Digitaltechnik & Rechnersysteme

Spezielle Schaltnetze, Don't Cares, Arithmetik

Martin Kumm

Hochschule Fulda
University of Applied Sciences

AI | Angewandte Informatik

WiSe 2023/2024

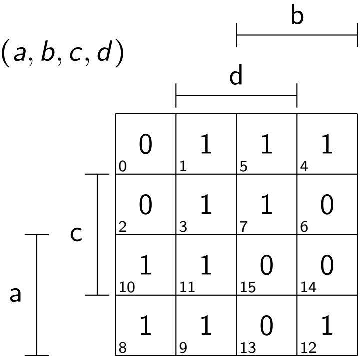
Was bisher geschah...

- Normalformen
  - Umrechnung in Normalformen (z.B. DNF in KDNF)
  - NAND/NOR Umformung
- KV-Diagramme
  - Aufbau
  - Ordnung von Termen
- Definitionen: Primimplikant, KPI, API, REPI
  - KPI: Enthält mindestens eine 1 die von keinem anderen Primimplikanten überdeckt wird
  - API: Alle 1en werden von KPIs überdeckt
  - REPI: Alle anderen
- Minimierung mit KV-Diagrammen
  - KPI werden immer benötigt
  - API werden nie benötigt
  - Aus REPIs muss eine geschickte Auswahl getroffen werden

Beispiel aus letzter Vorlesungsaufgabe

Markieren Sie alle Primimplikanten.

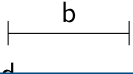
$f(a, b, c, d)$



0 0	1 1	1 5	1 4
0 2	1 3	1 7	0 6
1 10	1 11	0 15	0 14
1 8	1 9	0 13	1 12

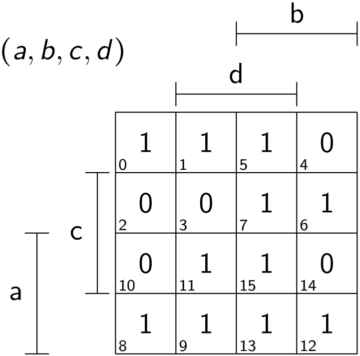
Welche Primimplikanten sind zur Minimierung notwendig?

$f(a, b, c, d)$



Vorlesungsaufgabe

$f(a, b, c, d)$



1 0	1 1	1 5	0 4
0 2	0 3	1 7	1 6
0 10	1 11	1 15	0 14
1 8	1 9	1 13	1 12

Bestimmen Sie die minimierte Funktion!

WrapUp: Darstellungsalternativen

Darstellungsformen für Schaltfunktionen (gleichberechtigt)

- Wahrheitstabelle
- KV-Diagramm
- Boolesche Funktion (Polynomdarstellung)
- Schaltbild (grafische Darstellung durch Schaltsymbole)

Für diese gilt:

- Für eine Wahrheitstabelle (KV-Diagramm) existieren mehrere Boolesche Funktionen und mehrere Schaltbilder
- Für eine Boolesche Funktion existieren mehrere Schaltbilder aber genau eine Wahrheitstabelle (KV-Diagramm)
- Für ein Schaltbild existiert genau eine Boolesche Funktion und genau eine Wahrheitstabelle (KV-Diagramm)

Eigene Aufgabe im 7. Übungsblatt

Im 7. Übungsblatt sollen Sie eine »Eigene Aufgabe« erstellen

Hier sollen Sie eine eigene kombinatorische Problemstellung überlegen welche mit den eingeführten *Tools* (Wahrheitstabelle, KV-Diagramm, etc.) bearbeitet und über einen alternativen Rechenweg überprüft werden soll.

Die Lösung soll per Moodle geteilt werden (extra Übungsmaterial)

Wenn genügend Aufgaben zusammenkommen werde ich eine davon in der Klausur abfragen (nach Ankündigung!) 😉

Inhalte

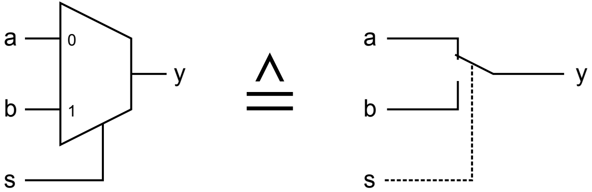
1. Wrap-Up
2. MUX
3. Decoder
4. Don’t Care
5. Addition/Subtraktion
   - Grundlegende Addition
   - Ripple-Carry Addierer
   - Grundlegende Subtraktion

Multiplexer (MUX)

Ein 2 : 1 Multiplexer (MUX) kann 2 Eingänge auf einen Ausgang schalten

Gesteuert wird dies über einen Steuer-Eingang (*Select*)

Funktionsweise: Wenn der Select-Eingang $s = 0$, wird Eingang 0 durchgeschaltet, wenn $s = 1$, wird Eingang 1 durchgeschaltet.

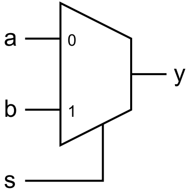


Anwendungsgebiete: Programmierbare (Rechner-)verbindungen

Multiplexer (MUX)

Funktionsweise: Wenn der Select-Eingang $s = 0$, wird Eingang 0 durchgeschaltet, wenn $s = 1$, wird Eingang 1 durchgeschaltet.

Schaltsymbol:



Wahrheitstabelle:

s	a	b	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

KDNF: $y = \overline{s}\, a\overline{b} + \overline{s}\, ab + s\overline{a}\, b + sab$

Multiplexer

Vereinfachung:

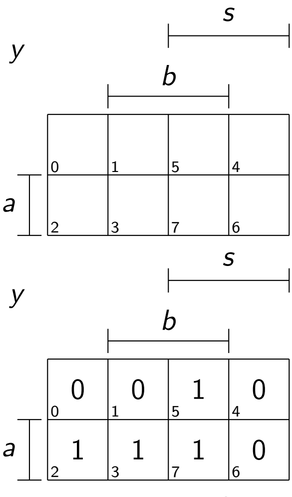
$$
\begin{aligned}
y &= \overline{s}\,a\overline{b} + \overline{s}\,ab + s\overline{a}\,b + sab \\
&= \overline{s}\,a(\overline{b} + b) + sb(\overline{a} + a) \\
&= \overline{s}\,a\underbrace{(\overline{b} + b)}_{=1} + sb\underbrace{(\overline{a} + a)}_{=1} \\
&= \overline{s}\,a + sb
\end{aligned}
$$

Multiplexer

Angewandte Informatik

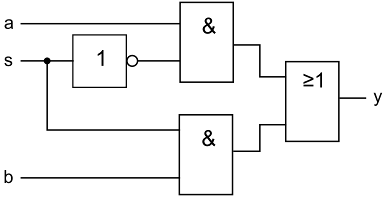
KV-Diagramm:

s	a	b	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

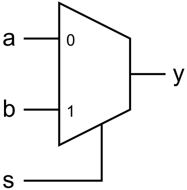


0	0	1	0
1	1	1	0

2 : 1 Multiplexer Symbol



Realisierung

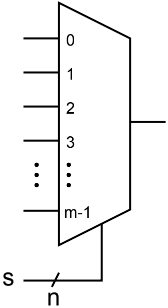


Symbol

Allgemeiner Multiplexer

Angewandte Informatik

Bei $m = 2^n$ Eingängen werden n Steuerleitungen für die Auswahl benötigt.



Die Ein/Ausgabe kann hier auch mehr Bits/Datenwörter umfassen.

Decoder

Schaltnetz, das n Eingänge auf 2^n Ausgänge abbildet.

Schaltet für jede Eingangskombination genau einen Ausgang auf 1 (auch *one hot* code genannt)

Allgemein können Ausgangsseitig weniger als 2^n Ausgänge vorgesehen sein: $m \leq 2^n$ bei n-zu-m-Decodern

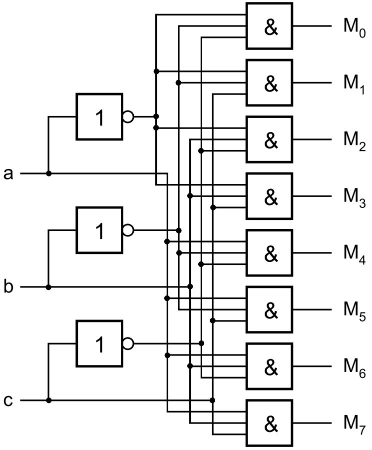
Beispiel: 3 : 8 Decoder

Wahrheitstabelle zum 3 : 8 Decoder:

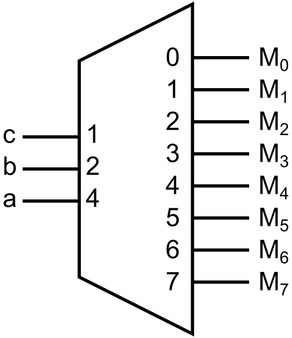
Index	a	b	c	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	0	1	0	0
3	0	1	1	0	0	0	0	1	0	0	0
4	1	0	0	0	0	0	1	0	0	0	0
5	1	0	1	0	0	1	0	0	0	0	0
6	1	1	0	0	1	0	0	0	0	0	0
7	1	1	1	1	0	0	0	0	0	0	0

Ausgang i entspricht Minterm M_i.

3 : 8 Decoder Schaltbild & Symbol



Realisierung



Symbol

Don't Care Belegungen

Häufig tritt die Situation auf, dass nicht für jede Belegung x der Wert der Funktion $f(x)$ zugeordnet werden muss oder kann. Es kann vielmehr offen bleiben, ob $f(x) = 1$ oder $f(x) = 0$ gesetzt wird.

Man bezeichnet solche Zuordungen mit **don't care** und spricht von einer **Redundanz** oder **Freistelle** der Funktion.

Statt einer 0 oder 1 wird häufig das Zeichen „–“ zugeordnet (oft auch: „d“ oder „*“).

Dies stellt aber keinen dritten Wert dar, sondern zeigt nur an, dass an dieser Stelle die Funktion wahlweise zu 0 oder zu 1 gesetzt werden kann. Das lässt sich bei der Minimierung ausnutzen!

Beispiel für Don't Care

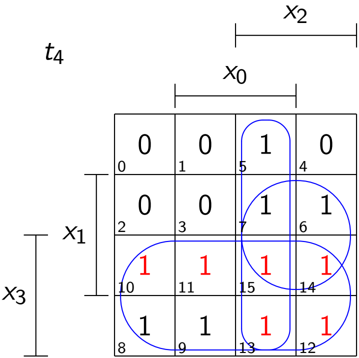
Angewandte Informatik

Thermometer-Code für die Ziffern 0 . . . 9

x_3	x_2	x_1	x_0	t_8	t_7	t_6	t_5	t_4	t_3	t_2	t_1	t_0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	1	1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	1	1	1	1
0	1	0	1	0	0	0	0	1	1	1	1	1
0	1	1	0	0	0	0	1	1	1	1	1	1
0	1	1	1	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1
1	0	1	0	–	–	–	–	–	–	–	–	–
1	0	1	1	–	–	–	–	–	–	–	–	–
1	1	0	0	–	–	–	–	–	–	–	–	–
1	1	0	1	–	–	–	–	–	–	–	–	–
1	1	1	0	–	–	–	–	–	–	–	–	–
1	1	1	1	–	–	–	–	–	–	–	–	–

Beispiel für Don't Care

x_3	x_2	x_1	x_0	t_4
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



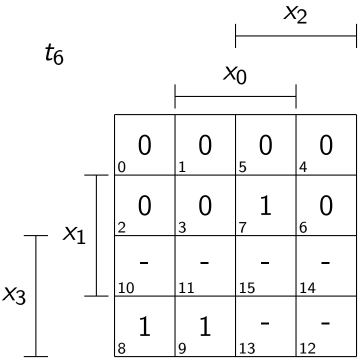
0 0	0 1	1 5	0 4
0 2	0 3	1 7	1 6
1 10	1 11	1 15	1 14
1 8	1 9	1 13	1 12

$$t_4 = x_3 + x_0x_2 + x_1x_2$$

Vorlesungsaufgabe

Bestimmen Sie die Primimplikanden für t_6!

x_3	x_2	x_1	x_0	t_6
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	–
1	0	1	1	–
1	1	0	0	–
1	1	0	1	–
1	1	1	0	–
1	1	1	1	–



0 0	0 1	0 5	0 4
0 2	0 3	1 7	0 6
- 10	- 11	- 15	- 14
1 8	1 9	- 13	- 12

Hauptklassen von Schaltfunktionen

Man definiert zwei Hauptklassen von Schaltfunktionen: Eine Schaltfunktion heißt

1. **vollständig** (definiert), wenn für alle Belegungen x ein Funktionswert $f(x) \in \{0,1\}$ fest zugeordnet wird.
2. **unvollständig** (definiert), wenn es mindestens eine Belegung x gibt, der kein Funktionswert $f(x) \in \{0,1\}$ fest zugeordnet wird.

Wegen $|\{0,1\}^n| = 2^n$ lässt sich bei unvollständigen Schaltfunktionen aus jeweils zwei Teilmengen die dritte bestimmen.

Grundlegende Addition

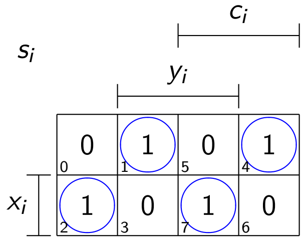
- Betrachten wir die Addition zweier ganzer Zahlen $X = 1010111100_2$ und $Y = 1010010_2$ $(700_{10} + 82_{10})$:

$$\begin{array}{rr} X & 1010111100_2 \\ +Y & +0001010010_2 \\ \hline =S & =1100001110_2 \end{array}$$

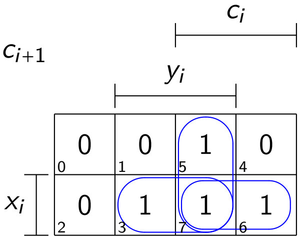
- Das Bit s_i (i-te Bit von S) hängt ab von x_i, y_i und vom aktuellen Übertragsbit c_i (Carry)
- Das nächste Übertragsbit c_{i+1} hängt ebenfalls ab von x_i, y_i und vom aktuellen Übertragsbit c_i
- Beides sind Boolesche Funktionen mit je drei Eingängen
- Die technische Funktionseinheit dieser Funktionen wird Volladdierer (VA) (engl. *full adder* (FA)) genannt

Volladdierer

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



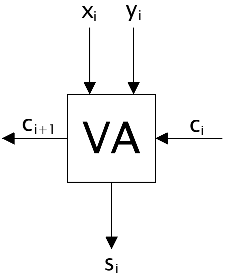
0 0	1 1	0 5	1 4
1 2	0 3	1 7	0 6



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

Volladdierer

Der Volladdierer (VA) abstrahiert die beiden Funktionen in einem Element



$$s_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

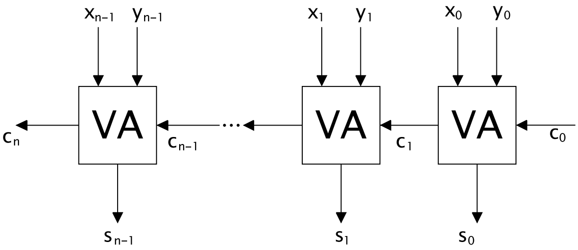
Ohne Carry-In ($c_i = 0$) wird aus dem VA ein Halbaddierer (HA):

$$s = x_i \oplus y_i \text{ und } c_o = x_i y_i .$$

Ripple-Carry Addierer

Ripple-Carry Addierer

- Durch die Weiterschaltung der Überträge zu höherwertigen VAs gelangt man zum Ripple-Carry-Addierer (RCA).
- Direkte Umsetzung der »Papier und Bleistift« Methode.
- Der VA für das LSB (rechts) kann durch einen HA ersetzt werden, wenn kein Carry-Eingang nötig ist.



Grundlegende Subtraktion

Wie funktioniert die Subtraktion?

Die Subtraktion wird über eine Addition mit dem Zweierkomplement realisiert: $D = X - Y = X + (-Y)$

Beispiel: $700_{10} - 82_{10}$ ($X = 01010111100_2$, $Y = 1010010_2$):

$$\begin{array}{rr} \overline{Y} & 11110101101 \\ +1 & 1 \\ \hline -Y = & 11110101110 \end{array}$$

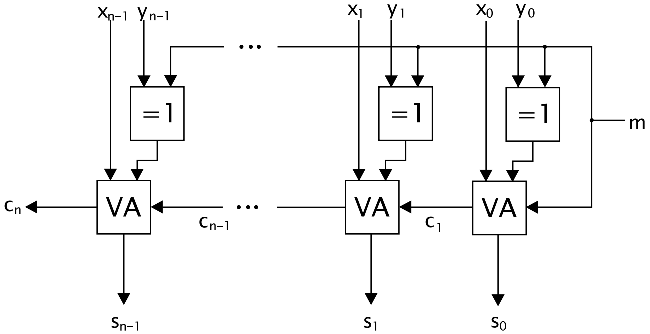
$$\begin{array}{rr} X & 01010111100 \\ +(-Y) & +11110101110 \\ \hline = D & = 1|01001101010 \end{array}$$

- Überträge werden im Zweierkomplement ignoriert!
- Alle Zahlen (auch Ergebnis) müssen gleiche Wortbreite haben!
- Alle Zahlen (auch Ergebnis) müssen auch darstellbar sein!

Ripple-Carry Addierer/Subtrahierer

Für die Subtraktion wird das Komplement gebildet und über das Carry-In eine Eins hinzuaddiert.

Ein Umschaltbarer Addierer/Subtrahierer lässt sich somit über zusätzliche XOR-Gatter realisieren ($m = 0$ Addieren, $m = 1$ Subtrahieren):



Moderne Addition

Der Ripple Carry Addierer ist die einfachste Addiererschaltung, leider auch die langsamste.

Durch die Weiterreichung der Überträge an die nächste Stelle müssen in einem *n*-Bit Addierer insgesamt *n* Volladdierer durchlaufen werden.

In modernen Addierern wird daher versucht Überträge »vorausschauend« zu berechnen (sog. *Carry-Look-Ahead*)

⇒ Bei Interesse sehen wir uns wieder im Modul »Computerarithmetik« im Master AI! 🙂