GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Informatik II

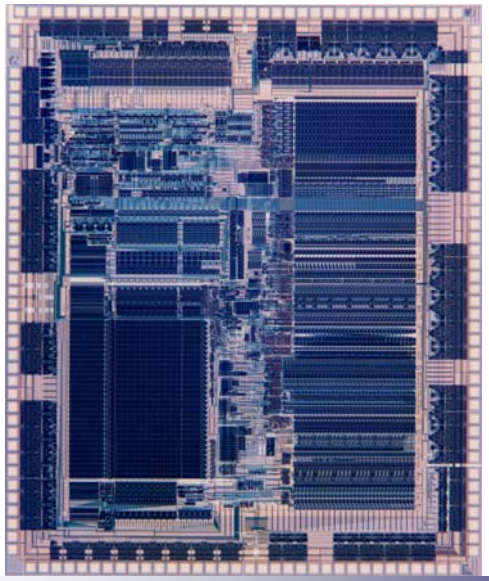
SS 2005

Kapitel 3: Rechnerarchitektur

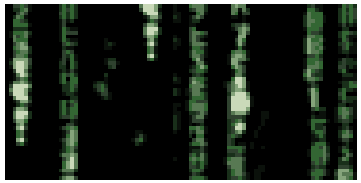
Teil 1: Schaltwerke

Dr. Michael Ebner
Dipl.-Inf. René Soltwisch

Lehrstuhl für Telematik
Institut für Informatik



ZIEL DER VORLESUNG:
Verstehen lernen, was in einem Rechner vorgeht. Wie er aufgebaut ist und wie die einzelnen Bits und Bytes den Rechner steuern….

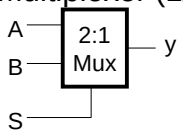


```
0000 0001 0000 1010 0101 0000 0010 0000
0000 0000 0001 0000 1010 0101 0101 0110
0000 0010 0000 0010 0010 0100 1010 0000
```


Kombinatorische Schaltelemente (Schaltnetze)

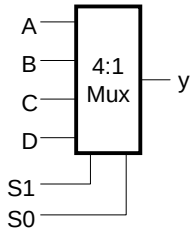
- Spezielle Verknüpfungselemente für Computer
  - Signalauswahl:
    - Multiplexer
    - Demultiplexer
  - Registerauswahl:
    - Dekoder
  - Werte vergleichen
    - Vergleicher
  - Addieren
    - Addierer
  - Arithmetisch-Logische Einheit (ALU)

Multiplexer (1/2)



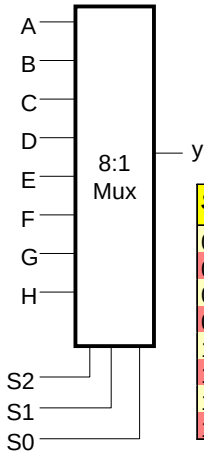
S	y
0	A
1	B

$\overline{S}A \vee SB$



S1,S0	y
0, 0	A
0, 1	B
1, 0	C
1, 1	D

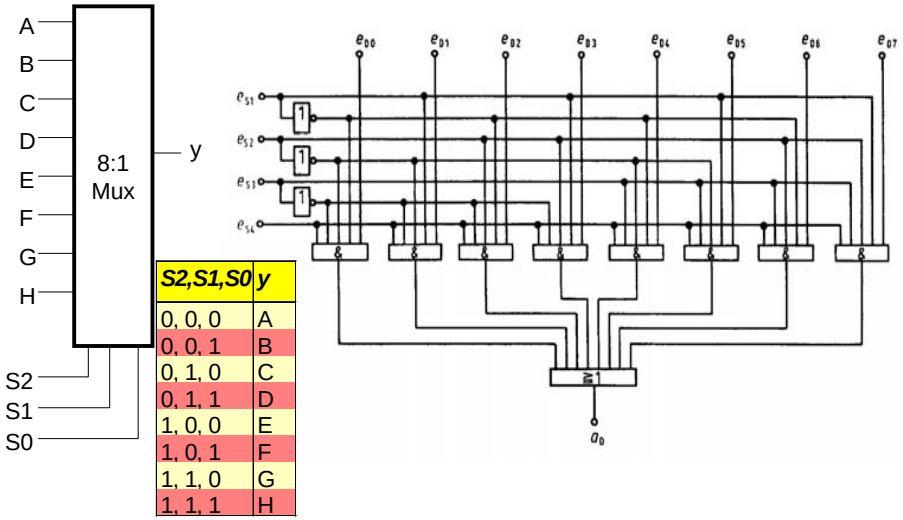
$(\overline{S1}\ \overline{S0}\ A) \vee$
$(\overline{S1}\ S0\ B) \vee$
$(S1\ \overline{S0}\ C) \vee$
$(S1\ S0\ D)$



S2,S1,S0	y
0, 0, 0	A
0, 0, 1	B
0, 1, 0	C
0, 1, 1	D
1, 0, 0	E
1, 0, 1	F
1, 1, 0	G
1, 1, 1	H

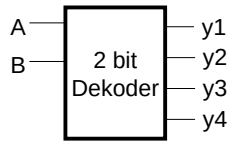
S_i .. Selektionseingänge

Multiplexer (2/2)

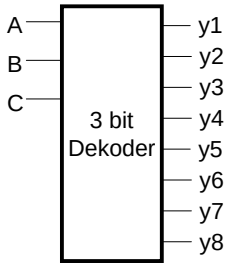


S2,S1,S0	y
0, 0, 0	A
0, 0, 1	B
0, 1, 0	C
0, 1, 1	D
1, 0, 0	E
1, 0, 1	F
1, 1, 0	G
1, 1, 1	H

Dekoder/Demultiplexer

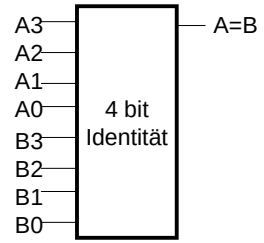


B, A	y4,y3,y2,y1
0, 0	0, 0, 0, 1
0, 1	0, 0, 1, 0
1, 0	0, 1, 0, 0
1, 1	1, 0, 0, 0

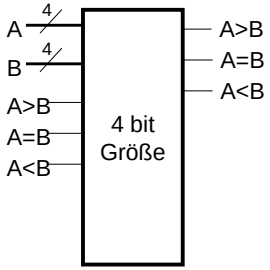


C,B,A	y8,y7,y6,y5,y4,y3,y2,y1
0, 0, 0	0, 0, 0, 0, 0, 0, 0, 1
0, 0, 1	0, 0, 0, 0, 0, 0, 1, 0
0, 1, 0	0, 0, 0, 0, 0, 1, 0, 0
0, 1, 1	0, 0, 0, 0, 1, 0, 0, 0
1, 0, 0	0, 0, 0, 1, 0, 0, 0, 0
1, 0, 1	0, 0, 1, 0, 0, 0, 0, 0
1, 1, 0	0, 1, 0, 0, 0, 0, 0, 0
1, 1, 1	1, 0, 0, 0, 0, 0, 0, 0

Vergleicher



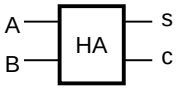
B, A	A=B
B ≠ A	0
B = A	1



A3,B3, A2,B2, A1,B1, A0,B0,	A>B,A=B,A<B	A>B,A=B,A<B
> - - - - - - -	1,	0, 0
< - - - - - - -	0,	0, 1
= > - - - - - -	1,	0, 0
= < - - - - - -	0,	0, 1
= = > - - - - -	1,	0, 0
= = < - - - - -	0,	0, 1
= = = > - - - -	1,	0, 0
= = = < - - - -	0,	0, 1
= = = = - 1 -	0,	1, 0
= = = = 1 0 0	1,	0, 0
= = = = 0 0 1	0,	0, 1

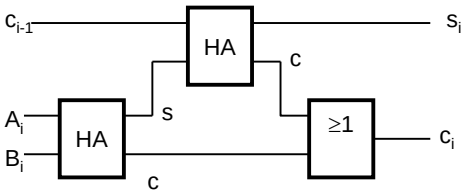
Addierer (1/4)

Halbaddierer

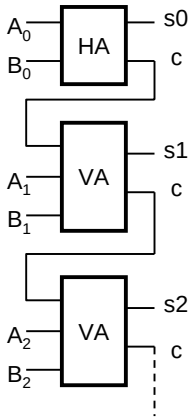


B, A	*s, c*
0, 0	0, 0
0, 1	1, 0
1, 0	1, 0
1, 1	0, 1

Volladdierer



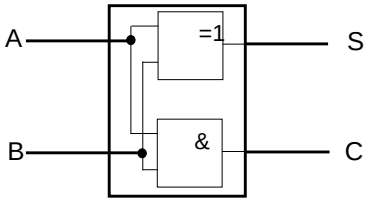
Mehrstelliger Addierer



Addierer (2/4)

- Einstelliger Addierer für Addition zweier Binärzahlen A und B, Ergebnis Summe S und Übertrag (Carry) C

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Mehrstellige Additionsaufgabe

Beispiel: 5 + 7
(5) 0101
(7) 0111
.. 111 <- Übertrag (Carry)
1100

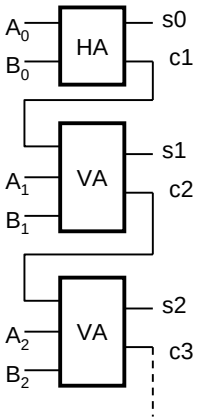
Allgemein:

1. Summand	A_4	A_3	A_2	A_1	A_0	
2. Summand	B_4	B_3	B_2	B_1	B_0	
Übertrag	C_5	C_4	C_3	C_2	C_1	0
Summe	S_5	S_4	S_3	S_2	S_1	S_0

$S_i = (A_i + B_i) + C_i$ (+ = Addition)
$C_{i+1} = C_i A_i \vee C_i A_i \vee A_i B_i$

Addierer (3/4)

„Serienaddierer“: ökonomisch, aber langsam



Beschleunigung mittels
Parallelisierung durch Carry-Vorausberechnung:
„carry look-ahead“

G = generate carry
(erzeuge Übertrag

C = propagate carry
(leite Übertrag weiter)

$C_{n+1} = \underbrace{(B_n A_n)}_{G_n} \vee \underbrace{(C_n(B_n \leftrightarrow A_n))}_{P_n}$

$C_{n+1} = G_n \vee C_n P_n$

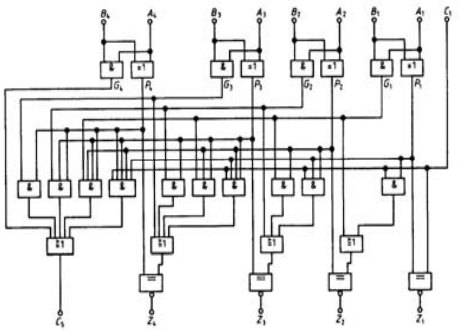
$C_2 = G_1 \vee C_1 P_1$
$C_3 = G_2 \vee C_2 P_2 = G_2 \vee (G_1 \vee C_1 P_1) P_2$
$= G_2 \vee G_1 P_2 \vee C_1 P_1 P_2$
$C_4 = G_3 \vee G_2 P_3 \vee G_1 P_2 P_3 \vee C_1 P_1 P_2 P_3$
$C_5 = G_4 \vee G_3 P_4 \vee G_2 P_3 P_4 \vee G_1 P_2 P_3 P_3 \vee C_1 P_1 P_2 P_3 P_4$

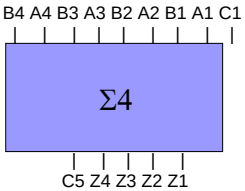
Addierer (4/4)

4 bit Paralleladdierer mit „carry look-ahead“ (paralleler Übertrag)

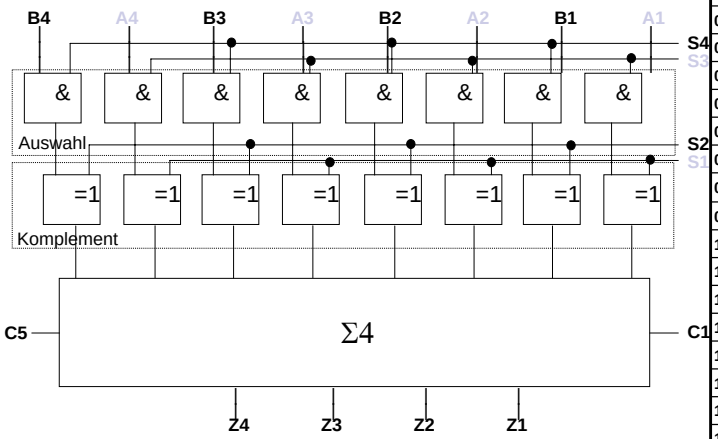
$C_2 = G_1 \vee C_1 P_1$
$C_3 = G_2 \vee C_2 P_2 = G_2 \vee (G_1 \vee C_1 P_1) P_2$
$= G_2 \vee G_1 P_2 \vee C_1 P_1 P_2$
$C_4 = G_3 \vee G_2 P_3 \vee G_1 P_2 P_3 \vee C_1 P_1 P_2 P_3$
$C_5 = G_4 \vee G_3 P_4 \vee G_2 P_3 P_4 \vee G_1 P_2 P_3 P_3 \vee C_1 P_1 P_2 P_3 P_4$

$C_{n+1} = \underbrace{(B_n A_n)}_{G_n} \vee \underbrace{(C_n(B_n \leftrightarrow A_n))}_{P_n}$





Addierer-Subtrahierer

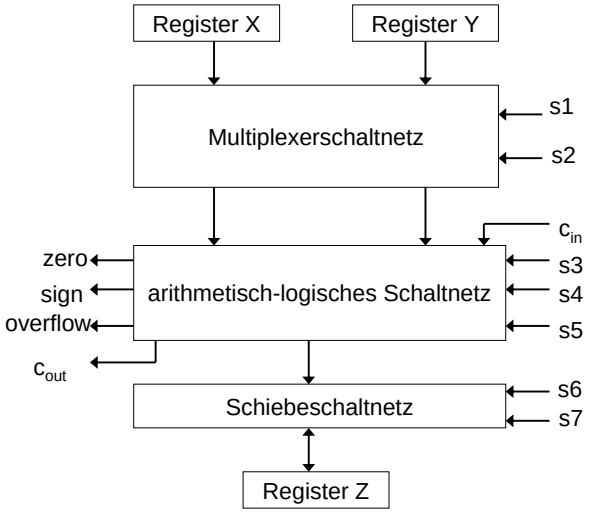


S4	S3	S2	S1	Z
0	0	0	0	C1
0	0	0	1	C1-1
0	0	1	0	C1-1
0	0	1	1	C1-2
0	1	0	0	C1+A
0	1	0	1	C1-A-1
0	1	1	0	C1+A-1
0	1	1	1	C1-A-2
1	0	0	0	C1+B
1	0	0	1	C1+B-1
1	0	1	0	C1-B-1
1	0	1	1	C1-B-2
1	1	0	0	C1+A+B
1	1	0	1	C1+B-A-1
1	1	1	0	C1+A-B-1
1	1	1	1	C1-A-B-2

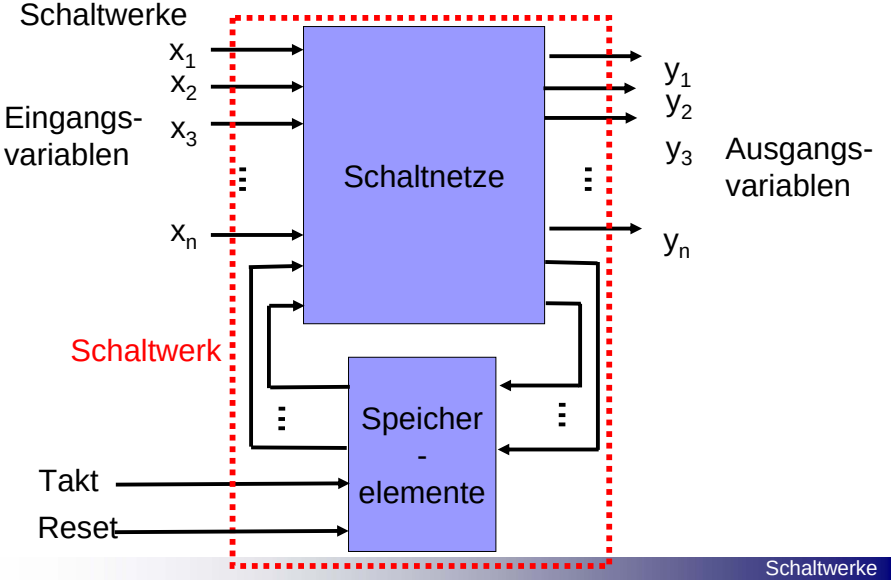
Arithmetisch-Logische Einheit (ALU) (1/2)

- funktionaler Kern eines Digitalrechners
- führt arithmetische und logische Operationen aus
- Eingabe: Daten und Steuersignale
- Ausgabe: Ergebnis und Statussignale
- Meist nur für Festkommazahlen (Gleitkomma wird oft von einem speziellen Arithmetik-Coprozessor ausgeführt oder in Festkommazahlen unterteilt)

Arithmetisch-Logische Einheit (ALU) (2/2)

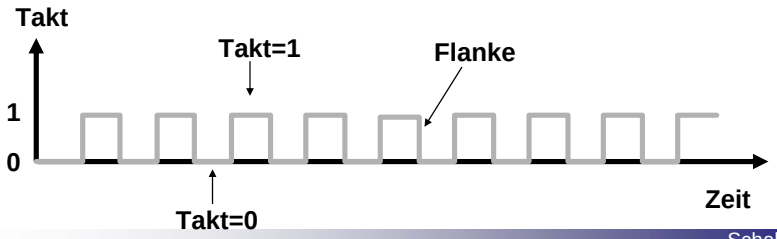


Schaltwerke



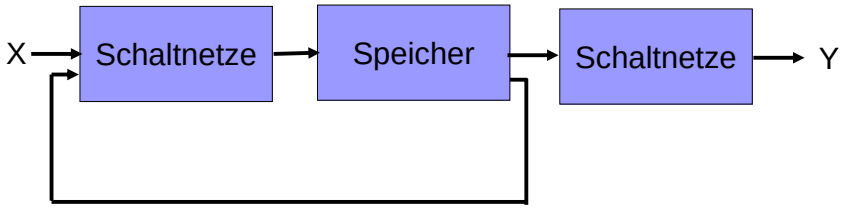
Takt

- Schaltwerke befinden sich zu gegebener Zeit in einem **Zustand** Q. Ein solcher Zustand ist beispielsweise durch die Werte der Ausgangsleitungen zu diesem Zeitpunkt charakterisiert.
- Zustände ändern sich nur zu bestimmten diskreten Zeitpunkten, bei Eintreffen eines Taktes, das heißt, wenn eine 1 auf der Taktleitung anliegt oder bei Pegeländerung auf der Taktleitung, d.h. bei positiver oder negativer Flanke.



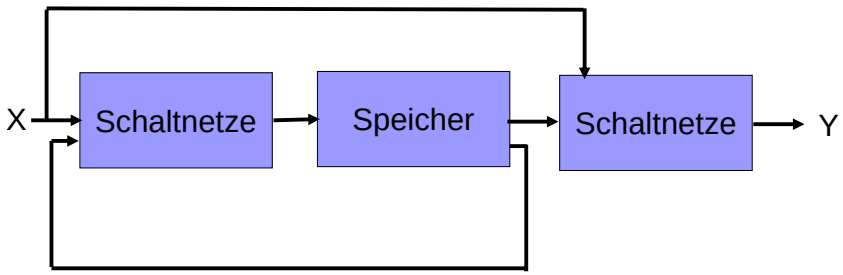
Moore- oder Zustandsautomat

- Ausgangsbelegungen sind nur vom Zustand des Schaltwerkes abhängig



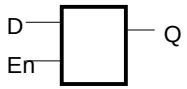
Mealy- oder Übergangsautomat

- Ausgangsbelegungen sind vom Zustand und vom Wert der Eingangsbelegungen abhängig



Speicherelemente

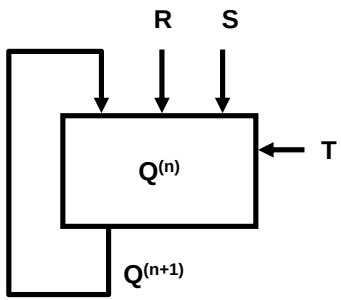
- Binäre Speicherelemente:
  - kleinste logische Bausteine zur Aufbewahrung von Information.
  - Speicherung durch Einnahme der Zustände “0” oder “1”.
  - Realisierung als Schaltwerke.



RS-Flipflop

Ein RS-Flipflop besitzt zwei besondere Eingänge zum
- **Setzen** (auf “1”; Set) und
- **Rücksetzen** (auf “0”; Reset)

des Inhalts.



Takt T
T=0: Zustand ändert sich nicht,
T=1: Zustand kann sich ändern.
Dann wird das RS-Flipflop durch folgende Tabelle beschrieben:

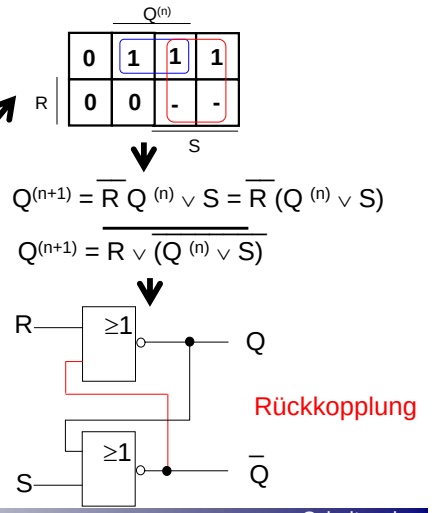
S	R	$Q^{(n+1)}$
0	0	$Q^{(n)}$
0	1	0
1	0	1
1	1	nicht def.

Realisierung eines RS-Flipflop

Funktionstabelle der Schaltfunktion für neuen Zustand $Q^{(n+1)}$ aus aktuellem Zustand $Q^{(n)}$

S	R	$Q^{(n)}$	$Q^{(n+1)}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undef.
1	1	1	undef.

Nebenbedingung: $r \wedge s = 0$



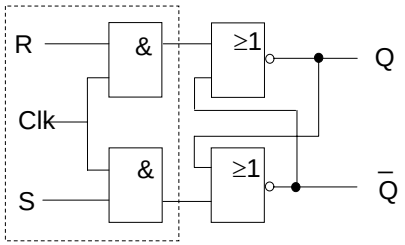
Erweiterung zum RS-Latch

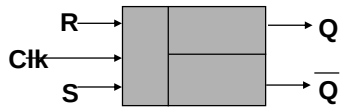
Funktionstabelle der Schaltfunktion für neuen Zustand $Q^{(n+1)}$ aus aktuellem Zustand $Q^{(n)}$

S	R	$Q^{(n)}$	$Q^{(n+1)}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undef.
1	1	1	undef.

Takt (Pegelsteuerung), „Clock“: Clk
Clk=0: Zustand ändert sich nicht,
Clk=1: Zustand kann sich ändern.

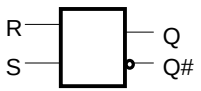
=> Ver**UND**ung von S und R mit Clk





RS-Flipflop

Symbol

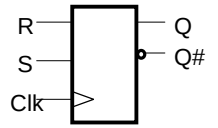


Tabelle

R S Q	Q
0 0 0	0
0 0 1	1
1 0 -	0
0 1 -	1
1 1 -	verb.

RS-Latch

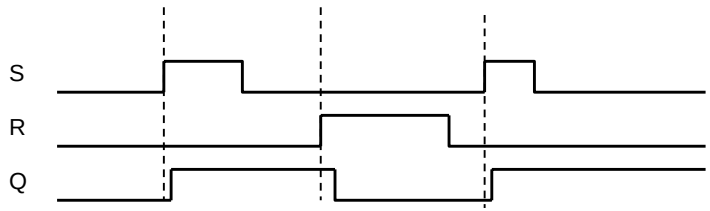
Symbol



Tabelle

Clk R S	Q
0 - -	Q
1 - -	Q
↑ 0 0	Q
↑ 1 0	0
↑ 0 1	1

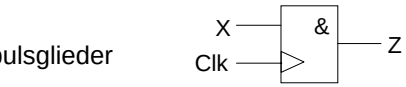
Zeitdiagramm für das RS Latch

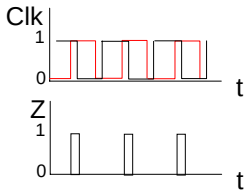


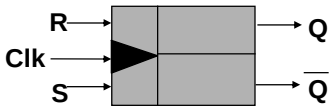
Taktflankensteuerung

Wir haben bisher ein taktpegelgesteuertes RS-NOR-Flipflop.

Taktflankensteuerung 2:
Ersetzen der UND-Glieder durch Impulsglieder
Positive Flankentriggerung

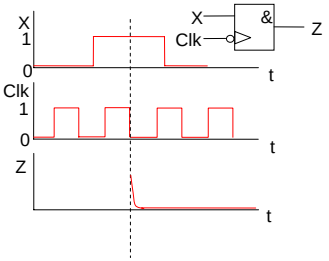




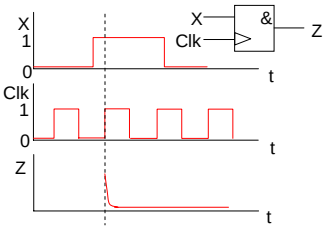


Taktflankengesteuerte Flipflops

Negative Flankentriggerung



Positive Flankentriggerung

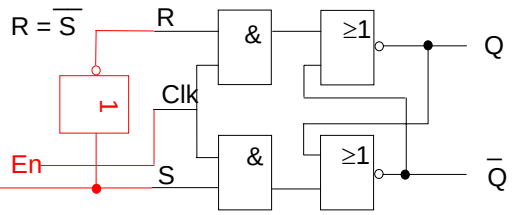


D-Latch und D-Flipflop

Häufig ist es wünschenswert, dass statt Set und Reset nur ein Eingangssignal benutzt wird.
Der Wert des Eingangssignals wird dann beim entsprechendem Enable- oder Taktsignal übernommen.

Eingangsseitige Erweiterungsbeschaltung von R und S.

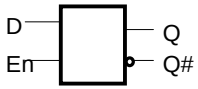
S	R	$Q^{(n)}$	$Q^{(n+1)}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undef.
1	1	1	undef.



D: „Vorbereitungseingang“

D-Latch

Symbol

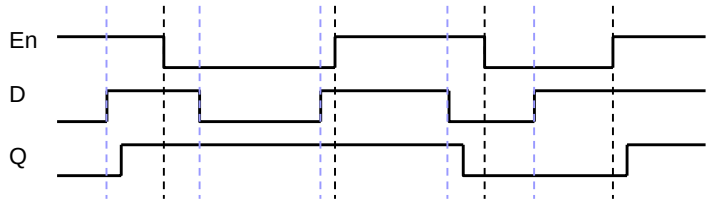


Tabelle

En	*Q*
0	Q
1	D

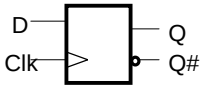
D = Delay

Zeitdiagramm



D-Flipflop -- Flankengesteuert

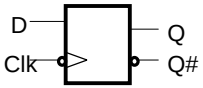
Symbol



Tabelle

Clk	Q
0	Q
1	Q
↑	D

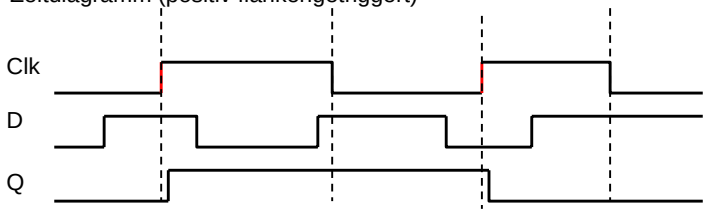
Symbol



Tabelle

Clk	Q
0	Q
1	Q
↓	D

Zeitdiagramm (positiv flankengetriggert)

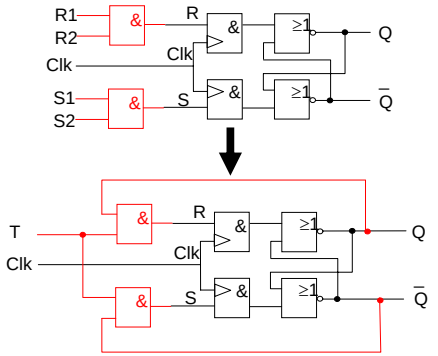


Meistgebrauchtes Flipflop: Basis-Flipflop für alle Register in Prozessoren und in Controllern

T-Flipflop (als Vorstufe zu JK Flipflop)

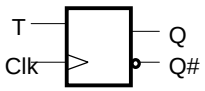
Soll bei jeder Taktflanke den Zustand wechseln, wenn an einem Steuereingang der Signalpegel 1 anliegt, sonst nicht („gesperrt“).

Erweiterung des RS-Flipflop



T-Flipflop

Symbol

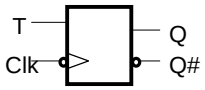


T = Toggle

Tabelle

Clk	T	Q	Q
0	-	-	O
1	-	-	O
↑	$\bar{0}$	-	O
↑	1	0	1
↑	1	1	0

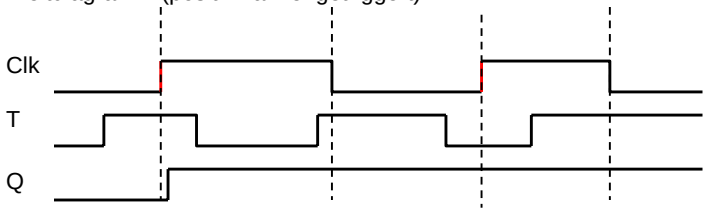
Symbol



Tabelle

Clk	T	Q	Q
0	-	-	O
1	-	-	O
↓	$\bar{0}$	-	O
↓	1	0	1
↓	1	1	0

Zeitdiagramm (positiv flankengetriggert)

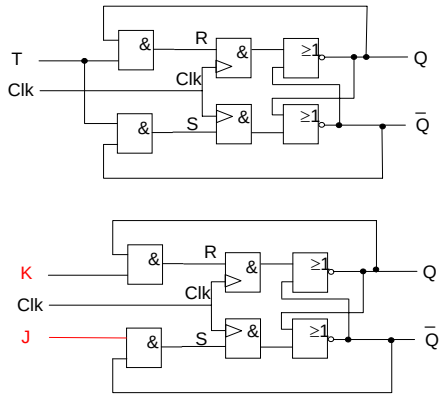


Hauptanwendung: Zählerschaltungen

JK-Flipflop (1/2)

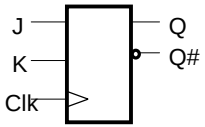
Möglichst vielseitiges Flipflop:
Speicherfall, Setzfall, Rücksetzfall, Kippfall: Auswahl durch 2 Steuerleitungen (J=Jump, K=Kill)

Modifikation des T-Flipflops



JK-Flipflop (2/2)

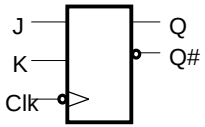
Symbol



Tabelle

Clk	J	K	Q
0	-	-	Q
1	-	-	Q
↑	0	0	Q
↑	1	0	1
↑	0	1	0
↑	1	1	/Q

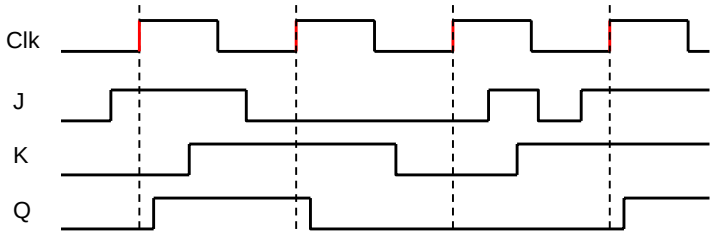
Symbol



Tabelle

Clk	J	K	Q
0	-	-	Q
1	-	-	Q
↓	0	0	Q
↓	1	0	1
↓	0	1	0
↓	1	1	/Q

Zeitdiagramm

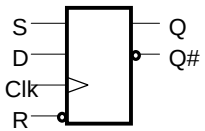


Bevorrechtigte Eingänge

- Oft möchte man Flipflops gezielt auf einen Wert setzen
  - Beispiele: Rücksetzen einer Schaltung beim Einschalten oder "Reset Taster"
- Gezieltes Laden eines Zählers auf einen Wert
  - Daher besondere Eingänge eines Flipflops zum Setzen / Rücksetzen
    - müssen nicht beide vorhanden sein
    - Reset häufiger als Set.
- Sondereingänge können direkt wirken
  - man nennt das asynchron
  - Vorteil: Wirkt immer, auch ohne Takt
  - Nachteil: Auch kurze Impulse können Reset auslösen (gefährlich!)
  - Also: ok für General-Reset, nicht für logische Funktion
- Gegensatz: Synchrone Eingänge wirken mit der Taktflanke
  - Auch für Logikfunktionen nutzbar
  - Aber: Taktgenerator muss laufen, sonst funktioniert z.B. Reset nicht.

Asynchrone bevorrechtigte Eingänge

Symbol: D-Flipflop mit Setz- und Rücksetzeingang

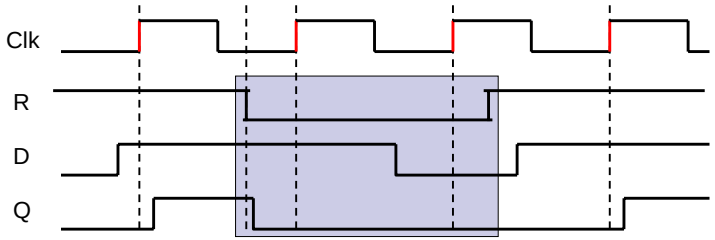


Tabelle

R	S	Clk	D	Q
0	-	-	-	0
1	1	-	-	1
1	0	0	-	O
1	0	1	-	O
1	0	↑	0	0
1	0	↑	1	1

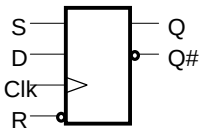
R und S wirken auch ohne Takt.
R ist bevorrechtigt.

Zeitdiagramm



Synchrone bevorrechtigte Eingänge

Symbol: D-Flipflop mit Setz- und Rücksetzeingang

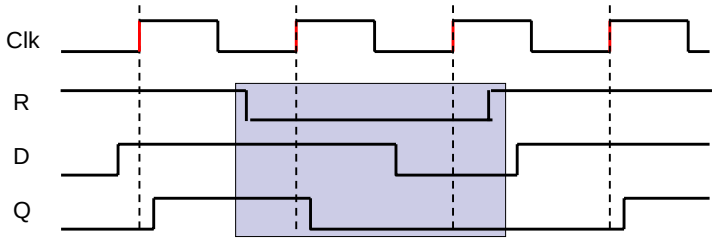


Tabelle

Clk	R	S	D	Q
0	-	-	-	Q
1	-	-	-	Q
↑	0	-	-	0
↑	1	1	-	1
↑	1	0	0	0
↑	1	0	1	1

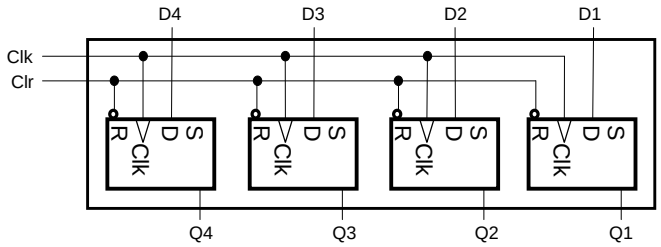
R und S wirken **nicht** ohne Takt.

Zeitdiagramm



Das Register

- Ein Register ist
  - eine Aneinanderreihung von D-Flipflops, z.B. 8, oder 16 oder 32 Stück ("Breite" des Registers)
  - Der Takt ist allen FFs gemeinsam



Das Register: Anwendung

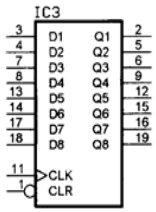
- Anwendung: Standardspeicher in Prozessoren
  - Für den Benutzer sichtbare Register
  - Viele Register für Zwischenwerte
- Ein Register hat oft bevorrechtigte Eingänge
  - Wie bei den Einzelflipflops, z.B. Set oder Reset (synchron oder asynchron)
  - Ein Eingang wirkt auf alle Flipflops des Registers
- Typischer bevorrechtigter Eingang: Clock Enable
  - CE muß 1 sein, damit Register synchrone Operation ausführt
  - Effekt: Register behält den alten Wert bei, "hält seinen Zustand„
  - Typische Anwendung zur Auswahl eines bestimmten Registers

Beispiel Register

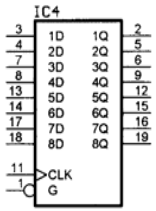
Tabelle

Clk	*Q*
0	Q_i
1	Q_i
↑	D_i

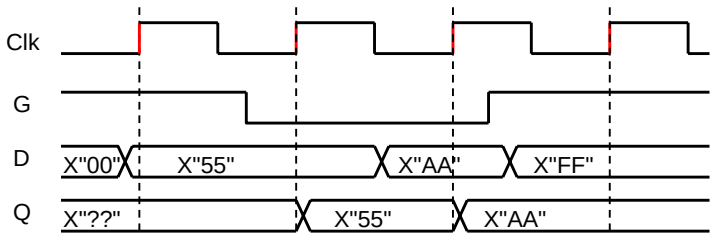
mit Reset



mit Enable



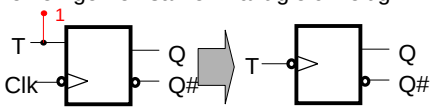
Zeitdiagramm für Register mit Enable

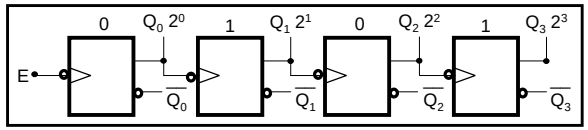


Der Zähler (1/3)

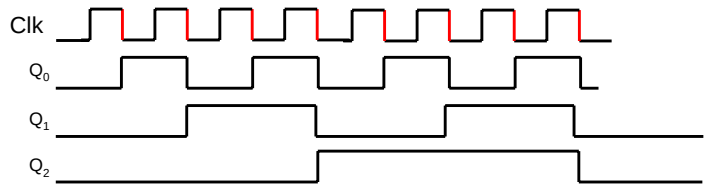
Asynchronzähler
Besteht aus Flipflops, die nicht durch einen gemeinsamen Takt gleichzeitig geschaltet werden.

Asynchrone Dualzähler
Duales Zahlensystem,
Aufbau aus T-Flipflops





Prinzipieller Aufbau



Der Zähler (2/3)

Clk	Q_i	Carry
↑	0 0 0 0	0
↑	0 0 0 1	0
↑	0 0 1 0	0
↑	0 0 1 1	0
↑	0 1 0 0	0
↑	0 1 0 1	0
↑	0 1 1 0	0
↑	0 1 1 1	0
↑	1 0 0 0	0
↑	1 0 0 1	0
↑	1 0 1 0	0
↑	1 0 1 1	0
↑	1 1 0 0	0
↑	1 1 0 1	0
↑	1 1 1 0	0
↑	1 1 1 1	1
↑	0 0 0 0	0

Grundsätzliche Zähler-Tabelle

Tabelle (Kurzform)

Clk	Q_i	Carry
-	Q_i	0
-	1111	1
↑	Q_i+1	0

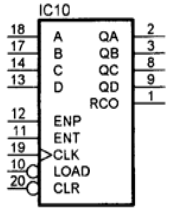
mit bevorrechtigten Eingängen

Clk, R, Ld, En, D_i					Q_i	Carry
-	1	-	-	-	0000	0
↑	0	1	-	D_i	D_i	0
↑	0	0	0	-	Q_i	0
↑	0	0	1	-	Q_i+1	0
-	0	0	0	-	1111	0
-	0	0	1	-	1111	1

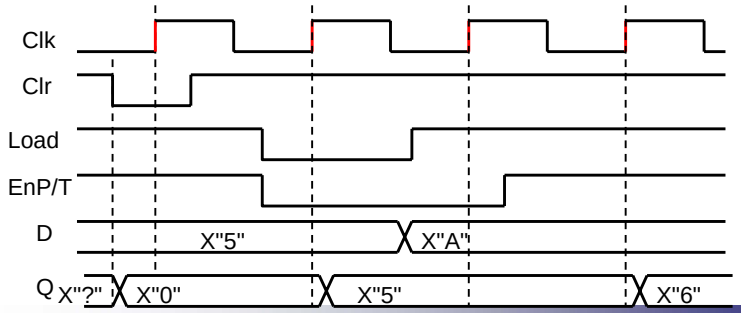
Wird zur Kaskadierung mit dem Enable-Eingang des nachfolgenden Zählers verbunden, der dann bei der nächsten positiven Flanke hochzählen kann.
Achtung: En beeinflusst Carry

Der Zähler (3/3)

Symbol



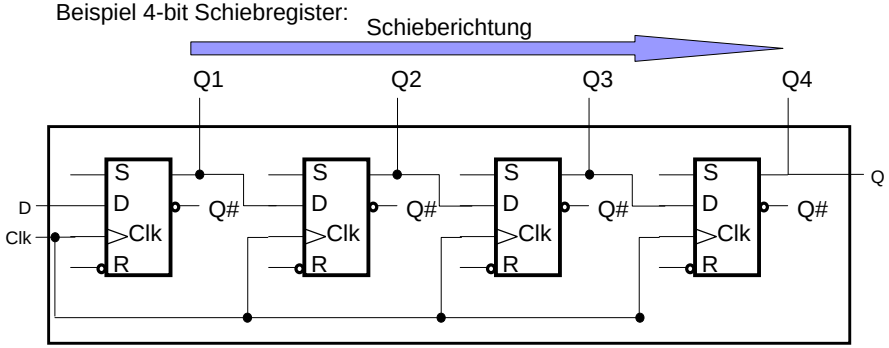
Zeitdiagramm



Das Schieberegister (1/5)

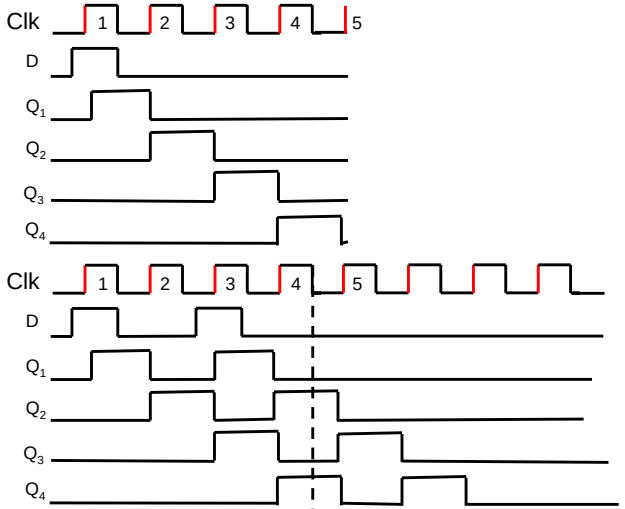
Aneinanderreihung von N Flipflops, deren Ausgang jeweils mit dem Eingang des nachfolgenden Flipflops verbunden ist und mit gemeinsamer Taktleitung.

Beispiel 4-bit Schiebregister:



Das Signal, das an D anliegt, wird mit jedem Clock-Signal zum nachfolgenden Flipflop weiter geschoben.

Das Schieberegister (2/5)

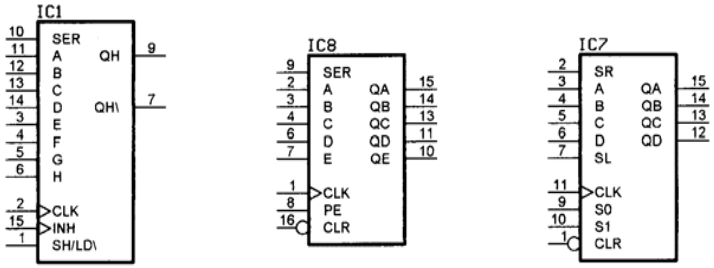


Das Schieberegister (3/5)

Tabelle

Clk, SerIn		*Q*
-	1	0000
↑	1	0001
↑	0	0010
↑	1	0101
↑	1	1011
↑	0	0110

verschiedene Implementierungen



Das Schieberegister (4/5)

- Anwendung:
  - Serielle Datenübertragung
    - Parallel – Serienwandlung
    - Serien – Parallelwandlung
  - Verzögerung
  - Rechenoperationen
- Verschiedene Ausführung, je nach Anwendung:
  - Nur Serien-Eingang und –Ausgang
    - für Verzögerungen
  - Serien-Eingang und -Ausgang plus FF-Ausgänge
    - zur Seriell-Parallel Wandlung
  - Serien-Eingang und -Ausgang plus FF Eingänge
    - zur Parallel-Seriell Wandlung
  - Universal Schieberegister: Alle Ein- Ausgänge

Das Schieberegister (5/5)

- Bevorrechtigte Eingänge:
  - Reset: setzt alle FFs auf 0
  - Set ist ungebräuchlich
  - Load: paralleles Laden des Registers
  - Shift Enable: Es wird geschoben wenn Enable 1 ist.
  - Direction: Nach rechts oder links schieben.
- Mathematische Bedeutung des Schiebens:
  Bsp:
  0000 0010 dezimal 2
  0000 0100 dezimal 4
  0000 1000 dezimal 8
  - Schieben nach links: Multiplikation mit 2
  - Schieben nach rechts: Division durch 2

Ausblick

- Mit diesen Bausteinen kann man nun komplexe Schaltwerke aufbauen, die aus in der Größenordnung 107 (!) Gattern und Latches bestehen. Solch hochkomplexe Systeme kann man natürlich nicht manuell und unsystematisch aufbauen. Generell gilt:
  - Realisiere Funktionen durch Gatterschaltungen
  - Realisiere Register und kritische Speicherplätze durch Master Slave Latches
  - Benutze zur Konstruktion **Hardwarebeschreibungssprachen** (Hardware Description Languages, HDL) und übersetze sie in Schaltungen, so wie man Programme in Maschinenprogramme transformiert.