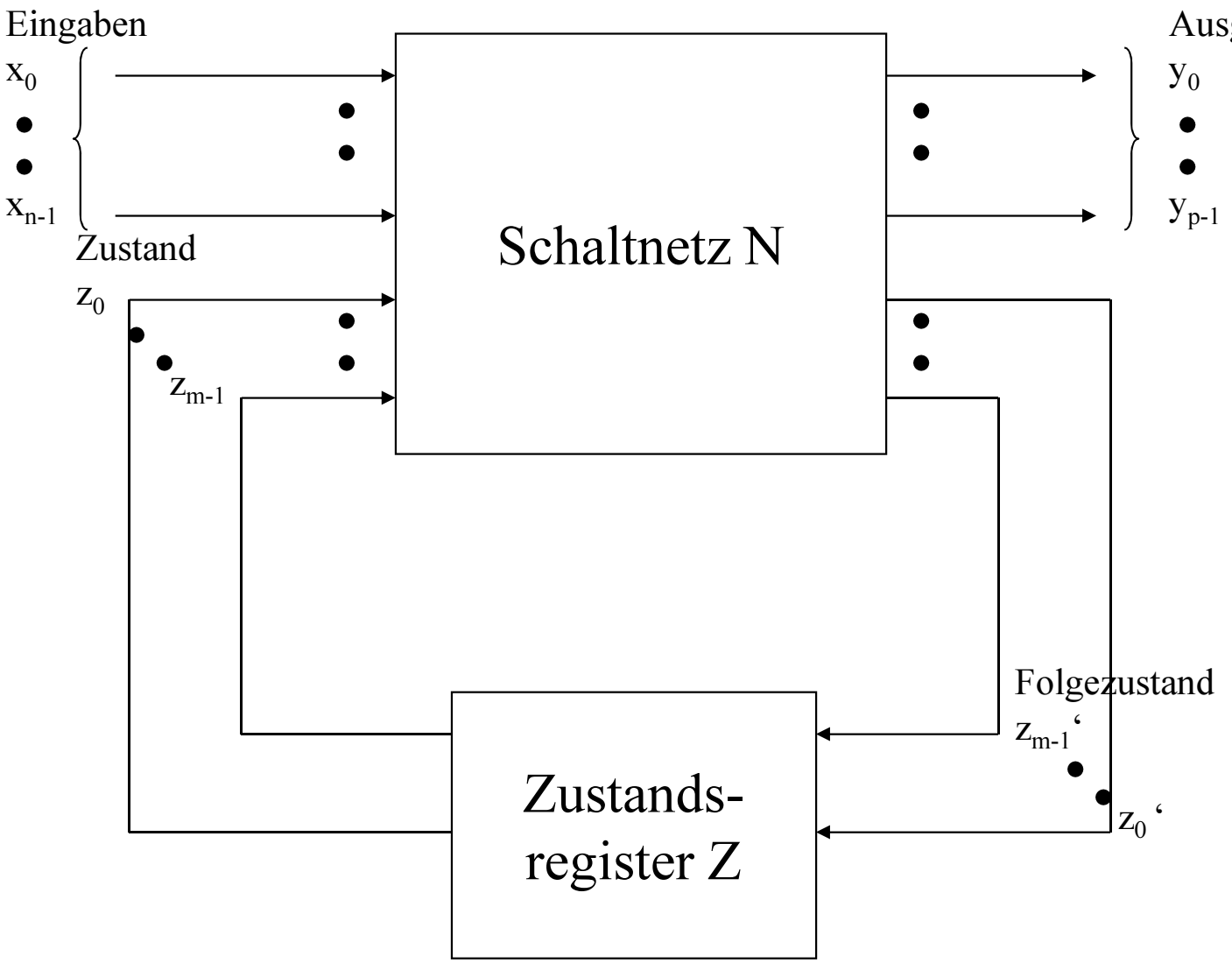
Schaltwerke

Bisher habe wir uns nur mit Schaltnetzen befasst, also Schaltungen aus Gattern, die die Ausgaben als eine Funktion der Eingaben unmittelbar (durch Schaltvorgänge) berechnen. Diese Schaltnetze (engl. combinatorial circuits) haben keine Zustände also kein Gedächtnis, in dem frühere Schaltvorgänge eingespeichert sind. Schaltnetze können nicht abhängig von einem gespeicherten Zustand auf die eine oder andere Weise auf die Eingaben reagieren. Dies ist aber bei einer elektronischen Schaltung wie einem Prozessor eines Computers erwünscht. Wir wollen uns daher jetzt mit Schaltwerken (engl. sequential circuits) beschäftigen, die man als technische Realisierung von endlichen deterministischen Automaten (finite state machines) aus der theoretischen Informatik betrachten kann.

Definition: Eine Schaltung, deren Ausgänge von der Belegung der Eingänge und ihrem inneren Zustand abhängt, wird ein **Schaltwerk** genannt.

Definition:

Eine Schaltung, deren Ausgänge von der Belegung der Eingänge und ihrem inneren Zustand abhängt, wird ein Schaltwerk genannt.


Eingaben
x_0
x_{n-1}
Zustand
z_0
z_{m-1}
Schaltnetz N
Ausgaben
y_0
y_{p-1}
Folgezustand
z_{m-1}‘
z_0 ‘
Zustands-
register Z

Jedes Schaltwerk enthält Speicherelemente, die den inneren Zustand speichern. Der aktuelle innere Zustand Z und die Belegung der Eingänge X bestimmen den Folgezustand Z' und die Ausgänge Y. Man kann sich die - beliebig in der Schaltung verteilten - Speicherelemente zu einem sogenannten **Zustandsregister** zusammengefasst denken.

Definition: Ein Speicherelement, das die beiden Werte 0 und 1 speichern (annehmen) kann, heißt **Flipflop**.

Der **Zustand** Z eines Schaltwerks ist ein binäres m-Tupel $(z_{m-1}, z_{m-2},...,z_1, z_0)$ aus Komponenten $z_i \in B = \{ 0, 1 \}$. Jede Komponente steht dabei für ein Flipflop.

Bevor wir auf die Realisierung dieser Flipflops eingehen, wollen wir uns ein Beispiel ansehen: Zu entwerfen ist eine Schaltung, die einen Strom aus eingehenden Bits empfängt und interpretiert. Pro Takt kommt über eine Leitung x ein Bit. Wenn hintereinander drei Einsen empfangen worden sind, soll am Ausgang eine Lampe y angehen. Also y ist 1, wenn die letzten drei empfangenen Bits 1 waren, sonst ist y 0. Ein Schaltnetz wäre mit dieser Aufgabe überfordert, da es kein Gedächtnis hat, in dem es beispielweise den vorletzten Wert speichern kann.

Um nun unser Schaltwerk zu bauen, veranschaulichen wir uns zunächst die Funktion anhand eines Schaltwerksgraphen (Automatengraphen).

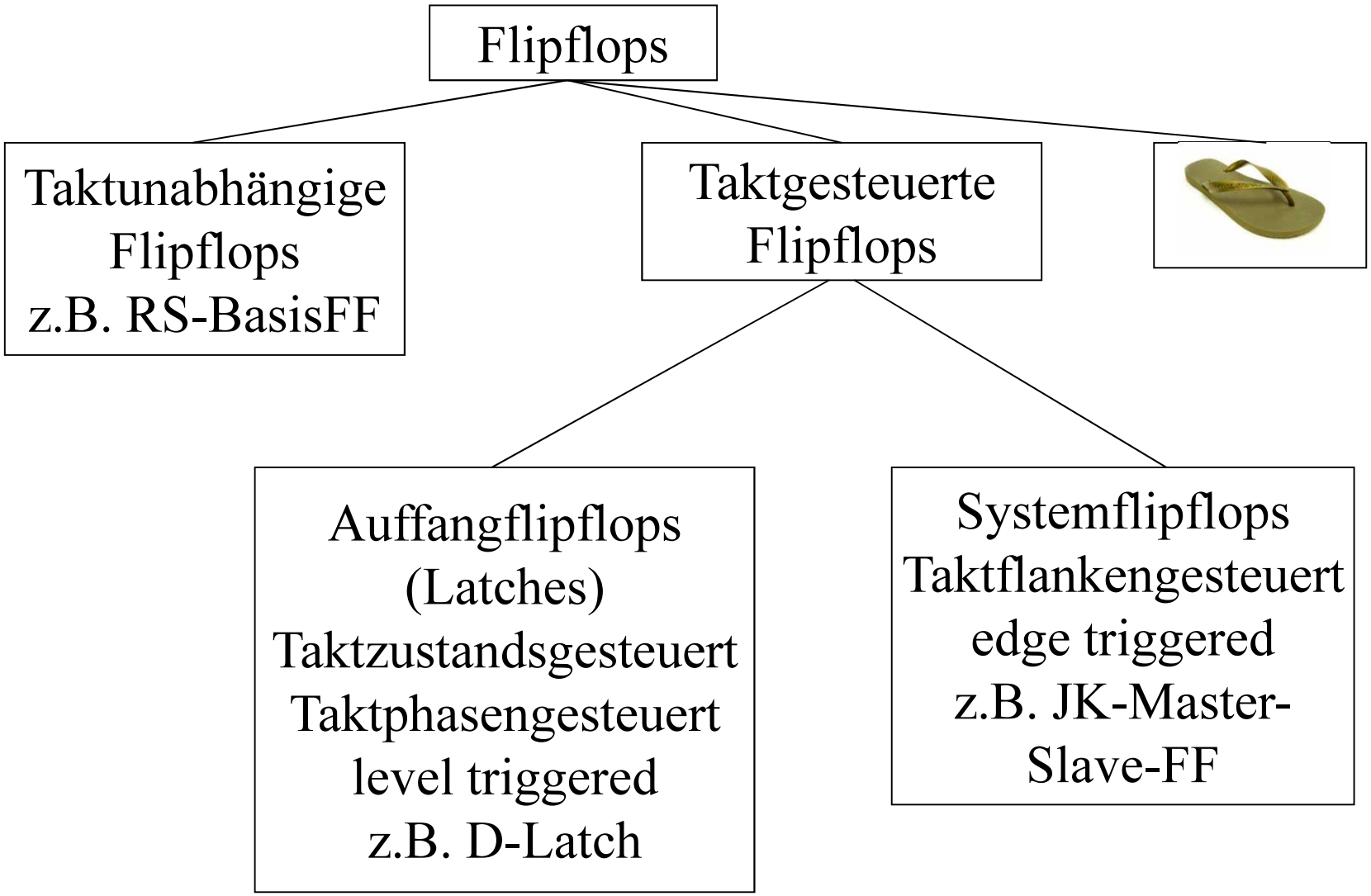
Definition:

Ein Speicherelement, das die beiden Werte 0 und 1 speichern (annehmen) kann, heißt Flipflop.

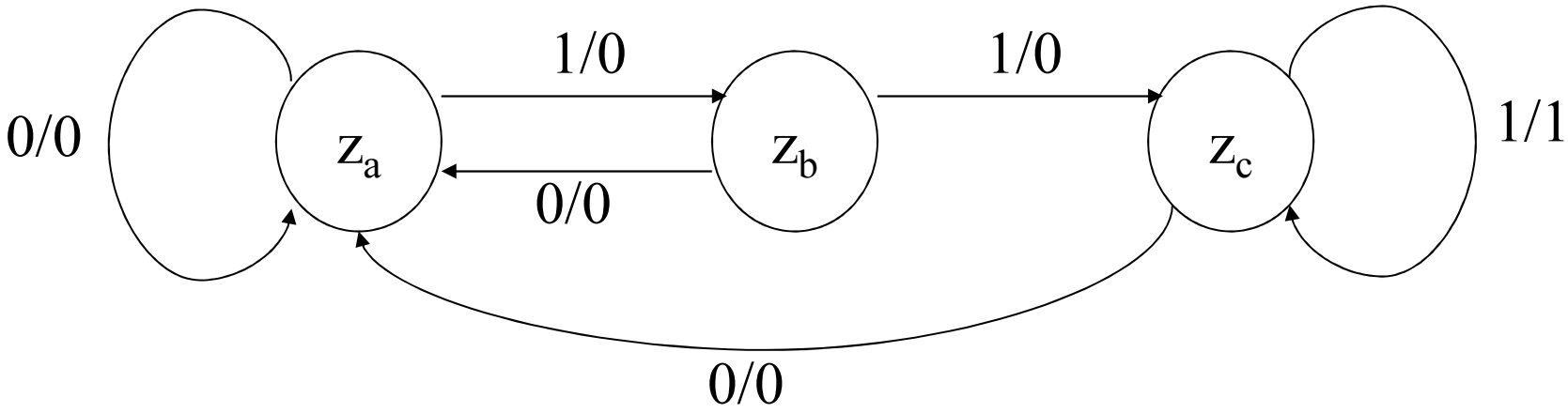
Ein Flipflop kann durch geeignete Ansteuerung von jedem seiner Zustände in den anderen übergehen.

Definition:

Der Zustand Z eines Schaltwerks ist ein m-Tupel $(z_{m-1}, z_{m-2},\ldots,z_1,z_0)$ aus Komponenten $z_i \in B=\{0,1\}$. Jede Komponente z_i steht dabei für ein Flipflop.


Flipflops
Taktunabhängige Flipflops z.B. RS-BasisFF
Taktgesteuerte Flipflops
Auffangflipflops (Latches) Taktzustandsgesteuert Taktphasengesteuert level triggered z.B. D-Latch
Systemflipflops Taktflankengesteuert edge triggered z.B. JK-Master-Slave-FF

Betrachten wir nun folgende Aufgabe. Ein Fließband hat eine Qualitätskontrolle eingebaut. Durch ein Verfahren wird für die dort nacheinander vorbeilaufenden Werkstücke ein Signal erzeugt, das das Werkstück als ok klassifiziert (x=1) oder als fehlerhaft (x=0). Wenn drei oder mehrere Werkstücke hintereinander fehlerhaft sind, soll eine Warnleuchte y aufleuchten.
Die an x generierten Signale kann man als bit-seriell ankommenden Strom von Eingabebits interpretieren, das Leuchten der Lampe als Ausgabebit.
Offenbar ist diese Aufgabe nicht mehr von einem Schaltnetz lösbar, denn wenn das zweite Werkstück eine Information für x liefert, ist das erste Bit bereits vergessen, da ein Schaltnetz kein „Gedächtnis“ hat. Stattdessen benötigen wir ein Schaltwerk.
Der folgende Graph zeigt das Verhalten eines solchen Schaltwerks. Die Kreise beschreiben die möglichen Zustände, in denen ein weiteres Bit gelesen wird. Das Gedächtnis besteht also in dem jeweiligen Zustand, in dem sich das Schaltwerk befindet.



Dieser Graph ist folgendermaßen zu interpretieren: Jeder Kreis ist ein möglicher Zustand, jeder Pfeil ein möglicher Zustandsübergang. An einem Pfeil steht eine Beschriftung vom Typ Eingabe/Ausgabe, d.h. bei Eingabe des Wertes vor dem / im Zustand, in dem der Pfeil beginnt, wird der Wert nach dem / ausgegeben und in den Zustand, auf den der Pfeil zeigt, gewechselt.

In unserem Falle beginnen wir im Zustand z_a. Bei Eingabe einer 0 bleiben wir in diesem Zustand und wir geben eine 0 aus (y=0, die Lampe leuchtet nicht). Bei Eingabe einer 1 wechseln wir in den Zustand z_b und geben eine 0 aus, denn bisher ist nur eine 1 eingegeben worden. Wenn nun eine 0 eingegeben wird, gehen wir wieder zurück in den Zustand z_a und geben eine 0 aus.

Wenn wir in z_b eine 1 eingeben, haben wir bereits zwei 1en hintereinander gesehen. Wir gehen in den Zustand z_c und geben eine 0 aus. Nun betrachten wir z_c. Wenn wir in diesem Zustand sind, haben wir als letzte beide Eingaben eine 1 gehabt. Wenn jetzt noch eine 1 kommt, dann muss die Lampe leuchten, d.h. unsere Ausgabe y muss auf 1 gehen. Aber welches ist in diesem Fall der Folgezustand? Nun, wiederum z_c, denn auch hier waren die beiden letzten Eingaben 1en. Wenn in z_c eine 0 eingegeben wird, gehen wir wieder mit Ausgabe 0 in z_a zurück, denn eine Folge von drei 1en müsste ganz neu beginnen.
Der nächste Schritt ist die Codierung der Eingaben, Ausgaben und Zustände als Binärzahlen. Nun, x und y können nur zwei Werte annehmen, daher sind sie bereits binär codiert. Für die Zustände wählen wir folgende Codierung: $z_a = 00$, $z_b = 01$, $z_c = 10$. Diese beiden Bits bezeichnen wir mit z_1 und z_0.
Jetzt können wir die Wertetabelle für das erforderliche Schaltnetz aufstellen:

x	z_1	z_0	z_1‘	z_0‘	y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	X	X	X
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	X	X	X

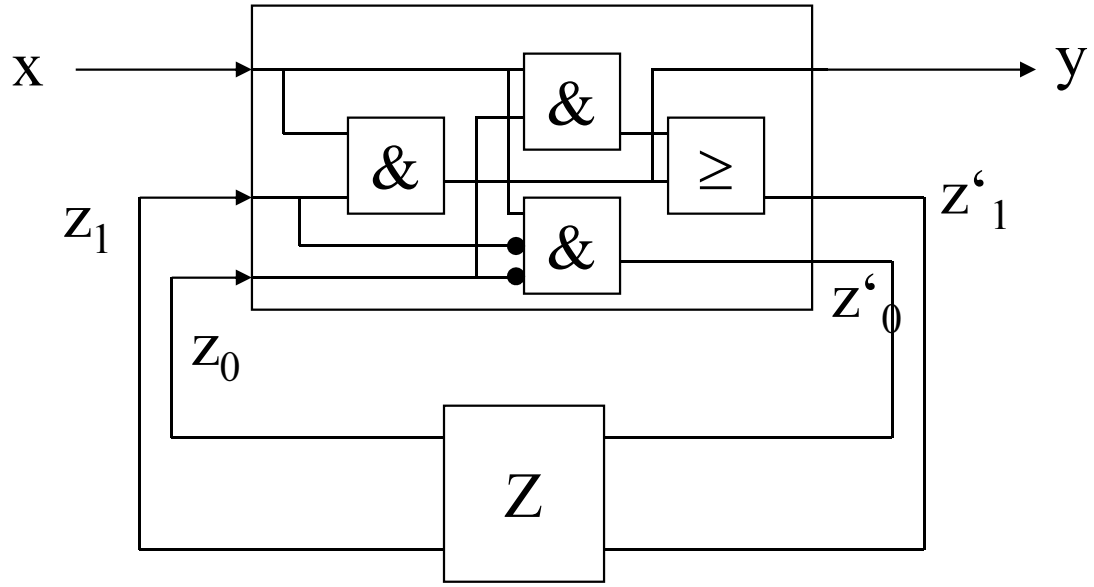
Die Anwendung von KV-Diagrammen führt uns zu folgenden disjunktiven Minimalformen für die z_i‘ und für y:

$z_0' = x\,\bar{z}_0\,\bar{z}_1$

$z_1' = x\,z_0 + x\,z_1$

$y = x\,z_1$

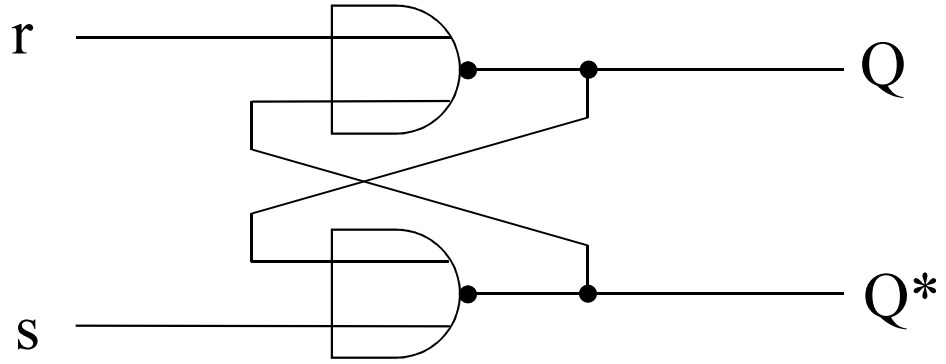
Das gesamte Schaltwerk kann also folgendermaßen realisiert werden:



Was uns nun noch fehlt, ist die Realisierung des Zustandsregisters Z. Wie sieht überhaupt ein Baustein aus, der ein Bit speichern kann, also das Flipflop?
Im folgenden werden wir einige verschiedene Flipfloptypen kennen lernen. Wir beginnen mit einfachen Flipflops, die wir dann mit zusätzlichen Funktionen ausstatten bis hin zu den sogenannten System-Flipflops, die wir schließlich für den Schaltwerksaufbau verwenden können.

Das r-s-Flipflop

Das einfachste Speicherelement ist das r-s-Flipflop. Das r steht für rücksetzen (reset), das s für setzen (set). Es kann mit zwei rückgekoppelten Nor-Gattern realisiert werden:

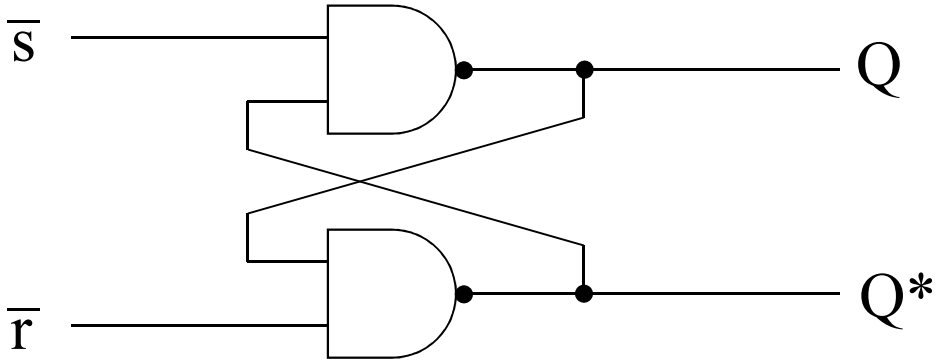


r	s	Q	Q*
0	0	Q_{alt}	Q^*_{alt}
0	1	1	0
1	0	0	1
1	1	0	0

Wenn r = 0 und s = 0 ist, speichert das Flipflop seinen alten Wert am Ausgang. Wenn r = 0 und s = 1 ist, wird es gesetzt (Q := 1); bei r = 1 und s = 0 wird es rückgesetzt (Q = 0).

Die Eingabe r = 1 und s = 1 ist verboten. Am Ausgang würde sich nämlich bei dieser Beschaltung der Zustand Q = 0 und Q* = 0 einstellen. Bei einer direkt darauf folgenden Eingabe r = 0 und s = 0 würde das Flipflop in einen instabilen Zustand geraten.

Das r-s-Flipflop kann auch mit NAND-Gattern aufgebaut werden. Bis auf die verbotene Eingabe ist die Wertetabelle identisch mit der des NOR-Flipflops (weil r und s invertiert eingehen).

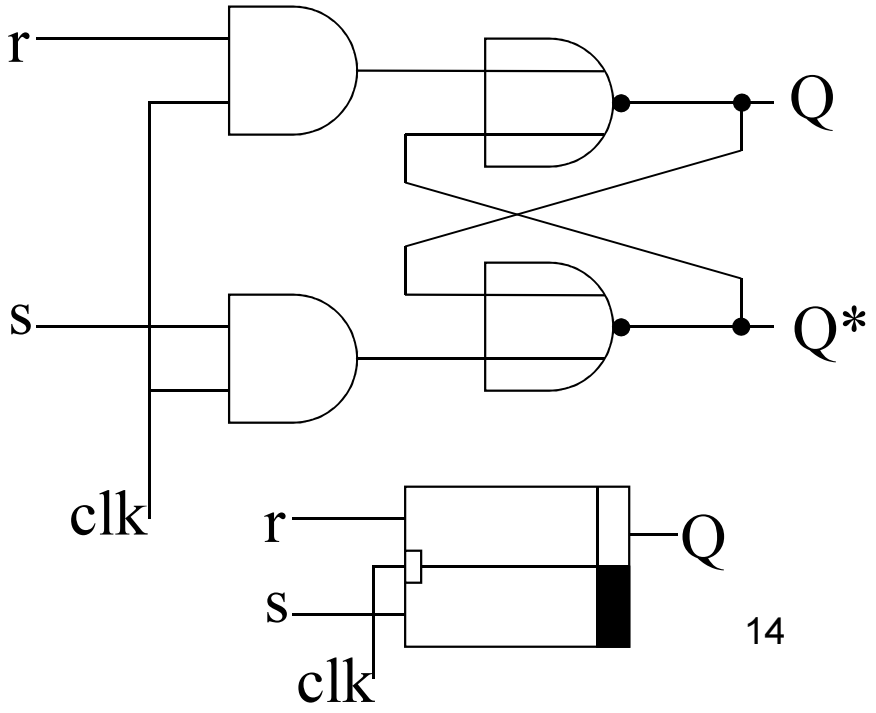


$\overline{r}$	$\overline{s}$	Q	Q*
1	1	Q_{alt}	Q^*_{alt}
1	0	1	0
0	1	0	1
0	0	1	1

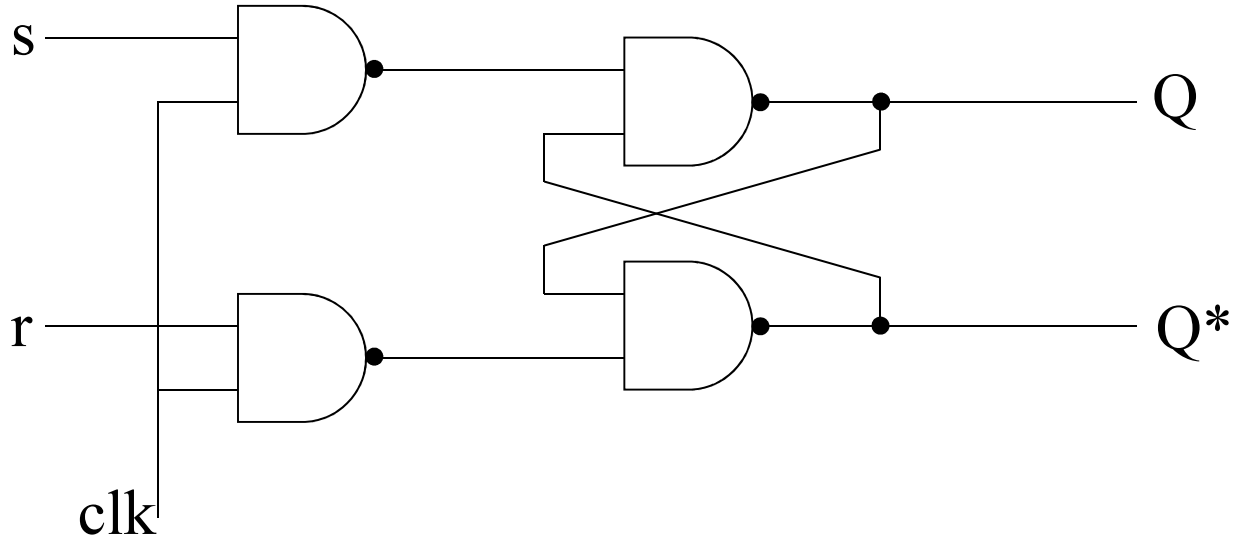
Ein solches Basisflipflop übernimmt beim Setzen oder Rücksetzen den Wert sofort an den Ausgang. Häufig ist diese Funktion aber nicht erwünscht, sondern man möchte den Zustand des Flipflops nur zu definierten Zeiten ändern. Zu diesem Zweck macht man das Übernehmen des Wertes abhängig von einem weiteren Eingang, z.B. einem Takteingang: Solange der Takt 0 ist, soll das Flipflop seinen alten Wert speichern. Wenn der Takt jedoch 1 ist, soll es gesetzt oder rückgesetzt werden können, es soll aber auch bei r=s=0 seinen alten Wert erhalten können. r=s=1 ist wiederum verboten.

Wertetabelle:

clk	r	s	Q	Q*
0	0	0	Q_{alt}	Q^*_{alt}
0	0	1	Q_{alt}	Q^*_{alt}
0	1	0	Q_{alt}	Q^*_{alt}
0	1	1	Q_{alt}	Q^*_{alt}
1	0	0	Q_{alt}	Q^*_{alt}
1	0	1	1	0
1	1	0	0	1
1	1	1	verboten	



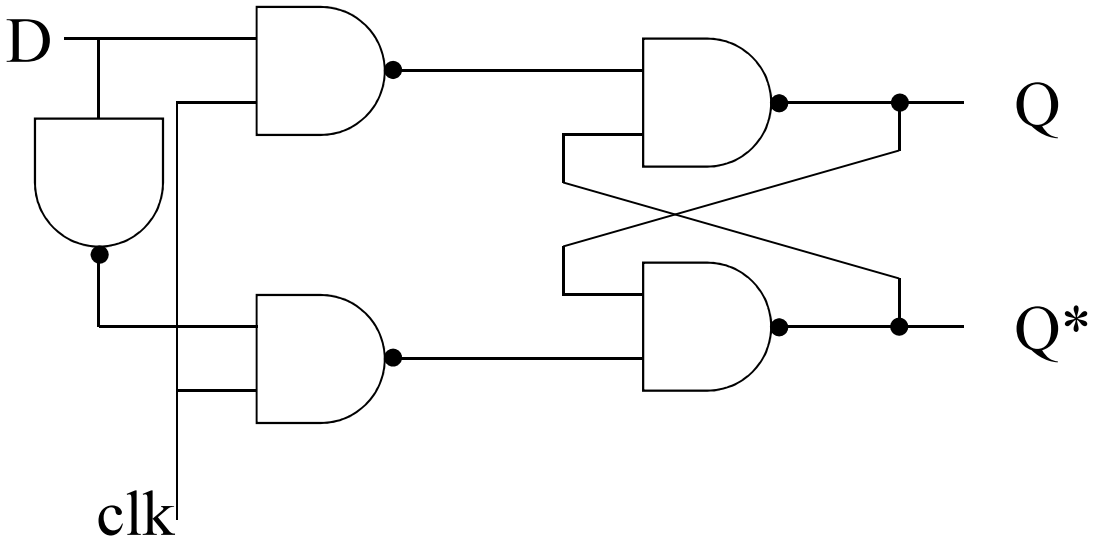
Dieses getaktete r-s-Flipflop nennt man auch ein r-s-Auffangflipflop (r-s-Latch). Es kann nämlich während der positiven Taktphase (also während clk=1 ist) einen Wert aufnehmen, der dann während der negativen Taktphase (clk=0) gespeichert bleibt.Man kann ein Latch natürlich wieder in NAND-Logik aufbauen.



Nun sind wir häufig lediglich daran interessiert einen Eingabewert, also ein Bit, unverändert zu übernehmen und zu speichern. Dann brauchen wir nicht die Funktionalität eines r-s-Flipflops, das ja immer den Nachteil der verbotenen Eingabekombination hat sondern ein D-Flipflop. D steht für delay. Wir sehen hier die Wertetabelle und eine mögliche Realisierung mit einem NAND-Basisflipflop.

Wertetabelle:

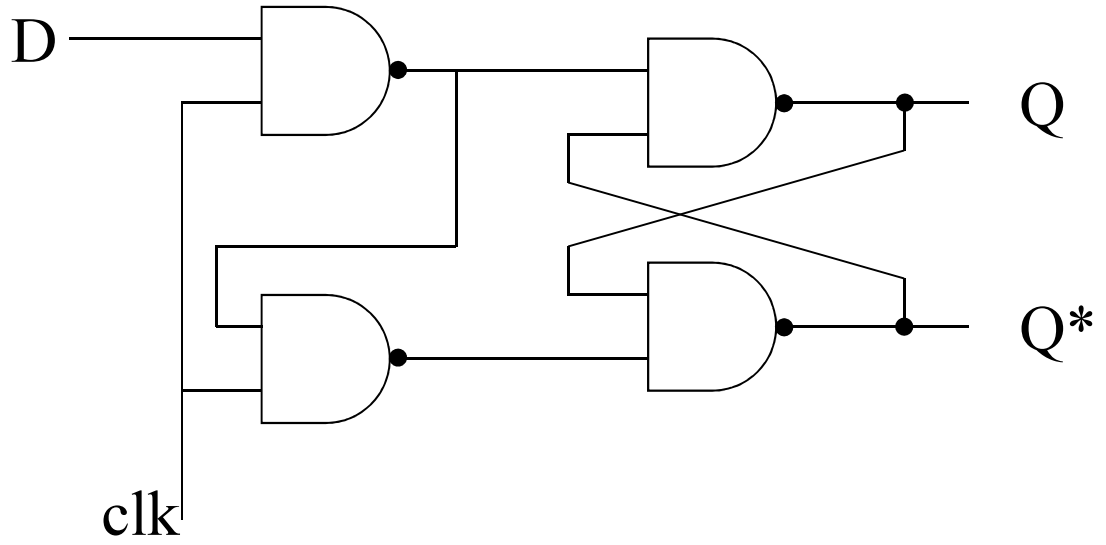
clk	D	Q	Q*
0	0	Q_{alt}	Q^*_{alt}
0	1	Q_{alt}	Q^*_{alt}
1	0	0	1
1	1	1	0



Man kann den Inverter einsparen, indem man das Layout geringfügig verändert. In dieser Version muß aber der Taktimpuls mindestens so lang sein wie eine Schaltzeit eines Nand-Gatters, da sonst bei clk = 1 das invertierte D sich nicht mehr auf dem unteren NAND-Gatter auswirken kann.

Wertetabelle:

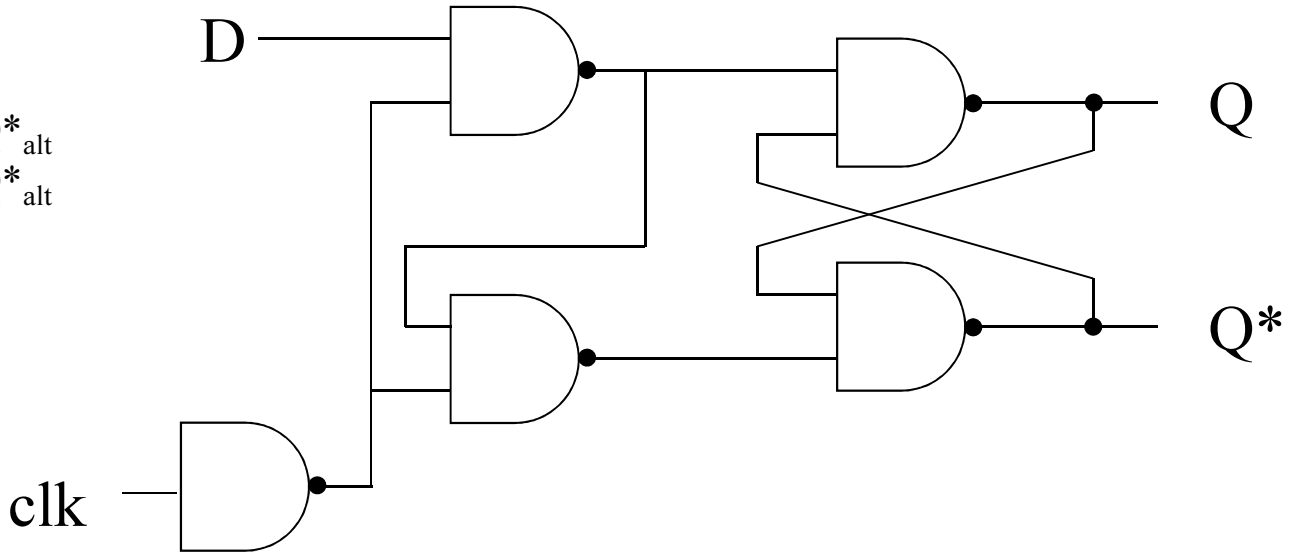
clk	D	Q	Q*
0	0	Q_{alt}	Q^*_{alt}
0	1	Q_{alt}	Q^*_{alt}
1	0	0	1
1	1	1	0



Alle bisher betrachteten Flipflops übernahmen die Eingangswerte, während der Takt auf 1 war und speicherten, wenn der Takt auf 0 war. Man nennt dies ein **positiv levelgesteuertes Auffangflipflop**. Man kann aber jetzt natürlich durch Inversion des Taktsignals ein negativ levelgesteuertes Auffangflipflop bauen, das bei clk = 0 übernimmt und bei clk = 1 speichert:

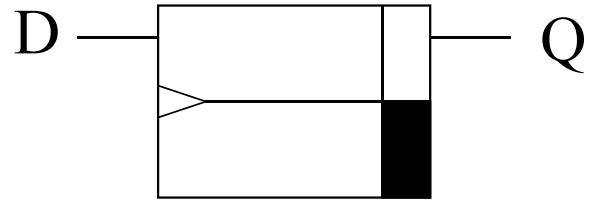
Wertetabelle:

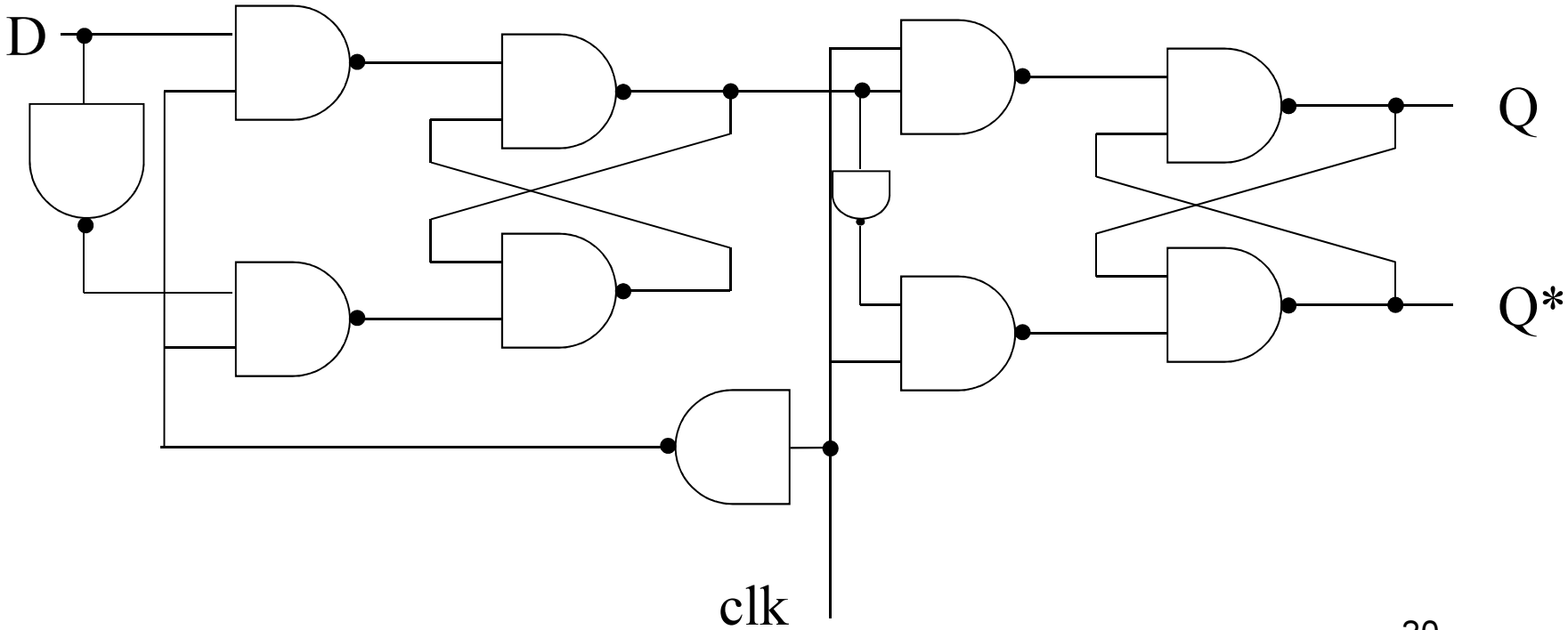
clk	D	Q	Q*
0	0	0	1
0	1	1	0
1	0	Q_{alt}	Q^*_{alt}
1	1	Q_{alt}	Q^*_{alt}



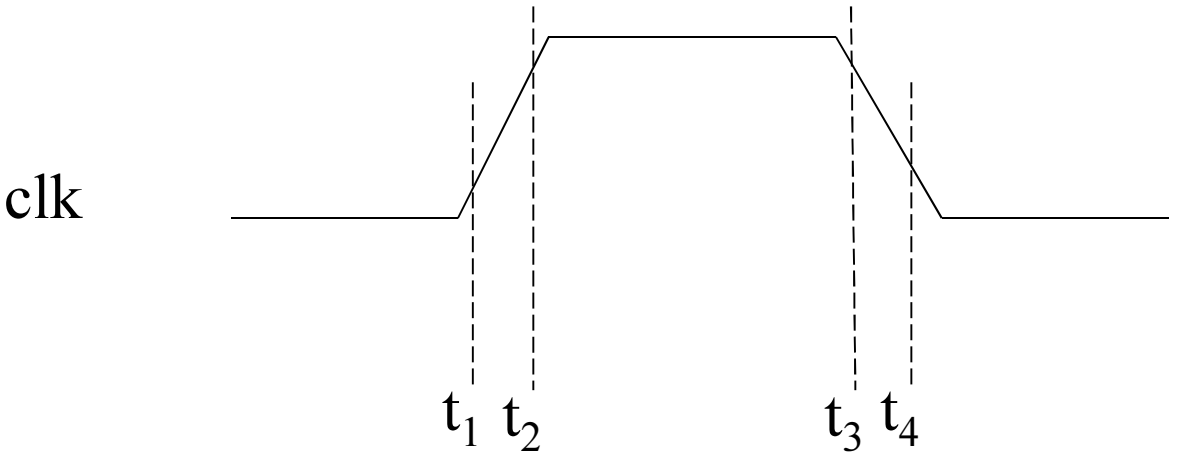
Leider genügen Auffangflipflops noch nicht den Anforderungen, die wir an die Speicherelemente unserer Schaltwerke stellen müssen. Was würde passieren? Sobald der Takt auf 1 ist, wird der Eingang der Flipflops als neuer Zustand an den Ausgang übernommen. Dieser neue Zustand liegt aber jetzt am Schaltnetz an, das nach einigen Gatterschaltzeiten neue Ausgänge produziert. Einige von diesen Ausgängen liegen als Folgezustand an den Eingängen der Flipflops. Da aber der Takt immer noch 1 sein kann, wirken diese sich nun wieder auf die Ausgänge der Flipflops aus und der Zustand wird wieder überschrieben. Mit anderen Worten: Die Signale würden einige Male in dieser Rückkopplungsschleife herumlaufen, bis irgendwann das Taktsignal auf 0 geht, und das Flipflop schließlich speichert.
Was wir also stattdessen benötigen ist ein Flipflop, das seinen Ausgang stabil lässt, solange die neuen Eingänge übernommen werden und dann, zu einem späteren Zeitpunkt, seine Eingänge nicht mehr übernimmt und stattdessen den übernommen Wert an den Ausgang weiterschaltet. Ein Flipflop, das dieses leistet, nennt man ein **System-Flipflop**.
Ein System Flipflop kann man als sogenanntes **Master-Slave-Flipflop** aus zwei Auffangflipflops zusammensetzen, und zwar einem negativ levelgesteuerten und einem positiv levelgesteuerten Latch.

Master-Slave D-Flipflop

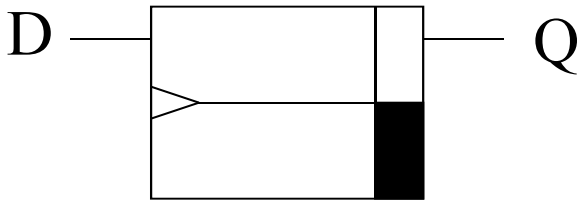




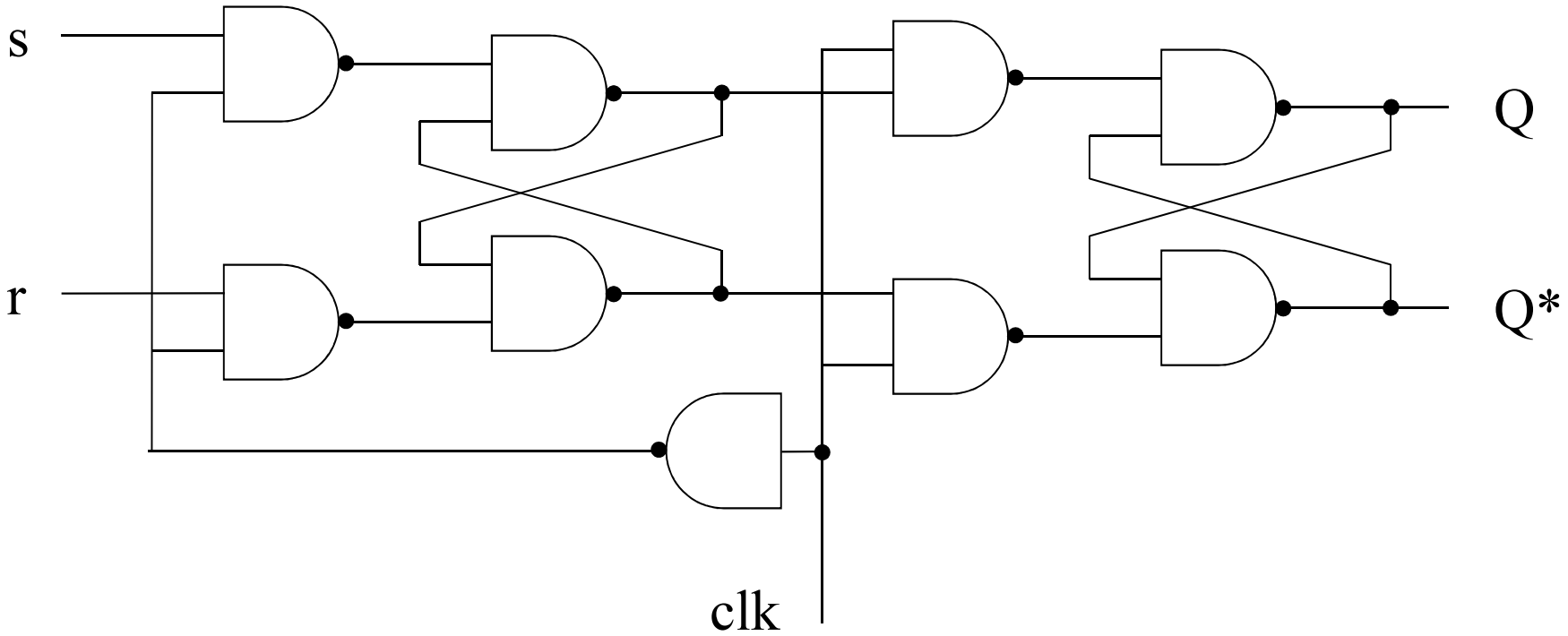
Master-Slave D-Flipflop, Timing



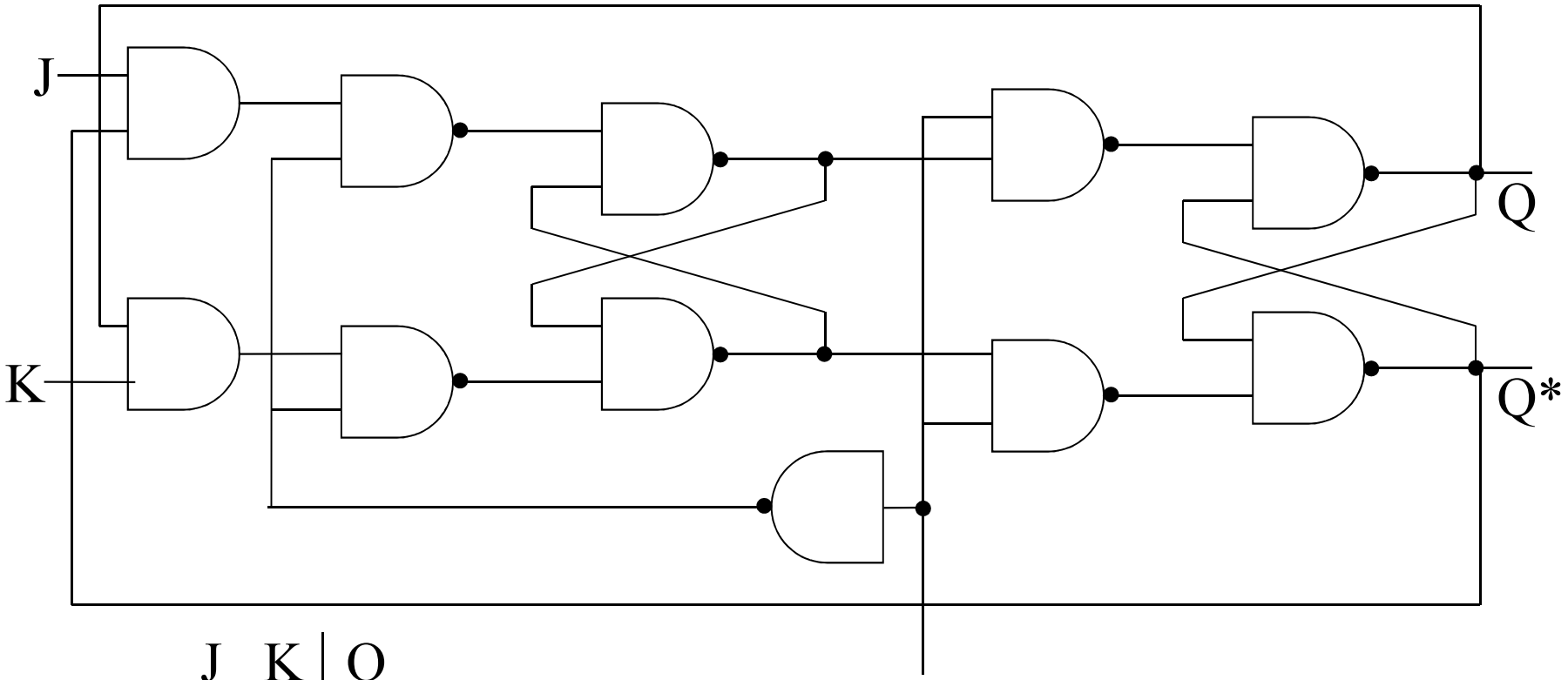
t_1: Eingang sperrt
t_2: Ausgang öffnet
t_3: Ausgang sperrt
t_4: Eingang öffnet



Entsprechend dem D-Flipflop kann man natürlich auch r-s-Flipflops als Master-Slave-Flipflops aufbauen. Allerdings bleibt das Problem mit der verbotenen Eingabe.

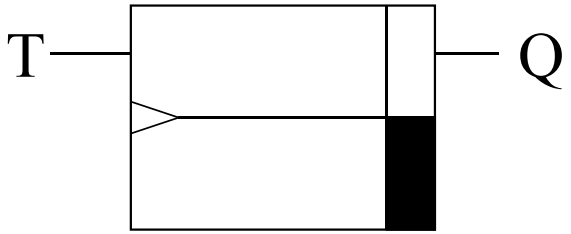


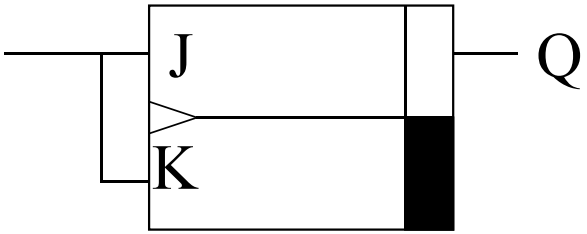
Um dieses Problem zu umgehen, kann man ein sogenanntes J-K-Flipflop bauen. Bei diesem sind die Ausgänge an die Eingänge



J	K	Q
0	0	Q_{alt}
0	1	0
1	0	1
1	1	$\overline{Q_{alt}}$

Gelegentlich benötigt man nur die Funktionalität der ersten und letzten Spalte der Wertetabelle eines J-K-Flipflops. In diesem Fall kann man ein Wechselflipflop (W-Flipflop oder auch T-Flipflop) verwenden, das nichts anderes ist als ein J-K-Flipflop mit verbundenen Eingängen:

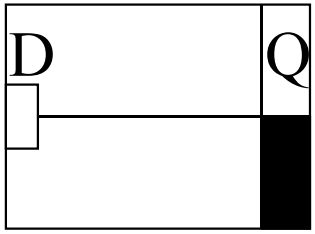




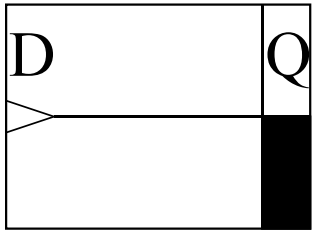
T	Q
0	Q_{alt}
1	$\overline{Q_{alt}}$

Alle diese Systemflipflops übernehmen die Werte des Eingangs während der negativen Taktphase und zeigen sie am Ausgang zu Beginn der positiven Taktphase. Sie wechseln also ihren Zustand mit der **positiven Flanke** des Taktes. Dies ist für Systemflipflops nicht zwingend erforderlich: Wenn man ein Master-Slave-Flipflop so aufbaut, daß der Master während der positiven Phase übernimmt und der Slave während der negativen, so hat man ein Systemflipflop, das bei der negativen Flanke den Zustand ändert. Man spricht auch von positiv oder negativ flankengesteuerten (oder auch flankengetriggerten) Flipflops. Entsprechend wurden Latches in positiv oder negativ pegelgesteuerte (oder auch levelgesteuerte) Flipflops eingeteilt:

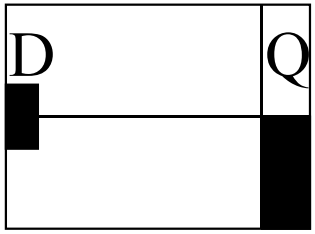
Positiv levelgesteuert



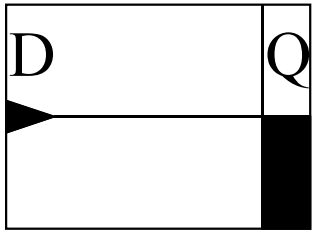
Positiv flankengesteuert



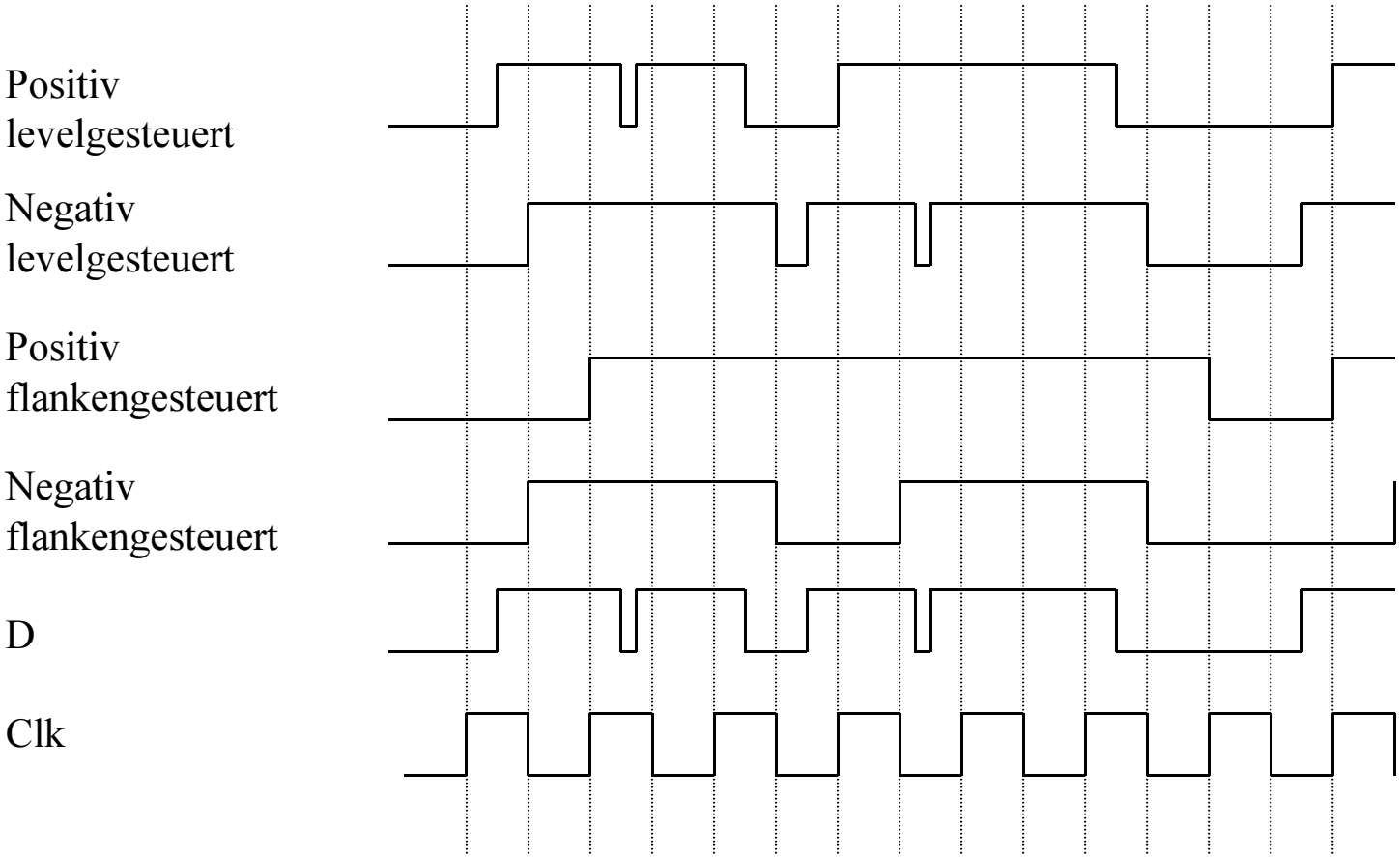
Negativ levelgesteuert



Negativ flankengesteuert



Hier sehen wir die Verläufe der Zustandsänderungen dieser vier Flipfloptypen für ein gemeinsames Eingangssignal D:



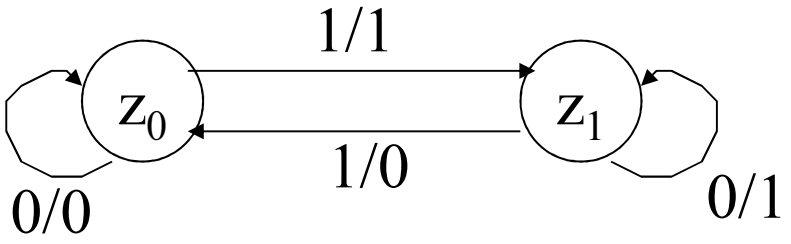
Beispiel:
Automat zur Erkennung der geraden Parität eines bit-seriell übertragenen Binärdatums.

$A = (X, Y, Z, \delta, \lambda)$ $\quad X = \{ 0,1 \}$

$Y = \{ 0,1 \}$

$Z = \{ 0,1 \}$

δ	0	1
0	0	1
1	1	0



λ	0	1
0	0	1
1	1	0

x	z	z‘	y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Beispiel:
Cola-Automat

$A = (X, Y, Z, \delta, \lambda)$

$X = \{ 0.50, 1.-, KE, Rück \}$
$Y = \{ Cola, 0.50, 1.-, KA \}$
$Z = \{ KS, 0.50, 1.- \}$

δ	0.50	1.-	KE	Rück
KS	0.50	1.-	KS	KS
0.50	1.-	KS	0.50	KS
1.-	KS	0.50	1.-	KS

λ	0.50	1.-	KE	Rück
KS	KA	KA	KA	KA
0.50	KA	Cola	KA	0.50
1.-	Cola	Cola	KA	1.-

Codierung:

X	KE	0.50	1.-	Rück
x_1x_0	00	01	10	11
Y	KA	0.50	1.-	Cola
y_1y_0	00	01	10	11
Z	KS	0.50	1.-	
z_1z_0	00	01	10	

Wertetabelle:

x_1	x_0	z_1	z_0	z_1'	z_0'	y_1	y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1	X	X	X	X
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	0	0	1	1
0	1	1	1	X	X	X	X
1	0	0	0	1	0	0	0
1	0	0	1	0	0	1	1
1	0	1	0	0	1	1	1
1	0	1	1	X	X	X	X
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	1
1	1	1	0	0	0	1	0
1	1	1	1	X	X	X	X

KV-Diagramme:

z_1 ‘

x_1 \ x_0				
				1
		d	d	
z_0	1	d	d	
			1	

z_1

y_1

x_1 \ x_0				
		1	1	
		d	d	1
z_0		d	d	
		1		

z_1

z_0 ‘

x_1 \ x_0				
			1	
		d	d	
z_0		d	d	1
	1			

z_1

y_0

x_1 \ x_0				
			1	
	1	d	d	1
z_0		d	d	
		1		

z_1

$$z_1' = \overline{x_0} x_1 \overline{z_0 z_1} + \overline{x_0 x_1} z_1 + x_0 \overline{x_1} z_0$$

$$z_0' = \overline{x_0} x_1 z_1 + \overline{x_0 x_1} z_0 + x_0 \overline{x_1 z_0 z_1}$$

$$y_1 = x_1 z_1 + x_0 z_1 + \overline{x_0} x_1 z_0$$

$$y_0 = x_1 z_0 + \overline{x_0} x_1 z_1 + x_0 \overline{x_1} z_1$$

Realisierung als FPLA: