

Boolesche Funktionen

a, b, c, d, e seien boolesche Variable, denen vermöge der Abbildungsvorschriften

$$(a, b, c) \rightarrow f(a, b, c) \quad \text{bzw.} \quad (a, b) \rightarrow f(a, b)$$

der boolesche Funktionswert $f(a, b, c)$ bzw. $f(a, b)$ zugeordnet wird.

Mit $\hat{a}$ oder NOT a bezeichnen wir die Negation von a .

1. Negation

a	$\hat{a}$
0	1
1	0

2. AND (Konjunktion)

$$(a, b) \rightarrow a \text{ AND } b \quad (\text{Vereinbarung: } a \text{ AND } b = a \cdot b = ab)$$

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

3. OR (Disjunktion)

$$(a, b) \rightarrow a \text{ OR } b \quad (\text{Vereinbarung: } a \text{ OR } b = a + b)$$

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

4. XOR (eXclusive OR)

$(a, e) \rightarrow a \text{ XOR } e$ (Vereinbarung: $a \text{ XOR } e = a \oplus e$)

a	e	$a \oplus e$	
0	0	0	
0	1	1	←
1	0	1	←
1	1	0	

Disjunktion der Konjunktionen:

$$a \text{ XOR } e = 0 + \hat{a} \cdot e + a \cdot \hat{e} + 0 = \hat{a} \cdot e + a \cdot \hat{e}$$

5. Gegeben ist die Zuordnung $(a, b, c) \rightarrow f(a, b, c)$ mittels folgender Wertetabelle; ermittle einen möglichst einfachen Funktionsterm:

a	b	c	$f(a, b, c)$	
0	0	0	0	
0	0	1	0	
0	1	0	1	←
0	1	1	0	
1	0	0	1	←
1	0	1	0	
1	1	0	1	←
1	1	1	0	

Disjunktion der Konjunktionen:

$$\begin{aligned}
 f(a, b, c) &= \text{NOT}a \cdot b \cdot \text{NOT}c + a \cdot \text{NOT}b \cdot \text{NOT}c + a \cdot b \cdot \text{NOT}c \\
 &= [\text{NOT}a \cdot b + a \cdot \text{NOT}b + a \cdot b] \text{NOT}c \\
 &= [\text{NOT}a \cdot b + a \cdot (\text{NOT}b + b)] \text{NOT}c \\
 &= [\text{NOT}a \cdot b + a \cdot 1] \cdot \text{NOT}c \\
 &= [\text{NOT}a \cdot b + a] \cdot \text{NOT}c \\
 &= [a + \text{NOT}a \cdot b] \cdot \text{NOT}c \\
 &= [(a + \text{NOT}a) \cdot (a + b)] \cdot \text{NOT}c \\
 &= [1 \cdot (a+b)] \cdot \text{NOT}c \\
 &= (a+b) \cdot \text{NOT}c
 \end{aligned}$$

Hinweis:

$$a(b+c) = ab + ac$$

$$a + bc = (a+b) \cdot (a+c)$$

Theoreme zum Rechnen mit Booleschen Variablen

Voraussetzung: a, e, u seien Boolesche Variable, mit $\hat{a}, \hat{e}, \hat{u}$ werden die Negationen von a, e, u bezeichnet.

Kommutativgesetz

$$(1) \quad a \cdot e = e \cdot a$$

$$(1)' \quad a + e = e + a$$

Assoziativgesetz

$$(2) \quad a \cdot (e \cdot u) = (a \cdot e) \cdot u$$

$$(2)' \quad a + (e + u) = (a + e) + u$$

Distributivgesetz

$$(3) \quad a \cdot (e + u) = a \cdot e + a \cdot u$$

$$(3)' \quad a + e \cdot u = (a + e) \cdot (a + u)$$

Absorptionsgesetz

$$(4) \quad a \cdot (a + e) = a$$

$$(4)' \quad a + a \cdot e = a$$

Tautologie

$$(5) \quad a \cdot a = a$$

$$(5)' \quad a + a = a$$

Gesetz über die Negation

$$(6) \quad a \cdot \hat{a} = 0$$

$$(6)' \quad a + \hat{a} = 1$$

Doppelte Negation

$$(7) \quad \text{NOT}(\text{NOT } a) = a$$

De Morgans Gesetz

$$(8) \quad \text{NOT}(a \cdot e) = \text{NOT } a + \text{NOT } e$$

$$(8)' \quad \text{NOT}(a + e) = \text{NOT } a \cdot \text{NOT } e$$

Operationen mit 0 und 1

$$(9.1) \quad a \cdot 1 = a$$

$$(9.1)' \quad a + 0 = a$$

$$(9.2) \quad a \cdot 0 = 0$$

$$(9.2)' \quad a + 1 = 1$$

$$(9.3) \quad \text{NOT } 0 = 1$$

$$(9.3)' \quad \text{NOT } 1 = 0$$

Wir ermitteln für $\mathbf{s}_i$ und $\mathbf{c}_{i+1}$ jeweils die disjunktive Normalform („Disjunktion der Konjunktionen“) und vereinfachen ggf. die booleschen Funktionsterme:

$$s_i = \bar{a}_i \cdot \bar{b}_i \cdot c_i + \bar{a}_i \cdot b_i \cdot \bar{c}_i + a_i \cdot \bar{b}_i \cdot \bar{c}_i + a_i \cdot b_i \cdot c_i$$

ohne Index i geschrieben:

$$s = \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + a \cdot b \cdot c$$

$$s = (\bar{a} \cdot b + a \cdot \bar{b}) \cdot \bar{c} + \bar{a} \cdot \bar{b} \cdot c + a \cdot b \cdot c$$

$$s = (a \oplus b) \cdot \bar{c} + (\bar{a} \cdot \bar{b} + a \cdot b) \cdot c$$

$$s = (a \oplus b) \cdot \bar{c} + (0 + \bar{a} \cdot \bar{b} + a \cdot b + 0) \cdot c$$

$$s = (a \oplus b) \cdot \bar{c} + (\bar{a} \cdot a + \bar{a} \cdot \bar{b} + a \cdot b + b \cdot \bar{b}) \cdot c$$

$$s = (a \oplus b) \cdot \bar{c} + [(\bar{a} + b) \cdot (a + \bar{b})] \cdot c$$

$$s = (a \oplus b) \cdot \bar{c} + [(\bar{a} + \bar{b}) \cdot (\bar{a} + \bar{b})] \cdot c$$

$$s = (a \oplus b) \cdot \bar{c} + [\overline{(a \cdot \bar{b})} \cdot \overline{(\bar{a} \cdot b)}] \cdot c$$

$$s = (a \oplus b) \cdot \bar{c} + \overline{[a \cdot \bar{b} + \bar{a} \cdot b]} \cdot c$$

$$s = (a \oplus b) \cdot \bar{c} + \overline{(a \oplus b)} \cdot c$$

$$s = (a \oplus b) \oplus c$$

mit Index i erhält man:

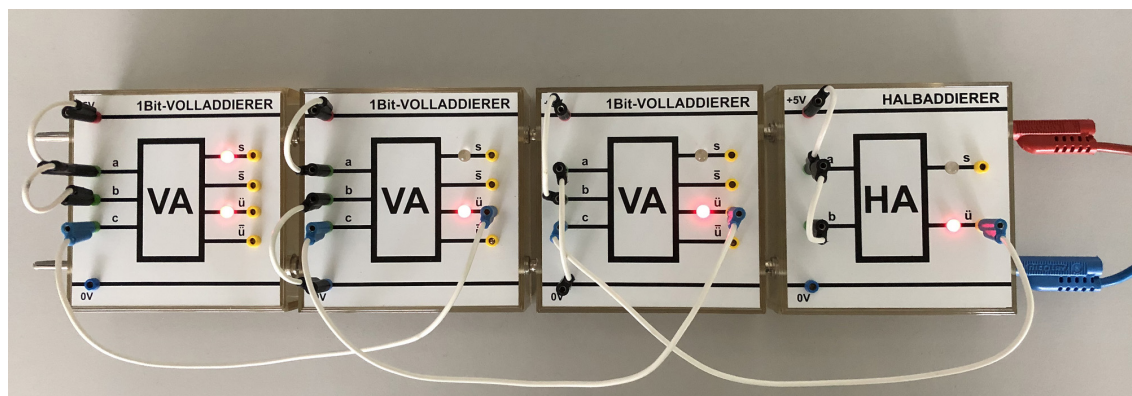
$$s_i = (a_i \oplus b_i) \oplus c_i$$

$$c_{i+1} = \bar{a}_i \cdot b_i \cdot c_i + a_i \cdot \bar{b}_i \cdot c_i + a_i \cdot b_i \cdot \bar{c}_i + a_i \cdot b_i \cdot c_i$$

$$c_{i+1} = (\bar{a}_i \cdot b_i + a_i \cdot \bar{b}_i) \cdot c_i + a_i \cdot b_i \cdot (\bar{c}_i + c_i)$$

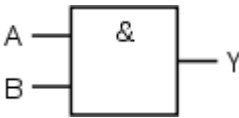
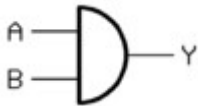
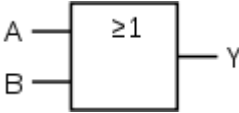
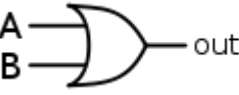
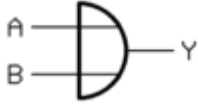
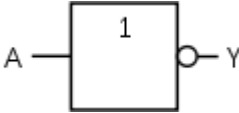
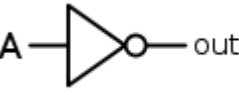
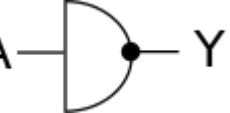
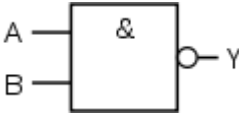
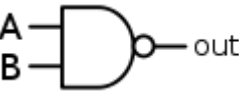
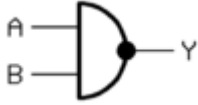
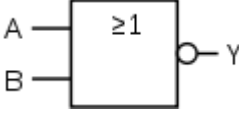
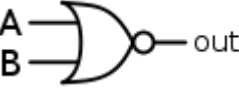
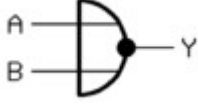
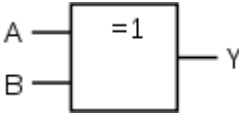
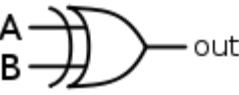
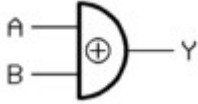
$$c_{i+1} = (\bar{a}_i \cdot b_i + a_i \cdot \bar{b}_i) \cdot c_i + a_i \cdot b_i \cdot 1$$

$$c_{i+1} = (a_i \oplus b_i) \cdot c_i + a_i \cdot b_i$$



Typen von Logikgattern und Symbolik

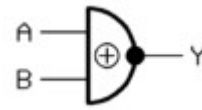
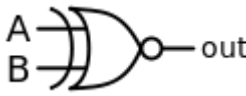
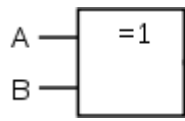
Logikgatter werden mit Schaltsymbolen bezeichnet, die nach unterschiedlichen, mehr oder weniger parallel existierenden Standards definiert sind.

Name	Funktion	Symbol in Schaltplan			Wahrheits-tabelle
		IEC 60617-12 : 1997 & ANSI/IEEE Std 91/91a-1991	ANSI/IEEE Std 91/91a-1991	DIN 40700 (vor 1976)	
Und-Gatter (AND)	$Y=A \cdot B$				A B Y 0 0 0 0 1 0 1 0 0 1 1 1
Oder-Gatter (OR)	$Y=A+B$				A B Y 0 0 0 0 1 1 1 0 1 1 1 1
Nicht-Gatter (NOT)	$Y=\overline{A}$				A Y 0 1 1 0
NAND-Gatter (NICHT UND) (NOT AND)	$Y=\overline{A \cdot B}$				A B Y 0 0 1 0 1 1 1 0 1 1 1 0
NOR-Gatter (NICHT ODER) (NOT OR)	$Y=\overline{A+B}$				A B Y 0 0 1 0 1 0 1 0 0 1 1 0
XOR-Gatter (Exklusiv- ODER, Antivalenz) (eXclusiveOR)	$Y=A \oplus B$				A B Y 0 0 0 0 1 1 1 0 1 1 1 0

XNOR-Gatter

(Exklusiv-Nicht-ODER, Äquivalenz)
(eXclusive Not OR)

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



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

Früher waren auf dem europäischen Kontinent die deutschen Symbole (rechte Spalte) verbreitet; im englischen Sprachraum waren und sind die amerikanischen Symbole (mittlere Spalte) üblich. Die IEC-Symbole sind international auf beschränkte Akzeptanz gestoßen und werden in der amerikanischen Literatur (fast) durchgängig ignoriert.

JK-Flipflop

Ein Flip-Flop (bistabile Kippstufe oder bistabiler Multivibrator) hat zwei stabile Zustände am Ausgang Q; die Zustände heißen „gesetzt“ (set) oder „zurückgesetzt“ (reset). Ein 1-Bit-Speicher lässt sich somit als FlipFlop realisieren.

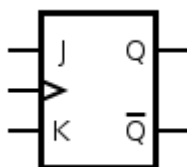
Ein JK-FlipFlop ist ein taktgesteuertes FlipFlop: die an den Eingängen J und K liegende Information wird mit einer Flanke (hier: der steigenden Flanke) des an C liegenden Taktsignals auf die Ausgänge Q und $\bar{Q}$ übernommen.

Mit dem Taktsignal (clock, C) und der Eingangsbelegung J = 1 und K = 0 wird am Ausgang Q eine 1 erzeugt und gespeichert, alternativ eine 0 bei J = 0 und K = 1.

Bei der Realisierung des JK-Flipflops als taktflankengesteuertes Flipflop kann der Eingang C für steigende Flanken (Wechsel von 0 auf 1) oder für fallende Flanken (Wechsel von 1 auf 0) ausgelegt sein.

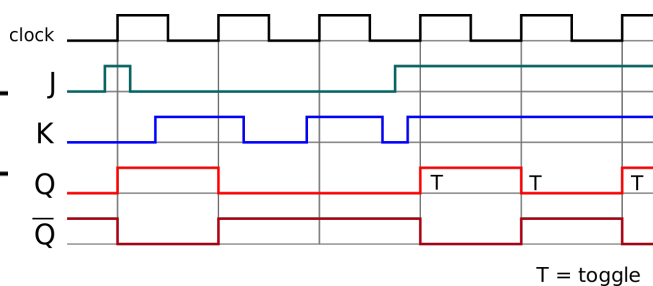
Name und Schaltzeichen

Flanken-gesteuertes JK-Flipflop



Signal-Zeit-Diagramm

Übernahme der Eingangsinformation durch steigende Flanke an C (clock)



T = toggle

Funktionstabelle

bis zur ... n-ten Taktflanke		nach der Taktflanke
J	K	Q_n
0	0	Q_{n-1} (unverändert)
0	1	0 (zurückgesetzt)
1	0	1 (gesetzt)
1	1	NOT Q_{n-1} (gewechselt)

(Wikipedia)

Halbaddierer (HA) und Volladdierer (VA)

Schaltungen

I. Volladdierer

Addition zweier Binärzahlen:

$$\begin{array}{r} a_3 a_2 a_1 a_0 \\ + b_3 b_2 b_1 b_0 \\ \hline s_4 s_3 s_2 s_1 s_0 \end{array}$$

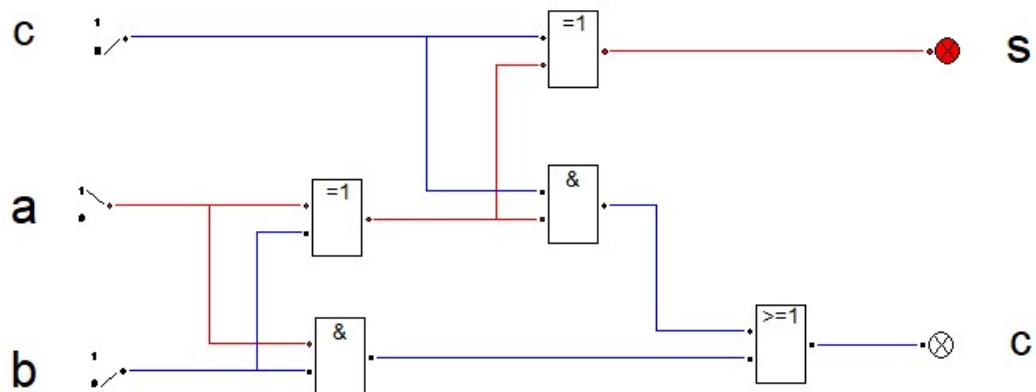
Boolesche Funktionen für den Volladdierer:

$$s_i = (a_i \oplus b_i) \oplus c_i$$

$$c_{i+1} = a_i b_i + (a_i \oplus b_i) c_i$$

Herleitung: Übungsaufgabe!
(Wahrheitstafel)

Schaltung:



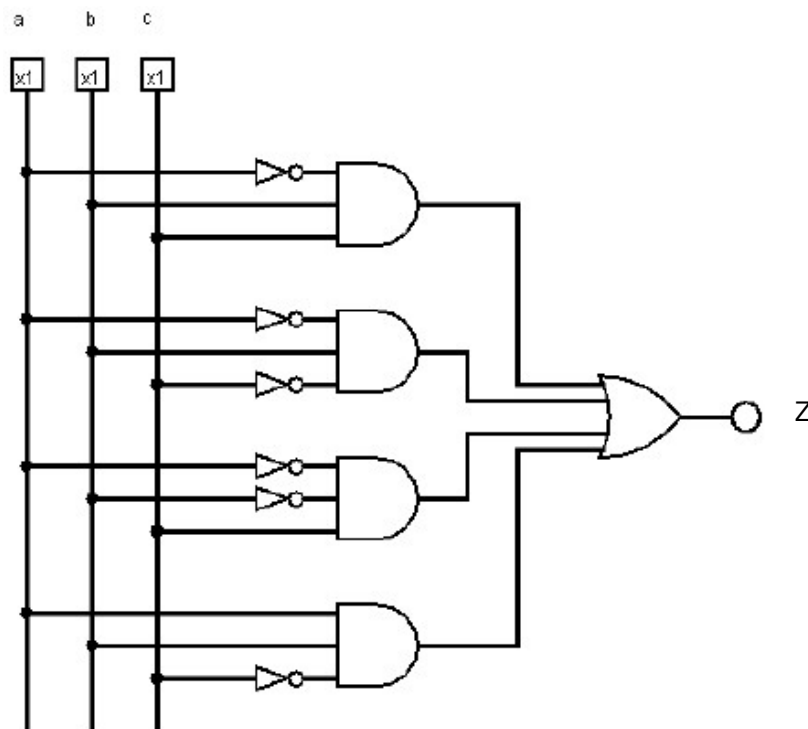
Informatik 13
Übungsaufgabe 07.10.2020

Gegeben ist die boolesche Funktion $z = f(a, b, c)$ vermöge folgender Wahrheitstafel:

a	b	c	z
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- a) Leite den Booleschen Funktionsterm für die Funktion f her.
- b) Vereinfache diesen Term mit Hilfe der Rechenregeln für Boolesche Terme.

1. Gegeben ist folgende digitale Schaltung mit den Eingängen a, b, c und dem Ausgang z:

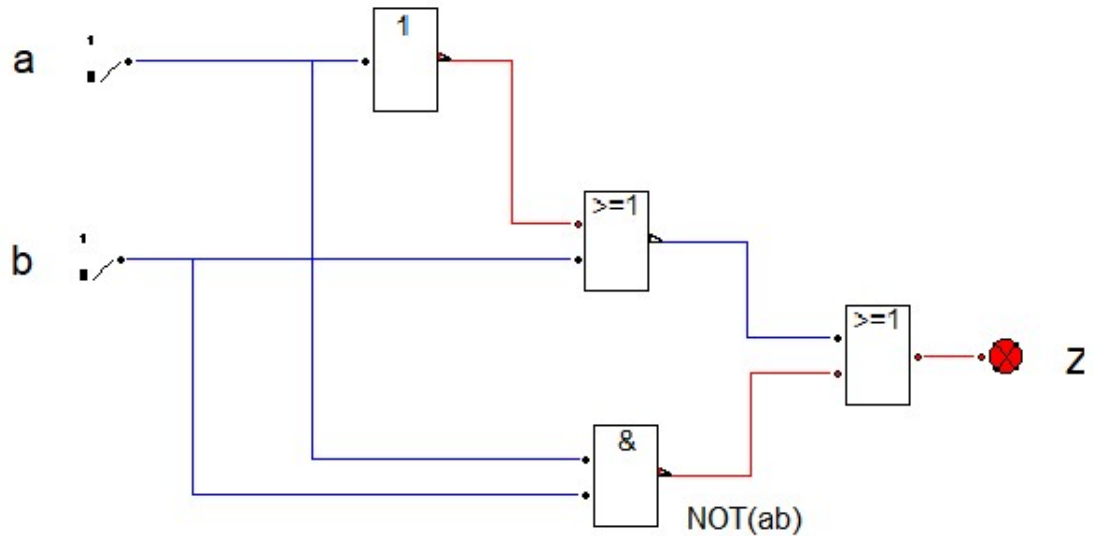


- a) Erstelle die Wahrheitstafel für diese Schaltung und ermittle die disjunktive Normalform für die Boolesche Funktion $z = f(a,b,c)$.
- b) Vereinfache den Funktionsterm für z und zeichne die vereinfachte Schaltung.
2. Die Boolesche Funktion $z = f(a,b,c)$ ist durch folgende Wahrheitstafel gegeben:

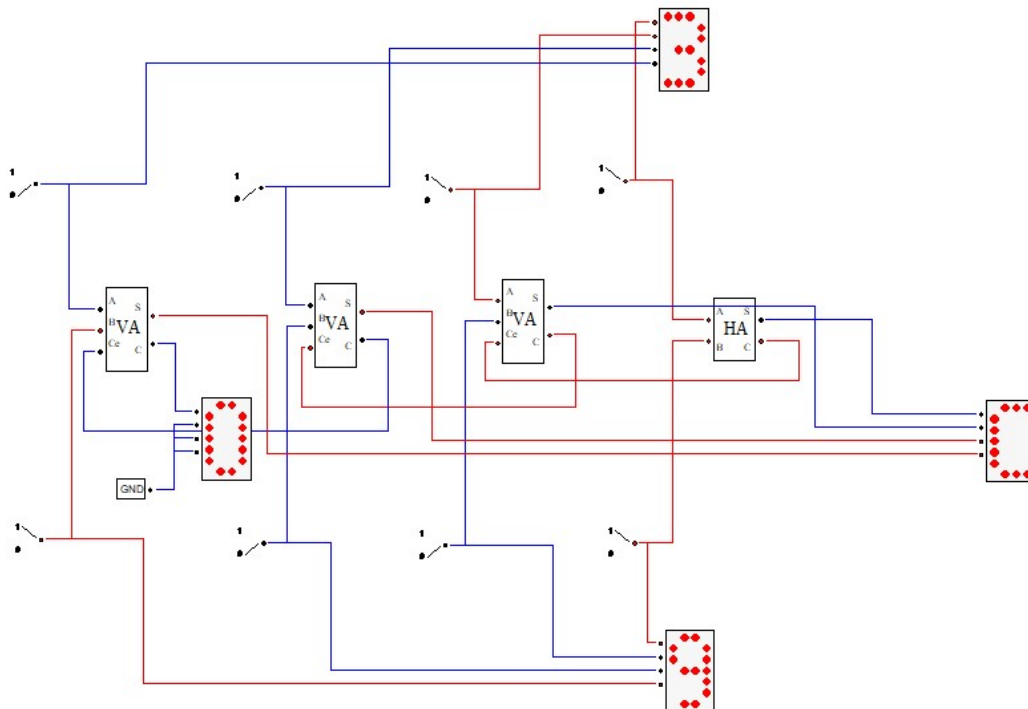
a	b	c	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- a) Ermittle die disjunktive Normalform für z und vereinfache den Funktionsterm.
- b) Zeichne den Schaltplan für die optimierte Funktion z.

3. Gegeben ist folgende digitale Schaltung mit den Eingangsvariablen a, b und der Ausgangsvariablen z:

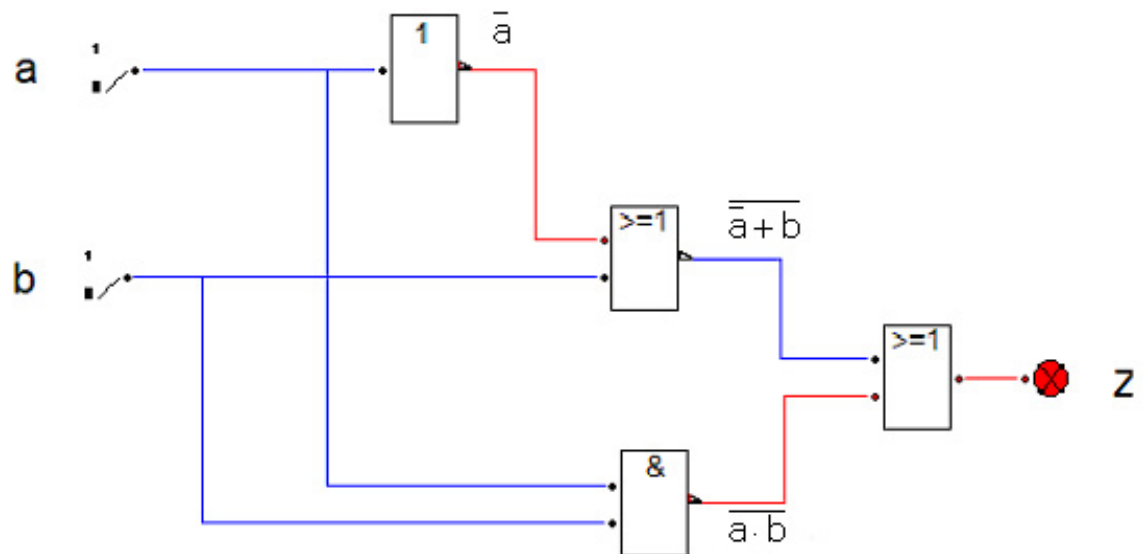


- Ermittle den Booleschen Term für die Boolesche Funktion $z = f(a,b,c)$. Hinweis: Notiere am Ausgang jedes Gatters jeweils den Booleschen Term (Beispiel: $\overline{a \cdot b}$ am Ausgang des NAND-Gatters).
 - Vereinfache den in a) erhaltenen Term unter Verwendung der Rechenregeln für Boolesche Ausdrücke; erstelle die Wahrheitstafel.
 - Zeichne das Schaltbild für den vereinfachten Funktionsterm und teste beide Schaltungsvarianten mit einem Digitalsimulator.
4. 4-Bit-Paralleladdierer mit Anzeige der Summanden und der Summe jeweils im Hexadezimalformat



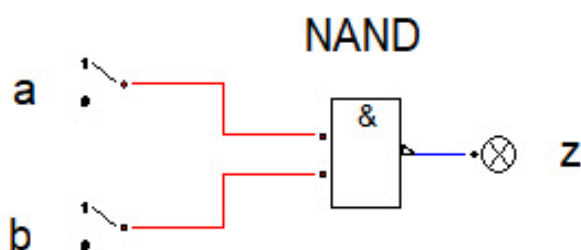
Erweitere die Schaltung „4-bit-Paralleladdierer.dsim“ (auf www.kalle2000.de downloadbar) zu einem 8-Bit-Addierer mit numerischer Anzeige.

Lösung zu Nr. 3 von Aufgabenblatt 2 vom 29.10.2020



$$\begin{aligned}
 z &= \overline{a+b} + \overline{a \cdot b} \\
 &= \overline{a} \cdot \overline{b} + (\overline{a} + \overline{b}) && \text{(2-mal de Morgan)} \\
 &= a \cdot \overline{b} + \overline{a} + \overline{b} && \text{(wegen } \overline{\overline{a}} = a \text{)} \\
 &= \overline{a} + \overline{b} \cdot a + \overline{b} && \text{(Kommutativgesetze)} \\
 &= \overline{a} + \overline{b} \cdot a + \overline{b} \cdot 1 && \text{(wegen } a = a \cdot 1 \text{)} \\
 &= \overline{a} + \overline{b} \cdot (a + 1) && \text{(Distributivgesetz)} \\
 &= \overline{a} + \overline{b} && \text{(wegen } a + 1 = 1 \text{)} \\
 &= \overline{a \cdot b} && \text{(de Morgan)}
 \end{aligned}$$

optimierte Schaltung:



Wertetabelle:

a	b	z
0	0	1
0	1	1
1	0	1
1	1	0

5. Für die Boolesche Funktion $y = f(a,b,c)$ ist folgende Wertetafel gegeben:

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

- Ermittle die DNF (disjunktive Normalform) für y .
- Vereinfache den Funktionsterm unter Verwendung der Booleschen Rechenregeln.
- Zeichne das Schaltbild für die vereinfachte Funktion.

Die Boolesche Funktion
 $z = f(a,b,c)$
 ist durch nebenstehende
 Wahrheitstafel
 gegeben:

a	b	c	z
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

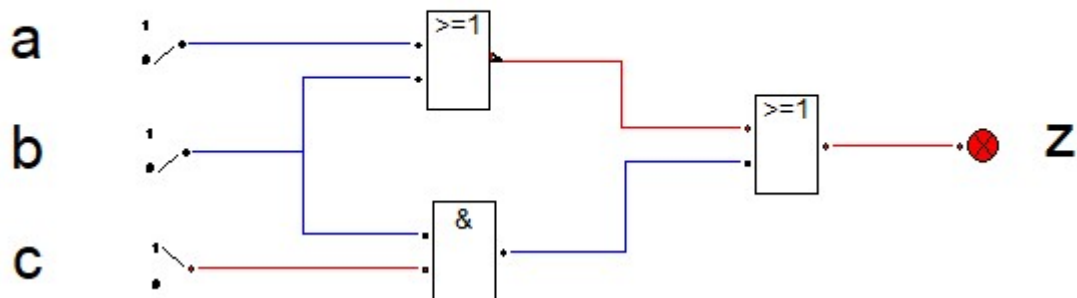
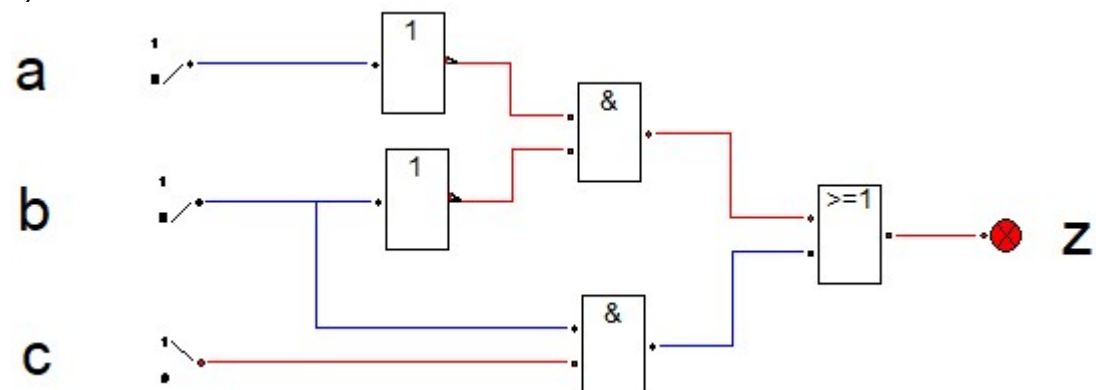
- a) Ermittle die disjunktive Normalform für z.
 b) Vereinfache den Funktionsterm unter Anwendung der Booleschen Rechengesetze.
 c) Zeichne den Schaltplan für die optimierte Funktion z.

Lösung:

a) $z = \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot c + a \cdot b \cdot c$

b) $z = \bar{a} \cdot \bar{b} \cdot (\bar{c} + c) + b \cdot c \cdot (\bar{a} + a)$ *Kommutativ- und Distributivgesetz*
 $= \bar{a} \cdot \bar{b} \cdot 1 + b \cdot c \cdot 1$
 $= \bar{a} \cdot \bar{b} + b \cdot c$
 $= \overline{a+b} + b \cdot c$

c)



Die wesentlichen Komponenten einer **CENTRAL PROCESSING UNIT (CPU)** bestehen aus der **CONTROL UNIT (CU)** und der **ARITHMETIC LOGIC UNIT (ALU)**.

Die **ALU** berechnet arithmetische und logische Funktionen, die **CU** decodiert die im Arbeitsspeicher abgelegten Befehle und führt sie aus.

In der Minimalkonfiguration beherrscht die **ALU** die arithmetische Funktion „**Addition**“ sowie die logischen Operationen „**Negation**“ (NOT) und „**Konjunktion**“ (AND). Zu Lasten der Rechenzeit lassen sich die übrigen arithmetischen und logischen Funktionen auf die genannten, minimal verfügbaren Operationen zurückführen.

1. Subtraktion

Die duale Subtraktion

$$\begin{array}{r} \quad a_3 \quad a_2 \quad a_1 \quad a_0 \\ - \quad b_3 \quad b_2 \quad b_1 \quad b_0 \\ \hline d_3 \quad d_2 \quad d_1 \quad d_0 \end{array}$$

läßt sich auf eine duale Addition nach folgendem Verfahren zurückführen:

- Bilde das Einerkomplement des Subtrahenden $b_3 \ b_2 \ b_1 \ b_0$, indem man alle Ziffern negiert (invertiert; aus 0 wird 1 und aus 1 wird 0).
 - Addiere das Einerkomplement und die Zahl 1 zum Minuenden.
 - Das Ergebnis ist die gesuchte Differenz; dabei bleibt der Überlauf unberücksichtigt.
- a) Verdeutliche das genannte Verfahren anhand einiger selbst gewählter Beispiele (ein Beweis des Verfahrens ist nicht erforderlich.).
- b) Ergänze die Schaltung „4-bit-Paralleladdierer.dsim“ so, daß man nach entsprechender Umschaltung wahlweise eine duale Addition oder eine duale Subtraktion durchführen kann.
Hinweise:
- Ersetze den HA für das least significant bit (LSB) durch einen VA, um erforderlichenfalls eine „1“ als Summand einspeisen zu können (wie?).
 - Die Invertierung der Ziffern des Subtrahenden gelingt z. B. durch den geeigneten Einsatz von XOR-Gattern.

2. Weitere Rechenoperationen

Gegeben sind die (im einfachsten Fall positiven ganzzahligen) Operanden a und b. Um zu verdeutlichen, wie man die „höheren“ Rechenoperationen mittels geeigneter Iteration auf die Grundoperationen „Addition“ und „Subtraktion“ zurückführen kann, schreibe und teste ein Python-Programm, welches die Operationen „Multiplikation“ ($a*b$), „Division“ (a/b , ganzzahlige Division) und „Potenzierung“ ($a**b$) realisiert.

3. Logische Operationen

Zeige exemplarisch, daß sich die logischen Verknüpfungen

- a) $a + b$
- b) $a \oplus b$
- c) $a \cdot (b + \bar{c})$

auf die Operationen NOT und AND zurückführen lassen.

11.11.20

SCHIEBREGISTER und DUALZÄHLER

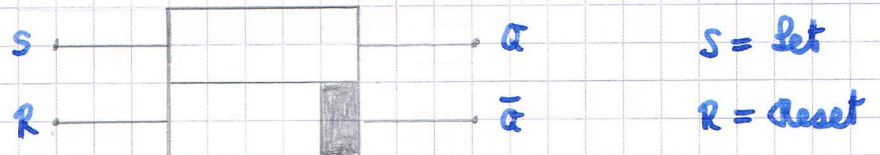
Definition: Unter einem Flip-Flop verstehen wir eine bistabile Tippstufe mit genau 2 stabilen Zustände, so dass der Informationsgehalt von 1 Bit gespeichert werden kann.

$1 \hat{=} \text{TRUE} \hat{=} 5 \text{ Volt}$

$0 \hat{=} \text{FALSE} \hat{=} 0 \text{ Volt}$

Schaltzeichen:

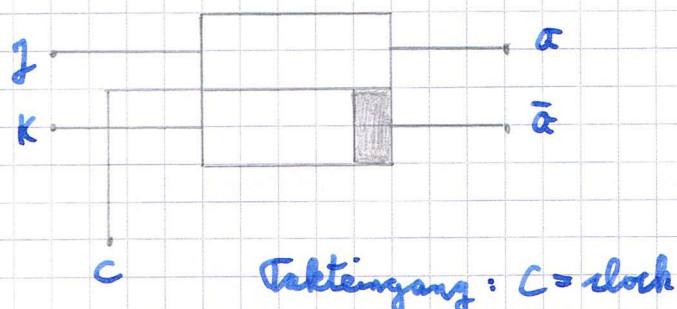
a) nicht getaktet:



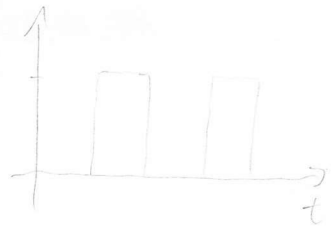
Zustandstabelle:

S	R	Ergesamand	
		Q	$\bar{Q}$
0	0	keine Änderung	
0	1	0	1
1	0	1	0
1	1	-	-

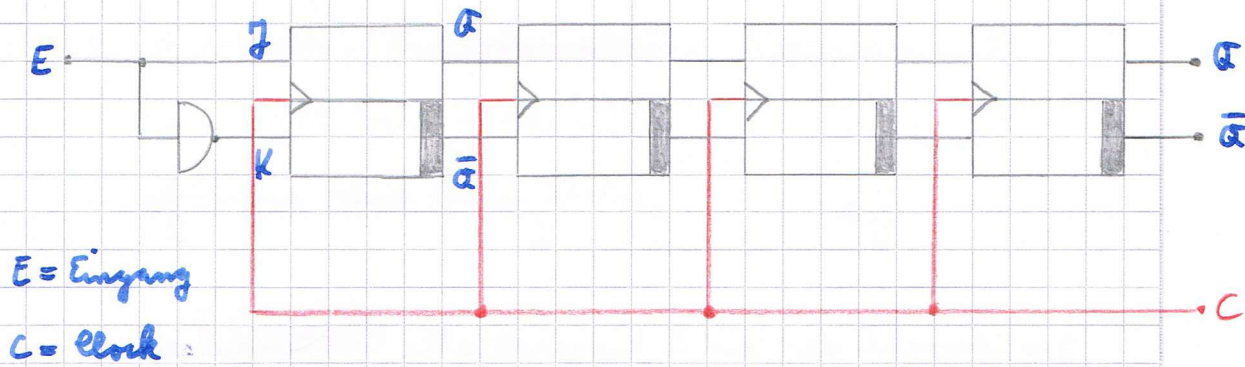
b) getaktet:



Die an den Eingängen J und K komplementär anliegende Information wird erst dann auf die Ausgänge Q und $\bar{Q}$ übertragen, wenn an C die fallende (oder auch die steigende) Taktflanke eines Rechtecksignals erscheint



SCHIEBEREGISTER (hier: 4-Bit-Register)



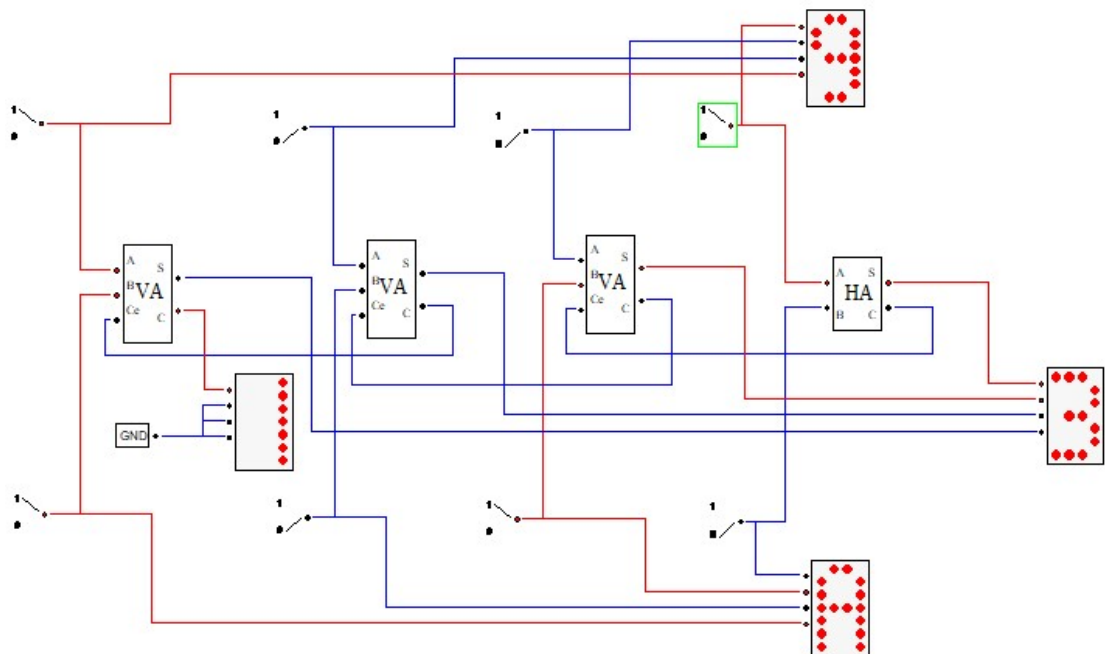
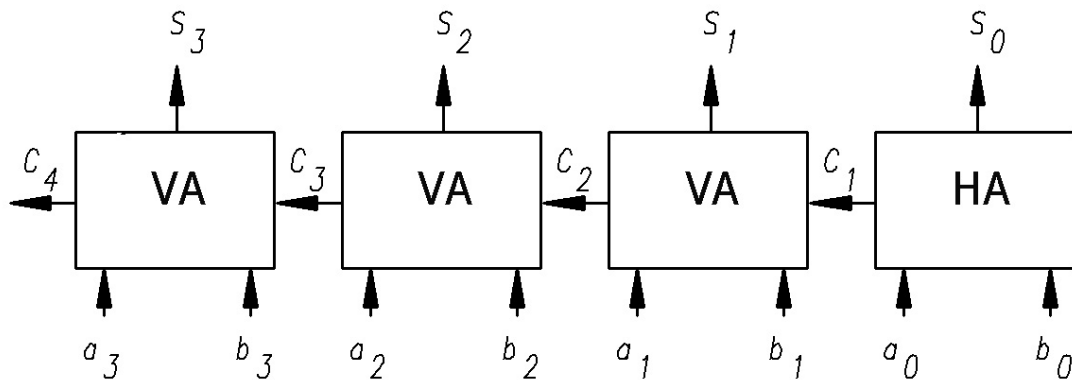
Bei jedem Taktimpuls an C wird die Information um eine Stelle nach „rechts“ geschoben.

Addier-Schaltungen für Dualzahlen

		a_3	a_2	a_1	a_0
+		b_3	b_2	b_1	b_0
	s_4	s_3	s_2	s_1	s_0

1. Paralleladdierer mit seriellem Übertrag (hier: 4-Bit-Addierer)

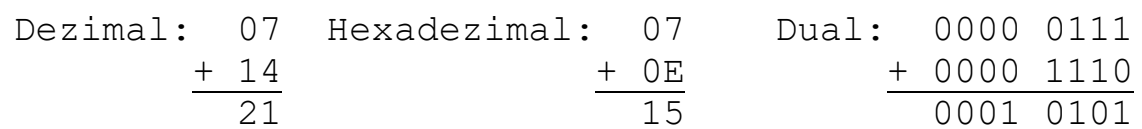
Für das Least Significant Bit (LSB) genügt ein Halbaddierer (HA); die höherwertigen Bits erfordern jeweils einen Volladdierer, da hier der Übertrag aus der vorherigen Stelle zu berücksichtigen ist.



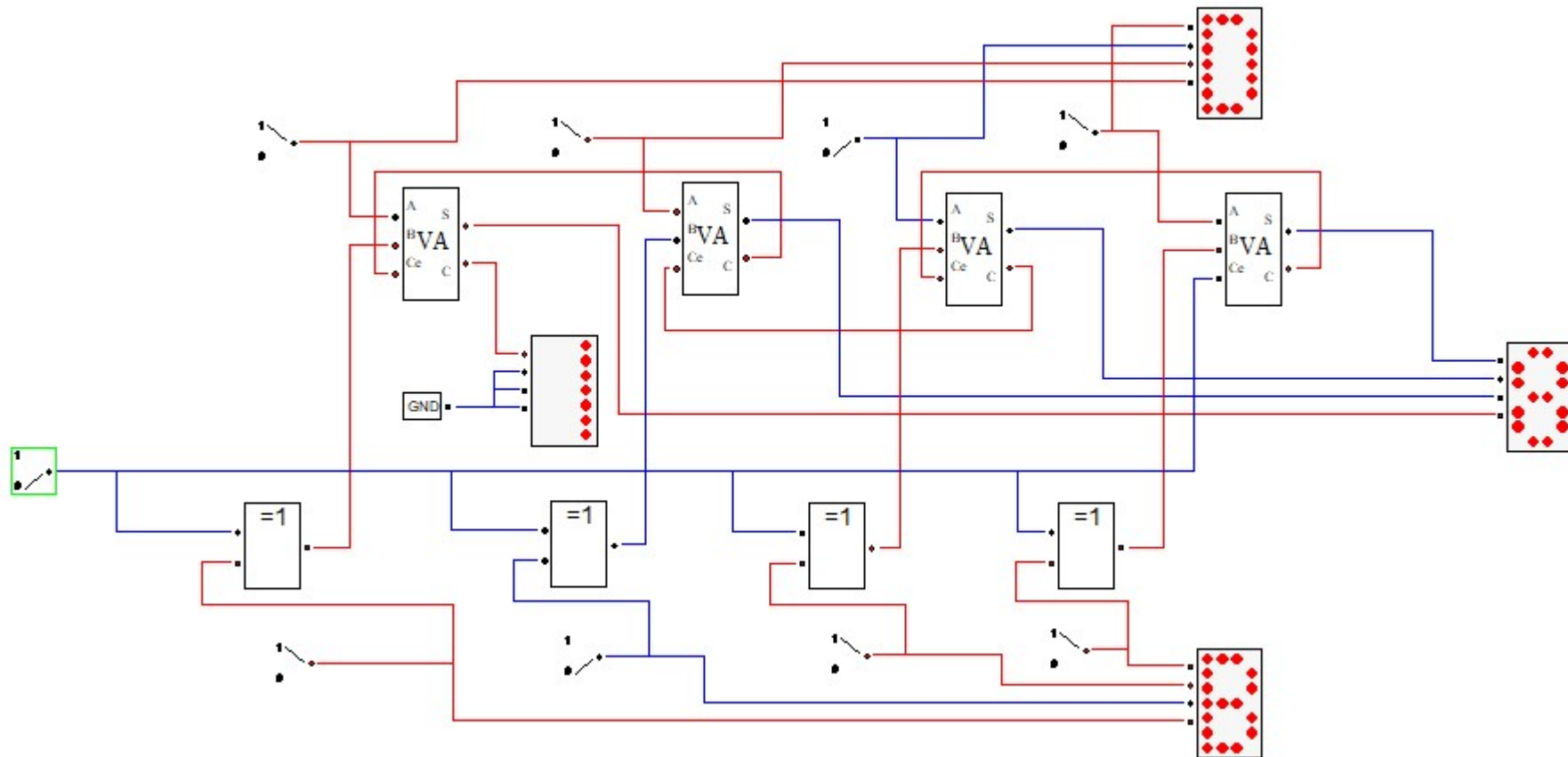
Dezimal:	09	Hexadezimal:	09	Dual:	0000 1001
	+ 10		+ 0A		+ 0000 1010
	19		13		0001 0011

2. Serieller 1-Bit-Addierer für 4-stellige Dualzahlen

Die Operanden werden jeweils in einem 4-Bit-Schieberegister abgelegt, nach 4 Taktimpulsen finden wir das Ergebnis (hier: die Summe) in einem weiteren 4-Bit-Schieberegister.



4-Bit-Paralleladdier mit Umschaltung auf 4-Bit-Parallelsubtraktion

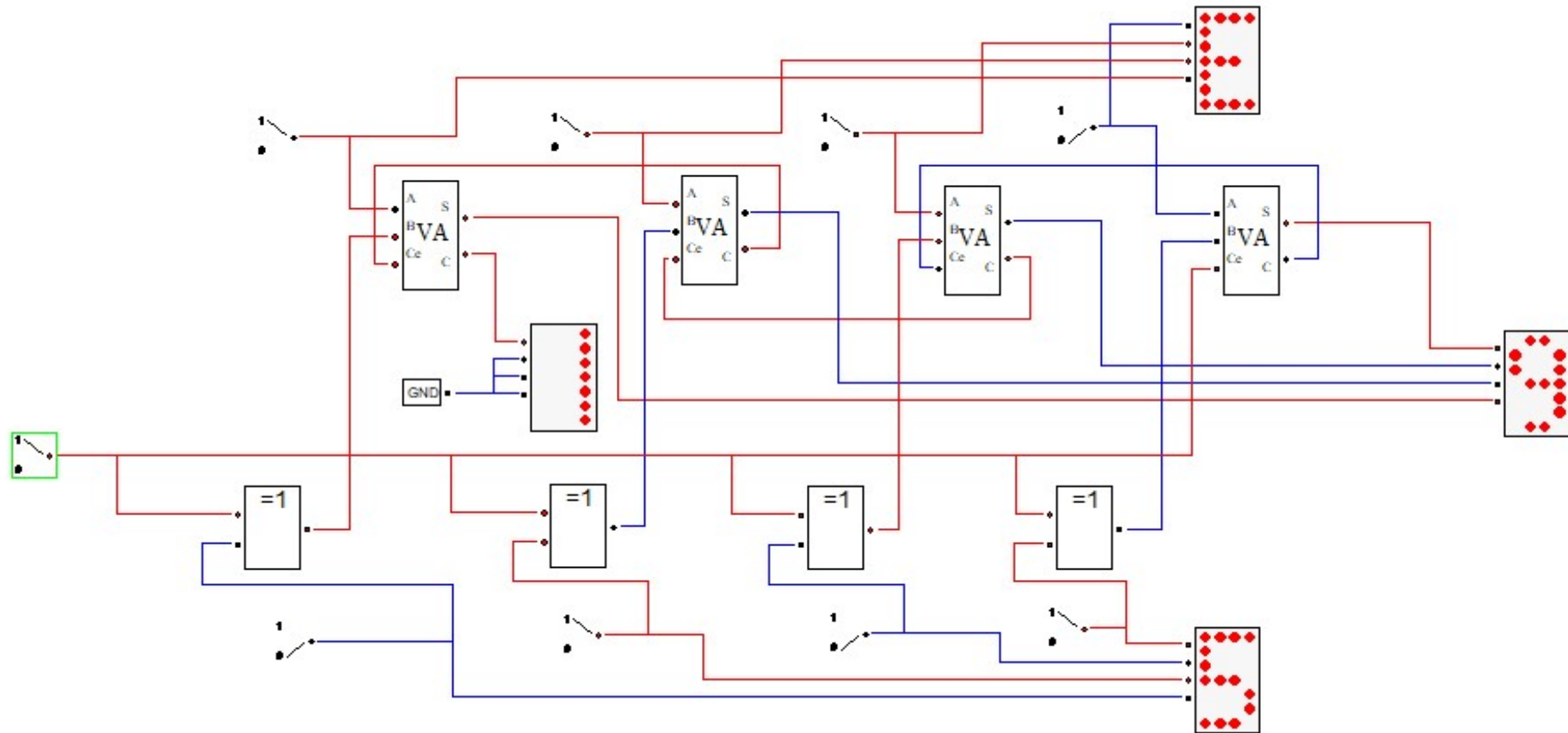


Dezimal: 13
 + 11
 ——
 24

Hexadezimal: 0D
 + 0B
 ——
 18

Dual: 0000 1101
 + 0000 1011
 ——
 0001 1000

4-Bit-Paralleladdier mit Umschaltung auf 4-Bit-Parallelsabtraktion



Dezimal: 14
 - 05

 09

Hexadezimal: 0E
 - 05

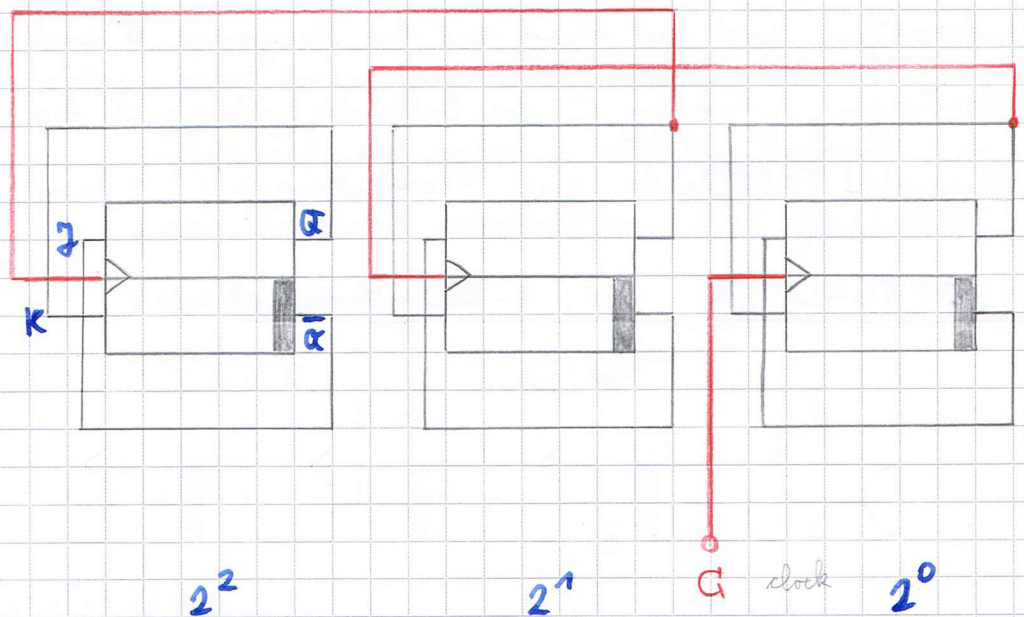
 09

Dual: 0000 1110
 - 0000 0101

 0000 1001

18.11.20

Dualzähler



Z = Anzahl der an C eingespeisten Taktimpulsen
 (genauer: die Anzahl der triggenden Taktflanken,
 hier: fallende Taktflanke)

	Z	2^2	2^1	2^0
nach 0 Taktimpulsen	0	0	0	0
nach 1 Taktimpuls	1	0	0	1
nach 2 Taktimpulsen	2	0	1	0
nach 3 Taktimpulsen	3	0	1	1
nach 4 Taktimpulsen	4	1	0	0
nach 5 Taktimpulsen	5	1	0	1
nach 6 Taktimpulsen	6	1	1	0
nach 7 Taktimpulsen	7	1	1	1

Bei 3 Flip-Flops: 8 Zustände

Bei n Flip-Flops: 2^n Zustände

Zahlbereich: 0 bis $2^n - 1$

4-Bit-Dualzähler

Schaltung mit alphanumerischer Anzeige des Zählergebnisses

Anzahl der Zustände: 2^4

Zählbereich: $0 \dots 2^4 - 1$

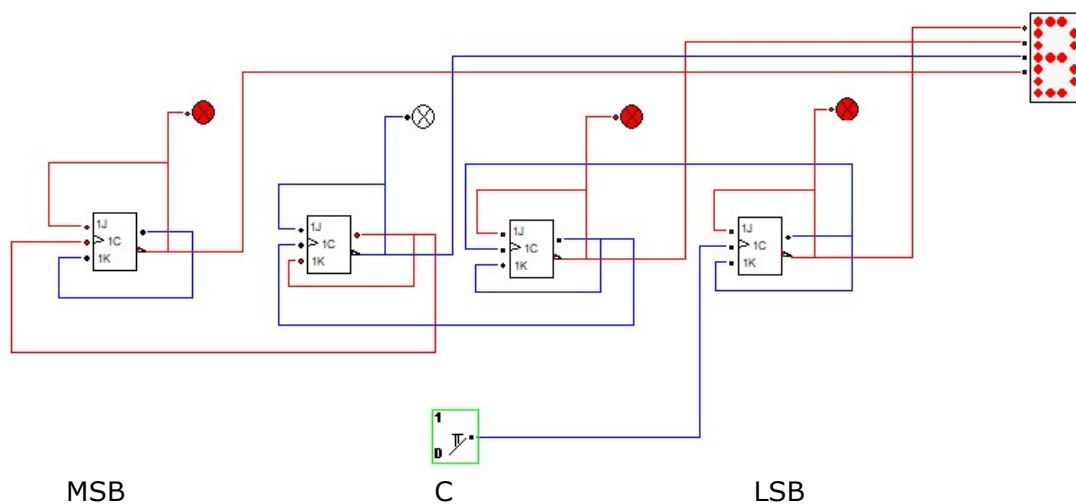
0000	-	1111	(dual)
0	-	F	(hexadezimal)
0	-	15	(dezimal)

Die am Eingang C eintreffenden Taktimpulse werden gezählt.

Beachte:

Bei Triggerung auf der fallenden Taktflanke erfolgt die Anzeige jeweils am Ausgang Q .

Bei Triggerung auf der steigenden Taktflanke erfolgt die Anzeige jeweils am Ausgang $\bar{Q}$.



Hier: Triggerung auf der steigenden Taktflanke, daher werden die Stellen (Bits) der Dualzahl jeweils dem Ausgang $\bar{Q}$ eines jeden Flip-Flops entnommen.