## Contents

# This Week With My Coleco ADAM 0009.18

by Richard F. Drushel

## I. Address Book Patch for ADAMserve.

In TWWMCA 0008.20 I described a 3-byte binary patch for SmartFiler, to enable it to function correctly under ADAMserve. (Prior to this patch, SmartFiler would hang ADAMserve when trying to read an existing database, necessitating a hard reboot of the ADAM client to clear the error.) The Address Book program, a phone number database for use with the Autodialer sideport expansion module (it uses pulse dialing), also hangs under ADAMserve. Since both SmartFiler and Address Book have similar database interfaces (and indeed, SmartFiler can read Address Book databases), I hypothesized that both programs shared the same core database code, and thus might share the same bug.

I'm happy to report that my hypothesis is correct: Address Book and SmartFiler share large blocks of identical code (though assembled at different addresses), and the same kind of patch fixes both of them under ADAMserve.

In brief, the "bug" is a single `CALL _START_RD_1_BLOCK` instruction which is used to force a rewind of the program tape if you bought the software on tape. ADAMserve's imperfect emulation of ADAMnet devices does not work at the low level of `_START_RD_1_BLOCK`, so instead of routing the read request to the ADAMserve serial line, it tries to read from the real ADAMnet device for tape 2. Naturally, nothing is there, and the application program does not have any error handler, so the program hangs. The "patch" is to replace the 3-byte `CALL` with 3 `NOP` (No Operation) instructions. The only downside to this patch is that the patched software, if run from a real tape drive, will not automatically rewind to the beginning after I/O is complete, so the next time the program tries to read the directory, you have to wait for it to rewind back to block 1.

Here are detailed instructions for patching Address Book under ADAMserve. Note that your Volume 0 might be organized slightly differently than mine; so be sure to verify the start block of the Address Book application *before* you make any changes.

1. run ADAMserve File Manager
2. Edit Catalog to find the entry for ADDRESS_BK
3. on my system, it starts at block 585; yours might vary
4. go to Media Functions/Edit Blocks
5. select the *1st* block of ADDRESS_BK (on my system, it's 585)
6. go to sector 6 (by hitting `HOME+down arrow` to page through the sectors)
7. byte offset 715 decimal, you should see the following 3 bytes in hex: CD A2 FC
8. change these 3 bytes to: 00 00 00
9. hit return and save the changes
10. `Shift-Undo` to return to HARDDISK
11. boot SmartFiler as usual from hard disk

I further hypothesize that the same kind of patch will be necessary for Recipe Filer. I will investigate this some time soon.

## II. SmartFiler Patch to Enable Database Creation Under ADAMserve/HARDDISK.

One annoying feature of SmartFiler, as it had been patched to run from the EOS partition of hard drive systems, was that it was impossible to create new databases; you could only read existing databases. I had always assumed that this inability was some kind of "safety" feature, since SmartFiler insists that the database occupy the entire disk or tape, and in the case of a hard drive system, this would mean an entire 1K volume. Since the hard drive version of SmartFiler was put into Volume 0, and this is the boot volume, inadvertently destroying it would be a major disaster. ADAMserve, being based upon the standard EOS hard drive organization, also shared this limitation.

Since the EOS hard drive under ADAMserve is just a 10MB MS-DOS file, however, it's easy to make a backup copy of a working hard drive image (I use PKZIP 2.04g), and then play around--if it gets destroyed, just unZIP the backup and start over. Looking through the directory listing of Volume 0, and comparing it to a directory listing of an original SmartFiler disk, I observed that Volume 0 was a missing file, TDS_MAP. (It's actually TDS_MA with P in the filetype byte). So, I copied it from the SmartFiler floppy to Volume 0 using File Manager, launched SmartFiler, and tried to create a new database on a blank, ADAM-formatted 160K floppy. It worked! I then tried to create a new database on `DRIVE B` (which is the device mapped to the ADAMserve hard drive). I got an error message which said that there were permanent files which could not be deleted to make room for a new database. Hooray, no chance to accidentally wipe out Volume 0, because there are a number of hidden and system files there. What about a disk/tape with existing user files? I tried another floppy with some SmartBASIC files on it, all A-type: SmartFiler told me that it would have to delete the files to create a new database, did I really want to? Again, a safety net to protect against accidental erasure!

So, I conclude that it's safe for hard drive systems to put the file TDS_MAP back into Volume 0. Maybe it was left out by accident, instead of as a precaution; I don't know.

Clearly TDS_MAP was some kind of code overlay. I was curious about what it does. I went back to my SmartFiler disassembly listing (still in progress) and traced out all the code which dealt with TDS_MAP. It turned out that all the SmartKey menus dealing with database creation had their executable code in an address space that was *outside* that of the 33K SMARTFILER file, and *inside* the address space where TDS_MAP gets loaded (A800H-BFFFH). Indeed, attempting to create a database causes SmartFiler to try to load TDS_MAP, and if it can't find the file, it aborts.

TDS_MAP is only 6K long, so it was pretty simple to generate a resolved disassembly listing using methods I have described previously (e.g., TWWMCA 9701.19) Aside from some garbage at the very end, TDS_MAP is mostly code with a few vector and data tables. It makes a few references to addresses in the SMARTFILER file address space, however, so any changes to SMARTFILER's memory map must be propagated through to TDS_MAP.

Address Book does not use an external overlay file (another possible reason why TDS_MAP might have been left out; what if both programs had an overlay named TDS_MAP and they were different binaries?), so after the 3-byte patch described above, it is able to successfully create new databases under ADAMserve.

## III. Next Time.

In order to trace out the SmartKey entry points into the TDS_MAP code, I had to figure out the data structures used to create SmartKey menus in SmartFiler. While somewhat complicated, this code shows one possible way to use SmartKey menus in Z80 assembly language programs for the ADAM. It might even be extracted as a single library module, to save future programmers the (serious) trouble of writing SmartKey menu-driven applications.

See you next week!

*Rich*

---

# This Week With My Coleco ADAM 0009.10

by Richard F. Drushel

## I. Finding a New Home for the *TWWMCA* Archive.

I used to archive past *TWWMCA* articles in an anonymous FTP directory at my ISP, APKnet. However, due to security concerns, APKnet shut off all anonymous FTP access in mid-August, 2000. (Some FTP clients can be cracked by the unscrupulous.) So, the *TWWMCA* archive needed to move.

I could have put the archive on one of the machines under my control here at CWRU, some of which run FTP servers. However, the lab machines tend to get moved around, or replaced, or given different IP addresses and hostnames. So, I didn't view these as very stable choices. APKnet sysadmins suggested that I make the archive available by HTTP (which is an "anonymous" protocol as well). Indeed, even binary files are often made available nowadays by HTTP. I could just move the files to my webpage space and make a simple index of links to each article. However, in for a penny, in for a pound: if they were going to be available via HTTP to web browsers, I thought that they should at least be reformatted to be webpages, written in HTML, instead of just plaintext dumps.

In addition, just after I posted *TWWMCA* 0008.13 to USENET, a reader of comp.os.cpm commented that perhaps the articles would reach more people if they were archived as HTML, or at least be more accessible. Why not, he suggested, post a pointer to new *TWWMCA* articles at the HTML archive, instead of posting the plaintext articles themselves?

The current subscribers to the Coleco ADAM mailing list will recall that I polled everyone to find out whether or not they wanted plaintext versions of *TWWMCA* to continue. The clear opinion was "whatever you do with the archive, keep posting them to the mailing list as plaintext". Since USENET readership is a secondary audience, I had to keep writing *TWWMCA* as plaintext.

## II. Converting *TWWMCA* Plaintext to HTML.

I thus decided that the archive should be HTML, but the original articles should continue to be plaintext. This meant that I needed to convert the plaintext articles into HTML.

To do the conversion by hand would be a very tedious job: there were 31 existing articles. Letting some word processor do an HTML export was not an option for me, either, as the HTML spewed out by word processors is ugly, verbose, and just *bad*. However, somehow automating the conversion was very desirable. What to do?

Fortunately for me, the plaintext *TWWMCA* articles have adhered very closely to a consistent layout format. All paragraphs are single-spaced, with a blank line in between them. Headings begin with Roman numerals, and there is always at least one blank line before and after each heading. Numbered lists begin with numerals in parentheses, like (1) and (2), and usually each item was on a separate line. Even the title header and author line had the same format in each article.

Thus, I decided that I could write a filter program (in QuickBASIC 4.5 for MS-DOS) which could read the original plaintext articles, add the necessary HTML header and footer, classify the text as title header, section header, paragraph, or numbered list, and add the appropriate HTML tags. Of course, there might be other manual editing to do (such as to add real links to URLs, or to show pictures instead of just having a reference to a picture file), but this would be a major drudgery saver.

I looked over a few of the original plaintext articles to see if there was anything important I would be missing with this simple, one-pass filter program. Three items became apparent:

1. Most of the numbered list items did not have blank lines between them, making it harder to determine where items ended.
2. Several articles had extensive sections of SmartBASIC or Z80 assembly code, whose formatting would be lost if blindly rendered as HTML (which ignores extra blank spaces in text, squeezing everything together).
3. The characters `<`, `>`, and `&` appeared in the text. These are reserved characters in HTML, to be used in HTML tags only. There are special HTML escape sequences which must be used to reproduce these characters as literals.

After a little thought, I came up with workarounds for these three cases:

1. Before running the filter program, manually add blank lines between numbered list items.
2. Manually add, on separate lines before and after blocks of code, the HTML tags `<PRE>` and `</PRE>`, and write the filter program so that it does not alter any text it encounters between a `<PRE>` and a `</PRE>` tag. `<PRE>` is used to specify pre-formatted text, which web browsers will render in a monospaced font with exactly the same spacing as the original text, with no space squeezouts.
3. Write the filter program so that it scans every input line of text for `<`, `>`, and `&`, and replace all occurrences *which are not in `<PRE>` blocks* with the special HTML escape sequences `&lt;`, `&gt;`, and `&amp;`, respectively.

The converter program, TXT2HTM4.RFD, is available for download:

- txt2htm4.rfd, text-to-HTML converter for *TWWMCA*, QuickBASIC 4.5

Note that the converter program also strips out leading and trailing spaces and `TAB` characters, *except* in `<PRE>` text blocks.

## III. Sample HTML Converter Input and Output Text.

Perhaps the best way to show how the automated conversion works is to give a "before" and "after" example. Here's a sample *TWWMCA*-format article in modified plaintext (i.e., with necessary blank lines and `<PRE>` and `</PRE>` tags added):

---

```
This Week With My Coleco ADAM  6211.03

by Richard F. Drushel  (drushel@apk.net)


I.  Administrivia.

    This is a test article for the TWWMCA text-to-HTML converter program
that I wrote.  This is only a test.  Note the appearance of HTML reserved
characters <, >, and &, which must be trapped.


II.  Sample Numbered List.

    Frequently TWWMCA has numbered lists, such as:

(1)  to list reasons why something will or won't work,

(2)  to show steps in debugging something, or

(3)  to give a list of programs or files.  Note that the filter program
     considers groups of single-spaced lines to be single entries in the
     numbered list.

Note that the begin and end tags for ordered lists, <OL> and </OL>, must
be added manually to the filtered output file.


III.  Sample Code Listing.

    It's necessary to allow code listings to retain their original spacing.
Otherwise, it becomes completely unintelligible to humans (although the
assemblers and compilers can still deal with them).

<PRE>
;Z80 assembly code fragment
;on entry, B=number of table entry
;on exit, A=table entry value

START:

    LD      HL,ADDR1          ;point to base of table
    LD      DE,1              ;length of table entry

LOOP:

    ADD     HL,DE             ;offset one entry
    DJNZ    LOOP              ;keep going until we reach Bth entry

    LD      A,(HL)            ;get the entry
    RET                       ;all done
```

```
</PRE>

    If the conversion works properly, it will look very nice.


IV.  Next Time.

    Who knows?  Something that hopefully won't bore you to tears.

    See you next week!

    *Rich*
```

---

Now here's what it looks like after the text-to-HTML converter program gets through with it (lines have been wrapped to 80 columns):

---

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<TITLE>This Week With My Coleco ADAM z</TITLE>
</HEAD>
<BODY>

<H1 ALIGN=CENTER>This Week With My Coleco ADAM  6211.03</H1>

<P ALIGN=CENTER>by <A HREF="mailto:drushel@apk.net">Richard F. Drushel
<I>(drushel@apk.net)</I></A></P>


<H2>I.  Administrivia.</H2>

<P>This is a test article for the TWWMCA text-to-HTML converter program
that I wrote.  This is only a test.  Note the appearance of HTML reserved
characters &lt;, &gt;, and &amp;, which must be trapped.
</P>


<H2>II.  Sample Numbered List.</H2>

<P>Frequently TWWMCA has numbered lists, such as:
</P>

<LI><P>to list reasons why something will or won't work,
</P></LI>

<LI><P>to show steps in debugging something, or
</P></LI>

<LI><P>to give a list of programs or files.  Note that the filter program
considers groups of single-spaced lines to be single entries in the
numbered list.
</P></LI>
```

```
<P>Note that the begin and end tags for ordered lists, &lt;OL&gt; and
&lt;/OL&gt;, must
be added manually to the filtered output file.
</P>


<H2>III.  Sample Code Listing.</H2>

<P>It's necessary to allow code listings to retain their original spacing.
Otherwise, it becomes completely unintelligible to humans (although the
assemblers and compilers can still deal with them).
</P>

<PRE>
;Z80 assembly code fragment
;on entry, B=number of table entry
;on exit, A=table entry value

START:

    LD     HL,ADDR1         ;point to base of table
    LD     DE,1             ;length of table entry

LOOP:

    ADD    HL,DE            ;offset one entry
    DJNZ   LOOP             ;keep going until we reach Bth entry

    LD     A,(HL)           ;get the entry
    RET                     ;all done
</PRE>

<P>If the conversion works properly, it will look very nice.
</P>


<H2>IV.  Next Time.</H2>

<P>Who knows?  Something that hopefully won't bore you to tears.
</P>

<P>See you next week!
</P>

<P>*Rich*
</P>


<HR>
<P><A HREF="wk.html">Next Article</A><BR>
<A HREF="wk.html">Previous Article</A><BR>
<A HREF="index.html"><I>TWWMCA</I> Archive Main Page</A></P>
<HR>
</BODY>
</HTML>
```

Now the manual tweaking begins:

- the *TWWMCA* date must be added to the `<TITLE>` text (a "`z`" is left as a placeholder).
- `<OL>` and `</OL>` tags must be added around the numbered list.
- all occurrences of the string "`TWWMCA`" must be replaced with the HTML "`<I>TWWMCA</I>`" to make it italics (since it's a title).
- the filenames for Next Article and Previous Article in the footer navigation bar must be added ("`wk.html`" is left as a placeholder).
- any other desired text formatting must be added (e.g., `<TT>` tags (teletype text, monospaced) for EOS function call names like `_READ_BLOCK`).

When these changes are made, the final HTML will be rendered as seen below:

---

# This Week With My Coleco ADAM 6211.03

by Richard F. Drushel

## I. Administrivia.

This is a test article for the *TWWMCA* text-to-HTML converter program that I wrote. This is only a test. Note the appearance of HTML reserved characters `<`, `>`, and `&`, which must be trapped.

## II. Sample Numbered List.

Frequently *TWWMCA* has numbered lists, such as:

1. to list reasons why something will or won't work,
2. to show steps in debugging something, or
3. to give a list of programs or files. Note that the filter program considers groups of single-spaced lines to be single entries in the numbered list.

Note that the begin and end tags for ordered lists, `<OL>` and `</OL>`, must be added manually to the filtered output file.

## III. Sample Code Listing.

It's necessary to allow code listings to retain their original spacing. Otherwise, it becomes completely unintelligible to humans (although the assemblers and compilers can still deal with them).

```
;Z80 assembly code fragment
;on entry, B=number of table entry
```

```
;on exit, A=table entry value

START:

    LD      HL,ADDR1            ;point to base of table
    LD      DE,1                ;length of table entry

LOOP:

    ADD     HL,DE               ;offset one entry
    DJNZ    LOOP                ;keep going until we reach Bth entry

    LD      A,(HL)              ;get the entry
    RET                         ;all done
```

If the conversion works properly, it will look very nice.

## IV. Next Time.

Who knows? Something that hopefully won't bore you to tears.

See you next week!

*Rich*

---

Next Article
Previous Article
*TWWMCA* Archive Main Page

---

---

## IV. Creating an Index Page.

Since the *TWWMCA* articles have covered such a broad range of topics, some kind of content index is necessary, so that you can find articles on a particular topic. The easiest way to do this was to create a large, 2-column table in HTML, with links to the articles on the left, and brief topic summaries for each on the right. Of course, I had to read through all the articles again to create the topic summaries...

I have arranged the index in reverse chronological order, i.e., newest articles first. To add a new *TWWMCA* article, I just add another entry to the top of the table, being sure to update the *previous* article's navigator footer to point to the *new* article as the Next Article.

I'm sure that there are ways to do this using HTML `<FRAME>` elements, but frames are overkill for me `:-)`

## V. Update on the SmartFiler Patch for ADAMserve.

Guy Bona tells me that he finally had a chance to try out my suggested patch for SmartFiler under ADAMserve (see TWWMCA 0008.20)...and it works for him! He can now read and write his database floppies. He has asked that I now look into a similar fix for Recipe Filer and Address Book, which (since they all use the SmartFiler database engine) are likely to have exactly the same bug as SmartFiler. This investigation will be the topic of future *TWWMCA* articles.

See you next week!

*Rich*

# This Week With My Coleco ADAM 0008.20

by Richard F. Drushel

## I. Patching SmartFiler for ADAMserve.

Since ADAMcon 0Ch last month, much of my ADAM time has been spent in disassembly and study of the SmartFiler database program, looking for the code responsible for its crashing under ADAMserve, my serially-linked device driver system to let ADAMs use PC hard drives and floppy drives. Before I had gotten very far with the disassembly, I had a pretty good idea what the problem was: SmartFiler was doing disk I/O at a level lower than ADAMserve EOS was emulating it. However, rather than immediately moving to test that hypothesis, I instead concentrated on getting the disassembly completely resolved, so that it could be edited and reassembled at will. At the last ADAM chat held at 9:00 PM EDT on Wednesday, 16 August 2000 (see http://www.adamcon.org for details on how to join this chat), Guy Bona's patient inquiries prompted me to try to solve his immediate problem (i.e., the crashes under ADAMserve) first, and leave the finer points of disassembly resolution for later.

Because I had already done a global search/replace for EOS global symbols in the SmartFiler disassembly listing, it was easy to find the probable offending code: a single call to `_START_RD_1_BLOCK`. For those not familiar with ADAM's I/O functions, let me just state that, because the original tape drives were so slow, especially to rewind, the operating system was designed to allow background tape I/O. Specifically, you could begin a read/write with `_START_RD_1_BLOCK` or `_START_WR_1_BLOCK`, then go off and do something else (like playing one level of a Super Game), letting the tape spin as needed, but without the user sitting at a static screen. For every `_START_xxx` function, there is an `_END_xxx` function, which can be polled at a low frequency to complete the transfer and reset ADAMnet for the next transfer. Anyway, ADAMserve (for reasons of code space, not for lack of my own cleverness) does not emulate ADAMnet devices as low as the `_START_xxx` or `_END_xxx` levels; it redirects only high levels like `_READ_BLOCK` and `_WRITE_BLOCK` to the server. Thus, any software which uses low-level I/O functions will not work correctly under ADAMserve. Fortunately, most ADAM software does not use the low-level I/O functions, and thus works fine under ADAMserve.

The fix was to "remove" the offending call by patching it out with `NOP` (No Operation) instructions, 00h, in the SmartFiler binary. I used File Manager to do this; but any block editing tool will work. I posted the specific instructions for doing this on an ADAMserve system to the Coleco ADAM mailing list; I will repeat them here.

1. run ADAMserve File Manager
2. Edit Catalog to find the entry for SMARTFILER
3. on my system, it starts at block 552; yours might vary
4. go to Media Functions/Edit Blocks
5. select the *2nd* block of SMARTFILER (on my system, it's 553)
6. go to sector 7 (by hitting `HOME+down arrow` to page through the sectors)
7. at byte offset 1004 decimal, you should see the following 3 bytes in hex: CD A2 FC

8. change these 3 bytes to: 00 00 00
9. hit `return` and save the changes
10. `Shift-Undo` to return to HARDDISK
11. boot SmartFiler as usual from hard disk

After this patch, SmartFiler did indeed work correctly under ADAMserve: it did not hang when trying to update an existing database.

Two additional verifications of this patch were performed:

1. an actual 160K floppy of SmartFiler was patched and then booted from disk (no ADAMserve). It worked fine.
2. an image file was made from this patched disk and then booted under Marcel de Kogel's ADAMEM emulator. It worked fine.

So, the original `_START_RD_1_BLOCK` was not needed...why? It might have been important; why not change it to `_READ_BLOCK`? Normally, this latter change would have been the correct one. However, from reading the context of the original code, this `_START_RD_1_BLOCK` call was merely being used as a "tape pre-wind" command, to get the tape drive spinning. All data is actually transferred with `_READ_BLOCK` and `_WRITE_BLOCK`. So, in this case, changing the `_START_RD_1_BLOCK` to `_READ_BLOCK` would have just duplicated all block reads. Omitting it actually was the correct thing to do.

You can experiment with these changes yourself on your own *backup* copy of SmartFiler.

## II. The SmartFiler Boot Block.

The patched 160K floppy disk created above still does not function as a boot disk under ADAMserve, if you try to boot from it using the Boot Software option under HARDDISK. It hangs ADAMserve...and in the PC status window, you see infinite attempts to read block 2. Why?

Disassembly of the boot block shows that it, too, uses `_START_RD_1_BLOCK` to load the program, assumed to occupy specific blocks on the disk! There is a directory which shows the location and size of all the needed files, but the boot code never reads the directory: it assumes that LOGOS is a 3-block file at blocks 2-4, and that SMART_FILER is a 30-block file at blocks 5-34! Treating a program as a sequence of blocks is fine for Super Games (that's how they are designed), but not a good idea for general user applications.

Why was it done this way? At the end of the boot block is the following "orphaned" message string:

```
        DEFB     "                              "
        DEFB     "BY SIERRA ON-LINE        "
        DEFB     3EH,41H,5DH,55H,59H,55H,41H,3EH
        DEFB     "@  PROGRAMMED BY DON MCGLAUFLIN@@"
        DEFB     "@   ",1DH,"1984 SIERRA ON-LINE, INC.@"
        DEFB     "@ PLEASE WAIT WHILE LOADING GAME@@"
```

Since it was written by a games company, it's not surprising that they wrote user applications in the same way; you do things they way you've always done them. (BTW, the hex codes in line 3 probably formed some graphic logo for Sierra On-Line; the 1DH is a (c) copyright symbol.)

If the boot loader is rewritten to use `_READ_BLOCK` instead of `_START_RD_1_BLOCK` (there's a little more to it than that, as the loader routine also peeks at ADAMnet device control blocks to determine when it's time to load the next block), then it boots correctly under ADAMserve *from the 160K disk*. (I did this and tested it. It also works fine under ADAMEM. But see an additional clarification below.) This boot loader doesn't work from an ADAMserve hard drive because the necessary files aren't stored at the absolute blocks that the loader is looking for. In fact, the hard drive (and ADAMserve) version of SmartFiler is just the 30K SMART_FILER file; there are no boot graphics.

The boot loader could be rewritten to access the files by file through the disk directory, instead of by hard-coded block. However, the SmartFiler application, once loaded and running, creates and maintains databases through absolute blocks! It assumes that there is one database per disk/tape, and that it takes the entire disk/tape. Furthermore, it copies some data stored within the SMART_FILER program itself from the boot drive to the database disk/tape *as absolute blocks*. Thus, the current hard drive (and ADAMserve) versions of SmartFiler are *incapable* of creating a new database; they can only read/write an existing SmartFiler database. Correcting this serious defect will require a detailed understanding of the program code, and a completely resolved disassembly; and it may not even be possible, if the ability to read/write old-format databases is to be retained.

In addition to the `_START_RD_1_BLOCK` calls in the program loader, there was one more which was rather puzzling. Here is the relevant section of the boot code, with necessary equates:

```
BLOCK_1_ADDR    EQU     0000H
PROG_START      EQU     0100H
ZERO            EQU     0000H
BLOCK_1         EQU     0001H   ;directory block
BLOCK_5         EQU     0005H   ;SMART_FILER start block

;Load the rest of SmartFiler, assuming a certain start block and length.

        LD       HL,PROG_START      ;load address for SMART_FILER
        LD       DE,BLOCK_5         ;start block of SMART_FILER
        LD       B,1EH              ;length of SMART_FILER
        CALL     LOAD_BLOCKS        ;load SMART_FILER
;
;This reads the directory block.
;How nice that it assumes it is only one block long...

        LD       A,(BOOT_DEVICE)    ;drive to read
        LD       HL,BLOCK_1_ADDR    ;load address
        LD       DE,BLOCK_1         ;block loword
        LD       BC,ZERO            ;block hiword
        CALL     _START_RD_1_BLOCK  ;read it
```

Why does it do this? It will overwrite the first 768 bytes of SMART_FILER! This is fatal, because the last executable statement in the boot loader is

```
        JP       PROG_START         ;and begin SmartFiler
```

However, the program clearly works as written. What gives?

The answer is truly evil. A quirk of ADAMnet is that it takes *2* reads of a block device to actually cause the data to be transferred into Z80 RAM. The first call makes the device read the data, but it doesn't get into Z80 RAM until the second consecutive call. (Look at the code for `_READ_BLOCK` in the EOS-5 commented listing and you will see the 2 calls to `_READ_1_BLOCK`.) So, the *effect* of a single call to `_START_RD_1_BLOCK` for block 1 is to just rewind the tape (if you're booting from data pack)--a major time savings for later.

This is a major trap for ADAM emulators (or programs like ADAMserve, which emulate ADAMnet devices). If this read-twice behavior is not correctly emulated, programs which depend upon this behavior will malfunction. Thus, in my rewritten SmartFiler boot block, I had to remove the entire read of block 1. (Initially, I changed it to `_READ_BLOCK`, because I had not grasped the purpose of the code; and it crashed spectacularly.)

My commented disassembly source of the original SmartFiler boot block is available:

- sfboot.asm
- sfboot.lst

## III. SmartFiler Boot Under ADAMEM: Displaying the Boot Screens.

The ADAMEM emulator uses disk and tape image files in place of actual media, so reading from an emulated disk or tape means using MS-DOS function calls to read from a file. This is very fast...so fast that you just see a blur of scrambled screen graphics when you boot an image of SmartFiler under ADAMEM.

I didn't like this, so I made a final modification to the new SmartFiler boot block that I made. I inserted a software delay after each boot screen was displayed. The delay is a simple counter, whose length can be varied by an input parameter. Here's the code for the programmable delay:

```
;Pause routine.  Delay is about 5 seconds with B=10.

WAIT:
        LD       B,10       ;change to vary the total delay
WAIT1:
        PUSH     BC         ;save outer loop counter
        LD       HL,0       ;initialize inner loop counter
WAIT2:
        DEC     HL          ;one less
        LD      A,H
        OR      L           ;is HL=0?
        JR      NZ,WAIT2    ;no, keep going
        POP     BC          ;yes, so restore outer loop counter
        DJNZ    WAIT1       ;keep going until B=0
;
        RET                 ;all done
```

With the delays in place, you can see the two boot screens very nicely under ADAMEM. What's more, the added delay is not objectionable if you boot from a real ADAM, or from an external floppy under ADAMserve. You might notice it if booting from tape, but I haven't tried this.

The commented source of the modified SmartFiler boot block, and a binary, are available:

- sfboot01.asm
- sfboot01.lst
- sfboot01.bin

This modified boot block also has read error detection (which the original lacks), so if there is an error reading any of the blocks, the program will abort to SmartWriter (instead of hanging forever).

## IV. A Small Rant on Program Design.

I might hope that all the "tricky" bits of the SmartFiler boot block were thoroughly documented in the original Sierra On-Line source code (now probably lost forever); but I wouldn't be surprised it it weren't. Which is too bad, because the code is poorly designed enough that it's not very easy to figure out without such comments. And what I have looked at of the SMART_FILER program per se also shows lots of poor, unobvious design. I have disassembled many other Coleco applications (EOS, SmartBASIC, SmartWriter, ADAMcalc, ADAMlink IVa), games (Pitfall), and other 3rd-party ADAM software (PowerPaint, File Manager, HARDDISK). SmartFiler has been one of the hardest to decipher.

The SmartFiler binary appears to be a debugger dump of RAM, with several modules loaded via serial port from a host computer as Intel HEX records. All the data areas contain non-SmartFiler program code, some of which is a debugger, based upon the text strings. Moreover, the debugger code has 2651 UART I/O routines, like the Coleco Graphics Processor cartridge, which could also import code and graphics in Intel HEX format. Many assemblers which generate output as Intel HEX do not initialize data areas with any starting value; instead, they skip over the data areas, so when the individual records are interpreted and loaded, RAM which corresponds to data areas gets left with whatever was in it previously. There are 2 such large areas of non- SmartFiler program code in the SmartFiler binary.

Ideally, code fixes made in a debugger are propagated back to the source code, and a new binary made from re-assembly of the corrected source. I hypothesize that this did not happen with SmartFiler. The version date (8/16/1984) is very late in the history of the ADAM, well after

Coleco had abandoned it; so perhaps the final version was produced under sub-optimal conditions, or even by someone who did not have access to the complete source code, perhaps not the original programmer(s).

Be that as it may, it is still very frustrating to have to understand and maintain a program which is not clear and straightforward in its design and implementation, or which exploits unusual or undocumented side effects, or (my biggest pet peeve) uses self-modifying code. SmartFiler is not organized internally into related modules; parts are interleaved and scattered all over. The use of `_START_RD_1_BLOCK` to effect a tape rewind without actually reading a block into Z80 RAM has been castigated previously. There is a long and complex routine which decides what sound effect to make depending upon what key you press, that is absolutely unintelligible unless you map it out as a branched decision tree on paper; talk about spaghetti `GOTO`s in BASIC! And the code used to sort database records is heavily self-modifying; there are data tables which actually contain Z80 opcodes, which are poked into various routines to change sorting parameters, and to terminate sorts. SmartFiler has lots of RAM left over; it doesn't need to resort to these tricks to save precious space.

Whether you're writing in assembler or C++, it's best to write clearly and logically *first*, and then worry about optimizing it for size or speed *if* field testing shows that it's necessary. Trying to write highly- optimized code on the first pass almost never works, and it's that much harder to debug. One programmer's opinion, anyhow...your mileage, of course, may vary.

## IV. Next Time.

I don't know what I'll do for next time, but it will be something interesting. Probably more work on the SmartFiler disassembly; it's almost completely resolved now, and once I can prove that it is, then I can start adding comments and changing some non-EOS global symbols to have meaningful names. And also there are Recipe Filer and Address Book to patch for ADAMserve, in probably exactly the same way as I have done for SmartFiler.

See you next week!

*Rich*

---

# This Week With My Coleco ADAM 0008.13

by Richard F. Drushel *(drushel@apk.net)*

## I. Collection of sources for ADAMserve.

Per Dale Wick's request from ADAMcon 0Ch almost a month ago, I have finished collecting all of the various sources for my ADAMserve serially- linked device driver software. This includes the x86 assembler source for the PC-side server, and the Z80 assembler source for ADAMserve boot, EOS, HARDDISK, and File Manager. The purposes for this collection are (1) to make sure it doesn't get lost, and (2) to put it on an ADAM software CD that Dale's brother Neil is reportedly working on.

As of last Sunday, I had located almost everything I needed, and put it all into various ZIP files. One item that I couldn't seem to find, however, was the source for the HARDDISK boot block. I remembered that it had been modified from an old, pre-ADAMserve version , so I figured that I could just regenerate it and compare the new binary against the old, to be sure that I hadn't made any mistakes. A few hours I thought it would take...

Well, I didn't get a few free hours until this evening, a week later. And after finally making a stand-alone .ASM file, I remembered that the "source" for the HARDDISK boot was present in the HARDDISK binary all along! Original HARDDISK author Tony Morhen liked to write programs which could create their own boot disk images, boot block, directory, and all. Since the ADAMserve version of HARDDISK was descended from Tony's original source, it still had this feature...so, sheepishly I must admit that my new boot .ASM program was superfluous...and so all the source code ought to be ready to send off to Dale.

## II. README files for ADAMserve.

*Ought*...but I realized that there were no "help" texts to go along with the source code...and some of it is a little hairy, because it prompts for configuration options. So, I started to write the README files...and it took me 5 hours to do it `:-(`

This isn't what I wanted to talk about in *TWWMCA* this week, but it's now very late, I need to get this out, and here is a pile of ADAM-related text. So, here it is `:-)` It is, of course, most useful if you are sitting with the various sources mentioned in front of you; but there is lots of useful by-the- way info about usage and program design, so I guess it will be of enough general interest. If not, oh well...

---

README72.TXT help file for ADAMserve 72

|  |  |
|---|---|
| READMESV.TXT | information about SERVE72x.ASM source code for PC server |
| READMEBO.TXT | information about ADAMserve boot disk images |
| READMESB.TXT | help file for sources to the ADAMserve boot block |
| READMEOS.TXT | information about the source code for ADAMserve EOS |
| READMEHD.TXT | information about the sources for ADAMserve HARDDISK 4.203 |
| READMEFM.TXT | help file for sources for ADAMserve File Manager 4.1 |

# README72.TXT -- help file for ADAMserve 72

by Richard F. Drushel
drushel@apk.net
13 August 2000

## PC Setup.

ADAMserve requires (minimum) an XT-class PC with 256K RAM, two 180K floppy disk drives, a serial port, a 20MB hard disk, and MS-DOS 2.0. A parallel port is required if you wish to bypass printing to the ADAM printer.

ADAMserve is known to run perfectly under Windows 95, Windows 98, OS/2 2.1 and 3.0 Warp. ADAMserve runs in a limited fashion under Windows NT 4.0: the ADAMserve hard disk can be accessed over the serial link, but PC floppy disk I/O is completely blocked. As of this writing, ADAMserve has not been tested under Windows ME or Windows 2000.

Connect the PC serial port to the ADAM serial port using a null modem cable. An RS-232 line tester with status LEDs is a good item to put in-line, because it can help you debug cabling problems. It is probably best to test the null modem connection using PC and ADAM telecommunications programs (e.g., ProComm, Telix, IMP, ADAMlink) *first*, before trying to run ADAMserve. If you can type at one keyboard and see characters on the other screen, and vice versa, then the serial link is working; if not, ADAMserve won't work.

Start the PC side of ADAMserve before booting the ADAM. The PC file is called SERVE72x.EXE, where x specifies the PC floppy mapping and PC serial port to be used:

| **x** | **date** | **A:** | **B:** | **serial** |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| A | 9609.02 | 1440K | 1200K | COM1: |
| B | 9609.02 | 1440K | 1200K | COM2: |
| C | 9609.02 | 320K | 320K | COM2: |
| D | 9609.02 | 1200K | 1200K | COM2: |
| E | 9609.02 | 320K | 720K | COM2: |
| F | 9609.02 | 1200K | 1440K | COM2: |
| G | 9609.08 | 720K | 320K | COM2: |

You can use the startup command line option /B to run in monochrome mode; default is color (black and cyan).

You should put ADAMserve into its own directory, e.g., C:\ADAMSERVE. Make sure that the two EOS hard disk image files EOSHD0.DSK and EOSHD1.DSK are present in this directory.

You must have the following line in your CONFIG.SYS file when you boot your PC:

`DEVICE=C:\DOS\ANSI.SYS` (or whatever path leads to ANSI.SYS)

Ignore the "File not found" error message at startup; it is from an as-yet- unimplemented configuration file feature.

## ADAM Setup.

ADAMserve requires (minimum) a stock ADAM with one tape drive and a third- party serial board (Eve, Orphanware, or Micro Innovations). A full printer/ power supply is necessary if you wish to print to the ADAM printer; otherwise, you may use a bare power supply and redirect printing to a PC parallel printer.

After the PC side of the server is running, put the ADAMserve boot tape/disk into any drive and pull the reset switch.

ADAMserve has built-in drivers for all third-party serial boards. However, you must configure your boot tape/disk to specify which serial port you are using for ADAMserve.

Block 0 boot of the ADAMserve boot disk has a reserved byte to specify the server serial baseport. Offset 2 (0-based) has one of following values:

| | |
|---|---|
| 44h | Orphanware port 68 |
| 4Ch | Orphanware port 76 |
| 54h | Orphanware port 84 |
| 5Ch | Orphanware port 92 |
| 18h | MIB2 port 1 |
| 10h | MIB2 port 2 |

Boot disk images are provided for all 6 serial baseports. However, you can use a block-editing utility like File Manager to change the baseport byte if you wish.

## Using ADAMserve: ADAM Side.

The ADAMserve boot will load HARDDISK from the boot tape/disk. Once HARDDISK is running, you can use it to launch programs from the hard disk or from tapes or (PC) floppy disks. For example, to run a program stored on Volume 5, change the Logical Drive to 5, then select "Boot From Hard Disk". From the menu of boot blocks, select the appropriate boot block. Your application will then run. To boot from an external drive, insert the tape/disk, then select "Boot Software" and select the appropriate drive.

From application programs, you can change the current Logical Drive of the hard disk by typing Shift-Tab (hold down both keys at once). You will hear a beep, then the current Logical Drive will be advanced by one (9 wraps back to 0).

To return to HARDDISK from any application, type Shift-Undo (hold down both keys at once). This reboots HARDDISK from the server hard drive.

## Using ADAMserve: PC Side.

The ADAMserve server program accepts the following function keys:

| | |
|---|---|
| `<escape>` | flushes the server receive buffer (may be useful after an I/O error). |
| `C/c` | ADAMcalc 'fudge'. Performs proper page eject when printing with either serial printer or "other" printer (parallel printer) in ADAMcalc. |

|  |  |
|---|---|
| `P/p` | PowerPaint 'fudge'. Ignores the hiword of block number (necessary for PowerPaint disk I/O). |
| `Q/q` | quits to DOS. |

The two 'fudges' are toggles: repeated typing turns them on and off. You should use the appropriate toggle before launching either ADAMcalc or PowerPaint from HARDDISK.

Note that status information is displayed: blocks being read or written, and any error messages.

## Assembling the Source Code.

Instructions for assembling both the PC and ADAM source code for ADAMserve are given in the respective ZIP archives containing the source.

## Known Bugs.

A small wrinkle seems to exist in the 1200K server floppy disk I/O: sometimes it can't read a 1200K disk correctly if it's the very first disk ADAMserve reads after it's started. If you read from a 160K or 320K floppy disk *first*, all subsequent attempts to read any 1200K disks are successful, regardless of whether you mix/match 160K/320K/1200K disks subsequently.

The ADAM printer emulator to parallel printer does not support underline printing from SmartWriter.

ADAM software which attempts to do raw ADAMnet I/O (by directly manipulating device control blocks), or does block I/O at levels below `_READ_1_BLOCK`, will not work with ADAMserve, due to incomplete ADAMnet emulation. Such software can often be patched, however, to be "better behaved". If you find such software, please let me know, and maybe I can patch it.

---

# READMESV.TXT -- information about SERVE72x.ASM source code for PC server

by Richard F. Drushel
drushel@apk.net
13 August 2000

## Source Code.

The following x86 code files are necessary to build the PC server for ADAMserve, using Borland's Turbo Assembler, TASM.EXE (or equivalent):

| | |
|---|---|
| `SERVE72x.ASM` | x=A,B,C,D,E,F,G (different floppy/serial port mixes) |
| `DATA_31x.ASM` | x=A,B,C,D,E,F,G |
| `CON_IO.ASM` | keyboard/screen |
| `FILE_IO.ASM` | DOS filesystem |
| `SER_IO4.ASM` | low-level serial port |
| `BLK_IO61.ASM` | block devices |
| `FDD_IO2.ASM` | low-level floppy disk |
| `CHAR_IO.ASM` | character devices |
| `TOGGLE_1.ASM` | ADAMcalc and PowerPaint toggle handlers |
| `SETUP_2.ASM` | configuration module (skeleton only) |

To assemble the server, type

```
TASM SERVE72X SERVE72X SERVE72X
```

at the MS-DOS prompt. Unlike Z80ASM+, TASM.EXE creates only object files; a separate linker program, TLINK.EXE, must be run to create the final SERVE72x.EXE application. Thus, after running TASM.EXE, type

```
TLINK SERVE72X
```

It is unknown whether SERV72x can assemble correctly under the MASM assembler. The source currently doesn't use any TASM-specific tricks that I know of.

## Limitations.

The various low-level hardware I/O routines, while perfectly acceptable in MS-DOS, may become increasingly useless as later Windows operating systems restrict direct hardware access, or imperfectly emulate it. For instance, while SERVE72x works perfectly well under Windows 98, the floppy disk I/O code does not work under Windows NT 4.0. Windows-specific development tools will be necessary at some point.

The most annoying feature of ADAMserve is that the PC side is not runtime- configurable by the user, rather relying upon special separately-assembled code bases to change serial port or combination of disk drives. This is mostly due to not wanting to write a menu interface in assembler, and not having given enough thought to how to write one in a higher-level language (e.g., C or QuickBASIC) and linking it in with the nuts-and-bolts assembly code.

---

# READMEBO.TXT -- information about ADAMserve boot disk images

by Richard F. Drushel
drushel@apk.net
13 August 2000

## ADAMserve Boot Disk Images.

The following 160K disk images are provided for booting ADAMserve:

| | |
|---|---|
| `ASBOOT16.IMG` | Micro Innovations port 2 |
| `ASBOOT24.IMG` | Micro Innovations port 1 |
| `ASBOOT68.IMG` | Eve/Orphanware port 44H |
| `ASBOOT76.IMG` | Orphanware port 4CH |
| `ASBOOT84.IMG` | Orphanware port 54H |
| `ASBOOT92.IMG` | Orphanware port 5CH |

These images are identical except for the serial baseport byte at offset 2 (0-based). They contain:

- the ADAMserve boot block
- the ADAMserve HARDDISK boot block
- ADAMserve HARDDISK 4.203
- EOS4.0
- EOS4.X
- File Manager 4.1

These images may be cloned to actual 160K disks using the appropriate TDOS program, CLONE.COM. They could also be block-copied onto disks starting from block 0, using File Manager and a source medium which is bigger than 160K. Only the first 47 blocks (blocks 0-46) actually have data on them, however, so it is possible to transfer less than the full 160 blocks. Though disk images, they may also be copied to tape for boot purposes; ADAMserve won't care.

## Usage.

For complete instructions, see README72.TXT.

---

# READMESB.TXT -- help file for sources to the ADAMserve boot block

by Richard F. Drushel
drushel@apk.net
13 August 2000

## Source Files.

The following source code files are necessary to build the ADAMserve block 0 boot program:

| | |
|---|---|
| `SBOOT20.ASM` | main boot code |
| `SBOOT20A.ASM` | special version for inclusion in HARDDISK |
| `SIOINI32.ASM` | code to initialize serial ports |

To assemble the ADAMserve boot block, type

```
Z80ASM+ SBOOT20.ASM
```

(Note: if you are using a CP/M emulator under MS-DOS, you will have to change the name of the assembler to Z80ASMP, because '+' is an invalid filename character in MS-DOS.)

The resulting 1K binary file must be block-copied to block 0 of the boot tape/ disk.

## Usage.

The ADAMserve boot block has built-in drivers for all third-party serial boards. However, you must configure your boot tape/disk to specify which serial port you are using for ADAMserve.

Block 0 boot of the ADAMserve boot disk has a reserved byte to specify the server serial baseport. Offset 2 (0-based) has one of following values:

| | |
|---|---|
| 44h | Orphanware port 68 |
| 4Ch | Orphanware port 76 |
| 54h | Orphanware port 84 |
| 5Ch | Orphanware port 92 |
| 18h | MIB2 port 1 |
| 10h | MIB2 port 2 |

Boot disk images are provided for all 6 serial baseports. However, you can use a block-editing utility like File Manager to change the baseport byte if you wish. You may also reassemble from source, changing the default value of the baseport.

## Limitations.

The ADAMserve boot block loads the HARDDISK boot block and passes control to it. See the documentation for the ADAMserve HARDDISK source for information on how to construct an entire ADAMserve boot disk.

---

# READMEOS.TXT -- information about the source code for ADAMserve EOS

by Richard F. Drushel
drushel@apk.net
13 August 2000

ADAMserve EOS is based largely upon the EOS-7 codebase (rather than the EOS-6 version present in ROM of every production ADAM I've ever seen), because EOS-7 is much smaller than EOS-6, and code space is at a premium: EOS cannot extend below 0E000H in RAM. The source for EOS-7 is the original .LST which was derived from the SmartBASIC 2.0 distribution. While this code is the original code, with the original symbols, it was stripped of comments. Any comments now found in the code are my own addition; but I have only commented those places where it was relevant to ADAMserve.

## Source Files.

The following source code files are necessary to build the two ADAMserve EOS binaries:

| | |
|---|---|
| `EOS551.ASM` | most of EOS |
| `EOSIO453.ASM` | ADAMserve protocol handshaking module |

To assemble ADAMserve EOS, type

```
Z80ASM+ EOS551.ASM
```

(Note: if you are using a CP/M emulator under MS-DOS, you will have to change the name of the assembler to Z80ASMP, because '+' is an invalid filename character in MS-DOS.)

The binary file known as EOS4.0`(^B)`, where (^B) is Control-B, `CHR$(2)`, is the "standard" ADAMserve EOS. The binary file known as EOS4.X`(^B)` is a "special" version with some bugfixing code to allow ADAMcalc and SmartLOGO to function correctly with ADAMserve. ADAMserve HARDDISK has code to detect if either ADAMcalc or SmartLOGO is being launched, and will load in the special EOS4.X version before running them. A soft reboot with Shift-Undo causes the normal EOS4.0 version to loaded before returning to HARDDISK.

In the .LST files provided, EOS551.LST corresponds to EOS4.0, and EOS551A.LST corresponds to EOS4.X.

## Assembly Options.

There are a number of assembly-time prompts for user input. Some are to enable debugging code; some are to enable functions that there is not enough RAM available for; and some are to select between 2 different final versions of the EOS binary. Here are the prompts and the required responses:

| | |
|---|---|
| `RAMdisk code?` | `FALSE` (no room for code) |

| | |
|---|---|
| Shift-Wildcard 64K RAM dump? | FALSE (debugging) |
| Enter amount of pre-pad: | 0 for both 4.0 and 4.X (may be needed to pad code forward so that current hard disk volume and volume offset table are at the required reserved RAM addresses) |
| Assemble for EOS? | TRUE (FALSE creates a BLOAD file for SmartBASIC) |
| Check for overrun/framing/parity errors? | FALSE (no room for code) |
| Assemble special ADAMcalc/SmartLOGO EOS? | FALSE for 4.0, TRUE for 4.X |
| Enter amount of post-pad: | 0 for 4.0, 5 for 4.X (may be required to pad code forward so that EOS jump table is at the required reserved RAM address) |

## Usage.

The files EOS4.0 and EOS4.X must be copied to both the ADAMserve boot disk and Volume 0 of the ADAMserve hard disk.

---

# READMEHD.TXT -- information about the sources for ADAMserve HARDDISK 4.203

by Richard F. Drushel
drushel@apk.net
13 August 2000

## Source Files.

The following files are needed to build ADAMserve HARDDISK 4.203 (using the Z80ASM+ assembler from SLR, or equivalent):

| |
|---|
| HD4203.ASM |
| S-SPEC2.Z80 |

| |
|---|
| DATA5.Z80 |
| SPATCH.Z80 |
| 1ST2BLK5.ASM |
| SBOOT20A.ASM |
| SIOINI32.ASM |

To assemble HARDDISK, type

```
Z80ASM+ HD4203.ASM
```

(Note: if you are using a CP/M emulator under MS-DOS, you will have to change the name of the assembler to Z80ASMP, because '+' is an invalid filename character in MS-DOS.)

There are macro prompts for input parameters (leftovers from the original ability to build all possible versions of HARDDISK for different hard disk controllers from a single source). You should type `FALSE` for everything except the `SERVER?` prompt, at which you should type `TRUE`.

NOTE: the output is an 11-block binary image of a 160K disk, HD4203.BIN. (This also is a leftover from Tony Morehen's original code, which spit out an entire boot disk image.) Block 0 is the HARDDISK BOOT block; block 1 is a DIRECTORY, and blocks 2-10 are the actual HARDDISK program.

## HARDDISK and ADAMserve Boot.

To make an ADAMserve boot disk, you must

1. copy the ADAMserve boot code to block 0 of the disk
2. copy the HARDDISK BOOT code to a file named SERVBOOT2`(^B)`, where `(^B)` is Control-B, `CHR$(2)`
3. copy the HARDDISK program code to a file named HARDDISK`(^B)`
4. copy the 2 special versions of EOS to files named EOS4.0`(^B)` and EOS4.X`(^B)`.

   If you are wise, you will also

5. copy ADAMserve File Manager (FILEMAN`(^B)`) to the boot disk; you may need it some day for disaster recovery!

The easiest way to do this is to get all the pieces onto EOS media as separate files, and then use File Manager to do the block and file copying. Getting the pieces onto EOS media can be a job in itself, especially if you are using an MS-DOS development environment. XMODEM transfer from PC to ADAM using your favorite terminal programs will work adequately.

Other README files describe how to assemble the ADAMserve boot block and the 2 versions of EOS.

## HARDDISK on the ADAMserve Hard Drive.

Volume 0 block 0 of the ADAMserve hard drive must contain the contents of the SERVBOOT2 file. Additionally, volume 0 must also contain the files HARDDISK, EOS4.0, and EOS4.X. Since Volume 0 of all EOS hard disks is in a very fragile state (with many programs which are at specific blocks and which must remain at those specific blocks in order to function correctly), it is not wise to use file copying methods to install these files from binary images. Assuming that you already have a working ADAMserve hard drive, just block copy onto the necessary pre-occupied blocks. HARDDISK on Volume 0 is set to occupy 12 blocks (despite having only 9K of code currently); *do not change this*.

For development purposes in an MS-DOS environment, I have used QuickBASIC programs to copy binaries from files to the proper positions in the 10 MB EOS hard disk image. These are not "intelligent" programs, as they don't read the directory to find out where to copy the files to; for simplicity they just assume that the files are at certain locations and block copy to the appropriate offsets. If someone wants to write these general-purpose utility programs, please go ahead :-)

## Limitations of ADAMserve HARDDISK.

1. You can't yet write an ADAMserve BOOT disk/tape from the menu.
2. 'Krunch' works, but *don't* use it on Volume 0, for the reasons described above. (This caveat applies to *all* versions of HARDDISK, for *any* hardware.)
3. Similarly, *don't* 'Init' Volume 0!
4. The logical-to-physical device table can't be edited yet, except for toggling between a genuine ADAM printer and emulating it on a PC-connected parallel printer. Thus, both disk drives are mapped to the PC; ADAMserve can't see a real ADAMnet disk drive. Implementing the remapping menu would be my first project for additional work on ADAMserve HARDDISK.

---

# READMEFM.TXT -- help file for sources for ADAMserve File Manager 4.1

by Richard F. Drushel
drushel@apk.net
13 August 2000

## Source Files.

ADAMserve File Manager was completely reverse-engineered from the File Manager 4.0 binary. All hard-coded device dependencies were removed; all devices are accessed at the EOS level, not at the ADAMnet device control block level. Thus, ADAMserve File Manager should work perfectly well with all Coleco and third-party disks, including Mini Wini and IDE hard disks.

The following source files are required to assemble ADAMserve File Manager 4.1:

| | |
|---|---|
| `FM023.ASM` | application |
| `FM4BOOT.ASM` | block 0 boot, for making a stand-alone disk |

TO assemble File Manager, type

```
Z80ASM+ FM023.ASM
```

(Note: if you are using a CP/M emulator under MS-DOS, you will have to change the name of the assembler to Z80ASMP, because '+' is an invalid filename character in MS-DOS.) To assemble the boot block, type

```
Z80ASM+ FM4BOOT.ASM
```

The resulting binaries must be copied to tape/disk for use. FILEMAN`(^B)`, where `(^B)` is Control-B, `CHR$(2)`, is the accepted name for File Manager. It may be file-copied to any desired destination. The boot block image, however, must be block-copied to block 0 of the desired tape/disk.

NOTE: On the ADAMserve hard drive, the Volume 0 copy of FILEMAN should *only* be updated by block-copying onto the existing image. There is a bug in the file launcher of HARDDISK which limits the display of launchable programs to only one screen's worth. If there are too many entries, you cannot page down to another screen. FILEMAN is *always* something you want to be able to launch, so it occupies a top position in the directory listing. If you move it to the end, you may not be able to see it some day, if your Volume 0 fills up with too many application software boot blocks.

## Bug Fixes.

A long-standing bug in File Manager was that the entire number of blocks of a tape/disk could not be copied in one pass, because of an off-by-one error in the block counter. This has been fixed in ADAMserve File Manager.

## Limitations.

The FORMAT menu merely gives the MS-DOS `FORMAT` command switches necessary to create disks of the desired size. Since the disk formatting protocol of ADAMserve has not been specified implemented, there is no way currently to format disks from within ADAMserve.

---

See you again next week!

*Rich*

---

# This Week With My Coleco ADAM 0008.06

by Richard F. Drushel

## I. Project: Disassembly of SmartFiler version 27D 8/16/1984.

Last week in my ADAMcon 0Ch report (*TWWMCA* 0007.30), I mentioned that one of the reasons that I brought a complete PC/ADAM/ADAMserve system with me to Toronto was so that I could debug some problems that Guy Bona was having with ADAMserve and SmartFiler. (For those who don't know, ADAMserve is software written by me to allow an ADAM to access PC drives and devices over a null-modem serial link. See *TWWMCA* 9709.26 for a complete technical description.) Guy had a database that he had created using SmartFiler under Marcel de Kogel's ADAMEM emulator, and wanted to use it under ADAMserve. Guy could read/write this database okay under ADAMEM, but when he cloned the .DSK image to a real 160K floppy and tried to read it under ADAMserve, SmartFiler would lock up the ADAM, forcing a hard reset to get out of it.

Guy demonstrated this behavior to me using my ADAMserve setup in my room at the convention. While it did turn out that the actual 160K floppy that we were using had been incorrectly cloned from the ADAMEM .DSK image, the fact remained that SmartFiler would hang whenever it tried to read/write an ADAMserve floppy disk drive. Moreover, two other Coleco programs which (I hypothesize) utilize the guts of the SmartFiler database engine, Recipe Filer and Address Book, *also* hang under ADAMserve when doing server drive I/O. So, I hypothesized that all three of these programs have the same "problem" with ADAMserve, and that the fix for one would fix all of them.

## II. Disassembly: A Review.

How do you "fix" a program which exists only as an executable binary image? You disassemble it--run a program which converts the binary image back into a text file of assembly code which, in principle, can be edited and then reassembled to give a new, "fixed" binary. Twelve years ago, when I was disassembling the EOS ROM and SmartBASIC 1.0 to see how they worked, I used a SmartBASIC program that I wrote, UNASMHEX, which did not attempt to create a re-assemblable source file. In the mid-1990s I discovered Kenneth Gielow's Z80DIS 2.2 for CP/M, a freeware Z80 disassembler which has some "artificial intelligence" to figure out what is code and what is data (always a hard problem: how do you know if the bytes represent instructions or just a message string?). Nowadays I run Z80DIS under a CP/M emulator for MS-DOS: formerly Joan Riff's Z80MU, but more recently (because of better emulation) Sydex's 22NICE. (Why not Simeon Cran's MYZ80, the "best" Z80 emulator for MS-DOS? Because it requires disk image files instead of being able to read/ write files directly from an MS-DOS directory; the latter is much more convenient.)

Even a "smart" disassembler's output requires manual inspection and tweaking. And once you have the disassembly, you have to interpret it--all of the original programmers' comments are gone! Now it's time for a good word processor to do global search/replace on all labels which represent EOS function calls, EOS global data structures, etc. Fortunately for me, I have compiled lists of all such EOS and OS-7 global symbols; and I also have commented source code for the EOS and OS-7 operating systems. These resources, combined with long experience, allow the creation of a reasonably interpretable disassembly listing.

If all you want to do is read the listing, *or* determine places to make patches to the existing binary which don't change the program's memory map footprint in any way, then you don't have to worry too much that you might not have completely or correctly resolved all of the code-vs.-data ambiguities. If, however, you'd like to make major changes which would cause data areas to be moved, then you must be absolutely sure that you've picked out every address or vector table and flagged them as such for the disassembler. Otherwise, the tables will get moved, but the address references won't (they will just be static binary data), so the "new" program will crash.

How can you tell that you have correctly resolved all of the ambiguities? The best way is to insert a `NOP` (no operation) statement after the first code statement in the program, reassemble, and run the new binary. This `NOP` will shift everything up by one byte. If *all* features of the program still work, then you have caught all the data references, and can go on to make whatever changes you like. If the program crashes, or if menu text strings display as shifted over by one letter, with the first letter being garbage, then you know that you have missed something.

I used this procedure to regenerate assemblable source code for both SmartWriter R80 (see *TWWMCA* 9710.20) and the ColecoVision cartridge game Pitfall (see *TWWMCA* 9701.19). Further hints and kinks about disassembly can be found in those articles.

## III. Normal SmartFiler Startup.

SmartFiler (like all other Coleco software) is normally booted by placing the program tape/disk into the drive and pulling the reset switch. This action causes block 0 of the tape/disk to be loaded into RAM at address 0C800H, followed by a `JP 0C800H` to execute the code there. This "boot block" is responsible for loading the rest of the application into RAM and then executing it. For systems with third-party hard drives, applications are installed on disk such that all the block 0 boot blocks are stored as separate files, and the rest of the application binaries and support files are stored in different "volumes" (akin to subdirectories). Under HARDDISK, the user specifies which application he wants to launch; then HARDDISK loads the appropriate boot block and changes the current "volume" to be where the rest of that application's files are. This is a workaround for the fact that the ADAM was designed as a single-user, single-application system, and all the Coleco software assumes that it can take over the entire machine.

The user booting SmartFiler from tape or disk sees first a message screen which shows a blue block graphic "ADAM" and white text `THE COLECOVISION(r) FAMILY COMPUTER SYSTEM PRESENTS SMART FILER(tm) (c) 1984 COLECO`. This is followed shortly by a complex graphic screen depicting lots of sheets of paper being filed, and a graphic version of the words "Smart Filer". Finally, the SmartFiler data entry window appears. (At this point, if you type ^R, Control-R, you will see the version number appear on one of the SmartKey menu strings; the latest is 27D dated 8/16/1984.)

Since the stock ADAM came only with a tape drive, and tape drives are slow compared to disk drives (especially considering rewind time), an important design feature of Coleco software is to get something on the screen quickly, to distract the user while the rest of the application is loading in from disk. There are EOS function calls to do tape/disk I/O in the background, so that the user can start interacting with the program (especially in Super Games) while the tape drive is spinning. Still, the wait can seem long, especially to those accustomed to modern computers.

## IV. Design of SmartFiler Code.

SmartFiler consists of the 1K boot block (BOOT), 3K of code and data for the graphic logo (LOGOS), and 33K of application (SMARTFILER). While these three pieces are organized as files on the SmartFiler tape/disk, the software (unfortunately) does not access them as files: instead, it assumes that they occupy certain contiguous blocks, and have certain specified lengths, and accesses them solely by absolute block number. LOGOS is assumed to be at blocks 2-4, and SMARTFILER at blocks 5-37. This is *bad* design! (The version of SmartFiler which was patched to run from hard drives must occupy a different set of absolute blocks in volume 0; if it is moved from those absolute blocks, it will crash.)

BOOT contains all the code and data necessary to load the first boot screen; it then loads and executes LOGOS, which creates the graphic boot screen; and then it loads SMARTFILER, finally jumping to the start of application code. SMARTFILER then relocates itself to a different address from which it was loaded by BOOT, before actually entering the main SmartFiler data entry screen. The relocator module of SMARTFILER is 1K in size and appears to have been tacked on to the beginning of a 32K debugger dump of addresses 0000H-7FFFH. About the first 2K (after the 1K relocator) and the last 2K of SMARTFILER are debugger code and menu strings which are orphaned from the main SmartFiler code. (Among other things, this debugger had a serial I/O module to communicate with a remote computer, using some unknown serial board which, I bet, is the same as the one used in the Coleco Graphics Processor cartridge. Hmm, maybe it even *is* CGP code...)

## V. Disassembly Source.

To date, I have completely disassembled and resolved, and partially commented both SmartFiler BOOT and LOGOS. The disassembly source can be reassembled (using SLR's Z80ASM+) to give binaries identical to the starting binaries. I haven't yet done the `NOP`-offset trick to absolutely verify that I have resolved the listings correctly; but these are short and straightforward programs, and I'm confident that I've gotten them right.

The disassembly source files are available:

- sfboot.asm
- sflogos.asm

I also have a fully-resolved disassembly of the 1K relocator module of SMARTFILER; but the remaining 32K module is still not completed. I have a lot of code tracing to do to disentangle all the debugger code (which is not executed by SmartFiler) from the rest. I also haven't yet started to look for where ADAMserve might be crashing. I won't do that until I am confident that I understand what the entire program is doing. It will take a few more weeks of study, I think. Stay tuned.

## VI. Next Time.

There are a number of programming design and style issues raised by the disassembly of the SmartFiler BOOT program (and a few rants to be ranted). I'll discuss these next time.

See you again next week!

*Rich*

---

# This Week With My Coleco ADAM (TWWMCA) 0007.30

by Richard F. Drushel

Yes, after a hiatus of almost 2.5 years, *TWWMCA* is back. ADAMcon 0Ch was just last week in Toronto, and ADAMcon 0Dh/13/XIII will be run by me in Cleveland next year, so I will have lots of weekly ADAM stuff to do and to talk about. It seemed to me that resurrecting *TWWMCA* would be a good way to help keep me on track for the next convention, and to help insure that there would be lots of new and interesting ADAM things to show off then.

I will kick off the new *TWWMCA* series with a report about ADAMcon 0Ch. Some attendees have already posted reports, and others are bound to follow. Photos from the convention can be seen at http://www.adamcon.org/, and I imagine that reports will probably end up archived there as well.

The theme of ADAMcon 0Ch was "Dinosaurs into the new millennium"; and I came back from the convention with the conviction that the ADAM isn't ready to go extinct yet!

*Rich*

# Report on ADAMcon 0Ch
# Toronto, Ontario, Canada
# 20-24 July 2000

by Richard F. Drushel

Here's some of what I did at ADAMcon 0Ch (that's 12 decimal for the hexadecimally-challenged). Some stuff may get left out...apologies in advance.

## Thursday, 20 July 2000.

My daughter Elanor (age 10) and I left Cleveland Heights, Ohio, at about 10:30 AM. We drove out to I-271 north, then to I-90 east, and onward along the "North Coast" of Ohio, Pennsylvania, and New York. We stopped to eat lunch at the Denny's in the Angola service plaza just before you get to Buffalo. (This service plaza has a neat layout: Denny's/ McDonald's/restrooms in a building on the median strip, with covered walkways over the highway to parking and gas stations on each side.) Elanor saw some road department fun from the walkway: cracks in the road had been recently filled with tar, and someone had painted a smiley face in tar on the pavement. We stopped for about 45 minutes, then drove on to Buffalo.

Once we reached Buffalo, we took I-190 west to the Peace Bridge to Canada, and then the QEW (Queen Elizabeth's Way) to Toronto. Traffic flowed nicely until we got to Oakville, where there is a Ford automobile plant: it was 3:30 PM, and day shift was driving home from work, so traffic came to near-standstill for about 15 minutes. From the QEW we took 427 north and then 401 east to Keele Street, where the Triumph Howard Johnson hotel stands. 401 was "interesting" driving at 4:00 PM, as it is divided into separate express and local "collector" highways, which intertwine with one another like some crazy macrame project...fortunately I made all the correct lane changes and didn't get run down by vehicles far exceeding the posted speed limit of 100 km/hr. We got to the hotel at about 4:30 PM; so driving time (minus the stop for lunch) was about 5 hours 15 minutes. The trip was 291.8 miles.

We checked into the hotel and met Ena Greenshields of MTAG (Metropolitan Toronto ADAM Group) in the lobby. I ran into Dale Wick, the convention host, when I wasn't looking for him, so I gaped a few blank looks before I realized who was talking to me. Sorry, Dale :-) Before we went up to our room (on the top floor, 1002), we saw Ron Mitchell come in. Ron is probably our all-time man-miles award winner, as he lives on Vancouver Island and most of the ADAMcons are in the midwest or on the east coast; so for Ron, ADAMcon means a transcontinental trip.

We then took our suitcases upstairs, then came back down to unload my computer equipment from the van. I had originally intended to come to ADAMcon 0Ch as just a spectator, but I got several last-minute requests to troubleshoot/talk about/help install my ADAMserve program, so I decided to bring a complete ADAMserve setup (a stock ADAM system/monitor with serial card, and a 486DX2-66 PC system/monitor). So, Elanor and I unloaded the equipment onto a cart and we brought it upstairs. I didn't unpack anything yet, since Dale had said we were to meet downstairs 5:30-6:00 PM to decide what to do about supper, as a group. Setting up the computers could wait until later in the evening.

By 6:00 PM everyone had arrived (all 20-odd of us), and we went to the hotel restaurant. After supper, everyone went up to the 10th floor (all of our rooms were adjacent) and hung out in the "hospitality suite"--a nice double-sized room with lots of chairs, pop, and munchies. We spent the rest of the evening catching up on the last year's events with all of our ADAM friends. I did find time to unpack my computers and set them up in my room. Fortunately, everything worked, including the ADAMserve serial link between ADAM and PC--very important, since I was supposed to give a session on this topic, and it would be hard to do if the hardware didn't work.

Elanor got tired around 11:00 PM and went to bed. I stayed around the lounge maybe another hour, then went back to my room to read for a while. Lights out was about 1:00 AM, I think. Ron Mitchell has observed that we ADAMconners are getting older, and lights-out is coming sooner and sooner every year `:-)`

## Friday, 21 July 2000

I got up at 6:00 AM, and I awakened Elanor at 6:30 AM (without too much prodding, amazingly). Breakfast was at 7:00 AM in the hospitality suite (conveniently right across the hall): a variety of cereals, fresh fruit, bagels, toast, etc. ADAMites trickled in gradually...we then went down to the basement level to the meeting room where we would be spending most of the convention. Dale Wick hadn't gotten all the computers set up yet for the 9:00 AM kickoff session, so we all pitched in and unpacked boxes and got 5 or so ADAM systems running.

The morning session was by John-Paul Gignac of MTAG, about "Conway's Game of Life", a famous computer simulation of population dynamics, with an ADAM implementation by Dale Wick. Briefly, the "game" is played on a grid of squares, each of which may be "alive" or "dead". The grid is seeded with an initial pattern of live or dead squares, and then the game is run: live squares can stay alive if they have a certain number of living neighbors (I forget how many), but die otherwise; and dead squares can come to life if they have a certain number of living neighbors, but stay dead otherwise. The simple rules give rise to a variety of stable and dynamic patterns. I remember writing a similar "artificial life" program in BASIC long ago with sharks and fish: the fish live rather like the rules for "Life", but the sharks prey upon the fish.

After lunch, Ron Mitchell talked about various methods for moving files between ADAMs and other computers (especially PCs, now that Marcel de Kogel's PC-based ADAMem emulator is so popular). Briefly, your options are modem transfer or direct reading/writing of ADAM disks in PC drives. For technical reasons, single-sided 160K disks (5.25-inch) are about the only format which can be read/written using PC BIOS function calls. Programs like 22DISK and ANADISK (from Sydex) bypass the BIOS and talk to the floppy disk controller directly; so does ADAMserve, which is also able to read/ write native ADAM disks.

After Ron's session, there was a lot of free time until supper, so I put it to use back in my room, doing some work with ADAMserve. During my system setup, I had discovered a 4-year-old bug in the ADAMserve HARDDISK program (for the uninitiated, HARDDISK is the shell/program launcher for ADAMs with hard drives under the EOS operating system). Brief technical digression: ADAMserve uses a serial link to allow an ADAM to access PC devices as if they were real ADAMnet devices. At startup on the ADAM side, the boot disk has one byte which specifies which of the 6 possible serial baseports is being used for the serial link. This baseport value is then stored in RAM and (theoretically) preserved across "soft" reboots (the SHIFT-UNDO key sequence, ADAM's version of Ctrl-Alt-Del). I said "theoretically" because it's supposed to be--but there was one case in which it was not preserved. ADAM has an EOS operating system in ROM, but ADAMserve replaces this at boot with a new EOS image from the boot disk (or hard disk if a soft reboot). There are actually 2 EOS images which can be used, depending upon what application software is launched from HARDDISK, since some ADAM software is "ill-behaved" and requires special handling under ADAMserve. You guessed it: when loading in the 2nd of these EOS choices, the global RAM byte containing the serial baseport was overwritten by the disk image. I never caught it because my serial card was configured with this same baseport. I did of course test ADAMserve boot using all 6 possible serial baseports, but I didn't test all apps with all ports...anyway, the fix was only a few lines of Z80 assembler, so by supper time I had made, assembled, and installed my bugfix version.

For supper, we all went down the street to Pizza Hut. This was good for Bob and Judy Slopsema, as they told us that it is their custom to eat pizza on Saturday night `:-)` Food was good, but service was slow: for some reason, restaurants can cope with 10 orders from individual people in line, but freak out if they get the same 10 orders from a group sitting at a big table. I have never worked in food service, so I don't know why this is; but we have seen it again and again at ADAMcons.

After supper, Guy Bona and I tried to reproduce some errors that he said he was getting with SmartFiler (a database program) if run under ADAMserve. He had a database which worked fine from a disk image file under the ADAMem emulator, but the same database cloned onto a real disk caused ADAMserve to crash. There were actually 2 separate problems: (1) the cloned disk was produced with the wrong interleave settings for the DCOPY program, and (2) SmartFiler/ADAMserve died when trying to write anything to an existing database (creating a new empty database and reading from an existing database worked). Interestingly, the other 2 Coleco programs which use the SmartFiler database engine (Recipe Filer and Address Book) *also* die in the same way as SmartFiler under ADAMserve. So, the best I can do for Guy is to start disassembling SmartFiler to discover what the problem is. My guess is that these programs are trying to write directly to ADAMnet's device control blocks (areas of memory-mapped I/O), instead of using EOS's `_READ_BLOCK` and `_WRITE_BLOCK` functions (which ADAMserve redirects to the serial link to the PC). It will take some weeks of work (probably), and will be future grist for the *TWWMCA* mill. Stay tuned.

In case you're wondering about Elanor, she attended the sessions, but got hooked on Steve Pittman's game ADAM Bomb II, and ended up either hanging out with the "Ladies' Auxiliary" of Judy and Meeka Slopsema and Dale's wife Jillian (who were playing it on laptops) or in our room playing it on my computer. (Hmm, is Jillian still J. Arnott, or is she now J. Wick? I don't know...Dale and/or Jillian, some clarification, please?)

More lounge conversation, then back to the room for a bit of reading and then lights out by about midnight.

## Saturday, 22 July 2000

The alarm still went off at 6:00, but I didn't listen to it until about 6:30. I got Elanor up at 7:00, and we went down to the hotel restaurant for breakfast. A few ADAMites were there (I think Guy Bona and Murray McCullogh), and the rest trickled in later. Continental-type breakfast (like Friday's) was free, add $CN3.00 per person to add all-you- can-eat pancakes/eggs/bacon. This ADAMcon was positively overflowing with food, in my opinion (Herman Mason take note), so we stuck to the Continental breakfast.

The 9:00 AM session was by Neil Wick (Dale's brother), about how to effectively use Web-based search engines. The sample search topic was, how could you find information about the hotel we were staying at, if all you knew was that it was a Howard Johnsons in Toronto? Neil had a nice handout comparing about 10 different search engines, the features each supported, etc. Of course, there is no native ADAM web browser (your only option is an 80x24 serial terminal using dialup to a UNIX or other shell which supports lynx), but nowadays ADAMites are not so insistent that their ADAMs be used for *everything* :-)

Lunch was a "working" lunch served in the meeting room--Greek barbecue (yum!). The first of 3 afternoon sessions was to be by Jillian and the rest of the "Ladies Auxiliary" about ADAM Bomb II. Before it got started, Richard Clee asked me if I'd like to skip it and go with him to his house nearby to pick up some software that I had previously wanted to purchase from him. (Richard runs ADAM Services in Toronto, one of the last full- service ADAM vendors.) Not wanting to pass up an opportunity to visit the Chateau Clee (and not being that interested in ADAM Bomb II, worthy adventure game though it is), I agreed, and off we went, in Richard's new truck-that- looks-like-a-big-van.

Richard and Frances Clee live in a nice brick house (but Richard tells me that *all* the houses in Toronto are brick), with one small room packed to the gills (but *neatly* packed) with racks of ADAM disks, tapes, and systems. Rich found the items I wanted to purchase (mostly public-domain volumes of in-house Coleco test software) and commenced to make disk copies. I also picked up a replacement image of the Donkey Kong Super Game tape (my original tape got mangled in 1989 or so and could only play 2 screens before hitting the mangled spot and aborting), an original unopened copy of Recipe Filer (for debugging the ADAMserve problem), and some blank 5.25- inch 360K floppy disks. Copies made, we returned to the hotel.

When we got back, the second afternoon session had been running for 45 minutes! I had thought that ADAM Bomb II would go for a couple hours, but no, only one...and this session was one that I wanted very much to hear: why old computers are still interesting, moderated by Ralph Veecock from MTAG. The consensus is that "old" computers are interesting because there's so much that you can do with them at all levels, low and high. You can tinker with them and hack them; the whole machine is at your disposal. My comment at the end was that those whose first computers were "old" microcomputers were part of unique user communities, created during the "adaptive radiation" period of microcomputer evolution--before the IBM PC architecture made almost everything else become extinct. Those user communities have remained as important parts of peoples' lives, even as computer technology has moved onward. There are no "Pentium Users Groups" with the same zeal and interest as the old C-64 and TI-99 and ADAM users groups (and probably will not be again, until some unforseen advance in computers requires that users form communities again in order to use their computers).

The rest of the afternoon was spent in Guy Bona's session on ADAMem and ancillary utility programs for creating disk and tape image files from native ADAM media. Guy has written QuickBASIC and Visual BASIC wrappers to make it easier to create the appropriate image files (the command-line switches on the apps themselves are somewhat difficult to understand).

Supper was at the hotel restaurant at 6:00 PM. At 8:00 PM, those with PC computers who had managed to install NetZero free internet access software (thanks to Bob and Doug Slopsema for having the 4 install disks conveniently available) dialed up and logged in to Dale Wick's Spaniel Chat server for the traditional Saturday Night ADAMcon Chat. In the past, this was held via CompuServe; but when CIS went graphics-only, there was no way for 8-bits with simple terminals to connect any more, and the ADAMites left. Most of the convention attendees managed to be near an online computer, and we were joined remotely by some non-attendees (Ron Mitchell's son Jeff is the one I can remember off the top of my head). I bailed out after about an hour to go help Ron Mitchell and George Koczwara debug some trouble with PowerPaint HD. After trying a few things, we narrowed it down to a bad boot block, which was restored from backup, thus fixing the problem.

I can't remember what I did next...probably talked some more in the lounge, then went to bed. I know I was in by 11:30 PM, and Elanor had gone to sleep at least an hour before that.

## Sunday, 23 July 2000

Again the alarm clock went off at 6:00 AM, and again I went back to sleep, this time for about 45 minutes. I got Elanor up about 7:15 AM, and we were ready to go downstairs for breakfast by about 7:45 AM. Again we were among the first to arrive, and the rest trickled in even later. I don't think that Dale and Jillian came down until almost 8:45 AM. So, it was a little past 9:00 AM when we started the first session.

The morning session was by Peter Searle of MTAG, a hands-on workshop on SmartLOGO. Peter's presentation was superbly prepared, with demo programs on disk at each workstation to save typing time. Most of his session explored uses of recursion to produce fractal patterns, and also to do animation cycles.

Another working lunch in the meeting room (various open-faced sandwiches; the red salmon ones were tastiest!), and then my session on ADAMserve. I had come to the convention prepared to install it on Richard Clee's PC-XT, going so far as to swap in a 360K 5.25-inch drive in my 486 system (replacing a 1.2M drive) to make it able to write suitable disks (360Ks written in 1.2M drives are unreliable in genuine 360K drives). However, Richard decided to wait...and my presentation reduced to theory of design, "fun" aspects of the implementation, and description of the bugfixing I had done while at the convention (the serial baseport bug, the SmartFiler crashes). I think that all the ADAMem people with easy-to-transport laptops would love it if my ADAM floppy disk I/O code could be added to ADAMem, so that they wouldn't need to lug ADAM consoles and NTSC monitors around...maybe it can be done. Though Doug Slopsema has yet to figure out how to get Marcel de Kogel's source code to compile...and while ADAMserve runs fine under WinNT 4.0, as far as the serial link between ADAM and PC is concerned, none of the floppy disk I/O code does--as it talks directly to the floppy disk controller, something which WinNT does not permit. (I'm surprised that it allows the serial I/O, as that also is written to talk directly to the 16550 UART.)

Dale Wick then gave another in a continuing series of sessions (over several ADAMcons) about writing games for the ADAM in assembler and/or C. This year was how to design non-player characters for games (e.g., the ghosts that try to eat Pac-Man). It seems that implementing them as finite state machines is the best way to go, especially if you want the characters to have complicated behaviors. My small experience in writing games in BASIC is that it is easy to write "enemy" characters which are too ruthlessly efficient in defeating you, and hard to find the subtle balance between challenge and player winability.

At the end of Dale's session, and right before the break to get ready for the banquet, we held the annual A.N.N. (ADAM News Network) meeting. Of late, the major business of this meeting has been to discuss the desire for another ADAMcon, and if interest warrants, to determine where it should be held, and who is going to run it. (The A.N.N. meeting also was used to decide, a few years ago, to suspend the awarding of ADAM Gallery of Honor memberships, as it was felt that all of the known worthies had already been enshrined.) There is always some tension toward the end of an ADAMcon about whether or not *this* one is going to be the last one. Well, we keep saying every year, some year it will end, but not this year, not yet. I had come to Toronto with permission from my wife, Joan, to offer to take the next ADAMcon if (a) there was "enough" interest that people would actually come, and (b) nobody else wanted to do it. I spoke about this with Richard Clee when we went to his house, to Ron Mitchell and George Koczwara when we were debugging PowerPaint HD, and to the Slopsema clan at breakfast on Sunday morning. I received nothing but encouragement; so at the A.N.N. meeting, I asked for support to take ADAMcon 0Dh/13/XIII to Cleveland next summer. The assembled delegates said that they were not ready to call it quits, and that I should plan to see them all in Cleveland...thus, I can officially announce that there *will* be another ADAMcon...details to be announced.

<SHEER_SPECULATION> I have a feeling that we will be set for ADAMcons at least through ADAMcon 16. I *think* that Scott Gordon is interested in having ADAMcon 14, I *think* that the Slopsemas are interested in another one on their turf in Michigan, and I *think* that the Vancouver Island contingent want another crack at a west-coast ADAMcon (they lost out to Seattle last year).</SHEER_SPECULATION>

As I said while proposing ADAMcon 13: 19 July 2001 is my 15th wedding anniversary. Joan and I can go to a hotel for 4 days around that date, and I can put an ADAM in the back of the van. Anyone who wants to meet us there during that time, bring an ADAM and we'll call it an ADAMcon. Otherwise, Joan and I will have 4 days of vacation. Some day, we may have to do this to keep ADAMcons going...but evidently, not yet.

The annual banquet began at 6:00 PM. Everyone wore the new ADAMcon 0Ch T-shirt (which has our pink, non-Barney dinosaur mascot, named, of course, Adam).

[I should mention something about T-shirts...most of us attending ADAMcons nowadays have been to most of them, so we have all the past T-shirts and wear them throughout the current ADAMcon. My first convention was ADAMcon IV, and since then I have missed only one (ADAMcon 10, but I designed the shirt for it, so I got one). So, I changed my shirt 3 times per day, morning/afternoon/ evening: Friday IV 5 6, Saturday 007 VIII 9, Sunday 10 11 and 0Ch.]

Back to the banquet...it was prime rib (also somewhat traditional) and very good. We didn't have a special ADAMcon layer cake this year, but dessert was an excellent death-by-chocolate cake. After the banquet came raffle and door prize drawings. There was another afghan crocheted by Jean Stone (she also crocheted handbags for all the womenfolk; Elanor got one, and since there was one left over, Jean gave it to Elanor to give to her mom), and a couple of complete ADAM systems in the raffle. The final door prize was for stewardship of Adam the dinosaur mascot. Elanor, who had been helping to draw tickets, drew the winner: herself! So, Adam came home with us from Toronto. At the very end was the traditional passing of the banner to the chairman of the next ADAMcon (namely, me; Elanor and fellow Clevelander and ADAMcons IV and VIII chairman George Koczwara were also in the acceptance party), and then the group photo. Normally, this is taken by ADAMcon photohistorian Bart "Zonker" Lynch, and includes a last-second dive into the picture after setting the time delay shutter on his camera; but Zonker wasn't able to make it this year, so Neil Wick and George Koczwara stood in for him. George made it easy at last: he brought a tripod this year! In the past, the camera has been stacked up on whatever could be scrounged...so, no dramatics, but I'm sure it will be a clear, stable picture :-)

After the banquet, everyone retired to the lounge, where a silent auction was in progress on various boxed lots of ADAM stuff, sent by who I can't remember. I got 2 boxes, one with lots of game cartridges and a Roller Controller, the other with lots of data packs. After this, I decided that I would rather just get ready and leave in the morning, so I took down my computers and packed them up in the van that night. All that I left upstairs was my and Elanor's suitcases. In the morning we could eat and check out. Elanor went off to bed, and I followed soon after; lights out was around midnight.

Actually, before I took the computers down, and while the silent auction was still running, I helped Bob and Doug Slopsema figure out if the DynoMite Audio Digitizer cartridge they had won in the raffle (part of one of the complete ADAM system setups) had a real-time clock chip in it or not. My SmartBASIC 1.x interpreter can recognize 3 different kinds of ADAM clock cards, so I used SB1.x to determine that yes, their cartridge also had the clock chip installed. (There is a socket inside; the clock was an add-on which cost extra. The clock chip itself is one of the Dallas Semiconductor kind which fit under a socketed EPROM.)

## Monday, 24 July 2000

Alarm at 6:00 AM, actually up at 6:30 AM, Elanor up at 7:00 AM. Another Continental breakfast in the lounge across the hall. Except for Guy Bona, who had a very early flight out, everybody else came to the lounge at one point or another. We ate, chatted, and said our goodbyes; then checked out at about 9:00 AM. There was a gas station next door, so I went there and filled up; then we hit the road at about 9:30 AM.

The return trip was the same as the way there, except that I came back through Niagara Falls, so that we could stop on the Canadian side. Thus, we got off the QEW at 420 east and drove to Niagara Falls. Arriving at about 11:00 AM, we drove along the parkway, past Horseshoe falls, and found some parking available in front of a greenhouse (probably a quarter mile from the Falls). It was bright sunshine and warm, and the air was misty from the Falls. Elanor and I walked back to the Falls, taking pictures along the way. We stopped in the little pavillion next to the Falls, and had lunch, and bought some trinkets for my other 3 daughters. We walked most of the way up toward the Rainbow Bridge, so that we could get a good view of the American Falls; then we walked back to the car. Two guys from Australia asked me to take their picture; I did, after they showed me which button to press on their camera :-)

We left at 1:00 PM, taking Niagara Falls Boulevard (US-62A and US-62) to I-290 east, then to I-90 west, and back toward Ohio. We stopped again at the Angola service plaza to get some pop and use the bathrooms, then we drove the rest of the way back as we had come. I stopped for gas at our usual gas station close to home, and got home at about 5:45 PM. The return trip took about 6 hours (not counting the stops), and was 302.8 miles.

I didn't feel like unpacking the car right away, so I left it until the next evening :-)

---

See you again next week!

*Rich*

---

# This Week With My Coleco ADAM 9802.16

by Richard F. Drushel

## I. Return from Hiatus.

Thank you for your patience...it's been since 17 November 1997 that I have produced a *TWWMCA*. My end-of-semester teaching crunch hit about then, and didn't let up until Christmas; but I had let so many things slip at home that I used Christmas Break for them. And of course, the start of the spring semester has its own crunch. Now that all the plates are spinning again (like the juggler at the circus), I have some space in my schedule again for *TWWMCA*.

## II. Rescuing Some "Dead" ADAMs.

A local computer junkie, Fred Horvat (some of you may remember him from his brief stopover at ADAMcon IV), had some ADAM pieces and parts which he said were nonfunctional, taking up space, and needed to be moved somewhere... pack rat that I am, I told him I'd take what he had and see what I could do with it. Since he lives way on the west side of Cleveland, and I live on the east side, we decided to meet at a "neutral site" (a Blockbuster Video store he passes on his way home from work in Akron) for the pickup.

When I met Fred, it turned out he hadn't been able to find all the stuff he wanted to get rid of--only a couple of ADAM system units. Perhaps as some compensation, he brought me a working original IBM-PC system (the one with the cassette port, not the later XT) and color monitor. Welcome to 1981 `:-)` but I can put it to good use--each of my 4 daughters now has her own computer.

The ADAM system units were a bit dirty, and one was missing the lid covering the 3 expansion slots. Each had one tape drive and one of the all- plastic type of drive bay placeholders (as opposed to the earlier kind which had a metal frame). There were no major scuffs or scratches, but one of the system units had a half-inch hole in it around the disk drive ADAMnet port.

Standard operating procedure for me with new-found ADAMs (reported to be working or not) is to completely disassemble them, clean them up, then test them under power. A few minutes with a Phillips-head screwdriver took care of the disassembly part. The plastic case pieces were put into a laundry tub full of hot water and Tide detergent. The metal shielding was intact on one system unit, missing except for the bottom memory board shield on the other. They weren't very corroded (sometimes they get quite rusty), but had lots of dust, so I threw them into the detergent as well.

For those of you who've never been inside an ADAM, there are two circuit boards: a rectangular "memory board" on the bottom, and a square "video board" on top. The memory board has the 64K of RAM, the EOS and SmartWriter ROMs. the 3 internal expansion slots, the sideport expansion connector, the tape drive connectors, and the ADAMnet master 6801 microcontroller.

The video board has the video circuitry, the Z80 microprocessor, the ColecoVision OS-7 ROM, the game controller connectors, and the game cartridge slot. The video board is actually just a modified version of the ColecoVision video game board; the Expansion Module 3 version of the ADAM (which connects to a ColecoVision) has just the memory board. In the ADAM, there is a ribbon cable with edgecard connector between the memory and video boards.

The circuit boards of both system units were revision D1-D3; PROMs (no version labels) instead of EPROMS; 3-ROM configuration (8K EOS, 8K and 16K SmartWriter) instead of 4-ROM (8K EOS, 3 x 8K SmartWriter); master and tape 6801s socketed; memory I/O controller soldered; lots of rework on the memory board (extra jumpers, capacitors, cut traces). Based upon this, I expected to find R80 SmartWriter when (if) the machines were booted.

There was one surprise inside the system unit which had the hole in the case--a complete mouse nest made of pink fiberglass insulation! This and other goodies were stuffed into the space between the outside of the case and the back wall of the internal expansion bay. The case plastic is about a quarter inch thick; I can't imagine some rodent chewing away at that for very long...but I don't know how else to interpret it. My daughters were pretty grossed out when I showed them the nest.

The cases cleaned up nicely in the detergent. Sometimes you will find old stickum from labels which have fallen off, or from tape, which the detergent won't remove. For this, I use lighter fluid (naphtha) and paper towels. If the case is badly discolored (too long under fluorescent lights, or else owned by a smoker), there's not much you can do. I have read reports that Formula 409 stain remover plus one of those green Scotchbrite pot scrubbers can remove this kind of heavy staining (at the expense of the finish), but I've never tried it myself.

Finally, I was able to put the circuit boards back in (minus all the shielding) for the first powered tests. The first system booted right away, with good video. The second system had a scrambled screen, and the printer didn't give the normal carriage return/line feed expected when booting into the Electronic Typewriter. So, which board was bad--the memory board or the video board? I decided to do a swap test using the first system unit; clearly it had 2 good boards. Memory board 1 plus video board 2 --> system worked! So, it must be a bad memory board. However, the reciprocal test of memory board 2 plus video board 1 also worked! This suggested a dirty edgecard connector on the ribbon cable between the memory and video boards. Cleaning the traces with a pencil eraser and reseating the connector a few time cured the problem.

You might be thinking that it's dangerous to troubleshoot by swapping--isn't it possible that a bad board will damage a good board, leaving you with a stack of blown boards? Yes, it is possible. My experience has been, however, that as long as the boards don't have dead short circuits in them (i.e., a direct connection between supply voltage and ground), there isn't much real danger. Shorted boards can be identified by the presence of burned traces, burned wires, or literally blown/fried chips. Such a board must have any damaged components removed and replaced, and the source of the short identified and fixed, before applying power. Component-level troubleshooting is best left to experts.

Now that I was sure both systems worked electrically, I put everything back together properly, including all the shielding. For the system which was missing most of the shielding, I went to my ADAM junkbox and got the necessary pieces. I tested both systems again after the shielding was installed, but before the cover was screwed down, just to make sure I didn't have any loose connections--it's really annoying to keep unscrewing the case more than a few times in one sitting `:-)`

The last items to test were the tape drives. Plug them in, put in a tape (my Donkey Kong Super Game was the closest at hand), pull the reset switch, and pray...luckily, both drives booted successfully.

So, for the cost of a scoop of detergent and an afternoon's time, I was able to add two more ADAM system units to my collection. I have found many times, with many different kinds of machinery, that simple disassembly, cleaning, and reassembly resurrects many "dead" items. Remember this the next time your ADAM flakes out.

## III. Next Time: ADAM Schematics.

I have some ADAM schematic diagrams in .GIF format. They aren't scans of Coleco schematics; I redid them using Canvas 3.5 graphics software, so they are neat and legible `:-)`. I will put these up for download and talk about them next time.

See you again next week!

*Rich*

# This Week With My Coleco ADAM 9711.17

by Richard F. Drushel

## I. Another Long Week, Short Article.

I've done a lot of ADAM stuff this week, something almost every day. Unfortunately, it doesn't translate into a very long article. Most of the work is preparatory for some larger efforts which may end up as future articles, though, so it's not wasted.

## II. Source code for ADAMlink V.

George Koczwara asked me if I could produce an assembly code listing for ADAMlink V, preparatory to HLM-GMK seeking a formal copyright on the program. I suspect that this is related to the demonstrated piracy of the program by Terry Fowler of ADAM's House. I'm not a lawyer, and don't play one on TV, so I will not attempt to discuss the legal ramifications. Instead, I will briefly outline what I've been doing to create such an assemblable source code listing from the disparate "source" pieces which went into making ADAMlink V.

ADAMlink V began as ADAMlink IValpha, a major overhaul of ADAMlink III/III+ by Tom Clary which was never officially released, but which has evidently had some "underground" circulation. A few years ago, HLM-GMK Co. tracked down Mr. Clary and purchased the rights to ADAMlink IValpha. This buyout was supposed to have included the original source code (ADAMlink IValpha is a de novo creation with no inherited code from Coleco's original ADAMlink I; I have the disassemblies to prove it), but it never appeared. Thus, back in 1991 or so, I began to disassemble the ADAMlink IValpha binary to see how it worked, with the thought of adding some extra features to it (such as support for the Micro Innovations-type serial ports, disk/tape catalog features, EOS RAMdisk support, hard drive support, and some other amenities).

Using my old UNASMHEX-based program which had been ported to MS-DOS, I created the disassembly and edited it as a WordPerfect 5.0 file. I pretty much totally commented it (it's a 25K+ program, about as big as SmartBASIC 1.0) and made an elegant listing, which I laser-printed and used as a reference. I figured out where I could patch/break into the code to make the changes that I wanted, then used a crufty DOS-based Z80 cross assembler to assemble just the code patches. (This is the same assembler that I used to create the SmartBASIC 1.x source code--ugh!) Using the assembly listing files, which gave the resulting machine code in hexadecimal, I manually entered the changed bytes on an ADAMlink IValpha binary on an ADAM using the block editor of File Manager. Thus, while the patches exist as commented source code (which with minor editing can be made compatible with my current assembler, SLR's Z80ASM+), the final ADAMlink V program does not.

How could I get an assembly-ready listing of ADAMlink V? One possibility would be to use my current dissasembler (Kenneth Gielow's Z80DIS 2.2 for CP/M), using the ADAMlink IValpha disassembly and mini-ADAMlink V assemblies as a guide for where to place the breakpoints. The major problem with this method is that all the comments, laboriously typed in over a few months, would be missing...and I have no desire (and no time) to retype them from scratch.

The next method would be to somehow alter the existing ADAMlink IValpha disassembly to be in assembly-ready format. Doing this by hand in a text editor would be as laborious (and error-prone) as retyping all the comments on a Z80DIS 2.2 disassembly. The alternative would be some sort of "intelligent" filter program which could read the ADAMlink IValpha listing and automatically convert it into Z80ASM+ format. It would take a while to write and debug, but when thoroughly tested, it could be unleashed upon the disassembly and in one step generate assemblable code, comments intact. Then, since the ADAMlink V patches already exist as assemblable code, I could easily do the minor editing necessary to integrate the patches with the main ADAMlink IValpha source. The final test would be to run the integrated source through Z80ASM+ and verify that it produced a binary identical to that of the current ADAMlink V.

The bulk of this week has been spent in writing such a filter program, which I call DIS2ASM. As always with me, I'm writing it in QuickBASIC 4.5 for MS-DOS and compiling it to run on my 486DX2-66 system. Currently, the DIS2ASM program is about 50% completed, and the parts which are completed have been tested and validated. It will probably take me the rest of the week and next weekend to complete it. I will then have some small editing or "regularizing" of the ADAMlink IValpha disassembly file to do, in order to make sure that certain "special cases" (which would break my simple-as- possible code) don't exist. The final DIS2ASM run will probably take about 5 minutes to run...even with the 2-week development time, though, the result will be far more accurate and painless to achieve than manual retyping/ editing.

## II. Prospect for SmartBASIC 1.0 and 2.0 Source Code Listings.

While I won't be able to release the completed ADAMlink V code (not if HLM-GMK are copyrighting it), there is an indirect payoff in that I have completed (in 1990!) disassemblies of SmartBASIC 1.0 and 2.0 (including the `STDMEM` and `EXTMEM` versions for 2.0) which are in exactly the same UNASMHEX format as the ADAMlink IValpha listing. This means that (again, with some "regularizing" editing) I should be able to run these disassembly listings through DIS2ASM and regenerate commented, assemblable source for them as well! No promises that I'll get to this anytime soon, but it could be done.

## III. Prospect for an ADAMlink VI.

Having real source for ADAMlink V in hand would allow for future expansion and enhancement, including EOS-8 type support for the ADAMnet serial and parallel devices, a decent VT100 terminal emulator, and maybe even ZMODEM file transfer capability. No firm promises for this ADAMlink VI, but it would be possible. Consider it "vaporware" for the time being.

See you again next week!

*Rich*

# This Week With My Coleco ADAM 9711.09

by Richard F. Drushel

## I. Source Code for SmartBASIC 1.x.

This week's article will be very short, since most of my ADAM-related typing has been devoted to the object of the article. Namely, I have finished converting my regenerated source code for SmartBASIC 1.x into Z80ASM+ format. The completed source assembles to give a binary identical to that which I got by `BSAVE`ing the SB1.x memory block (with data areas being set to zero in the source, of course). The source is available as a PKZIP 2.04g archive for download:

- sb1x20y.zip, SmartBASIC 1.x version 20Y source code

When you unzip this, you'll get a biiiigggg file (481628 bytes) called SB1X20Y.ASM. This is an ASCII text version of the WordPerfect 5.0 document I used as the basis for my editing.

As I mentioned in the last *TWWMCA,* part of my rationale for releasing this source is so that it ends up in multiple copies in multiple locations, so some disaster here doesn't result in its being lost forever. So, if you have enough disk space (and connect time to download the 90K .ZIP archive), please grab it and squirrel it away somewhere.

The source is only partially commented, because I have not had time (since 1994, the last time I worked on it) to type in all the handwritten comments I have on the original hand-assembled source code (and the hand-commented disassembly of SmartBASIC 1.0 from January 1990 or so!). So, it clearly violates all the Good Programming Practices that I've harped on previously...but at the time I did the disassemblies and reverse-engineering, the only computer I had access to was my Tandy 2800HD 286-12MHz laptop with a 20 MB hard drive. On that same machine, it takes 12 minutes to assemble the SB1X20Y.ASM source to binary using Z80ASM+ under the Z80MU.EXE CP/M emulator.

Remember also that this source code has not been reorganized to get rid of the spaghetti leftovers from my patching, and also from the original spaghetti in SmartBASIC 1.0. The next step is to do this reorganization, which probably could reduce the size of the program by 2K at least, probably more. This extra space would become available as extra workspace for user programs, or else for extra features in SB1.x.

## II. Next Time.

I don't know what's up for next time; goodness knows I have enough to do around here!

See you next week!

*Rich*

# This Week With My Coleco ADAM 9711.05

by Richard F. Drushel

## I. Administrivia.

I'm behind a week in my *TWWMCA* series because of a non-maskable interrupt from 2 weekends ago. In brief, my kids got wild playing in the kitchen and knocked over a plant sun-lamp onto the floor, shade-down, which destroyed some of the 25-year-old floor tile when it sat there undetected and hot for several minutes...guess who had to spend 18 hours retiling the entire floor?

We were lucky, though, because the old tile were *asbestos* (yes, that great flame-proof and falsely-maligned-as-more-carcinogenic-than-plutonium building material). If they had been the modern, not-(yet)-proven-to-cause-cancer-in-some-damned-rodent vinyl or polyurethane tile, they would have ignited quickly and we would be now be living in a Salvation Army shelter...

Anyhow, the Drushel household (and its attendant Coleco ADAM museum) is still in good shape, and the junior inhabitants have gotten a good scare... and they also owe me $132 when they get their first jobs `:-)`

## II. SmartWriter Reverse Engineering Progress.

My project to reverse engineer R80 SmartWriter to assemblable source code and make some bugfixes is stalled where it was before the floor tile adventure. That is, I have it patched to recognize drive D (disk 2, ADAMnet device 5), and it will be able to recognize an EOS RAMdisk (device 26) whenever EOS gets evolved enough to install it automatically at boot. There are some SmartKey menu inconsistencies between the GET FILE menus and the various STORE menus which I need to work out. Basically, I want the STORE menus to work exactly like the GET FILE menus. The reason for this is that, in the GET FILE setup, I can easily accommodate extra drives as SmartKey choices; while I can't under the current STORE menu arrangement. This is just time and tinkering. I still plan to put up the fixed binary for download for people to play with under Marcel de Kogel's ADAMEM emulator.

## III. Assembly Source for SmartBASIC 1.x.

While reorganizing the files on the computer that used to be my Tandy 2800 HD laptop (a 286-12 mHz whose guts now reside in a stand-alone CPU box after the built-in keyboard and LCD screen died; see *TWWMCA* 9609.30 in the archive for the whole story), I discovered a copy of the reverse-engineered source code for my SmartBASIC 1.x interpreter, from 1991. It is in the format for a very junky cross-assembler (runs under MS-DOS, writes Z80 binaries) which I found long ago at the famed (defunct) SIMTEL ftp archive at wsmr-simtel20.army.mil. I couldn't quickly lay my hands on the cross assembler, so by bits and pieces at lunchtime the last couple of weeks I've been editing it so it can assemble correctly under my current assembler, Z80ASM+

from SLR. I now have it down to only 2 errors during assembly. When I get it totally converted, I'm going to put the source up for anonymous ftp. It's fairly well commented, but it's not great, and not up to my current standards for documentation.

One of the reasons that it is so ugly is that it will exactly regenerate every single binary patch that was made to the original SmartBASIC 1.0 binary. I haven't done any cleanup or reorganization of the code, so it still has the same memory map as SmartBASIC 1.0 as described in Ben Hinkle's "The Hacker's Guide to ADAM Volume II". Another reason for its ugliness is that SmartBASIC 1.x was created without the benefit of an assembler program--I wrote code on paper, assembled it by hand, wrote the decimal values of the machine code as `DATA` statements, `POKE`d them in, and finally `BSAVE`d the entire binary. My paper code has lots of comments, but at the time I abandoned work on the reverse- engineered electronic version, I had typed in only some of these comments. For the rest, I still need to dig out the original paper source from 1990 and early 1991.

As you might imagine, the near-disaster in my kitchen has made me rather uncomfortable about the amount of irreplacable ADAM code and documentation that I'm sitting on here. I want to get some of this stuff out in multiple copies (preferably electronic) so that it doesn't get lost forever in some accident. And already I'm finding that I've misplaced things; to fulfill a Xerox request, I've been looking for Volume I of my *ADAM Technical Manual*, and I *can't* find it! I have Volume II with all the EOS-6 and OS-7PRIME source, but not Volume I with all the hardware descriptions, ADAMnet and EOS Users Manuals, and the descriptions of the ADAMlink modem and SuperGame software loaders. This is the 2-volume set that Shaun McCallum was selling at ADAMcon IV, and the only other set I know of is Chris Braymen's...

So, once I put this stuff up for download, please grab it and archive it away somewhere! As far as SB1.x source is concerned, I doubt I will ever sell another copy, so if you have the SLR assembler and want to play with it, go ahead--just don't sell it. The SB1.x manual source (a WordPerfect 5.0 file) I am holding onto for the time being, though (if HLM-GMK don't shoot me for giving away something they still sell) I might consider converting it into HTML and letting Dale Wick put it up on his ADAM Home Page (or include it with an ADAM CD/ROM collection, if such a thing ever gets made).

## IV. Disassembly of the ADAM CP/M 2.2 BIOS.

Also on the ex-laptop hard disk was a long-forgotten disassembly I had made in August 1992 of the ADAM CP/M 2.2 BIOS. I even remember how I made the disassembly: I used the CP/M `SAVE` command to dump the 64K RAM contents to disk, then used the IMP modem program to transfer it by direct serial link to my then-laptop. I then ran the BIOS portion through my UNASMHEX9.EXE disassembler (a SmartBASIC 1.x program converted to Microsoft QuickBASIC 4.5), and started to edit it in WordPerfect 5.0 for MS-DOS. I think the ADAM CP/M manual told what memory address the BIOS started at, or maybe I just looked around for the BIOS jump table; this part I don't remember. Anyway, the listing starts at the BIOS jump table. Since UNASMHEX9.EXE was never intended to generate assemblable source code, the disassembly listing can't directly be used to reverse-engineer the ADAM CP/M 2.2 BIOS `:-(`

More's the pity, because I commented some parts of it very thoroughly, specifically those relating to the AUX:, PUN:, and RDR: devices, which (according to the manual) were for use by the ADAMnet serial/parallel board (which was prototyped but never sold). As I recall now, my rationale was to figure out from this (presumably working and tested) code how the serial/ parallel board worked, so that I could emulate it correctly (on 3rd-party serial boards) in my new EOS-8. That I did so correctly is borne out by the fact that I could run SmartBASIC 2.0 under EOS-8 and use a serial terminal for console I/O using `IN#4` and `PR#4` (which were vectors to the ADAMnet serial device).

I did at that time, however, actually make a separate file of the BIOS code used for the serial board (AUX:, supplemented by disassembling the part of CONFIG.COM which actually sets the serial port baudrate, parity, etc., since the BIOS does not provide those functions directly). This is a nice package, so I'm putting it up for download:

- dev14.lst, ADAM CP/M 2.2. BIOS code for prototype ADAMnet serial board (device 14)

Seeing the final commented disassembly gives little hint how difficult it was to actually deduce from code alone how the thing worked `:-)`

I'm going to hold onto the disassembly listing of the entire BIOS for a little longer, in the hopes of cleaning it up somewhat. There are still some large areas which are pretty "raw" (e.g., data areas which aren't resolved as data but are still present as "garbage" assembly code).

## V. SmartBASIC 1.x and Clock Bugs?

Ron Mitchell has posted (to the mailing list) an interesting account of problems he's had with the SmartBASIC 1.x clock drivers, namely, the weekdays being off by one for certain clock hardware types. I haven't had time to investigate in detail. I know for *certain* that the motherboard ROM-based no-slot clock driver works perfectly well, because that's the clock type I have in my Mini Wini hard drive system. From Chris Braymen, I inherited one of Syd Carter's Dy-No-Mite Sound Digitizer/Clock cartridges; but there is no clock chip installed, so I can't immediately test it. As for the Eve/Orphanware clock, I never owned one.

Historically, my SB1.x clock driver code was written and tested using

1. my own no-slot clock, obtained from Steve Major's ADAM Connection (I got it free for purchasing so many $$$ of other stuff);
2. a Digitizer/Clock cartridge borrowed from Alan Neeley, for the express purchase of writing the driver code; and
3. the original wire-wrap prototype of the Orphanware clock, built by Orphanware founder John Lingrel and now owned by Herman Mason.

All these worthies are credited in the Acknowledgements section of the SB1.x manual, if memory serves. So far as I can remember, the final driver code worked properly with all three clock types. I will have to buy a Dallas SmartWatch from JDR Microdevices for my

Digitizer/Clock and beg/borrow the Orphanware clock prototype from Herman Mason to definitively address Ron Mitchell's bug report.

See you next week!

*Rich*

---

# This Week With My Coleco ADAM 9710.20

By Richard F. Drushel

## I. Status Report: SmartWriter Disassembly.

I'm happy to report that yesterday I finished my complete disassembly and reverse-engineering to assemblable source code of SmartWriter R80. This was done using Kenneth Gielow's Z80DIS 2.2 under the Z80MU CP/M emulator for MS-DOS. The new source is formatted for the Z80ASM+ assembler from SLR. The new source (minimally commented; it has all the EOS global symbols in it, as well as a few things I added on my own) is available for download:

- sw_r80.zip, regenerated source for SmartWriter R80

This file is a PKZIP 2.04g compressed version of the ASCII plaintext source. Note that it has no bugfixes or changes; it will exactly regenerate the original SmartWriter R80 binary, warts and all. Of course, now this source code can serve as the basis for bugfixes/upgrades to SmartWriter (including making it XRAM-friendly for a possible XRAM-based EOS).

## II. Verifying the New Source Code.

How can I be sure that I correctly reverse-engineered the source code? The new SmartWriter source was verified in two ways:

1. the new source was reassembled to a 32K binary image using Z80ASM+, then compared byte-by-byte with the original binary image of the R80 SmartWriter ROM. Except for padding out the unused ROM space with 00H instead of 0FFH, my new binary was identical to the old.
2. the new source was modified by inserting a `NOP` instruction after the first instruction, reassembled to a binary image, then tested using Marcel de Kogel's ADAM Emulator for MS-DOS, ADAMEM.EXE. The effect of the `NOP` is to displace all the code forward by one byte, without affecting the program operation (`NOP` means No OPeration, it doesn't do anything). Extensive testing of the new binary revealed no operational errors.

What is the rationale for (2)? In the code, there are many references to ROM data areas (e.g., message strings, sprite patterns), which of course are to the addresses in ROM at which those data sit. In a typical program, executable code and non-executable data are intermingled in the binary image. Since a disassembler program doesn't know anything about the internal structure of the binary it's disassembling, data areas often appear as "garbage" machine code instructions in the raw disassembly output. It is the task of the human disassembler to recognize such "garbage" code and edit the listing to properly mark it as data. Any data references which aren't caught, however, will no longer point to the proper place if the code length changes in editing, and the data doesn't end up in exactly the same place in ROM that it originally occupied. Thus, inserting a NOP to shift the reading frame of the code, then testing the new binary for errors, will reveal any missing data references as program bugs.

The initial `NOP`-shifted binary did not boot at all--I got a black screen. Clearly, I had missed some data references. I then started from the end of the program and worked forward, inserting a `NOP`, reassembling, and then retesting. `NOP`s inserted towards the end allowed the binary to boot, but the screen was a bit scrambled. One bug at a time--track down what kept it from booting first, and worry about the screen garbage later.

By repeated `NOP`-shifting, always bisecting the interval between what would boot and what gave the black screen, I found the place in the code where I had missed a data reference. In the code which loaded EOS from ROM to high RAM, the start address of the ROM was listed as "program label 6000H" instead of just the number 6000H. Any `NOP` inserted after 6000H wouldn't cause a problem, but a `NOP` before 6000H would cause "program label 6000H" to have the *value* 6001H--which would cause the whole of EOS to be copied to the wrong location in RAM, and thus crash as soon as any EOS function call was attempted. Fixing this in the new source allowed SmartWriter to boot--but with a scrambled screen, as noted above.

The scrambled-screen problems were due to references to a few video RAM (VRAM) addresses which happened to have the same numeric value as some program labels. (The disassembler just sees the final number, and if there is a program label with that number, it substitutes that label name in the code.) During my initial cleanup of the raw disassembly listing, I had recognized and fixed a few of these anyway, from my familiarity with how VRAM is typically used; but some others less obvious slipped through the cracks. Again, it was repeated `NOP`-shifting (always bisecting the interval between a shift that "broke" the program and one which had no effect) which allowed me to zero in on the references to program labels which needed to be changed into numbers. (The strategy is exactly like that in the "higher-lower" game, in which one person thinks of a number, and a second person tries to guess it, the first person answering only "higher" or "lower" depending upon the accuracy of the guesses.)

When the screen was unscrambled, there remained a small glitch: in the ADAM'S ELECTRONIC TYPEWRITER mode, the message string for SmartKey V was off by a few letters if I `NOP`-shifted the code at the beginning. Using the same procedure described above, I zeroed in on the offending part of the code, but couldn't see what was causing the problem: the critical address was the very address of the particular SmartKey V message string!

To make a long story short, the problem was that I had not correctly separated code from data areas in the subroutine which draws the SmartKey menus for the ELECTRONIC TYPEWRITER. In SmartWriter, SmartKey menus are drawn using an economical yet uncongenial-to-disassembly subroutine which includes the address of the string directly after the `CALL` to the menu- drawing routine. Here's the critical bit of code, correctly resolved:

```
C.0B07:
      CALL  DRAW_YELLOW_SK_4
;
      DW    I$12FC
;     -----------------
```

```
;
      CALL  DRAW_SK_5
;
      DW    I$132D
;     -----------------
;
      CALL  DRAW_SK_6
;
      DW    I$131C
;     -----------------
;
      RET
;
;     -----------------

;[...]

I$12FC:
      DB    ' ADAM''S ',0DH                 ;0DH is carriage return
      DB    ' ELECTRONIC TYPEWRITER',03H    ;03H is ETX, end-of-text marker
I$131C:
      DB    ' MARGIN',0DH
      DB    ' RELEASE',03H
I$132D:
      DB    ' MARGIN/',0DH
      DB    'TAB/ETC',03H
```

The subroutines of the form `CALL DRAW_YELLOW_SK_x` draw a yellow box from SmartKey I to SmartKey x, with the text of string at the address following the `CALL` printed in black. The subroutines of the form `CALL DRAW_SK_x` draw a blue (2 possible shades, depending upon which SmartKey) box in SmartKey x's space, with the text of the string at the address following the `CALL` printed in white. The `DW LABEL$` assembly directive means to store a Data Word containing the address of the program label `LABEL$`, whatever value `LABEL$` has at assembly time.

The first time through the disassembler, however, all these "inline" data references get treated as code. When I first recognized the existence of the `DRAW_YELLOW_SK_x` and `DRAW_SK_x` family of functions, I searched through the code listing and (supposedly) changed all the "garbage" code to DW references. In the case of the 'MARGIN/TAB/ETC' string, however, I missed one; and the (2DH, 13H) was treated as the following assembly code:

```
      CALL  DRAW_SK_5
;
      DEC  L               ;2DH
      INC  DE              ;13H
```

This meant that, when I `NOP`-shifted the program anywhere prior to `I$132D`, the new value of this label was *not* updated in what should have been the data reference after the `CALL DRAW_SK_5`. The result was that this `DRAW_SK_5` would *always* try to draw a SmartKey with whatever data it found at address 132DH, even if the actual message string had shifted to a different absolute address. Needless to say, putting in the correct reference as `DW I$132D` fixed the problem; and now I could edit the new SmartWriter source at will, without fear of such frameshift errors being introduced by the edits.

## III. Code Fossil: ADAMlink Modem I/O in SmartWriter.

My first disassembly of SmartWriter was done in 1992 or thereabouts, using the UNASMHEX disassembler I wrote for SmartBASIC 1.0 and 1.x (and ported to Microsoft BASIC). This disassembler did not make any attempt to generate assemblable source code; but I used it extensively, and it was my primary tool for figuring out how to patch PowerPaint for hard drives, HARDDISK 2.3 and 3.9 for RAMdisks, and ADAMlink IValpha to make ADAMlink V (not to mention SmartBASIC 1.0 itself to make SmartBASIC 1.x). I made a quick pass through SmartWriter just for fun, I guess, and at that time had discovered a truly amazing code fossil: SmartWriter knows about the ADAMlink modem, and tries to talk to it at startup!

I believe that I wrote a brief report of this discovery for A.N.N., but I can't be completely sure. At any rate, during the resourcing of SmartWriter, I came across the code again; so here it is, with some brief explanations:

```
MODEM_CTRL_PORT   EQU   5EH   ;ADAMlink modem control port (2651 UART)
MODEM_DATA_PORT   EQU   5FH   ;ADAMlink modem data port (2651 UART)

;      -----------------

;Load a program through ADAMlink modem!
;At least that's the intent.  It can't work, however, because
;there is no handshaking for any of the data reads; a 300 bps
;modem can't supply the characters that fast.  Maybe if the
;proper handshaking were added...it would be interesting.

C$103C:
      LD    BC,(D$1071)
      LD    A,B
      OUT   (C),A              ;send first init command
      LD    BC,(D$1073)
      LD    A,B
      OUT   (C),A              ;send second init command
      LD    A,(D$1075)
      LD    C,A                ;get data port
      IN    A,(C)              ;read for first magic number
      CP    0AAH               ;was it magic1?
      JP    NZ,J.1070          ;no, so exit
;
      IN    A,(C)              ;yes, so read second magic number
      CP    55H                ;was it magic2?
      JP    NZ,J.1070          ;no, so exit
;
      IN    E,(C)              ;yes, so get DE=count
      IN    D,(C)
      IN    L,(C)              ;get HL=load address
      IN    H,(C)
      PUSH  HL                 ;save start vector
J$1065:
      IN    A,(C)                ;read a byte
      LD    (HL),A               ;save it
      INC   HL                   ;point to next address
      DEC   DE                   ;one less count
      LD    A,E
      OR    D                    ;is D=E=0?
      JR    NZ,J$1065            ;no, so keep going 'til done
;
      POP   HL                   ;yes, so restore load address
      JP    (HL)                 ;and jump to program start
;
;     -----------------
J.1070:
      RET                        ;exit
;
;     -----------------

;initialization data for ADAMlink modem

D$1071:
      DB    MODEM_CTRL_PORT,00H      ;port, data
D$1073:
      DB    MODEM_CTRL_PORT,80H      ;port, data
D$1075:
      DB    MODEM_DATA_PORT          ;port
;     -----------------
```

As noted in the code comments, there's no way that this can work with the ADAMlink modem as we now have it; there's no handshaking for any of the modem data port reads. So, if you try to set this up now, with a second computer ready to send a program at 300 bps, it will fail. Now that I have reassemblable source for SmartWriter, though, it might be worth trying to make this code workable, just for kicks.

Why is this routine in here, to load and run a file over the ADAMlink modem? At this late date, only the SmartWriter programmers themselves could answer definitively; but I will offer this intriguing speculation as food for thought:

What if Coleco planned to start a nationwide BBS service, like CompuServe? They would have support areas for the ColecoVision and the ADAM, news about upcoming hardware and software, etc. Presumably the only way you could login would be via an ADAMlink modem (there were no third-party ADAMlink modems or serial interface boards to use standard Hayes-type modems). What if you wanted to try out a demo version of some new ColecoVision game or ADAM program, while you were on-line? In the appropriate forum, you could issue a command to start downloading a program (after making sure that your ADAMlink boot tape or disk was out of the drive), then pull the reset switch on your ADAM. Since you're still on-line, and loader code in SmartWriter doesn't hang up the phone, you could start receiving the file. This loader would grab the file, put it in RAM at the appropriate place, then run it. Presumably, the demo program would have a software escape mechanism to reboot the machine under ADAMlink so you could continue your online session. And this would be "safe" for Coleco because there would be no way for the users to save the program they had downloaded, because it wasn't downloaded to disk or tape, but rather directly to RAM, from which there is no escape (the ADAM not having a built-in debugger or program tracer).

In many respects, the Coleco ADAM is to the 1980s what TV-based "network computers" are to the 1990s. Technically, however, there is no way that 300 bps could have efficiently transmitted multimedia-type data. Even a 4K demo program would take over 2 minutes to download at 300 bps 8N1. I predict that "network computers" will be as commercially successful as the ADAM...

## IV. Possible Repair of SmartWriter R80 Bugs.

Now that SmartWriter can be changed at the source level (meaning, repairs are easy and can be as extensive as desired), you should all send me your lists of long-standing bugs to fix. The one that I remember is the inability to access disk 2 (drive D). I know where in the code the tape/disk detects are, and it looks to be a straightforward fix to add support for disk 2. (I may even start playing with this tonight, after my kids are in bed.) I also seem to recall something to do with there being an extra linefeed or half-linefeed at the bottom of pages during printing; if someone can send me a detailed description of how to produce this bug, I'd be grateful).

From the PRINT menu, there is an unused SmartKey menu entry for PRINT FILE. Making it print A-type (plain ASCII) files is trivial; making it interpret an existing SmartWriter file without actually loading it in and printing it from the workspace is probably a lot harder, but doable (I haven't looked).

Also, it might be nice to have native support for the PIA2-type parallel printer card (instead of having to run FASTPATCH or some equivalent). Again, adding the driver code is easy--and I've even published the core parallel printer driver in a previous *TWWMCA* (070907). Of course, anything requiring specific escape codes for parallel printers is out. A 132-column mode would probably involve deciphering and documenting SmartWriter's RAM data tables above 8000H, which I haven't done and wouldn't particularly relish doing.

It is also very easy to disable the XRAM detect routine, which would prevent SmartWriter from recognizing XRAM at all, thus making it more friendly to future XRAM-based operating system upgrades. If disk 2 support is being added, it probably would be worth adding RAMdisk (device 26) support at the EOS level (leaving it to the EOS RAMdisk driver to worry about how to manage the different memory bank switches that would be necessary).

So, send me your bug lists for SmartWriter! There are more than 3751 bytes free at the end of the 32K ROM to play with, which is lots of space in Z80 machine code!

See you next week!

*Rich*

# This Week With My Coleco ADAM 9710.13

by Richard F. Drushel

## I. Prelude to Commentary on *Ron'sWeek'n'ADAM* 9710.05

As promised in last week's *TWWMCA* (9710.06), I will respond here to some interesting and important questions raised by Ron Mitchell in the 5 October 1997 installment of his article series, *Ron's Week'n'ADAM.* I will do this in USENET style, with quotes from Ron's article prefaced by >, and my commentary following. This is also known as *Point-Counterpoint* style, after the old segment on the CBS news magazine *60 Minutes* featuring opposing viewpoints by (I think) James Kilpatrick (a curmudgeon) and Shana Alexander (Bambi in the headlights of the oncoming truck). Unlike the originals (who inspired a parody on NBC's *Saturday Night Live* in which Dan Akroyd, in the Kilpatrick role, responded to Jane Curtin with "Jane, you ignorant slut!"), I will not be mean or nasty. While Ron says, about his original article,

```
>There is gut reaction
>and there is, I think, more mellow and considered comment.
```

I have no difficulty in looking past the "gut reaction" and recognizing the "mellow and considered comment".

The following analysis is intended to be objective. It may come across as cold-blooded and cruel, but that is a by-product of my desire to be objective, not from a desire to denigrate the work of others. I'm putting on my "scientist" writing hat here, so my own work is as much under the gun as anybody else's.

## II. TDOS as an Example of Unmaintainable "Spaghetti Code".

A few *TWWMCA*s ago, in a survey of ADAM operating systems, I opined that TDOS (Tony's DOS, a CP/M-compatible operating system) in its current source code state is an unmaintainable mess of "spaghetti code", and that it would be better to start anew to build an ADAM-compatible CP/M than to make further use of the existing TDOS source code. To which Ron responded:

```
>The part about TDOS in Rich's work may indeed be accurate, but
>the manner in which it is presented leaves the impression that
>TDOS co-authors Tony Morehen and Guy Cousineau somehow produced a
>product that was less than satisfactory. Having known these two
>talented men as friends for several years, I do not believe they
>wrote any 'spaghetti' code whatever. I believe that they wrote
>code that made sense to them, and I know that code continues to
>perform yeoman service on several ADAM computers here. In fact,
>had it not been for TDOS, I would have dumped ADAM long ago.
```

I'm sure Ron knows this anyway, but I'll say it for the record: I am *not* knocking Tony Morehen and Guy Cousineau. Both are skilled programmers whose works are still in active, productive use. The ADAM community would be much diminished were it not for their efforts.

However, both Tony and Guy have moved on, leaving behind their programs for others (namely, me) to maintain. What does "maintain" mean? It means:

1. fix any bugs that may be discovered;
2. modify to support additional hardware that may be invented;
3. modify to add extra features that may be desired by users; and
4. modify to coexist with any changes to system software (e.g., operating system patches) which may be required by applying (1)-(3) to the operating system itself.

All program maintenance requires adding or modifying program code. This may be done in two ways:

1. modify the source (SmartBASIC, C, Pascal, Z80 assembler) and recompile/reassemble to a new executable binary file; or
2. patch the current executable binary file directly.

Modifying the source is the method of choice, because the source is typically written in a high-level language (like SmartBASIC) or, if in Z80 assembler, with useful variable names and subroutine names and some comments to explain how the program works. If all you have is a binary file, however, you must disassemble the binary to recreate a source file, then attempt to follow the program flow yourself (as if you were a Z80 CPU), and attempt to recreate and discover the thinking of the programmer. Even if you aren't doing a complete reverse-engineering of the program to source code (rather just disassembling what you think is the part you need to know about for a particular patch or bugfix), you have a difficult task--even with a "smart" disassembler tool like Kenneth Gielow's Z80DIS22 (which runs under CP/M).

In any case, whether you have the original source or a reverse-engineered source, your task as maintainer of someone else's code is to be able to read the code well enough to know what it's doing, how it's doing it, and whether there are any important side effects, so that the changes you make to it (1) do what you intend, and (2) don't break anything else by accident.

This task is greatly influenced by the underlying organization of the original code. If the original code is well-organized, hierarchical, modular, with related subroutines located close together, then it is easy to read. If, however, the code is cluttered, inconsistent, tricky (e.g., uses overlapping reading frames, self-modifying code, self-relocating code, overlays, conditional assembly, or many disjoint source files), and jumps all over the place, then it is very difficult to read. If the original source lacks comments (text explanations of what the program is doing at a particular point, or overviews of its organization), then the source is little better than a disassembly listing in terms of its readability.

The latter kind of code is the programming style referred to as "spaghetti", because like a plate of spaghetti noodles, it winds and twists and turns all around, with little hint of organization (and no comments or explanations from the programmer as to why it works or even what it's trying to do).

The TDOS source is such a "spaghetti" program, because

1. its internals are mostly uncommented;
2. it has an "inline" versus "modular" design;
3. it has a confusing amount of conditional assembly; and
4. the source is broken up into a large number of files without an obvious organization.

# III. Principles of Good Code Design.

Ron asks:

```
>1) [...By what standard is [spaghetti code] judged?
```

The answer is, by the people who have to maintain the code `:-)` Writing code in any assembly language is a difficult task, and even the best-designed, best-commented code may be rather unintelligible to its author 6 months later. (This is absolutely true for me.) Anything which makes the code easier to understand, be it comments or helpful variable names, is a boon to the code maintainer (even if that is the original author).

```
>2) What activities are included in the process of managing or
>   maintaining source code, and why are they important, given
>   that the product is out there and serving?
```

A program can be perfectly functional and easy-to-use from the end user standpoint, yet a nightmare of twisted "spaghetti" on the inside, completely hidden from the end user. If the end users are completely satisfied with the current operation of the program, and never demand either a change in its features or a change in its operating environment, then there is nothing to maintain, and the internal organization of the program is irrelevant. It is only when a maintainer is asked to fix a bug, or add a new feature, or ensure a backward compatibility, that the aims of the end user and the aims of the maintainer come into conflict. The first bugfix or hack or two can usually be made easily; but repeated patching upon patching results in a very "fragile" program--like a house of cards, the next card added may bring the entire structure down. And patches may have unforseen consequences which limit future patches.

```
>3) What is modular code, and why is having too many modular
>   source files a bad thing? After all, shouldn't we  be
>   carrying only the baggage that we need for a given application?
```

"Modular code" means that the code is organized into basic building blocks of functionality, and that unlike functions are not grouped together. The modules are often hierarchical, and higher-level functions are built up from calls to lower-level functions. Consider the following functional organization of the file system of ADAM's EOS operating system:

## High-Level Functions:

| | |
|---|---|
| _MAKE_FILE | create a file |
| _DELETE_FILE | delete a file |
| _OPEN_FILE | open a file |
| _READ_FILE | read from a file |
| _WRITE_FILE | write to a file |
| _CLOSE_FILE | close an open file |
| _POSITION_FILE | move the read/write pointer in a file |
| _RESET_FILE | reset the read/write pointer to the beginning |
| _TRIM_FILE | free up unused blocks at the end of a file |
| _QUERY_FILE | ind a file without file type |
| _FILE_QUERY | find a file with file type |
| _FMGR_INIT | initialize the file control blocks (FCBs) |
| _INIT_TAPE_DIR | initialize the directory of a disk or tape |

## Middle-Level Functions:

| | |
|---|---|
| _SCAN_FOR_FILE | see if a file exists |
| _SET_FILE | update the directory entry for a file |
| _CHECK_FCB | see if a FCB is in use (i.e., if a file is open) |
| _MODE_CHECK | see if an open file is open for read/write/execution |

### High-Level Block I/O:

| | |
|---|---|
| `_READ_BLOCK` | read a block from disk or tape (with 3 retries) |
| `_WRITE_BLOCK` | write a block to disk or tape (with 3 retries) |

### Low-Level Block I/O:

| | |
|---|---|
| `_RD_1_BLOCK` | read a block from disk or tape (no retries) |
| `_WR_1_BLOCK` | write a block to disk or tape (no retries) |

### Low-Level Hardware I/O:

| | |
|---|---|
| `_START_RD_1_BLOCK` | ADAMnet: start reading a block |
| `_END_RD_1_BLOCK` | ADAMnet: check if we're done reading a block |
| `_START_WR_1_BLOCK` | ADAMnet: start writing a block |
| `_END_WR_1_BLOCK` | ADAMnet: check if we're done writing a block |
| `_FIND_DCB` | ADAMnet: find the device control block (DCB) |
| `_REQUEST_STATUS` | ADAMnet: check the ADAMnet status of a device |
| `_SOFT_RES_DEV` | ADAMnet: reset a device |
| `_READ_DEV_DEP_STAT` | ADAMnet: read device-dependent status |

Note that higher levels provide more and more abstraction and insulation from the underlying hardware. This allows for portability for programs which only use the high levels; the low levels can be replaced with driver code for totally-different, non-ADAMnet hardware, as long as the same parameter passing conventions are observed. For example, ADAMserve breaks into `_RD_1_BLOCK` and `_WR_1_BLOCK` to reroute requests for ADAMnet device I/O to server devices, if appropriate. (It also maintains some dummy DCBs for the benefit of programs that try to read status data from ADAMnet directly by `PEEK`ing at the DCBs.)

To return to TDOS, part of the reason that I cannot "easily" come up with an ADAMserve version of TDOS, even given the complete original source code, is that nowhere in TDOS is there the equivalent of `_RD_1_BLOCK` and `_WR_1_BLOCK`. Perhaps to save space or gain speed, TDOS talks directly to ADAMnet, without any layers of abstraction like `_READ_BLOCK` (for block devices like disks and tapes) or `_READ_CHAR_DEV` (for character devices like keyboards and printers). I have not been able to spare the time or mental effort to find a "safe" way to either add this layer of abstraction or branch around the direct ADAMnet I/O code. It is not impossible, just not as straightforward and guaranteed-to-be-foolproof as rerouting `_RD_1_BLOCK` and `_WR_1_BLOCK` in EOS.

These direct accesses to hardware are called "inline" program design, because the code isn't written as a subroutine which is `CALL`ed; it's in the direct line of program flow. When optimizing a program for speed, inline design is the way to go, because you avoid the overhead of the `CALL` and the `RET`urn from the subroutine. If you must do the same thing at several different points in the program, however, you end up repeating the same bits of code over and over; thus, the program is larger than if you have a single subroutine which is `CALL`ed whenever needed. Subroutines are a way to optimize a program for size, because the overall code takes fewer bytes of machine code.

The programmer must decide where to optimize for speed and where to optimize for size. Optimizing for size can create dense, tricky, unreadable code that exploits every possible side effect of each machine code instruction. This is where helpful comments that describe the intent of the code, perhaps even show it in a straightforward, unoptimized version, are invaluable to the code maintainer. Optimizing for speed is important for timing-dependent operations (such as reading characters from a serial port at high bitrate and storing them in a buffer). For instance, the code in the terminal mode of ADAMlink IValpha and V has a 1-byte subroutine call (`RST`) to the serial port read/buffer routine about every 7 other instructions--the Orphanware and MI serial cards do not support interrupt-driven I/O, so the serial port must be polled at a fast rate in order to avoid losing characters at 19200 bps.

Ron comments:

```
>In all fairness, it could be noted here that Guy Cousineau once
>remarked (mused, lamented, complained) that TDOS was approaching the
>state where perhaps too many variations were 'out there' being
>used for one purpose and another. I think he realized that the
>'monster' was growing. In my mind however, that in no way
>diminishes the immense utility of TDOS in this place here. Each
>of my ADAM systems has a different setup. I have boot disks for
>each variation. It is not a problem.
```

Again, end-user utility is not necessarily a function of the elegance (or not) of the internal organization of the program. As for having boot disks for each hardware setup variation, this is fine as long as either (1) someone has assembled a version of TDOS for your particular setup, or (2) you have the TDOS source yourself and you "roll your own" version as needed. (2) is the typical situation for the general CP/M user these days; however, the TDOS source has *not* been released for general distribution. (I don't know if Dale Wick was "authorized" to let me have a copy, but that's where my copy came from.) In addition, the TDOS source is set up exclusively for the Z80ASM+ assembler from SLR, which, while an excellent and powerful assembler, (1) cost $150 when it was still available (I bought a copy at full price from Jay Sage at Sage Microsystems East), and (2) is no longer available now that SLR has sold out to Symantec. Porting the source to a different (freeware) assembler does not look to be an easy task.

The upshot is that the average TDOS user does not have the means to create non-standard versions of TDOS for unusual hardware configurations, or novel configurations (such as ADAMserve).

## IV. How Good Code Design Promotes Backward Compatibility.

Ron writes:

```
>Backward compatibility may be a problem for programmers, but
>programmers need to keep in mind that some users cling
>tenaciously to what they're familiar with unless there is a grand
>and handsome payoff. Perhaps the payoff has not been adequately
>sold.
```

Believe me, as a would-be operating systems programmer, I understand that users like to keep the applications they already have, and are using very productively. But it's extremely frustrating to me to find programs that, for no "good" reason (i.e., no payoff in speed or ease of use or novel use of system resources), do "bad" things like bypass the operating system, write directly to the hardware, etc.

But Ron also asks:

```
>Why would a program written in the early and mid-eighties do
>anything other than assume that it had the whole computer to
>itself? Was that not in fact true?
```

In the short-sighted short term, yes. However, even in the limited EOS operating system, there was some thought to what might come in future. The `_READ_BLOCK` and `_WRITE_BLOCK` functions take a 4-byte block number in the BC and DE registers; with 1K blocks, this is a maximum of 4 *gigabytes* of storage per block device. ADAMnet itself is a plug-and-play peripheral device network which modern Wintel systems have yet to achieve. And even though no communications protocol was ever defined, an ADAMnet device number (15) was reserved for a Gateway device which would have allowed ADAM-to-ADAM networking.

Some of these "hooks" for future expansion can be exploited today. Under ADAMserve, a 4GB ADAMnet hard drive could be created, or access provided to standard 650MB CD-ROMs--all because of support for "ridiculously high" block numbers in EOS-5. (Consider that tapes were 256K and original Coleco disk drives were 160K. Block 256 is B=1, C=0, D=0, E=0; a rather large "waste"!)

In the long term, however, the sloppy thinking that "the machine is *mine* and I can do what I want" is limiting. Part of the task of paving the way for future expansion is identifying all the "badly-behaved" programs and fixing them to be more considerate, when possible.

```
>When I
>hear this talk about so-called 'badly behaved programs' I have to
```

```
>wonder 'badly behaved' for whom?
```

They are badly-behaved for those who would try to provide additional features in operating systems while maintaining the user-demanded "backward compatibility".

```
>Speedywrite has often been cited as a fine example of a 'badly
>behaved program' in the ADAM world. The fact that it provided
>superior wordprocessing capability to anything then available for
>EOS seemed to be completely irrelevant. Speedywrite provides it's
>own methods of dealing with the computer's resources and by doing
>so provided an impressive example - in my estimation - of what a
>creative imagination can do with limited computing resources if
>left unfettered by conventions and rules.
```

Yes--from the end user's perspective. And as long as that end user is content to use SpeedyWrite in its original environment--taking over the entire machine, pull the reset switch when you're done--he has nothing to complain about. If he wants SpeedyWrite to run from a command line shell under an enhanced EOS which lives in XRAM, however, or can read files from a CD-ROM accessible by an ADAMserve connection, or can use an 80-column terminal, then he is (probably) out of luck, unless someone is willing to take SpeedyWrite to pieces and reassemble it in a "well-behaved" mode.

```
>In my humble
>estimation, a program of Speedywrite's quality, under any
>operating system, would be difficult replace.
```

And *that's* why, if there are too many "badly-behaved" legacy programs that end users want to keep using in their original form, then there is no point to trying to forge ahead with anything new and wonderful at the operating system level, because ultimately the result will be a nifty operating system which nobody will use but its author. I did that once with SmartBASIC 1.x; I don't wish to repeat the exercise.

Even Ron admits, after a laundry list of EOS's many shortcomings,

```
>it's probably fairly safe to say that
>there's plenty wrong with EOS about which plenty could be done.
>Whether or not it is worth doing except as an academic exercise
>remains debatable.
```

If the important applications that everybody wants to run had simply obeyed the injunction of the EOS programmers at the start of the EOS-6 code listing, it would have been soooo much easier to make enhancements to EOS:

```
          This absolute listing was generated to ease software
          development on the ADAM.  This listing provides the location
          of both released and unreleased entry points.  Released
          entry points begin immediately in this file with the jump
          table and end before the first code segment listed.
          Released entry points include the jump table, common data
          areas (EOS_COMN), common data tables, and equates which
          describe the released data structures.  Direct access
          to code segments is STRONGLY DISCOURAGED and may make
          your application incompatible with some ADAMs.  There is
          more than one version of EOS on the market at this time
          and updates are planned.
```

Much the same message is found at the beginning of the BIOS source code listing for the original IBM-PC (1983):

```
          THE BIOS ROUTINES ARE MEANT TO BE ACCESSED THROUGH
          SOFTWARE INTERRUPTS ONLY.  ANY ADDRESSES PRESENT IN
          THE LISTINGS ARE INCLUDED ONLY FOR COMPLETENESS,
          NOT FOR REFERENCE.  APPLICATIONS WHICH REFERENCE
          ABSOLUTE ADDRESSES WITHIN THE CODE SEGMENT
          VIOLATE THE STRUCTURE AND DESIGN OF BIOS.
```

And dealing with all the early MS-DOS applications which did *not* heed this injunction continues to cripple Wintel operating systems development.

## V. ADAM Hardware and Software Standards.

In closing, Ron has an excellent idea, which it's a pity that nobody had about 10 years ago:

```
>But perhaps we need something more. Perhaps we need not just a
>standard, or specification covering EOS and how it will work.
>Perhaps we also need a standard for programmers proposing to
>develop new software. (I'm assuming there will be some; perhaps
>that's a leap of faith). At the very least we need some sort of
>common understanding as to how ADAM's resources are to be
>accessed.
```

Those few of us who are left as programmers and stewards of the programs of others pretty much know what the standards should have been, and what the crippling no-nos are. Short of a massive effort to reverse-engineer the top five EOS applications to complete, reassemblable source code (PowerPaint, ADAMcalc, SpeedyWrite, SmartWriter, and ADAMlink V--I omit SmartBASIC and File Manager because I've already done them), there is no common base of "well- behaved" applications. *All* of the above 5 do bad things, or have bad internal organizations which make further (desired) expansion impossible. The ADAMserve version of File Manager (4.0, derived from a disassembly of 3.1) is now totally EOS-compliant; it does not bypass EOS to access hardware in any way at all. Its usage of XRAM buffers could be make RAMdisk-friendly without too much trouble, now that I have regenerated the source.

I can invent standards and promulgate/publish them through my *TWWMCA* articles; but unless part of my audience is ready to start writing some new applications (or willing to help reverse-engineer some existing ones), it is really just vanity press...

## VI. Conclusion.

On this note, Ron ends:

```
>'Nuff said. I do hope this discussion continues. I think it's
>marvellous that Dr. Drushel has initiated some thinking about
>these things. I hope we can all follow along, contribute to the
>best of our ability,  and learn in the process.
```

As I said at ADAMcon 08, I want to use my ADAM time and talents on a project which the remaining ADAM community deems to be of sufficient importance that it will actually *use it* if the project is completed, not simply ooh and ahh over it. Attaboys are *not* what I'm looking for. Frankly, I get a lot of intellectual pleasure in taking apart the sourceless binaries to see how they work, more so than trying to write user-friendly GUIs in Z80 asssembler `:-)`

See you next week!

*Rich*

# This Week With My Coleco ADAM 9710.06

by Richard F. Drushel

## I. What is a Bank-Switched EOS Operating System?

The last issue of *TWWMCA* (9709.30) mentioned in general terms some of the advantages to allowing the EOS operating system to move into expansion RAM (XRAM). I invited public comment on the idea; and in response, I received the following query from Bart "Zonker" Lynch:

```
>rich, could you put your "explanation hat" on and have a go at making this
>a bit more clear? i want to make sure i grasp the concept of what you are
>proposing before i make any comments. please remember to K.I.S.S! :)
>-zonker
```

I will try to clarify some of the technical issues, implications, and limitations of such an EOS redesign. First some basic facts:

## II. ADAM Memory Organization.

The Z80 microprocessor of the ADAM can access 64K of memory. This memory can be RAM or ROM. Additionally, the ADAM hardware is designed so that memory can be accessed in 32K segments or "banks", which correspond to the lower 32K (addresses 0000H-7FFFH) and the upper 32K (addresses 8000H-FFFFH) of the memory space. Different physical memories, ROM and RAM, are assigned to either the low 32K or upper 32K, and these banks may be swapped in and out, mixed and matched, to make different 64K memory maps. Only one RAM or ROM may be active (i.e., directly visible to the Z80 microprocessor) in each low/high 32K bank at any one time. Here is a summary diagram of the various banks of ROM and RAM and what place they occupy in the 64K address space:

```
low 32K                                high 32K

0000H         2000H     6000H          8000H                         FFFFH

<-------------intrinsic RAM------------><----------intrinsic RAM---------->
<-------------expansion RAM------------><----------expansion RAM---------->
                                        <----------cartridge ROM---------->
<--OS-7 ROM--><-----intrinsic RAM------>
<------------SmartWriter ROM----------->
                        <----EOS ROM--->
                                        <----------expansion ROM---------->
```

Memory configuration is selected by writing certain values to dedicated hardware I/O ports; the exact details need not concern us here. The key idea is that you have a memory smorgasbord, and can choose one from column A and one from column B for the current active memory map. Note that when the OS-7 ROM is selected, there is also 24K of RAM included to fill out the low 32K. (This is the ColecoVision memory map.) Also, a note about memory expanders bigger than

64K: they are arranged as parallel 64K banks, with a separate bank select port which must be written to. (This is because XRAM greater than 64K is aftermarket and there was no existing Coleco standard.)

There are *two* default memory maps, depending upon which reset switch you pull to start your ADAM! If you pull the game reset, the memory map is:

```
low 32K                                high 32K

0000H         2000H     6000H          8000H                            FFFFH

<--OS-7 ROM--><-----intrinsic RAM------><----------cartridge ROM---------->
```

If you pull the computer reset switch, however, you get the following:

```
low 32K                                high 32K

0000H         2000H     6000H          8000H                            FFFFH

<------------SmartWriter ROM-----------><----------intrinsic RAM---------->
```

In each case, program execution begins at address 0000H, in whatever RAM or ROM happens to be switched in.

For a game reset, the code looks to see if a game cartridge is plugged in. If the "magic" identifier byte sequences AAH, 55H ("game", uses COLECOVISION title screen) or 55H, AAH ("test", skips the title screen) are found at 8000H- 8001H, a cartridge is present and the game code is executed (using the pointer at 800AH to find the start of game code). If not, the `"TURN GAME OFF BEFORE INSERTING CARTRIDGE OR EXPANSION MODULE"` message is displayed and the system halts.

For a computer reset, however, the boot code is far more complex (and interesting `:-)` ). Here's the sequence of events:

1. Look for an expansion ROM (center slot; a "boot ROM" on one of the Micro Innovations I/O cards; *also* a "language card" French or German version of SmartWriter for European ADAMs). If found, jump to the program in it. There is a "magic" identifier byte sequence at 8000H-8001H of 66H, 99H which specifies a valid XROM; executable code begins at 8002H.
2. `low 32K                                high 32K`
3. 
4. `0000H         2000H     6000H          8000H` `FFFFH`
5. 
6. `<------------SmartWriter ROM-----------><----------expansion ROM---------->`
7. If there is no XROM present, swap in the high 32K RAM bank:
8. `low 32K                                high 32K`
9.

```
10.  0000H          2000H     6000H           8000H
   FFFFH
11.
12.  <------------SmartWriter ROM----------><----------intrinsic RAM------
   ---->
```
13. Then copy some code from the SmartWriter ROM to high 32K RAM and jump to it. This code then swaps in the EOS ROM and copies it to address E000H in RAM (where EOS usually resides).
```
14.  low 32K                                 high 32K
15.
16.  0000H          2000H     6000H           8000H
   FFFFH
17.
18.                           <----EOS ROM---><----------intrinsic RAM------
   ---->
```
19. The EOS loader code then issues the `_EOS_START` function call and tries to boot a program from disk or tape. Part of `_EOS_START` switches to an all-RAM memory map, which is the default environment for user programs:
```
20.  low 32K                                 high 32K
21.
22.  0000H          2000H     6000H           8000H
   FFFFH
23.
24.  <-------------intrinsic RAM-----------><----------intrinsic RAM------
   ---->
```
25. If no disk or tape is found, `_EOS_START` bails out to SmartWriter by issuing the `_GOTO_WP` function call, which swaps back in the SmartWriter ROM and jumps to the first byte of SmartWriter code at 0100H:
```
26.  low 32K                                 high 32K
27.
28.  0000H          2000H     6000H           8000H
   FFFFH
29.
30.  <------------SmartWriter ROM----------><----------intrinsic RAM------
   ---->
```

SmartWriter itself makes a number of bank switches (including to XRAM), but I will not describe them further at this time.

## III. The Perils of Bank-Switching.

Bank-switching gives the ADAM its great flexibility. Indeed, that's how the ADAM is able to implement a complete ColecoVision as a subset of its own functionalities--swap in the OS-7 ROM plus 24K RAM, and the cartridge ROM, and run the game.

However, some care must be used. For instance, you'd better not swap out the bank of memory that your code is currently executing in! That is, if the current instruction is executing in the upper 32K of memory, you *cannot* swap out the upper 32K of memory--if you do, your code becomes invisible to the Z80 (you swapped it out), and execution proceeds at whatever the next address is in the new bank of memory. Since this is likely to be garbage, there's no telling what will happen; in any case, you'll have to reboot the computer to get control again. (The only exception is the carefully designed case in which two banks of memory contain exactly the same code in exactly the same place; in this case, while there is a switch from one copy to the other, the program itself stays in phase, and there is no loss of control.)

You must also keep track of what bank of memory the program stack is in. The stack is a special area of read/write memory which is used for temporary storage of data, typically as CPU registers (AF, BC, DE, HL, IX, IY, PC). Any machine code which does `PUSH`es or `POP`s is using the stack. The current entry on the stack is pointed to by the SP (Stack Pointer) register. If you swap out the memory bank which contains the stack, you must not try to `POP` (because there will be garbage at the current SP) or `PUSH` (because you'll overwrite whatever is in the new bank of RAM, possibly program code). Both errors have disastrous results. Trying to `PUSH` or otherwise write to memory that is occupied by ROM is futile, because ROM is read-only as far as the Z80 is concerned. If your code needs to use a stack, you may need to allocate a temporary stack in the same bank of RAM you're in, for use only during the bank switch; of course, you have to save the current position of the *old* SP and restore it when you're done bank-switching.

Interrupts are also a problem during bank-switching. If enabled, maskable interrupts cause an automatic jump to address 0038H; non-maskable interrupts, to address 0066H. Both of these locations are in the lower 32K of memory. For a typical RAM-based program which does not do any bank- switching, the programmer provides interrupt handler code which resides at those addresses in RAM (typically a jump to somewhere else). If the program swaps out the lower 32K of RAM, however, and interrupts are still enabled, there is a chance that an interrupt will occur while the RAM is swapped out. If there is not appropriate code at 0038H and/or 0066H in the *new* bank of memory to deal with the interrupt, whatever garbage is present at those addresses will be executed as if it were code, and control will be lost. A trickier case is provided by SmartWriter, which has hard-coded jumps to locations in EOS RAM (upper 32K). If for some reason the SmartWriter ROM is swapped in but a copy of EOS code isn't present in the upper 32K, or some other memory bank (like XROM or cartridge ROM) is swapped in, then the interrupt will cause a jump to garbage code.

## IV. System Memory Maps.

Operating system code is like any other program code: it can be stored in RAM or ROM, and at any memory location which is convenient. The convention in both the EOS and CP/M operating systems is to put the operating system at the top of memory, leave the bottom 0000H-00FFH for interrupt vectors, and load user programs at address 0100H, extending upward in memory. Here's a diagram (turned on its side for convenience):

```
0000H              0100H                                                FFFFH

<---cold start----><-------user--------->  ...  <-------operating--------->
   RST,INT, NMI          program                         system
      vectors              code                           code
```

It is obvious from the above diagram that the maximum possible size of a (contiguous, non-bank-switched or non-overlayed) user program depends upon how big the operating system code is.

For example, if the operating system code is 8K, then the user program has less than 56K of space available to it. In the case of EOS-5 (the default ROM version in every ADAM), the operating system occupies over 11K (8K for EOS code, data, and ADAMnet device control blocks, 2K for user file I/O buffers, 1K for system file I/O buffer, and 105 bytes for file control blocks). This leaves less than 53K for user programs.

How does a user program know where the top of its available memory is? This is very important for a program which needs every bit of available space for user-created objects, such as ADAMcalc (spreadsheets), SmartBASIC (programs), or SmartWriter (text files). Some operating systems provide a dedicated function call to find out what the highest available free address is. This lets a user program tailor its memory needs dynamically, based upon how much free space is available. This also allows future revisions of the operating system to get bigger and grow downward in memory, because it can then be expected that every program will call this top-of-memory function and adjust its memory usage accordingly.

Alas, EOS-5 has no such top-of-memory function call. (EOS-8 does, for the reasons described above.) EOS-5 is loaded from ROM to RAM address E000H, and 3K of file I/O buffers and 105 bytes of file control blocks are allocated below that. This is the de facto upper limit for user program memory. Even the EOS-7 rewrite, which has a default of 3 user file buffers (instead of 2), squeezes in the third file buffer and file control block under the old EOS-5 top-of-memory limit. (This was accomplished by wholesale rewriting and optimization for size of the EOS code, freeing up over 1K compared to EOS-5.) Thus, the otherwise pathetic Disk Manager program launching shell (which loaded EOS-7 from disk) could successfully run any existing Coleco software which assumed that it could have all memory up to the EOS-5 top-of-memory point.

## V. Expanding EOS to Other Memory Spaces.

Given that all the Coleco software (and all of the SmartBASIC-based applications) is structured around this EOS-5 top-of-memory limit, and that reverse-engineering of binaries to source code for modification is a laborious task, it is thus an inescapable constraint that any future EOS revision which would run existing Coleco software *must* have an intrinsic RAM footprint no bigger than EOS-5. If EOS code and data overflow this limit, they will get clobbered by some application program which thinks it can use the space for itself.

Note carefully that I said "intrinsic RAM footprint", since that's where (with a few exceptions) all the Coleco software runs from--the base 64K of RAM. (Or in the case of SmartWriter, with the upper 32K of RAM, where EOS code is sitting.)

The EOS function calls are accessed via a jump table from FC30H-FD5EH in the upper 32K of intrinsic RAM. Thus, any program which makes EOS function calls must have the upper 32K of intrinsic RAM switched in at the time it makes the calls. When the function is completed, the memory map is the same as it was at the start of the call. What happens in between?

The answer is, *anything* can happen in between. Operating systems are supposed to be black boxes. There are defined entrances and defined exits, but what's inside the building itself is not defined, and indeed can be *anything* necessary to perform the requested functions. You can change the entire internal workings of the operating system, but if the defined entrances and exits are preserved, user programs shouldn't notice any difference. (This "black box" aspect of operating systems is what allows ADAMserve to substitute server disk drives and hard drive for genuine ADAM hardware--as long as `_READ_BLOCK` takes A=device number, BCDE=block number, HL=transfer address, and returns ZF=1 for okay, ZF=0 and A=error code for error, it doesn't matter where the block of data comes from, or what I have to do to get it.)

By extension, there is nothing that says that I have to stay with the current (entry) memory map while I'm performing an EOS function call. I should be able to switch to any memory configuration I want (subject to the Perils of Bank-Switching described above), as long as I put things back the way they were before I exit.

Thus, I should be able to put some of my operating system code into another memory space (e.g., XRAM), and with proper design and care, be able to transparently bank-switch in and out of that memory space, returning with the system memory map the same as it was at the start. I could fill an entire 64K of XRAM with EOS code, keeping only the jump table and the code necessary for bank-switching in intrinsic RAM. This way, user programs can keep their existing memory maps (i.e., they don't have to be reverse-engineered to source and patched to become dynamically-allocating), and EOS code can effectively be as big as it needs to be.

## VI. The Need for Controlled Access to XRAM.

If some of EOS code is made resident in XRAM, then XRAM becomes an important shared resource. User programs are no longer free to appropriate XRAM for themselves without regard to what might be stored there already, and in use by the operating system. If the user program overwrites the EOS code or data, then the system is corrupted and will crash if the garbage code is executed, or garbage data is used.

How can you tell if XRAM is present? As noted above, cartridge ROM and XROM have "magic" identifier bytes at reserved locations which signify that a valid ROM is present. At system reset, XRAM is wiped and contains data of undefined value, so there are no "magic" identifiers. However, the presence or absence of XRAM can be determined experimentally, by simply writing some data to the XRAM memory space, then trying to read it back. If XRAM is present, you will be able to read back what you wrote; if XRAM is absent, you'll get random data when you try to read it back. Since the data are random, it's best to write to more than one memory location (on the odd chance that random noise might give you the byte you wrote), and with a definite pattern of bits in the bytes. Typically, the "magic" bytes AAH and 55H are used, because AAH = 10101010 binary and 55H = 01010101 binary (an alternating pattern of set and clear bits).

XRAM greater than 64K can also be detected using this method. After selecting an XRAM memory configuration, each parallel bank of 64K can be selected (using the appropriate bank select port) and tested. Typically, to avoid having to exhaustively test every possible bank (a 2 MB expander has 32 parallel banks of 64K each), advantage is taken of the fact that memory expanders exist only in a few sizes (64K, 128K, 256K, 512K, 1 MB, and 2 MB), and only the "critical" bank for each size need be tested to determine its presence. Starting from the bottom, look at banks 0, 1, 3, 7, 15, and 31; the highest one detected determines the size of the XRAM.

Note that the XRAM detect method described (i.e, writing "magic" data and trying to read it back) is *destructive*--if XRAM is present, whatever data was already there at the test addresses gets wiped out. It is a simple matter to make detection non-destructive, however, by just reading the test addresses first, then doing the test write/read, and finally writing back the original contents if the test was successful. Unfortunately, none of the XRAM detection routines I have seen in the software that I have disassembled (SmartWriter, ADAMcalc, SmartBASIC 2.0, File Manager, PowerPaint, Walters RAMdisk) use non-destructive XRAM testing. Sad to say, I didn't either in the special HARDDISK 2.3/3.9 with RAMdisk that I created in 1991 for Herman Mason and George Koczwara; and I don't think I altered Tony Morehen's code for the XRAM detect in ADAMserve File Manager, either (though the necessary changes are trivial given that I have the source code). For shame!

One thing that I *did* do, however, in the HARDDISK 2.3/3.9 with RAMdisk was make sure that my XRAM-resident RAMdisk code sat in places other than where the XRAM detect routine of File Manager was making its (destructive) test writes. The need for this was experimentally-determined, after I had laid out my XRAM memory map as seemed most straightforward to me, implemented the RAMdisk driver, and found it got clobbered by File Manager. I don't remember if I had any problems with SmartWriter or ADAMcalc; and the hard drive version of PowerPaint had been patched by me anyway to use the hard drive instead of XRAM, so the latter wasn't a problem.

In any case, I hope you clearly see the difficulty in trying to create a shared resource when there are existing programs which use it with abandon. The choices are to alter the programs (usually very hard) or avoid using part of the shared resource that the programs want to use for their own (ultimately very limiting).

There is a final "easy" way out for me as programmer, but not for you as user: Ignore the first 64K of XRAM completely, and put EOS into the second 64K of XRAM, utilizing the leftovers (and any higher banks of XRAM) for a RAMdisk. This avoids the Coleco software problem because the Coleco software doesn't know about any XRAM above the first 64K. This is a disingenous choice, however, because it requires at least 128K of XRAM. 64K expanders are common enough that I can assume everyone left as an active ADAMite either has one or can get one; not so for the larger expanders. It is easier to try to get people to have their SmartWriter ROMs upgraded than to expect them to get 128K and greater memory expanders.

## VII. Synthesis.

Looking back to Zonker's injunction/plea to "K.I.S.S.", I'm not sure if he will perceive this article as "simple". I have tried to be more explicit in my reasoning, but there is some necessary complexity here, which I don't think has been treated as a coherent article within recent memory. (I'm sure there's some back issue of NIAD or the MTAG newsletter or Nibbles 'n' Bits somewhere which also covers this ground, maybe better or clearer than I have done here.)

Putting my "K.I.S.S." hat on very tightly, here's a go at the TV Guide blurb for the last couple of issues of *TWWMCA:*

"I want to put more functionality into EOS. I can't fit it all into 8K of EOS RAM, and I *must* to avoid breaking existing programs. Technically, I can move some of EOS into XRAM, but there are potential conflicts with interrupts and other programs which already use XRAM. I can either make EOS work around the conflicts, or eliminate them by patching the programs. Both are hard problems. Since everybody uses SmartWriter at some point (if only because of an abort), and SmartWriter uses XRAM, SmartWriter must be patched in ROM to *not* use XRAM, if an XRAM- based EOS is to be stable."

As always, comments and questions are welcome.

## VIII. Next Time.

Ron Mitchell's *Ron'sWeek'n'ADAM* for 5 October 1997 raises a number of interesting points about program design, style, and maintainability. I will devote next week's *TWWMCA* to a *Point-Counterpoint* discussion of his article.

See you next week!

*Rich*

# This Week With My Coleco ADAM 9709.30

by Richard F. Drushel

## I. Minutes of B.A.S.I.C. Meeting 9709.28.

The Cleveland B.A.S.I.C. users group held its monthly meeting in my basement on Sunday, 28 September 1997. It really wasn't too elaborate of a meeting, because only me and George Koczwara were in attendance. Pat Williams was out with the flu, and Jean Davies didn't want to come by herself. As for Herman Mason, well, nobody knows were he is `:-)`

George and I made the most of the situation, however. He had brought a disk containing the HTML source files for his personal webpage, which he had been working on. He wanted to make a wallpaper (background) consisting of the various logos for the 9 ADAMcons, and wanted to know if

1. I had photos of the banners,
2. if so, could I scan them, and
3. if I already had scanned photos, could we make a wallpaper?

At ADAMcon VIII, I had photographed all the banners which were on display at the banquet. I subsequently scanned the photos, used Adobe Photoshop on one of our lab PowerMacs to resize them and make them into JPEG images of suitable size for download (about 15-20K each), and sent them to Dale Wick, where he put them up on his Coleco ADAM Home Page. The URL is:

http://www.csclub.uwaterloo.ca/u/dmwick/adam.html

I still had copies of the final JPEGs on my home 486 system, so I gave these to George.

As for the ADAMcon 09 banner, I had photographed this as well at the ADAMcon 09 banquet, but had not yet scanned/processed the photographs. This is something I should do soon, some evening when I'm in the lab late and have accesss to the scanner. I'll get the JPEG of this (and of the ADAMcon 09 cake) to Dale as soon as I make it.

I then set out to make a webpage wallpaper out of the 8 banner JPEGs. Any image (GIF or JPEG) can be used as a basis for wallpaper; the web browser simply tiles it all over the screen (i.e., repeats it over and over like covering a floor with tiles) until the screen is covered. Text and other images are then displayed on top of it. To create the wallpaper image, I used a Win3.1 graphics program called Canvas 3.5. In brief, I loaded each of the 8 banner JPEGs and pasted them into a single window as separate objects (which means that I could move/manipulate each one separately). This window would be the basis for the final wallpaper image. I then moved each banner image into position, aligning them as best I could (the banners are of different sizes) so that a 2 column by 4 row array was created. I then created a solid blue rectangle (in ADAM cyan) and placed it *behind* the 8 banner images. Finally, I grouped all 9 objects together into a single bitmap image (i.e., there were no more separate objects, just one image) and saved it as a GIF file.

I wrote a quick and dirty HTML source file which I could load from disk in Netscape (3.0 for Win3.1) to see how the wallpaper looked. It looked quite nice, and was a small file for as large a bitmap as it was (about 60K). Unfortunately, both George and I agreed that the wallpaper was so visually "busy" that any text displayed on top of it would be rather illegible. Thus, George decided that he would find something else ADAM-oriented to serve as wallpaper for his webpage. What could be done, however, is to use Adobe Photoshop and its image manipulation features to tone down or fade the image, so that it would look more like background than foreground. If I get a chance, I may play with this. In any case, if there is interest, I can put the GIF wallpaper that we did make up for download. You could also go to the Coleco ADAM Home Page, grab the banner JPEGs yourself, and have your own go at it.

## II. Some Comments on ADAMserve.

I had couple of responses to my USENET repost of *TWWMCA* 9709.26, the one which documents the ADAMserve serially-linked device protocol. The gist of the comments was, "You're reinventing the wheel: the TCP/IP protocol does everything you're trying to do here and more. Why not use TCP/IP?"

TCP/IP is the main protocol used over the Internet to transmit data. Indeed, other protocols (like Real Audio for sending stereo sound) are themselves transmitted using TCP/IP. I would love to use something as robust and full-featured as TCP/IP with the ADAM.

The only problem with using TCP/IP for ADAMserve-like functions is that it takes too much Z80 machine code to implement TCP/IP. The entire ADAM EOS operating system must fit into 8K of RAM (less, if you leave room for the ADAMnet device control blocks at the very top of memory). Even after squeezing out all the dead space and optimizing the existing EOS code, there was only about 500 bytes available for all of ADAMserve. About 100 bytes of that is taken up by the low-level serial I/O routines, and another 64 bytes by the logical-to-physical device table. Managing NMIs (non-maskable interrupts) is another 20-odd bytes. There just isn't enough room to do much with what's left. I couldn't even put in a CRC (cyclic redundancy check) calculator for block I/O; I had to use simple (and not as capable of detecting transmission errors) checksums because they require minimal machine code to implement. Similarly, automatic retries in case of error are not implemented at the EOS level; the user program has to do the retries itself if desired.

Thus, ADAMserve in its current form represents a number of compromises from the ideal. About the best that I can say is that it does work: it allows the ADAM to use server hardware transparently (through the established ADAMnet device interface and standard EOS function calls), and it fits within 8K.

## III. Expanding ADAMserve.

One feature that I'd like to add (before I freeze the protocol) is a Set ADAMserve Version function:

```
*****************************
**                         **
**  SET ADAMSERVE VERSION  **
**                         **
*****************************


ADAM                                              SERVER
====                                              ======

V [version] [1's complement of version] ---------->


<-------------------------------------------------- ACK (or NAK
                                                            VER_NOT_AVAIL_ERR)


VER_NOT_AVAIL_ERR   EQU       8Fh       ;server doesn't support the requested
                                        ;level of ADAMserve functionality
```

This would allow server software to evolve to support greater levels of functionality (perhaps even on non-ADAM computers; I have in my moments of delusion of godhood imagined that ADAMserve could be a general-purpose way for 8-bit computers to access other hardware resources), while remaining backward-compatible with earlier versions of the client software. In this case, the client would tell the server what version of ADAMserve it was expecting. If the server supports the requested version, it will acknowledge the request, and all further ADAMserve transactions will follow this level of functionality. If the server does not support the requested version of ADAMserve, it returns an error code, so that the client can either attempt to "downshift" and negotiate a compatible level of ADAMserve functionality, or else abort gracefully. By default, the server should start up in the original (i.e., current) ADAMserve mode, and not change unless specifically requested by the client.

I will hereby define the default version to be the current ADAMserve 4.0 specification. Version 0 will be ADAMserve 5.0, when that specification is finally written and frozen. Note that any clients with 5.0+ functionality attempting to run on a 4.0 server will get an `INV_COMMAND_ERR` in response to the Set ADAMserve Version command, and hence should either downgrade to 4.0 level or abort.

## IV. Another Go at EOS-8?

The relative primitiveness of the ADAMserve protocol, compared to other available protocols which have already been invented and thoroughly debugged, is somewhat disappointing to me. Even though it works, ADAMserve is a dead end because of the stupid 8K EOS RAM limit; ADAMserve 4.0 will let existing applications run, but is not conducive to the development of new applications (assuming that there are programmers out there who will write them).

The only hope for more EOS functionality is to get rid of the 8K EOS RAM limit. The easiest way that I can forsee to do this is to move parts of EOS into expansion RAM. This would allow me to keep the 8K EOS RAM limit in the base 64K of RAM, but add as much functionality as I want in expansion memory (XRAM). The only programs which wouldn't work would be those that appropriate XRAM for themselves. These would have to be identified and modified to leave XRAM alone, or else access it only in a controlled way.

Here's what I propose: a bank-switched EOS. It would require at least a 64K memory expander, and whatever space wasn't used by EOS would be made available as an EOS RAMdisk (device 26, `d7` in SmartBASIC).

The only problem with this is the program that's built into every ADAM: SmartWriter. SmartWriter checks for the presence of XRAM, and takes it over as additional program workspace if it finds it. So as soon as anybody aborted to SmartWriter through the `_GOTO_WP` EOS function call, their ADAM would crash because EOS would be trashed. Thus, prior to any XRAM-banked EOS project, SmartWriter has to be modified, and new, XRAM-friendly SmartWriter ROMs installed in any ADAM wanting to use the new EOS.

This weekend, I made a preliminary disassembly-to-source of SmartWriter R80, using Z80DIS22 under the Z80MU.EXE CP/M emulator. (This is the same disassembler that I used to make ADAMserve File Manager and to fix the ColecoVision game Pitfall to have infinite lives.) There is some nasty code organization in SmartWriter, but nothing I haven't been able to handle so far. (The biggest impediment was what appears to have been an inline macro command for printing strings: a `CALL MY_PRINT_STRING` followed by the address of the string. This confused the disassembler and required tedious but simple manual intervention to separate code from data areas.)

I believe that I can very shortly create a version of SmartWriter which ignores XRAM. (I also will try to put in the ability to read/write to drive D (device 5), a long-standing omission in SmartWriter). I will test the new binary not by burning new SmartWriter ROMS, but rather by using Marcel de Kogel's ADAM Emulator. When I'm confident that I have been successful, I'll burn a ROM and try it in a real ADAM. When this works, I will release the binary and strongly encourage every ADAMite to upgrade to this new ROM. Only extremely large SmartWriter files which were created by someone with a memory expander would be unreadable under this new, XRAM-ignoring SmartWriter.

I have heard that there are releases of SmartWriter after R80 (I seem to remember an R84 and an R85). If anybody has the binaries of these, I would rather use them as the basis for this rewrite, since presumably they include the last available bugfixes (maybe even the drive D bug already). In the meantime, I will forge ahead with the R80 regeneration, since anything I may learn there will be needed for any of the later versions.

The process of burning new ROMs can be coordinated rather easily if we just agree to do it. The lowest-cost thing to do is to pull the existing EPROMs from your ADAM, send them to someone for reprogramming, the reprogrammer sends them back to you, you put them back in your ADAM, and you're done. Anybody with a Phillips-head screwdriver can open up an ADAM, and anybody with a small bladed screwdriver can pull an EPROM. This would be a good group activity for users groups, or something that could be done (by me or others) on a grand scale at ADAMcon 10. Of course, new PROMs or EPROMs could be made at material cost for those not wishing to have their original EPROMs wiped.

I have an EPROM burner and would be willing to reprogram them at shipping cost (about $1 or so) as a non-profit proposition. I would rather not deal with complete ADAM system units sent by people unwilling to open them up to get at the ROMS, because of the large expenses all the way around; but vendors might be willing to do this.

Just knowing that there was nothing intrinsic to the ADAM that would crash an XRAM-based EOS would make it a more reasonable proposition to implement advanced EOS-8 and ADAMserve functionalities.

Except for my disassembly of SmartWriter, this is all speculative: please discuss this publicly on the Coleco ADAM Mailing List, send me E-mail, etc. I want to know what people think.

## V. Programs Known to Use XRAM.

Here are the EOS programs I can think of that will appropriate a memory expander if they can find one:

- SmartWriter (extra workspace)
- ADAMcalc (as a background print buffer)
- PowerPaint (original version *requires* a memory expander)
- SmartBASIC 2.0 (`EXTMEM` command allows extra workspace)
- File Manager (won't use XRAM unless told to, but detection is destructive)
- SeQuel (MIDI I/O buffer)

I don't believe any of the Coleco database-type programs (SmartFiler, Recipe Filer, Address Book) use XRAM, nor indeed any of the other Coleco software. If you know of anything else which does use XRAM, please let me know.

Of the above list, the ones which will be well-nigh impossible to make compatible with an XRAM-based EOS are:

- SmartBASIC 2.0 in `EXTMEM` mode (`STDMEM` mode will be okay)
- SeQuel (overhead of going through an EOS RAMdisk will probably be too much timewise; I think it accesses XRAM through a maskable INT routine during MIDI interrupt--can you comment, Chris Braymen?)

The existing hard drive version of ADAMserve PowerPaint was already patched to ignore expansion memory. The print spooler in ADAMcalc can simply be disabled, forcing printing to the foreground (sorry, Frances Clee!). The destructive testing part of File Manager can be programmed around (indeed, I solved this problem in 1992 in the versions of Mini Wini HARDDISK 2.3 and 3.9 with RAMdisk).

See you next week!

*Rich*

# This Week With My Coleco ADAM 9709.26

by Richard F. Drushel

## I. Administrivia.

My apologies that this issue of *TWWMCA* is some days late; work-related tasks have taken more time than expected this week. However, as promised, here is the first published version of the ADAMserve protocol. Comments are welcome.

## II. The ADAMserve Protocol.

I list the version number at 4.0 because I have some comment references to a version 3.0 of the standards document. Unfortunately, I cannot find a copy of the 3.0 document, let alone 1.0 or 2.0. I may have uploaded 3.0 to Michael Hurst's BBS a few years ago; Michael, could you check for me?

So, I had to rewrite the standard from scratch, combining memory, intentions, and reference to the source code for the ADAM client and PC server software. It is usually bad news to let the implementation define the standard; but the implementations *were* written with that 3.0 standards document in front of me, so I'm pretty confident that it is a complete representation of the standard as it exists.

---

```
     **********************************************
     **                                          **
     **  ADAMserve Serially-Linked Device Protocol  **
     **                                          **
     **         for the Coleco ADAM computer       **
     **                                          **
     **   version 4.0  Friday, 26 September 1997   **
     **                                          **
     **    (c) 1994-1997 by Richard F. Drushel     **
     **                                          **
     **               drushel@apk.net              **
     **                                          **
     **********************************************
```

### Disclaimer.

This is a working standards document. The ADAMserve protocol is subject to change as dictated by expediency and ease of implementation. At some point in the near future, however, the standard will be frozen, which will make it "safe" for others to write servers on non-PC hardware, based only upon this standards document.

## Malediction.

The internal workings of ADAMserve concern only client and server software, not user applications. Anyone who writes a user application which depends upon the exact internals of the handshaking, etc., or on anything other than the public symbols which define the Standard Devices and Error Codes, is writing non-portable code which is liable to break in later versions of ADAMserve. Don't complain to me if this happens to you--you've been warned!

## Overview.

ADAMserve is a protocol to allow an ADAM client to utilize the hardware resources of a second server computer, the two computers being connected by their serial ports. The client maps ADAMnet devices to server devices through a modified version of the EOS operating system. ADAMserve EOS implements ADAMnet emulation and logical-to-physical remapping so that user applications doing device I/O through EOS function calls can have transparent access to server devices. The server software waits for ADAMserve requests and processes them as received. ADAMserve is a handshaking protocol with modest error detection and recovery capabilities. These capabilities are in practice limited by the stringent need to keep the size of the client EOS operating system code less than 8192 bytes (8K).

## Minimum Hardware and Software Requirements.

- ADAM: Base ADAM (1 tape drive), with one serial port (Orphanware or MI). While the MI serial port can be configured to 38.4 kbps, the Orphanware serial port maximum bitrate is 19.2 kbps. Thus, the client-server link serial port is currently set at the greatest common bitrate, 19.2 kbps, though this is subject to change in future.
- SERVER: Currently the server software exists only for x86-based IBM-PC compatibles (though any machine with a serial port which implements the ADAMserve protocol can in principle work successfully). Base PC: PC-XT (8088, 4 MHz), 256K RAM, one serial port, one 360K disk drive, one 20 MB hard drive, CGA monitor, MS-DOS 2.0 or greater, ANSI.SYS screen driver.
- CABLE: Standard RS-232C serial cable in null-modem configuration (either hardwired or with adapter).

## Standard ADAMserve Devices.

ADAMserve defines several standard (logical) devices. New devices may be added to the *end* of this list. Both client and server are free to map these logical names to any desired physical hardware. These are public global symbols.

```
FD0        EQU  0    ;floppy disk drive 0   A:                block
FD1        EQU  1    ;floppy disk drive 1   B:                block
HD0        EQU  2    ;hard drive 0          EOSHD0.DSK        block
HD1        EQU  3    ;hard drive 1          EOSHD1.DSK        block
SP0        EQU  4    ;serial port 0         COMx:             character
SP1        EQU  5    ;serial port 1         COMx:             character
PP0        EQU  6    ;parallel port 0       LPTx:             character
```

```
PP1         EQU  7    ;parallel port 1       LPTx:            character
CLOCK       EQU  8    ;real-time clock       BIOS clock       block
STATUS      EQU  9    ;status device         device summary   block
VIDEO       EQU  10   ;video graphics        video display    block
VT100       EQU  11   ;emulated VT100        video display    character
KYBD        EQU  12   ;alternate keyboard    server keyboard  character
```

The current version of the server software implements FD0, FD1, HD0, PP0, and PP1. PC disk drives can read/write ADAM floppy disks, so original ADAM disks may be used directly in A: (FD0) and B: (FD1). A 10 MB EOS hard drive partitioned into 10 logical drives is implemented as a virtual disk file, EOSHD0.DSK. PP0 emulates the bidirectional ADAM printer, while PP1 is just a standard parallel printer port. It is intended that the block structure of the CLOCK device be compatible with Chris Braymen's ADAMnet clock. The STATUS device is intended to return the current status of all server hardware in a single status block. The VIDEO device was reserved for the development of some high-level graphics display protocol (such as used by Prodigy) to allow client programs to display hi-res color graphics on the server monitor. The VT100 and KYBD devices are reserved for use by client telecommunications programs which need an 80x24 VT100 terminal for video display, and a keyboard with more function keys than are provided with the ADAM keyboard. It is intended that SP0 and SP1 will be fully interrupt-driven, ring-buffered serial ports which can be used with modems, terminals, and printers. Note that these are *not* the same as the server serial port used for the actual client-server link.

## Standard ADAMserve Status and Error Codes.

ADAMserve defines several status and error codes which are used in handshaking. The error codes begin at 81 hex (for compatibility with ADAMnet DCB status codes). New error messages may be added to the *end* of this list. These are public global symbols.

```
ACK             EQU     5       ;positive acknowledgement token, ASCII 05H
NAK             EQU     21      ;negative acknowledgement token, ASCII 15H

;these are ADAMnet error codes

CHECKSUM_ERR    EQU     81H     ;bad checksum/CRC
INV_BLOCK_ERR   EQU     82H     ;bad block number
MISS_MEDIA_ERR  EQU     83H     ;missing media in disk drive
INV_DEV_ERR     EQU     84H     ;invalid/unimplemented device
WRITE_PROT_ERR  EQU     85H     ;disk is write-protected
DEVICE_ERR      EQU     86H     ;general device fault

;these are additional error codes

INV_COMMAND_ERR EQU     87H     ;invalid ADAMserve command
INV_BAUD_ERR    EQU     88H     ;invalid baudrate set attempt for SP0/SP1
INV_PDS_ERR     EQU     89H     ;invalid parity/data/stop bit set attempt
                                ;for SP0/SP1
INV_CLOCK_ERR   EQU     8AH     ;invalid date/time set attempt
PRN_OFFLINE_ERR EQU     8BH     ;printer is offline
PRN_NOPAPER_ERR EQU     8CH     ;printer is out of paper
PRN_NOPOWER_ERR EQU     8DH     ;printer is turned off
NO_CHAR_RDY_ERR EQU     8EH     ;server timed out expecting a character from
                                ;the client
```

## ADAMserve Protocol.

All ADAMserve transactions are initiated by the client, through the use of a two-byte Request Header which is transmitted over the client-server serial link:

```
ADAM                                              SERVER
====                                              ======

[command] [device #] ------------------------------>
```

If the command is invalid (such as might occur if the client and server are out of synchronization due to a transmission error), the server responds by sending the `INV_COMMAND_ERR` code and then flushing its own link receive buffer (hopefully to clear it of garbage so that client and server can resynchronize).

The following commands are defined. These are public global symbols.

```
READ_CODE        EQU     "R"     ;read
WRITE_CODE       EQU     "W"     ;write
FORMAT_CODE      EQU     "F"     ;format
SETSERIAL_CODE   EQU     "S"     ;set SP0/SP1 serial port parameters
```

The device number is one of those defined in the list of Standard ADAMserve Devices. If the device number is invalid, the server responds by sending the `INV_DEVICE_ERR` code. Otherwise, based upon whether the device is a block device (such as a disk drive or hard drive) or a character device (such as a printer or serial port), the handshaking continues as illustrated below:

```
*****************************
**                         **
**  READ CHARACTER DEVICE  **
**                         **
*****************************


ADAM                                              SERVER
====                                              ======

R [char dev #]  ---------------------------------->


<------------------------------------------------- ACK (or INV_DEVICE_ERR
                                                           DEVICE_ERR)


ACK ---------------------------------------------->


<------------------------------------------------- char

<------------------------------------------------- 1's complement (char)
```

```
ACK ------------------------------------------------>
(or NAK if CPL(sent char)
not equal to sent 1's complement (char))



******************************
**                          **
**  WRITE CHARACTER DEVICE  **
**                          **
******************************


ADAM                                            SERVER
====                                            ======

W [char dev #]  ------------------------------------>


<--------------------------------------------------- ACK (or INV_DEVICE_ERR)


char ----------------------------------------------->

1's complement (char) ----------------------------->


<--------------------------------------------------- ACK (or CHECKSUM_ERR
                                                             PRN_NOPAPER_ERR
                                                             PRN_OFFLINE_ERR
                                                             PRN_NOPOWER_ERR
                                                             DEVICE_ERR)



*************************
**                     **
**  READ BLOCK DEVICE  **
**                     **
*************************


ADAM                                            SERVER
====                                            ======

R [block dev #]  ----------------------------------->


<--------------------------------------------------- ACK (or INV_DEVICE_ERR
                                                             DEVICE_ERR)


[lo] [hi] [lo] [hi] block # ----------------------->
 loword     hiword


<--------------------------------------------------- ACK (or INV_BLOCK_ERR)
```

```
ACK ---------------------------------------------->


                                                    [now server reads
                                                     physical media]


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;
;;
;;  NOTE:  the following handshake was in the original specifications, but
;;
;;  is not currently implemented due to space limitations in EOS RAM.
;;
;;  Without it, it is not possible to specifically trap hard read errors
;;
;;  on the server side.  The current behavior in case of hard read errors
;;
;;  is to abort on the server side and wait for the ADAM side to timeout.
;;
;;
;;
;;
;;
;;  <---------------------------------------------- ACK (or CHECKSUM_ERR
;;
;;                                                          MISS_MEDIA_ERR
;;
;;                                                          DEVICE_ERR)
;;
;;
;;
;;
;;
;;  ACK ------------------------------------------>
;;
;;
;;
;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;


<-------------------------------------------------- [block of data]

<-------------------------------------------------- [lo] [hi] checksum


ACK----------------------------------------------->
(or NAK if checksum error)
```

```
**************************
**                      **
**  WRITE BLOCK DEVICE  **
**                      **
**************************


ADAM                                              SERVER
====                                              ======

W [block dev #]  ----------------------------------->


<--------------------------------------------------- ACK (or INV_DEVICE_ERR
                                                             DEVICE_ERR)


[lo] [hi] [lo] [hi] block # ----------------------->
 loword     hiword


<--------------------------------------------------- ACK (or INV_BLOCK_ERR)


[block of data] ------------------------------------>

[lo] [hi] checksum -------------------------------->


<--------------------------------------------------- ACK (or CHECKSUM_ERR
                                                             WRITE_PROT_ERR
                                                             DEVICE_ERR
                                                             NO_CHAR_RDY_ERR)



***************************
**                       **
**  FORMAT BLOCK DEVICE  **
**                       **
***************************


ADAM                                              SERVER
====                                              ======

F [block dev #]  ----------------------------------->


<--------------------------------------------------- ACK (or INV_DEVICE_ERR
                                                             DEVICE_ERR)


                                                     [now server does the
                                                      physical format]
```

```
<-------------------------------------------------- ACK (or WRITE_PROT_ERR
                                                            DEVICE_ERR)



*********************************
**                             **
**  SET SERIAL PORT PARAMETERS  **
**                             **
*********************************


ADAM                                          SERVER
====                                          ======

S [char dev #]  ----------------------------------->


<-------------------------------------------------- ACK (or INV_DEVICE_ERR
                                                            DEVICE_ERR)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;
;;
;;  NOTE:  the remainder of the Set Serial Port Parameters handshaking is
;;
;;  currently undefined, because SP0 and SP1 have not yet been implemented.
;;
;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

## Implementation Notes.

1. Currently, due to space limitations in EOS RAM on the client side, there are no automatic retries implemented in case of errors. As long as this fact is made known, user programs can easily implement error retries if desired. Clearly, however, the more ideal situation would be for EOS to handle retries.
2. NMIs (non-maskable interrupts) are disabled during transfers for systems using Orphanware serial ports. Serial port overruns result if the user program has an active NMI routine and it remains active during client-server data transfers. NMIs are *not* disabled for systems using MI serial ports, because the 2681 UART has a 4-byte on-chip buffer which can hold several characters which may arrive during an active NMI routine. The client software is "smart" in that it will leave NMIs off if they were off already, and will restart them safely if they were on and turned off.
3. The ADAMnet floppy disk drive firmware responds to a `_WRITE_BLOCK` command to block 0FACEH by invoking a disk formatting routine. Current client EOS does *not* trap writes to block 0FACEH by issuing an ADAMserve format command; nor does the

current server software trap writes to this block and invoke disk formatting of its own accord. Probably the latter should be implemented.

4. The current server software has several server-keyboard-invokable "fudges" to permit certain ill-behaved applications to function correctly in an ADAMserve environment. One is for PowerPaint, which sends garbage as the hiword of block number when reading/writing blocks in the file dialogue box; the fudge tells the server to force the hiword of block number to zero. Another fudge is for ADAMcalc, to translate "magic" control characters sent by the ADAMnet serial printer driver into appropriate "standard" carriage returns and linefeeds.
5. The ADAMserve HARDDISK shell loads a special version of ADAMserve EOS if SmartLOGO is executed. SmartLOGO writes to VDP control registers directly instead of using EOS function calls (which save a copy of what was written to these write-only registers), so when enabling/disabling NMIs during ADAMserve transactions (which also requires a VDP register write), there's no way to tell what the current state is, so that it can be put back how it was when NMIs are restarted. The SmartLOGO-only version of ADAMserve EOS resorts to the evil expedient of poking 0EDH 45H (the machine code for `RETN`, RETurn from Nonmaskable interrupt) at addresses 102-103, waiting for 1/60th of a second to be sure that the NMI occurs and returns (but without a read of VDP register 8 to restart the NMIs), doing the serial data transfer, then restoring the original contents of 102-103 and reading VDP register 8.
6. Note the large handshaking overhead in character I/O transfers. For Read Character Device, 7 characters must be transferred to successfully read 1 character. For Write Character Device, 6 characters are transferred for each character written. At 19.2 kbps, 8 data bits, no parity, 1 stop bit (9 bits per byte) the maximum data transfer rate is 19200/9 = 2133 bytes per second. The effective rate for ADAMserve character reads is 2133/7 = 305 characters per second (about 2.7 kbps), and for writes is 2133/6 = 363 characters per second (about 3.3 kbps). The implication for ADAMserve-based telecommunications programs is that, even if the physical SP0 or SP1 is a 28.8 kbps modem, the effective rate seen on the ADAM screen will be only slightly better than 2400 bps. Of course, using an MI client serial port at 38.4 kbps would double all these rates. The users of Orphanware serial ports, however, are somewhat out of luck.
7. The handshaking overhead in block I/O transfers is not noticable for disks with 1024-byte blocks. The effective transfer rate, however, is still not much faster than a genuine ADAMnet digital data drive (i.e., tape). The calculation is left as an exercise for the reader `:-)`
8. The reason that Set Serial Port Parameters is currently undefined is that I'm still trying to decide whether or not I want to support the bizarre way that the prototype ADAMnet serial device sets these parameters (in brief, by sort of switching from a character device to a block device to send a parameter block, then switching back to single-character mode). Chris Braymen explained it to me once, but it's all gone hazy now.
9. The checksum computed in block I/O transfers is a simple sum. A CRC (cyclic redundancy check) would be far more robust, but there is no space available in EOS RAM to implement a CRC calculator. For local implementations (i.e., client and server physically close and connected by a short serial cable) this should not be much of a problem; but an arrangement in which client and server are connected by dialin telephone

lines would be much more subject to errors due to line noise, and CRC is safer than simple checksum in this case.

## III. For Next Time.

I should have another *TWWMCA* article written for 29 September 1997, because B.A.S.I.C. is having its monthly meeting this weekend, and I ought to be able to write about the meeting. It will be a short article, but I'll be back on schedule.

See you next week!

*Rich*

# This Week With My Coleco ADAM 9709.14

by Richard F. Drushel

## I. Rescuing the Coleco ADAM Forum on the Cleveland Freenet.

A couple weeks ago, a general notice was posted by the administrators of the Cleveland Freenet, to the effect that they were going to clean up (i.e., remove) any areas and SIGs which they deemed to be dead or abandoned. Unfortunately, the Coleco ADAM Forum was one of the SIGs that was slated for removal. I must admit, other than reposting my *TWWMCA* articles to the general bulletin board, I hadn't done much with it; and various technical glitches have prevented us from getting B.A.S.I.C. members Pat Williams and Jean Davies on-line and participating. Fortunately, as I am one of the sysops, I was able to intervene and save the Coleco ADAM Forum from destruction. I updated some of the informational text files, redid the menu structure a little, and have at least planned how I want to revamp the newsgroup and bulletin board areas which we have. If any of you have accounts on the Cleveland Freenet, all you have to do is type `"go adam"` at any `"Your Choice ==>"` prompt. If you don't have an account on CFN, but have telnet access and want to check us out, then do the following:

```
     telnet://freenet-in-a.cwru.edu
```

You'll get a nice ASCII art picture of the Cleveland skyline, and then the following prompt:

```
     Are you:
             1. A registered user
             2. A visitor

     Please enter 1 or 2:
```

You can enter `"2"`. You will then get another prompt:

```
     Would you like to:
             1. Apply for an account
             2. Explore the system
             3. Exit the system

     Please enter 1, 2 or 3:
```

You can enter `"2"` again, and you'll be able to look around the system. Once you get to a `"Your Choice ==>"` prompt, you should type `"term"` and set your terminal type (probably `"vt100"` would be best). Then you can `"go adam"` and look at the information files and newsgroups. Unfortunately, you won't be able to post anything if you are just a visitor. You can, however, apply for an account, as indicated on the menu above; accounts on CFN are free.

I welcome any suggestions about improving the look of the Coleco ADAM Forum. Let me know what you think! You can reach me and the other sysops (Herman Mason and George Koczwara) by sending mail to xx001@po.cwru.edu.

## II. Interrupts and Shared System Resources.

In last week's *TWWMCA* (9709.07), I promised to talk about the problem of the non-reentrancy of EOS VRAM routines, and a solution that I implemented as part of the EOS-8 project.

What do I mean by "non-reentrancy"? Basically it means that VDP (video display processor) is hardware which can only be accessed by one user (or subroutine or process) at a time, and while that user is accessing it, nobody else dare interrupt him, or the VDP will get messed up or confused. Think how hard it is to concentrate if you're talking to someone on the telephone and then your kids pick up the extension phones and all start talking at once. Unless you have remarkable powers of concentration, the phone is a non-reentrant resource--only one person can talk intelligibly at a time.

The Z80 microprocessor allows external conditions (such as a keyboard or button being pressed, a character arriving at a modem) to generated a signal called an interrupt. When an interrupt is received, the Z80 finishes the last machine code instruction it was executing, and then jumps to some special locations in memory and starts executing whatever program is there. (The programmer has to plan for this and provide the appropriate program at the appropriate location.) This program is called an interrupt service routine (ISR), because it is supposed to deal in some special way with whatever condition caused the interrupt. For example, in the case of a keyboard interrupt, the ISR might read the key that was pressed and save an ASCII code for it in a buffer somewhere. The ISR ends with a special form of the `RET` (return) instruction, which tells the Z80 to go back to the program that it had been working with and keep going, right where it left off. Interrupts are an efficient alternative to polling, which means sitting in a loop asking "Are you ready yet? Are you ready yet?" over and over until the answer is "Yes". With interrupts, when it's ready, it will come grab *you*, so you can do other things until you get grabbed.

How might the VDP get interrupted? It turns out that the VDP itself generates an interrupt 60 times per second (50 for European ADAMs running on 50 Hz line current instead of 60 Hz). What if the main program was in the middle of a write to the VDP (say to display a character on the video screen) when this VDP interrupt occurred, and the VDP ISR also tried to write to the screen? The result would be garbage on the screen and a confused VDP, because it hadn't finished the first write before starting the second.

The issue is not academic for the VDP, because of a quirk in the way the VDP responds to interrupts. Any VDP ISR must have, as its last action before returning from the interrupt, a read of VDP register 8 in order to acknowledge the interrupt and enable it for another cycle 1/60th (1/50th) of a second later. If *any* VDP read or write is already in progress, this acknowledgment read will disrupt it and leave the VDP in an unstable state.

There are two possible solutions to the problem: (1) keep the VDP interrupt disabled, so the main program can never get interrupted; or (2) somehow keep track of when the VDP is being used by a user program, so that an interrupt routine won't try to access the VDP if it's busy, but will instead wait until another time when the VDP is free).

SmartBASIC 1.0 uses both methods in different places.

1. In `TEXT` mode, the main program can `PRINT` to the screen, while the VDP interrupt routine manages blinking on and off if the `FLASH` command is active. Blinking can't safely occur, however, if a `PRINT` is in progress. SmartBASIC uses a status flag to indicate that `PRINT` is active, and the `FLASH` interrupt routine leaves the VDP alone; however, the `FLASH` interrupt routine also maintains a status flag for `PRINT` to tell it that an interrupt has occurred and that `PRINT` needs to do the acknowledgment read of VDP register 8, to restart the VDP interrupt for the next cycle. The result of this handshaking is that access to the VDP is regulated to one routine at a time. The `FLASH` rate during a `PRINT` is visibly slower, however, than if you are just sitting at the `]` prompt. (Try this: `TEXT`, `FLASH`, then `CATALOG` some disk/tape that will cause the directory listing to scroll off the screen. Observe the difference in blinking rate.)
2. In the graphics modes (`GR`, `HGR`, and `HGR2`), the VDP interrupt is turned off completely. Thus, the main program can read/write the VDP with impunity. Returning to `TEXT` mode restarts the VDP interrupt. (In SmartBASIC 1.x, a software clock is installed as part of the VDP interrupt routine, ticking at 60 (50) times per second; the clock stops whenever you are in a graphics mode, because the VDP interrupt is disabled.)

## III. Interrupt Deferral in EOS-8 VRAM Routines: Genesis.

At ADAMcon IV, whence the EOS-8 project was hatched, one of the topics of discussion was this very problem of non-reentrancy of the EOS VDP I/O routines. Based upon study of disassembled EOS code, Bruce Walters had suggested a particular code fix which he thought might solve the problem. In brief, the current EOS-5 code reads/writes the VDP one byte at a time, using `IN/OUT` instructions in a software loop. The Z80 also provides instructions which can read/write multiple bytes to/from a memory buffer without a software loop, namely `INIR/OUTIR` (in/out-increment-repeat). These commands use the HL register as a pointer to the buffer, the C register as the port to read/write, and the B register as a counter. Data is transferred via port C; HL is incremented and B is decremented; transfer stops when B is zero. Bruce had hoped that `INIR/OUTIR`, being single commands, were not interruptable (i.e., they would not respond to an interrupt until B=0); but unfortunately, they *are* interruptable.

I reproduce below a post to the Programmer's Forum on Mark Gordon's Micro Innovations BBS, dated 31 July 1992 (about 2 weeks after ADAMcon IV), in which Chris Braymen and I discuss the issue. It was in writing this post that I thought of a mechanism for VDP interrupt deferral for the EOS VRAM routines.

---

## ** VDP INTERRUPT HANDLING DISCUSSION **

[Note: I am using Internet E-mail format to respond to the technical discussion, so I can quote relevant parts of previous posts in my response. The T-BBS message editor does not allow this. -- Rich Drushel (75) ]

In a previous article, Chris Braymen (6) says:

```
>Hi Folks:  A discussion took place at ACIV that I want to make sure I
>understand..  The PROBLEM: You will end up with scrambled graphics if you
>read VDP register 8 (to restart the video interrupt) while you are in the
>process of accessing the VDP.
```

According to *The TMS 9118/28/29 Data Manual* (Texas Instruments, 1984),

"In an interrupt-driven environment (CPUs accepting interrupts), it is possible for an interrupt to occur before any one of the [multi-byte VDP access] sequences is finished. For example, an interrupt may occur immediately after loading address byte 1 or 2 during a write to VRAM operation. In this case, the interrupt service routine does not know where the interrupt occurred within the sequence. Therefore, it is necessary to disable and enable interrupts before and after every setup sequence. This action sequence prevents loss of continuity between the CPU and the VDP" (p. 2-9).

This means that *ANY* VRAM access (read data, write data, write register, read register 8) will be corrupted if an interrupting routine also tries to access VRAM.

```
>Most Coleco software solves this problem by maintaining a couple of flags
>that effectively delay the video interrupt restart until after the
>application is done using the VDP.
```

ColecoVision games employ an OS-7 routine which defers the writing of graphical objects (as defined with certain data structures) until desired by the programmer. Usually, VDP writes are deferred *until* an NMI occurs. At that point, the VDP is uninterruptable, so all the deferred writes are done until the deferral queue (or stack) is empty. This routine works only with the special graphical objects, not low-level VDP reads/writes or register access. In SmartBASIC, NMIs are simply disabled in `GR`, `HGR` and `HGR2`, ducking the problem. In `TEXT` mode, NMIs are enabled, and the NMI routine changes the pointer between normal and inverse pattern name tables (if `FLASH` is enabled) after a defined number of cycles. This is fine except that `PRINT` accesses the VRAM, as well as the actual `TEXT` command. The solutions are: The NMI routine checks to see if `PRINT` is executing, and if so, just does `RETN` (with no register 8 read to restart the NMI). The `PRINT` routine itself maintains the `FLASH` counter and will switch the name table pointers, and then issue a register 8 read. (Check the `FLASH` frequency when sitting waiting at the prompt versus when text is `PRINT`ing and scrolling on the screen; it is noticeably slower while `PRINT`ing.) Other programs leave the NMI disabled throughout.

```
>A proposal was made that this problem could be solved in the EOS system
>software by using OTIR instructions in place of the current OUT and OUTI
>instructions.
```

Having re-examined my disassembly of EOS-5, I believe that this is no longer a viable solution. See below.

```
>QUESTIONS:  My understanding of this problem holds that:
>
>      A) The problem occurs as a direct result of reading VDP register 8 and
>has absolutely nothing to do with the amount of time spent away from the VDP
>operation at hand during interrupt processing.
```

*Any* VRAM register read/write, or VDP data read/write, is vulnerable if interrupted by a routine which also attempts to access VRAM. It is not restricted to the register 8 read to restart the NMI for another cycle. Remember that the register 8 read is *OBLIGATORY* to restart the NMI; `RETN` is not sufficient.

```
>      B)  The problem will occur if register 8 is read between sending the
>2 bytes that make up a VDP command, OR if register 8 is read between
>reading or writing consecutive data using the auto incrementing VRAM
>address.
```

This is correct.

```
>Some people at the table indicated the problem only exists during VDP
>command access because the EOS already uses OTIR and INIR for data reads
>and writes.  This is not true!  The EOS functions Read_VRAM and
>Write_VRAM use OUTI and INI enclosed in loops, they do NOT use the
>non-interruptable commands OTIR and INIR.
```

You are correct. What's more, the register writes use separate `OUT` commands, the space between which is vulnerable to interrupts.

[note added 9709.14: `OTIR` and `INIR` *are* interruptable, so the point is moot]

```
>So my questions are:
>
>     1)  Am I correct in assuming the VDP hardware will get messed up if
>register 8 is read during Command sending OR during data read and write?
```

You are correct.

```
>     2) If so, will the EOS rewrite use OTIR and INIR in place of OUTI
>and INI in the Read_VRAM and write_VRAM routines?
```

It can't, because of 3) below.

```
>     3) If so, how will you incorporate the 2 NOP delay for the "Slow"
>VDP between each read/write?
```

You can't. So unless Bruce's code with `OTIR` and `INIR` actually does work, meaning the "slow" VDP on the design board wasn't really so "slow" in practice, I don't think you can use `OTIR` and `INIR` to make the write/read uninterruptable.

```
>And most importantly <grin>, if the OTIR and INIR opcodes are un-
interruptable
>(except by DMA), how can I make sure I'm not losing MIDI bytes during large
>transfers to the VDP?
```

Only by not using `OTIR` and `INIR` `:)`

```
>RECOMMENDATIONS:  I recommend leaving the system VDP access completely
>interruptable and making it the application programmers responsibility to
>handle the Video interrupt restart at a time when he/she is not doing
anything
>else with the VDP.
```

I agree 100% here. However, I would like to propose a mechanism whereby interrupt routines *WHICH ACCESS VRAM* could do so without fear of corrupting a main program VRAM access in progress.

[description omitted; see IV below]

```
>I can live with 2 byte interrupt disabling transfers, but I'll have trouble
>with any interrupt disabling transfer that's bigger than about 80 bytes.
```

If there isn't a bug in what I just proposed, you should be safe. INTs are enabled throughout all EOS VRAM function calls, and as long as your INT routine does not access VRAM, you could ignore the above.

```
>CLOSING: Please take this into consideration when designing the new EOS.
```

I am trying `:)`

```
>Spinner applications may also be adversely affected by an OTIR in write_VDP.
```

Absolutely.

```
>Thanks, Chris Braymen (6)
```

Regards, Rich Drushel (75)

---

## IV. Interrupt Deferral in EOS-8 VRAM Routines: Algorithms.

I reproduce here the design summary of the VRAM interrupt deferral routines for EOS-8, as posted to the Programmer's Forum on Mark Gordon's Micro innovations BBS on 6 October 1992. The algorithms are presented as pseudocode and as skeleton assembler. The complete assembler version as used in EOS-8 (and in a proof-of-concept version of SmartBASIC 1.x) is available for download.

- vram_def.asm, EOS-8 VRAM interrupt deferral routines

---

## EOS-8 VRAM INTERRUPT DEFERRAL ROUTINES.

designed by Richard F. Drushel 6 October 1992

Here is a mechanism whereby interrupt routines *WHICH ACCESS VRAM* could do so without fear of corrupting a main program VRAM access in progress. (If your interrupt routine does not access VRAM, there is no problem). It would work best with the NMI (because of its relatively low frequency; an INT can occur at any frequency) but might be useful for INTs as well. It would not improve existing interrupt routines, but could be used as a paradigm for future ones or rewrites. Anyway, code would be required both in the interrupt routine and in each EOS routine which accesses VRAM. EOS would define an interrupt status byte with 3 flag bits:

```
bit 0  ==>  vram_access       1 if an EOS function call accessing
                              VRAM is in progress, 0 if not.
bit 1  ==>  nmi_request       1 if an NMI routine needs
                              to be executed, 0 if not.
bit 2  ==>  int_request       1 if an INT routine needs
                              to be executed, 0 if not.
```

The handshaking of interrupt deferral is such that it is impossible to defer *BOTH* NMIs and INTs simultaneously. Indeed, since a `READ_REGISTER` (read VDP register 8) is required to restart NMIs after each NMI cycle, this effectively means that INT routines which access VRAM *cannot* be deferred if the NMI is active. The logic of the implementation follows:

```
INT:      DI                              ;disable further INTs
          IF vram_access=1                ;if VRAM I/O is in progress...
             THEN int_request=1           ;...then request a deferred INT
                  RET                     ;and return to VRAM I/O
                                          ;*NOTE* INTs *DISABLED*
             ELSE int_request=0           ;else wipe the INT request flag
                  CALL do_int             ;do the user INT routine
                  EI                      ;reenable INTs
                  RETI                    ;return from maskable interrupt
          ENDIF
do_int:   {user INT routine goes here}
          RET

NMI:      IF vram_access=1                ;if VRAM I/O is in progress...
             THEN nmi_request=1           ;...then request a deferred NMI
                  RET                     ;and return to VRAM I/O
                                          ;*NOTE* NMIs *DISABLED*
             ELSE nmi_request=0           ;else wipe the NMI request flag
                  CALL do_nmi             ;do the user NMI routine
                  read VDP register 8     ;restart the NMIs
                  RETN                    ;return from non-maskable interrupt
          ENDIF
do_nmi:   {user NMI routine goes here}
          RET
```

```
EOSVRAM:  IF vram_access=1              ;if VRAM I/O is already in
progress...
                                        ;i.e. this is a VRAM call made from
                                        ;inside another VRAM call, already
                                        ;flagged
             THEN JR do_vram_routine    ;...then just do it
                                        ;this allows the interrupt routine to
                                        ;use EOS VRAM functions, where it is
                                        ;safe from interruption
             ELSE vram_access=1         ;else mark that we're doing VRAM I/O
                                        ;*NOTE* now we can't be interrupted;
                                        ;INTs and NMIs will be deferred
                  CALL do_vram_routine  ;do the VRAM routine without fear
                  IF int_request=1      ;now if a deferred INT occurred
                                        ;while we were doing the VRAM I/O...
                     THEN int_request=0 ;...then clear the request
                          CALL do_int   ;do the user INT routine
                          vram_access=0 ;clear the VRAM usage flag
                          EI            ;*FINALLY* reenable INTs
                          RETI          ;and return from the interrupt
                                        ;We have done our main VRAM I/O *and*
                                        ;our INT VRAM I/O.
                     ELSE
                  ENDIF
                  IF nmi_request=1      ;now if a deferred NMI occurred
                                        ;while we were doing the VRAM I/O...
                     THEN nmi_request=0 ;...then clear the request
                          CALL do_nmi   ;do the user NMI routine
                          vram_access=0 ;clear the VRAM usage flag
                          read VDP reg8 ;*FINALLY* reenable NMIs
                          RETN          ;and return from the interrupt
                                        ;We have done our main VRAM I/O *and*
                                        ;our NMI VRAM I/O.
                      ELSE
                   ENDIF
                   vram_access=0        ;all done using VRAM
                   RET                  ;we're done!
```

The actual implementations are:

```
INT:
A56:
      PUSH AF                    ;save register
      LD A,(INTERRUPT_FLAGS)     ;point to interrupt flags
      BIT 0,A                    ;is a VRAM I/O in progress?
      JR Z,A71                   ;NO, so just do the routine
      SET 2,A                    ;YES, so request a deferred INT
      LD (INTERRUPT_FLAGS),A     ;save it back
      POP AF                     ;restore AF
      RET                        ;return with INTs *DISABLED*
A71:
      RES 2,A                    ;clear request flag for safety
      LD (INTERRUPT_FLAGS),A     ;save it
      CALL A83                   ;do the user INT routine (doesn't need
                                 ;to PUSH AF)
      POP AF                    ;restore AF
      EI                        ;reenable INTs
      RETI                      ;return from maskable interrupt
A83:
      JP user_int_routine       ;jump to the actual user code
                                ;it must save AF, HL and any other used
                                ;registers, and end in RET

NMI:
A102:
      PUSH AF                   ;save register
      LD A,(INTERRUPT_FLAGS)    ;get interrupt flags
      BIT 0,A                   ;is a VRAM I/O in progress?
      JR Z,A117                 ;NO, so just do the routine
      SET 1,A                   ;YES, so request a deferred NMI
      LD (INTERRUPT_FLAGS),A    ;save it back
      POP AF                    ;restore AF
      RET                       ;return with NMIs *DISABLED*
A117:
      RES 1,A                   ;clear request flag for safety
      LD (INTERRUPT_FLAGS),A    ;and save it back
      CALL A133                 ;do the user NMI routine (doesn't need
                                ;to PUSH AF)
      IN A,(191)                ;restart the NMI by reading VDP register 8
      LD (VDP_STATUS_BYTE),A    ;save it for EOS usage
      POP AF                    ;restore AF
      RETN                      ;return from non-maskable interrupt
A133:
      JP user_nmi_routine       ;jump to the actual user code
                                ;it must save AF, HL and any other used
                                ;registers, and end in RET

;end code which must be installed by the user application

;begin code which is in EOS RAM

VRAM:
      PUSH HL                   ;save register
      LD HL,INTERRUPT_FLAGS     ;point to interrupt flags
      BIT 0,(HL)                ;is a VRAM I/O in progress?
      JR NZ,DO_VRAM2            ;YES, so just do it
                                ;(interrupts are already deferred)
      SET 0,(HL)                ;NO, so mark it now in progress
      POP HL                    ;restore HL
      CALL DO_VRAM              ;do the VRAM routine
VRAM_COMMON_EXIT:
      PUSH HL                   ;save HL
      LD HL,INTERRUPT_FLAGS     ;point to interrupt flags
      BIT 2,(HL)                ;is there a deferred INT?
      JR NZ,DO_DEF_INT          ;YES, so do it
      BIT 1,(HL)                ;is there a deferred NMI?
      JR NZ,DO_DEF_NMI          ;YES, so do it
      RES 0,(HL)                ;NO, so no deferred interrupts; all done
      POP HL                    ;restore HL
      RET                       ;bye!
DO_DEF_INT:
      RES 2,(HL)                ;clear request flag
      CALL A83                 ;do user INT routine
      RES 0,(HL)               ;all done with VRAM routine
      POP HL                   ;restore HL
      EI                       ;finally reenable INTs
      RETI                     ;return from maskable interrupt
DO_DEF_NMI:
      RES 1,(HL)               ;clear request flag
      CALL A133                ;do user NMI routine
      RES 0,(HL)               ;clear VRAM flag
      POP HL                   ;restore HL
      PUSH AF                  ;save AF
      IN A,(191)               ;restart NMIs by reading VDP register 8
      LD (VDP_STATUS_BYTE),A   ;save it for EOS
                               ;VDP_STATUS_BYTE is in EOS global RAM
      POP AF                   ;restore AF
      RETN                     ;return from non-maskable interrupt
DO_VRAM2:
      POP HL
DO_VRAM:
     {routine here}
      RET
```

## V. For Next Time.

If I get any questions or other expressions of interest from the readers, I can talk about a few more details of the interrupt deferral code, which have been omitted above. For instance, there are some EOS VRAM routines which *cannot* be deferred; the technical reasons for this are perhaps of some interest.

Otherwise, I will start to write a technical description of ADAMserve (which I have been meaning to do for quite some time).

See you next week!

*Rich*

# This Week With My Coleco ADAM 9709.07

by Richard F. Drushel

## I. More About the EOS-8 Project.

I've had a few inquiries about the internals of EOS-8, my in-progress/abandoned upgrade to the EOS operating system. I was digging through some old correspondence and found some informational descriptions which might be of interest. I also decided that I would post some commented code for the universal serial and parallel drivers that I wrote, since these are in active use in ADAMserve (and also SmartBASIC 1.x, whence they were derived). This makes a rather easy *TWWMCA* article for me, since I did the actual writing in 1992 `:-)` But *you've* never seen it before, so I'm not blushing too badly about dredging it up.

## II. A Detailed Description of EOS-8.

Here is a helpfile I wrote for the benefit of the other programmers who were nominally helping with the EOS-8 project. This accompanied a complete source listing and bootable SmartBASIC 2.0 binary which had been uploaded to Mark Gordon's BBS. I haven't found my disk with this binary yet, but when I do, I will put an .IMG of the disk up for download.

Two errors of faulty recollection I made in *TWWMCA* 9708.31: the prototype ADAMnet parallel device is accessed as `PR #2` from SmartBASIC 2.0, and the serial device as `PR #4` and `IN #4`. I will correct the archived version of this article; newsletter editors and other archivers, please make this fix or grab my fixed version.

---

## *** EOS87.HLP *** help file for EOS8 rev.7 (9209.20)

by Richard F. Drushel

This file provides a brief overview of the design philosophy and implementation of EOS8, specifically rev.7. This should serve as a guide for outside reviewers and programmers as EOS8 is finalized.

---

### WHAT'S NEW IN EOS8 rev.7.

- Logical-to-physical device mapping. ADAMnet device numbers can be reassigned to non-ADAMnet physical devices, to allow existing programs to function with non-ADAMnet hardware (i.e., use a parallel printer whenever a call is made to the ADAM printer). Device numbers 32-63 are reserved for up to 32 non-ADAMnet devices; the following are defined:

| | | |
|---|---|---|
| 32 | `USERPAR_ID` | parallel port (64) |
| 33 | `USERSER_ID` | serial port (any Eve/Orphanware/MIB2) |
| 34 | `USERAMDISK_ID` | RAMdisk using standard XRAM |
| 35 | `USERHD_ID` | Hard disk (any Mini Wini/SASI/IDE) |

- This includes complete emulation of ADAMnet devices by non-ADAMnet equivalents. Physical devices use dummy DCBs which are set up as if they were controlling an actual ADAMnet device. This includes the status byte at (DCB+0) and the device-dependent status at (DCB+20). For serial devices, the (DCB+20) status bits for # characters pending and ok to send are maintained exactly like the prototype ADAMnet serial board.
- I/O for non-ADAMnet devices is *NOT* concurrent/background. Thus, all I/O initiated with a `_START_...` call is performed to completion before exiting the call, including updating the dummy DCB. This means that `_END_...` calls for non-ADAMnet I/O don't do anything except return the status from the dummy DCB.
- Defined overlay area for hard drives. The overlay has a defined code and data interface so that any type of hard drive (or other block device) can be used. The overlay includes the volume offset table which is used to specify the volume sizes of the particular EOS partition. This overlay is positioned so that the current HD volume remains at absolute address 58343, for compatibility with existing programs. The overlay must be less than 400 bytes long. The overlay interface accommodates up to 16 EOS volumes, and has provisions for 2 hard drives per physical interface:
- `HD_MW1_CODE    EQU   1`
- `HD_MW2_CODE    EQU   2`
- `HD_SASI1_CODE  EQU   3`
- `HD_SASI2_CODE  EQU   4`
- `HD_IDE1_CODE   EQU   5`
- `HD_IDE2_CODE   EQU   6`

Note: the current version was assembled with Mini Wini HD drivers in place, assuming 10 EOS volumes 1024 blocks each. You will have to reassemble it for your own HD characteristics to access any volume other than 0; 0 will always work :)

- Hotkey functions for hard drives. The following keys are trapped:

| | |
|---|---|
| `Shift-Undo` | reboots HD from volume 0/block 0 |

| | |
|---|---|
| `Shift-WildCard` | parks HD heads |
| `Shift-Tab` | toggles ahead the current HD volume, wrapping back to 0 when `MAX_VOLUMES` is exceeded |

- User-defined hotkey. This was implemented as part of EOS7. The user specifies a key to trap, and a vector to a handler routine. When this key is pressed, the handler routine is called; control returns to the main program.
- User-defined number of FCBs and DTAs. Instead of being hard-coded at 3, the user can change the contents of `NUM_FCBS` to have as many files open at once as desired. Make sure you `_FMGR_INIT` after changing `NUM_FCBS :)` This was also part of EOS7.
- NMI clock driver routine. Each `CALL` to this routine updates the clock by one tick, complete with weekday, day, month, and year rollover. Additional locations in `EOS_GLOBAL_RAM` are defined to hold 60th of second, second, minute, hour, and weekday.
- Auxiliary jump table for direct access to non-ADAMnet devices and extra EOS functions.

---

## WHAT'S MISSING FROM EOS8 rev.7.

- Hardware initialization routines for both Eve/Orphanware and MIB2 serial ports. A management decision is needed regarding the parameter passing mechanism--namely, will we remain compatible with the prototype ADAMnet serial board (device 14)? The difficulty with this approach is that the values 0-5 will have to become reserved data values which flag initialization parameters--you would lose the ability to send them as serial data independent of initialization, for the non-ADAMnet serial boards. If initialization compatibility is not maintained, then (1) dedicated init routines will be required for non-ADAMnet serial boards, and (2) it will not be possible for non-ADAMnet serial boards to emulate the initialization code for device 14. SmartBASIC 2.0 does not attempt to initialize device 14, but it is unknown if any other Coleco EOS software attempts to do so.
- Trapping of serial port overrun, parity, and framing errors. The source code has conditional assembly of traps for these errors. The enclosed binary has these left out, to avoid problems with SmartBASIC 2.0. (The way the code is in SmartBASIC 2.0, if you type too fast and overrun the port, the routine will loop forever; unlike SmartBASIC 1.x, it will not force a `PR#0/IN#0` if an error occurs.) You can reassemble the source with the traps put back in if you like `:)`
- Software RAMdisk routines. Since EOS8 ver.7 is currently larger than 8K, EOS code has spilled into what under EOS5 is the 2nd user DTA. The existing prototype RAMdisk routine (demonstrated at ADAMcon IV) appropriates this area for its own inter-bank transfers. While it is possible to rewrite the RAMdisk I/O to page through a smaller buffer, I have decided to wait until both the hardware clock I/O routines and the VRAM

interrupt deferral routines are implemented, so I know how large a RAMdisk buffer I will have available (either 256 or 512 byte).

- The 2nd user DTA. To my knowledge, only SmartBASIC actually allows the user to open 2 files at once. Moreover, because the file I/O routines in SmartBASIC are defective, there is no practical use for 2 open files. EOS8 allows the application programmer to both set the number of DTAs and to relocate the DTAs/FCBs, so future programs can have more than 1 user file open at once if proper setup is done. At least one existing program, however (ADAMlink V) uses the 2nd DTA at its absolute address under EOS5 (not relative to the pointer to start of DTAs) as a transfer buffer; these unfortunate programs overwrite the larger EOS8. SmartBASIC hackers also have been known to store machine code routines here.
- Hardware clock I/O routines. I can extract them from SmartBASIC 1.x, but have decided to wait for comments about what I have done already.
- VRAM I/O deferral during interrupts. The logic of this deferral has been described previously (in VRAMDISC.TXT).
- `BLOCKS LEFT`. EOS7 unfortunately does *NOT* write `BLOCKS LEFT` as the name of the "not a file" directory entry. Many existing programs look specifically for `BLOCKS LEFT`, and hence will crash under EOS7-8. I have not yet looked into fixing this.
- Bidirectional parallel printer emulation for SmartWriter. I have not included the routine which intercepts the printhead direction changes sent by SmartWriter to the ADAM printer. This means that text printed from SmartWriter will have every other line spelled backwards. (This does not happen from the Electronic Typewriter mode.) An 80-byte line buffer and line inversion software must be added to `__PR_CH` and `__PR_BUFF`.

---

## HOW TO LOAD EOS8 rev.7.

EOS8 rev.7 is currently part of a modified SmartBASIC 2.0 which lacks all the `EXTMEM` features. (I did it this way as a quick-and-dirty loader, also to test the logical-to-physical remapping of `PR#2/PR#4/IN#4`.) Put the disk in any drive and pull the reset. Once SmartBASIC 2.0 comes up, you can put any other bootable disk in the drive and `CALL 64560` (`_EOS_START`). The disk will be booted, but the EOS8 will still be in effect.

Since there aren't yet any serial port init routines, you must use some other program to init them *BEFORE* you load EOS8, or else boot SB1.x under EOS8 and use the `SERIAL` command. My primary testing tool has been SB1.x, because it lets me do EOS function calls directly via register variable setups and `CALL EOS(nn)`. Also, the default value at `USER_BASEPORT` is 0 (no serial port). You will have to `POKE 65206,baseport` in order to use device 14 as a non-ADAMnet serial port, where baseport maps as:

```
    SER68_PORT       EQU    68
    SER76_PORT       EQU    76
    SER84_PORT       EQU    84
    SER92_PORT       EQU    92
    SER_MIB21_PORT   EQU    24
    SER_MIB22_PORT   EQU    16
```

The current logical-to-physical mapping is:

```
    device 2 (ADAM printer)         --->  USERPAR_ID
    device 13 (Coleco Centronics)   --->  USERPAR_ID
    device 14 (Coleco RS-232)       --->  USERSER_ID
    device 24 (tape 2)              --->  USERHD_ID
```

---

## NEW GLOBAL ENTRY POINTS IN EOS8 rev.7

```
58343   HD_OVERLAY:
        HD_CURRENT_VOLUME:
            DS 1
58344   HD_VOL_OFFSET_TABLE:
            DS 16

;note:  the following 4 JPs are for *INTERNAL EOS USE ONLY*, but are
;fixed entry points which the overlay programmer must provide.

58360   HD_STATUS:                      ;get dummy DCB status
            JP __HD_STATUS
58363   HD_RESET:                       ;park heads
            JP __HD_RESET
58366   HD_WRITE:                       ;write 1 block
            JP __HD_WRITE
58369   HD_READ:                        ;read 1 block
            JP __HD_READ
58372   USER_HD:                        ;code for HD type
            DS 1
[...]

;SECONDARY EOS JUMP TABLE.
;     This is for the additional EOS function calls.  The bottom of the
;table is fixed, so new entries must be added to the *BEGINNING*.  Do *NOT*
;delete any entries!  Also, make sure that the JUMP2_COUNT variable is
;updated whenever you add jump table entries!

JUMP2_COUNT     EQU     9

64484   _GET_HW_CLOCK      EQU  $    ;read hardware clock time to system
clock
            JP      __GET_HW_CLOCK
64487   _SET_HW_CLOCK      EQU  $    ;write system clock time to hardware
clock
            JP      __SET_HW_CLOCK
64490   _UPDATE_NMI_CLOCK  EQU  $    ;system clock tick (every NMI)
            JP      __UPDATE_NMI_CLOCK
64493   _WRITE_PAR         EQU  $    ;write character to parallel port
            JP      __WRITE_PAR
64496   _READ_SER          EQU  $    ;read character from serial port
            JP      __READ_SER
64499   _WRITE_SER         EQU  $    ;write character to serial port
            JP      __WRITE_SER
64502   _INIT_SER          EQU  $    ;initialize serial port
            JP      __INIT_SER
64505   _GET_PHYS          EQU  $    ;get the physical device now using a
                                     ;logical device
            JP      __GET_PHYS
64508   _SET_PHYS          EQU  $    ;specify the physical device to be used
                                     ;by a logical device
            JP      __SET_PHYS

; *** _SET_PHYS IS THE LAST ENTRY!! ***
; *** DO NOT ADD TO THE END OF THIS TABLE!! ***

[...]

65196   TRIGGER_CHAR:
            DS 1
65197   USR_DFND_RTN:
            DS 2
65199   NUM_FCBS:
            DS 1

;begin EOS8 global RAM

65200   EOS_60TH:
            DS 1            ;RFD
65201   EOS_SECOND:
            DS 1            ;RFD
65202   EOS_MINUTE:
            DS 1            ;RFD
65203   EOS_HOUR:
            DS 1            ;RFD
65204   EOS_WEEKDAY:
            DS 1            ;RFD

65205   INTERRUPT_FLAGS:              ;store INT/NMI flags for VRAM routines
            DS 1            ;RFD

65206   USER_BASEPORT:                ;serial port to be used for ADAMnet
            DS 1            ;RFD      ;serial board emulations
65207   USER_SER_STATS:               ;in case we emulate ADAMnet serial
board
            DS 6            ;RFD      ;initializations (hold parameters)
            DS 3                      ;added by RFD to reserve space

65216   PCB:
            DS P_SIZE
65220   DCBS:                         ;only 12 real ADAMnet devices permitted
            DS 12*D_SIZE

;*********************************************************************
DUMMY_DCBS:
;     Note:  these will *NOT* be relocated if the PCB is
;relocated!  So if you application is switching to XRAM or
;ROM up here, you will *LOSE* these devices!!!

;     Note 2:  these are *NOT* user entry points!  This is for reference
;*ONLY*!!  Always access the DCBs indirectly, via _FIND_DCB.  The parallel
;DCB is currently stored at DCB_IMAGE, but this is subject to change.

65472   SERIAL_DCB:
            DS 21
```

```
65493   RAMDISK_DCB:
            DS 21
65514   HARDDISK12_DCB:
            DS 21
65535   RESERVED_BYTE:
            DS 1
```

---

## III. Universal Non-ADAMnet Serial and Parallel Drivers.

Here is commented source code for drivers for the Orphanware and MIB2 serial ports and the Orphanware parallel printer port. These were written to be a "universal" module--just include it in your program, and it can support all 6 serial ports as well as the printer port. For the serial ports, the serial baseport to use is passed in the C register; characters are passed in the A register; the Z flag reflects the status; if an error occurred, A has the error code.

The other great advantage of this code is that it implements a 10-second timeout. If the I/O is not successfully completed within 10 seconds (software loop timing assuming 4 mHz Z80), it times out and returns an error code. This prevents infinite polling loops which require a hard reboot to get out of if you specify the wrong port, or if a printer or modem isn't on-line.

Feel free to use this code. Note the conditional assembly of tests for overrun, framing, or parity errors for the serial drivers. I left them conditional in EOS-8 to save some space in EOS RAM; they are active in the SmartBASIC 1.x code. If these errors occur, the only way to clear them is to reinitialize the serial port.

I haven't included universal serial port initialization code here. It's present in the boot for the ADAMserve boot disk, again requiring the serial baseport passed in the C register, and the other serial port parameters (bitrate, parity, number of stop bits) passed in other registers. I can post this code at a later time.

```
;***************************************************************************
;Non-ADAMnet Serial Read Character With Timeout Check.
;     On entry, C=base port.  On exit, if the read was ok, ZF=1 and
;A=character, else ZF=0 and A=error code.  Routine retries approximately 10
;seconds before timing out.  C is preserved.

;errors

;CHAR_DEV_TIMEOUT_ERR   EQU  25       ;parallel or serial timed out
;CHAR_DEV_NO_PORT_ERR   EQU  26       ;no port found for parallel or serial
;CHAR_DEV_OR_F_P_ERR    EQU  27       ;serial had overrun/framing/parity
error
;PRINTER_OFFLINE_ERR    EQU  28       ;parallel printer is off-line

;serial baseports

;   SER68_PORT        EQU     68
;   SER76_PORT        EQU     76
;   SER84_PORT        EQU     84
;   SER92_PORT        EQU     92
;   SER_MIB21_PORT    EQU     24
;   SER_MIB22_PORT    EQU     16

;********

__READ_SER:
    PUSH HL                 ;save so we can use it for inner loop counter
    PUSH DE                 ;save so we can use it for outer loop counter
    PUSH AF                 ;not needed per se, but makes exits compatible
                            ;with SER_SEND_TCHK
    INC C                   ;point to status port
    LD D,8                  ;outer loop counter
    LD A,C                  ;get baseport
    CP SER68_PORT           ;is it Eve-Orphanware?
    JR NC,EVE_OWARE_READ    ;YES

MIB2_READ:
OUTR_LP_MI_RD:
    LD HL,65535             ;inner loop counter
INNR_LP_MI_RD:
    IN A,(C)                ;check the status port
    LD E,A                  ;save it in E
    CP 255                  ;does this port even exist?
    JR Z,NO_PORT            ;NO

IF OFP_CHK
    AND 240                 ;YES, so 11110000 check upper 4 bits
    JR NZ,OR_FR_P_ERR       ;sorry, errors
    LD A,E                  ;check the status again
ELSE
ENDIF

    BIT 0,A                 ;is there a character to read?
    JR NZ,READ_RDY_MI       ;YES (bit set)
    DEC HL                  ;NO, so one less inner loop
    LD A,H
    OR L                    ;down to zero yet?
    JR NZ,INNR_LP_MI_RD     ;NO, so keep going on inner loop
    DEC D                   ;YES, so one less outer loop
    JR NZ,OUTR_LP_MI_RD     ;reset the inner loop and keep trying
    JR TIME_OUT             ;sorry, we've timed out!
;********
READ_RDY_MI:
    POP AF                  ;restore character
    POP DE                  ;all done with outer loop counter
    INC C
    INC C                   ;point to data port
    IN A,(C)                ;get character
RDY2:
    DEC C
    DEC C
    DEC C                   ;restore C to entry state (baseport)
    JR SET_ZF2              ;ZF=1 for ok exit, leaving A=character
;********
EVE_OWARE_READ:
OUTR_LP_EO_RD:
    LD HL,65535             ;inner loop counter
INNR_LP_EO_RD:
    IN A,(C)                ;read status port
    LD E,A                  ;save it in E
    CP 255                  ;does the port exist?
    JR Z,NO_PORT            ;NO

IF OFP_CHK
    AND 56                  ;YES, 00111000 any framing/parity/overrun errors?
    JR NZ,OR_FR_P_ERR       ;YES, so exit
    LD A,E                  ;NO, so check status again
ELSE
ENDIF

    AND 2                   ;is there a character ready?
    JR NZ,READ_RDY_EO       ;YES, so get it
    DEC HL                  ;NO, not yet, so one less inner loop
    LD A,H
    OR L                    ;are we down to zero?
    JR NZ,INNR_LP_EO_RD     ;NO, so keep trying on inner loop
    DEC D                   ;YES, so one less outer loop
    JR NZ,OUTR_LP_EO_RD     ;not done yet, so reset inner loop and try again
    JR TIME_OUT             ;we have timed out!
;********
READ_RDY_EO:
    POP AF                  ;restore character
    POP DE                  ;all done with big loop counter
    DEC C                   ;back up to data port
    IN A,(C)                ;read the character
    INC C                   ;restore C=status port
    JR SET_ZF2              ;ZF=1 for ok exit, leaving character in A
;********
SET_ZF:
    PUSH HL
SET_ZF2:
    LD L,A              ;save A
    XOR A               ;ZF=1
    LD A,L              ;restore A
    POP HL
    RET


;***************************************************************************
;Non-ADAMnet Serial Send Character With Timeout Check.

;Modified PR #3 routine from SmartBASIC 1.x version 20Y by Richard F. Drushel
;Removed error jumps to force PR #0 and print error messages, and cleaned up
;some spaghetti left over from binary patching.

;     On entry, C=base port and A=character.  On exit, if the send was ok,
;ZF=1 and A=character, else ZF=0 and A=error code.  Routine retries
;approximately 10 seconds before timing out.  C is preserved.

;errors:

;CHAR_DEV_TIMEOUT_ERR   EQU  25       ;parallel or serial timed out
;CHAR_DEV_NO_PORT_ERR   EQU  26       ;no port found for parallel or serial
```

```
;CHAR_DEV_OR_F_P_ERR    EQU  27       ;serial had overrun/framing/parity
error

__WRITE_SER:
    PUSH HL                 ;save so we can use it for inner loop counter
    PUSH DE                 ;save so we can use it for outer loop counter
    PUSH AF                 ;save character
    INC C                   ;make status port
    LD D,8                  ;outer loop counter
    LD A,C                  ;get baseport
    CP SER68_PORT           ;is it Eve-Orphanware?
    JR NC,EVE_OWARE_SEND    ;YES

MIB2_SEND:
OUTR_LP_MI_SND:
    LD HL,65535             ;inner loop counter
INNR_LP_MI_SND:
    IN A,(C)                ;check the status port
    LD E,A                  ;save it in E
    CP 255                  ;does this port even exist?
    JR Z,NO_PORT            ;NO

IF OFP_CHK
    AND 240                 ;YES, so 11110000 check upper 4 bits
    JR NZ,OR_FR_P_ERR       ;sorry, errors
ELSE
ENDIF
    BIT 3,E                 ;can we send a character?
    JR NZ,SEND_RDY_MI       ;YES (bit set)
    DEC HL                  ;NO, so one less inner loop
    LD A,H
    OR L                    ;down to zero yet?
    JR NZ,INNR_LP_MI_SND    ;NO, so keep going on inner loop
    DEC D                   ;YES, so one less outer loop
    JR NZ,OUTR_LP_MI_SND    ;reset the inner loop and keep trying

TIME_OUT:
    LD A,CHAR_DEV_TIMEOUT_ERR  ;say we're timed out
SER_SEND_BYE:
SER_READ_BYE:
    POP DE                  ;clear AF off stack
    POP DE                  ;the real DE
    POP HL                  ;restore HL
    DEC C                   ;restore C to entry state (baseport)
    OR A                    ;ZF=0 for error
    RET
;********
NO_PORT:
    LD A,CHAR_DEV_NO_PORT_ERR ;missing port
    JR SER_SEND_BYE           ;error exit
;********
OR_FR_P_ERR:
    LD A,CHAR_DEV_OR_F_P_ERR ;mark overrun/framing error
    JR SER_SEND_BYE          ;error exit
;********
SEND_RDY_MI:
    POP AF                  ;restore character
    POP DE                  ;all done with outer loop counter
    INC C
    INC C                   ;point to data port
    OUT (C),A               ;send it
    JR RDY2                 ;restore C, set ZF=1 and exit with A=character
;********
EVE_OWARE_SEND:
OUTR_LP_EO_SND:
    LD HL,65535            ;inner loop counter
INNR_LP_EO_SND:
    IN A,(C)               ;read status port
    LD E,A                 ;save it in E
    CP 255                 ;does the port exist?
    JR Z,NO_PORT           ;NO

IF OFP_CHK
    AND 56                 ;YES, so 00111000 any framing/parity/overrun
errors?
    JR NZ,OR_FR_P_ERR      ;YES, so exit
    LD A,E                 ;NO, so check status again
ELSE
ENDIF

    AND 1                  ;is it ready to receive a character?
    JR NZ,SEND_RDY_EO      ;YES, so send it
    DEC HL                 ;NO, not yet, so one less inner loop
    LD A,H
    OR L                   ;are we down to zero?
    JR NZ,INNR_LP_EO_SND   ;NO, so keep trying on inner loop
    DEC D                  ;YES, so one less outer loop
    JR NZ,OUTR_LP_EO_SND   ;not done yet, so reset inner loop and try again
    JR TIME_OUT            ;we have timed out!
;********
SEND_RDY_EO:
    POP AF                 ;restore character
    POP DE                 ;all done with outer loop counter
    DEC C                  ;back up to data port
    OUT (C),A              ;send the character
    INC C                  ;restore C=status port
    JR SET_ZF2             ;ZF=1 for ok exit, leaving character in A


;************************************************************************
;Parallel printer send with timeout check.
;Modified PR #2 routine from SmartBASIC 1.x version 20Y by Richard F. Drushel

;On entry, A=character to send.  On exit, if the send was successful, ZF=1
and
;A=entry character.  If the printer timed out (either not on-line or non-
;existent), ZF=0 and A=error code.

;errors:

;CHAR_DEV_TIMEOUT_ERR   EQU  25       ;parallel or serial timed out
;CHAR_DEV_NO_PORT_ERR   EQU  26       ;no port found for parallel or serial
;PRINTER_OFFLINE_ERR    EQU  28       ;parallel printer is off-line
```

```
__WRITE_PAR:
    PUSH HL
    PUSH BC                 ;save for loop counters
    PUSH AF                 ;save character
    LD B,9                  ;big loop counter
LITTLE_LOOP:
    LD HL,65535             ;little loop counter
ONLINE_CHECK:
    IN A,(PAR_PORT)         ;read the parallel port
    CP 255                  ;does it exist?
    JR Z,NO_PAR_PORT        ;NO, so no parallel port error
    AND 1                   ;is it on-line?
    JR NZ,PAR_READY_CHECK   ;YES
PAR_NOT_READY:
    DEC HL                  ;NO, so decrement little loop counter
    LD A,H
    OR L                    ;are we down to zero?
    JR NZ,ONLINE_CHECK      ;NO, so keep trying
    DEC B                   ;YES, so decrement big loop counter
    JR NZ,LITTLE_LOOP       ;not done yet, so restart little loop
PAR_TIMEOUT:
    IN A,(PAR_PORT)         ;we've timed out!  so let's find out why
    BIT 1,A                 ;so was it off-line?
    LD A,PRINTER_OFFLINE_ERR  ;let's assume not on-line
    JR NZ,PAR_ERR_EXIT      ;YES
    LD A,CHAR_DEV_TIMEOUT_ERR  ;NO, anything else is busy
PAR_ERR_EXIT:
    OR A                    ;ZF=0 for error
    POP BC                  ;get rid of AF on stack
    JR PAR_BYE_BYE          ;and exit
;********
PAR_READY_CHECK:
    IN A,(PAR_PORT)         ;read the parallel port
    AND 2                   ;is it ready to receive another character?
    JR NZ,PAR_NOT_READY     ;NO, so keep trying
    POP AF                  ;YES, so restore character
    OUT (PAR_PORT),A        ;send it
    CALL SET_ZF             ;ZF=1 for ok exit while preserving A
PAR_BYE_BYE:
    POP BC
    POP HL
    RET
;**********
NO_PAR_PORT:
    LD A,CHAR_DEV_NO_PORT_ERR  ;no parallel port found
    JR PAR_ERR_EXIT            ;so error exit
```

## IV. For Next Time.

In my next article, I will discuss interrupt deferral techniques to prevent reentrancy in the EOS video RAM routines. This will include a transcript of some discussions that Chris Braymen and I had on the subject, as well a complete pseudocode for the algorithms, which I successfully implemented in EOS-8, and in a modified version of SmartBASIC 1.x (as proof of concept).

## V. Administrivia.

I finally unpacked my 486 system after ADAMcon 09, so now I can start working on all the leftovers in my buffer, including a master set of distribution disks for ADAMserve (including ADAMserve PowerPaint, and user-configurability of server hardware). I will get lynched soon if I don't clear this out.

See you next week!

*Rich*

---

# This Week With My Coleco ADAM 9708.31

by Richard F. Drushel

## I. Technology Note.

This week's *TWWMCA* is being composed using a rare (but probably not valuable) IBM PC/XT clone from 1983, a TAVA PC (made by TAVA Co., whoever or whatever they are or were). There must have been enough of them around at one time to make it worthwhile for Norton Utilities version 4.5 to specifically detect them in the SI (SYSINFO) program. Externally, it's a dead ringer for the original IBM PC; internally, however, aside from having an ISA bus and the same arrangement of DIP switches for setting the memory and video configuration, it's totally different. I got this as a freebie from a lab postdoc when I was in graduate school. Mostly it has been used as an MFM hard drive formatting/testing system, and also to house the interface card for my EPROM burner. But after ADAMcon 09, my wife Joan (am335@po.cwru.edu) decided that she wanted to start looking at USENET newsgroups for some 60s rock-n-roll groups she's interested in, using the Cleveland Freenet. I have ADAMs in abundance that I could have used, but this PC system was already set up, would already do 80x24 screen and VT100 terminal emulation. So, until such time as I can make a turnkey system using an ADAM, TDOS or CP/M, and my genuine H19 serial terminal, I guess I'll let her use this system. Those who remember the door prize giveaways at the ADAMcon 08 banquet will remember the Hayes SmartModem 2400 I won from HLM-GMK Co.; this is the modem that I'm using now.

The other reason for using this TAVA system is that I still have been too busy/lazy to unpack my travelling box from ADAMcon 09, which has my 486 system in it. Right now, this TAVA PC is the only computer I have operational in my basement.

## II. ADAM Operating Systems.

Most of what I want to do in this issue of *TWWMCA* is think out loud and hopefully bounce some ideas off of other people's heads, regarding a topic that was my initial interest in the ADAM, and which has occupied most of my ADAM programming efforts: enhanced ADAM operating systems.

To review, the ADAM has currently 4 operating systems for which user software is available: OS-7 (the ColecoVision video game operating system), EOS (the Elementary Operating System developed by Coleco), Digital Research's CP/M (Control Program/Microcomputer), and an enhanced CP/M clone called TDOS (Tony's Disk Operating System, named after its author, Tony Morehen). OS-7 and EOS are present in ROM on every ADAM (OS-7 only on ColecoVisions, and EOS only on Expansion Module #3s). CP/M and TDOS must be run from tape or disk (though former B.A.S.I.C. member Ron Collins had an ADAM in which the SmartWriter word processor ROMs were replaced with TDOS, so this ADAM would boot to TDOS in the absence of an external disk or tape; I saw this system myself so it's not folklore). Here is a table summarizing the circulating versions of the different ADAM operating systems:

| operating system | version | date |
|---|---|---|
| OS-7 | 7PRIME | 28 December 1982 |
| EOS | 5* | 8 October 1983 |
| EOS | 7** | September 1984 |
| CP/M | 2.2 | 1984 |
| TDOS | 4.59 | 1992 |

Notes:

* This is the value found at REV_NUM (FD60 hex) in all existing ADAM EOS ROMs. However, the actual version of the operating system is 6, not 5. This is explained in the following comment from the EOS-6 source code:

```
EOS_REV          EQU     005H    ;current EOS revision number
                                 ;---NOTE---   this is actually rev.
                                 ;06 but to match the production ROM
                                 ;which was labeled rev. 05 we fudge
                                 ;just a bit (two bits actually)
```

(For the binary-impaired, the joke in the comment is that 5=0101, but 6=0110; two bit positions must be changed to go from 0101 to 0110.)

** EOS-7 was distributed on disk with the Disk Manager software, as well as with the abandoned SmartBASIC 2.0. EOS-7 was a total internal rewrite with several additional functions, including

1. the ability to trap a keypress as a "hotkey" and jump to a user-supplied service routine, and
2. the ability to have as many files open simultaneously as desired, provided the user allocated sufficient memory for disk buffers.

The major liability of EOS-7 is that it uses only 2-byte instead of 4-byte disk blocks, thus limiting disks to 65,535 blocks or 64 megabytes, instead of 4,294,967,296 blocks or 4 gigabytes.

## III. Limitations of ADAM Operating Systems.

All 4 ADAM operating systems have significant limitations which have hindered the development of new software (and hardware):

## OS-7:

The operating system itself is quite well-designed, being object-oriented and flexible. The chief hindrance has been lack of programmer documentation and suitable development tools, especially for creating background graphics screens and composing songs and sound effects. I have a 20th-generation photocopy of the *ColecoVision Programmer's Manual* (obtained in 1992 from A.N.N. founder Barry Wilson), which I have restored and made several copies of for interested parties; as well as the OS-7PRIME source code listing from the ADAM Technical Manual set. To my knowledge, since the ADAM/ColecoVision were dropped from the market, no new cartridge games have been written except for Marcel de Kogel's Cosmo Fighter, which appeared in spring 1997; this was a byproduct of Marcel's ColecoVision and ADAM emulator projects.

## EOS:

The biggest limitation of EOS is the sequential file system, which requires that the blocks of each file be contiguous; there are no EOS functions to "squeeze" out holes left by deleted files. There is no direct support for subdirectories or for partitioning large physical media into smaller logical drives. Finally, the ROM version present on every ADAM leaves unimplemented (i.e., just a `RET` instruction) the `_POSITION_FILE` function, which is supposed to be used to move a read/write pointer around in a file, to provide random access file I/O. Because of this deficiency, most existing EOS programs bypass EOS file I/O functions and read/write raw blocks of the files--ick. Third-party hardware can only be supported by

1. applications including their own driver code, or
2. direct patches to EOS after it is loaded from ROM to RAM.

## CP/M:

Pre-existing CP/M programs run fine under ADAM CP/M 2.2, except that the CP/M world assumes your screen is 80 columns by 24 rows; the ADAM screen is 32 columns by 23 rows, so you're stuck with an annoying, horizontally-scrolling window as a poor man's 80 column screen. While CP/M supports I/O redirection to and from serial devices (like modems and terminals), the ADAMnet serial/parallel card never left the prototype stage, so there is no serial/parallel access. (The routines to access the prototype are present, but useless in the absence of the actual hardware.)

## TDOS:

TDOS is modelled after ZCPR2 (an early CCP replacement for CP/M) and contains built-in support for all available 3rd-party ADAM hardware (non-ADAMnet devices like serial cards, parallel cards, hard drives, floppy drives larger than 160K, and memory expanders larger than 64K), so it's possible to replace the ADAM video screen with a real 80x24 serial terminal and use all the existing CP/M software without the moving text window hassle. The chief problem with TDOS is that its creator, Tony Morehen, is no longer active in the ADAM community and is hence unavailable to maintain the code. The TDOS source code is an unmanagable mess of uncommented spaghetti, full of confusing conditional assembly directives, scattered over too many non-modular source files. You must assemble a new version of TDOS from source for every possible hardware configuration, and the source is in such disarray that it's no longer expandable to accommodate new hardware. (The source is probably not yet freely-distributable, either; there's been no recent word from Tony about what he wants to do with it. Binaries are supposed to be freely-distributable, however, and have been.)

## IV. The Future is the Past.

Progress in the realm of ADAM operating systems is limited by the same issues which retard progress on Wintel and MacOS platforms: namely, the large installed base of existing software which needs to be supported, because of the economic clout of the users of that software. This is also known as "the backward compatibility problem". In short, it doesn't matter how nifty a new version of the operating system is; if switching to it means that everyone has to get (i.e., buy) new versions of their existing application programs, then very few people will switch. There are other contributing factors, but much of the internal disarray of Win3.1 and Win95 is due to this economic pressure to continue to support DOS software from the early and mid 1980s--software which did "bad" things like take over all the system resources, modify its own code, take advantage of undocumented features (which perforce become "documented" as the need to retain the existing behavior becomes paramount). Win95 makes few concessions to DOS programs, and generally tries to make their continued use unpleasant; but most Win3.1 programs still run under Win95--with the penalty that they destabilize the system for the true-blue Win95 applications. A totally- from-scratch Win95 might have been "better" from a computer science viewpoint (and maybe from a user viewpoint as well, in that the system would be more stable), but a certain financial disaster in the real world.

The backward compatibility problem is acutely worse for an obsolete system like the ADAM, for which 98% of all the software that will every be written *has* already been written. Unless someone (or a team of someones) is also prepared to provide a suite of compatible applications, writing a new and improved operating system from scratch is a mere academic exercise. The result will be interesting, but nobody will actually use it, because none of the existing applications people have a large investment in (over 10 years of active use in some cases) will run.

On the CP/M side, there is a lot less to worry about; CP/M was designed to be portable to many different hardware architectures (all using the 8080/Z80 CPUs, of course), and there is a great conformance of CP/M applications to the operating system standards. The trick, then, is to have a suitably conforming version of CP/M running on your own hardware; then you will never have to patch applications (except in very precise ways, such as escape codes for terminal programs). TDOS with a serial terminal for display does this adequately for existing hardware, but cannot now easily be altered to accommodate new hardware. Many TDOS users have been waiting since ADAMcon 06 for a TDOS-compatible version of my ADAMserve serially-linked device driver program (which lets an ADAM control PC drives and other hardware via serial link). In theory, it should be simple: replace the BIOS-level I/O routines with the ADAMserve protocol. Alas, the TDOS BIOS is not really a well-encapsulated BIOS module; it is inline spaghetti all over the place. Disentangling it would take more ADAM time than I have available at present.

On the EOS side, the applications are really more important than the operating system. The Smart[fill in the blank] applications were designed to completely take over the machine. EOS itself is little more than a program loader with some minimal system services. In the EOS world, you put in the program disk/tape, pull the reset switch, and run the program; to switch programs, you put in another disk/tape and pull the reset switch again. There is no concept of a shell, of programs exchanging information with each other except perhaps through files, and especially of sharing hardware resources. The worst offense in this regard is in the use of expansion memory. Just about every EOS program that finds a memory expander appropriates it for itself. Ironically, the contents of XRAM remain intact across system resets (as long as you don't turn off the power, just pull the reset switch), so the idea of an expansion RAMdisk to share data between applications is attractive. Unfortunately, there are several RAMdisk implementations, all require patching of EOS in order to function, and the patches are incompatible.

## V. Escape Through the Universal Application?

The guiding design principle in my enhanced SmartBASIC 1.x interpreter (released 1991) was that all available ADAM hardware should be supported with high-level commands, and something simple like editing an ASCII text data file should be sufficient to change hardware configurations. Since I included all the driver code as part of the application, and didn't try to patch it into EOS, I was completely successful. I was literally creating my own independent programming environment, from which escape was not necessary because everything you needed to write anything you wanted was right there `:-)` But while I have sold 50-odd copies of SmartBASIC 1.x over the last 6 years, I know of only one programmer besides me who used SmartBASIC 1.x that way, i.e., for everything.

The ultimate failure of SmartBASIC 1.x was that it could not directly run existing SmartBASIC programs which were full of `PEEK`s, `POKE`s, and `CALL`s. I had provided high-level functions which made all those `PEEK`s, `POKE`s, and `CALL`s unnecessary; yet rather than write new programs with the new tools, people wanted to run their old ones unchanged. Perhaps if I had taken a year to write a suite of applications in SmartBASIC 1.x...but I didn't, and so now I imagine that I am the only one still using SmartBASIC 1.x for anything (I do still use it for everything).

## VI. The EOS-8 Project.

Before I had come round to this rather pessimistic viewpoint, I had considered that I could extend EOS in much the same way I had extended SmartBASIC. Why not make EOS user-configurable? Design overlay areas at defined locations in EOS RAM so you could swap in the driver for whatever serial card you had, or hard drive, etc.; provide pointers to global data structures so you had the freedom to change the internal organization of EOS, yet still provide user access to global data; protect the video routines from reentrancy during interrupts by building a deferral queue; use the ADAMnet device interface to non-ADAMnet devices by emulation. After ADAMcon 04, I started work on this project, which I called EOS-8 (since the last official version of EOS was EOS-7). This was ostensibly to be a group effort involving me, Chris Braymen, Guy Cousineau, and Bruce Walters, with the programmer's forum on Mark

Gordon's Micro Innovations BBS to be our means of communication. Other than getting a few comments from the others, I did all the work. (No flames intended.)

The EOS-8 project achieved some interesting proofs of concept. The ADAMnet emulator idea (with its logical-to-physical device mapping table) was quite workable, and it survives today in ADAMserve EOS. Using my own commented disassemblies (from 1988, before I knew that there was a broader ADAM community) and the source listing from the ADAM Technical Manual I bought at ADAMcon 04, I regenerated a commented, assemblable source listing of EOS-5, then optimized away all the junk and spaghetti and waste; the freed-up space was filled with new EOS-8 features. One version mapped an Orphanware serial board to the prototype ADAMnet serial device, and a PIA2 printer card to the prototype ADAMnet parallel device; when ADAMcalc was booted under this EOS, the icons for the prototypes showed up on the device detect screen--I was probably the first person outside of Lazer Microsystems to see them actually work--and I had 3 printers to print from instead of just the ADAM printer. SmartBASIC 2.0 also could talk to the serial/parallel card as `IN #4/PR #4` and `PR #2`; under EOS-8, I could use a serial terminal in place of keyboard and screen. Yet another EOS-8 version had drivers for both the MFM and IDE hard drives loaded simultaneously; I could read/write both hard drives and copy between them. These were exciting times for me, because I felt like some kind of software archaeologist, finding fossils of things which actually did exist once, and would again if awakened in the right way.

But the EOS-8 project could not achieve its intended goal of being a universal, all-in-one, support-everything-out-of-the-box operating system, because of one fundamental limitation: it had to fit into the 8K of EOS RAM beginning at address E000 hex. EOS provides no top-of-memory function (i.e., the address of the highest free memory address available to a user program without colliding with the operating system). Thus, all of Coleco's EOS software assumed that E000 hex was the top of memory, because that's where EOS-5 started. All the extra functionality I wanted to put into EOS-8, *working* functionality as demonstrated in tests, could not fit into that 8K limit, no matter how much I squeezed and optimized. If I spilled below E000 hex, my code would get clobbered by the applications. And in practical terms, there was no hope of disassembling every single Coleco program to figure out its memory map and change it (either by patching--very lucky--or by wholesale rewriting/reassembly--a huge job with no source).

A possible solution to the space problem would be to write a bank- switched EOS; that is, use XRAM as additional storage and either swap needed routines in and out of intrinsic RAM, or else transfer control directly to code located in XRAM. The difficulties with this solution include:

1. applications (like SmartWriter and File Manager) which usurp XRAM for their own use, regardless of what's there already;
2. interrupts (especially non-maskable) during XRAM bank switching; and
3. determining the current memory configuration (including application XRAM usage) so that the correct configuration can be restored at exit.

(2) and (3) are especially problematic because the Video Display Processor (VDP) registers and all the memory map control ports are write-only; you can't read them back to find out what their

current value is. Thus, unless the program saves the contents of all these write-only registers somewhere for its own reference, there's no way to tell what they are; and if the bank of RAM containing the copies gets switched out, there's no way to read it. The only foolproof way would be to add hardware (latch chips on every write-only control port) to allow reading back of current VDP and memory map status. Of course, anything which requires new ADAM hardware can never have widespread use; so again, what's the point? This last objection can be overcome if I switch to Marcel de Kogel's ADAM emulator; I can modify the emulator code to save any I/O port data I want, without having to actually build the hardware. But I'm not quite ready to discard the physical ADAM in favor of a virtual version.

## VII. Future Directions for ADAM Operating Systems.

I think that the most promising avenue for ADAM operating system advancement lies in the adaptation of CP/M 3.0. Now that Caldera, Inc. are giving away the source to both CP/M 2.2 and 3.0, it is becoming tempting for me to think about porting CP/M 3.0 to the ADAM. Why 3.0 instead of 2.2? For starters, 3.0 supports lots of the command shell enhancements that the TDOS users are accustomed to, as well as time/date stamping of files. The biggest attraction for me, however, is that CP/M 3.0 was designed to accommodate a bank-switched operating system. Some of the crufty optimizations of TDOS (in order to maximize the Transient Program Area, TPA, effectively the memory used by programs) could go away if some of the operating system could be moved to XRAM. I have not yet obtained the CP/M 3.0 source; but I'd like to, soon.

As for EOS, I don't think that much can be done beyond what already has been implemented as part of the HARDDISK shell and its ADAMserve variant. There are just too many existing "must-have" programs which are not maintainable/modifiable in their current state (such as PowerPaint and all the patches-to-SmartBASIC-type "applications"), and the requirements of backward compatibility are too extreme.

I would, of course, like to hear other opinions on this topic. My views are just that, mine, and not necessarily correct or the definitive word on the subject.

See you next week!

*Rich*

# This Week With My Coleco ADAM 9708.24

by Richard F. Drushel

## Administrivia.

Hello, everyone! After a long hiatus (since January 1997, I think), I am finally able to resume writing these weekly articles about what I've been doing with my Coleco ADAM computer. My topic buffer is fuller than ever; my personal buffer is also fuller than ever, but I'll try to set aside enough time each week to keep these articles coming out. The general reaction to the *TWWMCA* series has been, "We don't always understand what you're talking about, but we like having the articles around to read; we've missed them this spring and summer." I can't promise that you'll understand them any better, but I do hope to have them around.

Since tomorrow begins the first day of classes at Case Western Reserve University in Cleveland, Ohio (where I'm teaching 5 days a week this year), and I'm still trying to get a few last-minute things ready, this article will be somewhat short. Future articles will be longer, I'm sure :-)

## ADAMcon 09: The Ninth International Coleco ADAM Computer Convention.

ADAMcon 09 was held 14-17 August 1997 in Grand Rapids, Michigan, hosted by Bob and Judy Slopsema, representing the Southern Michigan ADAM Users Group (SMAUG), which has no non-Slopsema membership, I believe :-) I heard Bob conclude that there were 33 official delegates, and a few more walk-ins and non-delegate spouses. Though I don't have rigorous historical attendance figures at hand, my impresssion is that this was the best showing since ADAMcon 06 in Sarasota, Florida (October 6-9, 1994). As an aside, I think it would be nice for someone to collect and publish the complete attendance lists for all the ADAMcons; then we could put an end to the bickering I've heard between various past ADAMcon chairs about who had what attendance.

Until the end of July, I wasn't sure whether or not I'd be able to attend ADAMcon 09 at all. Some emergency travel expenses, as well as the need to save money for a new car, were significant financial constraints. Fortunately, I was able to do some consulting work in July which provided the necessary funds. As it turned out, my going was what enabled 2 other people to go--Jean Davies and Pat Williams from our Cleveland B.A.S.I.C. users group. Jean and Pat had originally planned on going together in Jean's van, sharing the driving (it's about 5 hours from Cleveland to Grand Rapids). Unfortunately, a few weeks before the convention, Pat broke her right foot and was unable to drive; and Jean felt that 5 hours of driving was too much for her to handle alone. Since now I was able to go, we all decided to travel together in my minivan (which we did, with me doing all the driving).

Dale Wick (dalew@truespectra.com) has already written a nice objective summary of all the sessions at ADAMcon 09 (posted to the Coleco ADAM Mailing List), so you might want to check that out to get an overview of the convention. I'm just going to write about a few personal things of the sort that don't make it into the "official" histories.

All throughout ADAMcon 09 I was *tired*. My brain just wouldn't wake up. Part of it was the killer work schedule I had in the weeks before the convention, but it was aggravated by the weather and the hotel air conditioning system. The weather outside was coldish and rainy, but inside the hotel it was warmish and humid. The air conditioner kept my room cool temperature-wise, but it didn't dehumidify the air, so it still felt stuffy and warmish. Put 30 people and 12 heat-generating ADAMs into one session suite and it was really stuporous. This isn't an especial knock at the hotel; I think every hotel air conditioning system is the same way. In any case, I felt myself dozing off during sessions, much to my embarrassment.

I was also extremely frustrated that I couldn't clear out my head enough to do any serious programming. I had intended to finally put the user configuration module into the PC-based server side of my ADAMserve serially-linked device driver system (currently the hardware configuration is static, compiled-in). Aside from about 30 minutes of clear thinking, in which I laid out a configuration datafile format and put the necessary hooks into the server code, I just couldn't concentrate enough to write any code. This was to the great disappointment of several people, I'm sure. This is the first ADAMcon of the ones I've been to (04 through 09) that I didn't get any useful programming done, and I'm not happy about it.

I gave 2 sessions at ADAMcon 09. The first session was on Friday night, talking about ADAMserve and some of the inside history and artifacts I had obtained from Randy Hyde (founder and president of Lazer Microsystems, the company which wrote SmartBASIC, ADAMcalc, the the ADAM port of CP/M 2.2) and Joel Lagerquist (a programmer at Lazer Microsystems, who worked on the above three programs). For those of you who remember the *TWWMCA* articles from Fall 1996, Randy Hyde is my "mystery man". I still have to write up the text of our E-mail correspondence; it should appear in *TWWMCA* this fall. The second session was Sunday afternoon, talking about converting PowerPaint image files to RGB TIFFs, the ColecoVision and ADAM emulators by Marcel de Kogel (m.dekogel@student.utwente.nl), and my LEGO robotics course at CWRU. The latter seemed to be of greatest interest to the audience, since I had brought with me an actual autonomous LEGO robot capable of robust obstacle avoidance, and I demoed it in the hallway outside the session room. The tenuous link of my robot course to the ADAM is that the robot controller board is a 68HC11, a later relative of the 6801 microcontrollers used in ADAMnet peripheral devices; and thus, in principle, one could write suitable software for the 68HC11 board such that it could function as an ADAMnet device (meaning you could control a robot from an ADAM, a nifty thought).

The audience seemed appreciative of my sessions, but I personally was quite worried about them, because (unlike previous sessions at previous ADAMcons) I had not rigorously prepared the presentations ahead of time. I sort of just winged it. Dale Wick said that was okay, that the material lent itself to a more informal presentation (unlike assembly language programming, which needs considerable forethought to manage the complexity in a beginner setting). I, however, felt like a one-legged tightrope walker over Niagara Falls without a net. If the audience never picked up on my anxiety, though, it must mean I'm getting to be a better teacher :-)

I must comment on the food at ADAMcon 09: it was uniformly outstanding, in quality and in quantity. Anybody who went to bed hungry, especially after the sumptuous all-you-can-eat smorgasbord for brunch on Sunday, has only himself to blame. I can't speak for ADAMcons 01-03, but my vote is that 09 had the best food of any ADAMcon 04-09. And Bob Slopsema says we still came out $100 in the black on the convention...way to go, Bob!

I also have to mention the fine work done by the de facto equipment manager at ADAMcon 09, Bob's son Doug. Doug kept all the ADAMs (and some of the PCs, too) running at peak form, had any needed cable or adapter or software ready at hand, and didn't break anything that I saw :-) At the banquet Sunday, I asked that he be recognized with a round of applause for his efforts, and I repeat that request here. Nice job, Doug!

Jean, Pat, and I left for Cleveland Monday morning about 7:30 AM. We made it back safely and in time for me to see my wife before she had to go to work that afternoon. My "travelling box" (which has attended every ADAMcon since 05, the "hard" one to get to in Salt Lake City, Utah) is still sitting in my basement with ADAMs unpacked. Hopefully this week I'll be able to get my ADAMs set up again (we'll see how busy classes are this week).

See you again next week!

*Rich*

---

# This Week With My Coleco ADAM 9702.02

by Richard F. Drushel

[Happy Groundhog Day to everyone! I haven't heard whether or not Puxatawney Phil saw his shadow or not this morning, but here's hoping he didn't--meaning the end of winter, according to folklore.]

## I. The last of the ColecoVision color palette discussion.

Although I have heard through the grapevine that some of my previous articles dealing with the TMS9918/28/29 VDP color palettes were causing MEGO Syndrome (My Eyes Glaze Over) among my readers, I still have one last bit of color palette discussion that I want to empty out of my buffer. So, to those of you who are MEGO-sensitive regarding this topic, please bear with me (or else skip ahead to the next section of the article).

As you may recall, I set out to determine RGB color triplets for the ColecoVision/ADAM VDP colors based upon technical information provided in TI's *VDP Data Manual*. I did this because I was dissatisfied with the colors used in Marcel de Kogel's COLEMDOS.EXE ColecoVision emulator (which was derived from an MSX2 emulator by Marat Fayzullin). After lots of calculations (see *TWWMCA* 9611.18 for details), I arrived at a color palette which I felt accurately represented what I saw on a TV set or NTSC color monitor. Here's what I got, compared to what Marat and Marcel used in their emulators:

| | | RFD | | | MF & MdK | | |
|---|---|---|---|---|---|---|---|
| | | 8-bit value | | | 8-bit value | | |
| hex | TMS9118 color | R | G | B | R | G | B |
| 0 | Transparent | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | Black | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | Medium Green | 71 | 183 | 59 | 32 | 192 | 32 |
| 3 | Light Green | 124 | 207 | 111 | 96 | 224 | 96 |
| 4 | Dark Blue | 93 | 78 | 255 | 32 | 32 | 224 |
| 5 | Light Blue | 128 | 114 | 255 | 64 | 96 | 224 |
| 6 | Dark Red | 182 | 98 | 71 | 160 | 32 | 32 |
| 7 | Cyan | 93 | 200 | 237 | 64 | 192 | 224 |
| 8 | Medium Red | 215 | 107 | 72 | 224 | 32 | 32 |
| 9 | Light Red | 251 | 143 | 108 | 224 | 96 | 96 |
| A | Dark Yellow | 195 | 205 | 65 | 192 | 192 | 32 |
| B | Light Yellow | 211 | 218 | 118 | 192 | 192 | 128 |
| C | Dark Green | 62 | 159 | 47 | 32 | 128 | 32 |
| D | Magenta | 182 | 100 | 199 | 192 | 64 | 160 |
| E | Grey | 204 | 204 | 204 | 160 | 160 | 160 |
| F | White | 255 | 255 | 255 | 224 | 224 | 224 |

Marat saw a repost of my derivations on one of the USENET newsgroups (alt.folklore.computers), and insisted the RGB triplets he used in his color palette for the MSX2 emulators (which has a TMS9938 VDP, which has a backwards-compatibility mode with the TMS9918/28/29 VDP), was the actual RGB palette used by the hardware (determined either by some 9938 function call or from the *9938 Data Manual*, he didn't say which). Marcel agreed with both of us: Marat was using the hardware palette, but mine looked just like what he saw on his composite color monitor.

Marcel told me that in AdamEm, his combined ADAM/ColecoVision emulator, he has included both palettes: mine is the default, but the "hardware" one is available as a startup option. How the same underlying "hardware" RGB palette can give rise to two totally different appearances on different display monitors (composite color versus VGA) must be due to the video modulator circuitry in composite monitors.

In any case, I'm glad that I worked through the math and derived the composite-color version of the palette. It's the one that I'm going to use in any image-conversion utility programs that I might develop (such as the one to convert PowerPaint 10K binary files to RGB TIFFs).

## II. Future directions for *TWWMCA*.

I still have a couple of old topics to clear out of my buffer, not the least of which is my "Mystery Man" interview. (I've even located a second "Mystery Man", and may have some information about who currently owns the rights to some of the ADAM software.) But after that, and after my review of the Telegames DYNA System (a ColecoVision-compatible game console), I will be free to take *TWWMCA* in any of a number of possible directions. Here are some options:

1. continue with the current porpourri, whatever happens to strike my fancy.
2. pick some topic of general interest, and systematically explore it (and eventually beat it to death). Such topics include: the internal workings of ADAMserve, or of the ColecoVision OS7 operating system, or a complete analysis of a ColecoVision game (like Pitfall).
3. pick some topic of interest to *me*, whether or not it's of interest to anybody else, and systematically explore it. Such topics include: trying to figure out what the Memory I/O Controller chip on the ADAM lower circuit board actually does (it interfaces the ADAMnet master 6801 and the Z80, manages memory bank switching, and controls dynamic RAM refresh, and probably more things), building an ADAMnet serial/parallel board that has working bidirectional serial I/O (the prototype serial/parallel can only send, it can't receive), or building an ADAMnet tape drive which can format new tapes (I believe it's possible).
4. use *TWWMCA* as a forum to publish important ADAM technical documents in electronic form, such as the *ColecoVision Programmers Manual* (including the *Sound Users Manual* and the *Graphics Users Manual)* and the *ADAM Technical Manual* set. Instead of writing new text for *TWWMCA*, I would be retyping existing documents from my collection. This would also be a place to document technical specs for various 3rd-party hardware (Eve, Orphanware, Micro Innovations, etc.) which are known (to me, anyway) only by disassembly.
5. anything else which I haven't thought of yet.

So, you, Gentle Readers, can help me choose the future direction of *TWWMCA*. I enjoy writing these articles; I would enjoy any of the things (1)-(5) that I suggested above. As I stated publicly at ADAMcon VIII, however, I want to use my time and efforts to the maximum benefit of the ADAM community. Since, by and large, my articles have not had the (intended) effect of generating conversation on the Coleco ADAM Mailing List, I am beginning to wonder if *TWWMCA* as I've been writing it is the right way to go.

Let's talk about this on the Mailing List (those of you who are subscribed to it, anyway; those of you who read *TWWMCA* through the reposts to alt.folklore.computers, comp.sys.misc, and comp.os.misc, you can send me E-mail).

See you next week!

*Rich*

---

# This Week With My Coleco ADAM 9701.27

by Richard F. Drushel

[Administrative note: this article is appearing a week late because of the observance of my wife's birthday on 26 January and family illness this week. Thanks for bearing with me. The regularly-scheduled article for 2 February is appearing on time as a separate article.]

## I. Low-memory code structure and code fossils in Pitfall.

I mentioned in my article on disassembling the ColecoVision game Pitfall by Activision (*TWWMCA* 9701.19) that, in addition to an unused sound, I discovered several complete code subroutines which were not used. I'm presenting them here with some comments. They are general-purpose routines which are called through the 1-byte `RST xxH` subroutine mechanism (designed so that frequently-used subroutines could be called using only one byte of machine code rather than the three required by `CALL nnnn`.

Since `RST xxH` causes a `CALL` to address `xxH`, and these addresses 08H-38H lie in the ColecoVision OS7 ROM, there are `JP nnnn` statements at each vector to defined locations in cartridge ROM address space. These defined locations are organized as a jump table (i.e., a series of consecutive `JP nnnn` statements). It is the responsibility of the cartridge programmer to make sure that each jump table entry points to his own routines. If a particular `RST xxH` is not used by the game, the jump table entry for that `RST` must point to a `RET`.

Although Pitfall(tm) makes no use of `RST xxH` commands, I presume that other Activision games do; I haven't looked, though.

Here follows the entire low memory setup of the Pitfall(tm) game cartridge, taken from my commented disassembly PITFALL.ASM. Some of the comments have been improved from the version I originally put up download as PITFALL.ZIP; I have now put up a fixed version, PITFALL2.ZIP, which supersedes the former. I indicate the division between the code that is common to (i.e., *required* by) all ColecoVision game cartridges, and the Pitfall-specific code. Symbols that begin with `MY_` and disassembler-generated symbols of the form `J$nnnn` or `D.nnnn` are Pitfall-specific; all the rest are OS7 public globals (see OS7SYM.ASM).

```
;**********************************************
;*                                            *
;*   start of Pitfall(tm) game cartridge ROM  *
;*                                            *
;*     the initial pointers and jump table    *
;*    are required for every game cartridge   *
;*                                            *
;**********************************************

CARTRIDGE:
        DB      55H,0AAH        ;magic identifier for "test" cartridge
                                ;(i.e., don't use ColecoVision boot screen)
```

```
LOCAL_SPR_TBL:
        DW      MY_LOCAL_SPR_TBL   ;copy of sprite name table (mirror of
VRAM)
SPRITE_ORDER:
        DW      MY_SPRITE_ORDER    ;sprite order table
WORK_BUFFER:
        DW      MY_WORK_BUFFER     ;scratch RAM for OS-7 function calls
CONTROLLER_MAP:
        DW      MY_CONTROLLER_MAP  ;data area to store state of game
controller
START_GAME:
        DW      MY_START_GAME      ;first executable code of the game
;       -----------------

;***********************************************
;*                                             *
;*   low-memory RST, INT, and NMI jump table   *
;*                                             *
;*      the OS-7 ROM has JPs to this code      *
;*                                             *
;***********************************************

RST_8H_RAM:
        JP      MY_RST_8H_RAM     ;for RST 08H
;
;       -----------------
RST_10H_RAM:
        JP      MY_RST_10H_RAM    ;for RST 10H
;
;       -----------------
RST_18H_RAM:
        JP      MY_RST_18H_RAM    ;for RST 18H
;
;       -----------------
RST_20H_RAM:
        JP      MY_RST_20H_RAM    ;for RST 20H
;
;       -----------------
RST_28H_RAM:
        JP      MY_RST_28H_RAM    ;for RST 28H
;
;       -----------------
RST_30H_RAM:
        JP      MY_RST_30H_RAM    ;for RST 30H
;
;       -----------------
IRQ_INT_VECT:
        JP      MY_IRQ_INT_VECT   ;for RST 38H or INT interrupt
;
;       -----------------
NMI_INT_VECT:
        JP      MY_NMI_VECT       ;for NMI interrupt
;
;       -----------------
GAME_NAME:

;This is where the title string is stored for cartridges
```

```
;which use the ColecoVision boot screen (magic number
;AA55 hex at addresses 0-1).  Pitfall(tm) doesn't need a
;title string, since it's a "test" cartridge (55AA).  Thus,
;the game specific code begins here.

;************************************
;*                                  *
;*    begin cartridge-specific code   *
;*                                  *
;************************************

MY_RST_8H_RAM:

;not used in Pitfall(tm).  Routine to pause for a certain
;number of interrupt cycles.  On entry, the stack contains
;a pointer to a count (nominally the return address of the
;CALL or RST, but in this case the data byte is in-line
;with the code, immediately after the CALL or RST.  This
;count is fetched, and the CPU put into the HALT state
;for the specified number of counts.  HALT disables the
;CPU until an external interrupt (INT or NMI) occurs;
;execution then begins at the first statement after the
;HALT (which here is a DJNZ loop).  On exit, return is to
;the address immediately after the in-line count byte.

;This code, if useful at all on the ColecoVision/ADAM, can
;only be used with INT interrupts.  The TMS9928 VDP, which
;generates NMIs, must receive a handshake after each NMI in
;order to keep them coming (namely, a read of VDP register 8).
;This code doesn't do the handshake after the HALT.  INT
;interrupts, however, will keep coming as long as they have
;been enabled with EI.

;calling code has the form:

;       RST     08H
;       DB      COUNT
;       [next opcode]

        EX      (SP),HL     ;get HL=return address off stack
        PUSH    BC          ;save a scratch register
        LD      B,(HL)      ;get count byte
        INC     HL          ;point ahead one to real return address
J$8028:
        HALT                ;halt the CPU and wait for an interrupt
        DJNZ    J$8028      ;interrupt occurred, so go back and wait again
;
        POP     BC          ;all done, restore BC
        EX      (SP),HL     ;put return address back on stack
        RET                 ;and exit
;
;       -----------------
MY_RST_10H_RAM:

;not used in Pitfall(tm).  Routine to get a vector from
;a table and JP there.  On entry, HL=address of table,
;and A=index of vector.  On exit, a JP is performed to
```

```
;the desired vector.

        ADD     A,A           ;A=A*2 for word offset
        ADD     A,L           ;add it to lobyte of table base
        LD      L,A           ;and back into L; was there a carry?
        JR      NC,J$8034     ;no
;
        INC     H             ;yes, so we have to INC H
J$8034:
        LD      A,(HL)        ;get lobyte of vector
        INC     HL            ;point to hibyte
        LD      H,(HL)        ;get hibyte of vector
        LD      L,A           ;now HL=vector
        JP      (HL)          ;go there
;
;       -----------------
MY_RST_18H_RAM:

;not used in Pitfall(tm).  Looks like a custom routine to
;return a random number.  The built-in RAND_GEN routine
;doesn't give the greatest sequence of random numbers.  On
;entry, A=count of how many times to call RAND_GEN, and
;HL=seed value.  On exit, A=random number.

        PUSH    HL
        PUSH    DE
        PUSH    BC
        PUSH    AF            ;save count
        LD      B,A           ;count in B
J$803E:
        CALL    RAND_GEN      ;OS7 call, return A=random number
;
        DJNZ    J$803E        ;keep going until done
;
        POP     BC            ;restore count
        XOR     A             ;A=0
        LD      D,A           ;DE=0
        LD      E,A
        EX      DE,HL         ;HL=0, DE=entry seed
J$8048:
        ADD     HL,DE         ;add seed
        ADC     A,00H         ;and add the carry (if set)
        DJNZ    J$8048        ;keep going until done
;
        POP     BC
        POP     DE
        POP     HL
        RET
;
;       -----------------
MY_RST_20H_RAM:

;not used in Pitfall(tm).  Multiplies HL*A*2, returning
;product in HL.  Could be another way to calculate an offset
;into a vector table.

        PUSH    DE
        LD      H,00H
        LD      D,H
        LD      E,H
        EX      DE,HL       ;DE=entry L, HL=0
J$8057:
        SLA     E
        RL      D           ;DE=DE*2
        SRL     A           ;A=A/2; did DE*2 give a carry?
        JR      NC,J$8060   ;no
;
        ADD     HL,DE       ;yes, so HL=HL+DE
J$8060:
        OR      A           ;is A=0 yet?
        JR      NZ,J$8057   ;no, so keep going
;
        POP     DE
        RET
;
;       -----------------
MY_RST_28H_RAM:

;not implemented in Pitfall(tm).

        RET
;
;       -----------------
MY_IRQ_INT_VECT:

;This is for the maskable interrupt routine (for the spinners, trackball,
;and driving wheel).  Not implemented in Pitfall(tm).

        RET
;
;       -----------------
MY_RST_30H_RAM:

;not implemented in Pitfall(tm).

        RET
;
;       -----------------
MY_NMI_VECT:

;NMI routine.  Does a JP to address stored at D.7000.

        PUSH    HL              ;save HL
        LD      HL,(D.7000)     ;get the vector
        EX      (SP),HL         ;put it on the stack and restore HL
        RET                     ;go to the vector address
;
;       -----------------
START_MAIN:

;starts the MAIN program.

[code omitted]
```

## II. The NMI routine: the heart of a ColecoVision game.

You may be somewhat disturbed by the lack of comments regarding the NMI routine at `MY_NMI_VECTOR`. That's because what happens during the NMI routine is extremely complicated (for Pitfall and for any other ColecoVision game). Stated briefly, the "main" program of a ColecoVision game is an idle loop,

```
IDLE_LOOP:
        JR IDLE_LOOP
```

waiting endlessly for some kind of interrupt (maskable or non-maskable) to occur. The game itself is played only after an interrupt occurs. In the case of Pitfall, which uses the 60 Hz (or 50 Hz on European systems) NMI as the system clock, everything happens as part of the `MY_NMI_VECT` routine. There is additional complexity to insure that the proper handshaking is performed with the TMS9928/9929 VDP (which generates the NMI) so that the NMI is re-enabled before the `RETN` command (return from NMI) is executed.

The NMI handler in Pitfall is quite interesting, but too complex for me to cover adequately at this time. I intend to make it a topic of discussion in a future issue of *TWWMCA*; stay tuned.

See you next week!

*Rich*

---

# This Week With My Coleco ADAM 9701.19

by Richard F. Drushel

## I. Administrivia.

Hello, patient readers, my end-of-semester/Christmas/start-of-semester hiatus is over. With this issue of *TWWMCA*, I'm resuming my weekly schedule. The next possible hiatus is at the end of April/beginning of May, which is the end of the Spring 1997 semester here at CWRU.

I've been busy with my ADAM over Christmas break, so I have a full buffer of topics for these articles. I hope you enjoy them over the coming weeks. And finally, a belated "Happy New Year" to everyone!

## II. Adding "immortality" to a ColecoVision game: Pitfall.

My kids spent a fair amount of their Christmas break playing ColecoVision games on their new gift from Santa Claus, a Telegames DYNA ColecoVision-compatible video game system. (I'm planning a formal review of the DYNA system for a future issue of *TWWMCA*, so stay tuned.) One of the games they had trouble with was Pitfall by Activision: they kept getting killed and using up their 3 lives less than 2 minutes into the game. While all games have a learning curve, Pitfall's seemed steeper than most, and none of my kids seemed to get any better with repeated playing.

So, I decided it would be an interesting project to try to hack Pitfall so that, no matter how many times you got killed, you never ran out of lives--"immortality", if you will. Of course, since the game itself is in ROM, unless I wanted to burn new ROMs for the cartridge, my hacked version would have to run either from disk on an ADAM, or else under Marcel de Kogel's ColecoVision or ADAM emulators. For ease of debugging, I decided to use the COLEMDOS emulator. (I know that the COLEMDOS project has been absorbed into Marcel's ADAM emulator, but I haven't gotten around to obtaining the latter and COLEMDOS works fine for my purposes.)

## III. Plan of attack.

To decide how I wanted to go about hacking into Pitfall, I had to consider a number of issues:

1. At some point, I was going to have to disassemble the Pitfall 16K binary to see how the code worked. But that 16K binary would be a jumble of code, image data, and sound data. Sorting those out would be a formidable task, and *then* the code would have to be traced out to see what it did. Brute-force begin-at-the-beginning-and-go-until-the-end would be very hard and take a long time, so I would have to try to limit the search space somehow. I didn't want to figure out more about the game than I needed to in order to add the immortality hack.

2. I needed to understand the behavior of the game (from a player's perspective), to know what actions caused you to die, to know what graphics, music, etc. occurred when you died, and when you ran out of lives. These would be things to look for in the disassembled code.

   In Pitfall, many things cause you to die: falling into any of the lakes (lake with alligators, lake with Tarzan rope, or appearing/disappearing lake), or touching various hazards (snake, fire, scorpion). When you die, the theme from the TV cop show "Dragnet" is played (DUMMM DA-DUM DUM DUMMM), and either (a) a new life is granted (if you have any left), or (b) the game ends (if that was your last life).

3. How would a typical programmer be likely to keep track of lives in a game? If I could guess the method, I could look for it in the disassembled code.

Putting (2) and (3) together, I hypothesized that the relevant code would have to look something like this:

```
MY_START_GAME:

    ...

    LD   A,3                    ;initial number of lives
    LD   (NUMBER_OF_LIVES),A    ;save it in workspace RAM

    ...

YOU_DIED:

    CALL PLAY_DEATH_SONG        ;DUMM DA-DUM DUM DUMMM

    ...

    LD   HL,NUMBER_OF_LIVES     ;point to number of lives
    DEC  (HL)                   ;one less life; any left?
    RET  NZ                     ;yes, so keep playing
    JP   GAME_OVER              ;no, so game over
```

This is not much to go on. This kind of code is commonplace, and the helpful labels given here are not going to show up in a disassembly listing:

```
J$81A0:

    ...

    LD   A,03H                  ;A=3
    LD  (I.721C),A              ;save something

    ...

C$A42F:

    CALL C$AC02                 ;do something
    ...

    LD   HL,I.721C              ;point to something
    DEC  (HL)                   ;one less, is it zero yet?
    RET  NZ                     ;no, so return
    JP   J$95A9                 ;yes, so go somewhere
```

Also, there's no guarantee that it was even written in this straightforward a manner. For example, all the RAM workspace data areas could be initialized by copying a single block of data from ROM to RAM, rather than setting the RAM variables directly. In this case, the variable you care about is buried anonymously in the block of data:

```
    LD   HL,I.B07E              ;ROM source
    LD   DE,I.7200              ;RAM destination
    LD   BC,27H                 ;39 bytes to copy
    LDIR                        ;copy them

    ...

I.B07E:

    DB  00H,00H,05H,0AH,00H,14H,29H,00H
    DB  08H,08H,00H,32H,11H,09H,00H,05H
    DB  0FFH,0EH,0FFH,0EFH,66H,55H,54H,09H
    DB  0C8H,0C8H,05H,00H,03H,01H,08H,10H
;                         ^^^
;                         ^^^ ROM $B09A --> RAM $721C
;
    DB  28H,33H,00H,0F8H,0F6H,00H,01H

I.B0A5:
```

Considering all these possibilities, however, it seemed certain that, whether they were contained in the same subroutine or not, there had to be some linkage between playing the "Dragnet" death theme and decrementing the count of lives remaining. Thus, my final plan of attack was:

1. Disassemble Pitfall enough so that the code and data areas are resolved. I don't care if I don't know what the data areas mean, just so they are all separate from the code areas.
2. Find the subroutine which plays the "Dragnet" theme.
3. Find all code which references the call to play the "Dragnet" theme, and look for anything which looks like it's decrementing a life count.
4. Patch the Pitfall binary to bypass the life count decrement, and play the patched version to see if "immortality" has been achieved.

## IV. Disassembling Pitfall.

Early in my ADAM career, I wrote a SmartBASIC program to disassemble code in memory, with the memory contents also displayed in hexadecimal, called UNASMHEX. I modified this to write the disassembly to disk. (By reading said disk in an IBM-PC disk drive, I could move the listing to a PC for further editing using better editors than SmartWriter; but that's a different story.) I further ported the program to Microsoft BASIC and modified it to disassemble from a file; with the last version of this program, UNASM2F9.BAS, I could disassemble to my heart's content, so long as I had managed to move the appropriate binary from the ADAM to the PC (usually by a null-modem transfer). This was all I had when I did my original disassemblies of EOS-5, EOS-7, SmartBASIC 1.0, SmartBASIC 2.0, PowerPaint, and ADAMlink IVa; and for simple disassembly to a list file, it was sufficient. Here's a sample (with some comments added after the fact). Note that everything not in the dump field is in decimal.

```
**********************************************************************
First 2 blocks of ADAMcalc binary file BASICPGM.
(named BASICPGM so that SmartBASIC can't INIT it by accident)
disassembled by Richard F. Drushel  1 August 1993
**********************************************************************
This is loaded by the cold boot/block 0 routine.
On entry, A=default device #, and addresses 0-25 contain the directory
entry for the ADAMcalc binary program BASICPGM.

12288 F3       DI                  ;disable maskable interrupts
12289 3100E0   LD SP,57344         ;set top of stack to bottom of EOS
12292 F5       PUSH AF             ;save default device # (from boot)
12293 219C37   LD HL,14236         ;address of data
12296 ED5B0D00 LD DE,(13)          ;get start block from directory entry
12300 13       INC DE              ;skip to 3rd block
12301 13       INC DE              ;(first 2 are already loaded)
12302 010000   LD BC,0             ;
12305 CDA2FC   CALL 64674          ;start read 1 block
12308 CD0632   CALL 12806          ;VDP register setup using data table.
**********************************************************************
VDP register setup data.  For awful, EOS-bypassing VRAM write routine.

12311 07       ;7
12312 07       ;6
12313 7F       ;5
12314 03       ;4
12315 FF       ;3
12316 06       ;2
12317 C2       ;1
12318 02       ;0
**********************************************************************
12319 CD1432   CALL 12820          ;some VRAM writing
12322 AF       XOR A               ;A=0
12323 110018   LD DE,6144          ;# bytes to fill
12326 210000   LD HL,0             ;start address
12329 CD26FD   CALL 64806          ;fill VRAM
```

Subsequently, as I have had need not only to disassemble a program to figure out how it works, but also to make substantial changes to it, I have needed a disassembler which can produce a listing which is assembler-ready. That is, I can use the disassembly output as the direct basis for program changes, and run it through an assembler to generate a new binary. Nothing in my UNASM family of disassemblers was designed to do this.

Fortunately, Kenneth Gielow's Z80DIS version 2.2, which runs under CP/M (and TDOS, too), disassembles a binary file to re-assemblable ASM source. Z80DIS is an "intelligent" disassembler: it can often recognize the breaks between code area and data areas (especially if the data consist of ASCII text strings), and you make it repeatedly analyze the binary, each cycle taking into account what it learned previously. Z80DIS creates what it calls a "break table", which is a sequential listing of start addresses for the different areas of the program. These breaks can be due to instructions (I), data bytes (B), data words (W), address tables (T), ASCII text (A), or empty space (S). Here is the equivalent break table for the fragment of ADAMcalc shown above:

```
    I3000
    B3017
    I301F
```

Note that the addresses in the break table are given in hexadecimal.

Typically, you let Z80DIS automatically make its best guess at where the breaks should be, cycling over and over until there is no further "improvement". At the end of each cycle, Z80DIS lists a table of "anomalies", which are places that don't seem to make sense based upon the current break table. When the anomaly table reaches a minimum, you exit the analysis mode and let Z80DIS write out the regenerated source ASM file. Now you have to look at the regenerated source and decide if you agree with Z80DIS's choices of breaks. If you don't, you have to write a new break table, feed that to Z80DIS, and let it write out the source ASM again.

The large advantage of Z80DIS is that its regenerated source file contains only as many reference labels as are necessary. In the UNASM output above, note that every statement is prefaced with a label (the RAM address in decimal). I could easily hack UNASM to omit the hex dump field and write `A12288` (a label) instead of `12288` (a number), and the output would be nominally acceptable to an assembler. The problem is, the vast majority of the labels would be unnecessary, because there are no `CALL`s or `JP`s to those addresses. Assemblers have a finite space to generate a symbol table, and giving every line a symbol would cause the assembler to rapidly run out of space. In the above code fragment, Z80DIS would generate *no* start-of-line labels, because *none* of it is the target of a `CALL` or `JP` anywhere else in the program (as I recall). (The data area is referenced by a "trick": the `CALL 12806` pushes the return address 12311 on the stack, which the 12806 routine immediately retrieves and uses as a pointer to the data; at the end, the correct return address of 12319 is put back on the stack before the `RET`, so execution resumes directly after the data table.)

The regenerated source file can be many hundreds of kilobytes in size. I believe that the popular CP/M editor VDE can't deal with a single file greater than 64K or so. In any case, if you're running Z80DIS under native CP/M, you'll need an editor capable of handling huge ASM files. I know that WordStar 3.3 can do it; if there are other such editors, especially if they are freeware/shareware, I'd like to hear about them.

Because I don't have WordStar 3.3 for CP/M and because I've never done much with ADAM CP/M 2.2 or TDOS (even though I have a 10MB partition for the latter on my ADAM hard drive), I prefer to use the text editors I have available on my PC system under MS-DOS, and to run Z80DIS and other CP/M programs under a DOS-based emulator. While Simeon Cran's MYZ80 is currently the "best" CP/M emulator around, I don't use it because it can't read/write files in the DOS filesystem--you have to create a virtual disk file, and use a special utility to import/export DOS files to/from the virtual disk. Instead, I use a more primitive and clunky emulator, Joan Riff's Z80MU. Despite many defects compared to MYZ80, Z80MU can directly read/write DOS files. Thus, simply keeping a /Z80 subdirectory on my PC hard drive, along with Z80MU.EXE and any CP/M .COM files I want, lets me easily switch back and forth between CP/M (for disassembly and assembly) and DOS (for editing).

- z80dis22.lbr, Kenneth Gielow's Z80 disassembler (CP/M)
- z80mub2b.zip, Joan Riff's CP/M emulator (DOS)
- myz80111.zip, Simeon Cran's CP/M emulator (DOS)

My final disassembly procedure was:

1. First-order disassembly of PITFALL.ROM using Z80DIS under Z80MU emulator. Automatic mode until anomaly table did not change; then write out PITFALL.ASM.
2. Examine PITFALL.ASM with WordPerfect 5.0 (DOS). Revise break table as seems appropriate.
3. Second-order disassembly, using revised break table (no automatic mode).
4. Repeat (2) and (3) until happy.
5. Reassemble PITFALL.ASM with SLR's Z80ASM+ (a commercial assembler) to create PITFALL0.ROM.
6. Use MS-DOS FC.EXE (File Compare) to make sure that PITFALL0.ROM is identical to PITFALL.ROM. This verifies that I got back what I started with.

   On a couple of occasions, further inspection of the code much later, during the cleanup process (see below), demanded some additional improvements to the break table (such as finding address tables buried inside byte data tables), which required re-disassembly to generate an improved PITFALL.ASM. This allowed the final verification of totally resolved disassembly:

7. Delete a couple bytes of unused, padded-out space from the beginning of the code, put it back at the end to keep the total bytecount the same, reassemble the new code, and play the game to make sure that everything still works. If there was a vector table still hidden in the data areas, moving the code will cause it not to point at the correct place any more, and the game will crash or have scrambled graphics, etc.

## V. OS7 global symbol substitution.

Now that I had Pitfall completely disassembled, I had to start to figure out how it worked--at least enough to find the "Dragnet" death music player. But before actually beginning to interpret the code, there was one final cleanup step to perform: substituting the official OS7 global symbol names wherever possible in PITFALL.ASM.

In the code samples given above in III, you can see what a difference meaningful symbol names bring to otherwise meaningless code. Except in the highly unlikely case that Activision completely bypassed the operating system and wrote directly to the hardware, Pitfall must be full of OS7 function calls and references to OS7 global data structures. There is no way that Z80DIS could know what these symbols were when it did its disassembly, so I would have to put them in manually, using the search/replace functions of my editor (WordPerfect 5.0 for DOS) and a listing of the OS7 global symbols. This list, OS7SYM.ASM, which I derived from the ColecoVision Programmer's Manual, is available for download.

- os7sym.asm, ColecoVision OS7 global symbols

The immediate effect of substituting the global symbol names was to make the code somewhat readable, even knowing nothing else about Pitfall. If you tracked down anything which had to do with sprites, for example, eventually you could find out which of the data areas had sprite pattern generators (i.e., the bitmaps for individual sprites) and colors. Similarly, if you tracked down anything which had to do with sounds (such as `PLAY_IT` or `PLAY_SONGS`), you could find out where the music and sound effects were stored. You can guess what I did next...

## VI. Finding the "Dragnet" theme player.

I searched through PITFALL.ASM for any references to the following OS7 function calls dealing with sounds:

| | |
|---|---|
| `PLAY_IT` | enable the playing of the song in B |
| `PLAY_SONGS` | actually write data to sound chip (low level routine) |
| `SOUND_INIT` | initialize the sound data areas |
| `SOUND_MAN` | manage note pointers for songs |
| `TURN_OFF_SOUND` | turns off 3 voices and noise channel |

I thus discovered a set of routines with the following form:

```
        LD   B,number                   ;song number in B
        CALL PLAY_IT                    ;play the song
        JR   common_exit                ;exit
```

These had to be routines to play individual songs and sound effects. Working backwards, I found that these were linked by a common routine, which I have named `PLAY_SOUND_IN_A`:

```
    PLAY_SOUND_IN_A:

    ;Play sound (or take sound action) specified by A.  On entry,
    ;A=index of sound action to perform (0=turn off sound, 1=init
    ;sounds, 2-10 are sounds to play).  On exit, the desired action
    ;is performed.  Aborts if A>10, which is weird because the
    ;SOUND_VECTOR_TABLE has an entry for A=11.

        CP   0BH                        ;is it less than 11?
        RET  NC                        ;no, so abort
;
        PUSH BC
        PUSH DE
        PUSH HL
        PUSH IX
        PUSH IY
        LD   E,A
        LD   D,00H                     ;DE=index
        LD   HL,SOUND_VECTOR_TABLE     ;point to vector table
        ADD  HL,DE                     ;offset twice
        ADD  HL,DE
        LD   A,(HL)                    ;get vector
        INC  HL
        LD   H,(HL)
        LD   L,A                       ;into HL
        JP   (HL)                      ;go there
;
;       -----------------
SOUND_VECTOR_TABLE:
        DW   MY_TURN_OFF_SOUNDS        ;turns off all sounds
        DW   MY_INIT_SOUNDS            ;initializes the sound manager
        DW   PLAY_SOUND_01H            ;
        DW   PLAY_SOUND_02H            ;
        DW   PLAY_SOUND_03H            ;
        DW   PLAY_SOUND_04H            ;
        DW   PLAY_SOUND_05H            ;
        DW   PLAY_SOUNDS_06H_07H       ;
        DW   PLAY_SOUNDS_08H_09H       ;
        DW   PLAY_SOUND_0CH            ;
        DW   PLAY_SOUND_0DH            ;
        DW   PLAY_SOUNDS_0AH_0BH       ;
;       -----------------
;
P_S_I_A_COMMON:
        POP  IY
        POP  IX
        POP  HL
        POP  DE
        POP  BC
        RET
```

All but the first 2 vectors in `SOUND_VECTOR_TABLE` point to routines with the `LD B,number / CALL PLAY_IT / JR common_exit` form (3 of them play 2 songs instead of 1). The first two vectors are for specialized functions: turning off all sounds, and initializing the sound manager. Note the interesting bit of orphaned code: there are 12 vectors in the table, but the routine only allows access to 11. We'll come back to what `PLAY_SOUNDS_0AH_0BH` actually does later.

The code segments of Pitfall are filled with

```
          LD   A,number
          CALL PLAY_SOUND_IN_A
```

and clearly some value of A will result in the playing of the "Dragnet" death theme. But which one? There's no easy way to tell.

Let's try find where the sound data is actually stored. The routine `MY_INIT_SOUNDS` gives us a clue:

```
    MY_INIT_SOUNDS:
        LD   HL,MY_SOUND_TABLE
        LD   B,04H
        CALL SOUND_INIT
    ;
        JR   P_S_I_A_COMMON

    MY_SOUND_TABLE:
        DW   DATA_SOUND_01H,I.72E0   ;
        DW   DATA_SOUND_02H,I.72E0   ;
        DW   DATA_SOUND_03H,I.72E0   ;
        DW   DATA_SOUND_04H,I.72E0   ;
        DW   DATA_SOUND_05H,I.72E0   ;
        DW   DATA_SOUND_06H,I.72E0   ;
        DW   DATA_SOUND_07H,I.72EA   ;
        DW   DATA_SOUND_08H,I$72F4   ;
        DW   DATA_SOUND_09H,I.72EA   ;
        DW   DATA_SOUND_0AH,I.72E0   ;
        DW   DATA_SOUND_0BH,I.72EA   ;
        DW   DATA_SOUND_0CH,I.72E0   ;
        DW   DATA_SOUND_0DH,I.72E0   ;
    ;       -----------------
```

Each 2-word record in `MY_SOUND_TABLE` contains a pointer to the actual data for a sound, and a pointer to a RAM workspace area used by the OS7 sound routines. Here are typical examples of the raw sound data:

```
    DATA_SOUND_01H:

        DB   0C1H,0FFH,41H,0AH,11H,0E0H,30H,0C3H
        DB   90H,0D1H,08H,15H,56H,0F9H,15H,10H

    DATA_SOUND_02H:

        DB   03H,00H,66H,04H,00H,00H,75H,11H
        DB   10H
```

Again, there's no way to tell from simple inspection of the raw sound data what kind of sound it's supposed to make.

How can you find out what any of these sounds sound like? One possibility would be to write a test program in Z80 assembler, to run on an ADAM under SmartBASIC (by `POKE`ing it into RAM, or maybe `BLOAD`ing it from disk and then `CALL`ing it). This was too much work for me...so I thought of an alternative:

If I changed all but the first two vectors in the `SOUND_VECTOR_TABLE` to the same vector, then played the altered game, then *every* *sound* would be the same sound. If I did this systematically, first all one vector, then all the next vector, etc., eventually I would know what vector actually made what sound. And knowing *that* information would tell me which A argument to `PLAY_SOUND_IN_A` corresponded to the "Dragnet" death theme. And *that* would narrow down very precisely which code I'd have to study to find the life-decrementing routine.

I used the Norton Utilities Disk Editor to make the changes to a backup copy of PITFALL.ROM. `SOUND_VECTOR_TABLE` begins at byte offset 36F4 hex, 14068 decimal (0-based). Remember that Z80 addresses are stored in lobyte/hibyte form, so the bytes will be swapped compared to their appearance in the table below. The table lists the absolute value of the vectors in hex, and gives (in the comment field) the corresponding value of A for the `PLAY_SOUND_IN_A` call, and a description of the sound produced by each vector.

```
    SOUND_VECTOR_TABLE:
        DW   B716                 ; 0  turns off all sounds
        DW   B71B                 ; 1  initializes the sound manager
    ;start changing vectors here
        DW   B759                 ; 2  [bwip!]            jumping
        DW   B760                 ; 3  [pfft!]            running
        DW   B767                 ; 4  [bwiyiyiyip!]      hitting a barrel
        DW   B76E                 ; 5  [eeeuuuuublump!]   falling down a
shaft
        DW   B775                 ; 6  ["Dragnet" theme]  death
        DW   B77C                 ; 7  ["Tarzan" yell]    swinging on a vine
        DW   B788                 ; 8  ["Charge!" theme]  getting treasure
        DW   B795                 ; 9  [eeuuwop!]         releasing a vine
        DW   B79D                 ;10  [tic!]             running into a wall
        DW   B7A5                 ;11  [dah-di]           gallop; orphaned
```

Note the "orphaned" sound A=11, which is not used in the actual game. It is a sort of low-pitched, fast gallopping sound. Either it was abandoned, or else it was a temporary sound for use until the other running sound was implemented; there is no way to know from the available evidence.

I encourage the adventurous to repeat my sound-substitution experiment: it's spooky to play the game and have every action result in the same sound over and over and over...

# VII. Finding the life counter.

On the basis of my sound experiments, I now knew that any code which said

```
        LD      A,06H
        CALL    PLAY_SOUND_IN_A
```

was playing the "Dragnet" death theme. Using the search functions of my editor, I located three subroutines which played the "Dragnet" death theme, and a fourth which called one of the first three. I named them (in order of appearance in the code) `DEATH_1`, `DEATH_2`, `DEATH_4`, and `DEATH_3`, respectively. I'm not sure what `DEATH_2` does; I think it handles the case of death when the 20:00 minute game timer runs out. `DEATH_4` is called by `DEATH_1` and `DEATH_3`; and looking at `DEATH_4`, it's what we were hoping to find:

```
    DEATH_4:

        LD   A,(LIVES_LEFT)     ;number of lives remaining
        SUB  01H                ;one less life
        LD   (LIVES_LEFT),A     ;save back new total
        RET  NC                 ;still lives remaining, so keep playing
    ;
        LD   HL,D.72AC          ;sorry, no lives left
        LD   (HL),01H           ;I
        LD   IX,I.7236          ;  don't
        LD   HL,C.A2FC          ;        know
        LD   (IX+18),L          ;              what
        LD   (IX+19),H          ;                   this
        RES  5,(IX)             ;                        stuff
        RES  0,(IX)             ;                              does
        RES  0,(IX+6)           ;                                   .
        LD   A,03H              ;                                    .
        LD   (D.71A0),A         ;                                     .
        LD   A,SOUND_05H        ;["Dragnet" theme] death
        CALL PLAY_SOUND_IN_A    ;play it
    ;
        POP  DE                 ;get rid of old return address
        RET                     ;and exit one level higher
```

This code isn't exactly the same as that which I hypothesized above that I'd find, but it does what I expected it would. On entry, you have just died. The life counter in `(LIVES_LEFT)` is decremented and the new value is saved. If any lives are left, you return to the caller, who plays the "Dragnet" death theme and returns to the game. Otherwise, the "Dragnet" death theme is played here, then the old return address is popped off the stack and you exit one level higher (presumably to a routine which waits for you to press a keypad button to begin a new game).

Since the "you-still-have-lives-left" exit condition is `NC` after the `SUB`, the entry value of `(LIVES_LEFT)` which causes the `NC` to fail is *0*, not 1. (1-1=0 no carry flag; 0-1=255 carry set.) This suggests that the initial value of `(LIVES_LEFT)` at the start of the game is *2*, not 3 (as it would be if `NZ` were the test condition):

```
    death      NZ test       NC test
     1       3-1=2   NZ    2-1=1   NC      ;2 lives left
     2       2-1=1   NZ    1-1=0   NC      ;1 life left
     3       1-1=0    Z    0-1=255  C      ;game over
```

Sure enough, in one of the subroutines called during game initialization, is found the following code:

```
        LD      A,02H            ;initial number of extra lives
        LD      (LIVES_LEFT),A   ;save it
```

## VIII. Final patching.

Based upon the code in `DEATH_4`, the best way to patch Pitfall to add "immortality" is to change `SUB 01H` to `SUB 00H`. If `(LIVES_LEFT)` is never decremented, the game will never end; after each death, you'll get another life. This patch has the additional merit of being only one byte, an 01H to an 00H.

I again used the Norton Utilities Disk Editor to make this patch (byte offset 09F8 hex, 2552 decimal (0-based)). I'm happy to state that this patched version of Pitfall does indeed render you "immortal"--when you die, another man just drops down from the upper left corner of the screen, and off you go.

When I told Herman Mason (aa337@po.cwru.edu) about this entire patching endeavor, his only comment was, "You should have done it for Burger Time!" Fortunately for Herman, the hacking principles described here are applicable to Burger Time or any other ColecoVision game.

## IX. Other goodies in Pitfall.

There are a number of interesting fossils and features in Pitfall. I have noted some of them in my final disassembly listing, PITFALL.ASM. I encourage the interested to take a look. I may talk about some of these discoveries in a future *TWWMCA* article.

- pitfall2.zip, regenerated source for Pitfall (revised)

See you next week!

*Rich*

# This Week With My Coleco ADAM 9612.10

by Richard F. Drushel

Hello gang, sorry to report that, due to it being Finals Week here at CWRU, and due to the end-of-semester LEGO Robot Egg Hunt, I don't have a decent *TWWMCA* for this week :-(

I have just one more article related to VDP color palettes, and then I can leave that topic for a while. If Dave Sands' last message was any indicator, my last few articles are causing MEGO Syndrome (My Eyes Glaze Over).

I really want to get my Mystery Man interview finished before Christmas...here's hoping that Santa Claus gives me an early gift of enough free time to do it.

I recently heard from Murray McCullough (irene@bora.dacom.co.kr), who's in Korea for the next year. Sadly, he won't be back in time to attend ADAMcon 09 :-( We missed Murray at ADAMcon VIII, because he had an unbroken attendance string since ADAMcon 01. Now, I believe that only Dale Wick and PJ Herrington are left who've been to all the ADAMcons.

B.A.S.I.C.'s Christmas Dinner meeting will be this Sunday, at a steakhouse called Broglio's. We expect Herman Mason, George Koczwara, Pat Williams, Jean Davies, and me in attendance.

See you next week!

*Rich*

# This Week With My Coleco ADAM 9612.01

by Richard F. Drushel

## I. More on RGB Values of the TMS9xx8 VDP Palette.

Two weeks ago (*TWWMCA* 9611.18) I presented an extensive derivation of the RGB triplet values corresponding to the color palette of the TMS9118 VDP (which is closely related to the TMS9928 VDP in the ColecoVision and ADAM). I've had some interesting responses from Marcel de Kogel (author of the ColEm and ADAMEm emulators under MS-DOS) and Marat Fayzullin (author of the original MSX emulator, upon which the ColEm and ADAMEm emulators were originally based). Marcel was happy with my RGB palette; but Marat claims that his palette (which I felt was not a good match to what I saw on my monitor screen with a real ADAM) was derived directly from the default RGB palette loaded into the TMS9938 VDP when starting in backward- compatible 9918/28 mode. I asked Marat if his palette came directly from a Texas Instruments *Data Manual* or from some other source; I haven't heard a reply yet.

In the meantime, I decided that I would follow up on one of my own suggestions about how to determine the RGB palette: create a color test pattern, use a video frame grabber to create an RGB TIFF image of the display, and then analyze the RGB triplets. I used a RasterOps 24STV 24-bit frame grabber on a PowerMac 7100 host computer, with an R80 ADAM's NTSC composite video output plugged into it. I wrote a simple SmartBASIC 1.x program to draw 15 256 x 12 horizontal bars in HGR2 mode, after setting the border color to white. (You can use regular SmartBASIC 1.0, but you have to use `POKE`s to set the colors; the `COLOR(x)=y` syntax of SB1.x is much cleaner, so I used the latter.) Here's the program (indenting used to promote clarity; it will disappear if you load this into SB1.x):

```
 1 REM colorbar / Created Saturday 30 November 1996 by Richard F. Drushel
 2 REM makes a test pattern of horizontal color bars
 3 REM Note:  SmartBASIC 1.x only!
 9 IF VER(0)<20 THEN TEXT: BEEP: PRINT "Requires SmartBASIC 1.x!": NEW
10 COLOR(2)=15:                 REM set border to WHITE
20 HGR2:                        REM 256 x 192 graphics mode
30 FOR n=0 to 15:               REM step through each color
40     COLOR(1)=n:              REM set HCOLOR color
50     FOR j=0 to 11            REM each bar is 12 lines high
60          y=(n*12)+j:         REM find our current scan line
70          HPLOT 0,y TO 255,y: REM draw the colored line
80     NEXT j
90 NEXT n
```

The software which comes with the RasterOps board (MediaGrabber 3.0) allows the current video to be grabbed and saved in a variety of image formats, including 24-bit color (RGB) TIFF. I grabbed the colorbar screen, moved the image to an MS-DOS machine, looked at it with a hex editor (Norton Utilities), and wrote a QuickBASIC 4.5 program to create an RGB frequency distribution of the pixels in each color bar. (There is always some noise in a video signal, meaning that each "color" would have some slight variations pixel-to- pixel.) Fortunately, for most of the colors, there was a vastly-predominating RGB triplet combination, so I could just select the triplets of greatest frequency as representative of each "color". The file ADAM.TIF is available for download.

adam.tif, framegrab of R80 ADAM (TMS9928A VDP) color bars

The following 2 tables give the RGB triplets from the colorbar framegrab, both as 8-bit values and percentages. For comparison, the values I calculated previously for the TMS9918 VDP are also presented. The file TMS9928A.TIF is a representation of the framegrab-derived palette.

tms9118.tif, TMS9118 palette, calculated from *Data Manual* parameters

tms9928a.tif, framegrab palette using only most-frequent RGB triplets (as tabulated above)

| | | 8-bit value | | | | | |
|---|---|---|---|---|---|---|---|
| | | TMS9928A (frame grab) | | | TMS9118 (calculated) | | |
| hex | color | R | G | B | R | G | B |
| 1 | Black | 0 | 16 | 0 | 0 | 0 | 0 |
| 2 | Medium Green | 16 | 191 | 0 | 71 | 183 | 59 |
| 3 | Light Green | 64 | 192 | 32 | 124 | 207 | 111 |
| 4 | Dark Blue | 64 | 63 | 255 | 93 | 78 | 255 |
| 5 | Light Blue | 96 | 95 | 255 | 128 | 114 | 255 |
| 6 | Dark Red | 160 | 80 | 16 | 182 | 98 | 71 |
| 7 | Cyan | 47 | 192 | 208 | 93 | 200 | 237 |
| 8 | Medium Red | 207 | 79 | 15 | 215 | 107 | 72 |
| 9 | Light Red | 224 | 111 | 32 | 251 | 143 | 108 |
| A | Dark Yellow | 160 | 175 | 0 | 195 | 205 | 65 |
| B | Light Yellow | 175 | 175 | 31 | 211 | 218 | 118 |
| C | Dark Green | 16 | 144 | 0 | 62 | 159 | 47 |
| D | Magenta | 159 | 79 | 160 | 182 | 100 | 199 |
| E | Grey | 159 | 160 | 143 | 204 | 204 | 204 |
| F | White | 191 | 207 | 175 | 255 | 255 | 255 |

| | | percent maximum value | | | | | |
|---|---|---|---|---|---|---|---|
| | | TMS9928A (frame grab) | | | TMS9118 (calculated) | | |
| hex | color | R | G | B | R | G | B |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | Black | 0.00 | 6.27 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | Medium Green | 6.27 | 74.90 | 0.00 | 27.56 | 71.65 | 22.86 |
| 3 | Light Green | 25.10 | 75.29 | 12.55 | 48.39 | 80.97 | 43.24 |
| 4 | Dark Blue | 25.10 | 24.71 | 100.00 | 36.52 | 30.37 | 100.00 |
| 5 | Light Blue | 37.65 | 37.25 | 100.00 | 49.90 | 44.42 | 100.00 |
| 6 | Dark Red | 62.75 | 31.37 | 6.27 | 71.12 | 38.22 | 27.76 |
| 7 | Cyan | 18.43 | 75.29 | 81.57 | 36.19 | 77.99 | 92.75 |
| 8 | Medium Red | 81.18 | 30.98 | 5.88 | 84.06 | 41.70 | 28.23 |
| 9 | Light Red | 87.84 | 43.53 | 12.55 | 98.06 | 55.70 | 42.23 |
| A | Dark Yellow | 62.75 | 68.63 | 0.00 | 76.16 | 80.23 | 25.50 |
| B | Light Yellow | 68.63 | 68.63 | 12.16 | 82.27 | 85.18 | 45.95 |
| C | Dark Green | 6.27 | 56.47 | 0.00 | 24.31 | 62.27 | 18.31 |
| D | Magenta | 62.35 | 30.98 | 62.75 | 71.15 | 39.06 | 77.92 |
| E | Grey | 62.35 | 62.75 | 56.08 | 80.00 | 80.00 | 80.00 |
| F | White | 74.90 | 81.18 | 68.63 | 100.00 | 100.00 | 100.00 |

There are clearly differences between the RGB values from the framegrab and the calculated values for the TMS9118 VDP; but what these mean isn't clear just from the numbers. Comparison of the framegrab image ADAM.TIF with the image of the calculated palette (TMS9118.TIF, also available for download) shows that the framegrab is "warmer": the colors are more saturated. Also, the "white" of the framegrab isn't really very white, but is instead rather grey. The framegrab "black" also has a distinct greenish cast to it.

To determine the exact nature of the color differences between the framegrab palette and the calculated TMS9118 palette, the RGB triplets were converted into luminance (Y), I, and Q signals. The I and Q signals were then used to calculate the magnitude of the chrominance vector (C) and the phase angle relative to the B-Y axis (P). The reader is referred to *TWWMCA* 9611.18 for discussion of color television signals and the necessary mathematics. The results are presented in the following table:

| | | TMS9928A (frame grab) | | | | |
|---|---|---|---|---|---|---|
| hex | color | Y | I | Q | C | P |
| 1 | Black | 0.04 | -0.02 | -0.03 | 0.04 | 241 |
| 2 | Medium Green | 0.46 | -0.17 | -0.38 | 0.41 | 238 |
| 3 | Light Green | 0.53 | -0.10 | -0.30 | 0.32 | 232 |
| 4 | Dark Blue | 0.33 | -0.24 | 0.23 | 0.33 | 347 |
| 5 | Light Blue | 0.44 | -0.20 | 0.20 | 0.28 | 348 |
| 6 | Dark Red | 0.38 | 0.27 | -0.01 | 0.27 | 126 |
| 7 | Cyan | 0.59 | -0.36 | -0.10 | 0.37 | 288 |
| 8 | Medium Red | 0.43 | 0.38 | 0.03 | 0.38 | 119 |
| 9 | Light Red | 0.53 | 0.37 | -0.00 | 0.37 | 123 |
| A | Dark Yellow | 0.59 | 0.18 | -0.23 | 0.29 | 174 |
| B | Light Yellow | 0.62 | 0.18 | -0.18 | 0.25 | 167 |
| C | Dark Green | 0.35 | -0.12 | -0.28 | 0.31 | 236 |
| D | Magenta | 0.44 | 0.09 | 0.16 | 0.19 | 61 |
| E | Grey | 0.62 | 0.02 | -0.02 | 0.03 | 172 |
| F | White | 0.78 | 0.00 | -0.05 | 0.05 | 210 |

For comparison, here is the equivalent data for the TMS9118 VDP, derived from values tabluated in the *Data Manual:*

| hex | color | TMS9118 (from Data Manual) | | | | |
|---|---|---|---|---|---|---|
| | | Y | I | Q | C | P |
| 1 | Black | 0.00 | 0.00 | 0.00 | 0.00 | --- |
| 2 | Medium Green | 0.53 | -0.11 | -0.24 | 0.27 | 237 |
| 3 | Light Green | 0.67 | -0.08 | -0.19 | 0.20 | 235 |
| 4 | Dark Blue | 0.40 | -0.19 | 0.23 | 0.30 | 354 |
| 5 | Light Blue | 0.53 | -0.17 | 0.21 | 0.27 | 354 |
| 6 | Dark Red | 0.47 | 0.23 | 0.04 | 0.23 | 114 |
| 7 | Cyan | 0.67 | -0.30 | -0.04 | 0.30 | 295 |
| 8 | Medium Red | 0.53 | 0.30 | 0.05 | 0.30 | 114 |
| 9 | Light Red | 0.67 | 0.30 | 0.05 | 0.30 | 114 |
| A | Dark Yellow | 0.73 | 0.15 | -0.18 | 0.23 | 173 |
| B | Light Yellow | 0.80 | 0.11 | -0.13 | 0.17 | 173 |
| C | Dark Green | 0.46 | -0.09 | -0.22 | 0.23 | 235 |
| D | Magenta | 0.53 | 0.07 | 0.19 | 0.20 | 53 |
| E | Grey | 0.80 | 0.00 | 0.00 | 0.00 | --- |
| F | White | 1.00 | 0.00 | 0.00 | 0.00 | --- |

The remarkable result of this comparison is that the phase angles of all the colors are virtually the same between the framegrab and the calculated 9118 values. The main differences are in

1. the luminance values, which are low in the framegrab, and
2. the chrominance values, which are greater in the framegrab.

The low luminance is the cause of the greyed-out "white" color in the framegrab, while the larger chrominances are responsible for the framegrab colors looking "warmer" and less pastel.

For fun, I rescaled the RGB values (by dividing by 0.78) so that the maximum luminance was 1.00 volts. Here's the result:

| hex | color | TMS9928A (frame grab) | | | | |
|---|---|---|---|---|---|---|
| | | Y/0.78 | I' | Q' | C' | P' |
| 1 | Black | 0.05 | -0.02 | -0.04 | 0.05 | 241 |
| 2 | Medium Green | 0.59 | -0.22 | -0.48 | 0.53 | 238 |
| 3 | Light Green | 0.68 | -0.13 | -0.38 | 0.41 | 232 |
| 4 | Dark Blue | 0.42 | -0.31 | 0.30 | 0.43 | 347 |
| 5 | Light Blue | 0.57 | -0.25 | 0.25 | 0.36 | 348 |
| 6 | Dark Red | 0.49 | 0.34 | -0.02 | 0.34 | 126 |
| 7 | Cyan | 0.76 | -0.46 | -0.13 | 0.48 | 288 |
| 8 | Medium Red | 0.55 | 0.49 | 0.04 | 0.49 | 119 |
| 9 | Light Red | 0.68 | 0.47 | -0.00 | 0.47 | 123 |
| A | Dark Yellow | 0.76 | 0.24 | -0.29 | 0.37 | 174 |
| B | Light Yellow | 0.80 | 0.23 | -0.22 | 0.32 | 167 |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| C | Dark Green | 0.45 | -0.15 | -0.36 | 0.39 | 236 |
| D | Magenta | 0.56 | 0.11 | 0.21 | 0.24 | 61 |
| E | Grey | 0.79 | 0.02 | -0.03 | 0.04 | 172 |
| F | White | 1.00 | 0.00 | -0.07 | 0.07 | 210 |

Two more remarkable results to note:

1. the luminances for all colors now matches that given in the *Data Manual;* and
2. the phase angles are unchanged.

If I understood this topic better, maybe I wouldn't be (or shouldn't be) surprised by this; but I find it quite unexpected nonetheless.

One possible explanation for the Y and C differences is that the brightness and hue controls for the RasterOps framegrabber board were not set at the correct defaults, and that the brightness was too low and/or the hue was too high. The tint setting, however, must be okay, because all the phase angles are correct. (Remember, "brightness" knob changes luminance, "hue" knob changes magnitude of chrominance vector, and "tint" knob changes phase angle of chrominance vector.) Another possibility is that the video circuitry of the ADAM I used is out of adjustment.

Speaking of settings, one setting I did play with in the MediaGrabber 3.0 software was the "gamma" setting. Gamma is a non-linear parameter introduced into the transmitted R,G,B color signals to compensate for non-linear behavior of the receiver. The transmitted signal is raised to the gamma power; the receiver takes the gammath root of the signal to restore the original. A gamma of 1.0 is for a perfectly linear response. A typical gamma for a television set is 2.2. When I changed the gamma of the RasterOps framegrabber board to 2.2, and looked at the live ADAM colorbar video display, I saw a color palette which looked (IMHO) very similar to Marat Fayzullin's palette for the TMS9938. Hmm...is it possible that the RGB values Marat used were gamma-corrected, and needed to be uncorrected? See the similarity yourself in the two files GAMMA22.TIF (the gamma-corrected view of my ADAM colorbar display) and MARAT.TIF (which is Marat's palette as extracted from COLEMDOS screen shots in Windows BMP format).

gamma22.tif, same screen as ADAM.TIF but gamma-corrected (gamma=2.2)

marat.tif, Marat Fayzullin's TMS9938 palette, used by Marcel de Kogel in the COLEMDOS ColecoVision emulator

Note that the order of color bars in the two TIFs is not the same; you should be able to figure out the colors, though. In principle, I should be able to gamma-uncorrect MARAT.TIF and get back something which looks like my TMS9118 palette. Something to try this week...

## II. Exploring ADAMnet: How the Z80 CPU Shares RAM With ADAMnet.

Chris Braymen recently cleared up a long-standing mystery (to me, anyhow) about why the EOS `_READ_BLOCK` function needed to make 2 ADAMnet read requests to read from a block device, while `_WRITE_BLOCK` needs only 1 ADAMnet write request. While checking out his answer by looking over the source code for the ADAMnet master 6801 microcontroller, I ended up lingering over the routines which actually read and write to the Z80 address space. (The ADAMnet Device Control Blocks, or DCBs, are memory-mapped I/O areas for communicating with each ADAMnet device.) I've spent a few hours with my ADAM schematics, my *6801 Reference Manual* from Motorola, and the master 6801 source, to try to understand this part of the ADAM hardware. It's complicated by the fact that there's a large logic gate array chip, the Memory I/O Controller (MIOC), sitting between the Z80 CPU and the master 6801, and made worse by the fact that the MIOC is a totally-undocumented black box.

I think I can make some headway, for my own amusement, but I don't have a "final" story worked out yet; so I'll be satisfied with just letting everybody know that I'm poking around with it. If I get it worked out (and if Chris Braymen doesn't have it worked out already), I'll write it up. Who knows, maybe enough can be gleaned to replace the MIOC with discrete logic chips someday; right now, the MIOC is the only non-replaceable chip on the ADAM motherboard.

## III. A Holiday Wish.

I hope all the Americans got lots of turkey this Thanksgiving weekend. To you Canadians, visit your closest ADAMite across the border and get some leftovers `:-)`

See you next week!

*Rich*

---

# This Week With My Coleco ADAM 9611.25

by Richard F. Drushel

It's the end-of-semester crunch time here at CWRU, and all the students in my Autonomous Robotics course are spending long hours in the lab...along with me. I've still managed to do some ADAM-related stuff, but I don't have any complete stories for you, like I did last week with the VDP color palette discussion. Thus, this is more of a progress report than a coherent article, so please bear with me.

## I. Marcel de Kogel's ADAM Emulator.

I finally traded some E-mail with Marcel de Kogel (m.dekogel@student.utwente.nl) about his ADAM emulator. He was looking for some information about ADAM floppy disk formats, and I sent him the x86 assembler source for the floppy disk I/O module of my ADAMserve file server. He said that the information was useful, but that it was so hardware-specific that it would likely not be useful for his emulator (i.e., not every computer has compatible floppy drive hardware). He also had a couple questions relating to ADAMnet I/O; I referred him to Chris Braymen (70057.2035@compuserve.com), but Marcel said he had already contacted him but was still waiting for a reply. Finally, Marcel was looking for information about (non-ADAMnet) serial and parallel ports, as well as the ADAMlink modem. I have all this information in various forms, I just need to put it together into a brief, coherent package. Marcel, bear with me `:-)`

As for the emulator itself, I don't have it yet, and have been avoiding it because it's too interesting and I have other stuff that people have been clamoring for me to do. Everything I've heard about it, though, has been very positive.

## II. COLEMDOS.EXE: Corrected Attribution.

I want to correct my previous misattributions of the MS-DOS emulator of the ColecoVision, COLEMDOS.EXE. In previous issues of *TWWMCA*, I have referred to it as Marat Fayzullin's (fms@freeflight.com) emulator. I have since learned that Marcel de Kogel is the true author of COLEMDOS.EXE; he started with Marat's MSX emulator code, but in the process of porting it to MS-DOS, he rewrote all of Marat's code. I apologize to Marcel de Kogel for this misattribution.

## III. More on TMS9xx8 VDP Colors.

While FTPing my floppy disk I/O code, Marcel also happened to grab the 24-bit color TMS9118.TIF file I put up last week.

TMS9118 VDP color palette, recalculated from TI *Data Manual* values

He commented that, like me, he initially thought that the colors were too washed-out, but when viewed side-by-side with his MSX's TMS9938 VDP, he was impressed by the color match. Marcel feels that the greens are possibly too bright. Possible sources of the difference are:

1. NTSC versus PAL video;
2. composite video versus color difference signals; and
3. errors in my RGB derivation.

I have access to a 24-bit framegrabber in my lab at CWRU. Perhaps I could bring in an ADAM, make a color bar test pattern display, and grab the frames to look at the RGB values.

## IV. ADAMnet Block Read Kinks.

The ADAMnet question Marcel has for Chris Braymen pertains to block reads from ADAMnet block devices. The code for the EOS `_READ_BLOCK` function makes two calls to the `_READ_1_BLOCK`; the corresponding `_WRITE_BLOCK` function makes only one call to `_WRITE_1_BLOCK`. Upon first inspection, this behavior sounds weird. Upon much further reflection, it's *still* weird. It doesn't make sense...but the second call to `_READ_1_BLOCK` is *necessary* for the block to be transferred. At ADAMcon 05 in 1993, Chris Braymen and I were playing with his ADAMnet 256K static RAMdisk, and we commented out the second `CALL _READ_1_BLOCK` (and associated code) with `NOP`s. The activity light would blink on the RAMdisk, but the requested data would not appear in Z80 address space. A second manual call to `_READ_1_BLOCK`, however, and the data appeared in Z80 address space.

Thanks to Chris Braymen, I have a copy of the original Coleco source code to the ADAMnet master 6801, the tape drive, and the daisy-wheel printer. I know that once I tried to work through the master 6801 source to identify the source of this two-read requirement, but didn't find anything. I decided to look again. As I started through the code, I discovered (or rather, was reminded of something I had forgotten) that Chris' original source listing was missing 2 pages from the master 6801 code. The missing pages weren't in a critical part of the code; in fact, there were parallel sections in the tape and printer source which seemed to duplicate what was missing.

The challenge of reconstructing the 2 missing pages of the listing was too attractive: I spent an evening doing it. Basically, the symbol table and cross-reference table at the end of the listing told me the line numbers on which certain symbols were originally defined. By analogy with the parallel sections in the tape and printer source, I was able to reconstruct probable comments; in fact, because the 2 missing pages were mostly all comments, I was able to account for every missing line number in the listing. For Chris's benefit, and for that of anybody else to whom he may have given copies of the listing, here's my reconstructed version. I'm confident that all but possibly a couple lines are exactly the way they appeared in the original. Warning: some of the lines are longer than 80 columns.

```
FILE: D_MMR:pADAMM       HEWLETT-PACKARD: D_MMR  (c) Coleco 1983 Confidential
Thu, 23 Jun 1983,  0:17     PAGE   2

LOCATION OBJECT CODE LINE     SOURCE LINE

                       26
****************************************************************
                       27 *
*
                       28 *  MODULE NAME:
*
                       29 *
*
                       30 *    D_MMR
*
                       31 *
*
                       32 *  FUNCTION(S):
*
                       33 *
*
                       34 *     1.  TO DECLARE THE DATA AREA "NIM_BLOCK."
*
                       35 *     2.  TO DECLARE THE D1_MODE_WORD.
*
                       36 *
*
                       37 *  NOTES:
*
                       38 *
*
                       39 * 1. NIM_BLOCK IS USED AS THE INTERFACE BETWEEN THE
*
                       40 *    MEDIUM ACCESS CONTROLLER AND THE RESIDENT
APPLICATION     *
                       41 *    PROGRAM.
*
                       42 *
*
                       43 * 2. THE INSTALLER IS RESPONSIBLE FOR LOCATING THIS
DATA       *
                       44 *    MODULE SO THAT THE LAST BYTE ENDS AT LOCATION
127 (DEC).  *
                       45 *
*
                       46
****************************************************************

....

FILE: D_MMR:pADAMM       HEWLETT-PACKARD: D_MMR  (c) Coleco 1983 Confidential
Thu, 23 Jun 1983,  0:17     PAGE   3

LOCATION OBJECT CODE LINE     SOURCE LINE

                       48                 GLB     D_MMR
                       49                 GLB     D1_MODE_WORD
```

```
                                         GLB     NIM_DMA_BLK
                                         GLB     CNFG_WORD
                                         GLB     A_SIGNAL
                                         GLB     A_SIG
                                         GLB     MEM_PTR
                                         GLB     MEM_PTR_HI
                                         GLB     MEM_PTR_LO
                                         GLB     MEM_LEN
                                         GLB     MEM_LEN_HI
                                         GLB     MEM_LEN_LO
                                         GLB     MEM_SIGNAL
                                         GLB     MEM_SIG
                                         GLB     EXPECTED_NODE
                                         GLB     NXT_STATE
                                         GLB     TIMEFORRESPONSE
                                         GLB     INDEXTOMEM
                                         GLB     TIME
                                         ORG     83H
       0083               D_MMR:

***************************************************************
                          *
*
                          *  DATA_WORD:
*
                          *
*
                          *    D1_MODE_WORD
*
                          *
*
                          *   FUNCTION:
*
                          *
*
                          *    CONTAINS THE STATE OF SEQUENCER PROCESSING
*
                          *
*

***************************************************************

                                         DATA
                                         ORG     90H
       0090               D1_MODE_WORD   RMB     1
```

....

Now that I've finished my detour, I hope that I can complete my original objective of going through the master 6801 code and figuring out why the two reads are necessary.

To those in the States, I wish you all a happy and safe Thanksgiving; eat lots of turkey and other good foods.

See you next week!

*Rich*

# This Week With My Coleco ADAM 9611.18

by Richard F. Drushel

## I. How to Reproduce ADAM Video Colors.

Some weeks back I described some crude methods to export PowerPaint 10K binary image files to other image formats, including greyscale TIFF. I also mentioned that I thought it would be rather simple to adapt these quick-and-dirty programs to be more general purpose, and also to allow exporting of the PowerPaint files in color. Believe it or not, I've spent about 2 weeks working on this "simple" adaptation.

The ADAM's video display processor (VDP), the Texas Instruments TMS9928, supports 15 distinct colors, plus a "transparent" color which can allow background elements to be visible through foreground elements which are on top of them (such as sprites). These colors are manipulated by the programmer according to a color number or index (0-15). A PowerPaint 10K binary file is basically a dump of the VDP memory, part of which is a table of foreground and background colors, all specified by these numbers 0-15.

Unfortunately, this is a hardware-dependent representation of the colors. A PC or Mac video system can't just load in these color numbers as-is and magically reproduce the colors you see on an ADAM. There must be some translation of the VDP color numbers to a data format which can specify the exact colors to use, regardless of what kind of display system is used to reproduce the image.

There are several color systems in use, which allow colored images to be translated to different display media. A broad division in color systems is between those used for print media (newspapers, magazines) and those used for electronic displays (televisions, monitors). Print media mix colored inks or paints to achieve a "rainbow palette", but electronic displays mix colored lights.

The most common color system used for electronic displays is called RGB color, named because it specifies colors as combinations of Red, Green, and Blue light. Look with a magnifying glass at the picture tube of a color television, and you will see individual red, green, and blue elements; at a certain distance from the screen, the individual elements merge, producing an effect in your eyes as if there were spots of unique colors, rather than separate red, green, and blue dots. Each RGB "triplet" thus represents one picture element, or pixel, on the screen. The picture tube of a color television actually has 3 electron guns inside, one for each primary color, and a television broadcast contains information to vary the intensities of each electron gun for each pixel on the screen.

You might imagine that a simple color image file might consist of nothing but a rectangular array of RGB triplets, one for each pixel in the image. That's basically the internal structure of a full-color TIFF file: 3 bytes per pixel, one byte each for the R, G, and B values, ranging 0-255. Thus, if I could determine what the RGB values for each of the 15 VDP colors were, I could use these values to convert a PowerPaint 10K binary image file into a color TIFF file. But how do you find out the RGB values for the VDP colors?

There are several possibilities:

1. hook an ADAM up to a video frame grabber, grab some colored screens, save them as RGB TIFF files, and then look at the RGB triplets with a binary editor (like Norton Utilities). The frame grabber hardware samples the analog video signals and digitizes them. One disadvantage of this is that noise in the analog signal will cause variations in the digitized values for a supposedly constant color; you'd have to average the values over large areas to get a representative sample. A more serious concern is that the composite video output of the ADAM is likely to be out-of-specification compared to ideal NTSC specifications (e.g., tint is off, brightness and contrast are too high or low), and thus all the color values so obtained would be systematically off. In the absence of any other methods, though, this would be my first choice.
2. try to match the colors by eye using an RGB color editor. Many graphics programs (such as Canvas for Macintosh and Windows) have a utility to create custom colors by specifying arbitrary RGB triplets. You could set up your ADAM next to your Mac and fiddle with the RGB controls until you had reasonable matches to the colors on the ADAM monitor. If you have good eyes, this is also feasible.
3. use the RGB triplets that the COLEMDOS.EXE ColecoVision emulator uses (never mind how Marat Fayzullin (fms@freeflight.com) determined them). I probably would have done this right away, except that I don't think his colors match the ColecoVision colors very well. In the absence of data to the contrary, I'd say that Marat made some quick-and-dirty guesses, and that his eye isn't very good `:-)` The way to get the emulator color assignments is to enable the screen capture mode, grab some game screens as Windows BMP files, then convert them into RGB TIFFs and look at the RGB triplets thus obtained.
4. use the actual RGB values designed into the VDP. Fortunately for me, my trusty *TMS9118/TMS9128/TMS9129 Data Manual* (1984) has a table which gives the factory specifications for the colors. I reproduce part of this table below:

```
5.
   =========================================================================
6.     |     |              |           TMS9118           |       TMS9128
   |
7.     |     |
   ===================================================
8.     |     |              |           |                |      |    color
   |
9.     |     |              |           |  chrominance   |      |
   difference  |
10.     |     |              |     DC    =================
   ===============
11.     | hex |    color     | luminance |   A'  | phase |   Y  | R-Y  |
   B-Y  |
12.
   =========================================================================
13.     |  0  | Transparent  |    0.00   | 0.000 |  ---  | 0.00 | 0.47 |
   0.47 |
14.     |  1  | Black        |    0.00   | 0.000 |  ---  | 0.00 | 0.47 |
   0.47 |
     |  2  | Medium Green |    0.53   | 0.267 |  237  | 0.53 | 0.07 | 0.20 |
     |  3  | Light Green  |    0.67   | 0.200 |  235  | 0.67 | 0.17 | 0.27 |
     |  4  | Dark Blue    |    0.40   | 0.300 |  354  | 0.40 | 0.40 | 1.00 |
     |  5  | Light Blue   |    0.53   | 0.267 |  354  | 0.53 | 0.43 | 0.93 |
     |  6  | Dark Red     |    0.47   | 0.233 |  114  | 0.47 | 0.83 | 0.30 |
     |  7  | Cyan         |    0.67   | 0.300 |  295  | 0.73 | 0.00 | 0.70 |
     |  8  | Medium Red   |    0.53   | 0.300 |  114  | 0.53 | 0.93 | 0.27 |
     |  9  | Light Red    |    0.67   | 0.300 |  114  | 0.67 | 0.93 | 0.27 |
     |  A  | Dark Yellow  |    0.73   | 0.233 |  173  | 0.73 | 0.57 | 0.07 |
     |  B  | Light Yellow |    0.80   | 0.167 |  173  | 0.80 | 0.57 | 0.17 |
     |  C  | Dark Green   |    0.46   | 0.233 |  235  | 0.47 | 0.13 | 0.23 |
     |  D  | Magenta      |    0.53   | 0.200 |   53  | 0.53 | 0.73 | 0.67 |
     |  E  | Grey         |    0.80   | 0.000 |  ---  | 0.80 | 0.47 | 0.47 |
     |  F  | White        |    1.00   | 0.000 |  ---  | 1.00 | 0.47 | 0.47 |
==========================================================================

     A' given as SQR( I^2 + Q^2 ), where I,Q are 33 degrees ahead of R-Y,B-Y
```

It's probably not obvious to you how this table is of any help in obtaining the RGB triplets for each of the VDP colors. It wasn't obvious to me, either--so, I decided I needed to learn about the electrical basis for color television. To that end, I borrowed two books from my Dad, and started reading:

- Slurzberg, M., Osterheld, W., and E.N. Voegtlin (1956). *Essentials of Television.* New York: McGraw-Hill. 687 pp.
- Terman, F.E., Helliwell, R.A., Petit, J.M., Watkins, D.A., and W.R. Rambo (1955). *Electronic and Radio Engineering.* New York: McGraw-Hill. 1078 pp.

While these books are old, they are elegantly written and very clear. Of course, some parts are technical and use lots of higher mathematics; but fortunately, you need only understand simple algebra and trigonometry to deal with the basics of television transmission.

I make no claims to be an electrical engineer. If I have made any errors in the ensuing discussion, I apologize in advance.

## II. Principles of Color Television.

In black-and-white (b&w) television transmission, the greyscale level for each pixel on the screen is transmitted as a voltage between 0.00 and 1.00 volts; this signal is called "luminance" and symbolized by Y. A luminance of 0.00 is black and 1.00 is white. Color television must transmit full RGB triplets for each screen pixel, but do it in such a way that (a) there is a luminance signal for b&w sets, and (b) the color signals aren't detected by a b&w set. The color television standard adopted by the NTSC in 1953 was a remarkable hack of existing b&w television, to allow b&w sets to receive color transmissions, and color sets to receive b&w transmissions, without any interference.

If b&w backward compatibility weren't an issue, a color television transmission could contain a separate channel/signal for each color (R,G,B), which would directly drive the appropriate electron gun in the picture tube. Color monitors for computers are actually constructed using this principle. "Analog RGB" monitors have 3 inputs, one for each color gun; modern VGA and SVGA monitors are variants of this. The signal inputs can also be digital, as in CGA and EGA monitors; in this case, discrete color levels are specified using multiple logic lines for each color input. Of course, neither direct-drive RGB nor digital monitors are compatible with b&w television.

Rather than transmit (R,G,B) as three separate signals, a means was devised mathematically to encode the information in only one signal, called "chrominance", and the encoded signal was shoehorned into unused bandwidth in the existing b&w television signal in such a way that b&w sets couldn't detect it. A luminance signal was generated by combining the (R,G,B) signals in proportions which reflect the sensitivity of the red, green, and blue photoreceptors of the human eye. A b&w set receiving a color transmission uses the luminance to show greyscale pixels, while ignoring the chrominance signal entirely. A color set receiving a color transmission mathematically decomposes the luminance and chrominance signals into the original (R,G,B) components, and uses those to drive the 3 electron guns in the picture tube. A color set receiving a b&w transmission has only a luminance signal to work with (chrominance signal is zero), and the luminance is decomposed into (R,G,B) values sufficient to make greyscale images.

The chrominance signal actually has some internal structure of its own. The initial encoding of the (R,G,B) signals produces Y (luminance) and two "color difference" signals, R-Y and B-Y. As their names imply, they are the difference in voltage levels between Y, R, and B. If you think of B-Y and R-Y forming a 2-dimensional coordinate system, any color can be represented as a point, whose coordinates are (B-Y,R-Y). Color points can also be represented in polar coordinate form, as a radius and an angle of rotation (r,theta).

```
               R-Y

                ^
                |       r = SQR( (B-Y)^2 + (R-Y)^2 )    ;Pythagorean theorem
                |
  (B-Y,R-Y)     |   theta = ATN( (B-Y) / (R-Y) )        ;definition of tangent
                |
       * - - - -|     B-Y = r * SIN( theta )            ;definition of sine
        \       |
       : \      |     R-Y = r * COS( theta )            ;definition of cosine
          \ r   |
       :   \   .|..
```

```
            \ . |  .
       :     \  |   .theta
              \ |   .
       :       \|   .
   <------------+-----------> B-Y
                |
                |
                |
                |
                |
                |
                v
```

The polar form is more relevant for the electrical implementation of the chrominance signal, because r can be transmitted as the signal amplitude, while theta can be transmitted as a phase shift of the carrier wave relative to an internal standard.

For reasons which aren't clear to me, however, the color difference signals R-Y and B-Y are not directly transmitted. Rather, two additional signals I and Q are defined, which are the result of rotating the (R-Y,B-Y) axis 33 degrees counter-clockwise. This is even harder to represent in ASCII art, but here goes:

```
               R-Y

                ^                           Location of saturated
                |                           colors (no white light),
    I           |           Q               relative to (B-Y) axis:
     \     90 degrees      /
      \       ..|..       /                  magenta   61 degrees
       \     .  |  .     /                   red       103 degrees
        \   . ..|.. .   /                    yellow    167 degrees
         \ . .  |  . . /                     green     241 degrees
          \ .   |   . /                      cyan      283 degrees
           \    |    /                       blue      347 degrees
            \   |   /
             \  |  /.
              \ | /  .33 degrees
               \|/   .                      I left out the chrominance
   <------------+-----------> B-Y           vector from the previous
               /|\                          diagram, because it's too
              / | \                         confusing; but if shown, it
             /  |  \                        would be pointing toward
            /   |   \                       about 9 o'clock.
           /    |    \
          /     |     \
         /      v      \
```

In the I-Q system, the value of the color or "hue" depends upon the angle of the chrominance vector, and the magnitude (length) of the chrominance vector determines the "saturation" of the colors (i.e., how pure or pastel they appear). Familiar controls on your color television set manipulate these parameters directly: The "tint" control changes the angle of *all* the chrominance signals (by phase shifting); you can make the picture purplish or greenish depending upon which direction you shift the phase. The "color level" control increases or decreases the magnitude of *all* the chrominance vectors (by differential amplification); you can make the picture gaudy-colored or completely b&w.

If you now refer back to the color table I reproduced from the VDP *Data Manual* you can see where all the entries come from. The TMS9118 VDP has an on-chip NTSC color encoder and puts out a direct composite video signal; thus, its colors are specified as luminance, chrominance, and phase angle relative to (B-Y). The TMS9128 VDP puts out luminance and color difference signals, so its colors are specified in terms of Y, R-Y, and B-Y.

I will spare you the derivation of the following formulas, but any interested reader is encouraged to give it a shot: it's simply 3 simultaneous linear equations `:-)`

The first three equations are givens:

| | |
|---|---|
| `Y = 0.30R + 0.59G + 0.11B` | RGB-to-greyscale; weighting is based upon photosensitivity of the human eye |
| `I = 0.60R - 0.28G - 0.32B` | coefficients empirically determined from color differences |
| `Q = 0.21R - 0.52G + 0.31B` | ditto |

What we want, however, is (R,G,B) in terms of (Y,I,Q):

| | |
|---|---|
| `R = Y + 0.95I + 0.62Q` | Terman *et al.* (1955) have a sign error in their derivation of this equation; I spent a week trying to reconcile data before I got fed up and re-derived it myself. |
| `G = Y - 0.28I - 0.64Q` | |
| `B = Y - 1.11I + 1.73Q` | |

## III. Calculating RGB Colors for the TMS9118 VDP.

Since the ADAM's VDP is a TMS9928, I decided I would calculate the RGB colors using the tabulated color difference and luminance data. Unfortunately, Texas Instruments (a) neglected to include the arithmetic signs of the color differences, and (b) did not provide unambiguous units for the values. "Percent of black/white voltage swing" is ambiguous because the total swing could be anywhere between 0.75 and 1.00 volts, depending upon whether colors or black/white/grey are being displayed. (0.00 is black, 1.00 is white, but no saturated color exceeds 0.75 volts.) Playing with the values and guessing at the signs did not give me any consistent results. If somebody can help me, please do; but the tabulated color difference values don't match anything given in my two reference textbooks.

I then tried to use the TMS9118 color values, even though the *Data Manual* says that the colors are slightly different between the two VDPs. Because the phase angles were given with reference to the B-Y axis, I had to subtract 33 degrees from all of them in order to get the correct hue. Using the trigonometric relations shown above in the R-Y/B-Y graph, I calculated I and Q values, and then substituted these and the Y values into the equations for R, G, and B. In a couple cases, values greater than 1.00 were produced, most likely due to accumulated roundoff errors; these were truncated at 1.00. Since most graphics programs which use RGB values take them either as bytes (0-255) or as percentages, I converted the RGB values from voltages 0.00-1.00 to the appropriate units. The results are tabulated below:

| TMS9118 color | voltage value | | | 8-bit value | | | percent max value | | |
|---|---|---|---|---|---|---|---|---|---|
| | R | G | B | R | G | B | R | G | B |
| Transparent | 0.000 | 0.000 | 0.000 | 0 | 0 | 0 | 0.00 | 0.00 | 0.00 |
| Black | 0.000 | 0.000 | 0.000 | 0 | 0 | 0 | 0.00 | 0.00 | 0.00 |
| Medium Green | 0.276 | 0.717 | 0.229 | 71 | 183 | 59 | 27.56 | 71.65 | 22.86 |
| Light Green | 0.484 | 0.810 | 0.432 | 124 | 207 | 111 | 48.39 | 80.97 | 43.24 |
| Dark Blue | 0.365 | 0.304 | 1.000 | 93 | 78 | 255 | 36.52 | 30.37 | 100.00 |
| Light Blue | 0.499 | 0.444 | 1.000 | 128 | 114 | 255 | 49.90 | 44.42 | 100.00 |
| Dark Red | 0.711 | 0.382 | 0.278 | 182 | 98 | 71 | 71.12 | 38.22 | 27.76 |
| Cyan | 0.362 | 0.780 | 0.928 | 93 | 200 | 237 | 36.19 | 77.99 | 92.75 |
| Medium Red | 0.841 | 0.417 | 0.282 | 215 | 107 | 72 | 84.06 | 41.70 | 28.23 |
| Light Red | 0.981 | 0.557 | 0.422 | 251 | 143 | 108 | 98.06 | 55.70 | 42.23 |
| Dark Yellow | 0.762 | 0.802 | 0.255 | 195 | 205 | 65 | 76.16 | 80.23 | 25.50 |
| Light Yellow | 0.823 | 0.852 | 0.460 | 211 | 218 | 118 | 82.27 | 85.18 | 45.95 |
| Dark Green | 0.243 | 0.623 | 0.183 | 62 | 159 | 47 | 24.31 | 62.27 | 18.31 |
| Magenta | 0.712 | 0.391 | 0.779 | 182 | 100 | 199 | 71.15 | 39.06 | 77.92 |
| Grey | 0.800 | 0.800 | 0.800 | 204 | 204 | 204 | 80.00 | 80.00 | 80.00 |
| White | 1.000 | 1.000 | 1.000 | 255 | 255 | 255 | 100.00 | 100.00 | 100.00 |

| 9118 color | Y | Yrgb | R-Y | B-Y | I | Q | C | Piq |
|---|---|---|---|---|---|---|---|---|
| Transparent | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | --- |
| Black | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | --- |
| Medium Green | 0.530 | 0.531 | -0.254 | -0.301 | -0.109 | -0.244 | 0.267 | 204 |
| Light Green | 0.670 | 0.670 | -0.186 | -0.238 | -0.075 | -0.185 | 0.200 | 202 |
| Dark Blue | 0.400 | 0.399 | -0.035 | 0.600 | -0.189 | 0.233 | 0.300 | 321 |
| Light Blue | 0.530 | 0.522 | -0.031 | 0.470 | -0.168 | 0.207 | 0.267 | 321 |
| Dark Red | 0.470 | 0.469 | 0.241 | -0.192 | 0.230 | 0.036 | 0.233 | 81 |
| Cyan | 0.670 | 0.671 | -0.308 | 0.258 | -0.297 | -0.042 | 0.300 | 262 |
| Medium Red | 0.530 | 0.529 | 0.311 | -0.248 | 0.296 | 0.047 | 0.300 | 81 |
| Light Red | 0.670 | 0.669 | 0.311 | -0.248 | 0.296 | 0.047 | 0.300 | 81 |
| Dark Yellow | 0.730 | 0.730 | 0.032 | -0.475 | 0.150 | -0.178 | 0.233 | 140 |
| Light Yellow | 0.800 | 0.800 | 0.023 | -0.340 | 0.107 | -0.128 | 0.167 | 140 |
| Dark Green | 0.460 | 0.460 | -0.217 | -0.277 | -0.087 | -0.216 | 0.233 | 202 |
| Magenta | 0.530 | 0.530 | 0.182 | 0.249 | 0.068 | 0.188 | 0.200 | 20 |
| Grey | 0.800 | 0.800 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | --- |
| White | 1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | --- |

$Y_{rgb}$ is the luminance calculated from the R,G,B voltages, as an internal check. R-Y and B-Y are color differences; note that they look nothing like the values provided by Texas Instruments. C is the magnitude of the chrominance vector (given as A' in the *Data Manual)*. $P_{iq}$ is the phase angle with respect to the I-Q axis; this is 33 degrees less than the angles with respect to the B-Y axis. The units of all values (except $P_{iq}$ and where otherwise specified) are volts.

What do the final RGB colors look like? I've created a sample color TIFF file, TMS9118.TIF, which you can retrieve. Take a look at it and let me know what you think.

tms9118.tif, RGB TIFF of VDP color palette

My first impression was that the colors were a little washed-out, especially the reds. Staring at an ADAM display on my Commodore 1701 monitor, side-by-side with TMS9118.TIF on my SVGA screen, though, changed my initial assessment. I wrote a program to slightly change the phase angle of the color values (the analog of a "tint" control") to see if there was any improvement, and (IMHO) there isn't. (To view the results of this tint test, look at TINT.TIF.) The color match is much better than Marat's in COLEMDOS.EXE; you can see his palette in MARAT.TIF, also available for download.

tint.tif, RGB TIFF of phase angle experiments with VDP color palette

marat.tif, RGB TIFF of Marat Fayzullin's ColecoVision emulator palette (note that the color bars are not in the same order as the other figures)

One possible reason for the colors tending toward the pastel is that the TMS9118 is subject to "rainbow" color distortions, due to interference from its on-chip NTSC composite video encoder circuitry. Keeping the color saturation down may minimize these effects. (Turn the "color level" control of your television set way up, and watch the color bleeding and ghosting which results.)

## IV. Converting PowerPaint 10K Binary Files to RGB Color TIFFs.

After all that trouble to figure out the ADAM color palette, I was ready to actually write the program to convert PowerPaint 10K binary files to RGB TIFFs. The program, 10K2RGB, in QuickBASIC 4.5 source (.BAS) as well as compiled DOS .EXE, are available for download. I've also provided two sample converted PowerPaint pictures: PPBOOTHD.TIF and PPBOOTAS.TIF. Let me know what you think of them. The former is proof of the origin of one pirated version of PowerPaint...thanks to Herman Mason (aa337@po.cwru.edu) for providing this gem.

- 10k2rgb.bas, QuickBASIC 4.5 source code, PowerPaint-to-RGB-TIFF utility
- 10k2rgb.exe, compiled MS-DOS executable, PowerPaint-to-RGB-TIFF utility

ppboothd.tif, RGB TIFF of PowerPaint 10K binary file, PowerPaint-HD boot screen

ppbootas.tif, RGB TIFF of PowerPaint 10K binary file, PowerPaint-ADAMserve boot screen

As for going back the other way, TIFF images to color PowerPaint 10K binary files, I'm still working on it. Right now, the program works pretty well if the source TIFF is exactly 256x160, but I haven't thought through all the math yet to make it work if the source TIFF is larger or smaller than this (though it should be easy--famous last words). Tune in again in a few weeks.

## V. My "Mystery Man" Interview.

Still working on it...a minor glitch occurred when a lab computer containing one of the interview E-mails crashed, and of course that was the one message I hadn't backed up yet :-( Fortunately, this particular message only had a few points of clarification in it, and I'm working with my interviewee to regenerate it. Hang in there, folks...

## VI. The Telegames DINA System.

I admit it, curiosity got the better of me and I bought one of the things. In case you hadn't heard of it before, it's a 3rd-party ColecoVision-compatible video game system. It has its own controllers, and can't use real ColecoVision/ADAM controllers, the Super-Action Controller, or any of the Expansion modules (Driving Controller, Roller Controller). I unpacked it long enough to try it out and make sure it works (no warranty or service from Telegames!), but I can't do a detailed review until after Christmas--I got it to give to my kids (as well as for me :-) ). So, mum's the word to Christina and Elanor until after 25 December.

A few quick takes on it, though...the built-in game, Meteoric Shower, is pretty good, a sort of Space Invaders game. The console is tiny and exudes cheapness. The DINA System nameplate is a sticker covering the true name--something in Chinese characters. Instead of the COLECOVISION boot screen, it's in Chinese. I did a quick exploratory operation inside the console and found a 4 mHz Z80, a TMS9928 VDP, a 32K ROM for Meteoric Shower (soldered in, but I'll fix that eventually), and an 8K ROM for the OS-7 operating system (also soldered in). I'm really anxious to dump those ROMs, especially the OS-7 ROM, to see if it's new software, or just the old with the pattern generators for COLECOVISION replaced with the Chinese characters. I'm also curious if Meteoric Shower is playable on a real ColecoVision/ADAM. The last item of interest is a second cartridge slot, for a 2x22-pin edgecard, plus some kind of contact switch activated when the cartridge would have been inserted. The minimal instructions with the unit say only "It was designed for a cartridge never marketed in North America, and you can only damage your unit if you try to plug anything into it."

Whew...this is a long article. Time to get it out.

See you again next week!

*Rich*

---

# This Week With My Coleco ADAM 9611.11

Hi gang, this is going to be a short status report...I missed last week because of flu which hit our household pretty hard, coupled with an out-of-town lecture I had to give. This weekend I finally got the flu myself, and to add insult to injury, we were hit by the nasty snowstorm which

1. buried us in over a foot of snow, and
2. knocked out our electricity for 25 hours, causing the indoor temperature to fall to 56 degrees F, which forced us to leave home and stay at my Dad's house until power was restored.

This weekend was also supposed to be our monthly B.A.S.I.C. users group meeting, but it got cancelled because of my illness. If I hadn't been sick, though, the power outage would have cancelled it anyway.

Hopefully this week I can finish some PowerPaint-related file conversion programs (for general consumption), and also finish writing up my interview with the ADAM Mystery Person (for which I know you're waiting with great anticipation).

Apologies for the shortness of *TWWMCA* this week, and I hope to make up for it next week.

*Rich*

---

# This Week With My Coleco ADAM 9610.28

by Richard F. Drushel

## I. Making a Boot Picture for ADAMserve PowerPaint.

This weekend I finally had a chance to start working on the new boot picture for the ADAMserve version of PowerPaint. My starting version was the picture I had modified (dated 2 September 1991!!!) for the Mini Wini hard disk version of PowerPaint. This in turn was based upon a special boot picture done by Tony Patterson and obtained from Tony in the summer of 1991, when he happened to be in town (he was a travelling children's photographer then) and we held our B.A.S.I.C. users group meeting in his motel room. (Boy was it crowded in there.)

I wanted to change the design so it was crystal-clear that this was the ADAMserve version of PowerPaint. To that end, I decided to replace the lower-right picture of a Mini Wini HD setup with something more appropriate: a bare ADAM and and an IBM PC-XT, linked by ADAMserve. I loaded the picture into PowerPaint and started to work.

It's been about 3 years since I did any serious drawing with PowerPaint, and when I did, I was using a 19-inch TV set as the monitor. My current monitor, a Commodore 1701, has beautiful color and resolution, but I was having a really hard time editing bitmaps on a 13-inch screen. My thoughts immediately turned to, "How can I export this picture to my 486 so I can edit the picture with Canvas?" Canvas is a fine product of Deneba Software, originally developed for the Macintosh, which I have used nearly every day in my job for image analysis and production of publication graphics. It's easy to edit bitmaps in Canvas, it has a zoom or "fatbits" mode so you can see what you're doing, and importantly, you can move parts of your bitmap to arbitrary locations on the screen--you're not limited to 8-pixel boundaries like you are in PowerPaint.

So, using ADAMlink V, I XMODEMed over the PowerPaint 10K binary file and took a look at it with Norton Disk Editor on my 486. After a brief inspection, and knowing something about the internal architecture of the TMS9928A video display processor, I had figured out the format of the file. Here's a brief description:

A PowerPaint 10K binary file has 5120 bytes of pattern information, followed by 5120 bytes of color information. Each bit in the pattern bytes is a pixel, 1 for on, 0 for off. Each pattern byte represents a stretch of 8 consecutive pixels. Each group of 8 pattern bytes represents an 8x8 graphic cell--it's *not* continuous raster image data. There are 640 such 8x8 cells, 32 across, 20 down. (Note that the physical screen is 24 cells down, but PowerPaint uses the bottom 4 for its own SmartKey menus, and this area is not available for user pictures.) The color bytes specify a 4-bit foreground color (pixel on) in the high nibble, and a 4-bit background color (pixel off) in the low nibble. The color bytes have the same spatial arrangement as the pattern bytes, i.e., 8x8 cells, 32x20. Because there is only 1 color byte for each pattern byte, each group of 8 pixels (corresponding to the horizontal boundaries of pattern cells) must have the same foreground/background colors. This is a well-documented hardware limitation of the

TMS9928A, and is responsible for the "color bleeding" effect seen if you try to set adjacent pixels within the same 8-pixel group to different foreground colors.

Fortunately for me, most of the boot picture was with a black foreground color. Thus, simply exporting the pattern bytes as a black-and-white raster image wouldn't lose much information. Besides, I was confident that I could do small touchups in PowerPaint itself after I had the majority of the image done under Canvas.

Most modern image file formats don't store their information in such a hardware-specific form as the PowerPaint 10K binary file. Usually the image is stored in some form of raster array--i.e., a continuous left-to- right sequence of pixels, starting at the upper left corner and going to the lower right corner, wrapping to the next line at the right margin. To convert the PowerPaint pattern information to a raster array, I wrote a QuickBASIC 4.5 program which decoded the file according to the above description, and wrote the pixels out to an intermediate file format which I developed myself for use in my research. I call it IMG; it's simply the first 2 bytes are the number of rows (lobyte, hibyte), the second 2 bytes are the number of columns (lobyte, hibyte), and the remaining bytes are the array of pixels, one byte per pixel. The encoding of the pixel bytes (i.e., what color they correspond to) can be anything you like. It can be grey levels 0-255, it can be an indexed color palette, etc. Usually I use it as an indexed color palette with colors 0-15, with the palette being that used by the IBM VGA display in 640x480x16 color mode.

The reason for making an IMG intermediate file is that I also had available a program I wrote to convert IMG files into greyscale TIFF files. TIFF (Tagged Image File Format) was one of the first image formats to be developed, and is very useful for moving greyscale and true-color images between different computer systems. My Canvas drawing program could read TIFF files, so I converted the IMG file to a greyscale TIFF and loaded it in. Now I could edit it to my heart's content :-)

When I was finished, I saved the final version as a greyscale TIFF. Since the drawing was still in black and white, I didn't need to worry about colors--yet. I still needed to convert the TIFF file's internal raster array of 1 byte per pixel into the encoded format required by the PowerPaint 10K binary file. I wrote another QuickBASIC program to perform this task. "Final" PowerPaint file in hand, I XMODEMed it back to the ADAM, and opened it up with PowerPaint.

The image pattern transfers had worked perfectly. Some of the background coloring was now no longer in the right place to match the foreground image, but that was easily cleaned up using DRAW BACKGROUND. As I had stated above, most but not all of the original picture had been with a black foreground color; it was now that I had to deal with the parts that had other foreground colors. A picture of a paintbrush with splotches of colored paint had been moved slightly to make room for additional artwork; when I moved the foreground patterns, I had *not* moved the color bytes. So, I had to retrace some of the picture in black (where new artwork had spilled into the old color information, causing it to appear in the old foreground colors), and re-color some things which were now black which should have been in color (the paint splotches). All in all, I was pretty happy with the result--and despite all the file transfers, new programming, etc., I'm convinced I did it faster than I would have had I slogged it out with PowerPaint. Not that PowerPaint is a bad tool; I just needed to be able to use it at the expert level, and I was very rusty--so I moved to where *I* could use *my* expert tools.

Now that I've seen how this has all turned out, it's trivial to modify the PowerPaint-to-IMG export program to

1. write greyscale TIFF files directly (I wrote library routines to read/write TIFFs, but the code was at work and all I had were the .EXE files so I went with what I had), and
2. write RGB color TIFF files, using the foreground/background color information.

   It's also not too bad to

3. import a greyscale TIFF file into PowerPaint format to yield black&white images.

   But the real problem would be

4. importing an RGB color TIFF into PowerPaint.

I'm not sure how I'd make it choose what colors to use if it ran into the "color bleeding" problem.

If there is any interest in (1), (2), (3), or even (4), please let me know and I'll put them on my "to do" list. Others are free to jump in if they like.

## II. Researching the History of the ADAM.

This section of *TWWMCA* will be rather short, because the project is still in progress, and I don't want to make a report until everything is complete.

Those of you who have "surfed the Web" know that there are several sites from which you can search for keywords on *any* web page that's been up for the last year or so. These "search engines" provide you with the URL (Uniform Resource Locator) of any matches, so you can go to the actual web page and check it out. My favorite of these is the Altavista Search Engine at http://www.altavista.com/. It has an Advanced mode in which you can combine your keywords with AND, OR, NOT, etc., group terms with parentheses, search for words which are NEAR one another (within 10 words forward or back). It's very powerful.

In my tinkering with Altavista, I've managed to locate a person who was very important in the history of the ADAM. I'm currently conducting an interview by E-mail; he's been gracious enough to consent to this. When I have written my summary, and he has had a chance to approve it, I'll post it to the mailing list.

To keep you on the edges of your seats, and to keep my contact from being flooded with well-intentioned E-mail requests, I'm not going to tell you who he is right now. Suffice it to say that, without this man, the Coleco ADAM as we know it would not exist. His story is fascinating, and he seems prepared to tell it now.

You are free to speculate, but I ain't tellin' yet. Keep reading the mailing list! :-)

## III. Miscellany.

Pat Williams mailed me the disk with her report of our B.A.S.I.C. meeting on Sunday 13 October 1996, but I haven't had a chance to convert it to plaintext and post it to the mailing list. I hope to do this in the next couple of days.

Ron Mitchell (ronaldm@mars.ark.com) sent me a lengthy E-mail this morning in which he exposes all the warts in the current version of ADAMserve. I knew they were there, and some aren't "bugs", they're "features" :-) He's writing a review for A.N.N. and wanted some info from the horse's mouth before the horse is sent off to the glue factory. I spent this morning replying to his questions. The horse hopes that he has managed to forestall his appointment with the knacker-man :-)

See you again next week!

*Rich*

# This Week With My Coleco ADAM 9610.21

by Richard F. Drushel

## I. Working with VideoTunes.

Last week, while backing up EOS volumes on my Mini Wini hard drive system, I rediscovered some songs that I had transcribed for VideoTunes. If you've been to any of the ADAMcons where Chris Braymen's (70057.2035@compuserve.com) ADAM Information Manager has been set up to display video advertizements with music, you may have heard some of these transcriptions: themes from *The Flintstones* and *Dinosaurs!*, Glenn Miller's *A String of Pearls* and *Sunrise Serenade*, and Glen Gray's Casa Loma Orchestra signature tune *Casa Loma Stomp.* If you have a MIDI interface, you can use the program VTplayer to play VideoTunes songs to a MIDI device, or even save the songs as MIDI files (whence they can be edited in Sequel, a MIDI sequencer program). These songs I transcribed are available if anybody wants to have copies of them; just let me know.

I also found one unfinished song: Glenn Miller's *In the Mood.* I had transcribed the final zig-zag trumpet chorus and started on the introduction when I stopped working on it (1993?). Being "in the mood", I spent an hour or so finishing the intro and putting in one cycle of the "A" theme.

It's somewhat difficult to transcribe Big Band tunes with only 3 voices (which is all you get with ADAM's sound chip), because [music theory mode on] there are so many 6th chords (i.e., 1-3-5 triad plus the 6th; on your piano, play C-E-G-A simultaneously to hear a C6 chord). Four tones and three voices ... the solution is moving or "broken" chords in which, as the song progresses, you hear 3 of the 4 tones, but never the same 3 simultaneously. You ear performs some kind of "temporal summation" and you hear all 4 tones correctly.

Another difficulty in transcribing swing music is that VideoTunes has no provision for triplet or "swung" eighth notes. In a swing tune in 4/4 time (4 beats per measure, quarter note is 1 beat), written eighth notes aren't played evenly--the first eighth note of a beat is played at 2/3 the length of a beat, and the second eighth note is played at 1/3 the length of a beat. The closest that VideoTunes can get to this rhythm is with dotted eighth/sixteenth note pairs, but the result doesn't "swing"--it sort of lurches along. [trying to think of examples that everybody has heard] For "straight" eighths, think of a polka or a Sousa march; for "swung" eighths, think of *In the Mood;* and for the dotted eighth/sixteenth note approximation, think of any of Lawrence Welk's "champaigne music".

Chris Braymen is the professional musician among us; maybe I should ask *him* to take a stab at explaining this :-)

## II. A Bug in ADAMcalc for ADAMserve.

As I noted in a prior post to the mailing list, after a conference call with Herman Mason (aa337@po.cwru.edu) and Bob Slopsema (72117.3003@compuserve.com), I realized (to my chagrin) that the version of ADAMserve I released at ADAMcon VIII didn't include the specially-patched ADAMcalc and PowerPaint versions required for these popular/indispensible programs to work under ADAMserve. I thought it would be an easy task to put together some disk images based upon my own server hard drive volumes (i.e., keep the vital programs, delete personal images, debugging stuff, etc.).

Well, for PowerPaint I'm *almost* there. To prevent confusion with other versions of PowerPaint which are floating around (original, RLE-enhanced, hard-drive-compliant), I figured I should edit the boot screens and pictures so that it would be clear that it's ADAMserve PowerPaint which is booting. The boot screen I fixed; it has a nice ADAMserve message in it now. The boot picture will take a little work in PowerPaint (I want to have a picture of an ADAM and an IBM side-by-side), which I haven't used seriously since 1991 or so (when I made my SmartBASIC 1.x logo for the disk labels). If I get stuck, I'm sure I can always call upon our PowerPaint expert PJ Herrington (76537.1271@compuserve.com) for assistance.

As for ADAMcalc...I thought *this* was the one that would be simple and straightforward. When I booted it up (ostensibly to look to see if there was anyplace I could stick some kind of "ADAMserve version" message), I idly tried to load a stock spreadsheet from the server hard drive. STORE/GET, then press a SmartKey to pick a disk/tape, and ... uh-oh, the SmartKey for the server hard drive (DRIVE II) did *not* appear! The two server floppy drives (DISK I and DISK II) showed up, as did the tape drive (DRIVE I). Hmmm...the icons for both tape drives appeared correctly during the boot device detect routine. Rats, it worked before...in the *previous* version of ADAMserve...in the version *before* I made the alterations required to get FileManager to work. @#$%&#! Strangely enough, if I don't select a drive from the SmartKey menu and press Done, ADAMcalc will happily read from the server hard drive, load spreadsheets from it, save workspaces to it, everything it's supposed to except show up on the SmartKey menu as a valid device. Double @#$%&#!

I have not yet had time to dig into ADAMcalc to find the actual code that is operating during the drive select operation (after STORE/GET). Of the 36 blocks of ADAMcalc, I have disassembled only 5. ADAMcalc is another of those programs which is beautiful for a user to use but utterly repulsive when a programmer "looks under the hood". It's a different kind of "evil" than PowerPaint, for example. PowerPaint is ugly on the inside because the programmer, Sol Swift, didn't know any better; it has lots of empty, unused spaces and inefficient code. ADAMcalc is ugly on the inside because the programmers were *very* *good* and were trying to pack as much functionality as possible into a small space; they used every trick in the book to optimize for size. As a result, the code (in uncommented, disassembly form) is very terse and very hard to read.

## III. More Basement Cleaning.

In preparation for our monthly users group meeting, I desperately tried to finish cleaning/reorganizing my basement, so that as many of my computers as possible were set up and functional, and also that the remainder were stored neatly and accessibly. I did not entirely succeed by 2:00 PM Sunday 20 October 1996 :-( I have 5 complete computer systems installed on 4 desks/ tables (3 ADAMs, my 486DX2-66, and my PC-XT), with one ADAM and my rescued 286 laptop project (see *TWWMCA* 9609.30) still in storage. Unfortunately, the rest of everything else is haphazardly stashed into boxes, pending a more careful sorting-out. In partial defense, I had to repair a number of items as I uncovered them (e.g., my 80-column video unit had some bad solder joints; my 256K memory expander had a broken addresser cable). I also rescued/repaired a 1972 Shibaden black&white TV set that a lab at work was junking, for use as a monitor. (The picture was fine, but there was no sound, and the on-off plastic knob had been lost. The speaker was blown; a $3 Radio Shack special fixed that. As for the knob, I had one in my spares box.)

A few things that I *know* I had down here appear to be missing, some of which I had as part of my "ADAM Museum" at ADAMcon VIII. I hope they're just mislaid in the remaining clutter, or (possibly) got mixed up with some of Herman Mason's or George Koczwara's (aa436@po.cwru.edu) stuff at the convention. (Most of all our respective stuffs rode back from Elyria in my van.) I'll be really disgusted if I ended up leaving any of it at the convention :-(

## IV. October B.A.S.I.C. Users Group Meeting.

This Sunday was the October meeting of our users group, B.A.S.I.C. (Best ADAM Support In Cleveland). In attendance were me, Herman Mason, George Koczwara, Pat Williams, and Jean Davies, plus my wife and kids at various points. Pat is supposed to be writing a summary of what went on at the meeting and mailing me a disk with the text. I even gave her a disk mailer :-) When I get the text, I'll post her report to the mailing list. After a little bit of wrangling with some balky hardware, we managed to have a productive meeting, with everybody having left by about 6:30 PM. During the meeting, I had a phone call from Michael Hurst (ao853@freenet.toronto.on.ca), who was having some difficulties trying to get the ADAMcon banner JPEGs I had put up for download. I was able to get to them without problems. Michael, try the URL as I listed it above and let me know what happens.

## V. New ColecoVision-Compatible Hardware from Telegames USA.

Telegames USA (http://www.telegames.com/) sells, in addition to new ColecoVision game cartridges (and cartridges and complete game systems for Sega, Nintendo, Atari, etc.) a *new* ColecoVision-compatible game machine called the DINA System. The info on their web page is sketchy (no pictures), but they say they redesigned the ColecoVision to have a sleeker box and smaller, Nintendo-style game controllers. It even has a built-in game called Meteor Shower. They say it will play any ColecoVision game cartridge which does *not* require a Super-Action Controller, a Roller Controller, or a Driving Module.

I requested (and received) the Telegames USA printed catalog. No pictures, just listings of game cartridges available for the many gaming systems they stock. The DINA System lists at $49.95. They sell genuine ColecoVision/ADAM game controllers at $19.95 each, and something called a QuickShot 3 Joystick for $29.95. I don't know if the DINA System (a) can accept a

ColecoVision/ADAM game controller, or (b) if the $49.95 DINA System price includes game controllers, or (c) if the $29.95 QuickShot 3 Joystick *is* the game controller for the DINA System.

Telegames USA currently has a number of special offers (bundles of cartridges, freebies for X number of cartridges bought, X free cartridges if you buy a Y system unit). Two of interest are SPS03 (buy a DINA System and get a free Donkey Kong Jr. cartridge) and SPH03 (buy any 3 ColecoVision cartridges and get a free DINA System). I'm curious about this DINA System; I'm tempted to spring for one as a "museum acquisition" :-)

Note: I'm not affiliated in any way with Telegames USA. According to Norm Sippel's ColecoVision Home Page (http://www.infinet.com/~ngsippel/cv.html), Telegames USA is the current copyright holder for the ColecoVision hardware and many/most of the ColecoVision games.

Well, that's it for this week. See you next week!

*Rich*

# This Week With My Coleco ADAM 9610.13

by Richard F. Drushel

## I. Goodies from Bob Slopsema.

This week I received two disks in the mail from Bob Slopsema (72117.3003@compuserve.com). The first was an ADAM disk containing a Spanish vocabulary tutor program for my daughter, Christina (age 9.5). The second was an IBM disk containing text versions of the playing instructions for a number of ColecoVision game cartridges. Let me talk about these two disks in a little more detail:

"The Spanish Vocabularian", © 1986 by Marathon Computer Press, is a `BRUN`able SmartBASIC program which lets you practice both English-to-Spanish and Spanish-to-English vocabulary. It has several dictionary files grouped by letter of the alphabet (you practice words which all begin with the same letter). Depending upon the mode chosen, the program gives you words in one language, and you have to type the equivalent in the other language. The program does tell you what part of speech the word is (noun, verb, adjective, etc.). Christina has been taking Spanish in school for the last 2 years, but it's all been conversational Spanish so far, so she doesn't know how to spell many of the words yet. Later on in her education, however, this program should be a useful tool for her. A final technical note: even though the program (as I received it) must be `BRUN`, when you exit normally to SmartBASIC, the program is left in memory in `LIST`able form. Maybe Bob put it into `BRUN` form from a regular A-type original?

Also on the disk, perhaps unbeknownst to Bob, were several SmartBASIC games. A bunch of these were A-type files masquerading as H-type (but a little work with FileManager fixed that). The games include GOLF (clearly based on a BASIC-11 version I played 15 years ago on a PDP-11/34 mini- computer, which BTW was one of the first games I ported to Microsoft BASIC for the IBM-PC waaaaay back when), HANGMAN (the classic guess-the-letters- in-the-word game), ICECREAM (really a `GR`-mode graphics demo; it asks you for ice cream flavors and draws a low-res picture of the resulting ice cream cone), GORKY (some kind of ADVENTURE/ZORK text-based adventure game), GRANPRI (a Grand Prix racing game), and FLUTE (another adventure game, the object being to retrieve a magic Golden Flute). My daughters didn't have much time this week to play these games, but I'm sure they will later.

As for the IBM disk with the ColecoVision game instructions, they will be very useful for figuring out some of those games that came with the ColecoVision emulator. Bob sent me them in some rather obscure word processing format (something called First Choice), leading me to believe that he typed all these in himself--a lot of work, for which I'm grateful. Possibly I can figure out a way to convert them into plain text so that others can read them. I can't recall if Norm Sippel has any ColecoVision game instructions on his home page (http://www.infinet.com/~ngsippel/cv.html).

## II. Backing Up an EOS Hard Drive.

I had mentioned a few weeks back that my Mini Wini hard drive system was dead due to a bad power supply. Using a power supply borrowed from George Koczwara (aa432@po.cwru.edu), I tried to bring the hard drive back to life. This particular hard drive has been on its last legs anyway--the motor bearings are shot, and it howls loudly--and all I wanted to do was back up all the data on it and then put it into my junk pile. I've had a spare 3.5-inch 20MB MFM replacement drive for over a year now, and just had never gotten around to doing the backups. Saturday was the day I'd get around to it.

Believe it or not, I don't think that there is *any* hard drive backup software available for the ADAM, EOS or TDOS. Since, as I've said before, I have hardly anything on the TDOS partition of my hard drive, I was only worried about backing up the 10 1-meg EOS volumes. How to do it?

Most authorities simply use FileManager to block-copy the contents of each volume onto a floppy disk. Before the advent of the Micro Innovations 1.44 MB floppy drives, most people I know would use a 720K disk and make sure that they never put more than 720K of data in any one EOS volume. The hardest volume to enforce this limit on is volume 0, because that's where all the boot blocks are stored, as well as entire software distributions that are so badly behaved that they will only run from particular blocks in volume 0 (I believe SmartLOGO is like this). Since I have a 1.44 MB disk drive (and also my ADAMserve systems will have access to a 1.44 MB PC disk drive), I decided to do block copies; but I decided *not* to use FileManager to do it.

Instead, I wrote two simple programs in SmartBASIC 1.x (the enhanced SmartBASIC interpreter I released in 1991): one to do backups of individual hard drive volumes to a disk, and another to restore volumes from a saved disk. Here are the programs. Remember, they're in SmartBASIC 1.x, and will abort with an error message if you try to run them under regular SmartBASIC 1.0.

```
  1 REM backuphd5 / created 9610.12 by Richard F. Drushel
  2 REM back up a 1024-block EOS volume onto a 1.44MB disk in d5
  3 REM note:  requires SmartBASIC 1.x!!!
  4 REM *****************************************************
  5 IF VER(0)>=20 THEN GOTO 10: REM we have SB1.x
  6 HOME: PRINT CHR$(4): INVERSE
  7 PRINT " WRONG SMARTBASIC VERSION ": NORMAL: PRINT: NEW
  9 REM *****************************************************
 10 LOMEM:MEM(0)+1024
 20 TEXT40: DSIZE(5)=1440: DSIZE(2)=1024
 25 PRINT "HARD DISK BACKUP TO d5": PRINT
 30 INPUT "Enter volume: ";v
 40 v=INT(v): IF v<0 OR v>9 THEN BEEP: GOTO 30
 50 POKE 58343,v
 54 PRINT
 55 y=VPOS
 56 REM *****************************************************
 60 FOR n=0 TO 1023
 65      n$=STR$(n)
 66      WHILE LEN(n$)<4
 67           n$="0"+n$
 68      WEND
 70      LOCATE y,1: PRINT n$;
 75      ONERR GOTO 1000
```

```
  80      BLREAD 2,n,MEM(0)
  85      CLRERR
  86      ONERR GOTO 2000
  90      BLWRITE 5,n,MEM(0)
  95      CLRERR
 100 NEXT n
 110 LOMEM:MEM(0): END
 999 REM ******************************************************
1000 BEEP: PRINT: PRINT "Bad Read ";n
1010 y=VPOS: RESUME 85
1099 REM ******************************************************
2000 BEEP: PRINT: PRINT "Bad Write ";n
2010 y=VPOS: RESUME 95
```

```
   1 REM restorhd5 / created 9610.12 by Richard F. Drushel
   2 REM restore a 1024-block EOS volume from a 1.44MB disk in d5
   3 REM note:  requires SmartBASIC 1.x!!!
   4 REM ******************************************************
   5 IF VER(0)>=20 THEN GOTO 10: REM we have SB1.x
   6 HOME: PRINT CHR$(4): INVERSE
   7 PRINT " WRONG SMARTBASIC VERSION ": NORMAL: PRINT: NEW
   9 REM ******************************************************
  10 LOMEM:MEM(0)+1024
  20 TEXT40: DSIZE(5)=1440: DSIZE(2)=1024
  25 PRINT "HARD DISK RESTORE FROM d5": PRINT
  30 INPUT "Enter volume: ";v
  40 v=INT(v): IF v<0 OR v>9 THEN BEEP: GOTO 30
  50 POKE 58343,v
  54 PRINT
  55 y=VPOS
  56 REM ******************************************************
  60 FOR n=0 TO 1023
  65      n$=STR$(n)
  66      WHILE LEN(n$)<4
  67           n$="0"+n$
  68      WEND
  70      LOCATE y,1: PRINT n$;
  75      ONERR GOTO 1000
  80      BLREAD 5,n,MEM(0)
  85      CLRERR
  86      ONERR GOTO 2000
  90      BLWRITE 2,n,MEM(0)
  95      CLRERR
 100 NEXT n
 110 LOMEM:MEM(0): END
 999 REM ******************************************************
1000 BEEP: PRINT: PRINT "Bad Read ";n
1010 y=VPOS: RESUME 85
1099 REM ******************************************************
2000 BEEP: PRINT: PRINT "Bad Write ";n
2010 y=VPOS: RESUME 95
```

For those of you who have never seen SmartBASIC 1.x, let me explain a few points of interest in the programs:

| | |
|---|---|
| line 5 | The `VER(0)` function returns an internal revision number for SmartBASIC 1.x. Under regular SmartBASIC 1.0, `VER(0)` will cause a default 11-element array to be allocated, all of whose elements are 0. Thus, if this statement is run under SmartBASIC 1.0, it will return 0, thus causing the program to fall into the ensuing error message code and abort. |
| line 10 | `MEM(0)` is a pointer to the default start of variable space in the SmartBASIC 1.x interpreter. Moving `LOMEM` to 1024 bytes above this pointer reserves a 1K disk block transfer area. By using the `MEM(0)` pointer instead of a hard-coded number (like 27407, the equivalent in SmartBASIC 1.0), I can run this program under any future version of SmartBASIC 1.x, which might have a different memory map and a different absolute address for `MEM(0)`, and it will still run correctly. Portability of programs was a key design consideration for SmartBASIC 1.x. |
| line 20 | The `DSIZE(drive)` command sets the maximum valid block number for the particular drive 1-7. This value is used for range- checking the `BLREAD` and `BLWRITE` block I/O commands. `DSIZE(0)` sets the number of directory blocks allocated with a subsequent `INIT` or `FORMAT` command. |
| line 50 | The "magic" address 58343 is where all known versions of EOS patched for hard drives (by HARDDISK) store the current volume for the hard drive. In a future revision of SmartBASIC 1.x, I should also have a pointer to this address as a system function; but in 1991, I didn't have a hard drive system of my own, or access to one. |
| line 55 | Note that `VPOS` does not require the dummy argument `VPOS(0)` like it does in SmartBASIC 1.0. |
| lines 66-68 | `WHILE...WEND` is a looping construct found in many other programming languages, such as C, Pascal, and Microsoft BASIC. As long as the condition in the `WHILE` statement is true, it will continue to execute all the statements up to the `WEND`. When the condition becomes false, execution begins at the first statement after the `WEND`. It's possible to use an `IF...GOTO` constuct to perform the same function as `WHILE...WEND`; the demonstration of this is left as an exercise to the reader `:-)` The purpose of the `WHILE...WEND` loop in these programs is simply to make the block counter a 4-digit 0-padded number, for convenience of display. |
| line 70 | The `LOCATE line,column` statement does the same thing as `VTAB line: HTAB column` in SmartBASIC 1.0. I borrowed this syntax from Microsoft BASIC. |
| lines 80,90 | The `BLREAD` and `BLWRITE` commands read and write blocks from a drive, respectively. These were the very first two commands I wrote for SmartBASIC 1.x, and the need for them was my inspiration to attempt SmartBASIC 1.x. I *hated* all the programs from |

| | |
|---|---|
| | NIAD et al. that were nothing but obscure `PEEK`s, `POKE`s, and `CALL`s. I also *hated* having to `POKE` in the same machine code into every program I wrote that needed to do block I/O. "Why not have a command...?" so I made one. To do this in SmartBASIC 1.0, you'll need to write some machine code, reserve space for it with `LOMEM`, `POKE` it in, and `CALL` it. |
| line 110 | Putting `LOMEM` back to `MEM(0)` deallocates my 1K transfer area and puts SmartBASIC 1.x's memory map back the way it was. |
| lines 1010,2010 | `RESUME` can take an optional line number in SmartBASIC 1.x, thus allowing you to precisely control what happens after an error occurs. Sometimes you *don't* want to just go back and try the same thing again. If you use `GOTO`, you are leaving the internal error handlers in an unstable state, and the system is likely to crash. Indeed, error handling in SmartBASIC 1.0 is *completely* broken; if another error occurs inside an error handler (i.e., code after the line you `ONERR GOTO`), the system crashes. I'm happy to say that I fixed it in SmartBASIC 1.x. |

Hmmm, this is turning into a big plug for SmartBASIC 1.x (which I hadn't really intended). The software is still available, from me direct, or from HLM-GMK. If you're interested, ask me questions and I'll post them to the mailing list. I must say that SmartBASIC 1.x has been a wonderful debugging environment for me, and has been an indispensible tool for investigating hard drives, serial communications, memory bank-switching, RAMdisks, and ultimately the ADAMserve system.

## III. Miscellany.

I had a few more things I'd intended to talk about this week, but it's getting late, and I think I'll save them for next time.

Just a note in closing: it's nice to see the progress/status reports from Ron Mitchell (ronaldm@mars.ark.com) and PJ Herrington (76537.1271@compuserve.com). I also have heard from some of the newer members of the mailing list. Nice job, folks, and keep up the good work!

See you again next week!

*Rich*

# This Week With My Coleco ADAM 9610.06

by Richard F. Drushel

## I. Building a Monitor Stand.

One of the best things I ever built for my dedicated ADAM system was a wooden stand for the monitor. While it's possible to set a monitor directly on top of the ADAM CPU (if it isn't too heavy), it's really not a good idea--the CPU case isn't so strong, you can get electrical interference between the ADAM video circuitry and the monitor, and it's really inconvenient if you need to open up the CPU to get at expansion cards.

My monitor stand design is basically a 4-legged table which fits over the CPU, with about 6 inches of space between the top of the CPU and the wooden table top (thus allowing access to the internal expansion slots). The space between the legs at the sides allows access to the power cable and disk drive ADAMnet cable (left side) and expansion port devices like serial modems (right side). To provide for convenient electrical hookups, I also have a 6-port powerstrip attached to the back, with a 6-foot cord. This way, wherever I take my ADAM, I only need to find one 120-volt outlet and I'm ready to go.

As I mentioned a few weeks ago, I've been slowly reorganizing my basement so that I can have as many working ADAM systems set up as possible, for use by my kids and for our users group, B.A.S.I.C. The first ADAM system I set up was not my "experimental" system, but I gave it the latter's wooden monitor stand. In order to set up more ADAMs, I would need more monitor stands; thus, I'd have to build them. This week, I built another one.

Here are abbreviated, no-picture plans for my monitor stand. The text should be pretty self-explanatory, but if anybody wants something fancy like a measured drawing, please let me know. Note: all dimensions are in feet and inches; you progressive Canadian metric types can convert :-)

### Materials.

One 8-foot 2x4 (make sure it's straight, no twists or bad knotholes), one 4-foot length of 0.75-inch square hardwood moulding, four 6x8 inch metal angle brackets (the kind that are stamped from steel and have a U-shaped channel in them, not just a right-angle bend of bar stock), 36 1-inch x 8 steel wood screws. You can get fancy and stain/varnish or paint the wood if you like, but bare wood travels better when piled into a van full of other computer stuff on its way to an ADAMcon :-) Optional: a 6-port powerstrip with or without AC and/or phone line surge protectors, circuit breaker, etc.

## Tools.

Hand or electric drill, bit to match wood screw diameter, saw (you can use a carpenter's saw or backsaw if you're really good, miter box with backsaw is better, best is radial arm saw or other console-mounted saw), large screwdriver (slotted or Phillips, to match head of wood screws), steel measuring tape or large carpenter's square, 150-grit sandpaper and sanding block or electric sander.

## Overview of Design.

Two 26-inch beams supported by two 10-inch legs each. Joint of beams and legs reinforced with angle brackets. Two assemblies connected by four crossbeams underneath main beams at an (outside) width of 12 inches. See the ASCII artwork below.

## Cutting the 2x4.

Measure the lengths of the two beams (26 inches each) and the four legs (10 inches each) on the 2x4. Remember to leave the width of the saw blade (kerf) between the measurements, or some of the pieces will end up too short. Cut the pieces. Sand them down smooth. If you cut them by hand, make sure that the edges are square (stand up the pieces on a flat surface and see if they veer to one side or another); sand them flat if not. (This can be a major chore if you don't have a power sander.)

## Assembling the Legs.

Lay out the two 26-inch beams side-by side, good wide side down. Stand up the legs onto the beams, one leg at each end. After you're sure that everything is lined up and square, position the angle brackets at each corner, 8-inch side down, 6-inch side up, centered on the beams/legs. You can get fancy and draw centerlines etc. if you want to, but by eye is good enough. With a sharp pencil, trace the screw holes onto the wood, being careful not to disturb the alignments. This can be tricky if you have shaky hands. My angle brackets have 6 holes per bracket; your mileage may vary. Use a hammer and a nail/centerpunch to mark the center of the holes, then drill them out. The drill bit has to be a little smaller than the shaft of the wood screws you're using, but not so small that it's too hard to put in the screws. (Don't make them too big, though, or the screws won't bite into the wood and they won't hold.) Don't drill all the way through, either! About 1 inch deep is right. Now screw everything together. If you are of average strength but are having a hard time putting in the screws, your pilot holes are too small; try making them a little bigger with some up and down strokes with the electric drill. Once they're assembled, turn the two assembiles over and make sure that they are level and aligned. You might even try a test with your intended monitor to make sure that it doesn't bend under load.

## Assembling the First Two Crossbeams.

Cut two 12-inch pieces of the 0.75-inch square moulding (again, remember to allow a kerf thickness between the pieces when measuring). Lay the completed leg/beam assemblies top down (legs in the air) side by side, 12 inches apart. Use a carpenter's square, a wall, etc. to keep the ends (legs) parallel. Lay the two crossbeams across the main beams, each one just past the tip of the angle bracket. Mark two holes at each end of each crossbeam, about 2 inches apart and about 1 inch in from each edge of the 26-inch beam. Again, you can measure exactly or lay it out by eye. Take the crossbeams aside and drill out the holes (you don't want to disturb the setup on the floor). After they're drilled, bring them back over, reposition them, and mark the holes onto the beams with a nail/ centerpunch. Drill out these holes. Finally, attach the two crossbeams with screws. Turn the almost-completed stand over and make sure it's still square and level.

## Assembling the Last Two Crossbeams.

Turn the stand top down, legs up again. Measure the distance between the inside edges of the angle brackets, across the center gap, back in the corner where the legs meet the beams. Depending upon how accurately you positioned the angle brackets, these lengths may not be the same at each end, but it's not too critical. Cut two more pieces of the 0.75-inch square moulding to match what you just measured. (Mine ended up 7 inches long apiece.) Position one in each corner across the center gap. Mark one hole at each end of each crossbeam; drill the holes. Now comes a tricky part: depending upon how big your drill is, you may or may not be able to get it in position to drill straight down through the existing holes in the crossbeams. If this is the case, you have no recourse but a heavy-duty screwdriver and elbow grease to get the screws in that last quarter- inch or so. In the best of all possible worlds, you could lay out all the screw holes before anything was assembled, drill them out, and then put it together and have everything line up correctly. In my experience, unless you have $20K of shop tools, you can't get it right by predrilling. You could unscrew the legs, but for me it was easier to just screw in the crossbeams. Screw them down, turn the stand over, and admire your handiwork!

## Attaching a Powerstrip.

Every powerstrip I've seen has some kind of holes for mounting it by screws. There may be a set of flanges on the outside of the case, but usually there are miserable lightbulb-shaped holes in the back: screw heads fit through the round part of the hole, then you slide the powerstrip right/left/up/down so that the shafts of the screws go into the slot part of the hole. In the latter case, the hard part is getting screws mounted so that they have the right spacing. Don't even try to measure them; instead, make a pencil rubbing of the back of the powerstrip onto a piece of paper. Lay a piece of paper over the back, and scribble lightly with the side of a soft pencil all around the holes, being careful not to move the paper (tape it down if you like). When you're done, you have a mirror image of the slots. Mark the centers of the round parts of the holes, poke through the centers with the pencil point, then turn the paper over and use it as a template for laying out the holes on the back of the monitor stand. Put in the mounting screws (predrilling isn't usually necessary, but you can if you want), and screw them in until there's just enough gap between the heads and the wood to equal the thickness of the metal/plastic back of the powerstrip. Make a test fitting with the screws a little loose; when you're sure it will fit, tighten the screws down a little at a time and keep refitting (press on, slide) until it's snug.

Congratulations, you're done!

Here's some ASCII art (not to scale) to show the general arrangement of the parts, in case my verbal description wasn't clear to you:

```
+----------------------------------------+
|                                        | <2"
|                                        |
+---+--------+-+-----------+-+-------+---+
|   |@@@@@@@@+-+     ^     +-+@@@@@@@|   |
|   |@@@ ^    ^      |            @@@|   |
|   |@@  |    |      |             @@|   |        FRONT/REAR VIEW
|   |@   |    |      |              @|   |
|   |@   |    |      |              @|   |
|   |    |    |      |               |   |
|   |    |    |      |               |   |
|   |    |    |      |               |   |
|   |    |    |      |               |   |
|   |    |    |      \               |   |
+---+    |    \       \              +---+
  ^      |     \       \              <2"
  |      |      \       \
  |      |       \       \
 10"   angle    12"     26"
 leg  bracket    cross   beam
                 beam


+---+--------+-+-----------+-+-------+---+
|   |        | |           | |       |   |
|   |@@@@@@@@|.|           |.|@@@@@@@|   |
|   |@@@@@@@@|.|           |.|@@@@@@@|   | <4"
|   |-+      | |           | |     +-|   |
+---+.|------| |-----------| |-----|.+---+
    | |      | |           | |     | |
    | |      | |           | |     | |
    | |      | |           | |     | |<-------\       BOTTOM VIEW
    | |      | |           | |     | |         \   (LEGS POINTING UP)
    | |      | |           | |     | |          \
+---+.|------| |-----------| |-----+.+---+       \
|   |-+      | |           | |     +-|   |        \
|   |@@@@@@@@|.|           |.|@@@@@@@|   |         |
|   |@@@@@@@@|.|           |.|@@@@@@@|   | <4"     |
|   |    ^   | |           | |       |   |         |
+---+----|---+-+-----------+-+-------+---+         |
  ^      |    ^      ^                <2"          |
  |      |    |      |                             |
  |      |    \      \                             |
  |      |     \      \                            |
 10"   angle   12"     26"    . is a screw     approx. 7"
 leg  bracket   cross   beam                    crossbeam
               beam


+-----+       +-----+
|     |<2"    |     |<----\
|     |       |     |      \
+-----+-------+-----+       \
|     |_______|     |         \
|     |   ^   |     |          \
|     |   |   |     |          |        END VIEW
|     |   |   |     |          |
|     |   |   |     |          |
|     |   |   |     |          |
|     |   |   |     |          |
|     |   |   |     |          |
|     |   |   |     |          |
|     |   |   |     |          |      Note: "2x4"s aren't really
+-----+   |   +-----+          |    2 inches by 4 inches any more;
   ^      |     <4"            |  they're more like 1-9/16 by 3-7/16
   |      |                    |
   |      |                    |                  :-(
  10"    end                 26"
  leg crossbeam              beam
```

## II. Pictures from ADAMcon VIII.

Dale Wick (dalew@truespectra.com) asked me to send him pictures of the eight ADAMcon banners which I photographed at the ADAMcon VIII banquet. (Traditionally, all the banners are hung up in the banquet room during the banquet; I hope that Bob Slopsema (72117.3003@compuserve.com) can find enough wall space for them for ADAMcon 09!) Dale wanted to put the pictures up on the Coleco ADAM Home Page (whose URL I'd give you except that the Coleco ADAM Home Page has moved and I don't know where), which means that somehow the photographs would have to be converted into graphic image files. Since I have access to the necessary equipment at my lab, I agreed to convert the photographs to image files.

The basic strategy is simple: use a device called a flatbed scanner to scan the photographs and produce a high-resolution color image file, optionally do some image processing on the image, then convert it into a file format compatible with common Web browsers like Mosaic or Netscape.

The typical modern scanner is capable of scanning images at 600 dots per inch (dpi) spatial resolution, with each dot (pixel) represented in 24-bit "true color" (i.e., 8 bits each for red, green, and blue light). It can also produce images in 8-bit color (3 bits for red and green, 2 bits for blue), 8-bit greyscale (0=black, 255=white, shades of grey in between), and 1-bit line art (a 0 bit=white, a 1-bit=black).

The greater the spatial resolution and more accurate the color, the bigger the resulting image file. For example, a 1-inch square, 8-bit greyscale image at 72 dpi is 72 x 72 = 5,184 bytes. The same 1-inch square image at 300 dpi and 24-bit color is 300 x 300 x 3 = 270,000 bytes! The size at 600 dpi/24-bit color is left as an exercise to the reader :-)

The larger the image file, however, the longer it will take to download to a computer for viewing. Assuming a 14400 bit-per-second (bps) modem at 8 data bits, no parity, and 1 stop bit, the 72 dpi greyscale image mentioned above will take (9 x 5184 / 14400) = 3.24 seconds to download, while the 300 dpi "true color" image will take (9 x 270000 / 14400) = 178.75 seconds!

The solution to minimizing image transfer time (aside from faster and faster modems) lies in the technique of image compression, i.e., the image data are encoded in a format which takes up less space than the original image. Some methods of compression do not alter the original data, such as the Lempl-Ziv-Welch (LZW) algorithm used in the CP/M CRUNCH.COM and earlier versions of the MS-DOS PKZIP.EXE; the compression algorithm is completely reversible. Other methods of compression, such as that used in JPEG images and MPEG digital movies, achieve great size reduction ratios by changing the original image through spatial averaging of colors. These latter forms of image compression are termed "lossy" because some of the original image details are lost during the compression. In the case of JPEGs and MPEGs, however, it is possible to specify different tradeoffs between image size and image resolution; you can pick how lossy you want the final compressed image to be. A final method of image compression is called palettization or indexed color: instead of having an infinite number of colors available, you pick a small subset (the palette), give each palette color an arbitrary number (the index), and map every pixel in the image to the index of the color in the palette which is closest to the original color. Typical GIF87A color images use this method of size reduction: part of the file specifies a table of 256 different colors (in 8-bit or 24-bit format), and the image proper is just an array of indices 0-255; the program reading the GIF file makes the appropriate color substitutions from the palette as it draws the image on the screen. An interesting sidelight about GIFs: not only do they use palette color, but their image array fields are themselves compressed with the LZW algorithm, thereby achieving an even greater size reduction.

Last week I finally finished out the roll of film that had my ADAMcon VIII pictures on it and got it developed, so at last I had the photographs. This last Saturday night, after the kids were put to bed, I went into the lab to make image files. Here's how I did it:

I scanned the original photographs at 300 dpi 24-bit color and saved them as LZW-compressed TIFF files. TIFF was one of the first image file formats developed and is good for lossless images. The scanner software (Ofoto for Apple Power Macintosh) has some nifty features like prescan (to see quickly what you've got, to make sure it's lined up, etc.) and image cropping (you can specify an arbitrary rectangular subset of the prescanned image, so you can just grab the parts you're interested in). These images were pretty huge, most 3-6 megabytes in size, so they definitely would have to be compressed in some way for practical viewing and downloading on-line.

I opened each TIFF image in Adobe PhotoShop 2.5.1 for Macintosh. This powerful image-enhancement program allows you to do all kinds of photographic retouching, special effects, you name it--including color-correction. Often you will get some color differences in scanned images compared to the originals. The image might be too green, or not have enough red, etc. This is a variable for each individual scanner. Also, you might be scanning an old photograph whose colors have changed with age, and you'd like to restore the original colors, contrast, etc. PhotoShop has sophisticated tools to do color and contrast corrections. The scanner I used made the images come out darker than the original photographs and also a bit too red. I toned down the red and turned up the blue to compensate.

Next, I converted the images from 300 dpi to 72 dpi without changing the actual image dimensions. This is a lossy compression which reduced the TIFF files to about 150K each. (If I had gone from 300 dpi to 72 dpi without constraining the image dimensions, the images would just have been displayed larger on the screen, with no shrinkage of the data.) This size is tolerable for a single image or so on a web page, but if you want people to look at all the pictures, 150K is too big.

The final compression step was to convert the 72 dpi 24-bit color TIFFs to 24-bit color JPEGs. I selected 75% image quality, 25% image size as my lossage parameters (higher means more of that parameter), and the final JPEGs were about 15-20K. That means only 10-15 seconds of download time each, and that's acceptable.

This morning I put all the final JPEGs up for download and sent mail to Dale Wick that he could grab them at his leisure. So can you! All the files are in the `/ac08` directory and have the extension `.jpg`. The files named `ac01` through `ac08` are the eight ADAMcon convention banners (`basic` is the B.A.S.I.C. user group banner). The files named `banq01` through `banq05` are banquet table shots. The names of the rest are self-explanatory, except for `rfd-jcd`: that's a special treat picture of me and my wife Joan (am335@po.cwru.edu) from 1995. Let me know what you think of it `:-)`

I case you're wondering why I made JPEGs instead of GIFs, I tried making some GIFs and really didn't like the way they came out; they were really spotty and over-palettized, especially the banner pictures.

[entering soapbox mode]

Just a final philosophical note: I have never felt bound by the stricture that "A *true* ADAMite only uses an ADAM; he never uses an IBM or an Apple or anything else; if he can't do it with the ADAM, he shouldn't be doing it." I used to hear this frequently in the early 1990s. Throughout my ADAM career (dating from 1988), I have felt free to use whatever computing resources I felt could best help me get my work done. I have no qualms about using other platforms to get work done for my ADAM (editors, word processors, assemblers, disassemblers, etc.). In fact, if I had been limited to SmartWriter's editing capabilities, I would never have been able to disassemble EOS-5 and SmartBASIC 1.0, and I wouldn't know any of the things that I know now. If the scanner's hooked up to a Mac, use the Mac, don't lament that there isn't SmartScan 1.0 for ADAM and never be able to show others my pictures. Heck, the prospects for a SmartMosaic web browser aren't very bright, so you'll *have* to use some other computer to view the images.

[exiting soapbox mode]

See you again next week!

*Rich*

---

# This Week With My Coleco ADAM 9609.30

by Richard F. Drushel (drushel@apk.net)

## I. Games, games, games!

Now that I have a full-time user ADAM system up and running in my basement, my older daughters have rediscovered the joys of playing ColecoVision game cartridges. Christina (age 9.5) does most of the playing, Elanor (age 6.5) offers comments and criticisms, and Diana (age 3) just watches. Most of their time has been spent with Carnival, Choplifter, Antarctic Adventure, some Keystone Kops game, and (ugh!) Space Panic. About the only game that Elanor has managed to figure out is Carnival; the ducks still get her every time, though.

This weekend, they also spent time trying out various ColecoVision games using the ColecoVision emulator program, COLEMDOS.EXE, by Marat Fayzullin (fms@freeflight.com), which I demonstrated at ADAMcon VIII. (Marat's home page is http://www.komkon.org/fms/ColEm/, which has links to ports to Unix, Macintosh, Windows, MS-DOS, and other operating systems.) I printed out a listing of the \ROMS directory (which contains binary images of game cartridges) so they would know what file names to use, and let them go at it. Most of the games in the 116-game archive I have never seen before, so I was little help for how to play some of the games. Their favorites were Tapper, Mr. Do, and Ladybug.

Some of the games do not appear to work correctly under the emulator: Qbert (locks after the first time you die), Victory (can't steer/rotate the ship), a Dr. Seuss puzzle game (can't rotate puzzle pieces if they get scrambled upside-down). I don't know if this is because of limitations in the emulator, or if the ROM images are bad. If I owned these cartridges, I could make my own ROM image and try them out.

ROM images...the vortex of copyright questions. It's very likely illegal to redistribute ROM images without consent of the original copyright holders. This means that there's no way to put this ROM archive in a conspicuously public place (like CompuServe) without getting someone in lots of trouble. At ADAMcon VIII, I discussed with Pat Herrington how to get around this so that she could safely use CompuServe to redistribute this emulator. The ColEm emulator itself can be freely redistributed--the source code is available for anybody to modify. (Hmm, in my copious spare time, it would be fun to make this into an ADAM emulator.) As for the ROM images, my suggestion to Pat was that a copy-cartridge-to-160K-disk program be written in SmartBASIC, and this program, along with Chris Braymen's ADAMDOS program (to read 160K ADAM disks in PC disk drives), be bundled with the emulator. This way, you can make ROM images of all the cartridges you own, for your own personal use, and not have to worry about copyright issues. I have a SmartBASIC 1.x program already which can copy the cartridges, but the distribution program really needs to be in vanilla SmartBASIC, `PEEK`s and `POKE`s and `CALL`s and all, since the latter is the only software that we may expect every ADAMite to own. If someone already has a program which can copy a game cartridge to a binary file (no headers, no fancy stuff), please speak up.

## II. Rescuing a laptop.

In September 1990, I bought a Tandy 2800HD '286 12 mHz laptop computer, with EGA-resolution LCD screen and internal 2400 bps modem, for the purposes of writing my Ph.D. dissertation. I also used it as a disassembly platform for SmartBASIC, ADAMlink IVa, PowerPaint, HARDDISK, the Coleco Graphics Processor, and many other programs which exist only as binary images. With the use of a CP/M emulator, Joan Riff's Z80MU.EXE, I could also run CP/M and TDOS programs (mostly uncrunchers and delibrary programs to deal with files I downloaded from George Koczwara's (aa436@po.cwru.edu) BBS, or ftp'ed from the famous, defunct Simtel archive (wsmr.simtel20.army.mil) at the White Sands Missile Range in New Mexico. With WordPerfect 5.0 as my editor and a DOS- based cross-assembler, I could write new Z80 code and assemble it to binaries; I could then either null-modem transfer it to an ADAM, or upload it to temporary storage on the Cleveland Freenet and download it to an ADAM via ADAMlink IVa. This machine served me well (you may remember seeing it at ADAMcon IV), and I took it everywhere there was ADAM code hacking to be done. (I also successfully completed my dissertation with it.)

Unfortunately, laptop computers of that vintage are subject to LCD screen wearout. Specifically, the fluorescent backlighting becomes dimmer and dimmer with time. There is a brightness adjustment to compensate, but eventually you can't get it bright enough to see what's on the screen. This happened to my laptop about 2 years ago. The only way to see what was on the screen was to sit either in a dark room (so you couldn't see they keyboard until your eyes got dark-adapted) or in direct sunlight. Needless to say, these were not optimal conditions.

The fatal blow to this laptop came when my daughter Elanor spilled a glass of milk onto the keyboard. I disassembled the computer and tried to rinse out the keyboard, but it was no use: the milk had gotten inside some of the "sealed" keyswitches and these keys no longer worked.

A $2500 piece of junk now, right? Not yet: fortunately, the Tandy designers had left some room for expansion. On the back of the laptop was an external keyboard port and an external color monitor port (DIP-switch selectable for either CGA or EGA). So, with the addition of an external keyboard and an external monitor, I could have a working computer again. In fact, even though a '286-12 is underpowered for any of my current PC needs, it would be just fine as a dedicated ADAMserve server--it has an external serial port, and its 20MB hard drive is plenty of space for a 10MB EOS partition.

So, I decided that I would pull the motherboard out of the laptop case, leaving behind the dead internal keyboard and dying LCD screen. I would mount the motherboard, hard drive, 1.44MB disk drive, and external power supply into a single box. This would then be a compact, ADAMserve-ready computer, which could be used with one of my spare ADAMs. Better to recycle the computer than to junk it, eh?

To achieve a small size, laptop computers use proprietary-design internal components, which do *not* fit in "standard" cases. Not only would I have to come up with a case, but I'd have to figure out how to mount it using all those nonstandard holes. Fortunately, the motherboard and power supply board of this laptop came out as a nice unit. I could set the board onto a sheet of thin plywood, trace the screw holes, drill them, and mount the board using aluminum sleeves as standoffs. (The board should have an airspace underneath it to allow for cooling.) As for a case, I happened to have a nice 2-piece ventillation-louvered plastic box, courtesy of a former member of B.A.S.I.C., George Harpster. Long ago, George (who worked for the NASA Lewis Research Center here in Cleveland, recently renamed NASA Glenn after Ohio-born astronaut John Glenn) got several of these boxes from cast-off equipment. He used one to build a custom case for his ADAM, which included an expansion bus of sideport connectors for plugging in multiple sideport devices. I got one of these boxes with the intent of doing the same someday, but never got around to it. I decided that it would be good to put the laptop guts into this box. One nice thing about the box was that the front and back panels were open: I could have stuff which stuck out the front and back without having to cut access holes in the box.

With little effort I cut a 1/8-inch plywood base, mounted the laptop motherboard and power supply, and installed it in the lower half of the new plastic case. The back of the motherboard had all the external jacks for keyboard, monitor, serial port, parallel port, and modem, and these faced the rear opening of the plastic case. To accommodate the hard drive and floppy drive, however, I had to build a shelf, because the box wasn't wide enough (front-to-back) to accommodate the drives on the same level as the motherboard. The shelf consisted of 2 1-inch tall pillars cut from a 2x4, and another piece of 1/8-inch plywood. The hardest part was drilling holes in the wood to match those in the bottom of the drives; I used paper templates to transfer the hole locations. The last task was to relocate the on/off switch to the front; I bought a big red switch and mounted it on the drive shelf.

The finished project has all the access ports in the back, and the floppy disk drive and power switch in the front. I didn't bother to fill in the remaining open panel spaces, because it would have been too much trouble. If I may brag, however, the new computer guts were fit together in such a way that no drilling, cutting, or alteration of the plastic box was necessary. If I ever decide to put something else in that box, the laptop guts will come out as a complete unit.

As for the cast-off laptop case, I took the LCD screen out, but reassembled everything else, including the dead keyboard, and gave it to my kids to play with `:-)`

Now I have to finish ADAMserve `:-) :-)`

## III. Making an ADAM disk image.

Ron Mitchell (ronaldm@mars.ark.com) posted to the mailing list last week that he couldn't get ADAMserve to boot. He'd gotten the latest version from Herman Mason (aa337@po.cwru.edu) at ADAMcon VIII and was desperate to try it out. From his description, it sounded as if Herman hadn't given him an ADAM boot disk/tape. So, I blithely offered to send (electronically) an image of a boot disk/tape.

First I'll give your the "approved" method. Then I'll tell you all the contortions I actually had to go through.

On paper, there's a simple procedure to create a binary image of an ADAM disk that is itself a stand-alone file (which can be copied, edited, etc.). Minimally, you need CP/M or TDOS and a utility program, IMAGE2.COM, which can save any desired number of consecutive blocks from an ADAM disk as a file, starting from any desired block number. The resulting file is called an image file and has the extension .IMG.

Disk images can be huge. If you're going to E-mail it to someone, there can be problems (due to technical reasons relating to how Internet E-mail is managed) if the message is bigger than 64K. Thus, for safety reasons, it's best to send a *compressed* version of the image. The recipient can *uncompress* it himself. The most common compressor I've seen ADAM TDOS people use is called CRUNCH.COM, which creates a compressed file of the extension .?Z?, where ? is any character; the Z means it's crunched. So, by the book, I would crunch the image file to yield something with extension .IZG.

Now I have to get it to the Internet. This means I'll have to upload it by telephone/modem to my local Internet Service Provider (ISP), and then use Internet tools to E-mail it. Using my favorite TDOS modem program (IMP, MODEM7, whatever), I dial into my ISP, initiate an XMODEM upload of my crunched image file, and send it.

Although things are changing slowly, Internet E-mail is still a 7-bit ASCII world. If you send 8-bit binary data, you run the risk that, at some point, all the high bits will be stripped out and your recipient will be left with a 7-bit version. Thus, you cannot directly E-mail a binary file as-is. It is possible, however, to encode an 8-bit binary file using only 7-bit ASCII characters, making it safe to transmit via Internet E-mail. The program to do this is called `uuencode`, and there's a reciprocal program called `uudecode` to convert a `uuencode`d file back into its 8-bit binary form. My ISP has `uuencode` available, so I `uuencode` my crunched image file, by convention creating a file with the extension .uue.

*Now* I can E-mail the file. I use my favorite mailer (elm, pine, mail, Eudora) to compose a message. I include the text of the `uuencode`d crunched image file, then send it off. My recipient must undo all of the steps in reverse order: he has to save the message to a file, `uudecode` it, download it to a TDOS disk, uncrunch it with UNCRUNCH.COM, and finally copy the image to an EOS disk ("clone" the disk, using CLONE5.COM). If my recipient did everything right, he should be able to put the final cloned disk into a drive and either boot it or be able to read/write it under the appropriate operating system (whatever was used to create it, EOS or CP/M or TDOS).

"The best-laid plans of mice and men gang oft agley." Here's what I *really* had to do to send Ron Mitchell his ADAMserve boot disk:

Historically, I've done very little with CP/M and TDOS. I don't have anything against them, I've just always had so much EOS work to do that I've never gotten around to the others. Thus, most everything I've ever had for CP/M and TDOS was on the TDOS partition of the Mini Wini hard drive I bought from HLM-GMK. Due to a recent power supply failure, however, my Mini Wini system is dead, and I haven't had time to troubleshoot it. Thus, all those helpful TDOS programs I mentioned above were sitting inaccessible.

My first purchase from Steve Major's ADAM Connection, back in 1991 or so, was ADAM CP/M 2.2. I knew I had made a backup copy of the original for use as a working disk, and I vaguely recalled that I had some TDOS-type utility software on that disk at one time or another. But where in my basement mess was that disk? After some scrounging, I located it, booted it, and saw that it had both IMAGE2.COM and CLONE5.COM. Alas, no CRUNCH.COM, and no modem program. Great, I can image the ADAMserve boot disk, but then what?

While scrounging, I also came across an ADAMlink IVa boot disk. This was one of my working disks while revising IVa to create ADAMlink V. This would do XMODEM transfers. So, if I could get the .IMG file from a CP/M disk to an EOS disk, I could at least get it up to my ISP, where I could further manipulate it and then send it to Ron. How to do the CP/M-to-EOS transfer?

Since I was working with a copy of the ADAM CP/M 2.2 boot disk, it contained the 2 utility programs CPMADAM.COM and ADAMCPM.COM, which are used for CPM-to-ADAM and ADAM-to-CPM transfers, respectively. Hooray for ADAM CP/M 2.2! These 2 files are *not* on any of the default TDOS boot disks, so far as I have seen.

In my collection of ADAMserve boot disks, however, I could not find one that was 160K media. ADAM CP/M 2.2 (unpatched) is limited to 160K disks, and all I could find was 720K and 1440K media. This meant that I had to boot EOS FileManager with a 1440K/160K dual-drive setup, copy all the necessary blocks to a 160K disk, then change to a dual-160K drive setup and reboot under ADAM CP/M 2.2. Arrgh!

Now that I had the ADAMserve EOS boot disk on appropriate media, I imaged the disk with IMAGE2.COM, then copied the ASBOOT.IMG file to an EOS disk with CPMADAM.COM. After setting up the nice serial board/2400 bps modem pair that I won as a door prize at ADAMcon VIII (thank you HLM-GMK), I booted ADAMlink IVa, dialed up my ISP, and sent ASBOOT.IMG by XMODEM.

I still needed to compress the binary file before uuencoding it, because the binary itself was 54K, and the encoding process makes it about half again as big. I thought about using the Unix `compress` utility, but I wasn't sure if Ron Mitchell had access to Unix `uncompress` or not. The same for `gzip/gunzip`. Rats, the only thing to do is download it to my PC, use DOS PKZIP.EXE to compress it, then upload it back to my ISP... which is what I did.

Finally, ASBOOT.ZIP in hand, I was able to `uuencode` it and mail it out to Ron, with a cover letter explaining what he had to do to resurrect the original image, and a note that the details of how his file had come to pass would be presented in my next This Week With My ADAM column.

As Paul Harvey says on the radio, "And now you know...the rest of the story."

Tell me, Ron, was it worth all the trouble? `:-)`

See you again next week!

*Rich*

---

Hi gang,

I'm going to try to write a weekly capsule summary of what I've done with my ADAM, no matter how small. The purposes of this are:

- to generate traffic for the mailing list;
- to generate articles for use by newsletters/A.N.N. disks;
- to force me to do something every week with my ADAM `:-)`

Anybody who likes is free to join in with his own periodic activity reports. Anybody who likes is free to redistribute my reports anywhere he thinks they will be seen by ADAMites.

---

# This Week With My Coleco ADAM 9609.22

by Richard F. Drushel *(drushel@apk.net)*

## I. Making Space For ADAM.

For the last few years, our users group, Cleveland B.A.S.I.C., has had its monthly meetings in my basement. As I have added computers, tools, and children to my collection, the basement has gotten more and more crowded, and more and more disorganized. This summer it finally reached the point that it was too hard to clean it up in time for a meeting, and I asked for some temporary changes of venue. The task of packing everything up for ADAMcon VIII was the last straw--the clutter was intolerable. When I returned from ADAMcon VIII, I left everything packed up in the boxes in preparation for a major cleanup.

One difficulty we've had with our B.A.S.I.C. meetings has been having working ADAMs set up for easy use. I have 4 complete ADAM systems, but usually the only one up and running was the one I was debugging ADAMserve on, and it was frequently in pieces or surrounded by printouts and other junk. A major objective of my basement cleanup was to create a space for user ADAMs that could be permanently set up and left alone, independent of my ADAM development system.

Another consideration is that my two oldest daughters, Christina (age 9.5) and Elanor (age 6.5) now have nightly homework assignments, and the dining room table isn't big enough for both of them to do their homework at the same time. I wanted to put some school desks in the basement so they could do their homework undisturbed. As for desks, I managed to get 2 complete study carrel sets (table, bookshelf, and chair) from the Sears Library here at Case Western Reserve University. CWRU has just built a new, centralized Kelvin Smith Library, and is demolishing two old peripheral libraries, the aforementioned Sears Library (on the old Case Institute of Technology campus) and the Freiberger Library (on the old Western Reserve College campus). All the furniture, bookshelves, etc. from these two libraries was free for the taking; what wasn't taken was to be junked. Naturally, there was great interest in all the free desks and chairs...I don't know about Freiberger, but Sears was stripped bare in about 4 hours. The two study carrels I got are not your modern-day sawdust-and-glue construction, but are all solid hardwood, and should survive anything short of a direct nuclear strike. These have now been installed, along with some new indoor/outdoor carpet and fluorescent lights; they are also available as workspaces during B.A.S.I.C. meetings.

I'm only about half-done with the cleaning/reorganization, but I've managed to get one complete ADAM system set up on its own table, with its own surge-protected powerstrip and overhead fluorescent lights. It's a basic R80 system, 2 tape drives, 1 1.44MB floppy drive, and a color TV set. I still have to set up my ADAMserve development system. I also have to move one of those assemble-it-yourself computer desk/hutch combinations from upstairs to the basement. I'll set up another ADAM user system on that.

## II. An ADAM Printer Problem And Its Solution.

While setting up the first R80 system (to be Christina's computer), I found a problem with the ADAM printer. The ribbon wasn't advancing when characters were typed, so the printing was bad. Remembering what Rich Clee had said during his Q&A session at ADAMcon VIII, I (for the first time) ventured to open up a Coleco printer ribbon. I didn't find anything unusual or which seemed to be jammed, so I put it back together and tried it again. Still no advance. I tried some brand-new Coleco ribbons; still no advance. Other ADAM printers didn't have any trouble advancing these ribbons. Hmmmm.

Upon close inspection, I noticed that the cross-headed ribbon advance pin did not fit snugly into the ribbon cartridge in this particular printer. Indeed, it seemed to me that the advance pin didn't really stick up far enough to catch. I tried taking off the cartridge plate and reseating it, but still that pin didn't stick up far enough to engage the cartridge, unless I held it down with my finger while it was typing. Unfortunately, there isn't any way to pull that pin up higher. What to do?

Staring at the cross-headed pin, something looked familiar. Where have I seen something like that before? Some of you may have heard that I teach a robotics course which uses LEGOs as the robot skeletons. *That* was it--a LEGO axle! I was sure that the crossed end of a black LEGO axle was the same as the crossed end of that ribbon advance pin. So, I made a quick trip to the robot lab and returned with a length of axle. I was right, it would fit, but it was just a tad larger than the original advance pin, so instead of easily fitting into the cartridge, it fit rather snugly. But it was very close.

My solution:

- sand down the LEGO axle so that it was the same diameter as the ribbon advance pin. A little work with some 100-grit sandpaper and it was done.
- cut off a thin cross-sectional slice of the axle (clamp it in a vise, use a hacksaw with a fine-toothed blade). "Thin" = about 2 millimeters or 1/16 inch. Clean up any burrs with sandpaper.
- flatten down the end of the ribbon advance pin. As it is, it's a little rounded; use the sandpaper to make it flat and to roughen it up a little.
- use Super Glue to attach the axle slice to the end of the ribbon advance pin. If you have bad eyes, make sure you use a magnifier. Forceps may also help to position the axle slice. You have to get it centered on the pin and get the crosses lined up perfectly *before* the glue sets! If you make a mistake, you have to pry off the axle and sand/scrape off the dried glue before attempting to reglue it.
- wait overnight to make absolutely sure that the glue is set.
- put in a ribbon cartridge, *gently* seat the cartridge over the ribbon advance pin, turning the advance knob clockwise as necessary to get it to seat correctly. Start typing and make sure that the ribbon is advancing with every 2nd character typed.

I have no real experience with printer problems. Is it possible that this is the source of difficulties with Coleco ribbon cartridges that are past half-used? Since the ribbon advance pin is so short, and since the front clamp doesn't really hold the cartridge down onto the plate very well, is it possible that, when the ribbon tension gets higher, the pin just slips out of the cartridge? If so, this modification might be of more general interest.

As for different types of glue...I used the standard Super Glue which is a runny liquid and sets very fast. I have never liked the Super Glue Gels and have no idea how they'd work here. Five-minute epoxies might be worth a try, but in my experience, they gradually absorb moisture from the air and, in about a year, get soft and break loose. Two-ton epoxies don't have this permanency problem, but they take several hours just to set. Your mileage may vary.

Well, that's it for this week. See you next week!

*Rich*

---