

Digitaltechnik

Wintersemester 2017/2018

7. Vorlesung



TECHNISCHE
UNIVERSITÄT
DARMSTADT





1. Einleitung
2. Sequentielle Schaltungen
3. Speicherelemente für sequentielle Schaltungen
4. Synchrone sequentielle Logik
5. Zusammenfassung

Einleitung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

1110111000101001110110010110000110110000
1110100111010011110111001110011011001110
1000101010001111011010110010010011011110
1011110111001010111010001011011010001001
0100010110000000011001011000101001001000
1100110001001110101001110111011101110000
0000101011000111010111101110001001110111
0100100011101111010001101100000101111000
0001001110000001100101100111011011000000
1100001101001011111011001100101001001111
0001010000101001000011011000010111100001
1011000100100101010001010100011001110110
1111000111110011011100001100111001011100
0011111000100101000011001101110011100010
0100000010011100111111110100111010011001
0110100111110100000101110000001000011000



- ▶ <https://nabla.algo.informatik.tu-darmstadt.de/>
(Rechenbeispiele für Zahlensysteme)



- ▶ <https://nabla.algo.informatik.tu-darmstadt.de/>
(Rechenbeispiele für Zahlensysteme)
- ▶ Vorgesehene Bearbeitungszeit für Übungen



- ▶ <https://nabla.algo.informatik.tu-darmstadt.de/>
(Rechenbeispiele für Zahlensysteme)
- ▶ Vorgesehene Bearbeitungszeit für Übungen
- ▶ Espresso Binär- und Beispieldateien verfügbar



- ▶ <https://nabla.algo.informatik.tu-darmstadt.de/>
(Rechenbeispiele für Zahlensysteme)
- ▶ Vorgesehene Bearbeitungszeit für Übungen
- ▶ Espresso Binär- und Beispieldateien verfügbar
- ▶ Vertretung durch Raad Bahmani für V8

Rückblick auf die letzten Vorlesungen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Kombinatorische Logik
 - ▶ Boole'sche Gleichungen
 - ▶ Boole'sche Algebra
 - ▶ Bubble Pushing
 - ▶ Logik-Realisierung mit Basis-Gattern
 - ▶ Karnaugh Diagramme
 - ▶ Algorithmische Logikminimierung
 - ▶ Vierwertige Logik
 - ▶ Zeitverhalten



Harris 2013
Kap. 2

Wiederholung: Espresso

Minimierung eines Prioritätsencoders: $\mathbb{B}^4 \rightarrow \mathbb{B}^4$



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ One-Hot-Encoding
des höchsten
gesetzten Eingangs
- ▶ Verwendet für
 - ▶ Bus-Arbitrierung
 - ▶ Interrupt-
Steuerung

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	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	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

Wiederholung: Espresso

Minimierung eines Prioritätsencoders: $\mathbb{B}^4 \rightarrow \mathbb{B}^4$



TECHNISCHE
UNIVERSITÄT
DARMSTADT

priorityencoder.esp

```
1  .i 4
2  .o 4
3  0000 0000
4  0001 0001
5  0010 0010
6  0011 0010
7  0100 0100
8  0101 0100
9  0110 0100
10 0111 0100
11 1000 1000
12 1001 1000
13 1010 1000
14 1011 1000
15 1100 1000
16 1101 1000
17 1110 1000
18 1111 1000
```

Wiederholung: Espresso

Minimierung eines Prioritätsencoders: $\mathbb{B}^4 \rightarrow \mathbb{B}^4$



TECHNISCHE
UNIVERSITÄT
DARMSTADT

priorityencoder.esp

1	.i	4
2	.o	4
3	0000	0000
4	0001	0001
5	0010	0010
6	0011	0010
7	0100	0100
8	0101	0100
9	0110	0100
10	0111	0100
11	1000	1000
12	1001	1000
13	1010	1000
14	1011	1000
15	1100	1000
16	1101	1000
17	1110	1000
18	1111	1000

espresso priorityencoder.esp

1	.i	4	Y_3
2	.o	4	\downarrow
3	0001	0001	
4	001-	0010	
5	01--	0100	
6	1---	1000	$\uparrow Y_0$

Wiederholung: Espresso

Minimierung eines Prioritätsencoders: $\mathbb{B}^4 \rightarrow \mathbb{B}^4$



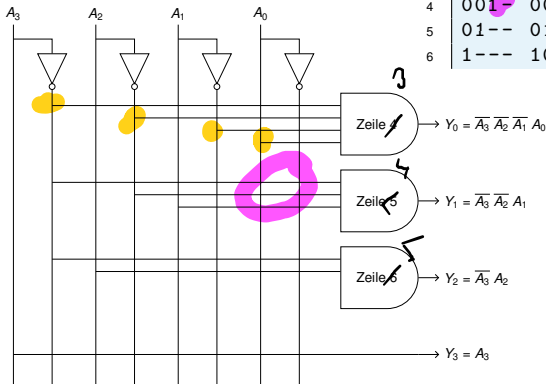
TECHNISCHE
UNIVERSITÄT
DARMSTADT

priorityencoder.esp

1	.i	4
2	.o	4
3	0000	0000
4	0001	0001
5	0010	0010
6	0011	0010
7	0100	0100
8	0101	0100
9	0110	0100
10	0111	0100
11	1000	1000
12	1001	1000
13	1010	1000
14	1011	1000
15	1100	1000
16	1101	1000
17	1110	1000
18	1111	1000

espresso priorityencoder.esp

1	.i	4
2	.o	4
3	0001	0001
4	001-	0010
5	01--	0100
6	1---	1000



Wiederholung: Espresso

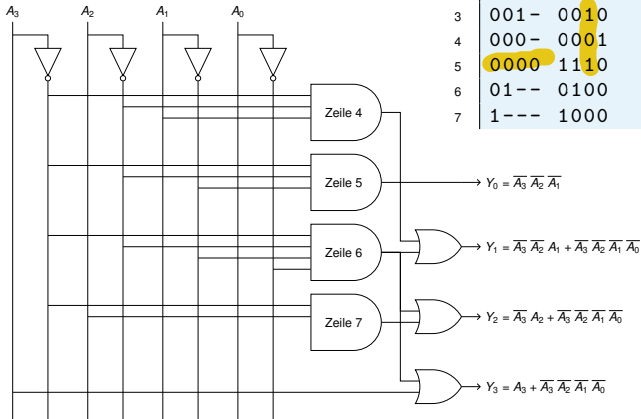
Minimierung eines Prioritätsencoders: $\mathbb{B}^4 \rightarrow \mathbb{B}^4$



TECHNISCHE
UNIVERSITÄT
DARMSTADT

priorityencoder.mod.esp

1	.i	4
2	.o	4
3	0000	1111
4	0001	0001
5	0010	0010
6	0011	0010
7	0100	0100
8	0101	0100
9	0110	0100
10	0111	0100
11	1000	1000
12	1001	1000
13	1010	1000
14	1011	1000
15	1100	1000
16	1101	1000
17	1110	1000
18	1111	1000



espresso priorityencoder.mod.esp

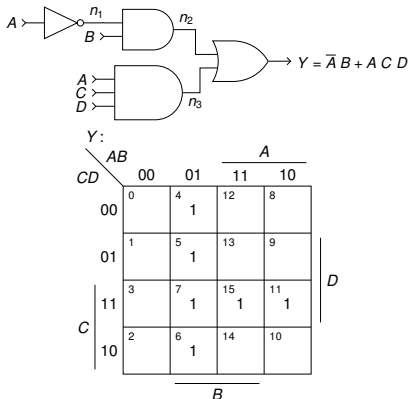
1	.i	4
2	.o	4
3	001-	0010
4	000-	0001
5	0000	1110
6	01--	0100
7	1---	1000

Wiederholung: Störimpulse

Abhängigkeit von Übergangsrichtung

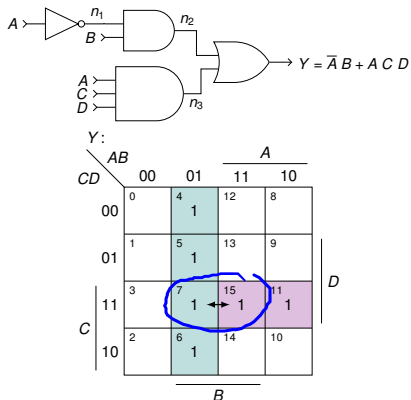


TECHNISCHE
UNIVERSITÄT
DARMSTADT



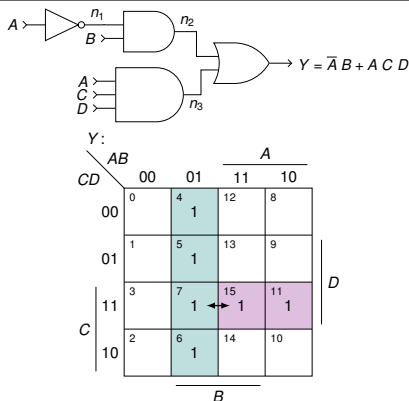
Wiederholung: Störimpulse

Abhängigkeit von Übergangsrichtung



Wiederholung: Störimpulse

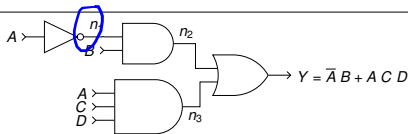
Abhängigkeit von Übergangsrichtung



	NOT	AND	AND3	OR
t_{pd}	1 ns	2 ns	2 ns	3 ns
t_{cd}	1 ns	2 ns	2 ns	3 ns

Wiederholung: Störimpulse

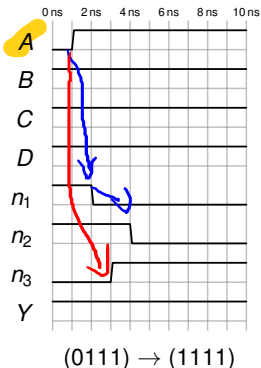
Abhängigkeit von Übergangsrichtung



Y:

AB		A			
		00	01	11	10
CD	00	0	4 1	12	8
	01	1	5 1	13	9
	11	3	7 1	15 1	11 1
	10	2	6 1	14	10
		B			

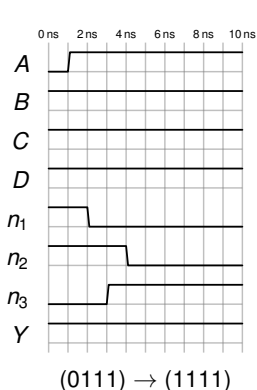
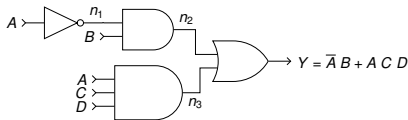
Blue arrows indicate a transition from state 7 (11, 1) to state 15 (11, 1) and back.



	NOT	AND	AND3	OR
t_{pd}	1 ns	2 ns	2 ns	3 ns
t_{cd}	1 ns	2 ns	2 ns	3 ns

Wiederholung: Störimpulse

Abhängigkeit von Übergangsrichtung

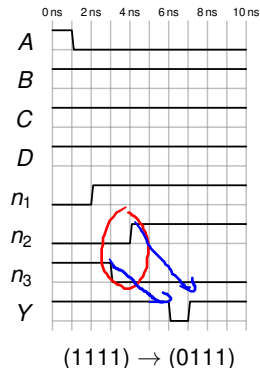


Y:

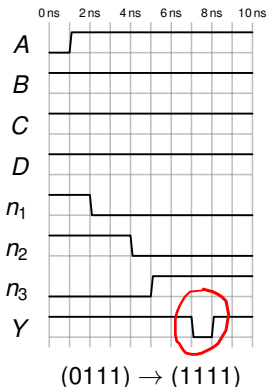
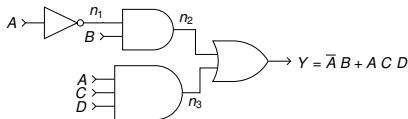
AB		A			
		00	01	11	10
CD	00	0	4 1	12	8
	01	1	5 1	13	9
C	11	3	7 1	15 1	11 1
	10	2	6 1	14	10
		B			

Arrows in the Karnaugh map indicate the transition from cell 7 (11, 01) to cell 15 (11, 11) for signal n3, and from cell 15 to cell 11 (11, 10) for signal n2.

	NOT	AND	AND3	OR
t_{pd}	1 ns	2 ns	2 ns	3 ns
t_{cd}	1 ns	2 ns	2 ns	3 ns



Wiederholung: Störimpulse Abhängigkeit von Gatter-Verzögerung

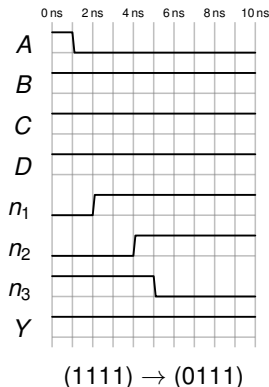


Y:

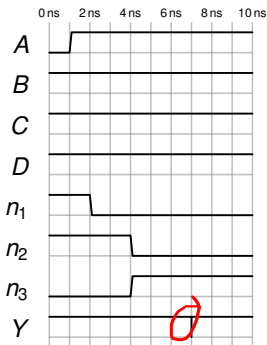
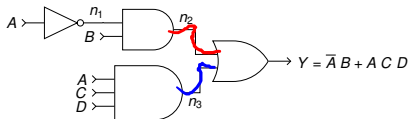
AB		A			
		00	01	11	10
CD	00	0	4 1	12	8
	01	1	5 1	13	9
C	11	3	7 1	15 1	11 1
	10	2	6 1	14	10
		B			

D

	NOT	AND	AND3	OR
t_{pd}	1 ns	2 ns	4 ns	3 ns
t_{cd}	1 ns	2 ns	4 ns	3 ns



Wiederholung: Störimpulse Abhängigkeit von Leitungsverzögerung



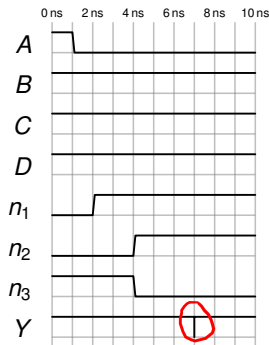
(0111) \rightarrow (1111)

Y:

AB		A			
		00	01	11	10
CD	00	0	4 1	12	8
	01	1	5 1	13	9
C	11	3	7 1	15 1	11 1
	10	2	6 1	14	10
		B			

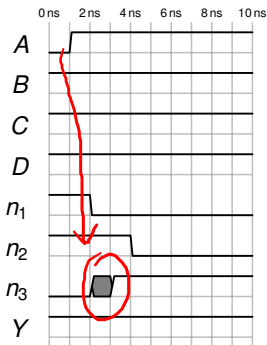
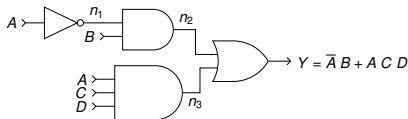
D

	NOT	AND	AND3	OR
t_{pd}	1 ns	2 ns	3 ns	3 ns
t_{cd}	1 ns	2 ns	3 ns	3 ns

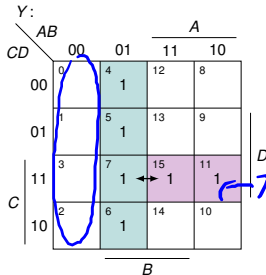


(1111) \rightarrow (0111)

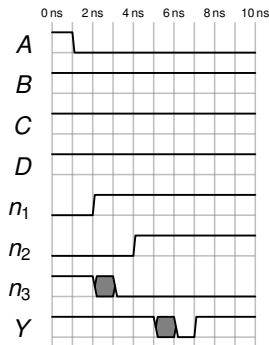
Wiederholung: Störimpulse Ausbreitung von Unsicherheit



(0111) → (1111)



	NOT	AND	AND3	OR
t_{pd}	1 ns	2 ns	2 ns	3 ns
t_{cd}	1 ns	2 ns	1 ns	3 ns



(1111) → (0111)

Überblick der heutigen Vorlesung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Sequentielle Logik
 - ▶ Sequentielle Schaltungen
 - ▶ Speicherelemente
 - ▶ Synchrone sequentielle Logik



Harris 2013
Kap. 3.1-3.3
Seite 103 - 117

Sequentielle Schaltungen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

0011010101011101111111110010111011110010
0101000000000100111101010000001111011000
1010001111011101110100110010010101100110
1110110011001100110110000011100101010001
1001101100010110111100111110010000000010
1100100100101101011101010100010110110100
0100011101111000101110000000110011100101
1011000110010101100100000101000011111
0100111100110001100011011111101111011010
1011100101111110100011001101111101101110
110111001111001111011100111011101101111
001001100111110010110101110010101111111
0100001110010110000110110111010010111101
0000111111110100100000101101100110111110
0010111000110001111101010011011100010010
10101000010011010001010001111110010011010



- ▶ Ausgänge hängen ab von
 - ▶ aktuellen Eingabewerten
 - ▶ **vorherigen Eingabewerten**
- ⇒ sequentielle Schaltung speichern **internen Zustand**
 - ▶ **(Kurzzeit-)Gedächtnis** repräsentiert Eingabesequenzen
 - ▶ realisiert durch **Rückkopplungen** von Ausgängen zu Eingängen
- ⇒ nicht kombinatorisch
- ▶ **Warum reichen kombinatorische Schaltungen nicht aus?**



- ▶ Ausgänge hängen ab von
 - ▶ aktuellen Eingabewerten
 - ▶ vorherigen Eingabewerten

⇒ sequentielle Schaltung speichern *internen Zustand*

- ▶ (Kurzzeit-)Gedächtnis repräsentiert Eingabesequenzen
- ▶ realisiert durch Rückkopplungen von Ausgängen zu Eingängen

⇒ nicht kombinatorisch

- ▶ Warum reichen kombinatorische Schaltungen nicht aus?

- ▶ (Zwischen-)Ergebnisse können nicht gespeichert / wiederverwendet werden
- ▶ kritische Pfade können nicht beliebig lang / divers werden

⇒ Zeitverhalten bei sequentiellen Schaltungen besser kontrollierbar

Speicherelemente für sequentielle Schaltungen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

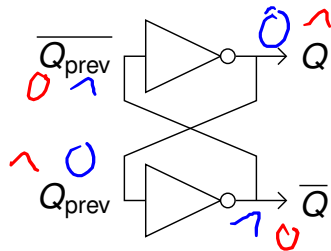
111110110101111001101000010001011110011111
1101001101111010010001010101101111100010
0001100101111011101011100100011010110110
0110010111000100011101100111101001010100
0111001111000000101111010010010101011001
0000001011011000101001101011101111010010
0101111010011100101101100101101101010010
000010000010111011001111110111010000
1000001011011110011101101100001111011011
0010100111111100001101101110101101010100
1100000010001101110010100000011011011110
1100111100101101011111001001100100010011
1010100000001010101100111110100011110111
0001110000001010010011001111100110110001
1010011001011101110001101101001011111001
1000111110110010000111110100111100011111

Bistabile Grundsaltung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

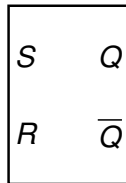


SR-Latch

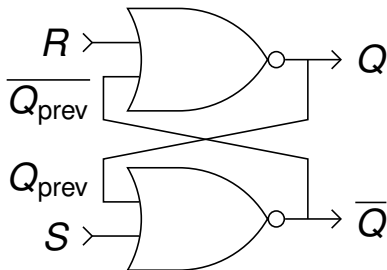


TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ bistabile Grundschaltung mit NOR statt NOT
- ▶ Interpretation der freien Eingänge S und R
 - ▶ $\overline{S} \overline{R} \rightarrow$ Zustand halten („latch“ = verriegeln)
 - ▶ $\overline{S} R \rightarrow$ Zustand auf 0 rücksetzen („reset“)
 - ▶ $S \overline{R} \rightarrow$ Zustand auf 1 setzen („set“)
 - ▶ $S R \rightarrow$ ungültiger Zustand ($Q = \overline{Q} = 0$)

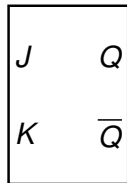


S	R	Q_{prev}	$\overline{Q}_{\text{prev}}$	Q	\overline{Q}
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	0	0	0

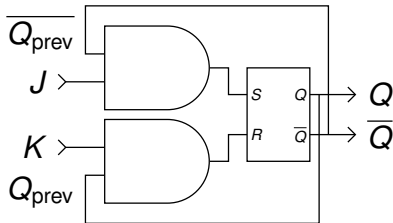


JK-Latch

- ▶ Ungültigen Zustand am SR-Latch verhindern
- ▶ 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_{prev}	\bar{Q}_{prev}	S	R	Q	\bar{Q}
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	1	0	1	0	0	0	1
0	1	1	0	0	1	0	1
1	0	0	1	1	0	1	0
1	0	1	0	0	0	1	0
1	1	0	1	1	0	1	0
1	1	1	0	0	1	0	1



D-Latch



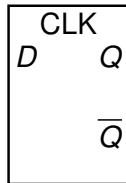
TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

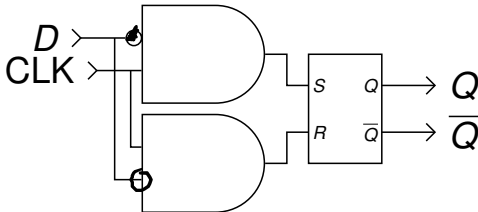
- ▶ CLK = 1 → Zustand auf D setzen (Latch transparent)
- ▶ CLK = 0 → Zustand halten (Latch nicht transparent)

⇒ ungültiger Zustand am SR-Latch wird vermieden

- ▶ Rückkopplung nur noch im SR-Latch



CLK	D	S	R	Q
0	0	0	0	Q_{prev}
0	1	0	0	Q_{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 Störimpulse übernommen



D-Flip-Flop



TECHNISCHE
UNIVERSITÄT
DARMSTADT

▶ Zwei D-Latches in Serie

- ▶ L_1 = Master
- ▶ L_2 = Slave
- ▶ komplementäre Taktsignale

▶ CLK = 0

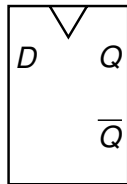
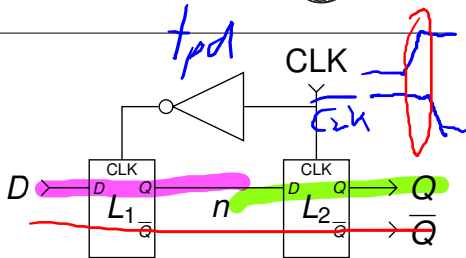
- ▶ Master transparent $\rightarrow n = D$
- ▶ Slave nicht transparent $\rightarrow Q$ bleibt unverändert

▶ CLK = 1

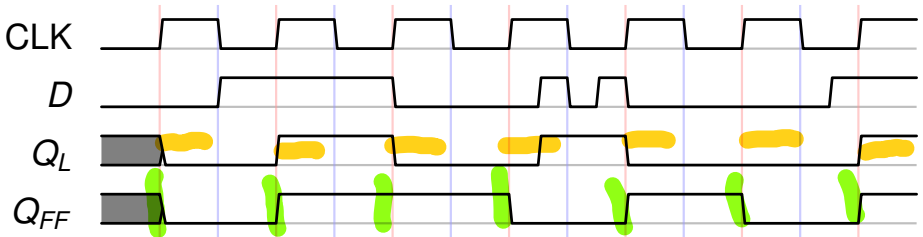
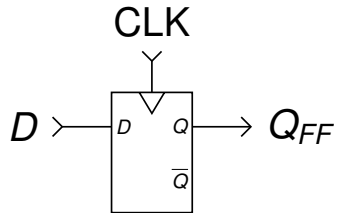
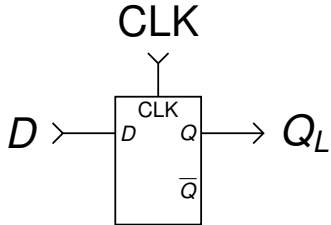
- ▶ Master nicht transparent $\rightarrow n$ bleibt unverändert
- ▶ Slave transparent $\rightarrow Q = n$

⇒ Taktflanken-gesteuert

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

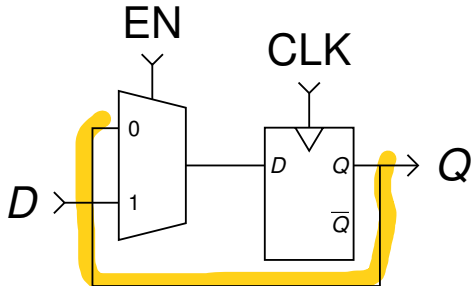
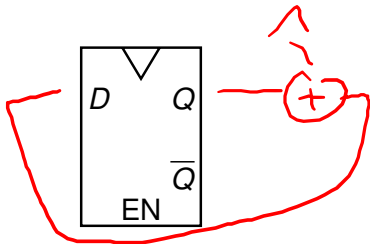


Vergleich D-Latch mit D-Flip-Flop



Flip-Flops mit Taktfreigabe

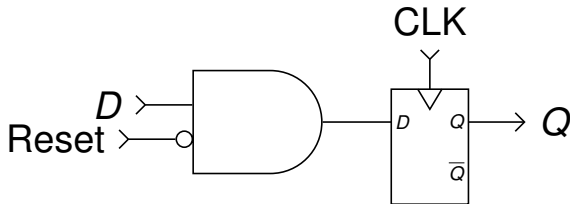
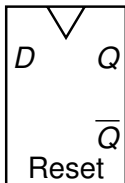
- ▶ Freigabeeingang (enable) steuert, wann Daten gespeichert werden
 - ▶ $EN = 1 \rightarrow D$ wird bei steigender CLK-Flanke gespeichert
 - ▶ $EN = 0 \rightarrow Q$ bleibt auch bei steigender CLK-Flanke unverändert
- ▶ Anwendungsbeispiele
 - ▶ Zähler
 - ▶ Speicher mit Adressdecoder



Zurücksetzbare Flip-Flops



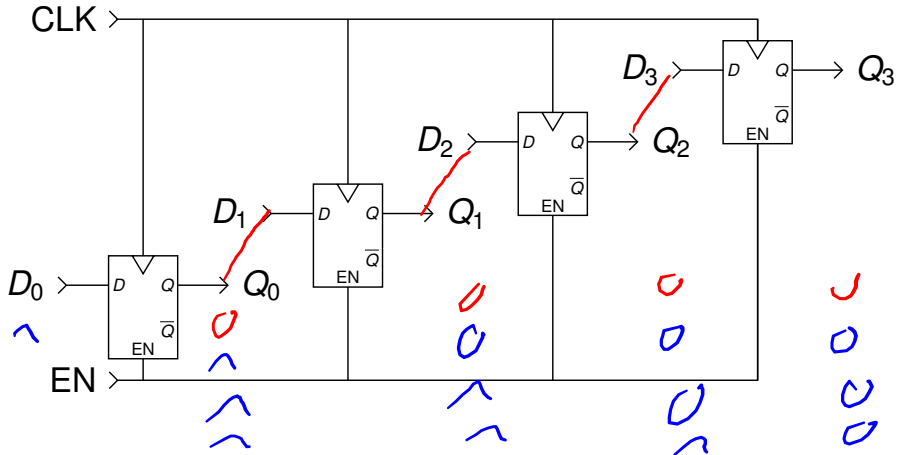
- ▶ **Reset** setzt internen Zustand unabhängig von D auf 0
 - ▶ **synchron**: nur zur steigenden Taktflanke wirksam
 - ▶ **asynchron**: jederzeit (unabhängig von CLK)
- ▶ Anwendungsbeispiele
 - ▶ **sequentielle Schaltung** in definierten Ausgangszustand versetzen
- ▶ setzbare Flip-Flops analog



Anwendungsbeispiel: (Shift-)Register



TECHNISCHE
UNIVERSITÄT
DARMSTADT



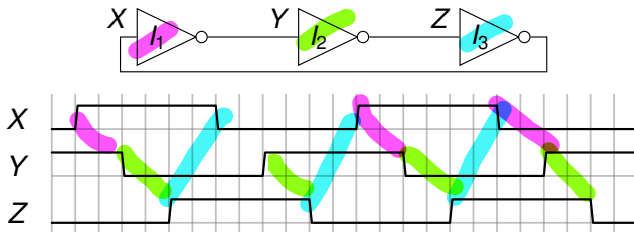
Synchrone sequentielle Logik



TECHNISCHE
UNIVERSITÄT
DARMSTADT

0010101110101100101001000011010101001000
1010011100100011010111110010010100000101
0001111101100001000111001001011011100011
0010011110101011000000001011011110100110
1010110011100110111011001100111011010101
0111100100110100110011000101010000111111
1101110011000010101001011110001110011011
01100011011000100101000100111100010011
0100011111110110011111001000101011100011
1011000110110101111000100011110110011001
1001011100001000111101110100000000011001
1111011001000010110111001111011011100111
1101110110111000011111001110110101101000
0111000111101110010101110011000010011001
1100001010111101101001110110001110001110
1101110011011101010001110110100011100000

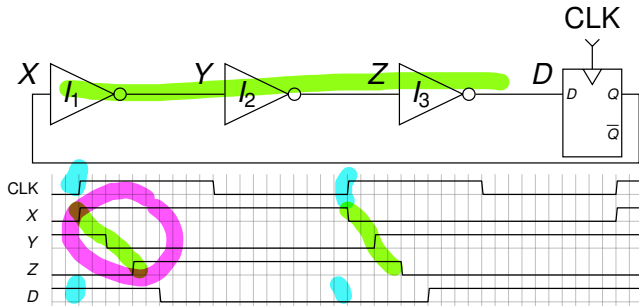
- ▶ alle nicht-kombinatorischen Schaltungen
- ▶ erlaubt Rückkopplungen, bspw:



⇒ instabile (oszillierende Schaltung)

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

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



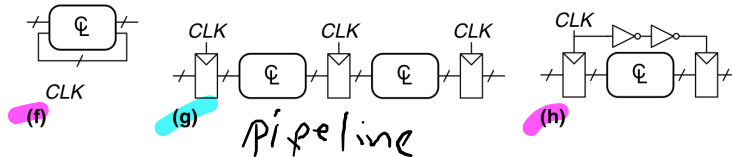
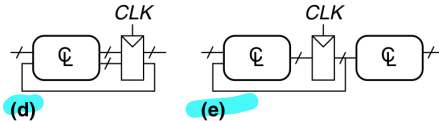
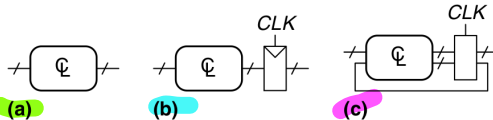


- ▶ Regeln für Aufbau
 - ▶ jedes Schaltungselement ist entweder Register oder kombinatorische Schaltung
 - ▶ mindestens ein Schaltungselement ist Register
 - ▶ alle Register werden durch gleiches Taktsignal gesteuert
 - ▶ jeder Taktzyklus enthält mindestens ein Register
- ▶ Anwendungsbeispiele
 - ▶ Endliche Zustandsautomaten
 - ▶ Pipelines

Kombinatorisch? Sequentiell? Synchron?



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Zusammenfassung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

00000111101111111110110111101111001101110
10101100110110011110111111100010111100110
0111011100100010110100011110110110100100
0000111101111011110001010001100001010010
1011000110100101000111101011011111010101
0110101100111110010010000001010100100110
0000011101101010010010001010001001000010
011101000000010011010101011000010001
1011110001101011000111100111101100111001
011010011101101111100100010000010011001
1110011010111001110010101001100101101010
0000000101001010011010100111011111100001
1001111110001000100000111010000010011111
1001101101111101111101100011100100100000
1111110000011000000110100101110110110110
0101010101011010110000000001110010100111



- ▶ Sequentielle Logik
 - ▶ Sequentielle Schaltungen
 - ▶ Speicherelemente
 - ▶ Synchrone Schaltungen

- ▶ Nächste Vorlesung behandelt
 - ▶ Endliche Zustandsautomaten