1.9 Atmega-Programmierung in ASM/LED-Ziffernanzeige

1.9.1 Idee

Bei der Programmentwicklung braucht man es ab und zu, dass man sich an bestimmten Stellen des Programms Variablenwerte anzeigen lässt.

Bei der Entwicklung in C auf PCs etwa fügt man an diesen Stellen einen Aufruf der Funktion `printf()` ein. Auf einem Mikrocontrollersystem dagegen besteht diese Möglichkeit nicht. Deshalb soll hier eine kleine Ziffernanzeige angeschlossen werden, die es erlaubt, den Inhalt eines 8-Bit-Registers dezimal auszugeben.

1.9.2 Hardware

Bezüglich der verwendeten Technologie gibt es viele Möglichkeiten, von denen hier nur einige genannt sind:

a) Vakuum-Fluoreszenz-Anzeige, bekannt aus Registrierkassen, oft 20 alphanumerische Stellen mit Punktmatrix, serielle Ansteuerung per RS232

b) LCD-Anzeige, oft 1x16 bis 2x40 alphanumerische Stellen mit Punktmatrix, parallele Ansteuerung mit HD44780-Controller

c) LED-Anzeige mit Punktmatrix, parallele Ansteuerung mit proprietären Protokollen

d) LED-Siebensegmentanzeige, eine Dezimalstelle pro Baustein, angesteuert über einen getrennten Decoder

e) LED-Siebensegmentanzeige, eine Dezimalstelle pro Baustein mit integriertem Decoder; auch als Punktmatrixanzeige

Die ersten drei Lösungen scheinen attraktiv, erfordern aber häufig komplizierte Ansteuerungsroutinen. Deshalb werden hier die letzten beiden Lösungen favorisiert.

Bei den Lösungen mit Siebensegmentanzeige ist noch zu fragen, in welchem Zahlensystem ausgegeben werden soll. Daraus ergibt sich auch, wie viele Stellen nötig sind:

a) dreistellige Ausgabe 000–255 im Dezimalsystem

b) zweistellige Ausgabe 00-FF im Hexadezimalsystem

c) zweistellige Ausgabe 00–99 dezimal und wahlweise 00–FF hexadezimal

Der Flexibilität halber soll wahlweise dezimal oder hexadezimal ausgegeben werden, und zwar mit einer zweistelligen Anzeige.

Für die hexadezimale Ausgabe gibt es mehrere Bausteine:

a) V40511 (RFT, ehemals DDR): Decoder, evtl. sehr günstige Restbestände; ähnlich: 74C915, CMOS 4311, CMOS 4368, F9368, D345, D346

b) HP5082-7340: Anzeige mit integriertem Decoder; sehr gut ablesbar und noch bezahlbar; ähnlich: TIL305, TIL311

Was die Ansteuerung mehrerer Anzeige von einem Mikrocontroller aus angeht, gibt es wieder mehrere Möglichkeiten:

a) Parallelansteuerung

b) Multiplexing

c) Charlieplexing

Hier soll aus Gründen der Einfachheit die Parallelansteuerung zweier Anzeigen mit integrierten Decodern (HP5082-7340) verwendet werden.

Beide Anzeigen werden über eine zehnpolige Verbindung (Belegung nach RN-Standard) an einen Mikrocontroller-Port angeschlossen werden. Die linke Anzeige ist mit den oberen vier Bits, die rechte Anzeige mit den unteren vier Bits verbunden (siehe Abbildung 1).

1.9.3 Hexadezimale Ausgabe

Da der Hexadezimal-Decoder bereits integriert ist, funktioniert die hexadezimale Ausgabe sehr einfach (`ziffanz1.asm`):

```
1  .include "/usr/share/avra/m32def.inc"
2          ldi r16, 0xFF
3          out DDRC, r16
4          ldi r16, 0
5          out PORTC, r16
6          ;------------------ Stackpointer setzen:
7          ldi r16, low(RAMEND)
8          out SPL, r16
9          ldi r16, high(RAMEND)
10         out SPH, r16
11         ;------------------- Hauptprogramm
12 main:
13         ldi r16, 42
14         rcall ziffhex
15 ende:
16         rjmp ende
17         ;------------------- Unterprogramm
18 ziffhex:
19         out PORTC, r16
20         ret
```

Die Ausgabe ist hier in ein einfaches Unterprogramm mit dem Namen `ziffhex` gepackt. In `r16` erwartet das Unterprogramm die Angabe der Zahl, die ausgegeben werden soll. `r16` ist hier also gewissermaßen der erste (und einzige) Parameter einer Prozedur bzw. Funktion (mit den Worten der höheren Programmiersprachen ausgedrückt).

Dies ist die einfachste Art, an Unterprogramme Parameter zu übergeben. Es gibt noch andere Arten, etwa

- über den Stack
- über globale Variablen

1.9.4 Dezimale Ausgabe einer zweistelligen Zahl

Nun soll der Wert dezimal ausgegeben werden. Der Registerinhalt 42 soll also nicht mehr mit `2A` angegeben werden, sondern mit `42`. Wie bekommt man aber mathematisch aus der 42 die 4 und

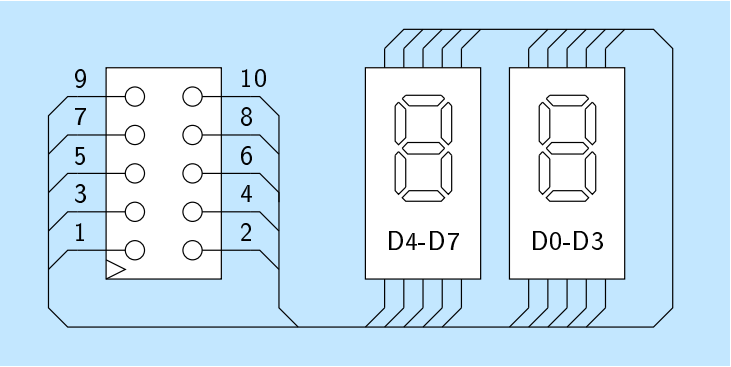

Abbildung 1: Anschluss der Ziffernanzeige

die 2 heraus?

1.9.4.1 Divisionsmethode Die Umwandlung einer Zahl in ein neues Zahlensystem kann über eine fortgesetzte Division durch die Basis des neuen Zahlensystems geschehen. In unserem Fall sind nur zwei Stellen vorhanden, was die Sache vereinfacht:

- Zehnerstelle: 4 = 42 geteilt durch 10 (bei ganzzahliger Division)
- Einerstelle: 2 = 42 modulo 10 (Divisionsrest)

Also ist die Sache klar: Eine einzige Division löst das Problem.

1.9.4.2 Division beim AVR Leider hat der AVR keinen Divisionsbefehl (Multiplizieren kann er aber). Deshalb führt man die Division auf die Subtraktion zurück[1]. Man zieht von 42 so oft die Zahl 10 ab, bis man eine Zahl kleiner als 10 erhält. Die Anzahl der Subtraktionen ist dann das Ergebnis. Abbildung 2 zeigt das zughörige Struktogramm. Daraus kann man ein Flussdiagramm

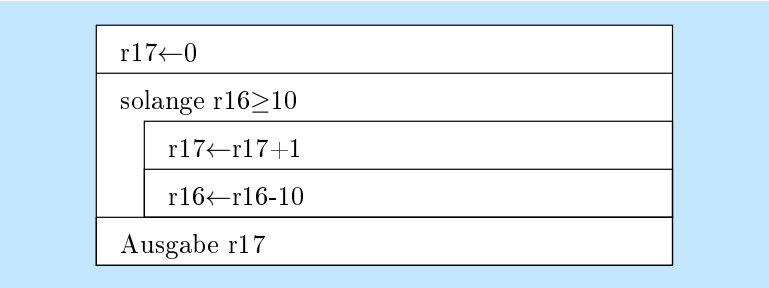

Abbildung 2: Division durch Subtraktion

erstellen, das sich schon näher am Assembler-Programm orientiert (Abbildung 3). Und schließlich

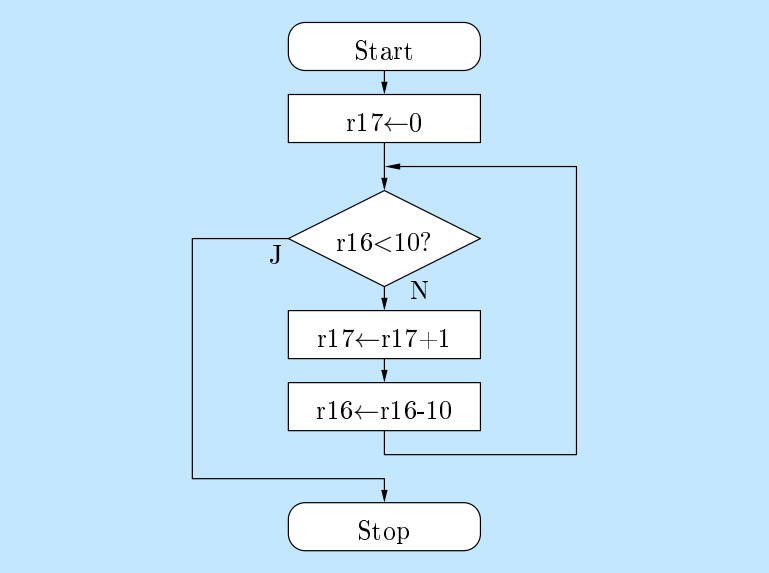

Abbildung 3: Division durch Subtraktion

entsteht daraus das Programm selbst (`ziffanz2.asm`):

[1] Es geht auch eleganter, wie bei Schmitt nachzulesen ist. Aber die dortige Methode ist komplizierter.

```
1  .include "/usr/share/avra/m32def.inc"
2          ldi r16, 0xFF
3          out DDRC, r16
4          ldi r16, 0
5          out PORTC, r16
6          ;------------------ Stackpointer setzen:
7          ldi r16, low(RAMEND)
8          out SPL, r16
9          ldi r16, high(RAMEND)
10         out SPH, r16
11         ;------------------ Hauptprogramm
12 main:
13         ldi r16, 42
14         rcall ziffzehner
15 ende:
16         rjmp ende
17         ;------------------ Unterprogramm
18 ziffzehner:
19         push r16
20         push r17
21
22         ldi r17, 0
23 _nochmal:
24         cpi r16, 10
25         brlt _weiter
26         subi r16, 10
27         inc r17
28         rjmp _nochmal
29 _weiter:
30         out PORTC, r17
31
32         pop r17
33         pop r16
34         ret
```

1.9.4.3 Bit-Verschiebung Noch wird die Ziffer 4 in der rechten Ziffernanzeige dargestellt, die von den Bits 0 bis 3 angesteuert wird. Will man sie auf die linke Anzeige bringen, muss man das Ergebnis um vier Bit nach links verschieben. Dazu gibt es den Befehl `lsl` (*logical shift left*), der ein Register um genau ein Bit nach links verschiebt und von rechts her Nullen auffüllt (entsprechend einer Multiplikation mit dem Faktor zwei). `ziffanz3.asm` zeigt das Ergebnis (Ausschnitt):

```
1  _weiter:
2          lsl r17
3          lsl r17
4          lsl r17
5          lsl r17
6          out PORTC, r17
```

1.9.4.4 Berechnung des Restes Wie kann man nun die Einerstelle darstellen? Sie entsteht automatisch bei der Division durch Subtrahieren in dem Moment, indem ein Subtraktionsergebnis kleiner als 10 ist. Dieses Ergebnis ist dann die gesuchte Stelle, wie man in `ziffanz4.asm` sieht (Ausschnitt):

```
_weiter:
        out PORTC, r16
```

1.9.4.5 Zusammenfassen der 8 Bit Nun müssen die vier Bit in `r17` und die vier Bit in `r16` noch zusammengefasst werden. Das gelingt entweder durch Addition oder einfach durch eine ODER-Verknüfung. Dazu gibt es den Befehl `or A, B`, der `A` und `B` bitweise ODER-verknüpft und das Ergebnis nach `A` schreibt. Das Ergebnis zeigt `ziffanz5.asm` (Ausschnitt):

```
_weiter:
        lsl r17
        lsl r17
        lsl r17
        lsl r17
        or r17, r16
        out PORTC, r17
```

1.9.4.6 Ergebnis Das Ergebnis sieht man in `ziffanz5.asm`:

```
.include "/usr/share/avra/m32def.inc"
        ldi r16, 0xFF
        out DDRC, r16
        ldi r16, 0
        out PORTC, r16
        ;------------------ Stackpointer setzen:
        ldi r16, low(RAMEND)
        out SPL, r16
        ldi r16, high(RAMEND)
        out SPH, r16
        ;------------------ Hauptprogramm
main:
        ldi r16, 42
        rcall ziffzehner
ende:
        rjmp ende
        ;------------------ Unterprogramm
ziffzehner:
        push r16
        push r17

        ldi r17, 0
_nochmal:
        cpi r16, 10
        brlt _weiter
        subi r16, 10
        inc r17
        rjmp _nochmal
_weiter:
        lsl r17
        lsl r17
        lsl r17
        lsl r17
        or r17, r16
```

```
35          out PORTC, r17
36
37          pop r17
38          pop r16
39          ret
```