*: (Halbleiterschalter/verstärker)
  Sehr klein & kleiner Stromverbrauch

Transistoren

Bipolartransistor
(bipolar junction transistor)
→ 3 Anschlüsse:
  - Basis (base)
  - Emitter (emitter)
  - lektor (collector)

→ stromgesteuert
Kleiner Steuerstrom auf der Basis-Emitter-Strecke steuert großen Strom auf Kollektor-Emitter-Strecke.

Feldeffekttransistor
(field effect transistor)
→ 3 Anschlüsse:
  - Quelle (source)
  - Senke (drain)
  - Steuerelektrode (gate)

→ spannungsgesteuert
Der Widerstand und somit der Strom der Drain-Source-Strecke wird durch die Gate-Source-Spannung gesteuert. Im statischen Fall fasst Stromlos

→ Wir beschränken uns im Folgenden auf Feldeffekttransistoren, da diese für Rechnertechnik (und speziell für integrierte Schaltunngen) wesentlich wichtiger sind

Feldeffekttransistoren

Transistorschalter

Kapitel 2

Schaltungstechnik II

Minimierung Boolescher Formeln

DEFINITION: Semantische & Syntaktische Äquivalenz

Seien φ und psi Boolesche Formeln, dann gilt:

- Wenn beide Formeln für alle Belegungen den gleichen Wahrheitswert haben, dann sind die Formeln

  Semantische äquivalent: $\varphi \equiv \psi$

  Wenn φ durch Äquivalenzumformungen in ψ umgeformt werden kann, dann sind die Formeln

  Syntaktische äquivalent: $\varphi = \psi$

Für die Syntaktische Äquivalenz ist der Nachweis oft wesentlich kürzer. Ein Abschluss der Äquivalenzprüfung ist allerdings nicht garantiert (kein Weg gefunden $\not\rightarrow$ $\varphi \not\equiv \psi$). Für die Semantische Äquivalenz kann der Nachweis sehr aufwendig sein. Zum Falsifizieren wird jedoch lediglich eine Wertkombination benötigt, die nicht equivalent ist.

2.1.1 Äquivalenzumformungen

Die Axiome der Booleschen Algebra erlauben es, alle semantische geltenden Äquivalenten auf syntaktischem Wege abzuleiten, denn es gilt:

Zwei Boolesche Formeln sind **genau dann** semantisch äquivalent, wenn sie syntaktisch äquivalent sind.

2.1.2 Systematische Vereinfachungsverfahren

Das Problem Äquivalenzumformungen ist, dass es keine klare Strategie zur Vereinfachung gibt. Besser wäre ein systematisches Vereinfachungsverfahren.

Karnaugh-Veitch-Diagramme

DEFINITION: Karnaugh-Veitch-Diagramme

Tabellarische Darstellungen Boolescher Funktion (wie Wahrheitstafeln, nur andere Auflistung der Funktionswerte).

- 2^n Felder für n Argumente.
- Anordung, dass ein Übergang zu einem Nachbarfeld den Wert nur genau einer der Variablen ändert (ein sog. Gray-Code)

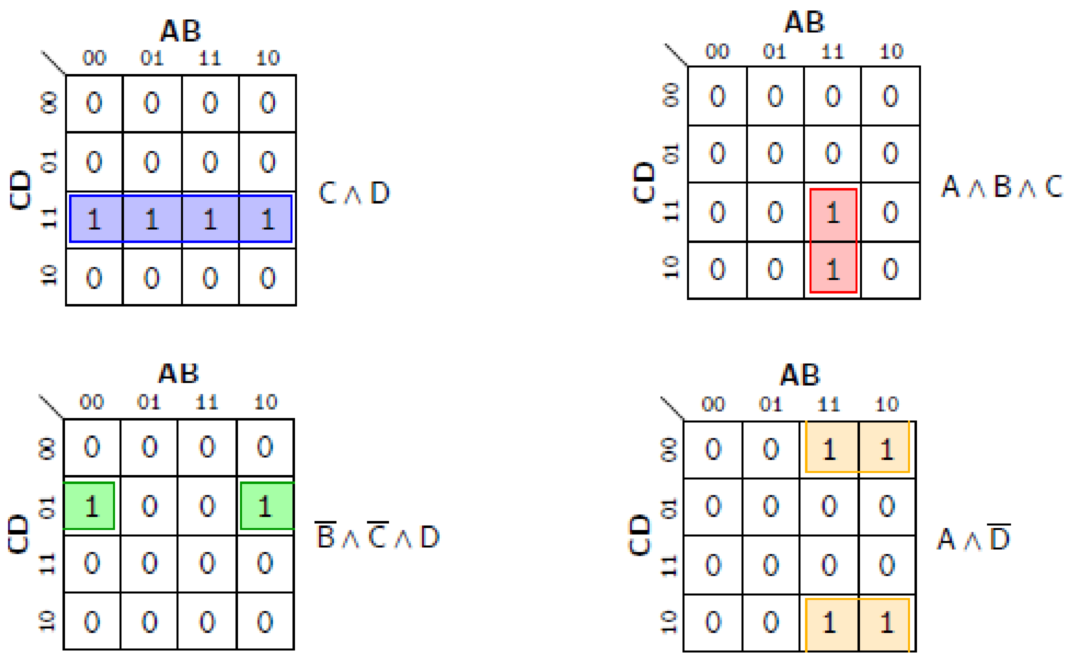
BEISPIEL: Zwei Variablen

Ein Gray-Code für zwei Variablen:
Anordnung ist bis auf zyklische Vertauschungen und Spiegelungen eindeutig.

00	01	11	10
01	11	10	00
11	10	00	01
10	00	11	10

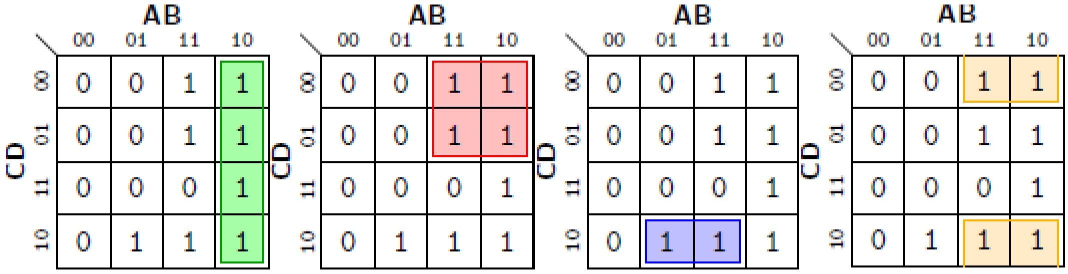
Sind zwei benachbarte Felder eines Karnaugh-Veitch-Diagramms beide 1, so zeigt dies an, dass eine bestimmte Variable in der ursprünglichen Funktion irrelevant ist. Beim Zusammenfassen von Feldern darf auch der Rand überschritten werden, da sich auch in diesem Fall der Wert nur einer Variable ändert: Wie oben gezeigt können auch mehr als 2 Felder Zusammengefasst werden, jedoch nur, wenn die Felderzahlen Zweierpotenzen sind.

Abbildung 2.1: Beispiel für Karnaugh-Veitch-Diagramms

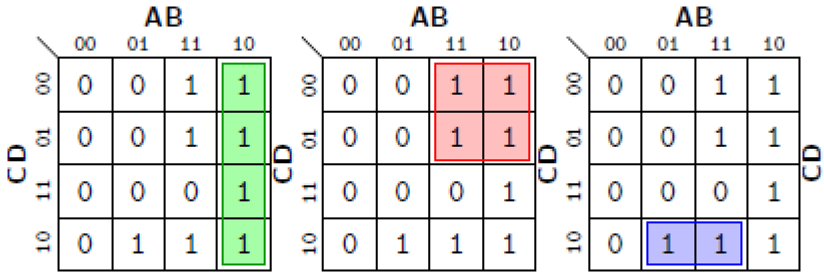


1. **Schritt:** Finde alle maximalen Zusammenfassungen von Feldern

   (dürfen überlappen, aber nicht in größeren Zusammenfassungen vorkommen)

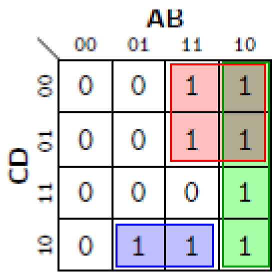


2. **Schritt:** Wähle möglichst wenige Zusammenfassungen, die alle Einsen abdecken



3. **Schritt:** Sammle die benötigten Ausdrücke:

$$
\begin{aligned}
0 = {} & (A \land \overline{C}) \\
& \lor (A \land B) \\
& \lor (B \land C \land \overline{D})
\end{aligned}
$$



INFO:

Natürlich kann die Minimierung nicht nur durch Zusammenfassen von Einsen, sondern, unter Rückgriff auf die Dualität der Booleschen Gesetze bzw. die Dualität der disjunktiven und der konjunktiven Normalform, auch durch Zusammenfassen von Nullen durchgeführt werden.
Da das Ergebnis eine Konjunktion von Disjunktionen ist (analog zur konj. Normalform) müssen die Variablenwerte, die die Zusammenfassungen beschreiben negiert werden

Quine-McCuskey-Algorithmus

Eine Minimierung logischer Funktionen mit bis zu sechs Argumenten ist zwar mit erweiterten Karnaugh-Veitch-Diagrammen im Prinzip möglich, aber unhandlich. Ein besserer Weg besteht in der Verwendung eines anderen Verfahrens des **Quine-McClunskey-Algorithmus**.

1. **Schritt:** Bilde die disjunktive (analog auch konmjunktive) Normalform der zu minimierenden Funktion.

2. **Schritt:** Finden der **Primimplikanten**.

   - Vereinige Terme, in denen eine einzelne Variable in dem einen Term als positives, im anderen als negatives Literal auftritt. (Der gleiche Term darf für mehrere Vereinigungen verwendet werden.)

$$
\begin{aligned}
(A_1 ... A_i ... A_n) \vee (A_1 ... \overline{A_i} ... A_n) &= A_1 ... A_{i-1} A_{i+1} ... A_n \wedge (A_i \vee \overline{A_i}) \\
&= A_1 ... A_{i-1} A_{i+1} ... A_n \wedge 1 \\
&= A_1 ... A_{i-1} A_{i+1} ... A_n
\end{aligned}
$$

   - Wiederhole dies Rekursiv mit den Vereinigungsergebnissen, bis keine weiteren Vereinigungen mehr möglich sind.
   - Vernachlässige alle Terme, die mit anderen Vereinigt wurden.
   - Übrig bleiben die sogenannten **Primimplikanten**

1er Implikanten			
1	$\overline{A}B\overline{C}\overline{D}$	0100	×
2	$A\overline{B}\overline{C}\overline{D}$	1000	×
3	$A\overline{B}\overline{C}D$	1001	×
4	$A\overline{B}C\overline{D}$	1010	×
5	$A\overline{B}CD$	1011	×
6	$AB\overline{C}\overline{D}$	1100	×
7	$ABC\overline{D}$	1110	×
8	$ABCD$	1111	×

Die 1er Implikanten sind die Minterme.

2er Implikanten			
1+6 → 9	$B\overline{C}\overline{D}$	-100	P_1
2+3 → 10	$A\overline{B}\overline{C}$	100-	×
2+4 → 11	$A\overline{B}\overline{D}$	10-0	×
2+6 → 12	$A\overline{C}\overline{D}$	1-00	×
3+5 → 13	$A\overline{B}D$	10-1	×
4+5 → 14	$A\overline{B}C$	101-	×
4+7 → 15	$AC\overline{D}$	1-10	×
5+8 → 16	ACD	1-11	×
6+7 → 17	$AB\overline{D}$	11-0	×
7+8 → 18	ABC	111-	×

4er Implikanten			
10+14, 11+13 → 19	$A\overline{B}$	10--	P_2
11+17, 12+15 → 20	$A\overline{D}$	1--0	P_3
14+18, 15+16 → 21	AC	1-1-	P_4

Primimplikanten (unvereinigte Implikanten) sind mit P_i bezeichnet.

3. **Schritt:** Aufstellen und Auswerten der Primimplikantentabelle

   - Spalten: Minterme, Zeilen: Primimplikanten
   - Finde wesentliche Primimplikanten (aufsuchen der Spalten mit nur einer Markierung)

   Eine systematische Methode für diese Auswahl von Primimplikanten ist **Petricks Algorithmus**

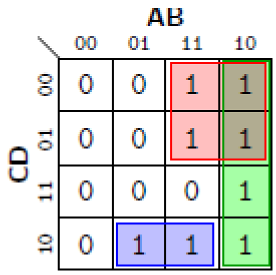
Primimplikanten			Minterme (Konjunktionen der disjunktiven NF)							
			0100	1000	1001	1010	1011	1100	1110	1111
P_1	-100	*	$\color{red}{\bullet}$					$\color{blue}{\bullet}$		
P_2	10--	*		$\color{blue}{\bullet}$	$\color{red}{\bullet}$	$\color{blue}{\bullet}$	$\color{blue}{\bullet}$			
P_3	1--0			$\color{gray}{\bullet}$		$\color{gray}{\bullet}$		$\color{gray}{\bullet}$	$\color{gray}{\bullet}$	
P_4	1-1-	*				$\color{blue}{\bullet}$	$\color{blue}{\bullet}$		$\color{blue}{\bullet}$	$\color{red}{\bullet}$

- $\color{red}{\bullet}$ Abdeckung nur durch einen Primimplikanten → wesentlich
- $\color{blue}{\bullet}$ Abdeckung auch durch wesentliche Primimplikanten.
- $\color{gray}{\bullet}$ Nicht benötigte Abdeckungen

INFO:

Primimplikanten des Quine-McCluskey-Algorithmus entsprechen Feld-Zusammenfassungen in Karnaugh-Veitch-Diagrammen.
Dementsprechend gibt es auch wesentliche Zusammenfassungen in Karnaugh-Veitch-Diagrammen. (Feldergruppen, die von keiner der anderen Gruppen abgedeckt werden)

Grün entspricht dem 4er Primimplikanten 10–
Rot entspricht dem 4er Primimplikanten -110
Blau entspricht dem 2er Primimplikanten -110
Gelb entspricht dem 4er Primimplikanten 1–0



2.1.3 Petricks Algorithmus

Nicht immer decken die wesentlichen Primimplikanten alle Minterme ab. In diesem Fall müssen aus den restlichen Primimplikanten geeignete ausgewählt werden, um die verbleibenden Minterme abzudecken.

1. **Schritt:** Bilde eine reduzierte Primimplikantentabelle

   Diese enthält nur noch die noch nicht abgedeckten Minterme und die nicht wesentlichen Primimplikanten.

2. **Schritt:** Ordne jedem Primimplikanten eine Auswahlvariable P_i zu.

3. **Schritt:** Bilde für jeden Minterm (Spalte) die Disjunktion der Auswahlvariablen.

   Alle Primimplikanten, die diesen Minterm abdecken werden mit einer Disjunktion verknüpft.

4. **Schritt:** Verknüpfe alle Disjunktionen zu einer Konjunktion C.

5. **Schritt:** Wandle Konjunktion C durch die Distributivgesetze in eine Disjunktion D' um

   Nun ergibt sich eine Disjunktion aus Konjunktionen, die jeweils alle Minterme abdecken.
   $$(P_1 \wedge P_2 \wedge P_3) \vee (P_2 \wedge P_3 \wedge P_4) \vee (P_3 \wedge P_5)$$

6. **Schritt:** Wähle die Konjunktionen aus D', mit den wenigsten Primimplikanten

   $P_3 \wedge P_5$

	Minterme (Konjunktionen)						
Primimplikanten		000	001	010	101	110	111
P_1	00-	•	•				
P_2	0-0	•		•			
P_3	-01		•		•		
P_4	-10			•		•	
P_5	1-1				•		•
P_6	11-					•	•

$(P_1 \vee P_2)$ (001)
$\wedge(P_1 \vee P_3)$ (001)
$\wedge(P_2 \vee P_4)$ (010)
$\wedge(P_3 \vee P_5)$ (101)
$\wedge(P_4 \vee P_6)$ (110)
$\wedge(P_5 \vee P_6)$ (111)

Programmierbare Logikarrays

Alle betrachteten Minimierungsergebnisse haben die folgende allgemeine Form:

$$o = (i_1 \wedge \overline{i_2} \wedge \overline{i_3} \wedge ...) \vee (\overline{i_1} \wedge i_2 \wedge \overline{i_3} \wedge ...) \vee (\overline{i_1} \wedge \overline{i_2} \wedge i_3 \wedge ...) \vee ...$$

Alle Boolesche Formeln können in diese Form gebracht werden, denn es handelt sich um eine Disjunktion von Konjunktionen von Literalen (sum of products, SOP)

DEFINITION: Programmierbare Logikarrays

Programmierbare Logikarrays (programmable logic array, PLAs) sind solche Formeln realisiert in Hardware".

- Alle Eingaben i_k und ihre Negation $\overline{i_k}$ sind verfügbar
- Die Eingaben werden über AND-Gatter verknüpft
- Die Ausgaben der AND-Gatter werden durch OR-Gatter verknüpft
- Ein PLA wird durch Entfernen von Verbindungen "konfiguriert"(pprogrammiert")

Gatterimplementierung von Originalfunktion und disjunkt. Normalform

Abbildung 2.2: Mehr Gatter, aber standardisierte Struktur

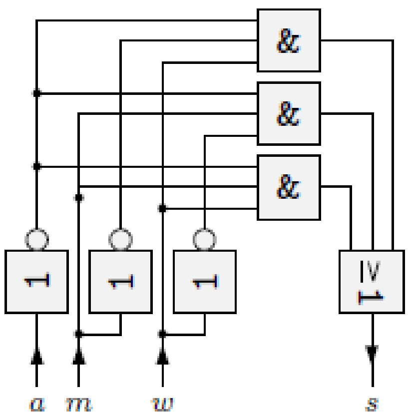
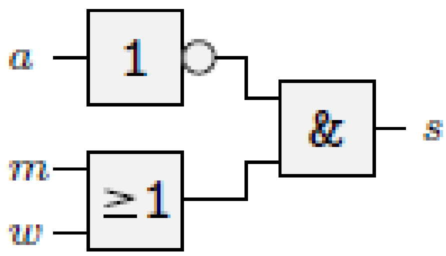


Abbildung 2.3: Weniger Gatter, aber Struktur abhängig von jeweiliger Funktion



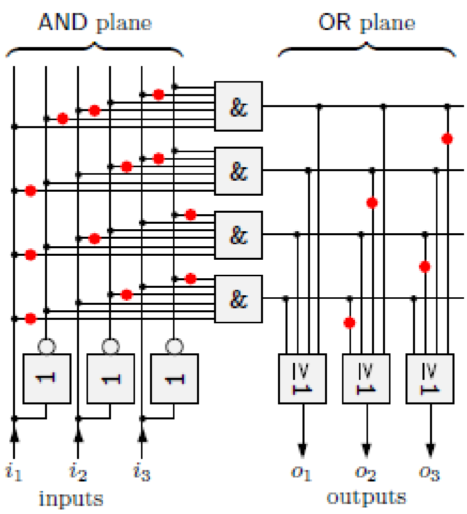
2.2.1 Allgemeine Struktur

Jede Funktion in disjunktiver Normalform kann durch eine Standard-Gatterstruktur dargestellt werden:

- Einem NOT-Gatter (Inverter), sodass für jede Eingabe negiert und unnegiert zur verfügung steht.
- Einem AND-Array, zur Berechnung der Konjunktionen
- Einem OR-Array, zur disjunktiven Verknüpfung der Konjunktionen

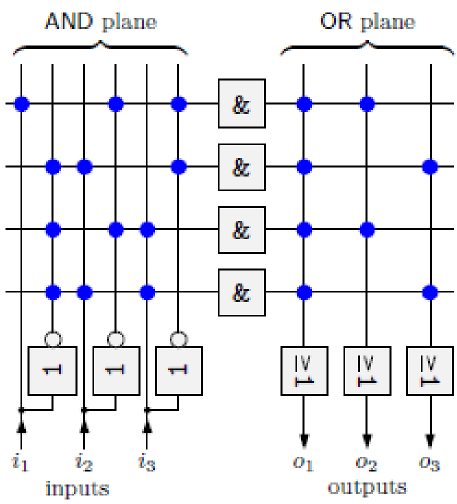
2.2.2 Hardware-Implementation

Abbildung 2.4: Originale Darstellung von Schaltungen



Bei Auslieferung sind Logikarrays komplett verbunden (an jedem UND-Gatter liegen alle negierte und unnegierte Eingaben an). Beim programmieren"des Logikarrays werden die rot markierten Verbindungen getrennt

Abbildung 2.5: Vereinfachte Darstellung von Schaltungen



Zur Vereinfachung stellt man nur eine Linie je Gatter dar und markiert die **Verbindungen**

Für eine elektronische Implementierung (durch Transistoren) sind AND- und OR-Gatter nicht besonders gut geeignet. Besser geeignet sind NAND- und NOR-Gatter (und ggf. NOT-Gatter).

Hardware-Beschreibungssprache

DEFINITION: Hardware-Beschreibungssprache

Eine formale Sprache, mit der Operationen von integrierten Schaltungen und ihr Design beschrieben sowie in Simulationen getestet werden können.

- Spezifieren von dem Verhalten von Bausteinen (Schnittstelle), sowie Implementierung aus Elementargattern.
- Simulator kann Elementargatter (und damit inderekt alle aus diesen zusammengesetzten Bausteine) simulieren
- Simuliertes Verhalten kann automatisch mit Schnittstelle verglichen werden (*automatische Fehlerüberprüfung*)

2.3.1 Aufbau

Hardware-Beschreibung eines Bauteils durch drei Dateien:

- ∗.*cmp* Beschreibung der Schnittstelle durch Ein-/Ausgabetupel
- ∗.*hdl* Beschreibung der Implementierung durch Gatterzusammenschaltung
- .*tst* Befehle zum Durchführen eines Funktionstests

LESEN:
The Elements of Computing Systems: Building a Modern Computer from First Principles
Noam Nisan & Shimon Schocken
MIT Press, Cambridge, MA, USA 2008
Bearbeiten: http://www1.idc.ac.il/tecs/projects/01/index.htm 66

Kapitel 3

Binärarithmetik & Ihre Implementierung

Erinnerung: Arithmetik

3.1.1 Zahlensysteme

Im prinzip kann jede beliebige Zahl als Basis eines Zahlensystems gewählt werden. Das in der Rechnertechnik verwendete *Binärsystem* (Basis 2) hat den Vorteil der kleinstmöglichen Zahl an Ziffernzeichen, nämlich nur zwei:

$$0,\ 1$$

Das auch häufig verwendete *Hexadezimalsystem* (Basis 16):

$$0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ A,\ B,\ C,\ D,\ E,\ F$$

3.1.2 Addition und Subtraktion

Addition und Subtraktion kann sehr leicht stellenweise in einem Stellensystem ausgeführt werden.
Einziges Problem ist die Behandlung von *Stellenüberlauf* und *Stellenunterlauf*. In diesen Fällen entsteht ein *Übertrag*.
Diese Rechenschemata sind nicht nur im Dezimalsystem, sondern im Prinzip in jedem Zahlensystem anwendbar.

$$\begin{array}{cccccccccc}
 & 4 & 6 & 7 & 8 & 5 & 3 & 9 & 9 \\
+ & 2_1 & 8 & 0_1 & 7 & 1 & 2_1 & 3_1 & 4 \\
\hline
= & 7 & 4 & 8 & 5 & 6 & 6 & 3 & 3 \\
 & 4 & 6 & 7 & 8 & 5 & 3 & 9 & 9 \\
- & 2_1 & 8 & 0_1 & 7 & 1 & 2_1 & 3_1 & 4 \\
\hline
= & 4 & 6 & 7 & 8 & 5 & 3 & 9 & 9
\end{array}$$

3.1.3 Multiplikation und Division

Die Multiplikation wird auf eine Summe von Stellenprodukten zurückgeführt.
Hierin besteht das Hauptproblem darin, den richtigen Stellenfaktor zu bestimmen.
Meist wird er geschätzt, anschließend ausprobiert und gegebenenfalls die Schätzung korrigiert.

Abbildung 3.1: Beispiel von Multiplikation

						4	6	7	8	5	3	9	9	
$\times$									9	6	4	3	1	
						4	6	7	8	5	3	9	9	1
+				1	4	0	3	5	6	1	9	7		3
+			1	8	7	1	4	1	5	9	6			4
+		2	8	0	7	1	2	3	9	4				6
+	4	2	1	0	6	8	4	9	1					9
=	4	5	1	1	5	6	2	8	1	0	9	6	9	

Binäre Arithmetik und Implementierung durch Schaltkreise

Die beiden Ziffern des Binärsystems können direkt durch Schalter Implementiert werden (0: offen (falsch), 1: geschlossen (wahr))

Addition

Die Addition ist die einfachste und am häufigsten verwendete Operation in der arithmetisch-logischen Einheit, da sie sich direkt in Wahrheitstafeln übersetzen lassen:

An der Addition von zwei Bits mit Wert 1 zeigt,
warum wir für den Ein-Bit-Addierer zwei Ausgänge benötigen:

- Die *Summe* (sum, s)
- Den *Übertrag* (carry, c)

Die bitweise Addition kann offenbar auch durch (Logik-)gatter implementiert werden.

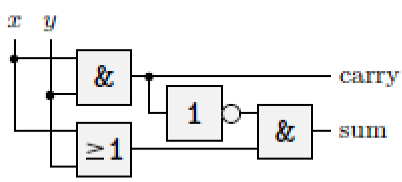
x	y	s	c
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

3.3.1 Halbaddierer

Wegen der möglichkeit eines Übertrags benötigt man für die Addition nicht nur eine Boolesche Funktion, sondern zwei (wobei die zweite den Übetrag bestimmt)

$c = x \wedge y$ $s = (x \wedge \overline{y}) \vee (y \wedge \overline{x})$

Abbildung 3.2: Halbaddierer

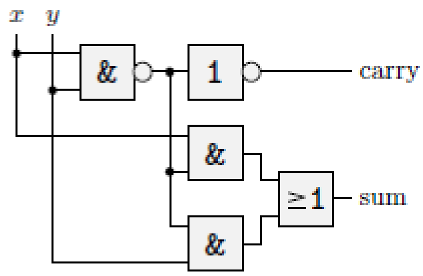


Verknüpfung der beiden Funktionen erlaubt eine Implementierung mit weniger Gattern:

$$s = (x \wedge \overline{(x \wedge y)} \vee (y \wedge \overline{(x \wedge y)}))$$

(Ableitbar über Axiome)

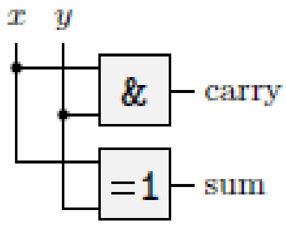
Abbildung 3.3: Halbaddierer mit weniger Gattern



Durch weiteres Ausnutzen der Booleschen Gesetze zur Vereinfachung und durch Nutzung anderer, zusammengesetzter Gatter ergibt sich dann eine deutlich kleinere Schaltung:

$$\begin{aligned} s &= (x \wedge \overline{y}) \vee (\overline{x} \wedge y) \\ &= x \oplus y \end{aligned}$$

Abbildung 3.4: Optimierter Halbaddierer



3.3.2 Volladdierer

Um Addition nicht nur für ein Bit, sondern für n Bits, $n \geq 2$, zu implementieren, braucht man einen
Addierer mit drei Eingängen:
Ein Volladdierer berücksichtigt den Übertrag einer vorangehenden Addition

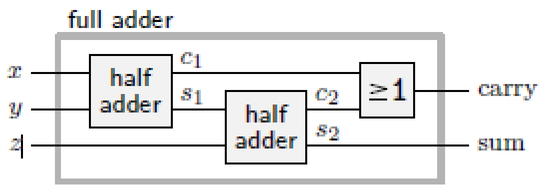
x	y	c_{in}	s	c_{out}
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Ein Volladdierer berechnet die Funktion

$$s = (x + y) + x_{in}$$

Aus diesem Grund wird er am einfachsten aus zwei Halbaddierern zusammengesetzt (mit $z = x_{in}$)

Abbildung 3.5: Volladdierer



3.3.3 n-Bit-Addierer

n-Bit-Übertragungskette-Addierer

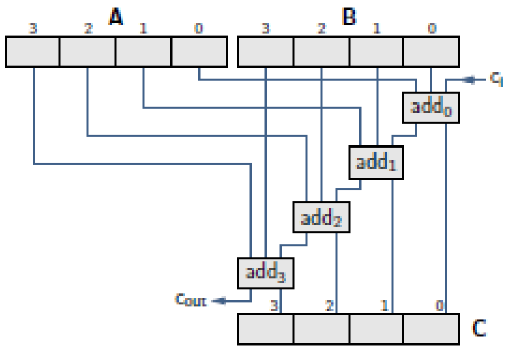
DEFINITION: Übertragungskette-Addierer

In einem n-Bit-Übertragungskette-Addierer wird mithilfe von n Volladdierern Addition durchgeführt

ein n-Bit-Übertr.kette-Addierer (carry ripple adder) für $C = A + B$ kann leicht aus n Volladdierern zusammengesetzt werden

Probleme:
- Übertrag breitet sich wellenartig durch die Addiererkette aus
- Volladdierer add_k kann erst anfangen, wenn add_{k-1} seine Berechnung abgeschlossen hat

Abbildung 3.6: Übertragzungskette-Addierer



n-Bit-Übertragsauswahl-Addierer

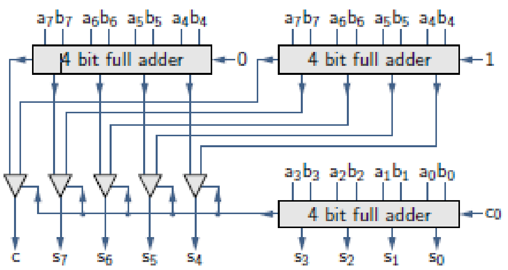
DEFINITION: Übertragsauswahl-Addierer

In einem n-Bit-Übertragsauswahl-Addierer werden die Summen des unteren Halbwortes (Bits 0 bis $\frac{n}{2} - 1$) und des oberen Halbwortes (Bits $\frac{n}{2} - 1$ bis n) parallel berechnet.
Dadurch umgeht man die Problematik des wellenartigen Übertrags.

Da der Wert des Übertrags aus dem unteren Halbwort noch nicht bekannt ist, wenn die Summenbildung für das obere Halbwort beginnt, wird diese Summe zweimal, in getrennten Schaltungen berechnet:
- Schaltung 1: $c_{in} = 0$
- Schaltung 2: $c_{in} = 1$

Wenn der Übertrag des unteren Halbworts berechnet ist, wird er über einen Multiplexer zur Auswahl des richtigen oberen Halbworts benutzt

Abbildung 3.7: Übertragungsauswahl-Addierer



INFO:

Bei längeren Binärzahlen wird das Prinzip des Übertragungsauswahl-Addierers rekusiv angewandt (d.h., die Halbwörter werden ihrerseits zerlegt)

3.3.4 Darstellung negativer Zahlen

Betrag & Vorzeichen
Höchstwertiges Bit gibt Vorzeichen an
<u>Problem</u>: Zahlen sollten gleiche Stellenzahl aufweisen

Einerkomplement
Negation durch Inversion der Zahl
<u>Problem</u>: Übertrag, Zwei Darstellungen für Null

Zweierkomplement
Negation durch Inversion und Addition von 1
<u>Problem</u>: Übertrag, Zwei Darstellungen für Null

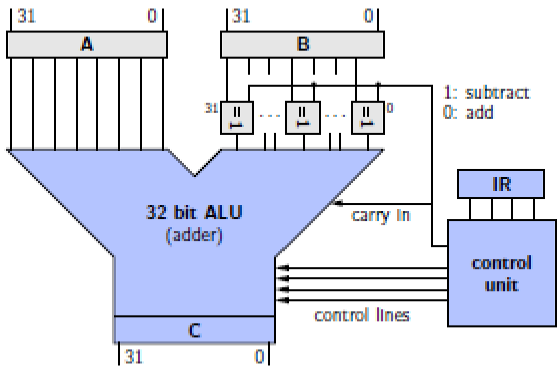
n-Bit-Addierer & -Subtrahierer im Zweierkomplement

DEFINITION: Addierer & Subtrahierer im 2er-Komplement

n-Bit-Subtrahierer bestehen aus eine, n-Bit-Addierer und Gattern, die die Negationsregel implementieren.

Idee: $A - B = A + (-B)$

Abbildung 3.8: Beispiel eines n-Bit-Subtrahierers



Multiplikation

Binärarithmetik p37ff

3.4.1 Multiplikation mit Potenzen der Basis

3.4.2 Multiplikation mit Zweierpotenzen: Bit-Schieben

3.4.3 Allgemeine Binäre Multiplikation

3.4.4 Negative Zahlen

3.4.5 Standardalgorithmusa

3.4.6 Booths Algorithmus

Die Arithmetisch-Logische-Einheit (Arithmetic Logical Unit. ALU)

3.5.1 Die ALU in der Hack-Architektur

3.5.2 Einbindung der ALU in den Prozessor der Hack-Architektur

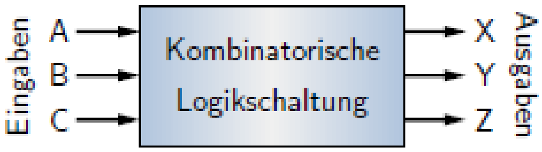
Kapitel 4

Sequentielle Logik

Sequentielle Logik

4.1.1 Logikschaltungen: Kombinatorische & Sequentielle Logik

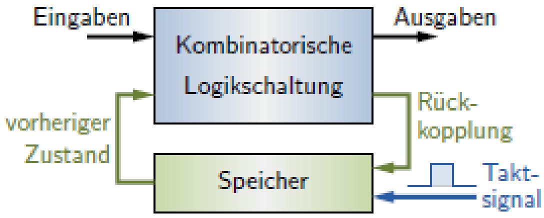
Kombinatorische Logik



Implementierung: Schaltnetze

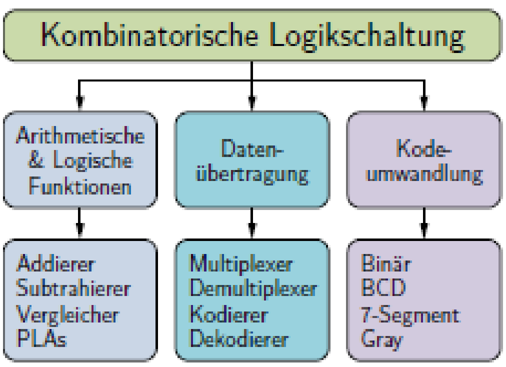
- Einfaches Berechnen von Ein- & Ausgaben
- Keine Informationsspeicherung
- Zustandslosigkeit
- verzögerungsfreie Berechnung

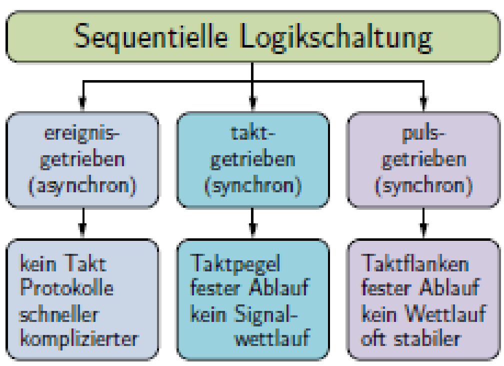
Sequentielle Logik



Implementierung: Schaltwerke

- Rückkopplung von Ausgaben auf Eingaben
- Explizite Informationsspeicherung
- Unterscheidung von Zuständen
- Gatterlaufzeiten explizit berücksichtigt

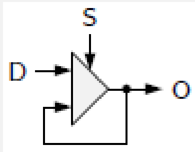




4.1.2 Prinzip der Rückkopplung

DEFINITION: Rückkopplung

Durch Rückkopplung werden Ausgaben als Eingabesignal verwendet. Hierdurch kann ein (Ausgabe-) Zustand festgehalten werden, bis ein Ergeignis ihn wieder ändert.



Probleme der Rückkopplung Durch Rückkopplung kann es zu (unerwünschten) *Schwingungen* kommen. Diese sind ein Beispiel für *Signallaufzeitprobleme*, die durch Rückkopplungen auftreten können).

Ein weiteres Beispiel sind sogenannte *Wettlaufsituationen (Race Conditions)*, die auftreten, wenn sich zwei Signale auf zwei oder mehr Wegen ausbreiten, und die Ausgabe davon abhängen kann, welches Signal schneller weitergegeben wird

Rückkopplungen lassen sich am leichtesten durch ein zentral erzeugtes *Taktsignal* vermeiden, das bestimmt, wann Eingaben übernommen werden.

4.1.3 Asynchrone und synchrone Schaltwerke

Asynchrone Schaltwerke

- Direkte Steuerung durch Änderung der Eingangssignale
- Wann und ob ein stabiler Zustand erreicht wird von Gatterlaufzeit abhängig
- Oft komplizierter, aufwendiger Entwurf
- Sehr schnelle Schaltwerke möglich

Synchrone Schaltwerke

- Steuerung durch Taktsignal
- Eingangssignale nur zu von Takt festgelegten Zeitpunkten übernommen
- Meist einfacher, systematischer Entwurf.
- Langsamstes Bauteil bestimmt maximal mögliche Taktfrequenz

4.1.4 Taktsignal (Clock Signal)

Bistabile Kippstufen

Register, Zähler und Speicher

Hardware-Simulation der sequentiellen Logik