Der vergessene Code

Gunther Zielosko

1. Einführung

Wohl jeder, der mit dem BASIC-Tiger® arbeitet, kennt die verschiedenen Darstellungen von Zahlen oder Buchstaben. Da gibt es den Binär-Code, den Hexadezimal-Code, den ASCII-Code, den BCD-Code und viele weitere, an die wir uns mehr oder minder gewöhnt haben. In der "Gründerzeit" der Datenverarbeitung waren noch viele weitere Codes populär, z.B. der Octal-Code, der Hamming-Code, der Gray-Code usw. Wir wollen uns in diesem Applikationsbericht mit einem dieser "vergessenen" Codes etwas näher beschäftigen, dem Gray-Code. Wir werden feststellen, daß er eigentlich unverdient ein Schattendasein führt, für den Elektronikbastler bietet er durchaus reizvolle Eigenschaften. Neben der grauen Theorie gibt es auch hier wieder Anregungen zum Basteln mit dem BASIC-Tiger®.

2. Die Welt der codierten Zahlen

Wozu benötigt man eigentlich Codierungen für Zahlen. Der Grund für die Konjunktur aller dieser Zahlen-Codes ist die Eigenschaft der Computer, daß diese im Gegensatz zu uns nur zwei Zustände unterscheiden können:

eingeschaltet	Ausgeschaltet	
Strom	kein Strom	
logisch High	logisch Low	
High-Pegel	Low-Pegel	
logisch 1	logisch 0	
L	0	usw.

Alle Daten, die darüber hinaus gehen, wie z.B. die Ziffern 0 bis 9, alle Zahlen, die Buchstaben des Alphabets, aber auch Bilder, digitalisierte Klänge und vieles andere mehr können nur über eine Kombination dieser beiden Grundzustände dargestellt werden. Die Folge ist z.B. der Binär-Code, der dies logisch einleuchtend als Summe der Potenzen von 2 darstellt (im Tiger-Basic werden Binärzahlen mit einem nachgestellten b gekennzeichnet):

Zahl 0	$0*2^3 + 0*2^2 + 0*2^1 + 0*2^0$	binär 0000b
Zahl 1	$0*2^3 + 0*2^2 + 0*2^1 + 1*2^0$	binär 0001b
Zahl 2	$0*2^3 + 0*2^2 + 1*2^1 + 0*2^0$	binär 0010b
Zahl 3	$0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$	binär 0011b
Zahl 4	$0*2^3 + 1*2^2 + 0*2^1 + 0*2^0$	binär 0100b
Zahl 5	$0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$	binär 0101b
Zahl 6	$0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$	binär 0110b
Zahl 7	$0*2^3 + 1*2^2 + 1*2^1 + 1*2^0$	binär 0111b
Zahl 8	$1*2^3 + 0*2^2 + 0*2^1 + 0*2^0$	binär 1000b
Zahl 9	$1*2^3 + 0*2^2 + 0*2^1 + 1*2^0$	binär 1001b
usw.		

Je nach Größe benötigen die Zahlen mehr oder weniger Potenzen von 2 (Bits). Die Zahlen 0 bis 15 kommen mit 4 Bit (Nibbel), die Zahlen bis 256 mit 8 Bit (Byte) und die Zahlen bis 65535 (Word) aus. Noch größere Zahlen oder Dezimalzahlen mit Vorzeichen bzw. Kommastellen brauchen noch mehr Bits. So weit, so gut, für die meisten Tiger-Nutzer ist das langweilig.
Neu für einige dürfte sein, daß es durchaus sinnvolle Überlegungen gibt, neben dieser einleuchtenden Bildungsvorschrift für Binärzahlen noch andere zu "erfinden" oder zu nutzen. Der Grund für die Entwicklung des Gray-Codes liegt in einer für bestimmte Anwendungen ungünstigen Eigenschaft des Binär-Codes.
Stellen wir uns einmal einen binärcodierten Drehwahlschalter vor, wie es sie vielfach zum Voreinstellen von Grenzwerten an Meßgeräten gibt (Bilder 1 bis 3). Der hat vorne ein Ziffernrad z.B. mit den Ziffern 0 bis 9, innen drin 5 Schleifer (ein gemeinsamer Abgriff, und 4 für die einzelnen Bits) sowie eine Segmentscheibe, mit der auf geschickte Weise der jeweilige Binär-Code erzeugt wird. Solange das Ziffernrad genau auf den Raststellungen fixiert ist, gibt es keine Probleme. Was aber passiert zwischen zwei Stellungen? Bewegt man das Rad z.B. langsam von der Position 7 auf 8, sollten zunächst die niederwertigen 3 Bits gesetzt sein (7 = 0111b) und dann alle Bits schlagartig umschalten auf (8 = 1000b). Tun sie aber nicht! Konstruktive Toleranzen, aber auch das kleinste Zittern des Bedieners, können dazu führen, daß die Bits in der Übergangsphase "verrückt spielen". Beispielsweise schaltet das höchstwertige Bit schon auf 1 um und die 3 niederwertigen liegen noch auf 1. Das wäre (15 = 1111b), ein völlig falscher Wert. Will man den Schalter mit einer Logik (oder dem BASIC-Tiger®) auswerten, muß man solche fehlerhaften Zwischenstellungen vermeiden. Was aber tut man, wenn man auf dem gleichen Prinzip einen Drehmelder aufbauen will, der keine Raststellungen haben und auch zwischendurch korrekte Werte abgeben soll?
Gäbe es einen Code, bei dem von Stellung zu Stellung nur ein einziges Bit verändert, wäre alles eindeutig, auch auf den "kritischen" Zwischenstellungen wird kein falscher Wert erzeugt, lediglich ein Springen zum Nachbarwert ist möglich. Einen solchen Code gibt es (sogar mehrere), im vorliegenden Applikationsbericht wählen wir den Gray-Code als Basis für verschiedene Experimente.

Bild 1 ein binär codierter Wahlschalter

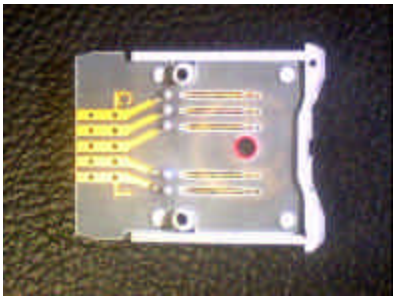

Bild 2 Rückansicht

Bild 3 Schleifer und codier–te Printplatte

3. Der Gray-Code

Der Aufbau des Gray-Codes ist ebenfalls systematisch, er läßt sich aber nicht mehr aus einer so einfachen mathematischen Beziehung ableiten wie der Binär-Code. Zunächst eine Darstellung analog der im vorigen Kapitel:

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

Noch deutlicher wird die Eigenschaft des Gray-Codes, nur 1 Bit pro Schritt zu ändern, in einer grafischen Darstellung. Die "1"-Bits sind rot, die "0"-Bits weiß gekennzeichnet. Links beginnt die Grafik mit der Zahl 0, im Beispiel werden nur 6 Bits dargestellt, der Gray-Code läßt sich natürlich auch mit mehr Bits realisieren:

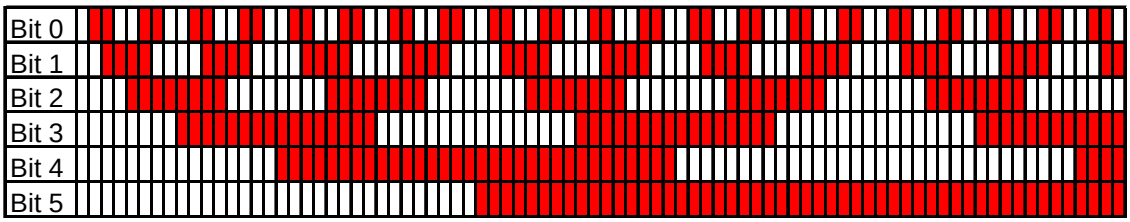

Bild 4 Der Gray-Code als Codelineal

Wie man sieht, hat auch dieser Code seine Systematik. Jede Position auf der X-Achse hat ihren eindeutigen Wert, pro Schritt wird nur ein Bit geändert. Dies gilt auch am Ende des Code-Lineals, so daß z.B. Kreis-Segmentscheiben mit "nahtlosem" Übergang vom Ende zum Anfang realisierbar werden. Voraussetzung dafür ist, daß alle Werte für die Gesamtbitzahl dargestellt werden, bei 8 Bit Breite also genau 256 Schritte. Wenn man nun aus diesem Wert einen binären (und damit sofort auch dezimalen) Wert bilden könnte, ist eine absolute Positionsbestimmung möglich. Im Tiger-BASIC kann man das entweder mit einer Tabellenzuordnung oder mit folgender Rechenvorschrift tun.

Gegeben sei eine Grayzahl a in 8 Bit (im Beispiel 00010100). Dazu gibt es eine weitere Zahl b, die gleich 0 gesetzt wird. Vom höchstwertigen Bit angefangen wird bitweise die XOR-Funktion aus a und b (=0) gebildet. Das Ergebnisbit wird als das XOR-Verknüpfungsbit für die nächste Stelle genommen. In der höchstwertigen Stelle ist a = 0, b als Ausgangszahl ebenfalls 0. Die XOR-Verküpfung liefert 0, wenn beide Argumente gleich sind, also auch 0. Diese 0 sowie die zweithöchste Stelle von a sind über XOR verknüpft ebenfalls 0. Das ist auch bei der nächsten Stelle so. Dann aber passiert etwas neues, die 4. Stelle von a ist 1, mit dem vorherigen Ergebnisbit (0) XOR-verknüpft gibt das eine 1. Das folgende Schema zeigt den Ablauf:

Grayzahl a 00010100 Hilfsvariable b 00000000

```
a = 00010100          b = 00000000
    |||||||               |
    0|||||| -> XOR  <- 0  /---- = 0
     0|||||| -> XOR  <----/---- = 0
      0|||||| -> XOR  <----/---- = 0
       1|||| -> XOR  <----/---- = 1
        0||| -> XOR  <----/---- = 1
         1|| -> XOR  <----/---- = 0
          0| -> XOR  <----/---- = 0
           0 -> XOR  <----/      = 0
                                  |
                                  \---> c = 00011000
```

Ergebnis Binärzahl c 00011000 (24)

Bild 5 Das Berechnungsschema Gray-Code zu Binär-Code

Wer möchte, kann im Bild 4 auf dem Codelineal des Gray-Codes nachsehen, ob das Ergebnis stimmt. Das 24. Feld von links (Feld 0 nicht vergessen!) müßte den Gray-Code 00010100 zeigen - stimmt!

4. Etwas Hardware

Wenn man sich die roten Flächen in der Gray-Code-Darstellung von Bild 4 als Kupferflächen und die weißen als Epoxy-Flächen einer Leiterplatte vorstellt, hat man bereits ein einfaches elektronisches Lineal. Die Kupferfläche liegt auf +5V, glücklicherweise sind alle Teile miteinander verbunden (das wäre beim Binär-Code nicht gegeben!). Jedes Bit bekommt einen Schleifkontakt, der mit einem Port des BASIC-Tiger® verbunden ist. Bei einem 8 Bit Codelineal könnte man 256 Positionen erfassen. Das Ganze kann man sich auch mit einer Kreisscheibe als Winkelgeber vorstellen (Bild 6), hier wird eine Umdrehung in 256 Einzelschritte zerlegt. Neben der einfachen Schleifkontakt-Technologie lassen sich auch kleine Lichtschranken einsetzen, dann besteht das Lineal oder die Segmentscheibe aus einer belichteten Folie. Wo diese schwarz ist, wird kein Licht durchgelassen. Ideal für solch eine Anwendung sind Miniatur-LED's (meist infrarot) und dazugehörige Miniatur-Fototransistoren (Bild 7), die sich im 1/10 Zoll-Abstand montieren lassen. Die eigentlichen Lichtschranken sollten dann in ihrer Empfindlichkeit einstellbar sein und Schmitt-Trigger-Verhalten aufweisen. Gut geeignet für eine solche Mini-Lichtschranke ist z.B. ein Aufbau mit CD4093 (Bild 8). Eingebaut in ein Gehäuse könnte das Ganze dann ein elektronischer "Wetterhahn" oder ein Drehmelder für die Antennenstellung werden.

Die hier vorgestellten Details sind für den Bastler vielleicht schon ausreichend, auf jeden Fall bieten sie Anregungen für eigene Experimente. Es gibt aber auch professionelle Winkelgeber mit viel höherer Genauigkeit, die haben allerdings ihren Preis. Wem das Prinzip der absoluten Positionsbestimmung ohne vorherigen Reset oder ohne Initialisierung gefällt, der kann sich z.B. bei im Internet umsehen, hier gibt es Winkel-Sensoren mit 8129 (!) Schritten pro Umdrehung (http://www.pepperl-fuchs.com/main_d.html).

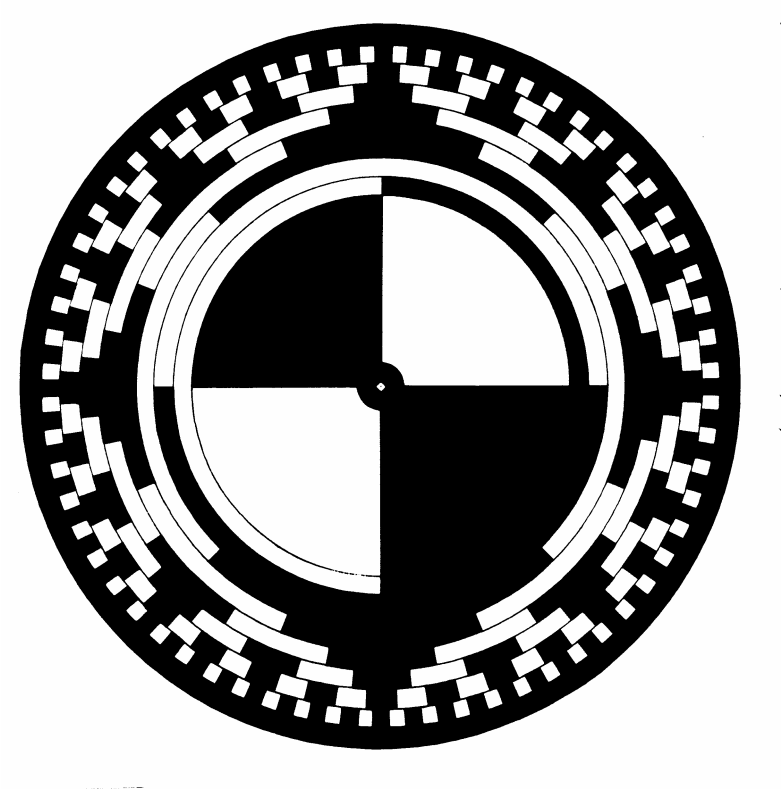

Bild 6 Eine Segmentscheibe für einen Winkelsensor

Bild 7 Eine selbstgebaute Fototransistor–zeile im 1/10-Zoll-Raster für 8 Bit

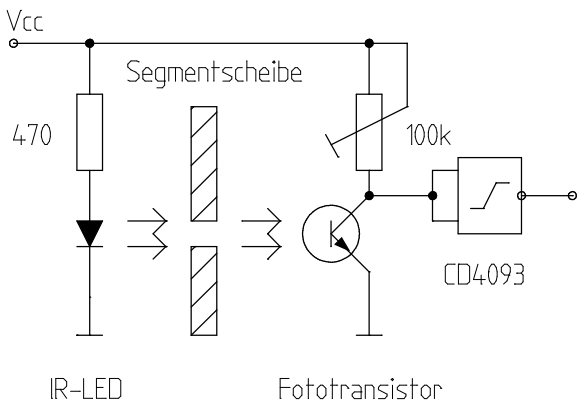

Bild 8 Schaltung mit CD 4093, hier nur ein Kanal dargestellt

5. Software

Wer bis hierher gekommen ist, weiß, daß jetzt ein Programm für den BASIC-Tiger® folgen muß, mit dem ein an einem Port ankommender 8-Bit-Wert aus dem Gray-Code in eine

Binärzahl umgewandelt werden kann. Diese wird dann auf dem Tiger-Display binär und auch dezimal angezeigt.

```
'--------------------------------------------------------------------
' Name: GRAY01.TIG
' Wandelt den am Port 8 anliegenden 8-Bit-Gray-Code in die entsprechende
' Binaerzahl um. Es wird der Gray-Code, die dazugehoerige Binaerzahl sowie
' die Dezimalzahl auf dem LCD-Display angezeigt
'--------------------------------------------------------------------
TASK MAIN                                   ' Beginn Task MAIN
  BYTE d, e, f                              ' Hilfsvariable zum Rechnen
  LONG n, a, b                              ' n als Zaehlvariable
                                            ' a als Gray-Eingangsvariable
                                            ' b als zu berechnende Binaerzahl
  INSTALL DEVICE #1, "LCD1.TDD"             ' LCD-Treiber installieren (BASIC-Tiger)

  DIR_PORT 8,255                            ' Port 8 als Eingang setzen
  USING "UD<1><1>   0.0.0.0.0"              ' Format fuer Binaer-Ausgabe

Anfang:                                     ' Marke Anfang
  b = 0                                     ' Vorbesetzung von b als Hilfsvariable
  IN 8, a                                   ' Gray-Code an Port 8 einlesen
  WAIT_DURATION 200                         ' etwas warten

  PRINT #1, "<1>Gray a = ";                 ' Ausgabe der eingelesenen Grayzahl a
                                            ' Gray-Code in Bitdarstellung
  FOR n = 7 TO 0 STEP -1                    ' Schleife ueber alle Bits
    PRINT USING #1, BIT (a,n);              ' bitweise Ausgabe von a
  NEXT                                      ' naechstes Bit
  PRINT #1, "b"                             ' Ausgabe der Binaer-Kennung

                                            ' Ausrechnung der Binaerzahl
  FOR n = 7 TO 0 STEP -1                    ' Schleife ueber die letzten 7 Bit
    d = BIT (a, n)                          ' das n-te Bit von a abfragen
    e = BIT (b, n + 1)                      ' das (n+1)-te Bit von b abfragen
    f = d BITXOR e                          ' beide mit XOR verknuepfen
    IF f = 1 THEN                           ' wenn das 1 ergibt (Bits nicht gleich)
      SET_BIT b, n                          ' das n-te Bit von b auf 1 setzen
    ENDIF                                   ' wenn nicht, das Bit auf 0 stehen lassen
  NEXT

                                            ' Binaerzahl in Bitdarstellung
  PRINT #1, "Bin. b = ";                    ' Ausgabe der neugebildeten Binärzahl b
  FOR n = 7 TO 0 STEP -1                    ' Schleife ueber alle Bits
    PRINT USING #1, BIT (b,n);              ' bitweise Ausgabe von b
  NEXT                                      ' naechstes Bit
  PRINT #1, "b"                             ' Ausgabe der Binaer-Kennung

                                            ' Dezimalzahl ausgeben
  PRINT #1, "b = ";b                        ' abschliessend noch den Dezimalwert b

  GOTO Anfang                               ' neuer Durchlauf von Anfang

END                                         ' Ende Task MAIN
```