Duale Hochschule Manheim DHBW

Kurs TINF21AI1

Rechnerarchitekturen I

Grundlagen Digital 1

Binäre Zahlen Grundlage

- **Zahlen zur Basis**
- **2^n**
- **Datentype Integer bzw. int**

Summe	A	B
0	0	0
1	1	0
1	0	1
0 (1)	1	1

2^3 = 8	2^2 = 4	2^1 = 2	2^0 = 1	Dezimal	Hexa-deimal
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	B
1	1	0	0	12	C
1	1	0	1	13	D
1	1	1	0	14	E
1	1	1	1	15	F

Binäre Rechenarten Addition

+	2^3	2^2	2^1	2^0
	1	0	1	0
	0	0	1	1
Übertrag = 0	1	1	0	1

+	2^3	2^2	2^1	2^0
	1	1	0	0
	0	1	0	0
Übertrag = 1	0	0	0	0

Binäre Vorzeichen

Es dient zur Darstellung Vorzeichen behafteter Zahlen. Hierzu wird das Most Significant Bit (MSB) einführt. Das Bit welches am weitesten Links steht wird für das Vorzeichen verwendet. 1 ist negativ und 0 ist positiv.

Zahl	Binär
3	011
2	010
1	001
0	000
-1	101
-2	110
-3	111

Subtraktion

Die Subtraktion im binären Zahlensystem erfolgt mit dem Zweierkomplement. Hierbei muss man zwei Fälle unterscheiden.
Minuend - Subtrahend
Fall1
Minuend größer Subtrahend
Fall2
Subtrahend größer Minuend

Der Subtrahend wird mit Zweierkomplement umgewandelt und zum Minuend addiert.

Zahl	Ergebnis	
0 - 0	0	
1 - 0	1	
1 - 1	0	
0 - 1	10	Überlauf

Subtraktion Zweikomplement

Wie müssen aus dem Subtrahend erst das Einerkomplement bilden zu dem wir 1 addieren.

Bespiel:
4 = 0100
Das Einerkomplement ergibt 1011
Für das Zweierkomplement addieren wir 1 hinzu somit haben wir dann 1100.

Fall 1 Minuend größer Subtrahend

8 – 4 = ?

1000 + 1100 = (1) 0100

0100 = 4

Fall 2 Subtrahend größer Minuend

2 – 4 =?

0010 + 1100 = 1110

Wir müssen dann das Einerkomplement von 1110 bilden zu 0001.

Mit dem Zweikomplement erhalten wir dann 0001 + 0001 = 0010.

Womit – 0010 als Ergebniss wie gewünscht zustande kommt.

Multiplikation

Die Multiplikation wird durch wiederholtes Addieren erreicht.
Hebei werden zwei Fälle unterschieden bei 1 wird der Wert dazu gerechnet mit einem Shift und bei 0 wird nur ein Shift gemacht.

1001 *1001							
				1	0	0	1
			0	0	0	0	
		0	0	0	0		
	1	0	0	1			
	1	0	1	0	0	0	1
				1			
0	**1**	**0**	**1**	**0**	**0**	**0**	**1**

Division

Die Division wird durch wiederholtes Subtrahieren erreicht. Hebei werden zwei Fälle unterschieden bei 1 wird der Wert abgezogen und bei 0 wird nur ein Shift gemacht.

	1100 : 11 = ?						
	1	**1**	**0**	**0**			
	1				<11		0
-	**1**	**1**			>=11		1
		0	0				
		0	**0**				0
			0	0			
			0	**0**			0

Subtraktion

Byte-Reihenfolge/Endianness

Im Rahmen der CPU Entwicklung haben sich zwei Formate durchgesetzt um Zahlen zu speichern, da man sich von den Formaten gewisse Vorteile versprochen hat bzw. hatte.

Integer	16.909.060 => 01020304(Hex)
BigEndian	01 02 03 04
LittleEndian	04 03 02 01

Im Netzwerk wird BigEndian verwendet.

Aufgabe

1. Was ist 8 und 11 in Bin und Hex
2. Was ist Hex A , D in Bin
3. 1100 + 1100
4. A + F
5. Was ist F + F in 4 Bit

Und/AND Gatter

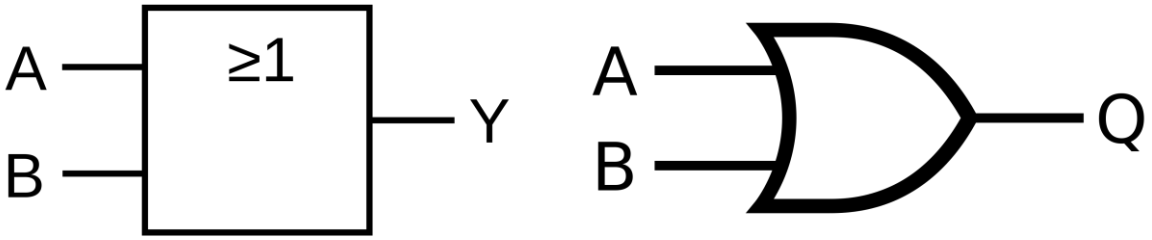
Funktion y = A ∧ B

Von User:Dbc334, User:Tobulax - Image:IEC_OR.svg, Gemeinfrei,
https://commons.wikimedia.org/w/index.php?curid=2329841
jjbeard - Own Drawing, made in Inkscape 0.43

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

Oder / Or Gatter

Funktion y = A V B

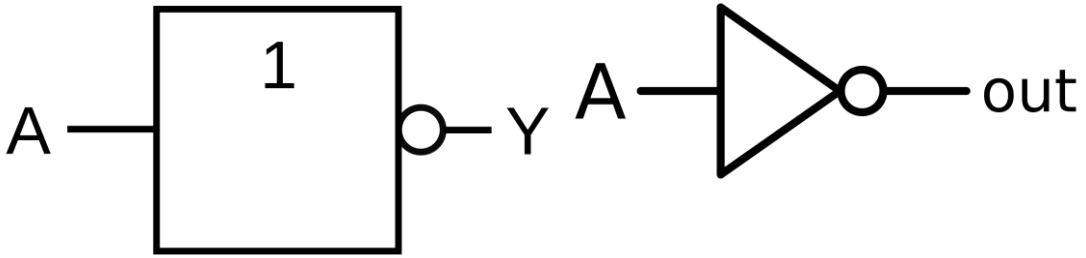


Von User:Dbc334, User:Tobulax - Image:IEC_OR.svg, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=2329841
Inductiveload - Own work

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

Nicht / Not Gatter

Funktion y = not (A)

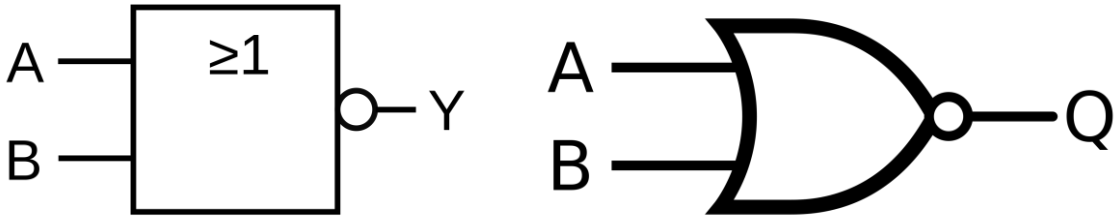


Von User:Dbc334, User:Tobulax - Image:IEC_NOT.svg, Gemeinfrei,
https://commons.wikimedia.org/w/index.php?curid=2329840
CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=829068

A	Y
0	1
1	0

Nor Gatter

Funktion y = not(A ∨ B)

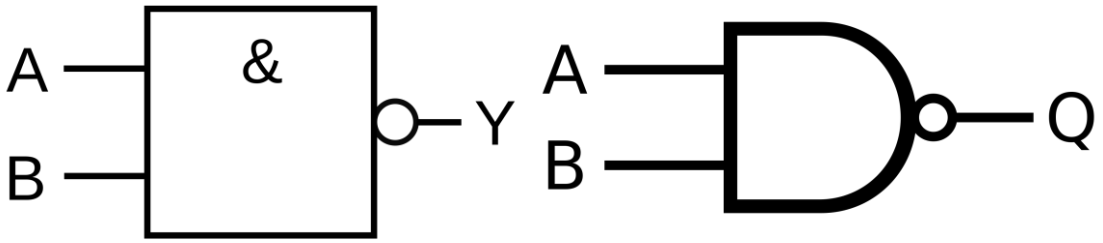


Von User:Dbc334, User:Tobulax - Image:IEC_OR.svg, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=2329841 Inductiveload - logOwn work

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

NAND Gatter

Funktion y = not(A ∧ B)

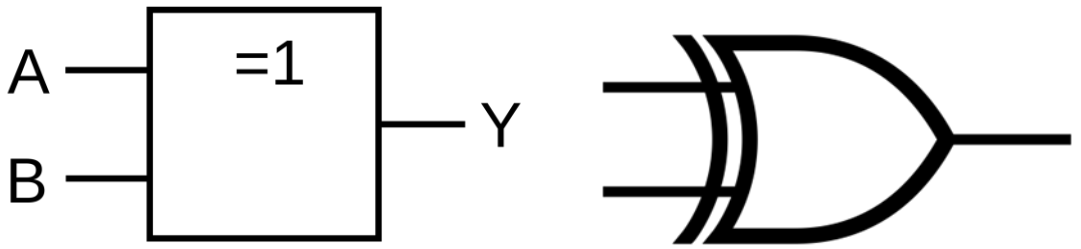


Von User:Dbc334, User:Tobulax - Image:IEC_OR.svg, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=2329841
By Inductiveload - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5729015

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

XOR Gatter

Funktion
$y = (\text{not } A \vee B) \wedge (A \vee \text{not } B)$

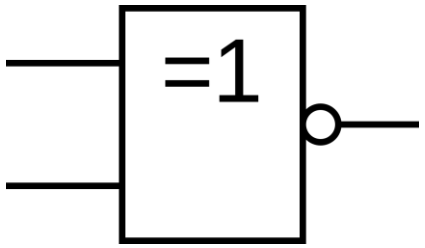


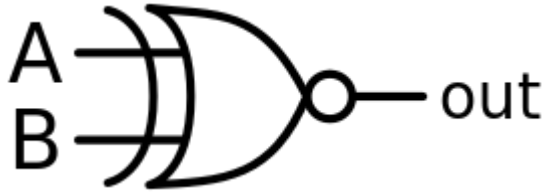
Von User:Dbc334, User:Tobulax - Image:IEC_OR.svg, Gemeinfrei,
https://commons.wikimedia.org/w/index.php?curid=2329841
jjbeard - Own Drawing, made in Inkscape 0.43

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

XNOR Gatter

Funktion
$y = \text{not}(A \vee B) \wedge \text{not}(\text{not } A \vee \text{not } B)$



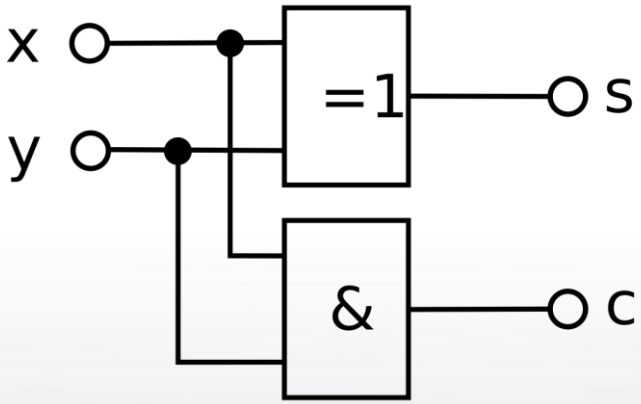


Traced by User:Stannered - en:Image:IEC XNOR.gif
Traced by User:Stannered - en:Image:IEC XNOR.gif

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

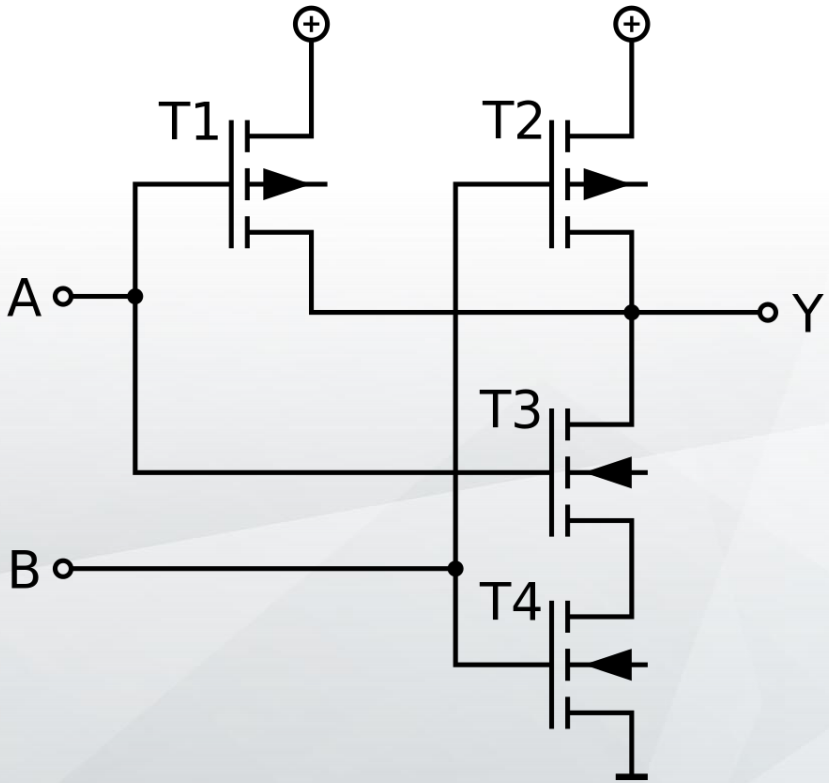
Übungen

1. $Q = A \wedge (A \wedge B)$
2. $Q = (A \wedge B \wedge C) \vee (\text{not}(A) \wedge B \wedge C)$
3. Q für EXOR
4. 1. in NAND
5. 3. in NAND
6. Siehe Bild



Warum NAND / NOR

Geringe Anzahl an Transistoren erforderlich. Mann kann mit NAND/NOR jedes Gatter nachbilden und erzeugen.



Clock/Tack

1. Synchronisieren
2. Störsignalauswirkung minimieren
3. Erzeugung hoher Frequenzen mit PLL
4. Nutzen zum refresh

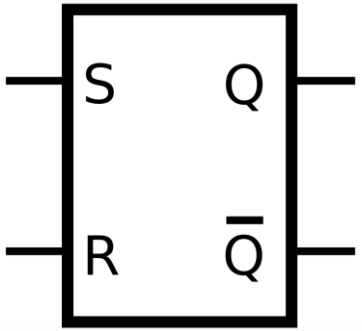
Wozu das alles

Um Signale zu verarbeiten und Speichern wurden mit den Grundelementen komplexe Schaltungen aufgebaut. Ein Element bilden die Flip Flops von diesen weitere Baugruppen abgeleitet werden.

SR Flip Flop

Funktion

$$Q = \overline{\overline{R} \vee Q^*} = \overline{R} \wedge \overline{Q^*} = \overline{R} \wedge (S \vee Q)$$



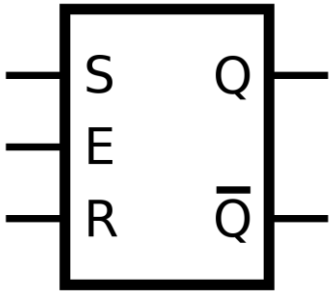
Von Dolicom - Eigenes Werk, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=28261155

S	R	Action	Q^m+1	Not Q^m+1
0	0	Hold	Q^m	Not Q^m
1	0	Set	1	0
0	1	Reset	0	1
1	1	bad	?	?

SR Flip Flop mit Takt

Funktion

$$Q = \overline{\overline{R} \vee Q^*} = \overline{R} \wedge \overline{Q^*} = \overline{R} \wedge (S \vee Q)$$

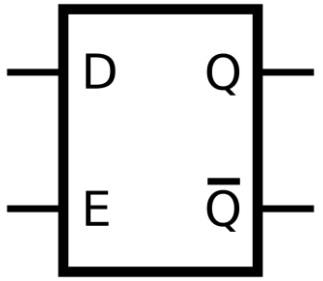


Von Dolicom - Eigenes Werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=28261155

S	R	T	Action	Q^m+1	Not Q^m+1
0	0	1	Hold	Q^m	Not Q^m
1	0	1	Set	1	0
0	1	1	Reset	0	1
1	1	1	bad	?	?
d	d	0	Hold	Q^m	Not Q^m

D Flip Flop

Funktion
SR Flip Flop mit Inverter von S auf R Eingang.
Keine Instabilität.

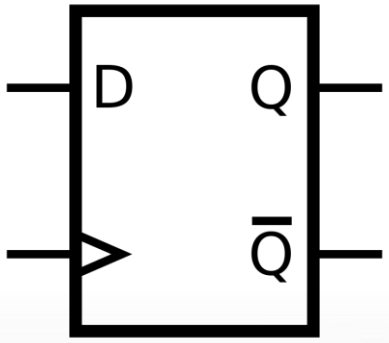


Von Inductiveload - Eigenes Werk, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=6712594

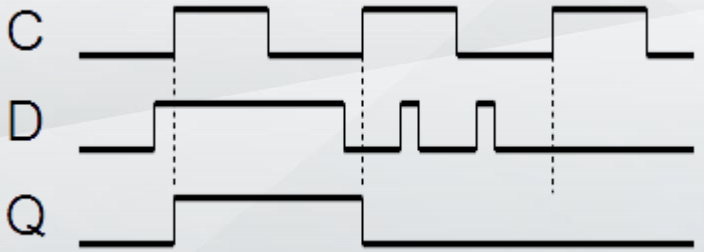
D	(R)	Action	Q^m+1	Not Q^m+1
0	1	Hold	Q^m	Not Q^m
1	0	Set	1	0

Unterschied am Beispiel D Flipp Flopp Flanken und Pegelgesteuert

In den vorhergehenden Beispielen wurde der Pegel zum Schalten verwendet. Dieses D Flipp Flopp hat einen Pegelsteuerung. An dem Dreieck am Clock kenntlich gemacht. Bei der steigenden Flanke wird das D Signal verarbeitet.
Ziel ist es mehr Bits über einen Taktzyklus zu übertragen.

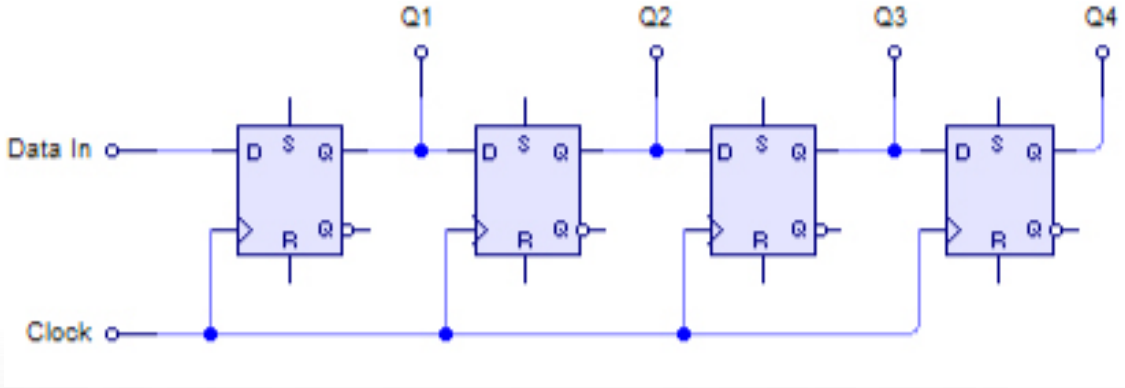


Von Inductiveload - Eigenes Werk, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=6693351



Schieberegister

1. Zum multiplizieren
2. Daten serialisieren
3. Addieren usw. (altes verfahren), da man wenig Platz auf dem Chip hatte
4. Bitmuster erzeugen
5. Und andere Operation

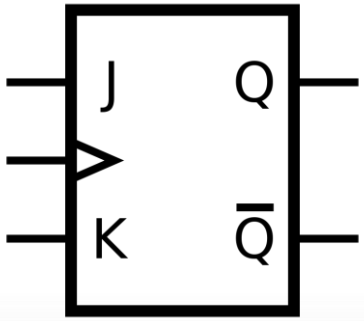


en:User:Jjbeard

JK Flip Flop

Funktion

$$Q_{\text{next}} = J\overline{Q} + \overline{K}Q$$



By Inductiveload - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=6693391

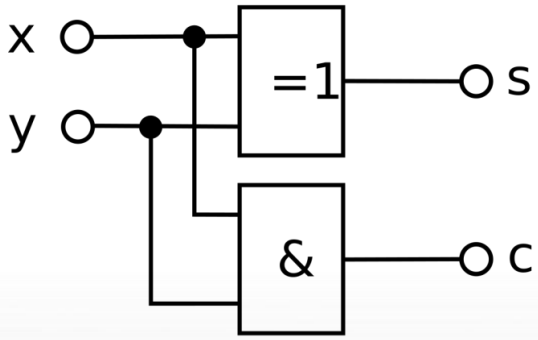
J	K	T	Action	Q^m+1	Not Q^m+1
0	0	1	Hold	Q^m	Not Q^m
1	0	1	Set	1	0
0	1	1	Reset	0	1
1	1	1	Toggel	Not Q^m	Q^m
d	d	0	Hold	Q^m	Not Q^m

Halbaddierer

Funktion

$$S = (\text{not } A \vee B) \wedge (A \vee \text{not } B)$$

$$C = A \wedge B$$



Von 30px MovGP0 - Eigenes Werk (Originaltext: selbst erstellt mit Inkscape), CC BY-SA 2.0 de, https://commons.wikimedia.org/w/index.php?curid=22912775

X	Y	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Quellen/Infos

https://de.wikipedia.org/wiki/Logikgatter
https://www.youtube.com/watch?v=jYmzJeMtUyU
https://de.wikipedia.org/wiki/Schieberegister

https://en.wikipedia.org/wiki/Main_Page
https://de.wikipedia.org/wiki/Wikipedia:Hauptseite

Digitaltechnik Klaus Fricke 9. Auflage