Commodore External RAM Expansion Cartridges

Dale A. Castello
Montgomery, AL

Transfer commands for your external storage area!

Editor's Note: *Although the 1700 and 1750 Expansion RAM modules will work on the C64, they draw about 200 milliamps and the C64 power supply can not handle the extra load. Should you wish to use either of these with the C64, you'll need a higher output power supply. However, the Commodore 1764 External RAM Expansion comes with a replacement power supply, and Dale's software will also work with the 1764. Naturally, the C128 supplies ample power for operating the expansion RAM in 64 mode with Dale's program.*

After many months of anticipation, the expansion RAM cartridge for the C128 is finally available at local stores and by mail. It comes in two versions: the 1700 contains 128K bytes of memory and the 1750 contains 512K bytes. Only the 1750 is readily available. This memory expansion cannot be directly addressed like the resident memory banks internal to the C128. Instead, access is established through the I/O space from $DF00 to $DF0A. Because the expansion cards use the computer's direct memory access (DMA) capability, a memory bank containing the C128 I/O space does not need to be turned on during the actual transfer. Commodore recommends that transfers be done with the 1MHz clock rate so as to avoid conflicts with the memory bus access. Transfers at 2MHz can be done, if the VIC screen is blanked and the instruction following the command execution does not make a write to memory.

The card offers four functions:

(1) FETCH – transfers from external RAM to internal RAM
(2) STASH – transfers from internal RAM to external RAM
(3) SWAP – exchanges internal and external RAM
(4) VERIFY – compares internal and external RAM

C128 BASIC implements the first three of these functions. The fourth function may be executed through use of pokes in C128 mode. A program to implement all four of these functions in C64 mode is discussed later in this article.

Physical Layout

The expansion RAM chips and DMA controller are housed in a C128-colored, plastic unit which is 5 1/4 inches wide and extends 4 1/2 inches behind the computer when plugged into the expansion port. There is no edge connector on the unit to permit other bus devices to be plugged into it. Inside the case are the DMA controller chip and 16 memory chips. The chips are either 64K by 1 bit for the 1700 or 256K by 1 bit for the 1750. Wire straps on the card indicate that Commodore designed the circuit card for 128K, 256K, and 512K byte configurations.

Internal Registers And Operation

The external RAM controller appears at I/O addresses $DF00 through $DF0A. Of these eleven addresses in the controller: one is for status, three for control, and the rest for addresses. All of the registers are read/write except the status register which is read only.

In order to activate an operation, the starting memory locations in internal and external RAM, the block size, some special options, and the command must be written to the controller. The actual transfer occurs either immediately following the write of the command or after the next bank switch of the C128. The latter feature permits the C128 banks to be reconfigured prior to the transfer so that memory under I/O may be transferred.

The internal computer RAM starting address is placed in $DF02/$DF03 in normal low/high byte order. The C128 bank configuration must be set in $FF00 or in location 1 if you are using a C64.

The external RAM is banked in increments of 64K bytes. Because it is only possible to address 64K memory locations using two bytes, the starting location in the external RAM requires three locations. The location is given in normal low to high order in $DF04 through $DF06. The values in $DF06 are limited to 0–1 for the 1700 and 0–7 in the 1750. If the block of data to be transferred extends across a bank boundary, the DMA controller automatically increments the bank register.

The size of the transfer is set in locations $DF07 and $DF08 in normal order. Transfers are limited to 64K bytes with all block sizes normal except size value of zero means 64K.

The DMA controller also permits an interrupt to be set when it completes its operation. Because the DMA controller disables normal CPU processing on the C128, this capability is not used on the C128. This means the interrupt must be processed by the user's program and will not be handled by the operating system. Location $DF09 is the interrupt mask for the controller. It works in the same way as the interrupt mask registers on other I/O devices. During a write, mask bit 7 determines if the interrupt will be enabled or disabled. Two conditions may be set: bit 6 causes an flag at the end of an operation and bit 5 sets a flag if a verify error occurs. The actual interrupt event is signalled by the setting of bit 7 in the status register. A read of $DF00 (the status register) will indicate which event caused the interrupt. Bits 6 and 5 of the status register have the same meaning as in the interrupt mask register. A read of the status register is destructive and will clear bits 5–7.

The status register has one more bit of interest. Bit 4 indicates whether a 1700 or 1750 is attached. If the bit is set, a 1750 is attached; otherwise, a 1700 is attached. The last two registers determine the operation of the controller. The register at $DF01 is called the command register and the one at $DF0A is the address control register.

During normal operation you will want both the internal and external addresses to increment as each byte is transferred. There are special cases where you would want to hold one address constant, such as a direct transfer with I/O. Bits 6 and 7 at $DF0A are normally zero which permits both addresses to increment. If bit 7 is set, the C128 address will be fixed. If bit 6 is set, the external RAM address will be fixed.

The register at $DF01 is the command register. It is set after all the other registers are set and determines the function to be performed. All bits must be set during a single write to the register. Bit 7 must always be set and it executes the function specified by the other bits and registers. Setting bit 5 enables the auto-reload feature. This causes the initial internal memory start address, the external memory start address, and block length to be reset after the function is completed to their values before the function. This option is of value if the same addresses are used repeatedly, such as the VIC screen in computer memory. The user need only set the addresses which change between commands. A disadvantage of the auto-reload feature is that the reload will occur even after an error is found during a verify operation. This destroys the address pointers to the errored byte.

Setting bit 4 enables the bank switch delay. When selected, the actual DMA transfer will not occur until the C128 bank is set by a store to location $FF00. This is the mode of operation used by C128 BASIC. It will not function properly in C64 operation. Finally, bits 0–1 of the command determine the function:

Bit	Function
0 0	Transfer from internal to external RAM (STASH)
0 1	Transfer from external to internal RAM (FETCH)
1 0	Exchange internal and external RAM (SWAP)
1 1	Compare internal and external RAM

After an operation is complete, the address registers will advance by the length register. The length register will be set to one unless auto-reload is enabled. If there is a bad byte detected during a verify operation, the internal and external address registers will point to one location beyond the mismatch.

C64 Operation

There are no commands built into the C64 BASIC to support the external RAM. Therefore, the program accompanying this article provides a BASIC extension of four new commands. The syntax of the commands is the same as in the C128 BASIC except an "@" has been added in front of each. The "@" is part of the keyword and no space should follow it. Any valid expression may be used for the arguments.

@FETCH <length>,<C64 addr>,<RAM addr>,<RAM bank>
@STASH <length>,<C64 addr>,<RAM addr>,<RAM bank>
@SWAP <length>,<C64 addr>,<RAM addr>,<RAM bank>
@COMPARE <length>,<C64 addr>,<RAM addr>,<RAM bank>

Where:
<length> range 0–65535 is size of memory block (0 means 64K)
<C64 addr> range 0–65535 is starting loc. in computer memory
<RAM addr> range 0–65535 is starting loc. in expansion mem.
<RAM bank>is expansion memory bank range 0–1 for 1700
range 0–7 for 1750

The wedge is activated by SYS 52992 and deactivated by SYS 53020. Care has been taken to permit other wedges to coexist with the expansion RAM wedge provided it is the last wedge activated. The code has been compacted so that it fits in $CF00–$CFFF.

Applications

The application program provided in this article will permit the graphics examples contained on the expansion–RAM demonstration disk to be executed on a C64, provided changes are made to C128 tokens and the graphics screen is properly positioned. Other graphics programs may also be modified. The author is currently working on a virtual disk which will permit some graphics adventure games to be played without disk access.

The availability of the space of three single sided disks at 1MHz transfer rates permits a entirely new realm of games and applications to be considered. One application I use is to place my assembler on RAM and "fetch" it into memory when ever I am ready to run it. I have also written a package to copy and modify text adventure games to use the external RAM. Text adventure games which have a lot of disk access come "alive" when RAM instead of disk is used. High speed, single drive copying of filled, single and double–sided disks without disk swapping is great.

Figure 1: C64/C128 Expansion RAM Register Definition

		7	6	5	4	3	2	1	0
$DF00	Status	Interrupt	End Function	Verify Error	512k RAM	Version Number			
$DF01	Command	Execute	Reserved	Auto-Load	No $FF00	Reserved	Reserved	Transfer Type 0-3	
$DF02	C128 Start	Low-Byte							
$DF03	Address	High-Byte							
$DF04	External	Low-Byte							
$DF05	RAM Start	High-Byte							
$DF06	Address	Bank-Byte 0-1(1700) 0-7(1750)							
$DF07	Block	Low-Byte ($0000 means $10000)							
$DF08	Length	High-Byte							
$DF09	Intr.Mask	On/Off	End Function	Verify Error	reserved				
$DF0A	Addr.Cntrl	Fix C128 Add	Fix RAM Addr	reserved					

Expansion RAM Commands: BASIC Loader

```
LN  1000 rem save' '0:xram64.bas' ',8
KF  1010 rem ** this program will create
HL  1020 rem ** a load and run module on
HF  1030 rem ** disk called ' 'xram64.obj' '
OF  1040 open 15,8,15: open 8,8,1,' '0:xram64.obj' '
PJ  1050 input#15,e,e$,b,c: if e then close 15
         : print e;e$;b;c: stop
LD  1060 for j=52992 to 53244: read x: print#8,chr$(x);
         : ch=ch+x: next: close8
GO  1070 if ch<>30308 then print ' 'checksum error!' '
         : stop
KO  1080 print ' '** finished! **' '
--  1090 print ' 'load xram64.obj,8,1 and sys52992 to enable' '
GJ  1100 print ' 'sys53020: rem to disable' '
GF  1110 end
EN  1120 :
FC  1130 data   0, 207, 162,  70, 160, 207, 204,   9
LI  1140 data   3, 240,  18, 173,   8,   3, 141,  68
JK  1150 data 207, 173,   9,   3, 141,  69, 207, 142
IA  1160 data   8,   3, 140,   9,   3,  96, 174,  68
FE  1170 data 207, 172,  69, 207, 200, 240,   7, 136
LP  1180 data 142,   8,   3, 140,   9,   3,  96,  83
PH  1190 data  84,  65,  83, 200,  70,  69,  84,  67
ID  1200 data 200,  83,  87,  65, 208,  67,  79,  77
AP  1210 data  80,  65,  82, 197,   0,  76, 255, 255
MK  1220 data 160,   0, 132,   2, 200, 177, 122, 201
FG  1230 data  64, 208, 242, 162,   0, 200, 177, 122
JA  1240 data  56, 253,  45, 207, 208,   3, 232, 208
JD  1250 data 244,  56, 233, 128, 208,   2, 240,  17
DE  1260 data 189,  45, 207,  48,   5, 240, 214, 232
IN  1270 data 208, 246, 230,   2, 232, 160,   1, 208
OM  1280 data 220, 200, 152,  24, 101, 122, 133, 122
CE  1290 data 144,   2, 230, 123,  32, 245, 207, 140
EP  1300 data   7, 223, 141,   8, 223,  32, 242, 207
JN  1310 data 140,   2, 223, 141,   3, 223,  32, 242
MN  1320 data 207, 140,   4, 223, 141,   5, 223,  32
OD  1330 data 242, 207, 201,   0, 240,   3,  76,  72
AF  1340 data 178, 173,   0, 223,  41,  16, 240,   4
OI  1350 data 192,   8, 144,   4, 192,   2, 176, 238
EE  1360 data 140,   6, 223, 165,   2, 160,   0, 140
HL  1370 data  10, 223, 140,   9, 223, 120, 162, 245
EF  1380 data 164,   1, 134,   1,  44,   0, 223,   9
EF  1390 data 144, 141,   1, 223, 165, 122, 208,   2
MJ  1400 data 198, 123, 198, 122, 173,   0, 223, 141
AJ  1410 data  12,   3, 173,   2, 223, 141,  13,   3
GP  1420 data 173,   3, 223, 141,  14,   3, 132,   1
JM  1430 data  88,  76,  67, 207,  32, 253, 174,  32
OB  1440 data 158, 173,  76, 247, 183
```

PAL Source Listing

```
EN  1000 rem save' '0:xram64.pal' ',8
GF  1010 rem ** pal 64 format **
LJ  1020 open 8,8,1,' '0:xram64.obj' '
HN  1030 sys700
DD  1040 .opt o8
HD  1050 *=$cf00
KJ  1060 ;
LH  1070 ; a program to implement external
IP  1080 ; ram function on a c-64 or
PB  1090 ; c128 in c64 mode
CM  1100 ;
IP  1110 ; dale a. castello
LJ  1120 ; 5964 oakleigh rd
DB  1130 ; montgomery al 36116
KO  1140 ;
CF  1150 ; implements basic extensions
KC  1160 ; @stash <bytes>,<addr1>,<addr2>,<bank>
HO  1170 ; @fetch <bytes>,<addr1>,<addr2>,<bank>
MN  1180 ; @compare <bytes>,<addr1>,<addr2>,<bank>
CN  1190 ; @swap <bytes>,<addr1>,<addr2>,<bank>
GC  1200 ;
NA  1210 ; where
ND  1220 ; <bytes> = number of bytes to transfer 0-65535
MM  1230 ;                0 => 65536 bytes
JK  1240 ; <addr1> = computer start address 0-65535
JG  1250 ; <addr2> = ram start address 0-65535
MP  1260 ; <bank>  = ram bank number
KN  1270 ;                0-1 for 1700
PO  1280 ;                0-7 for 1750
AI  1290 ;
EE  1300 ;  activate    sys 52992 ($cf00)
FN  1310 ;  deactivate sys 53020 ($cf1c)
OJ  1320 ;
FF  1330 ; on exit
JD  1340 ; areg  status  $20 okay
LB  1350 ;               $40 verify error
GM  1360 ;
CA  1370 ; xreg/yreg last computer address
KN  1380 ;
GG  1390 cmd     =    2        ;expansion command
DE  1400 txtptr  =    $7a      ;current byte of basic text
FD  1410 areg    =    $30c     ;storage of a reg
LM  1420 xreg    =    $30d     ;storage of x reg
NN  1430 yreg    =    $30e     ;storage of y reg
DO  1440 igone   =    $308     ;basic token eval
```

```
PL 1450 exp      =    $df00         ;dma controller
OJ 1460 c64      =    exp+2
DF 1470 ram      =    exp+4
LA 1480 bank     =    exp+6
LK 1490 leng     =    exp+7
CF 1500 ;
JL 1510 active   = *
IP 1520          ldx  #<parse
CA 1530          ldy  #>parse
LN 1540          cpy  igone+1       ;if page $cf
PD 1550          beq  inpl          ;already installed
OI 1560 ;
HG 1570          lda  igone
KG 1580          sta  oldvec+1
HI 1590          lda  igone+1
CI 1600          sta  oldvec+2
JC 1610          stx  igone
DE 1620          sty  igone+1
EN 1630 ;
IN 1640 inpl     = *
OF 1650          rts
CP 1660 ;
HL 1670 inact    = *
MO 1680          ldx  oldvec+1
OP 1690          ldy  oldvec+2
DE 1700          iny                ;if $ff is hi addr
FD 1710          beq  nogo          ;don't restore
OC 1720 ;
NH 1730          dey
LK 1740          stx  igone
FM 1750          sty  igone+1
GF 1760 ;
JK 1770 nogo     = *
AO 1780          rts
EH 1790 ;
HB 1800 table    = *
EH 1810          .asc ' 'stas' '
AD 1820          .byte $c8
GF 1830          .asc ' 'fetc' '
EE 1840          .byte $c8
ML 1850          .asc ' 'swa' '
EF 1860          .byte $d0
HN 1870          .asc ' 'compar' '
BF 1880          .byte $c5,0
IN 1890 ;
LB 1900 oldvec   = *
LE 1910          jmp  $ffff         ;address set to old error vector on activation
GP 1920 ;
IN 1930 parse    = *
HJ 1940          ldy  #0            ;scan basic text
OF 1950          sty  cmd           ;initial command number
ND 1960          iny                ;point to next character
CL 1970          lda  (txtptr),y
IB 1980          cmp  #' '@' '
NN 1990          bne  oldvec        ;no leading @
GE 2000 ;
CF 2010          ldx  #0            ;init table pointer
KF 2020 ;
MO 2030 nxt      = *
GP 2040          iny                ;get next input character
CA 2050          lda  (txtptr),y
NI 2060          sec
PA 2070          sbc  table,x       ;check text
MK 2080          bne  last          ;may be shifted
AK 2090 ;
HL 2100          inx                ;okay so far
PN 2110          bne  nxt           ;loop for next match
OL 2120 ;
FL 2130 last     = *
EO 2140          sec                ;check for shifted
IL 2150          sbc  #$80          ;check for shifted
JG 2160          bne  skip          ;character
AP 2170 ;
KM 2180          beq  found         ;matchs string
EA 2190 ;
NC 2200 ; no match found so advance to
HC 2210 ; next command string
CC 2220 ;
JB 2230 skip     = *
JJ 2240          lda  table,x
FB 2250          bmi  nxcmd         ;reached shifted char
KE 2260 ;
KJ 2270          beq  oldvec        ;error end of table
OF 2280 ;
OM 2290          inx
GH 2300          bne  skip          ;keep going
MH 2310 ;
CD 2320 nxcmd    = *
PD 2330          inc  cmd
AA 2340          inx
CB 2350          ldy  #1            ;dim in basic text
JH 2360          bne  nxt           ;search next command
IL 2370 ;
NM 2380 ; we have found the match
EN 2390 ; read parameters
GN 2400 ;
DN 2410 found    = *
KD 2420          iny                ;update basic pointer
FD 2430          tya
OP 2440          clc
EC 2450          adc  txtptr
KH 2460          sta  txtptr
AL 2470          bcc  nopage
GC 2480 ;
MD 2490          inc  txtptr+1
KD 2500 ;
JK 2510 nopage   = *
KN 2520          jsr  getint        ;get # bytes
JD 2530          sty  leng
FK 2540          sta  leng+1
BI 2550          jsr  arg           ;get c64 memory start
BF 2560          sty  c64
EN 2570          sta  c64+1
KC 2580          jsr  arg           ;get external ram start
PN 2590          sty  ram
CG 2600          sta  ram+1
IB 2610          jsr  arg           ;get bank
IO 2620          cmp  #0            ;check if out of range
KB 2630          beq  limit
GM 2640 ;
ED 2650 toobig   = *
AA 2660          jmp  $b248         ;illegal quantity
EO 2670 ;
GN 2680 limit    = *
HK 2690          lda  exp
LM 2700          and  #$10          ;check ram size
HF 2710          beq  r128
GB 2720 ;
BE 2730          cpy  #8            ;max bank for 512k +1
NN 2740          bcc  inside
ED 2750 ;
HG 2760 r128     = *
KF 2770          cpy  #2            ;max bank for 128k +1
IO 2780          bcs  toobig
MF 2790 ;
MM 2800 inside   = *
KN 2810          sty  bank
JA 2820          lda  cmd
JF 2830          ldy  #0
OF 2840          sty  exp+10        ;inc pointers
OB 2850          sty  exp+9         ;no interrupts
FI 2860          sei                ;open ram
HO 2870          ldx  #$f5          ;under basic and kernel
NK 2880          ldy  1             ;old value
IN 2890          stx  1             ;temp value
PD 2900          bit  exp           ;reset dma controller
DD 2910          ora  #$90          ;form command
CL 2920          sta  exp+1
JK 2930          lda  txtptr        ;dim in basic text
KC 2940          bne  notb
MP 2950 ;
BO 2960          dec  txtptr+1      ;page boundry
AB 2970 ;
OC 2980 notb     = *
MH 2990          dec  txtptr        ;single byte
DL 3000          lda  exp           ;return result
ID 3010          sta  areg
DH 3020          lda  c64           ;return last address
KL 3030          sta  xreg          ;accessed in computer
MG 3040          lda  c64+1
IH 3050          sta  yreg
LP 3060          sty  1             ;restore ram configuration
JP 3070          cli                ;interrupts on
OK 3080          jmp  oldvec        ;back to basic
II 3090 ;
HG 3100 ;subroutine to evaluate argument
MJ 3110 ;
OM 3120 arg      = *
HP 3130          jsr  $aefd         ;must have comma
KL 3140 ;
DC 3150 getint   = *
FD 3160          jsr  $ad9e         ;eval expression
LA 3170          jmp  $b7f7         ;fix it
CO 3180 ;
CF 3190 .end
```