Logiksynthese

- Optimierungstechniken für mehrstufige Schaltungen -

Peter Marwedel
Informatik 12

Optimierungstechniken für mehrstufige Schaltungen

Bei ≥ 2 Gatterstufen: Berücksichtigung gemeinsamer Teilausdrücke! Insbesondere bei Funktionenbündeln!
Beispiel: Zu berechnen:

$$y_1 = \bar{a}bcd \vee a\bar{b}cd \vee abc\bar{d} \vee abc\bar{d}$$

$$y_2 = \bar{a}bcd \vee a\bar{b}cd \vee efg$$

Eine einfachere, mehrstufige Realisierung ist:

$$h_1 = cd(\bar{a}b \vee a\bar{b})$$

$$y_1 = h_1 \vee ab(\bar{c}d \vee c\bar{d})$$

$$y_2 = h_1 \vee efg$$

mit der Hilfsfunktion h_1.

Optimierungstechniken für mehrstufige Schaltungen

Probleme: Kostenfunktion?

- Maximale Nutzung gemeinsamer Teilausdrücke starke kapazitive Belastung, langsam
- Keine Nutzung gemeinsamer Teilausdrücke, große Chipfläche

☞ Gegeneinander abzuwägende Ziele:
Minimierung von Chipfläche und Verzögerungszeit.

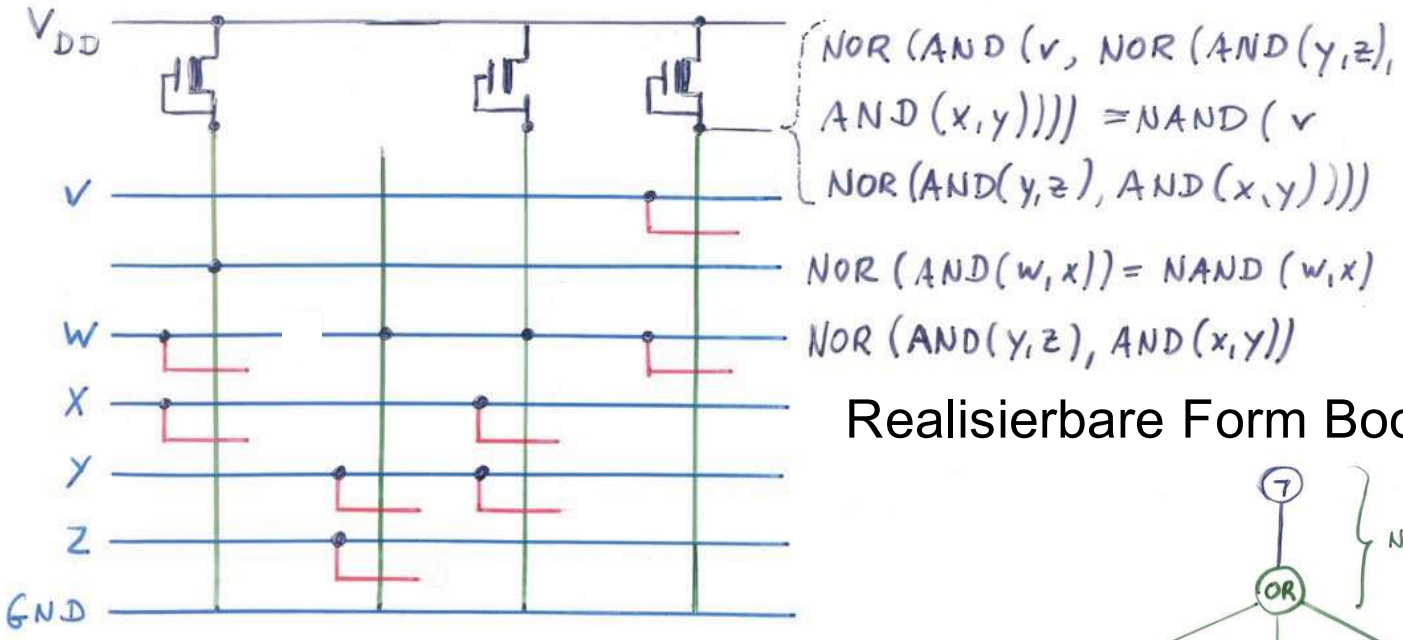
- Abbildung auf Bausteinbibliothek („*technology mapping*“)

Ansätze:

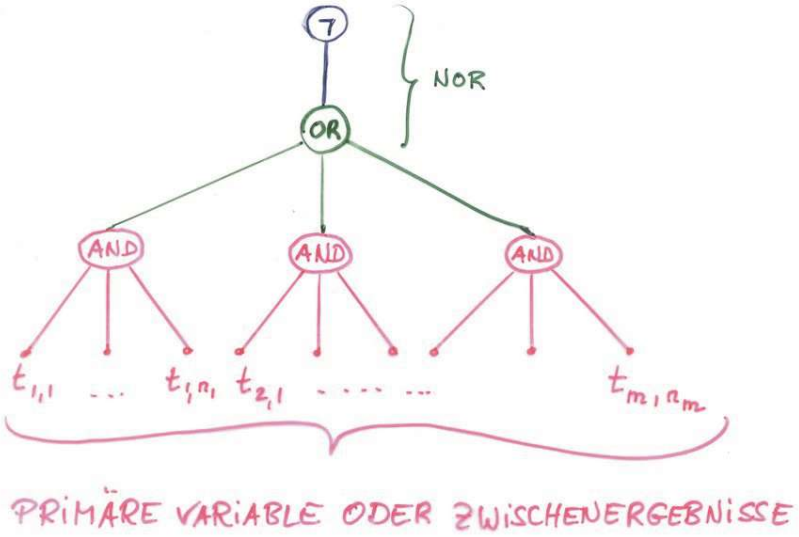
- LSS
- SOCRATES
- MIS, MIS-MV
- Synopsis
- ….

Reguläres Layout für mehrstufige Schaltungen

Beispiel: Weinberger Array (maskenprogrammiert)



Realisierbare Form Boolescher Ausdrücke



Synthese arithmetisch logischer Einheiten

Modulgeneratoren erzeugen Layouts für Boolesche Funktionen

- Irreguläre Funktionen ☞PLAs, Weinberger Arrays
- Reguläre Funktionen: effizienter! Beispiel: Addierer

EINGÄNGE			AUSGÄNGE		
x_i	y_i	c_i	c_{i+1}	z_i	–
0	0	0	0	0	$K_i = 1$ (KILL)
0	0	1	0	1	
0	1	0	0	1	$P_i = 1$ (PRO-PAGATE)
0	1	1	1	0	
1	0	0	0	1	
1	0	1	1	0	
1	1	0	1	0	$G_i = 1$ (GENERATE)
1	1	1	1	1	

$$K_i = \overline{x_i}\ \overline{y_i}$$

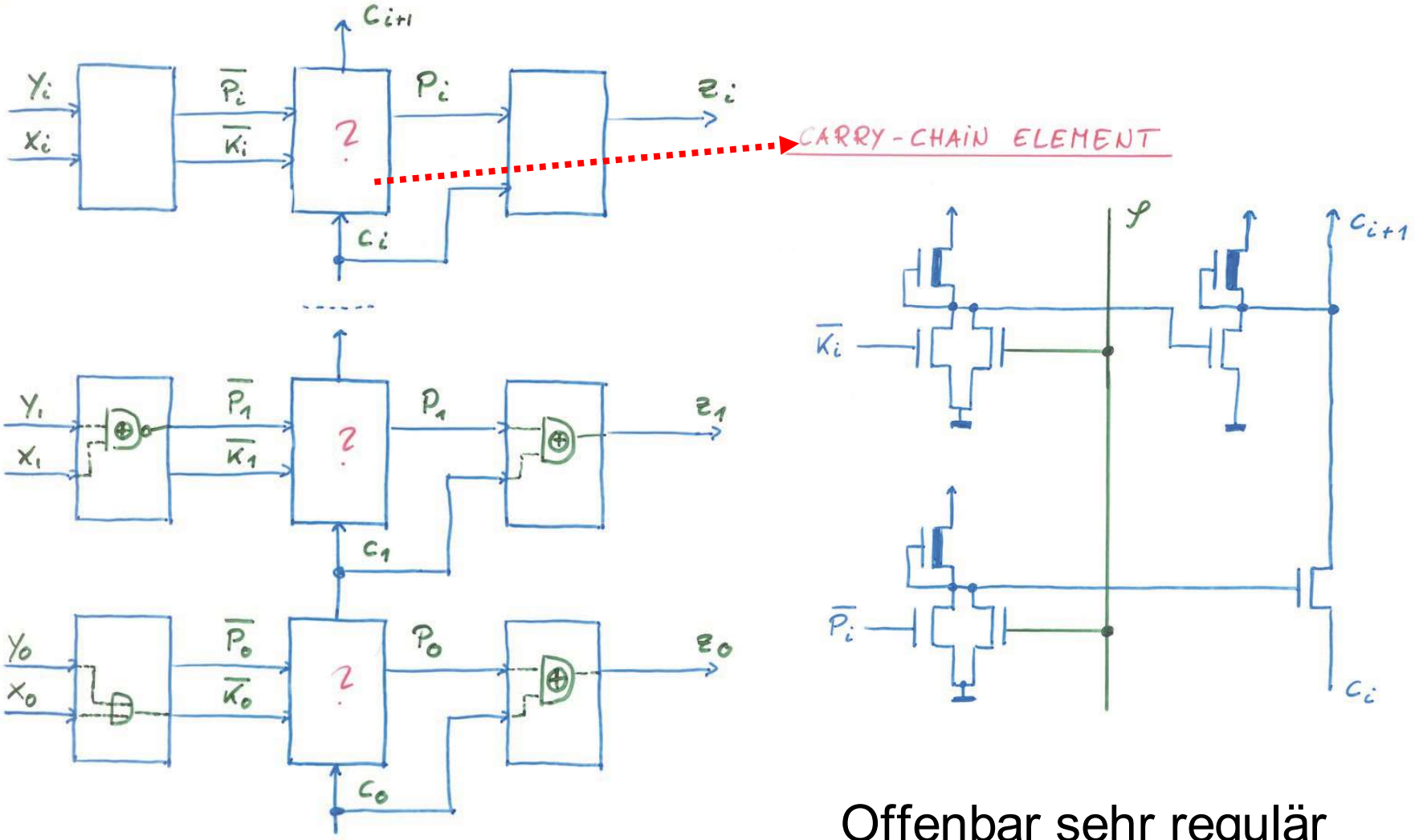
$$P_i = x_i \oplus y_i$$

$$G_i = x_i\, y_i$$

$$z_i = (c_i\, \overline{P_i}) \vee (\overline{c_i}\, P_i)$$

$$c_{i+1} = (c_i\, P_i) \vee G_i = (c_i\, P_i) \wedge \overline{K_i}$$

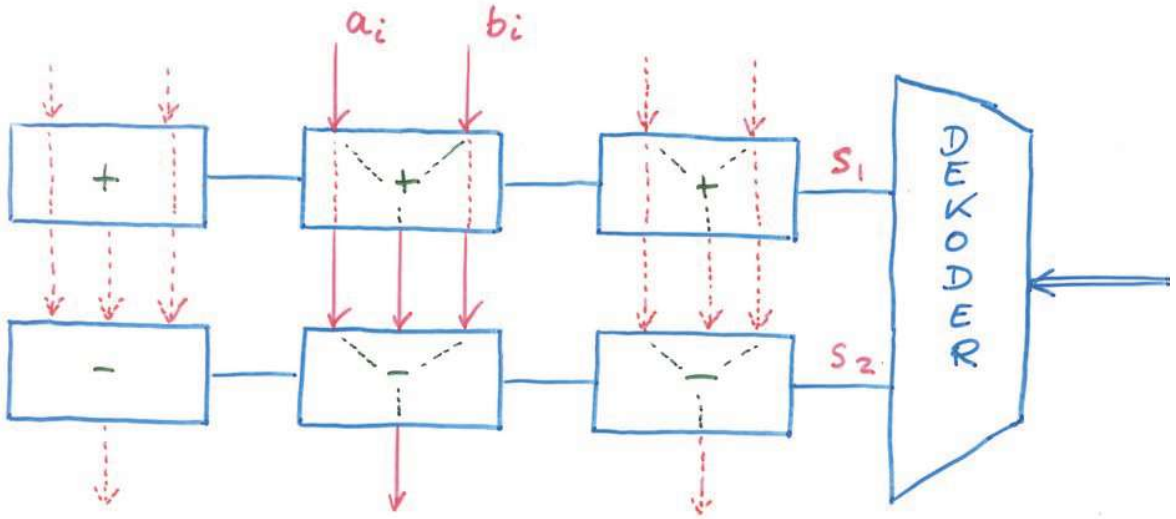
Blockschaltbild



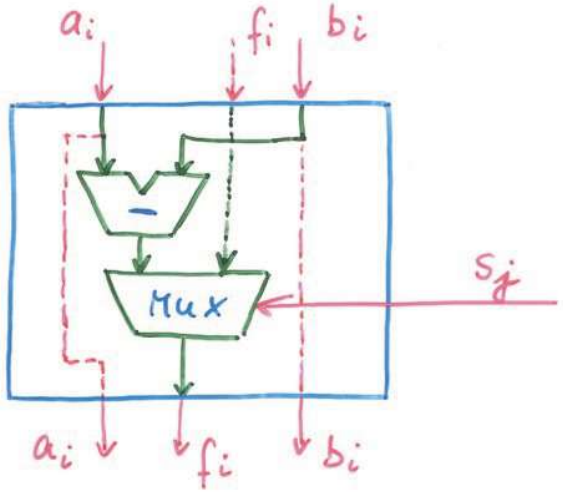
Offenbar sehr regulär

Multifunktionsbausteine

Auswahl der passenden Funktion über Zeilendekoder



Zu starke Vereinfachung: ähnliche Logik in unterschiedlichen Zeilen kann nicht genutzt werden.



Beschreibung der Funktion in CLASSY*

```
USE instr_format.ALL;
...
TYPE ctl : (add, sll, orleq);
...
ENTITY t4 IS
  PORT (a, b : IN word; c : IN bit;
        s : IN ctl; res : OUT word;
        ovr : OUT bit);
END t4;

ARCHITECTURE rt OF t4 IS
 BEGIN
  WITH s SELECT
    res <= (a + b) + c   WHEN add,
           a SLL 1       WHEN sll,
           (a OR b)      WHEN orleq;
  WITH s SELECT
    ovr <= overflow_add (a,b,c) WHEN add,
           a (a'left)           WHEN sll,
           a <= b               WHEN orleq;
END rt;
```

Ähnliche Logik bei verschiedenen Funktionen soll genutzt werden.

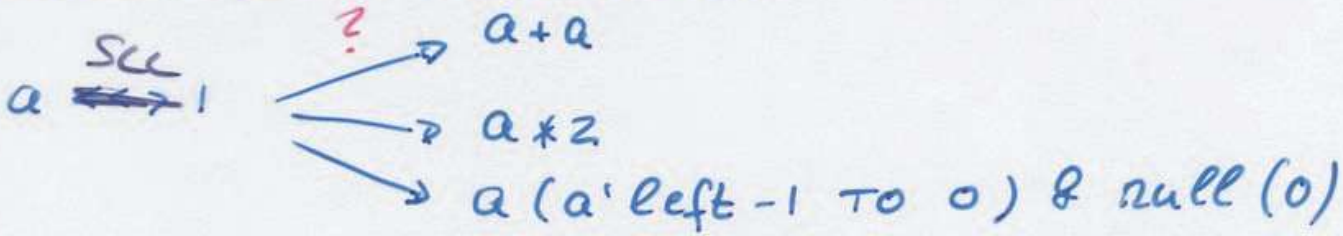
Symbolische Beschreibung des Kontrollcodes; erlaubt Einsatz symbolischer Minimierungstechniken.

* Lengauer et al., U Paderborn, ca. 1991

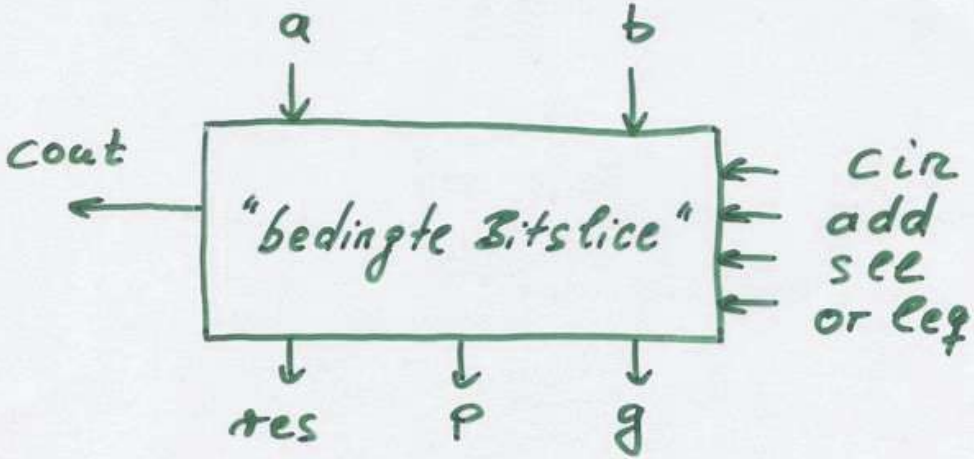
Ablauf der Synthese unter CLASSY

1. ALU-Zerlegung

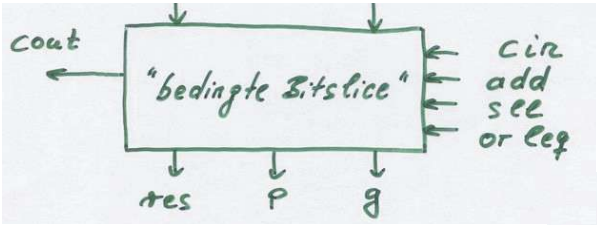
AUSWAHL EINER REALISIERUNG FÜR ARITHMETISCHE OPERATIONEN GEMÄß REGELSYSTEM



⇒ BRANCH- & BOUND - VERFAHREN



1. ALU-Zerlegung (2)



$res = /add, sll/ \quad cin \cdot \bar{p} \vee \overline{cin} \cdot p$

$/or\ leq/ \quad a \vee b$

$cout = /add, sll, or\ leq/ \quad g \vee p \cdot cin$

$p = /add/ \quad a\bar{b} \vee \bar{a}b$ ↓ für a+a

$/sll/ \quad 0 \qquad a\bar{a} \vee \bar{a}a$

$/or\ leq/ \quad ab \vee \bar{a}\bar{b} \qquad "="$

$g = /add/ \quad ab$ ↓ für a+a

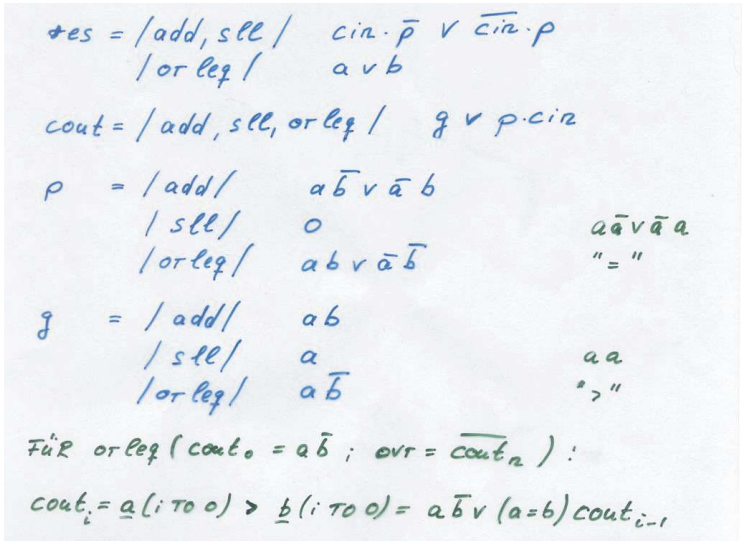
$/sll/ \quad a \qquad aa$

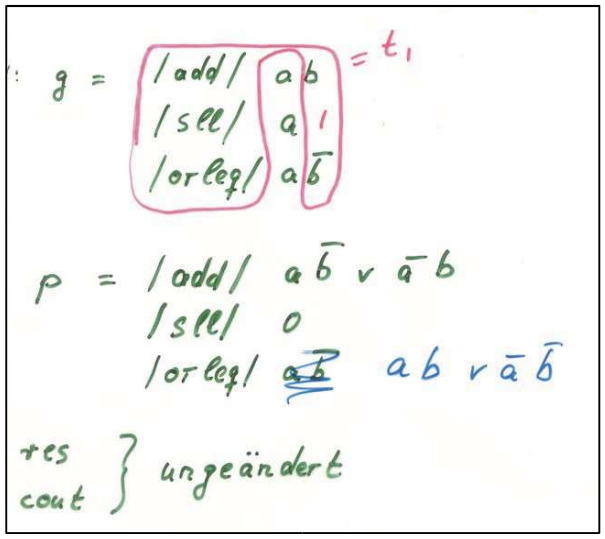
$/or\ leq/ \quad a\bar{b} \qquad ">"$

Für or leq ($cout_0 = a\bar{b}$; $ovr = \overline{cout_n}$) :

$cout_i = \underline{a}(i \text{ to } 0) > \underline{b}(i \text{ to } 0) = a\bar{b} \vee (a=b)\, cout_{i-1}$

2. Logik-Optimierung





☞ Mehrwertige symbolische Minimierung:

t_1 = /add/ b
/sll/ 1
/or leq/ $\bar{b}$

p = /add, or leq/ $a \cdot \bar{t}_1 \vee \bar{a} \cdot t_1$
/sll/ 0

g = /add, sll, or leq/ $a \vee t_1$

res = /add, sll/ $cin \cdot \bar{p} \vee \bar{cin} \cdot p$
/or leq/ $a \vee b$

cout = /add, sll, or leq/ $g \vee (p \cdot cin)$

Gleichzeitige Festlegung der Kodierung

t_1	=	/ add /	b
		/ sll /	1
		/ orleq /	$\bar{b}$
p	=	/ add, orleq /	$a\bar{t_1} \vee \bar{a} t_1$
		/ sll /	0
g	=	/ add, sll, orleq /	$a\, t_1$
res	=	/ add, sll /	$cin \cdot \bar{p} \vee \overline{cin} \cdot p$
		/ orleq /	$a \vee b$
cout	=	/ add, sll, orleq /	$g \vee (p \cdot cin)$

SYMBOLISCHE MINIMIERUNG

⇒ KODIERUNG DER STEUERSIGNALE

	cs_2	cs_1	cs_0
add	0	1	1
sll	1	0	1
orleq	1	1	0

$$t_1 = cs_0 \cdot b \vee cs_2 \cdot \bar{b}$$
$$= (add \vee sll)\, b \vee (sll \vee orleq)\, \bar{b}$$
$$= add \cdot b \vee sll \vee orleq \cdot \bar{b}$$

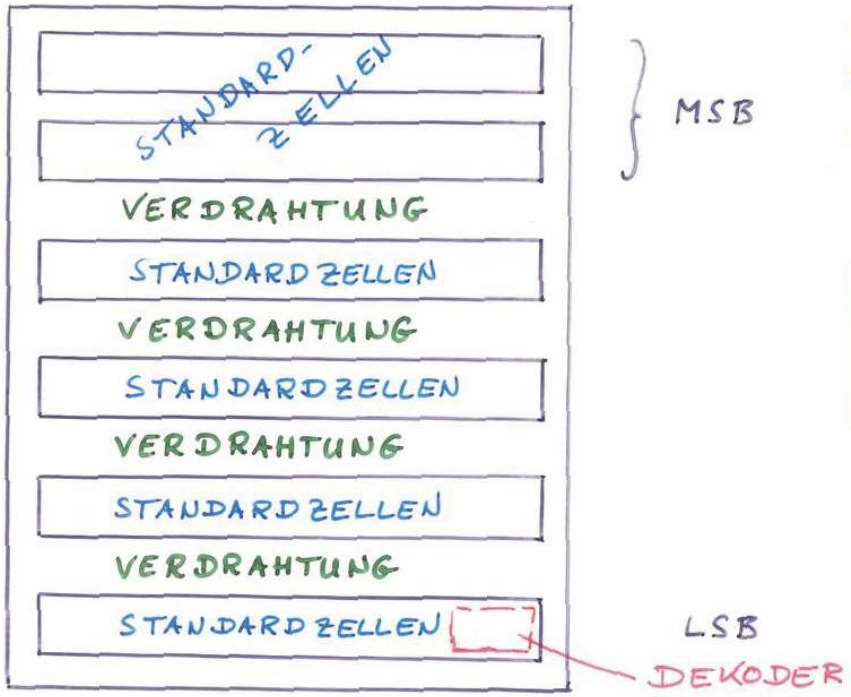
$$p = cs_1 \cdot (a \cdot \bar{t_1} \vee \bar{a} \cdot t_1)$$
$$= (add \vee orleq)(a\, \bar{t_1} \vee \bar{a}\, t_1)$$

$$res = cs_0 \cdot (cin \cdot \bar{p} \vee \overline{cin}\, p) \vee \overline{cs_0}\, (a \vee b)$$
$$= (add \vee sll)(cin \cdot \bar{p} \vee \overline{cin} \cdot p) \vee orleq\, (a \vee b)$$

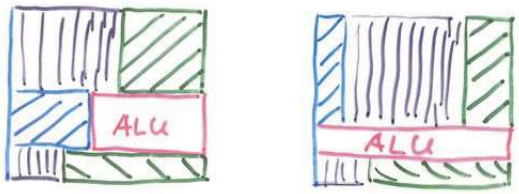
$$cout = a \cdot t_1 \vee p \cdot cin$$

Technology binding & Layout Synthese

1. Abbildung auf kommerzielle Bausteinbibliothek
2. Anordnung und Verdrahtung der Bausteine (Standardzellen)



ERZEUGUNG VON ALTERNATIVEN MIT UNTERSCHIEDLICHEM LÄNGEN/BREITEN VERHÄLTNIS (ENGL. "ASPECT RATIO")



AUSWAHL EINER ALTERNATIVE IM FLOORPLANNIG (= PLAZIERUNG + WAHL EINES "ASPECT RATIOS")

Zusammenfassung

- Layout für irreguläre mehrstufige Realisierungen
- Layout für reguläre mehrstufige Realisierungen
  - einfache *Manchester-carry-chains*
  - CLASSY