Ein bisschen Theorie

Dezimal, hexadezimal, oktal und binär.

Wenn man als Neuling in die Digitalelektronik einsteigt, wird man am Anfang vielleicht etwas unsicher, da man viele Bezeichnungen nicht versteht. Da ist die Rede von hexadezimalen Zahlen, von binären Bitmustern und von Abkürzungen, die fremd sind. Letztendlich stellt sich nachher heraus, dass alles gar nicht so kompliziert ist, wenn man die Zusammenhänge begriffen hat. Man muß etwas Zeit und Übung opfern, damit das Jonglieren von einem Zahlensystem in das andere leicht von der Hand geht.

Wir können eigentlich froh sein, dass die Römer uns ihr Zahlensystem nicht hinterlassen haben; es war keine technische Meisterleistung. Es wären vielleicht ganz andere Computer dabei herausgekommen. Erinnern wir uns an die Römischen Zahlen. Die 7 klassischen Zeichen kann man sich gut merken:

M = 1000 - D = 500 - C = 100 - L = 50 - X = 10 - V = 5 - I = 1.

Würden wir heute noch damit arbeiten, so hätten wir komplizierte Ausdrücke zu verarbeiten. Wir kämen auch nicht besonders weit mit diesen Zeichen, wenn man die Regel hinzuzieht, dass maximal drei gleiche Zeichen nacheinander folgen:

M M M I M = 3999 wäre unsere größte Zahl. Wer sich ein wenig mit Römischen Zahlen auseinandersetzen möchte, der kann sich des Programmes ROEMISCH.EXE auf der CD bedienen.

Dezimalzahlen

Kommen wir zu dem, was wir alle in der Schule gelernt haben: das Dezimalsystem. Die Zahl 10 ist hier die Grundlage für alle Berechnungen. Wahrscheinlich waren die 2 * 5 Finger auch ausschlaggebend für die Auswahl der Zahl 10.

Eine Dezimalzahl ist eine Folge von Dezimalziffern.

10.000.000	1.000.000	100.000	10.000	1.000	100	10	1
10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
Zehnmillionen	Million	Hunderttausend	Zehntausend	Tausend	Hundert	Zehn	Eins

Die Dezimalzahl 21834 können wir auch zerlegen in:

$2 * 10^4 + 1 * 10^3 + 8 * 10^2 + 3 * 10^1 + 4 * 10^0$

Die Rückwandlung ergibt natürlich wieder das Ausgangsergebnis:

2*10000 + 1*1000 + 8*100 + 3*10 + 4*1 = 21834

Formalistisch ausgedrückt: wir berechnen den Wert der Zahl, indem wir von rechts beginnen (quasi die nullrechteste Ziffer) mit dem Wert $10^0 = 1$. Danach kommt quasi die ‚erstrechteste' Ziffer mit dem Wert $10^1 = 10$ usw. usw.

Binärzahlen oder Dualzahlen

Eine Binärzahl ist genau so aufgebaut; allerdings wird ihr Wert nicht dargestellt aus Potenzen von 10, sondern aus Potenzen der Zahl 2:

128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10000000	01000000	00100000	00010000	00001000	00000100	00000010	00000001

Wieder formalisitsch ausgedrückt: wir berechnen den Wert der Zahl, indem wir von rechts beginnen (quasi die nullrechteste Ziffer) mit dem Wert $2^0 = 1$. Danach kommt quasi die ‚erstrechteste' Ziffer mit dem Wert $2^1 = 2$ usw. usw.

Im binären Zahlensystem gibt es nur die Ziffern 0 und 1.

Um z.B. den dezimalen Wert der Binärzahl 11001110 zu berechnen geht man so vor:

Beispiel 1:

11001110 =	$1 * 2^7 +$	$1 * 2^6 +$	$0 * 2^5 +$	$0 * 2^4 +$	$1 * 2^3 +$	$1 * 2^2 +$	$1 * 2^1 +$	$0 * 2^0$
	128 +	64 +	0 +	0 +	8 +	4 +	2 +	0
= 206								

Beispiel 2:

11111111 =	$1 * 2^7 +$	$1 * 2^6 +$	$1 * 2^5 +$	$1 * 2^4 +$	$1 * 2^3 +$	$1 * 2^2 +$	$1 * 2^1 +$	$1 * 2^0$
	128 +	64 +	32 +	16 +	8 +	4 +	2 +	1
= 255								

Beispiel 3:

10101010 =	$1 * 2^7 +$	$0 * 2^6 +$	$1 * 2^5 +$	$0 * 2^4 +$	$1 * 2^3 +$	$0 * 2^2 +$	$1 * 2^1 +$	$0 * 2^0$
	128 +	0 +	32 +	0 +	8 +	0 +	2 +	0
= 170								

Die Berechnung von Binärzahlen sollte einem später in Fleisch und Blut übergehen, denn bei der Ansteuerung von Hardware muß man häufig ‚bitweise' denken – da ist es hilfreich, wenn man überschlägig ein Byte in seine Bits zerlegen kann.

Die Wandlung einer Binärzahl in eine Dezimalzahl wurde oben gezeigt. Doch wie geht es jetzt umgekehrt? Eine Dezimalzahl ist gegeben – eine Binärzahl soll entwickelt werden.

Am einfachsten ist es natürlich, den Windows – Taschenrechner mit wissenschaftlicher Anzeige zu benutzen. Dezimalwert eingeben – auf binär klicken und schon steht das Ergebnis da. Dieser Taschenrechner ist sicherlich gut geeignet, um Ergebnisse nachzuprüfen. Doch hier soll das Ergebnis selbst ermittelt werden.

Die Methode ist umgekehrt zur oben beschriebenen Methode. Geben wir die Zahl 202 dezimal vor. Ihr Wert soll binär dargestellt werden. Diese Zahl ist mit 8 Bit darzustellen.

Erster Schritt: $202 - 2^7 = 74$. Ergebnis ist positiv, also 8. Bit = 1.
Zweiter Schritt: $74 - 2^6 = 10$. Ergebnis ist positiv, also 7. Bit = 1.
Dritter Schritt : $10 - 2^5 = -22$. Ergebnis ist negativ, also 6. Bit = 0.
Vierter Schritt : $10 - 2^4 = -6$. Ergebnis ist negativ, also 5. Bit = 0.
Fünfter Schritt : $10 - 2^3 = 2$. Ergebnis ist positiv, also 4. Bit = 1.
Sechster Schritt: $2 - 2^2 = -2$. Ergebnis ist negativ, also 3. Bit = 0.
Siebter Schritt : $2 - 2^1 = 0$. Ergebnis ist positiv, also 2. Bit = 1
Achter Schritt : $0 - 2^0 = -1$. Ergebnis ist negativ, also 1. Bit = 0.

Gesamtergebnis: 11001010

Die obige Prozedur einmal in Worte gefaßt:

Man prüft in aufeinander folgenden Schritten, welche Zweierpotenzen von der Dezimalzahl abziehbar sind (ohne dass ein negatives Ergebnis entsteht).

Passt eine Zweierpotenz "hinein", wird die entsprechende Binärstelle gleich 1 gesetzt, die Zahl wird um den dezimalen Wert der Zweierpotenz vermindert und dieselbe Prozedur mit der nächst niedrigeren Zweierpotenz wiederholt ...

Kann von der dabei verbleibende Rest-Dezimalzahl die nächst kleinere Zweierpotenz nicht abgezogen werden, wird die entsprechende Binärstelle gleich 0 gesetzt und dieselbe Prozedur mit der nächst niedrigen Zweierpotenz wiederholt ...

Setzen wir dies gleich in Programmcode für die CControl um:

```
Define wert byte
Define i    byte
pause 20        ' entprellen der Taste
Wert=202
print "dezimal ";wert; " = binaer: ";
For i=7 to 0 step -1
If wert and 1 shl i then print "1"; else print "0";
Next i
End
```

Trinärzahlen

Noch nie etwas von Trinärzahlen gehört? Ehrlich gesagt: ich auch nicht. Doch aus didaktischen Gründen sollten wir es einmal probieren. Wenn Binärzahlen auf der Basis 2 arbeiten, so nehmen wir jetzt die Basis 3. Wir werden bald sehen, dass sich die Basis 3 für ein anständiges Zahlensystem nicht eignet.

2187	729	243	81	27	9	3	1
3^7	3^6	3^5	3^4	3^3	3^2	3^1	3^0

Wir erkennen sofort, dass sich nicht alle Dezimalzahlen damit darstellen lassen, da alle Ergebnisse ungerade sind. Also weg mit den Trinärzahlen.

Oktalzahlen

Es gab in der Geschichte der Computertechnik auch Rechner, die mit Oktalzahlen rechneten. Nur kurze Zeit war diese Technik relevant. Jede Oktalziffer kann mit 3 Bits dargestellt werden.

111	110	101	100	011	010	001	000
$2^2+2^1+2^0$	2^2+2^1	2^2+2^0	2^2	2^1+2^0	2^1	2^0	2^0
7	6	5	4	3	2	1	0

Bei Oktalzahlen bezieht man sich auf die Basis 8. Wie beim Dezimalsystem gibt es auch hier die Wertigkeiten: $8^0 = 1$, $8^1 = 8$, $8^2 = 64$ usw.

Mit drei Oktalziffern lassen sich 512 + 64 + 8 = 584 verschiedene Zustände darstellen. Für eine Stelle (0 – 7) werden drei Bits benötigt. Damit sehen die Ziffern folgendermaßen aus:

Dezimal	Wertigkeit 64 (8^2)	Wertigkeit 8 (8^1)	Wertigkeit 1 (8^0)	Oktalzahl
0	000	000	000	000
1	000	000	001	001
2	000	000	010	002
3	000	000	011	003
4	000	000	100	004
5	000	000	101	005
6	000	000	110	006
7	000	000	111	007
8	000	001	000	010
9	000	001	001	011
15	000	001	111	017
16	000	010	000	020
23	000	010	111	027
24	000	011	000	030
25	000	011	001	031
63	000	111	111	077
64	001	000	000	100
65	001	000	001	101
137	010	001	001	211
255	011 (3*64=192)	111 (7*8=56)	111 (7*1=7)	377

Man erkennt, dass man zur Darstellung der Zahlen bis 255 3 * 3 = 9 Bits einsetzen muss, obwohl nur 8 Bits benötigt werden, 1 Bit geht verloren. Dieses ist recht ineffizient.

Hexadezimalzahlen - wie und warum?

Bei Hexadezimal- Zahlen erfolgt der Übertrag nicht nach der 9, sondern nach der (dezimal) 15. Stelle. Um alle Hexadezimal - Ziffern mit einem einzigen Symbol dazustellen, nimmt man für die Darstellung der (Dezimal-)Zahlen 10 bis 15 im Hexadezimal - System die Buchstaben A B C D E F (Groß- und Kleinschreibung erlaubt). Das hexadezimale C ist also gleich der dezimalen 12.

Mit einer einzigen Ziffer können im Hexadezimalsystem also 16 Zustände dargestellt werden.
Da 16 die vierte Potenz von 2 (2^4=16) ist, lassen sich mit einer Hexadezimal-Ziffer genau die Zustände von 4 binären Signalen (Bits) darstellen.

Eine Gruppe von 4 Bits heißt in der Informatik "Nibble"

Hex - Ziffer	Bit 3 (Wert 8)	Bit 2 (Wert 4)	Bit 1 (Wert 2)	Bit 0 (Wert 1)	Dezimalwert
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8

9	1	0	0	1	9
A	1	0	1	0	10
B	1	0	1	1	11
C	1	1	0	0	12
D	1	1	0	1	13
E	1	1	1	0	14
F	1	1	1	1	15

Hexadezimalzahlen lassen sich besonders vorteilhaft anwenden, um Gruppen binärer Zustände kompakt zu beschreiben. Man stelle sich z.B. eine Gruppe von 4 digitalen Signalleitungen vor, von denen jede den Zustand 0 und 1 annehmen kann, also genau die Zustände einer binären Ziffer. Ordnet man jeder Leitung eine bestimmte Zweierpotenz als Gewicht zu, so lassen sich die Zustände aller Leitungen zu einer einzigen Hexadezimal- Ziffer zusammenfassen.

Während die Binärdarstellung ein quasi grafisches Bild der Leitungszustände ist, erlaubt die Hexadezimaldarstellung eine wesentlich kompaktere Beschreibung – auch gegenüber der dezimalen Darstellung ist sie kompakter, wenn es um Ziffern über 9 geht.

Die obige Tabelle sollte man mit einiger Übung - ähnlich wie das kleine Einmaleins - im Kopf haben.

Die Kombination von 2 Nibbles im Sinne einer 2-stelligen Hexadezimal - ZAHL nennt man in der Informatik ein "Byte". Mit einem Byte lassen sich alle 256 möglichen Zustände von 8 binären Signalen darstellen. Die 256 Möglichkeiten, die man mit 8 Bits darstellen kann, können im hexadezimalen Sytem mit zwei Hexziffern (dezimal 255 = 3 Ziffern) dargestellt werden.

Dezimalzahl	Binärzahl	Nibble 1	Nibble 2	Hexzahl
255	11111111	1111	1111	FF
206	11001110	1100	1110	CE
127	01111111	0111	1111	7F
47	00101111	0010	1111	2F
11	00001011	0000	1011	0B = B

Um eine Hexadezimalzahl zu kennzeichnen, sind verschiedene Methoden gebräuchlich. Eine davon, die auch auf dieser Website angewandt wird, besteht darin, ein Dollarzeichen vor die betreffende Hex-Zahl zu stellen. $AFFE und $DEADFACE sind also gültige Hexadezimal- Zahlen.

Andere bekannte Darstellungsweisen stellen ein kleines "h" hinterher, also AFFEh bzw. DEADFACEh, oder - in der Programmiersprache C - wird "0x" vorangestellt, also 0xAFFE bzw. 0xDEADFACE. Bei der CControl benutzt man zur Kennzeichnung ein &H für die hexadezimale Darstellung und entsprechend werden Binärzahlen mit einem &B gekennzeichnet. Das beliebte $AFFE oder &HAFFE hat übrigens den dezimalen Wert 45054.

Die Binärdarstellung ist einfach zu realisieren, wenn eine Hexzahl vorliegt. Nibbleweise braucht man lediglich die Bitkombinationen hintereinanderzuschreiben. So wird aus dem &HAFFE:

A=1010 F=1111 F=1111 E=1110

&HAFFE = &B1010111111111110

Der entsprechende Dezimalwert 45054 hingegen kann selbst von erfahrenen Programmierern nicht leicht im Kopf in Bits umgerechnet werden. Die reine Binärzahl wiederum kann man sich schwer merken. Die Umrechnung von Bits in Zahlen und umgekehrt ist in der Praxis wichtig bei der Programmierung digitaler Ein- und Ausgänge und bei logischen Verknüpfungen. Daher sollten Sie die Hexadezimal- Darstellung nicht als Schikane oder Informatikspinnerei ansehen, sondern als eine aus der Sache begründete optimale Darstellung binärer Zustände anwenden.

Wie kann man Hexadezimalzahlen in Dezimalzahlen umrechnen?

Auch hier ist die einfachste Antwort: man nimmt den Windows – Taschenrechner und lässt ihn die Arbeit ausführen.
Die Umrechnung per Hand ist natürlich ebenso möglich; sie ist nur etwas umständlich. Man sollte jedoch aus Verständnisgründen dieses einmal gemacht haben. Bleiben wir bei dem Beispiel &HAFFE = 45054. Die Umrechnung in die Binärzahl war recht einfach. Jetzt müssen wir jeder Stelle wieder eine Wertigkeit zuordnen.

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0
32768		8192		2048	1024	512	256	128	64	32	16	8	4	2	

Die letzte Reihe addiert ergibt dann 45054.

Natürlich kann man das Ergebnis auf andere Weise ermitteln. Wir haben das hexadezimale Zahlensystem vorliegen. Die Basis beträgt 16. Somit ist jede Stelle mit einer Wertigkeit von 16^{X} belegt. Das Beispiel &HAFFE ist eine vierstellige Hexzahl. Die Wertigkeiten verteilen sich folgendermaßen:

Stellen	3. Stelle	2. Stelle	1. Stelle	0. Stelle
Wertigkeit	16^3	16^2	16^1	16^0
Dezimalwert	4096	256	16	1
Hexwert	**A**	**F**	**F**	**E**
Wert * Wertigkeit	10 * 4096	15 * 256	15 * 16	14 * 1
Ergebnis	40960	3840	240	14

Die letzte Reihe addiert ergibt dann 45054.

Man sollte durchaus ein wenig mit den verschiedenen Zahlensystemen herumexperimentieren, um einen schnellen Überblick über die Zahlen zu bekommen. Was bedeutet nun &HDEADFACE. Auch diese 8 - stellige Hexzahl soll hier zur Übung untersucht werden:

7. Stelle	6. Stelle	5. Stelle	4. Stelle	3. Stelle	2. Stelle	1. Stelle	0. Stelle
16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0
268435456	16777216	1048576	65536	4096	256	16	1
D	**E**	**A**	**D**	**F**	**A**	**C**	**E**
13	14	10	13	15	10	12	14
3489660928	234881024	10485760	851968	61440	2560	192	14
1101	1110	1010	1101	1111	1010	1100	1110

&HDEADFACE ergibt dezimal den Wert 3.735.943.886, binär den Wert &B11011110101011011111101011001110.

Dezimalzahlen in Hexzahlen umrechnen.

Es fehlt jetzt noch die Umrechnung einer Dezimalzahl in eine Hexzahl. Die Dezimalzahl wird „ausdividiert“, und zwar mit den Gewichten der Potenzen von 16. Bleiben wir bei unserem Beispiel &HAFFE = 45054.

7. Stelle	6. Stelle	5. Stelle	4. Stelle	3. Stelle	2. Stelle	1. Stelle	0. Stelle
16^7	16^6	16^5	16^4	16^3	16^2	16^1	1
268435456	16777216	1048576	65536	4096	256	16	
45054/268435456	45054/16777216	45054/1048576	45054/65536	45054/4096	4094/256	254/16	14
0	**0**	**0**	**0**	**10**	**15**	**15**	**1**

				Rest: 4094	Rest:254	Rest:14	Re
				A	**F**	**F**	

```
Sub Command1_Click ()
Dim wert As Double
wert = Val(text1)
For i = 7 To 0 Step -1
If (wert / (16 ^ i)) > 0 Then
anzahl = wert \ (16 ^ i)
wert = wert - anzahl * 16 ^ i
ausgabe = Trim(Str$(anzahl))
If anzahl > 9 Then ausgabe = Chr(55 + anzahl)
aus = aus + ausgabe
End If

Next i
If Len(aus) < 8 Then aus = aus + "0"
Print aus
End Sub
```

196	C4	11000100	304	197	C5	11000101	305	198	C6	11000110	306	199	C7	11000111	307
200	C8	11001000	310	201	C9	11001001	311	202	CA	11001010	312	203	CB	11001011	313
204	CC	11001100	314	205	CD	11001101	315	206	CE	11001110	316	207	CF	11001111	317
208	D0	11010000	320	209	D1	11010001	321	210	D2	11010010	322	211	D3	11010011	323
212	D4	11010100	324	213	D5	11010101	325	214	D6	11010110	326	215	D7	11010111	327
216	D8	11011000	330	217	D9	11011001	331	218	DA	11011010	332	219	DB	11011011	333
220	DC	11011100	334	221	DD	11011101	335	222	DE	11011110	336	223	DF	11011111	337
224	E0	11100000	340	225	E1	11100001	341	226	E2	11100010	342	227	E3	11100011	343
228	E4	11100100	344	229	E5	11100101	345	230	E6	11100110	346	231	E7	11100111	347
232	E8	11101000	350	233	E9	11101001	351	234	EA	11101010	352	235	EB	11101011	353
236	EC	11101100	354	237	ED	11101101	355	238	EE	11101110	356	239	EF	11101111	357
240	F0	11110000	360	241	F1	11110001	361	242	F2	11110010	362	243	F3	11110011	363
244	F4	11110100	364	245	F5	11110101	365	246	F6	11110110	366	247	F7	11110111	367
248	F8	11111000	370	249	F9	11111001	371	250	FA	11111010	372	251	FB	11111011	373
252	FC	11111100	374	253	FD	11111101	375	254	FE	11111110	376	255	FF	11111111	377

Wer diese Tabellen selbst erzeugen möchte, der kann das untenstehende Visual Basic 3 – Programm eingeben. Eine GRID.VBX muss geladen werden.

```
Sub Form_Load ()
Show
windowstate = 2
grid1.Width = 10000
grid1.Height = 8060
grid1.FixedRows = 1
grid1.FixedCols = 0
grid1.Rows = 67
grid1.Cols = 16

For i = 0 To grid1.Cols - 1
grid1.FixedAlignment(i) = 2
grid1.ColAlignment(i) = 2
Next i
grid1.Row = 0
For i = 0 To 3
grid1.Col = i * 4
grid1.Text = "dez."
grid1.ColWidth(i * 4) = 450
grid1.Col = i * 4 + 1
grid1.Text = "hex."
grid1.ColWidth(i * 4 + 1) = 450
grid1.Col = i * 4 + 2
grid1.Text = "bin."
grid1.ColWidth(i * 4 + 2) = 900
grid1.Col = i * 4 + 3
grid1.Text = "oct."
grid1.ColWidth(i * 4 + 3) = 450
Next i
grid1.Row = 1
zaehler = 0
For i = 0 To 63
For j = 0 To 12 Step 4
grid1.Col = j
grid1.Text = Format$(zaehler, "00#")
grid1.Col = j + 1
grid1.Text = Format$(Hex$(zaehler), "00#")
bin$ = ""
For b = 7 To 0 Step -1
If zaehler And 2 ^ b Then bin$ = bin$ + "1" Else bin$ = bin$ + "0"
Next b
```

```
grid1.Col = j + 2
grid1.Text = bin$
grid1.Col = j + 3
grid1.Text = Format$(Oct$(zaehler), "00#")
zaehler = zaehler + 1
Next j
grid1.Row = grid1.Row + 1
Next i
End Sub
```