2 Codierung und Zahlensysteme

2.1 Codes

Codes werden in der Digitaltechnik häufig verwendet, um ein Signal für einen Anwendungsfall optimal darzustellen. Ein Code bildet die Zeichen eines Zeichenvorrates auf die Zeichen eines zweiten Zeichenvorrates ab. Sinnvollerweise soll auch eine Dekodierung möglich sein, bei der aus dem codierten Zeichen wieder das ursprüngliche gewonnen wird.

Ein bekanntes Beispiel für einen Code ist der Morse-Code. Die Definition eines Codes geschieht durch eine Zuordnungstabelle wie sie in Tabelle 2-1 für den Morse-Code festgehalten ist. Dieser Code ist umkehrbar, da aus dem Buchstaben ein Morsezeichen, und daraus wieder der Buchstabe eindeutig ermittelt werden kann. Das gilt aber nur für einen Text, der in kleinen Buchstaben geschrieben ist, da der Morse-Code nicht zwischen Groß- und Kleinschreibung unterscheidet. Ein Text, der in Groß- und Kleinschreibung verfasst ist, kann daher strenggenommen aus dem Morse-Code nicht wieder dekodiert werden.

Tabelle 2-1 Morse-Code.

Alphabet	Morse-Code	Alphabet	Morse-Code	Alphabet	Morse-Code
a	· –	j	· – – –	s	· · ·
b	· – · ·	k	– · –	t	–
c	– · · ·	l	· – · ·	u	· · –
d	– · ·	m	– –	v	· · · –
e	·	n	– ·	w	· – –
f	· · – ·	o	– – –	x	– · · –
g	– – ·	p	· – – ·	y	– · – –
h	· · · ·	q	– – · –	z	– – · ·
i	· ·	r	· – ·		

Für jede Anwendung gibt es mehr oder weniger gut geeignete Codes. So ist für die Zahlenarithmetik in einem Rechner ein anderer Code sinnvoll als für die Übertragung von Zahlen über eine Nachrichtenverbindung. Dieses Kapitel untersucht die Unterschiede der einzelnen Codes und weist auf deren spezifische Anwendungen hin.

Die Kombination mehrerer Zeichen eines Codes nennt man ein Wort. Wir werden uns im Folgenden auf den technisch wichtigen Fall beschränken, dass alle Wörter eines Codes die gleiche Länge n haben. Im Morse-Code ist das nicht der Fall. Hat ein Code einen Zeichenvorrat von N Zeichen, so kann man N^n verschiedene Wörter der Länge n bilden. Werden alle N^n möglichen Wörter eines Codes verwendet, so spricht man von einem Minimalcode. Werden weniger als

K. Fricke, *Digitaltechnik,* DOI 10.1007/978-3-8348-2213-0_2,
© Springer Fachmedien Wiesbaden 2014

N^n Wörter verwendet, so nennt man ihn redundanten Code. Im Folgenden findet man eine Zusammenfassung der geläufigsten Codes, ausführliche Darstellungen findet man in [7-9].

2.2 Dualcode

Der Dualcode ist der wichtigste Code in digitalen Systemen, da er sehr universell ist. Durch die Beschränkung auf die Zeichen 1 und 0 ist eine Verarbeitung mit Bauelementen möglich, die als Schalter arbeiten. Der Dualcode erlaubt auch eine Arithmetik analog der des Dezimalsystems. Das duale Zahlensystem kann als eine Codierung des Dezimalsystems verstanden werden. Eine Dualzahl besteht aus einem Wort, welches aus den Zeichen $c_i \in \{0,1\}$ gebildet wird. Die Zeichen c_i eines Worts werden in der Digitaltechnik Bits genannt. Das Wort z_2 in der Dual-Darstellung entsteht durch die Aneinanderreihung der einzelnen Bits wie im Folgenden dargestellt:

$$z_2 = c_{n-1}c_{n-2}\cdots c_1c_0\ ,\ c_{-1}c_{-2}\cdots c_{-m+2}c_{-m+1}c_{-m} \tag{2.1}$$

Diese Dualzahl hat n Stellen vor dem Komma und m Stellen nach dem Komma. Den einzelnen Bits werden entsprechend ihrer Stellung i im Wort Gewichte 2^i zugeordnet. Damit kann die äquivalente Dezimalzahl z_{10} berechnet werden:

$$z_2 = \mathrm{g}(z_{10}) = c_{n-1}2^{n-1} + c_{n-2}2^{n-2} + \ldots + c_1 2^1 + c_0 2^0 + c_{-1}2^{-1} + \ldots + c_{-m}2^{-m} \tag{2.2}$$

Wir betrachten zum Beispiel die Dualzahl $10110{,}011_2$, die durch den Index 2 als Dualzahl gekennzeichnet ist. Sie wird interpretiert als:

$$\begin{aligned}\mathrm{g}(z_2) &= 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} = z_{10}\\ &= 22{,}375_{10}\end{aligned}$$

Der Dualcode wird als gewichteter Code bezeichnet, da die weiter links im Wort stehenden Bits ein höheres Gewicht besitzen. Gleichung 2.2 liefert eine Vorschrift für die Umwandlung von Dualzahlen in Dezimalzahlen.

Die Umwandlung von Dezimalzahlen in Dualzahlen ist komplizierter. Sie kann durch einen Algorithmus beschrieben werden, der für den ganzzahligen und den gebrochenen Teil der Dezimalzahl unterschiedlich ist. Am obigen Beispiel der Zahl $22{,}375_{10}$ soll der Algorithmus dargestellt werden:

- Zuerst wird der ganzzahlige Anteil in eine Dualzahl umgesetzt. Dazu wird der ganzzahlige Anteil fortwährend durch 2 geteilt und der Rest notiert, bis sich 0 ergibt.

  22:2 = 11 Rest 0
  11:2 = 5 Rest 1
  5:2=2 Rest 1
  2:2=1 Rest 0
  1:2=0 Rest 1

  ganzzahliger Anteil der Dualzahl

  Die zu 22_{10} gehörende Dualzahl ist also 10110_2.

- Im zweiten Schritt wird der gebrochene Anteil in den gebrochenen Anteil der Dualzahl umgesetzt. Zuerst wird der gebrochene Anteil mit 2 multipliziert. Der ganzzahlige Anteil wird abgetrennt, er bildet die niedrigstwertige Stelle der Dualzahl.

  Das Verfahren wird wiederholt, wie im folgenden Beispiel gezeigt.

$0{,}375 \cdot 2 = 0{,}75 \quad + 0$

$0{,}75 \cdot 2 = 0{,}5 \quad + 1$

$0{,}5 \cdot 2 = 0 \quad + 1$

gebrochener Anteil der Dualzahl

Man erkennt an dieser Stelle, dass sich der Rest 0 ergibt. Das ist nicht notwendigerweise so. Im Normalfall wird der gebrochene Anteil der äquivalenten Dualzahl unendlich viele Stellen haben. Man muss sich dann mit einer bestimmten Anzahl von Stellen hinter dem Komma begnügen und damit die Genauigkeit einschränken. In unserem Fall entspricht $0{,}375_{10}$ genau $0{,}011_2$.

Aus dem ganzzahligen und dem gebrochenen Anteil ergibt sich die gesuchte Dualzahl zu $10110{,}011_2$.

2.3 Festkomma-Arithmetik im Dualsystem

In diesem Kapitel wird die Arithmetik mit Festkommazahlen beschrieben. Festkomma-Arithmetik bedeutet, dass das Komma immer an einer festen Stelle steht. Die Stelle, an der das Komma steht, orientiert sich dabei an der Stellung im Speicher, in dem die Zahl steht. Das Komma braucht dabei nicht in der Hardware des Rechners implementiert zu sein. Es existiert in diesem Fall nur im Kopf des Programmierers. Wir beschränken uns auf eine konstante Wortlänge n, wie es in Rechnern der Fall ist. Dadurch kann auch das Problem der Bereichsüberschreitung diskutiert werden.

2.3.1 Ganzzahlige Addition im Dualsystem

Die ganzzahlige Addition zweier Zahlen A und B wird im Dualsystem genau wie im Dezimalsystem stellenweise durchgeführt. Wie dort müssen bei jeder Stelle die beiden Dualziffern a_n und b_n und der Übertrag von der vorhergehenden Stelle c_{n-1} addiert werden. Bei der Addition (Tabelle 2-2) entsteht eine Summe s_n und ein neuer Übertrag c_n.

Tabelle 2-2 Addition im Dualsystem mit den Summanden a_n, b_n und dem Übertrag von der vorhergehenden Stelle c_{n-1}. Die Summe ist s_n und der neue Übertrag c_n.

a_n	b_n	c_{n-1}	c_n	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

In der folgenden Tabelle ist der Übertrag explizit aufgeführt.

Bsp.:

	0	1	1	1	1	1	1	0
+	0	0	1	1	0	1	0	1
Übertrag	1	1	1	1	1	0	0	
=	1	0	1	1	0	0	1	1

Man beachte, dass im obigen Beispiel zwei 8 Bit lange Zahlen addiert wurden und das Ergebnis auch 8 Bit lang ist, so dass keine Bereichsüberschreitung stattfindet.

2.3.2 Addition von Festkommazahlen

Sollen zwei Festkommazahlen addiert werden, so ist es analog zum gewohnten Vorgehen im Dezimalsystem wichtig, dass die Kommata übereinander stehen. Daher muss bei der Addition von zwei 8Bit langen Zahlen das Komma bei beiden Zahlen zum Beispiel an der dritten Stelle stehen.

Bsp.:

	0	1	1	0	0,	0	1	0
+	0	0	1	1	0,	1	1	1
Übertrag	1	1	0	0	1	1	0	
=	1	0	0	1	1,	0	0	1

2.3.3 Einerkomplementdarstellung

Um den Hardware-Aufwand in Rechnern klein zu halten, hat man sich bemüht, Subtraktion und Addition auf *einen* Algorithmus zurückzuführen. Das gelingt, wenn negative Dualzahlen in ihrer Komplementdarstellung verwendet werden. Man unterscheidet zwischen Einer- und Zweier-Komplement. Das Einerkomplement wird gebildet, indem in einer Dualzahl alle Nullen gegen Einsen vertauscht werden und umgekehrt. Das Einerkomplement von 0001 ist also 1110. Das Einerkomplement einer Dualzahl A wird hier dargestellt als $\neg A$.

Es gilt offenbar bei einer Darstellung in n Bit-Worten:

$$\neg A + A = 2^n - 1 \tag{2.3}$$

Bsp. für eine Darstellung in 8 Bit-Worten:

$$10110011 + 01001100 = 11111111 = 2^8 - 1$$

Man kann Gleichung 2.3 so umformen, dass sie eine Rechenvorschrift für das Einerkomplement ergibt:

$$\neg A = 2^n - 1 - A \tag{2.4}$$

2.3.4 Zweierkomplementdarstellung

Das Zweierkomplement A_{K2} entsteht aus dem Einerkomplement $\neg A$ durch die Addition von 1:

$$A_{K2} = \neg A + 1 \tag{2.5}$$

Also gilt mit Gleichung 2.4:

$$A_{K2} = 2^n - A \tag{2.6}$$

Man erkennt, dass hier eine Darstellung vorliegt, in der $-A$ vorkommt, wodurch sich diese Darstellung für die Subtraktion eignet. Man beachte auch, dass 2^n in der dualen Darstellung n+1 Stellen hat. Hier ein Beispiel für das Zweierkomplement aus 10101100:

$$A_{K2} = \neg A + 1 = 01010011 + 1 = 01010100$$

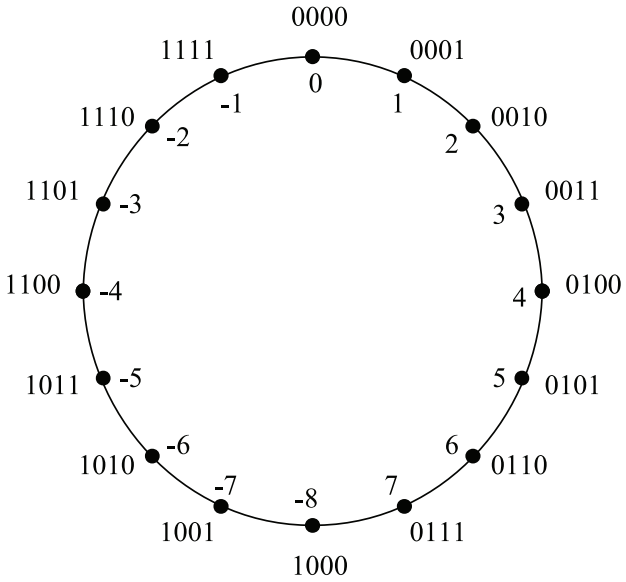


Bild 2-1 Darstellung von 4Bit-Wörtern in Zweierkomplementdarstellung.

Eine kreisförmige Darstellung in Bild 2-1 der 4Bit breiten Dualzahlen verdeutlicht den Zahlenbereich. Die betragsmäßig größte darstellbare positive Zahl ist 7_{10}, die betragsmäßig größte negative Zahl ist -8_{10}. Der Zahlenbereich ist also unsymmetrisch angeordnet, da es eine negative Zahl mehr gibt als positive. Die größte und kleinste darstellbare Zahl ist:

$$Z_{max} = 2^{n-1} - 1 \tag{2.7}$$

$$Z_{min} = -2^{n-1} \tag{2.8}$$

Dem Bild 2-1 entnimmt man, dass man betragsmäßig kleine Zweierkomplement-Zahlen daran erkennt, dass sie viele führende Einsen aufweisen, wenn sie negativ sind, und dass sie viele führende Nullen haben, wenn sie positiv sind. Betragsmäßig große Zweierkomplement-Zahlen haben eine weit links stehende Null, wenn sie negativ sind und eine weit links stehende Eins wenn sie positiv sind. Die Zahl 1000_2 (-8_{10}) ist ihr eigenes Zweierkomplement! Es ist auch wichtig festzustellen, dass es nur eine 0 in der Zweierkomplement-Darstellung gibt. Das erleichtert die Abfrage, ob ein Ergebnis 0 ist. In der Einerkomplementdarstellung gibt es dagegen die Dualzahl 0000_2 die der $+0_{10}$ entspricht und die Dualzahl 1111_2 die der -0_{10} entspricht.

2.3.5 Subtraktion in Zweierkomplementdarstellung

Es sollen nun zwei positive Dualzahlen A und B voneinander subtrahiert werden. Man kann die Subtraktion unter Verwendung des Zweierkomplements laut Gleichung 2.6 so durchführen:

$$A - B = A - B + B_{K2} - B_{K2} = A - B + B_{K2} - (2^n - B) \tag{2.9}$$

Zusammenfassen des rechten Ausdrucks ergibt:

$$A - B = A + B_{K2} - 2^n \tag{2.10}$$

Was bedeutet die Subtraktion von 2^n? Das soll am Beispiel der Subtraktion 7–3 = 4 im 4Bit-Dualsystem dargestellt werden. Die Summe des Dualäquivalents von 7 und des Komplements des Dualäquivalents von 3 ergibt:

Bsp.:

		0	1	1	1	7_{10}
+		1	1	0	1	-3_{10}
=	1	0	1	0	0	
–	1	0	0	0	0	
=		0	1	0	0	4_{10}

Die Subtraktion von 10000_2 entsprechend der Gleichung 2.10 ergibt das richtige Ergebnis 0100_2. Das kann in einem 4-Bit-Rechner einfach dadurch geschehen, dass die höchste Stelle ignoriert wird. Bei der Subtraktion mit Hilfe des Zweierkomplements ist also der höchste Übertrag c_4 nicht zu berücksichtigen. Vorsicht ist aber geboten in Zusammenhang mit einer Bereichsüberschreitung.

2.3.6 Bereichsüberschreitung

Es soll daher das Problem der Bereichsüberschreitung (Overflow) in Zusammenhang mit der Zweierkomplement-Darstellung betrachtet werden. Bereichsüberschreitungen können nur in 2 Fällen auftreten. Nämlich dann, wenn zwei positive Zahlen addiert werden oder wenn zwei negative Zahlen addiert werden. In allen anderen Fällen ist eine Bereichsüberschreitung ausgeschlossen. Dazu betrachten wir einige Beispiele in einer 4-Bit Darstellung:

- Beispiel einer Bereichsüberschreitung bei der Addition zweier positiver Zahlen.

  Bsp.:

		0	1	0	1	5_{10}
+		0	1	0	1	5_{10}
=	(0)	1	0	1	0	-6_{10}

  Das Ergebnis ist offensichtlich falsch. Der Fehler entsteht durch den Übertrag von der 3. in die 4. Stelle, wodurch eine negative Zahl vorgetäuscht wird. Dieser Übertrag c_3 wird in einer Darstellung mit n Bits allgemein als c_{n-1} bezeichnet. Der Übertrag c_4 (allgemein c_n) von der 4. Stelle in die 5. Stelle heißt Carry (Cy). Er ist in diesem Beispiel nicht aufgetreten.

- Beispiel einer Bereichsüberschreitung bei der Addition negativer Zahlen:

Bsp.:

		1	0	1	1	-5_{10}
+		1	0	1	1	-5_{10}
=	(1)	0	1	1	0	6_{10}

Auch in diesem Beispiel entsteht ein falsches Ergebnis. Es gab keinen Übertrag c_{n-1} von der 3. in die 4. Stelle aber einen Übertrag c_n von der 4. in die 5. Stelle.

- Zum Vergleich eine Addition zweier negativer Zahlen ohne Bereichsüberschreitung:

Bsp.:

		1	1	1	1	-1_{10}
+		1	1	0	1	-3_{10}
=	(1)	1	1	0	0	-4_{10}

Es gab die Überträge c_n und c_{n-1}.

Nun sollen diese Ergebnisse zusammen mit weiteren, hier nicht gezeigten Fällen in einer Tabelle zusammengefasst werden. Für zwei positive Dualzahlen A und B kann ein Überlauf bei der Rechnung im Zweierkomplement festgestellt werden, wenn die Überträge c_n und c_{n-1}, wie in der Tabelle 2-3 gezeigt, ausgewertet werden.

Tabelle 2-3 Überlauf bei der Addition in einer n Bit-Zweierkomplement-Darstellung ($A, B \geq 0$).

	Richtiges Ergebnis	Überlauf
$A+B$	$c_n = 0 \quad , \quad c_{n-1} = 0$	$c_n = 0 \quad , \quad c_{n-1} = 1$
$A-B$	$c_n = c_{n-1}$	nicht möglich
$-A-B$	$c_n = 1 \quad , \quad c_{n-1} = 1$	$c_n = 1 \quad , \quad c_{n-1} = 0$

Zusammenfassend kann festgestellt werden, dass ein richtiges Ergebnis vorliegt, wenn $c_n = c_{n-1}$ gilt, ein falsches, wenn $c_n \neq c_{n-1}$ ist.

2.3.7 Multiplikation

Die Multiplikation wird wie im Dezimalsystem ausgeführt. Hier ein Beispiel für die Multiplikation im Dualsystem $10_{10} \times 11_{10} = 110_{10}$:

Bsp.:

1	0	1	0	×	1	0	1	1	
					1	0	1	0	
				1	0	1	0		
		1	0	1	0				
		1	1	0	1	1	1	0	

Das größte zu erwartende Ergebnis E der Multiplikation zweier n-Bit-Wörter ist:

$$E = (2^n - 1)(2^n - 1) = 2^{2n} - 2^{n+1} + 1 \quad \leq \quad 2^{2n} - 1$$

Das Ergebnis einer Multiplikation zweier n-Bit-Zahlen ist also $2n$-Bit lang. Es ist aber kleiner als die mit $2n$ Bits maximal darstellbare Dualzahl $2^{2n}-1$.

Das Gesagte gilt für eine Multiplikation positiver Zahlen. Für das Rechnen in der Zweierkomplement-Darstellung können spezielle Algorithmen verwendet werden [41], oder man muss die Zweierkomplement-Zahlen vor der Multiplikation in ihre Beträge zurückverwandeln und das Ergebnis entsprechend dem Vorzeichen wieder in die gewünschte Darstellung überführen.

Bei der Multiplikation von Fixkommazahlen werden zunächst die Zahlen ohne Berücksichtigung des Kommas multipliziert. Dann wird das Komma entsprechend der Regel gesetzt: Die Multiplikation zweier Zahlen mit n und k Stellen hinter dem Komma ergibt ein Produkt mit $n+k$ Stellen hinter dem Komma.

2.3.8 Division

Die Division kann mit dem gleichen Algorithmus durchgeführt werden, wie er im Dezimalsystem verwendet wird. Das soll am Beispiel $10_{10}:2_{10}=5_{10}$ demonstriert werden.

Bsp.:

```
1 0 1 0 : 0 0 1 0 = 1 0 1
1 0
-----
  0 1 0
    1 0
  -----
      0
```

Entsprechend hat bei der Division einer Zahl mit n Stellen hinter dem Komma durch eine Zahl mit k Stellen hinter dem Komma, der Quotient n-k Stellen hinter dem Komma. So ergibt sich entsprechend dem obigen Beispiel:

$$10{,}10 \cdot 101{,}1 = 1101{,}110$$

Auch die Division von Zweierkomplement-Zahlen kann auf Multiplikation und Addition zurückgeführt werden [41].

2.4 Hexadezimalcode

In der Praxis hat sich neben dem Dualcode auch der Hexadezimalcode durchgesetzt, da er gegenüber langen Dualzahlen übersichtlicher ist. Die 16 Hexadezimalziffern sind definiert durch die Tabelle 2-4. Die Hexadezimalziffern größer als 9 werden durch die Buchstaben A-F dargestellt. Für die Umwandlung einer Dualzahl in eine Hexadezimalzahl fasst man jeweils 4 Ziffern der Dualzahl zusammen, die als eine Hexadezimalstelle interpretiert werden. Dadurch hat eine Hexadezimalzahl nur ein Viertel der Stellen wie eine gleichgroße Dualzahl.

Bsp.:

0 1 1 0	1 1 0 0	1 1 1 1
6	C	F

Es gilt also $011011001111_2 = 6CF_{16}$. Zur Kennzeichnung einer Hexadezimalzahl sind auch die Zeichen H und $ üblich. Die Umwandlung einer Hexadezimalzahl in eine Dezimalzahl und umgekehrt geschieht am einfachsten über die entsprechende Dualzahl. Es ist aber auch möglich, die Umwandlung über einen Algorithmus wie bei der Umwandlung einer Dualzahl in eine Dezimalzahl durchzuführen. Die umgekehrte Umwandlung würde analog zu der Gleichung 2.2 durchzuführen sein.

Tabelle 2-4 Hexadezimalzahlen.

Dezimal	Dual	Hexadezimal	Dezimal	Dual	Hexadezimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

2.5 Oktalcode

Der Oktalcode wird ähnlich verwendet wie der Hexadezimalcode, nur dass jeweils 3 Stellen einer Dualzahl zusammengefasst werden. Für den Oktalcode werden die Ziffern 0 bis 7 des Dezimalcodes verwendet, er wird oft auch mit dem Index O gekennzeichnet.

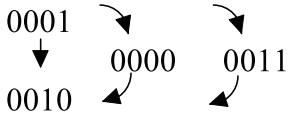
Bsp.:

1 1 0	1 0 1	1 0 0	0 1 1
6	5	4	3

Es gilt also $11010100011_2 = 6543_8$.

2.6 Graycode

Oft benötigt man in der Digitaltechnik eine Codierung für einen Zahlencode, bei dem beim Übergang von einer Zahl zur nächsten sich nur eine Ziffer ändern soll. Diese Bedingung ist notwendig, wenn durch technische Ungenauigkeiten der Zeitpunkt der Umschaltung nicht genau eingehalten werden kann. Bei einer Umschaltung von 2 Ziffern könnten sich daher Fehlschaltungen ergeben. Als Beispiel für einen derartigen Fehler soll die Umschaltung von 1_{10} auf 2_{10} im Dual-Code betrachtet werden:



Bei dieser Umschaltung ändern sich die Bits 0 und 1. Bei gleichzeitigem Umschalten wird die neue Zahl direkt erreicht. Wechselt erst das Bit 0, so erscheint zunächst die Zahl 0000 und erst wenn sich auch Bit 1 ändert, erhält man die richtige Zahl 0010. Ändert sich zuerst das Bit 1 und dann Bit 0, so wird zwischendurch die Zahl 0011 sichtbar. Gray-Codes vermeiden diesen gravierenden Fehler dadurch, dass sich von einem Codewort zum nächsten nur eine Stelle ändert. Die Tabelle 2-5 zeigt einen 4-stelligen Gray-Code. Der gezeigte Code hat zusätzlich die Eigenschaft, dass er zyklisch ist, da sich auch beim Übergang von der höchsten Zahl (15_{10}) zu der niedrigsten (0_{10}) nur eine Stelle ändert. Zyklische Gray-Codes können für alle geraden Periodenlängen konstruiert werden.

Tabelle 2-5 Beispiel für einen 4-stelligen Gray-Code.

Dezimal	Gray-Code	Dezimal	Gray-Code
0	0000	8	1100
1	0001	9	1101
2	0011	10	1111
3	0010	11	1110
4	0110	12	1010
5	0111	13	1011
6	0101	14	1001
7	0100	15	1000

2.7 BCD-Code

Will man zum Beispiel die Dezimal-Ziffern einer Anzeige ansteuern, so eignet sich ein Code, bei dem den einzelnen Dezimal-Ziffern dual codierte Code-Wörter zugeordnet sind. Dieser Code wird als BCD-Code (*B*inär-*c*odierte *D*ezimalzahl) bezeichnet. Eine Möglichkeit besteht darin, die Dezimal-Ziffern durch jeweils eine 4-stellige Dualzahl darzustellen. Da die einzelnen Stellen die Wertigkeiten 8, 4, 2 und 1 haben, wird der Code 8-4-2-1-Code genannt. Es gibt auch die Möglichkeit, einen BCD-Code mit den Gewichten 2, 4, 2, 1 aufzubauen (Aiken-Code). Andere BCD-Codes sind der 3-Exzess-Code und der BCD-Gray-Code [3].

Tabelle 2-6 BCD-Code.

Dezimalziffer	8-4-2-1-Code	Dezimalziffer	8-4-2-1-Code
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001