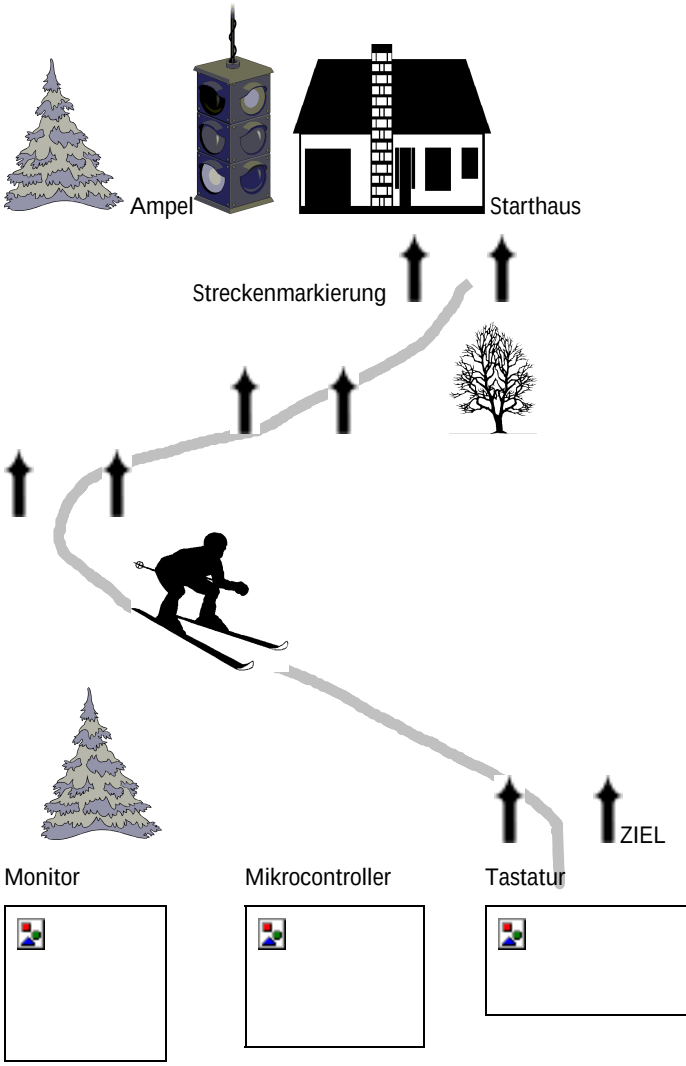
Informatik + Mikroelektronik = binäre Bäume und Mikrocontroller

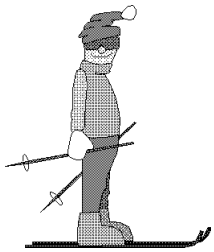
... eine sportliche Applikation rund um das Schirennen

Wilhelm Brezovits PCN-DSK-520:\baeume und DSK-521..526, 533..534

Applikation Schirennlauf



Kurzbeschreibung



- Die Ampel steht auf rot (Piste für den Schirennläufer gesperrt).
- Der Name des Schirennläufers wird erfaßt.
- Die Ampel wechselt auf grün (Piste frei).
- Der Schiläufer startet, die Zeitmessung beginnt.
- Die Ampel schaltet auf rot (Piste für den nächsten Läufer gesperrt).
- Der Schiläufer erreicht das Ziel, die Zeitmessung stoppt.
- Der Name und die Zeit des Schiläufers wird sortiert (1. nach der Zeit, bei gleicher Zeit alphabetisch nach Name) in einem binären Baum gespeichert.
- Ein Zwischenergebnis wird ausgegeben (1. Platz: Frau C, 2. Platz Herr B, 3. Platz Frau X, u.s.w.).
- Der Name des nächsten Schirennläufers wird erfaßt.
- Die Ampel wechselt auf grün (Piste frei).
- u.s.w.

Mittels Name nächster Schirennläufer = * wird dem Programm signalisiert, daß alle Läufer gestartet sind.

Es erfolgt der Ausdruck des Endergebnisses.

Anmerkung:

- Der binäre Baum hat den Vorteil, daß bei Programmstart die Anzahl der Schirennläufer nicht feststehen muß.
- Es wird nur bei Bedarf Speicherplatz angefordert!
- Weiters stehen die Daten jederzeit sortiert zur Verfügung!

Hinweis_1:

- *Demnächst ist die objektorientierte Version des C166-C-Compilers (C++) verfügbar.*
- *Diese Applikation ist extra für die PCNEWS edit geschrieben.*
- *Der binäre Baum in diesem Beispiel beschränkt sich auf das „sortierte Einfügen“ und „Ausgeben“, außerdem ist er „nur“ in ANSI-C codiert.*
- *Wenn sich mindestens eine Person bei der Redaktion meldet und Interesse bekundet (per Internet ?) bin ich gerne bereit, das ganze Beispiel zu erweitern (Speicherung in einer Liste / einem Baum; mit den Funktionen (Methoden) ausgeben, sortiert eintragen und löschen); natürlich in C++ codiert!*

Hinweis_2:

- *Um mit der Beschreibung nicht den zur Verfügung gestellten Platz in der PCNEWS edit zu sprengen, wurden sämtliche Dateien mit sehr viel Kommentar versehen (selbsterklärend!).*
- *Weiters wurden aussagekräftige Namen (Variablen, Funktionen) verwendet.*
- *Das Studium nachfolgender Dateien müßte also ausreichen, den Inhalt zu verstehen und nachvollziehen zu können.*

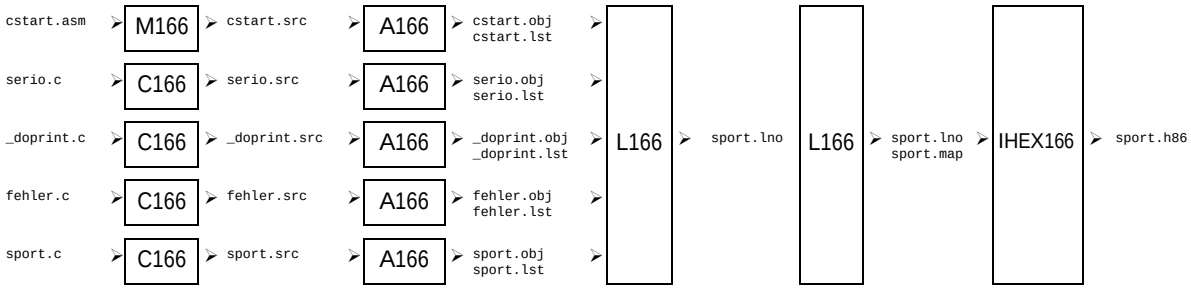
Hinweis_3:

- *Um unsere Applikation so kurz und überschaubar wie möglich zu halten (aber funktionsfähig), werden Benutzereingabefehler programmtechnisch nicht abgefangen.*
- *Weiters erfolgt die Zeitmessung in Sekunden (obwohl diese 16 bit Mikrocontrollerfamilie eine Auflösung von !!! 40 ns !!! für externe Signale bei nur 25 MHz Taktfrequenz schafft.*

REALISIERUNG DER APPLIKATION

1. Schritt

Ein Schaubild der verwendeten Dateien wird entworfen. Die notwendigen Compiler, Makroassembler, Linker und Locater- Aufrufe werden eingetragen.



C166	C-Compiler
M166	Makroassembler
A166	Assembler
L166	Linker/Locater
IHEX166	Hex-Konverter

`cstart.asm` Vom Compilerhersteller mitgelieferte "Startup-Datei". Der Inhalt dieser Datei wird vor " void main (void) { } " abgearbeitet. cstart.asm initialisiert den Mikrocontroller wie zum Beispiel: externer Bus: gemultiplext, nicht gemultiplext, Waitstates, ... Stackgröße des Mikrocontrollers; Watchdog-Timer u.s.w.

`serio.c` Vom Compilerhersteller mitgelieferte Datei zur Unterstützung der asynchronen, seriellen Schnittstelle des Mikrocontrollers damit Funktionen wie "printf" oder "scanf" nicht ins Leere schreiben oder von der binären Wüste lesen.

`_doprint.c` Vom Compilerhersteller mitgelieferte Datei. Zur Verbesserung des Laufzeitverhaltens stehen 3 Versionen von "printf" zur Verfügung.

`fehler.c` Bestimmte Fehler können während der Programmabarbeitung im Mikrocontroller auftreten (wie zum Beispiel: Stack-Überlauf, oder Stack-Unterlauf, auch das Bitmuster eines geschützten Befehles (Idle, Powerdown,...) kann vom externen Bus falsch eingelesen werden). Tritt so ein Fehler auf, springt der Mikrocontroller auf jedem Fall auf die dafür vorgesehene Interrupteinsprungadresse. Deshalb sollte auf dieser Interrupteinsprungadresse ein sinnvoller Befehl stehen (in unserem Beispiel ein Software-Reset).

`sport.c` Unsere Applikation (Schirennlauf) formuliert in ANSI-C mit einigen Erweiterungen die bei Mikrocontrollern notwendig sind.

Laden von sport.h86			ERTEC-MONITOR
Bootstrap-Loader aktivieren		➢	Stellung „BTL ON“
		➢	RESET-Taste
Mon16x	RETURN	➢	Ertec-Monitor wird geladen
load „sport.h86“	RETURN	➢	Anwenderprogramm wird geladen
exit	RETURN	➢	Monitor wird verlassen
Bootstrap-Loader deaktivieren		➢	S2
	RESET	➢	Anwendung läuft

2. Schritt:

Die Dateien `cstart.asm`, `serio.c` und `_doprint.c` werden nach Hardware/Anforderung angepaßt.

Die Steuerdateien für den Linker und Locater L166 werden geschrieben:

Datei cmd_link

```
; datei: cmd_link
;
;  CMD_LINK
; - steuerdatei für Linker l166
;

; Der heap wird benoetigt, wenn Speicherplatz waehrend
; der Laufzeit mittels
; malloc angefordert wird.
; 30000 Bytes reichen fuer ca. 1300 Schilauefer
; genau: HEAPSIZE / sizeof(KNOTEN)
HEAPSIZE (30000)

; Liste der Objektdateien die gebunden werden:
sport.obj fehler.obj  doprint.obj serio.obj cstart.obj

; Pfadangabe der Bibliotheken die mitgebunden werden:
LIBPATH(C:\bso_v5\lib\ext) C166S.LIB F166S.LIB

; gewuenschter Name der Output-Datei
TO sport.lno
```

Datei cmd_loc.e_e

```
; datei: cmd loc.e e
;
;  CMD LOC.E E
; - steuerdatei für locater l166 fuer ertec-board und ertec-monitor
;

;zu bearbeitende Datei:
sport.lno

;Interrupt-Einsprungtabelle wird erzeugt:
VECTAB(0)
;NOVECINIT

;Hinweis zur Hardware:
;Standard Speicherkapazität = 256 KByte
;(2x1MBit static RAM = 2x128KByte devices: 0H - 3FFFFH)

;diverse Code/Datensegmente werden bestimmten Adressen zugewiesen:
CLASSES ('CUSTACK' (  4000H-  0BFFFH))
CLASSES ('CPROGRAM'( 20000H-  3FFFFH))

;belegte Speicheradressen vom ertec_167_monitor
RESERVE MEMORY (   200H to    1FFFH )
;belegte Registerbank vom ertec_167_monitor
RESERVE MEMORY ( 0FCC0H to   0FCDFH )

;freies RAM, nicht bestückt:
RESERVE MEMORY ( 40000H to   7FFFFH )
;mit standard devices nicht erreichbare Speicheradressen:
RESERVE MEMORY ( 80000H to 0FFFFFFH )

;PEC-Pointer Reservierung:
RESERVE PECPTR ( PECC0 to PECC7 )
```

Die Datei `fehler.c` wird geschrieben:

Datei fehler.c

```
        /* datei: fehler.c                                          Seite 1/1
        ╔═══════════════════════════════════════════════════════════════════╗
        ┃ TRAP PRIORITY III, RESET FUNCTIONS:                               ┃
        ┃  » Hardware Reset,              Trap-Nummer: 00H, Vector: 00'0000 H┃
        ┃  » Software Reset,              Trap-Nummer: 00H, Vector: 00'0000 H┃
        ┃  » Watchdog Timer Overflow,     Trap-Nummer: 00H, Vector: 00'0000 H┃
        ╠═══════════════════════════════════════════════════════════════════╣
        ┃ TRAP PRIORITY II,  CLASS A HARDWARE TRAPS:                        ┃
        ┃  » Non-Maskable Interrupt       Trap-Nummer: 02H, Vector: 00'0008 H┃
        ┃  » Stack Overflow,              Trap-Nummer: 04H, Vector: 00'0010 H┃
        ┃  » Stack Underflow,             Trap-Nummer: 06H, Vector: 00'0018 H┃
        ╠═══════════════════════════════════════════════════════════════════╣
        ┃ TRAP PRIORITY I,   CLASS B HARDWARE TRAPS:                        ┃
        ┃  » Undefined Opcode,            Trap-Nummer: 0AH, Vector: 00'0028 H┃
        ┃  » Protected Instruction Fault, Trap-Nummer: 0AH, Vector: 00'0028 H┃
        ┃  » Illegal Word Operand Access, Trap-Nummer: 0AH, Vector: 00'0028 H┃
        ┃  » Illegal Instruction Access,  Trap-Nummer: 0AH, Vector: 00'0028 H┃
        ┃  » Illegal External Bus Access, Trap-Nummer: 0AH, Vector: 00'0028 H┃
        ╚═══════════════════════════════════════════════════════════════════╝*/

       interrupt ( 0x02 ) void laufzeit fehler trap a nmi   (void);
       interrupt ( 0x04 ) void laufzeit_fehler_trap_a_stkof (void);
       interrupt ( 0x06 ) void laufzeit fehler trap a stkuf (void);
       interrupt ( 0x0A ) void laufzeit_fehler_trap_b       (void);


    /*************************************************************************/
       interrupt ( 0x02 ) void laufzeit fehler trap a nmi (void)
       {
        _int166(0); // SRST...Software Reset
       } /* ende laufzeit fehler trap a nmi */

    /*************************************************************************/

    /*************************************************************************/
       interrupt ( 0x04 ) void laufzeit fehler trap a stkof (void)
       {
        int166(0); // SRST...Software Reset
       } /* ende laufzeit_fehler_trap_a_stkof */

    /*************************************************************************/


    /*************************************************************************/
       interrupt ( 0x06 ) void laufzeit fehler trap a stkuf (void)
       {
        _int166(0); // SRST...Software Reset
       } /* ende laufzeit fehler trap a stkuf */

    /*************************************************************************/
/
       interrupt ( 0x0A ) void laufzeit_fehler_trap_b (void)
       {
        _int166(0); // SRST...Software Reset
       } /* ende laufzeit_fehler_trap_b */

    /*************************************************************************/
```

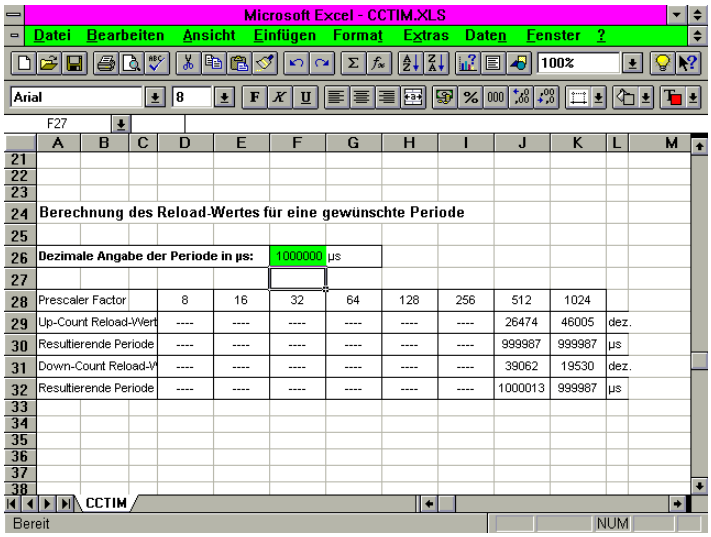
Die Datei `sport.c` unsere eigentliche Applikation wird geschrieben.

Datei sport.c

```
/*****************************************************************************
* Applikation Schirennlauf:                                                  *
*                                                                            *
*****************************************************************************
* Beschreibung:                                                              *
*                                                                            *
* ...die Ampel steht auf Rot (die Piste ist fuer Schirennlaeufer gesperrt).  *
* Die rote Lampe der Ampel wird mittels P7.0 vom Mikrocontroller angesteuert.*
* Die gruene Lampe der Ampel wird mittels P7.1 vom Mikrocontroller angesteuert.*
*                                                                            *
* Der Name des Schirennlaeufers wird erfasst.                                *
* Der externe Interrupt fuer Port 2.0 wird freigegeben.                      *
* Die Ampel wechselt auf Gruen (Piste frei).                                 *
*                                                                            *
* Der Schilaeufer startet, die Zeitmessung beginnt:                          *
* Mittels positiver Flanke an Port 2.0 wird ein externer Interrupt ausgeloest.*
* = START-SIGNAL                                                             *
* Dieser Interrupt startet Timer 0 und schaltet die Ampel auf Rot.           *
*                                                                            *
* Der Schilaeufer erreicht das Ziel, die Zeitmessung stoppt:                 *
* Mittels positiver Flanke an Port 2.1 wird ein externer Interrupt ausgeloest*
* = STOPP-SIGNAL                                                             *
* Dieser Interrupt stoppt Timer 0.                                           *
* Mittels globaler Variable schilaeufer_ziel_erreicht, wird dem Hauptprogramm*
* mitgeteilt, dass der Schilaeufer das Ziel erreicht hat und die Zeitmessung *
* abgeschlossen ist.                                                         *
*                                                                            *
* u.s.w. ...                                                                 *
*                                                                            *
*                                                                            *
*****************************************************************************/


#include <REG167.H>   /* Register Definitionen f r Mikrocontroller C167       */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "serio.h"

#define MAXNAME 15    /* maximale Anzahl von Zeichen-1 f r Schi-Laeufer-Namen  */

#define TRUE 1
#define FALSE 0

// Bitname fuer das Port-Richtungsbit AUSGANG_ROT definieren:
// AUSGANG_ROT = 0 bedeutet: Port 7.0 ist Eingang (hochohmiger Zustand)
// AUSGANG_ROT = 1 bedeutet: Port 7.0 ist Ausgang (PUSH-PULL)
// PUSH-PULL bedeutet: es wird aktiv 0 Volt oder 5 Volt ausgegeben
// (je nach Inhalt vom Portlatch P7.0)
sfrbit AUSGANG_ROT _atbit(DP7,0);

// Bitname fuer das Port-Richtungsbit AUSGANG_GRUEN definieren:
sfrbit AUSGANG_GRUEN _atbit(DP7,1);

// das bit ROT ist der Zustand der aktiv ( 0 oder 5 Volt) am Pin
// AUSGANG_ROT = P7.0 ausgegeben wird:
sfrbit ROT _atbit(P7,0);

// das bit GRUEN ist der Zustand der aktiv ( 0 oder 5 Volt) am Pin
// AUSGANG_GRUEN = P7.1 ausgegeben wird:
sfrbit GRUEN _atbit(P7,1);


// Definition eines Knoten vom binaeren Baum:
typedef struct knoten
{
    char name[MAXNAME];    /* Name des Schi-Laeufers         */
    int  zeit;             /* gelaufene Zeit in [s]          */
    struct knoten *links;  /* linker Nachkomme               */
    struct knoten *rechts; /* rechter Nachkomme              */
} KNOTEN;


/*==================== Prototypen ============================================*/
void main (void); /* Hauptprogramm = Interruptservice-Routine mit            */
          /* Interruptvektor (Einsprungadresse) auf Adresse 0         */
          /* Ereignis welches zum Aufruf f hrt: RESET                 */

void init_capcom(void);      /* Initialisierung der externen Interrupts       */
void init_timer_0(void);     /* Initialisierung von Timer 0                   */
void init_interrupt(void);   /* Initialisierung der Interrupt-Prioritaeten    */

KNOTEN *einordnen(KNOTEN *knot_zeig, char *name);
void drucke_binaeren_baum(KNOTEN *knot_zeig, int *nr);
KNOTEN *knoten_fuer_binaeren_baum_anlegen(void);

void externen_interrupt_fuer_start_freigeben(void);
/*============================================================================*/

/*==================== Prototypen Interruptservice-Routinen ==================*/
interrupt (0x10) void externer_interrupt_START (void);  /* Signal an Port 2.0 */
interrupt (0x11) void externer_interrupt_STOPP (void);  /* Signal an Port 2.1 */
interrupt (0x20) void timer_0_SEKUNDENINTERRUPT (void);
/*============================================================================*/

/*== globale Variablen zur Kommunikation: Hauptprogramm <-> Interrupts =======*/
char schilaeufer_ziel_erreicht;
int zeit_in_sekunden;
bit start_signal_freigabe;
bit stopp_signal_freigabe;
/*============================================================================*/


/*============================================================================*/
void main (void)
{
 KNOTEN *wurzel;       // Pointer der auf den binaeren Baum zeigt anlegen
 char wer[MAXNAME];    // Platz fuer den Namen des Schirennlaeufers schaffen
 int i=1, platz;

 ROT = TRUE;           // Ampelinitialisierung bei Programmstart
 GRUEN = FALSE;        // Ampelinitialisierung bei Programmstart
 AUSGANG_ROT = TRUE;   // Port 7.0 = AUSGANG (PUSH-PULL), P7.0 ist jetzt high
 AUSGANG_GRUEN = TRUE; // Port 7.1 = AUSGANG (PUSH-PULL), P7.0 ist jetzt low

 start_signal_freigabe = FALSE;
 stopp_signal_freigabe = FALSE;

 wurzel=NULL;          // Anfang des binaeren Baums festlegen

 init_serio();         // Initialisierung der seriellen Schnittstelle aufrufen
 init_capcom();        // Initialisierung der externen Interrupts
 init_timer_0();       // Initialisierung von Timer 0
 init_interrupt();     // Initialisierung der Interrupt-Prioritaeten

 IEN=1;                // Interruptsystem global freigeben

 while (1)
 {
    do
    {
         printf("\n\nName des am START stehenden Schilaeufers = ? ");
         scanf("%s",wer); fflush(stdin);

         if (wer[0]!='*')
         {
              schilaeufer_ziel_erreicht='n'; // n...nein
              zeit_in_sekunden=0;

              externen_interrupt_fuer_start_freigeben();

              printf("\n");
              printf("\n%5u Sekunden, ",zeit_in_sekunden);
              printf(" ( aktueller Zwischenstand Laeufer: ");
              printf("%15s )",wer);
              while (schilaeufer_ziel_erreicht == 'n')   // warten, bis die
              {                                          // Zeitmessung
                                        // beendet ist !!!
                   printf("\r%5u",zeit_in_sekunden);

              } /* ende while */
              printf("\n");

              wurzel=einordnen(wurzel,wer);
              printf("\n\nZwischenstand nach %d Laeufer(n)\n",i++);

         } /* ende if */

         else
              printf("\n\nEndergebnis:\n");

         printf("\n%12s %-15s %12s [Sekunden]\n","Platz","Name","Zeit");
         platz=1;
         drucke_binaeren_baum(wurzel,&platz);

    } while(wer[0]!='*');

    _int166(0); // SRST...Software Reset
              // Aus der Sicht eines PC-Programmierers eine ungewoehnliche
              // Art und Weise mittels malloc reservierten Speicher freizugeben
 } /* ende while (1) */

} /* ende main */
/*============================================================================*/

/*============================================================================*/
void init_capcom(void)
{
 CCM0 = 0x0011;   /* 0000 0000 0001 0001 B                                    */
           /* |||| |||| |||| 0001: CAPCOM1_Kanal_0                  */
           /* |||| |||| |||| ||01: pos. Flanke auf P2.0 = Signal START */
           /* |||| |||| |||| |0: capture mod = externer Interrupt      */
           /* |||| |||| |||| 0: zugeordneter Timer egal               */
           /* |||| |||| 0001: CAPCOM1_Kanal_1                        */
           /* |||| |||| ||01: positive Flanke auf P2.1 = Signal STOPP  */
```

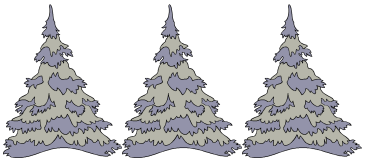
```
          /* |||| |||| |0: capture mod = externer Interrupt          */
          /* |||| |||| 0: zugeordneter Timer egal                    */
          /* |||| 0000: CAPCOM1_Kanal_2                              */
          /* 0000: CAPCOM1_Kanal_3                                   */

} /* ende init_capcom */
/*=========================================================================*/

/*=========================================================================*/
void init_timer_0(void)
{

 T01CON = 0x0006;  /* xxxx xxxx x0xx 0110 B                                */
           /*            |   |110: T0I:Timer/Counter 0 Input Selection*/
           /*            |   |          Teiler fuer fCPU = 512        */
           /*            |   0: T0M: Timer/Counter-Mode: 0=Timer Mode */
           /*           0: Timer/Counter 0 Run Control: Timer steht  */

 T0REL  = 26474;  /* Reload-Wert fuer Timer 0 Hochlauf fuer Periode=1 Sekunde */
          /* PT0 = ( (65536-T0REL) * 2 hoch(T0I+3) ) / fCPU =
          /*     = ( (65536-26474) * 2 hoch(6+3)   ) / 20 000 000 = 1 */
 T0=T0REL;

} /* ende init_timer */
/*=========================================================================*/

/*=========================================================================*/
void init_interrupt(void)
{
  // Interrupt Prioritaet Timer 0 (Sekundensignal): ILVL=10,GLVL=0
 T0IC  = 0x0068;   /* xxxx xxxx 01 1010 00 B                               */
           /*          || |||| 00: Group Level GLVL=0               */
           /*          || 1010: Interrupt Priority Level ILVL=10    */
           /*          |1: Interrupt Enable Control Bit setzen       */
           /*         0: Interrupt Request Flag loeschen            */

 // Interrupt Prioritaet Signal START: ILVL=14,GLVL=0
 CC0IC = 0x0078;   /* xxxx xxxx 01 1110 00 B, IR=0, IE=1, ILVL=14, GLVL=0    */

 // Interrupt Prioritaet Signal STOPP: ILVL=14,GLVL=1
 CC1IC = 0x0079;   /* xxxx xxxx 01 1110 01 B, IR=0, IE=1, ILVL=14, GLVL=1    */


} /* ende init_interrupt */
/*=========================================================================*/



/*=========================================================================*/
KNOTEN *einordnen(KNOTEN *knot_zeig, char *name)
{

 if (knot_zeig==NULL)
 {
     knot_zeig=knoten_fuer_binaeren_baum_anlegen();
     strcpy(knot_zeig->name,name);
     knot_zeig->zeit=zeit_in_sekunden;
     knot_zeig->links=knot_zeig->rechts=NULL;
 } /* ende if */

 else
     if (((zeit_in_sekunden-knot_zeig->zeit)<0) ||
     ((zeit_in_sekunden==knot_zeig->zeit) && (strcmp(name,knot_zeig->name)<0)))
          knot_zeig->links=einordnen(knot_zeig->links,name);
     else
          knot_zeig->rechts=einordnen(knot_zeig->rechts,name);

 return(knot_zeig);
} /* ende *einordnen */
/*=========================================================================*/

/*=========================================================================*/
void drucke_binaeren_baum(KNOTEN *knot_zeig, int *nr)
{
 if (knot_zeig!=NULL)
 {
     drucke_binaeren_baum(knot_zeig->links,nr);
     printf("%12d %-15s %12d\n",(*nr)++,knot_zeig->name,knot_zeig->zeit);
     drucke_binaeren_baum(knot_zeig->rechts,nr);
 } /* ende if */
} /* ende drucke_binaeren_baum */
/*=========================================================================*/

/*=========================================================================*/
KNOTEN *knoten_fuer_binaeren_baum_anlegen(void)
{
 KNOTEN *return_wert;

 return_wert=((KNOTEN *)malloc(sizeof(KNOTEN)));

 if (return_wert == NULL)
    printf("\n\n!!!ERROR: KEIN SPEICHERPLATZ VORHANDEN !!!"), _int166(0);

 return return_wert;

} /* ende KNOTEN *knoten_fuer_binaeren_baum_anlegen */
/*=========================================================================*/

/*=========================================================================*/
void externen_interrupt_fuer_start_freigeben(void)
{
 // Aufgabe: Start des Schilaufers vorbereiten: Ampel = Gruen

 start_signal_freigabe = TRUE;

 ROT   = FALSE;  /* Piste frei                                           */
 GRUEN = TRUE;   /* Piste frei                                           */

} /* ende externen_interrupt_fuer_start_freigeben */
/*=========================================================================*/

/*=========================================================================*/
interrupt (0x10) void externer_interrupt_START (void)
{
 // Aufgabe: Zeitmessung (Timer 0) starten und Ampel auf ROT schalten

 if (start_signal_freigabe)
 {
     T0R = 1; /* Timer 0 Run-Bit gesetzt -> Timer 0 laeuft (Zeitmessung laeuft)   */

     GRUEN = FALSE;  /* Piste fuer den naechsten Laeufer gesperrt                */
     ROT   = TRUE;   /* Piste fuer den naechsten Laeufer gesperrt                */

     start_signal_freigabe = FALSE;
     stopp_signal_freigabe = TRUE;
 } /* ende if */

} /* ende externer_interrupt_START */
/*=========================================================================*/

/*=========================================================================*/
interrupt (0x11) void externer_interrupt_STOPP (void)
{
 // Aufgabe: Timer_0 wird gestoppt.
 // mittels globaler Variable schilaeufer_ziel_erreicht wird dem Hauptprogramm
 // mitgeteilt, dass der Schilaeufer das Ziel erreicht hat, und die Zeitmessung
 // beendet ist.

 if (stopp_signal_freigabe)
 {
     T0R = 0; /* Timer 0 Run-bit geloescht -> Timer 0 steht (Zeitmessung beendet) */
     T0  = T0REL; /* Timer 0 Zaehlregister loeschen                                */

     stopp_signal_freigabe = FALSE;

     schilaeufer_ziel_erreicht = 'j'; /* j...ja...Ziel erreicht !!!               */
 } /* ende if */

} /* ende externer_interrupt_STOPP */
/*=========================================================================*/

/*=========================================================================*/
interrupt (0x20) void timer_0_SEKUNDENINTERRUPT (void)
{
 zeit_in_sekunden++;
} /* ende timer_0_SEKUNDENINTERRUPT */
/*=========================================================================*/
```

Bei der Berechnung des Sekundeninterrupts (Timer 0 Nachladewert) für die Zeitmessung ziehen wir die Excel-Dateien aus dem Buch „Arbeiten mit C166 Controllern von Karl-Heinz Mattheis“ zu rate:



!!! Alle Source-Dateien und Steuerdateien sind jetzt vorhanden !!!

Der nächste Schritt ist die Generierung der Hex-Datei sport.h86 mittels make, welche per Bootstrap-Loader und Monitor-Programm in unsere Zielhardware geladen wird:

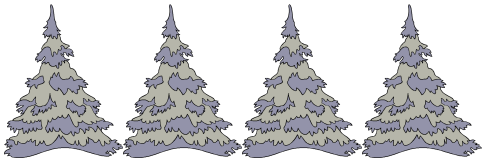


3. Schritt: Erstellung des makefile:

Datei makefile

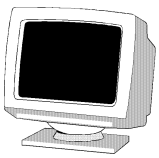
```
# datei: makefile
#
# ╔═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╗
# ║ MAKEFILE                                                                                                      ║
# ║                                                                                                               ║
# ║ Aufruf: mk166        // Generierung aller Dateien laut Toolkette                                               ║
# ║         mk166 clean  // Löschen aller erzeugten Dateien                                                       ║
# ╚═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╝
#
# ╔═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╗
# ║ Hinweis_1:                                                                                                    ║
# ║                                                                                                               ║
# ║ Wird das MAKEFILE mit "mk166 clean" von MS-DOS oder von MS-WINDOWS (Codewright) aufgerufen, dann werden alle unter dem Label
# ║ clean angegebenen Dateien gelöscht.                                                                           ║
# ╚═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╝
#
# ╔═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╗
# ║ Hinweis_2:                                                                                                    ║
# ║                                                                                                               ║
# ║ Wird das MAKEFILE mit "mk166" von MS-DOS oder von MS-WINDOWS (Codewright) aufgerufen, dann werden alle Dateien laut Toolkette
# ║ erzeugt.                                                                                                      ║
# ║ Bereits übersetzte und noch aktuelle Dateien werden nicht nochmals übersetzt -> Zeitersparnis !!!             ║
# ╚═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╝
#
# ╔═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╗
# ║ Hinweis_3:                                                                                                    ║
# ║                                                                                                               ║
# ║ Ein MAKEFILE liest man am besten von unten nach oben.                                                         ║
# ║ Wichtig für make ist die vom Betriebssystem für die zu bearbeitenden Dateien gesetzte Uhrzeit.                ║
# ║                                                                                                               ║
# ║ Es wird nur dann der Compiler, Assembler, Linker, Locater oder Objekt-Hex-Wandler aufgerufen, wenn die "Ergebnisdatei"
# ║ älteren Datums als die "Ursprungsdatei(en)" ist (sind) oder die "Ergebnisdatei" nicht vorhanden ist (Vorteil: Zeitersparnis).
# ║                                                                                                               ║
# ║ In den ersten beiden Zeilen des Makefile (Leserichtung von unten nach oben) steht:                            ║
# ║                                                                                                               ║
# ║ serio.src: serio.c serio.h                                                                                    ║
# ║  C166 -x -s -g -Ms serio.c                                                                                    ║
# ║                                                                                                               ║
# ║ Diese beiden Zeilen werden von mk166 folgendermaßen interpretiert:                                            ║
# ║ serio.src ist die "Ergebnisdatei" der "Ursprungsdateien" serio.c und serio.h                                  ║
# ║ Ist serio.src nicht vorhanden oder hat serio.src ein älteres Datum als serio.c oder serio.h dann wird serio.src (neu) erzeugt.
# ║ Wie das "Erzeugen" aussieht steht in der nächsten Zeile (mit TAB oder BLANK) eingerückt.                      ║
# ║ In unserem Beispiel ist es ein Compileraufruf C166.                                                           ║
# ║ -x bedeutet Mikrocontroller = C167                                                                            ║
# ║ -s bedeutet Assemblercode in der Protokolldatei anführen                                                      ║
# ║ -g erzeugt HLL-Debug-Information                                                                              ║
# ║ -Ms selektiert das verwendete Speichermodell (s...small)                                                      ║
# ╚═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╝



# OBJEKT-HEX-WANDLER:
# Verwandelt den OUTPUT vom Locater in ein Format, welches von EEPROM-Programmern oder in unserem Beispiel vom Monitor verstanden
# wird.
sport.h86: sport.out
 IHEX166 sport.out -i32 -o sport.h86

# LOCATER:
# Der Locater wandelt verschiebbare Programm/Datensegmente in ablauffaehige Segmente mit festen Adressen (er verteilt sie dabei
# auf den verfuegbaren Speicher.
sport.out: sport.lno cmd_loc.e_e
 L166 LOCATE @cmd_loc.e_e TO sport.out

# LINKER:
# Der Linker fuegt nach gewissen Regeln Programm-  und Daten-Segmente zusammen.
sport.lno: cstart.obj serio.obj _doprint.obj fehler.obj sport.obj cmd_link c:\bso_v5\lib\ext\c166s.lib c:\bso_v5\lib\ext\f166s.lib
 L166 LINK @cmd_link

# ASSEMBLER
# Der Assembler uebersetzt den vom Compiler/Makroassembler erzeugten mnemonischen Code in Maschinenkode
sport.obj: sport.src
 A166 sport.src TO sport.obj
fehler.obj: fehler.src
 A166 fehler.src TO fehler.obj
_doprint.obj:  _doprint.src
 A166 _doprint.src  TO _doprint.obj
serio.obj:  serio.src
 A166 serio.src  TO serio.obj
cstart.obj: cstart.src
 A166 cstart.src TO cstart.obj

# MAKROASSEMBLER
cstart.src: cstart.asm
 m166 cstart.asm

# COMPILER
sport.src: sport.c
 C166 -x -s -g -Ms sport.c
fehler.src: fehler.c
 C166 -x -s -g -Ms fehler.c
_doprint.src:  _doprint.c
 C166 -x -s -g -Ms _doprint.c
serio.src:  serio.c serio.h
 C166 -x -s -g -Ms serio.c
```

```
clean:
        $(exist sport.src del sport.src)
        $(exist sport.obj del sport.obj)
        $(exist sport.lst del sport.lst)
        $(exist sport.lno del sport.lno)
        $(exist sport.out del sport.out)
        $(exist sport.map del sport.map)
        $(exist sport.h86 del sport.h86)
        $(exist fehler.src del fehler.src)
        $(exist fehler.obj del fehler.obj)
        $(exist fehler.lst del fehler.lst)
        $(exist _doprint.src del _doprint.src)
        $(exist _doprint.obj del _doprint.obj)
        $(exist _doprint.lst del _doprint.lst)
        $(exist serio.src del serio.src)
        $(exist serio.obj del serio.obj)
        $(exist serio.lst del serio.lst)
        $(exist cstart.src del cstart.src)
        $(exist cstart.obj del cstart.obj)
        $(exist cstart.lst del cstart.lst)
```



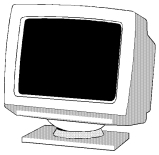
4. Schritt:

Es gibt jetzt 2 Möglichkeiten der Programmentwicklung:



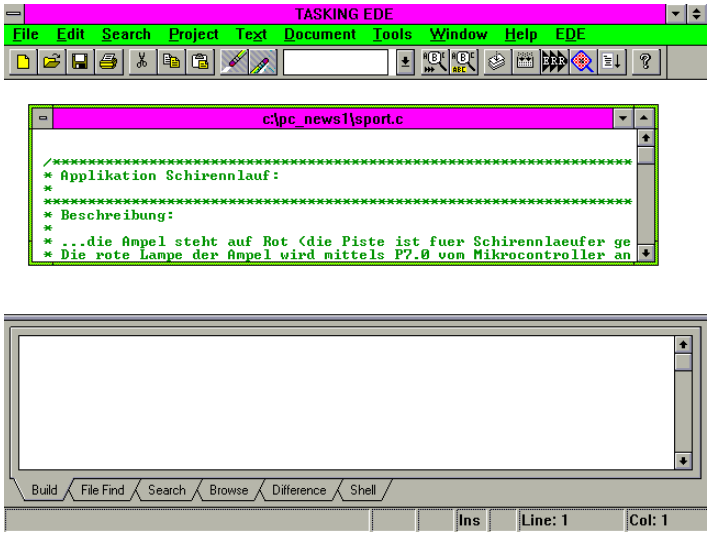
4.1) Programmentwicklung unter MS-DOS:

- Wir nehmen unseren MS-DOS Lieblings-Texteditor und bearbeiten sport.c.
- Anschließend starten wir das makefile mittels mk166.
- Danach laden wir `sport.h86` in unser Zielsystem.
- Befinden sich Syntaxfehler in `sport.c` bricht make (mk166) ab.
- Wir merken uns Fehler und Zeile die der Compiler meldet und beheben diese in unserem Source-File `sport.c`.

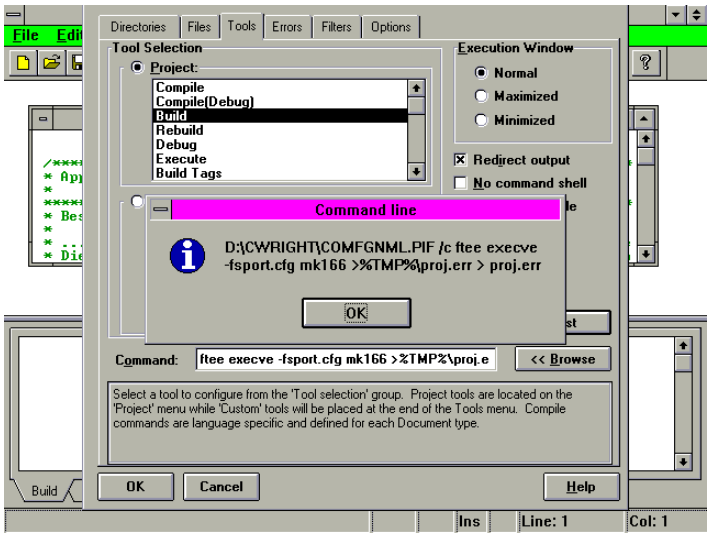


4.2) Programmentwicklung unter MS-WINDOWS und CodeWright:

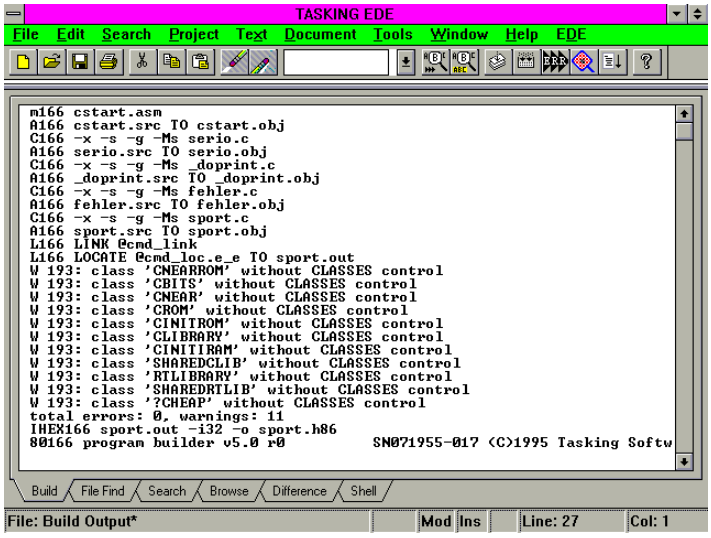
So sieht unsere Entwicklungsoberfläche aus:



Der Aufruf des makefile durch mk166 wird folgendermaßen in unsere graphische Benutzeroberfläche eingebunden:



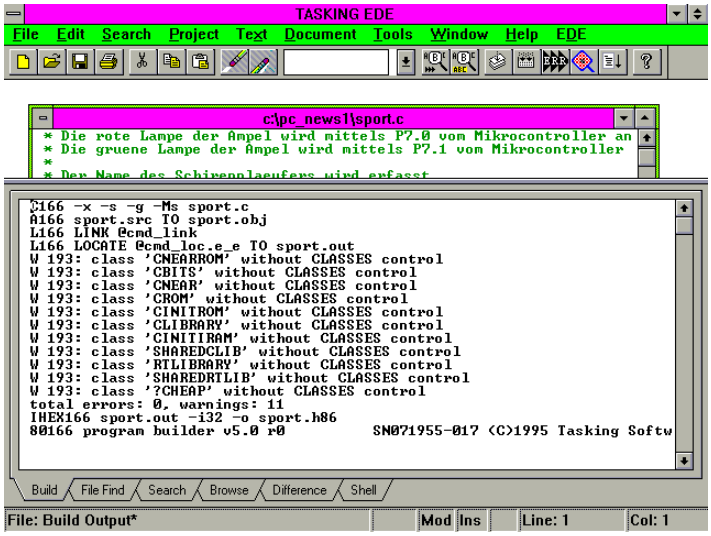
Wird jetzt Build aufgerufen dann sieht das so aus:



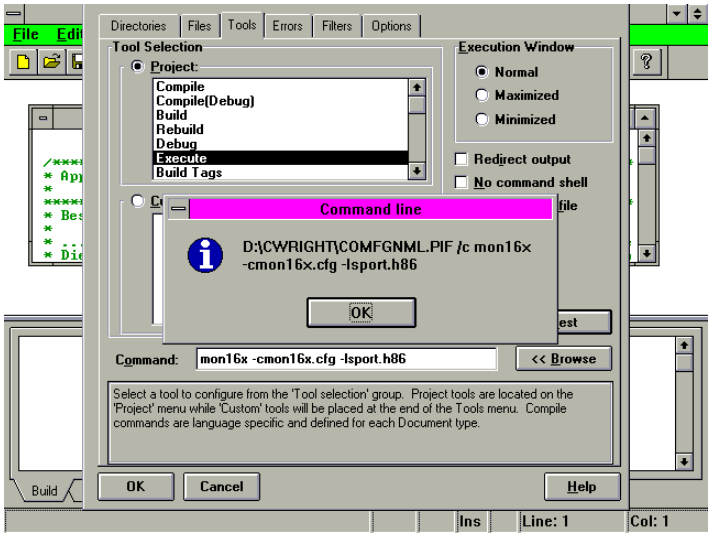
Das schöne an der Windows-Oberfläche ist, daß bei einem Syntaxfehler in `sport.c` mk166 abbricht, der Compiler die Fehler und Zeilennummern zurückmeldet, und der Texteditor automatisch im Source-File an der richtigen Stelle positioniert!

Das Schöne an mk166 ist, daß nur die Dateien neu generiert werden, die notwendig sind.

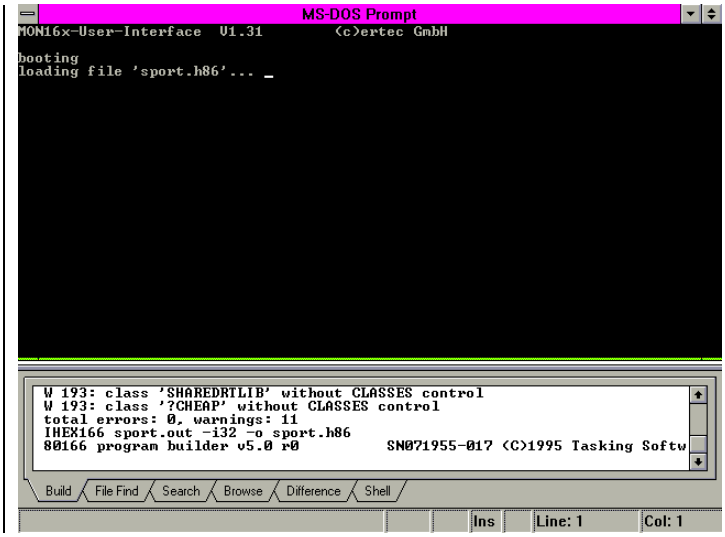
Ändern wir zum Beispiel nur `sport.c`, dann sieht der mk166-Lauf folgendermaßen aus:

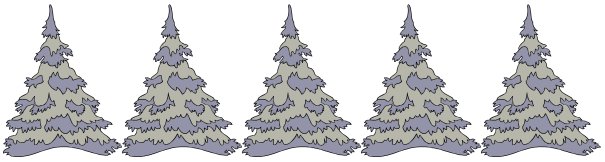


Das Laden und Ausführen von `sport.h86` wird auch in unsere Entwicklungsoberfläche eingebunden:



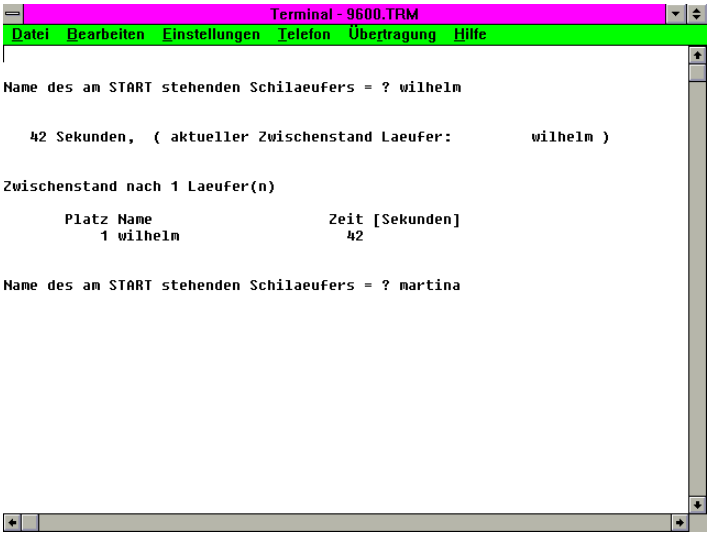
Das Laden des Monitors mittels Bootstrap-Loader und das Nachladen von `sport.h86` mittels Monitor sieht so aus:

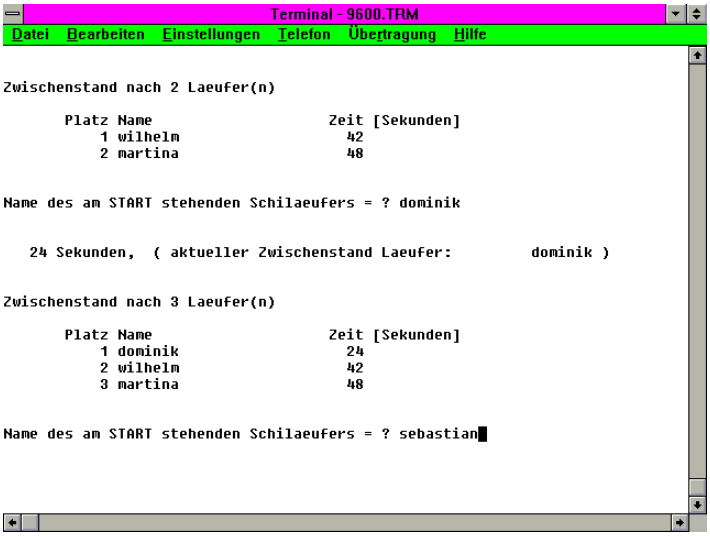


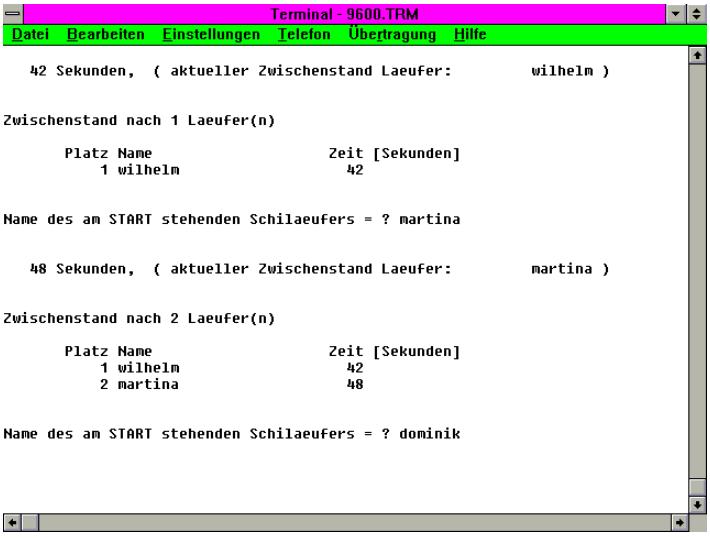


5. Schritt: Benutzerschnittstelle

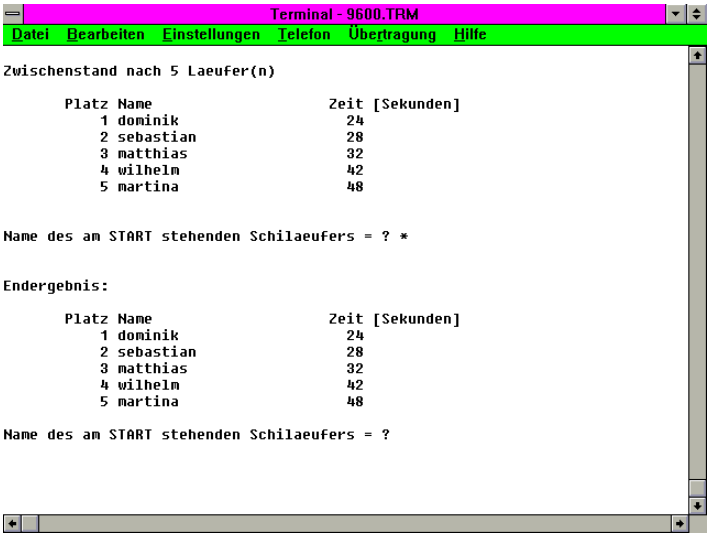
Egal ob wir unsere Applikation unter MS-DOS oder MS-WINDOWS entwickelt haben: jetzt ist es Zeit, die Benutzerschnittstelle als Hardcopy anzuführen. An den Mikrocontroller kann ein ASCII-Terminal oder einfach ein Terminalprogramm angeschlossen werden. Dies sieht dann so aus:







und so weiter, bis zum Endergebnis:



□