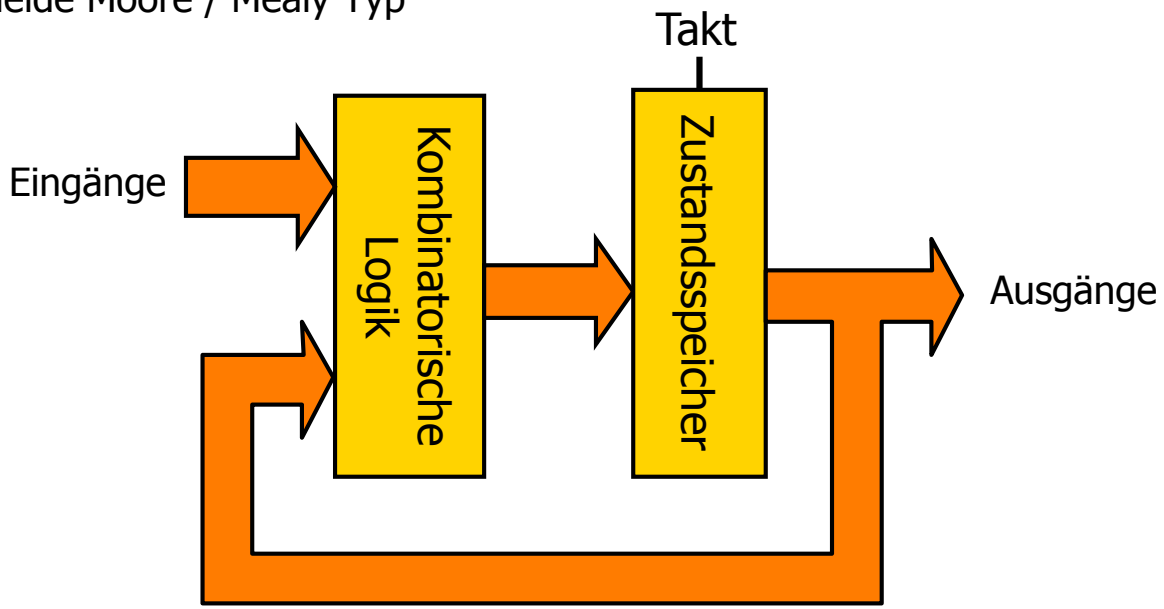
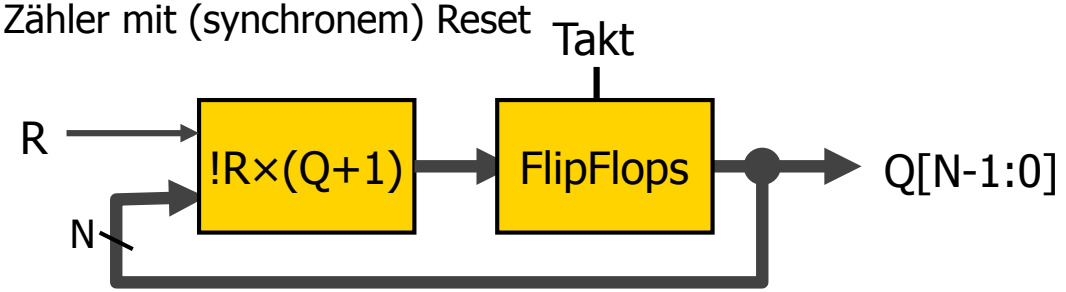
Getaktete Schaltungen

Sequentielle Logik

- Zum Speichern des Zustands eines Systems sind **Speicherelemente** notwendig
- Abhängig vom Zustand des Systems und von Eingangsvariablen soll sich der Zustand zu einem bestimmten Zeitpunkt (gegeben durch ein Taktsignal) ändern ⇒ **Zustandsmaschine** ('State machine').
- Später: Unterscheide Moore / Mealy Typ
- Beispiel:



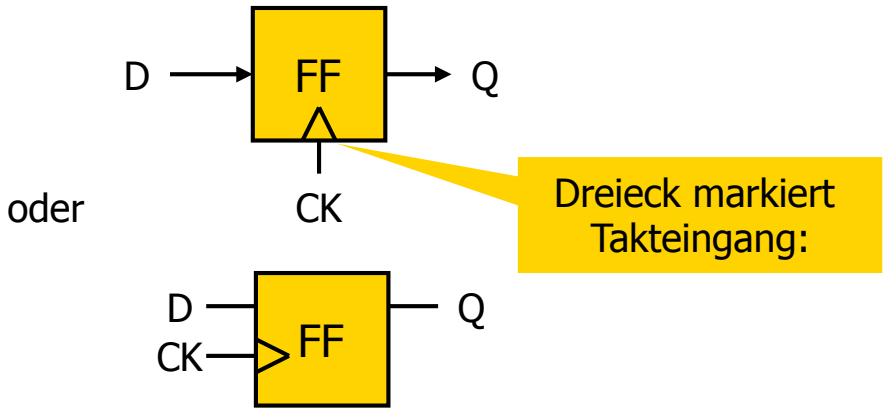
- Einfaches Beispiel: N Bit Zähler mit (synchronem) Reset



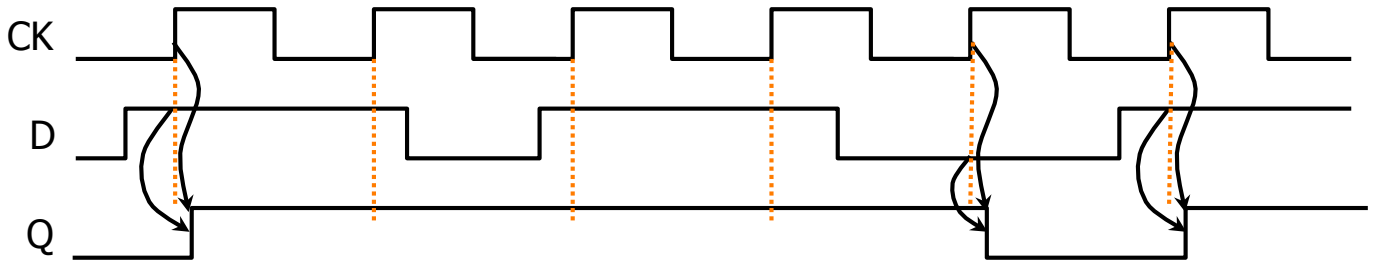
Wdh: Flankengetriggerte Flip-Flops

- Ein D-Flipflop überträgt den Wert seines
  - **D-Eingangs** an den
  - **Q-Ausgang** bei der (steigenden) Flanke (d.h. beim 0⇒1 Übergang) des Taktes
  - **Takt CK**

- Schaltsymbol

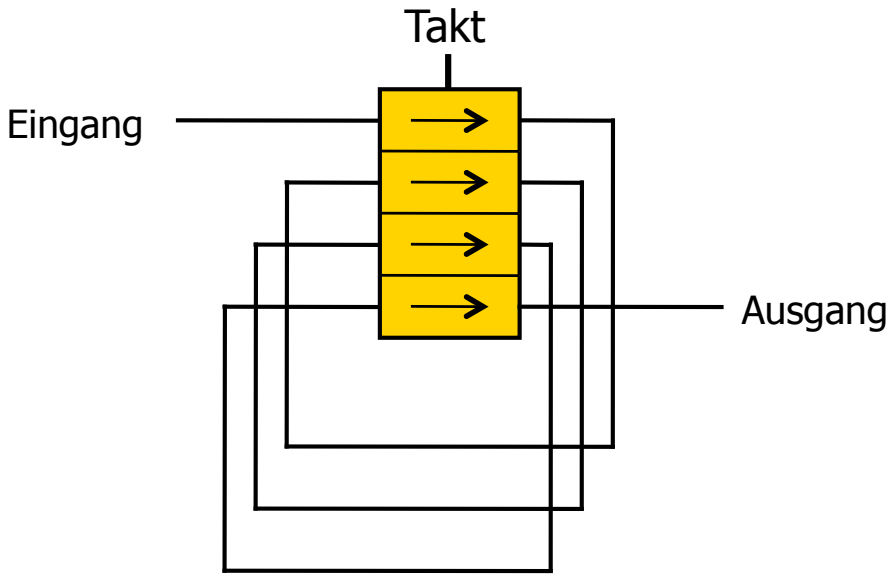


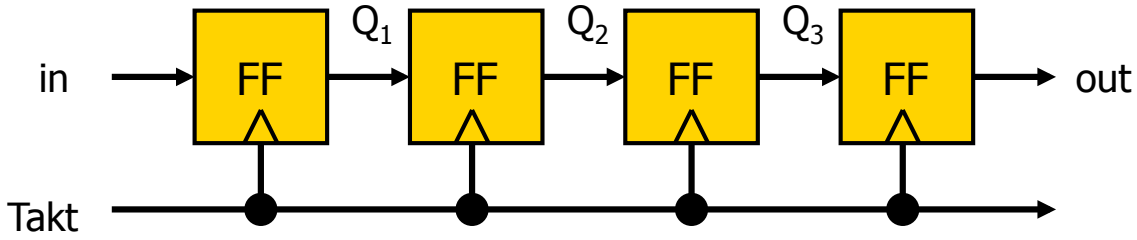
- Zeitliches Verhalten (Triggerung auf positive Taktflanke):



Schieberegister

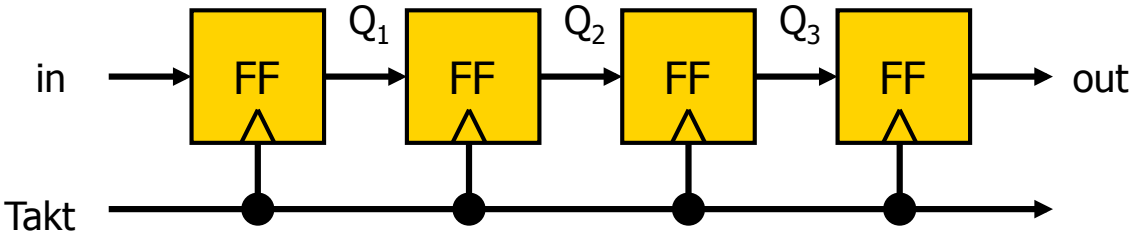
- Sehr einfach: Keine Logik, ein Eingang, ein Ausgang

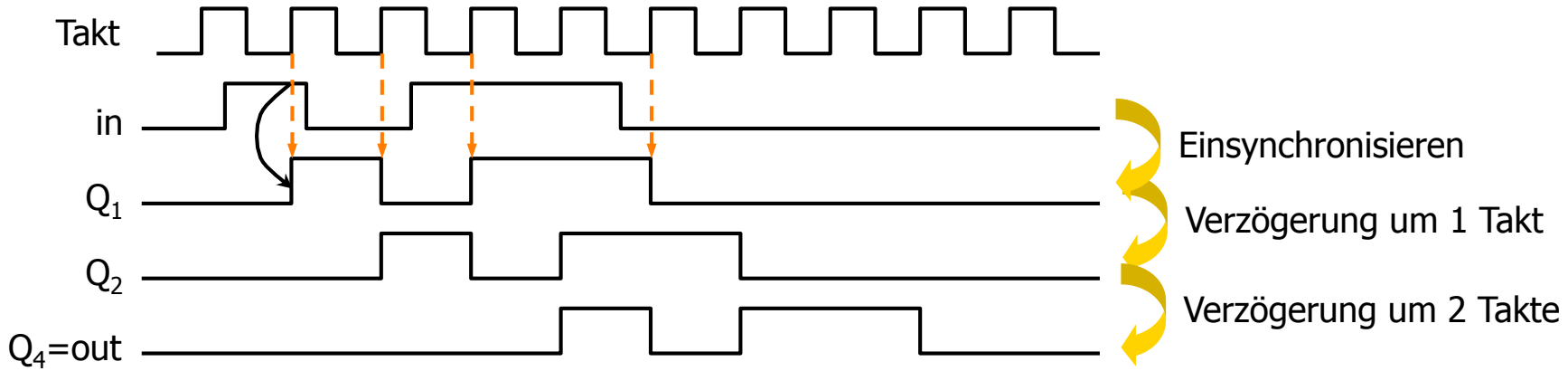




Schieberegister

- Schieberegister entstehen durch Hintereinanderschalten von FFs.
  Zwischen den Stufen ist keine (wenig) Logik:



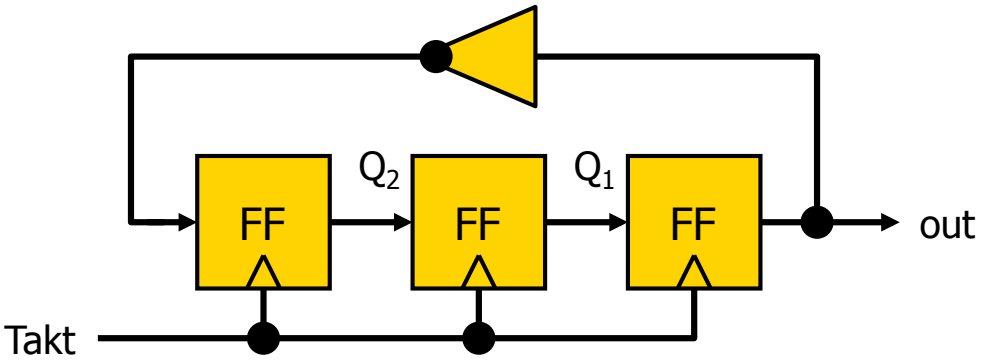


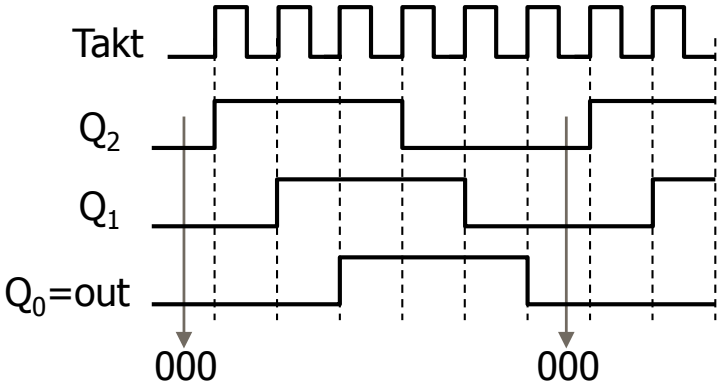
- **Vorsicht**: Die Hold-Zeit kann leicht verletzt sein. Daher fügt man manchmal Verzögerungen (Inverterketten) in den Datenpfad ein.
- Anwendungen:
  - Verzögerung von Signalen (z.B. bei Pipelining)
  - Einfache Zustandscodierung
  - spezielle Zähler (mit Rückkopplung)

‚Zähler' aus Schieberegistern: Johnson Zähler

- Sehr **einfach aufgebaute Zähler** werden durch **Linear Feedback Shift Register (LFSR)** erzeugt
- Das Zurücksetzen in einen Anfangszustand kann durch sync/async. Reset der FFs erfolgen

- Beim ‚Johnson Zähler' wird der Ausgang über einen Inverter zum Eingang rückgekoppelt.
- Der Zähler hat dadurch 2N Zustände



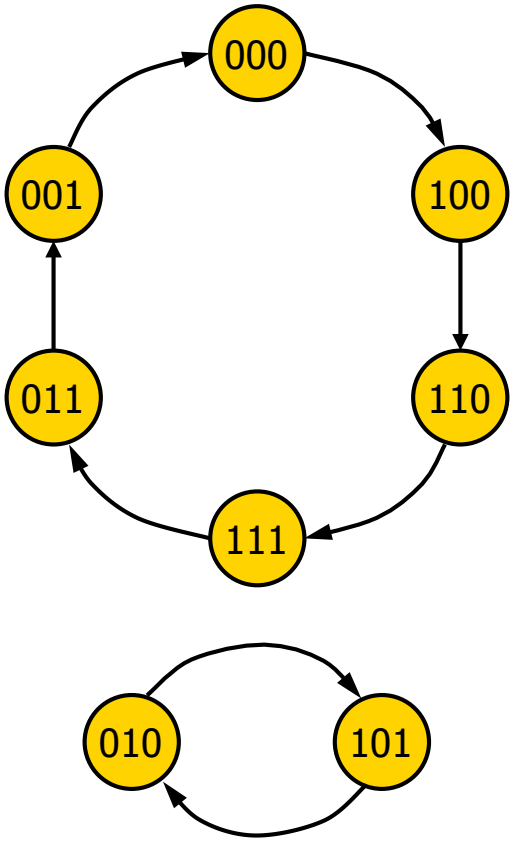


- Sehr einfache Aufbau
- Der Zählerstand kann durch die Abfrage von nur 2 (geeigneten) Bits ermittelt werden (bei einem Binärzähler müssen alle Bits einbezogen werden).
  Dies ist z.B. in FPGAs vorteilhaft, da die Logikblöcke dort nur ein begrenztes Fan-In haben.
- Beispiel für N=5:

Q0	0	1	1	1	1	1	0	0	0	0	…
Q1	0	0	1	1	1	1	1	0	0	0	…
Q2	0	0	0	1	1	1	1	1	0	0	…
Q3	0	0	0	0	1	1	1	1	1	0	…
Q4	0	0	0	0	0	1	1	1	1	1	…

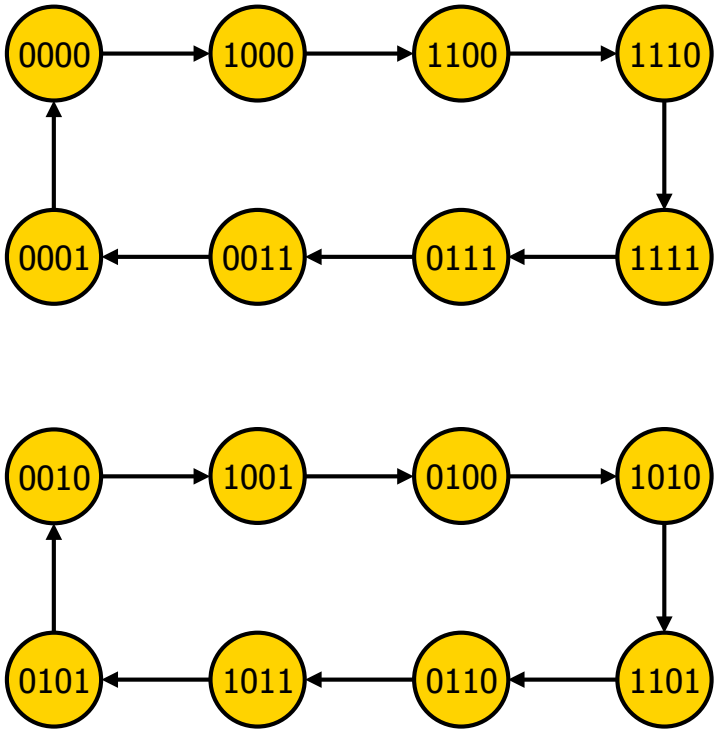
Johnson Zähler: Sprungdiagramm

- Bei N=3 gibt es $2^3 = 8$ mögliche Zustände.
- 6 davon werden vom Johnson Zähler durchlaufen:



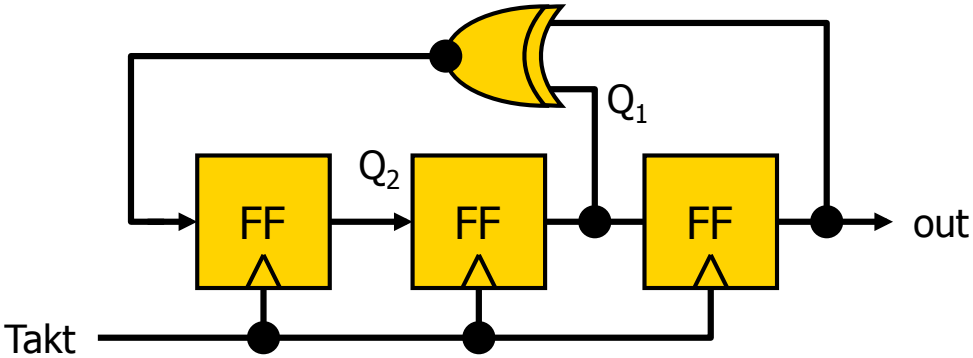
- Die verbleibenden beiden Zustände bilden einen eigenen Zyklus.
- Man muss mit einem Reset vermeiden hier zu starten!

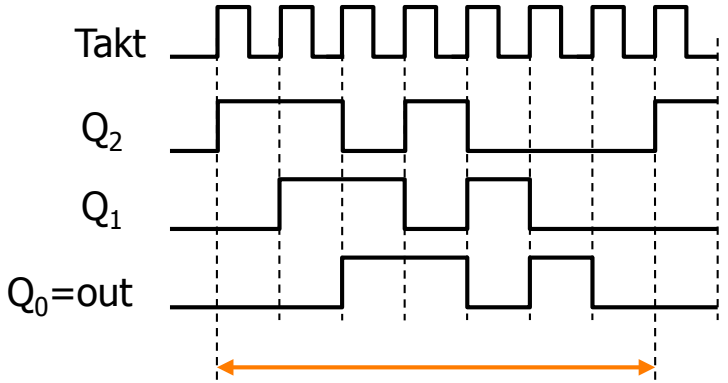
- Bei N=4:



Zähler aus Schieberegistern: PRBS

- Durch Rückkopplung des Ausgangs und eines (oder mehrerer) geeigneten Abgriffs (‚tap') kann bei N Flipflops eine Bitsequenz mit der Periode 2^N-1 entstehen (‚maximum length')
- Die Bitsequenz hat keine erkennbare Struktur und wird daher als Pseudo-Random-Bit-Sequence (PRBS) bezeichnet



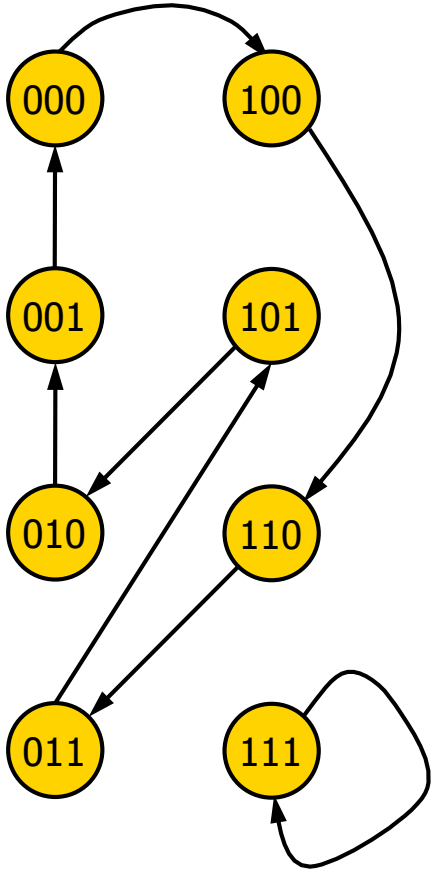


- Einige Eigenschaften:
  - In der gesamten Sequenz kommt nur genau eine Eins weniger vor als Nullen
  - Die Hälfte aller zusammenhängenden Einser-Blöcke ist einen Takt lang, ein Viertel ist zwei Takte lang, etc. (bis auf maximale Sequenzen von Einsen). Gleiches gilt für die Nullen. Beispiel: N=6, Periode = 63:
    000000111110111100111010110000101110001101101001000100110010101
  - Die Autokorrelationsfunktion ist 1/Periode für jede beliebige zeitliche Verschiebung der Sequenz.
  - Das Rauschspektrum ist konstant (‚weiß') innerhalb von 0.1dB bis zu 0.12xf_{clock}
  - Rückrechnen vom Bitmuster auf die Anzahl Takte ist sehr rechenaufwändig
  - Noch ein Beispiel: N=15.
    Start: 000000000000000111111111111110111111111111100111111111111…
    Nach 5000 Takten:
    100100101000100010010000110011001001110101010100101100000000010001011111111100110…

N	Abgriffe	Länge	Maximal ?
3	Q_1	7	ja
4	Q_1	15	ja
5	Q_2	31	ja
15	Q_1	32767	ja
16	Q_{12},Q_3,Q_1	65535	ja
16	Q_7		96.8%
39	Q_4	$5x10^{11}$	

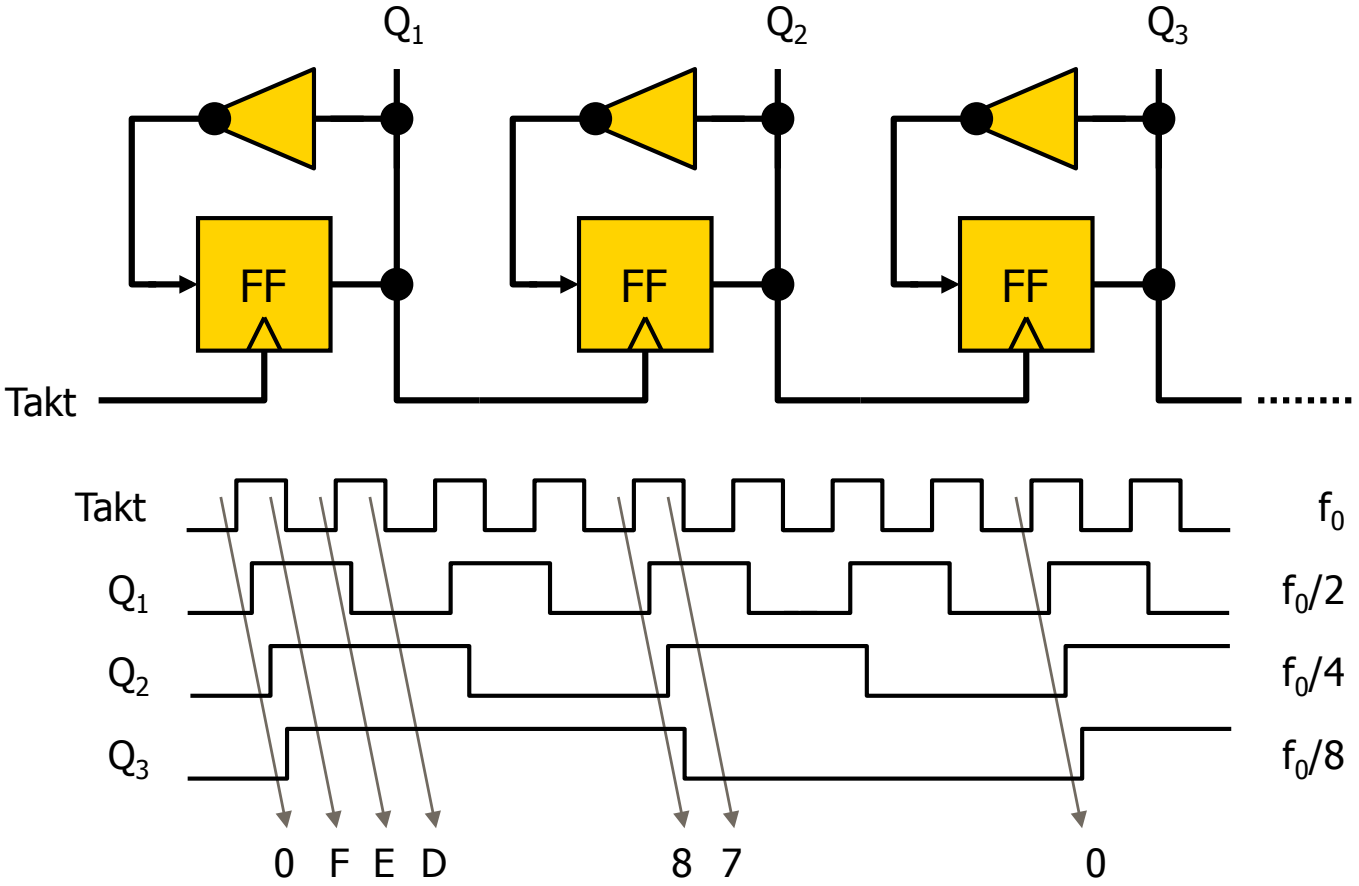
Max. Length LFRS: Sprungdiagramm

- Bei N=3 gibt es $2^3 = 8$ mögliche Zustände.
- 7 davon werden durchlaufen
- Zustand 111 ist (bei XOR feedback immer) stabil



Asynchrone Binärzähler (Ripple Counter)

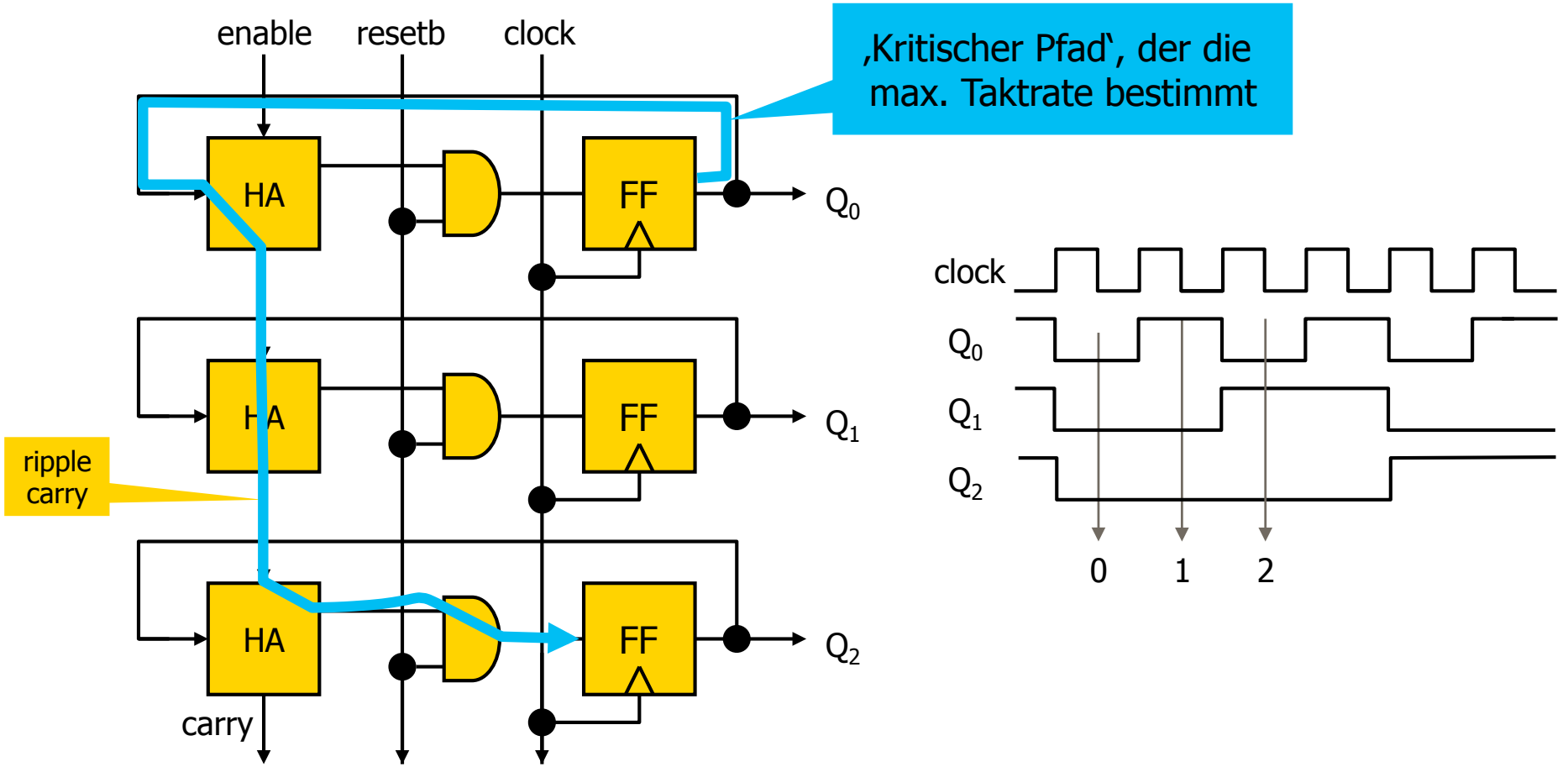
- Rückkopplung von !Q auf D erzeugt 'Toggle-FFs', die bei jedem Takt den Zustand ändern (0→1→0→...)
- Der Q-Ausgang eines Bits steuert das nächste Bit an (hier Rückwärtszähler):



- Wegen der Verzögerung der einzelnen Stufen sind die Flanken **nicht gleichzeitig** (daher *async*. Zähler)
- Sollte daher normalerweise vermieden werden.
- Anwendung: Frequenzteiler

Synchrone Binärzähler

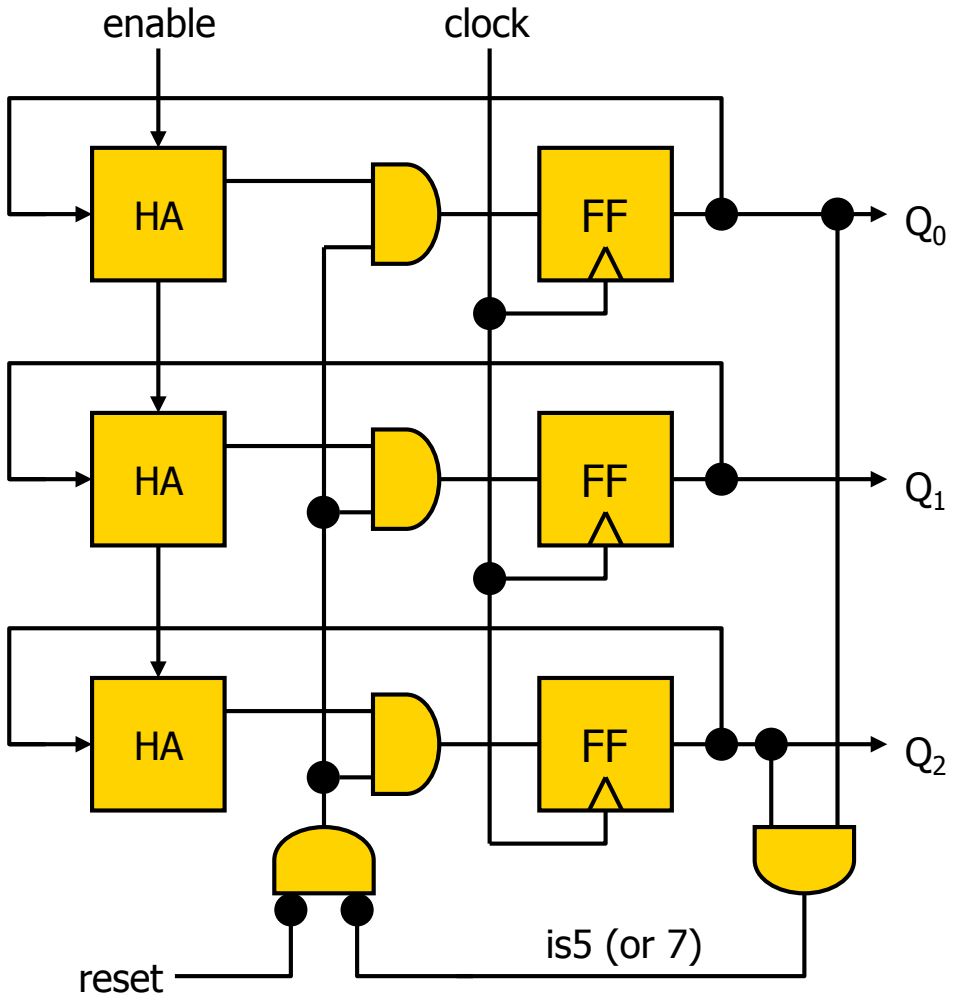
- Alle FFs werden gleichzeitig getaktet
- Die Eingänge werden so beschaltet, daß sich (z.B.) aufsteigend Binärzahlen ergeben
- Implementierung mit Halbaddierern (mit enable und reset):

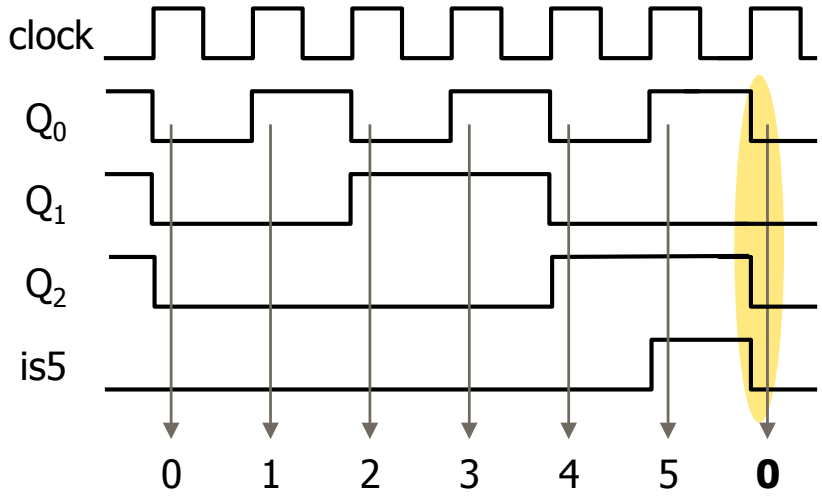


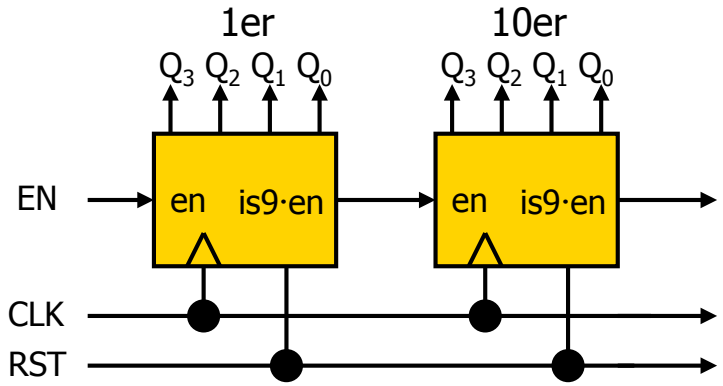
- Max. Taktfrequenz ist durch die Laufzeit des 'ripple' Carry begrenzt. Schnellere Zähler sehen wir später...

Kürzere synchrone Binärzähler (z.B. BCD Zähler)

- Gibt man das (synchrone) Reset-Signal bei einem bestimmten Zählerstand, so wird die Periode verkürzt.



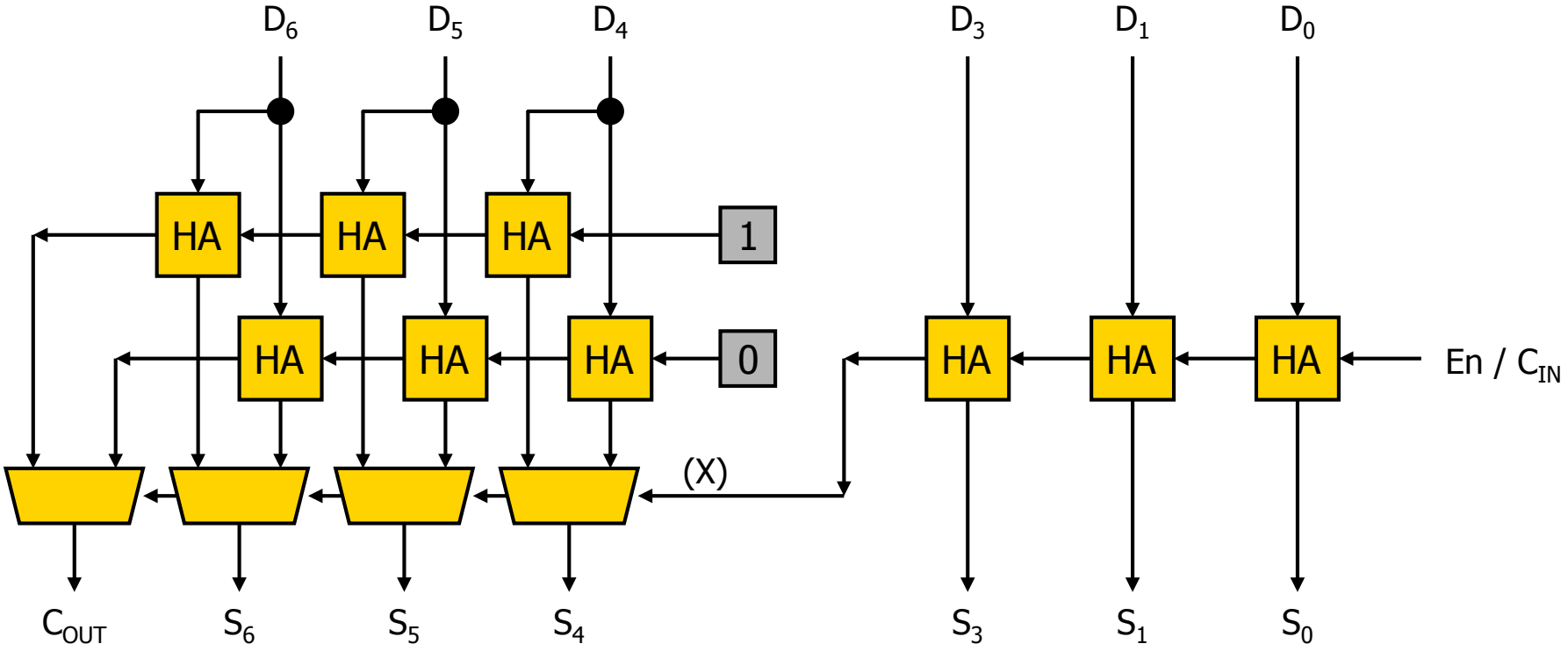




- Anwendung: BCD Zähler (Periode 10). 'is9 · en' gibt nächste Stufe frei.

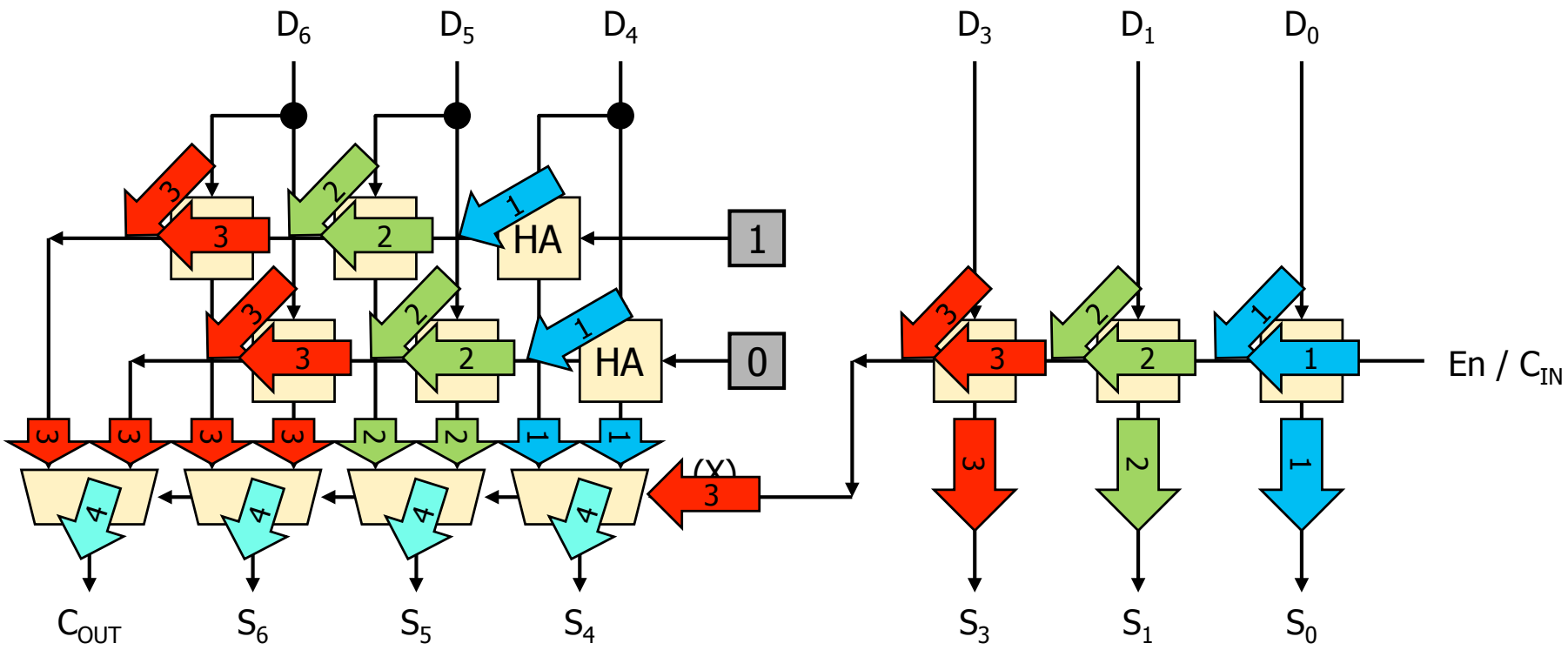
Vorgriff: Schnellere Zähler / Addierer

- Bei sehr großen Wortbreiten N muss das Carry-Signal sehr lange durch den Halbaddierer rippeln (N Stufen) und die Schaltung wird langsam.
- Es gibt viele Tricks, um das zu beschleunigen, z.B. den *Carry-Select* Addierer:
  - Berechne für Gruppen von Bits das C_{OUT} unter den ZWEI Annahmen C_{IN} = 0 oder C_{IN} =1. Das benötigt ZWEI Addierer.
  - Das C_{OUT} (X) der vorangehenden Gruppe wählt dann aus, welches Ergebnis benutzt wird
  - Im Fall von zwei Gruppen a N/2 reduziert sich der Delay auf etwa N/2+1
- Schlauer ist z.B. der *Carry-Lookahead* Addierer



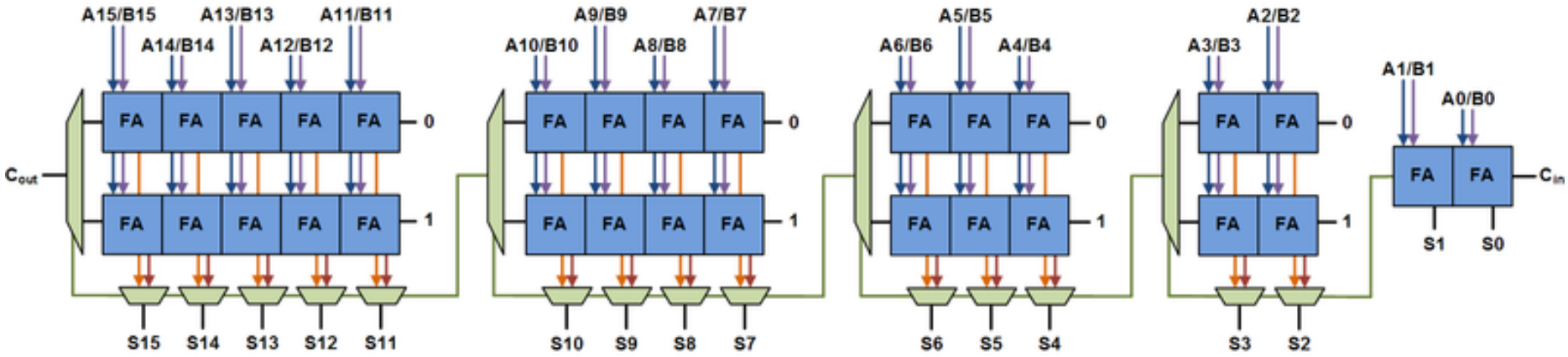
Verzögerung im Carry-Select Addierer

- Im Beispiel hat man 4 Gatter Verzögerung:
  - Die Berechnung des höherwertigen Teils ist gerade fertig, wenn die Entscheidung des niederwertigen Teils ankommt (rote Pfeile)



Optimierter Carry-Select Addierer

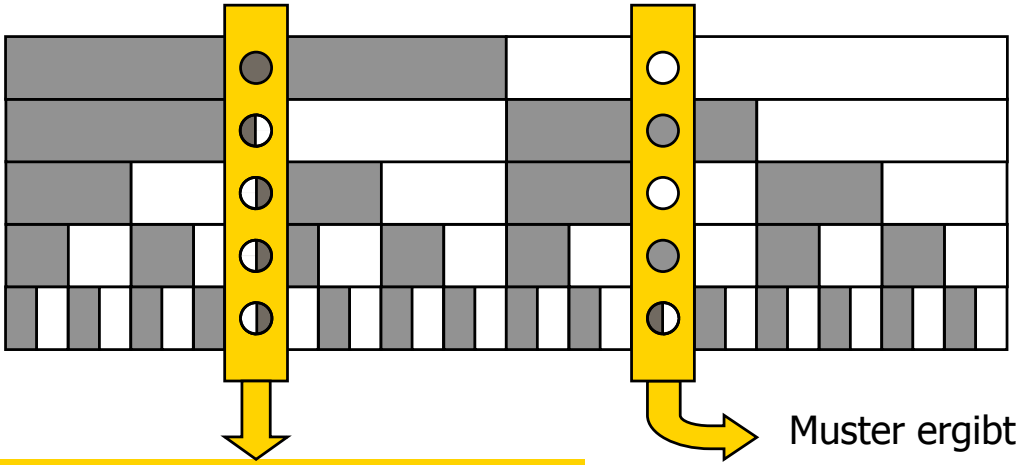
- Eine weitere Verbesserung erhält an durch mehr Gruppen mit ansteigender Breite:



Quelle: Wikipedia

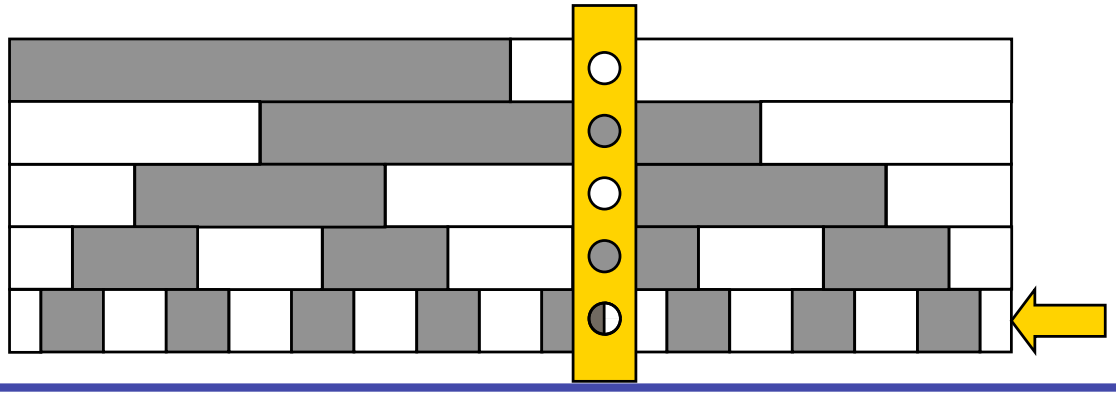
Gray Zähler: Wozu?

- Ein Gray Zähler ist *ein* möglicher Zähler mit Hammingdistanz H=1 (d.h. es ändert sich immer nur ein Bit)
- Betrachte z.B. einen linearen Maßstab zur Positionsmessung mit binärer Kodierung und Photosensor:



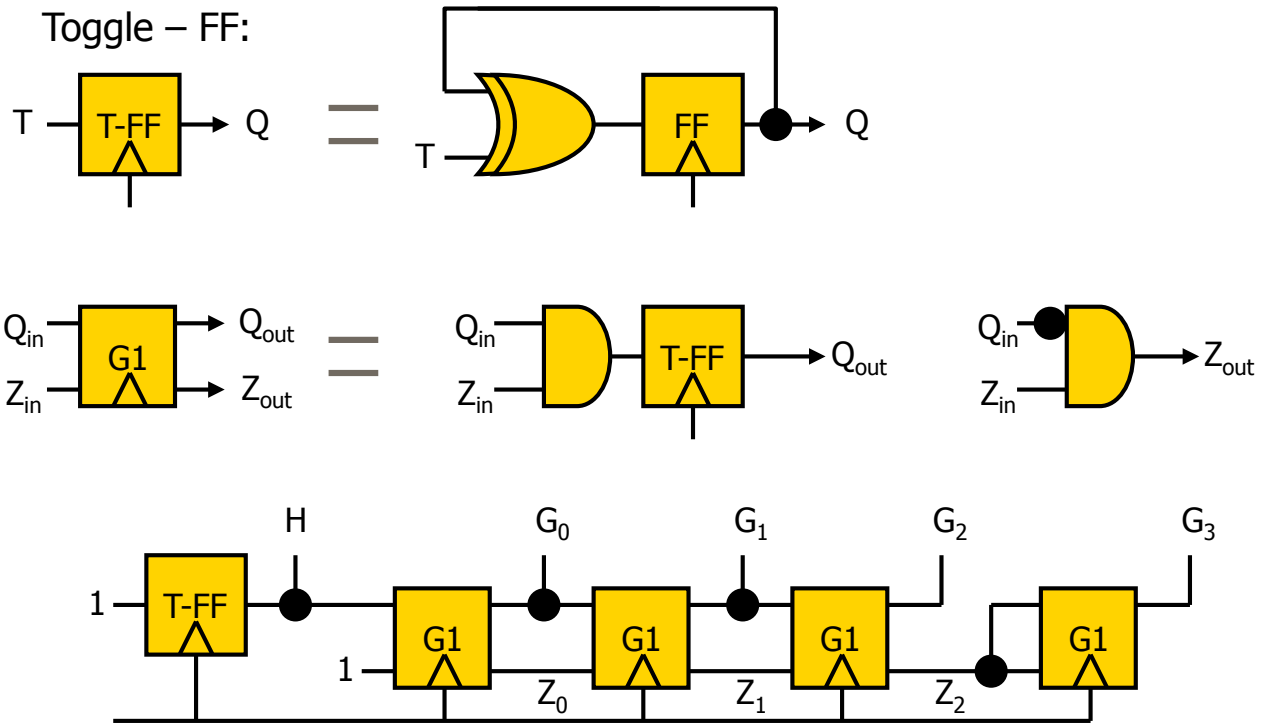
Problem: Wenn ein Sensor an einer der Kanten den falschen Wert meldet, ist die Position völlig falsch

- Lösung: An jeder Kante darf sich nur *ein* Bit ändern. z.B.: Gray Code: Ändere das niedrigste mögliche Bit



Gray Zähler: Implementierung

- Gray Zähler können 'direkt' implementiert oder aus Binärzahlen decodiert werden.
- Eine einfache Implementierung benutzt identische Blöcke pro Bit.
  - Ein Bit wird umgeschaltet (‚geflippt'), wenn die niederwertigeren Bits 1000... sind (rot).
    Die ‚000….' Information wird über eine Ripple-Kette erzeugt (Z-Signale: Z_i=1 heißt: H,Z_0…Z_{i-1}=0).
  - Trick: Für das niederwertigste Bit wird ein Hilfsbit benutzt (blau)
  - Das höchste Bit muß auch umschalten, wenn die niederwertigeren 0000... Sind (grün)



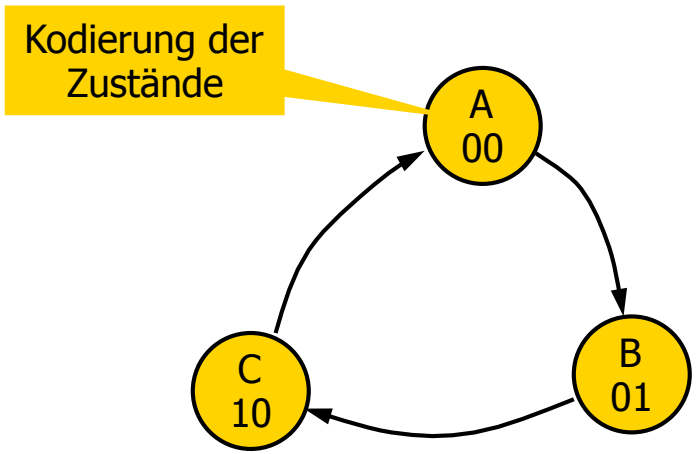
G3210	H	Z210
0000	1	000
0001	0	001
0011	1	000
0010	0	011
0110	1	000
0111	0	001
0101	1	000
0100	0	111
1100	1	000
1101	0	001
1111	1	000
1110	0	011
1010	1	000
1011	0	001
1001	1	000
1000	0	111
0000	1	000

- Der Ripple Pfad ist $Z_{in} \Rightarrow Z_{out}$

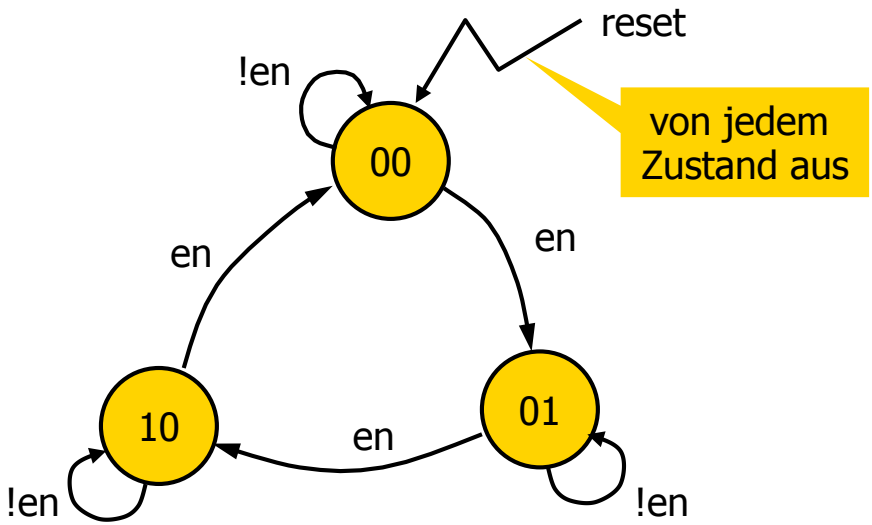
Zustandsautomaten

Zustandsautomaten

- Ein **Zustand** wird durch ein eindeutiges Bitmuster von Speicherelementen (Flipflops) definiert
- Der **Zustandsautomat (die Zustandsmaschine, 'state machine'**) befindet sich zu jedem Zeitpunkt (zumindest nach einem Reset) in einem erlaubten Zustand.
- Bei einem Taktsignal 'springt' er in einen neuen erlaubten Zustand (oder er bleibt im aktuellen Zustand)
- Ob ein neuer Zustand eingenommen wird (und welcher) hängt z.B. vom Zustand selbst und von externen Eingangsvariablen ab.
- Die Vorgänge werden in einem **Zustandsdiagramm** aufgezeichnet.
- Sehr wichtig zur **Ablaufsteuerung**
- Beispiel: Zähler, der binär periodisch von 0 bis 2 zählt:



ohne Enable



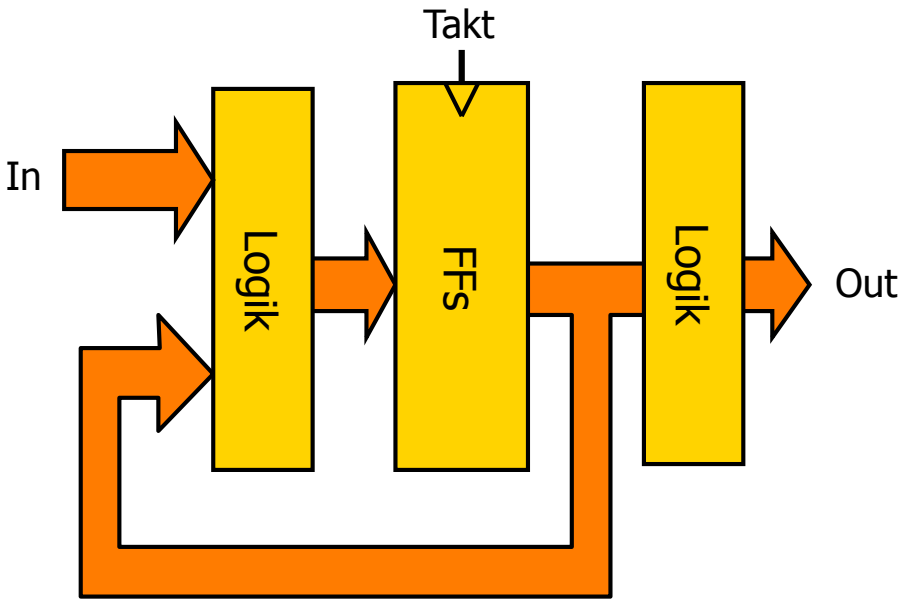
mit Enable und Reset

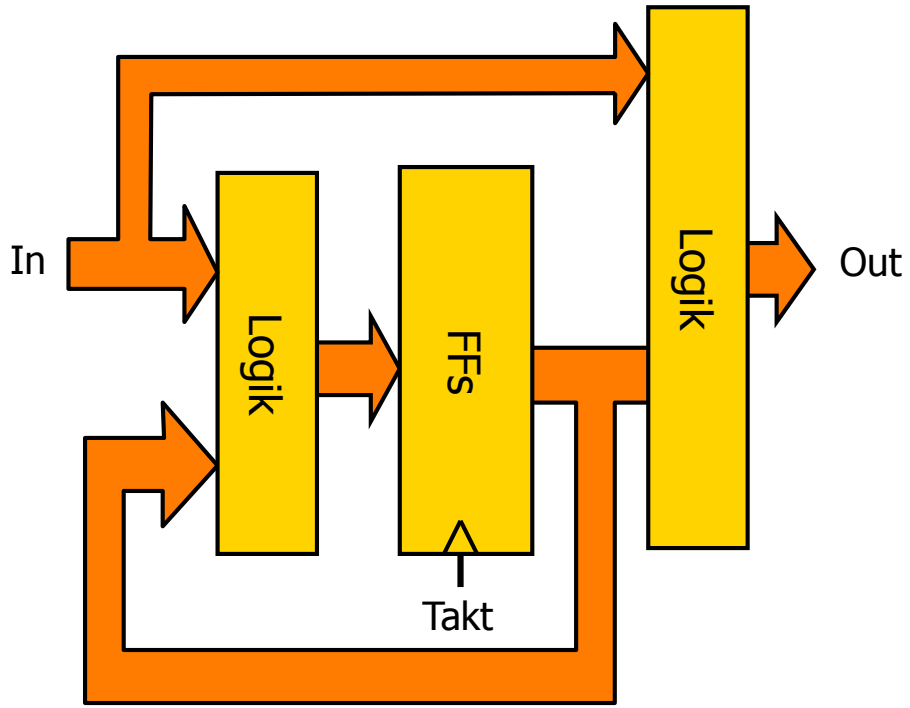
Klassifikation von Zustandsmaschinen

- 2 wichtige Typen:

- **Moore**-Maschine:
  - Ausgänge hängen **nur** vom Zustand ab (evtl. via kombinatorische Logik).
  - **Synchrone** Änderung der Ausgänge (bis auf Glitches in der Ausgangslogik)

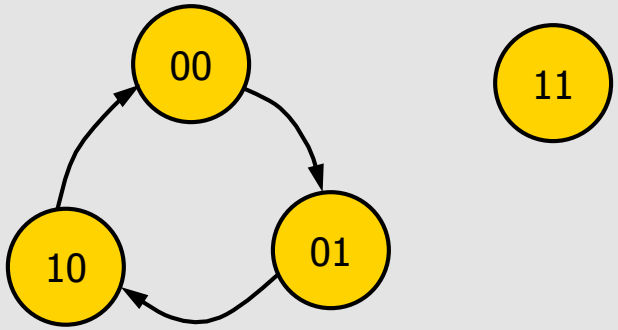
- **Mealy**-Maschine:
  - Ausgänge hängen **auch** von Eingängen ab.
  - **Asynchrone** Änderung der Ausgänge
  - Synchrone Variante durch weitere FFs direkt am Ausgang



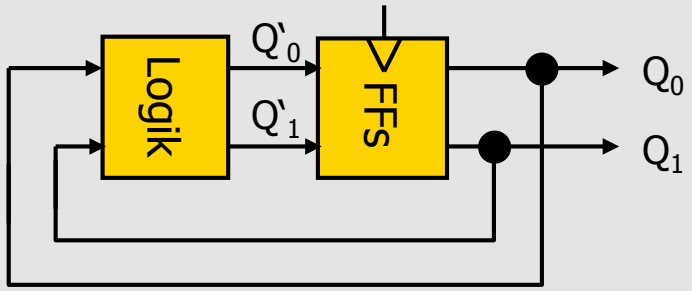


Implementierung: Beispiel Zähler 012012.. ohne Reset

- 1. Schritt: Zustandsdiagramm und Kodierung



- 2. Schritt: Architektur – hier Moore



- 3. Schritt: Wahrheitstabelle

	Q_1	Q_0	$Q`_1$	$Q`_0$
springen	0	0	0	1
	0	1	1	0
	1	0	0	0
Kommt nicht vor	1	1	X	X

- 4. Schritt: Gleichungen – hier mit KMAP

KMAP für $Q`_0$:

Q_1 \ Q_0	$\overline{Q_0}$
Q_1	0
$\overline{Q_1}$	1
$\overline{Q_1}$	0

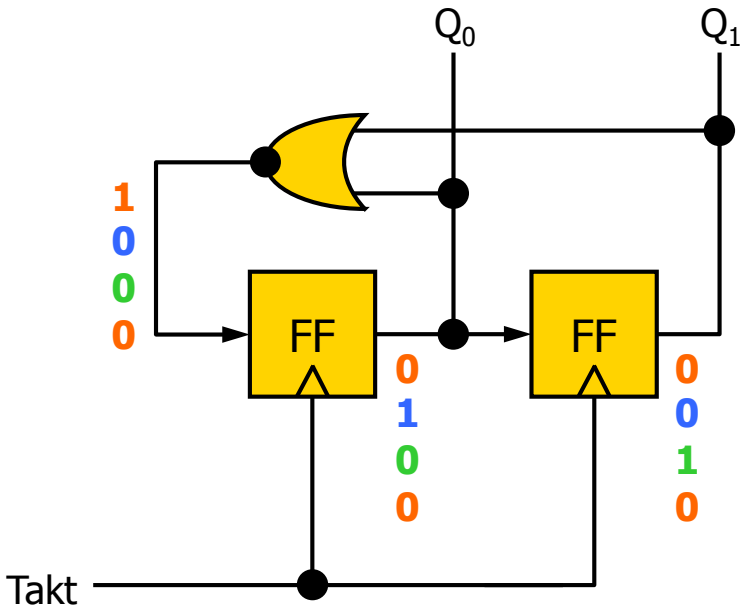
$$Q'_0 = \overline{Q_1 + Q_0} = \overline{Q_1} \cdot \overline{Q_0}$$

KMAP für $Q`_1$:

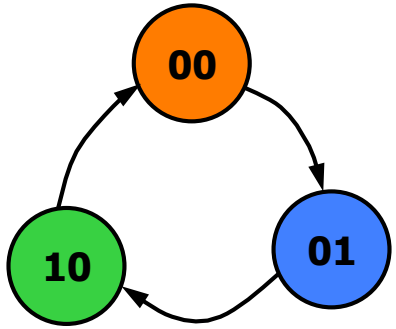
Q_1 \ Q_0	$\overline{Q_0}$
Q_1	0
$\overline{Q_1}$	0
$\overline{Q_1}$	1

$$Q'_1 = Q_0$$

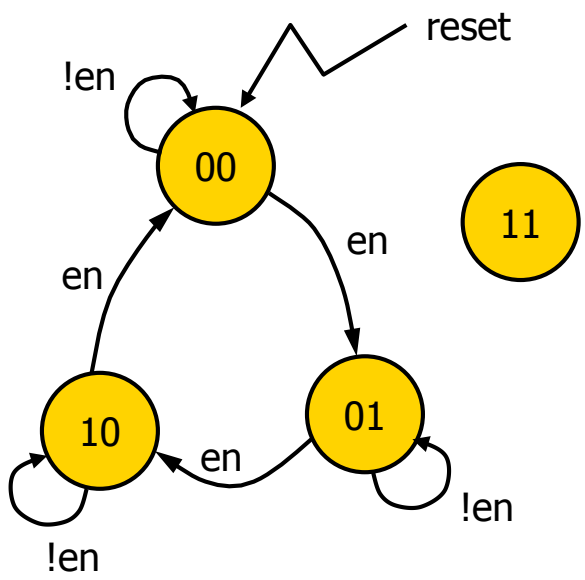
Implementierung

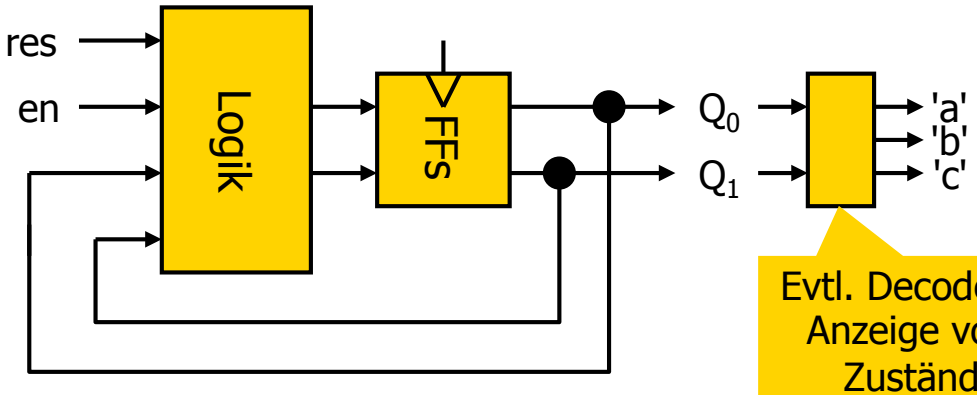


$$Q'_0 = \overline{Q_1 + Q_0} \qquad Q'_1 = Q_0$$



Beispiel: 3-Zustands-Zähler mit Reset und Enable





	res	en	Q_1	Q_0	$Q`_1$	$Q`_0$
warten	0	0	0	0	0	0
	0	0	0	1	0	1
	0	0	1	0	1	0
Kommt nicht vor	0	0	1	1	X	X
springen	0	1	0	0	0	1
	0	1	0	1	1	0
	0	1	1	0	0	0
Kommt nicht vor	0	1	1	1	X	X
reset	1	0	0	0	0	0
	1	0	0	1	0	0
	1	0	1	0	0	0
	1	0	1	1	0	0
	1	1	0	0	0	0
	1	1	0	1	0	0
	1	1	1	0	0	0
	1	1	1	1	0	0

Implementierung

res	en	Q_1	Q_0	Q'_1	Q'_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	X	X
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	X	X
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

KMAP für Q'_1:

res \ Q_0	Q_1=0	Q_1=1
en=0	0	1
en=0	0	0
en=1	0	0
en=1	0	X

res \ Q_0	Q_1=0	Q_1=1
en=0	0	X
en=0	0	0
en=1	0	0
en=1	0	1

$$Q'_1 = \overline{res} \cdot \overline{en} \cdot Q_1 + \overline{res} \cdot en \cdot Q_0$$

KMAP für Q'_0:

res \ Q_0	Q_1=0	Q_1=1
en=0	0	0
en=0	0	0
en=1	0	0
en=1	1	0

res \ Q_0	Q_1=0	Q_1=1
en=0	0	X
en=0	0	0
en=1	0	0
en=1	1	0

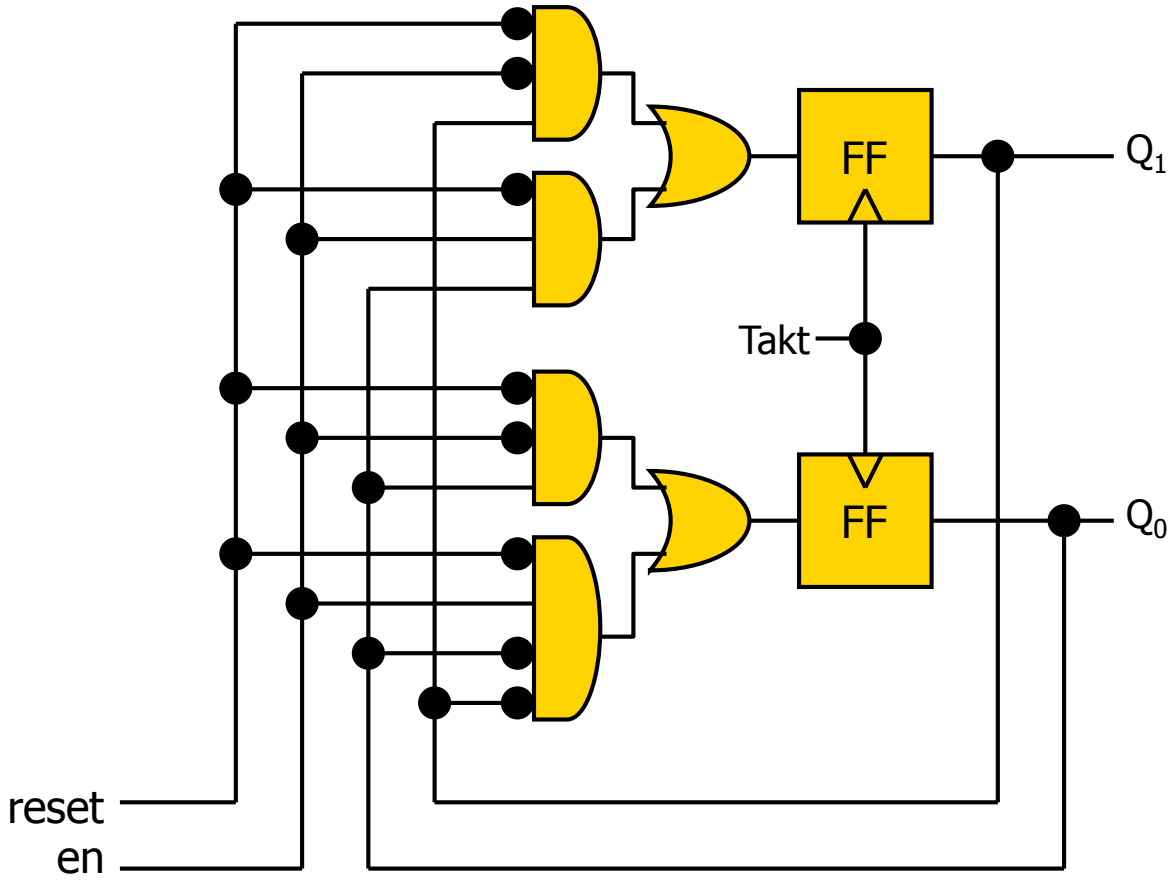
$$Q'_0 = \overline{res} \cdot \overline{en} \cdot Q_0 + \overline{res} \cdot en \cdot \overline{Q_1} \cdot \overline{Q_0}$$

NB: - Die 'X' im 'unmöglichen' Zustand 11 vereinfachen die Logik.
- Man muß dann mit einem Reset sicherstellen, daß die Zustandsmaschine nicht in diesem Zustand ‚fest hängt'!

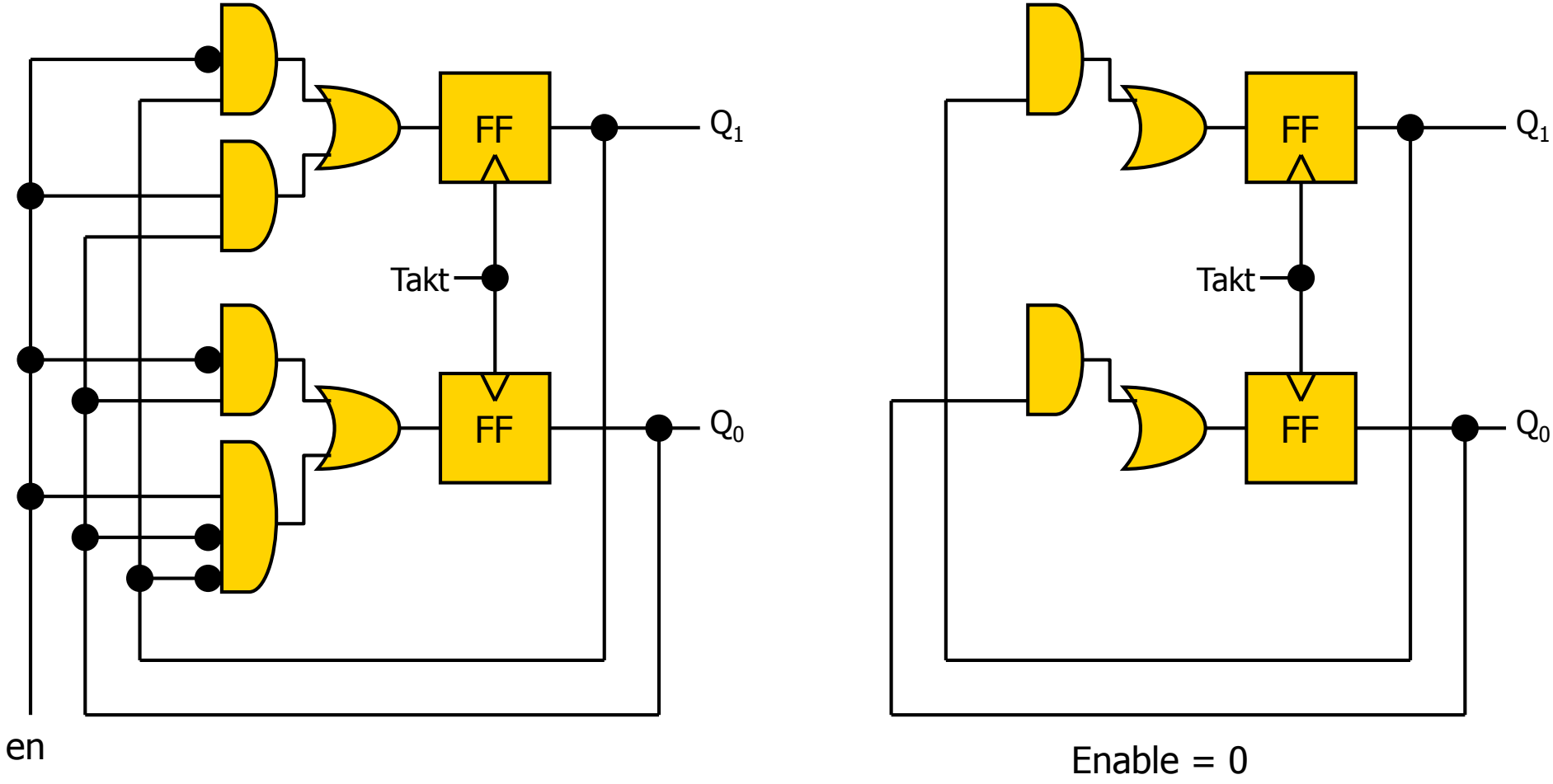
Implementierung

$$Q'_1 = \overline{res} \cdot \overline{en} \cdot Q_1 + \overline{res} \cdot en \cdot Q_0$$

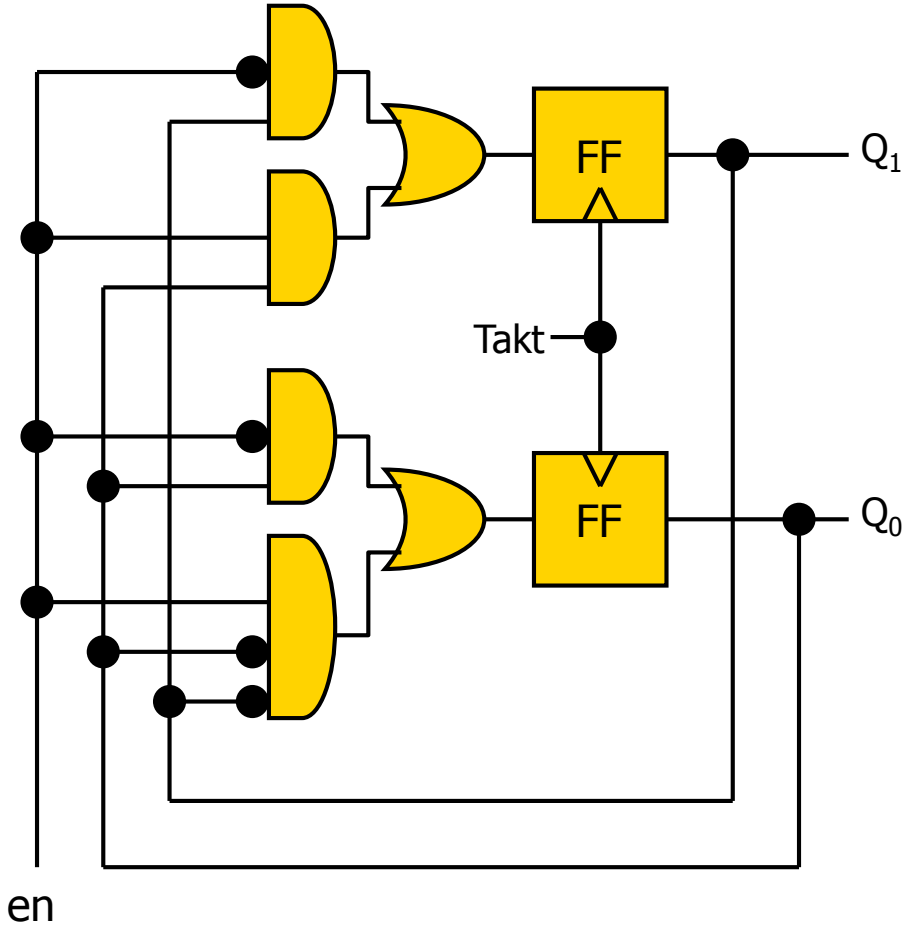
$$Q'_0 = \overline{res} \cdot \overline{en} \cdot Q_0 + \overline{res} \cdot en \cdot \overline{Q_1} \cdot \overline{Q_0}$$

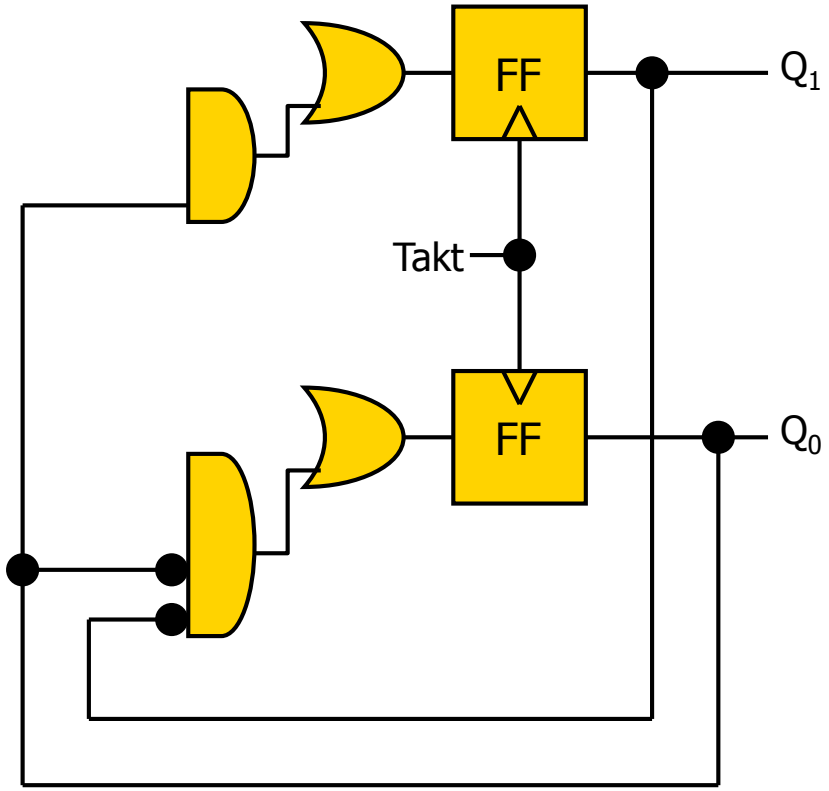


Test: Reset = 0



Test: Reset = 0

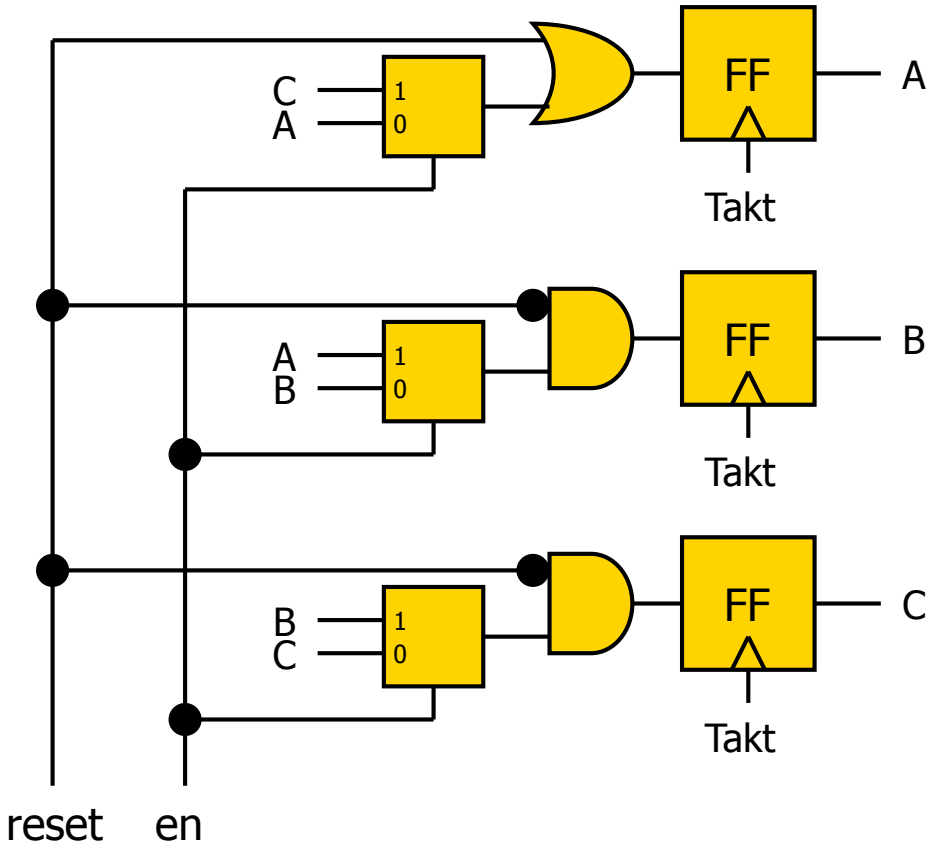




Enable = 1

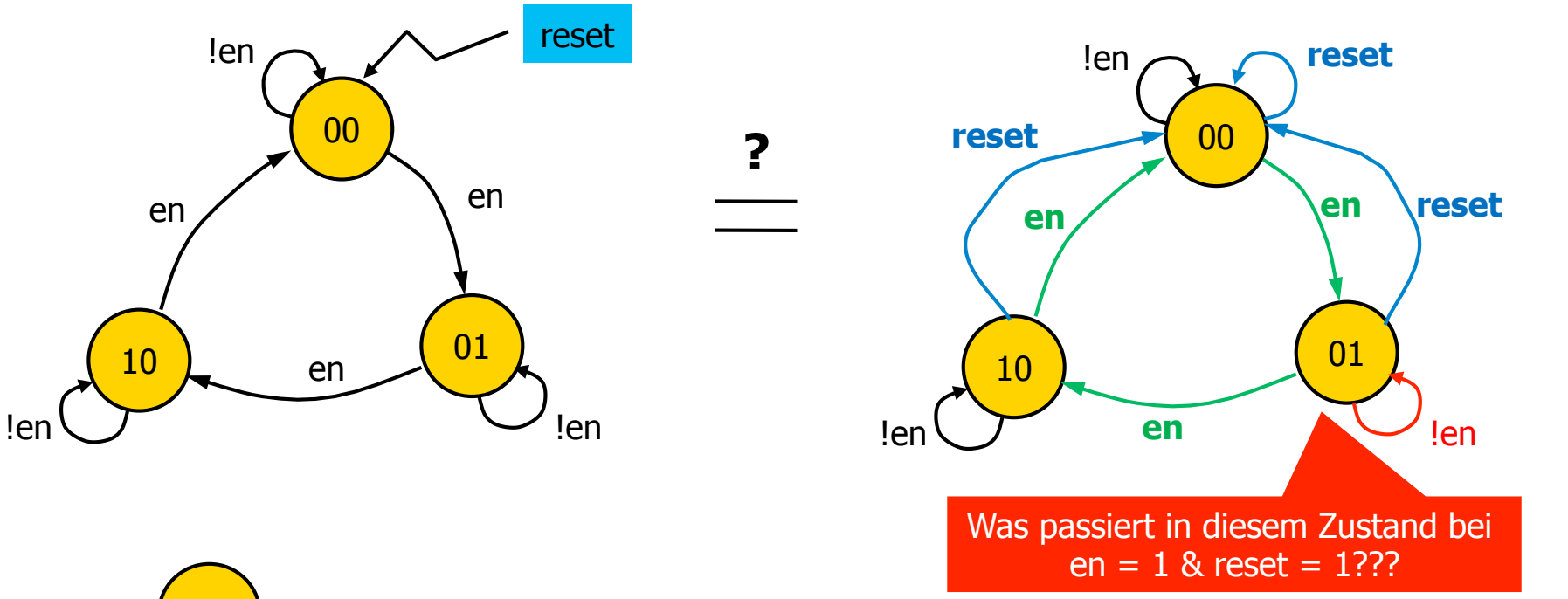
Andere Zustandscodierung

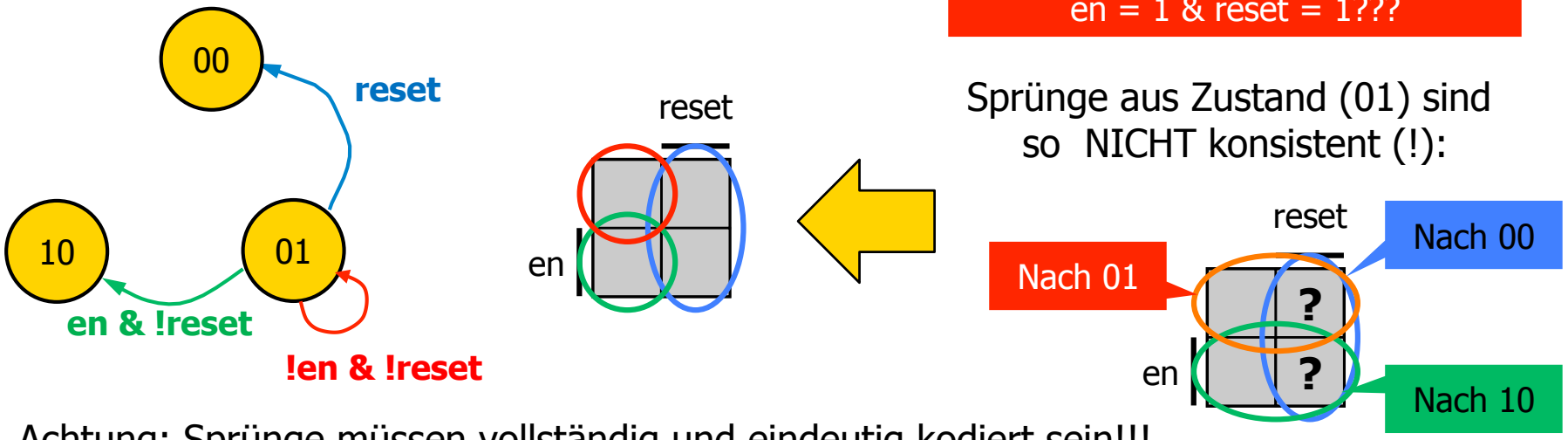
- Die 3 Zustände könnten auch mit 3 FFs als 001='a', 010='b', 100='c' codiert werden.
- Man nennt diese Art der Kodierung **'one hot encoding'**
- Ohne Herleitung sieht eine mögliche Implementierung dann so aus (= Schieberegister mit Enable):



- Merke: Durch andere Zustandskodierung *kann* sich die Ansteuerlogik vereinfachen (muss aber nicht)

Priorität der Sprünge



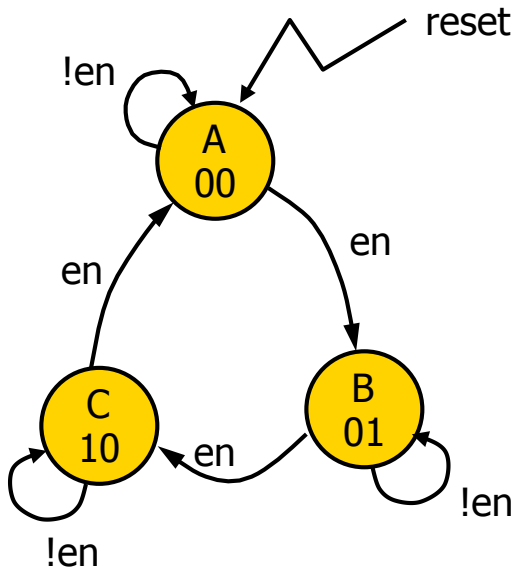


- Achtung: Sprünge müssen vollständig und eindeutig kodiert sein!!!

Beschreibung von Zustandsmaschinen mit ABEL

- Mit der Beschreibungssprache 'ABEL' können Zustandsmaschinen einfach beschrieben werden.
- Sie ist nur ein Beispiel für eine **Hardware Description Language (HDL)**.
- Beispiel 3-Zustands-Zähler:

```
module     counter3
title      'Three state machine';
U1 device 'p22v10';              „Art des Bauteils

clock      pin 1;
reset,en   pin 4,5;
q1,q0      pin 15,16;            "istype 'reg';

sreg = [q1,q0];                  "Zustandsregister
A = 0; B = 1; C = 2; XX = 3;     "Zustände, XX unbenutzt

state_diagram sreg;

State A:
  IF (en & !reset) THEN B ELSE A;
State B:
  IF (reset) THEN A ELSE IF (en) THEN C ELSE B;
State C:
  IF (!en & !reset) THEN C ELSE A;

end
```



- Dieses File kann mit dem Programm **ABEL** (oder Derivaten) übersetzt und simuliert werden (s. Demo).

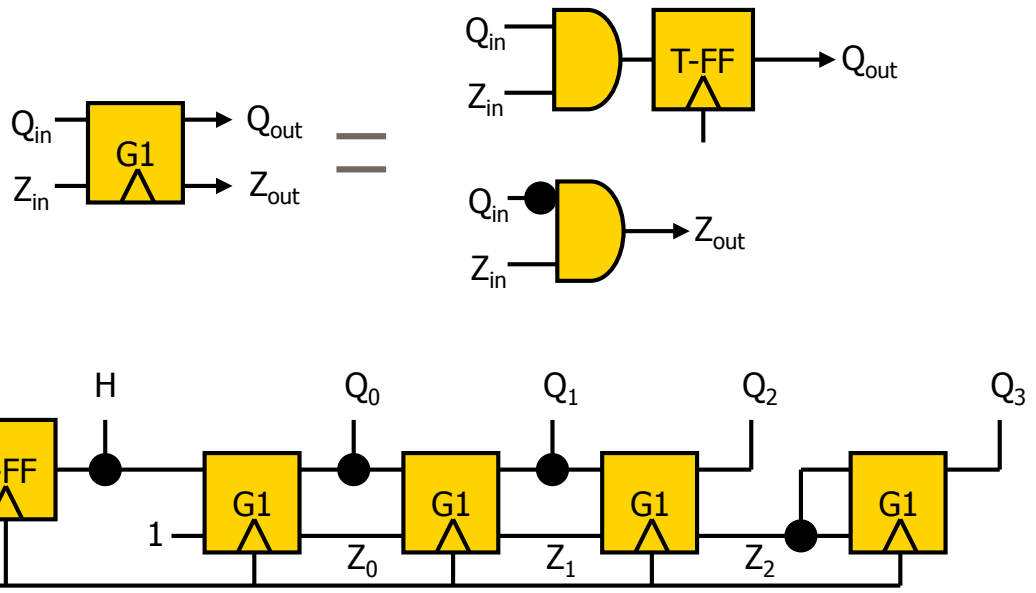
Testvektoren

- PALASM und ABEL können überprüfen, ob die gefundene Implementierung vorgegebene **Testvektoren** erfüllt:

```
test_vectors
([clock, reset, en] -> [q1,q0]);

[.c., 1, .x.] -> [0,0];    " reset
[.c., 0,  0 ] -> [0,0];    " warte
[.c., 0,  1 ] -> [0,1];    " State 0 -> State 1
[.c., 0,  1 ] -> [1,0];    " State 1 -> State 2
[.c., 0,  0 ] -> [1,0];    " warte
[.c., 0,  1 ] -> [0,0];    " zurück zu State 0

end
```

Zweites Beispiel: Gray Zähler

```
module    Gray_Counter
title     'Gray';
U1 device 'p22v10';

clock          pin 1;
reset          pin 4;
Q3,Q2,Q1,Q0,H  pin 14,15,16,17,18;

ON = 0; OFF = 1;

Z0 = 1  & !H;
Z1 = Z0 & !Q0;
Z2 = Z1 & !Q1;

equations
H  := !H # reset;


state_diagram Q0;
State ON:
  IF (reset) THEN OFF ELSE IF (H) THEN OFF ELSE ON;
State OFF:
  IF (reset) THEN OFF ELSE IF (H) THEN ON ELSE OFF;

state_diagram Q1;
State ON:
  IF (reset) THEN OFF ELSE IF (Q0 & Z0) THEN OFF ELSE ON;
State OFF:
  IF (reset) THEN OFF ELSE IF (Q0 & Z0) THEN ON ELSE OFF;

equations
Q2 := !reset & (Q2 $ (Q1 & Z1));  "$ is exclusive OR
```



Zweites Beispiel: Gray Zähler

```
state_diagram Q3;
State ON: IF (Z2 & !reset) THEN OFF ELSE ON;
State OFF:IF (Z2) THEN ON ELSE OFF;

test_vectors
([clock, reset] -> [Q3,Q2,Q1,Q0,H]);

[.c., 1] -> [0,0,0,0,1];    " reset
[.c., 0] -> [0,0,0,1,0];    " go...
[.c., 0] -> [0,0,1,1,1];
[.c., 0] -> [0,0,1,0,0];
[.c., 0] -> [0,1,1,0,1];
[.c., 0] -> [0,1,1,1,0];
[.c., 0] -> [0,1,0,1,1];
[.c., 0] -> [0,1,0,0,0];
[.c., 0] -> [1,1,0,0,1];
[.c., 0] -> [1,1,0,1,0];
[.c., 0] -> [1,1,1,1,1];
[.c., 0] -> [1,1,1,0,0];
[.c., 0] -> [1,0,1,0,1];
[.c., 0] -> [1,0,1,1,0];
[.c., 0] -> [1,0,0,1,1];
[.c., 0] -> [1,0,0,0,0];
[.c., 0] -> [0,0,0,0,1];    " back to start

end
```

Zusammenfassung

Zum Erstellen einer Zustandsmaschine sind folgende Schritte nötig:

1. Festlegen der Zustände
2. Festlegen der Übergänge (Zustandsdiagramm)

3. Festlegen des Maschinentyps (meist Moore)
4. Festlegen des ‚Encodings' (z.B. binär, one hot, encoded,…)

5. Aufstellen der Wahrheitstabelle
6. Reduktion der Gleichungen und Abbildung auf vorhandene Hardware

! Für einen wohl definierten Anfangszustand sorgen!
! Darauf achten, dass alle möglichen Übergänge spezifiziert sind !

Oszillation durch Rückkopplung

- Die Messung von sehr kleinen Verzögerungen ist schwierig (insbesondere, weil die Signale vom Chip noch durch andere Schaltungen müssen, die auch Verzögerungen beitragen, z.B. IO Pads)
- Daher mißt man die Verzögerung von Serienschaltungen mit N Elementen.
- Man baut einen **'Ringoszillator'** aus einer ungeraden Anzahl Inverter auf und mißt die Frequenz.
- Die Periode ist $\mathbf{T = 2 \times t_p \times N}$
- Meist ist noch ein NAND-Gatter zum kontrollierten Starten eingebaut