5. sequentielle Schaltungen

sequentielle Schaltungen: digitale Schaltung mit inneren Rückführungen
sie haben eine zeitsequentielle Arbeitsweise, wobei die einzelnen diskreten Zeitpunkte durch innere Zustände repräsentiert werden
die Ausgangsvariable ist eine Funktion der Eingangsvariablen x_i und der inneren Zustandsvariablen z_i:

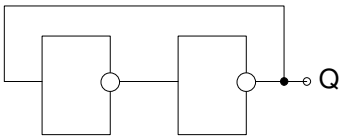
$y = f(x_0, x_1, ... x_{n-1}, z_0, z_1, ... z_{m-1})$

- Funktion f ist realisierbar durch Verknüpfung mit Grundgattern
- Zustandsvariable $\mathbf{z_i}$ werden durch binäre Speicherelemente "Flipflops FF" realisiert, die entweder 1 "gesetzt" oder auf 0 "rückgesetzt" werden

5.1. Flipflops

bekannte FF-Typen: RS-Flipflops, JK-FF, **D-FF**, T-FF

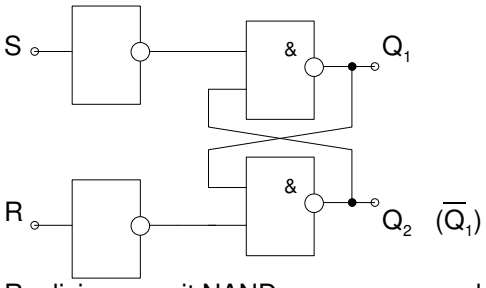
Grundprinzip: im einfachsten Fall erfolgt eine inneren Rückführung durch 2 Negatoren



$Q=\overline{\overline{Q}}$

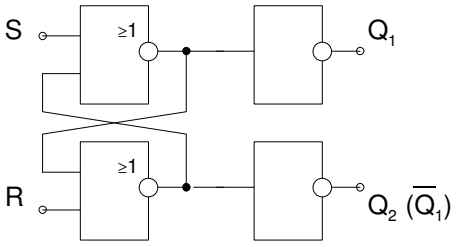
Zustand Q bleibt gespeichert, ist jedoch nicht beeinflußbar

RS-Flipflop: Realisierung mit Gattern (z.B. NAND, NOR)
2 Eingänge: **S** (Set, Setzen) und **R** (Reset, Rücksetzen)
2 Ausgänge: $\mathbf{Q_1}$ und $\mathbf{Q_2}$
i.A. existiert die Forderung nach einer komplementären Belegung, d.h. $Q_2 = \overline{Q}_1$



Realisierung mit NAND

bzw.



NOR-Gattern

Definition: Setzen: S = 1 und R = 0 → Q_1 = **1** Q_2 = 0
Rücksetzen: R = 1 und S = 0 → Q_1 = **0** Q_2 = 1

ST RS-FF:

S	R	Q_1	Q_1^*	Q_2^*	
0	0	0	0	1	Speichern
0	0	1	1	0	
0	**1**	0	**0**	1	Rücksetzen
0	**1**	1	**0**	1	
1	0	0	**1**	0	Setzen
1	0	1	**1**	0	
1	1	0	1	1	unbrauchbar
1	1	1	1	1	$Q_1 = Q_2$

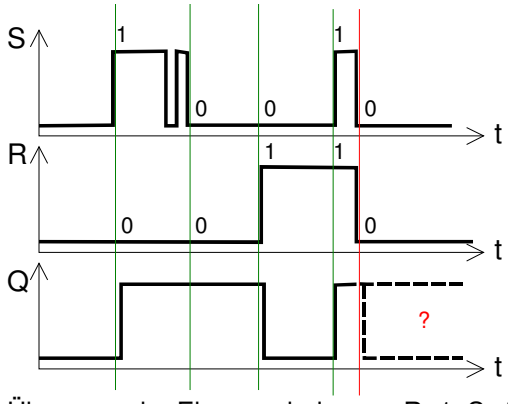
SF RS-FF: charakteristische Gleichung eines RS-FF:

$Q_1^* = S \vee (\overline{R}\, Q_1)$

Q: vor der Zustandsänderung
Q_1^*: nach der Zustandsänderung

HDL: `Q <= S OR (NOT R AND Q);`

Zeitdiagramm:

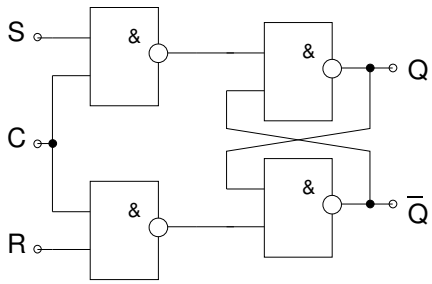


Problem: Übergang der Eingangsbelegung R=1, S=1 (Zustand $Q_1=Q_2=1$)
zu R=0, S=0 (Zustand speichern, wird $Q_1=1$, $Q_2=0$ oder $Q_1=0$, $Q_2=1$ gespeichert?)

Lösung: Unterdrückung einer kurzzeitig auftretenden Belegung R=1, S=1 durch **Taktung** bzw. Vermeidung der Eingangsbelegung R=1, S=1 durch **Verknüpfung** der Eingangssignale untereinander (z.B. R = $\overline{S}$, **D-FF**) oder mit den Ausgangssignalen (z.B. Q_1, Q_2, **JK-FF**)

<u>getaktetes RS-Flipflop</u>

Funktion: C = 0: Tor gesperrt (C ∧ S = **0**; C ∧ R = **0** Zustand Speichern)
C = 1: Tor geöffnet (C ∧ S = **S**; C ∧ R = **R** neuen Zustand einnehmen)



ST RS-FF:

C	S	R	Q	Q^*	$\overline{Q}^*$	
1	0	0	0	0	1	Speichern
1	0	0	1	1	0	
1	0	**1**	0	**0**	1	Rücksetzen
1	0	**1**	1	**0**	1	
1	**1**	0	0	**1**	0	Setzen
1	**1**	0	1	**1**	0	
1	1	1	0	1	1	vermeiden
1	1	1	1	1	1	
0	×	×	**0**	**0**	**1**	**keine Änderung**
0	×	×	**1**	**1**	**0**	

SF D-FF: vollständige charakteristische Gleichung des getakteten RS-FF

$Q^* = (S \vee \overline{R}\, Q)\, C \vee Q\ \overline{C}$

HDL:
```
Q <= ((S OR NOT R AND Q) AND C) OR (Q AND  NOT C);
```

oder besser:
```
process (S, R, C)
begin
if C='1' then
   if S='1' AND R='0' then Q <= '1';
   if R='1' AND S='0' then Q <= '0';
end if;
end process;
```

*Der **process** beschreibt, was bei Änderung eines der Eingangssignale (S, R, C) mit dem Ausgang Q geschieht. Erfolgt keine Änderung der Eingänge, bleibt Q unverändert – wird also gespeichert.*

vereinfachte Darstellung:

oft wird die letzte Zeile der ST nicht dargestellt, da getaktete FF grundsätzlich bei nicht aktiven Taktsignal (hier: C=0) keine Änderung am Ausgang Q zeigen.

vereinfachte charakteristische Gleichung:

$Q^* = S \vee (\overline{R}\, Q)$ Bedingung: C=1

die vereinfachte charakteristische Gleichung des getakteten RS-FF gilt nur während eines angelegten Taktes C=1,

für C=0 erfolgt keine Änderung des Ausganges (Speichern $Q^* = Q$)

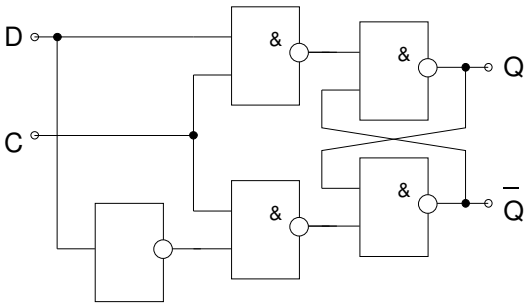
Schlussfolgerung:

Der Takt kann im Entwurf für eine vereinfachte Darstellung zunächst unberücksichtigt bleiben, da "keine Änderung" zu erwarten ist.
In der konkrete Schaltung muß natürlich dieser Takteingang beschaltet werden.

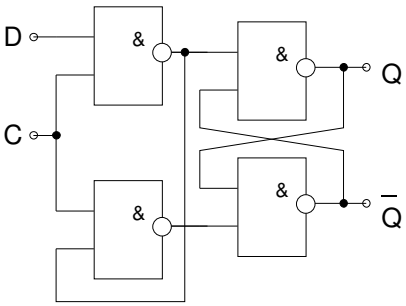
<u>D-Flipflop</u>

Verzögerungsspeicher (**D**elay = Verzögerung) oft auch als Latch ("Riegel") oder D-Latch bezeichnet

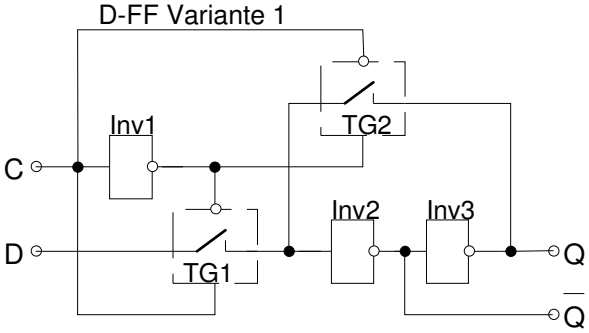
LP:



D-FF Variante 1

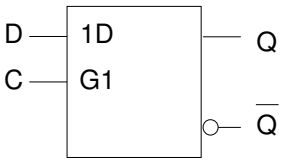


D-FF Variante 2



D-FF Variante 3 (mit Transmissionsschaltern)

Symbol:



Funktion: abgeleitet aus dem RS-FF mit S = D; R = $\overline{D}$, damit wird $\overline{S}$ = $\overline{R}$ = 0 vermieden, d.h. die Ausgänge (Q, $\overline{Q}$) sind immer komplementär zueinander

C = 0: Tor (NAND-Gatter) gesperrt (C ∧ D = **0** Zustand Speichern)

C = 1: Tor (NAND-Gatter) geöffnet (C ∧ D = **D** Eingangsbelegung übernehmen)

ST D-FF:

C	D	Q	Q^*	$\overline{Q}^*$	
1	0	0	0	1	Rücksetzen
1	0	1	0	1	
1	1	0	1	0	Setzen
1	1	1	1	0	
0	×	0	0	1	keine Änderung (Speichern)
0	×	1	1	0	

SF D-FF: vollständige charakteristische Gleichung des D-FF

$Q^* = D \wedge C \vee Q \wedge \overline{C}$

vereinfachte Darstellung:

oft wird die letzte Zeile der ST nicht dargestellt, da getaktete FF grundsätzlich bei nicht aktiven Taktsignal (hier: C=0) keine Änderung zeigen.

vereinfachte charakteristische Gleichung:

$Q^* = D$ Bedingung: C=1

die vereinfachte charakteristische Gleichung des D-FF gilt nur während eines angelegten Taktes C=1, für C=0 erfolgt keine Änderung des Ausganges

(Speichern $Q^* = Q$)

Schlußfolgerung:

Der Takt kann im Entwurf für eine vereinfachte Darstellung zunächst unberücksichtigt bleiben, da "keine Änderung" zu erwarten ist.

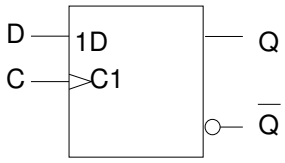
In der konkreten Schaltung muß natürlich dieser Takteingang beschaltet werden.

verkürzte Darstellung des Zustands Speichern in der ST:

C	D	Q	Q^*	$\overline{Q}^*$	
0	×	Q	Q	$\overline{Q}$	Speichern
1	D	×	D	$\overline{D}$	Übernehmen von D

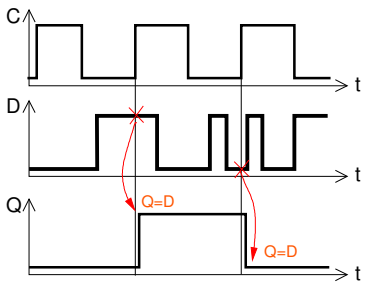
taktflankengetriggertes D-FF

die charakteristische Gleichung entspricht dem D-FF, die Übernahme der Daten erfolgt jedoch nicht während C=1, sondern während der 0→1Taktflanke von C:



HDL:

```
process (C)
begin
if rising_edge(C) then
     Q <= D;
end if;
end process;
```



Zeitdiagramm des taktflankengetriggerten D-FF

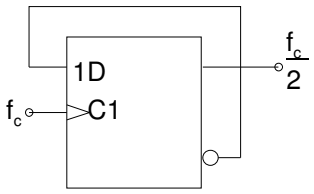
*Der **process** beschreibt, was bei einer positiven Taktflane (rising_edge) des Eingangssignals (C) mit dem Ausgang Q geschieht – er übernimmt D. Ansonsten bleibt Q unverändert – wird also gespeichert.*
Optional kann durch ein Reset (R) ein Anfangszustand z.B. 0 hergestellt werden.
Entsprechende Flipflops haben einen zusätzlichen R-Eingang:

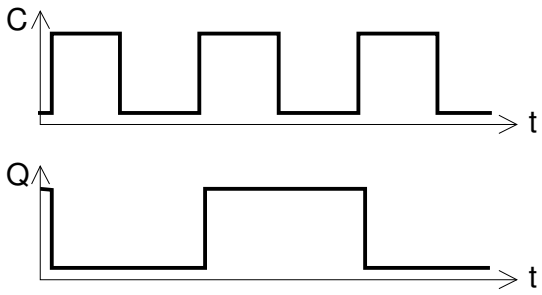
```
process (C,R)
begin
if R='1' then
    Q <= '0';
elsif rising_edge(C) then
    Q <= D;
end if;
end process;
```

Anwendungsbeispiel:

Binärteiler

Rückführung des $\overline{Q}$-Ausgangs auf den D-Eingang





Zeitdiagramm Frequenzteiler

5.2. Zustandsautomaten

allgemeiner Automat: $A = (X,Y,Z, f, g, Z_0)$
(X: Eingangsalphabet, Y: Ausgangsalphabet, Z: innerer Zustand,
f: Zustandsüberführungsfunktion, g: Ausgangsfunktion, Z_0: Anfangszustand)

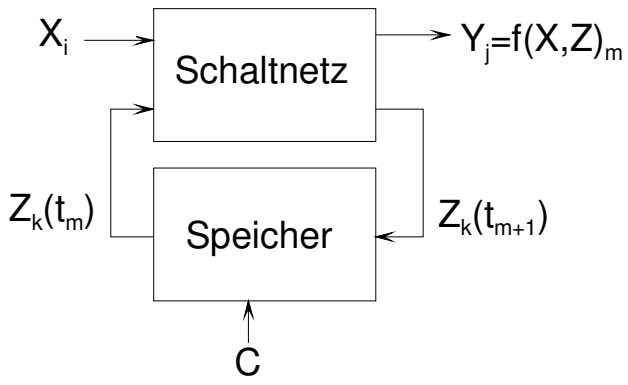
Digitale Automaten bestehen aus kombinatorischen und sequentiellen Schaltungen, kurz: Schaltnetzen und Speichern. Sie werden auch als Schaltwerke bezeichnet.
Da bei ihnen nur eine endliche Anzahl von Bauelementen existieren und eine Zufalls-Logikbelegung durch eindeutige Schaltungsauslegung nicht auftritt, handelt es sich um deterministische Automaten. Beispiele sind Mealy/Moore-Automaten, Zähler, Schieberegister.

allgemeiner Automat

$X_i = (x_0, x_1, ... x_{l-1})$	$0 \le i \le 2^l-1$	i Eingangsvektoren l-dimensional
$Y_j = (y_0, y_1, ... y_{v-1})$	$0 \le j \le 2^v-1$	j Ausgangsvektoren v-dimensional
$Z_k = (z_0, z_1, ... z_{w-1})$	$0 \le k \le 2^w-1$	k Zustandsvektoren w-dimensional

Der Automat wird zur Vereinfachung nur zu diskreten Zeitpunkten t_m betrachtet. (Synchroner Automat)
Die Ausgangsvektoren sind von den Eingangsvektoren X und den inneren Zuständen Z zu den diskreten Zeitpunkten m abhängig:

$Y_j = f (X, Z)_m$



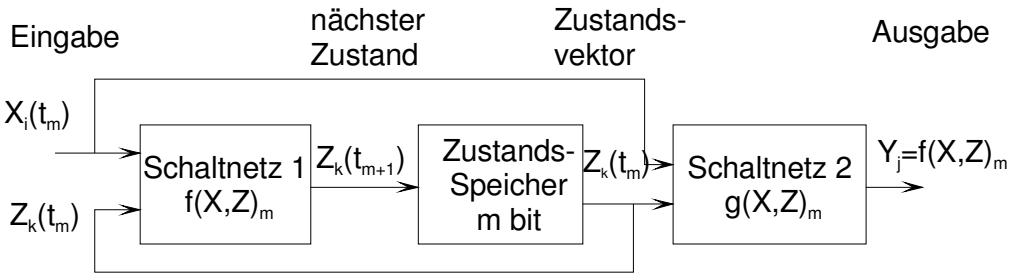
Systematisierung des Zusammenhanges ergibt die Zustandsgleichungen:

$Y_j(t_m) = g\,[\,X_i(t_m)\,,\,Z_k(t_m)\,]$ Ausgangs/Ergebnisgunktion

$Z_k(t_{m+1}) = f\,[\,X_i(t_m)\,,\,Z_k(t_m)\,]$ Überführungs/Übergangsfunktion

$Z_k\,(t_m) := Z_k\,(t_{m+1})$ Zeitfunktion (synchroner Automat)

daraus ist ein realisierbarer Zustandsautomat ableitbar (Mealy-Automat):



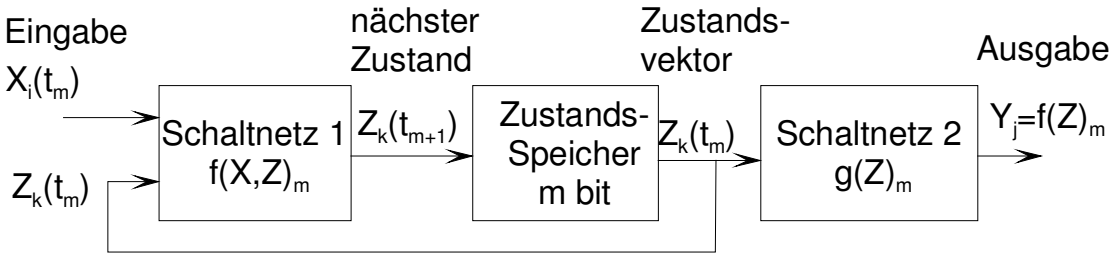
Mealy-Automat

ein Sonderfall ergibt sich, wenn der Ausgangsvektor nicht unmittelbar vom Eingangsvektor abhängig ist. (Moore-Automat)

$Y_j(t_m) = g\,[\,Z_k(t_m)\,]$ Ausgangs/Ergebnisfunktion

$Z_k(t_{m+1}) = f\,[\,X_i(t_m)\,,\,Z_k(t_m)\,]$ Überführungs/Übergangsfunktion

$Z_k\,(t_m) := Z_k\,(t_{m+1})$ Zeitfunktion (synchroner Automat)



Moore-Automat

Beschreibungsmöglichkeiten von Schaltwerken

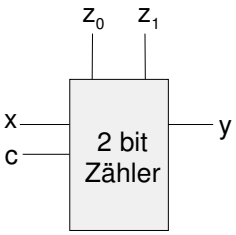
- Zeitlicher Verlauf der Eingangs- und Ausgangssignale (Schaltfolgediagramm, Zeitdiagramm, Timing-Diagramm)
- Tabellen (Automatentabelle, **Zustandsübergangstabelle**)
- **Zustandsgraph** oder Signalflußgraph
- **Zustandsgleichungen** (Schaltfunktion)
- HDL, Hardwarebeschreibungssprachen (VHDL)

Entwurf von Schaltwerken

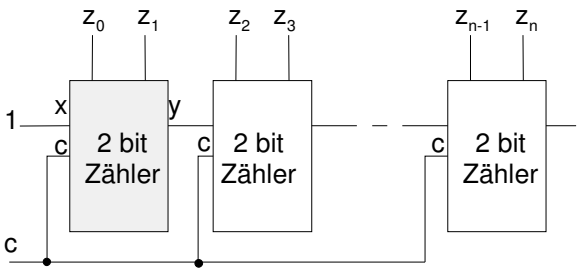
1. Beschreibung der Funktion, z.B. mittels Zustandsgraph
2. Ermittlung der Anzahl der notwendigen binären Zustandsspeicher w aus der Anzahl m der inneren Zustände $w \geq ld\,(m)$ (aufgerundet auf die nächsthöhere ganze Zahl)
3. Festlegung der Zustandskodierung (Dual, Gray, 1-aus-m, ...)
4. Festlegung des Flipfloptyps (D-FF, T-FF, ...)
5. Aufstellen der Zustandsübergangstabelle
6. Generierung der Flipflop-Eingangsbelegung (Übergangsfunktion) und der Ausgangsfunktion
7. Synthese des Schaltplanes

Beispiel: Es soll ein kaskadierbarer 2-Bit-Dualzähler mit Zählerfreigabe (x) und Übertrag (y) entworfen werden.

Zustandsfolge: $Z_m = (z_1, z_0)_m = (00, 01, 10, 11, 00 ....)$
Zähler zählt nur dann, wenn x = 1 ist (Freigabe) und der Takt c anliegt.
Der Übertrag wird zum Zustand 11 durch y =1 ausgegeben. (synchroner Übertrag)



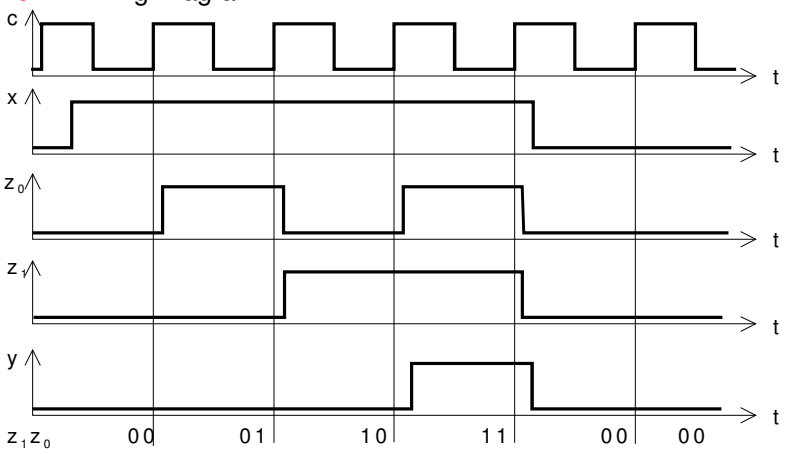
spätere Anwendung des Beispiel-Zählers:



- Eine hintereinanderschaltung (Kaskadierung) mehrerer 2-bit-Zähler ergibt einen n-bit-Zähler.
- Alle Zähler nutzen den gleichen Takt. → synchron (d.h. "gleichgetaktet")
- Die jeweils nachfolgende Stufe zählt genau nur dann um eins weiter, wenn die vorhergehende Stufe einen "Übertrag" y = 1 generiert.
- Dazu hat jeder Zähler einen Freigabeeingang x
- Im Beispiel wird einer dieser (identischen) 2-bit-Zähler entworfen.

Entwurfsschritte:

zu 1. Timing-Diagramm



zu 2. Zähler mit 4 binären Zuständen Z_0, Z_1, Z_2 Z_3 erfordert 2 Zustandsbits (2 FF)

zu 3. Kodierung : Dualcode: Z_0= 00, Z_1 = 01, Z_2 = 10, Z_3 = 11

Zustandsübergangstabelle

x Eingangsbelegung zur Zeit **m**

z_1, z_0 Zustand zur Zeit **m**

z_1^*, z_0^* Folgezustand zur Zeit **m+1**

y Ausgangsbelegung zur Zeit **m**

x	z_1	z_0	z_1^*	z_0^*	y
0	0	0	0	0	0
1	0	0	0	1	0
0	0	1	0	1	0
1	0	1	1	0	0
0	1	0	1	0	0
1	1	0	1	1	0
0	1	1	1	1	0
1	1	1	0	0	1

"Übertrag

Zustandsgleichungen

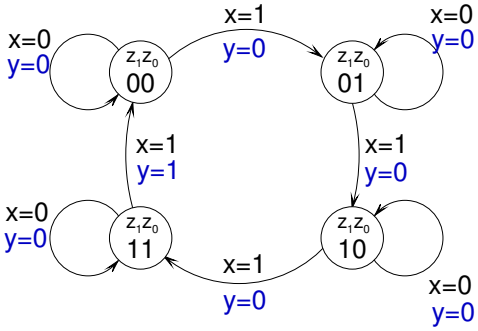
$z_0^* = x\ \overline{z_0} \vee \overline{x}\ z_0$

$z_1^* = z_1\ \overline{z_0} \vee \overline{x}\ z_1\ z_0 \vee x\ \overline{z_1}\ z_0$

Zustandsgraph

Knoten: Zustände, Zustandskodierung

Kanten: Eingangskombinationen, die zum Zustandsübergang führen



zu 4. D-Flipflops (Synchronzähler)

Zustandsübergangstabelle (Hilfsmittel: D-FF Synthesetabelle)

C	D	Q	Q^*	$\overline{Q}^*$	
0	0	0	0	1	SP
0	0	1	1	0	SP
0	1	0	0	1	SP
0	1	1	1	0	SP
1	0	0	0	1	RES
1	0	1	0	1	RES
1	1	0	1	0	SET
1	1	1	1	0	SET

abgeleitete **Synthesetabelle** D-FF

$Q \rightarrow Q^*$	D
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1

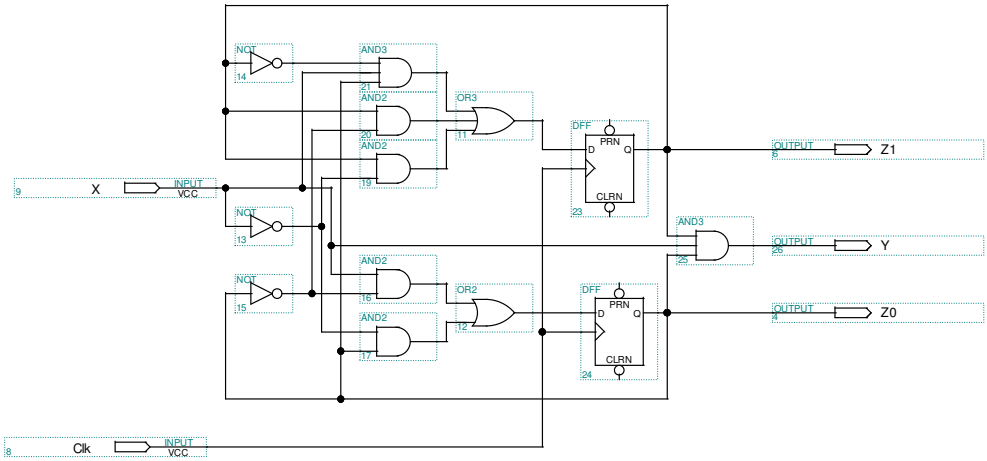
zu 5. Zustandsübergangstabelle mit D-Generierung

x	z_1	z_0	z_1^*	z_0^*	y	D_1	D_0
0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1
1	0	1	1	0	0	1	0
0	1	0	1	0	0	1	0
1	1	0	1	1	0	1	1
0	1	1	1	1	0	1	1
1	1	1	0	0	1	0	0

zu 6. Generierung von D_1, D_0 und von y

$D_0 = x\ \overline{z_0} \vee \overline{x}\ z_0 = x \oplus z_0$; $D_1 = \overline{x}\ z_1 \vee z_1\ \overline{z_0} \vee x\ \overline{z_1}\ z_0$ $y = x\ z_1\ z_0$

zu 7. Logikplan:



$D_0 = x\ \overline{z_0} \vee \overline{x}\ z_0 = x \oplus z_0$;

$D_1 = \overline{x}\ z_1 \vee z_1\ \overline{z_0} \vee x\ \overline{z_1}\ z_0$

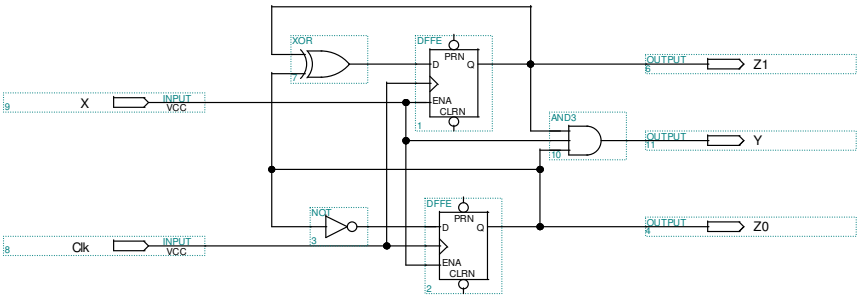
XOR-Umformung:

$D_1 = \overline{x}\ z_1 \vee x\ z_1\ \overline{z_0} \vee x\ \overline{z_1}\ z_0$ (unschädliche Erweiterung mit x)

$D_1 = \overline{x}\ z_1 \vee x\ (z_0 \oplus z_1)$ (x –gesteuerte Multiplexer im DFFE und zusätzliches XOR-Gatter)

$y = x\ z_1\ z_0$

$D_0 = \overline{x}\ z_0 \vee x\ \overline{z_0}$;



Verallgemeinerung (Herleitung später beim n-Bit-Zähler): $D_i = \overline{x}\ z_i \vee x\ (z_{i-1} \wedge z_{i-2} \ldots \wedge z_0 \wedge 1 \oplus z_i)$

HDL (1-Bit Zählstufe, vollständig)

```
LIBRARY IEEE; USE IEEE.STD_LOGIC_1164.ALL;

ENTITY bit_count is
port (
 signal X, Clk: in STD_LOGIC;
 signal Z: out STD_LOGIC_VECTOR (1 downto 0);
 signal Y: out STD_LOGIC );
end bit_count;

ARCHITECTURE arch_bit_count of bit_count is
signal Q: STD_LOGIC_VECTOR (1 downto 0):="00";
begin
Y <= X AND Q(1) AND Q(0);
Z <= Q;
process (Clk)
begin
  if rising_edge(Clk) then
    if X = '1' then
      Q(1) <= Q(1) XOR Q(0);
      Q(0) <= NOT Q(0);
    end if;
 end if;
end process;
end arch_bit_count;
```

Anstelle Q <= D wie beim FF notieren wir die logische Funktion, wenn X='1' gesetzt ist.
Bei X='0' soll sich nichts ändern.
Q(1) und Q(0) bleiben erhalten, ohne das explizit ein else Zweig für if X='1' angegeben wird.

Verallgemeinert:
Alle Zuweisungen in einem solchen Prozess werden gespeichert - sie beschreiben jeweils ein Flipflop.
Da der Ausgang Y nicht gespeichert werden soll, erfolgt die Zuweisung <u>außerhalb</u> des Prozesses.
Diese Zuweisung kann über oder unter dem Prozess notiert werden, da die AND Funktion immer berechnet wird – nebenläufig (parallel) zum Prozess.
*Z wird ebenso aus Q erzeugt, das ist nötig, da Z als Ausgang **out** nicht gelesen werden kann.*

Problem:
Der Anfangszustand der Flipflops Q wird zwar mit "00" festgelegt, das entspricht aber nicht der Schaltung bzw. der Hardware. Dazu sollte später besser der CLRN oder PRN Eingang (s. Bild) bzw. ein RES Signal benutzt werde.

Moore-Version des Beispielzählers

X	z_1	z_0	z_1^*	z_0^*	y
0	0	0	0	0	0
1	0	0	0	1	0
0	0	1	0	1	0
1	0	1	1	0	0
0	1	0	1	0	0
1	1	0	1	1	0
0	1	1	1	1	1
1	1	1	0	0	1

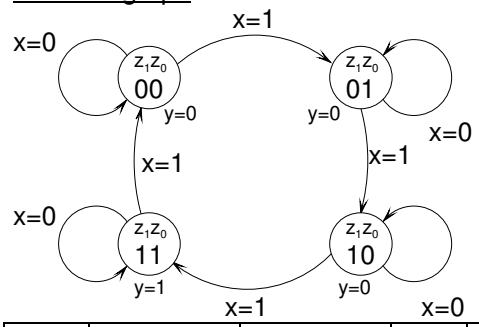
Übertrag

Zustandsgleichungen

$z_0^* = x\ \overline{z_0} \vee \overline{x}\ z_0$

$z_1^* = z_1\ \overline{z_0} \vee \overline{x}\ z_1\ z_0 \vee x\ \overline{z_1}\ z_0$

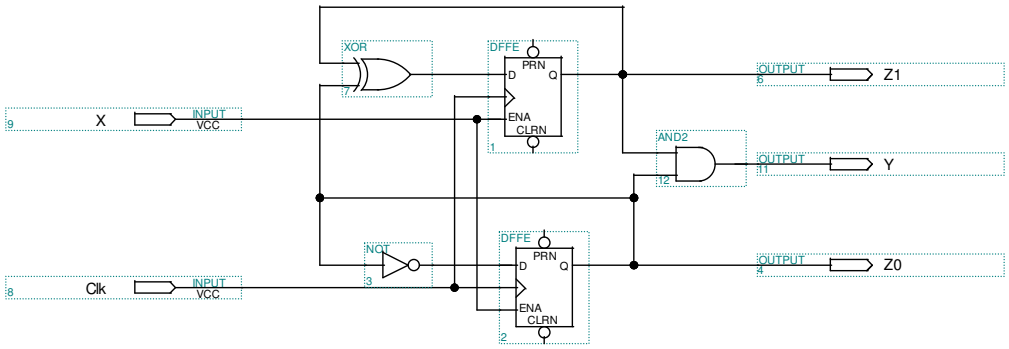
Zustandsgraph



x	z_1	z_0	z_1^*	z_0^*	y	D_1	D_0
0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1
1	0	1	1	0	0	1	0
0	1	0	1	0	0	1	0
1	1	0	1	1	0	1	1
0	1	1	1	1	1	1	1
1	1	1	0	0	1	0	0

zu 6. Generierung von D_1, D_0 und von y

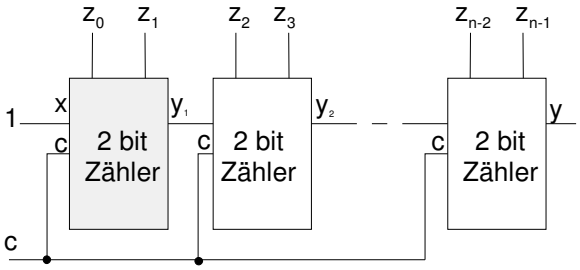
$D_0 = x\ \overline{z_0} \vee \overline{x}\ z_0 = x \oplus z_0$; $D_1 = \overline{x}\ z_1 \vee z_1\ \overline{z_0} \vee x\ \overline{z_1}\ z_0$ $y = z_1\ z_0$



VHDL:

```
Bis auf die Ausgangsfunktion Y <= Q(1) AND Q(0);
Gibt es keine Unterschiede zum Mealy-Zähler.
```

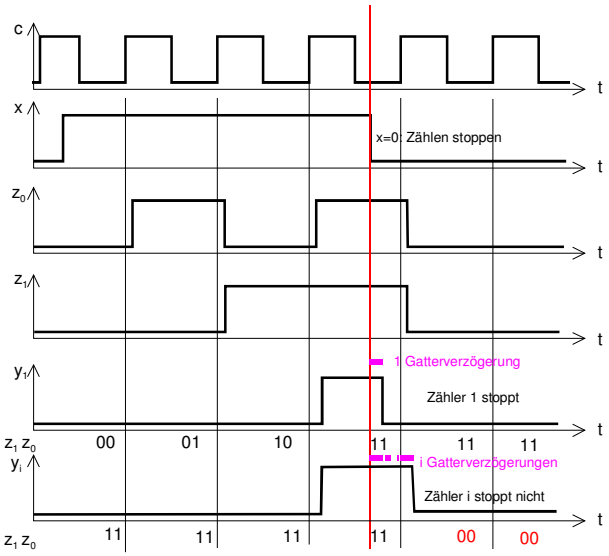
Anwendung des Beispiel-Zählers: Kaskadierung



Mögliche Timing-Probleme der Kaskadierung des Mealy-Zählers:

Eine Änderung des Freigabeeinganges x während des Zustandes Z_3=(11):
bewirkt mit einer Gatterlaufzeit-Verzögerung auch eine Änderung des Übertragsausgang y.

Bei einer Kaskadierung der Zähler ($x_n = y_{n-1}$) mit $0 \le n \le i$ addieren sich diese Zeiten, so daß sich bei hinreichend großer Stufenzahl i der Eingang x_i erst nach der Taktflanke ändert. Damit stoppt die Stufe i nicht korrekt, sondern zählt um 1 weiter.
Durch Verringerung der Taktfrequenz kann wieder eine korrekte Arbeitsweise erreicht werden.

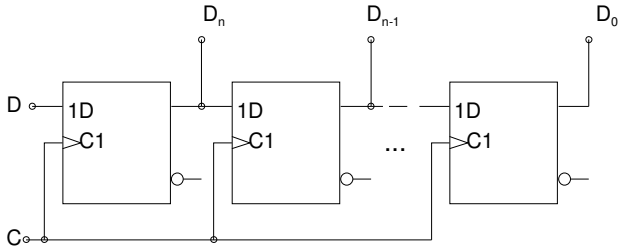


Mögliche Probleme der Kaskadierung des Moore-Zählers:
Kaskadierung von mehr als 2 Zählern ist nicht möglich, da das Übertragssignal y des 2. Zählers 4 Takte lang aktiv ist, so daß der 3. Zähler mehrere Takte lang zählt.

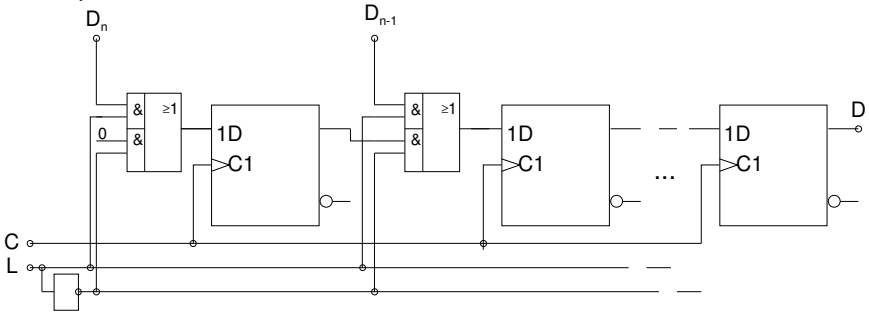
Architekturprinzipien bei der Verkopplung von Automaten:
- asynchrone Weitergabe von Signalen (Ripple-through-Prinzip)
  Mealy-Beispielzähler
  Vorteil: einfache Realisierung
  Problem: lange Verzögerungsketten erfordern langsamen Takt
- synchrone Weitergabe von Signalen (Pipeline-Prinzip)
  Vorteil: keine Abhängigkeit von Gatterlaufzeiten
  Problem: Verzögerung von Stufe zu Stufe um einen Takt, d.h. Baugruppen arbeiten untereinander verzögert.
  2. Moore-Beispielzähler
- Parallele und synchrone Weiterleitung von Signalen (Parallelarbeitsprinzip)
  Vorteil: keine Abhängigkeit von Gatterlaufzeiten
  Problem: höherer Hardwareaufwand, nicht beliebige Kaskadierbarkeit

spezielle Schaltwerke:

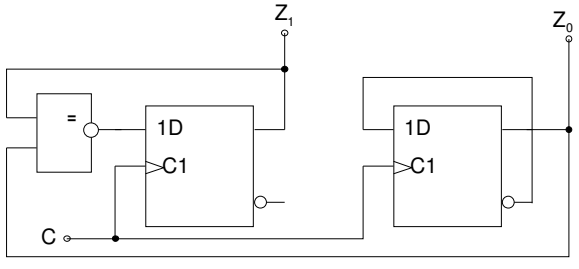
Serien-Parallel-Wandler
serieller Dateneingang D, paralleler Datenausgang D_n, D_{n-1}, ... D_0
Schieberegister:



Parallel-Serien-Wandler
paralleler Dateneingang D_n, D_{n-1}, ... D_0, serieller Datenausgang D
L = 1: paralleles Laden der Daten D_n, D_{n-1}, ... D_0

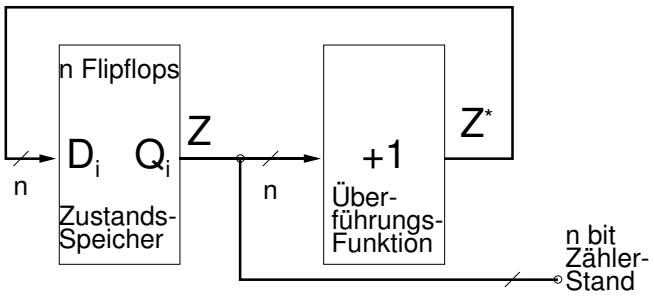


Autonome Schaltwerke
autonome Schaltwerke haben keine Eingangssignale X,
Anwendung: *Zähler*, *Frequenzteiler*
Beispiel: 2-bit-Zähler mit D-FF



$D_1 = z_1 \oplus z_0$ $D_0 = \overline{z_0}$

Verallgemeinerung (n-Bit-Zähler):



$Z = (z_{n-1} \dots z_0)$, $D = Z^* = Z + 1$ „Inkrementierer“

z_i	z_{n-1}	…	z_3	z_2	z_1	z_0
+	0	…	0	0	0	1
c_i	…	…	$z_2 \wedge z_1 \wedge z_0 \wedge 1$	$z_1 \wedge z_0 \wedge 1$	$z_0 \wedge 1$	
d_i	…	…	$z_3 \oplus (z_2 \wedge z_1 \wedge z_0 \wedge 1)$	$z_2 \oplus (z_1 \wedge z_0 \wedge 1)$	$z_1 \oplus (z_0 \wedge 1)$	$z_0 \oplus 1$

$d_0 = \overline{z_0}$

$d_1 = z_1 \oplus (z_0)$

n-Bit Zähler: $d_i = z_i \oplus \underbrace{(z_{i-1} \wedge z_{i-2} \wedge .... \wedge z_0)}_{c_{i-1}}$ $0 \le i \le n-1$

Rekursiv: $d_i = z_i \oplus c_{i-1}$ Aktuelle Zählstufe

mit $c_i = z_i \wedge c_{i-1}$ Übertrag zur nächsten Stufe

Für erste Stufe z_0 gilt: $c_{-1} = 1$ erste Stufe z_0 zählt

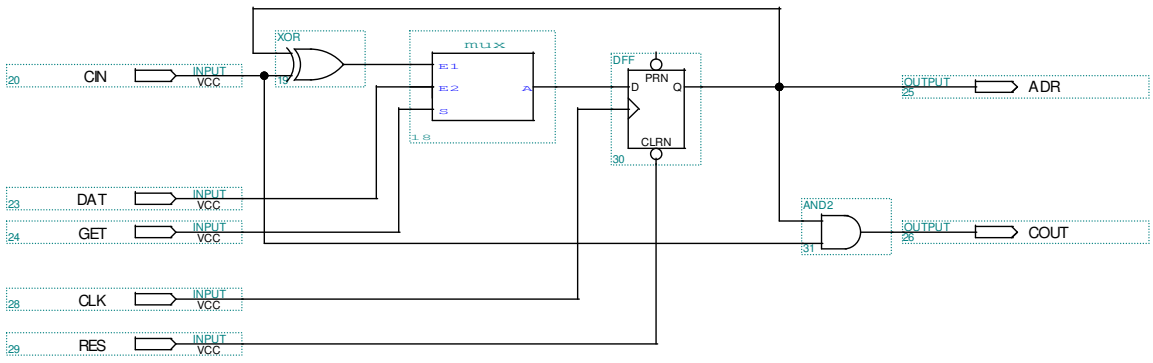
$c_{-1} = 0$ erste Stufe z_0 zählt nicht (und alle anderen auch nicht)

<u>Anwendung</u>:
1-Bit-Zählermodul mit Ladeeingang “GET”, Freigabe CIN und Übertragsausgang COUT

$D = GET\ DAT \vee \overline{GET}\ (Z \oplus CIN)$ (Multiplexer mit GET Steuerung)

$COUT = CIN \wedge Z$

$ADR = Z$



Dieses Modul kann zu einem N-Bit Zähler mit N Ausgängen
ADR(N-1), ADR(N-2), … ADR(1), ADR(0)
zusammengesetzt werden.

VHDL (vollständiger 1-Bit Zähler, rücksetzbar und ladbar)

```
LIBRARY IEEE; USE IEEE.STD_LOGIC_1164.ALL;
ENTITY bit_counter is
port (signal CIN, DAT, GET, CLK, RES: in STD_LOGIC;
      signal ADR, COUT: out STD_LOGIC);
end bit_counter;

ARCHITECTURE arch_bit_counter of bit_counter is
signal Z: STD_LOGIC;       -- int. ADR ohne in/out
begin
process (CLK,RES)
begin
 if RES = '0' then  -- RES=CLRN ist aktiv bei '0'
   Z <= '0' ;
 elsif rising_edge(CLK) then
   if GET = '1' then        -- Multiplexer,"wenn-dann"
     Z <= DAT;              -- laden
   else
     Z <= CIN XOR Z;        -- inkrementieren
   end if;
 end if;
end process;
ADR  <= Z;
COUT <= CIN AND Z;
end arch_bit_counter;
```

*Die Anweisungen im **process** werden von oben nach unten interpretiert.*
*Wie in einer gewöhnlichen Programmiersprache können daher **if, else** usw. geschachtelt werden, um das gewünschte Verhalten zu beschreiben.*

Die Funktion

$D = GET\ DAT \vee \overline{GET}\ (Z \oplus CIN)$

kann also statt
*Z <= (GET **AND** DAT) **OR** (**NOT** GET **AND** (Z **XOR** CIN);*

einfacher und übersichtlicher wie folgt beschrieben werden:
***if** GET = '1' **then** Z <= DAT;*
***else** Z <= CIN **XOR** Z;*
***end if**;*

Beachten:
*Die Zuweisung erfolgt immer auf Z, die Speicherung im D-FF erfolgt automatisch durch die **rising_edge** Bedingung.*
Z wird mit RES='0' korrekt in den Anfangszustand '0' rückgesetzt.

VHDL (vollständiger N-Bit Zähler, rücksetzbar, ladbar, zählt bei CIN=1, Übertrag COUT)

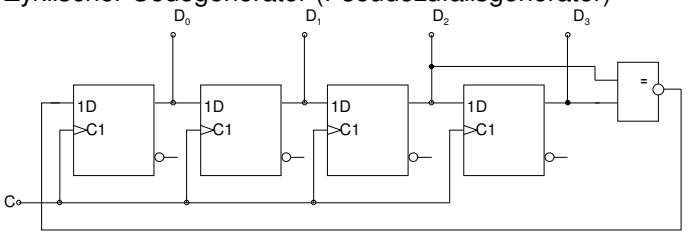
```
LIBRARY IEEE; USE IEEE.STD_LOGIC_1164.ALL;

ENTITY counter is
generic (N: INTEGER:=4);
port (signal CIN, GET, CLK, RES: in BIT;
      signal DAT:  in STD_LOGIC_VECTOR (N-1 downto 0);
      signal ADR: out STD_LOGIC_VECTOR (N-1 downto 0);
      signal COUT: out STD_LOGIC);
end counter;


ARCHITECTURE arch_ counter of counter is
component bit_counter is
port (signal CIN, DAT, GET, CLK, RES: in STD_LOGIC;
      signal ADR, COUT: out STD_LOGIC);
end component;
signal Z: STD_LOGIC_VECTOR (N-1 downto 0);    -- lokale Zustände
signal C: STD_LOGIC_VECTOR (N-1 downto 0);    -- lokale Überträge
begin
bit0: bit_counter port map(CIN, DAT(0), GET, CLK, RES, Z(0), C(0));
generate_bits:
  for I in 1 to N-1 generate                  -- I benötigt keine Deklaration
    bitX: bit_counter port map(C(I-1), DAT(I), GET, CLK, RES, Z(I), C(I));
  end generate;
ADR  <= Z;
COUT <= C(N-1);
end arch_compare;
```

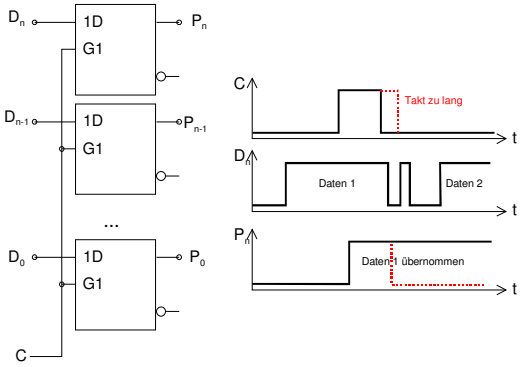
Es gibt weitere Lösungen für die Beschreibung von Zählern, wenn die mathematischen Operatoren + bzw. +1 auf STD_LOGIC_VECTOR angewendet werden können. Dazu sind spezielle Funktionen (packages) verfügbar, z.B. in der numeric_std oder STD_LOGIC_signed:
USE IEEE.STD_LOGIC_signed.ALL;

Zyklischer Codegenerator (Pseudozufallsgenerator)

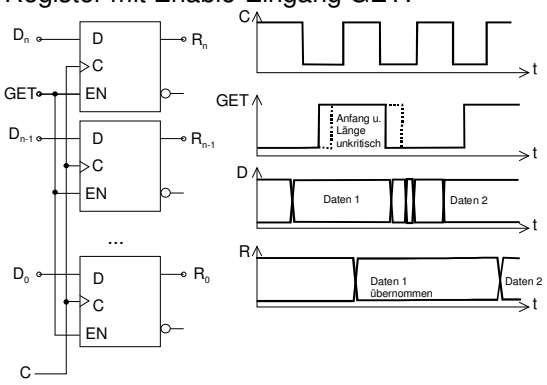


Speicherregister

Latch bzw. Parallel-Port:



Register mit Enable-Eingang GET:



Zusammenfassung sequentielle Schaltungen

- sequentielle Schaltungen haben ein zeitabhängiges Verhalten
- Die Zeit wird durch Taktung der Flipflops quantisiert
- durch Rückführungen werden Ausgangssignale und Eingangssignale verknüpft
- Rückführungen erfolgen aus Stabilitätsgründen immer über „geprüfte“ Flipflops
- digitale Automaten besitzen Zustandsspeicher, Überführungs- und Ausgangslogik
- der Mealy Automat läßt im Gegensatz zum Moore-Automaten eine kombinatorische Verknüpfung der Eingänge zum Ausgang zu.
- der Automatenentwurf erfolgt durch formale Beschreibung, Einführung von Flipflop-Zustandsspeichern und Generierung der Flipflop-spezifischen Überführungsfunktion und der Ausgangsfunktion
- bei synchronen Automaten besitzen alle Flipflops den selben Takt
- Der globale Takt und das Reset – Signal wird i.a. nachträglich in die Schaltung aufgenommen.
- Die Kopplung von Automaten ist durch eine asynchrone Signalweitergabe (ripple-through) oder synchrone Signalweitergabe (pipeline- oder parallel- Verarbeitung) möglich.

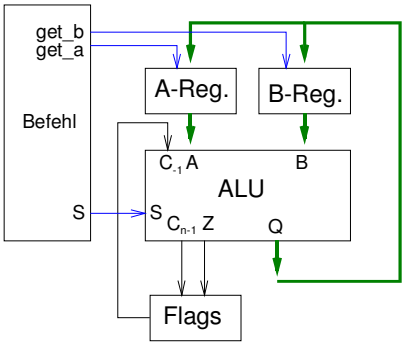
5.3. Rechenwerk

Die Zusammenschaltung einer n-Bit-ALU mit Speicherregistern über einen Datenbus ergibt eine Struktur, in der die Operanden und Ergebnisse der ALU zwischengespeichert werden können. Im allgemeinen Fall werden n-bit Register für die Operanden A und B sowie für das Ergebnis Q benötigt, damit können die folgenden Operationstypen realisiert werden:

Q := A ¤ B (¤: beliebiger ALU-Operator,
:= Zuweisung erfolgt mit einem Takt oder einem Enable-Signal)

Vereinfacht ist auch die Rückspeicherung nach A möglich, das ergibt den Operationstyp:

A := A ¤ B (¤: beliebiger ALU-Operator)
:= Takt oder Enable-Signal get_a in Verbindung mit dem Takt)



Beispiel einer 4-bit-ALU:

A=0010 (2)
B=1110 (14)
C_{-1}=0
S=[1,0,0,1] (Addition mit Übertrag)

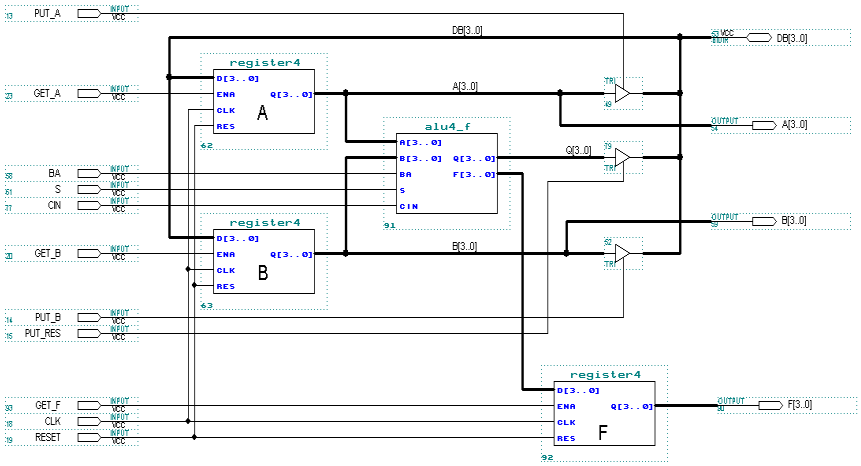
liefert nach der Verzögerungszeit der ALU:

Q=0000
Z=1
C_{n-1}=1

Der Wert von Q wird durch ein get_a Signal in das Ergebnisregister A (vier D-Flipflops) zurückgeschrieben.

Wenn der Datenbus mit den einzelnen Registern zusätzlich über Tristate-Treiber verbunden ist, können neben den Ergebnissen der ALU auch Werte von A oder B transportiert werden, z.B.

B := A, (:= Enable-Signal GET_B, Tristate-Enable PUT_A)
A := B, (:= Enable-Signal GET_A, Tristate-Enable PUT_B)
A := Q, bzw. A:= A ¤ B (:= Enable-Signal GET_A, Tristate-Enable PUT_RES)
B := Q, bzw. A:= A ¤ B (:= Enable-Signal GET_A, Tristate-Enable PUT_RES)



Es darf jeweils nur **ein** Tristate-Treiber aktiviert werden, um eine widersprüchliche Busbelegung zu vermeiden. Alternativ zu Tristate-Treibern können auch Multiplexer (hier: 3 auf 1) eingesetzt werden. Weiterhin ist ein Flag-Register F vorgesehen, welches der Speicherung der ALU-Ausgangssignale C und Z dient. Zusätzlich können das Vorzeichen Q_{n-1} (das MSB most significant bit) und ein Zweierkomplement-Überlauf Bit OV (ZK-Overflow) im F-Register abgespeichert werden. Für eine n-Bit ALU ergibt sich ein Überlauf, wenn im Ergebnis der Addition positiver Zahlen eine negative Zahl entsteht oder bei der Addition negativer Zahlen ein positives Ergebnis entsteht:

$$ZK = A_{n-1}\, B_{n-1}\, \overline{Q}_{n-1} \vee \overline{A}_{n-1}\, \overline{B}_{n-1}\, Q_{n-1}$$