## Hilfestellung bei der Klausurvorbereitung

Einige Vorlesungen Computersysteme
(experimentell, ab 4. Vorlesung, nicht die Vorlesungen im Chemie-Saal) wurden aufgezeichnet unter:

http://data.mip.informatik.uni-kiel.de:555/tos/CompSys/WS18-19/

User: stud
Passwort: 25.AU.XT.15 Ab heute abend freigeschaltet
(passwort wird in den Folien veröffentlicht)

Sie können diese zu Ihrer Vorbereitung nutzen, aber:

Die Videos dürfen NICHT irgendwo anders veröffentlicht werden (wie Youtube, eigene Server etc). Sie sind nur für Ihre privaten Vorbereitungszwecke zur Klausur zu nutzen. Da meine Persönlichkeitsrechte betroffen sind, muss und werde ich evtl. Verletzungen durch Veröffentlichung rechtlich verfolgen!

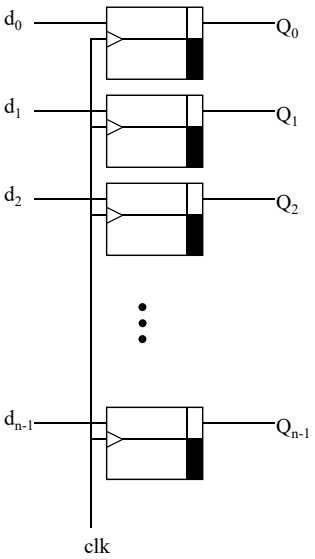
## 7. Spezielle Schaltwerke

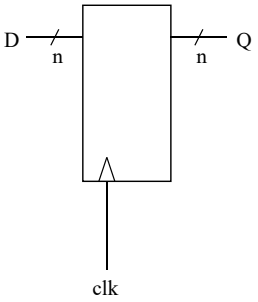
In diesem Abschnitt werden wir einige Schaltwerke kennenlernen, die als Basisbauteile überall im Aufbau digitaler Schaltungen verwendet werden.

### Das Register

Das Register oder der Wortspeicher ist eine Parallelschaltung von n Flipflops. In einem Register kann eine binäres Wort der Länge n gespeichert werden. Die Flipflops können Latches (Auffangflipflops) oder System-Flipflops (z.B. D-Flipflops) sein. Die folgende Folie zeigt den Aufbau eines Registers und sein Schaltbild.

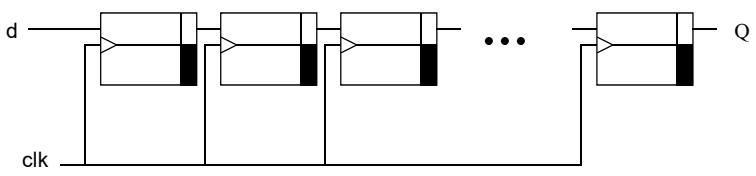
## Register



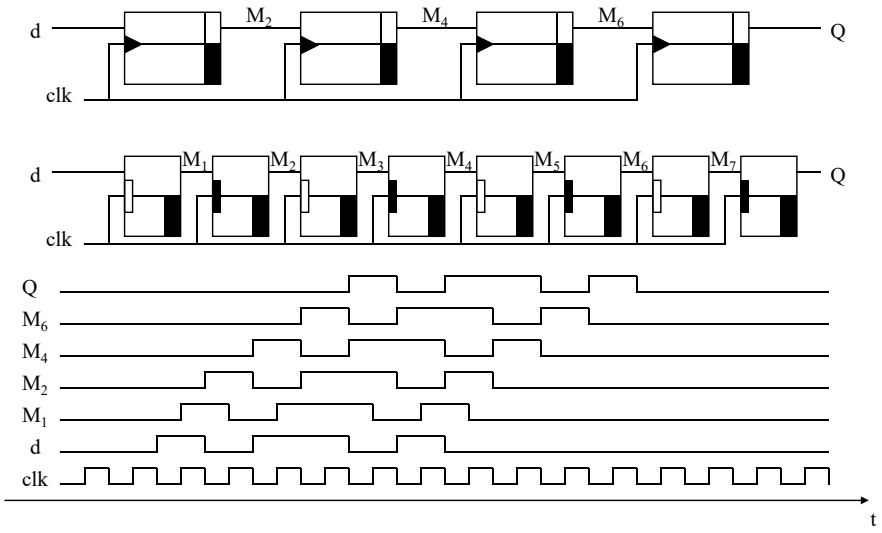




## Das Schieberegister

Das Schieberegister ist ein Wortspeicher mit bit-seriellem Zugriff. Es besteht aus einer Serienschaltung von n D-Flipflops. In einem Schieberegister kann eine binäres Wort der Länge n dadurch gespeichert werden, daß pro Takt ein Bit eingegeben wird. Entsprechend wird das gespeicherte Wort in n aufeinanderfolgenden Takten am Ende des Schieberegisters ausgegeben. Die Flipflops müssen System-Flipflops sein.
Wir wissen bereits, daß man ein System-Flipflop als Master-Slave-Flipflop aus zwei Latches aufbauen kann. Diesen Aufbau und den Verlauf eines Signals 001011010000 entlang des Schieberegisters sieht man auf der nächsten Folie.



## 4-Bit-Schieberegister



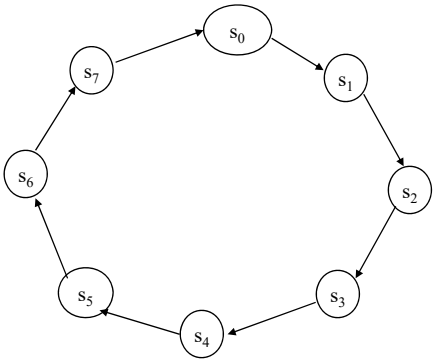

## Zähler

Zähler sind spezielle Schaltwerke, die in der Regel wenige (u.U. gar keine) Eingänge haben. Oft sind auch Ihre Ausgaben identisch mit den Zuständen. Der Zustand ist der Stand des Zählers. Häufig ist der Zählerstand eine binär codierte Zahl. Der Takt bewirkt, dass der Zähler zählt, also seinen Zustand wechselt. Wir zählen also quasi die Anzahl der ansteigenden Taktflanken.

Aus der Sicht der Automatentheorie haben wir bisher alle Schaltwerke als Mealy-Automaten gebaut. Wenn die Ausgabe aber nun nicht von der aktuellen Eingabe sondern nur vom aktuellen Zustand abhängt, so handelt es sich um einen Moore-Automaten. Wenn die Ausgabe identisch mit dem aktuellen Zustand ist, so sprechen wir von einem Medwedew-Automaten.

Wir werden in diesem Kapitel eine Reihe unterschiedlicher Zähler kennenlernen. Als einleitenden Beispiel wählen wir einen synchronen Modulo-8-Zähler. Auf der folgenden Folie ist der Automatengraph für diesen dargestellt: Der Modulo-8-Zähler wandert zyklisch durch 8 Zustände, wobei er bei jedem Takt einen Schritt macht.

## Modulo-8-Zähler




Kodierung der Zustände (D-FF):

| | $z_2$ $z_1$ $z_0$ |
|---|---|
| $s_0$: | 0 0 0 |
| $s_1$: | 0 0 1 |
| $s_2$: | 0 1 0 |
| $s_3$: | 0 1 1 |
| $s_4$: | 1 0 0 |
| $s_5$: | 1 0 1 |
| $s_6$: | 1 1 0 |
| $s_7$: | 1 1 1 |

## Modulo-8-Zähler mit D-FF

Wertetabelle:

| $z_2$ | $z_1$ | $z_0$ | $z'_2$ | $z'_1$ | $z'_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |

| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |

$z'_0$

| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |

$z'_1$

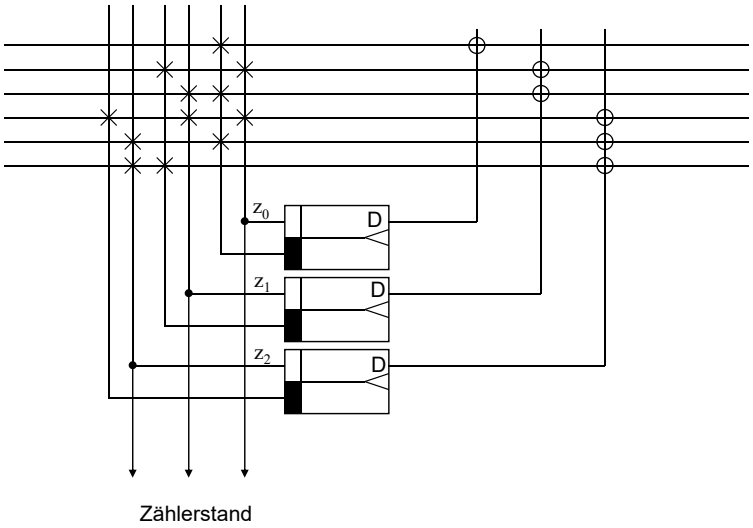
| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |

$z'_2$

Ergebnis:

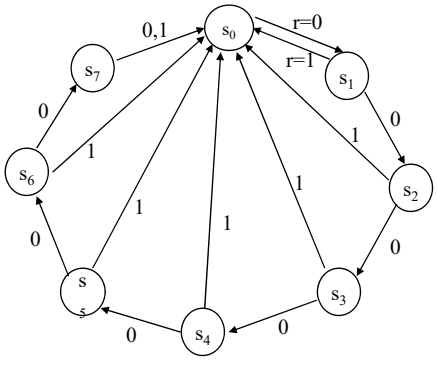
$$z'_0 = \overline{z}_0$$
$$z'_1 = z_0\overline{z}_1 + \overline{z}_0 z_1$$
$$z'_2 = z_0 z_1 \overline{z}_2 + \overline{z}_0 z_2 + \overline{z}_1 z_2$$

## FPLA für Modulo-8-Zähler mit D-FF




## Modulo-8-Zähler mit Reset-Erweiterung

Unser Modulo-8-Zähler hat noch einen kleinen Schönheitsfehler: Wir können ihn noch nicht auf einen definierten Anfangszustand setzen. Wenn wir das zusätzlich wollen, braucht das Schaltwerk doch einen Eingang, z.B. ein Signal „r“. Wenn dieses 1 ist, soll der Zähler in den Zustand $s_0$ gehen. Der entsprechend veränderte Zustandsgraph sieht dann so aus:



## Modulo-8-Zähler mit Reset-Erweiterung

Wertetabelle:

| r | $z_2$ | $z_1$ | $z_0$ | $z'_2$ | $z'_1$ | $z'_0$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Ergebnis:

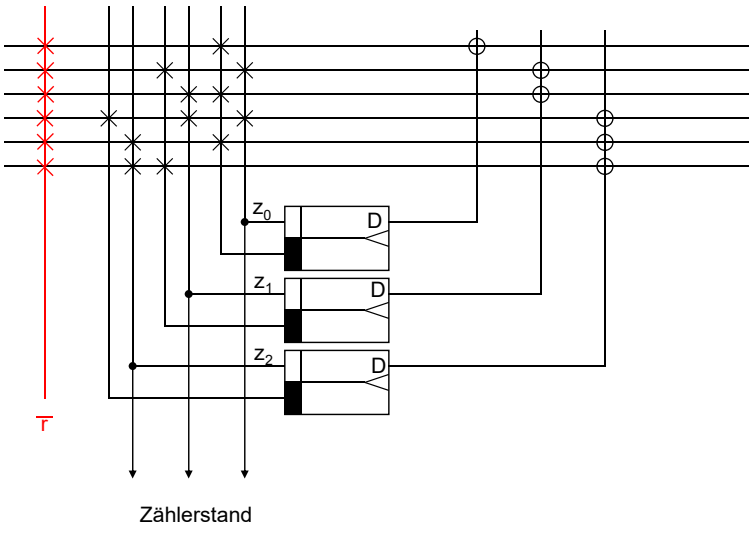
$z'_0 = \overline{r}\,\overline{z_0}$

$z'_1 = \overline{r}(z_0\overline{z_1} + \overline{z_0}\,z_1)$

$z'_2 = \overline{r}(z_0\,z_1\overline{z_2} + \overline{z_0}\,z_2 + \overline{z_1}\,z_2)$

=> Alle Primterme der Originallösung werden durch $\overline{r}$ erweitert

## Modulo-8-Zähler mit Reset-Erweiterung



## Modulo-8-Zähler mit r-s-FF

Während D-Flipflops in der Regel die geeigneten Bausteine zum Speichern von Zuständen in Schaltwerken sind, werden insbesondere bei Zählern häufig auch andere Flipfloptypen eingesetzt, weil sich dadurch Schaltungsaufwand in den Schaltnetzen einsparen läßt. Das folgende Beispiel ist einen Modulo-8-Zähler, der mit r-s-Flipflops aufgebaut werden soll. In der Wertetabelle können an vielen Stellen „dont cares“ eingesetzt werden, nämlich immer dann, wenn der erwünschte Wert erhalten bleiben soll, was durch zwei verschiedene Befehle erreicht wird:
„0“ nach „0“ durch „store“ (s=0 r=0) oder „reset“ (s=0 r=1) => (s=0 r=X)
„1“ nach „1“ durch „store“ (s=0 r=0) oder „set“ (s=1 r=0) => (s=X r=0)

| $z_2$ | $z_1$ | $z_0$ | $s_2$ | $r_2$ | $s_1$ | $r_1$ | $s_0$ | $r_0$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X | 0 | X | 1 | 0 |
| 0 | 0 | 1 | 0 | X | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | X | X | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | X | 0 | 0 | X | 1 | 0 |
| 1 | 0 | 1 | X | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | X | 0 | X | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | X | X | 0 | X |

$s_2 = z_0 z_1 \overline{z_2}$

| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | X | 0 | X |
| 1 | 0 | 0 | 1 | 0 |

$r_2 = z_0 z_1 z_2$

| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | X |
| 1 | 0 | 1 | 0 | X |

$s_1 = z_0 \overline{z_1}$

| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | 0 | 1 | 0 |
| 1 | X | 0 | 1 | 0 |

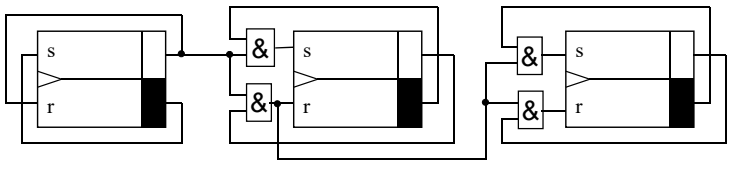
$r_1 = z_0 z_1$

| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |

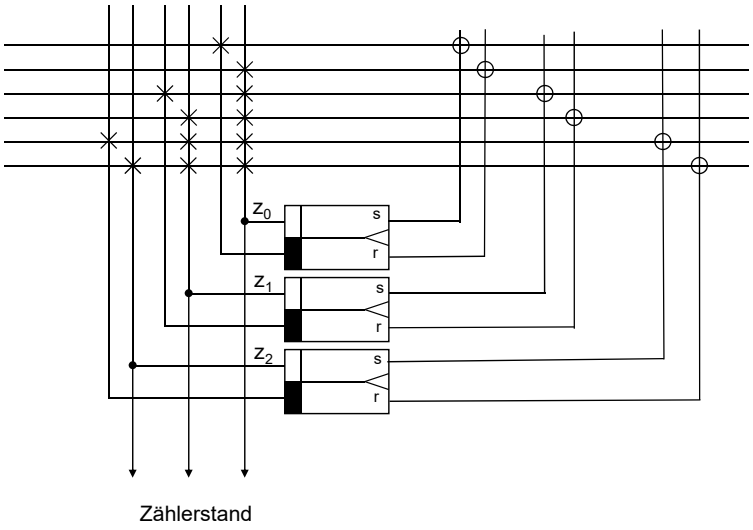
$s_0 = \overline{z_0}$

| $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |

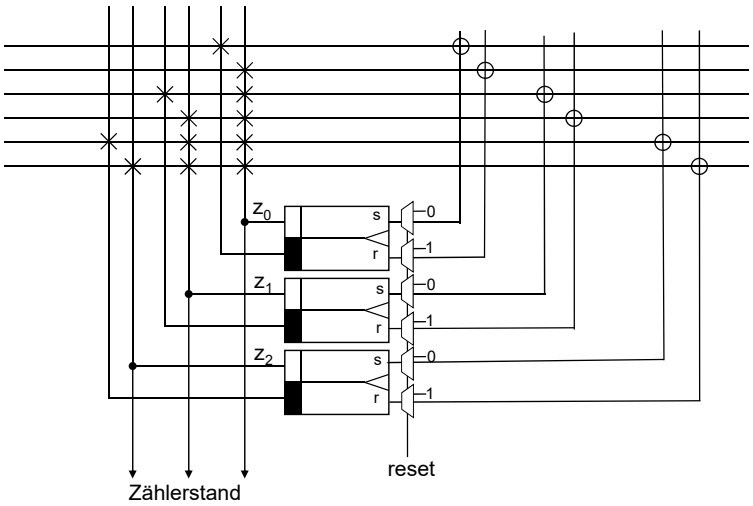
$r_0 = z_0$



## FPLA für Modulo-8-Zähler mit r-s-FF




## Reset-Erweiterung für Modulo-8-Zähler mit r-s-FF



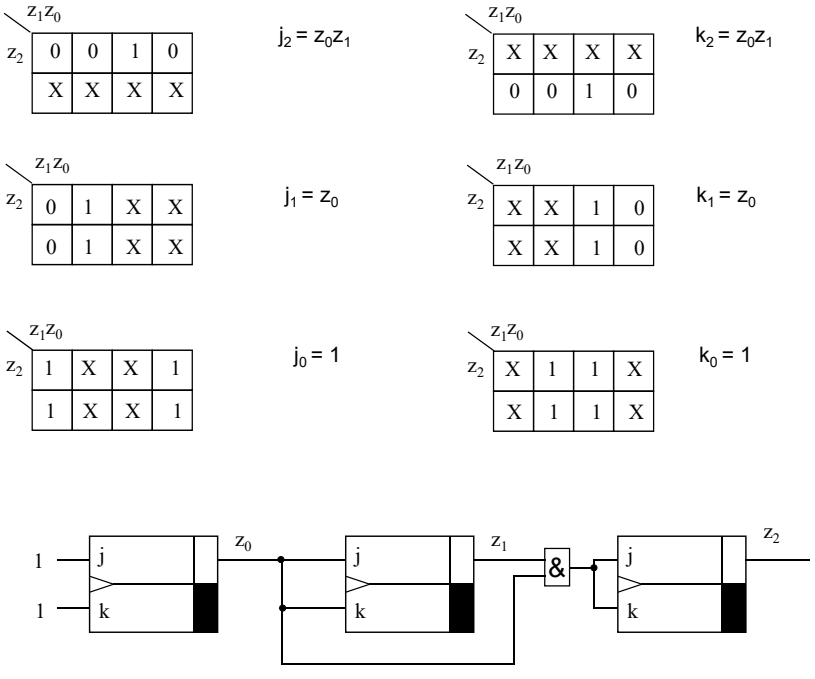

Die reset-Leitung schaltet Multiplexer wahlweise auf reset=1 (s=0,r=1) oder reset=0 (Zählen) um

## Einsatz von J-K-Flipflops zum Bau von Zählern

Wir können weiteren Schaltungsaufwand einsparen, wenn wir statt r-s-Flipflops J-K-Flipflops verwenden. Hier kommt uns zu Hilfe, daß die Eingabe j=k=1 zum Toggle des Zustands genutzt werden kann. Die Beibehaltung des Zustandes (0 oder 1) führt wie beim r-s-FF zu „don't care" (X), aber auch die Änderung des Zustands kann durch Toggle (j=1 k=1) zusammen mit „jump" und „kill" zu „don't care" führen:

„1" nach „1" durch „store" (j=0 k=0) oder „jump" (j=1 k=0) => (j=X k=0)
„0" nach „0" durch „store" (j=0 k=0) oder „kill" (j=0 k=1) => (j=0 k=X)
„0" nach „1" durch „toggle" (j=1 k=1) oder „jump" (j=1 k=0) => (j=1 k=X)
„1" nach „0" durch „toggle" (j=1 k=1) oder „kill" (j=0 k=1) => (j=X k=1)

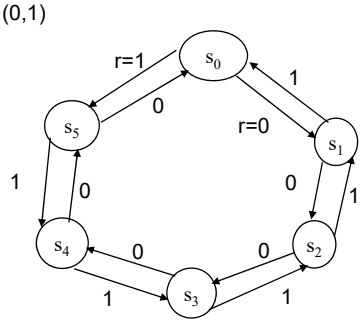
| $z_2$ | $z_1$ | $z_0$ | $j_2$ | $k_2$ | $j_1$ | $k_1$ | $j_0$ | $k_0$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X | 0 | X | 1 | X |
| 0 | 0 | 1 | 0 | X | 1 | X | X | 1 |
| 0 | 1 | 0 | 0 | X | X | 0 | 1 | X |
| 0 | 1 | 1 | 1 | X | X | 1 | X | 1 |
| 1 | 0 | 0 | X | 0 | 0 | X | 1 | X |
| 1 | 0 | 1 | X | 0 | 1 | X | X | 1 |
| 1 | 1 | 0 | X | 0 | X | 0 | 1 | X |
| 1 | 1 | 1 | X | 1 | X | 1 | X | 1 |



## Aufbau eines Modulo-6-vorwärts/rückwärts Zählers mit T-Flipflops

Dieser Zähler soll über ein Eingabesignal r (für rückwärts) so gesteuert werden, daß er aufwärts (r=0) oder abwärts (r=1) zählen kann. Zu seiner Realisierung sollen T-Flipflops verwendet werden, also solche, die bei Eingabe einer 0 den alten Zustand speichern und bei Eingabe einer 1 den Zustand wechseln (toggeln). Die Wertetabelle findet sich auf der nächsten Seite:

Eingabe r = (0,1)




Zustände s = (0,1,2,3,4,5)

Zustandsübergang:
r=0: s‘=(s+1) mod 6
r=1: s‘=(s -1) mod 6
Ungültige Zustände: X

## Wertetabelle eines Modulo-6-vorwärts/rückwärts Zählers mit T-Flipflops

| r | $z_2$ | $z_1$ | $z_0$ | $T_2$ | $T_1$ | $T_0$ | Zustandsübergang: |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | $s_0$ nach $s_1$ |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | $s_1$ nach $s_2$ |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | $s_2$ nach $s_3$ |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | $s_3$ nach $s_4$ |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | $s_4$ nach $s_5$ |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | $s_5$ nach $s_0$ |
| 0 | 1 | 1 | 0 | X | X | X | $s_6$ nach X |
| 0 | 1 | 1 | 1 | X | X | X | $s_7$ nach X |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | $s_0$ nach $s_5$ |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | $s_1$ nach $s_0$ |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | $s_2$ nach $s_1$ |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | $s_3$ nach $s_2$ |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | $s_4$ nach $s_3$ |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | $s_5$ nach $s_4$ |
| 1 | 1 | 1 | 0 | X | X | X | $s_6$ nach X |
| 1 | 1 | 1 | 1 | X | X | X | $s_7$ nach X |

## DMF des Modulo-6-vorwärts/rückwärts Zählers mit T-Flipflops

$T_0 = 1$

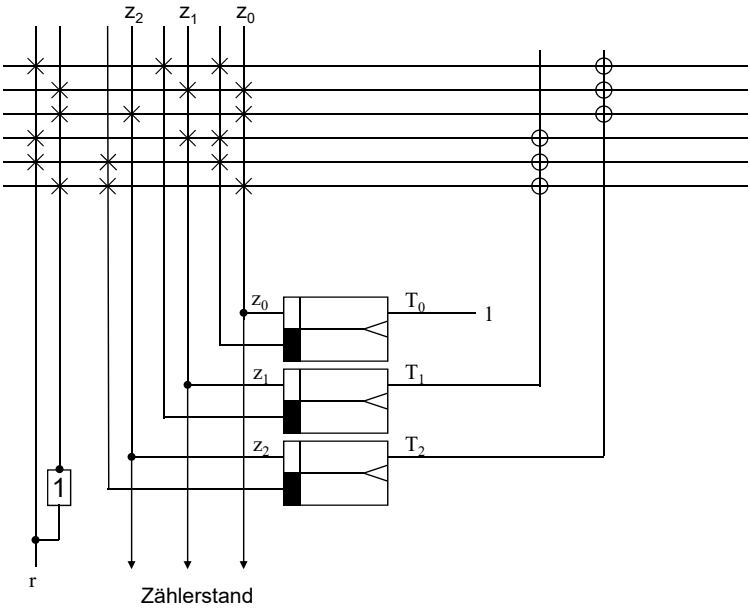
| r $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 0 | X | X |
| 11 | 0 | 0 | 0 | 1 |
| 10 | 1 | 0 | X | X |

$T_1 = \bar{z}_0 z_1 r + \bar{z}_0 \bar{z}_2 r + z_0 \bar{z}_2 \bar{r}$

| r $z_2$ \ $z_1z_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 1 | 0 |
| 01 | 0 | 1 | X | X |
| 11 | 1 | 0 | 0 | 0 |
| 10 | 1 | 0 | X | X |

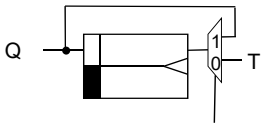
$T_2 = \bar{z}_0 \bar{z}_1 r + z_0 z_1 \bar{r} + z_0 z_2 \bar{r}$

## Realisierung als FPLA:



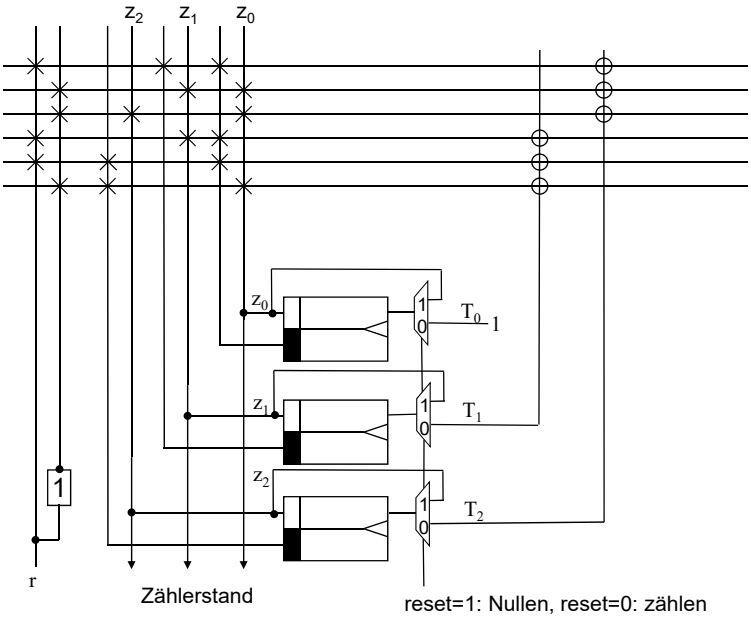
## Null-Setzen eines FPLA mit T-FF

Das T-FF hat keinen Set/Reset-Eingang, kann daher nicht direkt auf Null gesetzt werden. Allerdings kann die Toggle-Funktion ausgenutzt werden: Zum Reset kopple Q zurück auf T ($T=Q_{alt}$).
Wenn $Q_{alt}$ = 1, dann toggelt das FF. Der nächste Takt setzt Q auf 0.
Wenn $Q_{alt}$ = 0, dann speichert das FF, folglich bleibt Q auf 0.




reset=1: toggle auf 0
reset=0: schalte T durch

## Modulo-6-Vorwärts-Rückwärtszähler mit reset, Realisierung als FPLA:




Zählerstand

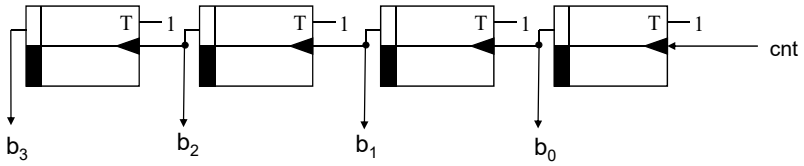
reset=1: Nullen, reset=0: zählen

## Asynchrone Zähler

Alle Schaltwerke, die wir bisher betrachtet haben, waren synchrone Schaltwerke. Das gilt insbesondere für alle Zähler.

Definition: Ein Schaltwerk heißt synchrones Schaltwerk, wenn alle Takteingänge mit einem einzigen, zentral verteilten Signal, genannt Taktsignal, beschaltet werden.

Es gibt viele gute Gründe, Schaltwerke synchron zu betreiben. Speziell bei den Zählern gibt es aber auch gebräuchliche asynchrone Varianten, die sich durch besonders einfachen Aufbau auszeichnen. Man betrachte den folgenden Modulo-16-Zähler aus T-Flipflops. Der Zählertakt wird rechts eingespeist in $cnt_0$ und pflanzt sich nach links durch die Kette an T-FF und erzeugt das 4-bit-Wort $b_3b_2b_1b_0$. Der $T_i$-Eingang liegt an 1, Ausgang $Q_i$ an $cnt_{i+1}$




## Wirkungsweise des asynchronen Zählers

Dieses Schaltwerk zählt die Anzahl der Impulse am Eingang cnt.
Sei der Zähler am Anfang im Zustand 0000. Die vier Flipflops sind negativ flankengetriggert. Nach dem ersten Impuls von cnt wird das rechte erste Flipflop getoggelt, der Zählerstand ist 0001. Alle anderen Flipflops haben noch keine negative Flanke am Takteingang gesehen, halten also ihren Zustand. Beim nächsten cnt-Impuls wechselt das erste Flipflop wieder seinen Wert (diesmal von 1 auf 0). Dadurch bekommt das zweite Flipflop seine erste negative Flanke, es toggelt, der Zählerwert ist 0010. Beim nächsten cnt-Impuls geht der Zähler auf 0011 usw.
Wir sehen, daß die Binärzahlen von 0 bis 15 mod N durchgezählt werden, der Zähler zählt hoch. Beim nächsten Impuls würden nacheinander alle Flipflops den Zustand wechseln, der Zähler fängt wieder bei 0000 an.

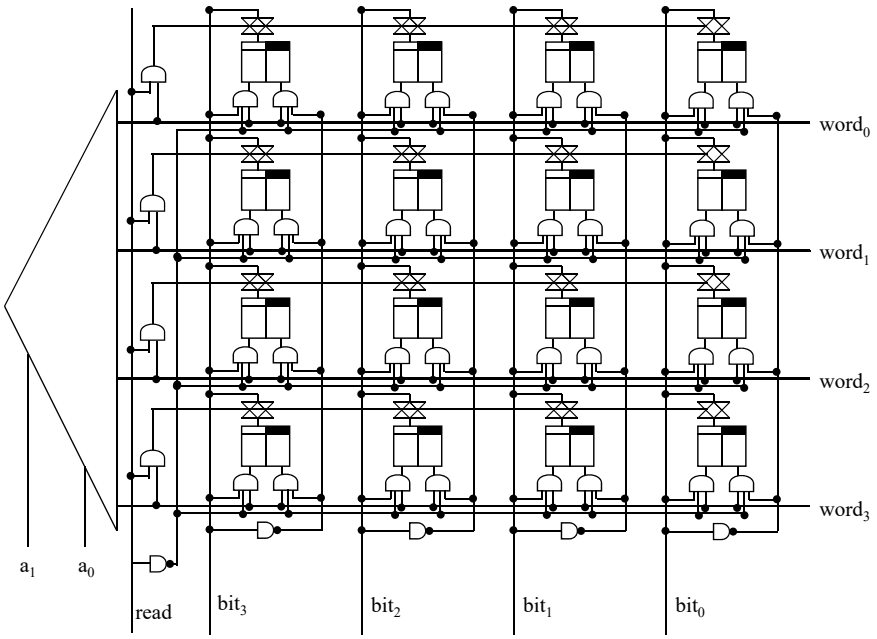
Ein positiv flankengetriggerter asynchroner Zähler zählt herunter, ebenfalls mod N. Natürlich funktioniert ein solcher asynchroner Modulo-N-Zähler genauso für alle Zweierpotenzen N mit n = log N Flipflops.

Problem: durch das asynchrone Schalten entstehen Zwischenzustände (Hazards), die zu Folgefehlern führen können!

## Das RAM

Das Random Access Memory ist ein Speicher für N Worte der Breite m Bit. Jedes dieser Worte ist durch eine Adresse identifizierbar. Die Adresse hat n = log N Bits. Wenn eine Adresse $a_{n-1}a_{n-2}...a_1a_0$ angelegt wird, kann auf das zugehörige Wort lesend oder schreibend zugegriffen werden. Zu diesem Zweck legt man die Adresse an einem Dekodierer an, der sie in einen 1-aus-N-Code dekodiert. Für jedes der N Worte gibt es nun eine sogenannte word-Leitung. Durch den Dekodiervorgang wird auf die gesuchte word-Leitung eine 1 gelegt, auf alle anderen eine 0. An jeder dieser word-Leitungen liegt nun ein Register der Breite m Bit. Durch das aktivieren der word-Leitung kann dieses Register nun von außen gelesen oder beschrieben werden.

Ein einfacher Aufbau eines RAM mit N = 4 und m = 4 ist auf der nächsten Folie dargestellt. Die Speicherbausteine sind r-s-Flipflops. Jeweils vier davon sind zu einem parallelen Wortspeicher zusammengeschaltet.



## Die Funktionsweise des RAM

Beim Schreiben wird an die Bit-Leitungen ein m-Bit-Wort angelegt. Dieses soll in die Zeile i geschrieben werden. Die Adresse i liegt am Decoder an. Dadurch wird die i-te word-Leitung auf 1 gelegt. Die And-Gatter vor den Flipflops der i-ten Zeile lassen somit am s-Eingang jedes r-s-Flipflops den Bit-Wert und am r-Eingang den invertierten Bit-Wert durch. Alle anderen word-Leitungen sind auf 0, d.h. die And-Gatter legen an alle anderen r-s-Flipflops die Speicherkombination r=0 und s=0 an. Somit wird in Zeile i das neue Wort „eingespeichert“. Alle anderen Zeilen speichern die alten Werte.

Beim Lesen wird an die Bit-Leitungen von außen nichts angelegt, sie sind im hochohmigen Zustand Z. Durch die Word-Leitung werden nun die Transmission-Gates der Ausgang der Flipflops in der i-ten Zeile geöffnet, die gespeicherten Werte gelangen auf die Bit-Leitungen. Somit werden die Werte der i-ten Zeile an den Ausgang des RAM transportiert.

In der Realität werden RAMs nicht aus r-s-Flipflops und And-Gattern und Transmission-Gates aufgebaut, da diese Realisierung zu aufwendig wäre. Man unterscheidet zunächst zwischen SRAM (statischem RAM) und DRAM (dynamischem RAM). Das statische RAM speichert einen Wert und hält diesen, solange die Versorgungsspannung eingeschaltet bleibt. Unser RAM aus r-s-Flipflops ist ein Beispiel für eine Realisierung eines statischen RAM.

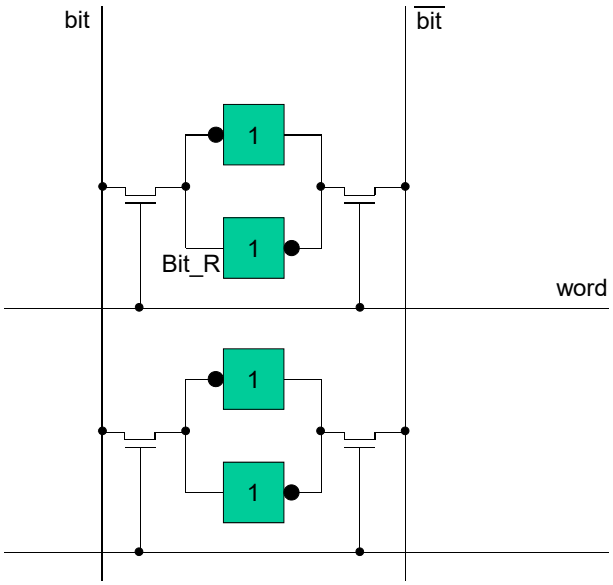
## DRAM

Das dynamische RAM speichert alle Werte nur für eine kurze Zeit. Und zwar wird als Speichermedium die Kapazität eines Transistor-Gates (polychristallines Silizium) gegenüber der Source/Drain (Diffusion) ausgenutzt. Natürlich ist ein solcher Speicher flüchtig. Daher muß jedes Bit in einem dynamischen RAM von Zeit zu Zeit aufgefrischt (refresh) werden. Das geschieht, indem automatisch in festen Zeitabständen zeilenweise alle Bits im RAM einmal gelesen und unverändert wieder geschrieben werden. Ein typisches Zeitintervall für den Refresh-Zyklus bei heutigen DRAMs ist eine Millisekunde.

SRAM ist schneller (kürzere Zugriffszeit) und teurer. Außerdem benötigt SRAM 6 Transistoren pro gespeichertem Bit, während DRAM mit einem Transistor pro Bit auskommt. Daher hat DRAM eine höhere Speicherkapazität pro Chipfläche.

Heutige Speicherchips sind quadratisch angeordnet. Es werden zwei Dekodierer verwendet, einen für die Zeile und einen für die Spalte. Zeilen- und Spaltenadresse sind dabei gleich lang. Auf diese Weise kann man mit der Hälfte der Adresspins auskommen, indem man die Adressleitungen im Zeitmultiplex verwendet. Immer wird zuerst die Zeilenadresse übertragen und dann über dieselben Pins die Spaltenadresse.

## Aufbau einer 6-Transistor SRAM-Zelle




## SDRAM

Eine Technik, mit der DRAMs schneller gemacht werden können sind die sogenannten SDRAMs (synchronous DRAMs). Es handelt sich im Prinzip um DRAM, bei dem über eine externe Taktung die Synchronisation an die maximale Geschwindigkeit des Prozessor-Speicher-Busses erzwungen wird.

RAMBUS ist eine vorwiegend für Intel-Prozessoren eingesetzte asynchrone Speichertechnik, bei der durch haargenaue Abstimmung der durch Kapazitäten, Induktivitäten, Widerstände verursachten Laufzeiten die Performance optimiert wurde. Diese Technologie ist aber gegenwärtig wieder auf dem Rückzug.

CDRAM (Cache DRAM) ist eine Kombination aus SRAM und DRAM. Es handelt sich im Prinzip um DRAM, bei dem aber die Zeile, aus der zuletzt gelesen wurde in einem kleinen separaten SRAM (Cache) gehalten wird. Da beim Zugriff auf den Speicher häufig mehrmals hintereinander auf dieselbe Zeile zugegriffen wird, kann jeder Zugriff mit der Geschwindigkeit des SRAM bedient werden. Trotzdem kann die hohe Speicherdichte des DRAM ausgenutzt werden.

Austin Semiconductor, Inc.

**SRAM**
**MT5C2564**

## 64K x 4 SRAM
SRAM MEMORY ARRAY

**AVAILABLE AS MILITARY SPECIFICATIONS**
- SMD 5962-88681
- MIL-STD-883

**FEATURES**
- High Speed: 15, 20, 25, 35, 45, 55, and 70
- Battery Backup: 2V data retention
- Low power standby
- High-performance, low-power, CMOS double-metal process
- Single +5V (±10%) Power Supply
- Easy memory expansion with CE\
- All inputs and outputs are TTL compatible

**OPTIONS** | **MARKING**

- **Timing**
  - 15ns access: -15
  - 20ns access: -20
  - 25ns access: -25
  - 35ns access: -35
  - 45ns access: -45
  - 55ns access: -55*
  - 70ns access: -70*
- **Package(s)**
  - Ceramic DIP (300 mil): C — No. 106
  - Ceramic LCC: EC — No. 204
- **Operating Temperature Ranges**
  - Industrial (-40°C to +85°C): IT
  - Military (-55°C to +125°C): XT
- 2V data retention/low power: L

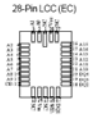
*Electrical characteristics identical to those provided for the 45ns access devices.

**For more products and information please visit our web site at *www.austinsemiconductor.com***

**PIN ASSIGNMENT**
(Top View)

24-Pin DIP (C) (300 MIL)

28-Pin LCC (EC)



### GENERAL DESCRIPTION

The Austin Semiconductor SRAM family employs high-speed, low-power CMOS and are fabricated using double-layer metal, double-layer polysilicon technology.

For flexibility in high-speed memory applications, Austin Semiconductor offers chip enable (CE\) on all organizations. This enhancement can place the outputs in High-Z for additional flexibility in system design. The x4 configuration features common data input and output.

Writing to these devices is accomplished when write enable (WE\) and CE\ inputs are both LOW. Reading is accomplished when WE\ remains HIGH and CE\ goes LOW. The device offers a reduced power standby mode when disabled. This allows system designs to achieve low standby power requirements.

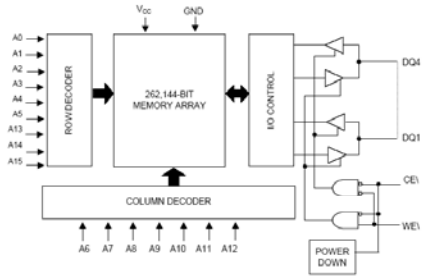
These devices operate from a single +5V power supply and all inputs and outputs are fully TTL compatible.




Austin Semiconductor, Inc.

SRAM
MT5C2564

### FUNCTIONAL BLOCK DIAGRAM

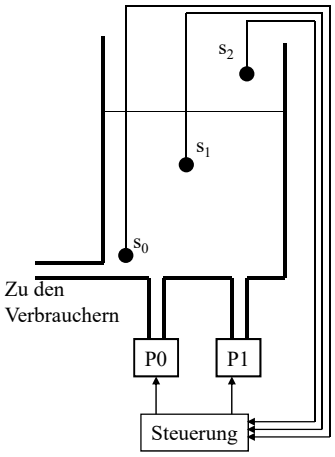



**TRUTH TABLE**

| MODE | CE\ | WE\ | DQ | POWER |
|---|---|---|---|---|
| STANDBY | H | X | HIGH-Z | STANDBY |
| READ | L | H | Q | ACTIVE |
| WRITE | L | L | D | ACTIVE |

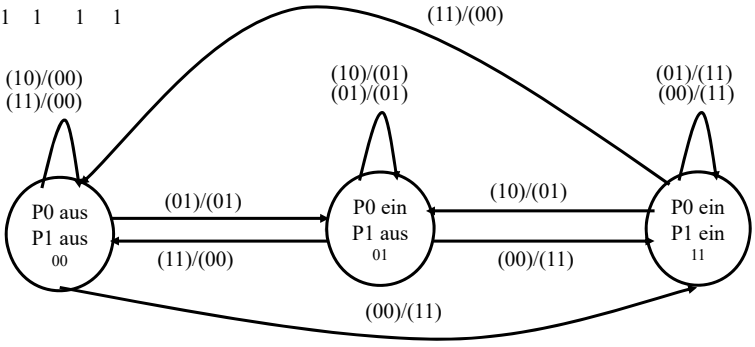
# Beispiel: Eine Pumpensteuerung

Ein Wasserturm soll mit zwei Pumpen aus einem Wasserwerk befüllt werden. Dabei sollen die Pumpen den Wasserstand immer zwischen den Markierungen $s_0$ und $s_2$ halten, wobei das Verhalten der Verbraucher unvorhersehbar ist. Sicher ist nur, dass beide Pumpen zusammen mehr liefern, als in der gleichen Zeit von den Verbrauchern entnommen wird. Nun sollen folgende Bedingungen erfüllt werden: Wenn der Wasserstand unter den Sensor $s_0$ fällt, sollen beide Pumpen eingeschaltet werden. Wenn er über $s_2$ steigt, sollen beide Pumpen ausgeschaltet werden. Wenn er über $s_1$ steigt, soll nur noch höchstens eine Pumpe arbeiten. Wenn er unter $s_1$ sinkt, soll mindestens eine Pumpe arbeiten.
Eine Pumpe, die durch Sinken des Füllstandes unter den Sensor $s_i$ eingeschaltet wird, soll aber frühestens bei erreichen des Füllstandes am Sensor $s_{i+1}$ wieder ausgeschaltet werden, um nicht ständig an- und abzuschalten, wenn der Füllstand leicht um die Höhe eines Sensors variiert.




Eingaben: $(s_2, s_1, s_0)$ Ausgaben (P1, P0), durch Codierung der Eingaben mit 2 Bit kann man die einzigen möglichen Fälle abdecken: $(x_1, x_0)$; Codierung der Zustände: $(P_0, P_1)$

| $s_2$ | $s_1$ | $s_0$ | $x_1$ | $x_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | X | X |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | X | X |
| 1 | 0 | 1 | X | X |
| 1 | 1 | 0 | X | X |
| 1 | 1 | 1 | 1 | 1 |

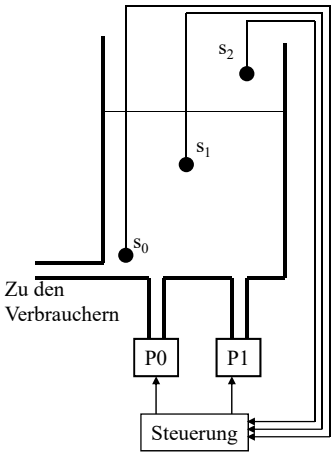


| $x_1$ | $x_0$ | P1 | P0 | P1‘ | P0‘ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | X | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | X | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | X | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | X | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

( X 1 ist immer ein definierter Zustand)

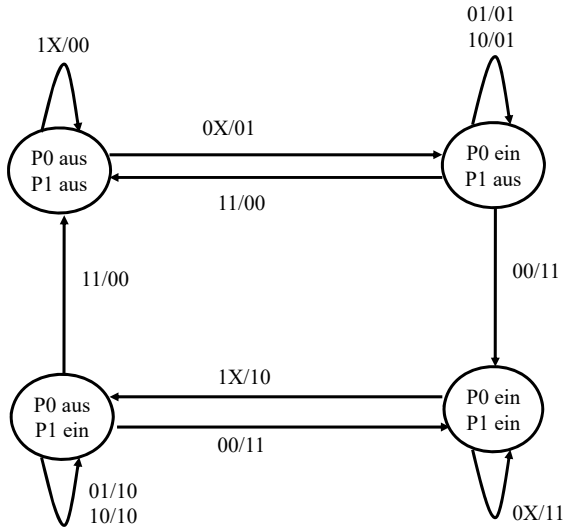
# Beispiel: Eine verbesserte Pumpensteuerung

Ein Wasserturm soll mit zwei Pumpen aus einem Wasserwerk befüllt werden. Dabei sollen die Pumpen den Wasserstand immer zwischen den Markierungen $s_0$ und $s_2$ halten, wobei das Verhalten der Verbraucher unvorhersehbar ist. Sicher ist nur, dass beide Pumpen zusammen mehr liefern, als in der gleichen Zeit von den Verbrauchern entnommen wird. Nun sollen folgende Bedingungen erfüllt werden: Wenn der Wasserstand unter den Sensor $s_0$ fällt, sollen beide Pumpen eingeschaltet werden. Wenn er über $s_2$ steigt, sollen beide Pumpen ausgeschaltet werden. Wenn er über $s_1$ steigt, soll nur noch eine Pumpe arbeiten. Dies soll P0 sein, falls beide Pumpen eingeschaltet sind. Wenn er unter $s_1$ sinkt, soll P1 eingeschaltet werden, falls beide Pumpen aus sind (um die Pumpen etwa gleichstark abzunutzen).




Eine Pumpe, die durch Sinken des Füllstandes unter den Sensor $s_i$ eingeschaltet wird, soll aber frühestens bei erreichen des Füllstandes am Sensor $s_{i+1}$ wieder ausgeschaltet werden, um nicht ständig an- und abzuschalten, wenn der Füllstand leicht um die Höhe eines Sensors variiert.

Eingaben: $(s_2, s_1, s_0)$ Ausgaben (P1, P0)
Codierung der Eingaben: $x_1x_0$: 00: $s_0$=0
01: $s_0$=1, $s_1$=0
10: $s_1$=1, $s_2$=0
11: $s_2$=1




| P1 | P0 | $x_1$ | $x_0$ | P1‘ | P0‘ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 |