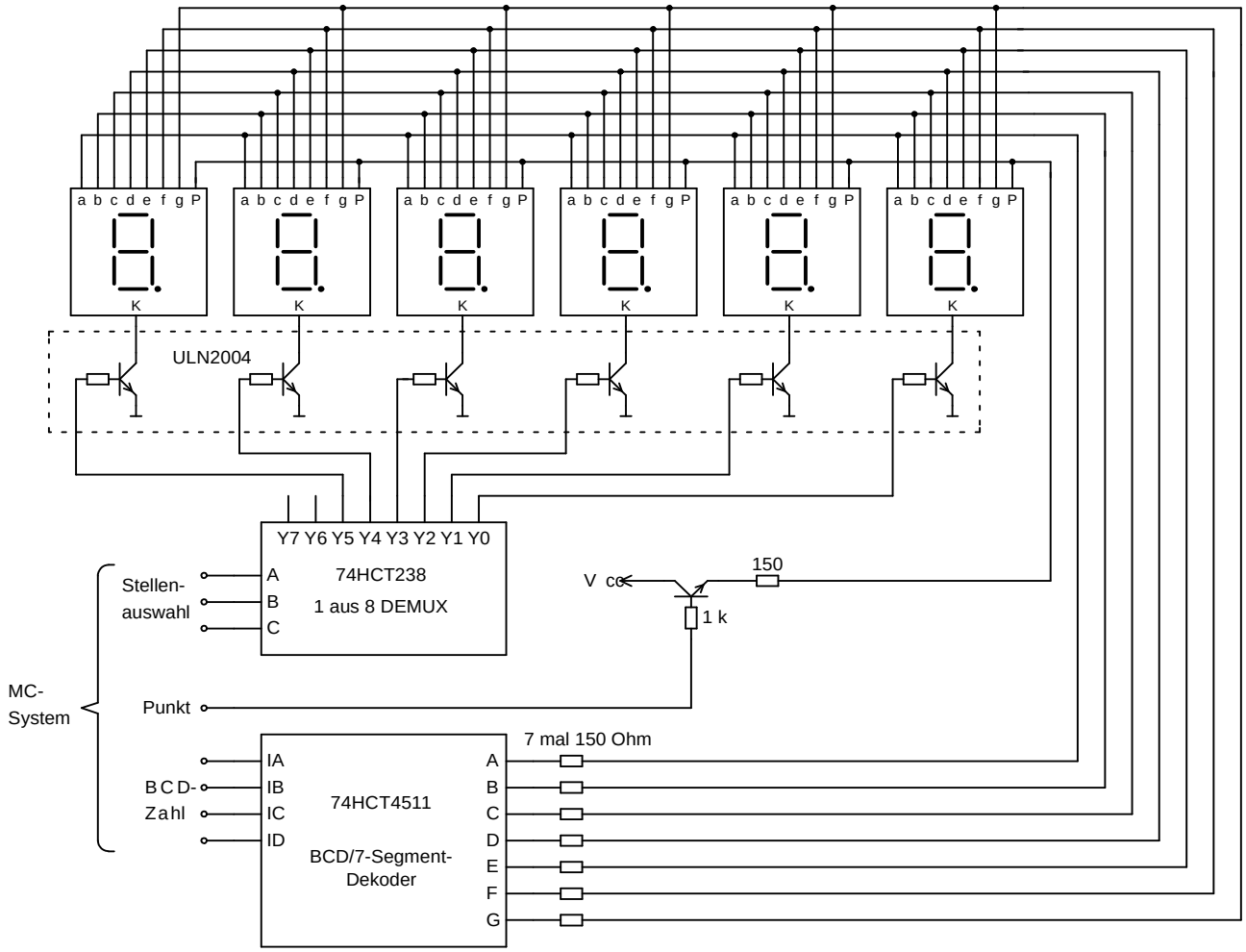
Eine Multiplex-Anzeige

Mit diesem Projekt wird das Ziel verfolgt, eine mehrstellige numerische Anzeige für Mikrocontroller-Systeme zu realisieren. Die Multiplex-Anzeige stellt eine preiswerte und ansprechende Möglichkeit dar, um zum Beispiel Messergebnisse gut sichtbar darzustellen. Sie lässt sich im Gegensatz zu einer LCD-Anzeige leicht programmieren und ist mit 6 Stellen für viele Anwendungen einsetzbar. In diesem Kapitel werden zunächst die Wirkungsweise, die schaltungstechnische Realisierung und ein Testprogramm beschrieben.

Blockschaltung der 7-Segment-Anzeige:



Eine 7-Segment-Anzeige besteht aus 8 einzelnen LEDs, welche einzeln angesteuert werden. Die Segmente a bis g gestatten eine Anzeige der Zahlen von 0 bis 9. Eine weitere LED ist für einen Dezimalpunkt vorgesehen.

7-Segment-Anzeige:

Alle LEDs haben einen gemeinsamen Kathodenanschluss. Im Schaltplan sind die Segmente a bis g aller Anzeigen parallel miteinander verbunden. Jede einzelne 7-Segment-Anzeige liegt mit ihrem Kathodenanschluss über einen Transistor an Masse. Ein High-Signal auf die Basis des entsprechenden Transistors schaltet die Kathoden nach Masse und lässt die entsprechende Anzeigenstelle leuchten.

Die erforderliche BCD-7-Segment-Codewandelung erfolgt durch den IC-Baustein 74HC4511. Die Wahrheitstabelle dieses Bausteines ist in folgender Tabelle dargestellt. An den Eingängen IA bis ID dieses Bausteines wird eine BCD-Zahl angelegt. Das sind die Dualzahlen von 0b0000 bis 0b1001, entsprechend den Dezimalzahlen von 0 bis 9. An den Ausgängen A bis G wird der 7-Segment-Code ausgegeben.

Wahrheitstafel des 74HCT4511:

Eingänge				*Ausgänge*							*Anzeige*
ID	*IC*	*IB*	*IA*	*G*	*F*	*E*	*D*	*C*	*B*	*A*	
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	0	0	0	1	1	0	1
0	0	1	0	1	0	1	1	0	1	1	2
0	0	1	1	1	0	0	1	1	1	1	3
0	1	0	0	1	1	0	0	1	1	0	4
0	1	0	1	1	1	0	1	1	0	1	5
0	1	1	0	1	1	1	1	1	0	0	6
0	1	1	1	0	0	0	0	1	1	1	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	0	0	1	1	1	9
1	0	1	0	0	0	0	0	0	0	0	blank
1	0	1	1	0	0	0	0	0	0	0	blank
1	1	0	0	0	0	0	0	0	0	0	blank
1	1	0	1	0	0	0	0	0	0	0	blank
1	1	1	0	0	0	0	0	0	0	0	blank
1	1	1	1	0	0	0	0	0	0	0	blank

Der 7-Segment-Code liegt jetzt zwar an allen Anzeigen an, leuchten wird jedoch nur die Stelle, bei welcher der zugehörige Transistor den Kathodenanschluss an Masse durchschaltet. Das Durchschalten eines Transistors erfolgt mit einem High-Pegel an der Basis.

Der IC-Baustein 74HCT238 ist ein 1-aus-8-Demultiplexer/Decoder. In Abhängigkeit von den Eingangssignalen A, B und C wird nur einer der 8 Ausgänge Y0 bis Y7 auf High-Pegel geschaltet. Liegt an den Eingängen A, B und C beispielsweise 0b000 an, so wird der Ausgang Y0 auf High geschaltet und bringt damit die Stelle 1 zur Anzeige.

Die vollständige Wahrheitstafel des 74HCT238 zeigt folgende Tabelle dargestellt.

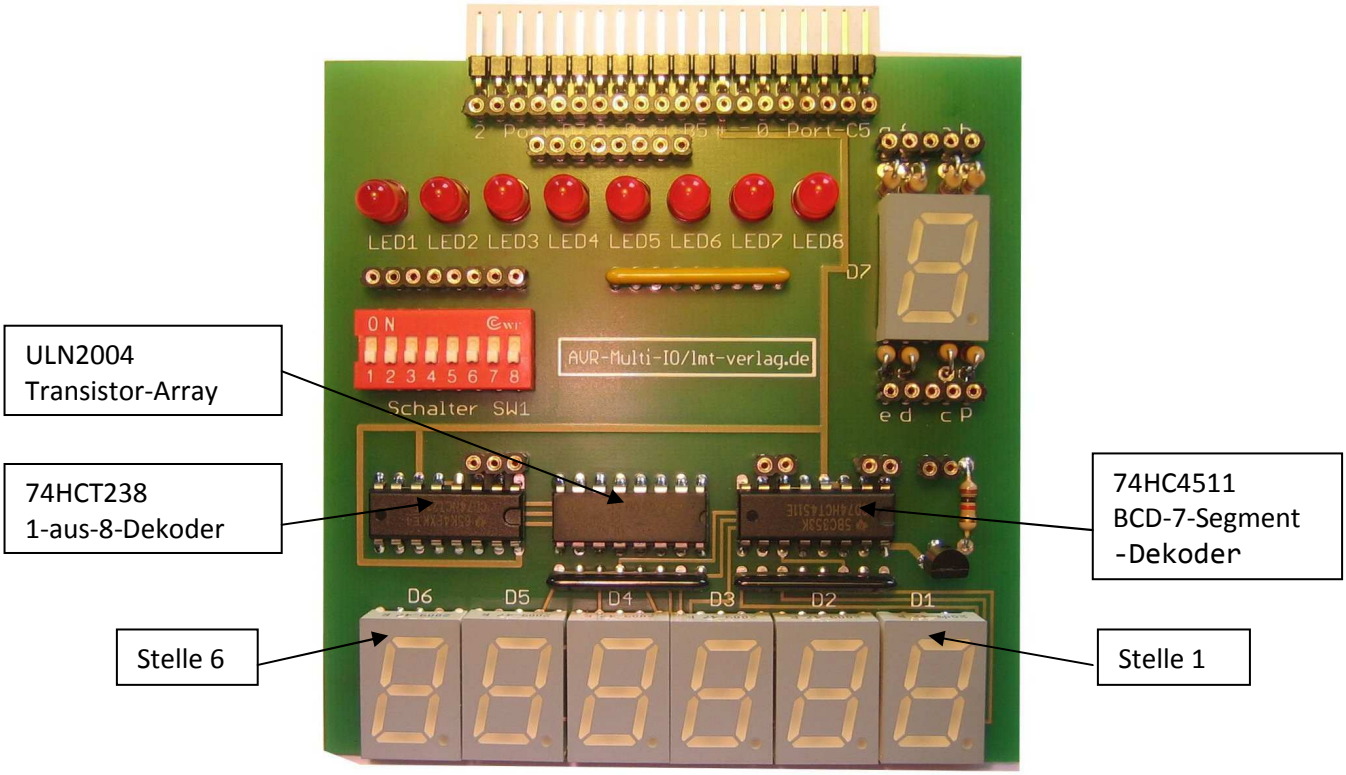
Wahrheitstafel des 74HCT238:

Eingänge			*Ausgänge*							
C	B	A	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Für das Verständnis der Multiplex-Anzeige ist zu beachten, dass zu einer Zeit immer nur eine Anzeige aktiv ist. Wählt man die Stellen genügend schnell nacheinander aus (Multiplex-Betrieb) so erscheint für das menschliche Auge eine ‚stehende' Anzeige. Um eine flimmerfreie Anzeige zu erhalten, sollten alle Stellen ca. 50 mal pro Sekunde ausgewählt werden.

Das bedeutet: Alle 1/50s = 20 ms sollte die komplette Anzeige erneuert werden. Das heißt wiederum bei einer 6-stelligen Anzeige, dass jede Stelle für 20ms/6 = 3,3ms (oder weniger) anzusteuern ist.

Das folgende Bild zeigt die Lage der Bauelemente auf der Platine.



Ein Testprogramm

Die Anzeige soll unabhängig von einem beliebigen Hauptprogramm erfolgen. Dazu wird eine Interrupt-Routine für den Timer0 eingerichtet, welche ca. alle 3ms aufgerufen wird. Global wird eine Software-Schnittstelle für die anzuzeigenden Werte und der aktuellen Stelle eingerichtet. Da sowohl das Hauptprogramm, wie auch die Interrupt-Routine auf diese Werte zugreifen, muss bei der Deklaration der Modifizierer `volatile` verwendet werden. Dadurch wird sichergestellt, dass die Werte bei einem Interrupt, welcher zu jedem beliebigen Zeitpunkt erfolgen kann, immer gültig sind. Bei Objekten, welche sowohl vom Hauptprogramm als auch von der Interrupt-Routine verändert werden können, sollte immer dieser Modifizierer verwendet werden.

```
...
volatile uint8_t wert[6]={0,0,0,0,0,0};        //Ausgabewerte
volatile uint8_t stelle=0;                     //Stellenzähler
...
```

In dem Array `wert[6]` werden die an den Stellen 1 bis 6 (Index 0 bis 5) auszugebenden Zahlenwerte eingetragen Die Variable `stelle` beinhaltet die Stelle an der das nächste Zeichen ausgegeben wird. Der Stellenzähler wird nach jedem Interrupt inkrementiert und bei Überschreiten von 5 auf 0 zurückgesetzt.

Das Programm:

```
/*****************************************
Datei: Beispiel_4_12_1.c
Beispiel: Mux-Anzeige
Belegung:
PD4 --> Pin 1 von 74LS138 - Stellenauswahl A
PD5 --> Pin 2 von 74LS138 - Stellenauswahl B
PD6 --> Pin 3 von 74LS138 - Stellenauswahl C
PB0 --> Pin 7 von 74HC4511 - Zahlenwert A0
PB1 --> Pin 1 von 74HC4511 - Zahlenwert A1
PB2 --> Pin 2 von 74HC4511 - Zahlenwert A2
PB3 --> Pin 6 von 74HC4511 - Zahlenwert A3
PB4 --> an Dezimalpunkt
*****************************************/
#include<avr/io.h>
#define F_CPU 3686400UL
#include <avr/interrupt.h>
volatile uint8_t wert[6]={0,0,0,0,0,0};             //Ausgabewerte
volatile uint8_t stelle=0;                          //Stellenzähler
int main(void)
{
DDRD|=(1<<PD4)|(1<<PD5)|(1<<PD6);                   //PD4...PD6 zur Ausgabe
DDRB|=(1<<PB0)|(1<<PB1)|(1<<PB2)|(1<<PB3)|(1<<PB4); //PB0...PB4 zur Ausgabe
TCCR0=0b00000011;           //interner Takt - F_CPU/64
TIMSK|=(1<<TOIE0);          //Timer0-Interrupt ermöglichen
TCNT0=83;                   //Startwert für ca. 3ms
sei();                      //Interrupts zulassen
```

```
while(1)                        //wiederhole immer
      {
      wert[0]=1;                //Ausgabewerte
      wert[1]=2;
      wert[2]=3;
      wert[3]=4;
      wert[4]=5;
      wert[5]=6;
      }
}
ISR(TIMER0_OVF_vect)
{
PORTB=wert[stelle];             //Wert ausgeben
PORTD=stelle<<4;                //Stelle ausgeben
stelle++;                       //nächste Stelle
if(stelle>5) stelle=0;          //Stellenzähler zurücksetzen
TCNT0=83;                       //Startwert des Timers
}
```

Ein Digitalvoltmeter

Eine analoge Eingangsspannung an PC0 des Mikrocontrollers von 0…5,1 Volt soll fortlaufend gemessen und auf einer dreistelligen Multiplex-Anzeige dargestellt werden. Bei einer Auflösung des A/D-Wandlers von 8 Bit ist die Genauigkeit der Messung 5,1 Volt/256 = 20 mV/Bit.

Die Darstellung der Spannung erfolgt mit einer Stelle vor und zwei Stellen nach dem Dezimalpunkt.

Die Referenzspannung von ca. 5,1V wird extern an Pin 21 des ATmega8 zugeführt (Vcc).

Das Programm:

```
/****************************************
Beispiel: Digitalvoltmeter
Belegung:
PD4 --> Pin 1 von 74LS138 - Stellenauswahl A
PD5 --> Pin 2 von 74LS138 - Stellenauswahl B
PD6 --> Pin 3 von 74LS138 - Stellenauswahl C
PB0 --> Pin 7 von 74HC4511 - Zahlenwert A0
PB1 --> Pin 1 von 74HC4511 - Zahlenwert A1
PB2 --> Pin 2 von 74HC4511 - Zahlenwert A2
PB3 --> Pin 6 von 74HC4511 - Zahlenwert A3
PB4 --> an Dezimalpunkt
****************************************/
#include<avr/io.h>
#include <avr/interrupt.h>
volatile uint8_t wert[6]={0,0,0,0,0,0};        //Ausgabewerte
volatile uint8_t stelle=0;                     //Stellenzähler

uint8_t GetADWert()
{
   ADMUX=0b00100000;                    //linksbündige Ausgabe; PC0
   ADCSRA|=(1<<ADEN|1<<ADPS2|1<<ADPS0); //ADEN=1 und Teilerfaktor 32
   ADCSRA|=1<<ADSC;                     //Wandelung starten
   while (ADCSRA & (1<<ADSC));          //warte bis ADSC = 0
   return(ADCH);
}
```

```
int main(void)
{
   uint8_t ADwert=0,tmp=0;
   DDRD|=(1<<PD4)|(1<<PD5)|(1<<PD6);                    //PD4...PD6 zur Ausgabe
   DDRB|=(1<<PB0)|(1<<PB1)|(1<<PB2)|(1<<PB3)|(1<<PB4); //PB0...PB4 zur Ausgabe
   TCCR0=0b00000011;                                    //interner Takt - F_CPU/64
   TIMSK|=(1<<TOIE0);                                   //Timer0-Interrupt ermöglichen
   TCNT0=83;                                            //Startwert für ca. 3ms
   sei();                                               //Interrupts zulassen

while(1)                                          //wiederhole immer
   {
      ADwert=GetADWert();
      wert[2]=((uint8_t)(ADwert/50))|0x010;     //Wert für Vorkommastelle mit Dezimalpunkt
      tmp=ADwert%50;
      wert[1]=(uint8_t)(tmp/5);    //Wert für Zehntelstelle
      wert[0]=(tmp%5)*2;           //Wert für Hundertstelstelle
   }
}

ISR(TIMER0_OVF_vect)
{
   PORTB=wert[stelle];             //Wert ausgeben
   PORTD=stelle<<4;                //Stelle ausgeben
   stelle++;                       //nächste Stelle
   if(stelle>2) stelle=0;          //Stellenzähler zurücksetzen
   TCNT0=83;                       //Startwert des Timers
}
```

Das folgende Bild zeigt den Aufbau mit dem AVR-Board.