All you ever wanted to know about: Atari Cartridges

Table of Contents

- All you ever wanted to know about: Atari Cartridges
- PART 1 RAM/ROM Control On An XL/XE Computer
- AUXILIARY ROM CONTROL
  - Atari-type Cartridges
  - OSS Super Cartridges
  - OSS Cartridge Examples
  - The SpartaDOS X Cartridge
  - The R-Time 8
  - The Multipllexer! Operating System (MUX)
  - Cold and Warm Starting and Parallel Devices (PD's)
- LATE NOTES
- LATER NOTE
- Part 2 - THE OSS SUPERCARTRIDGEs
- Part 3: Cartridge slot descriptions
  - Left slot pinout (all models) ARD4if cartridge maps to $8000-$9FFF area
  - Right slot pinout (only 800) AB-PHI2 1R/W
- Cartridge basics
- OSS carts
- SDX carts
- R-Time 8
- XEGS carts
- XEGS demo cartridge
- Telelink 2 cartridge
- JRC cartridges
- ATRAX cartridge
- Bounty Bob cartridge
- 128/256 K RamCart
- A/D Converter
- MPP Super Charger
- PILL carts
- Williams multibanked cartridge
- Thompson Proburner
- COS32
- Part 4: Converting games to Cartridge
- Universal Cartridge Copy / Run routine

This is a big collection of info-texts, that all deal with Atari cartridges. Although I did not write any of these texts, I collected them and now present them here for you. There are four different texts in total, thus you can read four different parts - ranging from general information (RAM/ROM control) to more and more specific information and finally some programming hints (and source!) on how to convert game files into Atari cartridges...

PART 1 RAM/ROM Control On An XL/XE Computer

by John Picken
AUXILIARY ROM CONTROL

Program control of Atari BASIC, OSS Super Cartridges, the R-Time 8 cartridge and, in a limited fashion, SpartaDOS X, is fairly simple but for one fact: there's nearly no documentation available on the subject. What I present here is gleaned from a bit of disassembly and a lot of experimentation (pretentious word for "try it until it don't crash").

Note that all following references to a cartridge being "present" imply that it is turned on if it's plugged in but turned off, consider it "absent". You may always consider the RT8 to be absent unless you're actually trying to access it. First let's look at the addresses used for, or in conjunction with, auxiliary ROM/RAM control. PORTB has already been covered; just keep in mind the function of the BASIC bit.

BASICF is a flag in low memory to tell the OS, on system reset, how to set bit 1 of PORTB. If this flag contains any non-zero value the BASIC ROM will be disabled.

TRIG3 is an address on the GTIA chip which was used for joystick trigger #3 on the 400/800. On the XL/XE it is a cartridge status indicator; if a cartridge is present it reads 1, otherwise it will be 0. There is no other possible reading at this address.

GINTLK is set, on boot, by the OS and is a copy of TRIG3. The OS compares GINTLK with TRIG3 during the deferred vertical blank interrupt and, if the two don't match, goes into a "soft" lockup (i.e. a reset will re-boot).

CARTCK holds a checksum, calculated on boot, by the OS. On a reset, if a cartridge is present as signalled by TRIG3, the OS re-calculates the sum and compares it with CARTCK. If the two don’t match, the OS assumes you've pulled or inserted a cartridge and immediately re-boots. Note that Mapping the Atari is vague on this: it applies to all XL/XE's, not just the 1200.

CARTCK, TRIG3 and GINTLK are effective for all cartridges except (in part) the RT8. One other important thing to note is that the TRIG3/GINTLK comparison occurs during the deferred vertical blank. This means you can fool around with a cartridge to your heart's content as long as the stage two vblank doesn't occur and you don't hit Reset. You can prohibit vblank2 in any of three ways: disable all NMI's, set CRITIC to a non-zero value or, most simply, use a SEI opcode.

The hardware address range for all cartridge control is $D500-$D5FF. Within that page, OSS cartridges use $D500-$D50F, the CSS MUX OS uses $D570-$D57F, the RT8 uses $D5B8-$D5B9, and SDX uses $D5E0-$D5EF. This sounds straightforward; unfortunately it isn't.

Atari-type Cartridges

I made no mention of Atari cartridges in the address ranges because once you stick one of the beasts into the slot, your only control over it comes with the power switch or by using SDX. An Atari cartridge can not be turned off by software unless SDX is present (even if turned off). SDX can control one because it sits between the computer and the cartridge and can, thereby, zap it electronically. However, the foregoing discussion of TRIG3 and GINTLK remains fully applicable.

OSS Super Cartridges

Though the control range is $D500 to $D50F, the cartridge address decode logic is only four bits wide. This means that any access (read from, write to, or otherwise manipulate) a $D5xy address affects the cartridge which ignores the "x". OSS cartridges react to the whole $D500 page based on the low four bits of the address.
To enable an OSS cartridge bank, add the bank number to $D500 and access that address (i.e. for bank n, STA $D50n, LDA $D50n, STA $D500,X where the x register holds n, etc.) In theory, a cartridge should be able to contain up to sixteen 8k banks and still allow you to turn it off. In practice, they contain two or three switchable 4k banks and one "master" 4k bank.

For OSS cartridges, the ROM bank number is found at location $AFFF. Valid values at $AFFF are 0, 3 and 4 for Action! and 0, 1 and 9 for MAC/65. Other bank values produce varying results. MAC/65 ignores bits 1 and 2 so any value from 0 to 7 results in selection of either the odd or even bank. With Action! attempts to select other bank numbers result in selection of one of the real ones or in selection of nothing i.e. a monitor shows a pile of $AF's in the $AF page just as when you examine page $D7 and get $D7 at all addresses. BASIC XE has banks 0, 1 and 9 but bank 9 is RAM. In bank 9, the BXE cartridge is off but TRIG3 stays high; a sneaky way to avoid having to worry about GINTLK while using the RAM under the cartridge.

There are two constants for cartridges: Addressing bit 3 alone turns cartridge ROM off and, bank 0 is the bank in which the cartridge boots and initializes. Here are "maps" of the banks in two cartridges:

**MAC/65**

```
0123456789ABCDEF: Bank Selection
rrrrrrrr r r r r: r=rom, empty=ram
01010101 9 9 9 9: bank #
01       9      : valid rom banks
```

**Action!**

```
0123456789ABCDEF: Bank Selection
rrrrrr r        : r=rom, empty=ram
000340 3        : bank #
0 34           : valid rom banks
```

**OSS Cartridge Examples#**

Following are several examples of cartridge and BASIC control with SDX not present. I'll start with equates for all examples from Mapping the Atari (XL edition):

```
WARMST = $08
BOOT?  = $09
CRITIC = $42
RAMTOP = $6A
COLDST = $0244
CARTCK = $03E8
BASICF = $03F8
GINTLK = $03FA
TRIG3  = $D013
PORTB  = $D301
NMIEN  = $D40E
EDITRV = $E400
```

Here's the simplest: turn off a cartridge and enable BASIC assuming both are actually present.

```
SEI         Kill stage 2 vblank
STA $D508   Kill any cartridge
LDA PORTB
AND #$FD   Drop basic bit
STA PORTB
LDA TRIG3   This should be 0
STA BASICF Flag it and
```
STA GINTLK  correct the cart shadow
CLI          Enable stage 2 vblank

Now let's access RAM under a cartridge:

SEI         Kill stage 2 vblank
LDA PORTB   Save Portb
PHA          Kill basic rom
OR #$02     STA PORTB
LDA #$08    Assume no cartridge.
LDX TRIG3   Check assumption.
BEQ GOTBNK  Go if none or off,
LDA $AFFF   else get the bank
GOTBNK
PHA          Save cartridge bank.
STA $D508   Kill any OSS cartridge
LDA TRIG3   Set the shadow before
STA GINTLK  the stage 2 vblank!
CLI          Enable stage 2 vblank

Do whatever in the RAM, then restore the previous status:

SEI         Kill stage 2 vblank
PLA          Recover cartridge bank
TAX          and restore the cart
STA $D500,X to it's prior status
LDA TRIG3   Reset Gintlk to
STA GINTLK  correct status
PLA          Restore prior Basic
STA PORTB   rom status
CLI          Enable stage 2 vblank

Just turning a cartridge off is simple:

SEI         Kill vblank 2
STA $D508   Enable cart bank 0
LDA TRIG3   Make sure it went
STA GINTLK  off and flag it
CLI          Enable stage 2 vblank

Cold-starting an OSS cartridge is only slightly more complex:

SEI         Kill vblank 2
STA $D500   Enable cart bank 0
LDA TRIG3   Set the shadow
STA GINTLK  correctly
LDA PORTB   PHA
ORA #$01
STA PORTB   Ensure OS is on
CLC          Calculate the
LDX #0      checksum
TXA          for reset.
CSLOOP
ADC $BFF0,X Note the sum
INX          includes the
BNE CSLOOP  first 240 bytes
STA CARTCK  of the OS ROM.
After cold-starting an OSS cartridge or BASIC, set WARMST to 0 to flag a boot so that the buffer
pointers are cleared (if you don't, you can, for example enter BASIC, type LIST and get an endless
display of zeros and/or a lockup). Then initialize the ROM. With Action! and BASIC this doesn't matter
as the initialization routines just RTS; with BASIC XE I'm not sure; with MAC/65 it's required. To
initialize any cartridge:

LDX #$FF Say we're on a boot
STA COLDST and make sure all
INX flags reflect this
STA WARMST
INX
STA BOOT? Go do it
JSR INIT
LDX #$FF Say we're back to
STA WARMST normal status, i.e.
INX what happens on Reset.
STA COLDST Note that some or all
INX carts play with some
STA BOOT? of these flags!
RTS

INIT
JMP ($BFFE) Cartridge init vector

After enabling or disabling a cartridge or BASIC, you also have to ensure top of RAM and screen
pointers are correct. To do this, execute a "GRAPHICS 0". In machine language terms, you set
RAMTOP and then close and re-open channel 0 to the "E:" device. You can do this in the traditional
manner via CIOV or more simply by calling the following subroutine with the accumulator holding $C0
if turning ROM off and $A0 if turning it on.

GRAPH0
STA RAMTOP Either $A0 or $C0
LDX #0 Indicate channel 0
LDY #2 Point to Close vector
JSR EDO
LDY #0 and now to Open vector
EDO
LDA EDITRV+1,Y
PHA
LDA EDITRV,Y
PHA
RTS

Be aware that turning off BASIC XE does not free up the RAM under the cartridge if you intend to
later restore the cartridge. BXE uses that RAM as well as that under the OS floating point routines
and also (undocumented) sets an interrupt vector in the last page of RAM ($FFFx).

One final note on turning ROM off or on: following the Graphics 0, an RTS under Sparta will usually
lock up the keyboard requiring a reset. Sparta installs its own E: handler so when you use the OS
handler to reopen E: Sparta's vectors are no longer valid. The simple way around this is to exit via a
JMP (DOSVEC). This is probably a good idea with any command-processor FMS where you can use
a batch file instead of an autorun to set things up.
The SpartaDOS X Cartridge

SDX boots in bank 0 but normally works in bank 1 with one subroutine call back to bank 0 via low RAM which I suspect is used to load files from CAR: The cartridge contains eight different ROM banks (0 to 7), but I have not discovered any single location containing a bank identifier and I doubt there is one as its existence essentially would mean a "hole" in the middle of each ROMdisk bank. The control address for the X cartridge is $D5E0 used similarly to $D500 with an OSS cartridge.

The following code will leave the currently selected bank in the Y register with version 4.20. With other versions, you’re on your own.

LDA $A004
LDY #7
LOOP
CMP XBANK,Y
BEQ GOTIT
DEY
BPL LOOP
LDY #$01    Can't figure, make it 1
GOTIT
RTS

XBANK
.BYTE $32,$1F,$02,$1D
.BYTE $D4,$1C,$61,$56

The ROMdisk directory is at the beginning of bank 2 and follows normal Sparta format except that the address of the first sector map is the actual starting location in ROM of the stored program. The first two entries look like this:

`.BYTE $08   Status: In use
.WORD 16384 Start: Bank 2 Offset 0
.WORD 598   Length
.BYTE 0     Length (high byte)
.BYTE "MAIN      "
.BYTE 1,1,70 Date
.BYTE 251,0,0 Time
;
`.BYTE $08   Status: In use
.WORD 16982 Start: Bank 2 Offset 598
.WORD 7288  Length
.BYTE 0     Length (high byte)
.BYTE "SPARTA  SYS"
.BYTE 6,2,89 Date
.BYTE 15,28,40 Time
;

To convert the starting address to a bank and offset within the bank:

\[
\begin{align*}
\text{bank} &= \text{int}(\text{address}/2000) \\
\text{offset} &= \text{address} - 2000 \times \text{bank}
\end{align*}
\]

CAR.COM uses the following combinations to control all three ROMs in the cartridge area. The values under "SDX" and "OSS" are offsets from $D5E0 and $D500 respectively and those under BAS are the value in bit 1 of PORTB. The "on" under the OSS column is the value found at $AFFF before the cartridge was last turned off and used to reenable it. The $0C value for SDX is what
causes it to latch any cartridge off and the $08 makes it transparent so that the cartridge ROM is accessible.

<table>
<thead>
<tr>
<th>SDX</th>
<th>OSS</th>
<th>BAS</th>
<th>OPERATING CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>$08</td>
<td>1</td>
<td>in DOS or low RAM</td>
</tr>
<tr>
<td>$0C</td>
<td>$08</td>
<td>1</td>
<td>in high RAM (X.COM)</td>
</tr>
<tr>
<td>$0C</td>
<td>$08</td>
<td>0</td>
<td>in BASIC</td>
</tr>
<tr>
<td>$08</td>
<td></td>
<td>1</td>
<td>in cartridge</td>
</tr>
</tbody>
</table>

If you're going to play with SDX banks remember that any read or write to a $D5Ex address will affect an OSS cartridge and TRIG3. Since TRIG3 is affected, GINTLK and CARTCK also come into play. So the example given of how to access RAM under a cartridge needs to be modified if SDX is present. Let's look at a subroutine to access RAM in the cartridge space taking in the possibility of the presence of BASIC, SDX, or an OSS cartridge.

```assembly
ROMCTL
  LDA PORTB
  PHA
  ORA #$02    Any Basic rom off
  STA PORTB
  LDA RAMTOP  This might be easy
  CMP #$A0+1
  BCC NOLUCK  Not quite.
  PLA         See note a. following
  STA BASICF  for an explanation
  JMP DOSTUFF

NOLUCK
  LDA TRIG3   Is a cart present?
  BNE CART    Yes, go.
  JSR DOSTUFF Still fairly easy
  PLA
  STA PORTB
  RTS

CART
  PLA         See note a. following
  STA BASICF  for an explanation.
  SEI         Kill stage 2 vblank
  LDA $AFFF   Get OSS bank number
  PHA         Save it
  STA $D5E8   Turn off both carts
  JSR DOSTUFF

We know a cartridge was on. Now we have to restore it correctly.

  PLA         Recover bank number
  CMP #$10    Valid for OSS? (note c.)
  BCS SDX     No, must be SDX
  TAY
  STA $D500,Y Restore OSS cart bank
  BCC CARXIT  Go always
  SDX
  STA $D5E1   Enable SDX normal bank
  STA $D508   Kill OSS cart (note b.)

CARXIT
  CLI
  RTS
```
a. The reason for discarding the PORTB entry value is to allow for 512k+ RAM expansions. As mentioned previously, the OS doesn't know extra RAM exists and has no way of knowing BASIC may not exist on large upgrades. As a result, it sets PORTB and BASICF based solely on the Option key at boot and uses BASICF to determine which status to restore on a reset. On large RAM upgrades this leads to major problems for programs using extra RAM as the program can end up in the wrong 256k bank. Unless BASIC is actually on, it is always advisable to flag it off and to set its bit high in PORTB.

b. We knew a cartridge was on or we never would have got to that portion of the code. As it wasn't the OSS cartridge, it had to be SDX. But, because the dumb OSS cartridge reacts to the $D5E1 address, we had to turn it off again after enabling SDX. For the same reason, a single access of $D5E8 was sufficient to turn both off.

c. The comparison of the bank number to 16 to determine its validity as an OSS bank number is that used by ICD in the code for the RT8 handler. The test is, I believe, made on the assumption that the X cartridge is in bank 1 where the value at $AFFF is 87 for version 4.20. There are two SDX banks where values less than 16 are found at $AFFF, namely 0 (value 7) and 4 (value 3).

The R-Time 8#

Control of the RT8 is built into all versions of Sparta from 3.2 on. As far as I know, all you can do with the RT8 is set or get time and date information. The only problem in doing this is that accessing the RT8 registers will affect an OSS cartridge. Because of this the RT8 has two identical user accessible registers $D5B8 and $D5B9. According to the RT8 source, addressing $D5B8 will turn off a cartridge and $D5B9 will turn one on.

The RT8 has seven internal registers which work in binary coded decimal. Starting from #0 they are: seconds, minutes, hour, day of month, month, year, and day of the week (#6). Seconds and minutes range from 0-59, hours from 0-23, day from 1-31, month from 1-12, year from 0-99. Day of the week ranges from 0 (Saturday) to 6 (Friday). When you read or write one of these registers the sequence is always the same:

1 Wait until the RT8 is not busy. 2 Store a value from 0 to 6 into $D5B8 or $D5B9 indicating the register you wish to address. 3 Read/write the same address to get/set the most significant digit (the low four bits are the valid data). 4 Read/write the same address to get/set the least significant digit (the low four bits are the valid data).

The source code released by ICD indicates that reading a register should be repeated up to three times accepting two values that match or, failing a match, the first one. When setting a register, it recommends reading it immediately afterward to ensure the value was really accepted and allowing 10 tries.

Here's one way to read and write RT8 registers without worrying about an OSS cartridge. Much of this is from the source released by ICD. In this example, the buffer is set up in the same order as the RT8 registers. With Sparta, you would have to cross refer to the order in which DOS saves time and date and keep a separate byte for day of the week.

```
* = $F0 Floating point zero page
TEMP1
  * = *+1
TEMP2
  * = *+1
RETRY
  * = *+1
BUFFER
```
SEI         Kill vblank2
LDA $AFFF   Get any cart bank
CMP #$10    Is it valid?
BCC SAVBNK  Yes go, else use
LDA #$08    the "off" value.
SAVBNK
PHA         Save cart bank

Verify rt8 present and working

JSR READ    Get seconds
CMP #60
BCS GLITCH  if >59 then error
STA TEMP1
LDA RTCLOK+2
ADC #90
WAIT
CMP RTCLOK+2 Wait about 1.5"
BNE WAIT
JSR READ    Read again
CMP TEMP1   Same as last?
BEQ GLITCH  Yes, not working
SEC
SBC TEMP1   Ensure <3
BCS CHECK3  It is
ADC #60     else did it roll over?
CHECK3
CMP #3
BCC RT8OK   Yes, rt8 is ok
GLITCH
LDA # <RT8ERR Set for error message
LDX # >RT8ERR
EXIT
STA ICBAL
STX ICBAH
LDA #9      Print to eol
STA ICCOM
STA ICBLH   Plenty of length
PLA
TAX         Restore cart bank
STA $D500,X
CLI         restore vblank2
LDX #0      Select channel 0
JMP CIOV    Exit with message

First read the clock regs into the buffer

RT8OK
LDX #6     Point to day of week
RCLOOP
JSR READ
DEX
BPL RCLOOP

Change values you want in the buffer and then write it back to the clock

LDX #6
RCLOOP
JSR WRITE
BNE GLITCH Exit if write failed
DEX
BPL RCLOOP else do all 7
LDA # <RT8SET Set success message
LDX # >RT8SET
BNE EXIT Branch always
RT8ERR
.BADIR "RT8 Error",155
RT8ERR
.BADIR "RT8 Set",155

; Subroutine: wait til clock is
; not busy or exit on time out.
; Enter: x=clock reg to access (0-6)
; Exit: x unchanged, clock ready,
; and clock register selected
WAITCL
LDY #$FF Timeout value
WAITC
LDA $D5B8 Get high byte
LDY $D5B8 Get low byte
AND #$0F Convert bcd to hex
STA TEMP1
ASL A Clears carry
ASL A
ADC TEMP1 Temp1=(high*10)
TYA Add in low byte
AND #$0F
ADC TEMP1 Return byte in a
RTS Note c=0 x=x y=trig3

; Subroutine: read rt8 reg once
; In x=reg#
; Out a=byte x=reg#
READ1
JSR WAITCL
LDA $D5B8 Get high byte
LDY $D5B8 Get low byte
AND #$0F Convert bcd to hex
STA TEMP1

; Subroutine: read a clock register
; and accept best 2 of 3 readings
; or the first if none match.
; in: x=reg#
READ
   JSR READ1
   STA TEMP1
   JSR READ1
   CMP TEMP1
   BEQ REXIT
   STA TEMP2
   JSR READ1
   CMP TEMP2
   BEQ REXIT
   LDA TEMP1
REXIT
   RTS

; Subroutine: write clock register
; with value stored in buffer offset
; by x. Allow 10 tries.
; in: x=reg#
; out: x=reg#, z flag set if ok

WRITE
   LDA #9
   STA RETRY
   WRT2
   LDA BUFFER,X
   LDY #$FF  Convert to bcd
   SEC
   SUB10
   INY
   SBC #10
   BCS SUB10
   ADC #10
   PHA           low byte
   TYA
   PHA           high byte
   JSR WAITCL    y=trig3
   PLA           High byte
   STA $D5B8
   PLA
   STA $D5B8    Low byte
   JSR READ     Verify it set
   CMP BUFFER,X correctly
   BEQ WRTXIT    It did!
   DEC RETRY
   BPL WRT2     Never 0 if failed
WRTXIT
   RTS

The Multiplexer! Operating System (MUX)#

The MUX OS makes frequent access of registers in the $D57x range. A cursory glance at the ROM reveals it uses the following registers on a read-only basis 1, 6 and 7. Registers 2, 3, B, C and E are accessed as write-only while 0 is read/write. Every one of these addresses will affect an OSS cartridge. I found no indication in the MUX code that it makes any effort to accommodate a cartridge, GINTLK or TRIG3.
While I have managed to work around an SDX cartridge and an RT8 in controlling BASIC, OSS cartridges, and even Atari cartridges (with SDX present), I can see no way of doing so with the MUX OS. I believe the idea with the MUX is that once the plug's in the port, you can't use a cartridge anyway. That's kind of unfortunate as it denies you use of a cartridge and access to the built-in monitor. As I don't have a MUX to experiment with, I leave that to someone else.

**Cold and Warm Starting and Parallel Devices (PD's)**

If you turn off an OSS cartridge and then re-boot under most versions of SpartaDOS, you get a "soft" lockup as DOS will enable a cartridge as part of its boot process in testing for BASIC XE. Simply hit Reset and the computer will boot again with the cartridge on. If you want to reenable the SDX cartridge after having turned it off using the COLD command, the method should be obvious by now. What is not obvious are a few other addresses and quirks in warm and cold starting.

On boot the OS calculates ROM checksums and compares them to ones stored in the OS itself. If these don't match, boot doesn't happen; you end up staring at the Self Test screen and a red bar under the heading "ROM". This can easily occur on a system with a PD because cold starting the computer does not cold start the PD any more than it does a cartridge. If PD ROM is enabled, as for a modem handler on a BBS, and you attempt to cold start, you will inevitably end up in Self Test because the ROM checksum will fail. It is supposed to include the floating point ROM at $D800, but instead gets a bank of the PD ROM.

Finally, if you're just going to warm start, decide whether or not you want to emulate a press of the Reset key. Jumping to the warm start vector at $E474 is not the same as pressing the key; the vector points past the hardware initialization routines. If you want to ensure you clear out all garbage (left over player missiles, keypress, etc.) you have to use the chip reset vector.

The following routine has varying results dependant on the entry point. To enable SDX, enter at XCART. To enable an OSS cart alone, enter at CART. To just cold start without touching the cartridges, enter at COLD. To simulate a press of Reset, enter at WARM.

```
XCART
SEI      Always before $D5xx access
STA $D5E0 Enable SDX

CART
SEI      Again, dependant on entry
STA $D500 Enable OSS cart

COLD
DEC COLDST Force a boot

WARM
SEI      Just in case
LDX #0
STX NMIEN Ditto
STX $D1FF Enable floating point
STX $D1E2 Kill MIO RAM (this and
DEX      following just in case)
STX $D1BC turn off BlackBox RAM
STX PORTB Ensure ROM OS is on to go
JMP ($FFFC) through chip reset vector
```

I won't go any further on this as the Black Box and MIO are, unlike Atari and OSS products, fairly well documented. Whew, when I started this I never thought it would turn into such a monster nor did I think it would take so long to come up with all the ins and outs. Now that you know how easy it is to make use of the RAM under cartridges, under the OS and in extra memory, I look forward to seeing some practical utilities. Here's few suggestions:

- "Pop-up" help screens for use in MAC/65, Action! or BASIC.
• An 8k RAM cache.
• A "pop-up" calculator.
• A resident DUP.SYS.
• etc.

LATE NOTES#

I recently was browsing through some old computer magazines and came across an article by Bill Wilkinson dealing with 130XE RAM control. In it he stated that 16k Atari cartridges such as Atari Writer Plus occupy the address space from $8000 to $BFFF rather than using bank switching.

As a final addition to this text, I've tacked on part 4 which is MAC/65 source code to produce two simple COM files to dump cartridges to disk for examination. The programs are not sophisticated but will do the trick.

Revision: 16 Feb 96

LATER NOTE#

A message by Bill Wilkinson posted on comp.sys.atari.8bit on 12 August 95 validates much of what has been said and revealed a bit more of cartridge construction. His memory was a bit off on address ranges, but his stuff about 4k banking is right on. I dumped MAC/65 and Action! to disk files and verified that, in each cartridge, the code from $B000-$BFFF is identical in all three banks. In effect, this indicates a mapping as follows (using MAC/65 as an example):

```
Addr: $D500 $D501 $D509 $D508
$A000   c0    c1    c9    RAM
$AFFF
$B000
$BFFF
```

Where: "Addr" is the access address used to enable the configuration, "c0" etc. is the switchable 4k bank and "cc" is the common 4k bank. Substituting "3" and "4" for "1" and "9" above would produce a map of the Action! cartridge.

```
.OPT NO LIST,NO EJECT
;  SAVE #D1:CARTDUMP.M65
;
;
; ASM ,,#D1:CARTDUMP.COM
;
; Copy OSS cartridge banks to
; D1:CARTIMAGE.CBx where x is bank
;
ICCOM = $0342
ICBAL = $0344
ICBLL = $0348
ICAX1 = $034A
CIOV = $E456
PORTB = $D301
DMACTL = $D400
SDMCTL = $022F
GINTLK = $03FA
TRIG3 = $D013
;
```
* = $2F00

; The first 64 bytes are dumped to
; the CARIMAGE.MAP file

RAMROM
* = *+16 0=rom $FF=ram

BANKNO
* = *+16 byte at $AFFF

BARRAY
* = *+16 0=valid rom bank

TRIG3A
* = *+16 trig3 reading for bank

CURBNK
* = *+1 Base test bank

TESTBK
* = *+1 Bank being tested vs base

CBSAVE
* = *+1 Entry cart status

PBSAVE
* = *+1 Entry portb status

; *= $3000

START
LDA #$60
STA START
LDX #0
TXA
ZLOOP
STA $2F00,X    Clear data page
INX
BNE ZLOOP

; LDA PORTB
STA PBSAVE
DEX
STX PORTB
LDY #$08
LDA GINTLK
BEQ HAVBNK
LDY $AFFF
HAVBNK
STY CBSAVE

; SEI          Kill vblank 2 before
LDX #15      playing with cart
LOOP0
STA $D500,X  First test
LDA TRIG3    if rom or ram
STA TRIG3A,X
LDA $AFFF    Get any bank number
STA BANKNO,X
INC $AFFF    Check ram
CMP $AFFF    No, it's rom
BEQ ISROM
STA $AFFF
DEC RAMROM,X Set rom/ram map to ram
DEC BARRAY,X Show no rom to test
ISROM
DEX
BPL LOOP0

; STA $D500,Y Cartridge normal
CLI vblank now ok

; STX CURBNK Start base at -1
LDY #$FF
LOOP1
INY CPY #16 All higher ones tested?
BCS NEXTBASE Yes, bump test base

LDA BARRAY,Y Unique rom this bank?
BMI LOOP1 No, ram or a dup

TYA
TAX

LOOP2
INX Test all higher
CPX #16 banks for duplicate
BCS LOOP1 All higher tested.

LDA BANKNO,X Was there a valid
CMP #16 bank number?
BCS FAIL No

EOR BANKNO,Y Same as current?
BNE LOOP2 No

FAIL
DEC BARRAY,X Yes, flag a duplicate
BMI LOOP2 Go always

NEXTBASE
INC CURBNK If at bank 15 there's
LDX CURBNK none higher to compare
CPX #15 with
BCS MAPWRT So go finish up

; LDA BARRAY,X Check if bank valid
BMI NEXTBASE No, ram or duplicate

STA $D500,X Enable it
LDY #0 Copy it to $6000
LDA #$A0
STA CLOOP+2
LDA #$60
STA CLOOP+5
CLOOP
LDA $A000,Y
STA $6000,Y
INY
BNE CLOOP
INC CLOOP+2
INC CLOOP+5
BPL CLOOP
JSR WRITE  Write it to disk
JMP NEXTBASE

MAPWRT
LDX CBSAVE
STA $D500,X
LDA PBSAVE
STA PORTB
LDX #$10
LDA #<MSPEC
STA ICBAL,X
LDA #>MSPEC
STA ICBAL+1,X
LDA #3
JSR GOCIO
BMI CLOSE

LDA #<RAMROM
STA ICBAL,X
LDA #>RAMROM
STA ICBAL+1,X
LDA #48
STA ICBLL,X
LDA #0
BEQ BPUT

WRITE
TXA
CLC
ADC #'0
CMP #'9+1
BCC ISHEX
ADC #6

ISHEX
STA BANKID
LDX #$10
JSR CLOSE
LDA #<FSPEC
STA ICBAL,X
LDA #>FSPEC
STA ICBAL+1,X
LDA #8
STA ICAX1,X
LDA #0
STA ICAX1+1,X
LDA #3
JSR GOCIO
BMI CLOSE
LDA #<$6000
STA ICBAL,X address
STA ICBLL,X length
LDA #>$6000
STA ICBAL+1,X address
LDA #>$2000

BPUT
STA ICBLL+1,X length
LDA #11  bput
JSR GOCIO
CLOSE
LDA #12
GOCIO
   STA ICCOM,X
   JMP CIOV

; Copy cartridge bank to $4000
;
BCOPY
   SEI
   STA $D500,X
   LDA # >$A000
   STA LOOP4+2
   LDA # >$4000
   STA LOOP4+5
   LDX #32
   LDY #0
LOOP4
   LDA $A000,Y
   STA $4000,Y
   INY
   BNE LOOP4
   INC LOOP4+2
   INC LOOP4+5
   DEX
   BNE LOOP4
   LDX CBSAVE
   STA $D500,X
   RTS

; FSPEC
   .BYTE "D1:CARIMAGE.CB"
BANKID
   .BYTE "0",155
MSPEC
   .BYTE "D1:CARIMAGE.MAP",155
   .OPT NO LIST
   .END

   .OPT NO LIST,NO EJECT
; SAVE #D1:SDXdump.M65
;
; ASM ,,#D1:SDXdump.COM
;
; Copy SDX cartridge banks to
; D1:SDXIMAGE.CBx where x is bank
;
ICCOM = $0342
ICBAL = $0344
ICBLL = $0348
ICAX1 = $034A
CIOV = $E456
PORTB = $D301
DMACTL = $D400
SDMCTL = $022F
GINLK = $03FA
TRIG3 = $D013
;
; First 64 bytes to SDX.MAP file
;
; *= $2F00
RAMROM
 *= +16  0=rom $FF=ram
BNUMBR
 *= +16  Value at $AFFF
BARRAY
 *= +16  0=valid, unique rom bank
TRIG3A
 *= +16  trig3 reading for bank
;
CURBNK
 *= +1  Base test bank
CBSAVE
 *= +1  Entry cart status
PBSAVE
 *= +1  Entry portb status

 *= $3000
START
LDA #$60
STA START
LDX #0
TXA
ZLOOP
STA $2F00,X  Clear data page
INX
BNE ZLOOP
LDA PORTB
STA PBSAVE
DEX
STX PORTB  Basic rom off
;
; SDX  OSS  BAS  OPERATING CONDITION
; $0C  $08  1   in high ram (x.com)
; $0C  $08  0   in basic
; $08  on  1   in oss cartridge
; $01  $08  1   in dos or low ram
;
LDA #$08  a=$08=oss off
LDX #$0C  x=$0C=sdx off
LDY TRIG3  Is this valid?
BEQ SAVEOX  Yes both are off
TAX
x=$08=sdx xprint
LDA $AFFF  a=ossbnk
CMP #$10  Valid?
BCC SAVEOX  x=$08=sdx xprint a=oss on
TXA  a=$08=oss off
LDX #$01  x=$01=sdx on
SAVEOX
STA CBSAVE
STX XCSAVE
SEI  Kill vblank2 before
LDX #$0F  playing with carts
LOOP0
STA $D5E0,X  Select SDX bank
STA $D508  Kill any OSS bank
LDA TRIG3  Save trigger value
STA TRIG3A,X
LDA $AFFF   Also bank id value
STA BNUMBR,X
INC $AFFF   Check ram
CMP $AFFF   No, it's rom
STA $AFFF
DEC RAMROM,X Set rom/ram map to ram
DEC BARRAY,X Show no rom to test
ISROM
DEX
BPL LOOP0
STX CURBNK
;
TEST1
INC CURBNK   Start sdx bank=0
LDX CURBNK
CPX #$10
BCS TESTED
LDA BARRAY,X Unique ROM to test?
BMI TEST1   No, ram or duplicate
JSR BCOPY   Copy to ram
JSR WRITE   Write to disk
LDX CURBNK
NXTTST
INX          Next higher bank
CPX #$10
BCS TEST1   All done, bump base
LDA BARRAY,X Ram or a duplicate?
BMI NXTTST   Yes, skip it
JSR BTEST   Go test it and set
JMP NXTTST  array if applicable.
;
TESTED
LDX XCSAVE
STA $D5E0,X
LDX CBSAVE
STA $D500,X
LDA PBSAVE
STA PORTB
LDX #$10
LDA # <MSPEC
STA ICBAL,X
LDA # >MSPEC
STA ICBAL+1,X
LDA #3
JSR GOCIO
BMI CLOSE
LDA # <RAMROM
STA ICBAL,X
LDA # >RAMROM
STA ICBAL+1,X
LDA #64
STA ICBLL,X
LDA #0
JSR BPUT
CLI
RTS
;
WRITE
TXA
CLC
ADC #'0
CMP #'9+1
BCC ISHEX
ADC #6
ISHEX
STA BANKID
LDX #$10
JSR CLOSE
LDA <$FSPEC
STA ICBAL,X
LDA >$FSPEC
STA ICBAL+1,X
LDA #8
STA ICAX1,X
LDA #0
STA ICAX1+1,X
LDA #3
JSR GOCIO
BMI CLOSE
LDA <$6000
STA ICBAL,X address
STA ICBLL,X length
LDA >$6000
STA ICBAL+1,X address
LDA >$2000
BPUT
STA ICBLL+1,X length
LDA #11    bput
JSR GOCIO
CLOSE
LDA #12
GOCIO
STA ICCOM,X
CLI
JSR CIOV
SEI
RTS
; BCOPY
STA $D5E0,X
STA $D508
LDA #$A0
STA BCLOOP+2
LDA #$60
STA BCLOOP+5
LDY #0
BCLOOP
LDA $A000,Y
STA $6000,Y
INY
BNE BCLOOP
INC BCLOOP+2
INC BCLOOP+5
BPL BCLOOP
RESTOR
LDY XCSAVE
OSS has recently introduced BASIC XL and ACTION in a cartridge referred to as the supercartridge. This article will explain the advantages of using such a cartridge and describe in detail how the cartridge works. The advantage of using the supercartridge hardware is that one can have 16K (K = 1 kilobyte) of ROM and 8K of RAM all within only 8K of memory address space. What this means is more of your precious memory is available for your programs and data instead of being used up by a 16K cartridge or a large applications program or interpreter. This efficient use of memory resources is accomplished by selectively activating 4K segments of two 2764 (8 Kilobyte x 8 bit) EPROMS. Only two 4K banks out of the possible 4 can be selected at one time. There are 2 EPROMS located on the supercartridge board that I have designated ROM A and ROM B. The upper 4K of ROM A will always reside in the $B000-$BFFF address range when the RAM is deselected. The other 4K banks (ROM B upper, ROM B lower, and ROM A lower) can selectively be mapped into the $A000-$AFFF address range. Optionally all the ROMs can be deselected and the computer's existing RAM can be accessed in the $A000-$BFFF memory range. The bank switching is accomplished by writing to a memory address within the range $D500-$D5FF. When this write occurs, address bits A0 thru A3
are latched into a 4 bit register located on the cartridge. The status of each bit determines the current mapping configuration to be put into effect. The following tables are used to show what each address bit actually does in the cartridge. A0 always controls the selection of ROM A. A1 always controls the selection of ROM B. A2 selects which half of ROM B is used when it is selected. A3 selects/deselects RAM. BIT A0 This table shows what portion of ROM A will be used in each address range dependent on the state of A0.

<table>
<thead>
<tr>
<th>ROM Address</th>
<th>ROM Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000-AFFF</td>
<td>B000-BFFFF</td>
</tr>
<tr>
<td>A0=0</td>
<td>nothing</td>
</tr>
<tr>
<td>A0=1</td>
<td>ROM A lower</td>
</tr>
</tbody>
</table>

BIT A1 This table shows when ROM B is selected. The half of ROM B used is determined by A2.

<table>
<thead>
<tr>
<th>ROM Address</th>
<th>ROM Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000-AFFF</td>
<td>B000-BFFFF</td>
</tr>
<tr>
<td>A1=0</td>
<td>ROM B select</td>
</tr>
<tr>
<td>A1=1</td>
<td>nothing</td>
</tr>
</tbody>
</table>

BIT A2 This table shows which half of ROM B is used when it is selected.

<table>
<thead>
<tr>
<th>ROM Address</th>
<th>ROM Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000-AFFF</td>
<td>B000-BFFFF</td>
</tr>
<tr>
<td>A2=0</td>
<td>Lower 1/2 of ROM B (A12=0)</td>
</tr>
<tr>
<td>A2=1</td>
<td>Upper 1/2 of ROM B (A12=1)</td>
</tr>
</tbody>
</table>

BIT A3 This table shows the effect of A3.

<table>
<thead>
<tr>
<th>ROM Address</th>
<th>ROM Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000-AFFF</td>
<td>B000-BFFFF</td>
</tr>
<tr>
<td>A3=0</td>
<td>ROM Selected/Ram deselected</td>
</tr>
<tr>
<td>A3=1</td>
<td>RAM selected/ROM deselected</td>
</tr>
</tbody>
</table>

Make note that some of these options can not be selected simultaneously. For example, an illegal option would be A0=1 and A1=0. In that case both ROM A lower and ROM B would be selected for the $A000-$AFFF address range. Possible Valid Configurations: These diagrams represent the segments of ROM and/or RAM that will be activated when the address shown is written to.

`$A000: ROM B : nothing :`
If one was to combine the above configurations into one diagram, you would get something like this representing the possible configurations:

```
$A000:   $A000:   $A000:   $A000:
  Rom A  Rom B   Rom B  Rom B
  lower  lower   upper   upper
$AFFF:________:________:________:________:
  A0=0   A0=0
  A1=1   A1=0
  A2=0,1 A2=1
  A3=0   A3=0

$B000: $B000: $B000: $B000:
  Rom B   Rom A   Rom A   Rom A
  upper   upper   upper   upper
$AFFF:________:________:________:________:
  A0=1   A0=0
  A1=1   A1=0
  A2=1   A2=0,1
  A3=0   A3=0

$D503 or $D507 $D504

$D500 $D500
```

$D508-$D50F
When ROM is swapped out for RAM, the entire range $A000-$BFFF no longer has ROM. The ROM can only be swapped out in the entire 8K range. In the ACTION! cartridge ROM A is designated MA and ROM B is designated LI. In the BASIC XL cartridge ROM A is designated as BAS-H and ROM B is designated as BAS-L.

CIRCUIT BOARD LAYOUT

| : | : | : | : |
| : | # | : | : |
| : | : | D 74LS02 | |
| # | 74LS175 | : | : |
| D | 74LS00 | : | : |
| -#- | -#- | : | : |

| : | : | : | : |
| 1 | 28: | : | 28: |
| 2 | 27: | : | 27: |
| 3 | 26: | : | 26: |
| 4 | 25: | : | 25: |
| 5 | 24: | : | 24: |
| 6 | 23: | : | 23: |
| 7 | ROM B | 22: | 7 | ROM A | 22: |
| 8 | 21: | : | 21: |
| 9 | 20: | : | 20: |
| 10 | 19: | : | 19: |
| 11 | 18: | : | 18: |
| 12 | 17: | : | 17: |
| 13 | 16: | : | 16: |
| 14 | 15: | : | 15: |
| : | : | : | : |
| : | : | : | : |
| : | : | : | : |
| 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 | 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 |

COMPONENT SIDE PINOUT

| : | : | : | : |
| : | : | : | : |
| : | : | : | : |
| A | B | C | D | E | F | H | J | K | L | M | N | P | R | S |

FOIL SIDE PINOUT

Part 3: Cartridge slot descriptions#

by Jindrich Kubec
Left slot pinout (all models) ARD4if cartridge maps to $8000-$9FFF area

(act H)1/S4 selects lower bank (act L)
BGND 2A3
CA4 3A2
DA5 4A1
EA6 5A0
FA7 6D4
HA8 7D5
JA9 8D2
KA12 9D1
LD3 10D0
MD7 11D6
NA11 12/S5 selects upper bank (act L)
PA10 13+5V
RR/WR/W - read (act H), write (act L)14RD5 if cartridge maps to $A000-$BFFF area (act H)
SB-PHI2buffered phase 2 clock15/CCTL active when written to $D5xx (act L)

Right slot pinout (only 800) AB-PHI2 1R/W

BGND 2A3
CA4 3A2
DA5 4A1
EA6 5A0
FA7 6D4
HA8 7D5
JA9 8D2
KA12 9D1
LD3 10D0
MD7 11D6
NA11 12/S4 selects lower bank (act L)
PA10 13+5V
RR/WR/W - read (act H), write (act L)14RD4 if cartridge maps to $8000-$9FFF area (act H)
SB-PHI2buffered phase 2 clock15/CCTL active when written to $D5xx (act L)

Parts are (always?) at back (XL)/bottom (XE) part of the cartridge. Connector orientation is S-A from front, 1-15 from back.

Cartridge basics

There are three different 'windows' or banks in the memory: 1 Bank $A000-$BFFF is selected by low state on S5. Its presence is signaled by high state on RD5. This signal is attached to TRIG3 in XL/XE series. Used in left carts only. 2 Bank $8000-$9FFF is selected by low state on S4. Its presence is signaled by high state on RD4. Used in both left and right slot carts. 3 'Bank' $D500-$D5FF is selected by low state on CCTL. It's used for bankswitching. (mapping ports etc.)

Czech speciality are cartridges with buttons. There are 2 different ways: 1 the button only connects RD5 to computer. So you push it, press reset, make the selection (if any) and then release the button. The program will be loaded from cartridge and will not take that much amount of memory (minimum is 8KB, obviously). 2 the button turns on RD5 and leaves it on. After reset, user could do some selections and cartridge will turn off when needed.
Romox released 'blank' 16KB cartridges which could be programmed in special machines. They were called Edge Connector Programmable Cartridges. Emulated: Atari800

OSS carts#

Banks: Usually contain 4 4KB banks. One of the banks is mapped in $B000-$BFFF. Other banks are mapped to $A000-$AFFF. Double eprom PCB (OSS DBL): Older scheme, uses 74LS175 (4bit register???), 2 eeproms and uses 4 bits of address bus. Physical order of banks: ROM A = 3, M; ROM B = 0, 4 PCB Top (42 KB JPG), PCB Bottom (42 KB JPG) (with parts)

A0 if 1, selects AL
A1 if 0, selects B
A2 if 0, selects BL, if 1 selects BH (A1 must be 0!)
A3 if 0, cart on, if 1, cart off.

AddrA000-AFFFFB000-BFFFFD5
0000BLAU1
0001AL+BL = shitAU1
0010nothing = FF'sAU1
0011ALAU1
0100BAU1
0101AL+BH = shitAU1
0110nothing = FF'sAU1
0111ALAU1
1xxxoffoff0

Single rom PCB (OSS SNG): Newer scheme, uses 2 d-flipflops, bits A0 and A3. Physical order of banks: M, 0, 9, 1.

A3A0A000-AFFFFB000-BFFFFD5 Values
00bank 0bank m10,2,4,6
01bank 1bank m11,3,5,7
10offoff0,A,C,E
11bank 9bank m19,B,D,F

Examples:

Action!, MAC/65 V1.00, MAC/65 1.01, MAC/65 1.02, Basic XE, Basic XL 1.02, Basic XL 1.03, Writer's Tool;

Emulated: Atari800

SDX carts#

Banks: Contain 8 8KB banks. Access to base+0-7 turns on banks 0-7 in $A000-$BFFF area. Access to base+8-F turns the cart off.

SDX: Base is $D5E0.

Diamond: Base is $D5D0.

Express: Base is $D570.

Better said: Cartridge looks for low CCTL and match in A4-A7. For SDX it is 0xE (1110), Diamond 0xD (1101), Express 0x7 (0111). If A3 is high, cart is turned off. If A3 is low, the bank whose number is in A0-A2 is selected. I'm not sure what happens when turning cart off, but I think that it turns off RD5 only.
Examples:
Sdx 4.18, Sdx 4.19, Sdx 4.20, Sdx 4.21, Sdx 4.22, Express, Diamond GOS 1, Diamond GOS 2, Diamond GOS 3, MIO Diagnostics
Emulated: Atari800

**R-Time 8**
Pass thru cartridge with battery-backup for real-time clock.
Emulated: Atari800 - read only is enough.

**XEGS carts**
Banks: Contain n 8KB banks. Bank n-1 is mapped at $A000-$BFFF. Banks 0 to n-1 are mapped to $8000-$9FFF by writing the number of bank to any of $D500 registers.

Examples (32KB): Archon, Blue Max, Crystal Castles, Into the Eagle's Nest, Food Fight, Star Raiders II;
Examples (64KB): Ballblazer, Battlezone, Choplifter!, David's Midnight Magic, Deflektor, Dark Chambers, Desert Falcon, Hardball, Mario Bros., Rescue on Fractalus, Tower Toppler, Thunderfox
Examples (128KB): (Bug Hunt and Lode Runner probably contain 64KB rom, but start from the bank 8.) Ace of Aces, Airball, Barnyard Blaster, Bug Hunt, Crime Buster, Crossbow, Fight Night, Flight Simulator II, Gato, Karateka, Lode Runner, Summer Games;
Emulated: Atari800

**XEGS demo cartridge**
Banks: 128 KBs.
Contains:
- XEGS 64 - Flight Simulator 2 (shortened demo version)
- XEGS 32 - One on One (same version as on XEGS cart)
- LS16 - Joust (same version as on cart)
- LS16 - Ms. Pacman (same version as on cart)
Does change game on each reboot (binary counter powered by capacitors). Emulated: Unemulated (not needed).

**Telelink 2 cartridge**
Banks: Just one 8kb bank at A000-BFFF. Additional hardware is one X2212 nonvolatile SRAM, containing 256 x 4 bits, for storing telephone numbers. It's accessed by reading/writing 9000-90FF. There is also read access to $D501 and write access to $D502, don't know yet for what, but probably for setting read/write mode of the SRAM. The cartridge uses RD4 hardwired to +5V, what means that it takes full 16KB of address space. Not very elegant solution.
Emulated: Unemulated
MD-DOS cartridge Banks: Same as OSS SNG cartridges. Uses two additional bits (A5,A4) for selecting 'subcart'. It thus has 4x16KB = 64KB. Also has button, don't know for what it is used (probably for calling the cartridge).

MD Dos Emulated: My experimental Atari800 emulator - but with possible bugs.

**JRC cartridges**

Toolbox III: Contains 64KB bankswitched cart with reset button.

Rambox2: EEPROM part same or very similar to Toolbox III. Additionally contains 256KB of RAM. This ram could be accessed by TT-Dos and BeWe Dos.

Toolbox III

Rambox2

EEPROM part: 8x8KB banks. Selected by D6, D5 & D4. Bank number is remembered when A7 is off (D500-D57F). If D7 is on, cartridge is off. If button is pressed, register is cleared, RD5 is on, main bank selected. RAM part:

```assembly
eram bank select:
X = 00 - 7F (7bits)
A = A | $F0 (4bits)
STA $D500,X
```

Together it's 11 bits of bank address. You can address $80 bytes of memory in region $D580 (7bits). 11 bits + 7 bits = 18 bits = 256KB of RAM.

memory write:

```assembly
LDA (BUFRLO), Y
STA $D580, Y
```

memory read:

```assembly
LDA $D580, Y
STA (BUFRLO), Y
```

Y = 00 - 7F. Uses previously set bank address.

Emulated: My experimental Atari800 emulator.

**ATRAX cartridge**

Banks: Contains 16 x 8KB banks (128KB). Uses A000-BFFF region. Bits 0-3 select bank. Bit 7 turns cart off. Menu program loads standard Atari dos executable, then turns cart off. I was told that 16 different carts exists.

ATRAX no.5

Emulated: Atari800
Bounty Bob cartridge#

Banks: Very strange bankswitching method. Contains main bank at A000-BFFF. From 8000-8FFF is first bank, 9000-9FFF is second bank. They are switched by accessing 8FF6-8FF9 (9FF6-9FF9). In each bank there's 4 x 4 KBytes. Totally, it's 2 x 4 x 4 + 8 = 40 KBytes. I was told that the cartridge is almost same as 5200 version and such cart contains only three chips: one 8KB rom, and two special self-switching 16KB roms. What circuitry is inside those chips is unknown to me.

Bounty Bob

Emulated: Atari800, BBEmu (Bounty-Bob Emu!)

128/256 K RamCart#

Banks: Unknown.

Emulated: Unemulated.

A/D Converter#

Banks: None. Uses lower four bits of $D500 for getting the digitized sound from mono analog input.

Emulated: Unemulated.

MPP Super Charger#

Banks: ???. May contain 'hardware accelerator chips'. Used together with game disk 'Assault Force'. (Still available from BEST Electronics!)

Emulated: Unemulated.

PILL carts#

The PILL!

Super PILL!

Super Cart (by Frontrunner)

Banks: None. Just a way to fool the computer into thinking that a cart was inserted (due to the copy protection of some carts).

Emulated: ?

Willams multibanked cartridge#

Banks: Contains 8 x 8kb banks. Writing to $D500 turns on first bank (on bootup), writing to $D507 turns on last bank. Writing to $D508 turns cart off. Video61 uses this design for their newly released 'big' cartridges. Emulated: Atari800 (from >1.2.2)

Thompson Proburner#

EEPROM burner. Banks: Contains 8kb in right slot, accesses D500, probably hardware for chip writing. Need more info.
COS32#

Banks: 16kb main bank. The other is banked in for a short period of time (would anybody compute that for me?) by any access to $D5xx area. Also contains the button on RD4/RD5. As I have other similar COS cartridge, I seem to remember there was some kind of service/toolkit which allowed to put anything on the cart.

Great source of information about memory and cartridge mappings is in an article by John Picken. (see Part 1!) this info (c) 1998-2002 Jindroush. Last modified: Fri Feb 22, 20:20:00 2002;

Part 4: Converting games to Cartridge#

by Nir Dary

(written 23/6/2001; mail: ndary@bigfoot.com)

We all love cartridges some of us even collect them. They are simple and quick to load, just plug it in your computer and the game starts. This time I would like to share my knowledge and show you all how you can convert your favorite games to cartridges, all you need is an empty EPROM chip, an EPROM burner, Cartridge PCB and an Assembler editor. Before we will start to explain how to port disk/tape games lets have a few words about cartridges...

The earlier ATARI model (800) had two 8K cartridge slots: Left and Right. The Right cartridge occupied memory address $8000 to $9FFF while the Left cartridge occupied memory address $A000 to $BFFF. Once a cartridge is inserted to the computer the memory address above becomes read only data. The XL/XE ATARI models are compatible with the 800 cartridges but they also have the possibility to read one 16K cartridge (that will occupy memory address $8000 to $BFFF), lets not forget the Bank Switchable cartridges, these are cartridges that contain more than 16K data, like the XEGS or OSS cartridges. The XEGS can contain up to 128K of game data i.e. 'Ace of Aces', 'Flight Simulator 2' carts etc.. Lets look at the cartridge connector:

Cartridge Slot ("Left" slot on all machines; "Right" slot on 800 only):

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>H</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>P</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

1. /S4                      A. RD4
2. A3                      B. GND
3. A2                      C. A4
4. A1                      D. A5
5. A0                      E. A6
6. D4                      F. A7
7. D5                      H. A8
8. D2                      J. A9
9. D1                      K. A12
10. D0                     L. D3
11. D6                     M. D7
12. /S5                    N. A11
13. +5V                    P. A10
14. RD5                    R. R/W
15. /CCTL                  S. B02
When the ATARI computer is booting, one of the first things the Operating system is checking is cartridge presence. If a cartridge is inserted its been executed, so a cartridge can be executed even before the rest of the OS routines are finished (i.e a Diagnostic Cartridge). The Last 6 bytes of the cartridge Memory location (addresses $BFFA to $BFFF) determine the RUN/INI address of the cartridge, if to Load DOS before running the cart software? etc\x{2026}

$BFFA/$BFFB - is the cartridge start address.
$BFFC - A non-zero number here tells the OS that there is no cartridge in the left slot
$BFFD - set to $05 if you want to boot Dos before running the cart $04 if no
$BFFE/$BFFF - Cartridge initialization address

The XEGS cartridges contain several 8K banks (depending on the cartridges size), the last bank is always shown at Memory area $A000 to $BFFF of the Atari computer while the rest of the 8K banks are mapped memory area $8000 to $9FFF depending on what value of data you set $D5XX. For example on a 64k cartridge the last 8k Bank of the cartridge (cartridge ROM memory location $E000 to $FFFF) are mapped to the XL memory location $A000 to $BFFF, and when accessing Atari location $D500 you can set which of the first seven 8K banks will be mapped to the ATARI memory location $8000 TO $9FFF.

Now lets get into business: You cannot convert every game to a cartridge, there are limitations...

1. You must feet the game into the cartridge memory space; you can use a 16k PCB for games that are less than 16k in size, or a XEGS pcb with larger ROMs. 2. The game must load in one part (single stage - one file). If the game is loading in several stages - then without modifying the game code it will try accessing TAPE/DISK when looking for the next stage to load *** 3. The game should not try to write to the cartridge memory address. Because after you insert the cartridge this is read only memory... 4. Some games are detecting cartridge presence and when a cartridge has been inserted the game will not run; this was done to prevent from some debugging cartridges to help hacking the game code

   • *** However there are some tricks to get even a multi-stage tape/disk game running from a cart (examples: Dandy, Summer Games, etc.) - but thats really a lot of work we will not explain here...

If your game is not restricted by the above limitations then we have a go. What we will do is simply write a routine that will copy the game data from the ATARI Cartridge Memory area back to the original game location and execute the game. This works for 90% of the games I tried !!

Next you will find a generic cartridge memory dumping routine that will work for most of the games. To explain how it works, lets take a game and make it work from a 16k cartridge PCB (“Danger Ranger” was never released on a cartridge).

Danger Ranger game loads into Memory Address: $0400 to $211F and is executed at address: $0488. The TABLE at the beginning of the source tells the copy routineto copy the Game Data from Address $A200-$BF20 To $0400-$211F. Here is the TABLE description:

| Byte 0   | What 8k bank to access, to use with XEGS carts (on 16k cart it should always be set to 00), if equal to $FF then end copy. |
| Byte 1,2 | Start Source address of data location on the Cartridge ROM, (on a 16k cart pcb it will be from $8000 to $BFFF, and on an XEGS cart from $A000 to $BFFF) |
| Byte 3,4 | Destination address where to copy the game data. |
| Byte 5,6 | Last byte of Source address to copy |
Universal Cartridge Copy / Run routine

; Universal Cartridge Copy / Run routine
; written by Nir Dary 13/6/2001

; GSA       = $0488 ; Game Start Address
DOSINI    = $0C ;
DOSVEC    = $0A ;
TMPFROM   = $CA ; Temp Source address
TMPTO     = $CC ; Temp Dest. address
TMPEND    = $CE ; Temp End Copy Addr
TMPTABLE  = $D0 ; Table address

; This Routine will be located at $A000
; but Assembled at $3000 mem area

Org $3000

TABLE .BYTE $00,$00,$A2,$00,$04,$20,$BF
    .BYTE $FF ; $FF ends the copy

START   STX $03E9 ; Cassette vector -
    STX $03EA ; disabled
    DEX
    STX $D301 ; Basic off - hardware
    LDA #$01
    STA $09 ; Boot? - say booted
    STA $42 ; Critic set
    STA $03F8 ; Basic off - software

LDA #$00 ; Short Move Routine
START
    STA TMPTO
    STA TMPPROM
    STA TMPTABLE
    TAY
    LDA #$30 ; Copy to PAGE 30
    STA TMPTO+1
    STA TMPTABLE+1
    LDA #$A0
    STA TMPPROM+1
L1    LDA (TMPPROM),Y
    STA (TMPTO),Y
    INY
    BNE L1
    JMP COPY ; ie. jmp 3000 regoin to
    ; move code out of cart.

COPY   LDA #$00 ; Enable Access to
    STA $D40E ; Shadow O.S Ram
    SEI
    LDA #$FE
    STA $D301
L2    LDY #$00 ; Load Bank# from table
    ; into X register
    LDA (TMPTABLE),Y ; deted if End
    CMP #$FF ; of Table then
BEQ EXITCOPY ;exit to run game
TAX
INC TMPTABLE

LDA (TMPTABLE),Y ;Load Source
STA TMPPFROM ;Address
INC TMPTABLE
LDA (TMPTABLE),Y
STA TMPPFROM+1
INC TMPTABLE

LDA (TMPTABLE),Y ;Load Dest
STA TMPTO ;Address
INC TMPTABLE
LDA (TMPTABLE),Y
STA TMPTO+1
INC TMPTABLE

LDA (TMPTABLE),Y ;Load Source
STA TMPEND ;end Address
INC TMPTABLE ;to copy
LDA (TMPTABLE),Y
STA TMPEND+1
INC TMPTABLE

; ;Start of Copy Routine
;
MAINLOOP STX $D500 ;Load Rom bank to
LDA (TMPFROM),Y
STA (TMPTO),Y
INC TMPFROM
BNE L3
INC TMPFROM+1
L3 INC TMPPFROM+1
L3 INC TMPTO
BNE L4
INC TMPTO+1
L4 LDA TMPFROM+1
CMP TMPEND+1
BNE MAINLOOP
LDA TMPFROM
CMP TMPEND
BNE MAINLOOP
BEQ L2 ; Do next Bank?
;
EXITCOPY LDA #$22
STA $D400 ;DMA control
LDA #$FE
STA $D301 ;RAM under ROM on
LDA #$40 ;Some games need $C0
STA $D40E ;NMI allowed
CLI

LDA <$GSA
STA DOSINI
STA DOSVEC
LDA >$GSA
STA DOSINI+1
STA DOSVEC+1
;
LDA #$C0 ;C000 high byte
STA $2E4 ;system size
STA $2E6 ;
STA $6A ;dlist ends a

; LDA #$00 ;cart interlock
; STA $03FA ;no cart status = off
STA $42 ;critic OK
STA $244 ;COLDST
LDA #$01
STA $08 ;WARMST
TAY
CLC
JMP GSA ;$E474
RTS

; Cartridge Run Address
; See Mapping the Atari
;
Org $BFFA
.WORD $0000
.BYTE $00
.BYTE $04
.WORD START+$7000

Once you assembled this routine make sure you add the game data to the correct source location. With this doc you will find a ready made ROM file of Danger Ranger game, simply burn the ROM data into a 27C64 (8k) EPROM, and insert it on the Right slot of a 16k cartridge pcb, (try the 16k Cartriges PCB from B&C computer visions or Best Electronics).

NIR DARY