Ingenieurbüro für Echtzeitprogrammierung

Terminal-Emulation

Dok-Rev. 1.0 vom 05.04.2000

Software-Rev. 1.0 vom 26.06.1996

Inhaltsverzeichnis

Revisionsliste:

Rev.	Datum	Na.	Änderung
1.0	05.04.2000	Ko	Übernahme

1 Urheberrecht und Haftung

Alle Rechte an diesen Unterlagen liegen bei der IEP GmbH, Langenhagen.

Die Vervielfältigung, auch auszugsweise, ist nur mit unserer ausdrücklichen schriftlichen Genehmigung zulässig.

In Verbindung mit dem Kauf von Software erwirbt der Käufer einfaches, nicht übertragbares Nutzungsrecht. Dieses Recht zur Nutzung bezieht sich ausschließlich darauf, daß dieses Produkt auf oder in Zusammenhang mit jeweils **einem** Computer zu benutzen ist. Das Erstellen einer Kopie ist ausschließlich zu Archivierungszwecken unter Aufsicht des Käufers oder seines Beauftragten zulässig. Der Käufer haftet für Schäden, die sich aus der Verletzung seiner Sorgfaltspflicht ergeben, z.B. bei unautorisiertem Kopieren, unberechtigter Weitergabe der Software usw.. Der Käufer gibt mit dem Erwerb der Software seine Zustimmung zu den genannten Bedingungen. Bei unlizensiertem Kopieren muß vorbehaltlich einer endgültigen juristischen Klärung von Diebstahl ausgegangen werden. Dies gilt ebenso für Dokumentation und Software, die durch Modifikation aus Unterlagen und Programmen von IEP hervorgegangen ist, gleichgültig, ob die Änderungen als geringfügig oder erheblich anzusehen sind.

Eine Haftung seitens IEP für Schäden, die auf den Gebrauch von Software, Hardware oder Benutzung dieses Manuskriptes zurückzuführen sind, wird ausdrücklich ausgeschlossen, auch für den Fall fehlerhafter Software oder irrtümlicher Angaben.

Das Einverständnis des Käufers oder Nutzers für den Haftungsausschluß gilt mit dem Kauf und der Nutzung der Software und dieser Unterlagen als erteilt.

1.1 Handhabung

Lesen Sie bitte zuerst sorgfältig diese Dokumentation bevor Sie anfangen zu programmieren. Sie sparen Zeit und vermeiden Probleme.

1.2 Erklärung

Wir behalten uns das Recht vor, Änderungen, die einer Verbesserung der Schaltung oder des Produktes dienen, ohne besondere Hinweise vorzunehmen. Trotz sorgfältiger Kontrolle kann für die Richtigkeit der hier gegebenen Daten, Schaltpläne, Programme und Beschreibungen keine Haftung übernommen werden. Die Eignung des Produktes für einen bestimmten Einsatzzweck wird nicht zugesichert.

2 Allgemeine Beschreibung

Die Terminalemulation stellt ein virtuelles Terminal an einem RTOS-UH Rechner zur Verfügung. Je nach Hardware wird ein komplettes Terminal mit Display und Tastatur oder nur Teile davon zur Verfügung gestellt.

Unter den Bezeichnungen `/AT`, `/BT` und `/CT` ist die Terminalemulation in der A-, B- oder C-Betriebsart unter der `LDN 4` mit den Drives 0, 2 und 6 erreichbar. Unter der Bezeichnung `/DT` wird der Ausgabekanal im Vollduplex-Betrieb mit der `LDN 14` angesprochen.

Zusätzlich zu den üblichen Betriebsarten kann die Terminalemulation auf `LDN 4` noch unter den Drives 64, 66 und 70 angesprochen werden. Die Terminalemulation liefert in diesem Fall bei Eingaben die Tastatur-Scancodes ohne Übersetzung in ASCII-Zeichen. Auch in diesem Fall ist ein Betrieb entsprechend der A-, B- oder C-Betriebsart möglich, jedoch findet grundsätzlich kein Echo statt. Hierbei ist besonders zu beachten:

- Bei Betriebsartenumschaltung Scan-Code/ASCII kann der Empfangspuffer noch Zeichen in der jeweils anderen Codierung enthalten. Zur Sicherheit sollte der Empfangspuffer daher durch ein erstes Lesen in der A-Betriebsart gelöscht werden.
- Bei Scan-Code-Betrieb müssen die Ausgaben ebenfalls über die Drives 64, 66 oder 70 erfolgen, da ansonsten wieder eine Rückschaltung in den ASCII-Betrieb erfolgt.

3 Steuerzeichen der Terminalemulation

Die Terminalemulation verhält sich weitgehend Televideo-kompatibel. Sie versteht die folgende Befehlssequenzen (Erläuterung der Angaben *Ps*, *Pc*, *Pn*, *r* und *c* siehe unten):

ESC U	Monitormode On
ESC X	Monitormode Off
ESC .*Ps*	Cursor style (0...4) 0 Cursor off 1 Blinking Block 2 steady Block 3 Blinking underline 4 steady underline Alle Zahlen als ASCII-'1' etc.
ESC G*Ps*	Define visual attributes: 0 normal video, default 1 invisible normal video 2 blinking normal video 3 invisible blinking 4 reverse background 5 invisible reverse 6 blinking reverse 7 invisible blinking reverse 8 underline 9 invisible underline : blinking underline ; invisible blinking underline < reverse underline = invisible reverse underline > blinking reverse underline ? invisible blinking reverse underline
ESC $	Special graphics mode on
ESC %	Special graphics mode off
ESC)	Enable invers
ESC (	Disable invers
ESC v	Autoscroll mode off
ESC w	Autoscroll mode on
CTRL J	Line feed
CTRL K	Cursor up

CTRL V	Cursor down
CTRL L	Cursor right
CTRL M	Carriage return
CTRL H	Cursor left
ESC =*rc*	Set Cursor *r*ow *c*olumn
ESC 1	Set tab stop
ESC 2	Clear tab stop
ESC 3	Clear all tab stop
ESC i	Move cursor to next tab stop
ESC I	Move cursor back one tab stop
ESC Q	Insert one space at cursor
ESC E	Insert one line of spaces
ESC W	Delete Char at cursor
ESC R	Delete current line to spaces
ESC T	Delete to end of line
ESC t	Delete to end of line
ESC Y	Delete to end of screen
ESC y	Delete to end of screen
ESC *	Delete screen
ESC ,	Delete screen
ESC ;	Delete screen
ESC +	Delete screen
CTRL Z	Delete screen
ESC :	Delete screen
ESC [=7h	Auto wrap on
ESC [=7l	Auto wrap off
ESC [*Pc*;*Pc*r	Set scroll region (*Pc*: start, end line)
ESC [*Pc*;*Pc*s	Set scroll region (*Pc*: start, end column)
ESC [*Pn*;*Pn*H	Set cursor position (*Pn*: column, line)
ESC [*Pn*A	Cursor up *Pn* rows
ESC [*Pn*B	Cursor down *Pn* rows
ESC [*Pn*C	Cursor right *Pn* columns
ESC [*Pn*D	Cursor left *Pn* columns
ESC ! x	Change forground colour (Blank + colour number)

Die Angaben für mit *Ps*, *Pc*, *Pn*, *r* oder *c* gekennzeichnete Zahlenwerte erfolgen durch Werte, die sich ASCII-codiert aus 32 + *Zahlenwert* errechnen.

Die Befehlssequenz zur Änderung der Schreibfarbe (ESC ! x) ist nicht zu gängigen Terminal-emulationen kompatibel. Neben der einfachen Änderung der Schreibfarbe kann durch Änderung des Grundwertes der Farbangabe folgendes Verhalten erzeugt werden:

Grundwert	Verhalten
32 (Leerzeichen)	Änderung der Schreibfarbe
64 (A)	Änderung der Hintergrundfarbe

Die Funktionstasten F1...F12 liefern standardmäßig die Zeichenfolge `SOH` *`zeichen`* `CR` mit folgendem *`zeichen`*:

Taste	ohne Shift	mit Shift
F1	@	`
F2	A	a
F3	B	b
F4	C	c
F5	D	d
F6	E	e
F7	F	f
F8	G	g
F9	H	h
F10	I	i
F12	K	k

Die Programmierung der Funktionstasten erfolgt mit der Befehlssequenz

`ESC | `*`p1`*` 1 `*`text`*` Ctrl Y`

und den Parametern

`p1`	Kennzeichnung der Funktionstaste, s.u.
`text`	gewünschte Tastenbelegung
`Ctrl Y`	Endekennung

Die Kennzeichnung der ausgewählten Funktionstaste erfolgt mit folgenden Zeichen:

Taste	ohne Shift	mit Shift
F1	1	<
F2	2	=
F3	3	>
F4	4	?
F5	5	@
F6	6	A
F7	7	B
F8	8	C
F9	9	D
F10	:	E
F11	;	F
F12	G	L

4 Grafikgrundfunktionen

Neben den unter RTOS-UH üblichen Grundfunktionen `SETPIX`, `GETPIX` und `LINE` (siehe RTOS-UH-Handbuch werden die im folgenden in alphabetischer Reihenfolge beschriebenen Funktionen unterstützt.

4.1 Besonderheiten der PVGA

Die Auflösung der PVGA beträgt 640 x 480 Pixel bei 16 Farben. Für jedes Pixel auf dem Bildschirm werden 4 Bit im Video-RAM abgelegt. Mit dem Parameter *col* kann die Farbe bei den entsprechenden Funktionen festgelegt werden. Zulässig sind Werte von 0 (Hintergrundfarbe) bis 15. Die oberen 3 Bits des Parameters *col* legen die Art des Zeichnens auf dem Bildschirm fest:

col	Zeichenart
0 0 0	absolut
0 0 1	not
0 1 0	and
1 0 0	or
1 1 0	exor

Folgende Farben sind den Farbpaletten zugeordnet:

Palettennr.	Farbe
0	schwarz
1	rot
2	grün
3	blau
4	gelb
5	hellblau
6	violett
7	hellgrau
8	orange
9	hellgrün
10	flieder
11	braun
12	graugrün
13	pink
14	dunkelgrau
15	weiß

Die Positionsangabe x, y ist in Pixel anzugeben. Die linke obere Ecke des Bildschirms ist der Nullpunkt. Positive x-Werte gehen nach rechts auf dem Bildschirm, positive y-Werte nach unten.

4.2 Grafikfunktionen

4.2.1 BIN_TEXT

```
BIN_TEXT: PROC( (x, y)      FIXED(15),
                col         FIXED(15),
                zeichen_adr FIXED(31)
              ) GLOBAL;
```

Die Prozedur `BIN_TEXT` überträgt ein Zeichen aus dem Zeichengenerator-ROM in der Farbe *`col`* auf den Bildschirm an die Position (*`x, y`*). Die Adresse im ROM wird mit *`zeichen_adr`* angegeben. Sie hängt von der Größe des auszugebenden Zeichens ab:

```
zeichen_adr = (xhöhe * ybreite + 7) / 8
```

4.2.2 BOX

```
BOX: PROC( (x,    y    ) FIXED(15),
           (Xend, Yend ) FIXED(15),
           col           FIXED(15)
         ) GLOBAL;
```

Die Prozedur `BOX` zeichnet in der Farbe *`col`* einen rechteckigen Rahmen mit den diagonalen Punkten (*`x, y`*) (linke obere Ecke) und (*`Xend, Yend`*).

4.2.3 BOX_FILLED

```
BOX_FILLED: PROC( (x, y       ) FIXED(15),
                  (Xend, Yend ) FIXED(15),
                  col           FIXED(15)
                ) GLOBAL;
```

Die Prozedur `BOX_FILLED` zeichnet ein in der Farbe *`col`* gefülltes Rechteck mit den diagonalen Punkten (*`x, y`*) und (*`Xend, Yend`*).

4.2.4 BOX_MOVE

```
BOX_MOVE: PROC( (Xstart, Ystart) FIXED(15),
                (Breite, Hoehe ) FIXED(15),
                (Xziel,  Yziel ) FIXED(15),
                mode             FIXED(15)
              ) GLOBAL;
```

Die Prozedur `BOX_MOVE` kopiert einen rechteckigen Bildausschnitt der Breite *`Breite`* und der Höhe *`Hoehe`* beginnend bei dem Startpunkt (*`Xstart, Ystart`*) auf einen Bildbereich gleicher Größe mit dem Startpunkt (*`Xziel, Yziel`*). Der Zielbereich kann außerhalb des sichtbaren Bildschirmbereichs liegen, d.h. *`Ywidth`* wird überschritten. *`mode`* legt die Zeichenart fest, im unteren Nibble sind folgende Werte zulässig:

dez.	hex	Art
12	0x0C	absolut
3	0x03	not
8	0x08	and
14	0x0E	or
6	0x06	exor

Das nächste Nibble legt die Schreibfarbe fest, d.h. für ein Pixel werden die 4 Bit aus dem Video-RAM entsprechend dem unteren Nibble gelesen, mit der Schreibfarbe "verundet" und wieder im Video-RAM abgelegt. Wird der Parameter *`mode`* auf absolut (=12) gesetzt, so wird der original Bildausschnitt an der Zielstelle abgelegt. Andere Werte sind experimentierfreudigen Anwendern vorbehalten.

4.2.5 BOX_READ

```
BOX_READ: PROC( (Xstart, Ystart) FIXED(15),
                (Breite, Hoehe ) FIXED(15),
                (Xziel,  Yziel ) FIXED(15),
                feld STRUCT[ REF CHAR(1) ]
              ) GLOBAL;
```

Die Prozedur `BOX_READ` kopiert einen rechteckigen Bildausschnitt der Breite *`Breite`* und der Höhe *`Hoehe`* beginnend bei dem Startpunkt (*`Xstart, Ystart`*) in den Speicherbereich, der mit *`feld`* angegeben wird. Dabei findet **keine** Überprüfung der Größe des Speicherbereiches statt. Ist das angegebene Feld zu klein, ist der Absturz des Rechners so gut wie sicher!

4.2.6 BOX_WRITE

```
BOX_WRITE: PROC( (Xstart, Ystart) FIXED(15),
                 (Breite, Hoehe ) FIXED(15),
                 (Xziel,  Yziel ) FIXED(15),
                 mode             FIXED(15),
                 feld STRUCT[ REF CHAR(1) ]
               ) GLOBAL;
```

Die Prozedur BOX_WRITE kopiert den Speicherbereich, der mit *feld* angegeben wird, auf einen rechteckigen Bildausschnitt der Breite *Breite* und der Höhe *Hoehe* beginnend bei dem Startpunkt (*Xstart, Ystart*). Dabei findet **keine** Überprüfung der Größe des Speicherbereiches statt. Ist das angegebene Feld zu klein, wird der folgende Speicher auf den Bildschirm geschrieben. Der Parameter *mode* ist bei der Prozedur BOX_MOVE beschrieben.

4.2.7 CIRCLE

```
CIRCLE: PROC( (Xmitte, Ymitte, Radius) FIXED(15),
              col                      FIXED(15)
            ) GLOBAL;
```

Die Prozedur CIRCLE zeichnet Vollkreise auf den Bildschirm.

4.2.8 CLEAR

```
CLEAR: PROC  GLOBAL;
```

Die Prozedur CLEAR löscht den **sichtbaren** Bildschirmbereich mit der Farbe 0.

4.2.9 GET_TEXT_WIDTH

```
GET_TEXT_WIDTH: PROC( (xhoehe, ybreite)  FIXED(15) IDENT
                    ) GLOBAL;
```

Die Funktion GET_TEXT_WIDTH liefert die Größe eines Zeichens in den Variablen *xhoehe* und *ybreite* zurück.

4.2.10 PLANE

```
PLANE: PROC( Farbnummer   FIXED(15),
             Palettenwert FIXED(31)
           ) GLOBAL;
```

Mit dieser Funktion können die Farbpaletten für die 16 Farben des Display-Prozessors gesetzt werden. Für *Farbnummer* sind Werte von 0 bis 15 zulässig. Folgende Bits vom *Palettenwert* werden genutzt:

Farbe	Bit-Nummer (PEARL)	Bit-Nummer (Assembler)
Rot	12...14	18...20
Grün	20...22	10...12
Blau	28...30	2... 4

Der Aufbau des Langwortes *Palettenwert* läßt sich durch die Einzelbit-Darstellung

```
[---- ---- ---r rr-- ---g gg-- ---b bb--]
```

veranschaulichen.

4.2.11 PLINIT

```
PLINIT: PROC GLOBAL;
```

Initialisiert den Display-Prozessor. PLINIT ist eine Prozedur ohne Parameter und muß vor Benutzung der Grafik-Routinen einmal aufgerufen werden.

4.2.12 SET_TEXT_WIDTH

```
SET_TEXT_WIDTH: PROC( (xhöhe, ybreite)  FIXED(15) IDENT
                    ) GLOBAL;
```

Die Funktion SET_TEXT_WIDTH setzt die Größe eines Zeichens für die Funktionen BIN_TEXT und TEXT.

4.2.13 TEXT

```
TEXT: PROC( (x, y)  FIXED(15),
            col     FIXED(15),
            string  STRUCT[ str CHAR(255), len FIXED(15) ]
          ) GLOBAL;
```

Die Prozedur TEXT schreibt die in *string.str* abgelegte Zeichenkette beginnend auf der Position (*x*, *y*) (linke, obere Ecke des ersten Zeichens) in der Farbe *col* auf den Bildschirm. Die Ausgabe endet, wenn *string.len* Zeichen ausgegeben wurden.

4.2.14 WIDTH

```
WIDTH: PROC( (Xwidth, Ywidth)  FIXED(15) IDENT
           ) GLOBAL;
```

Die Funktion WIDTH gibt in den Variablen *Xwidth* und *Ywidth* die Größe des Bildschirms in Pixel zurück.