

CDT Zusammenfassung


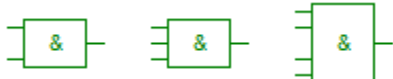
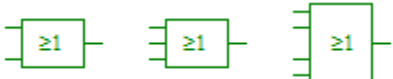
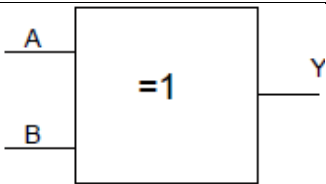
René Bernhardsgrütter, 08.04.2012

1 Basics und Darstellung von Zahlen

Bit ← binary digit: Eine Ziffer die nur 2 mögliche Werte annehmen kann.

Wenn das System kein Gedächtnis hat, dann spricht man von einem **kombinatorischen** System

Für N Eingänge hat man 2 mögliche Eingangskombinationen.

NOT	$Y = !A$	
AND	$Y = A \& B$	
OR	$Y = A \# B$	
XOR	$Y = A \$ B$ Wenn ein Eingang immer auf 1, dann ist das Gatter ein Inverter.	

2 Zahlensysteme

Name	Basis	Wertebereich
Dezimalsystem	10	0-9
Hexadezimalsystem	16	0-15 => 0-F
Binärsystem	2	0-1

Anzahl darstellbare Zahlen immer: Bei n Ziffern => Basisⁿ

Binärsystem: **MSB (Most Significant Bit d.h. Bit mit höchster Wertigkeit)** ist die am weitesten links stehende Ziffer. **LSB (Least Significant Bit d.h. Bit mit niedrigster Wertigkeit)** ist die am weitesten rechts stehende Ziffer.

Nibble = 4 Bits

Byte = 8 Bits = 2 Nibbles

Word = mehr als 8 Bits, oft eine Gruppe von 2 Bytes (= 16 Bit)

double Word = oft eine Gruppe von 32 Bits (= 4 Bytes)

2.1 Umrechnung

Wenn nicht von einem hohen System (Basis > 2), dann immer über das Binärsystem, welches als Zwischenschritt dient, zum Zielsystem.

- 101111000010 (Bin) = BC2 (Hex)
 - 4 Bit sind gruppiert für jede Hexadezimal-Ziffer
- 101111000010 (Bin) = 5702 (Oktal)
 - 3 Bit sind gruppiert für jede Oktal-Ziffer

Zuverlässige und merkbare Methode für alle Systeme:

z.B. Dezimal → Binär

- Durch 2 dividieren
- 47 (Basis 10) = 101111 (Basis 2)
- $47 : 2 = 23$ Rest 1
- $23 : 2 = 11$ Rest 1
- $11 : 2 = 5$ Rest 1
- $5 : 2 = 2$ Rest 1
- $2 : 2 = 1$ Rest 0
- $1 : 2 = 0$ Rest 1

z.B. Dezimal → Hexadezimal

- Durch 16 dividieren
- 453 (Basis 10) = 1C5 (Basis 16)
- $453 : 16 = 28$ Rest 5
- $28 : 16 = 1$ Rest 12 (C)
- $1 : 16 = 0$ Rest 1

Binär → Oktal

- 3-Bit gruppieren (immer mit LSB anfangen)
- 1010011111 (Basis 2) = 1237 (Basis 8)
- 001 010 011 111
- 1 2 3 7

Oktal → Binär

- Ziffer als 3-Bit schreiben
- 6513 (Basis 8) = 110101001011 (Basis 2)
- 110 101 001 011
- 6 = 110, 5 = 101, 1 = 001, 3 = 011

Binär → Hexadezimal

- 4-Bit gruppieren (immer mit LSB anfangen)
- 1010011111 (Basis 2) = 29F (Basis 16)
- 0010 1001 1111
- 2 9 F

Hexadezimal → Binär

- Ziffer als 4-Bit schreiben
- 7F3C (Basis 16) = 111111100111010 (Basis 2)
- 111 1111 0011 1100
- 7 = 0111, F = 1111, 3 = 0011, C = 1100

3 Schaltalgebra

Die Prioritätsreihenfolge sieht folgendermassen aus:

1.	()	Klammern	
2.	!	Negation	NOT
3.	&	Konjunktion	AND
4.	#	Disjunktion	OR

3.1 Axiome

X_1, X_2, \dots, X_n sind jeweils die verfügbaren Eingänge.

OR: Falls $X_1 = X_2 = \dots = X_n = 0$, dann ist die Schaltung = 0, sonst = 1.

AND: Falls $X_1 = X_2 = \dots = X_n = 1$, dann ist die Schaltung = 1, sonst = 0.

XOR: $1 \$ 1 = 0, 0 \$ 0 = 0, 0 \$ 1 = 1, 1 \$ 0 = 1$
 ➔ Wenn Eingänge verschieden, Ausgang = 1

3.2 Theoreme

Verknüpfung mit 0: $0 \& X = 0$
 $0 \# X = X$

Verknüpfung mit 1: $1 \& X = X$
 $1 \# X = 1$

Verknüpfung mit sich selbst: $X \& X = X$
 $X \# X = X$
 $X \& !X = 0$
 $X \# !X = 1$

Kommutativgesetze: $X_1 \& X_2 = X_2 \& X_1$
 $X_1 \# X_2 = X_2 \# X_1$

Assoziativgesetze: $(X_1 \& X_2) \& X_3 = X_1 \& (X_2 \& X_3)$
 $(X_1 \# X_2) \# X_3 = X_1 \# (X_2 \# X_3)$
 Die Klammern haben höchste Priorität (zuerst ausgeführt)

Distributivgesetze: $(X_1 \# X_2) \& X_3 = (X_1 \& X_3) \# (X_2 \& X_3)$
 $(X_1 \& X_2) \# X_3 = (X_1 \# X_3) \& (X_2 \# X_3)$

Vereinfachungsgesetze: $X_1 \# (X_1 \& X_2) = X_1$
 $X_1 \& (X_1 \# X_2) = X_1$
 $X_1 \# (!X_1 \& X_2) = X_1 \# X_2$
 $X_1 \& (!X_1 \# X_2) = X_1 \& X_2$

Gesetze von de Morgan (Inversionsgesetze):
 $!(X_1 \& X_2 \& \dots \& X_n) = !X_1 \# !X_2 \# \dots \# !X_n$
 $!(X_1 \# X_2 \# \dots \# X_n) = !X_1 \& !X_2 \& \dots \& !X_n$

Satz von Shannon: $!F(X_1, X_2, \dots, X_n, \&, \#) = F(!X_1, !X_2, \dots, !X_n, \#, \&)$

4 DNF: Disjunktive Normalform

Man verwendet die DNF, um von einem Ausgangszustand zu einem Zielzustand, der sogenannten Endlösung, zu kommen. Man möchte herausfinden, welche Kombinationen zur Lösung führen. Dazu macht man die DNF.

Die Zwischenlösungen der einzelnen Verknüpfungen heissen **Minterme**. Solche die 1 ergeben heissen **gute Minterme**, die anderen heissen **schlechte Minterme**.

Für n Variablen (Eingänge) gibt es 2^n Minterme.

K ist der Ausgang. K ist die OR-Verknüpfung **aller** guten Minterme. **K ist die disjunktive Normalform.** Sieht etwa so aus:

$$K = (..&..&..&) \# (..&..&..&)...$$

Vorgehen:

1. Wahrheitstabelle machen und gute Minterme herausuchen.
2. Diese mit den Gattern (also den logischen Verknüpfungen) aufschreiben.
3. Die Aussage vereinfachen, mit dem De Morgan Theorem oder den andere Regeln.
Darauf achten, dass wenn möglich nur Gleiche Gattertypen verwendet werden (dafür wird meistens NOT gebraucht).

5 KNF: Konjunktive Normalform

Statt für $K = 1$, wie bei der DNF, schreibt man bei der KNF die Funktionen für $K = 0$. Man invertiert dafür das K . Dies kann besser sein, wenn man so weniger schreiben oder rechnen muss.

Man muss die Maxterme verknüpfen: Ein Maxterm ist eine OR-Verknüpfung, welche die Eingangsgrößen entweder in direkter oder in negierter Form enthält.

Jeder Maxterm entspricht einer Zeile der WT, wobei die Variable in negierter Form einzusetzen ist, falls in der WT an der entsprechenden Stelle eine 1 steht, und in direkter Form, falls eine 0 steht.

$$!K = (..&..&..&) \# (..&..&..&)...$$

Dann das De Morgan Theorem anwenden:

$$K = (.. \# .. \# ..) \& (.. \# .. \# ..) \& (.. \# .. \# ..)$$

Die konjunktive Normalform ist die AND Verknüpfung **aller** Maxterme, die zu den Zeilen mit $K = 0$ gehören.

6 Signed und unsigned numbers

Bei einer **vorzeichenlosen Operation** zeigt das **Carry-Flag** (CF) eine Bereichsverletzung (Übertrag oder Unterlauf).

Bei einer **vorzeichenbehafteten Operation** zeigt das **Overflow-Flag** (CF XOR (Übertrag zum MSB des Resultats)) eine Bereichsverletzung (Über- oder Unterlauf).

6.1 Addition von vorzeichenlosen Zahlen:

Regeln für Addition

(Übertrag Resultat)

$0 + 0 = 00$
 $0 + 1 = 01$
 $1 + 0 = 01$
 $1 + 1 = 10$
 $1 + 1 + 1 = 11$

$$\begin{array}{r}
 101101 \\
 + 001010 \\
 \hline
 110111
 \end{array}
 \quad \downarrow$$

$$\begin{array}{r}
 110110 \quad (6 \text{ Bits}) \\
 + 101000 \quad (6 \text{ Bits}) \\
 \hline
 1011110 \quad (1 + 6 \text{ Bits})
 \end{array}$$

Carry-Bit = Übertrag → Nicht genug Bits vorhanden, um die Zahl korrekt darzustellen!

6.2 Subtraktion von vorzeichenlosen Zahlen:

Hier muss man die Zahlen wirklich subtrahieren, nicht wie bei den vorzeichenbehafteten Zahlen (dort kann man sie addieren)!

Regeln für Subtraktion

(Borrow Resultat)

$0 - 0 = 00$
 $0 - 1 = 11$
 $1 - 0 = 01$
 $1 - 1 = 00$
 $0 - 1 - 1 = 10$

$$\begin{array}{r}
 101101 \\
 - 001010 \\
 \hline
 100011
 \end{array}
 \quad \downarrow$$

$$\begin{array}{r}
 010110 \quad (6 \text{ Bits}) \\
 - 101000 \quad (6 \text{ Bits}) \\
 \hline
 1101110 \quad (1 + 6 \text{ Bits})
 \end{array}$$

6.3 Darstellung vorzeichenbehafteter Zahlen

Bei der Addition, Subtraktion kann ein Überlauf entstehen (Engl. **Overflow Bit = Überlauf**).

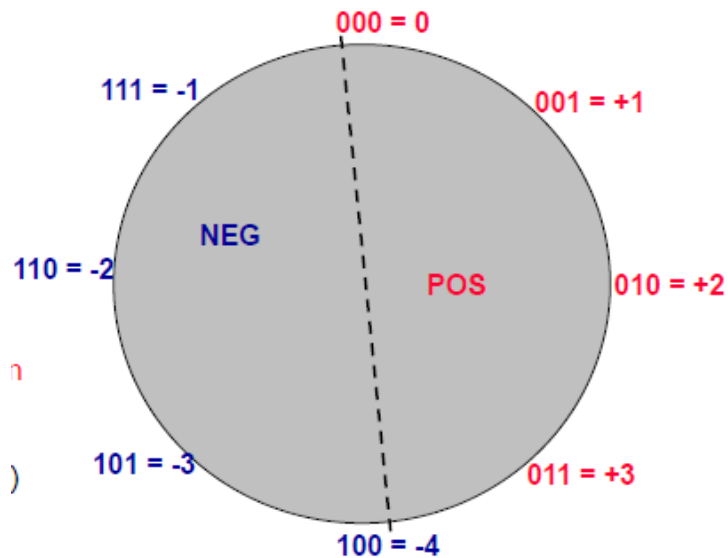
Bei vorzeichenbehafteten Zahlen ist das MSB (=Most Significant Bit, hier das linke) für die Darstellung des Vorzeichens reserviert:

- 0 = Positive Zahl
- 1 = Negative Zahl

Einerkomplement = Zahl invertieren.

Zweierkomplement = Zahlen invertieren und + 1 addieren. Immer für die Subtraktion verwenden!

Die Differenz zweier Zahlen ist die Addition des Zweierkomplements des Subtrahenden zum Minuenden.



Negative Zahlen = negativer Wert der ersten Stelle (MSB) plus die Summe der restlichen Stellen als positive Werte. So gehen keine Werte verloren (durch doppeltes Null).

7 Karunugh-Diagramm

Vereinfachung bis 4 Eingänge (nachher kompliziert). Immer gute 2^K -Minterme (=Gruppen) bilden, wo $Y=1$ ist. Möglichst grosse Gruppen.

8 Gray-Code

In drei Schritten zum Gray-Code aus Binärcode:

- X1: Dualzahl im **Binärcode**
- X2: **Rechts-Shift** der Dualzahl um 1 Bit
- X3: Modulo-2-Addition (**XOR-Verknüpfung**) von X1 und X2; dies ist die gewünschte Zahl im Graycode.