

## 15.2 Busprotokolle, Zustandsdiagramme

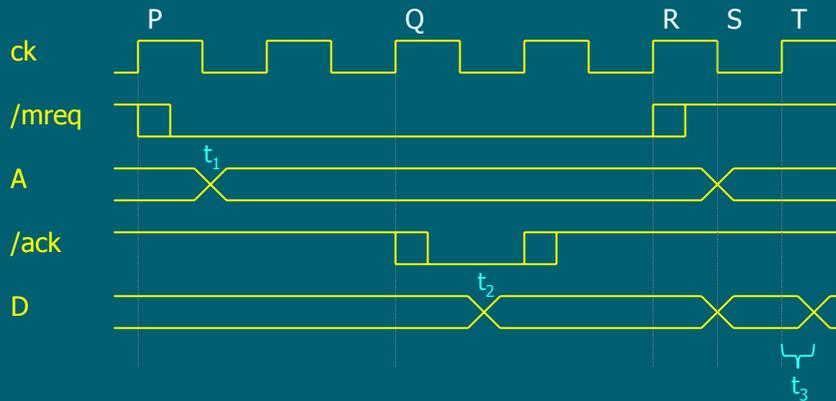
Bernd Becker – Technische Informatik II

### 15.2.1 Busprotokolle

Kommunikation zwischen Kontrolleinheit der CPU und den Kontrolleinheiten der Speicher erfolgt über ein festes **Busprotokoll** mit Hilfe der Signale `/mreq`, `/mw`, `/ack`

BB - TI II 15.2/2

## Lesezugriff hier in 3 Stufen



a) Lesen

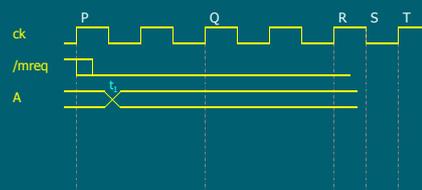
## Lesezugriff hier in 3 Stufen (ff)

1. Lesezugriff von CPU wird initiiert durch Aktivieren von `/mreq` an steigender Flanke P von `ck` (`/mw` inaktiv).

Nach Zeit  $t_1$  (nach P) garantiert CPU

gültige Adressen auf A.

### Lesezugriff hier in 3 Stufen

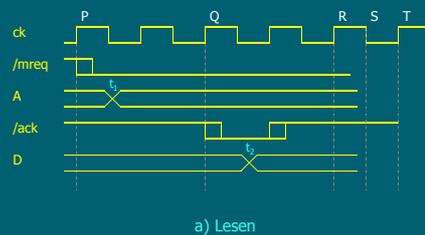


a) Lesen

## Lesezugriff hier in 3 Stufen (ff)

- Speicher aktiviert **Acknowledge-Signal /ack** für genau einen Takt an steigender Flanke Q. Nach Verzögerungszeit  $t_2$  nach Q garantiert Speicher korrekte Daten auf D.

Lesezugriff hier in 3 Stufen

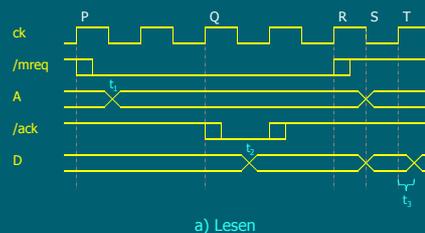


a) Lesen

## Lesezugriff hier in 3 Stufen (ff)

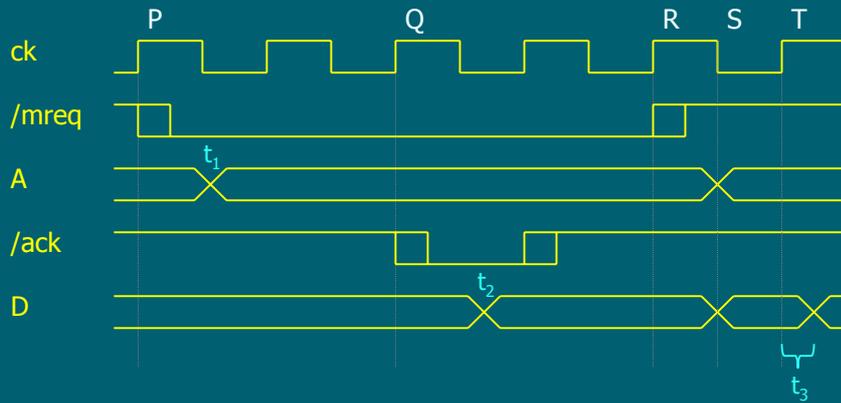
- CPU deaktiviert /mreq an steigender Flanke R. Speicher garantiert gültige Daten bis fallende Flanke S nach R und garantiert Disablen der Treiber auf D nach Zeit  $t_3$  nach nachfolgender steigender Flanke T

Lesezugriff hier in 3 Stufen



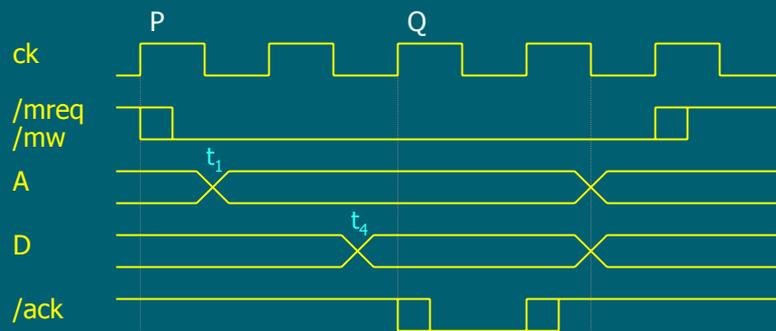
a) Lesen

## Lesezugriff hier in 3 Stufen



a) Lesen

## Schreibzugriff in 2 Stufen



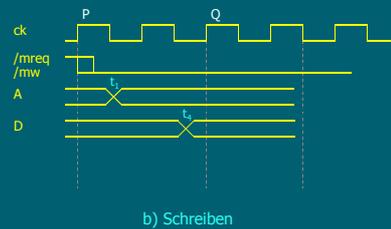
b) Schreiben

## Schreibzugriff in 2 Stufen (ff)

1. CPU aktiviert `/mreq` und `/mw` an steigender Flanke P von ck.

Nach  $t_1$  (nach P) gültige Adressen auf A garantiert,  
nach  $t_4$  auch gültige Daten auf D

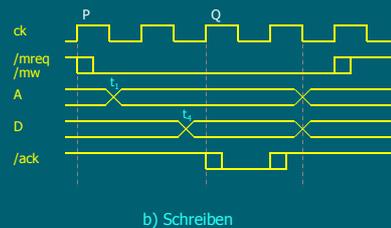
### Schreibzugriff in 2 Stufen



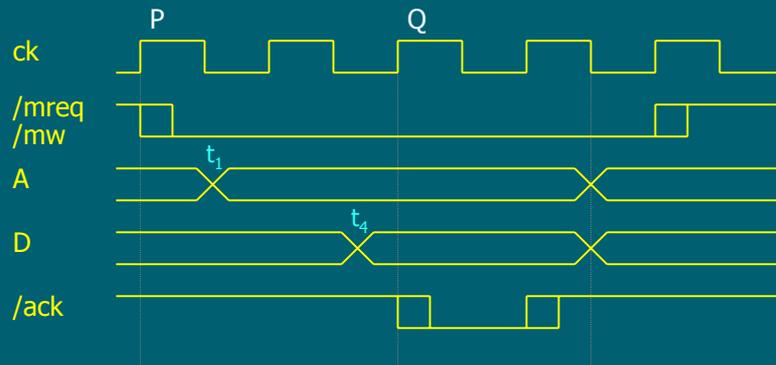
## Schreibzugriff in 2 Stufen (ff)

2. Speicher aktiviert `/ack` an Q für genau einen Takt.  
CPU garantiert stabile Adressen und Daten noch bis zur fallenden Flanke nach Deaktivieren von `/ack`.  
`/mreq` und `/mw` bleiben aktiv mindestens bis steigende Flanke nach Q.

### Schreibzugriff in 2 Stufen



## Schreibzugriff in 2 Stufen



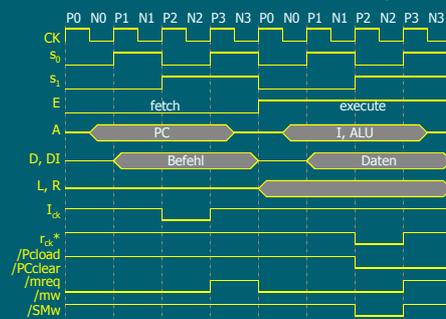
b) Schreiben

## Idealisiertes Timing mit Wait-Zyklen

Timing von /mreq und /mw wird leicht abgeändert:

**Bisher:**

Aktivieren bei P0,  
Deaktivieren bei P3.



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

## Idealisiertes Timing mit Wait-Zyklen

### Jetzt:

Fetch und Compute memory bleiben wie bisher;  
bei Load und Store (außer LOADI, MOVE) wird  
Aktivierungszeitpunkt auf P1 verschoben,  
um Zeit für Adressrechnung zu haben.

BB - TI II 15.2/13

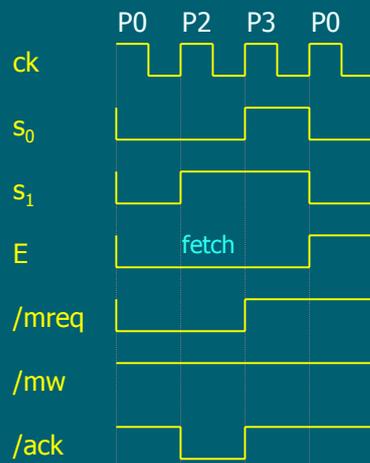
## Zunächst SRAM (schneller Speicher)

Kontrolle von SRAM wird so entworfen, dass  
`/ack` genau einen Takt nach `/mreq` aktiv wird.  
CPU-Kontrolle soll dann keine Wait-Zyklen erzeugen.

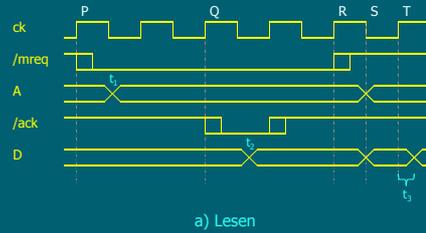
(Vgl. folgende Abbildungen)

BB - TI II 15.2/14

## Fetch für SRAM



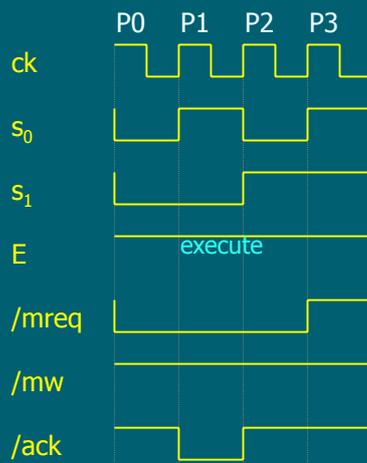
## Lesezugriff hier in 3 Stufen



a) Lesen

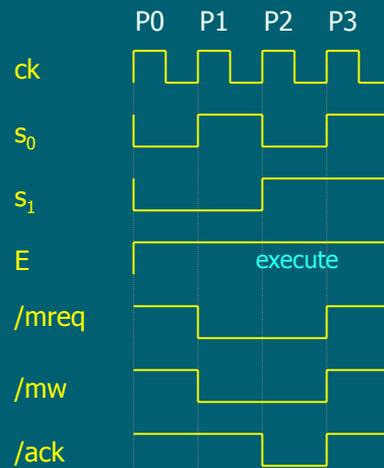
BB - TI II 15.2/15

## Compute memory für SRAM



BB - TI II 15.2/16

## Store für SRAM



BB - TI II 15.2/17

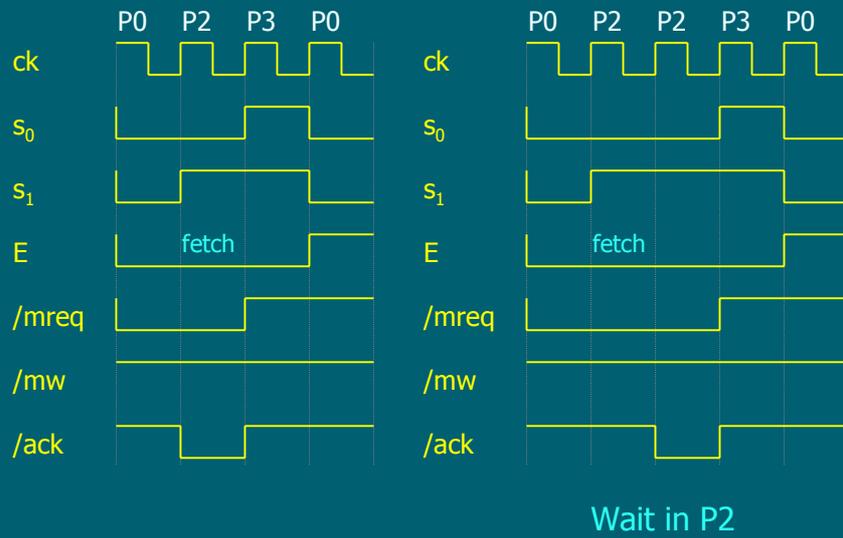
## Langsamere Speicher: Wait-Zyklen

Wenn /ack noch nicht aktiv ist, wird der aktuelle Takt wiederholt

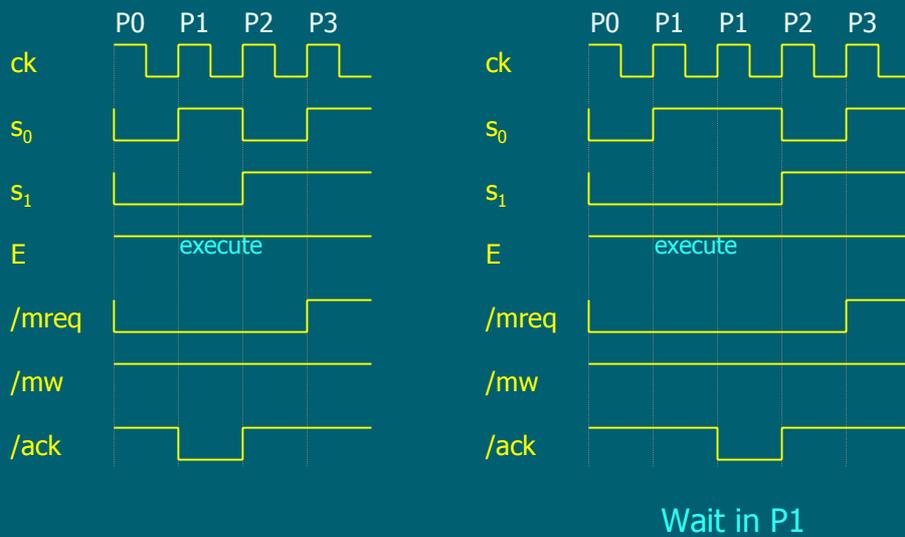
(Vgl. folgende Abbildungen)

BB - TI II 15.2/18

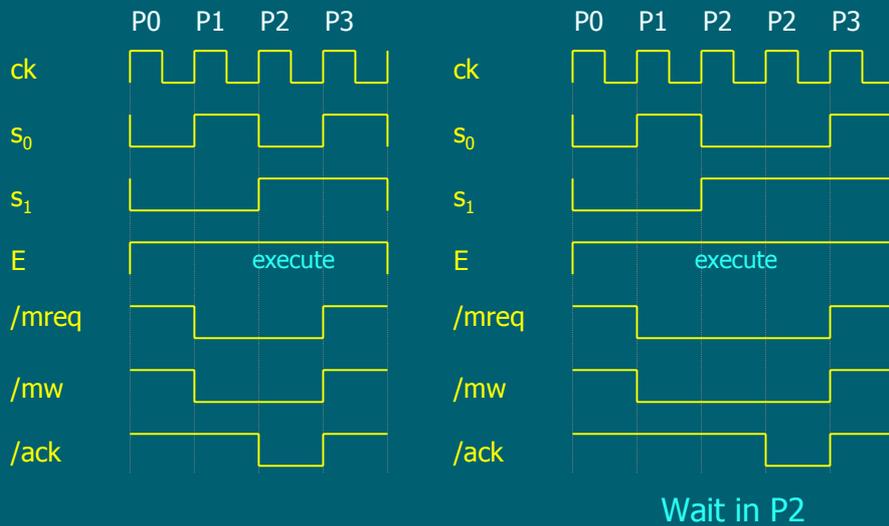
## Wait-Zyklen für Fetch



## Wait-Zyklen für Compute memory



## Wait-Zyklen für Store



## Wait-Zyklen (ff)

Wenn /ack noch nicht aktiv ist, so wird die aktuelle Uhrzeit wiederholt (wait!!), solange bis /ack aktiv, d.h. bei Fetch wird P2 wiederholt, bei Compute memory P1, bei Load/Store P2.

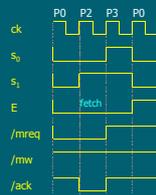
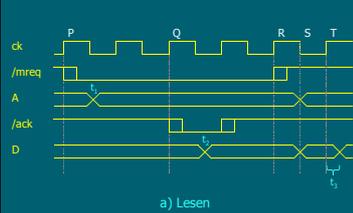
# Exaktes Timing mit neuem Protokoll für SRAM

Adressen sind stabil bei  
(Bezugspunkt ist Flanke bei der  
/mreq aktiviert wird):

**Fetch:**  $(10.5, 18.3) + (2, 8)$   
 /PCAdoe      enable-Zeit PCAd

Lesezugriff hier in 3 Stufen

Fetch für SRAM



## Neue Analyse

1. Neue Analyse von Compute Memory:  
 IAd wird enabled bei P0 von Execute, genauer  
 (vgl. vorige Abbildung)  
 IAdoe1 aktiv zur Zeit  $t_2 = \tau + (8.0, 12.0)$   
 P3 (fetch)      Delay PAL  
 Bezugspunkt!  
 IAdoe aktiv bei  $t_3 = t_2' + (2.5, 6.3) = \tau + (10.5, 18.3)$   
 AND-Gatter

## Zeitangaben zu Treibern

	Treiber 74F244	min	max
$t_{PZL}$	Enable-Zeiten	2.0	8.0
$t_{PZH}$	Enable-Zeiten	2.0	6.7
$t_{DZL}$	Disable-Zeiten	2.0	7.0
$t_{DZH}$	Disable-Zeiten	2.0	7.0
$t_{PHL}$	Umschaltverzögerung bei /OE = 0	2.5	6.2
$t_{PHE}$	Umschaltverzögerung bei /OE = 0	2.5	6.5

BB - TI II 15.2/23

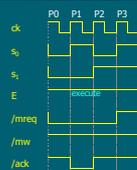
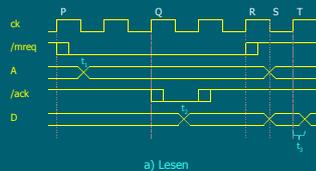
# Exaktes Timing mit neuem Protokoll für SRAM (ff)

Adressen sind stabil bei:

**Compute:**  $(10.5, 18.3) + (2, 8)$   
 /IAoe      enable-Zeit  
 IAd

Lesezugriff hier in 3 Stufen

Compute memory für SRAM



BB - TI II 15.2/24

# Exaktes Timing mit neuem Protokoll für SRAM

## Zeitangaben zu Treibern

Treiber 74F244		min	max
$t_{PD}$	Enable-Zeiten	2.0	8.0
$t_{PZ1}$	Enable-Zeiten	2.0	6.7
$t_{DZ}$	Disable-Zeiten	2.0	7.0
$t_{PDZ}$	Disable-Zeiten	2.0	7.0
$t_{OL1}$	Umschaltverzögerung bei /OE = 0	2.5	6.2
$t_{OL2}$	Umschaltverzögerung bei /OE = 0	2.5	6.5

Adressen sind stabil bei:

**LOADINj, STOREINj:**

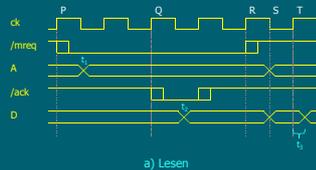
$$(8, 12) + (2, 8) + 46 + 6.5 - \tau$$

IRdoe
enable-Zeit
ALU
ALUAd

IRd

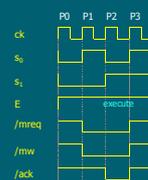
Adressrechnung geht in P0 los, /mreq kommt erst in P1

### Lesezugriff hier in 3 Stufen



a) Lesen

### Store für SRAM



BB - TI II 15.2/25

# Exaktes Timing mit neuem Protokoll für SRAM (ff)

Wegen  $\tau \geq 52.0$  gilt:

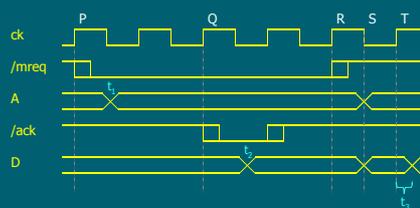
$$(8, 12) + (2, 8) + 46 + 6.5 - \tau \leq 72.5 - 52 = 20.5$$

Insgesamt gilt:

Adressen sind stabil bei

$$t_1 = 26.3$$

### Lesezugriff hier in 3 Stufen



a) Lesen

# Exaktes Timing mit neuem Protokoll für SRAM: Lesezugriff

Zu  $t_2$  :

Gültige Daten liegen auf dem D-Bus zur Zeit

$$26.3 + 6.5 + 45 + 6.5 = 84.3$$

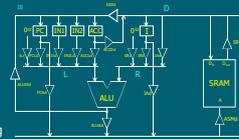


falls wie bisher  
früh genug enabled

Daten auf D

→ Daten auf D spätestens zur Zeit

Treiber-  
verzögerung



$$t_8 = \max(\max(t_2) + 6.5, \max(t_7))$$

schon enabled,  
wenn Daten gültig

nicht „rechtzeitig“  
enabled

$$= \max(3/2 \tau + 84.0, 2\tau + 20.0)$$

\*\* genauer:  
Enablen geschieht  
> 1.5 = 8.0 - 6.5 ns  
bevor Daten gültig  
werden

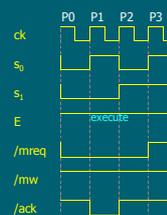
# Exaktes Timing mit neuem Protokoll für SRAM: Lesezugriff (ff)

Compute memory für SRAM

Dies ist

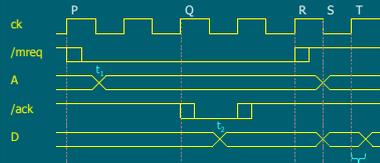
$$t_2 = 84.3 - \tau \leq 84.3 - 52 = 32.3$$

nach P1, bei der /ack für  
Compute memory aktiviert



Lesezugriff hier in 3 Stufen

Lesezugriff hier in 3 Stufen (ff)



- Speicher aktiviert **Acknowledge-Signal /ack** für genau einen Takt an steigender Flanke Q. Nach Verzögerungszeit  $t_2$  nach Q garantiert Speicher korrekte Daten auf D.

B - T1 II 15.2/28

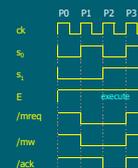
BB - T1 II 15.2/28

## Exaktes Timing mit neuem Protokoll für SRAM: Lesezugriff (ff)

Zu  $t_3$  :

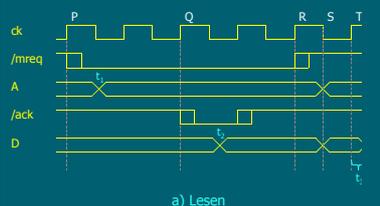
Bei P3 wird /mreq deaktiviert,  
Frühestens 2 Takte später anderer  
Treiber auf D-Bus, also muss gelten  
 $t_3 \leq \tau$ , problemlos möglich bei  $\tau \geq 52.0$

### Store für SRAM



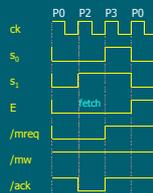
P0 N0 P1 N1 P2 N2 P3 N3 P0 N0 P1 N1 P2 N2 P3 N3

### Lesezugriff hier in 3 Stufen



a) Lesen

### Fetch für SRAM



### Lesezugriff hier in 3 Stufen (ff)

- CPU deaktiviert /mreq an steigender Flanke R.
- Speicher garantiert gültige Daten bis fallende Flanke S nach R und garantiert Disablen der Treiber auf D nach Zeit  $t_3$  nach nachfolgender steigender Flanke T

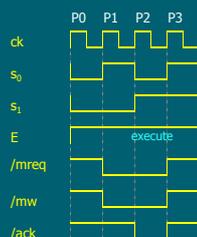
BB - TI10 15.2/17

## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff

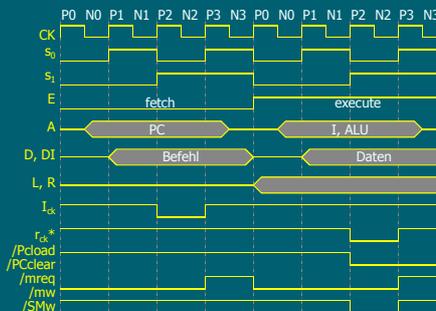
Nach Lesezugriff jetzt noch Schreibzugriff:

/SMw und /ack sind zeitgleich bei P2 aktiv

### Store für SRAM



BB - TI10 15.2/17



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

## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

Überprüfe der Reihe nach Bedingungen zum Funktionieren des SRAMS und Einhalten des Protokolls

### RAM CY7C191-45

Symbol	Bezeichnung	min	max
$t_{acc}$	Lesezugriffszeit	3	45
$t_{SAW}$	Setup-Zeit von A bis W	0	
$t_{SAEW}$	Setup-Zeit von A bis Ende W	35	
$t_{HWA}$	Hold-Zeit von A nach W	0	
w	Schreibpulsweite	22	
$t_{SDEW}$	Setup-Zeit von D bis Ende W	15	
$t_{HWD}$	Hold-Zeit von D nach W	0	

### Schreibzugriff in 2 Stufen (ff)

1. CPU aktiviert  $/mreq$  und  $/mw$  an steigender Flanke P von ck.  
Nach  $t_1$  (nach P) gültige Adressen auf A garantiert, nach  $t_2$  auch gültige Daten auf D

BB - T11 | 15/29

## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

Adressen sind nach  $t_1 = 26.3$  auf A garantiert,  
 $26.3 + 6.5 = 32.8$  nach P1 liegen sie an SM an,  
 wegen  $\tau \geq 52.0$  ist dies vor P2;  
 $t_{SAW}$  und  $t_{SAEW}$  werden eingehalten

$t_1$  ASMd

### RAM CY7C191-45

Symbol	Bezeichnung	min	max
$t_{acc}$	Lesezugriffszeit	3	45
$t_{SAW}$	Setup-Zeit von A bis W	0	
$t_{SAEW}$	Setup-Zeit von A bis Ende W	35	
$t_{HWA}$	Hold-Zeit von A nach W	0	
w	Schreibpulsweite	22	
$t_{SDEW}$	Setup-Zeit von D bis Ende W	15	
$t_{HWD}$	Hold-Zeit von D nach W	0	

### Schreibzugriff in 2 Stufen (ff)

1. CPU aktiviert  $/mreq$  und  $/mw$  an steigender Flanke P von ck.  
Nach  $t_1$  (nach P) gültige Adressen auf A garantiert, nach  $t_2$  auch gültige Daten auf D

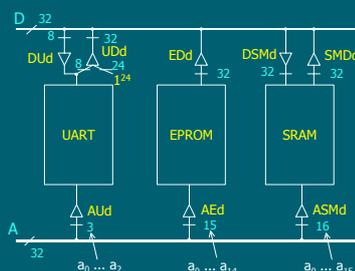
## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

Ebenso wird die Schreibpulsweite eingehalten, wir kümmern uns nun um D:

### RAM CY7C191-45

Symbol	Bezeichnung	min	max
$t_{acc}$	Lesezugriffszeit	3	45
$t_{SAW}$	Setup-Zeit von A bis W	0	
$t_{SAEW}$	Setup-Zeit von A bis Ende W	35	
$t_{HWA}$	Hold-Zeit von A nach W	0	
W	Schreibpulsweite	22	
$t_{SDEW}$	Setup-Zeit von D bis Ende W	15	
$t_{HWD}$	Hold-Zeit von D nach W	0	

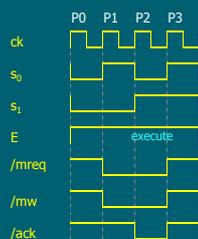
### Datenpfade (graphisch)



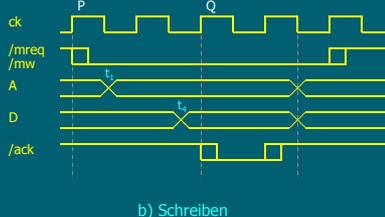
## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

stabile Daten hinter DSMd sollen an P2 vorhanden sein,

### Store für SRAM



### Schreibzugriff in 2 Stufen

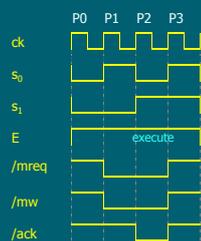


BB - TIH 15.2/17

## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

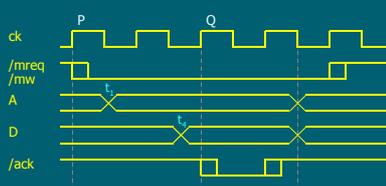
d.h. einen Takt nach P1 (Aktivierung von /mreq, /mw),  
 also  $t_4 + 6.5 \leq \tau$   
 muss gelten.

### Store für SRAM



BB - T11 | 15.2/17

### Schreibzugriff in 2 Stufen

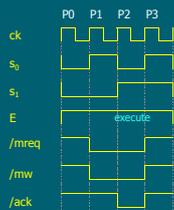


b) Schreiben

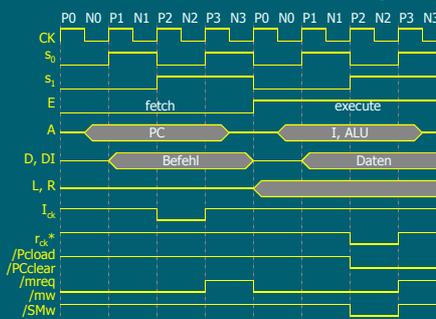
## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

Wegen  $\tau \geq 52.0$  genügt es  $t_4 \leq 45.5$  einzuhalten.  
 Dies geschieht, da ACCDd zur Zeit P1 enabled wird,  
 Daten sind also zur Zeit  $t^+ + (2, 8) = (10, 20)$  stabil.

### Store für SRAM



BB - T11 | 15.2/17

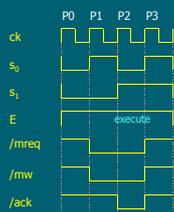


(\* r = PC, IN1, IN2, ACC)

## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

Damit ist insbesondere die Zeit  $t_{SDEW}$  für das SRAM eingehalten, es bleibt der Nachweis der Hold-Zeiten und der Konsistenz mit Schritt 2 des Protokolls beim Schreibzugriff

Store für SRAM



BB - T110 15.217

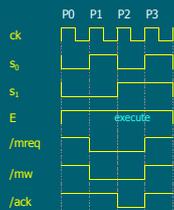
RAM CY7C191-45

Symbol	Bezeichnung	min	max
$t_{ecc}$	Lesezugriffszeit	3	45
$t_{SMW}$	Setup-Zeit von A bis W	0	
$t_{SDEW}$	Setup-Zeit von A bis Ende W	35	
$t_{HWA}$	Hold-Zeit von A nach W	0	
w	Schreibpulsweite	22	
$t_{SDEW}$	Setup-Zeit von D bis Ende W	15	
$t_{HWD}$	Hold-Zeit von D nach W	0	

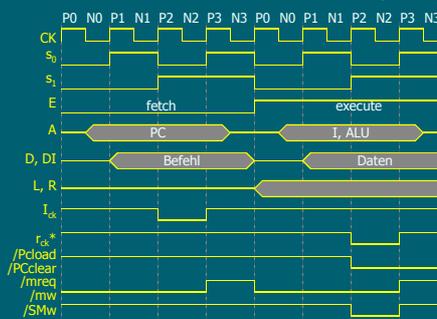
## Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

Zur Einhaltung des Protokolls müssen Adressen und Daten mindestens bis N3 stabil bleiben (dies ist stärker als die vom Speicher geforderten Hold-Zeiten)

Store für SRAM



BB - T110 15.217



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

## Exaktes Timing mit neuem Protokoll für SRAM (ff)

Wir zeigen im folgenden für alle Speicherbausteine:  
Das Protokoll wird eingehalten mit

$$t_1 = 26.3$$

$$t_2 = 32.3$$

$$t_3 \leq 52$$

$$t_4 \leq 45.5$$

BB - TI II 15.2/39

## 15.2.2 Zustandsdiagramme

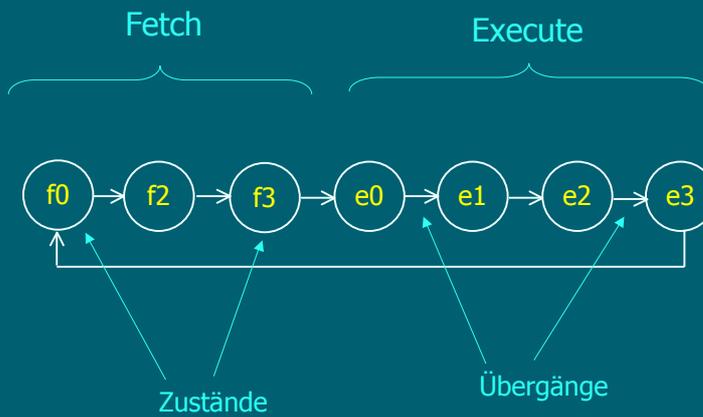
Aufgaben:

- Erzeuge Uhrzeit in Abhängigkeit von Befehlen und /ack-Signal
- Erzeuge für jeden Speicher eine Kontrolleinheit

→ **Zustandsdiagramme**

BB - TI II 15.2/40

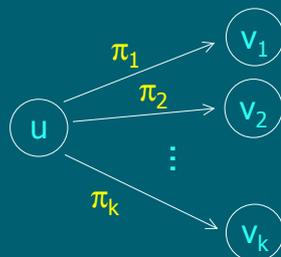
## Uhrzeit bisher:



BB - TI II 15.2/41

## Allgemeinere Zustandsdiagramme:

- Knoten evtl. mehr als eine ausgehende Kante
- Übergang von Knoten  $u$  zu Knoten  $v_i$  in Abhängigkeit von Bedingung  $\pi_i$ :

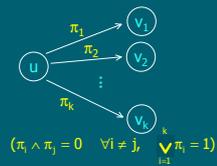


$$(\pi_i \wedge \pi_j = 0 \quad \forall i \neq j, \quad \bigvee_{i=1}^k \pi_i = 1)$$

BB - TI II 15.2/42

## Allgemeinere Zustandsdiagramme:

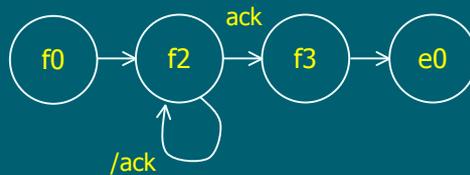
- Zustandsdiagramme können als Schaltung realisiert werden



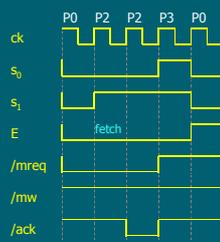
- Wir führen dies exemplarisch vor

BB - TI II 15.2/43

## Zustandsdiagramm für Uhrzeiten: Fetch



### Wait-Zyklen für Fetch



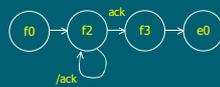
### Kodierung der Zustände für Fetch

Zustand	E	s <sub>1</sub>	s <sub>0</sub>
f0	0	0	0
f2	0	1	0
f3	0	1	1
e0	1	0	0

## Spezifikation der Übergänge

- $E := /E * s_1 * s_0$  ; Übergang von f3 nach e0
- $s_1 := /E * /s_1 * /s_0$  ; Übergang von f0 nach f2  
 +  $/E * s_1 * /s_0$  ; Übergang ausgehend von f2
- $s_0 := /E * s_1 * /s_0 * ack$  ; Übergang von f2 nach f3

Zustandsdiagramm für Uhrzeiten:  
Fetch



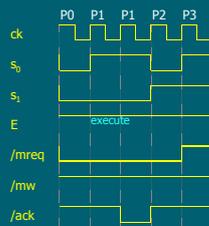
Kodierung der Zustände für Fetch

Zustand	E	s <sub>1</sub>	s <sub>0</sub>
f0	0	0	0
f2	0	1	0
f3	0	1	1
e0	1	0	0

## Zustandsdiagramm für Uhrzeiten: Execute

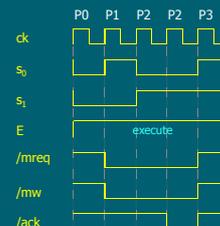
- Wir unterscheiden zwischen Compute memory – Befehlen, Load/Store – Befehlen (außer LOADI, MOVE) und CPU-internen Befehlen
- Dementsprechend gibt es drei Bedingungen cm, ls und in, die die Übergänge bestimmen.

Wait-Zyklen für Compute memory



Wait in P1

Wait-Zyklen für Store



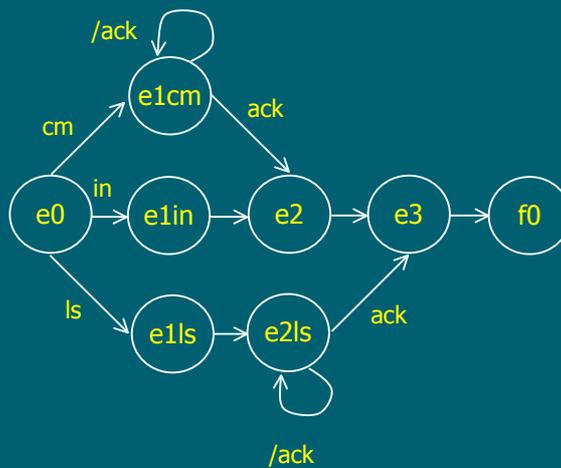
Wait in P2

## Erklärung zu den Zustandsdiagrammen

- $cm = 1 \Leftrightarrow$  Es handelt sich um  
Compute memory – Befehle (waits in P1)
- $ls = 1 \Leftrightarrow$  Es handelt sich um  
Load/Store – Befehle (außer LOADI, MOVE)  
(waits in P2)
- $in = 1 \Leftrightarrow$  Es handelt sich um  
CPU-internen Befehl (keine waits !)

BB - TI II 15.2/47

## Zustandsdiagramm für Uhrzeiten: Execute



BB - TI II 15.2/48

# Erklärung zu den Zu...

Bestimme cm, ls, in folgendermaßen

■  $cm = /I31 * /I30 * I29$

■  $in = /I31 * /I30 * /I29 + /I31 * I30 * I29 * I28 + I31 * /I30 * I29 * I28 + I31 * I30$

■  $ls = /I31 * I30 * /I29 + /I31 * I30 * /I28 + I31 * /I30 * /I29 + I31 * /I30 * /I28$

Jump

## Compute - Befehle (ff)

Typ	MI	F	Befehl	Wirkung
0 0	0	0 1 0	SUBI i	[ACC]:=[ACC] - [i] <PC> := <PC> + 1
		0 1 1	ADDI i	[ACC]:=[ACC] + [i] <PC> := <PC> + 1
		1 0 0	OPLUSI i	ACC := ACC ⊕ 0 <sup>8</sup> i <PC> := <PC> + 1
		1 0 1	ORI i	ACC := ACC ∨ 0 <sup>8</sup> i <PC> := <PC> + 1
		1 1 0	ANDI i	ACC := ACC ∧ 0 <sup>8</sup> i <PC> := <PC> + 1
0 0	1	0 1 0	SUB i	[ACC]:=[ACC] - [M(<i>)] <PC> := <PC> + 1
		0 1 1	ADD i	[ACC]:=[ACC] + [M(<i>)] <PC> := <PC> + 1
		1 0 0	OPLUS i	ACC := ACC ⊕ M(<i>) <PC> := <PC> + 1
		1 0 1	OR i	ACC := ACC ∨ M(<i>) <PC> := <PC> + 1
		1 1 0	AND i	ACC := ACC ∧ M(<i>) <PC> := <PC> + 1
0 1	1 0	0	LOADINZ i	M(<INZ> + [i]) <PC> := <PC> + 1 ACC := 0 <sup>8</sup> i <PC> := <PC> + 1
0 1	1 1	0	LOADI i	ACC := 0 <sup>8</sup> i <PC> := <PC> + 1

## Store, Move - Befehle (ff)

Typ	Modus	Befehl	Wirkung
1 0	0 0	STORE i	M(<i>) := ACC <PC> := <PC> + 1
1 0	0 1	STOREIN1 i	M(<IN1> + [i]) := ACC <PC> := <PC> + 1
1 0	1 0	STOREIN2 i	M(<IN2> + [i]) := ACC <PC> := <PC> + 1
1 0	1 1	MOVE S D	D := S <PC> := <PC> + 1

außer bei D = 0 0 (PC)

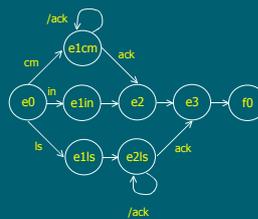
# Erklärung zu den Zustandsdiagrammen

Nach Binärcodierung der Zustände kann Zustandsdiagramm (= endlicher Automat) z.B. durch Register-PAL realisiert werden.

e1 zerlegt in 3 Zustände e1cm, e1in, e1ls

e2 aufgeteilt in 2 Zustände e2, e2ls

## Zustandsdiagramm für Uhrzeiten: Execute



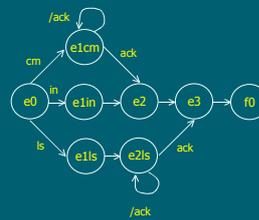
## Zustandskodierung

Erweiterung der Kodierung vom Fetch-Diagramm,  
zwei neue Signale r1, r0  
zur Unterscheidung der e1 und e2 Teil-Zustände

Kodierung der Zustände für  
Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

Zustandsdiagramm für Uhrzeiten:  
Execute



BB - 11.8 15.248

## Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

## Realisierung durch PAL-Gleichungen

Hier nur einige Kommentare:  
r0 ergibt sich durch Ersetzen  
von in und Ausmultiplizieren  
bei

$$r0 := E * /s1 * /s0 * in$$

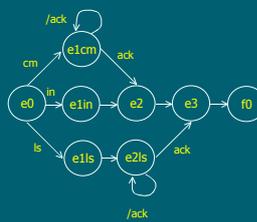
$$+ E * /s1 * s0 * r1 * /r0$$

$$+ E * s1 * /s0 * r0 * /ack$$

### Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

### Zustandsdiagramm für Uhrzeiten: Execute



BB - 11 II 15.2/48

## Realisierung durch PAL-Gleichungen

Bei r1 ersetze ls

$$r1 := E * /s1 * /s0 * ls$$

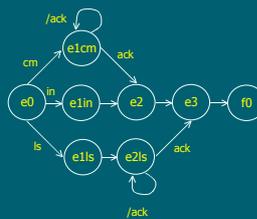
E wird aktiviert nach f3 und  
gehalten bis e3

s1 wird aktiviert nach f0, e1cm  
mit ack, e1in und e1ls;  
gehalten nach f2, e2 und e2ls

### Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

### Zustandsdiagramm für Uhrzeiten: Execute



BB - 11 II 15.2/48

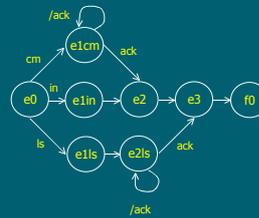
# Realisierung durch PAL-Gleichungen

s0 wird aktiviert nach f2 mit ack,  
 nach e2ls mit ack und nach e2;  
 gehalten nach e1cm mit /ack

## Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	E	s <sub>1</sub>	s <sub>0</sub>	r1	r0
f0	0	0	0	0	0
f2	0	1	0	0	0
f3	0	1	1	0	0
e0	1	0	0	0	0
e1cm	1	0	1	0	0
e1in	1	0	1	0	1
e1ls	1	0	1	1	0
e2	1	1	0	0	0
e2ls	1	1	0	0	1
e3	1	1	1	0	0

## Zustandsdiagramm für Uhrzeiten: Execute



BB - 11.8 15.248

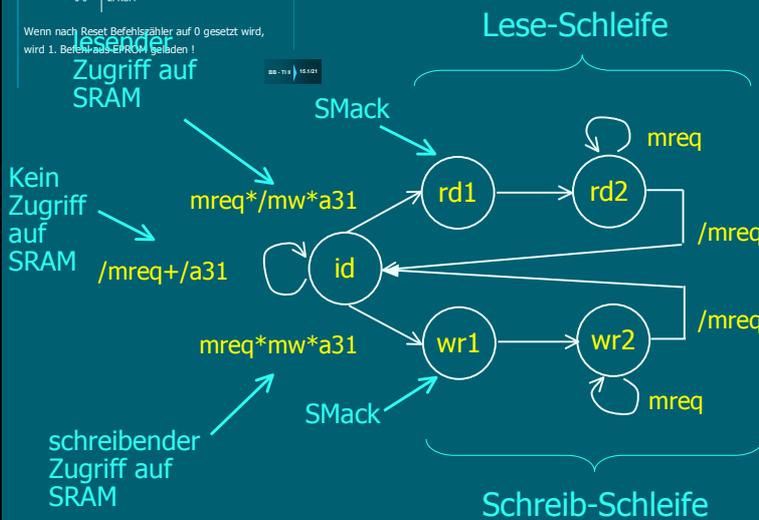
## Memory Map

Festlegung, welcher Speicher unter welcher Adresse angesprochen wird durch Memory Map.

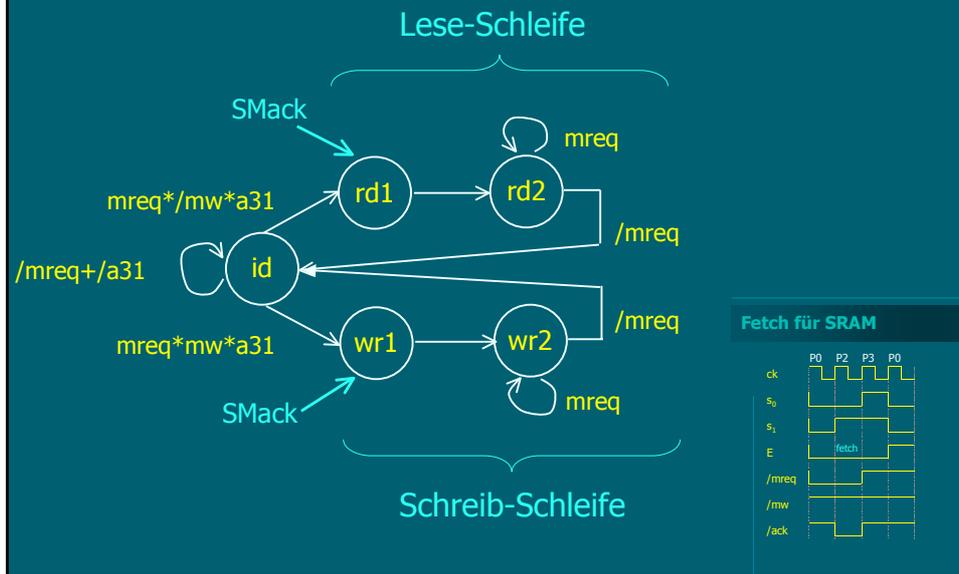
A[31:30]	Einheit
1 1	SRAM
0 1	URAM
0 0	EPROM

Wenn nach Reset Befehlszähler auf 0 gesetzt wird, wird 1. Befehl aus EPROM geladen!

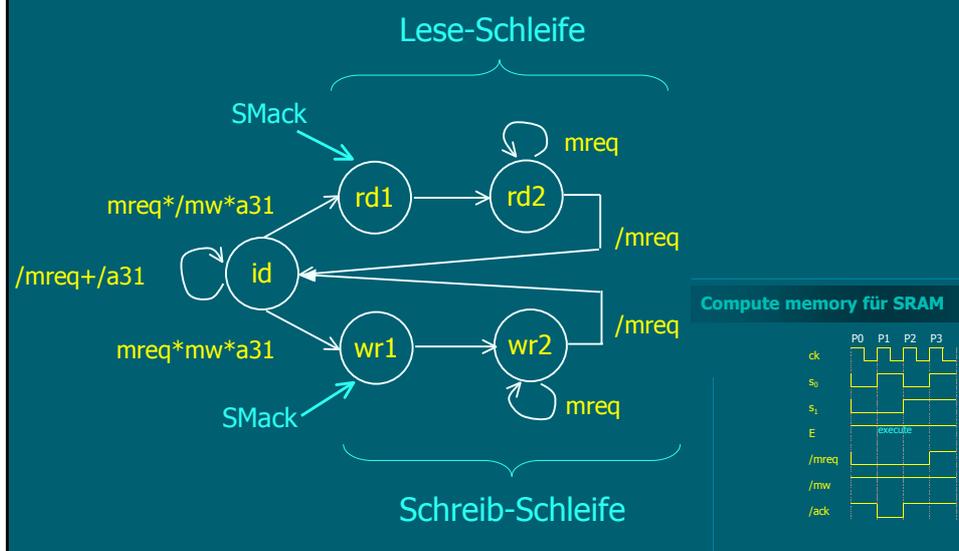
## Programm für Speicher:



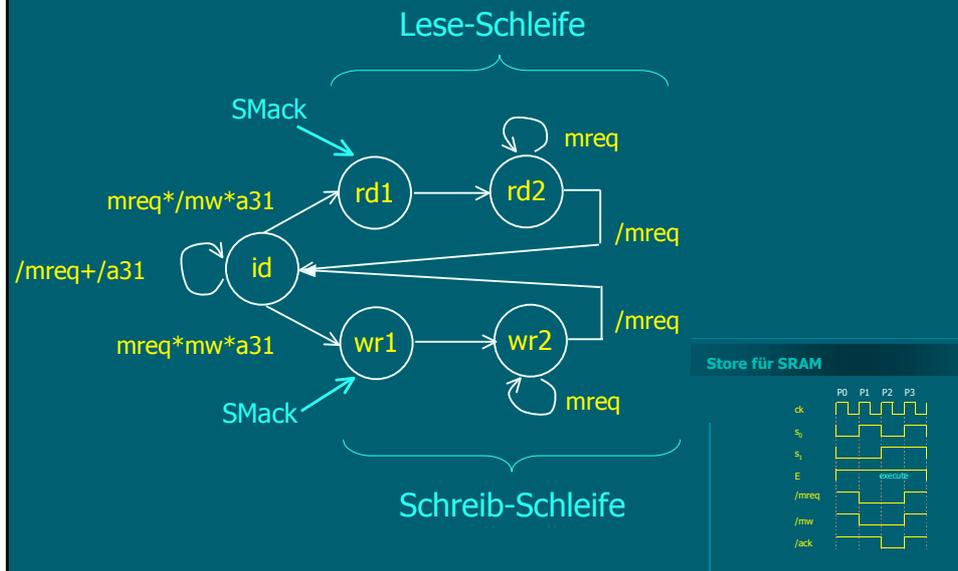
## Korrespondenz zu Fetch auf CPU-Seite



## Korrespondenz zu Compute memory auf CPU-Seite



## Korrespondenz zu Store auf CPU-Seite



## Kontrollsignale aufgrund der Speicherkontrolle

$SMDd\phi$ ,  $SMw$ ,  $SMack$  werden aufgrund der Zustände der Speicherkontrolle berechnet:

$SMack$  aktiviert in rd1, wr1 für einen Takt

$SMDd\phi$  aktiviert in rd1, rd2, disabled im Zustand nach rd2

$SMw$  aktiviert in wr1 für einen Takt

## Realisierung des Automaten und der Kontrollsignale

Benutze P-PALs und 3 Signale sr0, sr1, sr2 zur Kodierung der 5 Zustände

→ einfache Übung

BB - TI II 15.2/61

## Detailliertes Timing für SRAM-Zugriffe

### Exaktes Timing mit neuem Protokoll für SRAM: Lesezugriff

Zu  $t_2$   
Gültige Daten liegen auf dem D-bus zur Zeit

$$26.3 + 6.5 + 45 + 6.5 = 84.3$$

- Nach unserer alten Analyse gilt: Adressen sind stabil auf A nach  $t_1 = 26.3$  (gerechnet ab Flanke P, bei der /mreq aktiviert wird)
- Wegen  $\tau \geq 52.0 \geq t_1 + 15$  ist die Zeit ausreichend für setup der PALs und die Kontrollsignale für des SRAM werden im nächsten Takt Q erzeugt
- Damit liegen zur Zeit  
 $26.3 + 6.5 + 45 + 6.5 = 84.3$   
gültige Daten auf dem D-Bus. Dies ist  
 $t_2 = 84.3 - \tau \leq 84.3 - 52 = 32.3$   
nach Q, bei dem /ack aktiviert wird



BB - TI II 15.2/62

D

Wir zeigen im folgenden für alle Speicherbausteine:  
Das Protokoll wird eingehalten mit

$$t_1 = 26.3$$

$$t_2 = 32.3$$

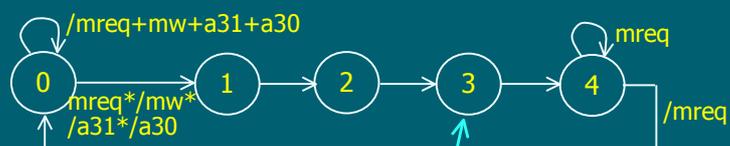
$$t_3 \leq 52$$

$$t_4 \leq 45.5$$

BB - TI II 15.2/39

BB - TI II 15.2/63

## Zustandsdiagramm für Speicher: EPROM



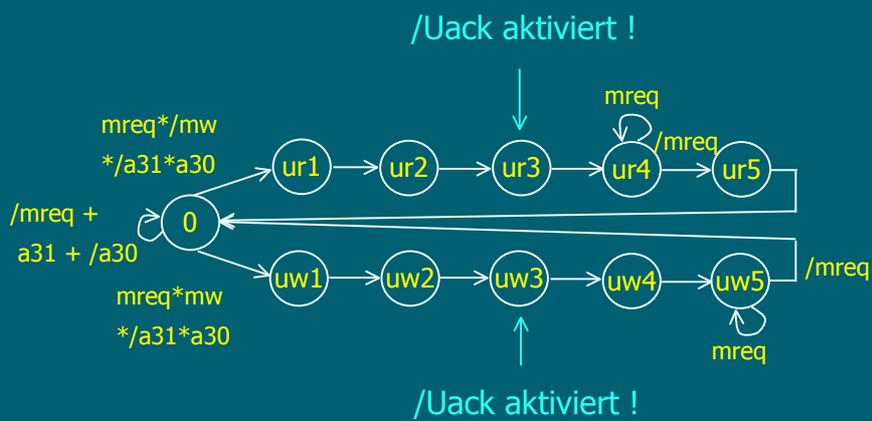
Eack aktiviert

## EPROM

wie SRAM, aber nur Lesen und 2 Wait-Zyklen,  
da **Eack** erst in Zustand (3) aktiviert wird  
(**EDdœ** aktiv von (1) bis (4) )

BB - TI II 15.2/65

## Zustandsdiagramm für Speicher: UART



## UART

/Uack aktiviert in ur3 bzw. uw4

→ bei Lesen 2 Wait-Zyklen,  
bei Schreiben 3 Wait-Zyklen

(ur5 wird gebraucht, um Lesezykluszeit zu garantieren)