

# Digitaltechnik

Wintersemester 2025/2026

Vorlesung 8



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



**ENCRYPTO**  
CRYPTOGRAPHY AND  
PRIVACY ENGINEERING

## 1 Arithmetische Grundschaltungen 2

## 2 Intro: Sequentielle Schaltungen

## 3 Speicherelemente

## 4 Synchrone sequentielle Logik



Harris 2016  
Kap. 3.1 - 3.3

Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen










Abgabefrist für Hausaufgabe C zu  
Vorlesungen 05 und 06 **diese** Woche  
Freitag 23:59!  
Wöchentliches Moodle-Quiz nicht vergessen!

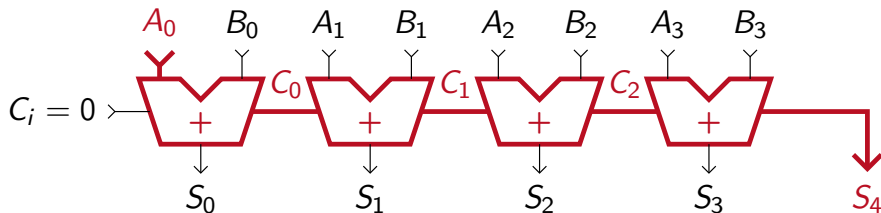
## 1 Arithmetische Grundsaltungen 2

## 2 Intro: Sequentielle Schaltungen

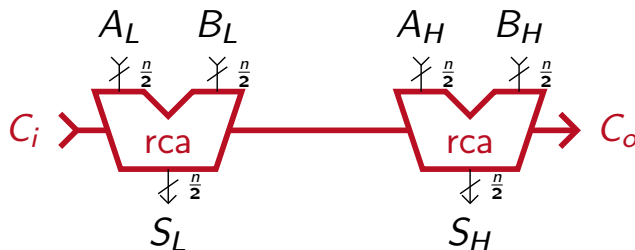
## 3 Speicherelemente

## 4 Synchrone sequentielle Logik

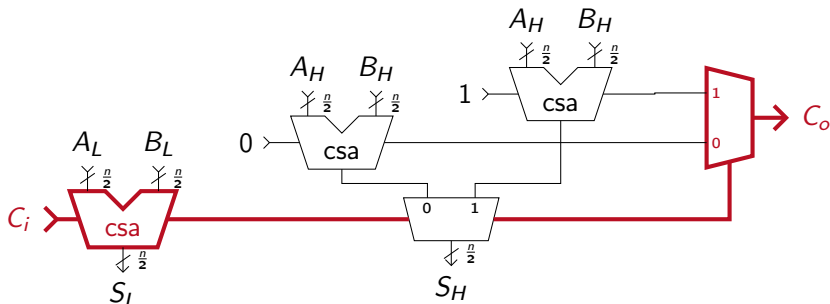
Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen



- Überträge werden über Kette von 1 bit Volladdierern vom LSB zum MSB weitergegeben
  - ⇒ langer **kritischer Pfad** (steigt linear mit Bitbreite)
  - ⇒ schnellere Addierer müssen Übertragskette aufbrechen, benötigen mehr Hardware



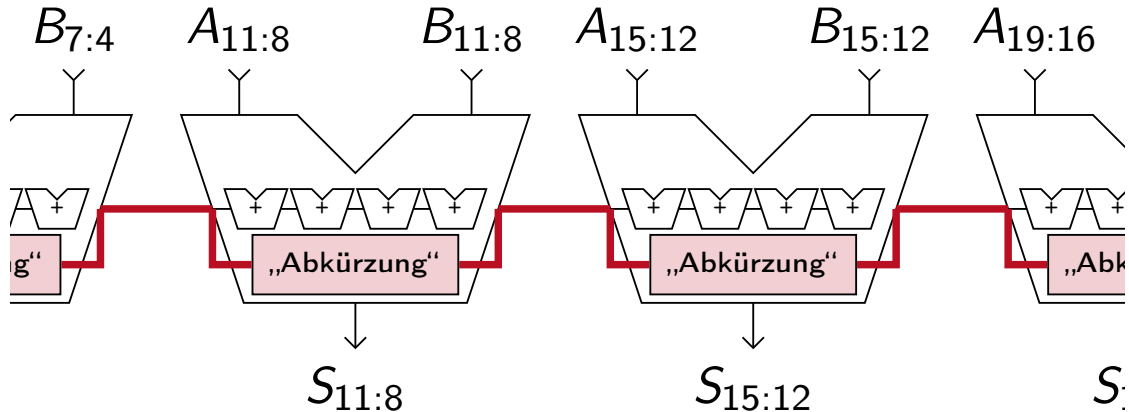
- Aufteilen in unteres (Low) und oberes (High) Halbwort („Divide and Conquer“)
  - zweiter Addierer muss auf Übertrag aus erstem Addierer „warten“
- ⇒ **kritische Pfade** beider Teiladdierer werden addiert
- für schnellen Addierer müssen *oberes und unteres Halbwort gleichzeitig* bearbeitet werden



- Übertrag vom unterem (L) in oberes (H) Halbwort kann nur 0 oder 1 sein
  - für beide Optionen kann oberes Halbwort schon mal vorberechnet werden
  - Auswahl des richtigen Ergebnisses, sobald tatsächlicher Übertrag bekannt
- ⇒ nach halbem CSA folgt nur noch ein MUX auf **kritischem Pfad**

# Carry Lookahead Adder (CLA)

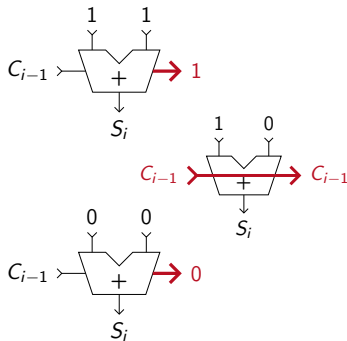
Idee: Aufbau



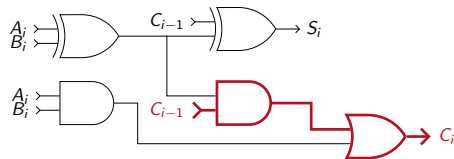
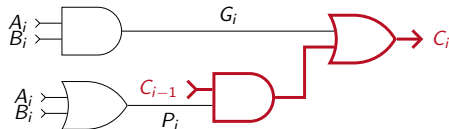


	1	0	1	1	0	Übertrag
		1	0	0	1	Summand $A$
+		1	0	1	1	Summand $B$
<hr/>						
=	1	0	1	0	0	Summe

- für  $A_i B_i = 1$  ist  $C_i = 1$  unabhängig von  $C_{i-1}$   
⇒ Spalte  $i$  *generiert* einen Übertrag („generate“)
- für  $A_i + B_i = 1$  ist  $C_i = 1$  wenn  $C_{i-1} = 1$   
⇒ Spalte  $i$  *leitet* Übertrag *weiter* („propagate“)
- für  $A_i + B_i = 0$  ist  $C_i = 0$  unabhängig von  $C_{i-1}$   
⇒ Spalte  $i$  *leitet* Übertrag *nicht weiter*



- Generate-Flag für Spalte  $i$ :  $G_i = A_i B_i$
- Propagate-Flag für Spalte  $i$ :  $P_i = A_i + B_i$
- ⇒ Übertrag aus Spalte  $i$ :  $C_i = G_i + P_i C_{i-1}$
- Bei naiver Verwendung davon (links) kein Vorteil ggü. Volladdierer (rechts):  
In beiden Fällen AND und OR auf kritischem Pfad zwischen  $C_{i-1}$  und  $C_i$





- Generate- und Propagate-Flags können über mehrere Spalten kombiniert werden (hier gezeigt für  $k = 4$  Spalten)

- $k$ -Spalten Block *propagiert* Übertrag, wenn jede einzelne Spalte propagiert

$$\Rightarrow P_{3:0} = P_3 P_2 P_1 P_0$$

- $k$ -Spalten Block *generiert* Übertrag, wenn eine der Spalten generiert, und alle anderen Spalten darüber propagieren

$$\Rightarrow G_{3:0} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$

- Übertrag überspringt  $k$  Spalten auf einmal:

$$\begin{aligned} C_n &= G_{n:n-k+1} + C_{n-k} \cdot P_{n:n-k+1} \\ &= (G_n + P_n (G_{n-1} + \dots + (P_{n-k+2} G_{n-k+1}))) + C_{n-k} \cdot \prod_{i=n-k+1}^n P_i \end{aligned}$$

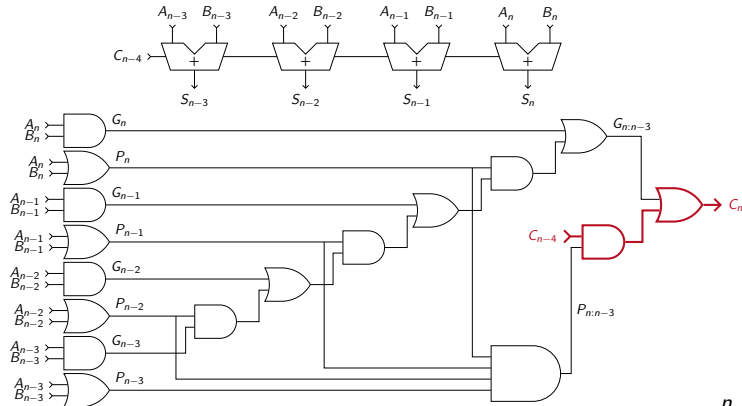


# Carry Lookahead Adder (CLA)

## Block für $k = 4$ Spalten

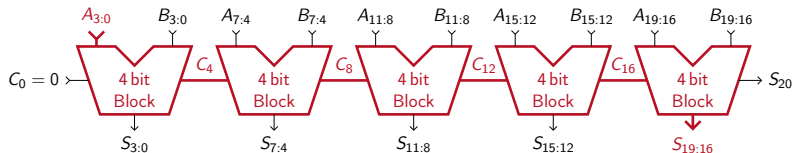


**ENCRYPTO**  
CRYPTOGRAPHY AND  
PRIVACY ENGINEERING



$$C_n = (G_n + P_n (G_{n-1} + \dots + (P_{n-k+2} G_{n-k+1}))) + C_{n-k} \cdot \prod_{i=n-k+1}^n P_i$$

- Propagate und Generate Signale in allen Blöcken gleichzeitig berechnen
  - für große Bitbreiten  $N$  dominiert  $\frac{N}{k} \cdot (t_{pd,AND} + t_{pd,OR})$
- ⇒ Blöcke möglichst groß wählen (kostet aber mehr Ressourcen)
- Kritischer Pfad für  $N = 20$  bit und  $k = 4$ :
    - RCA:  $\approx 20 \cdot (t_{pd,AND} + t_{pd,OR})$  (siehe Übung 07)
    - CLA:  $\approx 10 \cdot (t_{pd,AND} + t_{pd,OR})$  (siehe Übung 08)
      - G/P von Block 1
      - Carry zum letzten Block
      - RCA im letzten Block





- Parallel Prefix Adder
  - *alle*  $C_i$  per Generate und Propagate möglichst schnell bestimmen
  - Kritischer Pfad logarithmisch in Bitbreite  $N$
- Carry-Save Adder
  - Verwendet parallele Volladdierer, um 3 Werte in Vektor aus Carries  $C_i$  und Summen  $S_i$  zu addieren












Harris 2013/2016  
Kap. 5.2.1

1 Arithmetische Grundsaltungen 2

2 Intro: Sequentielle Schaltungen

3 Speicherelemente

4 Synchrone sequentielle Logik

Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen





- kombinatorische Logik („Schaltnetz“)
  - Ausgänge hängen nur von *aktuellen* Eingangswerten ab
- Warum reichen kombinatorische Schaltungen nicht aus?
  - Nicht alle Funktionalitäten lassen sich sinnvoll als kombinatorische Schaltungen realisieren
    - (Zwischen-)Ergebnisse können nicht gespeichert/wiederverwendet werden
    - kritische Pfade können nicht beliebig lang werden
  - Zeitverhalten bei kombinatorischen Schaltungen schwer kontrollierbar (siehe Timing-Analyse in früheren Vorlesungen)












- Ausgänge hängen ab von
  - aktuellen Eingabewerten und
  - *vorherigen* Eingabewerten
- ⇒ sequentielle Schaltung speichert *internen Zustand*
  - (Kurzzeit-)Gedächtnis repräsentiert bisherige Eingabesequenzen
  - realisiert durch Rückkopplungen von Ausgängen
- ⇒ nicht kombinatorisch

1 Arithmetische Grundsaltungen 2

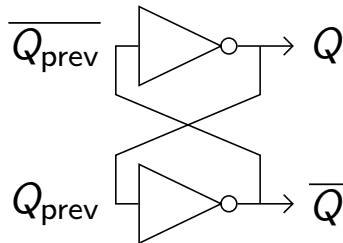
2 Intro: Sequentielle Schaltungen

3 Speicherelemente

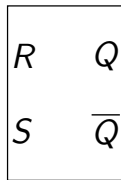
4 Synchrone sequentielle Logik

Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen

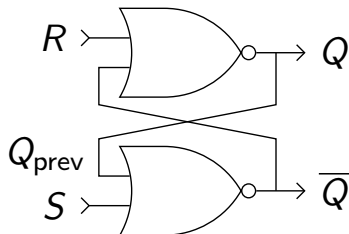
- Grundlage des Zustandsspeichers
- zwei Inverter mit Rückkopplung:  
 $Q_{\text{prev}}$  (previous  $Q$ )
- zwei Ausgänge:  $Q, \overline{Q}$
- speichert 1 bit durch zwei stabile Zustände
  - $Q = 0 \Rightarrow \overline{Q} = 1 \Rightarrow Q = 0$
  - $Q = 1 \Rightarrow \overline{Q} = 0 \Rightarrow Q = 1$
- *keine* Eingänge
  - $\Rightarrow$  gespeicherter Zustand kann nicht beeinflusst werden



- bistabile Grundschtaltung mit NOR statt NOT
- NOR: Ausgang 0 wenn einer der Inputs 1 ist
- Interpretation der freien Eingänge  $S$  und  $R$ 
  - $\bar{S} \bar{R} \rightarrow$  Zustand halten („latch“ = verriegeln)
  - $\bar{S} R \rightarrow$  Zustand auf 0 rücksetzen („reset“  $R$ )
  - $S \bar{R} \rightarrow$  Zustand auf 1 setzen („set“  $S$ )
  - $S R \rightarrow$  ungültiger Zustand ( $Q = \bar{Q} = 0$ )

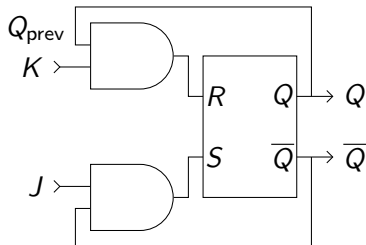
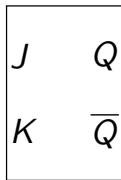


$S$	$R$	$Q_{\text{prev}}$	$Q$	$\bar{Q}$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0



- Ungültigen Zustand  $SR$  am SR-Latch verhindern
- Historisch unklar, woher die Bezeichnung "JK" kommt
- Interpretation der freien Eingänge  $J$  und  $K$ 
  - $\bar{J} \bar{K} \rightarrow$  Zustand halten
  - $\bar{J} K \rightarrow$  Zustand auf 0 rücksetzen, falls nötig
  - $J \bar{K} \rightarrow$  Zustand auf 1 setzen, falls nötig
  - $J K \rightarrow$  Zustand invertieren („toggle“)

$J$	$K$	$Q_{\text{prev}}$	$S$	$R$	$Q$	$\bar{Q}$
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	0	0	1
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	1	0	1

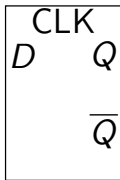


- Daten-Latch mit Taktsignal (CLK) und Dateneingang ( $D$ )

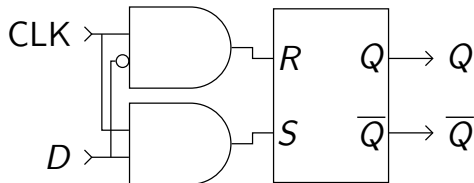
- CLK = 1  $\rightarrow$  Zustand auf  $D$  setzen (Latch transparent)
- CLK = 0  $\rightarrow$  Zustand halten (Latch nicht transparent)

$\Rightarrow$  ungültiger Zustand am SR-Latch wird vermieden

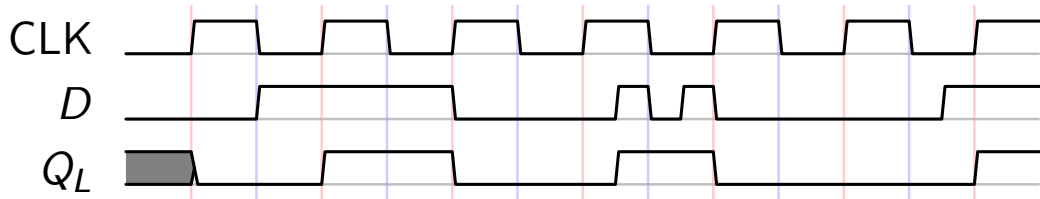
- Rückkopplung nur noch im SR-Latch



CLK	$D$	$S$	$R$	$Q$
0	0	0	0	$Q_{\text{prev}}$
0	1	0	0	$Q_{\text{prev}}$
1	0	0	1	0
1	1	1	0	1



- periodische Taktsignale üblicherweise symmetrisch
  - 0-Phase und 1-Phase gleich lang
- D-Latch ist Taktphasen-gesteuert
  - für Hälfte der gesamten Zeit transparent
  - sequentielle Schaltungen mit D-Latches für Hälfte der Zeit kombinatorisch
- breites „Abtastfenster“ sorgt für Unschärfe
  - bspw. unklar, ob Störimpuls übernommen wurde





## ■ Zwei D-Latches in Serie

- First ( $F$ )
- Second ( $S$ )
- komplementäre Taktsignale

## ■ CLK = 0

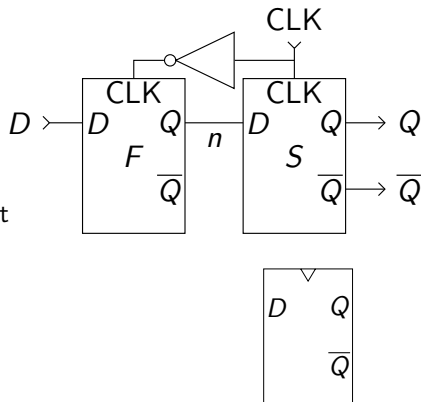
- First transparent  $\rightarrow n = D$
- Second nicht transparent  $\rightarrow Q$  bleibt unverändert

## ■ CLK = 1

- First nicht transparent  $\rightarrow n$  bleibt unverändert
- Second transparent  $\rightarrow Q = n$

## $\Rightarrow$ Taktflanken-gesteuert

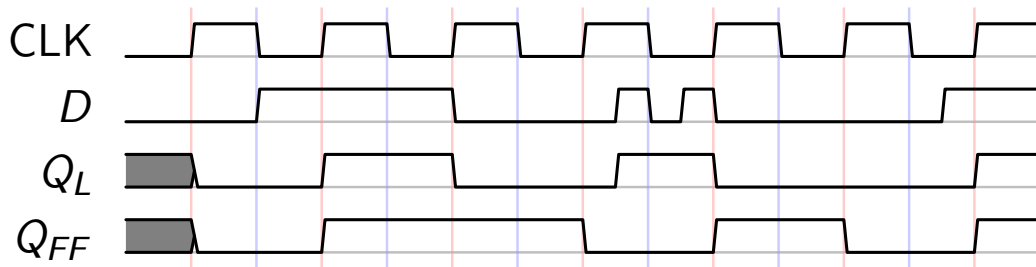
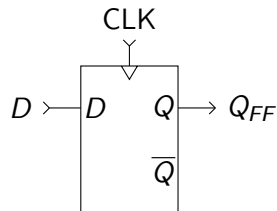
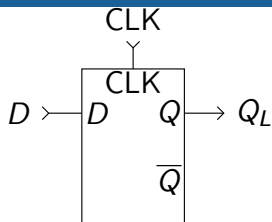
- genau bei steigender CLK Flanke wird  $Q = D$
- es wird der Wert von  $D$  übernommen, der **unmittelbar vor** der Taktflanke anliegt



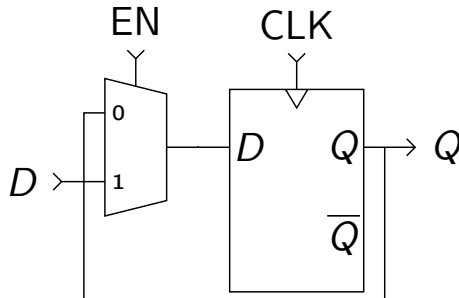
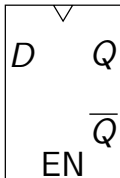
# Vergleich D-Latch mit D-Flip-Flop



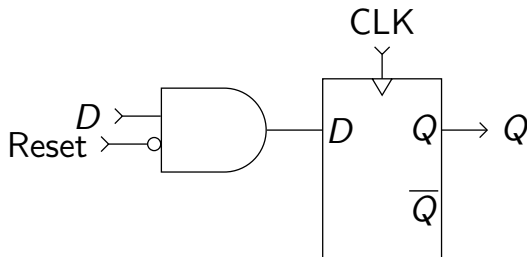
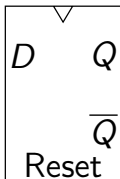
ENCRYPTO  
CRYPTOGRAPHY AND  
PRIVACY ENGINEERING



- Freigabeeingang (enable EN) steuert, wann Daten gespeichert werden
  - $EN = 1 \rightarrow D$  wird bei steigender CLK-Flanke gespeichert
  - $EN = 0 \rightarrow Q$  bleibt auch bei steigender CLK-Flanke unverändert
- Anwendungsbeispiele
  - Zähler
  - Speicher mit Adressdecoder

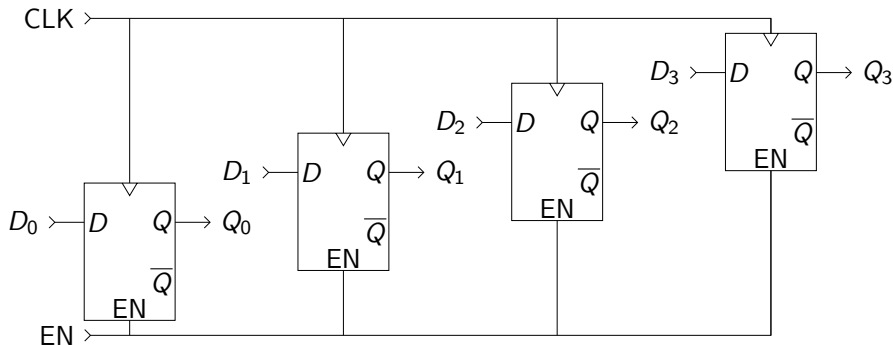


- Reset setzt internen Zustand unabhängig von  $D$  auf 0
  - synchron: nur zur steigenden Taktflanke wirksam
  - asynchron: jederzeit (unabhängig von CLK)
- Anwendungsbeispiele
  - sequentielle Schaltung in definierten Ausgangszustand versetzen
- setzbare Flip-Flops analog





- Register bestehend aus parallelen D-Flip-Flops
- Bei Shift-Register ist  $D_i = Q_{i-1}$












1 Arithmetische Grundsaltungen 2

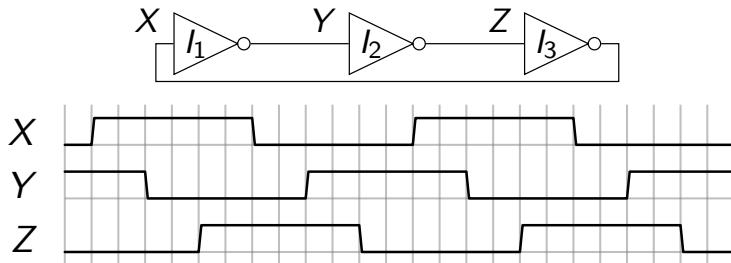
2 Intro: Sequentielle Schaltungen

3 Speicherelemente

4 Synchrone sequentielle Logik

Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen

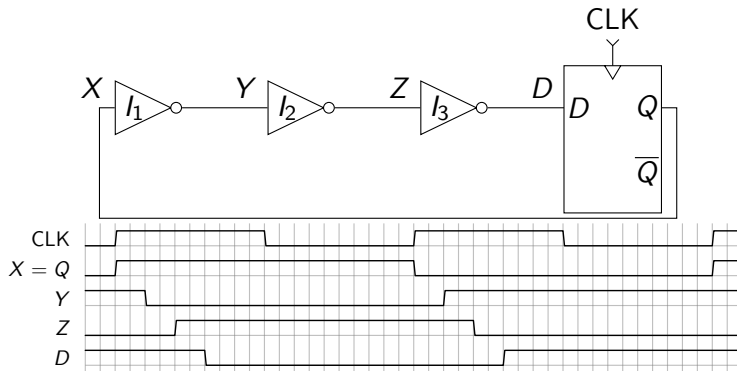
- alle nicht-kombinatorischen Schaltungen
- erlaubt Rückkopplungen, beispielsweise:



⇒ instabile (oszillierende) Schaltung

- Verhalten abhängig von Herstellungsprozess, Spannung, Temperatur
- nicht vorhersagbar

- Rückkopplungen durch Register aufbrechen
  - halten den Zustand der Schaltung
  - ändern Zustand nur zur Taktflanke
- ⇒ gesamte Schaltung *synchronisiert* mit Taktflanke







- Regeln für Aufbau
  - jedes Schaltungselement ist entweder Register oder kombinatorische Schaltung
  - mindestens ein Schaltungselement ist ein Register
  - alle Register werden durch gleiches Taktsignal gesteuert
  - jeder zyklische Pfad enthält mindestens ein Register
- Anwendungsbeispiele
  - Pipelines (nächste Vorlesung)
  - Endliche Zustandsautomaten (übernächste Vorlesung)
- Wie schnell kann so eine Schaltung betrieben werden?
  - ⇒ Was ist die kürzeste Taktperiode? (nächste Vorlesung)



1 Arithmetische Grundsaltungen 2

2 Intro: Sequentielle Schaltungen

3 Speicherelemente










4 Synchrone sequentielle Logik

nächste Vorlesung beinhaltet

- Zeitverhalten synchroner sequentieller Logik
- SystemVerilog für sequentielle Logik
- Parallelität und Pipelines

**Hausaufgabe C zu Vorlesungen 05 und 06 muss bis diese Woche Freitag 23:59 abgegeben werden.**

**Wöchentliches Moodle-Quiz nicht vergessen!**

Anwendungssoftware 	Programme
Betriebssysteme 	Gerätetreiber
Architektur 	Befehle Register
Mikroarchitektur 	Datenpfade Steuerung
Logik 	Addierer Speicher
Digital-schaltungen 	UND Gatter Inverter
Analog-schaltungen 	Verstärker Filter
Bauteile 	Transistoren Dioden
Physik 	Elektronen