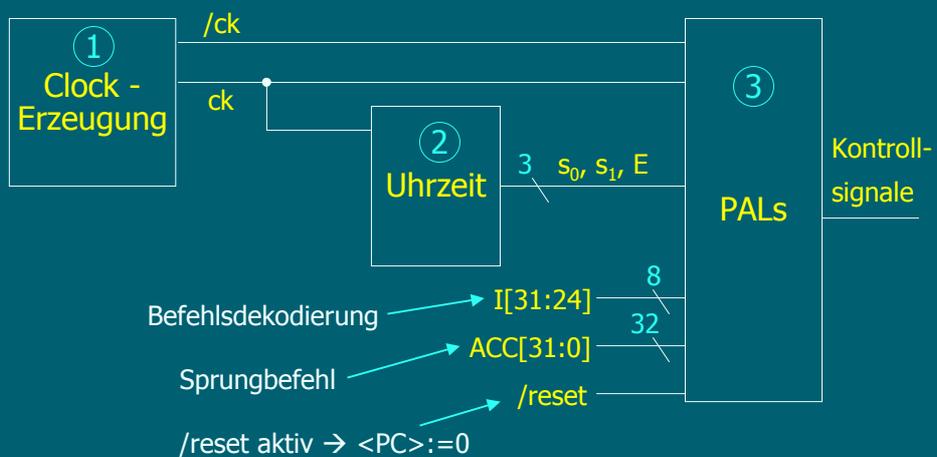


## 14.3 Kontrolllogik

Bernd Becker – Technische Informatik II

### Allgemeines:

Kontrolllogik wird in 3 Stufen realisiert:



## 14.3.1 Clock- und Phasensignale

BB - TI II 1.3/3

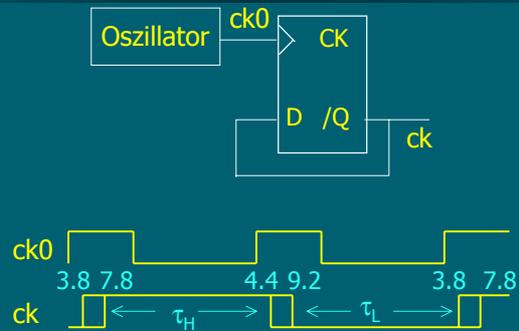
### zu 1.:

Clocks werden durch Oszillatoren generiert  
(aber evtl. nicht symmetrisch!)

→ Erzeugung einer (nahezu) symmetrischen Clock  
durch nachfolgendes D-FF  
(vgl. folgende Skizze)

BB - TI II 1.3/4

## Skizze



$\tau_H$  : Zeit, in der ck high

$\tau_L$  : Zeit, in der ck low

ck symmetrisch, wenn  $\tau_H = \tau_L$

Maximale Abweichung?

BB - TI II 1.3/5

## Berechnung zur Skizze



$$\tau_H = (P + 9.2) - 3.8$$

$$\tau_L = (P + 3.8) - 9.2$$

$$|\tau_H - \tau_L| = 2 (9.2 - 3.8)$$

Maximale Abweichung:

$$|\tau_H - \tau_L| \leq 2 (9.2 - 3.8) = 10.8 \text{ ns}$$

In der Realität ist  $|\tau_H - \tau_L|$  wesentlich kleiner.

Vereinfachende Annahme für die Timing-Analyse:

Clock ist symmetrisch!!!

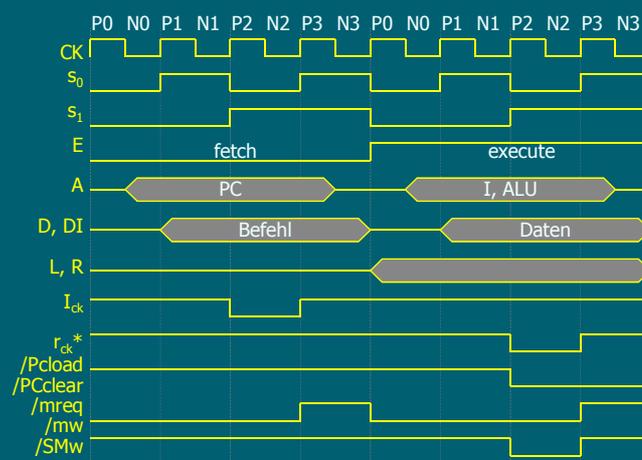
BB - TI II 1.3/6

## zu 2. und 3.:

2.  $s_0, s_1, E \cong 3$  niederwertigste Bits eines Zählers 74F163
3. Verwendete PALs: 20R8  
20L8

BB - TI II 1.3/7

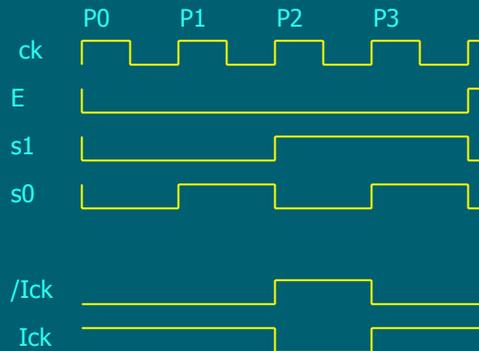
## 14.3.2 Clocksignale



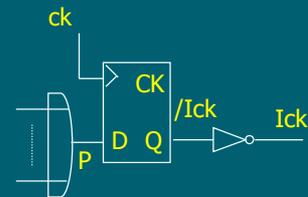
(\* r = PC, IN1, IN2, ACC)

BB - TI II 1.3/8

## Verlauf von Ick



a)



b)

## Ick

Da Ick active high, schreibe Gleichung für /Ick:

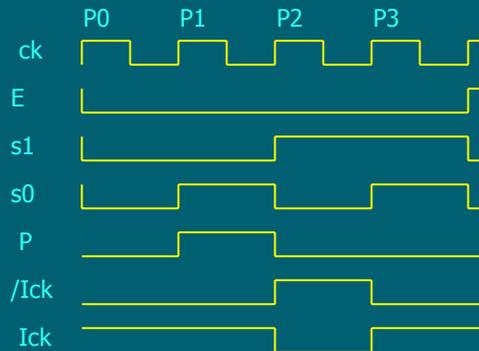
$/Ick := P$  mit P Polynom.

(Verlauf siehe folgende Skizze)

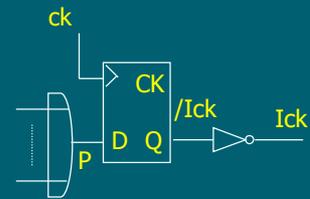
Polynome werden für **Ausgänge der OR-Gatter** geschrieben!

Um bei P2 eine 1 ins FF zu bringen, muss diese schon im Takt vorher am Eingang anliegen!

## Verlauf von Ick



a)



b)

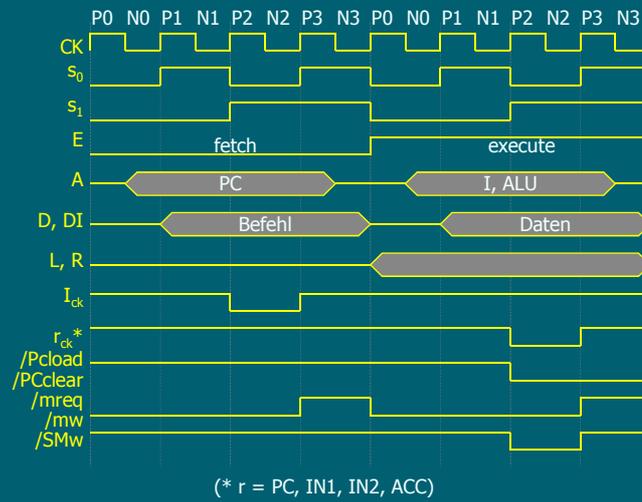
$$P = /E * s0 * /s1$$

## Ick (ff)

Polynom P muss also Signal erzeugen,  
das /Ick um 1 Takt vorseilt!!

$$\rightarrow /Ick := /E * /s1 * s0$$

## PCck



BB - TI II 1.3/13

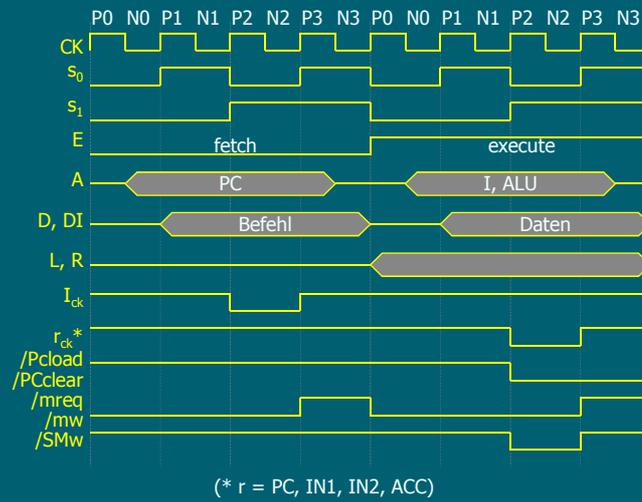
## PCck

PC gecltkt immer bei P2 von execute (E)

→  $/PCck := E * /s1 * s0$

BB - TI II 1.3/14

## ACCck



BB - TI II 1.3/15

## ACCck

ACC geclockt während execute für

- Compute mit D = ACC
- Load mit D = ACC
- MOVE mit D = ACC

BB - TI II 1.3/16

## ACCck (ff)

Compute: /I31 \* /I30

Load: /I31 \* I30

MOVE: I31 \* /I30 \* I29 \* I28

D = ACC: I25 \* I24

Kodierung	Register
0 0	PC
0 1	IN1
1 0	IN2
1 1	ACC

BB - TI II 1.3/17

## ACCck (ff)

/ ACCck := E \* /s1 \* s0 ; Start bei P2 von execute

\* /I31 \* /I30 ; compute

\* I25 \* I24 ; D = ACC

+ E \* /s1 \* s0

\* /I31 \* I30 ; Load

\* I25 \* I24

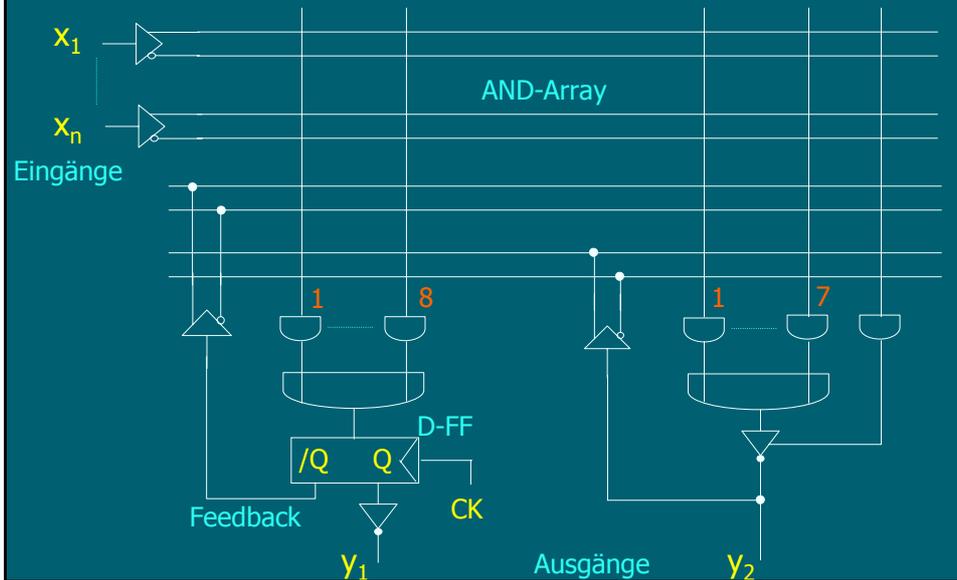
+ E \* /s1 \* s0

\* I31 \* /I30 \* I29 \* I28 ; MOVE

\* I25 \* I24

BB - TI II 1.3/18

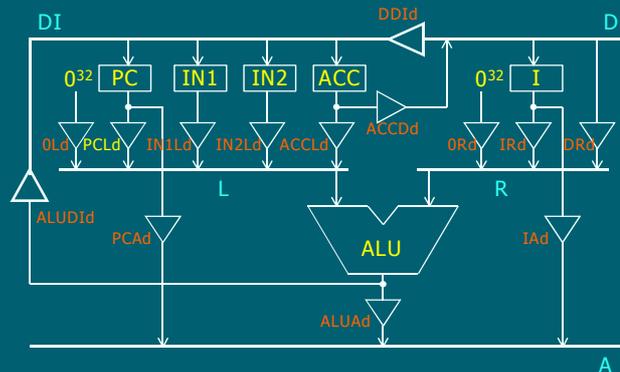
## Schaltbild eines PAL



## Clock für IN1ck, IN2ck:

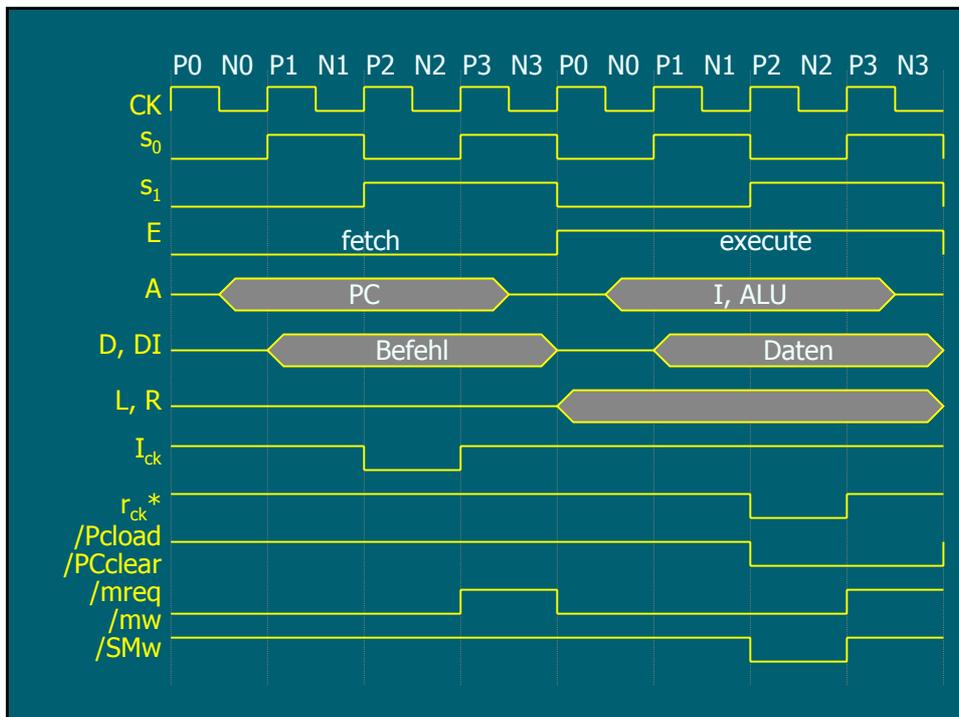
Analoge Gleichungen für IN1ck, IN2ck !

### 14.3.3 Output enable - Signale



Beispiel:  $/PCLd\bar{o}e$

BB - TI II 1.3/21



## /PCLdœ

enabled bei P0 von execute für Befehle

- JUMP
- Compute mit D = PC
- MOVE mit S = PC

BB - T1 II 1.3/23

## Beispiel (ff)

```
PCLdœ := /E * s1 * s0 * I31 * I30 ; Start bei P0 von Jump
+ /E * s1 * s0
  * /I31 * /I30 ; Compute
  * /I25 * /I24 ; D = PC
+ /E * s1 * s0
  * I31 * /I30 * I29 * I28 ; MOVE
  * /I27 * /I26 ; S = PC
+ ... ★
```

BB - T1 II 1.3/24

## Problem:

Aktivierung aber für 4 Takte!!

Bei analogem Vorgehen also  $4 \cdot 3$  Monome nötig.

(PAL stellt nur 8 zur Verfügung.)

Ausweg: Nutze Rückkopplung! → *Halteterme*

## Beispiel (ff)

- ★ +  $PCLd\phi * E * /s1 * /s0$  ; bei P1 halten
- +  $PCLd\phi * E * /s1 * s0$  ; bei P2 halten
- +  $PCLd\phi * E * s1 * /s0$  ; bei P3 halten

Analog: andere Output enable – Signale

→ **siehe Übung**



## Aufgaben:

1. Funktions-Select-Signale  $f[2:0]$  der ALU korrekt ansteuern
2.  $c_{in}$  setzen
3. `sext` und `fill` berechnen

Alle Signale werden mit kombinatorischen PALs berechnet.

BB - TI II 1.3/29

## zu 1.:

Kodierung der Compute-Befehle ist so gewählt, dass  $I[28:26]$  gerade die Belegung für  $f[2:0]$  liefert.

$$\rightarrow f[2:0] = \begin{cases} I[28:26] & \text{falls typ = compute} \\ 011 & \text{d.h. Addition sonst} \end{cases}$$

PAL-Gleichung **siehe Übung**

(  $IF(VCC) / f0 = \dots$  etc. )

BB - TI II 1.3/30

## zu 2.:

$c_{in} = 1$  bei Subtraktion

→ Übung

( IF (VCC) /cin = ... etc. )

## zu 3.:

keine Sign-Extension bei

- Compute immediate mit log. Operationen
- LOADI

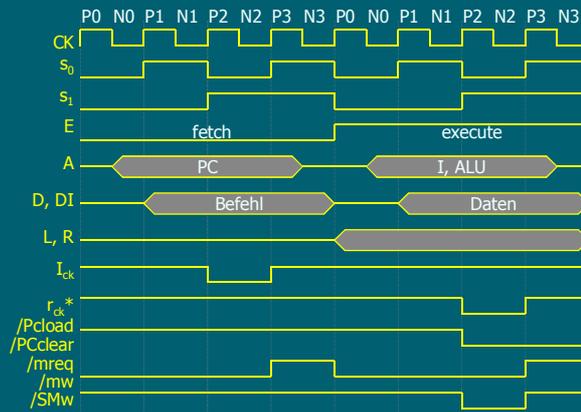
→ Übung

( /sext = ... )

$fill = sext * I[23]$  , d.h.

$/fill = /sext + /I[23]$

## 14.3.5 Laden des Befehlszählers



BB - TI II 1.3/33

## Aktivierung, Deaktivierung

/Pcload wird aktiviert bei P2 von Execute,  
deaktiviert bei P2 von Fetch,  
bei

- Compute mit D = PC
  - Load mit D = PC
  - MOVE mit D = PC
  - JUMP mit erfüllter Sprungbedingung
- } Bedingung analog /ACCck

BB - TI II 1.3/34

## JUMP

JUMP wird ausgeführt  
abhängig von C-Feld  
(I[29:27]) und ACC[31:0] ,  
genauer wenn

- ( I[29] = 1 und [ACC] < 0 ) oder
- ( I[28] = 1 und [ACC] = 0 ) oder
- ( I[27] = 1 und [ACC] > 0 )
- [ACC] < 0  $\Leftrightarrow$  ACC[31] = 1
- [ACC] = 0  $\Leftrightarrow$  /ACC[31] \* ... \* /ACC[0] = 1

### Kodierung nach Schema:

nur I [29] = 1  $\Leftrightarrow$  < wird abgefragt  
nur I [28] = 1  $\Leftrightarrow$  = wird abgefragt  
nur I [27] = 1  $\Leftrightarrow$  > wird abgefragt

Andere Abfragen durch Kombinationen, z.B.  
C = 101 : < oder > , also  $\neq$

BB - TI II 1.3/35

## Problem

bei Realisierung mit PAL 20L8 bzw. 20R8!

Lösung: Aufteilung in Gleichung für ZH und ZL und  
Berechnung von /ZH und /ZL in getrennten  
kombinatorischen PALs

(IF(VCC)) ZH := /ACC31 \* ... \* /ACC16

(IF(VCC)) ZL := /ACC15 \* ... \* /ACC0

[ACC] = 0  $\Leftrightarrow$  ZL \* ZH = 1

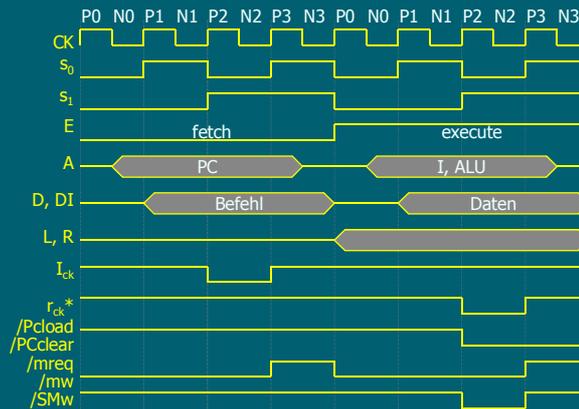
[ACC] > 0  $\Leftrightarrow$  ACC[31] = 0  $\wedge$  (/ZL = 1  $\vee$  /ZH = 1)

Zusammenbauen  $\rightarrow$  PAL-Gleichung für /PCload

BB - TI II 1.3/36

## 14.3.6 Reset

/PCclear sorgt für  $\langle PC \rangle := 0$



BB - TI II 1.3/37

## Reset

- /PCclear sorgt für  $\langle PC \rangle := 0$ , d.h. Reset der Maschine.  
(aktiviert bei P2 von Execute, deaktiviert bei P0 von Fetch)
- An die Maschine wird ein Reset-Request gestellt,
  - wenn der Strom eingeschaltet wird (power up reset)
  - ein Reset-Knopf gedrückt wird

BB - TI II 1.3/38

## Reset (ff)

Sei eine physikalische Schaltung gegeben,  
die einen Reset-Request in einem Flipflop  
mit Ausgang /resetreq abspeichert,  
d.h. bei einem Reset-Request dafür sorgt,  
dass für mind. 1 Takt lang /resetreq = 0 wird.

## Reset (ff)

/resetreq aktiviert ein Signal /reset,  
das bis P2 von Execute gehalten wird:

reset := resetreq

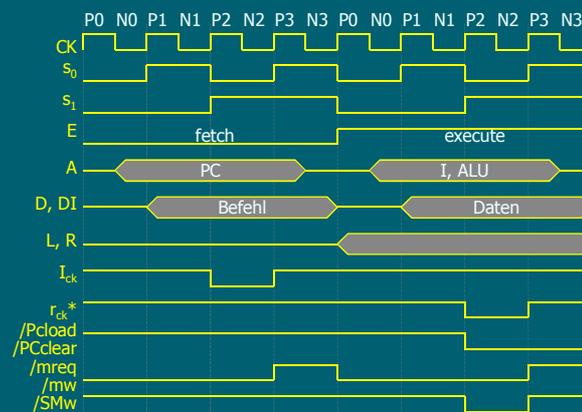
+ reset \* E \* s1 ; halten an P3 von E, P0 von F  
+ reset \* /E ; halten an P1,P2,P3 von F,  
P0 von E  
+ reset \* E \* /s1 \* /s0 ; halten an P1 von E

## Reset (ff)

$PC_{clear} := E * /s_1 * s_0$  ; Start bei P2 von E  
 \* reset  
 $+ PC_{clear} * E * s_1 * /s_0$  ; halten an P3 von E

BB - TI II 1.3/41

## 14.3.7 Speichermanagement



BB - TI II 1.3/42

## Speicheransteuerung

Speicher wird gebraucht bei

- Fetch
- Compute Memory
- LOAD, LOADINj
- STORE, STOREINj

→ /mreq läßt sich mit 8 Termen realisieren

→ Übung

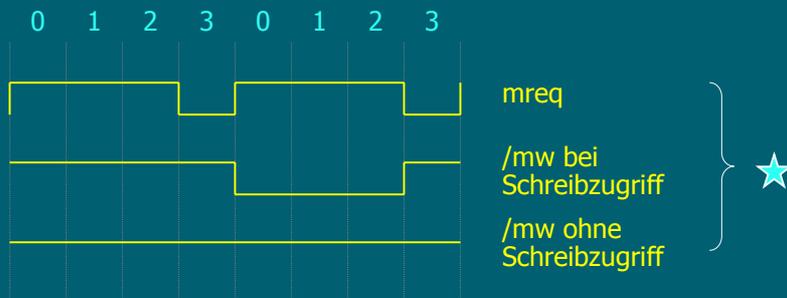
BB - TI II 1.3/43

## Weiteres zur Speicheransteuerung

- PAL-Gleichung für /mw durch Streichen der Terme für Fetch, Compute, Load
- Output enable für SMDd
- SMw

BB - TI II 1.3/44

## Speicheransteuerung (ff)



★ → Und-Verknüpfung + 1 Takt verschoben!

## Speicheransteuerung (ff)

- Output enable für SMDd:  
 $SMD\phi_e := mreq * /mw$
- $SMw := mw$  ; Schreiben  
 $* E * /s1 * s0$  ; Start bei P2 von E