Mikrocontrollerboard für 80C537 oder 80C517A

und parallele Ein- Ausgabeerweiterung für diese Mikrocontroller (oder andere).

Hermann Schönbauer PCN-DSK-520:/uc537

Diese Applikation besteht aus zwei einseitigen Europakarten. Die erforderliche 2. Verdrahtungsebene ist mittels Drahtbrücken realisiert und dadurch auch im Schul- oder Ausbildungsbereich nachbaubar. Keine Durchkontaktierungen!!

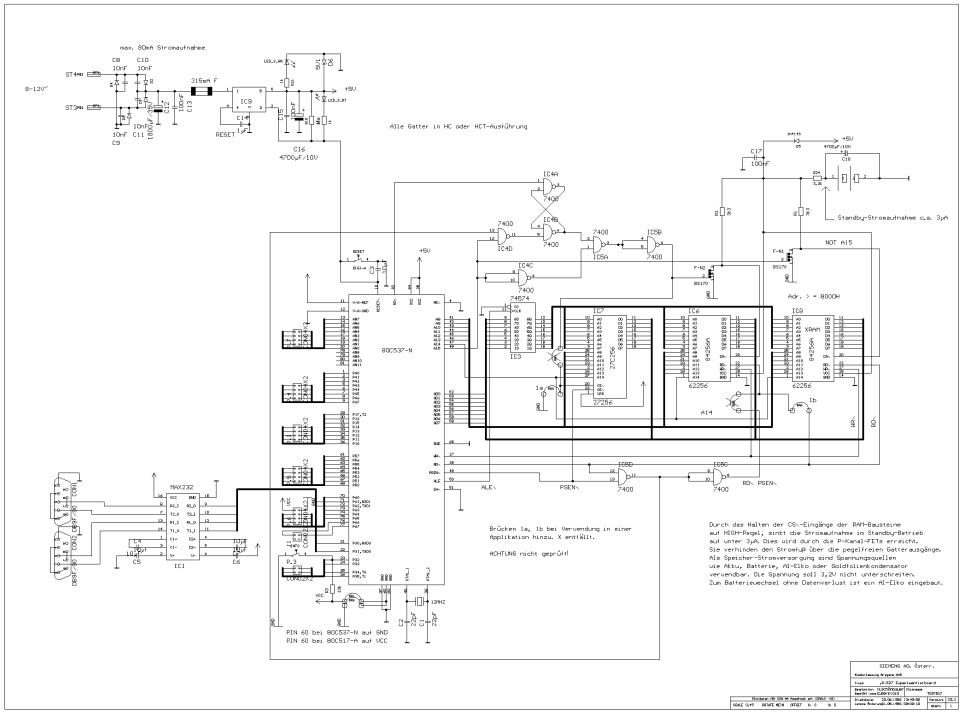
Mikrocontrollerboard

Das CMOS-RAM des Mikrocontrollerboards ist mit Batterie oder Akku bufferfähig (3µA). Es sind alle Ports ausgenommen Port 8 (4 A/D-Eingänge) herausgeführt und die zwei seriellen Schnittstellen auf V24 umgesetzt. Die Stromversorgung ist integriert, so daß nur eine Einspeisung von 8-12V Wechselspannung erforderlich ist. Bei Verwendung des 'KEIL-MONITORs' kann ein HEX-File über die S1-Schnittstelle downgeladen werden. Der Monitor (EPROM) schaltet sich dann auf die Adr. 8000H, wobei das downgeladene Programm im RAM steht.

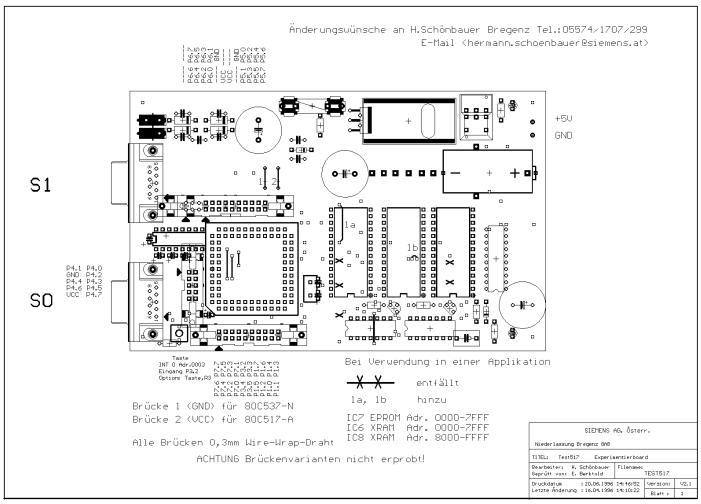
Das Handling mit einem Monitorprogramm ist der jeweiligen Monitorbeschreibung zu entnehmen. Es kann aus lizenzrechtlichen Gründen nicht weiter darauf eingegangen werden.

Alle Logikschaltungen in HC- oder HCT-Ausfürung.

Schaltbild



Layout



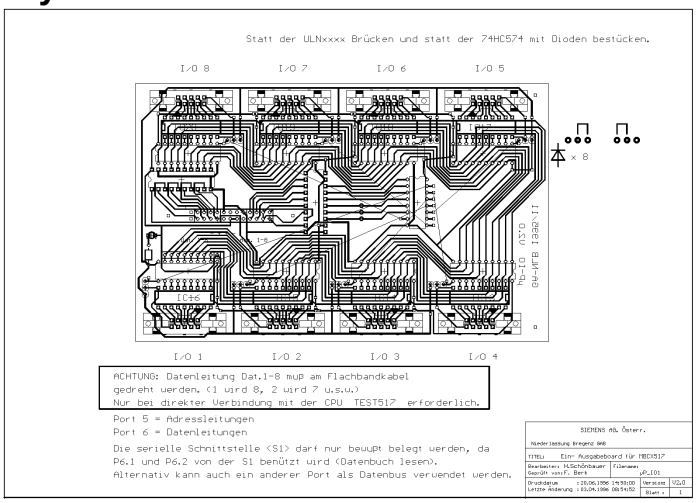
Ein- Ausgabeerweiterung

Sie ermöglicht die Ansteuerung bzw. Abfrage von bis zu 192 I/Os. Hierbei wird ein Port (8-Bit) für die Bausteinadressierung und ein zweites Board für die Datenvermittlung verwendet. Zur Adressierung werden 3 Bit pro Baugruppe (2'0 - 2'2) und weitere 3 Bit (2'3-2'5) zur Platinenselektion benötigt. 2'6 ist der Übernahmetakt für den gerade adressierten Baustein und 2'7 bestimmt die Datenrichtung (Lesen oder Schreiben).

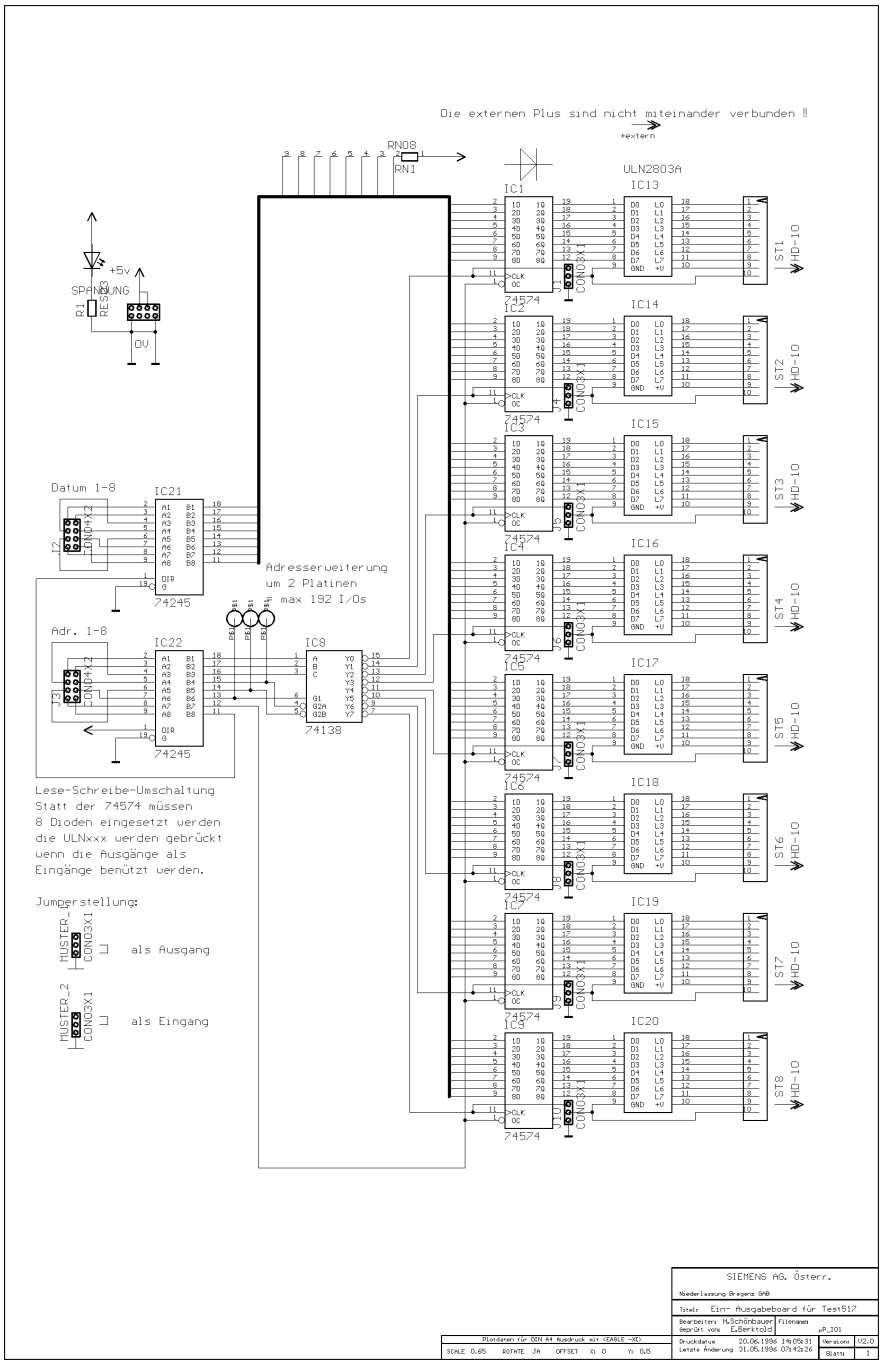
Eine höhere Dekodierung der Baugruppen wäre möglich (7x8 statt 3x8), aber nur bei sehr langsamen Applikationen sinnvoll (>100ms) und ist bei dieser Hardware nicht realisiert.

Alle Logikschaltungen in HC- oder HCT-Ausführung.

Layout



Schaltbild



Wie mache ich meine Eingänge störungsunempfindlich?

Zur Philosophie:

Die Eingänge sollen möglichst oft abgefragt werden, um eine vernünftige Entprellung (Ausblenden von Störeinflüssen) zu gewährleisten.

Die Ausgänge sollen innerhalb von 10ms bedient werden.

Zur Entprellung:

1.) Eine Methode (hier nicht verwendet) ist, den Zustand 1 oder 0 als Addier- oder Subtrahierkriterium herzunehmen und die Auswertung über ein Trendbit und Grenzwerte vorzunehmen. Diese Art benötigt eine relativ hohe Abfragerate und ist recht aufwendig.

Sie ist nur für Extremfälle sinnvoll, da ein Abtastabstand von etwa 1ms erforderlich ist. Dafür ist sie FUZZY-ähnlich und kann mit entsprechenden Schwellwerten ausgestattet werden.

2.) Eine andere Methode, die hier zur Anwendung kommt, ist:

Man pollt die Eingänge ca. alle 10ms, wobei das Lesen (Portzugriff) nur eine Übernahmezeit von 1µs benötigt und akzeptiert den Wert nach etwa 8 gleichen Ergebnissen pro Bit.

Nach 8 x 10ms = 80ms wird dieses Ergebnis einer Auswerteroutine übergeben (ein für die meisten Eingänge annehmbarer Wert).

Allgemeines über Störungen innerhalb von Geräten oder Maschinen

Eine Störung kann durch Lichtbögen oder durch Wechselmagnetfelder entstehen. Den Einfluß von Funkstörungen schließe ich bei dementsprechendem Aufbau und Außenbeschaltung aus. Ein Lichtbogen ist außer mit der Methode 1 kaum auszuschließen (ähnlich Funkstöreinflüssen). Ein Wechselmagnetfeld induziert hingegen seine Störung relativ langsam 10ms in eine und 10ms in die andere Richtung (bei 50Hz), wobei das Nutzsignal einmal verstärkt und einmal geschwächt wird. Verbleiben also 10ms in denen sich die Störung auswirkt. Wenn man die Abtastrate asynchron zu den 50Hz setzt wird die Wahrscheinlichkeit einer Störung stark reduziert. Es sollten 8 Abfragen nach jeweils 10ms für eine sichere Eingangsbestimmung reichen.

Bei Anwendung der Methode 2 steht das Eingangsergebnis also nach 80ms fest.

Tritt eine Störung innerhalb dieser Zeit auf, so wird der Zyklus von 8 Abfragen wiederholt. Es tritt eine dementsprechende Verzögerung ein. Diese Verzögerung ist in den meisten Fällen tragbar. Wenn man noch bedenkt, daß die Abfragezeit jeweils nur 1µs beträgt und der Abfragezyklus 10ms, so entsteht ein Verhältnis von 1:10.000 und die Wahrscheinlichkeit einer Störung wird dementsprechend gering. Man kann auch pro Störung in der Abfragephase den Entprellwert inkrementieren und so eine schnellere Gültigkeit erreichen.

Es gibt natürlich Anwendungen wo dies nicht möglich ist und eine andere Lösung gesucht werden muß (Hardwareaufwand). Nach meiner langjährigen Erfahrung sind solche Fälle ausgesprochen selten und wenn, dann vorhersehbar.

Softwaremodul-Beschreibung

Datenfelder:

`def_feld[max_port]`

Bestimmt die Verwendung als Eingang, Ausgang oder nicht benützt.

1=Ausgang

0=Eingang

2=NICHT

`entprell[max_port][8]`

Vorgabe der Abfrageanzahl pro Eingang (Bit) - Entprellzahl

`entp_count[max_port][8]`

Temporäre Abfrageanzahl wird aus `entprell[][]` bei Eingangsänderung nachgeladen und, wenn größer `NULL`, bei Gleichheit dekrementiert. Zur Quittierung wird eine `80H` in dieses Byte geschrieben, um eine mehrfach Auswertung zu verhindern. Mit einer Zahl unter 80H kann eine "REPEAT-Funktion" erreicht werden.

`ein_aus[max_port]`

Speicher der Portzustände, daß heißt die Ausgänge werden in 'ein_aus' geschrieben und von dort an die Ports per Timerinterrupt weitergegeben. In der Leserichtung steht immer der letzt gelesene Eingangspegel in 'ein_aus'. Aber erst nach einer Abfrage von `'entp_count[][]'` auf `"IST=NULL"`, ist diese Information auch gültig.

Um sich Schreibaufwand zu ersparen und die Lesbarkeit eines Programmes zu erhöhen, ist es von Vorteil, Abfrage- ,Quittierungs- und Repetitions-Makros zu schreiben. Diese kann man dann in einen eigenen Header (`xxx.h`) unterbringen.

z.B.:

```
// Taste 1 Abfragen auf LOW-aktiv
#define   t1_akt   (entp_count[x][y]&0x7f==0)&&(!ein_aus&z)
```

Im Programm sieht dies so aus:

```
   'if(t1_akt) {....;}'
       x ... Baustein  (0-7)
       y ... Bit       (0-7)
       z ... x*8+y
```

Programmodule:

`I_O()`	Selektion der Portrichtung (Schreiben/Lesen/Unbenutzt)
`ausg()`	Ausgabe des `'ein_aus[]'` Feldes auf den Port
`eing()`	Einlesen des Ports in das Feld `'ein_aus[]'`, Entprellen der einzelnen Eingänge eines Ports
`init_mpio()`	Löschen der Ausgänge im `'ein_aus[]'`-Feld und deren Ausgabe; Vorbesetzen der Entprellbytes

Alle weiteren Module dienen der Kontrolle der Ein- und Ausgänge und sind je nach Anwendung zu gestalten.

Programmaufbau in diesem Beispiel:

```
main()
{
  init_mpio()  // Ausgänge passiv setzen
  init_int0()  // EXTERN Interrupt 0 bei Bedarf (Ausstieg aus
               // dem KEIL-Monitor
  b9600()      // Serielle 0 mit 9600Baud initialisieren
  init_sp()    // V24-Spooler löschen und initialisieren
  init_t()     // Timer 2 mit 10ms autoreload setzen
               // und Interrupt vorbereiten
  forever      // Ab hier nur zur Demonstration
  disp_ios()   // Anzeige der Ein- und Ausgänge über V24 (S0)
  V24_ausw()   // Auswertung der seriellen Schnittstelle 0
               // wenn Datensatz vollständig
}

t2_int()       // Aufruf alle 10ms per Timer2-Interrupt
{
  v24_abf()    // Pollen der S0-Schnittstelle
               // und deren Bewertung
  I_O()        // Bedienung der I/Os je nach Verwendung
  {
    ausg()     // Ports alle 10ms beschreiben, unabhängig
               // von einer Änderung. So werden etwaige
               // Störungen der Ausgänge unterbunden.
    eing()     // Ports lesen und entprellen.
  }
}
```

Die Unterlagen zum Nachbau der hier verwendeten CPU-, I/O-Platine und eine Demosoftware kann über 'hermann.schoenbauer@siemens.at' bezogen werden

Die Hardware ist mit ‘EAGLE 3.02’, die Software mit ‘µVISION/51 für Windows’ Version 1.12 und C51-Compiler V5.0 von ‘KEIL’ erstellt worden.

Unter der Telefonnummer '05574 1707 299' stehe ich gerne für weitere Fragen zur Verfügung. □

```
/*****************************************************************
*                   Ein- Ausgabe-Programm                        *
*  fuer Platine "MPIO" ab Version V2.0 der GA-Bregenz            *
*  Benuetzte Entprellroutine (c) by H. Schoenbauer 1986          *
********************************************** 31-03-1996 *********/

#include        <reg517a.h>
#include        <stdio.h>
#include        <ctype.h>
              // 3 Platinen x 8 Bausteine x 8 Pin = 192 Pin
              // 3 Platinen x 8 Bausteine            =  24 Pin
#define         max_port        24
                   // I/O-Port
#define         daten_p      P6
              // Select und Clock
#define         adr_p        P5
              // verwendete Registerbank fuer Timer-Interrupt
#define         reg_bank     2
              // Eingangsrepetition z.B.:
              // fuer Tasteneingaenge
#define         repeat          8
#define         reload          -10000    //10ms TimerInterrupt
/* Eine praktikable Entprellzeit betraegt 80ms, dem entsprechend
   muss 'MPIO' per Timerinterrupt aufgerufen werden.
   Die Ausgaenge sind in einem Daten-Array jederzeit setz- und
   ruecksetzbar. Die Bedienung erfolgt per Timerinterrupt.
   Es werden max 1x8 Ein- oder Ausgaenge pro Interrupt bedient,
   um das Hauptprogramm nicht zu sehr zu bremsen.

Aufbau der Portadressen (adr_p):

b^0-b^2         Bausteinauswahl 0-7
b^3-b^5         Platinenauswahl 0-2       0 0 1   1.Platine
                                          0 1 0   2.Platine
                                          1 0 0   3.Platine
b^6             Ausgabetakt/Einlesetakt         LOW-aktiv
b^7             Schreibe/Leseumschaltung fuer 74HC245
                        gesteuert ueber 'def_feld'
                                        1       Schreiben
                                        0       Lesen
def_feld (Definitionsfeld)
        Bestimmung der Erweiterungsports als Ein- bzw. Ausgaenge
                                        0 = Eingang
                                        1 = Ausgang
                                        2 = nicht verwendet oder vorhanden
max_port (max. I/O-Portanzahl (Port=8bit))   1-24
*/
//
// Timerinitialisierung, T2 mit 'reload'-Wert
#include        "T2.c"
// V24 Schnittstelle 'S0' mit 9600 Baud initialisieren
#include       "b9600.c"
/*********************************************************
*                       V 24 - H A N D L E R             *
*                                                        *
**********************************************************
*/

#define               max_s_len       10
        // Eingangs-Spooler
xdata unsigned char     feld[max_s_len];
        // Ein- Ausgabezeiger
xdata unsigned char     *ein_z;
xdata unsigned char     *aus_z;
xdata unsigned char     diff;       // Differenzzaehler fuer V24-Empfang
idata unsigned char     text[9];
idata unsigned char     temp[4];

        // Eingangsspooler loeschen, Zeiger setzen
void init_sp(void)
{
  for(diff=0;diff<max_s_len;diff++)
          { feld[diff]=0; }
  ein_z=&feld[0];
  aus_z=&feld[0];
  diff=0;
}

void v24_abf(void)         using reg_bank
{
  if(RI0)   {
              *ein_z=S0BUF;
              if((*ein_z>0x2f) && (*ein_z<0x3a) || (*ein_z=='+') || (*ein_z=='-'))
                {
                diff++;
                S0BUF=*ein_z; TI0=0;          // Echo
                if(++ein_z>=&feld[max_s_len])     ein_z=&feld[0];
                }
              RI0=0;
              }
}

code unsigned char def_feld[max_port] = {
                         0,0,2,2,1,1,1,1,  // 2xEingang,2xnicht,4xAusgang..
                         2,2,2,2,2,2,2,2,  // nicht verwendet.............
                         2,2,2,2,2,2,2,2   // ............................
                                        };
   // 'entprell' = Anzahl der Einlesezyklen pro Eingang, NICHT pro Port
   // normal 80ms. Wenn alle 10ms eine Portbehandlung erfolgt, ergeben
   // sich 8 x 10ms = 80ms. Damit besteht die Moeglichkeit stoeranfaellige
   // Eingaenge hoeher zu entprellen.
   // Bereich 1-127, 128 (0x80) wird zur Quittierung beim Abarbeiten
   // verwendet.
code unsigned char entprell[max_port][8] = {
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8,
                                8,8,8,8,8,8,8,8, 8,8,8,8,8,8,8,8
                                                 };

// Gegenstueck von 'entprell' im RAM, wird beim Generieren mit dem
// Wert aus 'entprell' beschrieben und bei jeder Eingangsaenderung
// nachgeladen.
```

```
xdata unsigned char entp_count[max_port][8];

//Shadow-Ports, Inhalt entspricht dem momentanen I/O-Zustand. Je nach Verwendung des
Erweiterungsportes befindet sich //der Eingangszustand IST unabhaengig vom Entprellstatus
im Speicher 'ein_aus[x]'.
//Oder der Ausgangszustand wird in 'ein_aus[x]' geschrieben und beim entsprechenden
Timerinterrupt ausgegeben.

xdata unsigned char ein_aus[max_port];

// Zaehler fuer Ein-/Ausgabe beim Timerinterrupt
idata unsigned char     zaehler;

// Fuer Aenderungskennung eines Einganges
bit                           ungleich;

void ausg(void)    using reg_bank
{
  idata unsigned char   n, x;
  n=zaehler&0x07;      // Bausteinauswahl:     Maske fuer Baustein
                // Baugruppenauswahl:   Maske fuer Baugruppe
  if(zaehler<8)            n|=0x20;  // 1. Baugruppe
  else if(zaehler<16)   n|=0x10;   // 2. Baugruppe
  else                       n|=0x08;  // 3. Baugruppe
      n|=0x80;             // DIR auf Ausgabe, Ausgabedatum holen
  x=ein_aus[zaehler];
  n|=0x40;        // OC ON       OC==H     Daten ausgeben
  adr_p=n;    // OC OFF      OC==L
  daten_p=x;
      n&=0xbf;        // OC OFF      OC==L     und Trennen
  adr_p=n;
}

void eing()       using reg_bank     // Selbe Registerbank wie T2-Interrupt
{
  idata unsigned char   n,mex,x;
  adr_p=0xbf;
  adr_p=0x3f;               // OC==L , DIR auf Eingabe
  x=zaehler&0x18;           // Baugruppenauswahlmaske
  n=zaehler&0x07;    // Bausteinauswahl
                            // die Pin-Adressierung ist ohne Bedeutung,
                            // da alle 8 Bit
                            // gemeinsam eingelesen werden.
                            // Baugruppe bestimmen
  if(x<8)        n|=0x20;      // 1. Baugruppe
  else if(x<16)     n|=0x10;      // 2. Baugruppe
  else               n|=0x08;      // 3. Baugruppe
  adr_p=n;                           // Adresse und CLK
      daten_p=0xff;               // Port auf Eingabe
      mex=daten_p;              // Port lesen
  adr_p=0x3f;                     // Baugruppe deselektieren
      //     Entprellvorgang Anfang
  x=mex;                                  // Neu-Daten merken
  mex^=ein_aus[zaehler];          // Unterschied zum letzten Lesen -> mex
                                             // 'mex' bedeutet Merker EXOR Alt<->Neu
  ein_aus[zaehler]=x;              // Neuen Wert merken
  x=0x01;                                 // Bit-Maske
  for(n=0;n<8;n++)                   // Bit-Zaehler
          {
        if(mex&x)                                       // Eingang veraendert?
           entp_count[zaehler][n]=entprell[zaehler][n];    // Counter nachladen
        else if(entp_count[zaehler][n] & 0x7F)         // Eingang gleich aber nicht
entprellt
          {entp_count[zaehler][n]--;                     // Entprellcounter -1
            if(entp_count[zaehler][n]==0)    ungleich=1;
           }                   //      else    ; Eingang ist gleich und schon entprellt -
nichts tun
        x=x<<1;                                // naechstes Bit, - Masken-SHIFT
        }  }  //          Entprellvorgang Ende
/* Bei einer einzelnen Abfrage der Ports empfehle ich eine  switch(timer_zaehler) {
case(0).......case(x)  } Bedienung
Der Vorteil, bestimmte Programmteile koennen so in regelmaessigen kurzen Abstaenden bedient
werden, waehrend die I/Os zyklisch in groesseren Schritten ausgefuehrt werden. Dadurch
bleibt fuer das Hauptprogramm mehr Ausfuehrungszeit. */


void I_O(void)  using reg_bank
{
  zaehler=0;
  while(zaehler<max_port)
  {
                                      // Portverwendung selektieren
  if(def_feld[zaehler]==1)           ausg();
  else if(def_feld[zaehler]==0)       eing();
  zaehler++;
  }
}

/*******************************************************
        Initialisierung besetzt die Entprellbytes
          und setzt die Ausgaenge auf HIGH-Pegel
********************************************************/
void init_mpio(void)
{
  idata unsigned char   j;
  for(zaehler=0;zaehler<max_port;zaehler++)
          {
          if(def_feld[zaehler]==1)
           {
                  ein_aus[zaehler]=0x00;       // Ausgang loeschen
                  ausg();                          // und ausgeben
            }
          else if(def_feld[zaehler]==0)
            {for(j=0;j<8;j++)        // Entprellcounter vorbesetzen
                  entp_count[zaehler][j]=entprell[zaehler][j];
            }
          }
}

init_int0(void)
{
  IT0=1;              // Initialisieren von Interrupt EX0
  EX0=1;              // Flankengesteuert
  IE0=0;              // Fuer Monitorabbruch bei Programmlauf im CMOS-RAM
}

t2_int(void)    interrupt 5            using reg_bank
{
  v24_abf();
  I_O();
  TF2=0;
}
```

```
// Bitmuster der Ausgaenge und Eingaenge darstellen

void disp_ios(void)       //      Darstellmodul der I/Os            ANFANG
{
  idata unsigned char     i,j,x1;
  for(i=0;i<max_port;i++)
         {
         if(def_feld[i]==1)      // Ausgabebaustein
           {
            j=0;
             for(x1=0x80;x1!=0x08;x1=x1>>1)
                   { if(ein_aus[i]&x1)   { text[j]='1'; }
                      else                { text[j]='0'; }
                     j++;
       }
            text[j]='|'; j++;
            for(;x1>0x00;x1=x1>>1)
                  { if(ein_aus[i]&x1)   { text[j]='1'; }
                    else                  { text[j]='0'; }
                    j++;
         }
           printf("%c%c%c%c%c%c%c%c ",text[0],text[1],text[2],text[3],
                                      text[4],text[5],text[6],text[7],
                                      text[8]);
           }


    else if(def_feld[i]==0)// Eingabebaustein
           {
            j=0;
             for(x1=0x80;x1!=0x08;x1=x1>>1)
                   { if(ein_aus[i]&x1)   { text[j]='H'; }
                     else                  { text[j]='L'; }
                     j++;
       }
            text[j]='-'; j++;
            for(;x1>0x00;x1=x1>>1)
                  { if(ein_aus[i]&x1)   { text[j]='H'; }
                    else                  { text[j]='L'; }
                    j++;  }
           printf("%c%c%c%c%c%c%c%c ",text[0],text[1],text[2],text[3],
                                      text[4],text[5],text[6],text[7],
                                      text[8]);
           }

         else                  // Nicht benuetzt
           {
                 printf("----|---- ");
           }
       }
  printf("\n");
}       //       Darstellmodul der I/Os            ENDE

v24_ausw()      //  V24_Auswertung       ANFANG
{
  idata unsigned char     i, x1;
       temp[0]=*aus_z;
       diff--;
       temp[0]&=0x31;
       if(++aus_z >= &feld[max_s_len]) { aus_z=&feld[0]; }
       temp[1]=*aus_z;
       diff--;
       if(++aus_z >= &feld[max_s_len]) { aus_z=&feld[0]; }
       temp[2]=*aus_z;
       printf("\tAusgang %c%c%c ",temp[0],temp[1],temp[2]);
       diff--;
       if(++aus_z >= &feld[max_s_len]) { aus_z=&feld[0]; }
       temp[3]=*aus_z;
       if(*aus_z=='+')
       { printf(" ON\n");}
       else
       { printf(" OFF\n");}
       diff--;
       if(++aus_z >= &feld[max_s_len]) { aus_z=&feld[0]; }
  temp[0]=(temp[0]&0x01)*100;
  temp[0]=(temp[1]&0x0f)*10;
  temp[0]+=temp[2]&0x0f;
  temp[2]=temp[0]/8;            // Baustein bestimmen
  x1=temp[0]%8;                 // Bit-Position
  if(temp[2]<24)                // Baustein-Max begrenzen
  {
  if(def_feld[temp[2]]==1)      // Dann ist es eine Ausgabe
         {
    i=0x01; i=i<<x1;            // Bitposition einstellen
    if(temp[3]=='+')            // Ausgang ON
          {
          ein_aus[temp[2]] | =i;
          }
    else                          // Ausgang OFF, es ist eine Eingabe
          {
          i^=0xff;
          ein_aus[temp[2]] & =i;
          }
      }
  else
         { printf("\t\t\t\t U N G U E L T I G\n"); }
    printf("\n");
       }
  disp_ios();
  printf("Ihre Eingabe:  ");
}       //      V24_Auswertung          ENDE


void main(void)
{
  init_mpio();
  init_int0();
  b9600();
  RI0=0;
  init_sp();
  init_t();
  zaehler=0;
  printf("%c\n\t\t\tI/O-Karte mit 192 Ein- Ausgaengen",temp[0]);
  printf("\n\t\t\tgesteuert von MP-Karte 'TEST517'\n");
  printf("\n\t\tAusgang  ON   mit Ausgangnummer 0-191 und + , z.B.: 001+\n");
  printf("\t\tAusgang  OFF  mit Ausgangnummer 0-191 und - , z.B.: 001-\n\n");
  printf("Darstellung der Eingaenge mit H/L, der Ausgaenge mit 1/0\n\n");
  printf("Ihre Eingabe:  ");

  while(1) {
      P52=0;
      if(ungleich)
             {
              ungleich=0;
              printf("\n\n");
              disp_ios();
             }
      if(diff>=4)           // xxx = Ausgang   y=+  == ON  y!=+  == OFF
             {
             v24_ausw();
       }
      }
}
□
```

Real programmers don't.....

- *Real programmers don't write specs. Users should consider themselves lucky to get any programs at all and take what they get.*
- *Real programmers don't comment their code. If it was hard to write, it should be hard to read.*
- *Real programmers don't write application programs, they program right down on the bare metal. Application programming is for feebs who can't do systems programming.*
- *Real programmers don't eat quiche. Real programmers don't even know how to spell quiche. They eat Twinkies, Coke and palate-scorching Szechwan food.*
- *Real programmers don't draw flowcharts. Flowcharts are, after all, the illiterate's form of documentation. Cavemen drew flowcharts; look how much it did for them.*
- *Real programmers don't read manuals. Reliance on a reference is a hallmark of the novice and the coward.*
- *Real programmers programs never work right the first time. But if you throw them on the machine they can be patched into working in only a few 30-hours debugging sessions.*
- *Real programmers don't use Fortran. Fortran is for wimpy engineers who wear white socks, pipe stress freaks, and crystallography weenies. They get excited over finite state analysis and nuclear reactor simulation.*
- *Real programmers don't use COBOL. COBOL is for wimpy application programmers.*
- *Real programmers never work 9 to 5. If any real programmers are around at 9 am, it's because they were up all night.*
- *Real programmers don't write in BASIC. Actually, no programmers write in BASIC, after the age of 12.*
- *Real programmers don't document. Documentation is for simps who can't read the listings or the object deck.*
- *Real programmers don't write in Pascal, or Bliss, or Ada, or any of those pinko computer science languages. Strong typing is for people with weak memories.*
- *Real programmers know better than the users what they need.*
- *Real programmers think structured programming is a communist plot.*
- *Real programmers don't use schedules. Schedules are for manager's toadies. Real programmers like to keep their manager in suspense.*
- *Real programmers think better when playing adventure.*
- *Real programmers don't use PL/I. PL/I is for insecure momma's boys who can't choose between COBOL and Fortran.*
- *Real programmers don't use APL, unless the whole program can be written on one line.*
- *Real programmers don't use LISP. Only effeminate programmers use more parentheses than actual code.*
- *Real programmers disdain structured programming. Structured programming is for compulsive, prematurely toilet-trained neurotics who wear neckties and carefully line up sharpened pencils on an otherwise uncluttered desk.*
- *Real programmers don't like the team programming concept. Unless, of course, they are the Chief Programmer.*
- *Real programmers have no use for managers. Managers are a necessary evil. Managers are for dealing with personnel bozos, bean counters, senior planners and other mental defectives.*
- *Real programmers scorn floating point arithmetic. The decimal point was invented for pansy bedwetters who are unable to "think big."*
- *Real programmers don't drive clapped-out Mavericks. They prefer BMWs, Lincolns or pick-up trucks with floor shifts. Fast motorcycles are highly regarded.*
- *Real programmers don't believe in schedules. Planners make up schedules. Managers "firm up" schedules. Frightened coders strive to meet schedules. Real programmers ignore schedules.*
- *Real programmers like vending machine popcorn. Coders pop it in the microwave oven. Real programmers use the heat given off by the cpu. They can tell what job is running just by listening to the rate of popping.*
- *Real programmers know every nuance of every instruction and use them all in every real program. Puppy architects won't allow execute instructions to address another execute as the target instruction. Real programmers despise such petty restrictions.*
- *Real programmers don't bring brown bag lunches to work. If the vending machine sells it, they eat it. If the vending machine doesn't sell it, they don't eat it. Vending machines don't sell quiche.*
- *Real programmers know that the word is disk, not disc. Disc is a definite commie plot put forth by blubbering quiche eaters.*