

Digitaltechnik Zusammenfassung

Thomas Jund <info@jund.ch>
Laurent Cohn <info@cohn.ch>
Andrew Mustun <andrew@mustun.com>

31. Dezember 2002

Version 1.0

Formelsammlung für das Fach Digitaltechnik des Studiengangs IT an der ZHW.

Inhaltsverzeichnis

1	Schaltalgebra	1
1.1	Elementare Schaltalgebra	1
1.2	Grundgesetze	1
1.3	Morgansche Gesetze (Inversionsgesetze)	2
1.4	Disjunktive Normalform (DNF)	2
1.5	Konjunktive Normalform (KNF)	2
1.6	Umformung für Realisierung mit NAND	2
2	Karnaugh-Diagramm	3
2.1	Karnaugh-Diagramm mit Wahrheitstabelle 4 Variablen	3
2.2	Karnaugh-Diagramm mit Wahrheitstabelle 3 Variablen	3
2.3	Karnaugh-Diagramm - EX-OR und EX-NOR	4
2.4	Karnaugh-Diagramm - Spezialfälle	5
3	BCD-7-Segment	6
3.1	BCD-7-Segment-Codeumsetzer	6
4	Code	7
4.1	Elementares (Einerkomplement, Zweierkomplement)	7
4.2	BCD-Code	8
4.3	GRAY-Code	9
4.4	3-Exzess-Code	10
4.5	Aiken-Code	11
5	VHDL - VHSIC Hardware Description Language	12
5.1	Grundelemente der Programmierung	12
5.2	Verhaltensbeschreibung eines D-Flip-Flops	13
5.3	BCD Beispiel	14
6	Multivibratoren	15
7	Flip Flops	16
7.1	Unterscheidung nach Taktart	16
7.2	Unterscheidung nach Typ	16
7.3	RS-FF	17
7.3.1	Ungetaktetes RS-FF	17

INHALTSVERZEICHNIS

7.3.2	Taktzustandgesteuertes RS-FF	19
7.3.3	Master Slave RS-FF (Pulsgetriggertes RS-FF)	21
7.3.4	Flankengetriggertes RS-FF	22
7.4	D-FF	23
7.4.1	Taktzustandsgesteuertes D-FF (D-Latch)	23
7.4.2	Flankengetriggertes D-FF	24
7.5	Toggle-FF	25
7.6	JK-FF	26
8	Automaten	27
8.1	Daten zur Realisierung	27

1 Schaltalgebra

1.1 Elementare Schaltalgebra

AND-Funktion	AND-Theoreme
$1 \& 1 = 1$	$A \& 0 = 0$
	$A \& 1 = A$
$0 \& 0 = 0 \& 1 = 1 \& 0 = 0$	$A \& A = A$
	$A \& !A = 0$
	$!!A = A$

Allgemein formuliert: $X_1 \& X_2 \& X_3 \& X_4 \& \dots \& X_i = 1$ - falls alle $X_i = 1$, sonst 0

Dies lässt sich am einfachsten ausdrücken als Minimums-Funktion - min();

OR-Funktion	OR-Theoreme
$0 \# 0 = 0$	$A \# 0 = A$
	$A \# 1 = 1$
$0 \# 1 = 1 \# 0 = 1 \# 1 = 1$	$A \# A = A$
	$A \# !A = 1$
	$!!A = A$

Allgemein formuliert: $X_1 \# X_2 \# X_3 \# X_4 \# \dots \# X_i = 0$ - falls alle $X_i = 0$, sonst 1

Dies lässt sich am einfachsten ausdrücken als Maximums-Funktion - max();

1.2 Grundgesetze

Kommutativgesetz

$$A \& B \& C = C \& A \& B$$

$$A \# B \# C = C \# A \# B$$

Assoziativgesetz

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

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

$$A \& (!A \# B) = A \& B$$

$$A \# (!A \& B) = A \# B$$

Distributivgesetze

$$A \& (B \# C) = (A \& B) \# (A \& C)$$

$$A \# (B \& C) = (A \# B) \& (A \# C)$$

$$A \& (A \# B) = A$$

$$A \# (A \& B) = A$$

1.3 Morgansche Gesetze (Inversionsgesetze)

Dies sind die sogenannten Inversionsgesetze:

$$\begin{aligned} \overline{\overline{X_1 \& X_2 \& X_3 \& \dots \& X_n}} &= \overline{\overline{X_1} \# \overline{\overline{X_2}} \# \overline{\overline{X_3}} \# \dots \# \overline{\overline{X_n}}} \\ \overline{\overline{X_1 \# X_2 \# X_3 \# \dots \# X_n}} &= \overline{\overline{X_1} \& \overline{\overline{X_2}} \& \overline{\overline{X_3}} \& \dots \& \overline{\overline{X_n}}} \end{aligned}$$

1.4 Disjunktive Normalform (DNF)

Die disjunktive Normalform ist die OR-Verknüpfung aller guten Minterme. Diese kann aus dem Karnaugh-Diagramm ohne Probleme herausgelesen werden. Minterme sind AND-Verknüpfungen und können als Minimum-Funktion aufgefasst werden.

e.g.
 $Y = (\overline{A} \& B \& C) \# (A \& \overline{B} \& C) \# (A \& B \& \overline{C}) \# (A \& B \& C)$

1.5 Konjunktive Normalform (KNF)

Die konjunktive Normalform hat keine grosse Bedeutung in der Praxis. Sie kann als AND-Verknüpfung aller Maxterme aufgefasst werden.

e.g.
 $Y = (A \# B \# C) \& (A \# B \# \overline{C}) \& (A \# \overline{B} \# C) \& (\overline{A} \# B \# C)$

1.6 Umformung für Realisierung mit NAND

Um eine Schaltung mit NAND zu realisieren, sucht man zuerst die DNF der Schaltung. Danach kann man mit der doppelten Inversion arbeiten, womit die Funktion unverändert bleibt. Speziell ist zu beachten ist die Gruppierung mit der doppelten Inversion, diese ist sehr wichtig bei der Verwendung von NAND-Elementen mit nur zwei Eingängen.

e.g. (Für eine bessere visuelle Übersicht ist hier eine andere Schreibweise verwendet)

Realisierung mit NAND	
$Y = A1 \& A0 \& B0 \& B1$	$Y = A1 \& A0 \& B0 \& B1$
$Y = \overline{\overline{A1 \& A0 \& B0 \& B1}}$	$Y = \overline{\overline{(A1 \& A0 \& B0 \& B1)}}$
$Y = \overline{\overline{\overline{\overline{A1 \& A0 \& B0 \& B1}}}}$	$Y = \overline{\overline{\overline{\overline{(A1 \& A0) \& \overline{\overline{(B0 \& B1)}}}}}}$

2 Karnaugh-Diagramm

2.1 Karnaugh-Diagramm mit Wahrheitstabelle 4 Variabeln

Um mit einem Karnaugh-Diagramm optimal arbeiten zu können gibt es folgende Darstellungsvariante: Diese ermöglicht eine sehr schnelle Übertragung der Werte aus der Wahrheitstabelle in das Diagramm.

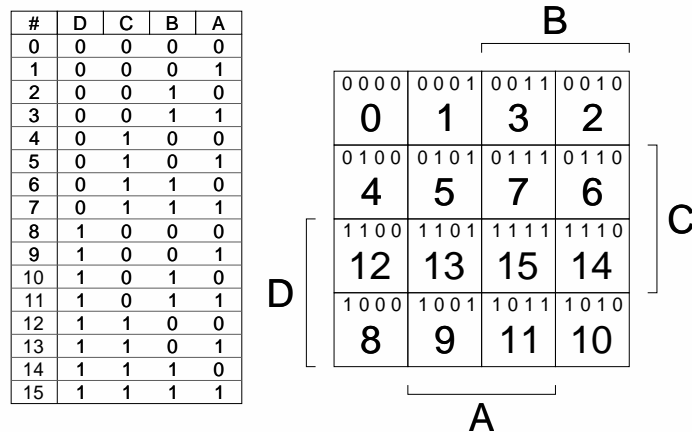


Abbildung 1: Wahrheitstabelle und Karnaugh-Diagramm

tabelle in das Diagramm. Man kann dabei im Stile eines Algorithmus die Werte übertragen. Man muss dabei immer die Nummer drei und vier vertauschen.

2.2 Karnaugh-Diagramm mit Wahrheitstabelle 3 Variabeln

Das Karnaugh-Diagramm mit der zugehörigen Wahrheitstabelle sieht wie folgt aus:

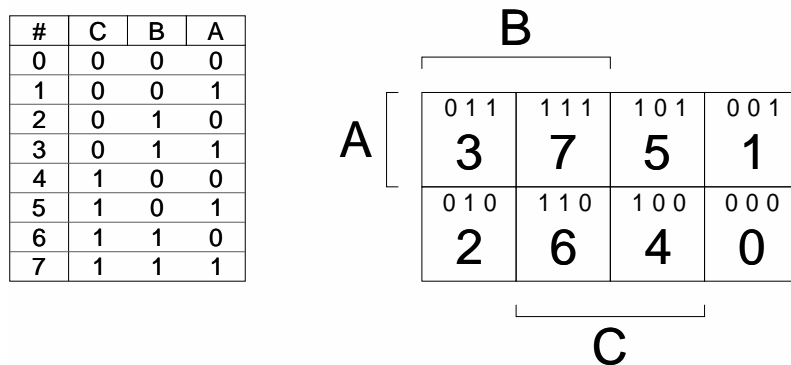


Abbildung 2: Wahrheitstabelle und Karnaugh-Diagramm

2.3 Karnaugh-Diagramm - EX-OR und EX-NOR

Die EX-OR Funktion in einem Karnaugh-Diagramm lässt sich einfach an der Schachbrettförmigen Verteilung der Nullen und Einsen erkennen. Anhand der Wahrheitstabelle kann man diese Funktion auch sehr gut erkennen.

		B		
		┌───┐	└───┘	
	0000	0001	0011	0010
	0	1	0	1
	0100	0101	0111	0110
	1	0	1	0
	1100	1101	1111	1110
	0	1	0	1
D	┌───┐	└───┘	└───┘	└───┘
	1000	1001	1011	1010
	1	0	1	0
		A		

Abbildung 3: Karnaugh-Diagramm EX-OR und EX-NOR

Bei diesem etwas komplexeren Beispiel handelt es sich um folgende Funktion:

$$((A\$B)\&(C!\$D))\#((A!\$B)\&(C\$D))$$

2.4 Karnaugh-Diagramm - Spezialfälle

Das KV-Diagramm für vier Variablen hat genau genommen eine Kugelform. Daher sind Plätze, die sich an allen Aussenseiten des Diagramms gegenüberliegen, einander benachbart.

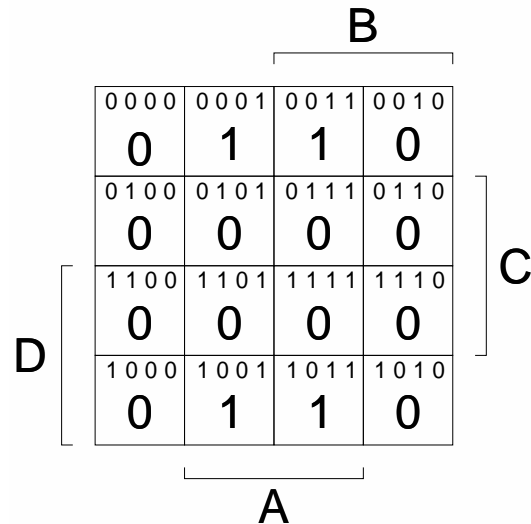


Abbildung 4: Karnaugh-Diagramm $A \& !C$

Dies gilt auch für die Ecken, Bei folgendem Beispiel ergibt sich daher $!A \& !C$

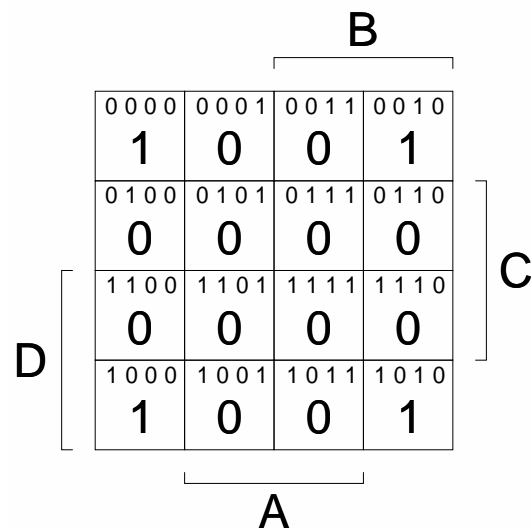


Abbildung 5: Karnaugh-Diagramm $!A \& !C$

3 BCD-7-Segment

3.1 BCD-7-Segment-Codeumsetzer

Der BCD-Code wird in grossem Umfang angewendet. Entsprechend häufig sollen BCD-codierte Informationen über 7-Segment-Anzeigeeinheiten ausgegeben werden. Um dies zu realisieren stellt man am besten das Karnaugh-Diagramm für jedes Segment auf und liest die Disjunktive Normalform heraus:

#	BCD-Code				7-Segment-Code						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	0	1

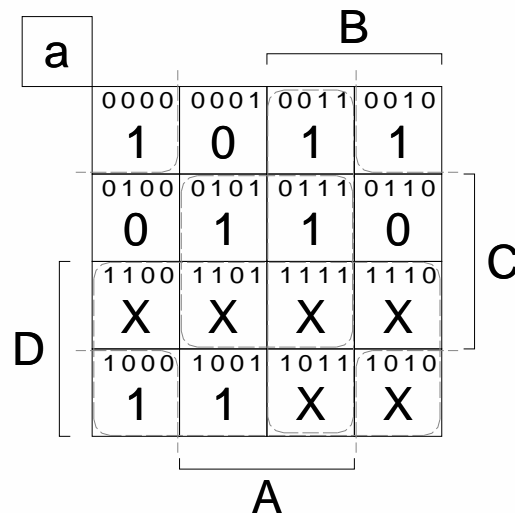


Abbildung 6: BCD-7-Segment

Die Disjunktiven Normalformen für die Segmente lauten daher:

Segment a: $D\#(!A\&!C)\#(A\&C)\#(A\&B)$

Segment b: $!C\#(A\&B)\#(!A\&!B)$

Segment c: $A\#!B\#C$

Segment d: $(!A\&B)\#(!A\&!C)\#(B\&!C)\#(A\&!B\&C)$

Segment e: $(!A\&B)\#(!A\&!C)$

Segment f: $D\#(!A\&!B)\#(!A\&C)\#(!B\&C)$

Segment g: $(!A\&B)\#(!B\&C)\#(B\&!C)\#D$

4 Code

4.1 Elementares (Einerkomplement, Zweierkomplement)

Die Differenz zweier Zahlen a und b ist definiert durch $D = a - b$. Man kann sie aber auch auf einem anderen Weg berechnen: $D = a + b^* - R$ mit $b^* = R - b$, das heisst b^* ist das Komplement zu b . Dieser Umweg kann sich lohnen, wenn die Rechnung dadurch einfacher wird.

Komplemente:

- $R = B^N$ ist das B-Komplement (B: Basis, N: Stellenzahl). Für das Binärsystem ist $B = 2$ und damit das Zweierkomplement

- $R = B^N - 1$ ist das B-1-Komplement. Im Dualen ist es das Einerkomplement.

Die invertierte abzuziehende Zahl wird oft als Einerkomplement bezeichnet. Wenn man zu dem Einerkomplement 1 hinzuaddiert, erhält man das gesuchte Zweierkomplement.

e.g.

$$\begin{array}{r} 1\ 0\ 1\ 1\ 1\ 1 \\ -\ 1\ 1\ 0\ 1\ 1 \end{array} = 47 - 27$$

$$\begin{array}{r} 0\ 1\ 1\ 0\ 1\ 1 \\ 1\ 0\ 0\ 1\ 0\ 0 \\ +1\ 1 \\ 1\ 0\ 0\ 1\ 0\ 1 \end{array} \begin{array}{l} \text{abzuziehende Zahl} \\ \text{invertierte abzuziehende Zahl} \\ \text{addieren um Zweierkomplement zu bilden} \\ \text{Komplement} \end{array}$$

$$\begin{array}{r} 1\ 0\ 1\ 1\ 1\ 1 \\ 1\ 0\ 0\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1\ 0\ 0 \end{array} \begin{array}{l} = 47 \\ \text{Komplement (Addieren)} \\ = 20 \end{array}$$

Begründung der Korrektheit:

$$D = a + b^* - R \quad \leftrightarrow \quad a + b^* = D + R \quad \leftrightarrow \quad 20 = 47 - 27$$

4.2 BCD-Code

Die Abkürzung BCD steht für Binary Coded Decimal. Dabei wird jede Stelle einer Dezimalzahl durch eine 4 Bit Dualzahl codiert. Da es nur 10 unterschiedliche Dezimalzahlen gibt, mit 4 Bit aber $2^4 = 16$ Zahlen codiert werden können, bleiben 6 Möglichkeiten ungenutzt. Es gibt theoretische viele Möglichkeiten, 10 Dezimalzahlen in irgendeiner Form durch 4 Bit zu codieren. Am häufigsten benutzt man jedoch die normale Dezimal-Dual Umrechnung, bei der die einzelnen Stellen der Dualzahl die Wertigkeiten $8 - 4 - 2 - 1$ besitzen. Dieser $8 - 4 - 2 - 1$ BCD-Code wird einfach BCD-Code genannt. Mit den restlichen 6 Stellen werden manchmal noch die Hexadezimalstellen codiert.

8 - 4 - 2 - 1 BCD-Code				
#	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
b	1	0	1	1
C	1	1	0	0
d	1	1	0	1
E	1	1	1	0
F	1	1	1	1

4.3 GRAY-Code

Der GRAY-Code ist ein einschrittiger Code, dies bedeutet dass er beim Übergang von einer Tetrade auf die nächste nur ein Bit ändert. Dies ist nicht so Fehleranfällig wie mehrschrittige Codes. Seine Anwendungsbereichs sind zum Beispiel Winkelkodierung. Er ist nicht sehr praktisch um zu implementieren.

GRAY-Code				
#	G	R	A	Y
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

4.4 3-Exzess-Code

Der Exzess-3-Code hat keine Stellenbewertung für die Dezimalziffer 0 – 9. Die verschiedenen Kombinationen aus 0 und 1 sind den Dezimalzahlen zugeordnet, deshalb wird der Exzess-3-Code auch als ein Zuordnungscode bezeichnet. Der Exzess-3-Code ist durch einfache Korrekturen zum Rechnen geeignet. Erfolgt bei einer Addition einer Dezimalstelle ein Zehnerübertrag, so entsteht auch in der Binären Rechnung ein Übertrag. Daraus folgt, dass als Korrektur zum Ergebnis 0011 addiert werden muss. Falls kein Übertrag stattfindet, muss dem Ergebnis als Korrektur 1101 addiert werden.

e.g.

0100 1011 = Exzess-3 Tetraden für 18

0100 0110 = Exzess-3 Tetraden für 13

1001 0001

1101 0011 = Exzess-3 Korrekturtetraden

0110 0100 = Exzess-3 Tetraden für 31

3-Exzess-Code				
#	D	C	B	A
	0	0	0	0
	0	0	0	1
	0	0	1	0
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

4.5 Aiken-Code

Der Aiken-Code ist für elektronische Zähler sehr geeignet. Eine interessante Eigenschaft des Aiken-Code ist, dass bei ungeraden Dezimalzahlen die letzte Binärstelle eine 1 und bei geraden Dezimalzahlen die letzte Binärstelle eine 0 hat, wobei die erste Binärstelle bei den Werten 0 – 4 das Zeichen 0 und bei den Werten 5 – 9 das Zeichen 1 hat. Bei Aiken-Code bewirkt ein Dezimalübertrag auch eine Tetradenübertrag.

e.g.

1101 = Aiken Tetrade für 7

1100 = Aiken Tetrade für 6

1001 = Aiken Pseudotetrade und Übertrag

-0110 = Korrekturtetrade (-6)

0001 0011 = 2 Aiken Tetrade für 13

Aiken-Code				
#	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
	0	1	0	1
	0	1	1	0
	0	1	1	1
	1	0	0	0
	1	0	0	1
	1	0	1	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

5 VHDL - VHSIC Hardware Description Language

5.1 Grundelemente der Programmierung

Hier nur das wichtigste zu VHDL:
VHDL ist nicht Case Sensitive.

Entwurfsobjektet (Design Entities)

Schnittstellenbeschreibung - Entity (Interface Description)

```
entity entity_show is
generic generic_list;
port interface_list;
declarations
```

```
begin
concurrent_statements;
end entity_show;
```

Implementierung - Architectural Body (Architecture)

```
architecture identifier of
entity_show is
declarations
```

```
begin
concurrent_statements;
end identifier;
```

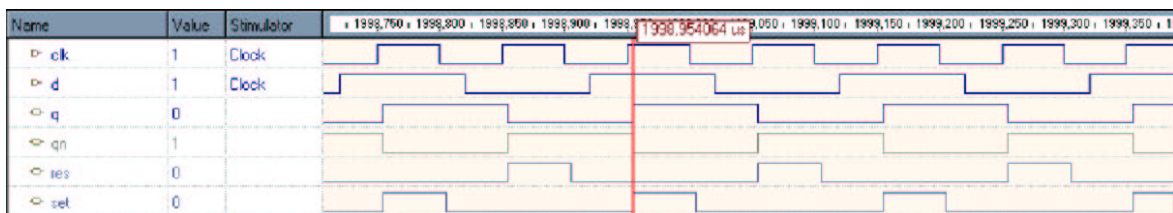
Es unterscheidet strikt zwischen Signalen und Variablen.
Variablen werden mit := und Signale werden mit <= zugewiesen.

5.2 Verhaltensbeschreibung eines D-Flip-Flops

```

ENTITY dff IS
  PORT (d      : IN bit := '1';      -- data
        clk    : IN bit := '1';      -- clock
        set    : INOUT bit;          -- set
        res    : INOUT bit;          -- reset
        q      : OUT bit;            -- dff-out
        qn     : OUT bit);           -- dff-out inverted
END dff;
ARCHITECTURE behavior OF dff IS
BEGIN
  PROCESS
  BEGIN
    IF      d = '1' AND clk = '1' THEN
      set <= '1';
      res <= '0';
    ELSIF  d = '0' AND clk = '1' THEN
      set <= '0';
      res <= '1';
    ELSIF  d = '0' AND clk = '0' THEN
      set <= '0';
      res <= '0';
    ELSIF  d = '1' AND clk = '0' THEN
      set <= '0';
      res <= '0';
    END IF;
    IF      set = '1' AND res='0' THEN  -- set f"ur RS-FF
      q <= '1';
      qn <= '0';
    ELSIF  set = '0' AND res= '1' THEN  -- reset f"ur RS-FF
      q <= '0';
      qn <= '1';
    ELSIF  set = '0' AND res='0' THEN  -- save f"ur RS-FF
      -- nichts wird ge"andert, FF ist schon initialisiert
    END IF;
  END PROCESS;
END behavior;

```



5.3 BCD Beispiel

In diesem Beispiel wird ein Nibble (4-Bit Binäre Nummer) in BCD gewandelt, um diese auf zwei 7-Segment-Anzeigen auszugeben:

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;

entity BIN_BCD_CONV_4BIT is port (
    bin_data : IN STD_LOGIC_VECTOR (3 DOWNTO 0); -- binary inputs
    bcd_data_u : OUT STD_LOGIC_VECTOR (3 DOWNTO 0); -- units of bcd output
    bcd_data_t : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)); -- tens of bcd output

end BIN_BCD_CONV_4BIT;

ARCHITECTURE arch_bin_bcd_conv_4bit OF bin_bcd_conv_4bit IS
BEGIN
    PROCESS
    BEGIN
        CASE bin_data IS
            WHEN "0000" => bcd_data_u <= "0000"; bcd_data_t <= "0000";
            WHEN "0001" => bcd_data_u <= "0001"; bcd_data_t <= "0000";
            WHEN "0010" => bcd_data_u <= "0010"; bcd_data_t <= "0000";
            WHEN "0011" => bcd_data_u <= "0011"; bcd_data_t <= "0000";
            WHEN "0100" => bcd_data_u <= "0100"; bcd_data_t <= "0000";
            WHEN "0101" => bcd_data_u <= "0101"; bcd_data_t <= "0000";
            WHEN "0110" => bcd_data_u <= "0110"; bcd_data_t <= "0000";
            WHEN "0111" => bcd_data_u <= "0111"; bcd_data_t <= "0000";
            WHEN "1000" => bcd_data_u <= "1000"; bcd_data_t <= "0000";
            WHEN "1001" => bcd_data_u <= "1001"; bcd_data_t <= "0000";
            WHEN "1010" => bcd_data_u <= "0000"; bcd_data_t <= "0001";
            WHEN "1011" => bcd_data_u <= "0001"; bcd_data_t <= "0001";
            WHEN "1100" => bcd_data_u <= "0010"; bcd_data_t <= "0001";
            WHEN "1101" => bcd_data_u <= "0011"; bcd_data_t <= "0001";
            WHEN "1110" => bcd_data_u <= "0100"; bcd_data_t <= "0001";
            WHEN "1111" => bcd_data_u <= "0101"; bcd_data_t <= "0001";
            WHEN others => bcd_data_u <= "----"; bcd_data_t <= "----";
        END CASE;
    END PROCESS;
END arch_bin_bcd_conv_4bit;
```

6 Multivibratoren

Es gibt drei Typen von Multivibratoren:

- Der **Monostabile Multivibrator** (one-shot) hat nur einen stabilen Zustand. Er erzeugt einen einzelnen Puls, getriggert von einem Eingangssignal.
- Der **Bistabile Multivibrator** (flip-flop) hat zwei stabile Zustände. Er kann die zwei Zustände SET und RESET unendlich lange behalten. Oft gebraucht als Grundbaustein für Zähler, Register und Speicher.
- Der **Instabiler Multivibrator** hat keinen stabilen Zustand. Er wird vor allem als Oszillator eingesetzt um periodische Pulsformen für Zeitsteuerungen zu erzeugen.

7 Flip Flops

7.1 Unterscheidung nach Taktart

Taktart			
Transparent		Nicht Transparent	
Ungetaktet	Taktzustand- gesteuert	Flanken-getriggert	Master-Slave pulsgetriggert
Ausgang reagiert immer sofort auf Eingang	Ausgang reagiert auf Eingang, wenn Clock aktiv	Ausgang reagiert nur im Zeitpunkt der entspr. Clock-Flanke auf Eingang	Master speichert bei pos. Clock-Flanke den Eingangszustand. Auf neg. Clock-Flanke wird Zustand im Master auf Slave weitergegeben.

7.2 Unterscheidung nach Typ

FF-Typ		
RS-FF	D-FF	JK-FF
Grundbaustein		Nie transparent! Universalstes FF.
	$D \hat{=} S$ beim RS-FF	$J \hat{=} S$ und $K \hat{=} R$ des RS-FF
Zustand $S=R=1$ verboten	Zustand $R=S$ nicht möglich, da $R=\bar{S}$	Bei $J=K=1$ und Clock wechselt Q zu \bar{Q} (toggle)

7.3 RS-FF

7.3.1 Ungetaktetes RS-FF

Eigenschaften

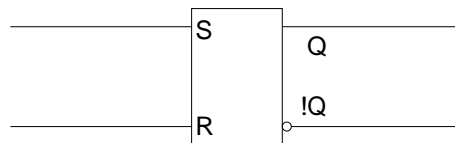
Transparent

Beim Einschalten der Betriebsspannung stellt sich ein undefinierter Zustand ein. Er ist von den Eigenschaften und Toleranzen der Bauteile abhängig. Die Bezeichnung $I = 0$ im Schaltzeichen eines RS-FF besagt, dass direkt beim Einschalten der Betriebsspannung der Speicherzustand = 0 ist, d.h. der Ausgang Q führt ein 0-Signal.

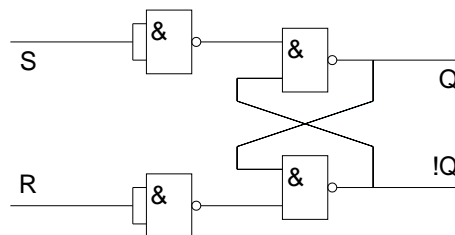
Zeichen

S		Eingang Set
R		Eingang Reset
Q		Ausgang
$!Q$		Invertierter Ausgang

Symbol



Realisierung

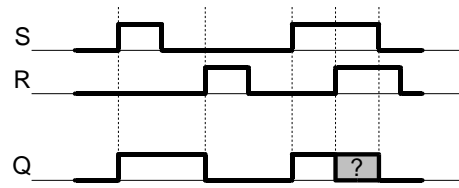


Wahrheitstabelle

S	R	Q_{n+1}	$!Q_{n+1}$	
0	0	Q_n	$!Q_n$	(Speichern)
0	1	0	1	(Reset)
1	0	1	0	(Set)
1	1	–	–	(Undefiniert)

Timing Diagram

7 FLIP FLOPS



7.3.2 Taktzustandgesteuertes RS-FF

Eigenschaften

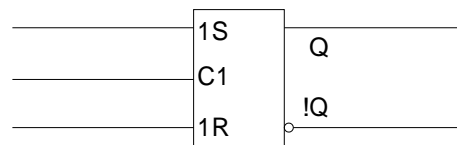
Transparent

Ein RS-FF hat einen Setzeingang S und einen Rücksetzeingang R . Es besitzt zwei stabile Zustände, wirkt also als Speicherelement. Ein Impuls mit dem Wert 1 an S ergibt den Ausgangswert 1, ein Impuls an R den Ausgangswert 0.

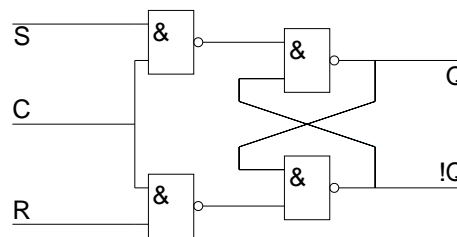
Zeichen

1S	Eingang Set (abhängig von C1)
1R	Eingang Reset (abhängig von C1)
C1	Taktsignal 1 (steuert 1R und 1S)
Q	Ausgang
!Q	Invertierter Ausgang

Symbol



Realisierung

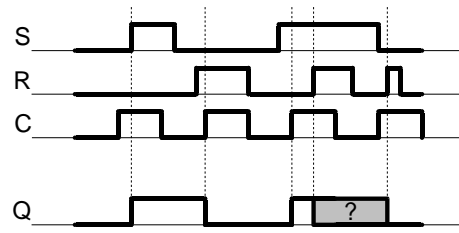


Wahrheitstabelle

S	R	C	Q_{n+1}	$!Q_{n+1}$	
X	X	0	Q_n	$!Q_n$	(Speichern)
0	0	1	Q_n	$!Q_n$	(Speichern)
0	1	1	0	1	(Reset)
1	0	1	1	0	(Set)
1	1	1	–	–	(Verboten)

Timing Diagram

7 FLIP FLOPS



7.3.3 Master Slave RS-FF (Pulsgetriggertes RS-FF)

Eigenschaften

Nicht transparent

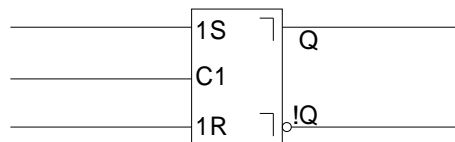
Pulsgetriggert: **Zur Triggerung wird ein vollständiger Puls benötigt** (ansteigende und abfallende Clock-Flanke)

Schaltet bei negativer Clock-Flanke

Zeichen

1S		Eingang Set (abhängig von C1)
1R		Eingang Reset (abhängig von C1)
C1		Taktsignal 1 (steuert 1R und 1S)
Q*		Ausgang Master
Q		Ausgang Slave

Symbol



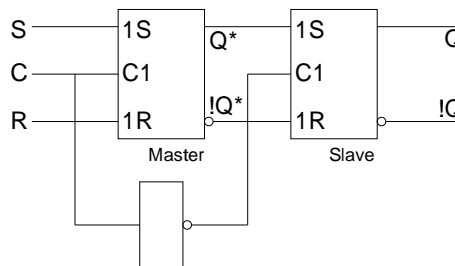
Funktionsweise

Clock: high

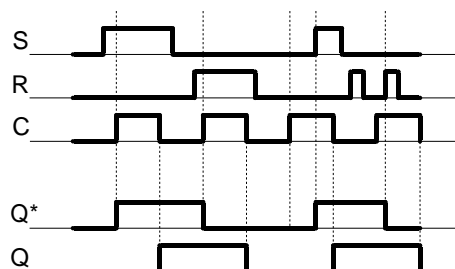
Master übernimmt ständig Zustände von S und R und überträgt diese auf Q*. Q ändert nicht.

Clock: neg. Flanke

Q* bleibt konstant, unabhängig von R und S. Slave übernimmt Zustände des Masters.



Timing Diagram



7.3.4 Flankengetriggertes RS-FF

Eigenschaften

Nicht transparent

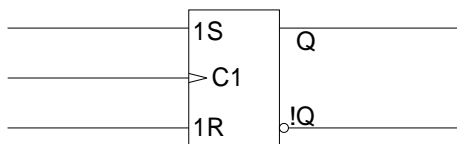
Schaltet bei positiver oder negativer Clock-Flanke

Die entsprechende Flanke löst einen Impuls aus, der gerade lang genug ist, um als Taktsignal für das Flipflop zu dienen.

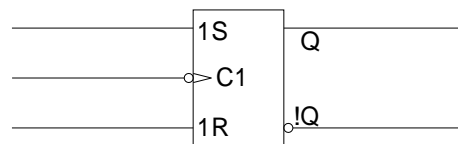
Zeichen

1S		Eingang Set (abhängig von C1)
1R		Eingang Reset (abhängig von C1)
C1		Taktsignal 1 (steuert 1R und 1S)
Q		Ausgang

Symbol

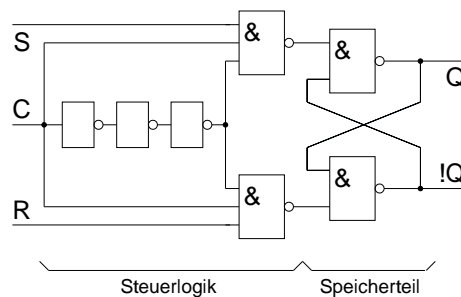


Positiv flankengetriggert

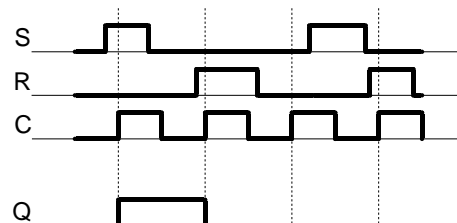


Negativ flankengetriggert

Realisierung



Timing Diagram



7.4 D-FF

7.4.1 Taktzustandsgesteuertes D-FF (D-Latch)

Eigenschaften

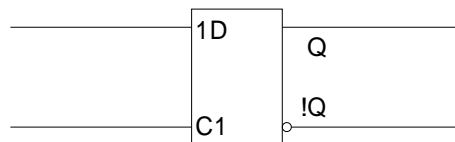
Transparent

Das D-FF heisst eigentlich Delay-FF. Es dient dazu, ein Eingangssignal so lange zu verzögern, bis das Taktsignal kommt. Erst dann wird das Eingangssignal an der Ausgang Q übergeben.

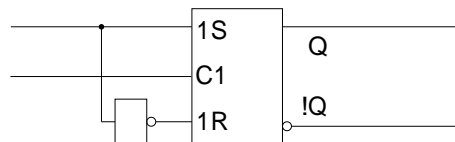
Zeichen

1D	Eingang Set/Reset (abhängig von C1)
C1	Taktsignal 1 (steuert 1D)
Q	Ausgang

Symbol



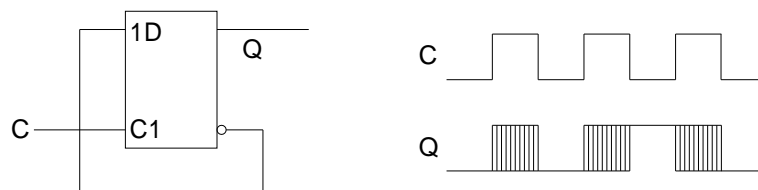
Realisierung



Wahrheitstabelle

D	C	Q_{n+1}	$!Q_{n+1}$	
X	0	Q_n	$!Q_n$	(Speichern)
0	1	0	1	(Reset)
1	1	1	0	(Set)

Verhalten mit Rückführung $D = !Q$



Ausgang Q oszilliert (unstabil).

7.4.2 Flankengetriggertes D-FF

Eigenschaften

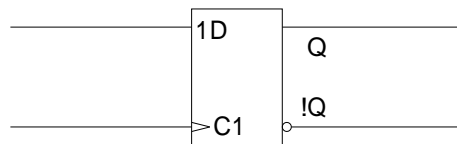
Nicht transparent

Das D-FF hat nur einen Signaleingang und einen Takteingang. Der Ausgangszustand ist jeweils eindeutig. Er wechselt bei jedem Eingangsimpuls.

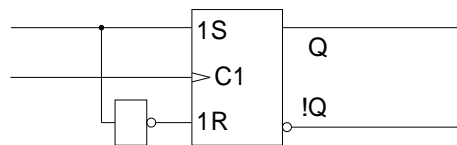
Zeichen

1D	Eingang Set/Reset (abhängig von C1)
C1	Taktsignal 1 (steuert 1D)
Q	Ausgang

Symbol



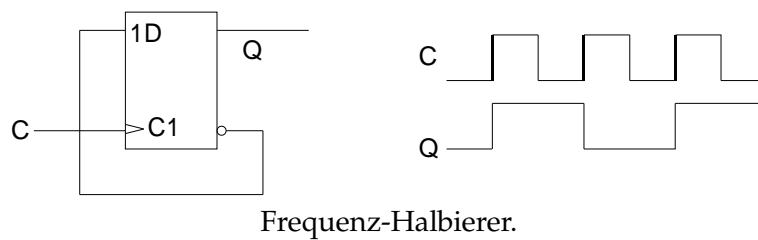
Realisierung



Wahrheitstabelle

D	C	Q_{n+1}	$!Q_{n+1}$	
X	0	Q_n	$!Q_n$	(Speichern)
0	1	0	1	(Reset)
1	1	1	0	(Set)

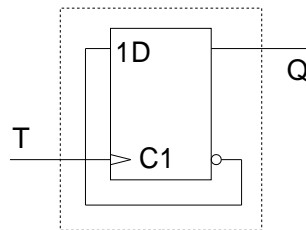
Verhalten mit Rückführung $D = !Q$



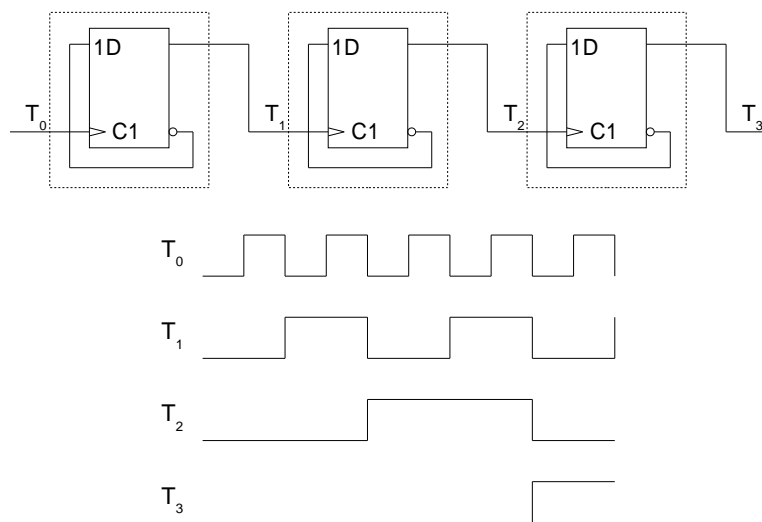
7.5 Toggle-FF

Ein Toggle-FF ist ein Flankengetriggertes D-FF mit Rückführung des Ausgangs \overline{Q} an den Eingang D . Es wird als Frequenzhalbierer eingesetzt.

Symbol



Die Frequenz kann auch durch eine Potenz von 2 geteilt werden, indem mehrere Toggle-Flip-Flops kaskadiert werden.



7.6 JK-FF

Eigenschaften

Nicht transparent

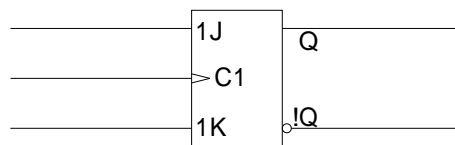
Universellstes und meistgebrauchtes FF

Der *J*-Eingang (Jump) entspricht dem Set-, der *K*-Eingang (Kill) dem Rücksetz-Eingang eines RS-Speichers.

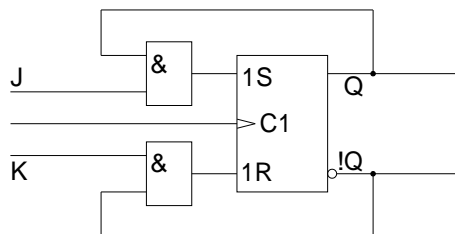
Zeichen

1J	Eingang Set (abhängig von C1)
1K	Eingang Reset (abhängig von C1)
C1	Taktsignal 1 (steuert 1D)
Q	Ausgang

Symbol



Realisierung



Wahrheitstabelle

J	K	Q_{n+1}	$!Q_{n+1}$	
0	0	Q_n	$!Q_n$	(Speichern)
0	1	0	1	(Reset)
1	0	1	0	(Set)
1	1	$!Q_n$	Q_n	(Toggle)

Schaltalgebraisch

$$Q_{n+1} = (J_n \& !Q_n) \# (!K_n \& Q_n)$$

8 Automaten

8.1 Daten zur Realisierung

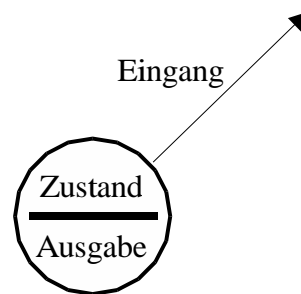
Übergangstabelle

Überg.	R	S
0 → 0	X	0
0 → 1	0	1
1 → 1	0	X
1 → 0	1	0

Überg.	J	K
0 → 0	0	X
0 → 1	1	X
1 → 1	X	0
1 → 0	X	1

Überg.	D
0 → 0	0
0 → 1	1
1 → 1	1
1 → 0	0

Für das Zustandsdiagramm werden Kreissymbole verwendet, die folgendermassen bezeichnet werden.



Das Symbol wird als eine Art Symbollegende in das Zustandsdiagramm mit den konkreten Variablen eingezeichnet (zum Beispiel rechts unten).