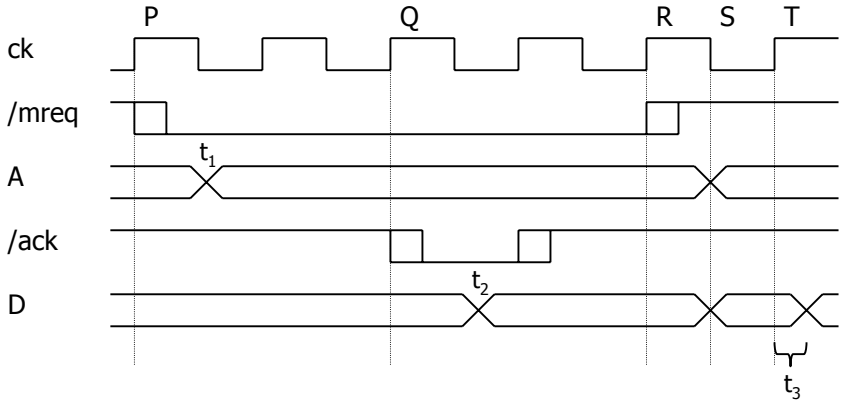
15.2
Busprotokolle, Zustandsdiagramme

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

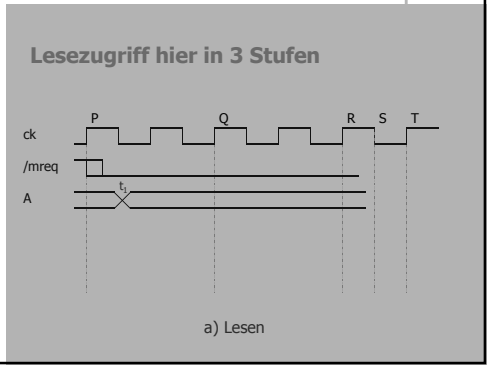
Lesezugriff hier in 3 Stufen



a) Lesen

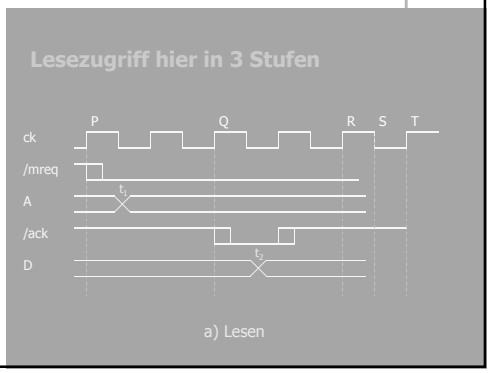
Lesezugriff hier in 3 Stufen (ff)

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.



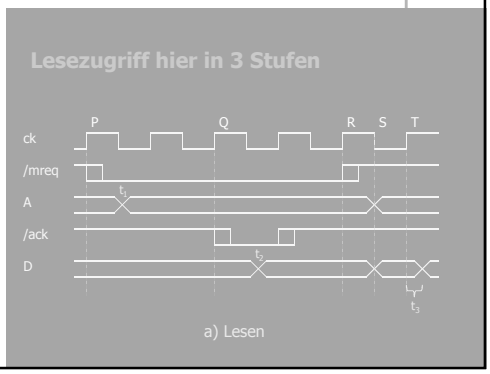
Lesezugriff hier in 3 Stufen (ff)

2. 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.

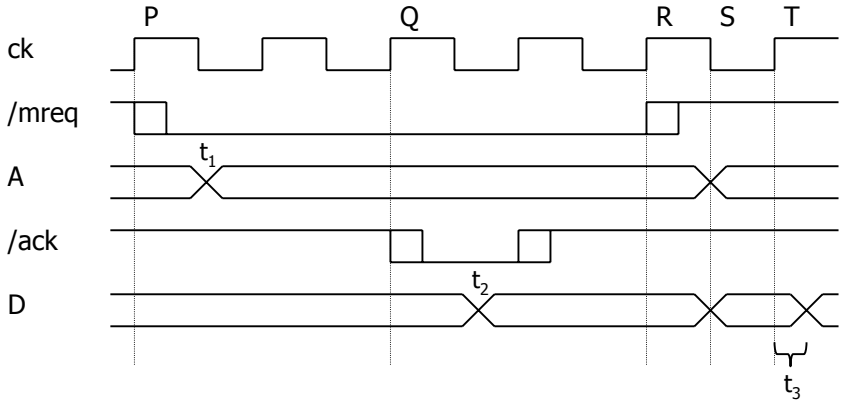


Lesezugriff hier in 3 Stufen (ff)

3. 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

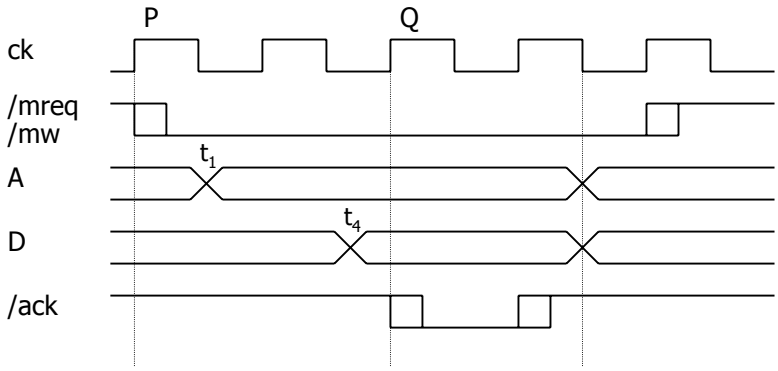


Lesezugriff hier in 3 Stufen



a) Lesen

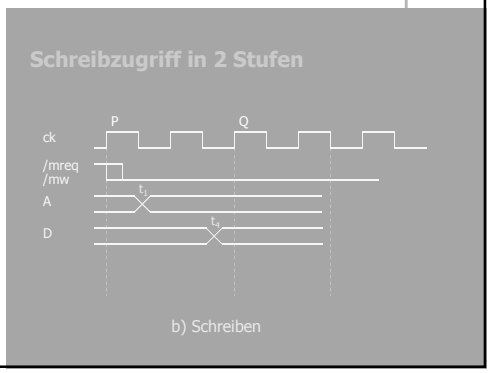
Schreibzugriff in 2 Stufen



b) Schreiben

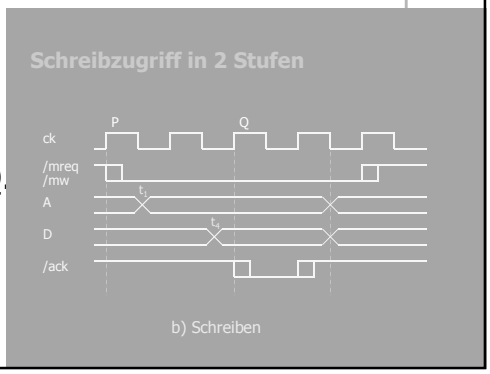
Schreibzugriff in 2 Stufen (ff)

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

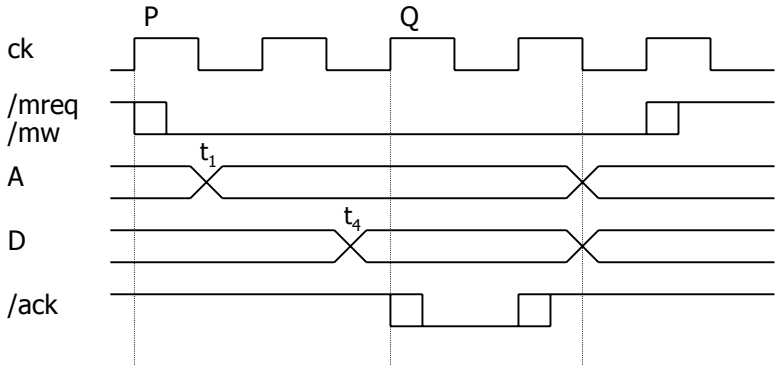


Schreibzugriff in 2 Stufen (ff)

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



b) Schreiben

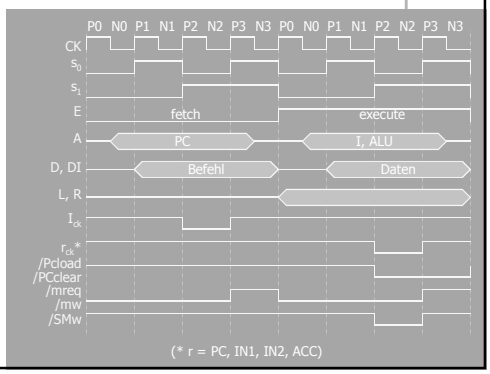
Idealisiertes Timing mit Wait-Zyklen

Timing von /mreq und /mw wird leicht abgeändert:

Bisher:

Aktivieren bei P0,

Deaktivieren bei P3.



Idealisiertes Timing mit Wait-Zyklen

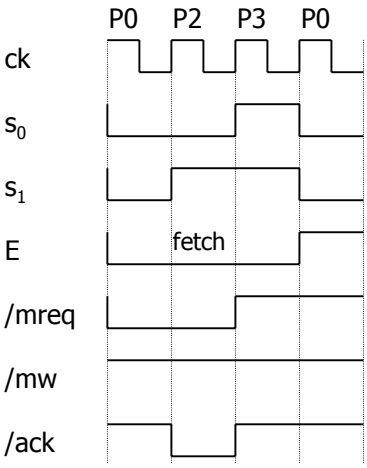
Jetzt:
Fetch und Compute memory bleiben wie bisher;
bei Load und Store (außer LOADI, MOVE) wird
Aktivierungszeitpunkt auf P1 verschoben,
um Zeit für Adressrechnung zu haben.

Zunächst SRAM (schneller Speicher)

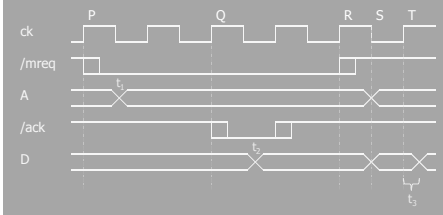
Kontrolle von SRAM wird so entworfen, dass
/ack genau einen Takt nach /mreq aktiv wird.
CPU-Kontrolle soll dann keine Wait-Zyklen erzeugen.

(Vgl. folgende Abbildungen)

Fetch für SRAM

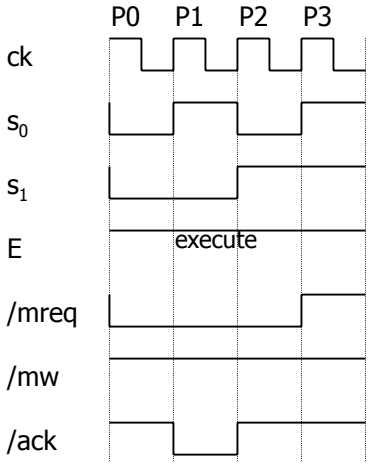


Lesezugriff hier in 3 Stufen

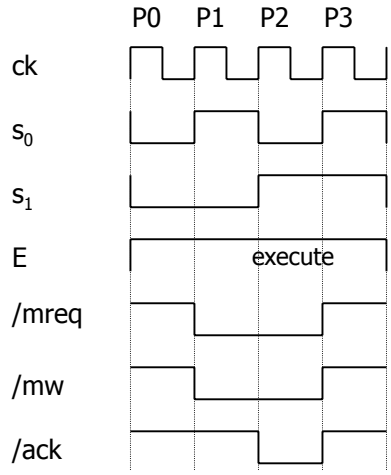


a) Lesen

Compute memory für SRAM



Store für SRAM

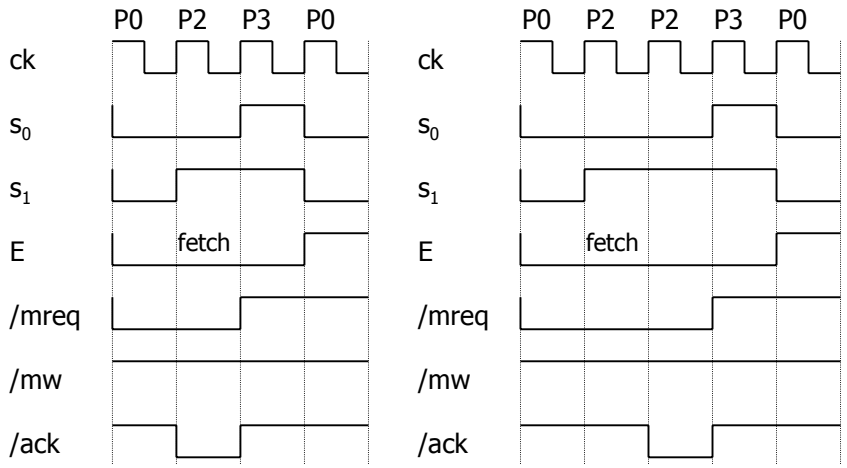


Langsamere Speicher: Wait-Zyklen

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

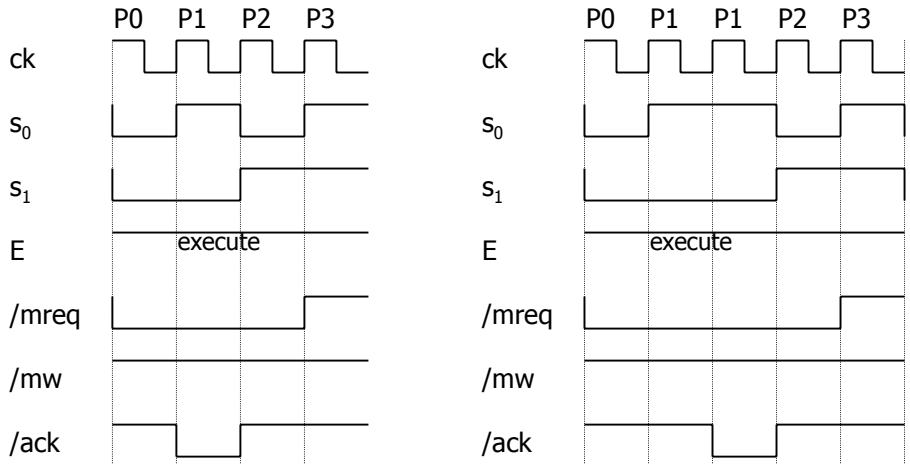
(Vgl. folgende Abbildungen)

Wait-Zyklen für Fetch



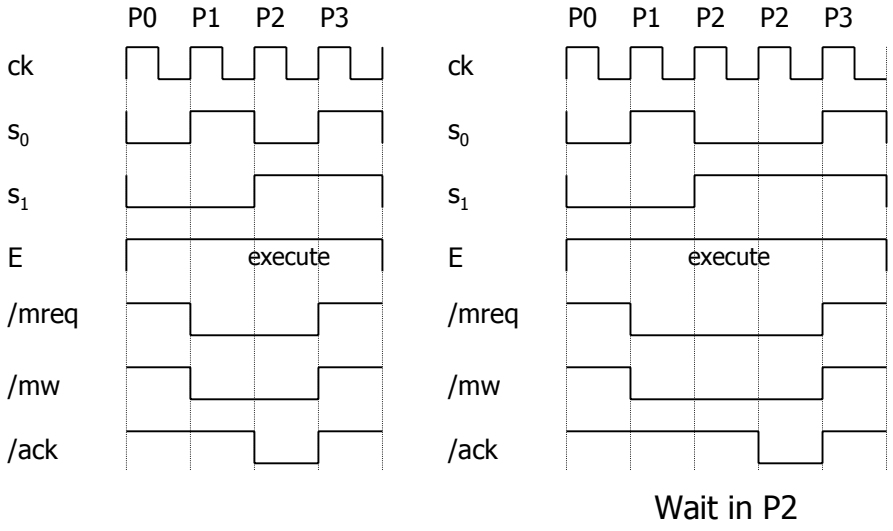
Wait in P2

Wait-Zyklen für Compute memory



Wait in P1

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

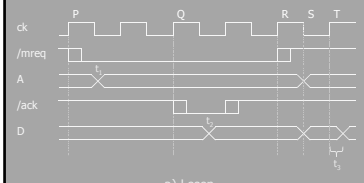
Neue Analyse

1. Neue Analyse von Compute Memory:
   IAd wird enabled bei P0 von Execute, genauer (vgl. vorige Abbildung)
   IAdœ1 aktiv zur Zeit $t_2' = \tau + (8.0, 12.0)$
   P3 (fetch) Bezugspunkt! — Delay PAL
   IAdœ aktiv bei $t_2 = t_2' + (2.5, 6.3) = \tau + (10.5, 18.3)$
   AND-Gatter

Zeitangaben zu Treibern

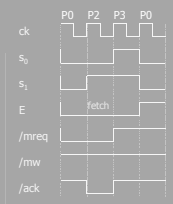
	Treiber 74F244	min	max
t_{PZL}	Enable-Zeiten	2.0	8.0
t_{PZH}	Enable-Zeiten	2.0	6.7
t_{PLZ}	Disable-Zeiten	2.0	7.0
t_{PHZ}	Disable-Zeiten	2.0	7.0
t_{PLH}	Umschaltverzögerung bei /OE = 0	2.5	6.2
t_{PHL}	Umschaltverzögerung bei /OE = 0	2.5	6.5

Lesezugriff hier in 3 Stufen



a) Lesen

Fetch für SRAM



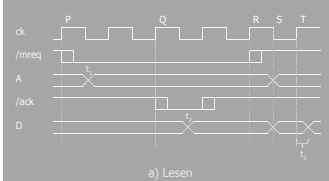
Exaktes Timing mit neuem Protokoll für SRAM (ff)

Adressen sind stabil bei:

Compute: (10.5, 18.3) + (2, 8)

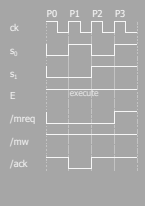
/IAdoe enable-Zeit IAd

Lesezugriff hier in 3 Stufen



a) Lesen

Compute memory für SRAM



Exaktes Timing mit neuem Protokoll für SRA

Adressen sind stabil bei:

LOADINj, STOREINj:

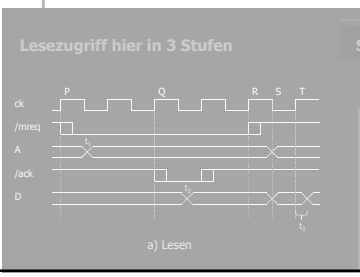
$(8, 12) + (2, 8) + 46 + 6.5 - \tau$

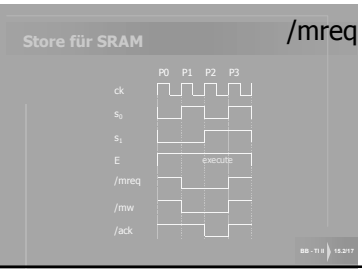
IRdoe — enable-Zeit IRd — ALU — ALUAd

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

Zeitangaben zu Treibern

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





Exaktes Timing mit neuem Protokoll für SRAM (ff)

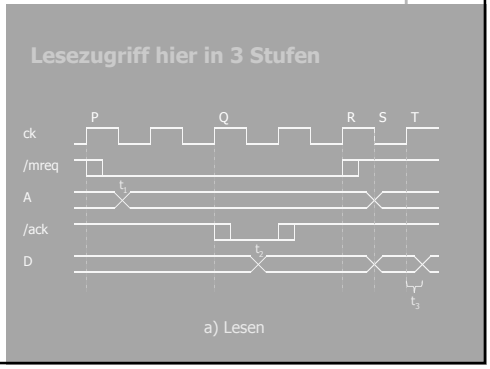
Wegen $\tau \geq 52.0$ gilt:

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

Insgesamt gilt:

Adressen sind stabil bei

$t_1 = 26.3$



Exaktes Timing mit neuem Protokoll für SRAM: Lesezugriff

Zu t_2 :

Gültige Daten liegen auf dem D-Bus zur Zeit

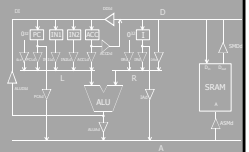
26.3 + 6.5 + 45 + 6.5 = 84.3

t_1 ASMd SM SMDd,

falls wie bisher früh genug enabled

Daten auf D

→ Daten auf D spätestens zur Zeit



$t_8 = \max(\max(t_5) + 6.5, \max(t_7))$

schon enabled, wenn Daten gültig **

nicht *„rechtzeitig"* enabled

= max(3/2 τ + 84.0, 2τ + 20.0)

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

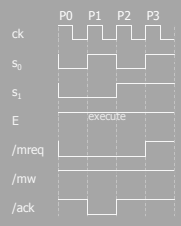
Exaktes Timing mit neuem Protokoll für SRAM: Lesezugriff (ff)

Dies ist

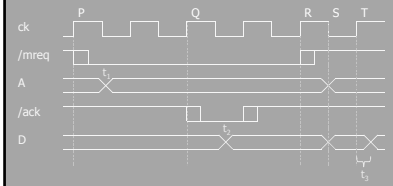
$t_2 = 84.3 - \tau \leq 84.3 - 52 = 32.3$

nach P1, bei der /ack für Compute memory aktiviert

Compute memory für SRAM



Lesezugriff hier in 3 Stufen



a) Lesen

Lesezugriff hier in 3 Stufen (ff)

2. 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.

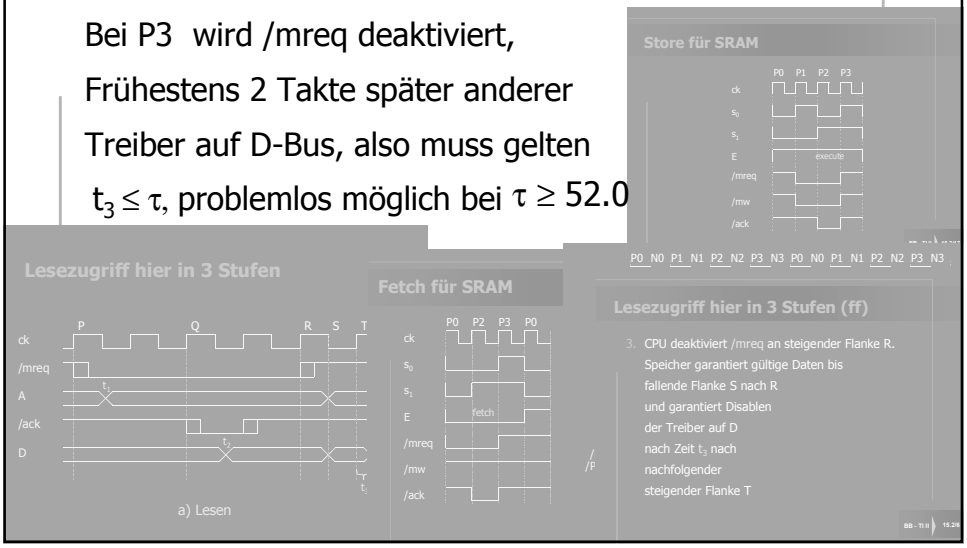
Exaktes Timing mit neuem Protokoll für SRAM: Lesezugriff (ff)

Zu t_3 :

Bei P3 wird /mreq deaktiviert,

Frühestens 2 Takte später anderer Treiber auf D-Bus, also muss gelten

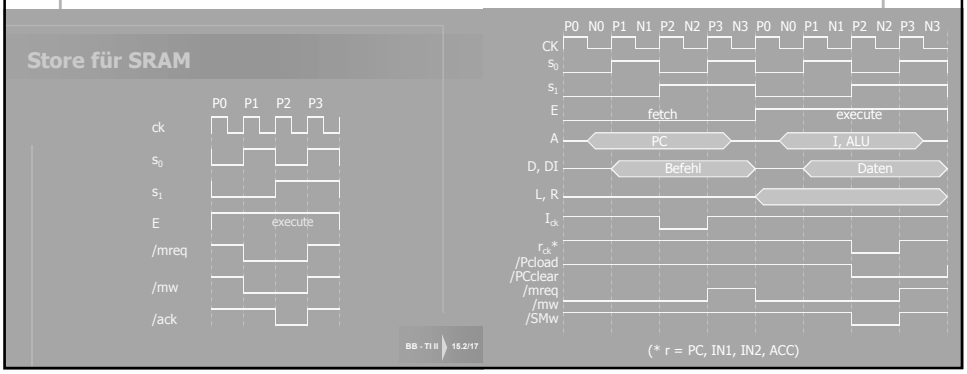
$t_3 \leq \tau$, problemlos möglich bei $\tau \geq 52.0$



Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff

Nach Lesezugriff jetzt noch Schreibzugriff:

/SMw und /ack sind zeitgleich bei P2 aktiv



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_4 auch gültige
   Daten auf D

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,

t_1 ASMd

wegen $\tau \geq 52.0$ ist dies vor P2;

t_{SAW} und t_{SAEW} werden eingehalten

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_4 auch gültige
   Daten auf D

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

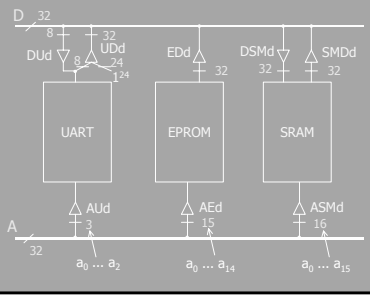
Ebenso wird die Schreibpulsweite eingehalten,

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)

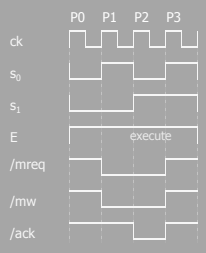


Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

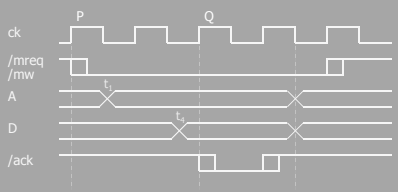
stabile Daten hinter DSMd sollen an P2

vorhanden sein,

Store für SRAM



Schreibzugriff in 2 Stufen



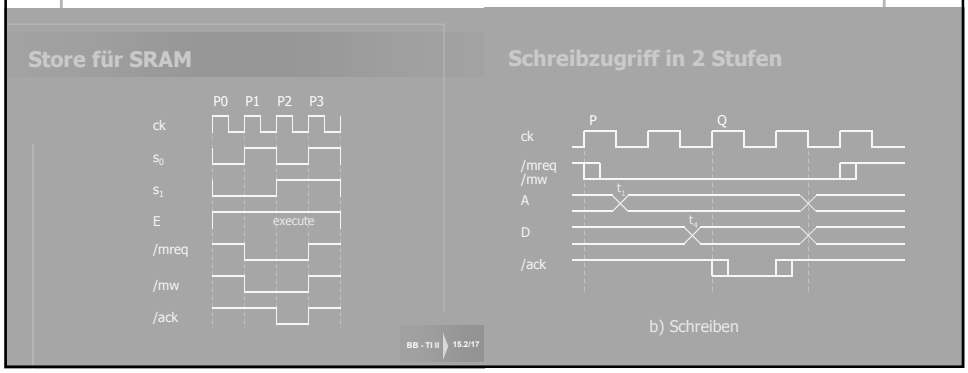
b) Schreiben

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.

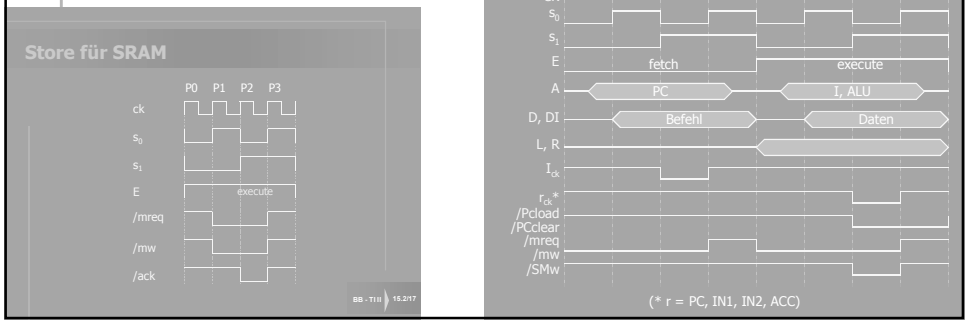


Exaktes Timing mit neuem Protokoll für SRAM: Schreibzugriff (ff)

Wegen $\tau \geq 52.0$ genügt es $t_4 \leq 45.5$ einzuhalten.

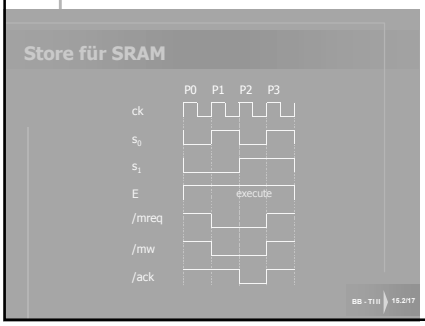
Dies geschieht, da ACCDd zur Zeit P1 enabled wird,

Daten sind also zur Zeit $t^+ + (2, 8) = (10, 20)$ stabil.



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

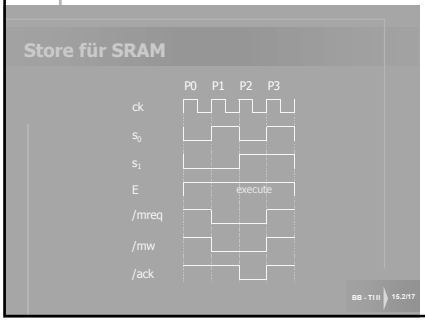


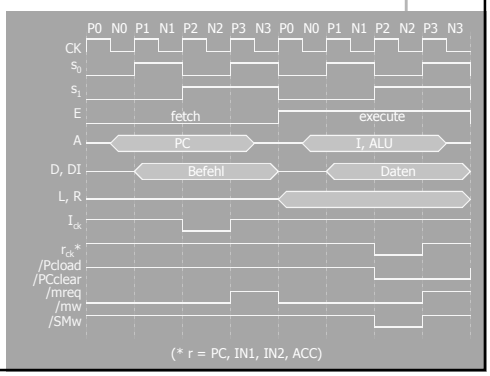
RAM CY7C191-45

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	

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)





Exaktes Timing mit neuem Protokoll für SRAM (ff)

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

$t_1 = 26.3$
$t_2 = 32.3$
$t_3 \leq 52$
$t_4 \leq 45.5$

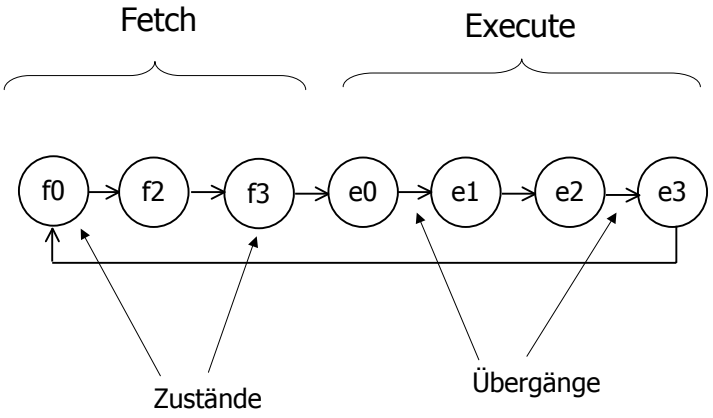
15.2.2 Zustandsdiagramme

Aufgaben:

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

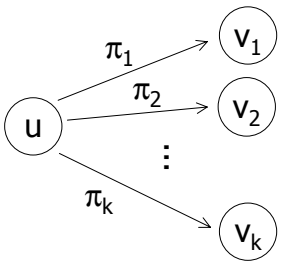
→ Zustandsdiagramme

Uhrzeit bisher:



Allgemeinere Zustandsdiagramme:

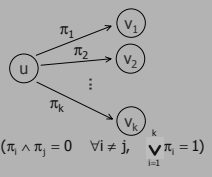
- Knoten evtl. mehr als eine ausgehende Kante
- Übergang von Knoten u zu Knoten v_i in Abhängigkeit von Bedingung π_i :



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

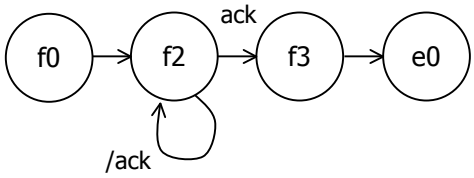
Allgemeinere Zustandsdiagramme:

- Zustandsdiagramme können als Schaltung realisiert werden

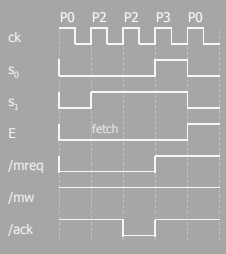


- Wir führen dies exemplarisch vor

Zustandsdiagramm für Uhrzeiten: Fetch



Wait-Zyklen für Fetch



Wait in P2

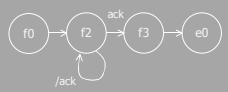
Kodierung der Zustände für Fetch

Zustand	E	s_1	s_0
f0	0	0	0
f2	0	1	0
f3	0	1	1
e0	1	0	0

Spezifikation der Übergänge

- E := /E * s1 * s0 ; Übergang von f3 nach e0
- s1 := /E * /s1 * /s0 ; Übergang von f0 nach f2
  + /E * s1 * /s0 ; Übergang ausgehend von f2
- s0 := /E * s1 * /s0 * ack ; Übergang von f2 nach f3

Zustandsdiagramm für Uhrzeiten: Fetch



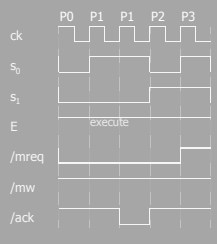
Kodierung der Zustände für Fetch

Zustand	E	s_1	s_0
fo	0	0	0
f2	0	1	0
f3	0	1	1
e0	1	0	0

Zustandsdiagramm für Uhrzeiten: Execute

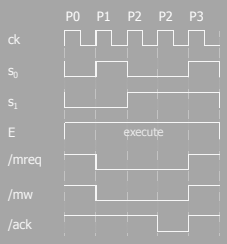
- Wir unterscheiden zwischen Compute memory – Befehlen, Load/Store – Befehlen (außer LOADI, MOVE) und CPU-internen Befehlen
- Dementsprechend gibt es drei Bedingungen cm, ls und in, die die Übergänge bestimmen.

Wait-Zyklen für Compute memory



Wait in P1

Wait-Zyklen für Store

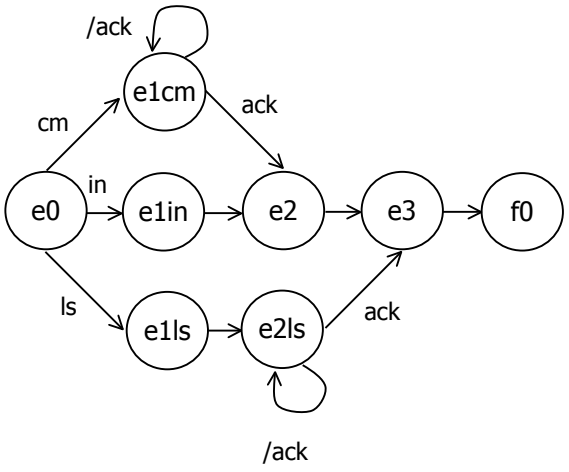


Wait in P2

Erklärung zu den Zustandsdiagrammen

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

Zustandsdiagramm für Uhrzeiten: Execute



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

Jump

- ls = /I31 * I30 * /I29
  + /I31 * I30 * /I28
  + I31 * /I30 * /I29
  + I31 * /I30 * /I28

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 ⊕ 0^8 i	<PC> := <PC> + 1
		1 0 1	ORI i	ACC : = ACC ∨ 0^8 i	<PC> := <PC> + 1
		1 1 0	ANDI i	ACC := ACC ∧ 0^8 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	LOADIN2 i	M(<IN2> + [i])	<PC> + 1
0 1		1 1	0	LOADI i	ACC := 0^8 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)

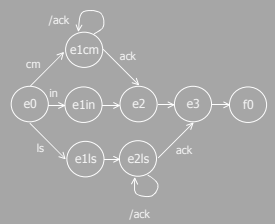
Erklärung zu den Zustandsdiagrammen

Nach Binärkodierung 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



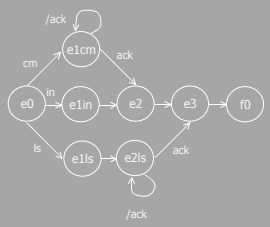
Zustandskodierung

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_1	s_0	r1	r0
fo	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



Kodierung der Zustände für Zustandsdiagramme auf CPU Seite

Zustand	**E**	**s_1**	**s_0**	r1	r0
fo	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:

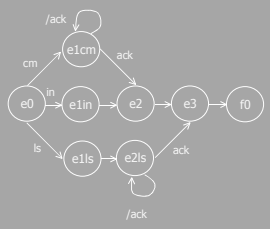
r0 ergibt sich durch Ersetzen von in und Ausmultiplizieren bei

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_1	s_0	r1	r0
fo	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



Realisierung durch PAL-Gleichungen

Bei r1 ersetze ls

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

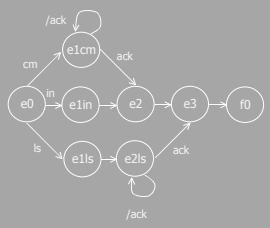
E wird aktiviert nach f3 und gehalten bis e3

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_1	s_0	r1	r0
fo	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



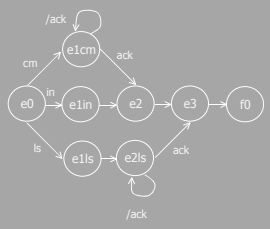
Realisierung durch PAL-Gleichungen

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_1	s_0	r1	r0
fo	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



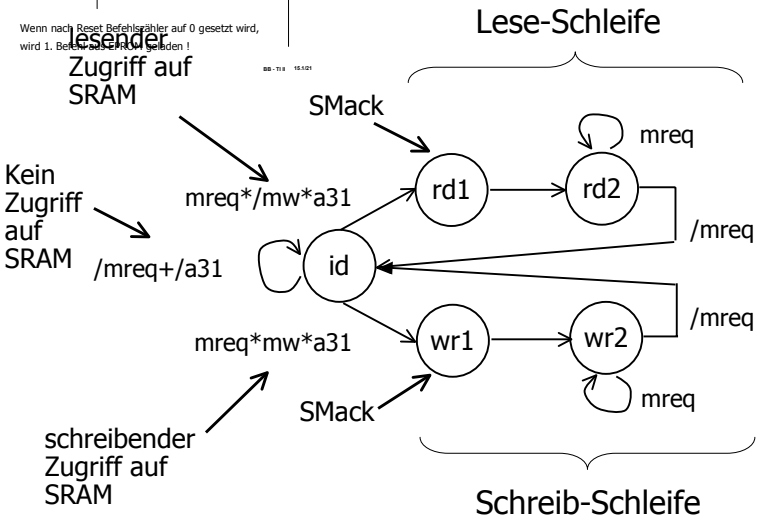
Memory Map

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

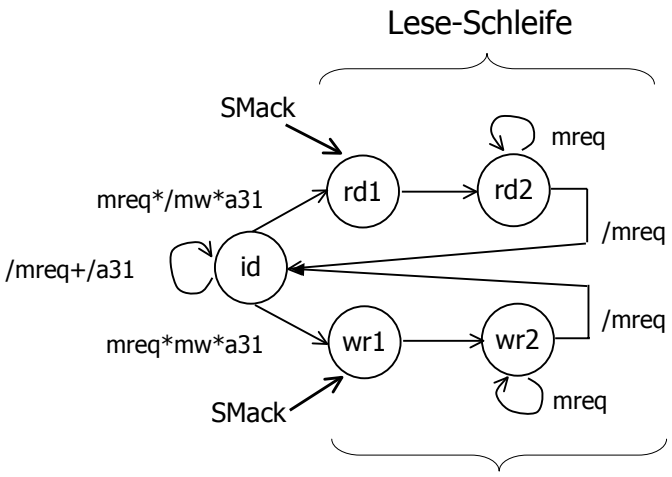
A[31:30]	Einheit
1 *	SRAM
0 1	UART
0 0	EPROM

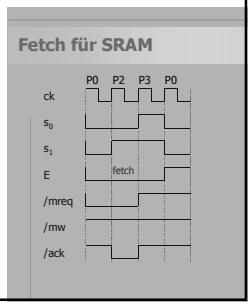
Wenn nach Reset Befehlszähler auf 0 gesetzt wird, wird 1. Befehl aus EPROM geladen !

...gramm für Speicher:

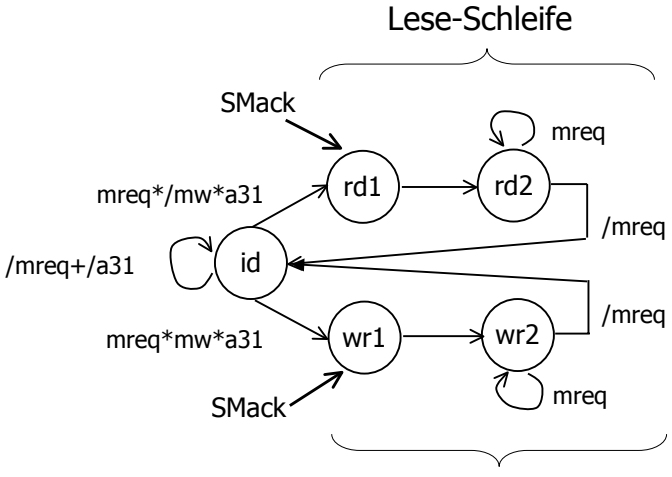


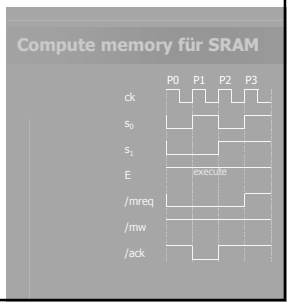
Korrespondenz zu Fetch auf CPU-Seite



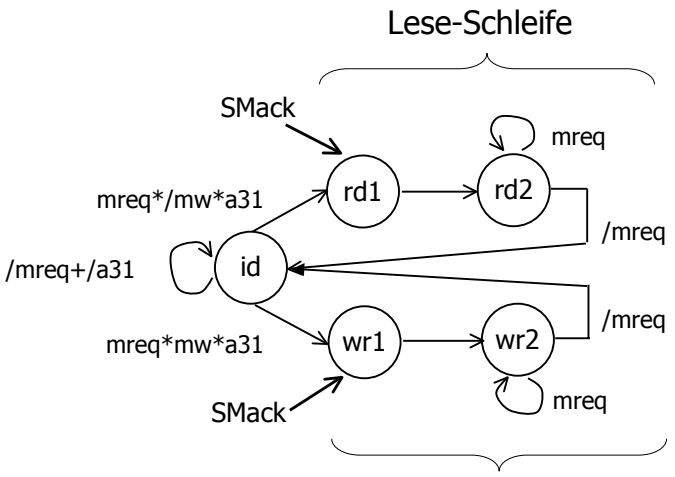


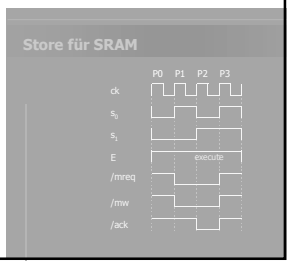
Korrespondenz zu Compute memory auf CPU-Seite





Korrespondenz zu Store auf CPU-Seite





Kontrollsignale aufgrund der Speicherkontrolle

SMDdœ, SMw, SMack werden aufgrund der Zustände der Speicherkontrolle berechnet:

SMack aktiviert in rd1, wr1 für einen Takt

SMDdœ aktiviert in rd1, rd2, disabled im Zustand nach rd2

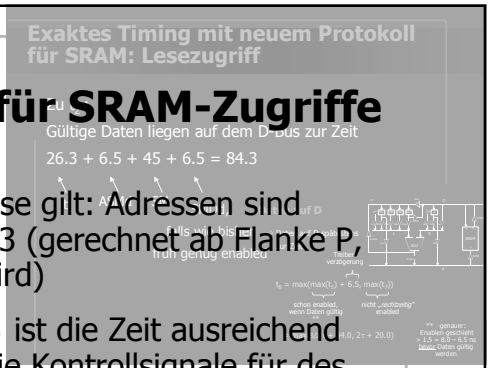
SMw aktiviert in wr1 für einen Takt

Realisierung des Automaten und der Kontrollsignale

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

→ einfache Übung

Detailiertes Timing für SRAM-Zugriffe



- 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

D

Wir zeigen im folgenden für alle Speicherbausteine:

Das Protokoll wird eingehalten mit

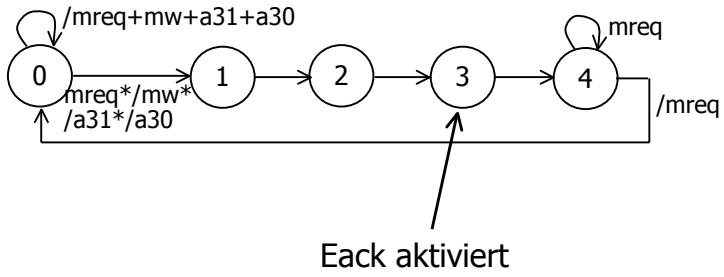
$t_1 = 26.3$

$t_2 = 32.3$

$t_3 \leq 52$

$t_4 \leq 45.5$

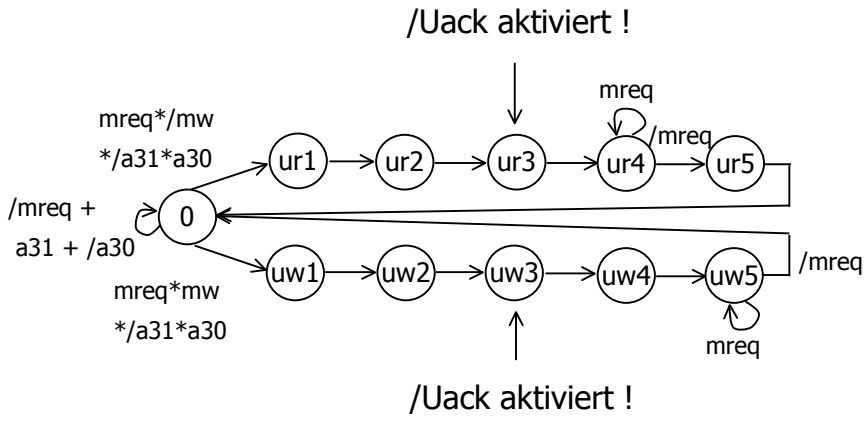
Zustandsdiagramm für Speicher: EPROM



EPROM

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

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)