TI-99 SAMS Memory Card

From an article in Micropendium

by Bruce Harrison.

Found on ATARIAGE.COM TI-99 4A Development board

With addendum by Brian Fox, Jan 6, 2019 for Forth

SAMS CARD ACCESS and MAPPING

To use other than the default setup, you have to do two things. First, you have to "turn on" the card's memory in the >4000 block and write to the mapping registers there. Second, you have to "turn on" the mapper function to make what you've written into the >4000 block take effect. The first step in all this is to set Register 12 to the value >1E00, which is the CRU address of the AMS card. Now doing a SBO 0 instruction will enable the >4000 memory block for reading and writing. In that block, each word starting with >4000 and ending with >401E acts to control what page maps into what memory location.

The subroutine AMSINI (see sidebar) initializes all the mapping registers to power-up condition. Thus, for example, >4004 is set for page 2, >4006 to page 3, etc. In our own program, we're mapping things into the >A000 through >F000 blocks. Thus the registers we write to in >4000 are at >4014 thru >401E. To make a mapping selection, we write the desired page number to the high order byte of the control register.

Software

Low-level routines

You will have noted that the original SuperAMS does not map any memory at >4000-5FFF, thus cannot contain any DSR. This is because it is a memory expansion card, and these do not need DSRs. However, we still need some way to control memory paging. This must be done by the application program.

The principle is very easy. You can set a page number by setting CRU bit 0 to 1, and writing a page number into 8 among the 16 available registers.

Register Address	Paged area
>4000	>0000-0FFF
>4000	>1000-1FFF
>4004	>2000-2FFF
>4006	>3000-4FFF
>4014	>A000-AFFF
>4016	>B000-BFFF
>4018	>C000-CFFF
>401A	>D000-DFFF
>401C	>E000-EFFF
>401E	>F000-FFFF
Others	Inactive

It may seem harmless to store arbitrary values in the remaining registers, as mapping never occurs in the area that they control. However, these should not be used to store data irrelevant to mapping, as a future version of the card may want to extend mapping to the >4000-7FFF area, for instance.

As mentioned above, CRU bit 1 can be used to toggle between mapping and transparent mode. A SBO 1 enables mapping in the whole area >2000-3FFF and >A000-FFFF, a SBZ 1 puts it in transparent mode (page 2 at >2000-2FFF, page 3 at >3000-3FFF, etc.)

Let's take a concrete example

Suppose we want pages 4, 5 and 6 of the AMS to behave as if they were at >A000, >B000, and >C000. It's this easy:

```
LI  R12,>1E00 CRU ADDRESS
SBO 0         TURN ON CARD'S >4000 BLOCK
LI  R1,>0400  PAGE NUMBER IN LEFT BYTE
LI  R3,>4014  MAPPING REGISTER FOR >A000 BLOCK
MOV R1,*R3+   WRITE TO >4014 AND ADD 2 TO R3
AI  R1,>100   LEFT BYTE OF R1=5
MOV R1,*R3+   WRITE 5 TO >4016 AND ADD 2 TO R3
AI  R1,>100   LEFT BYTE OF R1=6
MOV R1,*R3    WRITE 6 TO LEFT BYTE AT >4018
```

This sets the registers, but does not actually start the mapper working. To put the mapping in effect, we have to perform one more operation:

```
SBO 1          TURN ON THE MAPPER
```

Now that the mapper is turned on, anything written to >A000 thru >CFFF will actually be written into pages 4 thru 6 of the AMS memory instead of into pages >A thru >C. Similarly, any memory reading from >A000 thru >CFFF will actually read the contents of pages 4 thru 6 of the AMS memory. Thus your program can write and read as if to >A000 thru >CFFF, but the mapper will make things actually access the pages 4 thru 6 in the AMS card. You can turn off access to the card's >4000 block and still leave the mapper in effect by the instruction SBZ 0.

(Provided R12 still contains >1E00.) You can turn off the mapper by SBZ 1, and then the areas >A000 thru >CFFF will revert to "normal", reading and writing to pages >A thru >C of the AMS card.

In the finished AMS version of Video Titler, the program works by frames, where each frame accounts for three pages of the AMS memory. We use an odd-even scheme, so that odd numbered frames (1,3,5,...19) go into pages mapped to >A000 thru >CFFF, and even numbered ones go into pages mapped to >D000 thru >FFFF.

This way the program always has immediate access to two frames at any given time.

Example 2

Let's say for example that the user asks to load a TI-Artist picture (with color) into Frame 1.

We take the desired frame number, multiply that by three, add one, then put that in the left byte of a workspace register. Since this is an odd frame, we use the mapping registers at >4014 thru >4018, and put 4, 5, and 6 into the left bytes of those mapping registers, respectively. (For an even page, we'd use >401A thru >401E.)

Now when the program reads the file, it dumps the contents of the pattern and color parts into >A000 thru >CFFF, which really goes into pages 4 thru 6 of the AMS memory.

In similar fashion, frame 2 goes into pages 7 thru 9, frame 3 into pages 10 thru 12, frame 4 goes to pages 13 thru 15, and so on until frame 20, which goes into pages 61 thru 63. Clear as mud? Look at Part Two of the Sidebar, which may help.

Meanwhile, how about a little different perspective?

The AMS Owner's Viewpoint

The Super AMS card is a most welcome and needed addition to the TI 99/4A. The physical aspects are a well-made, high-quality card which inserts in place of the 32K card. In normal use, it duplicates the 32K card exactly, except for the absence of an indicator light. In expanded mode, the card provides from 128K to 1 meg of paged memory, depending on the chips installed.

There are three disks of software provided with the card, of which two are archived. Included is an excellent memory tester that writes and reads to every bit. This assures the user that everything is functioning properly. Also included is an excellent disk copy program, AMS Copy. The program reads an entire 720 sector disk into memory, then writes to the copy disk in one pass. Contents of the master disk remain in memory, allowing as many copies to be made as needed without re-reading.

Also there were two games, which I could not get to run. The remainder of the software is mainly for programmers and of little consequence for the end user. So, aside from testing the memory and copying disks, what can be done with the Super AMS card? Well, prior to the middle of July, the answer was "not too much."

Enter Bruce Harrison.

When I first purchased the Super AMS card back in May at Cleveland, Bruce rather poo-pooed it, saying that he could write any program for the TI and keep it well within the 32K space. No argument here. Then I posed the hypothetical question....how many titles could the Video Titler program hold in 256K of memory? To which Bruce answered, "I'll think about it."

Well, Bruce did a lot of thinking and a lot of programming. The result.... the Video Titler can now hold up to twenty color graphic files in memory on a 256K card. Super. They are instantly available using any of the many different wipes in the program. If you make a mistake, or change your mind, any frame can be reloaded without changing the others. The sequence can also be started at any frame.

Anyone who has previously used the Video Titler will most certainly appreciate the new features, added versatility, speed, and power of the new AMS Video Titler. The entire operation of paging twenty titles into memory, then using them is totally seamless and transparent to the user. It is as though the TMS9900 had twenty address lines.

Substantial documentation is included with the program, but I dare say that most folks won't read them. Aside from possibly looking up which keys to use for what wipes, the Titler is that easy to use. When you are ready to quit, the Titler returns nicely to the E/A menu screen.

It's rather ironic.... To the best of my knowledge, Bruce Harrison is the first programmer outside of the SAMS development team to make any real productive software for the Super AMS card. Yet, he is the one programmer that never received a card to use. The AMS Video Titler beta copy worked flawlessly the very first time. This is a real tribute to outstanding programming ability.

Programmer's Update

Since the above was written, I have received a set of the AMS documentation and an offer of a card from David Ormand of the SW99ers. David has been sent a copy of the AMS Video Titler and a letter accepting the offer of a card to use for programming and testing my new products designed for use with AMS.

```
* SIDEBAR PART ONE - A TEST PROGRAM
* COMPLETE AS SHOWN
*
* TEST PROGRAM FOR AMS CARD
* SPECIAL BY B. HARRISON
* FOR TITLER AMS VERSION
* AMST/S
       REF  KSCAN,VMBW
       DEF  START

KEYADR EQU  >8374        Key-unit address
KEYVAL EQU  >8375        Reported keystroke
STATUS EQU  >837C        GPL STATUS BYTE
GPLWS  EQU  >83E0        GPL workspace
GR4    EQU  GPLWS+8      GPL Reg 4
GR6    EQU  GPLWS+12     GPL Reg 6
STKPNT EQU  >8373        stack pointer
LDGADD EQU  >60          load grom address address
XTAB27 EQU  >200E        a storage location
GETSTK EQU  >166C        get stack
************************************************
START  LWPI WS           LOAD OUR WORKSPACE
       BL   @AMSINI      USE AMS INITIALIZE SUBROUTINE (BELOW)
       C    R1,@>401E    AMS CARD PRESENT? 0=NO 401E=0F0F=YES
       JNE  NOAMS        IF NO AMS CARD, JUMP AHEAD
       SBO  1            TURN ON MAPPER
       CLR  R1           R1=0
       CLR  R4           R4=0
       MOV  R1,@>401E    SET >F000 TO PAGE >00
       MOV  R1,@>F000    SET FIRST WORD PAGE 0 TO 0
       LI   R3,>8000     R3 TO PAGE >80
       MOV  R3,@>401A    SET >D000 TO PAGE >80
       LI   R1,>0F00     START AT PAGE 15
AMSLP1 AI   R1,>100      ADD ONE PAGE
       INC  R4           INC NUMBER FOUND
       MOV  R1,@>401C    SET >E000 TO PAGE 16, 17, 18, ETC.
       MOV  R1,@>E000    LOAD >E000 WITH PAGE #
       C    R1,@>E000    DID IT LOAD?
       JNE  AMSDO        IF NOT, WE'RE BEYOND LAST PAGE
       C    @>F000,@>E000     CHECK PAGE 0 FOR CHANGE
       JEQ  AMSDO        IF EQUAL, JUMP
       CI   R1,>9000     LIMIT NUMBER?
       JL   AMSLP1       IF LESS, REPEAT
       C    @>D000,@>E000     CHECK PAGE 80 FOR CHANGE
       JNE  AMSLP1       IF NOT EQUAL, REPEAT
AMSDO  DEC  R4           ONE LESS (R4 EQUALLED PAGE NOT FOUND)
       MOV  R4,@AMS      SAVE NUMBER OF PAGES ABOVE 15
       BL   @AMSINI      AMS INITIALIZE (BACK TO "NORMAL")
       SBZ  0            TURN OFF CARD - MAPPER STILL ON
       SBZ  1            TURN OFF MAPPER
       JMP  REPORT       JUMP TO REPORT SECTION
NOAMS  LI   R0,11*32+3   ROW 12, COL 4

       LI   R1,NOSTR     NO AMS MESSAGE
       BL   @DISSTR      DISPLAY THAT
```

```
       JMP  KEYEX        JUMP TO EXIT ROUTINE
REPORT LI   R0,10*32+3   ROW 11, COL 4
       LI   R1,PGSTR     PAGES STRING
       BL   @DISSTR      DISPLAY THAT
       A    R2,R0        ADD LENGTH TO POINTER
       MOV  @AMS,R3      GET NUMBER OF PAGES PAST 15
       MOV  R3,@>835E    PUT AT >835E
       BL   @SHWINT      SHOW NUMBER ON SCREEN
       LI   R0,12*32+3   ROW 13, COL 4
       LI   R1,FRMSTR    FRAMES STRING
       BL   @DISSTR      SHOW THAT
       A    R2,R0        ADD LENGTH
       CLR  R2           CLEAR R2
       LI   R1,3         SET R1 TO 3
       DIV  R1,R2        DIVIDE R2-R3 BY 3
       AI   R2,4         ADD 4 FOR PAGES 4 THRU 15
       MOV  R2,@>835E    QUOTIENT PLUS 4 (FRAMES) TO >835E
       BL   @SHWINT      SHOW THAT NUMBER (1/3 OF PAGES NUM + 4= FRAMES)
KEYEX  BLWP @KSCAN       SCAN KEYBOARD
       LIMI 2            ALLOW INTS
       LIMI 0            STOP INTS
       CB   @ANYKEY,@STATUS KEY STRUCK?
       JNE  KEYEX        IF NOT, REPEAT
       LWPI GPLWS        LOAD GPL WORKSPACE
       B    @>6A         BACK TO GPL INTERPRETER
*

* SUBROUTINES SECTION
*
* AMSINI SETS AMS CARD TO "POWER-UP" CONDITION
*
AMSINI LI   R12,>1E00    AMS CRU BASE
       SBO  0            TURN ON AMS
       LI   R1,>FEFF     (THIS IS ->0101)
       LI   R0,>4000     START OF MEMORY
AMSLP  AI   R1,>0101     ADD 1 PAGE
       MOV  R1,*R0+      MOVE 2 BYTES TO MEM-MAPPER
       CI   R0,>4020     ALL DONE?
       JLT  AMSLP        NO, INIT MORE
       RT                RETURN
*
* DISSTR DISPLAYS STRING POINTED BY R1 AT SCREEN
* LOCATION POINTED TO BY R0
*
DISSTR MOVB *R1+,R2      LENGTH BYTE TO R2
       JEQ  DISX         IF ZERO, SKIPIT
       SRL  R2,8         RT JUST
       BLWP @VMBW        WRITE STRING TO SCREEN
       A    R2,R1        ADD LENGTH TO POINTER
DISX   RT
*
* FOR SHWINT, FIRST PUT INTEGER AT >835E,
* AND POINT R0 AT DESIRED LOCATION
* THEN BL  @SHWINT
*

SHWINT BLWP @GPLLNK      USE GPLLNK VECTOR
       DATA >2F7C        DATA FOR INT TO STRING
       MOVB @>8361,R2    LENGTH TO R2
       SRL  R2,8         RT JUST.
       MOVB @>8367,R1    ADDR TO R1
       SRL  R1,8         RT JUST
       AI   R1,>8300     ADD OFFSET
       BLWP @VMBW        WRITE STRING
       RT
```

```
*  GENERAL PURPOSE GPL LINK
*  by Doug Warren/Craig Miller
*
GPLLNK DATA GLNKWS
       DATA GLINK1
RTNAD  DATA XMLRTN
GXMLAD DATA >176C
       DATA >50
GLNKWS EQU  $->18
       BSS  >08
GLINK1 MOV  *R11,@GR4
       MOV  *R14+,@GR6
       MOV  @XTAB27,R12
       MOV  R9,@XTAB27
       LWPI GPLWS
       BL   *R4
       MOV  @GXMLAD,@>8302(R4)
       INCT @STKPNT

       B    @LDGADD
XMLRTN MOV  @GETSTK,R4
       BL   *R4
       LWPI GLNKWS
       MOV  R12,@XTAB27
       RTWP
*

* DATA SECTION
*
WS     BSS  32           OUR REGISTERS
AMS    DATA 0            PAGES BEYOND 15
NOSTR  BYTE 19
       TEXT 'NO AMS CARD PRESENT'
PGSTR  BYTE 14
       TEXT 'PAGES FOUND = '
FRMSTR BYTE 9
       TEXT 'FRAMES = '
ANYKEY BYTE >20
       END
*
* END OF COMPLETE PROGRAM
*
```

```
* PART TWO OF SIDEBAR - A "SNIPPET"
*
* SUBROUTINE SETFRM (FROM AMS VIDEO TITLER)
* ON ENTRY, R3 CONTAINS THE CURRENT FRAME NUMBER (1 THRU XX)
* WHERE XX CAN BE 9 (128K CARD), 20 (256K CARD), OR 41 (512K CARD)
* PAGE NUMBERS START WITH 0, SO FOR EXAMPLE ON A 256K CARD, PAGES
* ARE NUMBERS 0 THRU 63, FOR A TOTAL OF 64 PAGES
*
* THE PROGRAM "KNOWS" HOW MANY FRAMES ARE AVAILABLE, AND
* WON'T USE THIS SUBROUTINE BEYOND THE AVAILABLE FRAMES.
*
SETFRM MOV  R3,R1        COPY CURRENT FRAME # TO R1
       MPY  @THREE,R3    MULTIPLY R3 BY 3
       INC  R4           ADD 1 TO PRODUCT IN R4 (PAGE NUMBER ON AMS = 4 THRU YY)
* YY IS THE HIGHEST PAGE -2
       SWPB R4           SWAP TO LEFT BYTE
       COC  @ONEWD,R1    CHECK CURRENT FRAME FOR ODD/EVEN
       JNE  SETEVN       IF NOT 1 IN LSB, EVEN
SETODD LI   R8,>4014     MAP FOR >A000
       LI   R1,>A000     SET R1=>A000
       JMP  SETPGS       JUMP
SETEVN LI   R8,>401A     MAP FOR >D000
       LI   R1,>D000     SET R1=>D000
SETPGS LI   R12,>1E00    AMS CRU ADDR
       SBO  0            TURN ON CARD
       SBO  1            TURN ON MAPPER
       LI   R5,3         THREE WRITES TO MAKE (3 PAGES PER FRAME)
SETLP  MOV  R4,*R8+      SET CURRENT PAGE AND INCT R8

       AI   R4,>100      NEXT PAGE
       DEC  R5           DEC COUNT
       JNE  SETLP        RPT IF NOT 0
       SBZ  0            TURN OFF CARD MEMORY (>4000)
* THE MAPPER STAYS ON
       RT                RETURN

* UPON RETURN, R1 IS USED TO DETERMINE WHERE THE FRAME IS SENT OR GOTTEN FROM
* DEPENDING WHETHER WE'RE WRITING OR READING A FRAME
*
* END OF PART 2 OF SIDEBAR
```

Addendum: Accessing SAMS with Camel99 Forth

The SAMS card can be accessed with Forth. This takes a little more space than using machine code operations, however it is a little more convenient to modify. This example code lets you access multiple 64k segments via a 4K window in memory located at HEX 3000. The key word is called PAGED. Given an address PAGED decides is that memory is already sitting in the window or not. If not it, it flips the correct range of the 64k segment into the 4k window. Using PAGED, we create four words to access SAMS memory by byte or word.

```
\ SAMS CARD support. 64K segmented memory fetch and store

NEEDS SBO   FROM DSK1.CRU2

HEX
     VARIABLE BANK#      \ current mapped bank
1000 CONSTANT 4K         \ bytes per bank = 4K
3000 CONSTANT PMEM       \ paged memory block location
     VARIABLE SEG        \ holds current 64K segment

\ safely set the 64K segment that you want to use
: SEGMENT ( 1..F -- )
          DUP 01 10 WITHIN 0= ABORT" BAD segment selected"
          SEG ! ;  \ don't allow segment 0

1 SEGMENT ( set default segment)

: SAMS-OFF ( --) 1E00 CRU! 1 SBZ ;
: SAMS-ON  ( --) 1E00 CRU! 1 SBO ;

\ * SAMSINI sets card to "power-up" condition
: SAMSINI
       1E00 CRU! 0SBO    \ turn on card
       0                 \ 1st value
       4000 20           \ register address, #regs
       BOUNDS
       DO
           DUP I !       \ I is reg. address
           0101 +        \ next value
       2 +LOOP
       0SBZ              \ turn off card
       DROP
;

: PAGED  ( addr -- addr')
         SEG @ 4K UM/MOD  ( -- offset bank#)
         DUP BANK# @ =             \ are we using the same PAGE
         IF
             DROP                  \ Yes! Drop bank# and get out
         ELSE
             DUP FF00 AND ABORT" SAMS Err!"
             DUP BANK# !           \ update bank# variable
             ><                    \ swap bytes, bank# must be in left byte
            1E00 CRU! 0SBO         \ enable SAMS card
           ( bank#) 4006 !         \ store bank in 3K SAMS register
             0SBZ                  \ disable SAMS card
        THEN  PMEM +               \ then add offset to paged mem block
;

\ paged memory fetch and store
: C@P    ( addr -- n)    PAGED C@ ;   \ fetch a byte
: C!P    ( n 32addr -- ) PAGED C! ;   \ store a byte
: @P     ( 32addr -- n)  PAGED @ ;    \ fetch an int
: !P     ( n 32addr -- ) PAGED ! ;    \ store an int
```

SAMS Test Code

Timings were made with machine code version. Forth is approximately 45% slower on byte and word access operations.

```
HEX
7FFF CONSTANT 32K
FFFF CONSTANT 64K
1000 CONSTANT 4K

1 SEGMENT  \ select the segment we will use

: ERASE    0 FILL ;
: BLANKS  BL FILL ;

\ 64k single byte writes to paged memory
: 64KBYTES    64K 0 DO  I     I C!P    LOOP ; ( 47 secs)

\ 64K single byte writes to single address
: 64KBTEST    64K 0 DO  I  3000 C!    LOOP ; ( 15 secs)

\ 32k word writes to paged memory
: 32KWORDS    64K 0 DO  I  I     !P   2 +LOOP ; ( 25.5 secs)

\ 32K word writes to single address
: 32KTEST     64K 0 DO  I  3000  !    2 +LOOP ; ( 9.6 secs)

\ 4K block fill to paged memory
: 64KBLANKS   64K 0 DO I PAGED 4K BLANKS  4K +LOOP ; ( 1.5 secs)

\ 4K block fill to CPU memory
: 64KTEST     64K 0 DO  3000 4K BLANKS    4K +LOOP ; ( 1.5 secs)

: SEE32K      64K 0 DO  I @P . 2+ LOOP ;

: 64KERASE    64K 0 DO I PAGED 4K ERASE   4K +LOOP ;

: ERASEALL    10 1 DO  I SEG !   64KERASE   LOOP ;  ( 20.7 secs)
```