2.1 Codierung/Was ist Codierung?

2.1.1 Was ist Codierung?

Wenn man sich den RAM-Inhalt eines Computers oder den Inhalt einer Datei auf einem Massenspeicher oder einen Mitschnitt des Verkehrs an einer Netzwerkschnittstelle ansieht, dann fällt zunächst auf, wie unübersichtlich das ist: Enorm viele Nullen und Einsen, aber nirgends eine Anleitung, was was bedeutet. Daten sind nämlich immer abhängig vom Kontext, in dem sie sich befinden. So gibt es

- Zahlen in Datenbanken, kaufmännischen Programmen, Tabellenkalkulationen
- Buchstaben und Satzzeichen in Texten und Konfigurationsdateien
- Befehlswort in Programmen

Man sagt, die Daten wurden – je nach Anwendungsbereich – *codiert*. Codierung ist die Zuordnung zwischen Zeichen oder Zeichenfolgen zu Bedeutungen.

2.1.2 Einteilung von Codes

Jetzt kann man Codes in zwei Dimensionen einteilen:

a) Wie viele verschiedene Zeichen habe ich (=mein Alphabet)?

   1) Bei einem binären Code werden die Zeichen meistens mit 0 und 1 bezeichnet (seltener mit 0 und L, auch selten mit L und H).
   2) Bei einem ternären Code gibt es drei mögliche Zeichen, z. B. 0, 1 und -1.

b) Welche Bedeutungen müssen unterschieden werden?

   1) Bei einem *numerischen* Code werden Zahlen unterschieden, z. B. Gleitkommazahlen bei IEEE-754
   2) Bei einem *alphanumerischen* Code werden Schriftzeichen unterschieden, also Buchstaben, Satzzeichen und Ziffernzeichen
   3) Bei einem *Befehlssatz* werden CPU-Befehle unterschieden, meistens für eine spezielle CPU

Bedeutungen/Alphabet	Binärcodes	Ternärcodes
Numerische Codes	Natürlicher Binärcode Zweierkomplement BCD IEEE-754	z. T. bei DSPs
Alphanumerische Codes	CCITT-2 ASCII UNICODE	Morsecode
Befehlssätze	AVR, AMD-64, ...	

Tabelle 1: Übersicht über Codes mit Beispielen

Tabelle 1 zeigt Beispiele.

2.1.3 Numerische Codes

Es gibt viele Arten, im Computer Zahlen darzustellen. Die meisten sind an eine Systematik, z. B. ein Zahlensystem, angelehnt. Häufig kommen vor:

a) Zahlen im natürlichen Binärcode mit 8, 16, 32, 64 oder 128 Bit Breite

b) Zahlen im Zweierkomplement mit 8, 16, 32, 64 oder 128 Bit Breite

c) Gleitkommazahlen nach IEEE-754

d) Festkommazahlen im BCD-Code

   1) mit 4 Bit Breite für eine Ziffer (*packed BCD*)
   2) mit 8 Bit Breite für eine Ziffer (*unpacked BCD*, die vier führenden Bits sind dabei 0)

Andere Formate sind seltener und für bestimmte Zwecke optimiert.

Im Bereich der Weg- oder Winkelmessung kommt häufig der Gray-Code zum Einsatz. Es handelt sich dabei um einen Code, bei dem sich im Übergang von einem Wert zum nächsten immer nur ein Bit ändert (Beispiel für 4 Bit siehe Tabelle 2). Man sagt, dieser Code ist *einschrittig*. Das verhindert Fehlmessungen beim Übergang zwischen zwei Werten, die beim sonst üblichen

I3	I2	I1	I0	Wert
0	0	0	0	0
0	0	0	1	1
0	0	1	1	2
0	0	1	0	3
0	1	1	0	4
0	1	1	1	5
0	1	0	1	6
0	1	0	0	7
1	1	0	0	8
1	1	0	1	9
1	1	1	1	10
1	1	1	0	11
1	0	1	0	12
1	0	1	1	13
1	0	0	1	14
1	0	0	0	15

Tabelle 2: Graycode

natürlichen Binärcode passieren könnten (siehe Abbildung 1): Der schräge Balken soll einen Träger

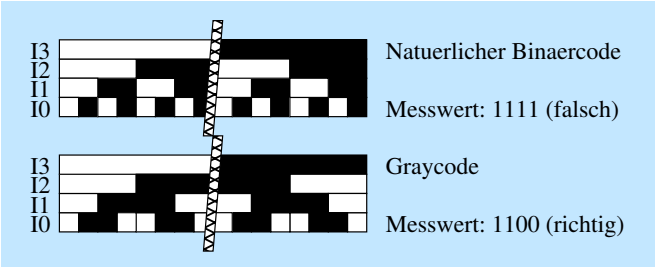

Abbildung 1: Fehlmessung beim natürlichen Binärcode

für optische Sensoren darstellen, der naturgemäß nie *ganz* senkrecht zum Verfahrweg stehen kann.

Beim natürlichen Binärcode entsteht dadurch ganz kurzzeitig ein falscher Wert; das könnte fatale Auswirkungen haben. Beim Graycode dagegen kann kein falscher Wert entstehen, da sich immer ein Bit ändert.

2.1.4 Alphanumerische Codes

Alphanumerische Code sind zur Darstellung von Texten geeignet. Texte können enthalten:

- Buchstaben (das Alpha aus alphanumerisch)
- Ziffernzeichen (das Numerisch aus alphanumerisch)
- Satzzeichen
- Sonderzeichen wie Zeilenumbruch, Seitenvorschub, Tabulator

Ein sehr früher Code für diesen Zweck ist der Morsecode. Er hat ein Alphabet aus drei Zeichen:

- Punkt (`.`)
- Strich (–)
- Pause (␣)

Damit handelt es sich um einen ternären Code. Tabelle 3 zeigt einen Teil der Zuordnung. Man

Code			.	-	..	.-	-.	--
Zeichen			e	t	i	a	n	m
Code	...	..-	.-.	.--	-..	-.-	--.	---
Zeichen	s	u	r	w	d	k	g	o
Code	....	...-	..-.	..--	.-..	.-.-	.--.	.---
Zeichen	h	v	f	ü	l	ä	p	o
Code	-...	-..-	-.-.	-.--	--..	--.-	---.	----
Zeichen	b	x	c	y	z	q	ö	ch
Code	.....	....-	...--	..---	.----			
Zeichen	5	4	3	2	1			
Code	-...	--...	---..	----.	-----			
Zeichen	6	7	8	9	0			

Tabelle 3: Morsecode (Auszug)

sieht, dass der Morsecode nur Kleinbuchstaben enthält, allerdings auch Umlaute (und Akzente anderer europäischer Sprachen). Vorhanden, aber hier nicht aufgeführt sind Satzzeichen (Punkt, Komma, Strich, ...) und eine ganze Reihe von Steuerzeichen (hier Betriebszeichen genannt) wie BREAK oder ERROR (8 Punkte). Im Morsecode haben die einzelnen Codes unterschiedliche Länge: Die häufigeren Codes sind kürzer, die selteneren dafür um so länger. Von Menschen ist das leicht zu decodieren, von Maschinen eher nicht. Deshalb haben sich für das maschinelle Lesen (Fernschreibempfänger) von Nachrichten solche Codes durchgesetzt, bei denen jedes Zeichen gleich viele Bits hat. Dazu gehören der Baudot-Code (CCITT-1) und der Baudot-Murray-Code (CCITT-2). Das sind bereits binäre Codes. Beide benötigen für die Übertragung eines Zeichens nur fünf Bit. Beim Baudot-Code gehört beim Senden/Eintippen zu jedem gesetzten Bit eine mit einem Finger gedrückte Taste, daher die Fünf. Damit können 32 Zeichen unterschieden werden, was aber für 26 Buchstaben und 10 Ziffern nicht ausreicht. Deshalb haben die beiden Codes jeweils einen Umschaltcode hin zu Buchstaben (LS, *letter shift*) und einen Umschaltcode hin zu Ziffern und Sonderzeichen (FS, *figure shift*).

Mit der Entwicklung der Computertechnik wollte man weitere Zeichen ein- und ausgeben können, zum Beispiel Kleinbuchstaben und weitere Zeichen, die auf den Tastaturen der Schreibmaschinen zu finden waren. So wurden mehrere Codes eingeführt, die leider untereinander nicht kompatibel waren. Der bekannteste davon ist EBCDIC, ein 8-Bit-Code der Firma IBM, der heute noch auf Großrechnern verwendet wird.

Das Nebeneinander der verschiedenen Codes führte dazu, dass man sich über einen einheitlichen Standard-Code zum Austausch zwischen verschiedenen Systemen einigen musste. Dazu wurde der ASCII (*american standard code for information interchange*) entwickelt.

2.1.5 Befehlssatz einer CPU

Der Befehlssatz einer CPU ordnet jedes Bitmuster aus dem RAM oder ROM, das zum Befehlsdecoder gelangt, einer bestimmten Aktion, einem sogenannte Befehl (=*instruction*) zu. In Tabelle 4 findet man einen Auszug aus dem Befehlssatz der ATMEL-AVR-Chips. Die Bitmuster sind 16 Bit breit, damit sind theoretisch bis zu 65536 verschiedene Befehle möglich. In der Praxis bleiben manche ungenutzt. Außerdem haben einige Befehle Bits, mit denen zwischen verschiedenen Register oder Adressen ausgewählt werden kann. Beim ATmega8 kommt man damit (je nach Zählung) auf ca. 67 Befehle. In der Praxis nimmt man beim Programmieren statt der Bitmuster die sogenann-

Bitmuster	Mnemonic	Bedeutung
0000 11ab bbbb aaaa	ADD	Addiere a und b, Ergebnis in b
1001 0101 1001 1000	BREAK	CPU vorläufig anhalten
1111 01aa aaaa a001	BRNE	Springe a Adressen nach vorn, falls Ergebnis 0 war
1001 010a aaaa 0000	COM	Negiere jedes Bit in Register a
1001 0100 0000 1001	IJMP	Springe an die Adresse, die im Z-Register liegt
1011 0aab bbbb aaaa	IN	Lade Register b mit Eingangsport a
1001 010a aaaa 0011	INC	Erhöhe Register a um 1
1110 aaaa bbbb aaaa	LDI	Speichere Wert a in Register b
0010 11ab bbbb aaaa	MOV	Kopiere Register a nach Register b
0000 0000 0000 0000	NOP	tue nichts
1011 1aab bbbb aaaa	OUT	Lade Ausgangsport a mit Register b
1100 aaaa aaaa aaaa	RJMP	Springe a Adressen nach vorn

Tabelle 4: Befehlssatz der CPU des Mikrocontrollers ATMEL ATmega8

ten *Mnemonics*, Hilfswörter, die die Befehle erklären und dennoch sehr kurz sind. Ein sogenannter *Assembler* (=Assemblierer) wandelt jedes Mnemonic in das entsprechende Bitmuster um. So wird bei der CPU ATmega8 der Befehl INC r7 (erhöhe Register 7 um eins) zu folgendem Bitmuster[1]:
`1001 010`**`0 0111`**` 0011`
Glücklicherweise brauchen nur manche Spezialisten Programme in dieser Assembly-Sprache (=Assembler-Sprache) zu schreiben, und zwar immer dann, wenn es um Geschwindigkeits- und Speicherplatz-Optimierung geht. Alle anderen können in den wesentlich einfacher zu erlernenden Hochsprachen (C, Rust, Java, Python, ...) programmieren.

[1] In den fett gedruckten Bits ist die Nummer des betreffenden Registers codiert