

Rechnerstrukturen WS 2012/13

- ▶ Boolesche Funktionen und Schaltnetze
- ▶ KV-Diagramme (Wiederholung)
- ▶ Algorithmus von Quine und McCluskey (Wiederholung)
- ▶ Unvollständig definierte Funktionen
 - ▶ Einleitung
 - ▶ Minimalpolynome für partiell definierte Funktionen
- ▶ Hazards
 - ▶ Einleitung
 - ▶ Schaltungshazards
- ▶ Programmierbare Bausteine
 - ▶ Einleitung
 - ▶ Einsatz von PLAs

Minimalpolynombestimmung mit KV-Diagramm

Wieso suchen wir eigentlich Minimalpolynome?

Aufgabe Bestimme für $f: \{0,1\}^4 \rightarrow \{0,1\}$ ein Minimalpolynom.

Vorgehen

1. Eintragen der Funktion ins KV-Diagramm
2. Finden aller maximaler Zweierpotenz-Rechtecke
3. Finden eines Primimplikanten für jedes Rechteck
4. Finden einer Überdeckung aller Einsen durch eine minimale Monomauswahl

jetzt noch ein **Beispiel**

$f: \{0,1\}^4 \rightarrow \{0,1\}$ mit $(1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1)$

Eintragen der Funktion ins KV-Diagramm

$f: \{0, 1\}^4 \rightarrow \{0, 1\}$ mit $(\overset{0}{1}, \overset{1}{0}, \overset{2}{1}, \overset{3}{1}, \overset{4}{0}, \overset{5}{1}, \overset{6}{0}, \overset{7}{1}, \overset{8}{0}, \overset{9}{1}, \overset{10}{0}, \overset{11}{1}, \overset{12}{0}, \overset{13}{0}, \overset{14}{1}, \overset{15}{1})$

$x_1 \ x_2$

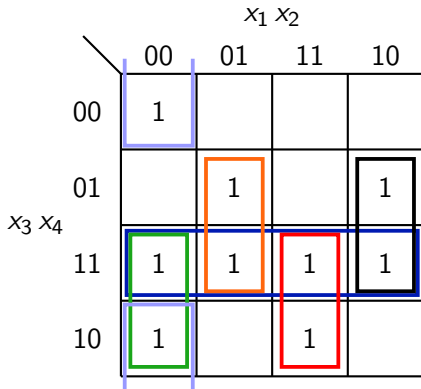
	00	01	11	10
00	1			
01		1		1
11	1	1	1	1
10	1		1	

$x_3 \ x_4$

- 0 = (00 00)₂
- 1 = (00 01)₂
- 2 = (00 10)₂
- 3 = (00 11)₂
- 4 = (01 00)₂
- 5 = (01 01)₂
- 6 = (01 10)₂
- 7 = (01 11)₂
- 8 = (10 00)₂
- 9 = (10 01)₂
- 10 = (10 10)₂
- 11 = (10 11)₂
- 12 = (11 00)₂
- 13 = (11 01)₂
- 14 = (11 10)₂
- 15 = (11 11)₂

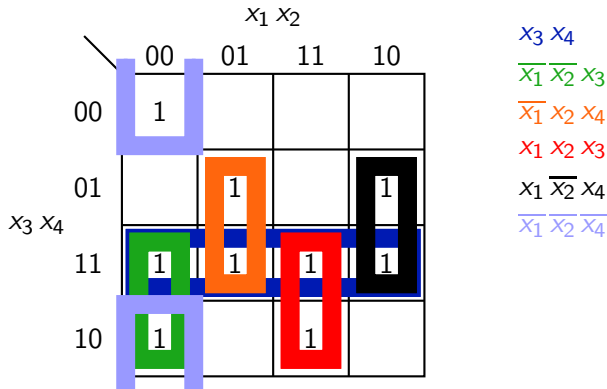
Finden aller maximaler Zweierpotenz-Rechtecke

$f: \{0, 1\}^4 \rightarrow \{0, 1\}$ mit $(1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1)$



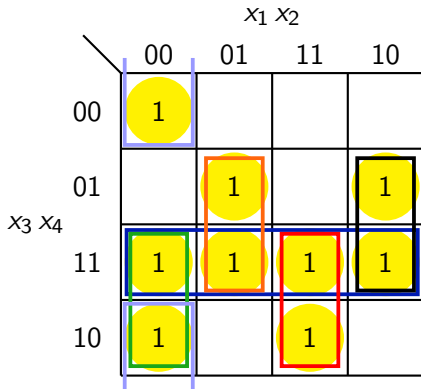
Finden eines Primimplikanten für jedes Rechteck

$f: \{0, 1\}^4 \rightarrow \{0, 1\}$ mit $(1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1)$



Überdeckung aller Einsen durch minimale Monomauswahl

$f: \{0, 1\}^4 \rightarrow \{0, 1\}$ mit $(1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1)$



$x_3 x_4$ beste Wahl

$\overline{x_1} \overline{x_2} x_3$

$\overline{x_1} x_2 x_4$ erforderlich

$x_1 x_2 x_3$ erforderlich

$x_1 \overline{x_2} x_4$ erforderlich

$\overline{x_1} \overline{x_2} \overline{x_4}$ erforderlich

also $x_3 x_4 \vee \overline{x_1} \overline{x_2} x_3 \vee \overline{x_1} x_2 x_4 \vee x_1 x_2 x_3 \vee x_1 \overline{x_2} x_4 \vee \overline{x_1} \overline{x_2} \overline{x_4}$
 Minimalpolynom

Fazit KV-Diagramme

Mit KV-Diagrammen effizient beide Schritte zur Minimalpolynomberechnung durchführbar

1. Bestimmung aller Primimplikanten
2. Bestimmung einer minimalen Überdeckung

für Funktionen $f: \{0, 1\}^n \rightarrow \{0, 1\}$ für $n \in \{3, 4\}$

klar Das reicht nicht aus.

Wie bestimmen wir Minimalpolynome für $f: \{0, 1\}^n \rightarrow \{0, 1\}$ für größeres n ?

klar Wir suchen einen Algorithmus.

Der Algorithmus von Quine und McCluskey

Willard van Orman Quine (1955)

Edward J. McCluskey (1956)

Algorithmus von Quine/McCluskey

Eingabe Fkt. f als Liste aller Minterme zu einschlägigen Indizes

Ausgabe Minimalpolynom zu f

1. Berechne PI, Menge aller Primimplikanten von f .
2. Berechne minimale f überdeckende Auswahl aus PI.

Algorithmus von Quine/McCluskey: Erster Teil

Algorithmus 11 (Berechnung von PI)

Eingabe L_0 : Liste aller Minterme zu einschlägigen Indizes von f

Ausgabe PI: Menge aller Primimplikanten zu f

1. $i := 0$; PI := \emptyset
2. So lange $L_i \neq \emptyset$
3. $L_{i+1} := \{m \mid \exists x: \{m x, m \bar{x}\} \subseteq L_i\}$ (Resolution)
4. PI := PI $\cup \{m \in L_i \mid m \text{ hat keine echte Verkürzung in } L_{i+1}\}$
5. $i := i + 1$
6. Ausgabe PI

Beispiel PI-Berechnung

1. $i := 0; PI := \emptyset$
2. So lange $L_i \neq \emptyset$
3. $L_{i+1} := \{m \mid \exists x: \{m x, m \bar{x}\} \subseteq L_i\}$
4. $PI := PI \cup \{m \in L_i \mid m \text{ hat keine echte Verkürzung in } L_{i+1}\}$
5. $i := i + 1$
6. Ausgabe PI

$$PI = \{\bar{x}_1 \bar{x}_2 x_3, \bar{x}_1 x_2 x_4, x_1 \bar{x}_2 x_4, x_1 x_2 x_3, x_3 x_4\}$$

$$L_0 = \{\bar{x}_1 \bar{x}_2 x_3 \bar{x}_4, \bar{x}_1 \bar{x}_2 x_3 x_4, \bar{x}_1 x_2 \bar{x}_3 x_4, \bar{x}_1 x_2 x_3 x_4, \\ x_1 \bar{x}_2 \bar{x}_3 x_4, x_1 \bar{x}_2 x_3 x_4, x_1 x_2 x_3 \bar{x}_4, x_1 x_2 x_3 x_4\}$$

$$L_1 = \{\bar{x}_1 \bar{x}_2 x_3, \bar{x}_1 x_3 x_4, \bar{x}_2 x_3 x_4, \bar{x}_1 x_2 x_4, x_2 x_3 x_4, x_1 \bar{x}_2 x_4, \\ x_1 x_3 x_4, x_1 x_2 x_3\}$$

$$L_2 = \{x_3 x_4\}$$

$$L_3 = \emptyset$$

Der Algorithmus von Quine und McCluskey

Algorithmus von Quine/McCluskey

Eingabe Fkt. f als Liste aller Minterme zu einschlägigen Indizes

Ausgabe Minimalpolynom zu f

1. Berechne PI, Menge aller Primimplikanten von f . ✓
2. Berechne minimale f überdeckende Auswahl aus PI.

schwieriges kombinatorisches Problem

nur heuristische Lösung
mit gewissen Freiheitsgraden

Das Überdeckungsproblem

Aufgabe Finde minimale Überdeckung von f mit PI.

PI-Tafel

- ▶ eine Zeile für jeden Primimplikanten
- ▶ eine Spalte für jede Eins-Eingabe
- ▶ Eintrag = $\begin{cases} 1 & \text{Primimplikant überdeckt Eins-Eingabe} \\ 0 & \text{sonst} \end{cases}$

klar PI-Tafel meist riesig groß

klar verkleinern

dazu Verkleinerungsregeln

Freiheit durch

- ▶ Reihenfolge der Regelanwendung
- ▶ Art der Regelanwendung

Problem Lösung oft trotzdem nicht ablesbar

Lösung Backtracking oder Heuristiken

Beispiel PI-Tafel

	0010	0011	0101	0111	1001	1011	1110	1111
$\overline{x_1} \overline{x_2} x_3$	1	1						
$x_3 x_4$		1		1		1		1
$\overline{x_1} x_2 x_4$			1	1				
$x_1 x_2 x_3$							1	1
$x_1 \overline{x_2} x_4$					1	1		

<

Streichung überdeckender Spalten

Beispiel PI-Tafel

	0010	0101	0111	1001	1011	1110	1111
$\overline{x_1} \overline{x_2} x_3$	1						
$x_3 x_4$			1		1		1
$\overline{x_1} x_2 x_4$		1	1				
$x_1 x_2 x_3$						1	1
$x_1 \overline{x_2} x_4$				1	1		

<

Streichung überdeckender Spalten

Beispiel PI-Tafel

	0010	0101	1001	1011	1110	1111
$\overline{x_1} \overline{x_2} x_3$	1					
$x_3 x_4$				1		1
$\overline{x_1} x_2 x_4$		1				
$x_1 x_2 x_3$					1	1
$x_1 \overline{x_2} x_4$			1	1		

<

Streichung überdeckender Spalten

Beispiel PI-Tafel

	0010	0101	1001	1110	1111
$\overline{x_1} \overline{x_2} x_3$	1				
$x_3 x_4$					1
$\overline{x_1} x_2 x_4$		1			
$x_1 x_2 x_3$				1	1
$x_1 \overline{x_2} x_4$			1		

Streichung überdeckter Zeilen

Beispiel PI-Tafel

	0010	0101	1001	1110	1111
$\overline{x_1} \overline{x_2} x_3$	1				
$\overline{x_1} x_2 x_4$		1			
$x_1 x_2 x_3$				1	1
$x_1 \overline{x_2} x_4$			1		

Kernimplikanten-Zeilen

also $f(x_1, \dots, x_4) = \overline{x_1} \overline{x_2} x_3 \vee \overline{x_1} x_2 x_4 \vee x_1 x_2 x_3 \vee x_1 \overline{x_2} x_4$

Unvollständig definierte Funktionen

formal $f: \{0, 1\}^n \rightarrow \{0, 1, *\}$

Bedeutung $f(x) = *$ Funktionswert für x egal (“*don't care*”)

klar Schaltnetz realisiert immer eine
„normale“ boolesche Funktion

Definition

$f': \{0, 1\}^n \rightarrow \{0, 1\}$ heißt **Realisierung** (oder auch **Erweiterung**) von
 $f: \{0, 1\}^n \rightarrow \{0, 1, *\}$ $\Leftrightarrow \forall x \in \{0, 1\}^n: (f'(x) = f(x)) \vee (f(x) = *)$

klar größere Freiheiten

Wie nutzen wir die?

Wahl der Realisierung

Wunsch Finde Realisierung f' mit minimalem Minimalpolynom.

nützliche Begriffe

- ▶ minimale Erweiterung f_0 von f

$$f_0(x) := \begin{cases} f(x) & \text{falls } f(x) \in \{0, 1\} \\ 0 & \text{sonst} \end{cases}$$

- ▶ maximale Erweiterung f_1 von f

$$f_1(x) := \begin{cases} f(x) & \text{falls } f(x) \in \{0, 1\} \\ 1 & \text{sonst} \end{cases}$$

Über extreme Erweiterungen

Beobachtungen

- ▶ Primimplikanten von f_0 sind **keine** echten Verkürzungen von Primimplikanten von f_1
- ▶ Primimplikanten von f_1 **können** echte Verkürzungen von Primimplikanten von f_0 sein

Warum?

klar zusätzliche Einsen verkürzen Primimplikanten höchstens

Also hat f_1 das kürzere Minimalpolynom?

So allgemein gilt das nicht!

denn f_1 kann viel mehr Primimplikanten haben

Über unvollständige definierte Funktionen

Satz 12

Minimalpolynome einer unvollständig definierten Funktion $f: \{0, 1\}^n \rightarrow \{0, 1, *\}$ enthalten ausschließlich Primimplikanten von f_1 .

Beweis Sei $p = m_1 \vee m_2 \vee \dots \vee m_k$ Minimalpolynom von f .

Beobachtung p ist Realisierung von f .

Sei m_i beliebiger Primimplikant in p .

Beobachtung m_i ist Implikant von f_1 .

denn $m_i(x) = 1 \Rightarrow (f(x) = 1) \vee (f(x) = *) \Rightarrow f_1(x) = 1$

also Verkürzung m' von m_i ist Primimplikant von f_1

aber Ersetzen wir m_i in p durch m' , erhalten wir Polynom für f

klar m' kann nicht echte Verkürzung von m_i sein,
sonst wäre p nicht Minimalpolynom zu f

Minimalpolynome unvollständig definierter Funktionen

Algorithmus

1. Berechne Menge aller Primimplikanten $PI(f_1)$ zu f_1 .
2. Berechne minimale Überdeckung von f_0 durch Monome aus $PI(f_1)$.

also algorithmisch keine neuen Probleme

Einwand Vielleicht lässt sich f_0 gar nicht so überdecken?

Beobachtung Überdeckung von Nullen von f_0 **unproblematisch**

Wieso eigentlich?

Primimplikanten von f_1 können nur Einsen oder
don't cares von f überdecken!

Hazards

ärgerlich, aber unvermeidlich Abweichungen zwischen
Modell und Realität

Das gilt auch für technisch realisierte Schaltnetze.

Was passiert, wenn die Eingabe wechselt?

klar Die Ausgabe kann wechseln.

Wunsch

- ▶ wenn f die Ausgabe wechselt, wechselt das Schaltnetz genau einmal seine Ausgabe
- ▶ wenn f die Ausgabe nicht wechselt, wechselt das Schaltnetz seine Ausgabe nicht

Realität Schaltnetz kann Ausgabe „zwischen durch“ wechseln.

Das heißt **Hazard**.

Hazards: Begriffe

Hazards sind

- ▶ **statisch** Ausgabewert soll gleich bleiben, ändert sich aber.
- ▶ **dynamisch** Ausgabewert soll sich ändern, ändert sich aber mehrfach.
- ▶ **Funktionshazard** „schon in der Funktionsdefinition enthalten“
- ▶ **Schaltungshazard** Hazard, der kein Funktionshazard ist

also vier verschiedene Fälle

Statischer Funktionshazard

Definition

$f: \{0, 1\}^n \rightarrow \{0, 1\}$ hat **statischen Funktionshazard**, wenn es $a = a_1 a_2 \cdots a_n \in \{0, 1\}^n$ und $b = b_1 b_2 \cdots b_n \in \{0, 1\}^n$ mit $a \neq b$ und $f(a) = f(b)$ gibt, sowie $c = c_1 c_2 \cdots c_n \in \{0, 1\}^n$ mit $c_i \in \{a_i, b_i\}$ für alle $i \in \{1, 2, \dots, n\}$ und $f(c) \neq f(a)$.

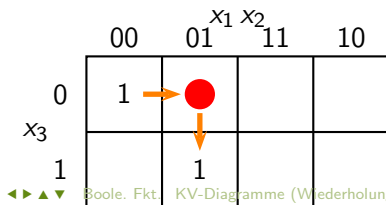
Beispiel

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$a = 000, f(a) = 1$

$c = 010, f(c) = 0$

$b = 011, f(b) = 1$



Dynamischer Funktionshazard

Definition

$f: \{0, 1\}^n \rightarrow \{0, 1\}$ hat **dynamischen Funktionshazard**, wenn es $a, b \in \{0, 1\}^n$ mit $a \neq b$ und $f(a) \neq f(b)$ gibt, sowie $c = c_1 c_2 \cdots c_n$ und $c' = c'_1 c'_2 \cdots c'_n$ mit $c_i \in \{a_i, b_i\}$, $c'_i \in \{c_i, b_i\}$ ($i \in \{1, 2, \dots, n\}$) und $f(c) \neq f(a)$ sowie $f(c) \neq f(c')$.

Beispiel

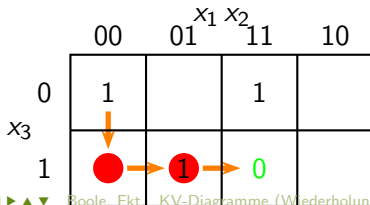
x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$$a = 000, f(a) = 1$$

$$c = 001, f(c) = 0$$

$$c' = 011, f(c') = 1$$

$$b = 111, f(b) = 0$$



Zusammenfassung: Statische/Dynamische Hazards

Wir haben korrektes Schaltnetz S für f .

Wir betrachten Eingabewechsel von a auf b .

1. Funktionswert bleibt gleich: $f(a) = f(b)$
Am Ausgang von S liegt kurzzeitig ein **anderer Wert** an.
statischer Hazard
2. Funktionswert ändert sich: $f(a) \neq f(b)$
Am Ausgang von S ändert sich der Wert **mehrfach**.
dynamischer Hazard

Weitergehende Klassifikation

- ▶ Manche Hazard heißen **Funktionshazards**.
- ▶ Die übrigen Hazards heißen **Schaltungshazards**.

Wie kommt es zum Funktionshazard?

Betrachte Eingabewechsel von a nach b mit
entweder $f(a) = f(b)$ statisch
oder $f(a) \neq f(b)$ dynamisch

Es gibt statisch: 1 andere Eingabe c
dynamisch: 2 andere Eingaben c, c'

zwischen a und b ,
so dass der Funktionswert beim Weg von a nach b

statisch über c wechselt
dynamisch über c und c' mehrfach wechselt

zentral andere Eingaben echt „dazwischen“

Was bedeutet „dazwischen“?

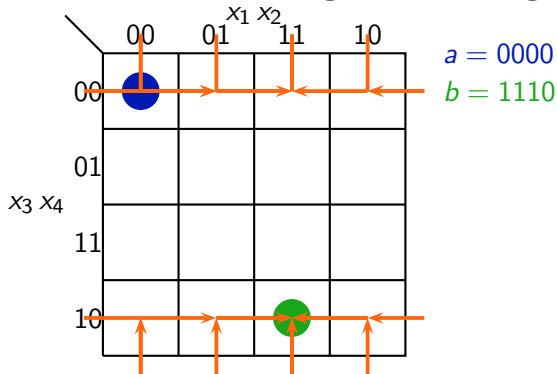
in Bezug auf Eingabewechsel

- ▶ Wertänderung einiger Eingabebits (alle Bits mit $a_i \neq b_i$)
- ▶ nicht völlig gleichzeitig
- ▶ also „dazwischen“ = kann als Zwischenschritt bei Wertänderung der Eingabebits vorkommen

beim Blick auf KV-Diagramm

- ▶ auf einem kürzesten Weg von a nach b

Kürzeste Wege im KV-Diagramm



Beobachtung 3 Bits verschieden
 kürzeste Wege haben Länge 3
 (i.a.: n unterschiedliche Bits \Rightarrow kürzeste Weglänge = n)

Funktions hazards

klar Funktions hazards sind in Realisierungen von Schaltnetzen nicht zu vermeiden

zur Kenntnis nehmen Es gibt Hazards.

für uns interessanter Schaltung hazards

Definition Schaltung hazard Funktion hat bezüglich a, b keinen Hazard, Schaltnetz aber schon

Fragen

1. Wie kann das passieren?
2. Kann man das vermeiden?

Definition Schaltungshazard

Definition

Schaltnetz S für f hat einen **statischen Schaltungshazard**, wenn es $a, b \in \{0, 1\}^n$ gibt, so dass f bezüglich a, b keinen statischen Funktionshazard hat, aber beim Eingabewechsel von a nach b am Ausgang von S **nicht notwendig** stabil $f(a)$ anliegt.

Definition

Schaltnetz S für f hat einen **dynamischen Schaltungshazard**, wenn es $a, b \in \{0, 1\}^n$ gibt, so dass f bezüglich a, b keinen dynamischen Funktionshazard hat, aber beim Eingabewechsel von a nach b am Ausgang von S mehr als ein Funktionswertwechsel **auftreten kann**.

Beachte: Auftreten nicht garantiert!

Beispiel statischer Schaltungshazard

$$f(x_1, x_2, x_3) = x_1 x_2 \vee \overline{x_2} x_3$$

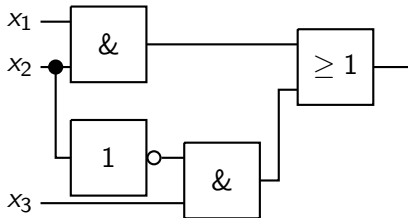
$$a = 111, f(a) = 1$$

$$b = 101, f(b) = 1$$

klar kein Funktionshazard

Gar kein c zwischen a und b !

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Vermeidung von Schaltungshazards

Wir konzentrieren uns auf

- ▶ statische Schaltungshazards,
- ▶ Schaltnetze, die direkt Polynome realisieren.

Sind in solchen Schaltnetzen Schaltungshazards ganz vermeidbar?

gute Nachricht ja

schlechte Nachricht nicht kostenlos

Wie vermeidet man statische Schaltungshazards?

Satz

Sei S ein Schaltnetz, das direkt ein Polynom p einer booleschen Funktion f realisiert. S hat genau dann keinen statischen Schaltungshazard, wenn S für jeden Primimplikanten von f ein Und-Gatter enthält.

Beweis durch Widerspruch:

statischer Schaltungshazard \Leftrightarrow ein Primimplikant fehlt

Beweistrategie

1. **Beweis:** statischer Schaltungshazard \Rightarrow ein Primimplikant fehlt
2. **Beweis:** Primimplikant fehlt \Rightarrow statischer Schaltungshazard

Hazard \Rightarrow Primimplikant fehlt

Voraussetzungen

- ▶ Es gibt statischen Hazard bezüglich a und b .
- ▶ Es gibt keinen Funktionshazard bezüglich a und b .
- ▶ a und b unterscheiden sich in genau einem Bit

1. Fall $f(a) = f(b) = 0$

klar alle Und-Gatter permanent 0

also Wert am Ausgang permanent 0

also kein Hazard ✓

2. Fall $f(a) = f(b) = 1$

Sei m_a Minterm zu a , m_b Minterm zu b

Beobachtung \exists Monom m , Variable x : $\{m_a, m_b\} = \{m x, m \bar{x}\}$

klar m ist Implikant von f

also S enthält Und-Gatter für Verkürzung m' von m

Beobachtung Dieses Und-Gatter ist permanent 1 (da unabh. von x).

Primimplikant fehlt \Rightarrow Hazard

Voraussetzungen

- ▶ S realisiert direkt Polynom p für f
- ▶ \exists Primimplikant m von f : kein Und-Gatter für m in S

Definiere Eingaben a und b .

- ▶ $x_i \in m \rightsquigarrow a_i = b_i = 1$
- ▶ $\bar{x}_i \in m \rightsquigarrow a_i = b_i = 0$
- ▶ sonst $\rightsquigarrow a_i = 0, b_i = 1$

Beobachtung $m(a) = m(b) = 1$

also $f(a) = f(b) = 1$

Beobachtung kein Hazard \Leftrightarrow ein Und-Gatter permanent 1

klar So ein Und-Gatter realisiert Verkürzung von m .

also So ein Und-Gatter gibt es in S nicht.

also Hazard □

Zusammenfassung: Hazards

Fazit Hazards

- ▶ Hazards sind ärgerlich.
Wann liegt der richtige Funktionswert vor?
- ▶ Funktions hazards sind so nicht vermeidbar.
- ▶ Statische Schaltung hazards können mit zusätzlichem Aufwand vermieden werden.
- ▶ Eine grundsätzliche Lösung ist wünschenswert.

mögliche grundsätzliche Lösung: Taktung der Schaltung ⇒ später

Realisierung von Schaltnetzen

Gedanken zur Anwendung

1. Problem
2. boolesche Funktion
3. Schaltnetz-Entwurf
4. Schaltnetz-Realisierung

Realisierungen

- ▶ hoch-integrierte Schaltung
teuer Lohnt sich nur bei großen Stückzahlen.
- ▶ direkte Umsetzung mit Gattern
umständlich

Realisierung mit Gattern

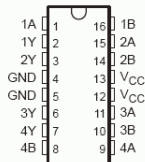
Datenblatt von Texas Instruments (www.ti.com)

74AC11008 QUADRUPLE 2-INPUT POSITIVE-AND GATE

SCAS014C – AUGUST 1987 – REVISED APRIL 1996

- Flow-Through Architecture Optimizes PCB Layout
- Center-Pin V_{CC} and GND Configurations Minimize High-Speed Switching Noise
- EPIC™ (Enhanced-Performance Implanted CMOS) 1- μ m Process
- 500-mA Typical Latch-Up Immunity at 125°C
- Package Options Include Plastic Small-Outline (D) and Thin Shrink Small-Outline (PW) Packages, and Standard Plastic 300-mil DIPs (N)

D, N, OR PW PACKAGE
(TOP VIEW)



description

This device contains four independent 2-input AND gates. It performs the Boolean function $Y = A \cdot B$ or $Y = \overline{A} + \overline{B}$ in positive logic.

The 74AC11008 is characterized for operation from -40°C to 85°C .

FUNCTION TABLE
(each gate)

INPUTS		OUTPUT
A	B	Y
H	H	H
H	L	L
L	H	L
L	L	L

Alternative Realisierung

- ▶ massenhaft produzierte
- ▶ darum **preisgünstige**
- ▶ nach der Fertigstellung in ihrer Funktion noch beeinflussbare
- ▶ **funktional vollständige**
- ▶ also **universelle** Standardbausteine

Programmable Logic Array (PLA)

Varianten PAL, PROM, FPGA, ...
(zum Teil eingeschränkte Funktionalität)

Programmable Logic Array (PLA)

Datenblatt von Lattice (www.latticesemi.com)



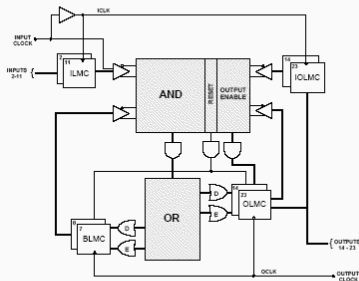
GAL6001

High Performance E²CMOS FPLA
Generic Array Logic™

Features

- HIGH PERFORMANCE E²CMOS® TECHNOLOGY
 - 30ns Maximum Propagation Delay
 - 27MHz Maximum Frequency
 - 12ns Maximum Clock to Output Delay
 - TTL Compatible 16mA Outputs
 - UltraMOS® Advanced CMOS Technology
- LOW POWER CMOS
 - 90mA Typical I_{cc}
- E² CELL TECHNOLOGY
 - Reconfigurable Logic
 - Reprogrammable Cells
 - 100% Tested/100% Yields
 - High Speed Electrical Erasure (<100ms)
 - 20 Year Data Retention
- UNPRECEDENTED FUNCTIONAL DENSITY
 - 78 x 64 x 36 FPLA Architecture
 - 10 Output Logic Macrocells
 - 8 Buried Logic Macrocells
 - 20 Input and I/O Logic Macrocells
- HIGH-LEVEL DESIGN FLEXIBILITY
 - Asynchronous or Synchronous Clocking
 - Separate State Register and Input Clock Pins
 - Functional Superset of Existing 24-pin PAL®

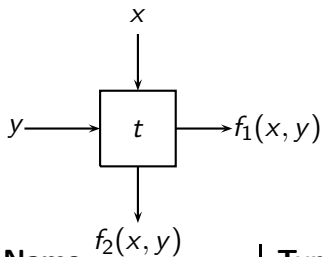
Functional Block Diagram



Macrocell Names

ILMC	INPUT LOGIC MACROCELL
IOLMC	I/O LOGIC MACROCELL
BLMC	BURIED LOGIC MACROCELL

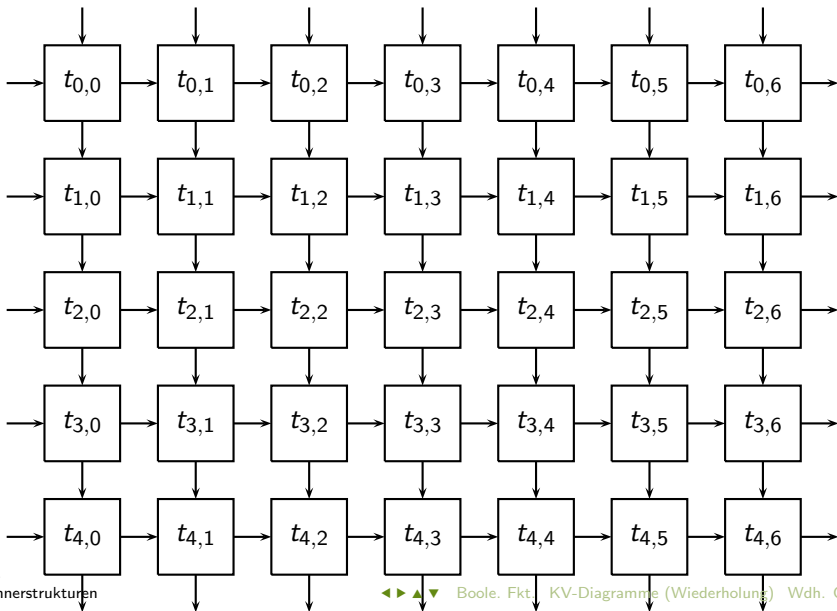
PLA Grundbausteine



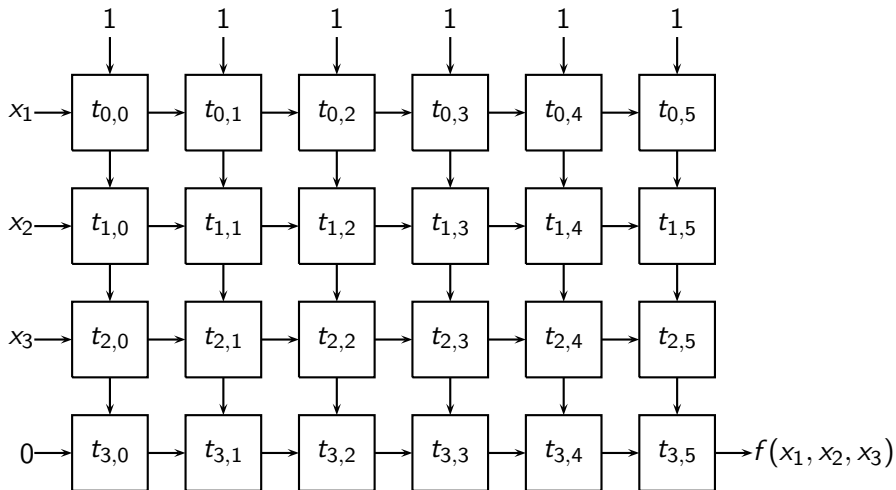
Name	Typ	$f_1(x, y)$	$f_2(x, y)$
Identer	0	y	x
Addierer	1	$x \vee y$	x
Multiplizierer	2	y	$x y$
Negat-Multiplizierer	3	y	$x \bar{y}$

klar funktional vollständig

PLA



PLA für $f: \{0, 1\}^3 \rightarrow \{0, 1\}$



PLA: Bausteinwahl

klar jede Funktion als **Polynom** darstellbar

Erinnerung Polynom = Disjunktion einiger Monome

erster Schritt

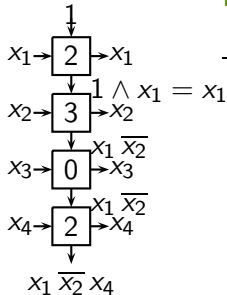
Wie realisieren wir Monome?

exemplarisch am Beispiel $x_1 \overline{x_2} x_4$

PLA: Monomrealisierung

Beispiel Monom $x_1 \bar{x}_2 x_4$

Erinnerung



Name	Typ	$f_r(o, l)$	$f_u(o, l)$
Identer	0	$/$	o
Addierer	1	$o \vee l$	o
Multiplizierer	2	$/$	$o l$
Negat-Multiplizierer	3	$/$	$o \bar{l}$

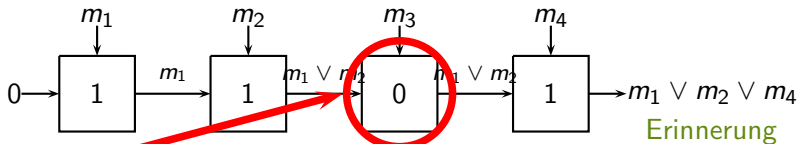
also jedes Monom leicht realisierbar

- ▶ falls Variable fehlt Typ 0
- ▶ falls x_i vorkommt Typ 2
- ▶ falls \bar{x}_i vorkommt Typ 3

PLA: Polynomrealisierung

gesehen für $f: \{0, 1\}^n \rightarrow \{0, 1\}$
 k verschiedene Monome m_1, m_2, \dots, m_k
in n Zeilen und k Spalten realisierbar

Wie können wir f realisieren, z. B. $f = m_1 \vee m_2 \vee m_4$?



Ist das sinnvoll?

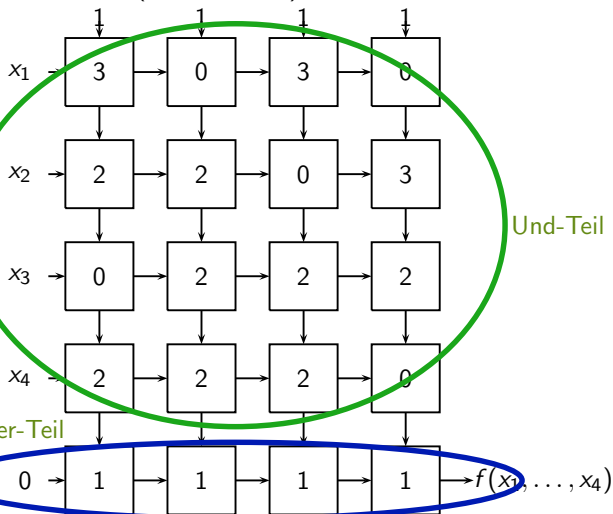
klar für
 $f: \{0, 1\}^n \rightarrow \{0, 1\}$

nicht
aber für
 $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$
schon

Name	Typ	$f_r(o, l)$	$f_u(o, l)$
Identer	0	l	o
Addierer	1	$o \vee l$	o
Multiplizierer	2	l	$o l$
Negat-Multiplizierer	3	l	$o \bar{l}$

PLA: Ein konkretes Beispiel

Beispiel $f(x_1, x_2, x_3, x_4) = \overline{x_1} x_2 x_4 \vee x_2 x_3 x_4 \vee \overline{x_1} x_3 x_4 \vee \overline{x_2} x_3$



Fazit zur PLA-Nutzung

also Wir können jede Funktion $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$,
für deren Polynom insgesamt k Implikanten ausreichen,
mit einem PLA mit $n + m$ Zeilen und k Spalten realisieren.

klar Wir wünschen uns k klein.

dafür Minimalpolynome

Wie findet man Minimalpolynome für Funktionen
 $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$?

Minimalpolynome für $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$

Notation statt $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$
 (f_1, f_2, \dots, f_m) mit $f_i: \{0, 1\}^n \rightarrow \{0, 1\}$
 für $i \in \{1, 2, \dots, m\}$

Definition Ein **Minimalpolynom** für $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$
 ist (p_1, p_2, \dots, p_m) mit minimalen Kosten,
 dabei ist p_i Polynom für f_i . Bei den Kosten
 zählen mehrfach vorkommende Monome nur einmal.

Also suchen wir Minimalpolynome p_i für f_i ? **Nein!**

Beispiel $p_1(x_1, x_2, x_3) = x_1 x_3 \vee \overline{x_2} x_3$ ist Minimalpolynom
 $p_2(x_1, x_2, x_3) = \overline{x_1} x_2 \vee x_2 x_3$ ist Minimalpolynom

$p'_1(x_1, x_2, x_3) = x_1 x_2 x_3 \vee \overline{x_2} x_3$ stellt auch p_1 dar
 $p'_2(x_1, x_2, x_3) = \overline{x_1} x_2 \vee x_1 x_2 x_3$ stellt auch p_2 dar

Gesamtkosten $(p_1, p_2) = (n + m) \times k = (3 + 2) \times 4$

Gesamtkosten $(p'_1, p'_2) = (n + m) \times k' = (3 + 2) \times 3$

Monome für Minimalpolynome für $f: \{0, 1\}^n \rightarrow \{0, 1\}^k$

Welche Monome übernehmen die Rolle der Primimplikanten?

Definition Ein Monom m ist ein **multipler Primimplikant** von $f = (f_1, f_2, \dots, f_k)$ mit $f_i: \{0, 1\}^n \rightarrow \{0, 1\}$, wenn m Primimplikant von $\bigwedge_{i \in I} f_i$ ist für eine nicht-leere Menge $I \subseteq \{1, 2, \dots, k\}$.

Theorem Minimalpolynome für $f: \{0, 1\}^n \rightarrow \{0, 1\}^k$ enthalten nur multiple Primimplikanten von f .

Beweis und **Algorithmus** zur Berechnung \rightarrow Skript

Multiple Primimplikanten berechnen

Wie finden wir alle multiplen Primimplikanten?

Theorem $m \in \text{PI} \left(\bigwedge_{i \in I} f_i \right) \Rightarrow \forall i \in I: \exists m_i \in \text{PI}(f_i): m = \bigwedge_{i \in I} m_i$

Beweis

klar $m \in \text{PI} \left(\bigwedge_{i \in I} f_i \right)$ ist für alle $i \in I$ Implikant von f_i

Sei $m_i \in \text{PI}(f_i)$ Verkürzung von m ($i \in I$)

klar $\bigwedge_{i \in I} m_i$ ist Verkürzung von m

weil $m \in \text{PI} \left(\bigwedge_{i \in I} f_i \right)$, kann $\bigwedge_{i \in I} m_i$ keine echte Verkürzung von m sein

also $\bigwedge_{i \in I} m_i = m$



Berechnung aller multipler Primimplikanten

Algorithmus

berechnet für $I \subseteq \{1, 2, \dots, k\}$ die Menge M_I der multiplen Primimplikanten $m = \bigwedge_{i \in I} m_i$

1. Für $i \in \{1, 2, \dots, k\}$ berechne $M_{\{i\}} := \text{PI}(f_i)$.
2. $M := \bigcup_{i=1}^k \{\{i\}\}$; $M' := \bigcup_{i=1}^k M_{\{i\}}$.
3. Wiederhole
4. Für $I, J \in M$ mit $I \cap J = \emptyset$
5. $M_{I \cup J} := \{m \neq 0 \mid m = m' m'' \text{ mit } m' \in M_I, m'' \in M_J, \text{ keine echte Verkürzung von } m \text{ in } M_{I \cup J}\}$.
6. $M := M \cup \{I \cup J\}$; $M' := M' \cup M_{I \cup J}$.
7. So lange, bis M' eine Iteration unverändert bleibt.

Minimalpolynomberechnung für $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$

Algorithmus

1. Für alle f_i berechne alle Primimplikanten.
2. Berechne alle multiplen Primimplikanten.
(ergeben sich potentiell als paarweise Verknüpfung "normaler" Primimplikanten)
3. Berechne eine möglichst günstige Überdeckung.

also günstigste Darstellung für PLA-Realisierungen
mit uns bekannten Mitteln berechenbar
allerdings **sehr aufwendig**

PLA als ROM

Aufgabe Speichere 2^n „Wörter“ der Länge m .

$$w_0 = w_{0,0} w_{0,1} w_{0,2} \cdots w_{0,m-1} \in \{0, 1\}^m$$

$$w_1 = w_{1,0} w_{1,1} w_{1,2} \cdots w_{1,m-1} \in \{0, 1\}^m$$

$$w_2 = w_{2,0} w_{2,1} w_{2,2} \cdots w_{2,m-1} \in \{0, 1\}^m$$

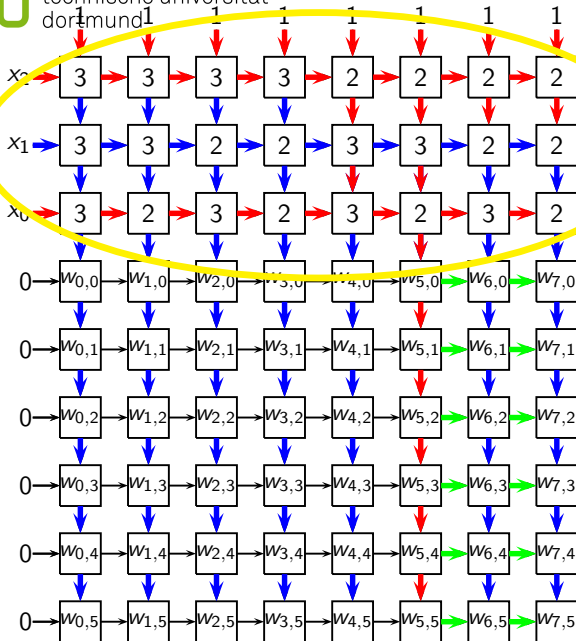
\vdots

$$w_{2^n-1} = w_{2^n-1,0} w_{2^n-1,1} w_{2^n-1,2} \cdots w_{2^n-1,m-1} \in \{0, 1\}^m$$

Benutze PLA mit $n + m$ Zeilen, 2^n Spalten

Beobachtung $m \cdot 2^n$ Zellen für $m \cdot 2^n$ zu speichernde Bits
 mindestens erforderlich

Adressierung mit jeweils n Bits
 in n zusätzlichen Zeilen



Beispiel

$n = 3, m = 6$

$2^3 = 8$ Wörter
jeweils 6 Bits

Beobachtung

Und-Teil fest

nur von Größe
abhängig

Anmerkung

PLAs mit festem
Und-Teil werden
als PROM
verkauft.

Zwischen-Fazit PLAs

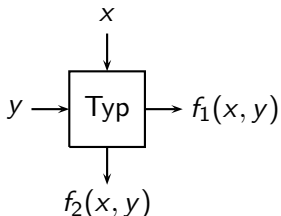
PLAs sind preiswerte, universelle Bausteine, die

- ▶ beliebige boolesche Funktionen leicht realisierbar machen,
- ▶ Minimalpolynomdarstellungen motivieren,
- ▶ Speicherung von 2^n Wörtern der Länge m in einem $(n + m) \times 2^n$ -PLA erlauben.

Nachteil nur einmal programmierbar

Wie kann man PLAs beliebig neu programmierbar machen?

„Software“-PLAs



Typ	s	t	$f_1(x, y)$	$f_2(x, y)$
0	0	0	y	x
1	0	1	$x \vee y$	x
2	1	0	y	$x y$
3	1	1	y	$x \bar{y}$

klar vier verschiedene Typen, mit zwei Bits codierbar

Idee erweitere PLA-Baustein um zwei zusätzliche Eingaben, die den Baustein-Typ codieren

jetzt f_1 und f_2 als Funktionen von (s, t, x, y) darstellen

- ▶ $f_1(s, t, x, y) = y \vee \bar{s} t x$
- ▶ $f_2(s, t, x, y) = \bar{s} x \vee s x (t \oplus y)$

Bemerkung zum Einsatz

klar für ein $n \times m$ -PLA werden zur Programmierung $2nm$ Bits gebraucht.

Beobachtung Man kann diese $2nm$ Bits gut in einem PROM speichern.

Fazit

- ▶ einfache und günstige Realisierung von booleschen Funktionen
- ▶ besonders geeignet für kleine Stückzahlen
- ▶ besonders geeignet bei nur temporärem Gebrauch