# RTOS–UH

Prof. Dr.-Ing. W. Gerth

Besonderheiten des

## RTOS-UH Testsystems

für den ATARI-Emulator WinSTon

April 2001

# Inhaltsverzeichnis

# Kapitel 1: Ein-/Ausgabe-Dations

Der Zugriff auf die PC-Festplatte erfolgt vom RTOS-UH Testsystem unter dem ATARI-Emulator WinSTon über die neuen Dations

- `/X0` (virtuelle ATARI-Partition `C:`),
- `/X1` (virtuelle ATARI-Partition `D:`),
- `/X2` (virtuelle ATARI-Partition `E:`),
- `/X3` (virtuelle ATARI-Partition `F:`).

Die virtuellen ATARI-Partitionen können über die Einstellungen des WinSTon auf beliebige Verzeichnisse der Windows Partitionen abgebildet werden. Auf diese Weise ist ein einfacher Datentransfer zwischen RTOS-UH und Windows-Programmen möglich. Achtung: Im Treiber dieser Dations sind noch nicht alle Funktionen (wie z.B. `SEEK`) implementiert.

Über die Floppy-Dations `/F0` und `/F1` kann auf die Disc-Images des WinSTon, falls eingerichtet, zugegriffen werden.

Folgende Dations des Original ATARI-Systems werden **nicht** unterstützt :

- `/PP` (parallele Schnittstelle),
- `/A2`, `/B2`, `/C2` (serielle Schnittstelle),
- `/A3`, `/B3`, `/C3` (Midi-Schnittstelle),
- `/H0`, `/H1`, `/H2` und `/H3` (Festplatte).

# Kapitel 2: Tastatur und Terminalemulation

## 2.1 Belegung der Tastatur

Der nachfolgenden Tabelle sind die Sonderbelegungen der PC-Tastatur und ihre Abbildungen auf die Atari-Tastatur (bei Verwendung der mitgelieferten Tastatur-Konfigurationsdatei `rtos.map`) zu entnehmen.

| **PC** | **Atari** | **Funktion** |
|---|---|---|
| Rollen | Help | Hauptbildschirm |
| Shift+Rollen | Shift+Help | Letzter Bildschirm |
| Ctrl+Cursordown | Ctrl+Cursordown | Wechsel der Schriftfarbe |
| Pos 1 | F5 | WE: Dateianfang |
| Ende | F6 | WE: Dateiende |
| Bild ab | F7 | WE: Seite weiter |
| Bild auf | F8 | WE: Seite zurück |
| Backspace | F9 | WE: Backspace |

Die Funktionstasten F5 bis F9 des ATARI werden in der Datei `AUTOC` auf die oben genannten Funktionen des Editors WE umprogrammiert (s. Funktionstastenprogrammierung auf S. 6).

## 2.2 Terminalemulation des Atari

Die Terminalemulation der Atari-ST-Serie versteht die folgenden Steuerzeichen:

| | |
|---|---|
| `$07` | Ring the bell |
| `$08` | Cursor einen Schritt nach links (Backspace) |
| `$0A` | Cursor einen Schritt nach unten (Linefeed) |
| `$0B` | Cursor einen Schritt nach oben (Vertical tab) |
| `$0C` | Cursor einen Schritt nach rechts (Formfeed) |
| `$0D` | Cursor an den Anfang der Zeile (Carriage return) |
| `$1A` | Bildschirm löschen, Cursor Home |
| `$1E` | Cursor in die linke obere Bildschirmecke setzen (Home) |

Die folgenden Escape-Sequenzen sind z.B. zum Televideo 910 kompatibel:

| | |
|---|---|
| `Esc *` | löscht den gesamten Bildschirm |
| `Esc Y` | löscht ab Cursor-Position den Rest des Bildschirms |
| `Esc y` | wie `Esc Y` |
| `Esc T` | löscht ab Cursor-Position den Rest der Zeile |
| `Esc t` | wie Esc T |
| `Esc =r,c` | Cursor positionieren: r=Row c=Column; zu den Werten für r und c sind immer `$20` zu addieren |
| `Esc 8` | Smooth-Scroll einschalten |
| `Esc 9` | Auf normalen Scroll umschalten |
| `Esc b` | den gesamten Bildschirm invertieren |
| `Esc d` | normaler Bildschirm |
| `Esc G 0` | Video-Attribut: normale Schrift |
| `Esc G 4` | Video-Attribut: inverse Schrift |
| `Esc G 8` | Video-Attribut: unterstrichene Schrift |
| `Esc G <` | Video-Attribut: inverse und unterstrichene Schrift |
| `Esc E` | deutschen Zeichensatz laden |
| `Esc R` | US-ASCII Zeichensatz laden |

## 2.3 Eigener Zeichensatz

Der Zeichensatz liegt im RAM-Bereich des Rechners. Vom Nutzer kann also auch ein eigener Zeichensatz geladen werden. Der Zeichensatz beginnt mit dem Zeichen `$00`. Jeder Buchstabe ist mit 16 (bei Farbe 8) Byte abgelegt. Den Anfang der Zeichensatztabelle findet man, indem man `$00000320` zum Inhalt der Speicherzelle `$81E` addiert. Der Platz reicht für einen kompletten Zeichensatz mit 256 Zeichen, der ab der angegebenen Stelle ins RAM kopiert werden kann. Nach einem Abort ist der US-ASCII Zeichensatz geladen.

## 2.4 Eigene Tastaturbelegung

Die Tastaturtabellen liegen im RAM-Bereich des Rechners. Sie setzen die Codes, die von der Tastatur geliefert werden, in ASCII- oder Sonderzeichen um. Jede Tastaturtabelle ist 128 Byte lang und kann erreicht werden, indem der Offset aus der Tabelle auf den Inhalt der Speicherzelle `$81E` addiert wird.

| **Tastensatz** | **Offset** |
|---|---|
| normal | `$1320` |
| shift | `$13A0` |
| capslock | `$1420` |
| control | `$14A0` |
| alternate | `$1520` |

## 2.5 Tastaturcodes

Über die Datenstation `/A1` können direkt die Tastaturcodes - ohne Umsetzung über eine Tabelle - eingelesen werden, wenn dabei die Drives 64, 66 und 70 benutzt werden. Die Station verhält sich bei Drive=64 wie eine `A`-Station, bei Drive=66 wie eine `B`-Station und bei Drive=70 wie eine `C`-Station.

**Beispiel**:

```
SYSTEM;
.
TastA:LD/0,64/(TFU=1)<-;
TastB:LD/0,66/(TFU=1)<-;
TastC:LD/0,70/(TFU=1)<-;
.
PROBLEM;
.
SPC (TastA,TastB,TastC) DATION IN ALPHIC CONTROL(ALL);
.
.
GET char FROM TastA;
```

Wenn sowohl Tastaturcodes als auch normale ASCII-Zeichen eingelesen werden sollen, muss beachtet werden, dass beim Einlesen von ASCII-Zeichen über das B-Port noch Tastaturcodes im Puffer stehen können. Werden die ASCII-Zeichen über das A-Port gelesen, wird der Puffer gelöscht.

# Kapitel 3: Zusätzliche Bedienbefehle

## C O N / C O F F — Cursor On/Off

Syntax: `CON` oder `COFF`

Beschreibung: Mit dem Befehl `COFF` kann der blinkende Cursor des Hauptbildschirms ausgeschaltet werden. Der Befehl `CON` dient zum Wiedereinschalten des Cursors.

## F 1 . . . F 1 0 — Funktionstastenprogrammierung

Syntax: `F1` *text* ... `F10` *text*

Beschreibung: Mit den Befehlen `F1` bis `F10` können die Funktionstasten des Atari programmiert werden. Die Belegung der Tasten bleibt auch nach einem Abort erhalten. Die Tastenbelegung kann jederzeit durch erneuten Aufruf von `F1` bis `F10` geändert werden.

Der Text *text* darf aus ASCII-Zeichen bestehen. Seine Länge ist pro Taste auf 80 Zeichen beschränkt. Um Control-Zeichen einzugeben, ist das Zeichen "^"zu benutzen.

Beispiel 1: Die Taste `F1` soll so belegt werden, dass man die Systemübersicht erhält.

Kommando: `F1 ^AS^M`

Es wird zuerst `Ctrl A` eingegeben. Dann folgt das `S` für die Systemübersicht. Als letztes wird ein `CR`, entspricht `Ctrl M`, eingegeben.

Beispiel 2: Es soll bei Betätigung der Taste `F9` eine Datei vom Laufwerk `/F0` in ein `ED`-File kopiert werden.

Kommando: `F9 ^ACOPY /F0/XYZ > /ED/XYZ`

Zuerst wird wieder `Ctrl A` eingegeben, um das Bedieninterface aufzuwecken. Dann folgt der `COPY`-Befehl. Hier wurde kein `CR` angehängt, so dass der Befehl noch von Hand losgeschickt werden muss.

# Kapitel 4: Systemadressen und Prozessinterrupts

## 4.1 Systemadressen

| | | | |
|---|---|---|---|
| videopointer | `$81A` | long | Zeigt auf die Startadresse des Hauptbildschirms. Er darf nicht verändert werden. |
| hide_screen | `$83A` | long | Zeigt auf den Bildschirm, der zuletzt genutzt wurde, der Bildschirm muss nicht sichtbar sein. |
| change_screen | `$84A` | long | Ist der Wert ungleich Null, wird dieser Wert beim nächsten vertikalen Bildrücklauf in den Videoshifter geschrieben. Das hintere Byte wird dabei nicht berücksichtigt und change_screen wird auf Null gesetzt. |
| key_answer | `$8E6` | 8 words | Hier können bis zu 8 Worte stehen, die vom Tastaturprozessor als Statusmeldung geschickt werden. Alle Meldungen, auch kürzere, beginnen bei `$8E6`. Die Meldung wird in der Reihenfolge abgelegt, wie sie von Atari dokumentiert ist. Bei Mausbewegungen z.B. liegt der Mausstatus bei `$8E6`, die X-Koordinate bei `$8E7` und die Y-Koordinate bei `$8E8`. |

## 4.2 Prozessinterrupts

Zwei Prozessinterrupts sind angeschlossen.

| | |
|---|---|
| `EV 80000000` | Dieser Interrupt wird ausgelöst, wenn eine komplette Statusmeldung des Tastaturprozessors vorliegt. Damit kann z.B. auf Bewegungen der Maus von PEARL aus reagiert werden (s. Systemadresse `$8E6`). |
| `EV 40000000` | Dieser Interrupt wird bei jedem vertikalen Bildrücklauf gefeuert. Er kann benutzt werden, um synchron zum Bildrücklauf - in der Austastlücke - zwischen verschiedenen Bildschirmen umzuschalten. |

# Kapitel 5: Erweiterte Grafik-Einbaufunktionen

Die erweiterten Einbaufunktionen sind nicht im Compiler angeschlossen, müssen daher vor Benutzung im Programm spezifiziert werden. Je nach Einstellung im WinSTon wird eine Auflösung von 640x400 Pixel oder von 640x200 Pixel (4 Farben) unterstützt. Es stehen folgende Funktionen zur Verfügung:

| | |
|---|---|
| `CALL SCLOAD(modname,drive,filename)` | Bildschirm von Floppy laden. |
| `CALL SCSAVE(modname,drive,filename)` | Bildschirm auf Floppy sichern. |
| `CALL SCRSET(modname)` | Bildschirm im System einrichten. |
| `CALL SCKILL(modname)` | Bildschirm aus dem System entfernen. |
| `CALL SCRCHG(modname)` | angezeigten Bildschirm wechseln. |
| `CALL PLANE(planenr,colour)` | Farbregister setzen. |
| `CALL PEN(colour)` | Schreibfarbe setzen. |
| `CALL SPRITL(xpos,ypos,xlen,ylen,modname,array)` | Sprite laden. |
| `CALL SPRITS(xpos,ypos,xlen,ylen,modname,array)` | Sprite schreiben. |
| `CALL WIDTH(xwidth,ywidth)` | Auflösung feststellen. |
| `CALL CLEAR` | Bildschirm löschen. |
| `CALL CIRCLE(xpos,ypos,radius,colour)` | Kreis zeichnen. |
| `CALL TEXT(xpos,ypos,colour,text)` | Text an Position ausgeben. |

Parameter:
`modname CHAR(24)`, `filename CHAR(24)`, `text CHAR(80)`,
`array ( , ) BIT(16) IDENT`, `(xwidth,ywidth) FIXED(15) IDENT`,
`sonst FIXED(15)`

Die Funktionen sind dabei wie nachfolgend beschrieben zu verwenden.

`SCLOAD` Von der Diskette (Laufwerk `/F0` für `drive=0`, Laufwerk `/F1` für `drive=1`) wird der mit drivename bezeichnete File in den mit modname bezeichneten Bildschirm geladen. Es wird geprüft, ob der angegebene Bildschirm eingerichtet ist. Wenn nicht, erfolgt eine Fehlermeldung und der Ladevorgang wird abgebrochen.

`SCSAVE` Der mit modname bezeichnete Bildschirm wird auf der Diskette (Laufwerk `/F0` für `drive=0`, Laufwerk `/F1` für `drive=1`) unter dem mit filename angegebenen Namen abgelegt. Existiert der Bildschirm nicht im System, erfolgt eine Fehlermeldung und der File wird nicht angelegt.

`SCRSET` Es wird ein verdeckter Bildschirm im System eingerichtet. Damit ist es möglich, Graphiken im Hintergrund zu erstellen. Der Bildschirm erhält den Namen, der mit modname übergeben wird. Vor modname wird automatisch ein `#` eingefügt, um ein Entladen mit `UNLOAD` zu verhindern. In den Taskworkspace der aufrufenden Task wird die Adresse des Bildschirms eingetragen. Diese Adresse wird von `SETPIX`, `LINE`, `CIRCLE` und `CLEAR` benutzt. Ist ein Bildschirm mit modname schon im System vorhanden, wird kein zweiter eingerichtet, sondern es erfolgt nur die Eintragung der Adresse. Die Adresse wird bei jeder Aktivierung der Task gelöscht, so dass z.B. eine zyklisch eingeplante Task jedesmal `SCRSET` aufrufen muss. Steht nicht mehr genügend Speicher zur Einrichtung des Bildschirms zur Verfügung, wird die Task suspendiert. Sie läuft bei genügend freiem Speicher weiter.

`SCKILL` Mit `SCKILL` kann ein verdeckter Bildschirm aus dem System entfernt werden. Dabei wird automatisch die Adresse des Bildschirms im Taskworkspace gelöscht. Diese Routine steht auch als Bedienkommando zur Verfügung. Dann muss vor dem Modulnamen aber das `#` mit eingegeben werden. Vorsicht ist geboten, wenn ein Bildschirm entfernt wird, in dem noch gemalt wird: es kann zu Systemabstürzen kommen!

`SCRCHG` Der angegebene Bildschirm wird zur Anzeige gebracht. Wird das Umschalten des Schirmes mit dem Bildrücklauf synchronisiert, können, je nach Anzahl der Bilder, kleine Trickfilme realisiert werden. Dieser Befehl steht auch als Shell-Erweiterung zur Verfügung. Dann muss beim modname aber ebenfalls das `#` mit angegeben werden.

`PLANE` Die Farbinformation in den Farbpalettenregistern wird geändert. Damit ändert sich die Farbe des gesamten Bildschirmes für alle Bilder. Bei schwarz/weiss-Darstellung wird nur das letzte Bit im Plane-Register berücksichtigt. In der mittleren Auflösung, die von RTOS-UH unterstützt wird, werden die Plane-Register 0-3 berücksichtigt. Die Organisation der Farbpalettenregister sieht folgendermaßen aus:

```
FEDCBA9876543210
-----rrr-ggg-bbb
```

Die mit `r` bezeichneten Bits bestimmen den Rot-Anteil, die mit `g` bezeichneten den Grün- und die mit `b` bezeichneten den Blau-Anteil der Farben. Je größer der Wert dieser drei Bits ist, um so intensiver ist die Farbe.

`PEN` Es wird die Schriftfarbe festgelegt. Die Farbe 0 entspricht dem Hintergrund. Mit den Farben 1-3 kann das entsprechende Paletten-Register angewählt werden.

`SPRITL` Von dem angegebenen Bildschirm wird ein Ausschnitt der Größe `xlen`, `ylen` in ein zweidimensionales `BIT(16)` PEARL-Feld gerettet. Es wird geprüft, ob der Bereich innerhalb des Bildschirms liegt, wenn nicht, erhält man die Fehlermeldung `Out_of_screen`. Desweiteren wird überprüft, ob der Ausschnitt in das angegebene PEARL-Feld passt. Dabei werden die einzelnen Feldgrenzen überprüft. Reicht eine nicht, erhält man die Fehlermeldung `X-Dim._of_Array_to_small` oder `Y-Dim._of _Array_to_small`. Sind `xlen`/`ylen` kleiner oder gleich Null, wird die Meldung `Xlen_negativ` oder `Ylen_negativ` ausgegeben. Bei jedem der beschriebenen Fehler wird die aufrufende Task suspendiert und bei einem Continue terminiert. Die X-Dimension des Feldes muss die Größe `(xlen/16)` haben. Wenn der Rest dieser Division ungleich Null ist, muss das Feld ein Wort größer sein. Die Y-Dimension hat die Größe von `ylen`. Es wird nur der tatsächlich angegebene Ausschnitt gerettet, ist z.B. `Xlen=5`, so werden in X-Richtung 5 Bits linksbündig in das Feld geschrieben.
Beispiel:

```
Bildschirm: ooooxxxxxxooo...    Feld: xxxxxx....
            ooooxxxxxxooo...          xxxxxx....
            ooooxxxxxxooo...          xxxxxx....
            ................          ..........
Aufruf:     DCL Feld (1,5) BIT(16);
            CALL SPRITL(4,0,6,3,'MAIN ',Feld);
```

In dem Feld liegen die Bits wie auf dem Bildschirm. Um in dem obigen Beispiel z.B. auf das Pixel mit den Koordinaten (3,1) (hervorgehoben) zuzugreifen, muss mit `BIT=Feld(1,2).BIT(4);` gearbeitet werden.
Bei Farbdarstellung werden beide Planes gerettet. Zuerst wird das erste Plane und danach das Zweite komplett im Feld abgelegt. Es wird also die doppelte Y-Länge des Feldes benötigt. Da die Auflösung in Y-Richtung bei Farbe auch nur 200 Pixel beträgt, kann der gesamte Bildschirm sowohl bei Farbe wie auch bei Schwarz/Weiß in ein Feld der Größe (40,400) `BIT(16)` gerettet werden.
Anmerkung: Mit `SPRITL/SPRITS` können Bildausschnitte über den Bildschirm bewegt werden. Dazu muss ein Bildausschnitt mit SPRITL gerettet werden, dann mit dem Ausschnitt überschrieben und zuletzt mit SPRITS restauriert werden. Wenn sowohl die X-Koordinate als auch `Xlen` ohne Rest durch 8 teilbar sind, wird der Transfer in Hochgeschwindigkeit durchgeführt! Je größer der Bereich ist, desto mehr wirkt sich der Geschwindigkeitsvorteil aus. Wird z.B. der ganze Bildschirm in ein Feld geschrieben, geht es ca. 6 mal schneller, als wenn ein Bit weniger gerettet würde.

`SPRITS` Ein Feld(ausschnitt) wird auf einen Bildschirm geschrieben. Alle Überprüfungen erfolgen genauso, wie bei `SPRITL`.

`WIDTH` Die aktuelle Bildschirmauflösung wird in die Variablen `xwidth` und `ywidth` eingetragen (z.B. hohe Auflösung `xwidth=640`, `ywidth=400`). Damit ist es möglich, Programme zu schreiben, die bei verschiedenen Auflösungen lauffähig sind.

`CLEAR` `CLEAR` löscht den Bildschirm, der im Taskworkspace mit `SCRSET` eingetragen wurde. Ist kein eigener Bildschirm eingerichtet, so benutzt `CLEAR`, ebenso wie `LINE`, `SETPIX` und `CIRCLE`, den Hauptbildschirm.

`CIRCLE` Es werden Vollkreise auf den Bildschirm, der im Taskworkspace eingetragen ist, gezeichnet. Ist kein Bildschirm eingetragen, wird auf den Hauptbildschirm gezeichnet. Ist der Kreis größer als der Bildschirm, werden nur die entsprechenden Segmente gezeichnet. Eine Zerstörung anderer Speicherstellen ist ausgeschlossen.

`TEXT` Es kann ein beliebiger Text auf den Bildschirm, der im Taskworkspace eingetragen ist, ausgegeben werden. Der Text darf maximal 80 Zeichen lang sein. Er wird solange ausgegeben, bis nur noch Leerzeichen kommen. Diese werden unterdrückt, um auch kürzere Texte ausgeben zu können. Der Text wird nur auf den Bildschirmspeicher geschrieben, es kann nicht zu einer Systemzerstörung kommen. `xpos`, `ypos` geben die linke, obere Ecke des ersten Zeichens an, Schreibrichtung ist nach rechts.

Ist `modname='MAIN '`, so wirken die Routinen auf den Hauptbildschirm. Der Aufruf von `SCRSET` ist nicht nötig. Wird bei den Routinen `LINE`, `SETPIX`, `CIRCLE` und `TEXT` für die Farbe eine negative Zahl eingesetzt, werden die Bits auf dem Bildschirm invertiert. Durch zweimaligen Aufruf mit den gleichen Koordinaten kann z.B. eine Linie wieder gelöscht werden.

**Beispiel:**

```
MODULE;
...
PROBLEM;
SPC CLEAR ENTRY GLOBAL;
SPC WIDTH ENTRY (FIXED(15),FIXED(15)) GLOBAL;
SPC SCRSET ENTRY (CHAR(24)) GLOBAL;
SPC SCRCHG ENTRY (CHAR(24)) GLOBAL;
...
DRAWLINE:TASK;
DCL (xwidth,ywidth) FIXED(15);
...
```

```
   CALL WIDTH(xwidth,ywidth);
    .
   CALL SCRSET('BILD');
   CALL CLEAR;
    .
   CALL LINE(0,0,xwidth,ywidth,1);
    .
   CALL SCRCHG('BILD');
...
END;
...
MODEND;
```