
Digitaltechnik

Kapitel 5, Sequenzielle Schaltungen

Prof. Dr.-Ing. M. Winzker

Nutzung nur für Studierende der Hochschule Bonn-Rhein-Sieg gestattet.
(Stand: 20.03.2019)

Einleitung

Einordnung

- Kombinatorische Schaltungen (Schaltnetze):
 - Die Ausgangswerte hängen nur von den **aktuellen** Werten am Eingang ab
- Sequenzielle Schaltungen (Automaten):
 - Ausgangswerte hängen von **aktuellen und vorangegangenen** Werten am Eingang ab
 - ➔ Sequenzielle Schaltungen haben ein **Gedächtnis**

Beispiel: Fernseher mit „+“ und „-“ Taste für Kanal

- Der aktuell angezeigte Kanal hängt von den bisher gedrückten Tasten ab
- Der Kanal wird über **Zustände** beschrieben, d.h. die Nummer des aktuellen Kanals

Der **Zustand** speichert Informationen aus der Vergangenheit, die für die Funktion erforderlich sind.

5.1 Speicherelemente

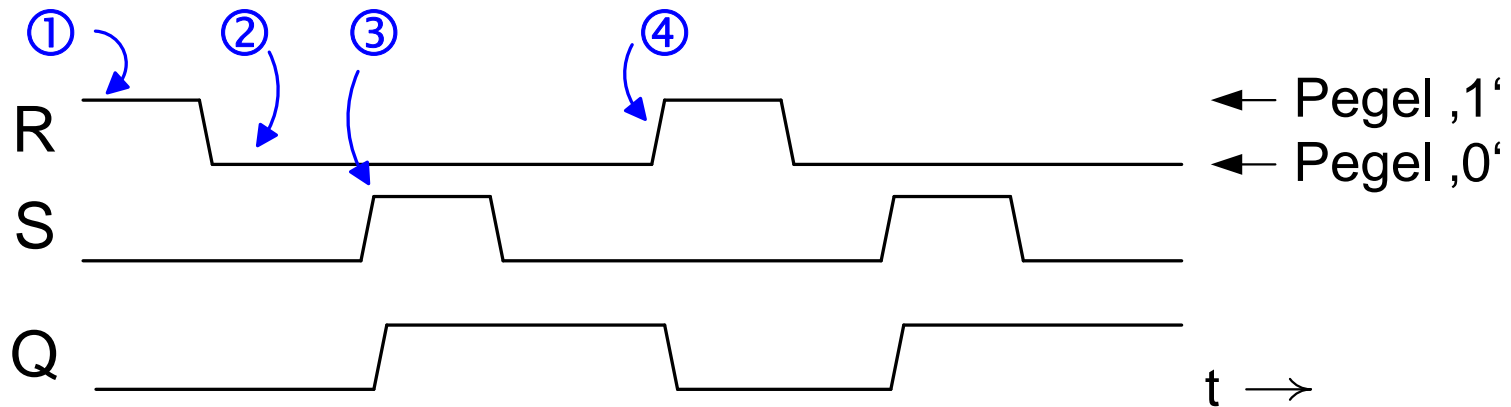
- Die Speicherung innerhalb von Schaltungen erfolgt durch Flip-Flops (FFs)
 - Ein Flip-Flop speichert ein Bit
- Das RS-Flip-Flop ist eine **Grundform** der Flip-Flops
- Ein RS-Flip-Flop hat zwei **Eingänge** R und S
 - R (**Reset**) dient zum Zurücksetzen des FFs (auf 0)
 - S (**Set**) dient zum Setzen des FFs (auf 1)
 - Wenn kein Eingang aktiv (auf 1) ist, wird der Zustand gehalten
 - Beide Eingänge dürfen nicht gleichzeitig aktiv (1) sein
 - Setzen und Zurücksetzen schließt sich gegenseitig aus
- Die zeitlich aufeinander folgenden Werte für Q werden durch Indizes n, n+1 angegeben



R	S	Q ⁿ	Q ⁿ⁺¹	
0	0	0	0	} Speichern
0	0	1	1	
0	1	0	1	} Setzen
0	1	1	1	
1	0	0	0	} Rücksetzen
1	0	1	0	
1	1	0	-	} verboten
1	1	1	-	

R	S	Q ⁿ⁺¹	
0	0	Q ⁿ	Speichern
0	1	1	Setzen
1	0	0	Rücksetzen
1	1	-	verboten

Zeitverlauf eines RS-FFs



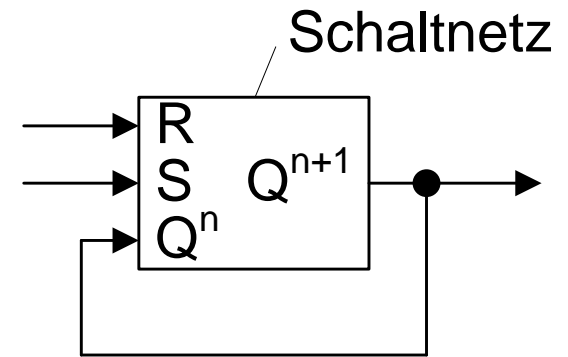
- Darstellung abstrahiert:
 - Nur Pegel 0 und 1
 - Zeitablauf durch schräge Signalübergänge angedeutet

Ablauf

- ① Der Eingang R ist 1, das RS-FF wird zurückgesetzt und Q ist 0.
- ② Beide Eingänge sind 0 und das RS-FF speichert den vorherigen Wert 0 für Q.
- ③ S wird 1 und setzt das RS-FF. Der Ausgang Q wird 1 und speichert diesen Wert auch wenn S wieder auf 0 geht.
- ④ Mit Aktivierung von R wird das RS-FF wieder auf 0 gesetzt.

Struktur des RS-Flip-Flops

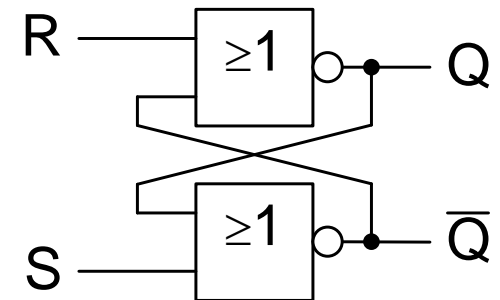
- Die Generierung eines neuen Wertes für Q^{n+1} aus den Eingängen R , S , Q^n erfolgt durch ein Schaltnetz
- Der generierte Wert wird wieder auf den Eingang des Schaltnetzes gegeben
- Die **Datenspeicherung** erfolgt über die **Rückkopplung** des Ausgangswertes
- Die Funktion des Schaltnetzes kann mit bekannten Verfahren (Karnaugh-Diagramm) in eine Schaltung umgewandelt werden
- Die gezeigte Implementierung mit zwei NOR-Gattern ist sehr effizient und liefert neben Q auch den invertierten Ausgang $\neg Q$



Für zu Hause:

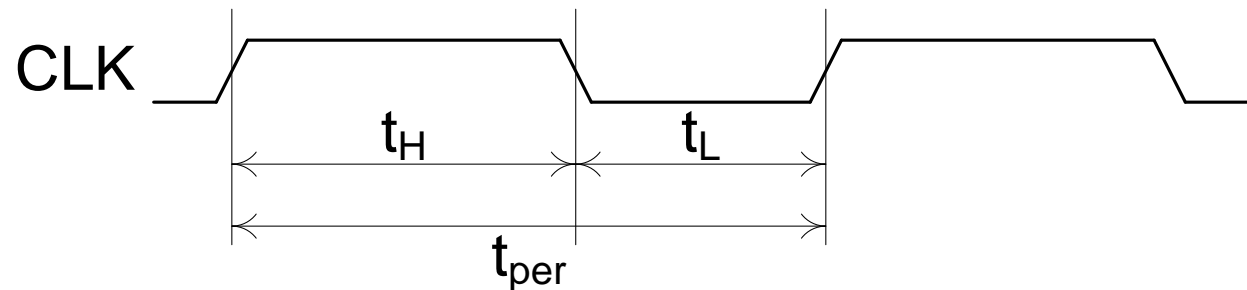
Erstellen Sie mit LogiFlash ein RS-FF aus zwei NOR-Gattern und simulieren Sie es.

Was passiert bei der verbotenen Kombination $R = S = 1$?



Taktsteuerung von Flip-Flops

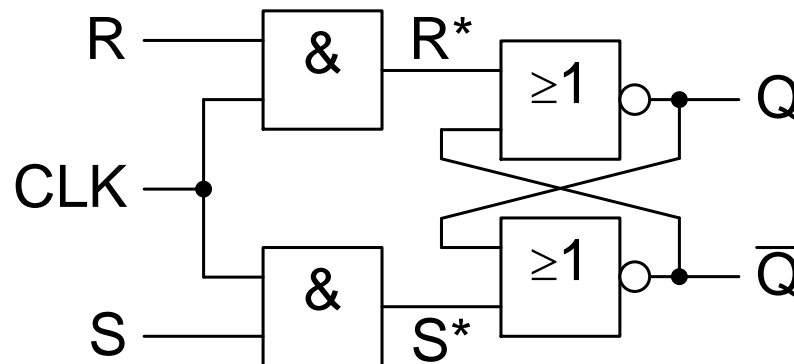
- Die meisten praktischen Automaten benutzen einen **Takt** zur Steuerung des Ablaufes
- Der Takt ist ein periodisches Signal, welches in gleichmäßigem Rhythmus zwischen 0 und 1 wechselt
- Die englische Bezeichnung für Takt ist **Clock**; das Signal wird oft als **CLK** abgekürzt



- Kennzeichnend für einen Takt sind:
 - Periodendauer T_{per} und Frequenz $1/T_{per}$
 - Tastverhältnis („duty cycle“): T_H/T_{per} (oder T_L/T_{per})
- **Taktfrequenzen** sind im Bereich zwischen 10 MHz (einfache Schaltung) bis zu über 3 GHz (aktuelle CPUs)
- Die **Periodendauern** liegen entsprechend von 100 ns bis unter 0,3 ns
- Der „**duty cycle**“ sollte möglichst etwa 50% sein, also im Bereich 45%-55%

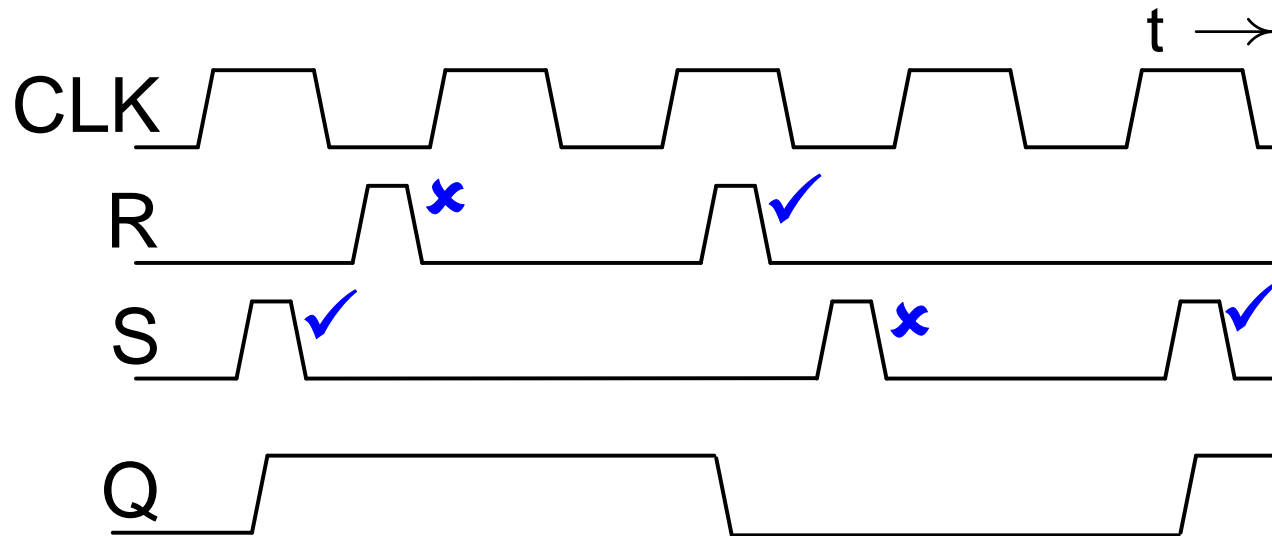
Steuerung durch Taktzustand und Taktflanke

- Die Speicherung im Flip-Flop kann gesteuert werden durch
 - Taktpegel, also Takt ist 1 (oder 0)
 - Taktflanke, also Takt wechselt von 0 auf 1 (oder von 1 auf 0)
- **Taktzustandsgesteuerte Flip-Flops** sind eine einfache Erweiterung von RS-FFs
- Steuereingänge werden mit Takteingang verknüpft



Zeitablauf taktpiegelgesteuerter Flip-Flops

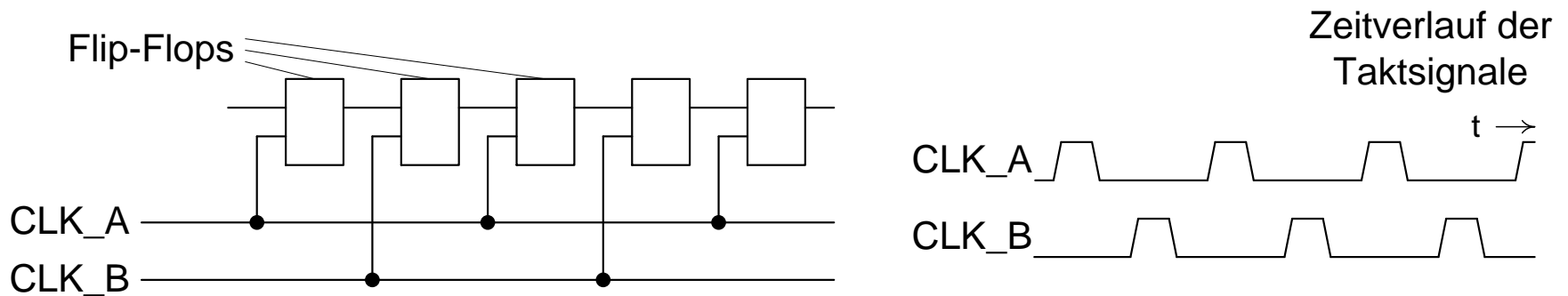
- Nur bei Takt gleich 1 werden die Signale R und S ausgewertet (✓)
- Bei Takt gleich 0 sind auch Signale R* und S* auf 0 und das FF ändert seinen Speicherinhalt nicht (✗)



Zweiphasentakt

- Problem taktpegelgesteuerter Flip-Flops
 - Bei direkter Verbindung „läuft“ eine Information direkt durch alle Flip-Flops
- Daher sind abwechselnd Flip-Flops mit einem **Zweiphasentakt**, also zwei verschiedenen Taktsignalen ohne Überlappung, angesteuert

Flip-Flop-Kette mit Zweiphasentakt

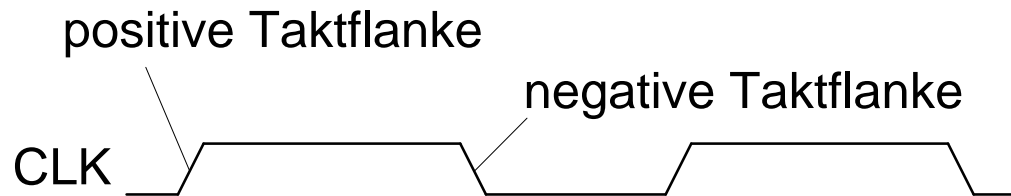


Vergleich

- Kanalschleuse mit zwei Schleusentoren
 - Tore werden abwechselnd, niemals gleichzeitig geöffnet
- ➔ Taktpegelgesteuerte Flip-Flops werden heute praktisch **nicht** mehr verwendet

Steuerung durch Taktflanke

- Flip-Flops werden meist als **taktflankengesteuerte** FFs ausgeführt
- Ein taktgesteuertes Flip-Flop wird durch jede **positive** oder **negative Flanke** des Taktsignals aktiviert
 - Eine Aktivierung durch **beide** Flanken ist **nicht** möglich
- Bei einer Flanke wird der Ausgang entsprechend der Steuereingänge aktualisiert

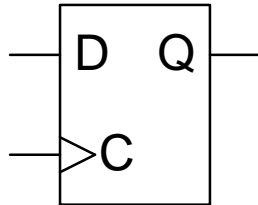


- Eine Ansteuerung erfolgt nicht über R und S
- Ein Datenwert D am Eingang wird bei der Taktflanke übernommen und gespeichert

Taktflankengesteuertes D-Flip-Flop

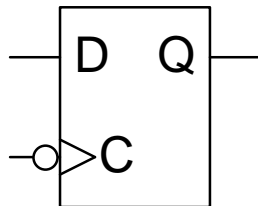
- Das D-Flip-Flop (D-FF) ist das **einfachste** und am **häufigsten verwendete** Flip-Flop
- Der Eingang D wird bei einer Flanke übernommen und am Ausgang Q ausgegeben

Positive Taktflanke:



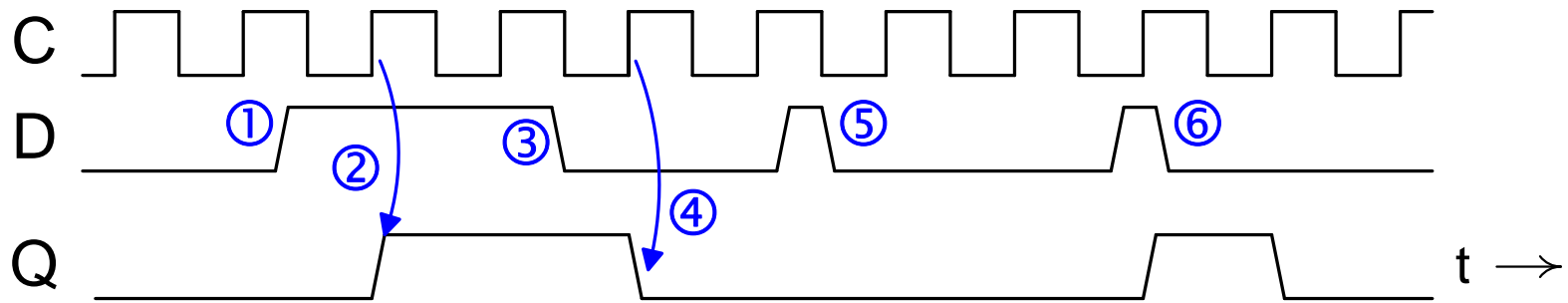
D	C	Q^{n+1}
0		0
1		1
X	0	Q^n
X	1	Q^n

Negative Taktflanke:



D	C	Q^{n+1}
0		0
1		1
X	0	Q^n
X	1	Q^n

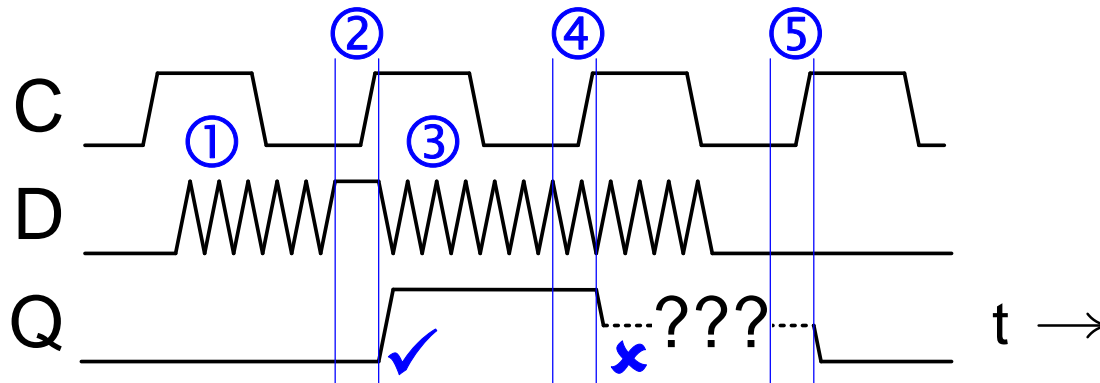
Zeitablauf D-Flip-Flop



- ① Der Eingang D wird 1.
- ② Bei der nächsten steigenden Taktflanke speichert das D-Flip-Flop den Eingangswert und gibt ihn am Ausgang aus. Q wird 1.
- ③ Der Eingang D wird 0.
- ④ Bei der nächsten steigenden Taktflanke speichert das D-Flip-Flop wieder den Eingangswert. Q wird 0.
- ⑤ D wird 1 und vor der nächsten steigenden Taktflanke wieder 0. Der gespeicherte Wert im Flip-Flop und der Ausgang Q ändern sich nicht.
- ⑥ D wird wieder kurz 1, dann 0. Da in dieser Zeit eine steigende Taktflanke auftritt, wird der Ausgang für einen Takt gleich 1.

Reales Zeitverhalten

- Das D-Flip-Flop übernimmt den Eingangswert bei der positiven Taktflanke.
- Dabei darf sich der Eingangswert zum Zeitpunkt der Taktflanke nicht ändern, sondern muss kurz vor und eventuell kurz nach der Taktflanke stabil sein



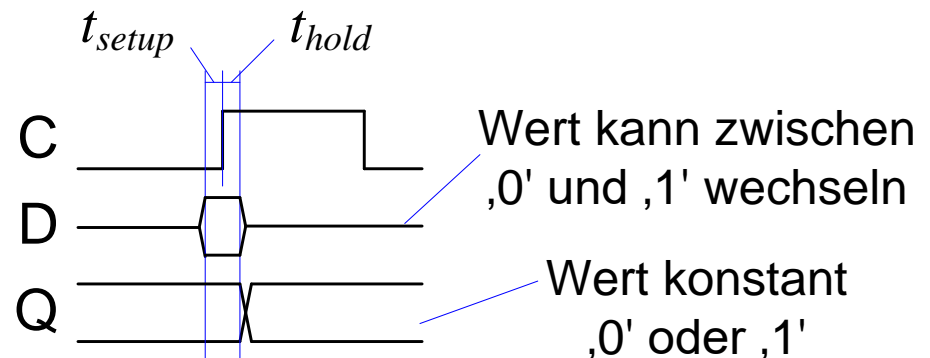
- ① Der Dateneingang D wechselt vor der Taktflanke.
- ② Kurz vor und nach der Taktflanke ist D stabil und wird korrekt übernommen (✓).
- ③ Nach der Taktflanke kann D wieder wechseln.
- ④ Während der nächsten Taktflanke ist D nicht stabil und wird nicht korrekt übernommen (✗). Der Ausgang des Flip-Flops ist undefiniert.
- ⑤ Bei der nächsten Taktflanke ist D stabil. Wenn man Pech hat, benötigt das Flip-Flop einige Zeit um sich zu „fangen“. Dies wird als Metastabilität bezeichnet.

Setup- und Hold-Zeiten

- **Setup-Zeit:** Zeitraum **vor der Taktflanke**, in der sich ein Eingangssignal nicht ändern darf
- **Hold-Zeit:** Zeitraum **nach der Taktflanke**, in der sich ein Eingangssignal nicht ändern darf
- Zwischen Setup und Hold-Zeit ist das „Zeitfenster für gültige Eingangssignale“ („**Data-Valid-Window**“)
$$t_{valid} = t_{setup} + t_{hold}$$
- Aktuelle FFs haben eine Hold-Zeit die Null (oder negativ) ist
 - Dadurch gibt es keine Probleme beim Weiterreichen eines Signals von FF zu FF

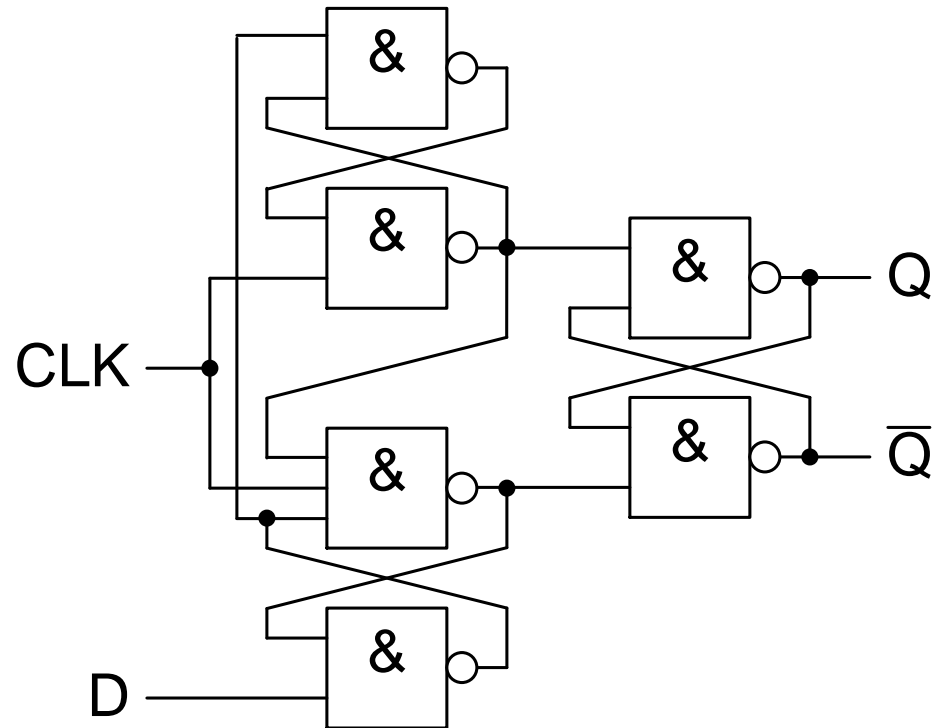
Übliche Darstellung

- Horizontaler Strich in der Mitte zwischen 0 und 1:
 - Wert darf beliebig wechseln.
- Zwei parallele Striche bei 0 und 1:
 - Konstanter Wert 0 oder 1.



Aufbau eines D-Flip-Flops

- Für den Aufbau eines D-Flip-Flops gibt es mehrere Möglichkeiten
 - Unterschiede in Größe, Zeitverhalten und Stromverbrauch
- Bild zeigt eine Möglichkeit
 - Links: Vorstufe erkennt steigende Taktflanke
 - Rechts: RS-FF



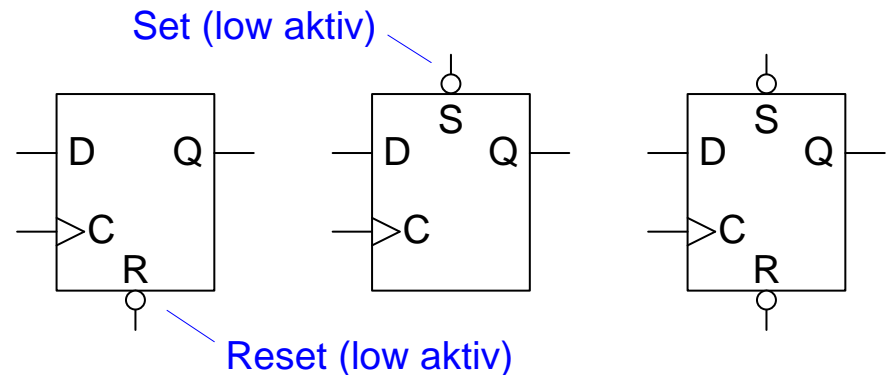
Nach: <http://www.ti.com/lit/ds/symlink/sn7474.pdf>

Asynchroner Set, Reset

- Der Ausgang eines taktf flankengesteuerten FF kann sich nur ändern, wenn eine Taktflanke auftritt
- Manchmal soll ein FF jedoch sofort, unabhängig vom Takt, beeinflusst werden
 - Setzen auf einen Startwert, beim Einschalten eines Geräts
 - Überwachen sehr kurzer Pulse
- Hierfür können FFs zusätzliche Steuereingänge haben
- Diese Steuereingänge sind **Asynchroner Set** und **Asynchroner Reset**
 - Asynchrone Steuereingänge sind Eingänge, die nicht synchron, also zeitlich abgestimmt mit dem Takt, wirken
 - Set setzt den Ausgang und den intern gespeicherten Wert Q^n **sofort** auf 1
 - Reset setzt den Ausgang und den intern gespeicherten Wert Q^n **sofort** auf 0
- Eine Änderung des Ausgangs erfolgt bei asynchronen Steuereingängen also nicht unbedingt erst bei der nächsten Taktflanke

Asynchroner Set, Reset (II)

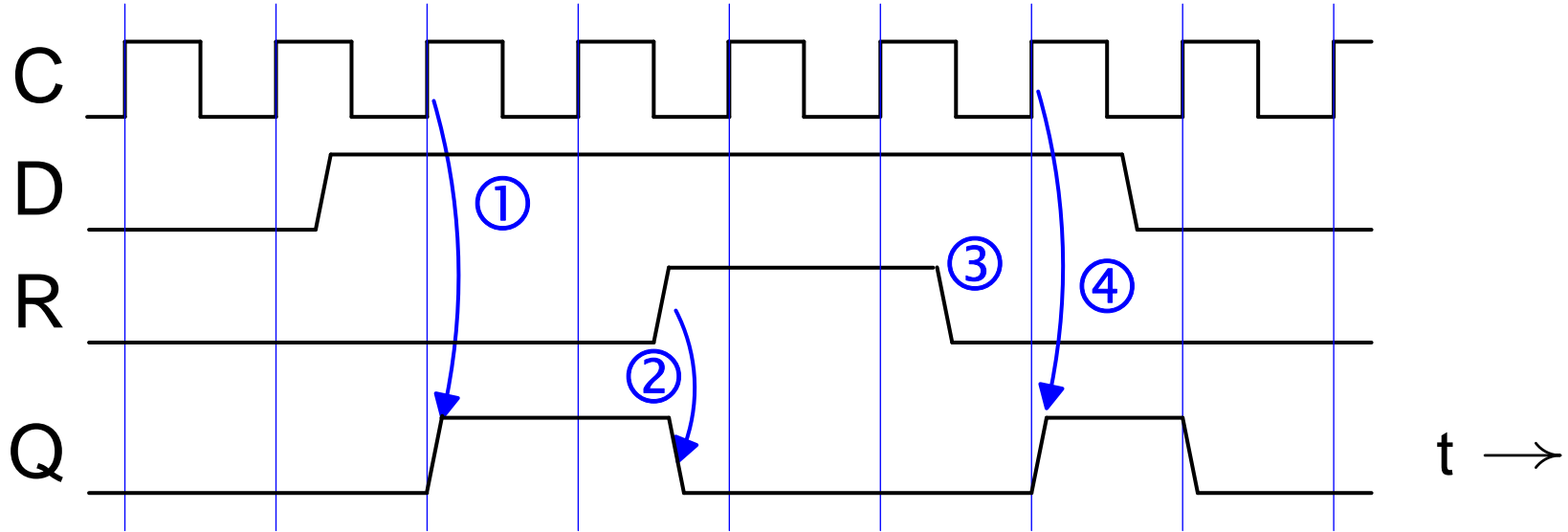
- FFs haben meist nur Set **oder** Reset
 - Wenn beide Eingänge vorhanden sind, muss definiert sein, welcher Eingang Priorität hat (meist Reset)
 - Achtung: Es ist praktisch **nicht möglich**, Set und Reset **wirklich gleichzeitig** zurückzunehmen
 - Falls Set und Reset gleichzeitig benötigt werden, muss das Zurücknehmen des Set/Reset also in **definierter Reihenfolge** erfolgen



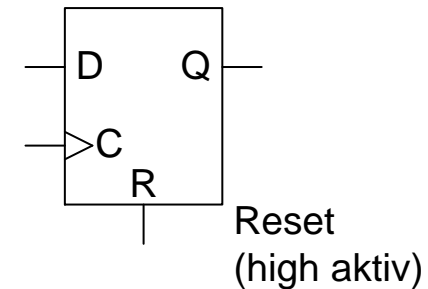
Praktischer Einsatz

- Initialisierung:** Beim Einschalten einer Digitalschaltung werden die Flip-Flops auf den gewünschten Startwert gesetzt
- Erkennung kurzer Impulse:** Ein Interrupt ist eventuell sehr kurz und schon vor der nächsten Taktflanke beendet.
 - Flip-Flop wird auf 0 gesetzt, Interrupt liegt am asynchronen Set
 - Wenn das Flip-Flop auf 1 ist, lag ein Impuls am Set-Eingang vor

Zeitablauf für D-FF mit asynchronem Reset



- ① Mit der steigenden Taktflanke wird der Wert 1 des Eingangs D gespeichert.
- ② Durch eine 1 am Reset wird das D-FF sofort auf 0 gesetzt, also ohne auf die nächste Taktflanke zu warten.
- ③ Reset wird wieder 0, also inaktiv. Dies hat aber noch keine Auswirkung auf den gespeicherten Wert.
- ④ Erst mit der nächsten Taktflanke wird der Wert von D wieder ausgewertet und der Ausgang Q wird 1.



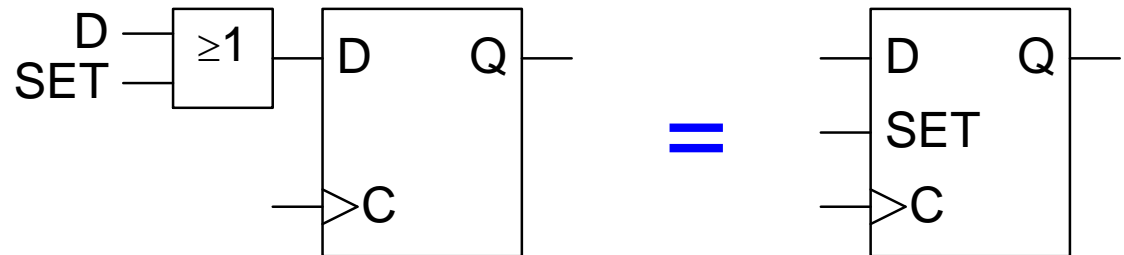
Erweiterung von D-FFs

- Neben asynchronen Set, Reset gibt es weitere Optionen der synchronen Ansteuerung eines D-FF
- Diese Ansteuerung kann prinzipiell durch ein Schaltnetz vor dem D-FF erfolgen
- Darstellung und Umsetzung als Steuereingang teilweise einfacher

Erweiterungen zum D-FF

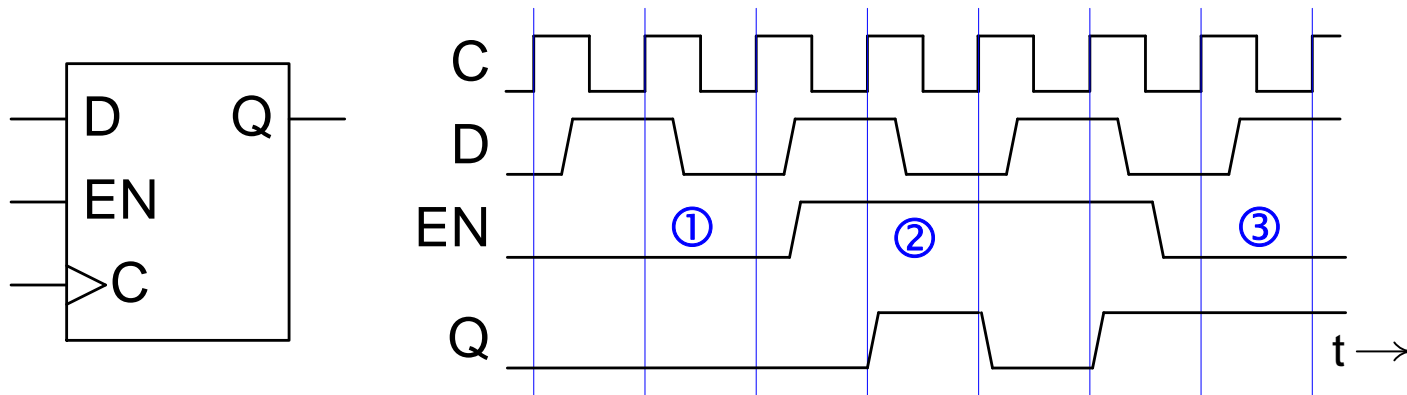
- **Synchroner Set, Reset:** Dieses Signal wird nur bei der Taktflanke ausgewählt und „überstimmt“ den D-Eingang
 - Ebenfalls zum Initialisieren geeignet, wenn Reset sicher bei Taktflanke anliegt

Beispiel: Synchroner Set



Erweiterung von D-FFs (II)





- **Enable:** Bei der Taktflanke wird der D-Eingang nur übernommen, wenn Enable (EN) gleich 1 ist. Ansonsten wird der Ausgang Q^n beibehalten.

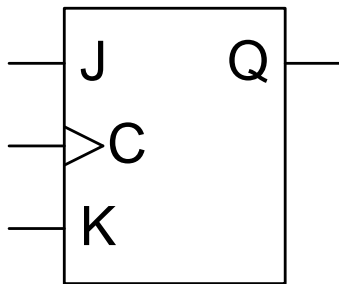


- ① EN ist 0 und das Flip-Flop behält seinen Wert.
- ② EN ist 1 und bei jeder steigenden Taktflanke wird der Wert von D übernommen.
- ③ EN ist 0 und das Flip-Flop behält seinen Wert.

Selten: JK-Flip-Flop

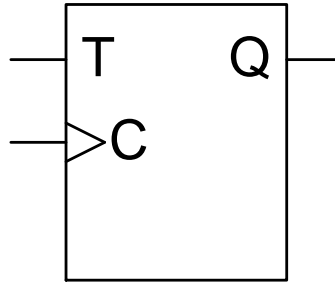
- Ähnliche Funktion wie RS-Flip-Flop, allerdings flankengesteuert
- Zwei Eingänge, J und K:
 - J = 1 setzt den Ausgang auf 1
 - K = 1 setzt den Ausgang auf 0
 - Merken durch:
 - J=„Jump“
 - K=„Kill“
 - J = 1 und K = 1 ist erlaubt und invertiert den Ausgang



J	K	CLK	Q^{n+1}
0	0		Q^n
0	1		0
1	0		1
1	1		$\overline{Q^n}$
<i>ansonsten</i>			Q^n



Selten: Toggle-Flip-Flop

- Ein Steuereingang, T:
 - T = 1 invertiert den Ausgang
(umschalten = engl. „toggle“)
 - T = 0 lässt den Ausgang unverändert
- Kurzbezeichnung: T-FF



T	CLK	Q^{n+1}
0		Q^n
1		$\overline{Q^n}$
ansonsten		Q^n

- Auch JK-FF und T-FF lassen sich, z.B. durch asynchronen Reset, erweitern

Einsatz

- JK-FFs und T-FFs wurden früher eingesetzt, als Digitalschaltungen noch durch einzelne diskrete Schaltungen aufgebaut wurden.
- Durch geschickte Ansteuerung konnten Logikgatter eingespart werden.
- Heutzutage werden **praktisch nur D-FFs** verwendet.

5.2 Flip-Flops in VHDL

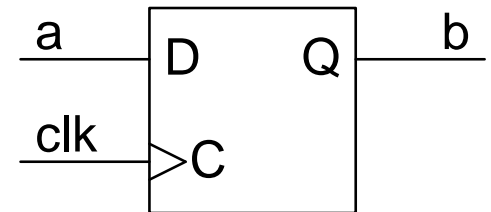
- VHDL beschreibt Flip-Flops über einen **Prozess**
 - Der Prozess beschreibt die Abhängigkeit des Ausgangs vom Takt
- Die Daten werden durch die if-Abfrage mit der **steigenden Taktflanke** übernommen
 - Durch „*falling_edge*“ könnte auch die fallende Flanke benutzt werden

```
signal a , b : std_logic;  
[...]  
  
process  
begin  
    wait until rising_edge(clk);  
  
    b <= a;  
end process;
```

Steigende Taktflanke

Der Wert ‚b‘ wird für einen Taktperiode gespeichert.
Dies entspricht einem Flip-Flop

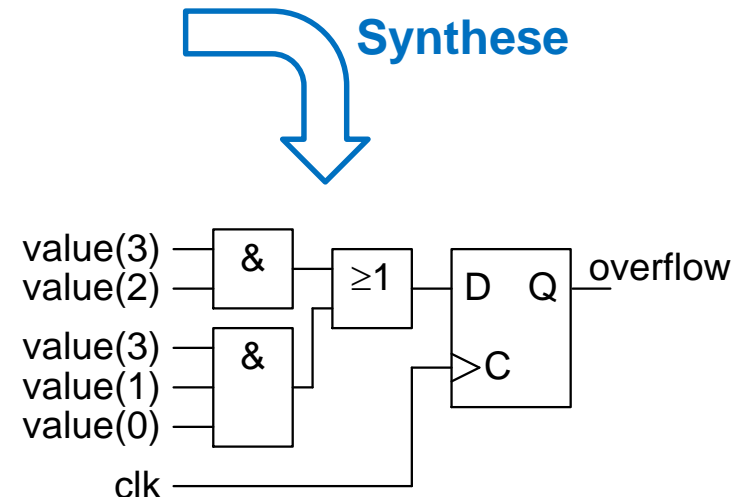
- Syntheseprogramme erkennen die besondere Beschreibung mit dem Schlüsselwort „*rising_edge*“
- Aus der Beschreibung wird ein Flip-Flop generiert



Erweiterte Funktion des getakteten Prozesses

- Fünf Zeilen Code für ein Flip-Flop hört sich zunächst umständlich an
- Der besondere Vorteil liegt darin, dass innerhalb der rising_edge-Bedingung weitere Funktionen beschrieben werden können
 - Weitere if-Abfragen, case-Bedingungen und logische Verknüpfungen, ...
- Das Synthese-Programm generiert aus der Beschreibung dann die Schaltung
- **Beispiel:** Erkennung einer Zahl größer 10

```
signal value : integer range 0 to 15;  
signal overflow : std_logic;  
[...]  
process  
begin  
    wait until rising_edge(clk);  
  
    if (value > 10) then  
        overflow <= '1';  
    else  
        overflow <= '0';  
    end if;  
end process;
```



Synchroner Reset und Set

- Synchroner Reset und Set wird durch eine if-Abfrage beschrieben
- Die synchrone Funktion wird nur bei der Taktflanke ausgeführt, darum ist ihre if-Abfrage dem rising_edge-wait nachgeordnet

```
process
begin
    wait until rising_edge(clk);

    if (reset = '1') then
        f <= '0';
        g <= '1';
    else
        f <= a or b;
        g <= b and c and d;
    end if;
end process;
```

Flip-Flop für f mit
synchronem Reset

Flip-Flop für g mit
synchronem Set

Enable

- Auch ein Enable wird durch eine if-Abfrage beschrieben
- Es ist eine synchrone Funktion, also ist ihre if-Abfrage dem rising_edge-wait nachgeordnet
- Die Enable-Abfrage enthält keine else-Beschreibung
 - Wenn enable aktiviert ist, wird der neue Wert übernommen, ansonsten findet keine Änderung statt

```
process
begin
    wait until rising_edge(clk);

    if (enable = '1') then
        f <= a or b;
        g <= b and c and d;
    end if;
end process;
```

Flip-Flop für f und g mit Enable

5.3 Kippstufen

- Flip-Flops werden in der Literatur auch als **Bistabile Kippstufen** bezeichnet
 - Bistabil meint, dass beide „Kippwerte“, also 0 und 1 stabil sind

Diese Bezeichnung legt nahe, dass es auch andere Kippstufen gibt

- Eine **Monostabile Kippstufe** hat nur einen stabilen Zustand; der instabile Zustand geht nach einer Verzögerungszeit in den stabilen Zustand über
 - **Beispiel:** Bei einer Treppenhausbeleuchtung geht der instabile Zustand „Licht an“ nach 2 Minuten in den stabilen Zustand „Licht aus“
 - **Anwendung:** Reset mit Mindestdauer
- Eine **Astabile Kippstufe** hat keinen stabilen Zustand, sondern wechselt periodisch zwischen den beiden Zuständen
 - **Anwendung:** Taktgenerator, Implementierung z.B. mit Schwingquarz

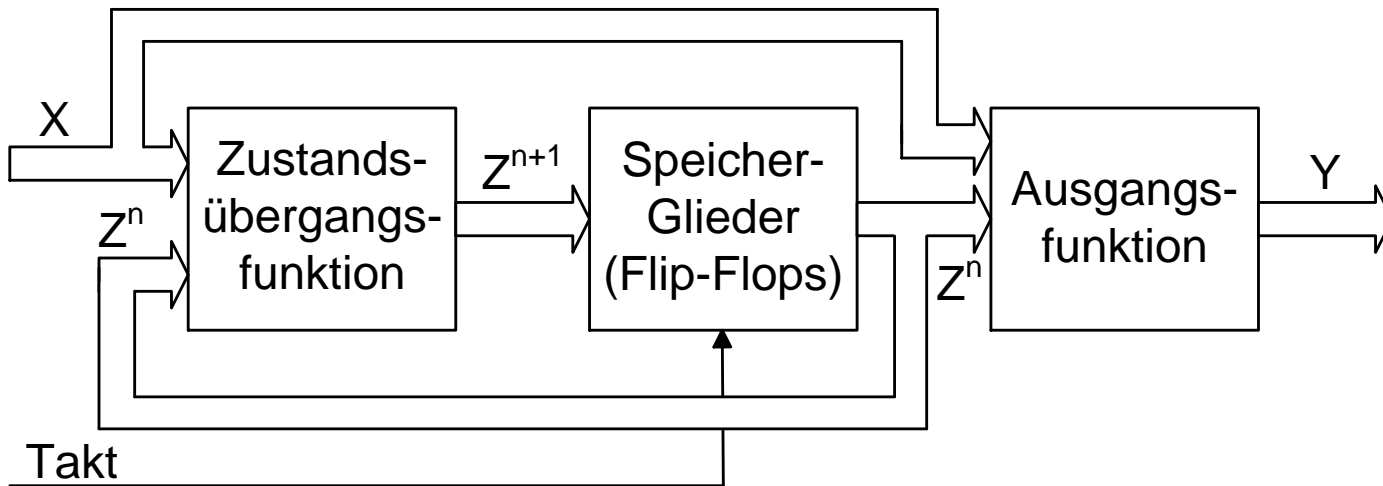
Mono- und Astabile Kippstufen werden meist als verfügbare Schaltungselemente eingesetzt

5.4 Endliche Automaten

- Ein Automat ist eine digitale Schaltung, dessen Verhalten durch aktuelle **Eingangsvariablen** und interne **Zustandsvariablen** bestimmt ist
- Die Zustandswerte, oder auch Zustände, beschreiben die „Vorgeschichte“ des Automaten
- Weil die Anzahl der internen Zustände endlich ist, werden Automaten auch als **Endliche Automaten** bezeichnet
- Die englische Bezeichnung lautet „Finite State Machine“ (FSM)
- In der theoretischen Informatik werden auch Automaten mit unendlich vielen Zuständen untersucht
 - Die unendlich vielen Zustände sind z.B. durch einen unendlich großen Speicher gegeben
 - Beispiel: Turing-Maschine

Mealy-Automat

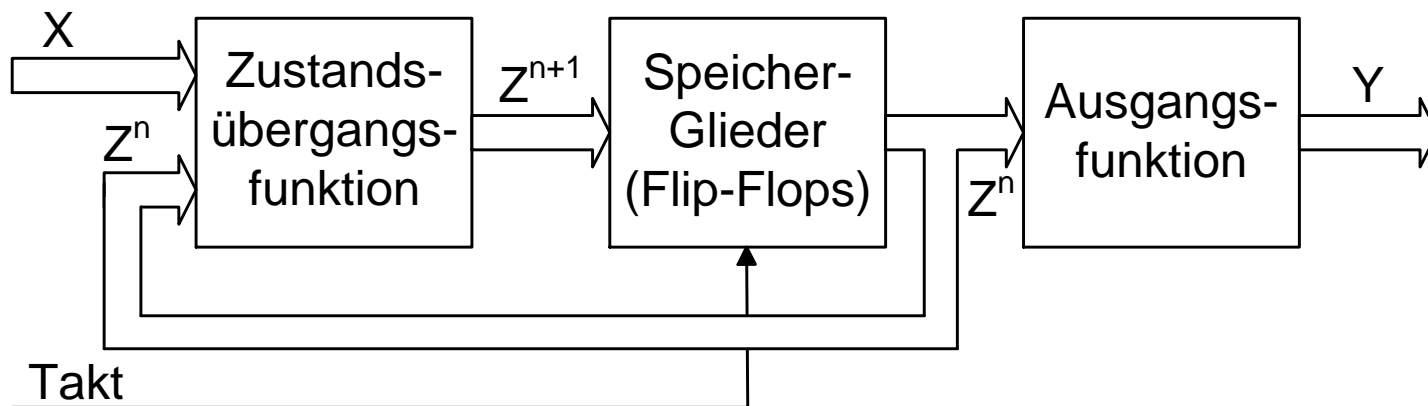
- Ein Automat lässt sich beschreiben durch:
 - Eingangsvariablen: X , eventuell mit mehreren Stellen: $X(0), X(1), \dots$
 - Ausgangsvariablen: Y , eventuell mit mehreren Stellen: $Y(0), Y(1), \dots$
 - Zustandsvariablen: Z , eventuell mit mehreren Stellen: $Z(0), Z(1), \dots$
 - Zustandsübergangsfunktion: $Z^{n+1} = f(X, Z^n)$ ❖
 - Ausgangsfunktion: $Y = g(X, Z^n)$



- ❖ Mit den aktuellen Zustandsvariablen Z^n (Z von aktuellen Zeitschritt n) wird die neue Zustandsvariable Z^{n+1} (für den nächsten Zeitschritt $n+1$) berechnet

Moore-Automat

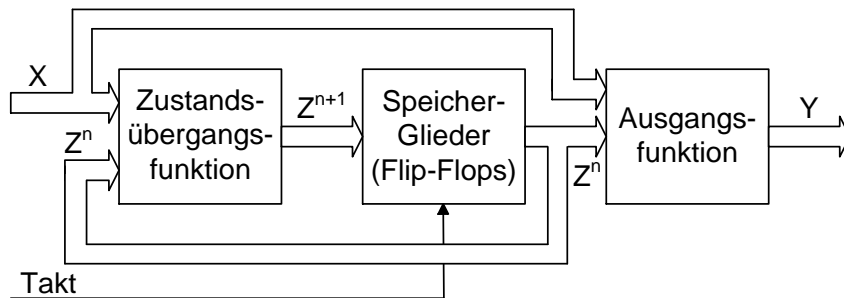
- Eine andere Struktur hat der **Moore-Automat**
- Bei einem Moore-Automaten ist die Ausgangsfunktion **nicht** von den **Eingangsvariablen** abhängig
- Die Ausgangsfunktion ist **nur** von den **Zustandsvariablen** abhängig:
 - Ausgangsfunktion: $Y = g(Z^n)$



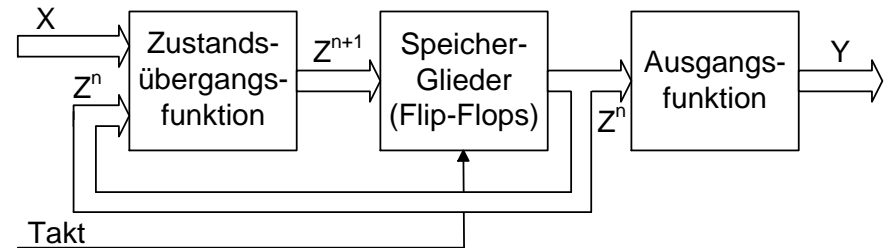
- Bei einem **Medwedew-Automaten** sind die Zustandsvariablen gleichzeitig die Ausgangsvariablen
- Der Medwedew-Automat ist ein Spezialfall des Moore-Automaten und wird zunächst nicht weiter betrachtet

Vergleich: Mealy- und Moore-Automat

- Der **Mealy-Automat** kann für **einen** Zustand **verschiedene** Kombinationen von Ausgangswerten erzeugen (abhängig von den Eingangsvariablen)
 - Der **Moore-Automat** kann für **einen** Zustand nur **eine** Kombinationen von Ausgangswerten erzeugen
- ➔ Der Mealy-Automat ist darum flexibler, aber auch aufwändiger



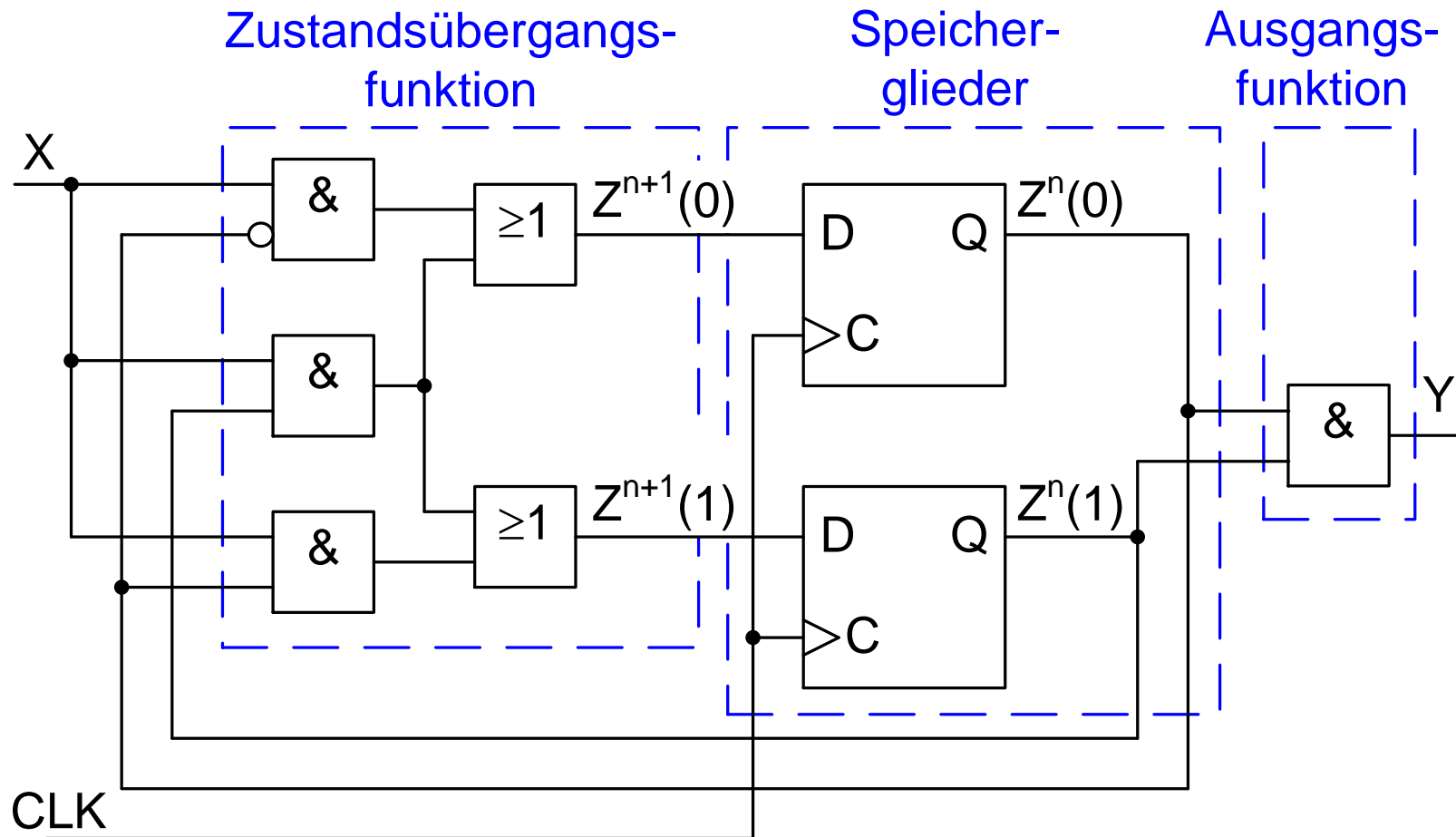
Mealy-Automat



Moore-Automat

- Grundsätzlich können für praktische Problemstellungen stets beide Automaten verwendet werden
- Für manche Problemstellungen ist ein Mealy-Automat besser geeignet, für andere ein Moore-Automat

Beispiel: Analyse eines Automaten



Beispiel: Analyse eines Automaten (II)

- Der Automat hat:
 - Eine Eingangsvariable: X
 - Zwei Zustandsvariablen: $Z(0)$, $Z(1)$
 - Eine Ausgangsvariable: Y
- Der Automat ist ein **Moore-Automat**, da die Ausgangsvariable Y nur von den Zustandsvariablen abhängt
- Mit zwei Zustandsvariablen können vier verschiedene Zustände gespeichert werden
- Eine Analyse der Zustandsübergangsfunktion und Ausgangsfunktion ergibt:

X	$Z^n(1)$	$Z^n(0)$	$Z^{n+1}(1)$	$Z^{n+1}(0)$	$Z^n(1)$	$Z^n(0)$	Y
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	0	1			
1	0	1	1	0			
1	1	0	1	1			
1	1	1	1	1			

Beispiel: Analyse eines Automaten (III)

- Anstelle der Codierung eines Zustands durch die Zustandsvariablen Z sollen die Zustände durch Namen bezeichnet werden
 - Die vier Zustände werden als A, B, C, D bezeichnet
 - Der aktuelle Zustand ist s^n
 - Der Folgezustand ist s^{n+1}
- Mit der Bezeichnung der Zustände kann die **Zustandsfolgetabelle** aufgestellt werden
- Die Zustandsfolgetabelle beschreibt für jeden Zustand s^n , welcher Folgezustand s^{n+1} und welcher Ausgabewert z sich für die möglichen Kombinationen an Eingangsvariablen ergibt

Codierung		Zustand
Z(1)	Z(0)	s
0	0	A
0	1	B
1	0	C
1	1	D

Funktion:

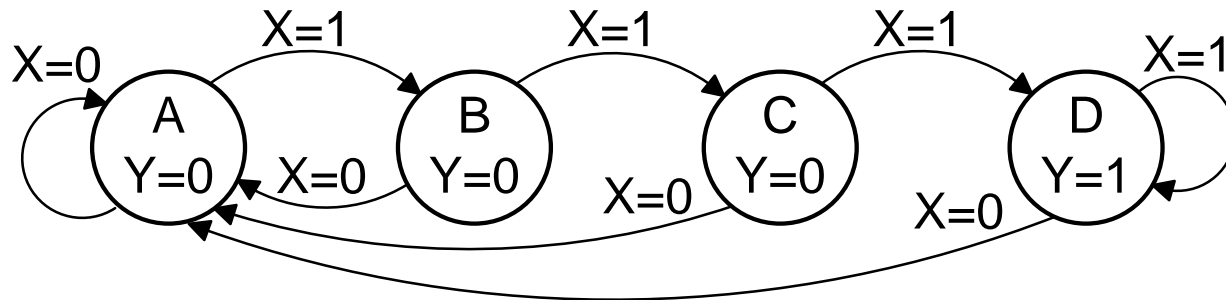
Der Automat erkennt Folgen von 1 am Eingang X. Ab drei aufeinander folgenden Werten 1, führt jede 1 zum Setzen des Ausgang Y auf 1. Wenn eine 0 am Eingang anliegt, wird Y sofort wieder 0.

s^n	s^{n+1}		
	X=0	X=1	Y
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1



Beispiel: Analyse eines Automaten (IV)

- Die Zustandsfolgetabelle kann auch graphisch als **Zustandsfolgediagramm** dargestellt werden
- Jeder Zustand wird durch einen Kreis mit seinem Namen dargestellt
- Die Zustandsübergänge sind durch Pfeile angegeben
 - Der Automat geht mit jedem Takt einen Pfeil weiter
- Für jeden Zustandsübergang ist ein Ausgangswert Y definiert



- Jeder Zustand hat eine bestimmte **Bedeutung**:
 - A: Der letzte Eingangswert war 0
 - B: Es gab bisher eine 1
 - C: Es gab bisher zweimal eine 1
 - D: Drei oder mehr Eingangswerte waren 1

Der **Zustand** speichert Informationen aus der Vergangenheit, die für die Funktion erforderlich sind.

5.5 Entwurf von Moore-Automaten

Der Ablauf beim Entwurf eines Automaten ist in etwa umgekehrt zur Analyse:

1. Spezifikation des Verhaltens
2. Aufstellen der Zustandsfolgetabelle
3. Minimierung der Zustände
4. Codierung der Zustände
5. Aufstellen der Ansteuerungstabelle
6. Logikminimierung

Spezifikation des Verhaltens

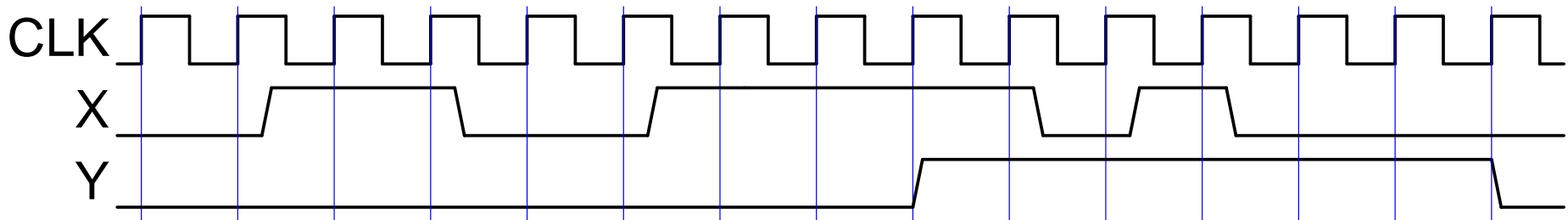
Das gewünschte Verhalten ist meist in Textform gegeben.

Spezifikation:

Zum Entprellen eines Tasters soll ein Automat entwickelt werden. Der Automat soll am Ausgang Y den entprellten Wert des Eingangs X angeben. Wenn am Eingang drei Takte lang der gleiche Wert 0 oder 1 anliegt, soll der Ausgang Y diesen Wert annehmen. Ansonsten soll der letzte Eingangswert, der mindestens drei Takte anlag ausgegeben werden.

Beim Einschalten soll der Wert 0 ausgegeben werden.

Ein **Zeitdiagramm** kann die Spezifikation ergänzen.



Spezifikation des Verhaltens (II)

- Zeitdiagramme dienen der **Unterstützung** und Illustration einer Spezifikation. Sie sind **kein Ersatz** für eine Spezifikation
- Die Angabe aller möglichen Abfolgen von Eingangskombinationen und Zuständen ist in einem Zeitdiagramm nicht sinnvoll (und meist gar nicht möglich)
- Während ein einfacher Automat in einem Absatz beschrieben werden kann, erfordert eine komplexe Schaltung (z.B. eine CPU) mehrere 100 Seiten Spezifikation
- Bei der Umsetzung einer Spezifikation können Unklarheiten auftreten, z.B. weil das Verhalten beim Einschalten oder einem bestimmten unerwarteten Eingangsverhalten nicht spezifiziert ist
- Diese Unklarheiten müssen durch Rückfrage geklärt werden

In einem größeren Projekt wird eine Spezifikation üblicherweise mehrfach geändert, weil:

- spezifizierte Eigenschaften nicht oder nur sehr aufwändig zu realisieren sind,
- weitere Eigenschaften zusätzlich benötigt werden, z.B. wegen gesetzlicher Bestimmungen oder Anforderungen des Marktes.

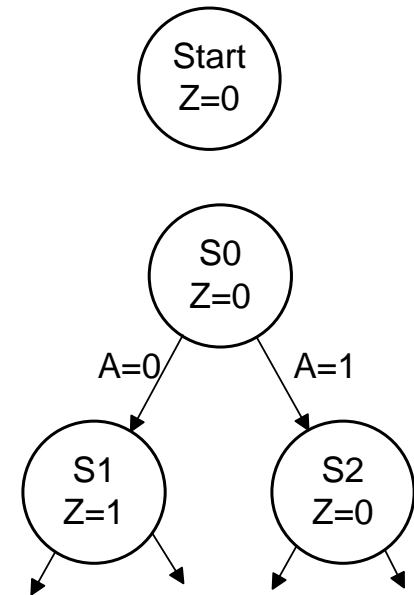
Aufstellen der Zustandsfolgetabelle

- Das Aufstellen der Zustandsfolgetabelle ist der eigentliche **kreative Schritt** der Entwicklung des Automaten
- Die Zustandsfolgetabelle kann am besten zunächst **graphisch** als **Zustandsfolgediagramm** aufgestellt werden
- Als erstes muss entschieden werden, ob eine Implementierung als Mealy- oder Moore-Automat erfolgen soll
 - Bei Übungsaufgaben ist normalerweise der Typ vorgegeben. Hier soll ein **Moore-Automat** erstellt werden
 - Wenn Sie mehrere Automaten entworfen haben, können Sie selbst beurteilen, welcher Automatentyp günstiger ist
 - Die praktischen Unterschiede der Automaten werden später diskutiert

Regeln für die Erstellung des Zustandsfolgediagramms

Für Moore-Automat:

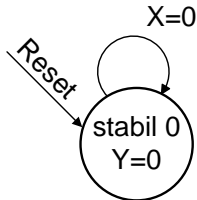
- Jeder Zustand wird durch einen Kreis dargestellt
 - Der Kreis enthält einen kurzen Kommentar
 - Der Kreis enthält die Ausgabe (oder Ausgaben) für den Zustand
 - Für jede mögliche Kombination an Eingangswerten muss ein Übergang zu einem Folgezustand vorhanden sein
 - Ein Übergang wird durch einen Pfeil dargestellt
 - Die Eingangskombination, für die der Übergang erfolgen soll, wird an den Pfeil angetragen
 - Ein Zustand kann auch sich selbst als Folgezustand haben; dann führt ein Pfeil auf den Zustand zurück
-
- Für einen **Mealy-Automat** gelten etwas andere Regeln
 - Die Unterschiede werden später erläutert



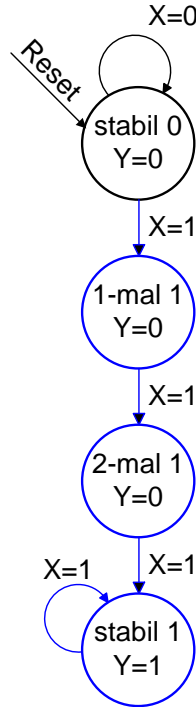
Der Automat geht mit jedem Takt einen Pfeil weiter.

Entwicklung des Zustandsfolgediagramms

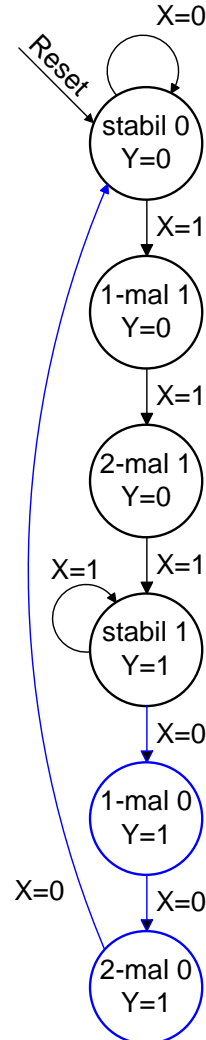
1. Startzustand



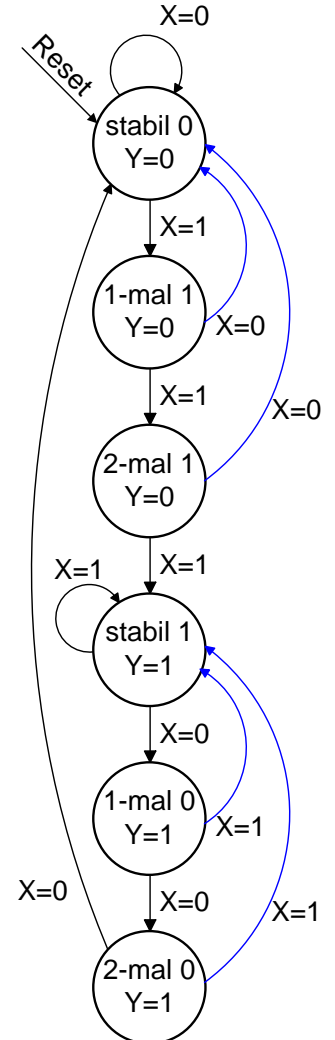
2. Übergang zu Y=1



3. Rückkehr zu Y=0



4. Fehlende Übergänge



Zustandsfolgetabelle

Die Zustandsfolgetabelle ist eine textuelle Form des Zustandsfolgediagramms

- Für jeden **Zustand** wird eine **Zeile** angelegt
- Für jede mögliche **Eingangskombination** wird eine **Spalte** angelegt
- Für die **Ausgangswerte** wird eine weitere **Spalte** angelegt
- In die Felder der Tabelle werden für jede Kombination aus Eingangswerten und Zustand der **Folgezustand** eingetragen
 - Für die Zustände sollten kurze oder prägnante **Abkürzungen** gewählt werden
- Für alle Zustände werden die **Ausgangswerte** eingetragen

s^n	s^{n+1}		Y
	X=0	X=1	
stabil 0*	stabil 0	1-mal 1	0
1-mal 1	stabil 0	2-mal 1	0
2-mal 1	stabil 0	stabil 1	0
stabil 1	1-mal 0	stabil 1	1
1-mal 0	2-mal 0	stabil 1	1
2-mal 0	stabil 0	stabil 1	1

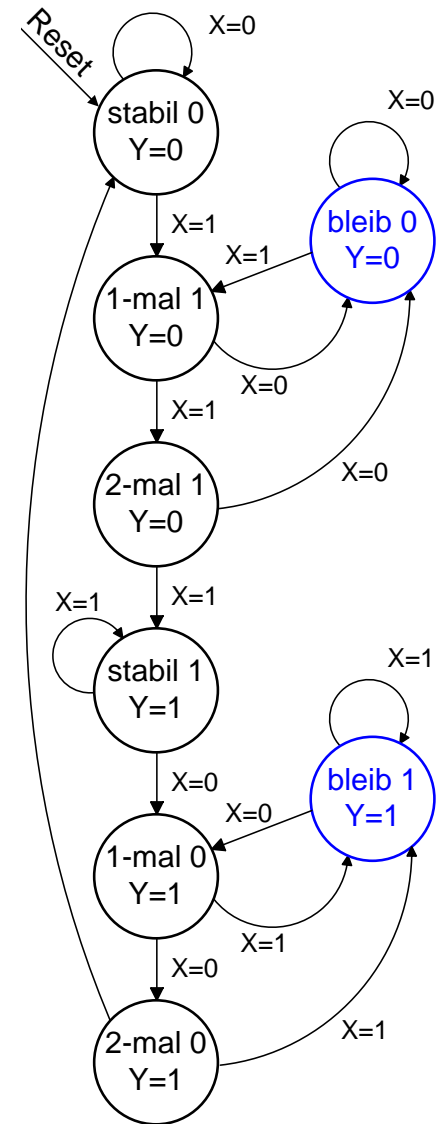
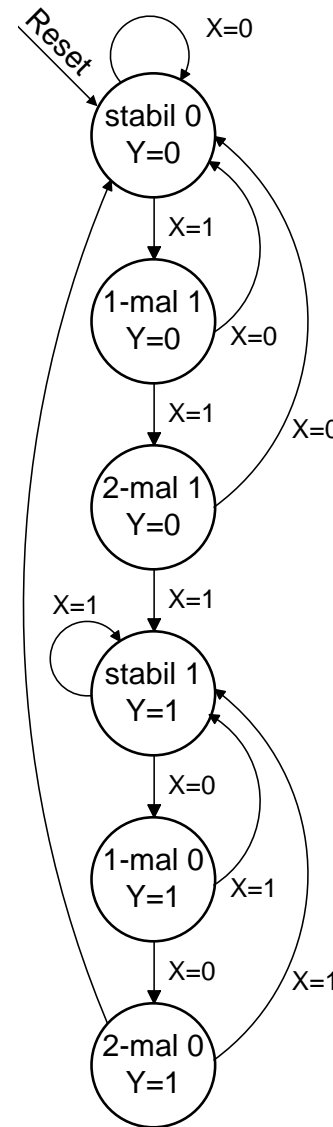
* = Reset

Minimierung der Zustände

- Für die gegebene Spezifikation hat ein Automat mit 6 Zuständen die minimale Anzahl an Zuständen
- Es sind jedoch auch Automaten mit mehr Zuständen möglich

Beispiel:

- In der rechten Zustandsfolgetabelle sind zwei zusätzliche Zustände „bleib 0“, „bleib 1“
- Diese Zustände treten auf, wenn ein kurzer Wechsel des Eingangspegels von nur ein oder zwei Takten auftritt. Dies reicht nicht aus, um die Ausgabe zu wechseln
- „bleib 0“ und „bleib 1“ sind **äquivalent** mit „stabil 0“ bzw. „stabil 1“ und können zusammengefasst werden



Minimierung der Zustände (II)

- Zwei **Zustände** sind **äquivalent**, wenn Ihre Ausgangswerte gleich sind und die Folgezustände für alle Eingangskombinationen gleich oder äquivalent sind
 - Äquivalente Zustände können zusammengefasst werden (**Zustandsverschmelzung**)
- Zwei **Automaten** sind **äquivalent**, wenn ihr Verhalten für alle möglichen Eingangskombinationen gleich ist
- Durch Minimierung der Anzahl der Zustände entsteht ein „**äquivalenter minimaler Automat**“
- Zur Minimierung der Zustände gibt es einen Algorithmus
- In der Praxis wird aber oft keine formale Zustandsminimierung durchgeführt, denn:
 - Durch geschicktes Aufstellen des Zustandsdiagramms entstehen keine (oder nur wenige) äquivalenten Zustände
 - Bei kleineren Automaten können äquivalente Zustände durch „genaues Hinschauen“ erkannt werden
 - Wenige zusätzliche Zustände führen meist nur zu geringem (oder keinem) Mehraufwand



Codierung der Zustände

- Die Zustände des Automaten müssen für eine Implementierung durch eine **Zustandscodierung** dargestellt werden
- Mit der Codewortlänge n müssen alle m Zustände dargestellt werden können, Mathematisch ausgedrückt muss also gelten: $2^n \geq m$
- Umgeformt nach n gilt: $n \geq \lg m$
 - Mit \lg als Zweierlogarithmus
 - Berechnung mit Zehnerlogarithmus als: $\lg m = \log m / \log 2$
- Für den Automaten mit 6 Zuständen gilt:
 - $\lg 6 = \log 6 / \log 2 = 0,778 / 0,301 = 2,58$
 - Es sind also mindestens 3 Stellen als Codewortlänge erforderlich
- Ermittlung über Zweierpotenz oft einfacher:
 - $2^3 = 8 \geq 6 \quad \rightarrow$ 3 Stellen reichen aus
 - $2^2 = 4 < 6 \quad \rightarrow$ 2 Stellen sind zu wenig
 - ➔ Kleine Zweierpotenzen sind meist bekannt: 2, 4, 8, 16, 32, ...

Codierung der Zustände (II)

- Ziel der Codierung ist,
 - ein möglichst geringer **Aufwand** für den gesamten Automaten
 - eine möglichst hohe **Geschwindigkeit** des Automaten
- Für die Codierung gibt es so viele Möglichkeiten, dass diese nicht alle ausprobiert werden können
 - Für eine Beurteilung bezüglich Aufwand und/oder Geschwindigkeit wäre die komplette Entwicklung bis auf Ebene der Logikgatter nötig
- Da dieser Aufwand viel zu hoch ist, wird versucht eine möglichst gute Codierung, und nicht die theoretisch beste Codierung zu wählen
- Man unterscheidet:
 - Codierung mit minimaler Codewortlänge
 - Codierung mit redundanter (nichtminimaler) Codewortlänge

Codierung mit minimaler Codewortlänge

- Mehrere Möglichkeiten, z.B. einfache Nummerierung
 - Die Zustände werden der Reihe nach mit Dualzahlen codiert
- Für eine Codierung mit minimaler Codewortlänge sind 3 bit nötig
 - Das Codewort wird als $z(2:0)$ bezeichnet
 - 2 Codierungen sind unbenutzt
 - Freie Codierungen werden zur Optimierung benutzt

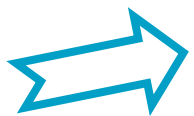
s^n	$Z(2:0)$
stabil 0	000
1-mal 1	001
2-mal 1	010
stabil 1	011
1-mal 0	100
2-mal 0	101

- Bei sehr sicherheitskritischen Schaltungen:
 - Für die unbenutzten Codeworte wird der Startzustand (oder ein anderer sinnvoller Zustand) als Folgezustand definiert
 - Bei ungültigem Codewort z.B. durch kurzfristigen Einbruch der Spannungsversorgung kann sich der Automat „aufhängen“

Aufstellen der Ansteuerungstabelle

- Mit der Codierung und den gewählten Flip-Flop-Typ, wird die Ansteuerungstabelle für die Zustandsübergangsfunktion und Ausgangsfunktion aufgestellt
 - Unbenutzte Codierungen erhalten als **Folgezustand** und **Ausgabe** den Wert „don't care“

Damit ergibt sich folgende Ansteuerungstabelle:

s^n	s^{n+1}		Y		s^n	Z^n	Z^{n+1}		Y
	X=0	X=1					X=0	X=1	
stabil 0*	stabil 0	1-mal 1	0		stabil 0*	000*	000	001	0
1-mal 1	stabil 0	2-mal 1	0		1-mal 1	001	000	010	0
2-mal 1	stabil 0	stabil 1	0		2-mal 1	010	000	011	0
stabil 1	1-mal 0	stabil 1	1		stabil 1	011	100	011	1
1-mal 0	2-mal 0	stabil 1	1		1-mal 0	100	101	011	1
2-mal 0	stabil 0	stabil 1	1		2-mal 0	101	000	011	1
* = Reset					-	110	- - -	- - -	-
					-	111	- - -	- - -	-

Logikminimierung

Aus der Ansteuerungstabelle ergeben sich folgende Logikfunktionen (aus Karnaugh-Diagramm):

$$Z^{n+1}(2) = \bar{X} \& Z^n(1) \& Z^n(0) \vee \bar{X} \& Z^n(2) \& \overline{Z^n(0)}$$

$$Z^{n+1}(1) = X \& Z^n(2) \vee X \& Z^n(1) \vee X \& Z^n(0)$$

$$Z^{n+1}(0) = X \& Z^n(2) \vee X \& Z^n(1) \vee X \& \overline{Z^n(0)} \vee Z^n(2) \& \overline{Z^n(0)}$$

$$Y = Z^n(2) \vee Z^n(1) \& Z^n(0)$$

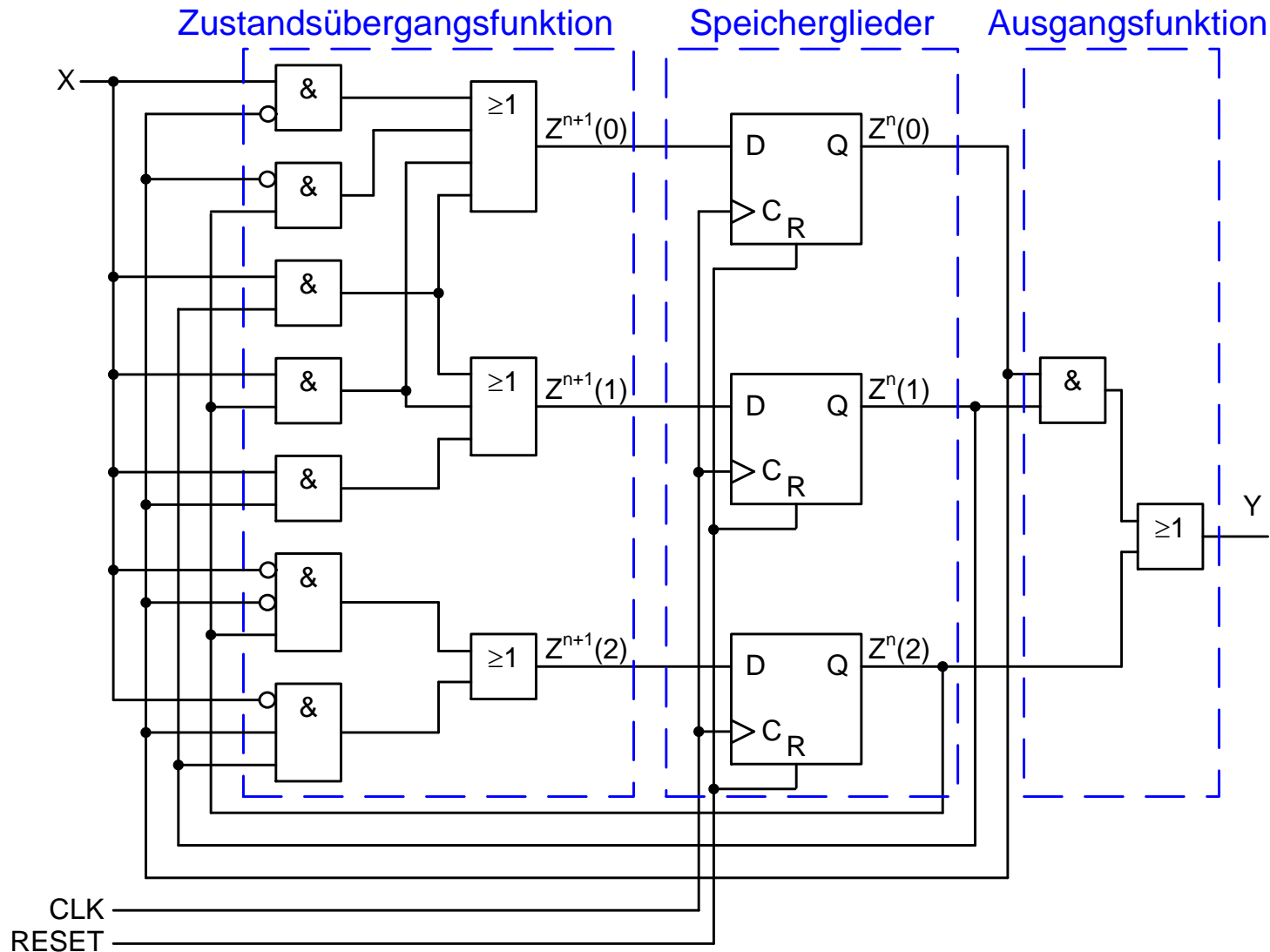
Der Startzustand "stabil 0" hat die Codierung "000"

Darum wird der Reset so geschaltet, dass alle Flip-Flops auf 0 gesetzt werden

Eine Animation des Automaten finden Sie auf:

<http://youtu.be/u0mxrlAxjX8>

Schaltbild des Automaten „Entprellen eines Tasters“



Codierung mit redundanter Codewortlänge

- Als andere **Strategie zur Zustandskodierung** wird eine redundante, also nichtminimale Codewortlänge gewählt
 - Nachteil: Es sind zusätzliche Flip-Flops erforderlich
 - Vorteil: Die kombinatorischen Schaltungen für Zustandsübergangsfunktion und Ausgangsfunktion sind (vermutlich) besonders einfach und schnell
 - ➔ Dieser Vorteil ist nicht garantiert; es ist eine **Strategie**
- Häufig verwendete Codes sind:

„One-Hot“-Codierung (1-aus-n-Code)

- Codewortlänge n gleich der Anzahl der Zustände
- In jedem Codewort ist nur eine Stelle 1 („Hot“), alle anderen 0

„Zero-One-Hot“-Codierung

- Ähnlich „One-Hot“, zusätzlich gibt es das Codewort mit sämtlichen Stellen 0
- Die Codewortlänge n ist gleich Anzahl der Zustände minus eins

	„One-Hot“ Z(5:0)	„Zero-One-Hot“ Z(4:0)
s^n		
stabil 0	000001	00000
1-mal 1	000010	00001
2-mal 1	000100	00010
stabil 1	001000	00100
1-mal 0	010000	01000
2-mal 0	100000	10000

Codierung mit redundanter Codewortlänge (II)

- Für den Entprell-Automaten ergeben sich dann 6 definierte Zustandscodierungen und 58 don't-cares

- $58 = 2^6 - 6$

- Logikfunktionen mit Quine-McCluskey erstellt, da 7 Eingangsvariablen

<http://sourceforge.net/projects/qmcmimizer/>

Es fällt auf:

- Funktionen deutlich einfacher
 - Wert $Z^n(5)$ wird nicht verwendet
- ➔ Effektiv also „Zero-One-Hot“, allerdings andere Variante als auf vorheriger Folie

s^n	$Z^n(5:0)$	$Z^{n+1}(5:0)$		Y
		X=0	X=1	
stabil 0*	000001*	000001	000010	0
1-mal 1	000010	000001	000100	0
2-mal 1	000100	000001	001000	0
stabil 1	001000	010000	001000	1
1-mal 0	010000	100000	001000	1
2-mal 0	100000	000001	001000	1
	<i>sonst</i>	- - - - -	- - - - -	-

* = Reset

$$Z^{n+1}(5) = \bar{X} \& Z^n(4)$$

$$Z^{n+1}(4) = \bar{X} \& Z^n(3)$$

$$Z^{n+1}(3) = X \& \overline{Z^n(1)} \& \overline{Z^n(0)}$$

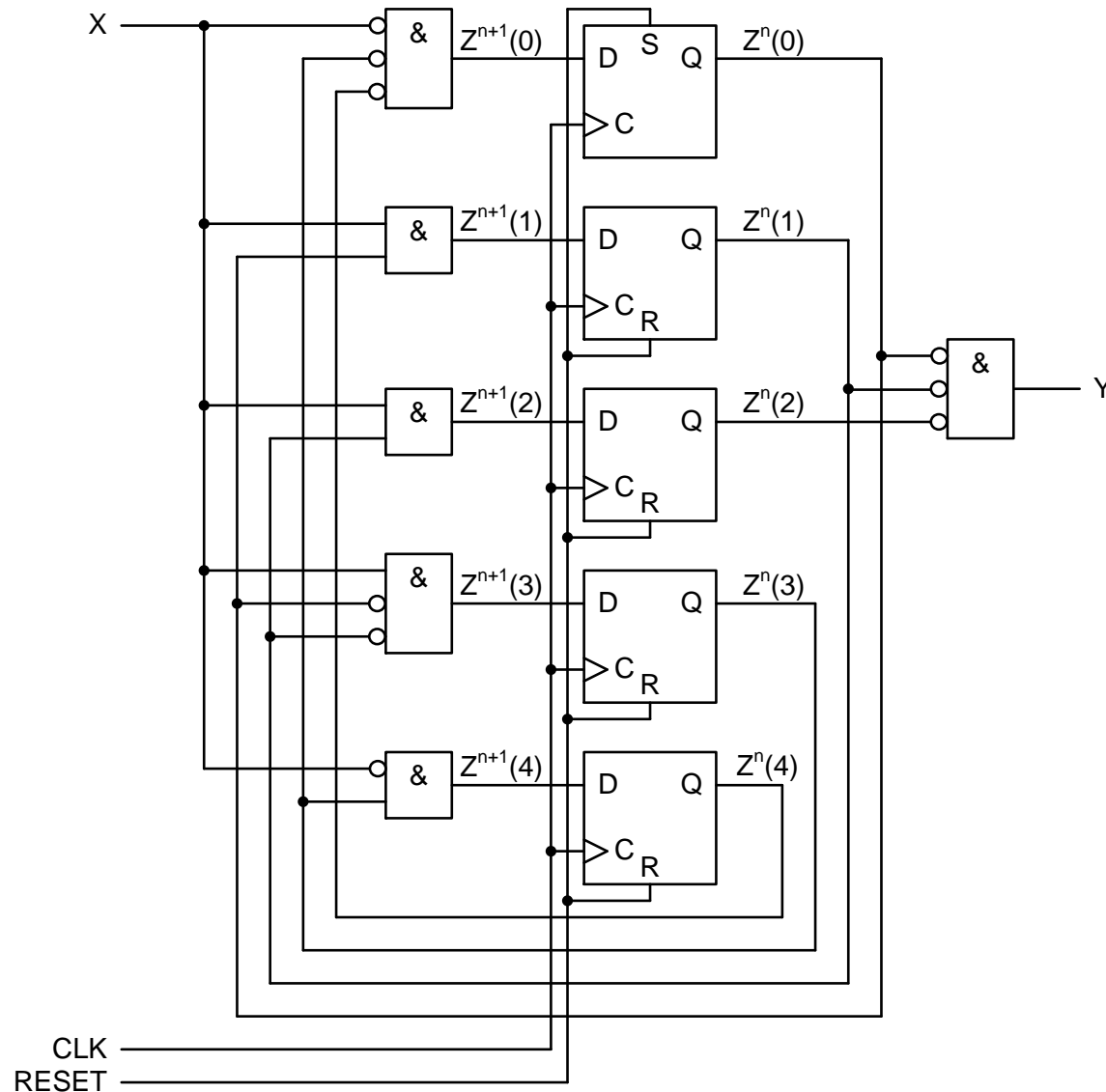
$$Z^{n+1}(2) = X \& Z^n(1)$$

$$Z^{n+1}(1) = X \& Z^n(0)$$

$$Z^{n+1}(0) = \bar{X} \& \overline{Z^n(4)} \& \overline{Z^n(3)}$$

$$Y = \overline{Z^n(2)} \& \overline{Z^n(1)} \& \overline{Z^n(0)}$$

„Entprellen eines Tasters“ mit redundanter Codewortlänge



Beachten Sie:
Initialisierung
setzt $Z^n(0)$ auf 1

Weitere Alternative: Optimierte Codierung

Als weitere Variante kann die Zustandsfolgertabelle genauer analysiert werden

- ① Drei der Zustände können nur bei $X=0$ als Folgezustand auftreten, die drei anderen Zustände nur bei $X=1$
- ② Für drei Zustände ist die Ausgabe $Y=0$, für die drei anderen Zustände ist $Y=1$.

s^n	s^{n+1}		Y
	$X=0$	$X=1$	
stabil 0*	stabil 0	1-mal 1	0
1-mal 1	stabil 0	2-mal 1	0
2-mal 1	stabil 0	stabil 1	0
stabil 1	1-mal 0	stabil 1	1
1-mal 0	2-mal 0	stabil 1	1
2-mal 0	stabil 0	stabil 1	1

* = Reset

Dies kann für die Codierung genutzt werden:

- ① Die Zustandsvariable $Z(0)$ wird entsprechend des Folgezustands gewählt.
 - Zustände die Folgezustand bei $X=0$ sind, werden mit $Z(0)=0$ codiert, die anderen Zustände (Folgezustand bei $X=1$) mit $Z(0)=1$.
- ② Die Zustandsvariable $Z(1)$ wird entsprechend des Ausgabewertes gewählt.
 - Zustände mit Ausgangswert $Y=0$, werden mit $Z(1)=0$ codiert, die Zustände mit $Y=1$, haben $Z(1)=1$ als Code.
- ③ Weitere Zustandsvariable werden frei gewählt, um eindeutige Codewörter zu erhalten

Optimierte Codierung (II)

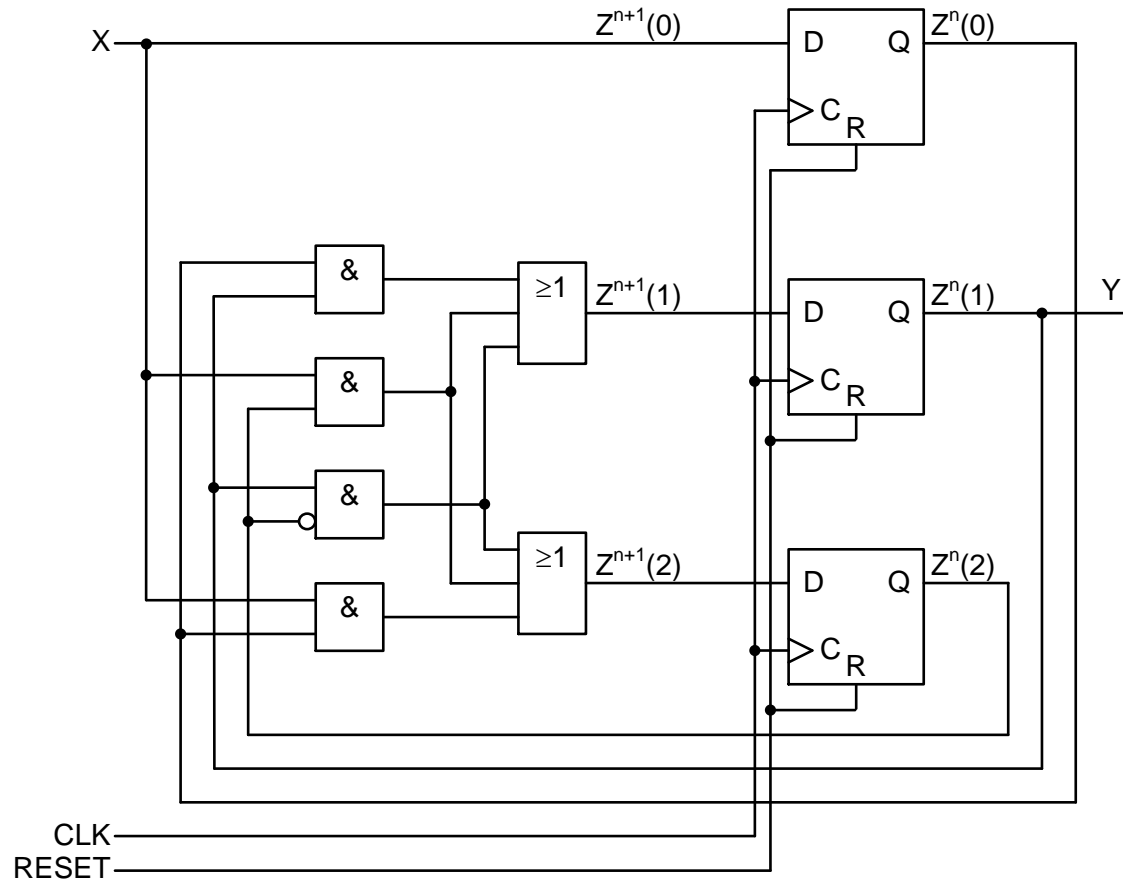
Codierung

s^n	$Z(2:0)$	
stabil 0	0 0 0	① Zustände, die bei $X=1'$ folgen
1-mal 1	0 0 1	
2-mal 1	1 0 1	
stabil 1	1 1 1	② Zustände mit Ausgabe $Y=1'$
1-mal 0	0 1 0	
2-mal 0	1 1 0	

Ergebnis der Optimierung

- ① Keine Logikgatter für $Z(0)$ erforderlich
- ② Keine Logikgatter für Y erforderlich

Schaltbild



Vergleich der Codierungen

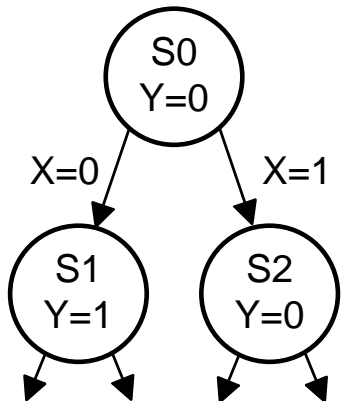
Drei Varianten der Codierung

- Minimaler Codewortlänge und einfache Durchnummerierung
- Redundanter Codewortlänge und One-Hot-Codierung
- Optimierte Codierung durch Analyse der Zustandsfolgetabelle
- **Alle Automaten sind äquivalent!**
 - Sie ergeben bei gleicher Eingabe auch die gleiche Ausgabe
 - Damit sind sie von außen nicht zu unterscheiden
- Eine „gute Codierung“ hängt ab von Struktur des Automaten, Anforderungen und Technologie der Schaltungsimplementierung
- Die Varianten sind Strategien, die zur Optimierung probiert werden können
- In der Praxis wird Aufwand zur Optimierung und erzielter Nutzen beachtet
 - ➔ Arbeitszeit für eine optimierte Codierung lohnt sich meist nicht
- Entwurf normalerweise in VHDL
 - ➔ Sie sollten Meldungen des Computer verstehen („Choosing One-Hot-Coding“)

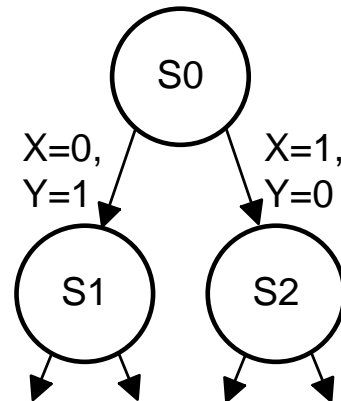
5.6 Entwurf von Mealy-Automaten

- Der Entwurf eines Mealy-Automaten gleicht in weiten Teilen dem eines Moore-Automaten
- Der wesentliche Unterschied ist, dass die **Ausgabe** nicht von den Zuständen sondern den **Zustandsübergängen** abhängt
 - ➔ Die Ausgabe wird darum nicht in die Zustandskreise sondern **an die Pfeile** der Zustandsübergänge eingetragen
- Dies bedeutet für die Zustandsfolgetabelle
 - ➔ Die Ausgabe hat keine eigene Spalte, sondern wird für **jeden Zustandsübergang** angegeben

Moore



Mealy



Moore

s^n	s^{n+1}		Y
	X=0	X=1	
S0	S1	S2	0
S1	S3	S4	1
...

Mealy

s^n	s^{n+1}, Y	
	X=0	X=1
S0	S1,1	S2,0
S1	S3,0	S4,1
...

Beispiel für einen Mealy-Automaten

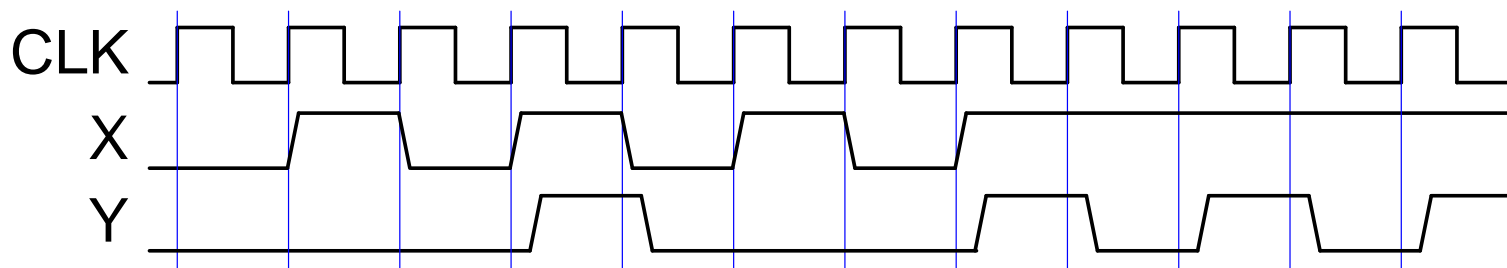
- Die Unterschiede erscheinen zunächst etwas formell
 - Sie eröffnen jedoch weitere Möglichkeiten für den Entwurf eines Automaten

Spezifikation eines Mealy-Automaten

Ein Automat soll die Anzahl von Takten mit dem Wert 1 halbieren. Wenn am Eingang X der Wert 1 anliegt, soll für jeden zweiten Wert eine 1, ansonsten eine 0 am Ausgang Y ausgegeben werden. Die Zählung soll durch Eingangswerte 0 nicht beeinflusst werden. Bei einer 0 am Eingang soll 0 ausgegeben werden.

Beim Einschalten soll für die erste 1 der Wert 0 ausgegeben werden.

Zeitdiagramm



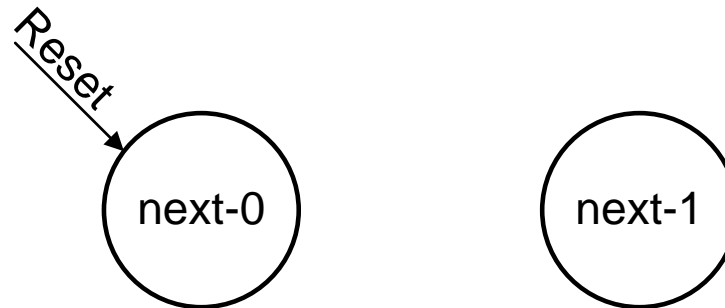
Mealy-Automat „Halbieren der 1-Werte“

Aufstellen der Zustandsfolgetabelle

- Welche Informationen muss sich der Automat merken?
- Grundidee:
 - Der Automat gibt nur jede zweite 1 am Eingang weiter und unterdrückt die jeweils andere 1.
 - Er muss sich merken, ob die nächste 1 weitergegeben oder unterdrückt wird.
- Start mit zwei Zuständen:
 - next-0: Die nächste 1 am Eingang wird unterdrückt.
(Startzustand laut Spezifikation)
 - next-1: Die vorherige 1 wurde unterdrückt, also wird die nächste 1 des Eingangs an den Ausgang weitergegeben.

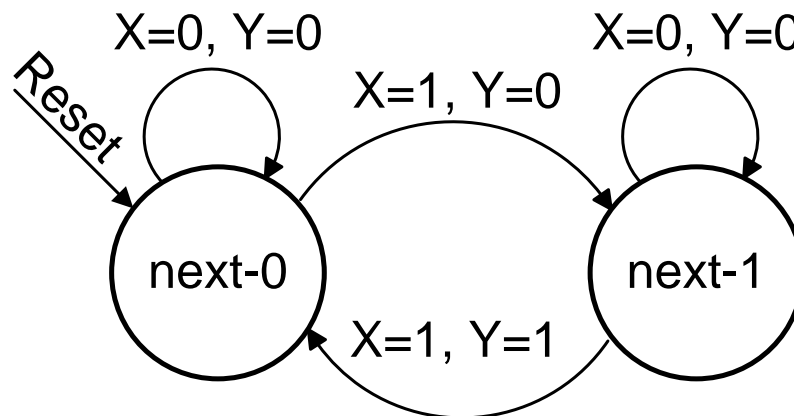
Beachten Sie:

- Für die Zustände ist keine Ausgabe definiert, da ein Mealy-Automat entworfen wird



Mealy-Automat „Halbieren der 1-Werte“ (II)

- Für beiden Zustände wird nun überlegt, was laut Spezifikation im Falle der Eingaben $X=0$ und $X=1$ passieren muss.
 - Für $X=0$ wird eine 0 ausgegeben. Das Zählen der 1-Werte wird nicht beeinflusst, der Automat ändert seinen Zustand nicht.
 - Für $X=1$ gibt es zwei Fälle:
 - Bei next-0 wird die 1 unterdrückt, also 0 ausgegeben. Der Automat merkt sich, die nächste 1 wird weitergegeben und wechselt nach next-1.
 - Bei next-1 wird die 1 weitergegeben, also 1 ausgegeben. Der Automat merkt sich, die nächste 1 wird unterdrückt und wechselt nach next-0.



Mealy-Automat „Halbieren der 1-Werte“ (III)

- Für alle Zustände sind beide möglichen Folgezustände definiert
 - Das Zustandsfolgediagramm ist komplett
 - Es werden zwei Zustände benötigt, die sich nicht zusammenfassen lassen
- Der Unterschied zum Moore-Automaten ist die Definition der Ausgangswerte
- Hier sind die Ausgänge für die Zustandsübergänge, also für die Pfeile definiert

Zustandsfolgetabelle

- Die Zustandsfolgetabelle wird direkt aus dem Diagramm erstellt
- Die Ausgabe ist abhängig von Zustand und Eingang.
- Darum wird sie mit dem Folgezustand in der Form „ s^{n+1}, Y “ für jede Spalte angegeben.

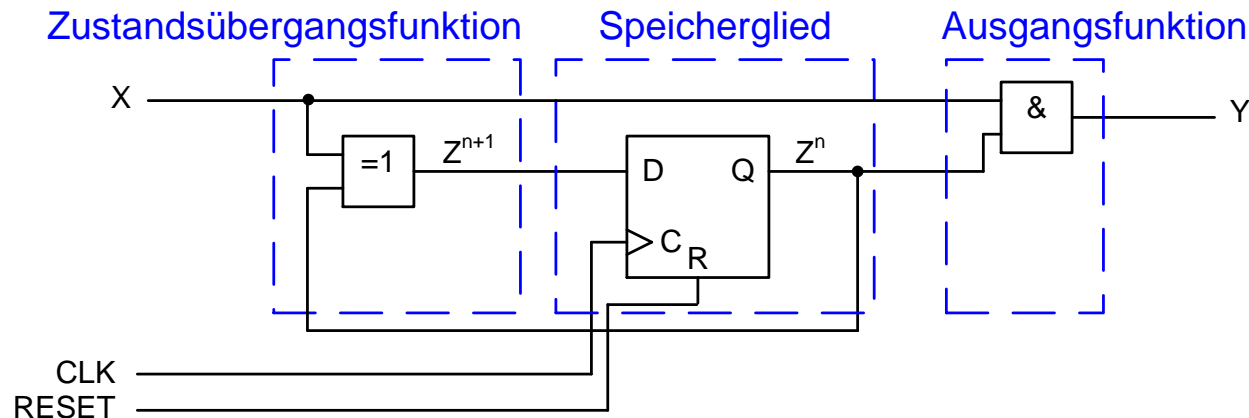
s^n	s^{n+1}, Y	
	$X=0$	$X=1$
next-0 *	next-0, 0	next-1, 0
next-1	next-1, 0	next-0, 1

* = Reset

Mealy-Automat „Halbieren der 1-Werte“ (IV)

Codierung der Zustände

- Zwei Zuständen, darum Codewort mit nur einer Stelle
- Kaum Optionen für Codierung, darum
 - next-0: $Z=0$
 - next-1: $Z=1$
- Implementierung mit 1 FF, 1 XOR-Gatter, 1 UND-Gatter



Vereinfachte Darstellung des Zustandsfolgediagramms

- Bei komplexen Automaten sind die Zustandsfolgediagramme ebenfalls komplex und können daher unübersichtlich werden
- Zur besseren Darstellung sind Vereinfachungen möglich
- ① Ein- und Ausgänge müssen nicht mit X, Y bezeichnet werden, sondern können Abkürzungen entsprechend der Spezifikation haben, z.B. A1, A2, P, T
- ② Eingangs- und Ausgangswerte müssen nicht stets neu benannt werden, sondern können in einer festen Reihenfolge angegeben werden
Empfohlene Reihenfolge: Eingangswerte, Schrägstrich, Ausgangswerte
- ③ Haben mehrere Eingangskombinationen denselben Folgezustand, reicht ein gemeinsamer Übergangspfeil
- ④ Zustände können einfach durchnummeriert werden (S0, S1, ...), die Bedeutung wird als Liste dokumentiert.

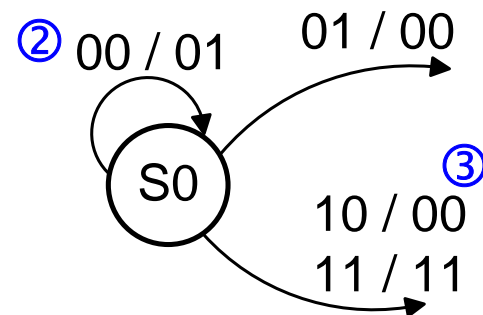
Ein- /Ausgänge

A1,A2 / P,T ①

Zustände:

S0 - Start ④

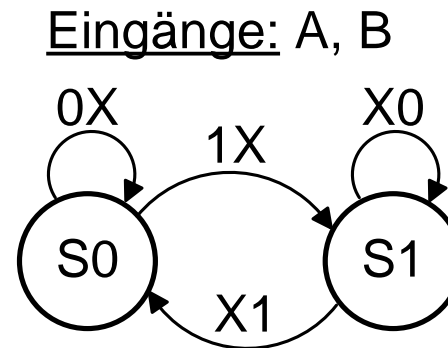
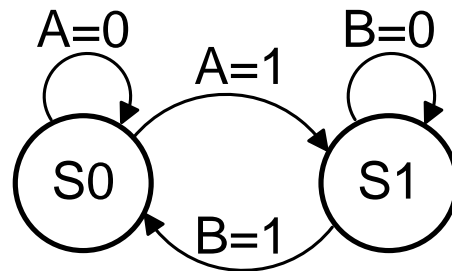
S1 - ...



Vereinfachte Darstellung des Zustandsfolgediagramms (II)

Auch Vereinfachung für Zustandsübergänge möglich

- Wenn Zustandsübergang nur ein Teil der Eingangsvariablen beachten muss:
 - Angabe der erforderliche Eingabe (links)
 - Nicht erforderliche Eingabe wird mit ‚X‘, für "Eingang beliebig" bezeichnet (rechts)



Achtung:

- Alle 2^n Eingangskombinationen bei n Eingangswerten berücksichtigen
- Das Zustandsfolgediagramm darf nicht kryptisch kurz werden. In der Praxis muss man nach 2 Wochen, 2 Monaten oder 2 Jahren das Diagramm noch lesen und **verstehen** können.

Vergleich von Mealy- und Moore-Automat

Vorteil Moore-Automat

- Weniger Ausgabewerte definiert, darum übersichtlicher
- Höhere Wartbarkeit, also Möglichkeit einen Entwurf später einmal zu ändern und anzupassen

Vorteil Mealy-Automat

- Höhere Reaktionsgeschwindigkeit möglich
 - Z.B. Weitergabe der „halbierten Pulse“ noch im gleichen Takt
 - Kann für Bussysteme (z.B. PCI-Bus im PC) erforderlich sein

Verwendung beim Automatenentwurf

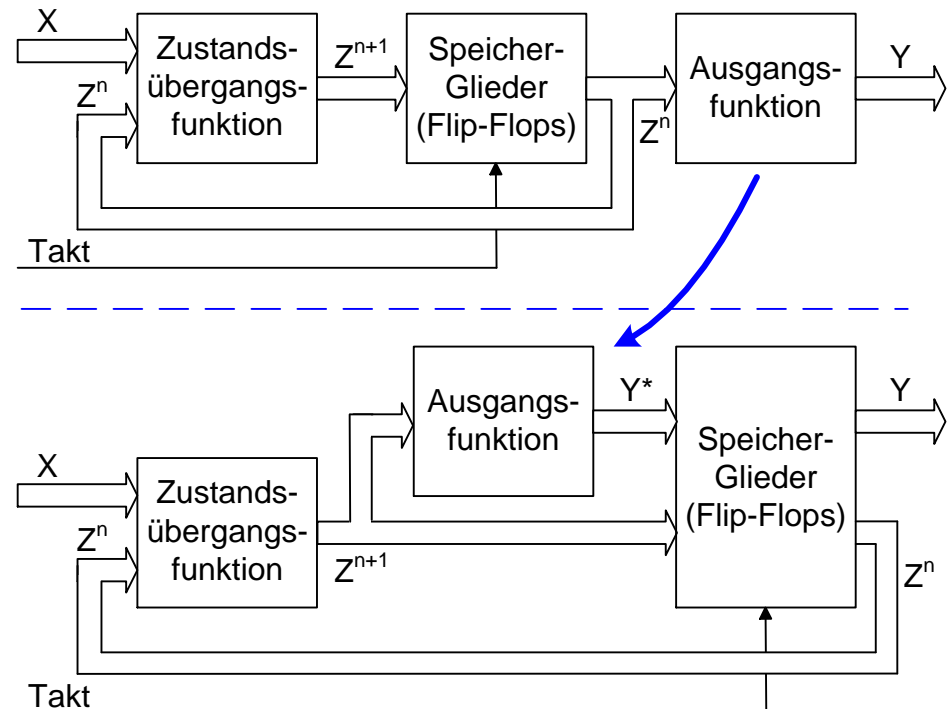
- Für den Normalfall wird empfohlen einen Moore-Automaten zu entwerfen
- Wenn der Automat noch im gleichen Taktzyklus eine Antwort ausgeben muss, empfiehlt sich der Einsatz eines Mealy-Automaten

Kombinatorische Ausgabe und Registerausgabe

- Für die Automaten erfolgt die Ausgabe der Signalwerte Y nach einer kombinatorischen Verknüpfung
- In der Praxis ist es vorteilhaft, wenn Teilschaltungen klare Schnittstellen zu den folgenden Teilschaltungen haben
- Daher oft als **Taktkonzept**: Ausgänge von Teilschaltungen müssen immer aus einem Flip-Flop stammen (Registerausgabe)

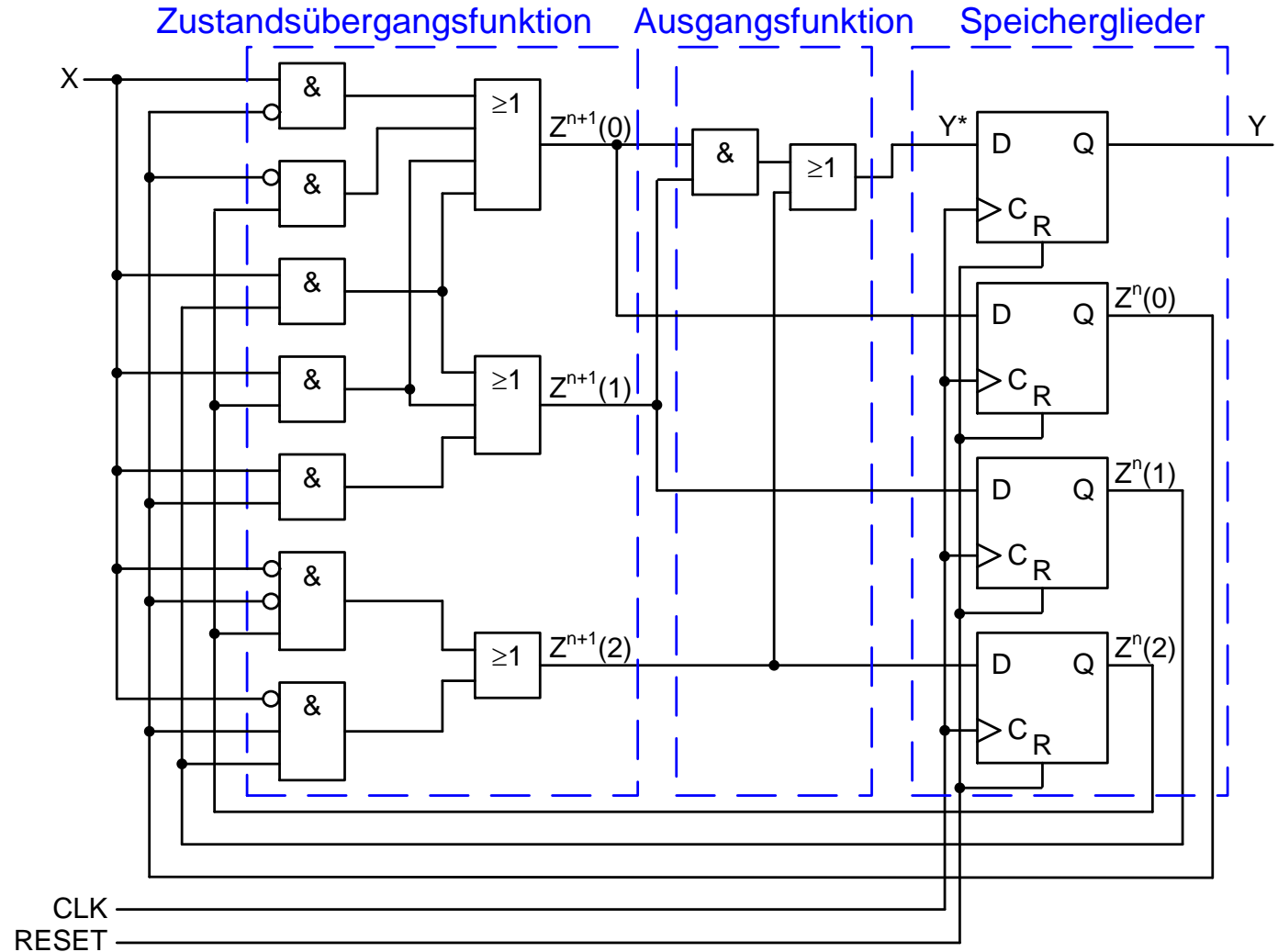
Umsetzung durch Veränderung des Blockschaltbilds

- Ausgangsfunktion wird mit neuer Zustandsvariable Z^{n+1} berechnet
- Ergebnis Y^* der Ausgangsfunktion wird in Registerstufe gespeichert
 - Die Ausgangsfunktion wird also vor die Flip-Flops geschoben
- Beide Moore-Automaten sind äquivalent, haben also die gleiche logische Funktion



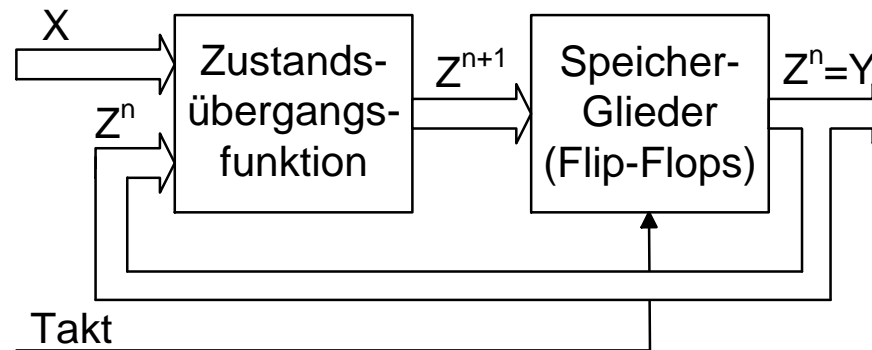
Beispiel für Registerausgabe

➤ Vergleiche Folie 50



Registerausgabe mit Medwedew-Automat

- Der Medwedew-Automat ist ein Spezialfall des Moore-Automaten
 - Die Ausgangsvariablen Y sind gleich den Zustandsvariablen Z^n .
 - ➔ Für den Medwedew-Automat sind darum keine weiteren Ausgangs-Flip-Flops erforderlich



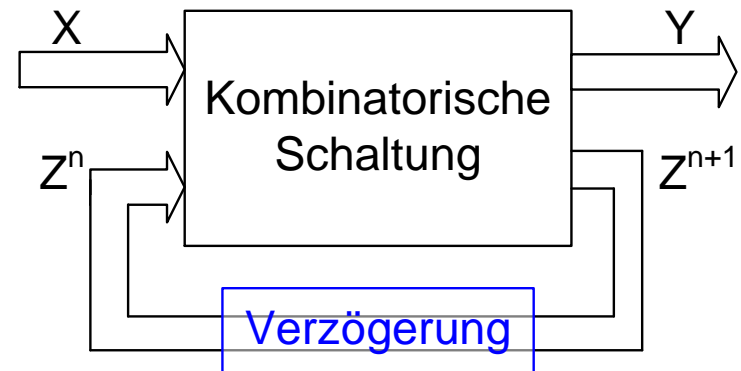
- Für bestimmte Anwendungen lässt beim Moore-Automaten einplanen, dass die Zustandsvariablen auch als Ausgangsvariablen verwendet werden
 - Entprell-Automaten mit optimierter Codierung
 - Zähler, der nacheinander Zahlenwerte ausgibt, wie 0, 1, 2, 3, ... Diese Zahl wird als Zustand gespeichert und ist die Ausgabe.
- Medwedew-Automat nicht unbedingt verlangt, denn Aufwand für zusätzliche Ausgangs-Flip-Flops ist oft akzeptiert

5.7 Asynchrone Automaten

- Bei asynchronen Automaten sind keine Flip-Flops zur Datenspeicherung vorhanden
- Die Zustandsinformation wird direkt vom Ausgang des Schaltnetzes mit dem Eingang verbunden
- Die Speicherung der Information findet in der Verzögerung des Schaltnetzes und der Verbindungsleitungen statt

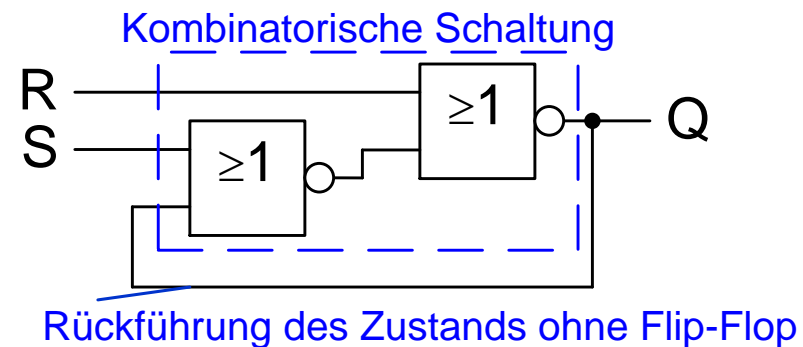
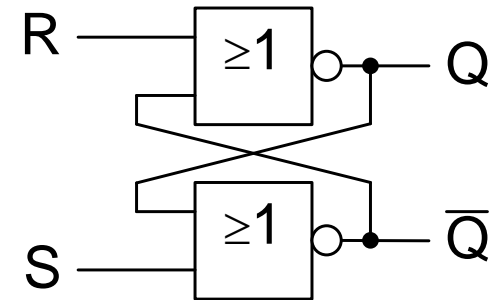
Vorteile:

- Höhere Geschwindigkeit
 - Der Takt muss nicht auf den langsamsten Fall warten
- Niedrigerer und gleichmäßigerer Stromverbrauch
 - Bei synchronen Schaltungen schalten bei den Taktflanken Tausende von FFs gleichzeitig
- Geringere Störausstrahlung, da kein Takt



Beispiel eines asynchronen Automaten

- Das RS-Flip-Flop ist ein Beispiel für einen asynchronen Automaten
- Das üblicherweise gezeichnete Schaltbild kann umgezeichnet werden
- Das Signal Q wird als Zustand vom Schaltnetz erzeugt und wieder als Eingang des Schaltnetzes verwendet



Entwurf asynchroner Automaten

- Beim Entwurf asynchroner Automaten sind wesentlich mehr Bedingungen zu beachten als bei synchronen Automaten
 - Das System ist nur stabil, wenn die Änderung einer Zustandsvariablen durch einen Eingang nicht immer wieder zu Änderungen von Zustandsvariablen führt
 - Ansonsten kann der Automat schwingen
 - Die kombinatorische Schaltung darf keine kurzzeitigen Zwischenwerte ausgeben
 - Ansonsten kann der Automat in einen falschen Zustand übergehen
 - Aufgrund des komplexeren Entwurfs sind asynchrone Automaten auch wesentlich **schwieriger zu entwerfen**
 - Da Fehler nur schwer entdeckt werden können, ist das **Risiko beim Entwurf** eines asynchronen Automaten relativ hoch
- ➔ In der Praxis werden asynchrone Automaten so gut wie nicht entworfen.
- Nur bewährte und besonders geprüfte Grundsaltungen wie RS-Flip-Flop oder andere Flip-Flops werden eingesetzt

Es gibt regelmäßig (vergebliche) Vorschläge, für den Einsatz asynchroner Automaten

5.8 Automaten in VHDL beschreiben

- Automaten, die als **Zustandsübergangsdiagramm** vorliegen, können **direkt** in VHDL beschrieben werden

Für den Zustand wird ein eigener Datentyp mit vier möglichen Werten definiert
Achtung: Hier wird nur der Datentyp selbst, aber noch kein Signal definiert

```
architecture behave of debounce is
  type state_type is (stabil_0, einmal_1, zweimal_1,
                     stabil_1, einmal_0, zweimal_0);
  signal state      : state_type;
begin
  process
  begin
    wait until rising_edge(clk);
    if (reset='1') then
      y <= '0';
      state <= stabil_0;
    else
      case [...]
    end if;
  end process;
```

Ein Signal mit dem neuen Datentyp wird definiert

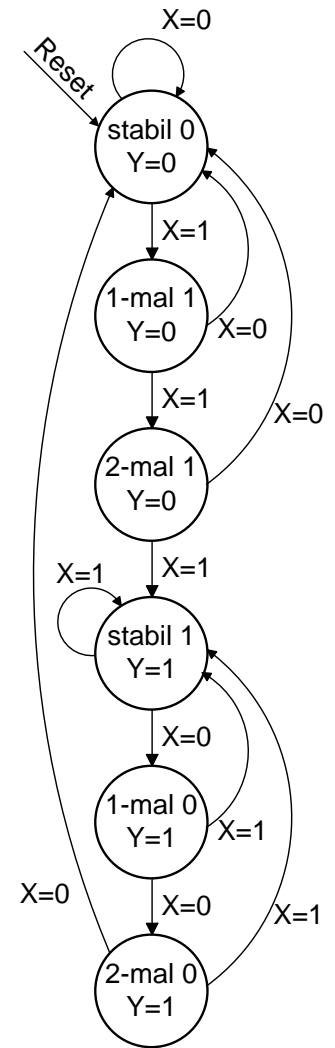
In einem getakteten Prozess werden in einer case-Anweisung die Zustandsübergänge beschrieben
(siehe nächste Folie)

Automaten in VHDL beschreiben (II)

```
case state is
  when stabil_0 =>
    if (x='0') then
      state <= stabil_0;  y <= '0';
    else
      state <= einmal_1;  y <= '0';
    end if;
  when einmal_1 =>
    if (x='0') then
      state <= stabil_0;  y <= '0';
    else
      state <= zweimal_1; y <= '0';
    end if;
  when zweimal_1 =>
    [...]
end case;
```

Beschreibung entspricht
Registerausgabe (Folie 66, 67)

Für jeden Zustand wird
Folgezustand und dessen
Ausgabe beschrieben



Automaten in VHDL beschreiben: Komplettes Beispiel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity entprell is
    port ( clk      : in  std_logic;
          reset    : in  std_logic;
          x        : in  std_logic;
          y        : out std_logic);
end entprell;

architecture behave of entprell is

    type state_type is (stabil_0, einmal_1, zweimal_1,
                        stabil_1, einmal_0, zweimal_0);
    signal state      : state_type;

begin

process
begin
    wait until rising_edge(clk);
    if (reset='1') then
        y <= '0';
        state <= stabil_0;
    else
        case state is
            when stabil_0 =>
                if (x='0') then state <= stabil_0; y <= '0';
                else           state <= einmal_1; y <= '0';
            end if;
        end case;
    end if;
end process;
```

[...]

Automaten in VHDL beschreiben: Komplettes Beispiel (II)

```
[...]
    when einmal_1 =>
        if (x='0') then state <= stabil_0; y <= '0';
        else state <= zweimal_1; y <= '0';
        end if;
    when zweimal_1 =>
        if (x='0') then state <= stabil_0; y <= '0';
        else state <= stabil_1; y <= '1';
        end if;
    when stabil_1 =>
        if (x='1') then state <= stabil_1; y <= '1';
        else state <= einmal_0; y <= '1';
        end if;
    when einmal_0 =>
        if (x='1') then state <= stabil_1; y <= '1';
        else state <= zweimal_0; y <= '1';
        end if;
    when zweimal_0 =>
        if (x='1') then state <= stabil_1; y <= '1';
        else state <= stabil_0; y <= '0';
        end if;
    end case;
end if;
end process;
end behave;
```

Screenshots: Altera Quartus II

- 6 Zustände

← 6 Stellen der Codierung

