

# Digitaltechnik

Andrej Scheuer  
ascheuer@student.ethz.ch  
3. Dezember 2020

## Gates

### AND

$$Y = A \wedge B \quad Y = A \cdot B$$



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

### NAND AND aus NOR



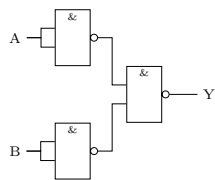
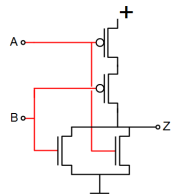
### OR

$$Y = A \vee B \quad Y = A + B$$



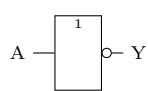
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

### NOR OR aus NAND

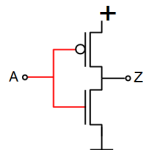


### NOT

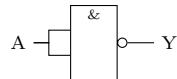
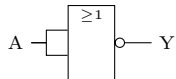
$$Y = \overline{A}$$



A	Y
0	1
1	0



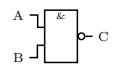
### NOT aus NOR NOT aus NAND



## Weitere Gates

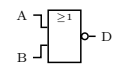
### NAND

$$C = \overline{A \wedge B}$$



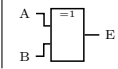
### NOR

$$D = \overline{A \vee B}$$



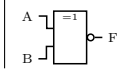
### XOR

$$E = A \oplus B$$



### XNOR

$$F = \overline{A \oplus B}$$

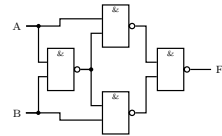


A	B	C	D	E	F
0	0	1	1	0	1
0	1	1	0	1	0
1	0	1	0	1	0
1	1	0	0	0	1

$$XOR = (A \wedge \overline{B}) \vee (\overline{A} \wedge B)$$

$$XNOR = (A \wedge B) \vee (\overline{A} \wedge \overline{B})$$

### XOR aus NAND



**XOR aus NOR:** Gleiches Schema wie NAND + 1 Inverter

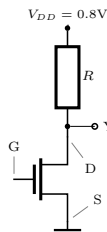
**XNOR aus NAND:** Gleiches Schema wie XOR aus NOR

**XNOR aus NOR:** Gleiches Schema wie XOR aus NAND

Es versteht sich natürlich, dass wenn von „Gleichem Schema wie...“ gesprochen wird, die Gates trotzdem getauscht werden müssen.

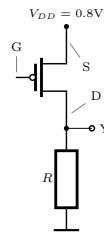
## CMOS

### NMOS



G	Schalter	Y
0	offen	1
1	zu	0

### PMOS



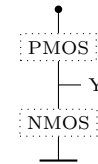
G	Schalter	Y
0	zu	1
1	offen	0

### Konstruktion von CMOS-Gates

Regeln für CMOS-Schaltungen

1. CMOS-Gates bestehen aus gleich vielen NMOS und PMOS.
2.  $m$  Eingänge:  $m$  NMOS und  $m$  PMOS.
3. NMOS in Serie  $\rightarrow$  PMOS parallel
4. NMOS parallel  $\rightarrow$  PMOS Serie

## Allg. Aufbau CMOS

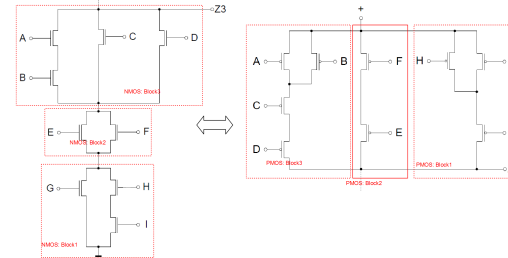


Pull-up: PMOS  
Pull-down: NMOS

Pfade sind komplementär  
(Serie  $\Leftrightarrow$  Parallel)

### Umwandlung Pull-up zu Pull-down

1. Teilbereiche (Blöcke) identifizieren.
2. Schritt 1 wiederholen, bis nur noch einzelne Transistoren vorkommen.
3. Falls Pull-down:
  - Von GND aus mit äusserstem Block beginnen.
  - PMOS  $\rightarrow$  NMOS
4. Falls Pull-up:
  - Von  $V_{DD}$  aus mit äusserstem Block beginnen.
  - NMOS  $\rightarrow$  PMOS.



### Funktionsgleichung

parallel:  $\vee$  | Pull-Up:  $y = 1$  | alle I : 0  $\rightarrow$  I invert.  
Serie:  $\wedge$  | Pull-Down:  $y = 0$  | alle I : 1  $\rightarrow$  Gl. invert.

## Boolesche Algebra

### Grundregeln

### Kommutativität

$$A \wedge B = B \wedge A$$

$$A \vee B = B \vee A$$

### Assoziativität

$$A \wedge (B \wedge C) = (A \wedge B) \wedge C$$

$$A \vee (B \vee C) = (A \vee B) \vee C$$

### Distributivität

$$(A \wedge B) \vee (A \wedge C) = A \wedge (B \vee C)$$

$$(A \vee B) \wedge (A \vee C) = A \vee (B \wedge C)$$

$$\text{Nicht} \quad \overline{\overline{A}} = A$$

$$\text{Null-Th.} \quad A \vee 0 = A \quad A \wedge 0 = 0$$

$$\text{Eins-Th.} \quad A \vee 1 = 1 \quad A \wedge 1 = A$$

$$\text{Idempotenz} \quad A \vee A = A \quad A \wedge A = A$$

$$\text{V. Komp.} \quad A \vee \overline{A} = 1 \quad A \wedge \overline{A} = 0$$

$$\text{Adsorp.} \quad A \vee (\overline{A} \wedge B) = A \vee B$$

$$A \wedge (\overline{A} \vee B) = A \wedge B$$

$$\text{Adsorp.} \quad A \vee (A \wedge B) = A$$

$$A \wedge (A \vee B) = A$$

$$\text{Nachbar.G.} \quad (A \wedge B) \vee (\overline{A} \wedge B) = B$$

$$(A \vee B) \wedge (\overline{A} \vee B) = B$$

## De Morgan

$$1. \text{ Regel} \quad \overline{A \wedge B} = \overline{A} \vee \overline{B}$$

$$2. \text{ Regel} \quad \overline{A \vee B} = \overline{A} \wedge \overline{B}$$

Regeln gelten auch für  $n$  verknüpfte Terme.

## Normalformen

Minterm	Maxterm
AND-Ausdruck	OR-Ausdruck
Output: 1	Output: 0
$n$ Schaltvar. $\rightarrow 2^n$ mögl. Minterme.	$n$ Schaltvar. $\rightarrow 2^n$ mögl. Maxterme.
nicht-invertierte Var: 1	nicht-invertierte Var: 0
invertierte Var: 0	invertierte Var: 0

### Disjunktive Normalform

1. Identifiziere WT-Zeilen mit Output 1
2. **Minterme** für diese Zeilen aufstellen
3. Minterme mit **OR** verknüpfen

### Konjunktive Normalform

1. Identifiziere WT-Zeilen mit Output 0
2. **Maxterme** für diese Zeilen aufstellen
3. Maxterme mit **AND** verknüpfen

A	B	Y	Minterme	Maxterme
0	0	1	$A \wedge \overline{B}$	
0	1	0		$A \vee \overline{B}$
1	0	0		$\overline{A} \vee B$
1	1	1	$A \wedge B$	

$$\text{DNF} \quad Y = (\overline{A} \wedge \overline{B}) \vee (A \wedge B) \quad 1 \text{ Mint. erf.} \rightarrow 1$$

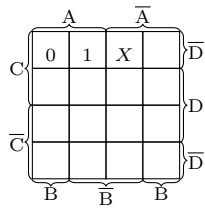
$$\text{KNF} \quad Y = (A \vee \overline{B}) \wedge (\overline{A} \vee B) \quad 1 \text{ Maxt. erf.} \rightarrow 0$$

### NAND/NOR Schaltungen

Schaltung nur aus:

- NOR: KNF  $\rightarrow 2 \times$  Negieren  $\rightarrow 1 \times$  De Morgan
- NAND: DNF  $\rightarrow 2 \times$  Negieren  $\rightarrow 1 \times$  De Morgan

## Karnaugh Diagramme (KVD)

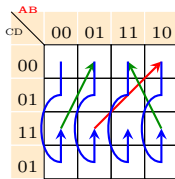
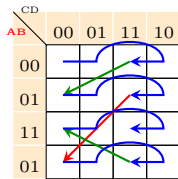


CD \ AB	00	01	11	10
00	0	1	X	
01				
11				
01				

Hat das Karnaugh Diagramm 5 Dimensionen, wird die 5te Dimension auf zwei Tabellen aufgeteilt.

**Don't-Care-Zustände**  $X \in \{0, 1\}$  Redundante, überflüssige oder unmögliche Kombinationen der Eingangsvariablen werden mit einem  $X$  markiert.

### Schema zum Ausfüllen



### Päckchen

- Päckchen immer rechteckig (Ausnahme: über Ecken).
- Umfassen möglichst grosse Zweierpotenz.
- Dürfen über Ecken und Grenzen hinausgehen und sich überlappen.

### DNF

- KVD ausfüllen.
- Päckchen mit **1** uo  $X$ .
- Vereinfachte Minterme aufstellen.
- Minterme mit OR verbinden.

### KNF

- KVD ausfüllen.
- Päckchen mit **0** uo  $X$ .
- Vereinfachte Maxterme aufstellen.
- Maxterme mit AND verbinden.

### Hazard

Kurzzeitige, unerwünschte Änderung der Signalwerte, die durch Zeitverzögerung der Gatter entstehen.

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	1	0	0
01	1	1	0	0

**Statische Hazards** Stellen im KVD, an denen sich Päckchen orthogonal berühren, aber nicht überlappen.  
**Lösung** Berührende **Päckchen** mit zusätzlichen (möglichst grossen) **Päckchen** verbinden.

## Zahlensysteme

$D$  zu berechnende positive Zahl  
 $R$  Basis/Radix von  $D$   
 $b_i$  Koeffizient

$$D = \sum_{i=0}^{\infty} b_i \cdot R^i$$

Darstellung  $D$  in Basis  $R$ :  $\dots b_2 b_1 b_0 . b_{-1} b_{-2} \dots R$

Dezimal	10	$b_i \in \{0, 1, \dots, 9\}$
Dual/Binär	2	$b_i \in \{0, 1\}$
Oktal	8	$b_i \in \{0, 1, \dots, 7\}$
Hexa	16	$b_i \in \{0, 1, \dots, 9, A, B, C, D, E, F\}$

### Umwandlung Zahlensysteme

- Ganzzahlige Division mit  $R$ :  $D/R = Q_0 + r_0$ .
- $Q_i/R = Q_{i+1} + r_{i+1}$   
bis  $Q_i = 0$ .
- Erste Operation gibt MSB, letzte Operation gibt LSB (aka. unten nach oben lesen.)

Für  $1 > D \geq 0$

$$D \cdot R = P_0 \quad K_{-1} = \text{floor}(P_0) \quad a_{-1} = P_0 - K_{-1}$$

$$a_{-1} \cdot R = P_{-1} \dots$$

$K_i$ : Koeffizienten für Zahlensystem. Erste Operation gibt **MSB**, letzte Operation gibt **LSB** (aka von oben nach unten lesen).

### Byte

$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
128	64	32	16	8	4	2	1

$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$
0.5	0.25	0.125	0.0625

### Binär zu Hex

0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

### Zweierkomplement

Sign Bit  
0: positiv 1: negativ

### Konstruktion

- Zahl  $|Z|$  in Binär  $B$  umwandeln.
  - $B$  bitweise invertieren
  - 1 zu LSB addieren (! Übertrag)
  - Sign Bit hinzufügen (zuvorderst).
- Ist die Blocklänge länger als Zahl, vorangehende 0(-en) miteinbeziehen.

### 2<sup>er</sup> Komplement zu Dezimal

$$D_{(10)} = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

Wertebereich 2<sup>er</sup>-Komp.  $[-2^{n-1}, 2^{n-1} - 1]$

## mQn

$$D_{(10)} = -b_m \cdot 2^m + \sum_{i=0}^{m-1} b_i \cdot 2^i + \sum_{i=1}^n b_i \cdot 2^{-i}$$

$m$ : Vorkommabits,  $n$ : Nachkommabits  
Sign-Bit muss nur einmal vor dem  $m$  codiert werden.

### Binäre Rechenoperationen

#### Addition

Bitweise Addition der Binärzahlen. Leere Slots werden mit 0 aufgefüllt.

#### Subtraktion

Addition via 2<sup>er</sup> Komp. Übertrag von MSB ignorieren.

### Multiplikation

- Bitweise Multiplikation des Multiplikanden  $a$  mit  $b_i$  des Multiplikators.
- Sukzessive Multiplikationen werden um ein Bit (0) nach links verschoben.
- Anzahl Nachkommabits ergibt sich aus der Summe der Anzahl Nachk.bits der Operatoren.

$$\begin{array}{r} b_0 \cdot a \\ + b_1 \cdot a \cdot 0 \\ + b_2 \cdot a \cdot 00 \\ + b_3 \cdot a \cdot 000 \\ \hline = \text{Sum} \end{array}$$

### Division

- Identifiziere Teil des Divident  $>$  Divisor (Unterblock). Für jede Stelle, sodass Divident  $<$  Divisor, 0 in Quotient.
- Unterblock – Divisor, 1 an Quotient anhängen, Rest behalten.
- An das Resultat der Subtraktion Bits des Dividenten anhängen. Wiederholen bis Subtraktion 0 ergibt.

## Parity-Bits

Hilft Bit-Fehler zu finden.  
Bitsequenz wird in 4 Bits unterteilt. Pro Nibble wird ein **Parity-Bit** angefügt. Nach 4 Blöcken folgt ein **Prüfwort**.

Parity-Bit	Anz. 1	PB	Nibble + PB
Even $P_E$	ungerade	1	gerade
	gerade	0	
Odd $P_O$	ungerade	0	ungerade
	gerade	1	

01010 11011 10111 00101 00011

### Korrekt $P_E$

0	1	0	1	0
1	1	0	1	1
1	0	1	1	1
0	0	1	0	1
0	0	0	1	1

### Fehler $P_E$

0	1	0	1	0
1	1	1	1	1
1	0	1	1	1
0	0	1	0	1
0	0	0	1	1

## Latches und FlipFlops

**Kombinatorische Schaltung**  
Output hängt von Inputs und Verknüpfungen ab.

**Sequentielle Schaltung**  
Enthält Rückkopplungen, Outputs hängen von vorherigen Werten ab.

## Latch

**(Takt)zustandgesteuerte Schaltung** → Änderungen am Eingang können während der **ganzen aktiven Taktphase** den Output beeinflussen.

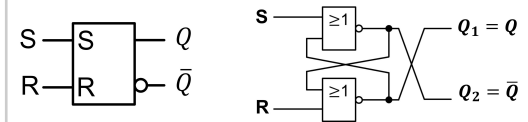
## FlipFlops

**Taktflankengesteuerte Schaltung** → Input zum Zeitpunkt der Taktwechsels wird wirksam.

### Latches

Alle taktzustandgesteuerte Schaltungen sind gegenüber **Störimpulsen** empfindlich, da bei  $T = 1$  jede Änderung übernommen wird.

### SR-Latch

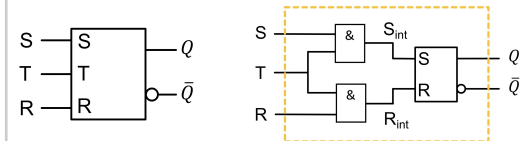


**S** Set → setzt  $Q$  auf 1  
**R** Reset → setzt  $Q$  auf 0

$$Q_{n+1} = S \vee (Q_n \wedge \bar{R})$$

Fall	S	R	$Q_{n+1}$	
1	0	0	$Q_n$	speichern
2	0	1	0	zurücksetzen
3	1	0	1	setzen
4	1	1	-	unzulässig

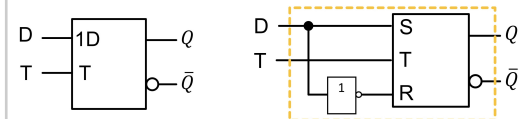
### SRT-Latch



**T**  $S_{int}$   $R_{int}$   
0 → 0 0 → Datenspeicherung  
1 → S R → Normales SR-Latch

Änderungen werden nur übernommen, wenn T/CLK aktiv ist.

### D-Latch



Bauelement, das Daten für die Periodendauer eines Taktes speichern kann.

$$Q_{n+1} = (Q_n \wedge \bar{T}) \vee (D \wedge T)$$

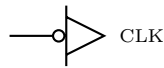
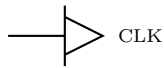
**T**  $Q_{n+1}$   
0  $Q_n$  → alter Ausgang gespeichert  
1 D → Input übernommen

D-Latch transparent

letzter Zustand gespeichert



## FlipFlops



Input beim Übergang von 0 → 1 von CLK wirksam.

Input beim Übergang von 1 → 0 von CLK wirksam.

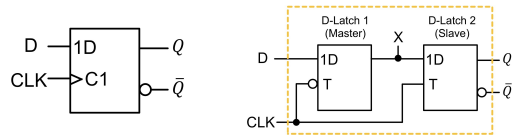


Positive/steigende Taktflanke



Negative/fallende Taktflanke

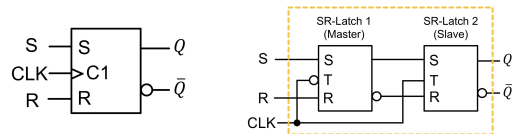
## D-FlipFlop



$Q_{n+1} = D$  wenn CLK 0 → 1

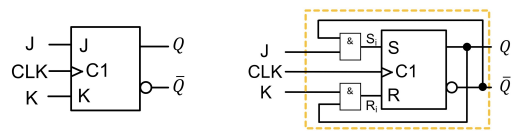
Master low-active CLK = 0  
Slave high-active CLK = 1

## SR-FlipFlop



$Q_{n+1} = S \vee (\bar{R} \wedge Q_n)$  wenn CLK 0 → 1

## JK-FlipFlop



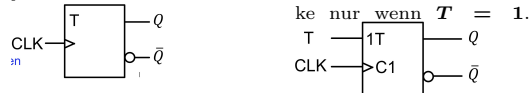
$Q_{n+1} = (J \wedge \bar{Q}_n) \vee (\bar{K} \wedge Q_n)$  wenn CLK 0 → 1

Fall	J	K	$Q_{1n+1}$	$Q_{2n+1}$	
1	0	0	$Q_{1n}$	$Q_{2n}$	speichern
2	0	1	0	1	zurücksetzen
3	1	0	1	0	setzen
4	1	1	$\bar{Q}_{1n}$	$\bar{Q}_{2n}$	wechseln

Bei J = K = 1 wechselt Output. (toggle)

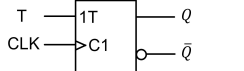
## T-FlipFlop

**V1** Ausgang wechselt bei jeder aktiven Taktflanke.



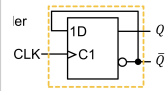
$Q_{n+1} = \bar{Q}_n$   
wenn CLK 0 → 1

**V2** Ausgang wechselt bei aktiver Taktflanke nur wenn T = 1.

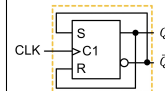


$Q_{n+1} = \bar{Q}_n$   
wenn CLK 0 → 1 ∧ T = 1

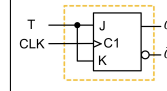
## BS V1



## BS V1

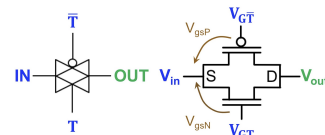


## BS V2



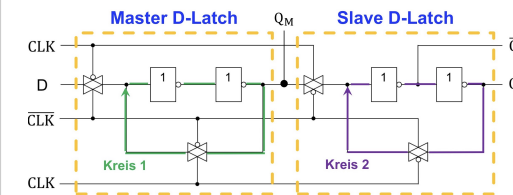
## D-FlipFlop in CMOS-Technik

### Transmission Gates

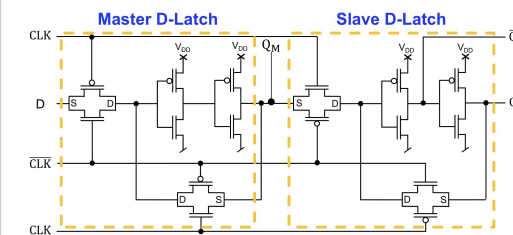


IN	T	Widerstand	OUT
0	0	hochohm.	-
0	1	niederohm.	0
1	0	hochohm.	-
1	1	niederohm.	1

TG sperrt wenn Widerstand hochohmig ist. (T = 0)



CLK 0 Input ins erste Latch übertragen  
CLK 1 Latch verriegelt, Wert im Kreis gefangen



## D-FlipFlop ⇔ JK-FlipFlop

1. JK-FF kann immer durch D-FF ersetzt werden.

D-FF:  $D_n = (J \wedge \bar{Q}_n) \vee (\bar{K} \wedge Q_n)$  :JK-FF

2. Ein D-FF kann nur durch JK-FF ersetzt werden wenn:  
a) Schaltung eine Rückkopplung enthält.  
b) Input D als  $(F_1 \wedge \bar{Q}_n) \vee (F_2 \wedge Q_n)$  geschrieben werden kann.

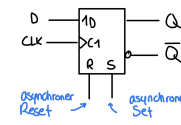
### Gleichung für D-FF → JK-FF

- Wahrheitstabelle mit Einängen und Rückkopplung.
- Wahrheitstabelle in  $Q_n$  und  $\bar{Q}_n$ .
- Separat Päckchen in  $Q_n$  und  $\bar{Q}_n$  machen.
- Päckchen mit OR verbinden. Ggf.  $Q_n$  und  $\bar{Q}_n$  ausklammern.

BC	00	01	11	10
00				
01				
11				
01				

## Asynchroner Set/Reset Input

Können gespeicherte Zustände asynchron zu CLK überschreiben.



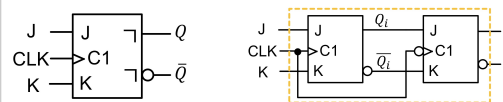
## Verzögerungszeiten

$t_s$	Setup-Zeit	Solange muss Signal <u>vor</u> aktiver Taktflanke stabil anliegen.
$t_h$	Hold-Zeit	Solange muss Signal <u>nach</u> aktiver Taktflanke stabil anliegen.
$t_{pd}$	Verzögerungszeit	Durchlaufzeit

$$T_{min} \geq t_{pd1} + t_{pd,ks} + t_{s2} \quad f_{max} = \frac{1}{T_{min}}$$

$t_h$  kann bei der Berechnung von  $f_{max}$  vernachlässigt werden.

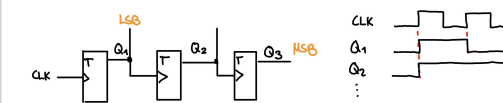
## Zwischenspeicher-FF



FlipFlop, dass Input bei steigender Taktflanke übernimmt und bei der nächsten fallenden Taktflanke ausgibt. (oder umgekehrte Flanken)

- ⌋ Ausgabe bei fallender Flanke
- ⌋ Ausgabe bei steigender Flanke

## Frequenzteiler und Zähler



Kaskadieren von T-FlipFlops führt zu einer Frequenzreduktion von CLK um Faktor 2.

Kann als Bitzähler verwendet werden (ohne CLK). MSB ist längste Frequenz.  $n_{T,ff} \rightarrow 0 \dots (2^n - 1)$

## Diverses

### Physikalische Zuordnung logischer Zustände

0	Low 0 V	Ground
1	High 0.8 V	VDD

Toleranzen:

- GND: 0 V ... 0.15 V
- VDD 0.7 V ... 0.9 V

### Schaltelemente

#### Multiplexer

Sendet eines von  $2^n$  Eingangssignalen an den Ausgang. Hat  $n$  Auswahlbits.

#### Demultiplexer

Sendet 1 Eingangssignal an einen von  $2^n$  Ausgängen.  $n$  Auswahlbits.

## Halbaddierer

Addiert 2 Binärzahlen A und B. Produziert Summe und Carry-Out.

$$SUM = A \oplus B \quad CO = A \wedge B$$

## Volladdierer

Nimmt einen zusätzlichen Input CI entgegen.

$$SUM = (A \oplus B) \oplus CI \quad CO = (A \wedge B) \vee (S_{AB} \wedge CI)$$

## Serienaddierer

Addition einer Stelle pro Taktschritt.

## Paralleladdierer (Normalform)

Addition aller Stellen pro Taktschritt.

### Vorteile

- Maximal 3 Grundgatter zwischen Input und Output.
- Laufzeit ist unabhängig von Stellenzahl der Summanden.

→ Schnell aber Schaltungsaufwendig

**Nachteile** Bei Addition von  $n$ -stelligen Summanden müssen  $\sim n \cdot 2^{2n-1}$  Min-/Maxterme verknüpft werden.

## Ripple-Carry Addierer (Paralleladdierer)

### Vorteile

- Durch Kaskadierung einfach skalierbar.
- Schaltungsaufwand linear zur Stellenzahl.

### Nachteile

- SUM und CO für die  $i$ -te Stelle können erst nach der Berechnung der  $(i-1)$ -ten Stelle gebildet werden.
- Addierzeit linear zu Stellenzahl

Langsamer als Normalformaddierer aber einfacher zu realisieren.

## Carry-Look-Ahead Addierer (Paralleladdierer)

Kombination der Vorteile des Normalform- und Ripple-Carry-Addierer → schnelle Schaltung mit begrenztem Aufwand.

**Praktische Realisierung** Addierer werden kaskadiert, Berechnung der Überträge erfolgt parallel zur Summenbildung.

Berechnungsaufwand ist linear zur Stellenzahl, Laufzeit bleibt konstant.

## Booth-Algorithmus

Dient der Multiplikation von Binärzahlen ( $A$  &  $B$ ). Berechnung über Zwischenprodukte  $P_i$ .

Division durch 2 bedeutet: Verschiebung des Kommas nach links (shift), mit Vorzeichenverdoppelung falls nötig.

$a_i$	$a_{i-1}$	Operation
0	0	$P_i = P_{i-1}/2$
0	1	$P_i = (P_{i-1} + B)/2$
1	0	$P_i = (P_{i-1} - B)/2$
1	1	$P_i = P_{i-1}/2$

Anfangswerte:  $P_{-1} = 0, a_{-1} = 0$   
Beim letzten Schritt entfällt die Division durch 2.