Spuler v4.5

Martin Rogge
Möllenholt 24
24107 Kiel

27. Juli 1995

Inhaltsverzeichnis

1 Lieferumfang

Das System SPULER v4.5 besteht aus folgenden Dateien:

`SPULER45.APP`	Oberfläche mit integriertem Filespuler
`SP45_TSR.PRG`	Hintergrundprogramm
`SP45_CFG.PRG`	Konfigurationsprogramm
`SP45_CFG.RSC`	zugehörige Resource-Datei
`SPULER45.IMG`	Zustandsdiagramm im GEM-Imageformat
`SPULER45.DVI`	Dokumentation als TEX-Datei
`SPULER45.TXT`	Dokumentation im ASCII-Format

Die Weitergabe des unvollständigen Systems ist untersagt.

2 Copyright

SPULER v4.5 ist nicht PD, d.h. ich, Martin Rogge, behalte alle Rechte an dem Programm. Es ist aber erlaubt und erwünscht, SPULER v4.5 zu benutzen und komplett mit allen Dateien weiterzugeben, auch in komprimierter Form. Dabei darf jedoch nur der zur Deckung aller Unkosten benötigte Geldbetrag gefordert werden. Falls jemand irgendwo mehr bezahlen mußte, so möge er oder sie mir bitte schreiben. Auch Fehlerberichte und Fragen sind mir willkommen.

Postanschrift:	Martin Rogge Möllenholt 24 24107 Kiel
MausNet:	Martin Rogge @ KI
Internet:	martin_rogge @ ki.maus.de

Selbstverständlich geschieht die Benutzung auf eigene Gefahr, ich übernehme keinerlei Haftung für irgendwelche Datenverluste oder sonstwas. Nach diesen strengen Worten kann ich Dich, lieber Leser, aber beruhigen: Ich persönlich benutze das Programm seit Jahren (in der jeweils aktuellen Version) und habe bestimmt schon Gigabytes hindurchgepumpt, ohne jedes Problem.

3 Beschreibung

3.1 Wozu das Ganze

Eine der unangenehmsten Eigenschaften des Atari-Betriebssystems ist die Tatsache, daß I/O-Operationen (mit Ausnahme serieller Kanäle) nicht im Hintergrund durchgeführt werden, obwohl die Hardware dafür ausgelegt ist. Wer hat nicht schon minutenlang in die Röhre gesehen, während der Rechner einen Ordner mit 100 Dateien von Laufwerk A: nach Laufwerk B: kopiert hat, obwohl dabei nur eine effektive Prozessorleistung von gerundet 0% benötigt wird? Indes ist bei Floppy-Operationen auch kaum Abhilfe möglich, da die einzige

Instanz des Betriebssystems, die (etwa beim Dateizugriff) begrenzt in die Zukunft blicken könnte, nämlich das DOS, eine abgeschlossene Einheit bildet.[1] Außerdem bräuchte man zusätzliche Kommunikationsmittel, damit die Direktprogrammierung der Hardware, wie sie von fast allen Kopier- und Formatierprogrammen angewandt wird, nicht zum Absturz führt.

Dagegen kann der Datentransfer über die Parallelschnittstelle relativ einfach in den Hintergrund gedrängt werden. Das genau ist die Aufgabe von SPULER V4.5. Eine weitere Aufgabe von SPULER V4.5 ist es, auf einfache Art und Weise aus GEM-Anwendungen heraus die Spulfunktion überwachen und kontrollieren zu können. Ferner ist seit Version 4.4 ein Filespuler integriert. Das Hauptaugenmerk dabei liegt darauf, daß all dieses auf möglichst saubere Art und Weise geschieht, also möglichst betriebssystemkonform und mit geringstmöglichen Einbußen bei der allgemeine Performance.

Übrigens ist SPULER V4.5 kein völlig neues Programm, sondern seit 1988 mehr oder weniger kontinuierlich gewachsen. Die Version 3.0 ist in der c't 6/90 abgedruckt worden.

3.2 Funktionsweise des Spoolers

Bei der Darstellung des Spulprozesses muß man seit Version 4.4 terminologisch aufpassen, da es jetzt zwei verschiedene Mechanismen gibt, die in der Umgangssprache beide „Spooler" genannt werden. Nennen wir den einen „Interruptspuler" und den anderen „Filespuler". Dabei ist der Interruptspuler der wichtigere der beiden. Er ist auch historisch der Kern von SPULER V4.5.

Der Interruptspuler legt einen ausreichend großen Datenpuffer an und biegt alle relevanten BIOS-Ausgabevektoren um. Soll nun fürderhin ein Zeichen ausgegeben werden, so wandert es in den Puffer. Danach steht der Pozessor sofort für weitere Aufgaben zur Verfügung. Der Spulprozeß läuft selbsttätig im Interrupt: Immer dann, wenn der Drucker die Busyleitung zurücksetzt, wird ein Interrupt der Priorität 6 ausgelöst. Dieser Interrupt gibt dann das nächste Zeichen aus. Die Ausgabefrequenz kann man übrigens nicht vorhersehen, sie wird vollständig vom Drucker gesteuert und ist daher maximal. Sie interessiert aber auch nicht, da der Prozeß ja im Hintergrund abläuft (einen hinreichend großen Puffer vorausgesetzt).

Obige Darstellung ist natürlich etwas vereinfacht, so müssen etwa die Fälle eines vollen oder leeren Puffers sowie Eingriffe des Steuerprogramms berücksichtigt werden.

Was macht nun der Filespuler? Er ist im Steuerprogramm angesiedelt (das auch als Accessory benutzt werden kann) und muß explizit angestoßen werden. Man gibt ihm einen Dateinamen und er versucht dann, diese Datei in gleichmäßigen Abständen häppchenweise auf den Drucker auszugeben. Dazu sieht er regelmäßig nach, ob genug Platz im Puffer für ein weiteres Häppchen ist. Dank des Interruptspulers kommen beim Drucker trotzdem keine Datenhäppchen an, sonder ein kontinuierlicher Datenstrom, der darüberhinaus auch noch maximale Datenrate hat.

[1] Seit dem Erscheinen von MagiC 3 hat sich dieser Sachverhalt allerdings geändert.

3.3 Installation

Um normale Druckerausgaben im Hintergrund durchzuführen, reicht es vollständig, das TSR zu installieren. Am einfachsten kopiert man zu diesem Behufe die Datei `SP45_TSR.PRG` in den Auto-Ordner. Für den Interruptspulbetrieb selbst wird das Steuerprogramm nicht benötigt. Das Steuerprogramm `SPULER45.APP` erlaubt es einem, den Füllstand des Druckpuffers zu kontrollieren, den Puffer zu löschen, und Dateien im Hintergrund zu drucken. Wenn man das Steuerprogramm `SPULER45.APP` in `SPULER45.ACC` umbenennt, kann man es als gewöhnliches Accessory betreiben. Das ist letztlich nur bei einem Singletasking-TOS sinnvoll. Unter MagiC und MultiTOS funktioniert es aber auch. Der Hauptunterschied besteht darin, daß das Accessory auch bei geschlossenem Fenster weiterarbeitet, während die Applikation sich beim Schließen des Fensters beendet. (Sollte der Filespuler noch am Werken sein, bleibt die Applikation allerdings so lange resident, bis die Datei ausgedruckt ist.)

3.4 Konfiguration

Um das TSR zu konfigurieren, startet man die Datei `SP45_CFG.PRG`. Zunächst erfragt das Programm den Namen des TSR mit Hilfe der Fileselectorbox und testet die angegebene Datei. Als nächstes erscheint eine Dialogbox, in die man die gewünschte Puffergröße eintragen kann. Man kann ferner zwei Verzögerungswerte angeben, deren Funktion ich im folgenden Absatz kurz erklären möchte. Wer mit so technischem Zeugs nix am Hut hat, kann beim übernächsten Absatz weiterlesen.

Es hat sich nämlich gezeigt, daß der Interruptspuler, der von mir auf höchste Effizienz getrimmt wurde, bei schnellen Beschleunigerkarten Probleme hat. Befindet sich die ganze Routine zum Beispiel im prozessorinternen Cache eines mit 48 MHz getakteten 68030, erscheinen auf dem Bus nur noch die Buszugriffe auf die I/O-Bausteine, und zwar unmittelbar hintereinander. Da der Soundchip eine nur schwache Treiberleistung hat, folgen die Spannungspegel in den Leitungen nicht schnell genug – vor allem bei längeren Kabeln. Deswegen sind jetzt zwei Verzögerungsschleifen eingebaut: die erste verzögert die Aktivierung des Strobe, nachdem die Daten schon angelegt wurden, und die zweite verzögert die Deaktivierung des Strobe. Im Inneren jeder Verzögerungsschleife findet ein Lesezugriff auf den I/O-Bereich statt, der von keiner Beschleunigungshardware verkürzt wird.

Wenn im Ausdruck falsche Zeichen ankommen, sollte der erste Wert erhöht werden. Werden Zeichen verschluckt, muß der zweite Wert erhöht werden. Vermutlich ist es auch sinnvoll, den zweiten Wert nicht kleiner als den ersten zu wählen. In den meisten Fällen wird man die Werte aber einfach auf 0 stehen lassen können, das funktioniert bei mir (68030 mit 48 MHz) problemlos.

3.5 Der Normalbetrieb

Aktiviert man das Steuerprogramm, so öffnet sich ein Fenster, aus dem sich der aktuelle Füllstand des Puffers und seine Größe ergeben. Zu diesem Zweck wird ein Balken gezeichnet, der bei sich änderndem Füllstand aktualisiert wird.

Hatte man beim Start zusätzlich die linke Shifttaste gedrückt, so erscheint stattdessen eine Alertbox, die fragt, ob der Puffer gelöscht werden soll. Zeitgleich mit dem Erscheinen bewußter Alertbox ist auch die Zeichenausgabe im Interrupt gestoppt worden. Man kann dann in Ruhe entscheiden, ob man löscht oder weiterdruckt. Beim Löschen wird nicht nur der Puffer des Interruptspulers gelöscht, sondern auch der Filspuler abgebrochen. Eine andere Möglichkeit, den Löschdialog zu starten, besteht darin, bei gedrückter linker Shifttaste in das offene Fenster zu klicken.

Hält man beim Aktivieren des Steuerprogramms oder Klicken ins Fenster die Controltaste gedrückt, erscheint statt alledem der Fileselektor. Jetzt kann man eine Datei auswählen, die vom Filespuler ausgedruckt werden soll.

Die Bedienung läßt sich also auf den Merksatz bringen: „Keine Taste – Fenster, Linke Shifttaste – Löschdialog, Controltaste – Filespuler“.

3.6 Zusammenarbeit mit GEMINI

GEMINI benutzt seit Version 1.2 das sogenannte AV-Protokoll zur Kommunikation mit Accessories und Applikationen. Dieses ist auch in SPULER V4.5 implementiert.

Für den Anwender heißt das, daß nicht nur `SPULER45.APP` durch Doppelklick auf das Dateiicon gestartet werden kann, sondern auch `SPULER45.ACC` – falls das Accessory installiert ist. Bezüglich der Shift- und der Controltaste gilt das oben Gesagte. Zieht man ein Dateiicon auf das Spulericon oder das Spulerfenster, wird der Filespuler in Gang gesetzt, um die Datei auszudrucken.

Sämtliche Tastendrücke werden an GEMINI weitergereicht. Die Fensterfunktionen von GEMINI beziehen sich auch auf das Spulerfenster. Zwischen einem GEMINI-Fenster und dem Spulerfenster sollte kein Unterschied feststellbar sein.

In früheren Versionen des Spulers wurde stets der Zustand des Fensters in `GEMINI.INF` abgespeichert, und auch nach der Rückehr aus einer Applikation wurde der alte Zustand wiederhergestellt. Dies wird seit Version 4.5 nicht mehr gemacht.

3.7 Zusammenarbeit mit MagiC und MultiTOS

`SPULER45.APP` prüft stets anhand der Funktion appl_getinfo, welche Möglichkeiten das aktuelle Betriebssystem bietet und sollte daher mit MultiTOS und MagiC zurecht kommen.

Zur Zeit besteht die spezielle Anpassung darin, daß

- es als Accessory auf den act_pd-Hack verzichtet
- der Filespuler bei gesperrtem Bildschirm weiterläuft
- es sich bei Empfang von AP_TERM beendet.

Für spätere Versionen ist evtl. die Implementierung der Protokolle DRAG&DROP und ICONIFY geplant.

Abbildung 1: Zustandsdiagramm von SPULER v4.5

Als Druckprogramm unter `MAGXDESK` ist `SPULER45.APP` übrigens jetzt schon hervorragend geeignet, da `MAGXDESK` das AV-Protokoll verwendet.

3.8 Technisches

Auf einer gewissen Abstraktionsebene ist SPULER v4.5 (wie eigentlich alle eventorientierten Programme) einfach beschreibbar (ohne das jetzt näher zu definieren) durch einen endlichen Automaten. So etwas ähnliches (nennen wir es Zustandsdiagramm) stellt Abbildung 1 dar. Ein Bild sagt halt doch mehr als tausend Worte. Wie aus der Abbildung ersichtlich ist, besteht SPULER v4.5 im wesentlichen aus zwei zentralen Zuständen, die im Programmcode tatsächlich durch Eventschleifen repräsentiert sind. Die kleinere von beiden wartet eigentlich nur auf die Aktivierung des Fensters. Daher frißt SPULER v4.5 praktisch keine AES-Zeit, wenn das Fenster zu ist. Die größere Schleife schaut alle 0.2s nach, ob der Füllstand des Puffers sich verändert hat. Dank eines effizienten Bindings für evnt_multi() wird aber auch dort keine Rechenzeit verdorben.

Etwas anders sieht das Bild aus, wenn der Filespuler aktiv ist. Dieser liest im Timerevent gegebenenfalls 4KB von Platte ein und schiebt sie gleich in den Puffer des Interruptspulers. Ursprünglich wollte ich die Daten über das BIOS ausgeben, um so eventuelle Systemerweiterungen des Users (Filter usw.) nicht zu umgehen. Bei einem schnellen 68030 geht das durchaus noch, aber ein 68000 mit 8 MHz wird durch einige tausend BIOS-Calls pro Sekunde fast bis zum Stillstand gebremst. Damit der Filespuler nicht direkt in den Puffer des Interruptspulers schreiben muß, habe ich die Softwareschnittstelle des TSR um eine weitere Funktion bereichert (s.u.). Insgesamt ist der Filespuler dadurch so schnell geworden, daß auch auf einem ST mit 8 MHz keine störende Verlangsamung beim Arbeiten feststellbar ist. Trotzdem wird die Datei verzugslos und mit maximaler Druckergeschwindigkeit ausgedruckt.

4 Bugs oder Features

- Der Interruptspuler bedient sich der xcon-Vektoren des BIOS. Diese gibt es jedoch erst seit TOS 1.02, dem Blitter-TOS. Die Alternative, den Trap umzubiegen, halte ich für eine Zumutung. Wer heute noch TOS 1.0 benutzt, wird vermutlich für ein Betriebssystemkonzept wie I/O-Transfer im Hintergrund ohnehin nichts übrig haben.
- Ist der Drucker ausgeschaltet oder offline, so können beim normalen Ausdrucken folgende zwei Fälle auftreten. Falls die Druckdaten weniger als die Pufferlänge sind, so wandern sie vollständig in den Puffer. Sobald man den Drucker online schaltet, wird alles ausgedruckt. Sind es dagegen mehr Daten, als in den Puffer passen, so wird erst einmal der Puffer gefüllt. Bei den folgenden Zeichen verhält sich die Ausgaberoutine dann wie die originale BIOS-Routine. Erst wird 10s gewartet, danach ein Fehlerflag gesetzt und alle nachfolgenden Zeichenausgaben mit Fehler abgebrochen. (Erst eine fünfsekündige Druckpause macht das Flag wieder unwirksam). Der Puffer wird aber nicht gelöscht, das bleibt dem Anwender überlassen.
- Da mir keine Möglichkeit bekannt ist, an die Druckerkonfiguration des XBIOS heranzukommen (ein Trap bei jeder Zeichenausgabe ist indiskutabel), wird ein serieller Drucker nicht mehr unterstützt. Das ist bei meinem TEX-Druckertreiber allerdings ein Feature: Durch Ausgabe auf einen seriellen Drucker zwinge ich ihn, das BIOS überhaupt zu benutzen. Sonst gibt er alles direkt auf den Port, wohl in der Einbildung, beim Warten auf den Drucker weniger Zeit zu verschwenden. (Ein ähnlich kläglicher Versuch war auch 'mal in c't 10/91, natürlich ohne jeden Benchmark.)
- Aus naheliegenden Gründen sollte der Puffer leer sein, bevor man ein Programm startet, das den Port direkt programmiert.
- Es werden auch die Vektoren von Bconstat() und Bconin() für den Drucker umgebogen: Sie geben stets „Kein Zeichen verfügbar“ bzw. CTRL-Z zurück. Damit ist jetzt auch bei TOS 1.02 die GEMDOS-I/O-Umlenkung auf den Drucker möglich.
- Wünschenswert wäre natürlich eine dynamische Speicheranforderung zur Laufzeit, d.h. daß Pufferspeicher in kleinen Stücken genau dann vom Betriebssystem angefordert wird, wenn er gebraucht wird und auch stückweise zurückgegeben wird. Das ist jedoch mit den Mitteln des GEMDOS für speicherresidente Programme nicht machbar. An eine Lösung mit Hilfe der AES (durch Auswertung von AC_CLOSE) will ich nicht einmal denken, da der Spulprozeß selbst *unabhängig* vom GEM sein soll, TOS-Programme dann Schwierigkeiten bereiten und es außerdem mangels Handshake Timingprobleme mit den AES-Messages gibt. Ein sicherer Betrieb wäre Zufall.
- Noch lieber hätte ich einen Mechanismus, der Teile des Puffers auf die Platte auslagert und bei Bedarf wieder lädt. Wollte man dazu das normale Filesystem benutzen, müßten GEMDOS-Aufrufe aus dem Interrupt heraus gemacht werden, was Atari sei Dank unmöglich ist. Auch einen pollenden AES-Prozeß zu benutzen, ist keine gute Idee: ein TOS-Programm oder ein schlecht programmiertes GEM-Programm könnte einen Deadlock erzeugen. Fazit: mit den Mitteln des Betriebssystems ist das ganze nicht sauber machbar.

Mir ist zwar bekannt, daß es Spooler gibt, die BIOS-Traps abfangen und die Druckdaten mit einem Trick in eine Datei schreiben, aber so etwas ist in meinen Augen ein Hack und entspricht nicht meiner Vorstellung von sauberem Programmieren.

- Was ich noch machen wollte, ist ein CPX zur Spulersteuerung. Allerdings wäre damit vermutlich kein Filespuler möglich. Falls sich aber jemand daran macht (die Softwareschnittstelle des TSR ist ja dokumentiert), bitte ich um Nachricht.

- Wo wir gerade bei der Softwareschnittstelle sind: Das Konzept, über einen Cookiepointer aus dem Speicher eines anderen Prozesses zu lesen bzw. dessen Code auszuführen, läuft natürlich der Memory Protection von MiNT zuwider. Hier hilft nur, den Speicher des TSR als global zu kennzeichnen, oder aber das Steuerprogramm nicht zu benutzen.

- Während der Filespuler läuft, wird die Bconout-Schnittstelle *nicht* gesperrt. Ich erlaube es einfach einem AES-Prozeß nicht, in das BIOS einzugreifen, denn das BIOS soll nach wie vor transparent und autonom funktionieren. Das Konzept des Lockings von Systemressourcen gibt es halt beim TOS nicht.

  Es liegt in der Verantwortung des Benutzers, nur einen Prozeß zur Zeit drucken zu lassen. Das Schlimmste, was andernfalls passiert, ist die Vermischung der verschiedenen Druckdaten.

- Auch zum ewigen Thema GEMDOS ist etwas zu sagen: Es scheint so zu sein, daß die Filehandles des GEMDOS für den Rechner global sind, es also keine Benutzerfiledeskriptorentabelle (für jeden Prozeß) gibt. Jeder Prozeß kann auf jedes Handle zugreifen, wenn er es nur kennt. Trotzdem wird der Prozeß vermerkt, der die Datei geöffnet hat, damit bei dessen Ende die Datei automatisch geschlossen wird. Da nun ein Accessory im SingleTOS stets unter dem aktuellen GEMDOS-Prozeß läuft, gibt es Probleme, wenn eine Datei über einen AES-Aufruf hinweg offen bleiben soll — der beim Öffnen aktive GEMDOS-Prozeß könnte ja zwischenzeitlich zuende sein und das Filehandle weg oder gar neu vergeben. Zur Lösung dieses Problems könnte man jeden Dateizugriff in Fopen/Fclose schachteln, was aber ineffizient ist. Ich habe mich daher schweren Herzens entschlossen, (im Falle von Accessory & SingleTOS) den bekannten aber unschönen Hack zu benutzen, dem GEMDOS beim Öffnen des zu spulenden Files einen anderen Prozeß (eben den Spuler) vorzuspiegeln. Lesen und Schließen geschieht dann wieder unter dem gerade aktuellen Prozeß.

- Ich sollte noch ein paar Worte zur Konzeption des Filespulers verlieren. Es war *nicht* geplant, den Filespuler als zentralen Druckmanager oder als Druckerdämon zu entwerfen. Ein solches Konzept wäre zum einen dem TOS etwas fremd, zum anderen wäre es ein extrem aufwendiges Projekt, das die Kontrolle über den gesamten Druckbetrieb übernehmen müßte und vielfältige Konfigurationsmöglichkeiten bräuchte — man denke an Druckerinitialisiserung, Formfeed, Fehlerbehandlung, usw.

  SPULERsollte dagegen schon immer ein kleines und effizientes System sein. In Version 4.5 belegen TSR und APP im Speicher (!) 17 KB zuzüglich Puffer. Unter MagiC reichen 8 KB Puffer völlig aus. Auch die Belastung für das Restsystem ist minimal: Es werden keine Traps abgefangen, das TSR ist optimal in Assembler codiert und über die geringe AES-Belastung ist weiter oben schon geschrieben worden. Trotzdem könnte

Spuler v4.5 von, sagen wir, einem Dämon benutzt werden, der dem Steuerprogramm VA_START-Messages schickt.

Kleinere Druckjobs sollten nach Möglichkeit wie gewöhnlich abgewickelt werden. Der Interruptspuler sorgt schon für das Verschwinden eventueller Denkpausen. Erst bei größeren Jobs wird die Benutzung des Filespulers sinnvoll, wenn man gleichzeitig auf dem Rechner noch etwas anderes zu tun hat. Unter SingleTOS muß es sich dabei aber um GEM-Applikationen handeln, da sonst der Filespuler nicht parallel dazu arbeiten kann. Während ich zum Beispiel diesen Text hier in einem GEM-Editor tippe, druckt die Nadelsäge neben mir ein über 200-seitiges Postscriptdokument aus.

Um den Filespuler überhaupt benutzen zu können, müssen die Druckdaten als Datei vorliegen. Dazu muß man das druckende Programm so einstellen, daß es in eine Datei druckt. Geht das nicht, so kann man es mit der Ausgabeumlenkung des GEMDOS versuchen. Programme, die stets über BIOS oder – schlimmer – durch Direktprogrammierung der Hardware drucken, können den Filespuler nicht nutzen. Glücklicherweise gehören Ghostscript und die meisten DVI-Treiber nicht in diese Kategorie.

Eine A4-Seite verschlingt bei 360∗360 dpi übrigens bis zu einem Megabyte auf der Platte. Man braucht für das Filespulen von Vollgrafik also gehörige Reserven.

- Unter MagiC wird bei der BIOS-Ausgabe und vollem Puffer immer noch eine Warteschleife (bis zum Timeout) durchlaufen. Bis ich die Programmierdoku verstehe, bleibt das auch erst mal so.

5 Was ist neu

5.1 in Version 4.0

- Fast der gesamter Code ist neu. Lediglich die Interrupt- und BIOS-Routinen sind (leicht überarbeitet) von Version 3 erhalten geblieben. Die gesamte Accessoryverwaltung ist nun in C geschrieben.
- Der Timeout wurde von 30s auf 10s verkürzt.
- Die BIOS-Vektoren 0x506 und 0x50A für die Zeichenausgabe der internen Hardcopyroutine werden nun umgebogen.
- Die Vektorverbiegung entspricht jetzt dem XBRA-Standard. Die XBRA-Kennung von Spuler v4.0 ist „spMR“.
- Der Füllstand wird jetzt intuitiv erfaßbar als Balkengrafik dargestellt. Außerdem läuft die Anzeige auch während der Ausgabe.
- Das AV-Protokoll wurde implementiert.
- Bei Start als Hauptapplikation konfiguriert sich Spuler v4.0 selbst.

5.2 in Version 4.1

- Leider mußte ich nach der Freigabe von SPULER V4.0 noch einen ärgerlichen Fehler feststellen: In der Schleife zur Bearbeitung der Rechtecklisten wurden die falschen Variablen getestet, so daß zuviele wind_get(WF_NEXTXYWH, ...) getätigt wurden. Der Fehler fällt nicht auf, weil nach endlich vielen Schritten die AES immer so frei sind, als x-Koordinate eine 0 zu liefern, zumindest bei meinem Rechner. Da jedoch TOS das erste nichtdeterministische Betriebssystem ist, kann man sich wohl kaum darauf verlassen.
- Die Fensterkoordinaten werden nun als Relativgrößen gespeichert.

5.3 in Version 4.2

- Der Spuler ist jetzt zweigeteilt in ein TSR und ein Accessory. Die Kommunikation geschieht über den cookie jar. Dazu legt das TSR ein cookie mit Namen „spMR“ an. Der zugehörige Parameter ist ein Pointer auf folgende Struktur (in C-Syntax):

```
struct
{   long length;            /* Pufferlänge */
    long lobuf;             /* Startadresse des Puffers */
    long hibuf;             /* Endadresse des Puffers */
    long inmark;            /* Eingabepointer */
    long outmark;           /* Ausgabepointer */
    void (*stop)();         /* Stoppt Hintergrundprozeß */
    void (*restart)();      /* Neustart desselben */
}
```

  Diese (hiermit dokumentierte) Struktur ist nur gültig, wenn length!=0 ist. Dadurch bleibt die Definition offen für zukünftige Entwicklungen. Der Puffer ist genau dann leer, wenn inmark==outmark ist. Allgemein liefert das Programmstück

```
l=inmark-outmark;
if (l<0) l+=length;
```

  den aktuellen Füllstand des Puffers in der Variablen l. Die übrigen Einträge der Struktur tun genau das, was man anhand der Kommentarzeile von ihnen erwarten würde. Jede andere Eigenschaft des TSR gelte als undokumentiert.

- Der Non-Autovektor-Interrupt-Pointer des MFP für das Busy-Signal an 0x100 wird nun einfach überschrieben und nicht mehr in einer XBRA-Struktur vor der neuen Interruptroutine gerettet. Das war nötig, weil das TOS diesen Pointer bei einem Reset nicht zurücksetzt und dann ein Zykel in der XBRA-Kette entsteht (bei gleicher Systemkonfiguration). Das hinwiederum sägt einige der vektorüberwachenden Programme ab.
- Der Busy-Interrupt wird nun stets initialisiert, wenn ein Byte in den leeren Puffer übertragen wird. Das war nötig, da z.B. einige TEX-Druckertreiber einen eigenen Spuler einrichten. Das ist an sich nicht schlimm, nur ist nach dem Ende dieser Programme der Busy-Interrupt disabled und der Interruptvektor überschrieben.

- Die Messages des AV-Protokolls werden nun nicht mehr einfach an ap_id 0 geschickt. Stattdessen wird bei jeder Einstellung des Minimalprotokolls versucht, die Applikationen AVSERVER oder GEMINI zu finden. Nur wenn das fehlschlägt, wird weiterhin ap_id 0 verwendet.

5.4 in Version 4.3

- Im wesentlichen wurde nur das TSR überarbeitet. Ich wollte eigentlich nur testen, ob ein Burst-Fill in der Interruptroutine sinnvoll ist. Zumindest für meinen NEC P6 ist es das aber nicht. Stattdessen gelang es mir, die Interruptbehandlung um über 10 Mikrosekunden (bei einem Standard-ST) zu verkürzen.
- Es wird jetzt bei jedem Bconout geprüft, ob der MFP-Interrupt noch korrekt initialisiert ist. Falls nicht, wird kurzerhand alles neu initialisiert. In Version 4.2 geschah das nur bei leerem Puffer.
- Das TSR gibt jetzt eine Einschaltmeldung aus.
- Das ACC kann jetzt im Zeitalter des Multitasking auch als PRG gestartet werden.
- Dafür ist jetzt ein eigenständiges Konfigurationsprogramm beigelegt.

5.5 in Version 4.4

Es hat eine mehr oder weniger drastische Überarbeitung aller Programmteile gegeben.

- Beim TSR von Version 4.3 hatte ich schlauerweise ein bset mit einem bclr verwechselt. Das führte dazu, daß die stop-Routine ihrem Namen nicht gerecht wurde.
- In die Ausgaberoutine wurden zwei Verzögerungsschleifen eingebaut, die mit dem Konfigurationsprogramm konfiguriert werden können. Die Funktion dieser Schleifen wurde oben schon beschrieben. Damit sollten jetzt alle Probleme mit schnellen Beschleunigerkarten der Vergangenheit angehören.
- Es ist eine weitere aufrufbare Funktion implementiert worden. Sie hat (in C-Notation) den Prototypen

  ```
  long write(long count, char *buffer)
  ```

  und funktioniert ähnlich wie Fwrite: Es wird versucht count viele Zeichen von buffer an in den Puffer des Interruptspulers zu übertragen. Zurückgegeben wird die Anzahl der tatsächlich übertragenen Zeichen. Das können durchaus weniger als count sein, wenn nicht genug Platz im Puffer ist. Auf keinen Fall wartet write auf das Freiwerden von Platz. Wie bei Bconout wird geprüft, ob der Spulerinterrupt noch korrekt initialisiert ist, und der Spulprozeß wird nötigenfalls angestoßen.

  Die Parameterübergabe erfolgt gemäß der Pure-C-Konvention: count in D0, buffer in A0, Rückgabewert in D0.
- Um auf die neuen Elemente zugreifen zu können, wurde die Softwareschnittstelle erweitert. Der Parameter im Cookie „spMR“ zeigt nun auf folgende Struktur (int habe 16 bit):

```
struct
{   long length;                /* Pufferlänge */
    long lobuf;                 /* Startadresse des Puffers */
    long hibuf;                 /* Endadresse des Puffers */
    long inmark;                /* Eingabepointer */
    long outmark;               /* Ausgabepointer */
    void (*stop)(void);         /* Stoppt Hintergrundprozeß */
    void (*restart)(void);      /* Neustart desselben */
    long (*write)(long, char*); /* Schreibt in den Puffer */
    int  delay1;                /* Verzögerung vor Strobe */
    int  delay2;                /* Verzögerung nach Strobe */
}
```

Offenbar ist diese Struktur aufwärtskompatibel zu der alten. Für die ersten sieben Einträge gilt das unter Version 4.2 Gesagte.

- Das Konfigurationsprogramm hat nun eine externe RSC-Datei, was die Wartung wesentlich vereinfacht. Das ist von der Handhabung her kein Nachteil, denn schließlich wird das Konfigurationsprogramm nur selten benutzt.
- Bevor das Konfigurationsprogramm seinen Dialog öffnet, fragt es bereits nach dem TSR, um die aktuellen Werte in die Dialogbox zu übernehmen.
- Die herausragende Neuerung im ACC bzw. PRG ist der Filespuler. Die gesamte Dokumentation wurde entsprechend umgearbeitet, so daß an dieser Stelle nichts mehr zu sagen bleibt.

5.6 in Version 4.5

SPULER V4.5 wurde im wesentlichen nur an MagiC (und damit an MultiTOS) angepaßt:

- Die Distribution enthält jetzt das APP anstelle des ACC, was unter MagiC sinnvoller ist. Das ist halt die neue Konfiguration des Autors. Am besten trägt man das Steuerprogramm bei Magxdesk unter Optionen–Programme ein.
- Unter MagiC (oder MultiTOS) wird der Filespuler nicht mehr von wind_update blockiert. Das hat den sehr angenehmen Seiteneffekt, daß man für das Filespulen den Puffer des TSRs nun sehr klein wählen kann. Bereits 8 KB reichen völlig aus, den Drucker immer mit Stoff zu versorgen. Für die BIOS-Schnittstelle braucht man allerdings wie bisher einen großen Puffer.
- Auch als Accessory verzichtet der Filespuler auf den oben erwähnten act_pd-Hack, wenn MagiC (oder MultiTOS) aktiv ist.
- Das Steuerprogramm wurde um diejenigen Funktionen des AV-Protokolls erleichtert, die Gemini stets über die Lage des Fensters auf dem Laufenden halten. Das ist letztlich eine Spielerei, die 1.3 Kilobytes kostet. Das sei nicht viel? Nun, es drückt die Dateilänge unter 8192 und spart unter Umständen einen großen Plattensektor. ;-) Durch ein einfaches Compilerflag kann der Support aber jederzeit wieder eincompiliert werden.

- Das Steuerprogramm prüft jetzt zyklisch nach, ob der MFP-Interrupt noch korrekt initialisiert ist. In einer Multitasking-Umgebung ist es einfach zu wahrscheinlich, einmal auf ein Programm zu stoßen, das selber spulen möchte.[2] Dazu wurde die Softwareschnittstelle des TSRs um einen neuen Eintrag erweitert:

```
struct
{   long length;                /* Pufferlänge */
    long lobuf;                 /* Startadresse des Puffers */
    long hibuf;                 /* Endadresse des Puffers */
    long inmark;                /* Eingabepointer */
    long outmark;               /* Ausgabepointer */
    void (*stop)(void);         /* Stoppt Hintergrundprozeß */
    void (*restart)(void);      /* Neustart desselben */
    long (*write)(long, char*); /* Schreibt in den Puffer */
    int  delay1;                /* Verzögerung vor Strobe */
    int  delay2;                /* Verzögerung nach Strobe */
    void (*check)(void);        /* Test und evtl. Neustart */
}
```

- Die Blocklänge des Filespulers wurde auf 4096 erhöht, was beim Autor gerade einem Plattensektor entspricht und dadurch unter Umständen die Performance erhöht. Zum Ausgleich wurden die Zeitintervalle für evnt_timer auf 200 ms verlängert.

  Man kann als Programmierer `:-)` mit diesen Parametern in weiten Grenzen spielen. Es sollten nur die Daten mit einer höheren Rate in den Puffer hineinkommen als der Drucker sie herausholt. Auf langsamen Rechnern sollten die evnt_timer-Intervalle nicht zu kurz sein, damit die Gesamtperformance nicht leidet. Umgekehrt sind zu große Blocklängen unter Singletaskingsystemen zu vermeiden, da der Rechner während der Plattenzugriffe blockiert ist. Das gilt übrigens auch für MultiTOS. Unter MagiC mit Hintergrund-DMA ist es dagegen völlig egal. Hier sind große Blöcke sogar vorzuziehen, da DMA immer billiger ist als Taskswitching.

Literatur

[1] Jankowski, Rabich, Reschke: ATARI ST Profibuch, Sybex, 1988

[2] Amler, Zobel: Einer für alles, c't 9/88, Seite 144

[3] Rogge: Von der Spule, c't 6/90, Seite 212

[2] Im Löschdialog (Button „zurück“) konnte man den Interrupt schon immer manuell neu Installieren.