Der Klassiker unter den zeitabhängigen Kippstufen ist der NE555, der sowohl als monostabile als auch als astabile Kippstufe betrieben werden kann.

NE555
SA555 - SE555

GENERAL PURPOSE SINGLE BIPOLAR TIMERS

- LOW TURN OFF TIME
- MAXIMUM OPERATING FREQUENCY GREATER THAN 500kHz
- TIMING FROM MICROSECONDS TO HOURS
- OPERATES IN BOTH ASTABLE AND MONOSTABLE MODES
- HIGH OUTPUT CURRENT CAN SOURCE OR SINK 200mA
- ADJUSTABLE DUTY CYCLE
- TTL COMPATIBLE
- TEMPERATURE STABILITY OF 0.005% PER°C

N
DIP8
(Plastic Package)

D
SO8
(Plastic Micropackage)

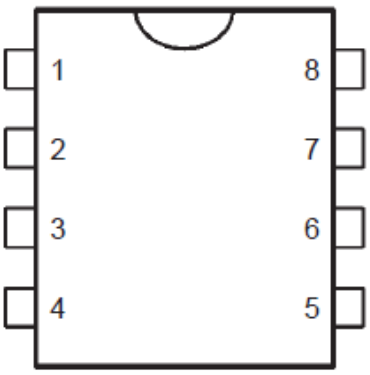
DESCRIPTION

The NE555 monolithic timing circuit is a highly stable controller capable of producing accurate time delays or oscillation. In the time delay mode of operation, the time is precisely controlled by one external resistor and capacitor. For a stable operation as an oscillator, the free running frequency and the duty cycle are both accurately controlled with two external resistors and one capacitor. The circuit may be triggered and reset on falling waveforms, and the output structure can source or sink up to 200mA. The NE555 is available in plastic and ceramic minidip package and in a 8-lead micropackage and in metal can package version.

ORDER CODES

Part Number	Temperature Range	Package	
		N	D
NE555	0°C, 70°C	•	•
SA555	–40°C, 105°C	•	•
SE555	–55°C, 125°C	•	•

PIN CONNECTIONS (top view)



1 - GND
2 - Trigger
3 - Output
4 - Reset
5 - Control voltage
6 - Threshold
7 - Discharge
8 - V_{CC}

5.4 Anwendung von Flipflops

5.4.1 Register

Register haben die Aufgabe, ganze **Bitvektoren** fester Länge, z.B. binäre (Daten)Worte, für eine bestimmte Zeit zu speichern.

Sie sind eine geordnete Menge von parallel angesteuerten Speicherelementen (oft 8, 16, 32 oder 64 Bit parallel).

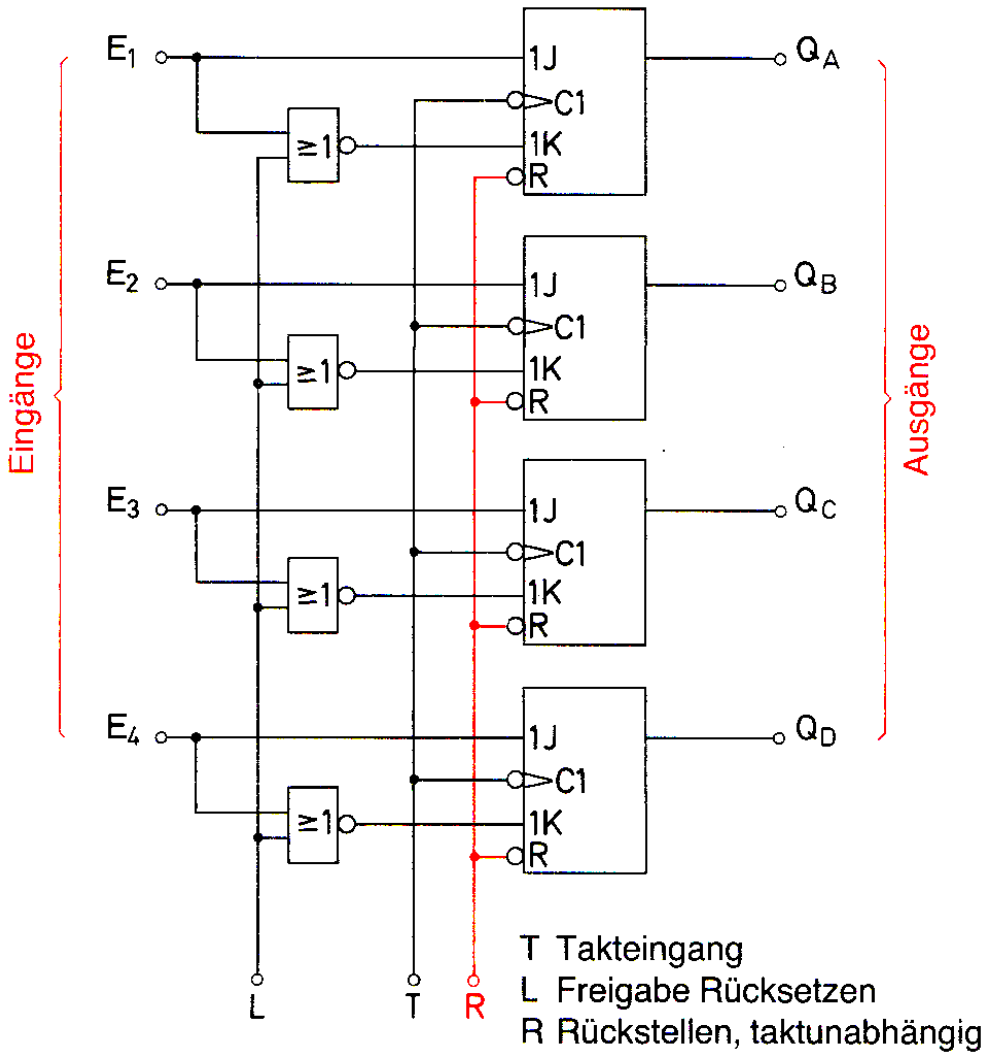
Register verfügen dazu über ein gemeinsames Übernahme-taktsignal, bei dem alle anliegenden Bits synchron und parallel übernommen werden.

Je nach Anwendung sind auch gemeinsame Steuerleitungen zum Freischalten des Taktes (Clock Enable), taktsynchronen oder asynchronen Rücksetzen ("Löschen", Clear) u.ä. aller Registerbits vorgesehen.

Register sind ein wesentlicher Bestandteil von komplexeren digitalen Systemen wie Mikroprozessoren, mit dem durch ein „synchrones Design“ laufzeitabhängige Effekte (Hazards) auszuschließen sind.

Die technische Realisierung erfolgt mit den bekannten Flipfloptypen, meist D-Flipflops.

Beispiel: 4-Bit-Register mit gesteuertem synchronen (L) und taktunabhängigem (asynchronem) Rücksetzen (R)



5.4.2 Schieberegister

Schieberegister sind Register, die ein Bitmuster taktgesteuert Bit für Bit aufnehmen und pro Takt um eine Stelle verschieben (shiften).

Schieberegister werden i. d. R. durch hinter einander geschaltete taktflankengesteuerte D-, RS- oder JK-Flipflops aufgebaut, die mit dem selben Takt betrieben werden.

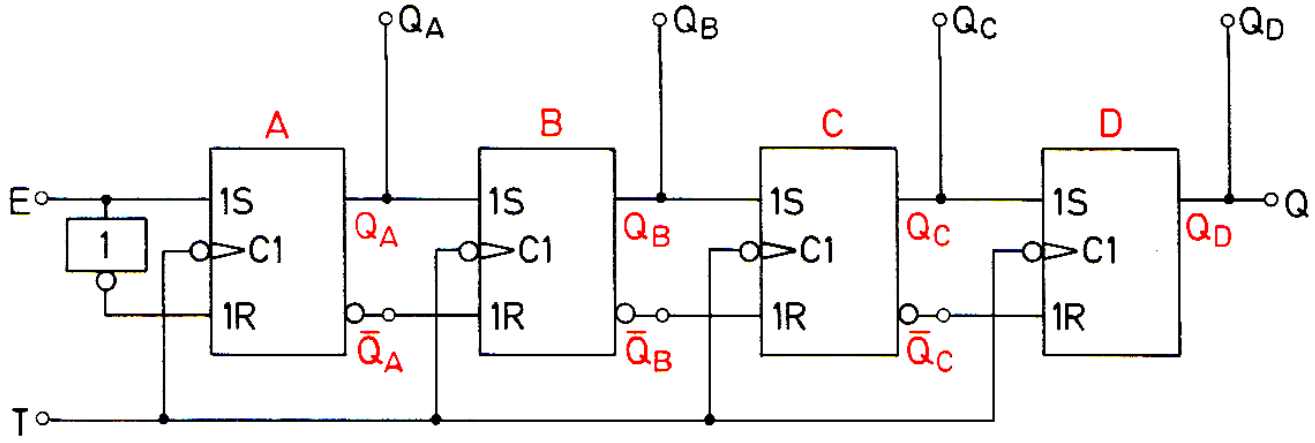
Sie eignen sich zur

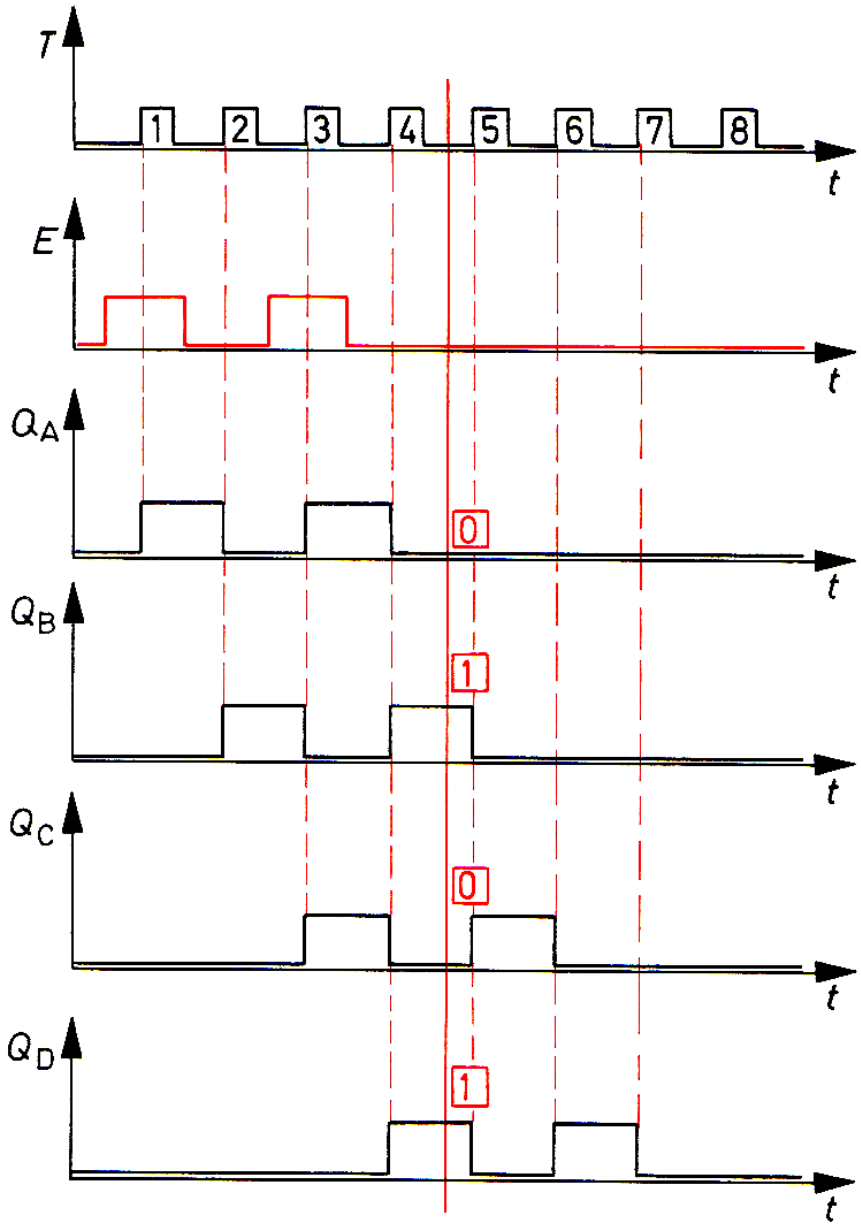
- Seriell-Parallel-Wandlung
- Parallel-Seriell-Wandlung
- Verzögerung von Daten um N Takte
- (bit)serielles Rechnen
- Multiplikation / Division von Dualzahlen
- Erzeugung von Pseudozufallszahlen

und vieles mehr.

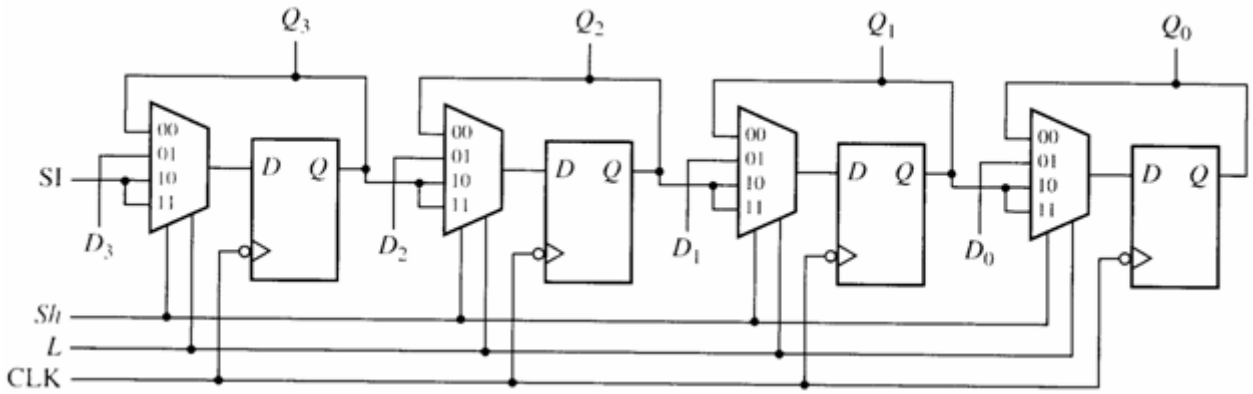
Schieberegister mit Parallelausgabe

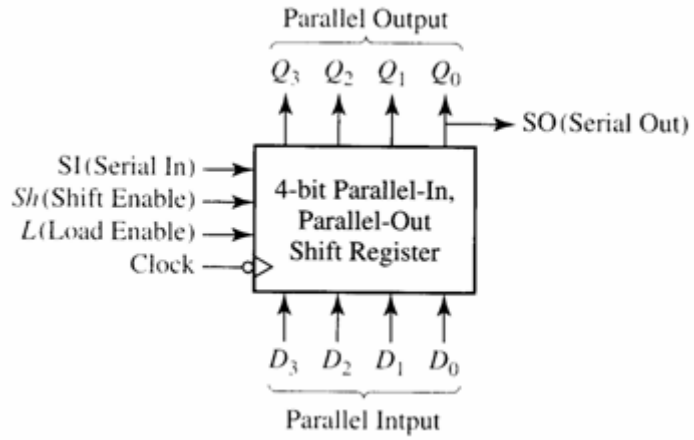
(z.B. für Seriell-Parallel-Wandlung oder Verzögerung eines seriellen Datenstroms)





Schieberegister mit parallelem Laden und Schiebe-Freigabe
(hier Realisierung mit D-Flipflops und Multiplexer)





Eingänge		Folgezustand				Funktion
Sh (shift)	L (Load)	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
0	0	Q_3	Q_2	Q_1	Q_0	keine Änderung
0	1	D_3	D_2	D_1	D_0	Laden
1	x	SI	Q_3	Q_2	Q_1	Rechts-Shift (Seriell-Parralel-Wandlung)

VHDL-Code für ein parallel ladbares Schiebregister

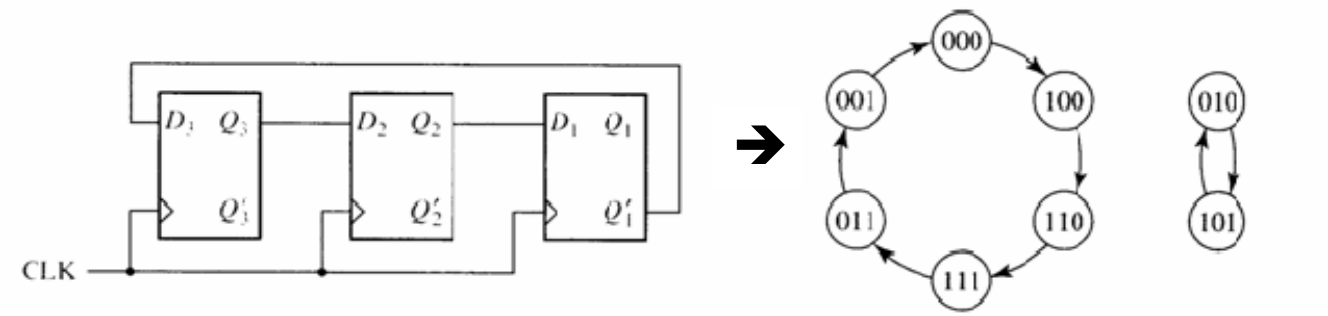
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

---------------------------------
-- 4-bit Parallel-Load Shift   --
--    Right Shift Register     --
---------------------------------

ENTITY shift_b IS
  PORT(n_ck, pl: IN std_logic;
       par_data: IN std_logic_vector (3 DOWNTO 0);
       q       : OUT std_logic_vector(3 DOWNTO 0));
END shift_b ;
```

```
ARCHITECTURE arc OF shift_b IS
    SIGNAL reg : std_logic_vector (3 DOWNTO 0);
BEGIN
    PROCESS (n_ck,pl)
    BEGIN
        IF (pl='1') THEN
            reg<=par_data;
        ELSIF (n_ck'EVENT AND n_ck='0' ) THEN
            reg(3)<= '0';     -- logical shift
            reg(2)<= reg(3);
            reg(1)<= reg(2);
            reg(0)<= reg(1);
        END IF;
        q<=reg;
    END PROCESS;
END arc;
```

Analyse von Schieberegister-Schaltungen

Die Zustände, in die ein Schieberegister von Takt zu Takt wechselt, lassen sich als Zustandsgraphen darstellen und dadurch ihr Verhalten veranschaulichen.

Beispiel: Invers zurückgekoppeltes 3-Bit-Schieberegister (Ringshift)



Hier gilt:

$$Q_3^+ = \overline{Q}_1$$
$$Q_2^+ = Q_3$$
$$Q_1^+ = Q_2$$

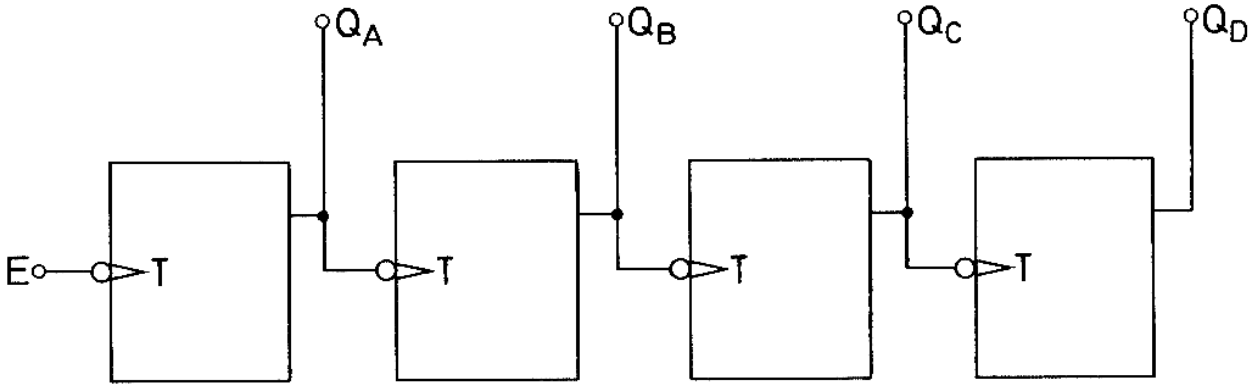
In gleicher Weise kann auch allgemein die Funktionsweise aus der Beschaltung von (Schiebe-) Registern aus einem Zustandsgraphen und/oder der Verschaltung abgeleitet werden (s. unten bei „Schaltwerken“).

5.4.3 Zähler

Zähler (Zählschaltungen) sind eine Gruppe von Flipflops, die ihren Zustand bei jedem Takt in einer vorgegebenen Zählweise (Zählsequenz) ändern. Man unterschiedet zwischen:

- Vorwärts- und Rückwärtszählern
- synchronen und asynchronen Zählern
- Zählern für verschiedene Codes

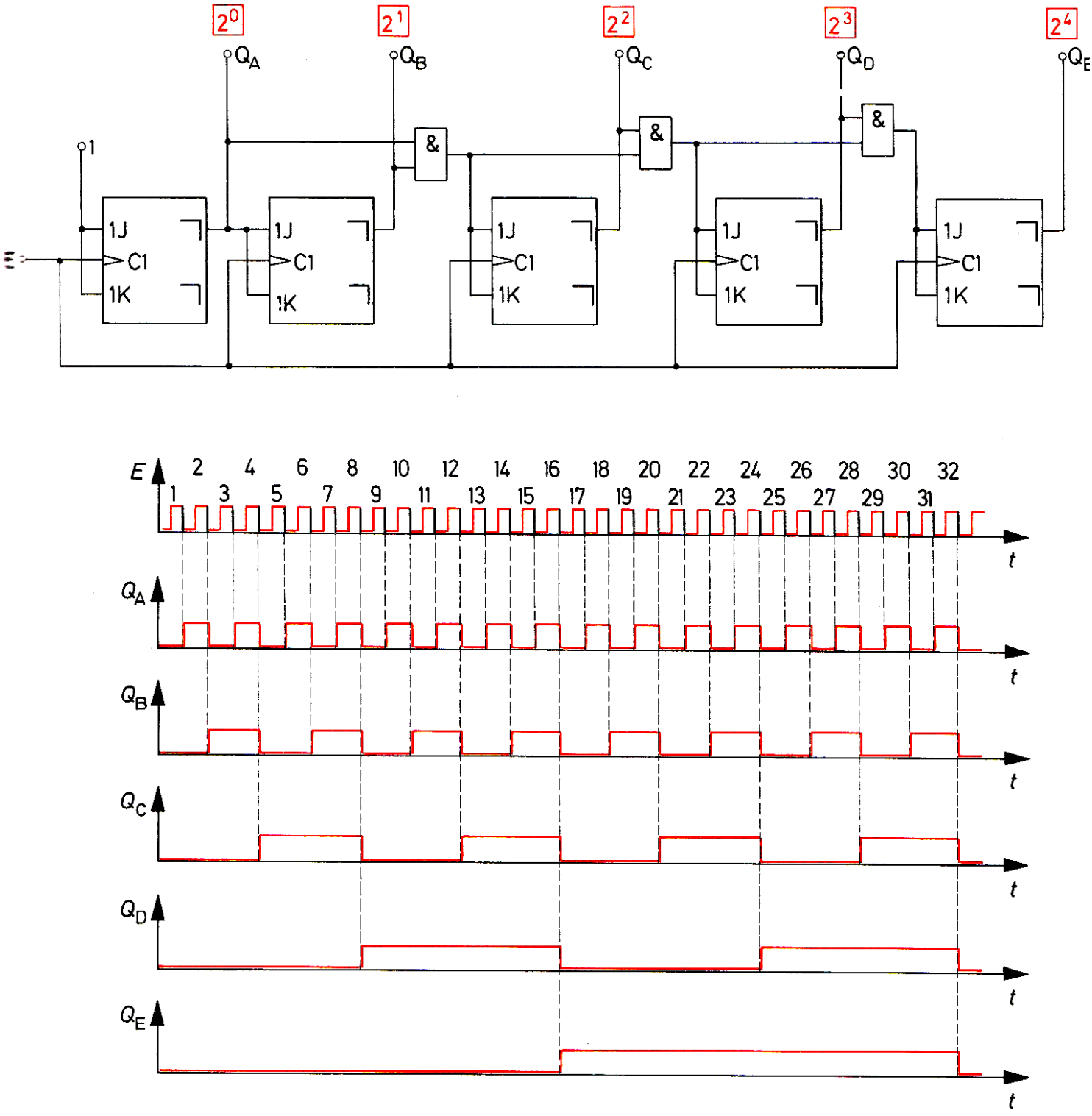
Dualzähler zählen von Null an bis zu ihrem Höchstwert von 2^N-1, schalten beim nächsten Takt wieder auf Null zurück und beginnen dann den Zählvorgang erneut.



Einfacher asynchroner 4-Bit-Dual-Vorwärtszähler
(mit negativ flankengetriggerten T-Flipflops)

Aktueller Zustand				Folgezustand			
Q_D	Q_C	Q_B	Q_A	Q_D'	Q_C'	Q_B'	Q_A'
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	0	1	0	0
…	…	…	…	…	…	…	…
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	1
…	…	…	…	…	…	…	…

Bei asynchronen Zählern treten i. d. R. beim Zählen durch Propagieren der internen Zustände (Ripple) ungewollt kurzfristig Übergangszustände als unplausible Ausgangszustände („Glitch“) auf. Um diese zu vermeiden, werden synchrone Zählschaltungen verwendet.



Synchroner 5-Bit-Dual-Vorwärtszähler

Zustandstabelle für den 5-Bit-Zähler

Aktueller Zustand Q_E, Q_D, Q_C, Q_B, Q_A	Flipflop-Eingänge $JK_E, JK_D, JK_C, JK_B, JK_A$	Folgezustand $Q_E', Q_D', Q_C', Q_B', Q_A'$
0 0 0 0 0	0 0 0 0 1	0 0 0 0 1
0 0 0 0 1	0 0 0 1 1	0 0 0 1 0
0 0 0 1 0	0 0 0 0 1	0 0 0 1 1
0 0 0 1 1	0 0 1 1 1	0 0 1 0 0
0 0 1 0 0	0 0 0 0 1	0 0 1 0 1
. . .	. . .	. . .

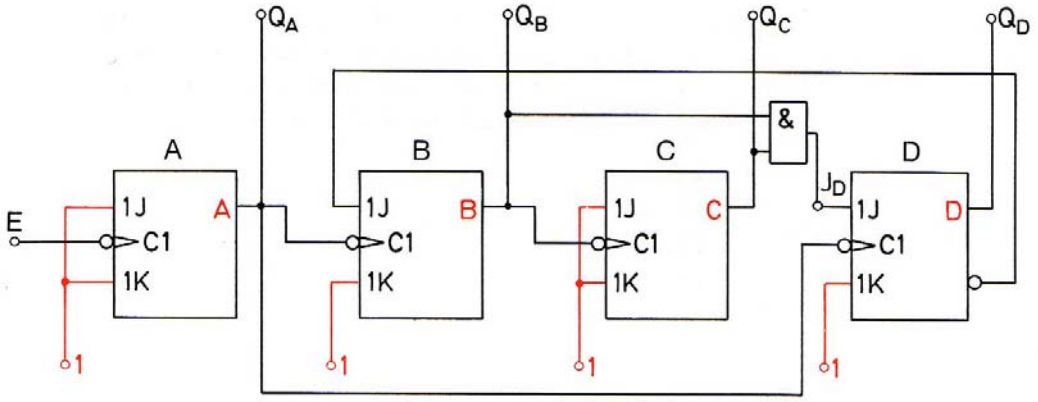
VHDL-Code für einen ladbaren 5-Bit-Dual-/Modulo-32-Vorwärtszähler

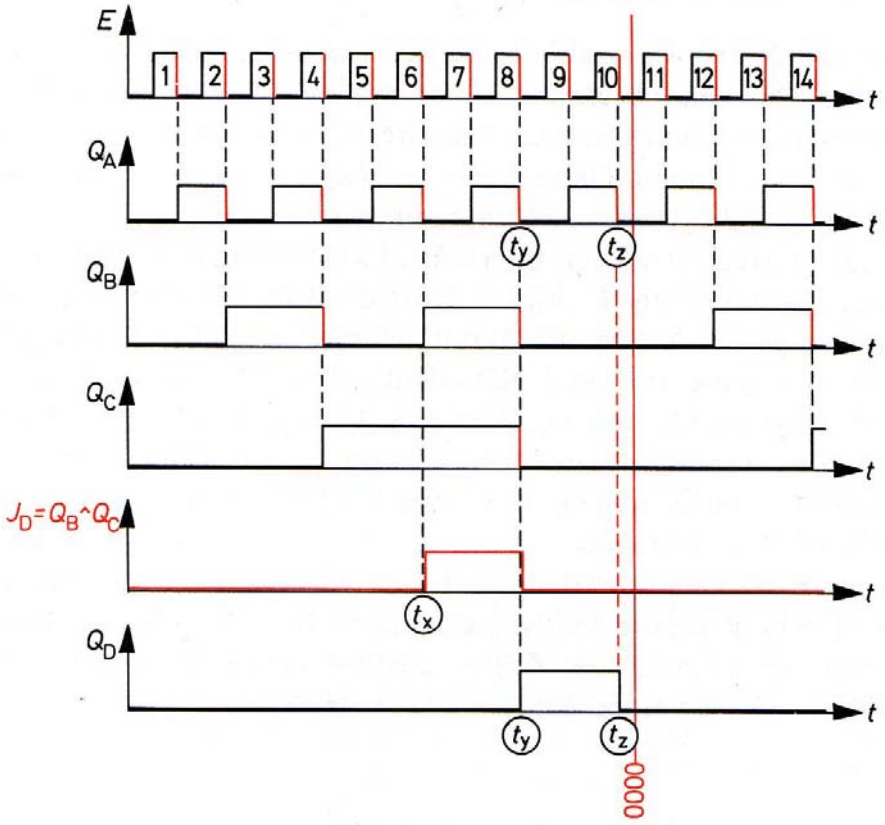
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-------------------------------
-- Mod-32 Up-Counter with load
-------------------------------
ENTITY mod32up IS
    PORT(n_ck, n_rd: IN  std_logic;
         q         : BUFFER integer RANGE 0 TO 31);
END mod32up;
```

```
ARCHITECTURE arch OF mod32up IS
BEGIN
    PROCESS (n_ck, n_rd)          -- negative active
    BEGIN
        IF (n_rd='0') THEN        -- reset direct
            q <= 0;
            ELSIF (n_ck'EVENT AND n_ck='0') THEN
                  q <= q+1;       -- count
        END IF;
    END PROCESS;
END arch;
```

Durch entsprechende Verknüpfung der Zählerbits lassen sich Zähler in beliebigen Codes realisieren (z.B. Aiken- oder Gray-Code). Am weitesten verbreitet sind BCD-Zähler.

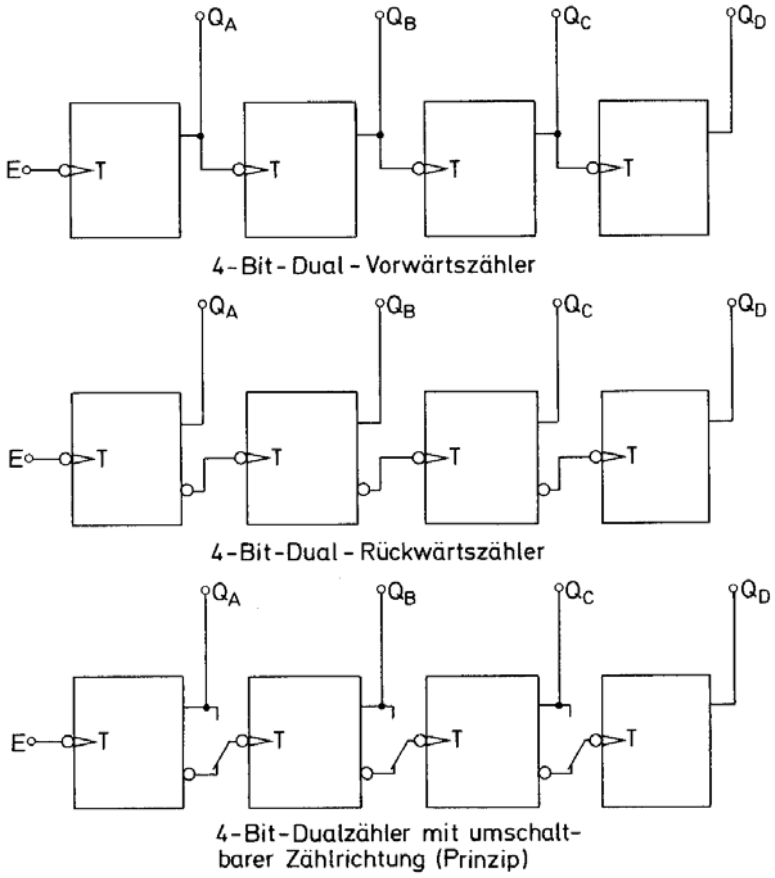
Beispiel: Negativ flankengetriggerter asynchroner BCD-Vorwärtszähler





Zähler können auch um Zusatzsignale für die Freischaltung des Zählers (Enable), die Zählrichtung, (asynchrones) Rücksetzen oder Setzen (Laden) erweitert werden.

Beispiel: Asynchroner BCD-Zähler mit umschaltbarer Zählrichtung

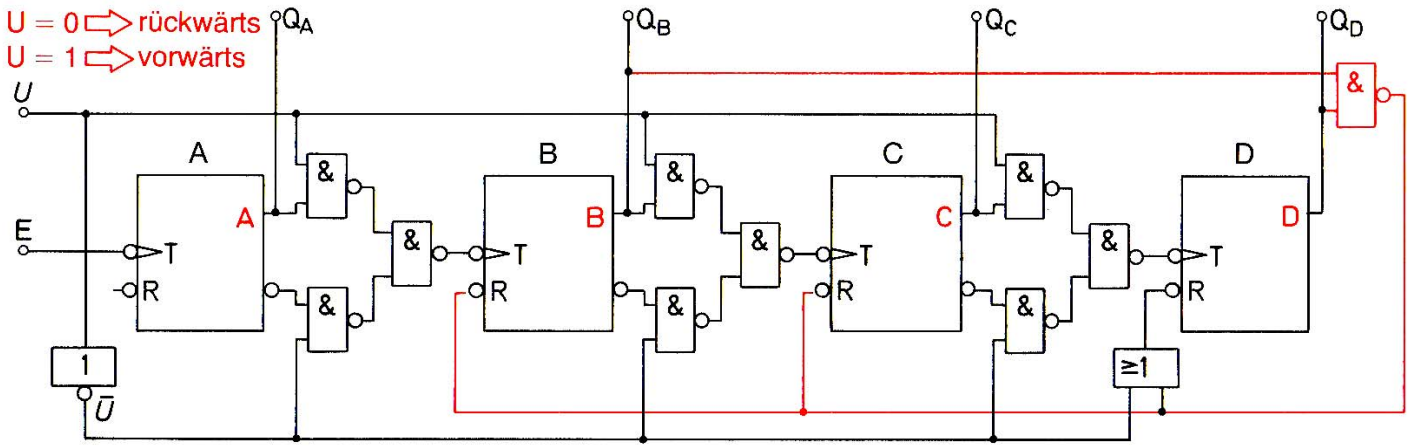


4-Bit-Dual-Vorwärtszähler

4-Bit-Dual-Rückwärtszähler

4-Bit-Dualzähler mit umschaltbarer Zählrichtung (Prinzip)

Prinzipielle Entwicklung zur umschaltbaren Zählrichtung



VHDL-Code für einen ladbaren, synchronen (glitch-freien) Modulo-10-Zähler

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-----------------------------------
-- Mod-10 Glitch-Free Up-Counter --
-----------------------------------

ENTITY mod10up IS
    PORT(n_ck, n_rd: IN  std_logic;
         q         : BUFFER integer RANGE 0 TO 15);
END mod10up12;
```

```
ARCHITECTURE arch OF mod10up IS
BEGIN
    PROCESS (n_ck, n_rd)
    BEGIN
        IF  (n_rd='0') THEN      -- reset
            q <= 0;
            ELSIF (n_ck'EVENT AND n_ck='0') THEN
                IF (q=9) THEN    -- clocked overflow
                   q <= 0;        -- detection
                   ELSE q <= q+1; -- count
                END IF;
        END IF;
    END PROCESS;
END arch;
```

Je nach Abstraktion der Entwurfsebene hat der Entwickler keine oder nur eine geringe Kontrolle über das zeitliche Übergangsverhalten des Zählers.

Systematischer Entwurf von Synchronzählern

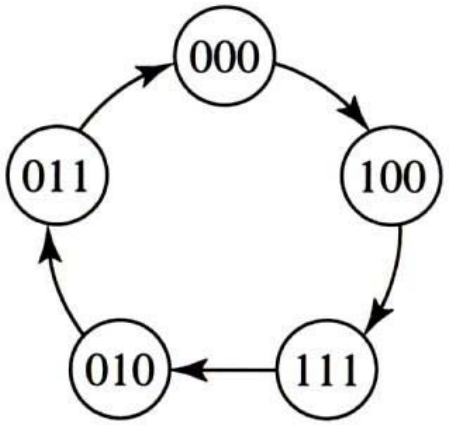
Ablauf zum Entwurf von Synchronzählern:

- Aufstellen der Wahrheitstafel gemäß der gewünschten Zählfunktion
- Aufstellen und Vereinfachen der Funktionsgleichungen
- Bestimmen der charakteristischen Gleichungen der zu verwendenden Flipflops
- Bestimmen der Verknüpfungsgleichungen zur Ansteuerung der Flipflops durch Koeffizientenvergleich
- Zeichnen des Schaltbilds nach den Verknüpfungs-gleichungen

<u>Ansteuerung für gewünschte Folgezustände bei typischen Flipflop-Typen</u>

Flipflop-Typ	Eingang	$Q^n = 0$		$Q^n = 1$	
		$Q^{n+1}=0$	$Q^{n+1}=1$	$Q^{n+1}=0$	$Q^{n+1}=1$
Delay	D	0	1	0	1
Trigger	T	0	1	1	0
RS	S	0	1	0	x
	R	x	0	1	0
JK	J	0	1	x	x
	K	x	x	1	0

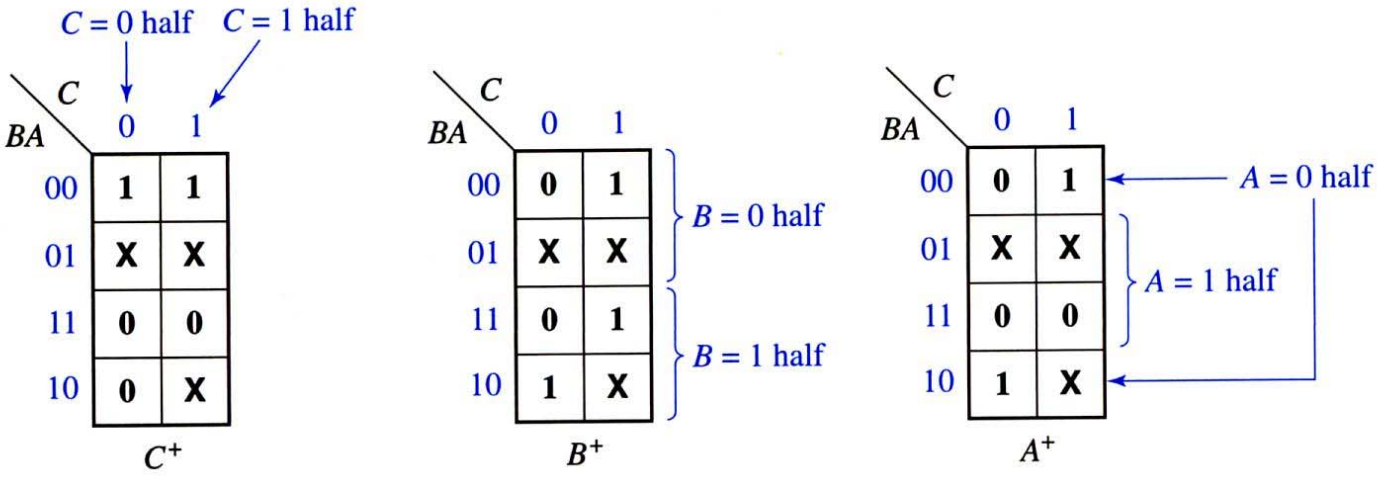
Beispiel: Zähler für folgende vorgegebene Zählfolge



Zustandsübergangstabelle:

C^n	B^n	A^n	C^{n+1}	B^{n+1}	A^{n+1}
0	0	0	1	0	0
0	0	1	-	-	-
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	-	-	-
1	1	0	-	-	-
1	1	1	0	1	0

KV-Diagramme aus der Übergangstabelle:

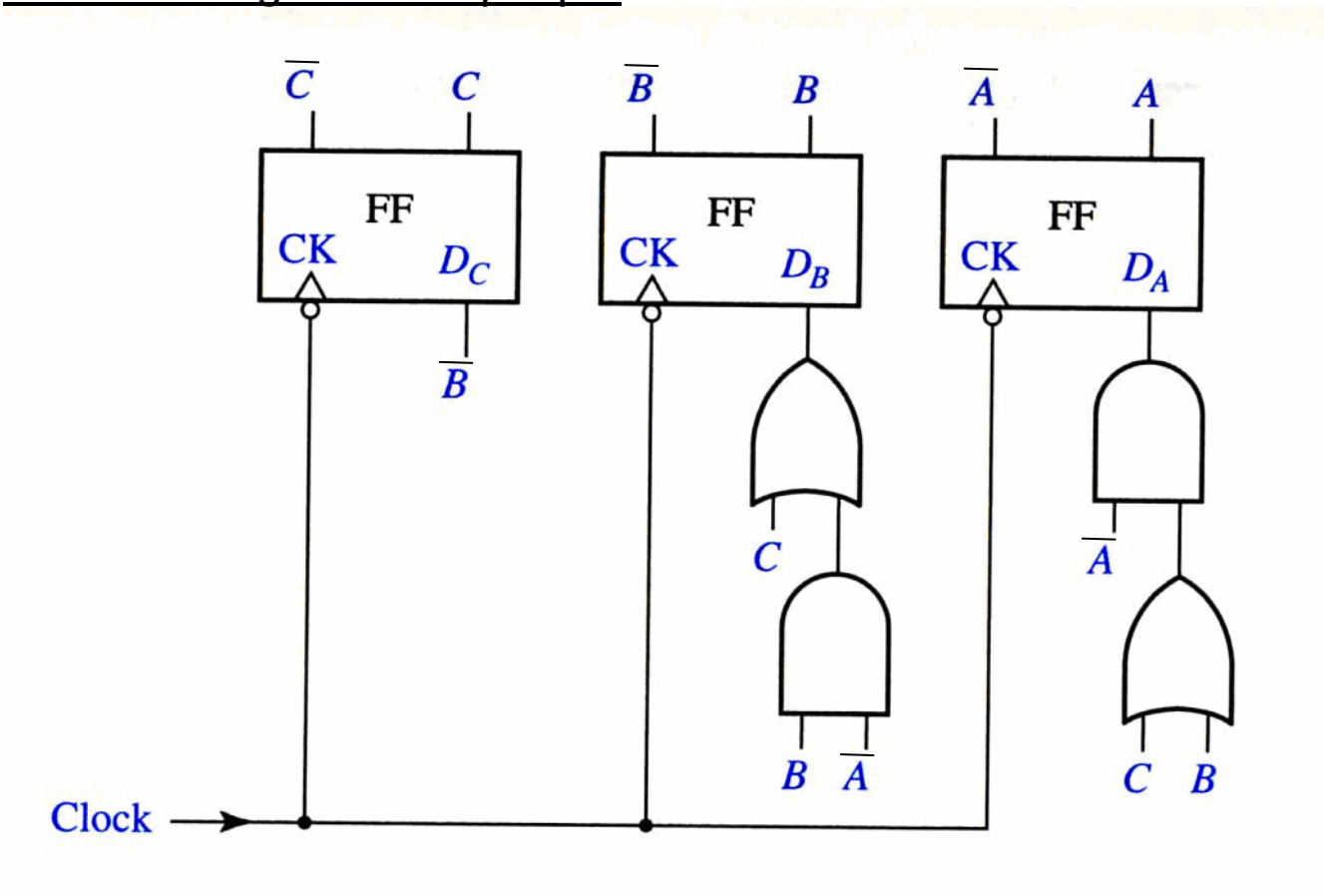


Ansteuergleichungen für D-Flipflop-Realisierung:

$$D_C^n = C^{n+1} = \overline{B}^n, \qquad D_B^n = B^{n+1} = C^n + B^n\overline{A}^n$$

$$D_A^n = A^{n+1} = C^n\overline{A}^n + B^n\overline{A}^n = \overline{A}^n(C^n + B^n)$$

Verschaltung der D-Flipflops:



Ansteuergleichungen für alternative Realisierung mittels RS-Flipflops (analog herzuleiten)

$$S_A = \overline{A}(B+C) \qquad R_A = A$$

$$S_B = C \qquad R_B = A\overline{C}$$

$$S_C = \overline{B} \qquad R_C = A$$

5.4.4 Frequenzteiler

Frequenzteiler sind (eine Sonderform von) Zählschaltungen, die die Frequenz binärer Signale in einem bestimmten Verhältnis runterteilen.

Jeder Zähler kann auch als Frequenzteiler eingesetzt werden.

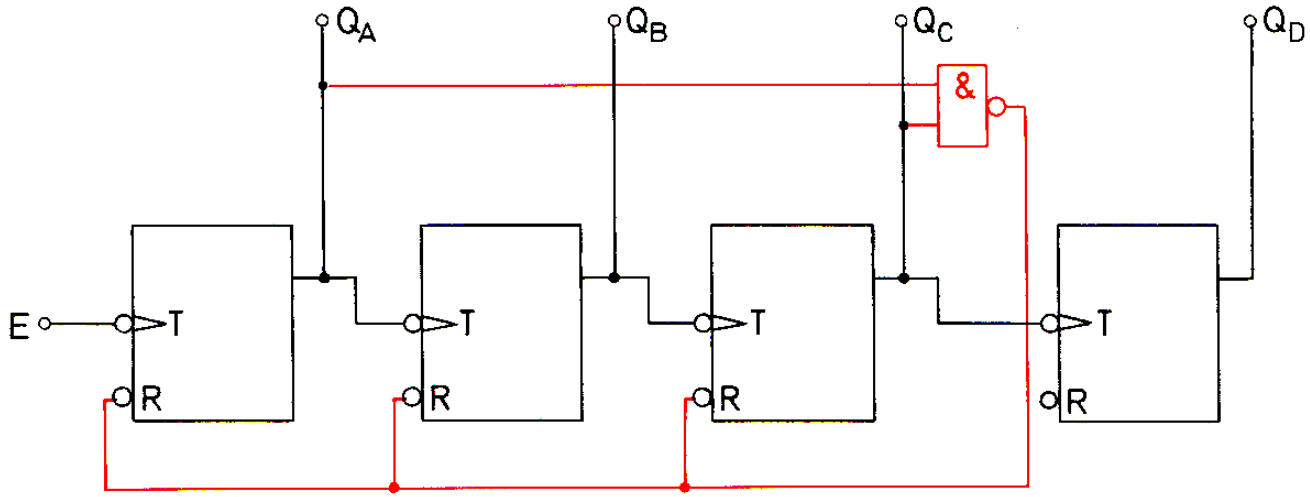
Bei *n*-Bit-Dualzählern stehen die geteilte Frequenz f_t am Ausgang und die Eingangsfrequenz f_E in dem Verhältnis:

$$f_t = \frac{f_E}{2^n}$$

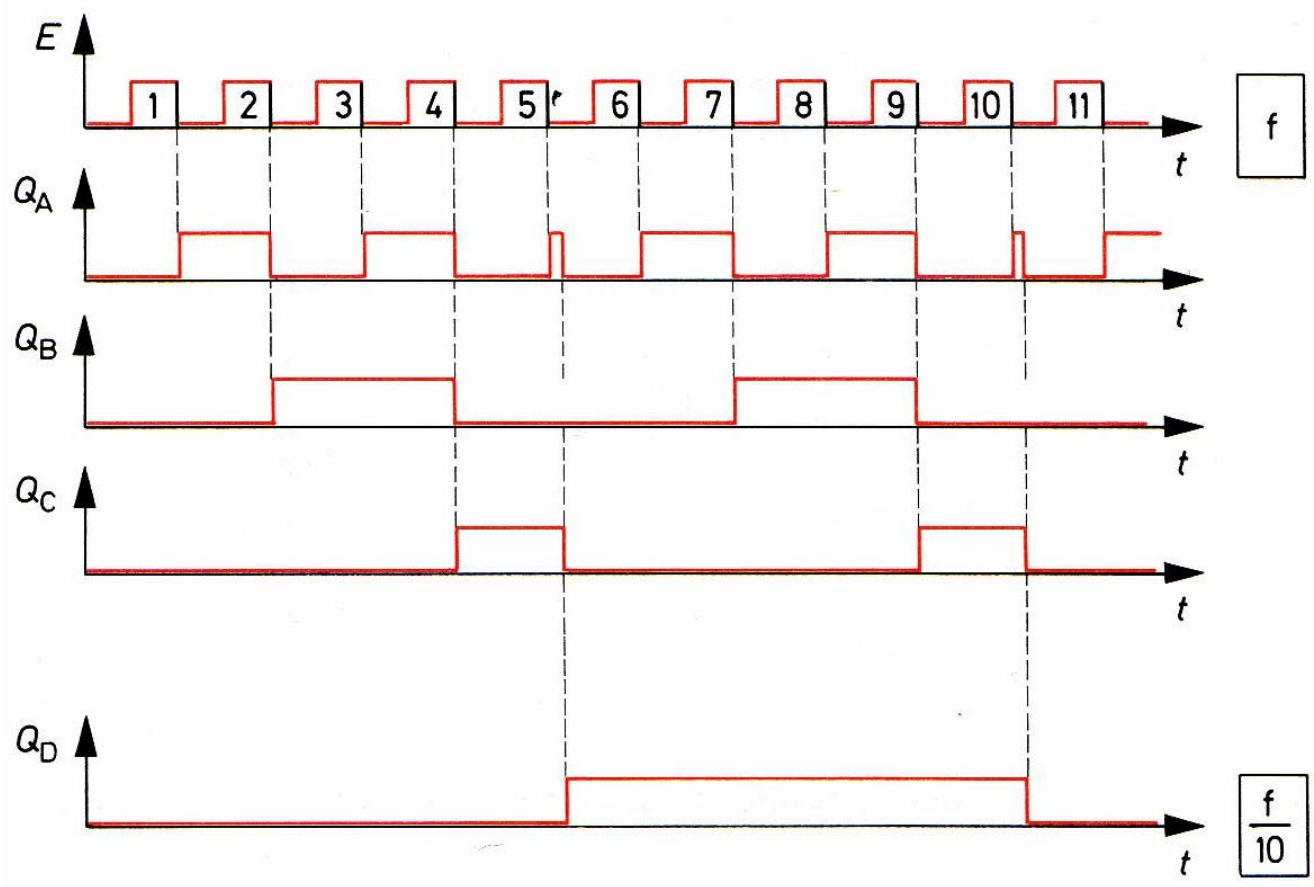
Um andere Teilerverhältnisse zu erzielen, werden die Zählbits beim Erreichen des maximalen Zählerstandes über eine entsprechende Verknüpfung durch die Rücksetzeingänge der Flipflops zurückgesetzt.

Dabei entstehen i. d. R. asymmetrische Impulse. Für gerade Teilerverhältnisse kann dies durch eine Frequenzhalbierung in der letzten Stufe ausgeglichen werden.

Frequenzteiler mit einem Teilerverhältnis von 10:1



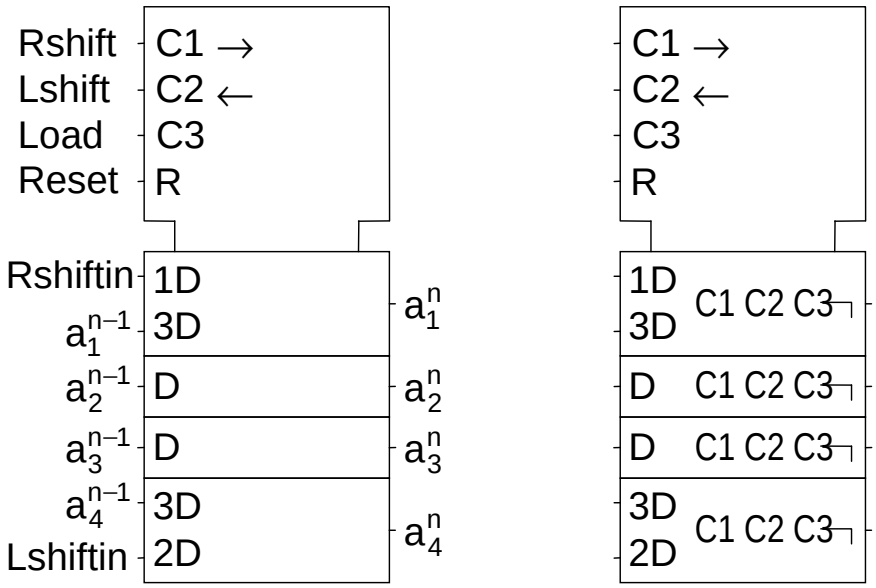
Impulsdiagramm für einen Frequenzteiler 10:1



Normierte Schaltsymbole für Register (DIN)

Wenn Steueranschlüsse auf mehrere andere Anschlüsse wirken, wird ihnen eine eindeutige Identifikation(snummer) gegeben und bei den Ein-/Ausgängen angegeben, worauf sie wirken.

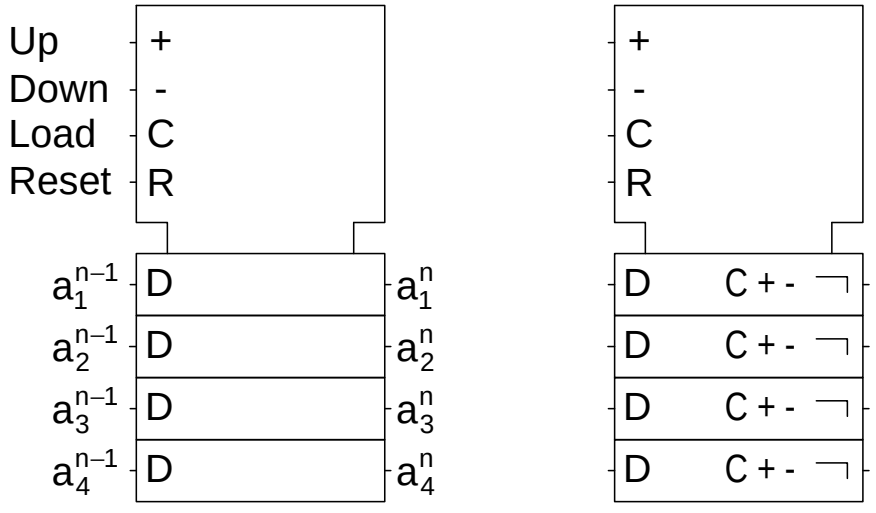
Universalregister mit Ladeeingängen und Links/Rechtsshift



positiv einflankengetriggert

Master-Slave

Vorwärts/Rückwärtszähler mit Ladeeingängen und Reset



positiv einflankengetriggert

Master-Slave

5.5 Registerbänke und Speicher

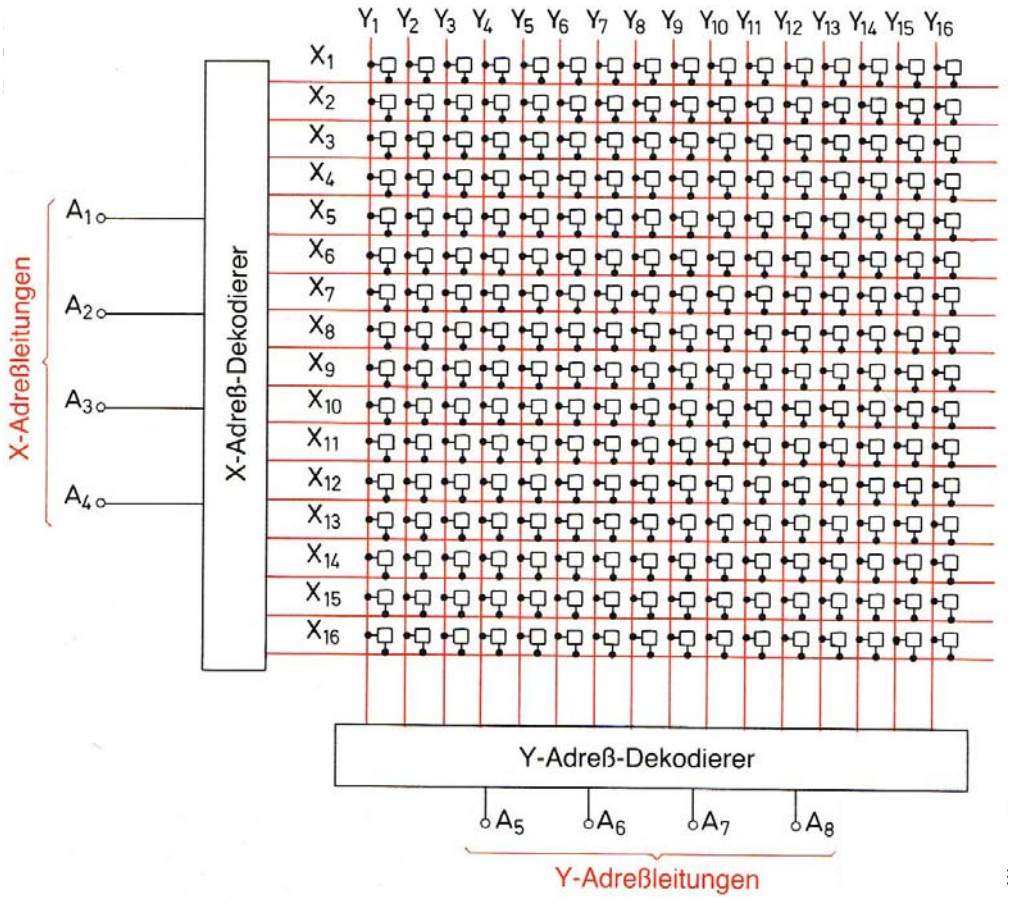
Mehrere Register werden vielfach zu Registerbänken zusammengefasst, bei denen immer wortweise nur ein Register gleichzeitig gelesen oder geschrieben wird.

Die Auswahl des zu lesenden oder zu schreibenden Registers erfolgt über *Adressierung*, die die Ausgänge bzw. das Schreiben des jeweiligen Registers freischaltet (enable).

Bei einer großen Anzahl von Registern wächst der Aufwand für die Adressdekodierung und Auswahl der Register stark an.

Speicher werden deshalb matrixförmig organisiert und wortweise spalten- **und** zeilenweise angesteuert (Adresse und Schreib-/Lesesignal, ggf. auch *Chip-Select*).

Schema eines 256×1-Speichers



Speichertypen

Speicher werden in verschiedenen Techniken realisiert.

Flüchtige Speicher:
(verlieren den Speicherinhalt bei Abschalten der Versorgungs-spannung)

- RAM — Random Access Memory
- SRAM — Static RAM
- DRAM — Dynamic RAM
- Dual-ported RAM — Speicher, auf den zwei Einheiten (gleichzeitig) zugreifen können

Nichtflüchtige Speicher
(behalten den Speicherinhalt bei Abschalten der Versorgungs-spannung)

- ROM — Read Only Memory
- PROM — Programmable ROM
- EPROM — Erasable PROM
- EEPROM — Electrically Erasable PROM
- Flash-ROM — wie EEPROM, nur andere Technologie

VHDL-Code für ein einfaches ROM

```
-- ROM-Beispiel für 8 Worte mit 4 Bit
ENTITY ROM_example is
    PORT(address: IN  BIT_VECTOR(2 DOWNTO 0);
         data   : OUT BIT_VECTOR(0 TO 3);
END ENTITY;
```

```
ARCHITECTURE ROM OF ROM_example IS
TYPE ROM8X4 IS ARRAY(0 TO 7) OF BIT_VECTOR(0 TO 3);
CONSTANT ROM1: ROM8X4 := ("1100", "1001", "0000",
         "0101", "1111", "0000", "0000", "1101");
BEGIN
    data := ROM1(address);
END ROM;
```

5.6 Registertransfer

5.6.1 Registeroperationen

Durch die getaktete Arbeitsweise von Digitalschaltungen kann man insbesondere bei einer Taktflankensteuerung oder beim Master-Slave-Prinzip erreichen, dass alle Laufzeiteffekte, die bei einer Verknüpfung in (mehrstufigen) Schaltnetzen auftreten, abgeklungen sind, wenn die nächste aktive Flanke/Taktphase kommt. Weil Hazards so keine Rolle mehr spielen, wird sichergestellt, dass Signalzustände immer stabil sind, bevor sie übernommen werden.

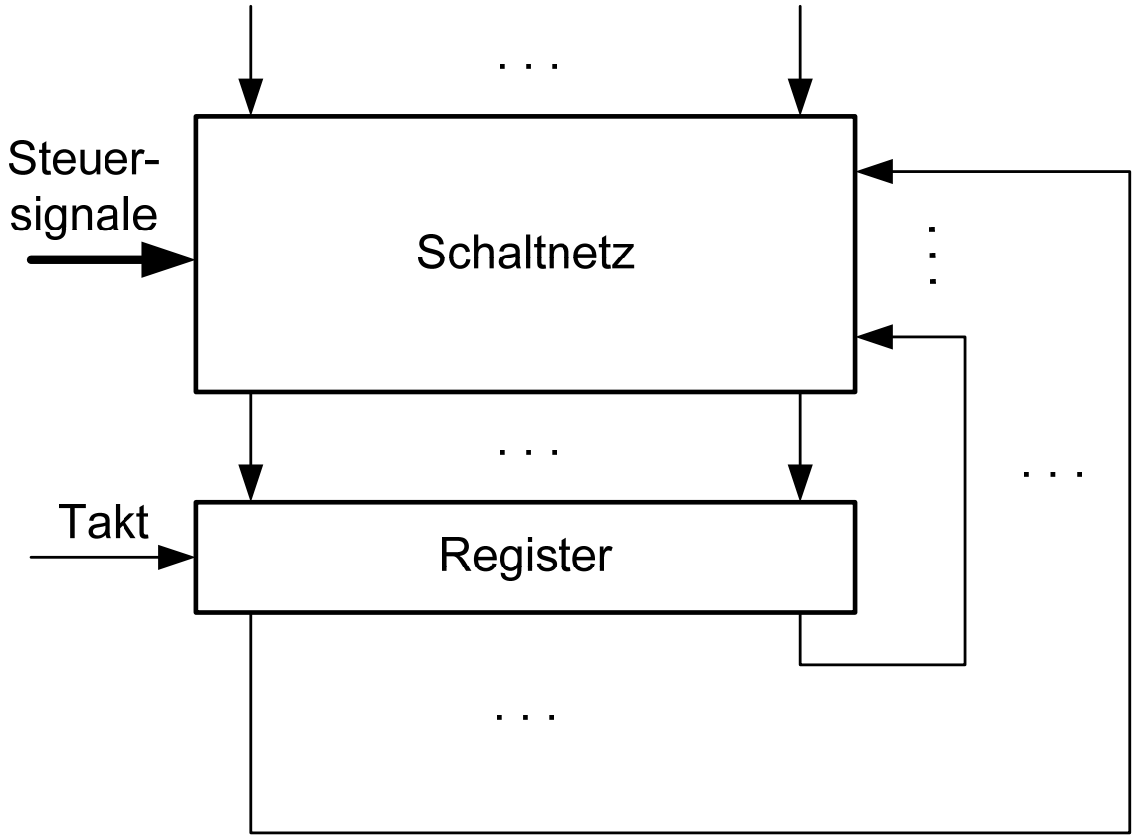
Das führt zur synchronisierten Verarbeitung mit synchronem (Zwischen-)Speichern der Ergebnisse in Registern mit Schaltnetzen in den dazwischen liegenden Verarbeitungsschritten.

Digitale Systeme, bei denen Daten in Registern (zwischen) gespeichert und deren Inhalte durch Schaltnetze verknüpft werden, spielen deshalb eine zentrale Rolle in der Technischen Informatik.

In komplexeren digitalen Schaltungen nutzt man dieses Prinzip systematisch aus und holt die zu verarbeitenden Daten aus einem Register und speichert die Verarbeitungsergebnisse, nachdem sie stabil sind, wiederum in einem Register.

Die Verarbeitung erfolgt also **getaktet** innerhalb des digitalen Systems durch den **Transfer** von Daten zwischen zwei **Registern**, zwischen denen sich Verknüpfungsschaltungen (Schaltnetze) befinden. Dabei können Quell- und Zielregister auch identisch sein.

Diese **Registertransfer**-Sicht ist eine wichtige Abstraktionsebene beim Entwurf digitaler Schaltungen und Systeme. Ihr liegt folgende allgemeine Struktur zugrunde:

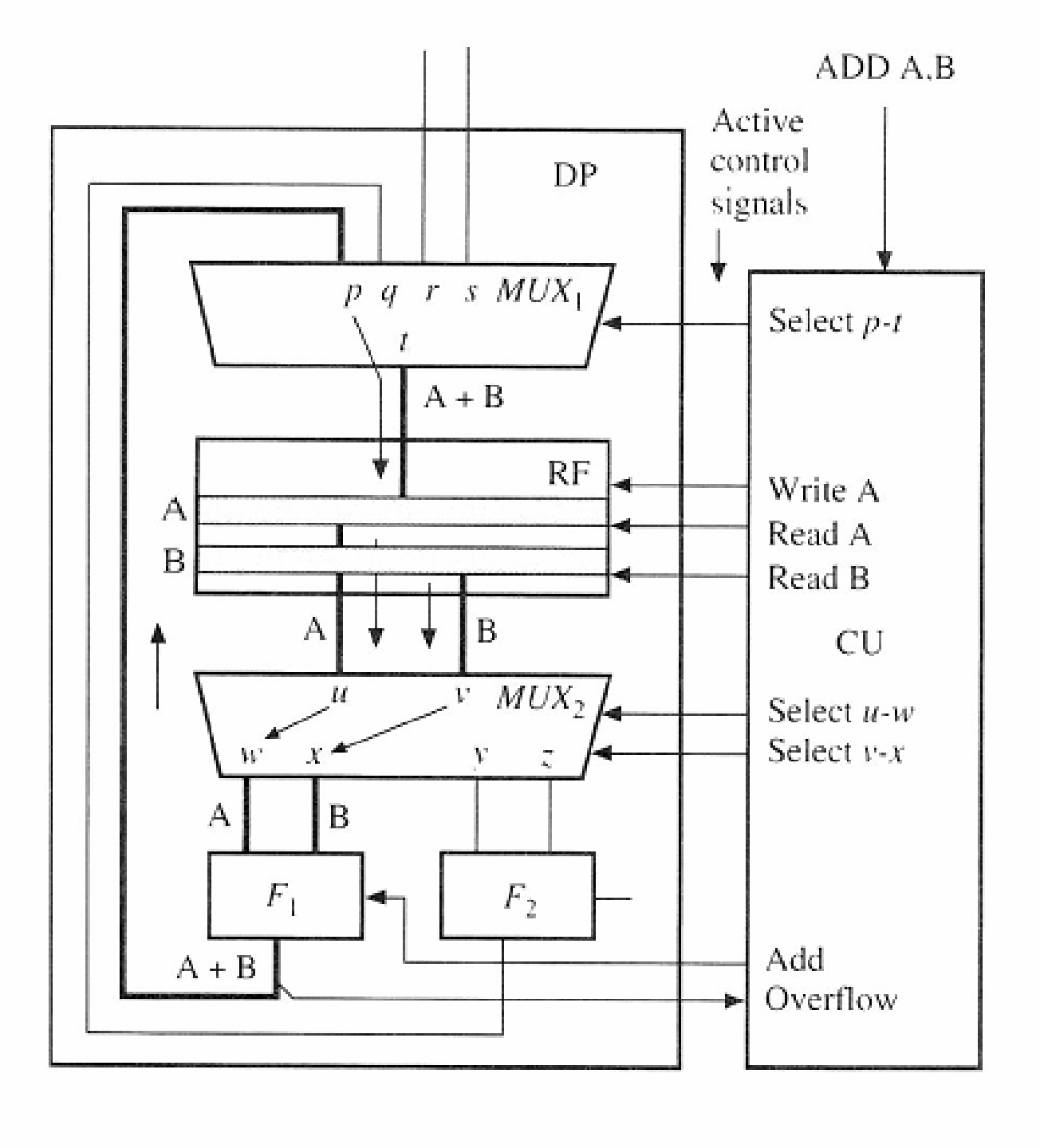


Die **allgemeine Notation** für einen Registertransfer ist:

Ziel(register) ← *Ausdruck über Quellen(register)*

Bei dieser Registertransfer-Beschreibung wird die Taktung dann implizit angenommen.

Beispiel: Einfacher Prozessor (Ein-Zyklus-Operation)



Mit:
- DP = Data Path,
- CU = Control Unit,
- MUX = Multiplexer,
- RF = Register File,
- F_1, F_2 = Functional Units

Das Lesen und Schreiben von Registern sowie die Verarbeitung der transferierten Daten wird durch eine separate (übergeordnete) Steuereinheit koordiniert, die den Ablauf der Verarbeitung und des Datentransfers steuert (Ablaufsteuerung, Schaltwerk).

Auf diese Weise wird beim Entwurf komplexer digitaler Schaltungen klar in die Verarbeitung (von Daten im „**Operationswerk**“) und Ablaufsteuerung (im „Kontrollwerk“, „**Steuerwerk**“) strukturiert und durch die getaktete Verarbeitung auch von Laufzeiteffekten abstrahiert.

Der Entwurf kann dadurch auf einer abstrakteren Ebene, der **Registertransfer-Ebene** (RT-Ebene; RTL: Register Transfer Level), erfolgen, die nur noch den Transfer von Register zu Register und die dazwischen stattfindende Operation spezifiziert. Sie ist beim Entwurf digitaler Systeme eine wichtige Beschreibungsebene oberhalb der Logikebene (Gatter, Flipflops), durch die eine größere Komplexität beherrschbar wird.

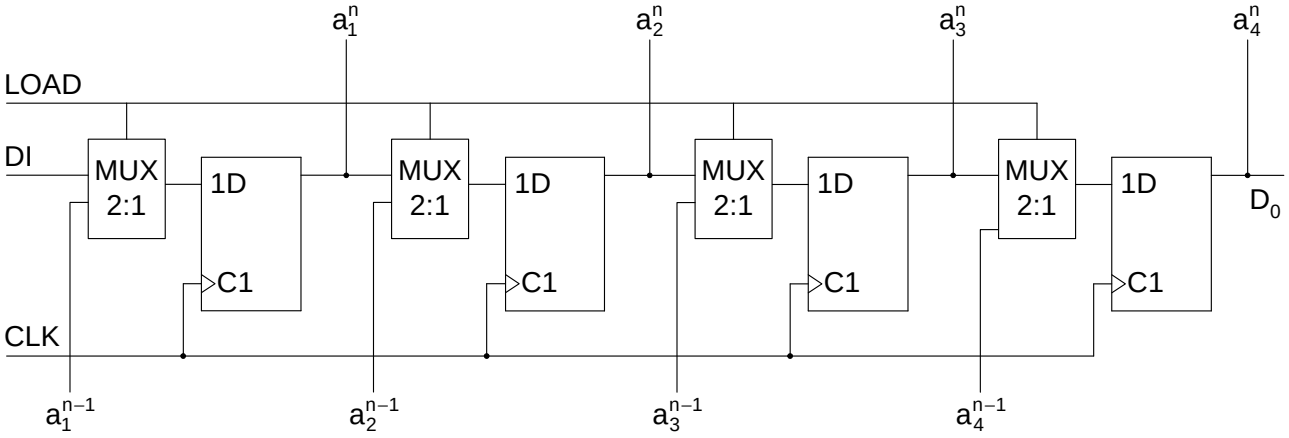
Obwohl dabei die **Taktung meist implizit** angesetzt und das Taktsignal nicht mehr explizit dargestellt wird, wird beim Entwurf auf Registertransfer-Ebene das zeitliche Verhalten **taktgenau festgelegt**!

Die Umsetzung einer Registertransfer-Beschreibung in die logischen Grundfunktionen ist einfach (und automatisiert) möglich, wie wir später noch sehen werden.

5.6.2 Operationen mit einem Register

Beispiel: Ladbares Schieberegister mit paralleler Ein/Ausgabe

Realisierung: D-Flipflops mit Multiplexer



Operation:

Load = 1 → $a_1^n \leftarrow a_1^{n-1}, a_2^n \leftarrow a_2^{n-1}, a_3^n \leftarrow a_3^{n-1}, a_4^n \leftarrow a_4^{n-1}$

Load = 0 → $a_1^n \leftarrow DI, a_2^n \leftarrow a_1^{n-1}, a_3^n \leftarrow a_2^{n-1}, a_4^n \leftarrow a_3^{n-1}$

Register-Transferfunktion: (beim nächsten Takt)

if Load = 1 **then** $a_{1:4}^n \leftarrow a_{1:4}^{n-1}$

if Load = 0 **then** $a_1^n \leftarrow DI, a_{2:4}^n \leftarrow a_{1:3}^{n-1}$

Anwendung z. B. zur Parallel/Serien- und Serien-/Parallelwandlung

Andere Operationen auf einem Register

Operation: **Nullfunktion, Löschen A ← 0**
Register A,
mit a_i Registerelemente von A (i = 1, 2, ..., N)

$\rightarrow a_i = 0$

Umsetzung: D-Flipflop: $D_i = a_i = 0$

JK-MS-Flipflop: $J_{a_i} = 0, \quad K_{a_i} = 1, \quad i = 1 \ldots N$

Operation: **Einsfunktion A ← 1**

Umsetzung: D-Flipflop: $D_N = a_N = 1$
$D_i = a_i = 0, \qquad i = 1 \ldots N\text{-}1$

JK-MS-Flipflop: $J_{a_i} = 1, \quad K_{a_i} = 0, \quad i = N$

$$J_{a_i} = 0, \quad K_{a_i} = 1, \quad i = 1 \ldots N - 1$$

Für andere Konstanten **A ← c** Bitmaske entsprechend setzen.

Operation: **Negation, Invertierung,**
Einerkomplement A ← $\overline{A}$

Umsetzung: D-Flipflop: $D_i = a_i = \overline{Q_i}$

JK-MS-Flipflop: $J_{a_i} = 1, \quad K_{a_i} = 1, \quad i = 1 \ldots N$

oder $J_{a_i} = \overline{Q}_{a_i}, \quad K_{a_i} = Q_{a_i}, \quad i = 1 \ldots N$

Operation: **Shift, Schieben $a_i \leftarrow a_{i-1}$** für Rechtsshift
bzw. **$a_i \leftarrow a_{i+1}$** für Linksshift

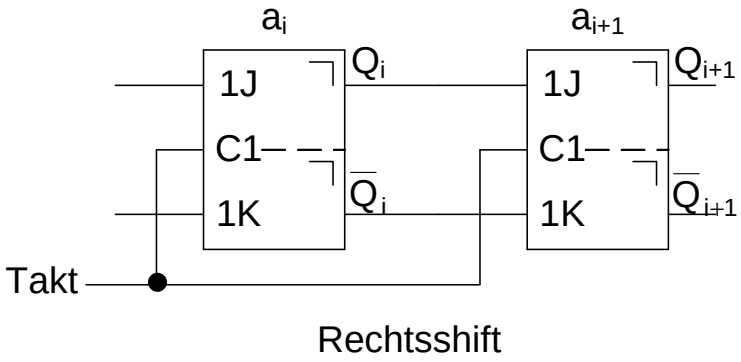
Umsetzung:

D-Flipflop: $D_{ai} = Q_{ai\pm1}$

JK-Flipflops:

$J_{ai} = Q_{a,i+1}$ Linksshift
$_{(-)}$

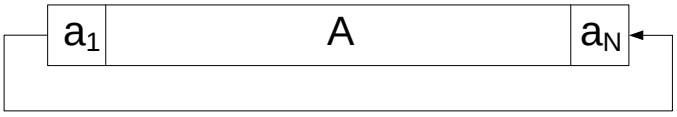
$K_{ai} = \overline{Q}_{a,i+1}$ (Rechtsshift)
$_{(-)}$



Rechtsshift

Operation: **Ringshift**
$a_i \leftarrow a_{i-1}$ mit **$a_N \leftarrow a_1$** für Linksshift
bzw. **$a_i \leftarrow a_{i+1}$** mit **$a_1 \leftarrow a_N$** für Rechtsshift

Umsetzung:



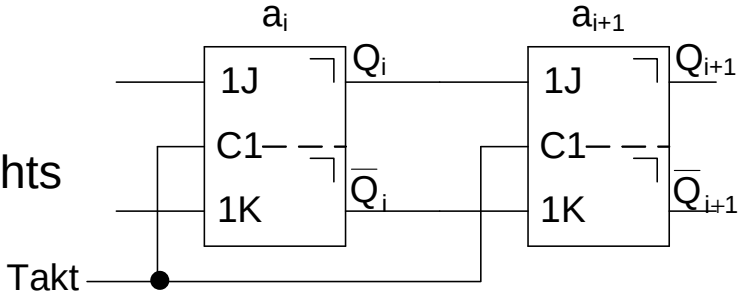
Ringshift (links)

D-Flipflop: $D_{ai} = Q_{ai\pm1}$
und $D_{aN} = Q_{a1}$
bzw. $D_{a1} = Q_{aN1}$

JK-Flipflops:

$J_{aN} = Q_{a1}$ Ringshift links
$K_{aN} = \overline{Q}_{a1}$

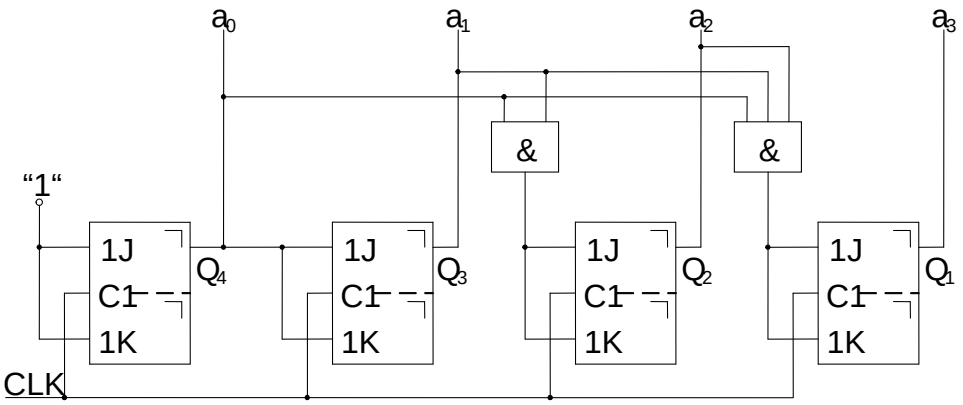
$J_{a1} = Q_{aN}$ Ringshift rechts
$K_{a1} = \overline{Q}_{aN}$



Rechtsshift

Operation: **Zählen A ← A + 1**

Umsetzung: Synchroner Dualzähler
(hier mit JK-Flipflop als T-Flipflop)



Takt- bzw. Zählimpulse

Eingangsfunktion der i-ten Stelle:
(Prinzip:Togglen, wenn alle niederwertigeren Stellen 1 sind)

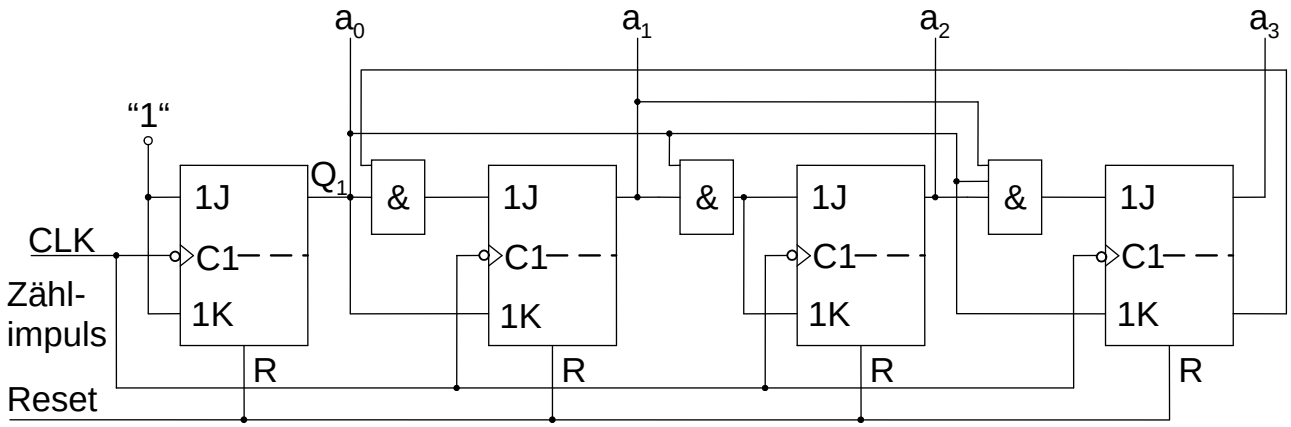
$$J_{a_0} = K_{a_0} = 1$$

$$J_{a_i} = K_{a_i} = Q_{a_0} * Q_{a_1} * \ldots * Q_{a_{i-1}} \qquad (i = 1, 2, \ldots, N-1)$$

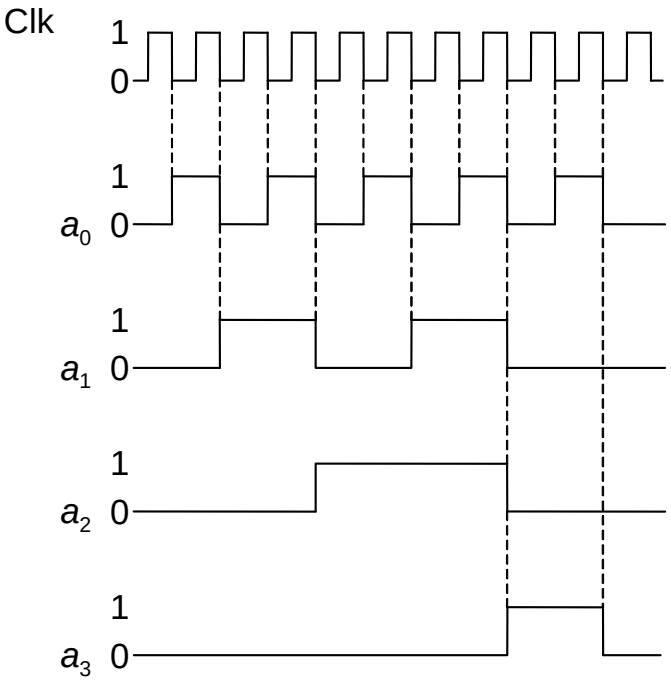
Zählimpulse	a_3	a_2	a_1	a_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
15	1	1	1	1
16	0	0	0	0

Operation: **BCD-Zähler A ← (A + 1) mod 10**

Umsetzung: Synchroner Zähler, hier mit JK-Flipflops und asynchronem Reset



A	a_3	a_2	a_1	a_0
	2^3	2^2	2^1	2^0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0



Weitere Varianten von Zählern mit entsprechender Verschaltung der Eingänge, z. B.:

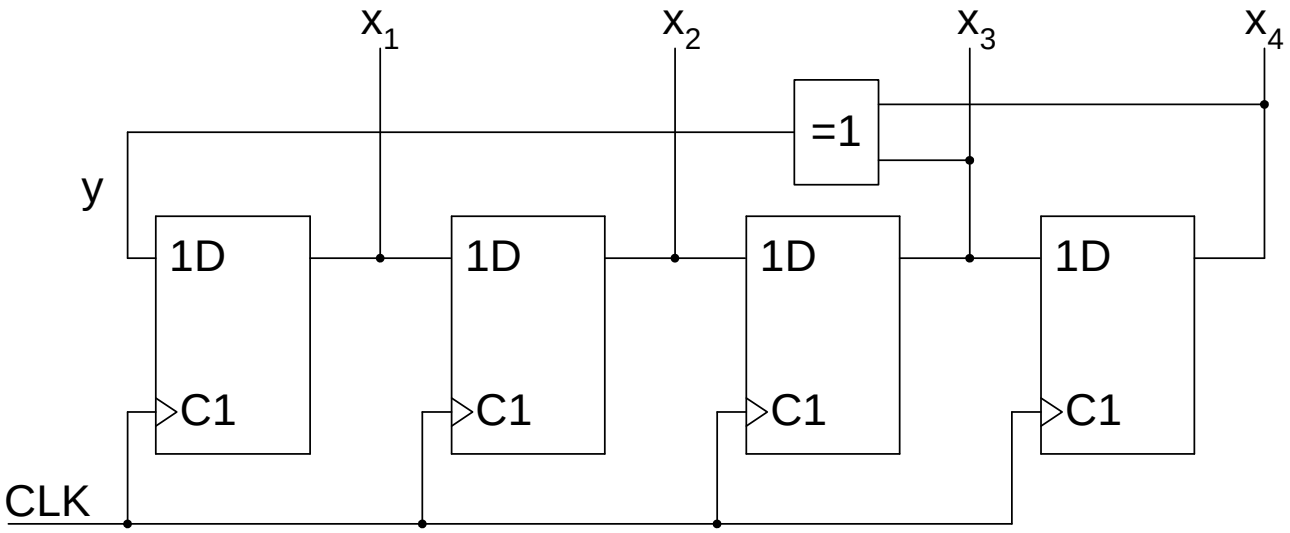
- verschiedene Codes
- Vorwärts-/Rückwärts-Zähler
- Zähler mit parallelen Ladeeingängen

...

Operation: **Generierung von Pseudozufallszahlen**

Umsetzung: Rückgekoppeltes Schieberegister

Beispiel: Pseudo-Zufallszahlengenerator mit n = 4 Bit (Periode $N = 2^n - 1 = 15$)



Zustandstabelle (Anfangswert 0000 unzulässig)

Takt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_1	1	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1
x_2	0	1	0	0	1	1	0	1	0	1	1	1	1	0	0	0
x_3	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0	0
x_4	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0
y	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1	0

Für andere N sind entsprechende Rückkopplungsschaltungen für eine maximale Zyklenlänge berechenbar.

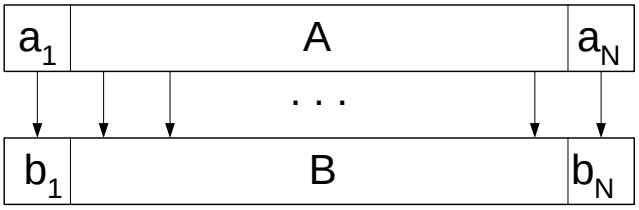
Weitere Anwendungen rückgekoppelter Schieberegister:

Codierer/Decodierer für zyklische Codes, Signatur-analyse usw.

5.6.3 Operationen mit zwei Registern

Datentransport

Paralleltransport B ← A



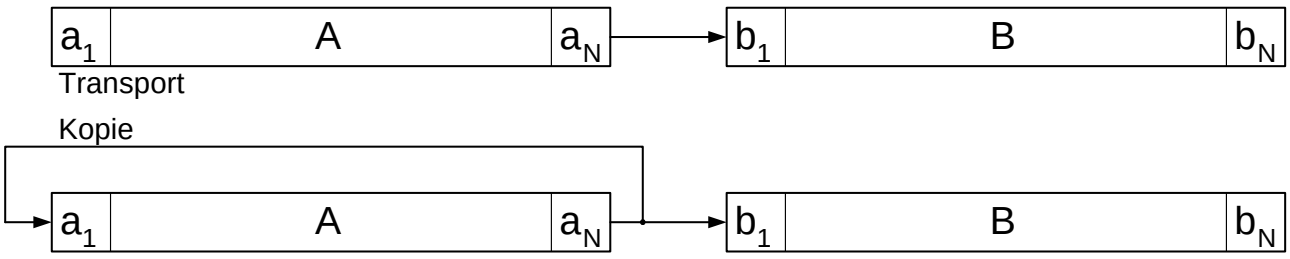
Für D-Flipflops: $D_{bi} = Q_{ai}$

Für JK-Flipflops: $J_{bi} = Q_{ai}$
$K_{bi} = \overline{Q}_{ai}$

➔ **1** Taktimpuls für Datentransport

N Leitungen für Daten
+ Leitungen für Takt und Schreib-/Lesesignale

Serientransport B ← A



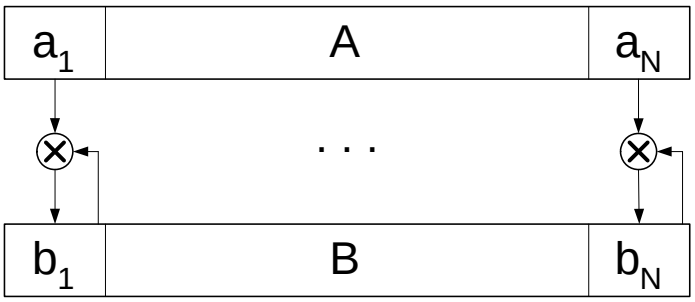
➔ **N** Taktimpulse für Datentransport

1 Leitung für Daten
+ Leitungen für Takt und Schreib-/Lesesignale

Zweistellige Funktionen oder Verknüpfungen

$B \leftarrow A \times B$ oder $A \leftarrow A \times B$

Parallelverknüpfung



N Verarbeitungseinheiten und 1 Takt erforderlich

<u>Beispiele:</u>

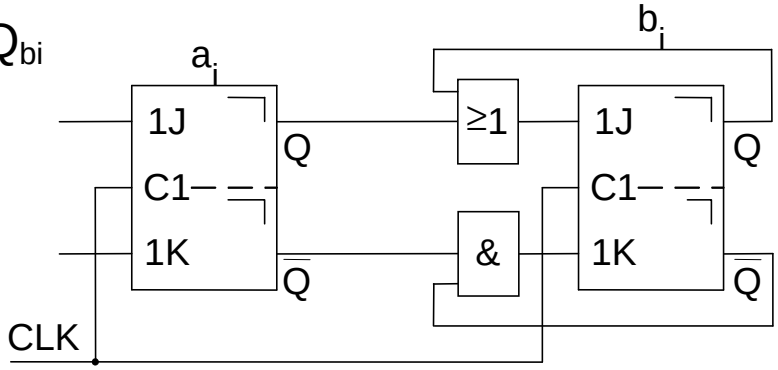
Disjunktion (**ODER**-Verknüpfung): $b_i \leftarrow a_i + b_i$

Für D-Flipflops: $D_{bi} = Q_{ai} + Q_{bi}$

Für JK-Flipflops:

$J_{bi} \leftarrow Q_{ai} + Q_{bi}$

$K_{bi} \leftarrow \overline{Q}_{ai} * \overline{Q}_{bi}$



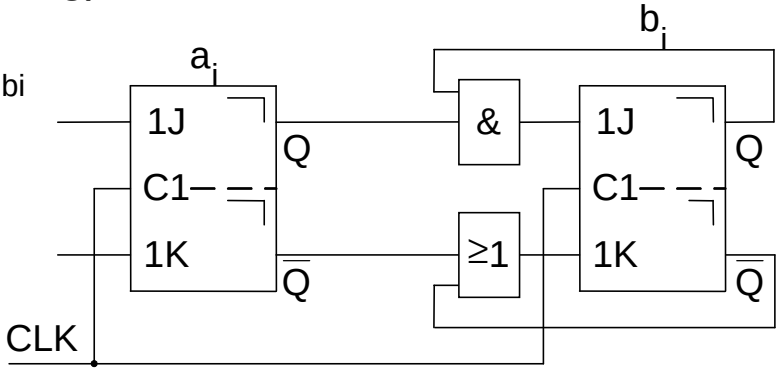
Konjunktion (**UND**-Verknüpfung): $b_i \leftarrow a_i * b_i$

Für D-Flipflop: $D_{bi} = Q_{ai} * Q_{bi}$

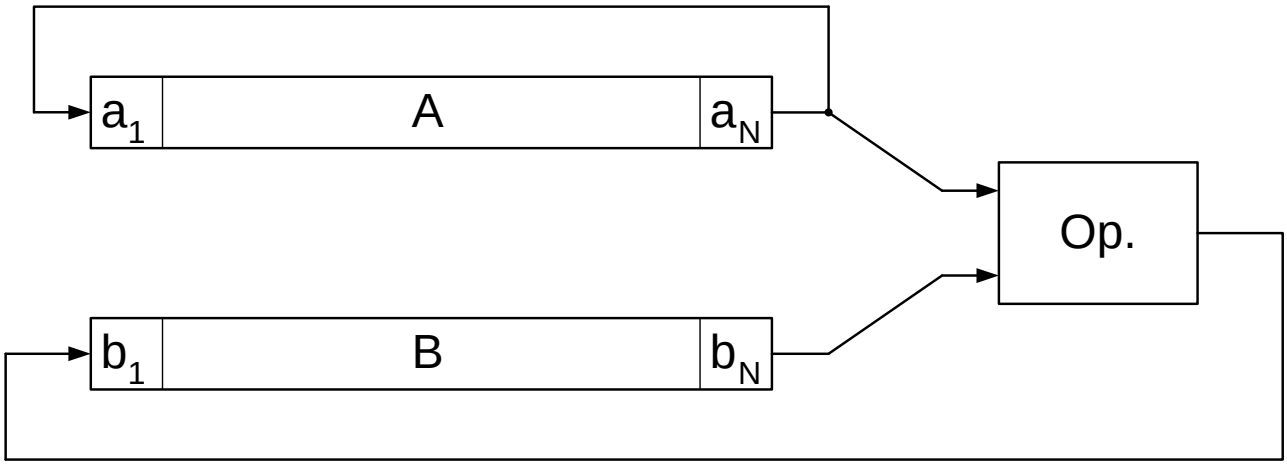
Für JK-Flipflop:

$J_{bi} \leftarrow Q_{ai} * Q_{bi}$

$K_{bi} \leftarrow \overline{Q}_{ai} + \overline{Q}_{bi}$



Serienverknüpfung



N Takte erforderlich, aber

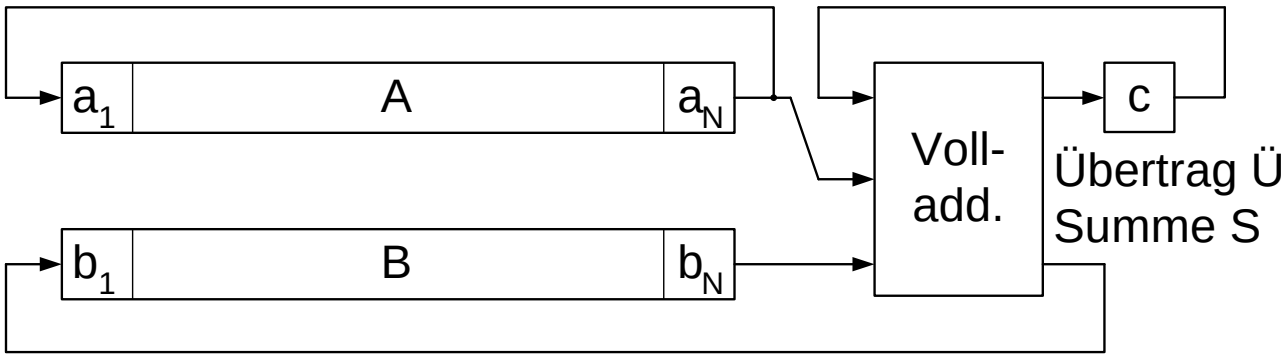
nur 1 Verarbeitungseinheit

5.6.4 Operationen mit drei und mehr Registern

Serielle Verarbeitung

z. B. $C \leftarrow A \times B$
$A \leftarrow A \times B \times C$
$B \leftarrow A \times B \times C$

Beispiel: Serienaddierwerk für $B = A + B + c_0$



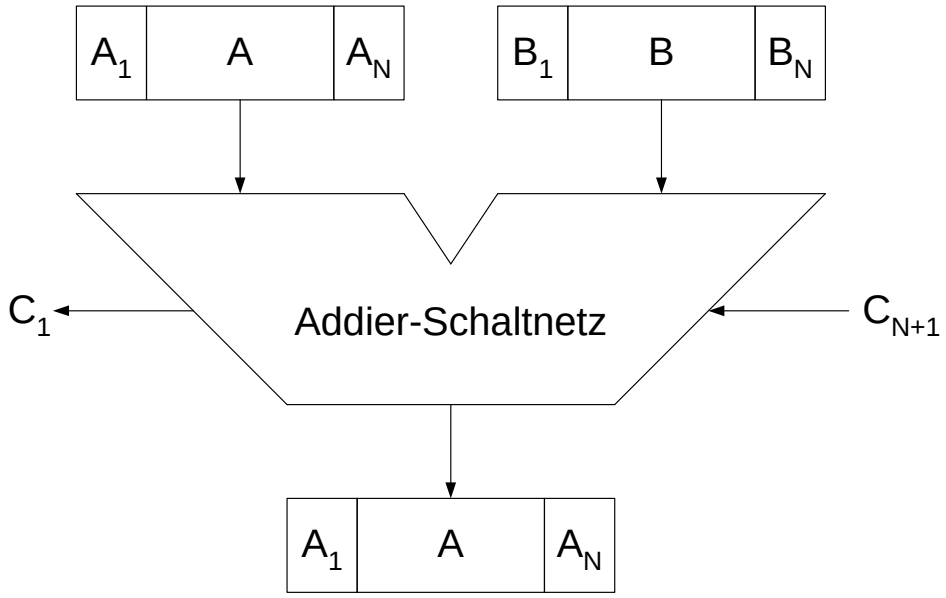
Der Volladdierer berechnet zur i-ten Taktzeit (i = 0,1, ..., N-1)

$$S = a_{N-i} \oplus b_{N-i} \oplus c_{N-i+1}$$

$$Ü = a_{N-i}\, b_{N-i} + a_{N-i}\, c_{N-i+1} + b_{N-i}\, c_{N-i+1}$$

Komplexere Operationen

Beispiel: Asynchrones Paralleladdierwerk



‘Asynchron‘ heißt hier, dass der Übertrag vollständig im Schaltnetz behandelt wird. (Die Register können durchaus synchron arbeiten.) Realisierung z. B. als Addierer mit durchlaufendem Übertrag (Ripple-Carry) oder parallelem Übertrag (Carry-Lookahead).

Es sind auch synchrone Addierwerke bekannt, bei denen der Übertrag taktsynchron abgebaut wird (Carry-Save-Adder).

Vielfach werden auch Verknüpfungsschaltnetze eingesetzt, die unterschiedliche Funktionen realisieren können. Steuerleitungen kontrollieren dann das aktuelle Verhalten ebenso wie das Speichern in Registern.

In Mikroprozessoren erfolgt beispielsweise die Verarbeitung von Daten meist in ALUs (Arithmetic Logical Units), die eine Kombination mehrerer arithmetisch/logischer Funktionen (z. B. ADD, SUB, AND, OR, ...) enthalten, ganz analog.

Datenquelle und Datensenke sind auch hier jeweils Register.

5.7 Datentransfer zwischen Registern

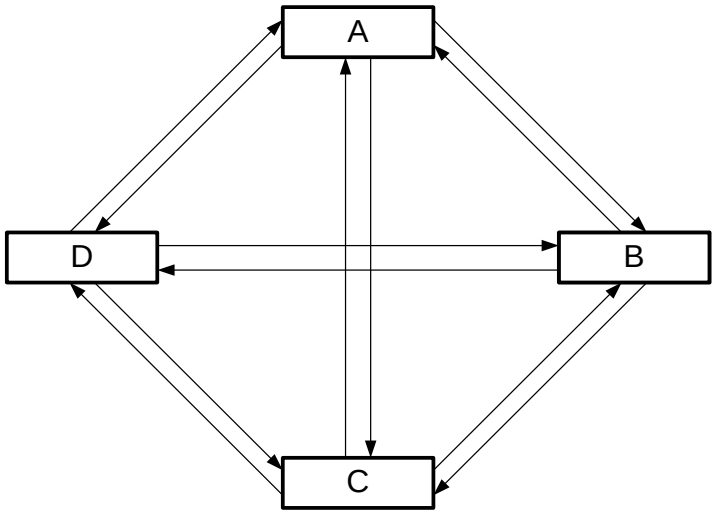
In komplexeren digitalen Systemen transferieren Busse N-Bit Daten (Vektoren) zwischen mehreren Registern. Sie machen selbst keine logischen Verknüpfungen.

Der Transfer von Daten kann je nach Hardwarerealisierung uni- oder bidirektional sein.

Typische Busstrukturen

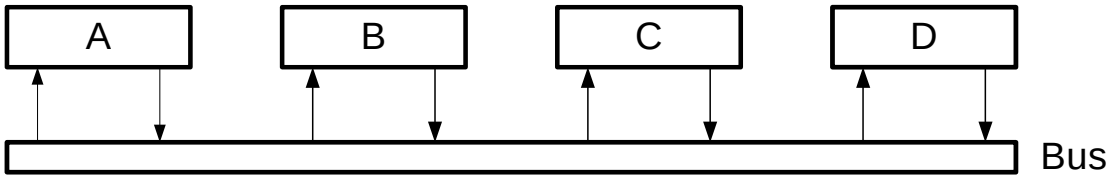
Dedizierte Busse (Punkt-zu-Punkt Verbindungen):

- Aufwand k·(k-1) bei k Einheiten
- schnell (große Bandbreite)



Gemeinsamer Bus (common shared bus):

- nur ein Transfer pro Zeiteinheit
- geringer Verdrahtungsaufwand
- Kontrolllogik erforderlich



Buszugriff

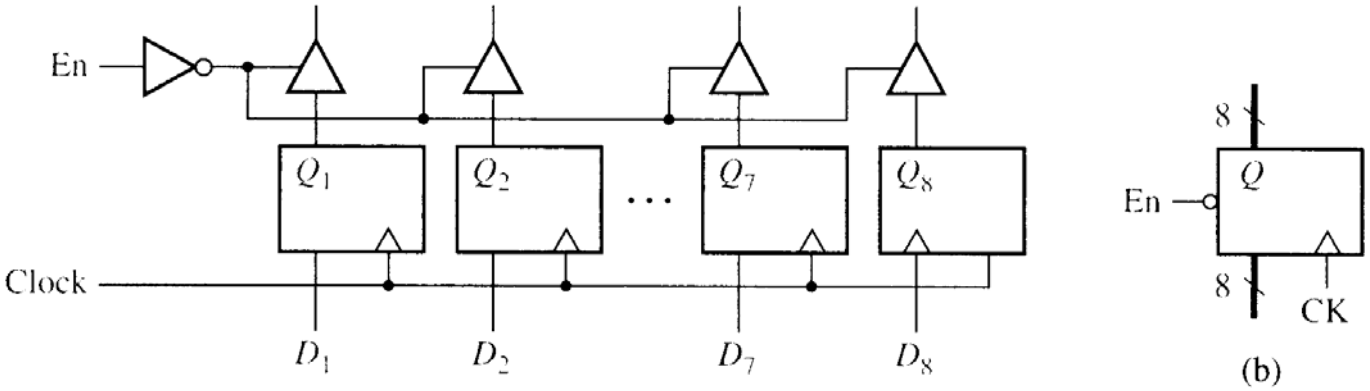
Oft verbinden Busse mehrere Komponenten so mit einander, dass mehrere Komponenten lesend und/oder (alternativ) schreibend auf den Bus zugreifen können.

Dabei darf immer **nur eine** Komponente schreibend auf den Bus zugreifen, beliebig viele lesend.
Das ermöglicht Broadcast und Multicast.

Für den konfliktfreien Zugriff hat die Kontrolllogik des Busses (Steuerwerk) zu sorgen.

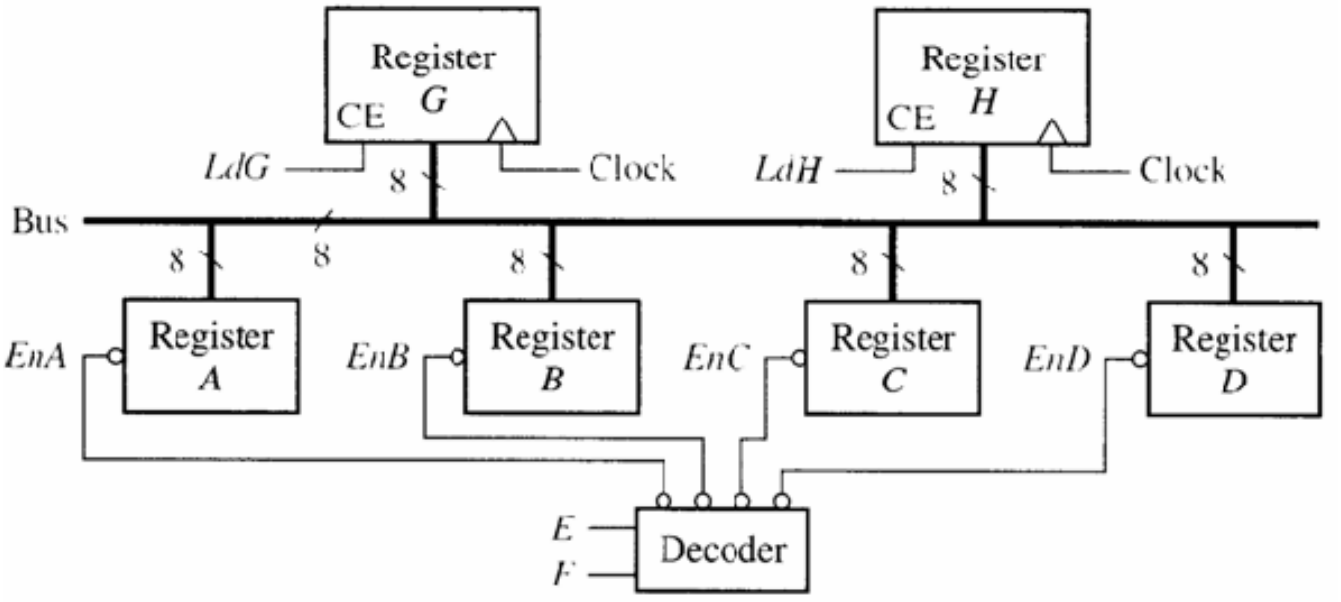
Damit mehrere Register auf gemeinsamen Datenleitungen arbeiten können, kann der Ausgang eines jeden Registers durch ein Freigabesignal (*enable*) aktiv oder hochohmig (*tristate*) geschaltet werden.

➔ *Tristate-Busse*



Der Datentransfer (hier 8 Bit) zwischen den Registern wird durch die Auswahl (*Adressierung*) des schreibenden Registers und des lesenden Registers über Freigabesignale (*enable*) für das Schreiben auf den Bus bzw. die Datenübernahme vom Bus gesteuert.

Anwendungsbeispiel:



Auf funktionaler Ebene sind alle Daten- und Steuerleitungen, oft auch die dazu gehörige Steuerlogik, zu einem "Bus" (z.B. *Datenbus*, *Adressbus* von Prozessoren) zusammengefasst und stehen allen Komponenten zum Datentransport zur Verfügung. Sie treten beim Entwurf auf Registertransfer-Ebene nicht (mehr) explizit in Erscheinung.

Folglich werden beim Entwurf Baugruppen unterschieden zum

- Verarbeiten (Schaltnetze) ,
- Speichern (Register, Speicher),
- Transport (Busse)

von Daten und eine übergeordnete *Kontrolleinheit* zur Koordinierung des Ganzen.