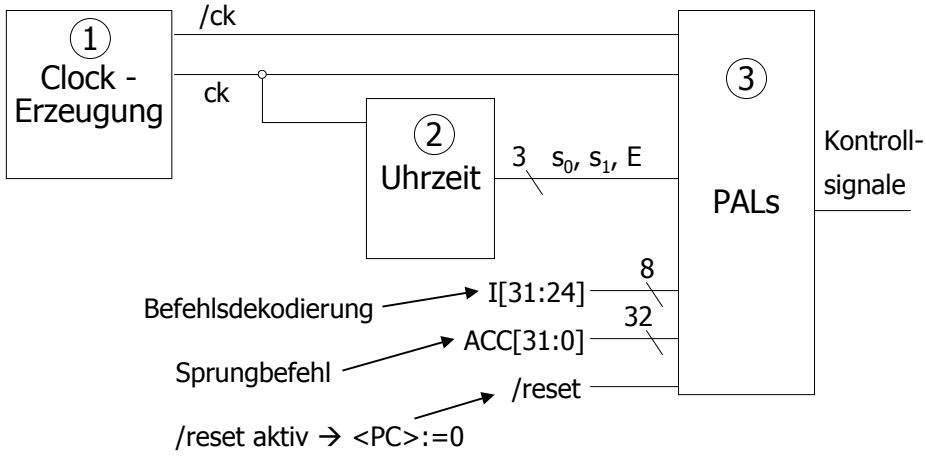
14.3 Kontrollogik

Allgemeines:

Kontrollogik wird in 3 Stufen realisiert:



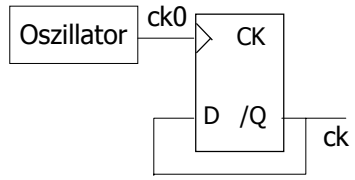
14.3.1 Clock- und Phasensignale

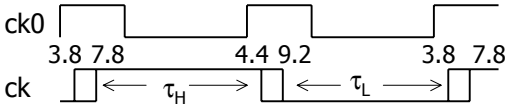
zu 1.:

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

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

Skizze





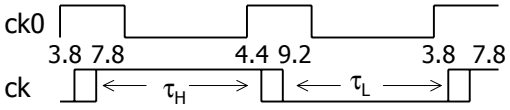
τ_H : Zeit, in der ck high

τ_L : Zeit, in der ck low

ck symmetrisch, wenn $\tau_H = \tau_L$

Maximale Abweichung?

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:

$| \tau_H – \tau_L | \leq 2 (9.2 – 3.8) = 10.8$ ns

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

Vereinfachende Annahme für die Timing-Analyse:

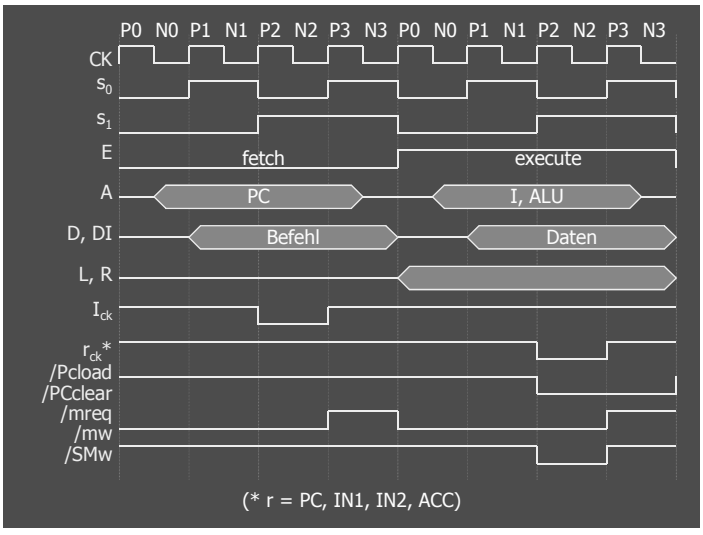
Clock ist symmetrisch!!!

zu 2. und 3.:

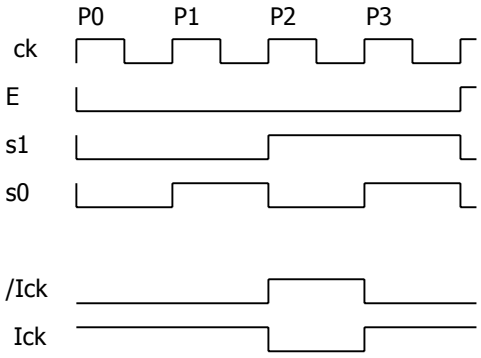
2. s0, s1, E ≅ 3 niederwertigste Bits
   eines Zählers 74F163

3. Verwendete PALs: 20R8
   20L8

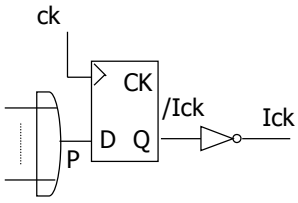
14.3.2 Clocksignale



Verlauf von Ick



a)

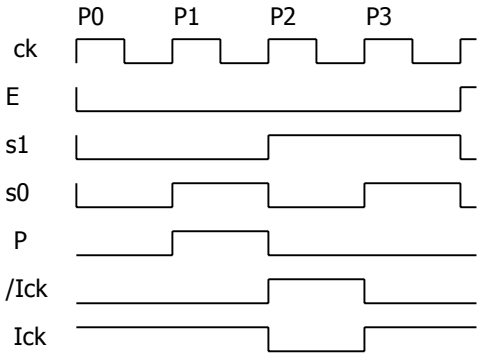


b)

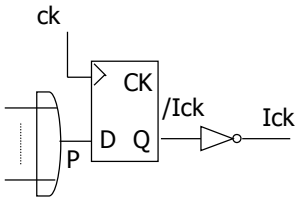
Ick

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)

$P = /E * s0 * /s1$

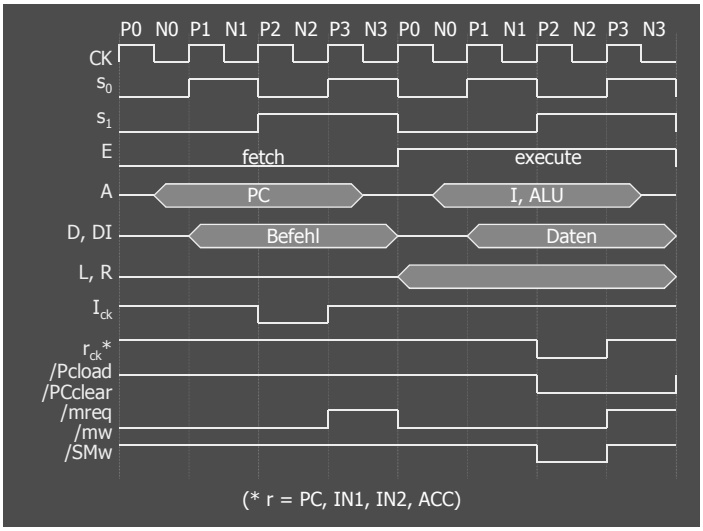
Ick (ff)

Polynom P muss also Signal erzeugen,

das /Ick um 1 Takt vorauseilt!!

→ $/Ick := /E * /s1 * s0$

PCck

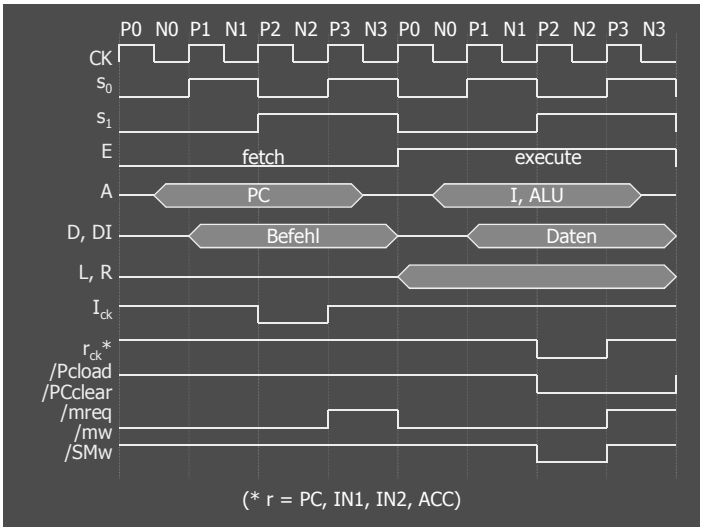


PCck

PC geclockt immer bei P2 von execute (E)

→ /PCck := E ∗ /s1 ∗ s0

ACCck



ACCck

ACC geclockt während execute für

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

ACCck (ff)

Compute: /I31 * /I30

Load: /I31 * I30

MOVE: I31 * /I30 * I29 * I28

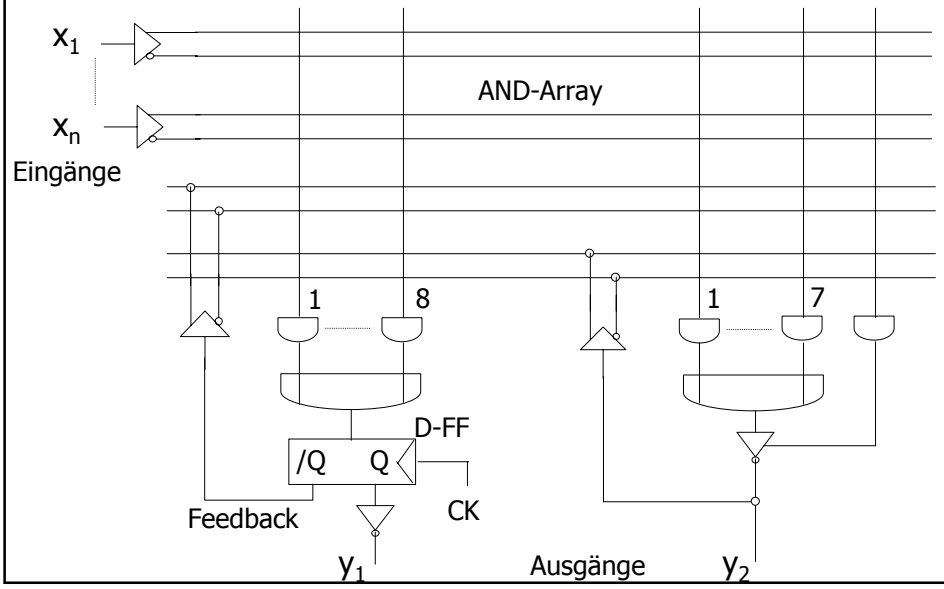
D = ACC: I25 * I24

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

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
```

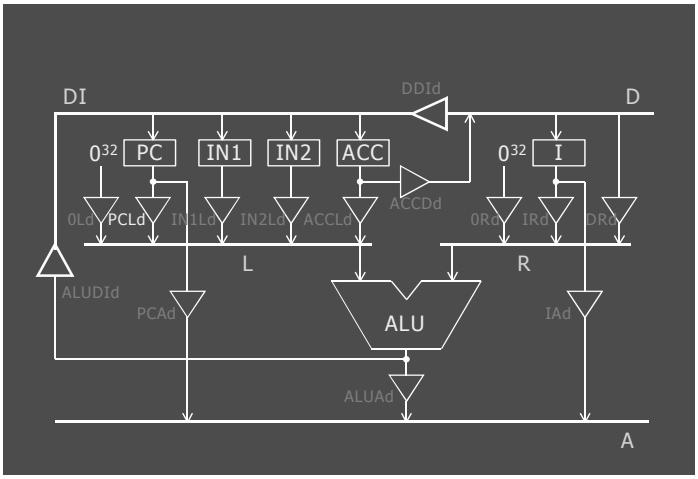
Schaltbild eines PAL



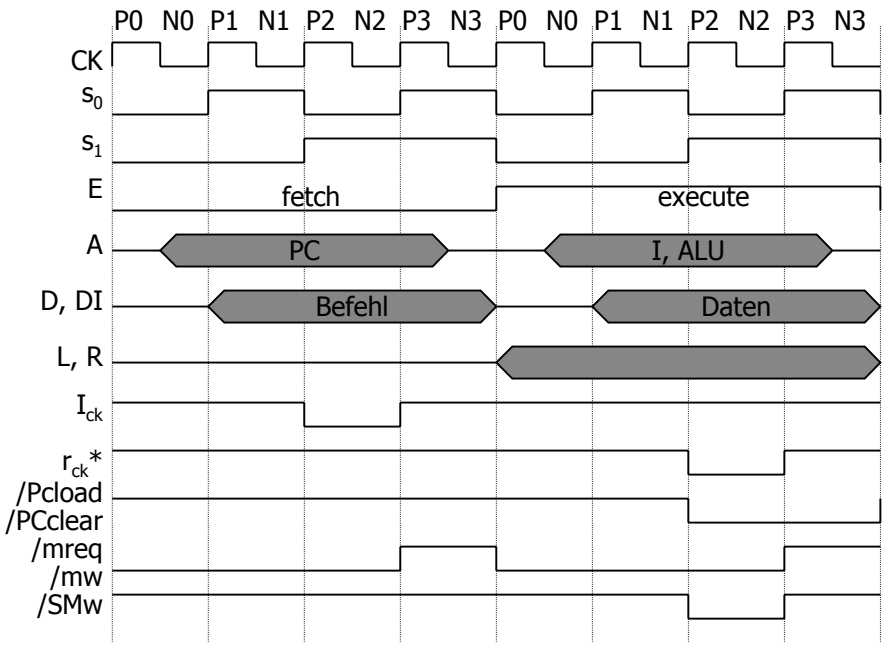
Clock für IN1ck, IN2ck:

Analoge Gleichungen für IN1ck, IN2ck !

14.3.3 Output enable - Signale



Beispiel: /PCLdœ



/PCLdœ

enabled bei P0 von execute für Befehle

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

Beispiel (ff)

```
PCLdœ  := /E * s1 * s0 * I31 * I30        ; Start bei P0 von Jump
        +  /E * s1 * s0
           * /I31 * /I30                  ; Compute
           * /I25 * /24                   ; D = PC
        +  /E * s1 * s0
           * I31 * /I30 * I29 * I28       ; MOVE
           * /I27 * /I26                  ; S = PC
        +  ...
```

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)

☆ + PCLdœ ∗ E ∗ /s1 ∗ /s0 ; bei P1 halten

+ PCLdœ ∗ E ∗ /s1 ∗ s0 ; bei P2 halten

+ PCLdœ ∗ E ∗ s1 ∗ / s0 ; bei P3 halten

Analog: andere Output enable – Signale

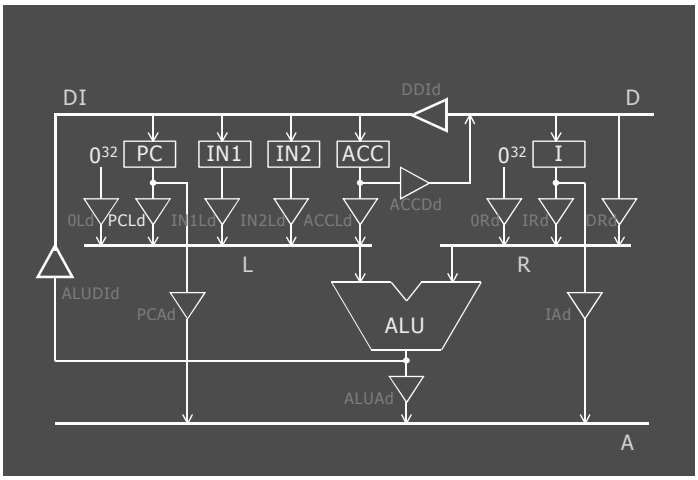
→ siehe Übung

Sonderbehandlung:

Treiber auf Adressbus werden enabled bei N0, disabled bei N3

→ Berechnung in PALs,
die mit /ck getaktet werden!

14.3.4 Kontrolle der ALU und Sign-Extension



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.

zu 1.:

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

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

PAL-Gleichung siehe Übung

(IF(VCC) /f0 = ... etc.)

zu 2.:

c_{in} = 1 bei Subtraktion

→ Übung

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

zu 3.:

keine Sign-Extension bei

- Compute immediate mit log. Operationen
- LOADI

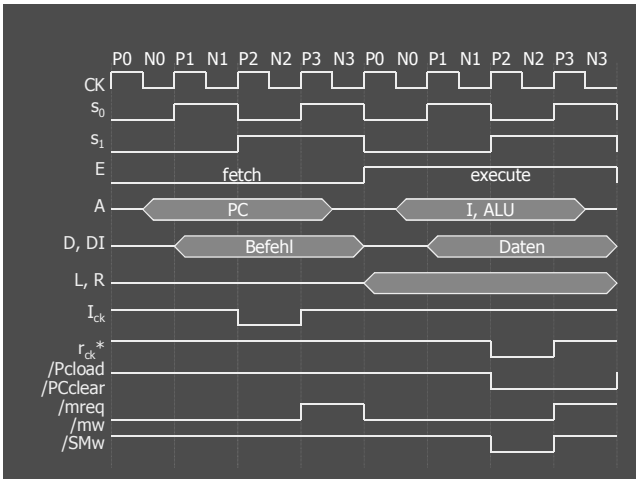
→ Übung

(/sext = ...)

fill = sext ∗ I[23] , d.h.

/ fill = / sext + / I[23]

14.3.5 Laden des Befehlszählers



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

} Bedingung analog /ACCck

- JUMP mit erfüllter Sprungbedingung

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 ⇔ ACC[31] = 1
- [ACC] = 0 ⇔ /ACC[31] * ... * /ACC[0] = 1

Kodierung nach Schema:

nur I [29] = 1 ⇔ < wird abgefragt
nur I [28] = 1 ⇔ = wird abgefragt
nur I [27] = 1 ⇔ > wird abgefragt

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

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

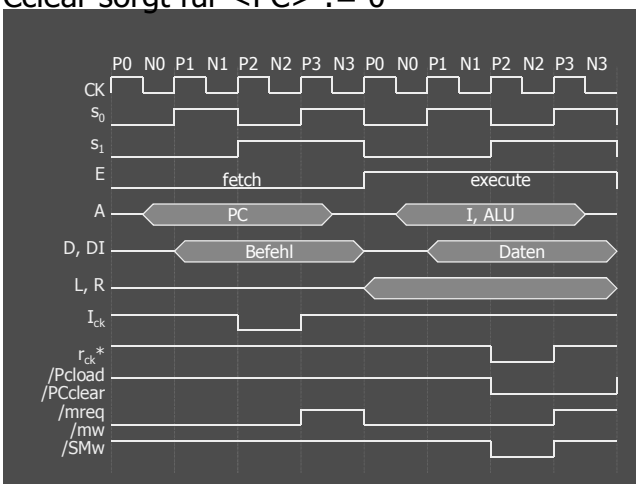
[ACC] = 0 ⇔ ZL * ZH = 1

[ACC] > 0 ⇔ ACC[31] = 0 ∧ (/ZL = 1 ∨ /ZH = 1)

Zusammenbauen → PAL-Gleichung für /PCload

14.3.6 Reset

/PCclear sorgt für <PC> := 0



Reset

- /PCclear sorgt für <PC> := 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

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

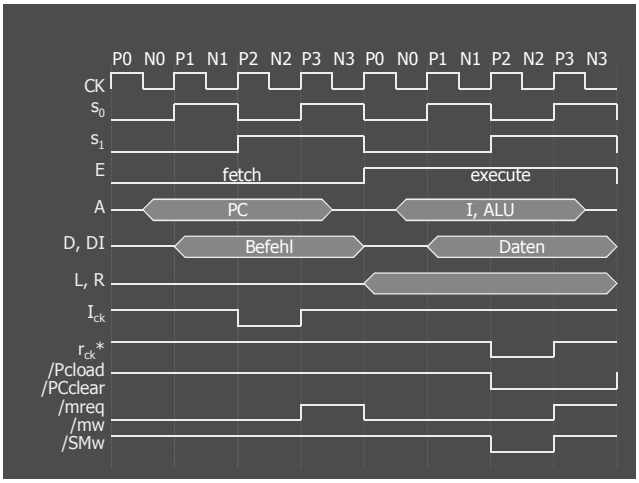
Reset (ff)

PCclear := E * /s1 * s0 ; Start bei P2 von E

* reset

+ PCclear * E * s1 * /s0 ; halten an P3 von E

14.3.7 Speicheransteuerung



Speicheransteuerung

Speicher wird gebraucht bei

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

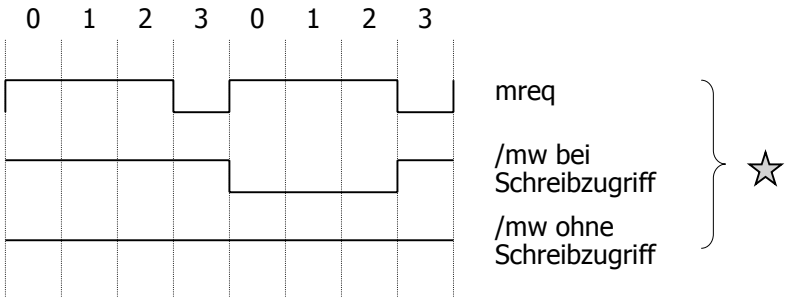
→ /mreq läßt sich mit 8 Termen realisieren

→ Übung

Weiteres zur Speicheransteuerung

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

Speicheransteuerung (ff)



☆ → Und-Verknüpfung + 1 Takt verschoben!

Speicheransteuerung (ff)

- Output enable für SMDd:

  SMDoe := mreq * /mw

- SMw := mw ; Schreiben

  * E * /s1 * s0 ; Start bei P2 von E