SAMS Memory Expansion Guide

Author: Lee Stewart

Introduction

SAMS and 32 KiB Expansion RAM (ER) cards are mutually exclusive. A SAMS card in the system will supply the 32 KiB ER. It is important to note that there is no DSR on a SAMS card. The only consequence of setting the card's (CRU address = 1E00h) bit 0 is that access to the SAMS mapper registers is enabled. After powerup, the card's 0 and 1 bits are cleared. This puts the SAMS card in transparent mode (see below) and disables access to the mapper registers. Of the 16 mapper registers, only 8 are active. Though the SAMS registers are only 8 bits (1 byte) to manage each bank's 256 4-KiB pages of SAMS RAM, they are 16 bits apart for two reasons:

1. They only respond to even addresses.
2. A SAMS card with more than 1 MiB RAM has the bits above 255 (00FFh) tied to a latch that selects a bank of 256 SAMS pages.

The mapper registers start at the beginning of the 4000h memory space, which is why they are only accessible for reading/writing by setting the card's bit 0. This is typically thought of as turning on the card for access to the 8-KiB memory space, 4000h..5FFFh. Each SAMS register manages the mapping of a 4-KiB SAMS page to a 4-KiB window of CPU RAM. Each RAM window starts on a 4-KiB boundary of the TMS9900's 64-KiB memory space. Table 1 shows the SAMS mapper registers and the memory windows to which they map SAMS pages (unavailable SAMS mapper registers and memory windows are shaded).

Table 1: SAMS Mapper Registers and their Expansion RAM windows

SAMS Mapper Register		Memory Window Address
Number	**Address**	
0	4000h	0000h
1	4002h	1000h
2	4004h	2000h
3	4006h	3000h
4	4008h	4000h
5	400Ah	5000h
6	400Ch	6000h
7	400Eh	7000h
8	4010h	8000h
9	4012h	9000h
10	4014h	A000h
11	4016h	B000h
12	4018h	C000h
13	401Ah	D000h
14	401Ch	E000h
15	401Eh	F000h

Transparent Mode

Transparent mode is the powerup mode, *i.e.*, default mode, for the SAMS card. This mode is in effect when the card's bit 1 is cleared (SBZ 1). Transparent mode is identical to mapping a SAMS bank 0 page, numerically equivalent to said SAMS mapper register number, to a mapper register's ER window, as shown in Table 2. The programmer should initialize the SAMS card to this mapping before leaving transparent mode to avoid unpleasant surprises.

Table 2: SAMS Transparent Mode Mapping to Expansion RAM

SAMS Register #	SAMS (Bank 0) Page #	Memory Window Address
2	2	2000h
3	3	3000h
10	10	A000h
11	11	B000h
12	12	C000h
13	13	D000h
14	14	E000h
15	15	F000h

Mapping Mode

Mapping mode is enabled by setting bit 1 (SBO 1). Changing this mode bit does not require enabling access to the mapper registers (bit 0). Mapped SAMS memory is also always available when bit 1 is set regardless of the state of bit 0. SAMS memory should be initialized to the transparent-mode mapping of Table 2 before entering mapping mode to avoid accidentally mapping out of a working program.

Mapping SAMS Pages

To map a SAMS page to an ER window, its bank and page numbers are written via a single MOV (16-bit word) to the SAMS register of the relevant ER window. Because the MUX circuitry for the 8-bit PEB bus writes a word in LSB-MSB order and an 8-bit SAMS register only responds to an even address, both bytes are written to the 8-bit SAMS register, with only the MSB surviving the transfer (more later). This means that the SAMS page must be in the MSB. Because the bank latch only responds to the LSB of the MOV, the bank number must appear in the LSB.

I think it is simpler to manage SAMS memory without considering the bank/page hierarchy. Though I still use "page" to describe each 4 KiB segment of SAMS memory in **fbForth**, I will use "segment" for these degenerate (non-hierarchical) pages in this discussion to avoid confusion. Because a bank can contain only 256 pages (0..255), the highest page number fills exactly 1 byte. This allows the "segment" idea to work because adding 1 to segment 255 (00FFh or bank 0, page 255) is segment 256 (0100h or bank 1, page 0). Here are a few segments with their corresponding bank/page representations:

- Segment 255: 00FFh (bank 0, page 255)
- Segment 256: 0100h (bank 1, page 0)
- Segment 400: 0190h (bank 1, page 144)
- Segment 511: 01FFh (bank 1, page 255)
- Segment 3119: 0C2Fh (bank 12, page 47)

- Segment 4095: 0FFFh (bank 15, page 255) <--highest possible with current **SAMS** circuitry
- Segment 8191: 1FFFh (bank 31, page 255) <--only possible in the **Classic99** emulator

To write such a segment word to a **SAMS** mapper register requires swapping bytes before the MOV to properly effect the desired mapping.

The procedure for mapping a **SAMS** segment to an **ER** window, *e.g.*, segment 400 to A000h, is as follows:

1.	Set the **CRU** register (R12) to 1E00h, the address of the SAMS card:	`LI  R12,>1E00`
2.	Put segment to map into a **CPU** register, *e.g.*, R0:	`LI  R0,400`
3.	Swap bytes for writing to **SAMS** mapper register:	`SWPB R0`
4.	Enable access to mapper registers by setting bit 0:	`SBO  0`
5.	Write R0 to **SAMS** mapper register 10:	`MOV  R0,@>4014`
6.	Disable access to mapper registers by clearing bit 0:	`SBZ  0`
7.	Enable mapping (if not yet enabled) by setting bit 1:	`SBO  1`

Reading the Mapper Registers

You can read the value in a **SAMS** mapper register with the MOV instruction, but, given that an 8-bit **SAMS** register only responds to an even address and that its mapped bank # (MSB of segment # before swapping bytes) is not stored, the value returned will have a copy of its page # (LSB of segment # before swapping bytes) in both bytes. This means that by reading a **SAMS** register, you can only ever retrieve the mapped segment's page #, never its bank #. For example, pointing **SAMS** mapper register 3 (**ER** window = 3000h) to segment 1841 (0731h) with

```
LI   R12,>1E00     set SAMS card CRU address
LI   R0,>0731      load segment 1841 (>0731)
SWPB R0            swap bytes to >3107 for SAMS mapping
SBO  0             enable access to SAMS mapper registers
MOV  R0,@>4006     to SAMS mapper reg 3 for RAM window at >3000
SBZ  0             disable access to SAMS mapper registers
```

when read back with

```
LI   R12,>1E00     set SAMS card CRU address
SBO  0             enable access to SAMS mapper registers
MOV  @>4006,R0     value in SAMS mapper reg 3 to R0
SBZ  0             disable access to SAMS mapper registers
```

will return 3131h in R0. That said, I should note that the **Classic99** emulator does, in fact, return the value previously written and in the same order, *viz.*, in page#-bank# order. Consequently, the above example will return 3107h in R0 in **Classic99**.

TMS9900 Assembly Language Code (ALC) Routines

The following ALC routines are slightly modified from the actual routines I use in **fbForth** to manage SAMS memory:

SMSINI—Initialize SAMS card

```
*****************************************************************
****** SMSINI  [SAMS card initialization routine] ***************
*****************************************************************
***                                                           ***
*** Originally ported from TurboForth code courtesy of Mark Wills ***
*** and modified to explicitly set pages to power-up defaults,    ***
*** viz., pages >0002, >0003, >000A..>000F mapped to CPU RAM      ***
*** >2000, >3000, >A000..>F000, respectively.                    ***
***                                                           ***
*** This routine ends with SAMS mapping enabled, i.e., bit 1 set. ***
***                                                           ***
*** CAUTION:                                                  ***
***    This routine should be run from memory that is unaffected  ***
***    by SAMS mapping (ROM, Scratchpad RAM, ...) or with SAMS    ***
***    mapping disabled prior to loading, as at powerup.          ***
***                                                           ***
*****************************************************************

SMSINI LI   R12,>1E00         CRU address of SAMS card
       SBO  0                 enable access to mapper registers
       SBZ  1                 disable mapping while we set it up
       LI   R0,>4004          SAMS mapper register 2 for RAM >2000
       LI   R1,>0200          map SAMS page >0002...
       MOV  R1,*R0+           ...into RAM >2000..INCT to SAMS reg 3
       LI   R1,>0300          map SAMS page >0003...
       MOV  R1,*R0            ...into RAM >3000
*
* Now set up the banks for high memory...
*
       LI   R0,>4014          SAMS mapper register 10 for RAM >A000
       LI   R1,>0A00          start loop with SAMS page >000A
       LI   R2,6              loop count
SMSIN2 MOV  R1,*R0+           map next SAMS page..INCT to next SAMS reg
       AI   R1,>0100          INC SAMS page
       DEC  R2                finished?
       JNE  SMSIN2            loop if not
       SBO  1                 enable mapping
       SBZ  0                 lock the SAMS mapper registers
       RT                     return to caller
```

SMSCHK—Check for Presence of SAMS Card

```
*
*++ SAMS flag (SAMSFL)
*++     Value in SAMSFL is only relevant after running SMSINI.
*++     Its value reflects the highest available 4-KiB SAMS page.
*++     A value of 0, of course, reflects no SAMS card.
*
SAMSFL BSS  2           SAMS flag

*****************************************************************
****** SMSCHK  [Check for presence of SAMS card] *****************
*****************************************************************
***                                                           ***
*** SAMS flag (SAMSFL) will be set to highest available page #.  ***
***                                                           ***
*** A check-value of >994A is written to the starting address of  ***
*** the RAM window at >E000 before starting the test loop. The    ***
*** test loop will start with the highest possible amount of SAMS ***
*** memory (32 MiB), proceeding by halving the range until 128    ***
*** KiB is reached. 128 KiB is assumed to be the lowest viable    ***
*** SAMS. The actual page tested will be >000E higher than the    ***
*** lowest page in the upper half of the range. To test, map      ***
*** >000E + lowest page in upper half of SAMS range to >E000. For ***
*** 32 MiB, this is >100E. We initially store >0010 (LSB,MSB) in  ***
*** R3 to allow a circular right shift each round before MOVing   ***
*** to R0 to then add >0E00 (LSB,MSB) for the next test. If the   ***
*** test fails at >001E, the last viable SAMS (128 KiB), R3 will  ***
*** go to >0800, at which point the loop exits, setting R3 to 0,  ***
*** effectively reporting "no SAMS".                           ***
***                                                           ***
*** This routine should only be run after running SMSINI, which   ***
*** insures that SAMS memory is mapped as in transparent mode.    ***
*** This routine presumes that mapping is enabled, which is the   ***
*** case after running SMSINI.                                 ***
***                                                           ***
*****************************************************************

* Set up SAMS check.
*
SMSCHK LI   R2,>994A     check-value
       MOV  R2,@>E000    check-value to check-location
*
* Classic99 emulator can do 32 MiB
*
       LI   R3,>0010     lowest page # in upper half of SAMS to R3 (LSB,MSB)
       LI   R12,>1E00    CRU address of SAMS
SMSCK2 MOV  R3,R0        lowest page in upper half of SAMS range
       AI   R0,>0E00     get >000E pages higher
       SBO  0            enable SAMS mapper registers
       MOV  R0,@>401C    poke SAMS mapper register 14 for RAM >E000
       SBZ  0            disable SAMS mapper registers
       C    @>E000,R2    examine check-location for test value
       JNE  SMSXT1       exit if SAMS mapped, i.e., no match
       SRC  R3,1         shift right circularly to halve the SAMS range
       CI   R3,>0800     too far?
```

```
       JNE  SMSCK2       no..try SAMS half the size of last pass
       CLR  R3           yes..no SAMS, so set flag to 0
       JMP  SMSXT2       we're outta here
SMSXT1 SWPB R3           restore page #
       SLA  R3,1         double value (highest page # + 1)
       DEC  R3           decrement to highest page #
SMSXT2 MOV  R3,@SAMSFL   save SAMS flag
       JEQ  SMSEND       if no SAMS, no need to restore default mapping
*
* Remap default SAMS page >0E to RAM >E000.
* R12 should still have correct CRU address (>1E00).
*
       LI   R0,>0E00     load SAMS page >000E
       SBO  0            enable SAMS mapper registers
       MOV  R0,@>401C    poke SAMS mapper register 14 for RAM >E000
       SBZ  0            disable SAMS mapper registers
SMSEND RT                return to caller
```

MAP—Map a SAMS Page to a 4-KiB CPU RAM Window

```
*****************************************************************
****** MAP  [SAMS page-mapping routine] ************************
*****************************************************************
***                                                           ***
*** Originally ported from TurboForth code courtesy of Mark Wills ***
*** and modified to handle 128 KiB..32 MiB.                   ***
***                                                           ***
*** Inputs...                                                 ***
***    R1: Address of RAM mapping window, which should be a valid ***
***        address on a 4K boundary (e.g., >2000, >3000, >A000,   ***
***        >B000, >C000, >D000, >E000, >F000). R1 value is forced ***
***        to a 4-KiB boundary by virtue of the SRL instruction   ***
***        at the start of the routine and the fact that writing  ***
***        to a SAMS register address ignores the rightmost bit   ***
***        (LSb) of the address.                                  ***
***    R2: SAMS page # to map to above RAM window. R2 should be a ***
***        number between 0 and SAMSFL (maximum possible page of  ***
***        current SAMS card.                                     ***
***                                                           ***
*** It is presumed that mapping mode is set, so nothing is done   ***
*** with SAMS card bit 1 in this routine.                      ***
***                                                           ***
*** It is a good idea to check whether the SAMS page about to be  ***
*** mapped is higher than SAMSFL prior to calling this routine or ***
*** add that error check to the MAP routine itself. Code similar  ***
*** to the following would do the trick:                       ***
***                                                           ***
***    C    R2,@SAMSFL      requested SAMS page too high?         ***
***    JH   ERROR_HANDLER   yes..deal with the error             ***
***                                                           ***
*****************************************************************

MAP    SRL  R1,11        0-based location on card of SAMS register
       AI   R1,>4000     address on card of SAMS register
       SWPB R2           reverse byte order of page for SAMS register
       LI   R12,>1E00    CRU address of SAMS
       SBO  0            enable SAMS mapper registers
       MOV  R2,*R1       poke SAMS mapper register (LSb ignored)
       SBZ  0            disable SAMS mapper registers
       RT                return to caller
```