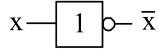


Junktoren

1. Negation

Die Verneinung eines Eingabewertes.

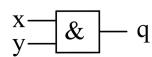


x	NOT
0	1
1	0

2. Konjunktion

Die Und-Verknüpfung von zwei Eingabewerten.

„Nur wenn x UND y wahr sind, dann...“

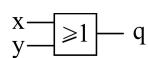


x	y	AND
0	0	0
0	1	0
1	0	0
1	1	1

3. Disjunktion

Die Oder-Verknüpfung von zwei Eingabewerten.

„Wenn x ODER y wahr sind, dann...“

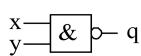


x	y	OR
0	0	0
0	1	1
1	0	1
1	1	1

4. NAND (not And)

Eine Negierung einer Konjunktion.

„Nur wenn x oder y NICHT wahr sind, dann...“

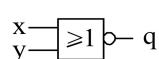


x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

5. NOR (not Or)

Eine Negierung einer Disjunktion.

„Nur wenn x und y NICHT wahr sind, dann...“

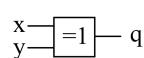


x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

6. EXOR

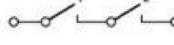
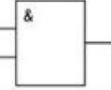
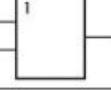
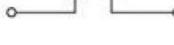
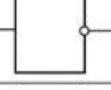
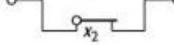
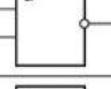
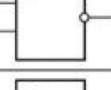
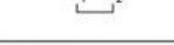
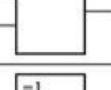
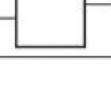
Auch ausschließende Disjunktion genannt.

„Nur wenn x ODER y NICHT wahr sind, sich aber nicht gleichen, dann...“



x	y	EXOR
0	0	0
0	1	1
1	0	1
1	1	0

Bauteilübersicht

Boolesche Funktion	Mathematische Darstellung	Kontaktanordnung	Wertetabelle	Symbol									
Konjunktion (UND)	$x_1 \wedge x_2 = y$		<table border="1" data-bbox="881 303 976 393"> <tr><td>x_1</td><td>\emptyset</td><td>L</td></tr> <tr><td>\emptyset</td><td>\emptyset</td><td>\emptyset</td></tr> <tr><td>L</td><td>\emptyset</td><td>L</td></tr> </table>	x_1	\emptyset	L	\emptyset	\emptyset	\emptyset	L	\emptyset	L	
x_1	\emptyset	L											
\emptyset	\emptyset	\emptyset											
L	\emptyset	L											
Disjunktion (ODER)	$x_1 \vee x_2 = y$		<table border="1" data-bbox="881 415 976 505"> <tr><td>x_1</td><td>\emptyset</td><td>L</td></tr> <tr><td>\emptyset</td><td>\emptyset</td><td>L</td></tr> <tr><td>L</td><td>L</td><td>L</td></tr> </table>	x_1	\emptyset	L	\emptyset	\emptyset	L	L	L	L	
x_1	\emptyset	L											
\emptyset	\emptyset	L											
L	L	L											
Negation (NICHT)	$\bar{x} = y$		<table border="1" data-bbox="881 527 976 617"> <tr><td>x</td><td>y</td></tr> <tr><td>\emptyset</td><td>L</td></tr> <tr><td>L</td><td>\emptyset</td></tr> </table>	x	y	\emptyset	L	L	\emptyset				
x	y												
\emptyset	L												
L	\emptyset												
Negiertes UND (NAND)	$\overline{x_1 \wedge x_2} = y$		<table border="1" data-bbox="881 640 976 729"> <tr><td>x_1</td><td>\emptyset</td><td>L</td></tr> <tr><td>\emptyset</td><td>L</td><td>L</td></tr> <tr><td>L</td><td>L</td><td>\emptyset</td></tr> </table>	x_1	\emptyset	L	\emptyset	L	L	L	L	\emptyset	
x_1	\emptyset	L											
\emptyset	L	L											
L	L	\emptyset											
Negiertes ODER (NOR)	$\overline{x_1 \vee x_2} = y$		<table border="1" data-bbox="881 752 976 842"> <tr><td>x_1</td><td>\emptyset</td><td>L</td></tr> <tr><td>\emptyset</td><td>L</td><td>\emptyset</td></tr> <tr><td>L</td><td>\emptyset</td><td>\emptyset</td></tr> </table>	x_1	\emptyset	L	\emptyset	L	\emptyset	L	\emptyset	\emptyset	
x_1	\emptyset	L											
\emptyset	L	\emptyset											
L	\emptyset	\emptyset											
Äquivalenz (Exklusiv-NOR)	$(\bar{x}_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2) = y$		<table border="1" data-bbox="881 864 976 954"> <tr><td>x_1</td><td>\emptyset</td><td>L</td></tr> <tr><td>\emptyset</td><td>L</td><td>\emptyset</td></tr> <tr><td>L</td><td>\emptyset</td><td>L</td></tr> </table>	x_1	\emptyset	L	\emptyset	L	\emptyset	L	\emptyset	L	
x_1	\emptyset	L											
\emptyset	L	\emptyset											
L	\emptyset	L											
Antivalenz (Exklusiv-ODER)	$(\bar{x}_1 \wedge x_2) \vee (x_1 \wedge \bar{x}_2) = y$		<table border="1" data-bbox="881 977 976 1066"> <tr><td>x_1</td><td>\emptyset</td><td>L</td></tr> <tr><td>\emptyset</td><td>\emptyset</td><td>L</td></tr> <tr><td>L</td><td>L</td><td>\emptyset</td></tr> </table>	x_1	\emptyset	L	\emptyset	\emptyset	L	L	L	\emptyset	
x_1	\emptyset	L											
\emptyset	\emptyset	L											
L	L	\emptyset											

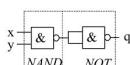
Realisierung von Verknüpfungen aus nur NAND-Gattern

Inverter



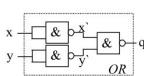
$$q = \bar{x} = (\bar{x} \wedge x)$$

AND-Verknüpfung



$$q = x \wedge y = \overline{\bar{x} \wedge \bar{y}} = (\bar{x} \wedge y) \wedge (\bar{x} \wedge \bar{y})$$

OR-Verknüpfung



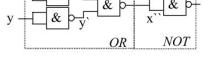
$$q = x \vee y = \overline{\bar{x} \wedge \bar{y}} = \overline{\bar{x} \wedge \bar{y}} = (\bar{x} \wedge x) \wedge (\bar{y} \wedge y)$$

NAND-Verknüpfung



$$q = \bar{x} \wedge \bar{y}$$

NOR-Verknüpfung



$$q = \overline{x \vee y} = \overline{\bar{x} \wedge \bar{y}} = \overline{\bar{x} \wedge \bar{y}} = (\bar{x} \wedge x) \wedge (\bar{y} \wedge y)$$

EXOR-Verknüpfung

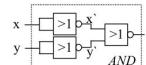


$$\begin{aligned} q &= (\bar{x} \wedge y) \vee (\bar{x} \wedge y) = (\bar{x} \wedge y) \vee (\bar{x} \wedge y) = (\bar{x} \wedge y) \wedge (\bar{x} \wedge y) \\ &= (\bar{x} \wedge y) \wedge ((\bar{x} \wedge y) \wedge (\bar{x} \wedge y)) \end{aligned}$$

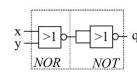
Realisierung von Verknüpfungen aus nur NOR-Gattern



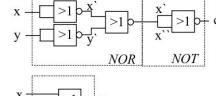
$$q = \bar{x} = (\bar{x} \wedge x)$$



$$q = x \wedge y = \overline{\bar{x} \wedge \bar{y}} = \overline{\bar{x} \wedge \bar{y}} = (\bar{x} \vee x) \wedge (\bar{y} \vee y)$$



$$q = x \vee y = \overline{\bar{x} \wedge \bar{y}} = \overline{\bar{x} \wedge \bar{y}} = (\bar{x} \vee y) \wedge (\bar{y} \vee x)$$



$$q = \bar{x} \wedge \bar{y} = \overline{\bar{x} \wedge \bar{y}} = \overline{\bar{x} \wedge \bar{y}} = (\bar{x} \vee x) \wedge (\bar{y} \vee y)$$



$$q = \overline{x \vee y}$$

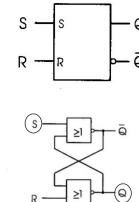
$$\begin{aligned} q &= (\bar{x} \wedge y) \vee (\bar{x} \wedge y) = (\bar{x} \wedge y) \vee (\bar{x} \wedge y) = (\bar{x} \wedge y) \wedge (\bar{x} \wedge y) \\ &= ((\bar{x} \wedge y) \wedge (\bar{x} \wedge y)) \vee ((\bar{x} \wedge y) \wedge (\bar{x} \wedge y)) \end{aligned}$$

Flip Flops

Flip Flops sind elektronische Schaltungen, die zwei stabile elektronische Zustände besitzen. Durch Eingangssignale kann dieser Zustand verändert werden. Typische Anwendung finden sie bei Schieberegistern, Speichern, Zählern und Frequenzteilern.

1. RS-Flip-Flop (NOR-Gatter)

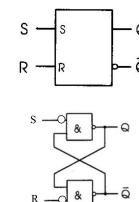
Der Eingabewert bleibt im Flip Flop erhalten. Ein Hoch bei beiden Eingängen sollte vermieden werden, da der Zustand nicht bestimmbar ist.



S	R	Q	\bar{Q}
0	0	Q_1	\bar{Q}_1
0	1	0	1
1	0	1	0
1	1	?	?

2. RS-Flip-Flop (NAND-Gatter)

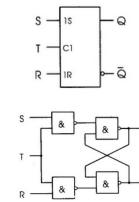
In der Industrie werden häufig NAND-Gatter verwendet. Also wird der RS-Flip Flop meist durch NAND-Gatter dargestellt. An der Funktion ändert sich nichts.



S	R	Q	\bar{Q}
0	0	Q_1	\bar{Q}_1
0	1	0	1
1	0	1	0
1	1	?	?

3. RS-Flip-Flop mit Takteingang

Ist eine Erweiterung zum R-S Flip-Flop. Nur wenn T eins ist, kann ein Input übernommen werden. Wenn T null ist wird der letzte Zustand gespeichert.

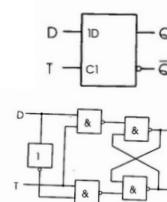


T	S	R	Q	\bar{Q}
1	0	0	Q_1	\bar{Q}_1
1	0	1	0	1
1	1	0	1	0
1	1	1	?	?
0	X	X	Q_1	\bar{Q}_1

Gleich mit R-S
Flip-Flop
⇒ Speichert
Zustand

4. D-Flip-Flop

Vom Aufbau ähnelt das D-Flip Flop dem RS-Flip Flop. Jedoch wurde hier durch nur einen Input der Fehler des nicht definierten Zustands behoben.

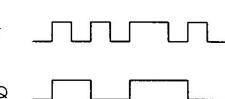
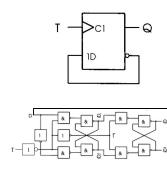


T	D	Q	\bar{Q}
0	X	Q_1	\bar{Q}_1
1	0	0	1
1	1	1	0

Speichert
Zustand
⇒ Transparent

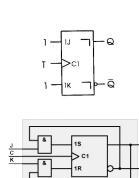
5. T-Flip-Flop

Toggle-Flip-Flops erhält man, wenn man den invertierten Ausgang eines Taktgesteuerten D-Flip-Flops wieder als Dateneingang benutzt. Diesen wechselnden Zustand nennt man Frequenzteiler.



6. J-K-Flip-Flop

Der J-K-Flip-Flop ist eine Erweiterung der R-S-Flip-Flops. Durch Rückkoppelung der komplementären Ausgänge gibt es keinen undefinierten Zustand. Sind beide Inputs 1, toggled dieser wie ein T-Flip-Flop. In diesem Zustand teilt dieser den Takt durch zwei.



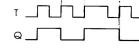
J	K	Q	\bar{Q}
0	0	h	h
0	1	0	1
1	0	1	0
1	1	X	X

Speichert
Zustand
⇒ Setzen von
Werten
⇒ Wechselbetrieb

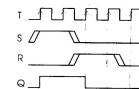
- **Ungetaktete Flip-Flops:** Ihr Zustand wird nur von den Setz-/Rücksetz-Eingängen gesteuert.
- **Getaktete Flip-Flops:** Der Zeitpunkt der Informationsübernahme wird durch ein Taktsignal vorgegeben.
- **Zustandsgesteuerte Flip-Flops:** Die Informationsübernahme wird durch einen Pegel des Steuersignals veranlasst.
- **Flankengesteuerte flip-Flops:** Die Informationsübernahme wird durch einen Zustandswechsel veranlasst.

Schaltsymbol	Flip-Flop	Steuerung
	T-Flip-Flop	getaktet, einflankengesteuert
		nicht getaktet, zustandsgesteuert
	RS-Flip-Flop	getaktet, einzustandsgesteuert
		getaktet, einflankengesteuert
	JK-Flip-Flop	getaktet, zweizustandsgesteuert
		getaktet, zweiflankengesteuert
	D-Flip-Flop	getaktet, einzustandsgesteuert
		getaktet, einflankengesteuert

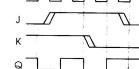
Das **T-Flip-Flop** teilt die Impulsanzahl des TAKTES durch zwei. Bei annähernd konstanter TAKTFREQUenz spricht man von einem **Frequenzteiler**.



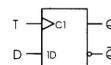
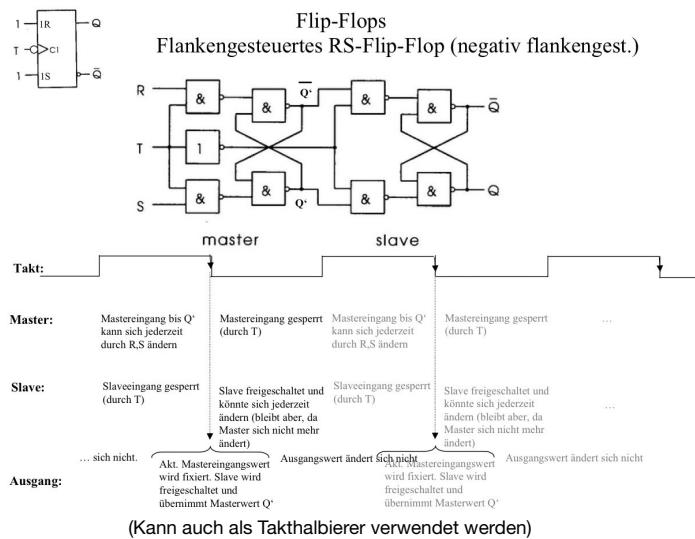
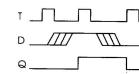
Der **RS-Flip-Flop** wird durch die positive Flanke des TAKTS gesetzt, wenn der Setz-Eingang auf H liegt. Wiederholtes Setzen ändert den Zustand nicht. Ist das Reset-Signal zum Zeitpunkt der positiven TAKTFLANKE H, wird das Flip-Flop rückgesetzt. R = S = 1 führt zum Zeitpunkt der TAKTFLANKE zu einem irregulären Zustand. Ansonsten ist die Kombination zulässig.



Beim taktflankengesteuerten **JK-Flip-Flop** hängen die Ausgangssignale von den asynchronen Vorbereitungseingängen J und K ab. Bei der Kombination $J = K = 1$ arbeitet das Flip-Flop als T-Flip-Flop, bei der Kombination $J = K = 0$ speichert es den vorherigen Zustand.

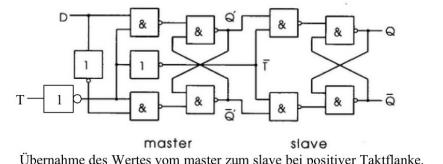
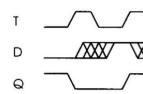


Das **D-Flip-Flop** übernimmt mit der positiven TAKTFLANKE den Wert am Dateneingang. Die rampenartigen Flanken des D-Signals deuten an, dass der Zeitpunkt des PEGELWECHSELS innerhalb des gezeichneten Intervalls gleichgültig für die Funktion des Schaltkreises ist.



Flankengesteuertes D-Flip-Flop (positiv flankengesteuert)

Die Transparenz des D-Flip-Flops geht verloren, wenn man zwei hintereinander schaltet. Die beiden Flip-Flops werden mit komplementären TAKTsignal angesteuert.



Diese Anordnung bezeichnet man als **master-slave-Flip-Flop**.

Der Q-Ausgang des masters folgt dem D-Signal, solange $T = 0$ anliegt. Das slave-Flip-Flop bleibt solange verriegelt. Geht das TAKTsignal auf 1, so verriegelt das master-Flip-Flop, und das folgenden slave-Flip-Flop übernimmt den logischen Zustand des Q-Ausgangs des masters.

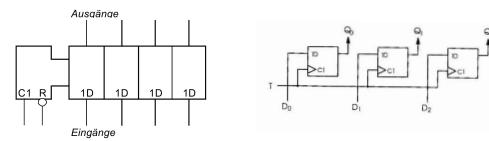
Ein Zustand wird also immer nur bei einer **Flanke** übernommen, man spricht von einem **dynamischen Flip-Flop**.

Register/Zähler

Flip Flops können hintereinandergeschaltet mehrere Aufgaben erfüllen. Sie können als Speicher verwendet werden oder zu Zählern zusammengeschlossen werden.

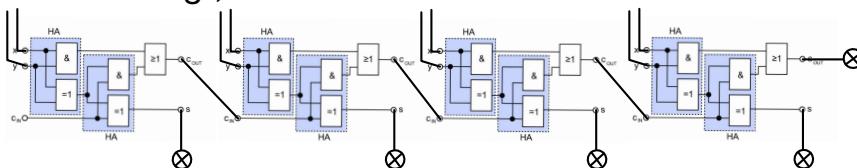
1. Speichermodul

Eine Parallele Anordnung von D-Flip Flops kann zur Zwischenspeicherung von Signalzuständen verwendet werden.



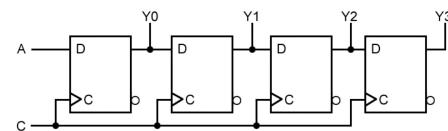
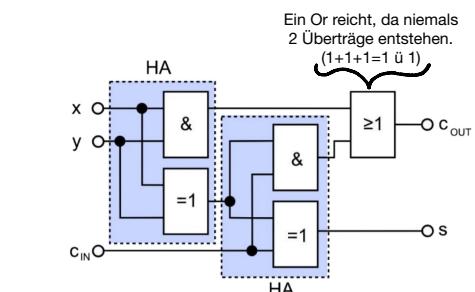
2. Halbaddierer/Addierer

Ein Halbaddierer (HA) addiert zwei 1bit Zahlen zusammen. Als Outputs gibt es ein Ergebnis- und ein „Übertragsbit“. Schaltet man zwei Halbaddierer hintereinander, so wird auch das Übertragsbit berücksichtigt, und es wird ein Volladdierer.



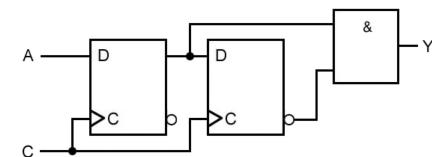
3. Schieberegister

Bei einem Schieberegister sind mehrere Flipflops hintereinander geschaltet. Der Takt schiebt den gespeicherten Zustand dann einen Flipflops weiter. Dadurch lässt sich ein gleichbleibenden sequenziellen Verhalten realisieren.



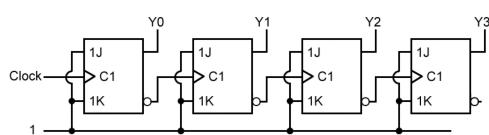
4. Synchrones Monoflop

Ein synchrones Monoflop ist eine Art Schalter, der sich selbstständig zurücksetzt. Der Vorteil ist, dass man nur eine Eingabe braucht, um nacheinander 2 Zustände auszulösen. Ein Beispiel wäre ein Licht, welches automatisch ausgeht.



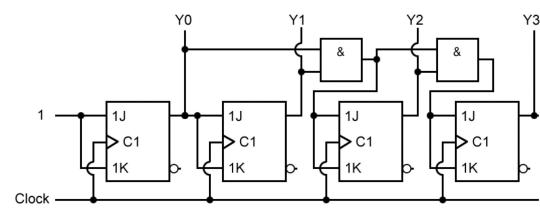
5. Asynchroner Zähler

Der asynchrone Zähler ist inkonsistent in den Ausgangssignalen. Längere Verzögerungen oder fehlerhafte Zustände sind unvermeidbar, dafür ist der Aufbau einfacher.



6. Synchroner Zähler

Der synchrone Zähler ist schnell und bietet zuverlässige Ausgangssignale. Deswegen wird dieser deutlich häufiger verwendet.



Zahlensysteme

Zahlensysteme sind vor allem in der Elektronik ein essenzieller Bestandteil. Unterschiedliche Darstellungen von Zahlen bieten jeweils andere Vorteile.

1. Dezimalsystem

Das Dezimalsystem gehört zu den meist benutzen Zahlensystemen. Wahrscheinlich hat es sich durch unsere Anatomie mit 10 Fingern etabliert.

Elemente: 1,2,3,4,5,6,7,8,9,0

Rechenregeln: Hoffentlich bekannt.

2. Binärsystem

Das Binärsystem wird häufig in der Elektronik eingesetzt, da man es auch durch einen hohen und niedrigen Strompegel darstellen kann.

Elemente: 1,0

Rechenregeln:

2.1: Umformen

135	—————	111101
68 : 2 = 62	Rest: 1	0 : 2 = 1 1
62 : 2 = 31	Rest: 0	1 : 2 = 1 3
31 : 2 = 15	Rest: 1	3 : 2 = 1 7
15 : 2 = 7	Rest: 1	7 : 2 = 1 15
7 : 2 = 3	Rest: 1	15 : 2 = 1 31
3 : 2 = 1	Rest: 1	31 : 2 = 1 62
1 : 2 = 0	Rest: 1	62 : 2 = 1 135
		⇒ 135

Das Horner-Schema

135	—————	111101
68 : 2 = 62	Rest: 1	0 : 2 = 1 1
62 : 2 = 31	Rest: 0	1 : 2 = 1 3
31 : 2 = 15	Rest: 1	3 : 2 = 1 7
15 : 2 = 7	Rest: 1	7 : 2 = 1 15
7 : 2 = 3	Rest: 1	15 : 2 = 1 31
3 : 2 = 1	Rest: 1	31 : 2 = 1 62
1 : 2 = 0	Rest: 1	62 : 2 = 1 135
		⇒ 135

Die Umrechnungstabelle

2 2 2 2 2 1
32 16 8 4 2 1
4 1 1 1 0 1

$$⇒ 32 \cdot 16 + 8 \cdot 4 + 1 = 61$$

2.2: Rechnungen

1 0 1 0 1 0 1 0 1 0	—————	1 0 0 0 · 1 0 1 0
+ 1 1 0 0 1 0 0 1 0 0		0 0 0 0
—————		1 0 0 0

1 0 0 0 · 1 0 1 0	—————	1 1 1 0 0 : 1 0 0 = 1 1 1
1 0 0 0		1 0 0
0 0 0 0		0 1 1 0
—————		1 0 0

2.3: Komplemente

7:0 1 1 1 6:0 1 1 0 5:0 1 0 1 ← 4:0 1 0 0 3:0 0 1 1 2:0 0 1 0 1:0 0 0 1 0:0 0 0 0 -1:1 1 1 1 -2:1 1 1 0 ← -3:1 1 0 1 -4:1 1 0 0 -5:1 0 1 1 -6:1 0 1 0 -7:1 0 0 1 -8:1 0 0 0	—————	Bei der zweierkomplementdarstellung wird das Most significant Bit (ganz links) als Vorzeichen verwendet. Mit dieser kann man von $-2^{(n-1)}$ bis $2^{(n-1)} - 1$ alle Zahlen darstellen. Eine Umwandlung ins Dezimalsystem ist bei positiven Komplementen immer möglich, bei negativen musst man wieder zurückrechnen und dann ein Minus hinzufügen.
0010 ⇒ 1101 + 1 ⇒ 1110		1011 ⇒ 0100 + 1 ⇒ 0101

8' 8' 8' 8' 8' 8' 8' 8'	—————	0 0 0 2 4 5
0 0 0 2 4 5		⇒ 42818 + 1 = 42819

3. Oktalsystem

Das Oktalsystem war damals das vorherrschende Zahlensystem für Computer, da es sich leicht ins Binärsystem umwandeln lässt, denn 0=000 und 7=111.

Elemente: 1,2,3,4,5,6,7,0

Rechenregeln: Umrechnungstabelle

4. Hexadezimalsystem

Das Hexadezimalsystem löst heutzutage das Oktalsystem größtenteils ab. Es ist kürzer und damit besser zu lesen und zu schreiben. Außerdem sind heutige Datenwerte fast nur noch in 16,32 oder 64Bit dargestellt. Genau wie bei dem Oktalsystem lässt sich auch leicht als Binär darstellen. (0=0000, F=1111)

Elemente: 1,2,3,4,5,6,7,8,9,0,A,B,C,D,E,F

Rechenregeln: Umrechnungstabelle

Normalformen

Eine Schaltung, die durch boolsche Funktionen definiert wurde, hat nahezu unendlich viele verschiedene Realisierungen. Um dies in einem standardisierten eindeutigen Therm definieren zu können, existieren die Normalformen.

1. Minterm/Maxterm

Ein Minterm ist die konjunktive (Und) Verknüpfung aller Eingangsvariablen.

Ein Maxterm ist die disjunktive (oder) Verknüpfung aller Eingangsvariablen.

A	B	C	f(A, B, C)	Minterm
0	0	1	1	$\rightarrow (\neg A \wedge \neg B \wedge C)$
1	0	0	1	$\rightarrow (A \wedge \neg B \wedge \neg C)$

A	B	C	f(A, B, C)	Maxterm
0	0	1	0	$\rightarrow (A \vee B \vee \neg C)$
1	0	0	0	$\rightarrow (\neg A \vee B \vee C)$

2. Kanonische Disjunktive-/Konjunktive Normalform

Die kanonischen Normalformen verknüpfen dann diese Minterme/Maxterme dann disjunktiv/konjunktiv und stellen so dann einen eindeutigen Therm der Schaltung dar. Hier müssen alle Eingangsvariablen enthalten sein (keine Vereinfachung)

$$f(A,B,C) = (\neg A \wedge B \wedge C) \vee (\neg A \wedge B \wedge \neg C)$$

$$f(A,B,C) = (\neg A \vee B \vee C) \wedge (A \vee B \vee \neg C)$$

3. Disjunktive-/Konjunktive Normalform

In den Normalformen werden abschließend die kanonischen Normalformen zusammengefasst. Wenn mehr wahre Ergebnisse (1) rauskommen, verwendet man die Disjunktive, andernfalls die Konjunktive Normalform.

a	b	c	f	Normalform (kanonisch)	Vereinfacht
0	0	0	0		
0	0	1	1	$\rightarrow (\neg a \wedge \neg b \wedge c) \vee$	
0	1	0	0	$(\neg a \wedge c)$	
0	1	1	1	$\rightarrow (\neg a \wedge b \wedge c) \vee$	\vee
1	0	0	1	$\rightarrow (a \wedge \neg b \wedge \neg c) \vee$	$(a \wedge \neg c)$
1	0	1	0		
1	1	0	1	$\rightarrow (a \wedge b \wedge \neg c)$	
1	1	1	0		

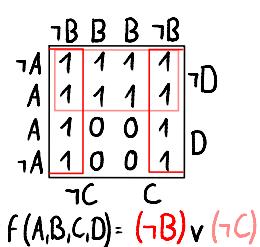
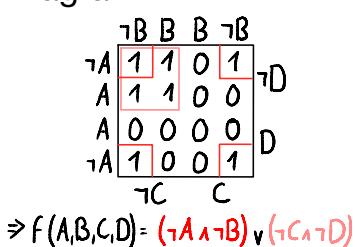
a	b	c	f	Normalform	Vereinfacht
0	0	0	1		
0	0	1	0	$\rightarrow (a \vee b \vee \neg c) \wedge$	
0	1	0	0	$\rightarrow (a \vee \neg b \vee c) \wedge$	
0	1	1	1		
1	0	0	0	$\rightarrow (\neg a \vee b \vee c) \wedge$	
1	0	1	1		
1	1	0	0	$\rightarrow (\neg a \vee \neg b \vee c)$	
1	1	1	1		

$$f(A,B,C) = (\neg a \wedge c) \vee (a \wedge \neg c)$$

$$f(a,b,c) = (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c)$$

4. KV-Diagramm

Das KV-Diagramm dient zur übersichtlichen Darstellung und Vereinfachung boolscher Funktionen. Mit diesem lässt sich jede beliebige Disjunktive Normalform in einen minimalen Ausdruck umwandeln. Jeder einzelne Ausgabewert benötigt ein KV-Diagramm.

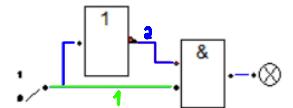


Glitches/Hazards

Laufzeitverzögerungen von Bauteilen können gravierend sein, wenn die ungewollten Störimpulse Auswirkungen auf die nachfolgende Logik hat. Deswegen versucht man, die Laufzeiten einer Schaltung im Blick zu haben.

1. Hazards

Bei einem Wechsel der Eingabe ist die Ausgabe vor Erreichen des korrekten Zustandes kurzzeitig falsch. Ursache sind Laufzeitunterschiede von verschiedenen Wegen im Schaltnetz. Grund dafür können Verzögerungen bestimmter Bauteile sein. Eine direkte Verbindung (1) ist demnach etwas schneller als eine Verbindung durch ein Bauteil (2).

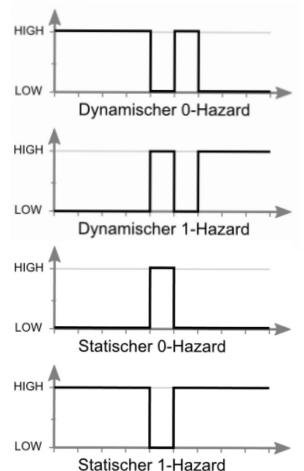
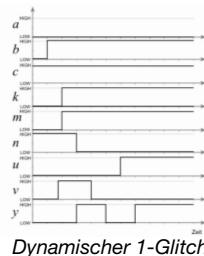
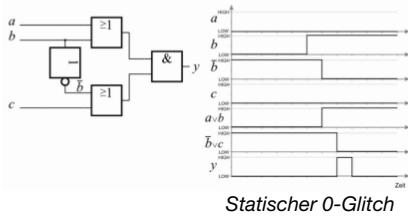


2. Statische- / Dynamische-Glitches

Ein **statischer Glitch** tritt auf, wenn kein Wechsel im Ausgabewert erfolgen soll, durch einen Glitch aber **kurzzeitig** ein **anderer Wert** ausgegeben wird. Bei einem **dynamischen Glitch** wechselt der korrekte Ausgabewert, wird jedoch vom Glitch **kurzzeitig** auf den **alten Wert** zurückgesetzt.

3. 0-/1-Glitches

Bei einem 0-Glitch erwartet man 0 als Ausgabewert, bekommt aber kurzzeitig 1. Bei einem 1-Glitch wiederum erwartet man eine 1, bekommt aber kurzzeitig eine 0.



4. Struktur-Glitches

Betrachtet man eine Schaltung in einem KV-Diagramm, gibt es Einser/Primterm-Blöcke. Bei einem Übergang von 2 getrennten Primtermblöcken kann es zu Glitches kommen, da die Schaltung kurz einen Zwischenzustand bekommen kann. Durch redundante Blöcke kann man die vermeiden.

	$\neg a \neg b$	$\neg ab$	ab	$a \neg b$
$\neg c \neg d$	0	0	0	0
$\neg cd$	0	1	0	0
cd	0	1	1	1
$c \neg d$	0	0	1	1

Eine Schaltung mit Strukturglitch

	$\neg a \neg b$	$\neg ab$	ab	$a \neg b$
$\neg c \neg d$	0	0	0	0
$\neg cd$	0	1	0	0
cd	0	1	1	1
$c \neg d$	0	0	1	1

Eine Schaltung mit Redundanz ohne Glitch

5. Funktions-Glitches

Bei einer Schaltung mit mehreren Eingangsvariablen kann es dazu kommen, dass sich die Zustände zeitversetzt verändern. Eine darauf abgestimmte Einteilung von Zuständen kann diesem Problem vorbeugen, in dem sich zum Beispiel immer nur ein Bit verändern muss.

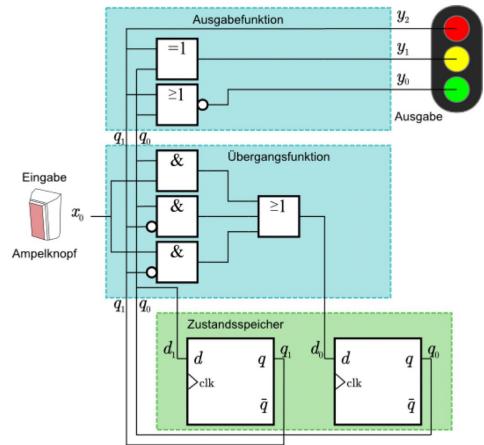
Anstatt: 00, 11, 01, 10 definiert man besser: 00, 01, 11, 10

Automaten

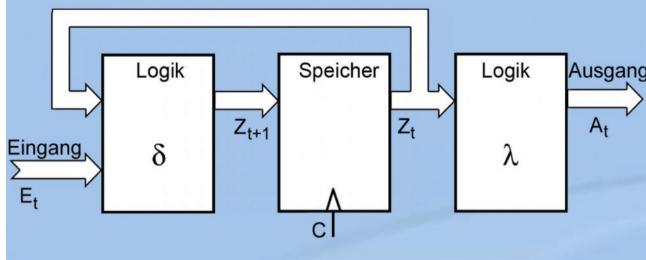
Ein Automat ist ein sequenzielles System, dessen Zustand um einen vom aktuellen Zustand und zum anderen von den Inputs abhängt.

1. Automaten

Im Gegensatz zu Zählern oder Registern sind Automaten nicht nur von ihrem internen Zustand abhängig. Zusätzlich zu einem Speichermodul und dem Logikmodul besitzt ein Automat ein zusätzliches Logikmodul, welches Eingangswerte verarbeiten kann. Dadurch wird die kombinatorische Logik meist extrem vereinfacht und ein Automat kann in Situationen eingesetzt werden, in dem man den laufenden aber vordefinierten Zustand verändern kann. Ein gutes Beispiel wäre ein Ampelsystem, bei dem es zwar eine fest definierte Reihenfolge gibt, der Rotzyklus aber erst mit dem Ampelknopf ausgelöst werden soll.

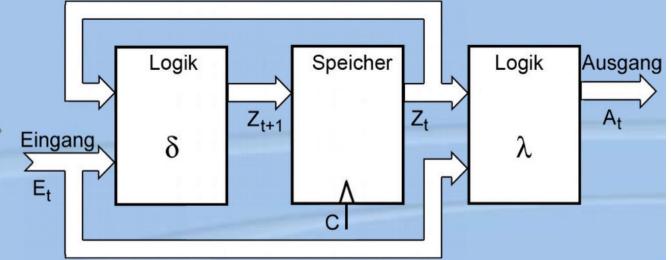


1. Moore-Automat



Bei dem Moore-Automaten wird der Input zusammen mit dem gespeicherten vorherigen Zustand in den Logikblock geleitet. Dieser speichert dann diesen neuen Zustand und gibt diesen anschließend über einen weiteren Logikblock aus und stellt diesen gleichzeitig wieder für den Input wieder zur Verfügung. Diese Schaltung ist **nicht transparent** und erst nach einer Speicherung des Zustandes wird die Ausgabe geändert.

2. Mealy-Automat



Der Moore-Automat gibt den Input im Gegensatz zum Mealy-Automaten direkt an beide Logikblöcke. Also gibt der Automat direkt den einen Wert aus, während der Zustand gleichzeitig gespeichert wird. Dies ermöglicht zwar eine direkte und **transparente** Ausgabe, verkompliziert aber auch die Logikblöcke, da hierbei in beide sowohl den gespeicherten Zustand als auch die Inputbits verarbeiten müssen.