11 Serially accessed ROMs and the *ROM filing system

The Electron has been designed to use software contained in ROM cartridge packs. The ROM packs which plug into the Plus 1 expansion may contain up to two paged ROMs, The ROM pack paged ROMs may contain up to about 16K of data and/or programs which is paged into memory as required. On the BBC microcomputer the facility also extends to phrase ROMs (PHROMS) associated with the speech upgrade. When the programs or data stored in these ROM packs are required it may be loaded into user RAM in the same way as programs or data may be loaded off tape or disc.

These ROM packs are intended to provide a reliable and rapidly accessible medium for the distribution of programs. The market for such a product being amongst owners of tape based machines who would otherwise have to rely upon the much slower and inherently less reliable medium.

The advantage to the software producer is that there is no requirement for a special version of the program to be written. A system is required for the formatting of the program for inclusion in a ROM pack but no modification of the program itself is required.

The *ROM filing system is a subset of the tape filing system. Paged ROMs are interrogated to determine whether they contain information intended for this filing system and are then serially accessed by the *ROM filing system.

Paged ROMs containing information intended for access via the *ROM filing system are no different from other paged ROMs. They are service type ROMs and as such have sevice entry points. They are distinguishable as *ROM filing system ROMs only by their response to paged ROM service calls issued by the *ROM filing system code. When the user selects the *ROM filing system any further requests for files result in the *ROM filing system section of the operating system scanning the paged ROMs for these files. A paged ROM containing files intended for the *ROM filing system should respond to one of two paged ROM service calls.

The two service calls and the responses expected from ROMs containing *ROM data are described in detail below. One call expects the ROM to prepare to yield any data it has and the second call is used to extract this data, one byte at a time. The data should be formatted in a similar way to the data stored on tape but is modified in such a way as to minimise the storage overheads involved in using such a format. The reason for adopting this format is to minimise the requirements for extra code in the operating system while utilising the exhaustive error checking already in existence. Accompanying these advantages there is a concurrent reduction in response time performance but this is of little importance to the users of tape based machines who are still able to appreciate a substantial improvement on their system's existing performance.

11.1 Converting files to *ROM format

In order to produce a ROM containing files which will be recognised by the *ROM filing system it is necessary to fulfill two criteria, The first requirement is for some header code which will recognise the *ROM filing system paged ROM service calls and respond accordingly. The second requirement is that the data which makes up the files is formatted in the manner in which the *ROM filing system expects to find it.

11.2 The header code

As has been stated above a paged ROM which is to be recognised by the *ROM filing system is a perfectly standard paged ROM which responds to the apporpriated service calls. As a result of this requirement the first part of each *ROM filing system ROM consists of a standard format paged ROM header followed by a small amount of code which responds to the necessary service calls. By convention *ROM paged ROMs do not respond to the

*HELP sevice call but should these ROMs anounce their presence in this way it would obviously leave less space for programs and data.

The two paged ROM service calls which should elicit a response from *ROM paged ROMs are described in the next two paragraphs.

11.3 Paged ROM service call with A=&D

This call is the *ROM filing system initialise call. When the filing system is active and wishes to scan the next ROM this call is issued.

The initialise service call is made with the ROM number of the next ROM to be scanned in the Y register. Having received this service call a filing system ROM should only respond if its own ROM ID (stored in location &F4) is greater than or equal to the ROM number passed in the Y register.

Having decided to claim this service call the ROM should place its own ROM number in location &F5 which marks it as the currently active *ROM filing system ROM. It should then write the address of the start of the data it contains in locations &F6 and &F7. This provides a zero page pointer which is used by the filing system code to extract bytes of data serially from the ROM.

Having performed these two operations the service routine should return with the accumulator containing zero to indicate that the call has been claimed, In the case of the paged ROM ID being less than the ROM number in the Y register the service routine should exit with &D in the accumulator and the operating system will then offer the call to the next ROM.

The actual mode in which the *ROM filing system ROM numbers are represented differs from the way in which the paged ROM IDs are usually represented (i.e. as stored in &F4, a number 0 to 15). The filing system ROM numbers are represented by a value which is 15 minus the physical paged ROM number. One way of converting numbers from one form to another is, given the number to be converted in the accumulator,

```
        EOR #&FF
        AND #&F
```

which returns the inverted number in the accumulator. These instructions will always convert a number into the other representation.

11.4 Paged ROM service call with A=&E

Having obtained a response from a paged ROM to service call &D the *ROM filing system will use this service call to read bytes from the data contained in the ROM.

There is a difference in how the service routine can be implemented on the BBC Microcomputer OS 1.00 and later OS versions (including the Electron). The actual response required from the service call is essentially the same however.

When called by OS 1.00 a paged ROM should only respond to this call if its own ROM ID is the same as the current *ROM filing system ROM number. A comparison of the contents of memory location &F4 (current paged ROM) should be made with the inverted contents of &F5 (current *ROM) If these are not the same the call should be returned unclaimed.

The service routine for OS 1.00 should return the byte of data pointed to by the pointer in &F6 and &F7 in the Y register (e.g. LDA (&F6),Y:TAY) and increment this pointer so that it is ready for the next call.

Later operating system versions contain a routine (OSRDRM) which given the paged ROM ID of the current *ROM filing system ROM in the Y register will read a byte from this paged ROM using the pointer at &F6+&F7. Thus this paged ROM service call may be serviced by the highest priority *ROM filing system ROM and the operating system does not have to scan all the ROMs before getting a response. This leads to a significant improvement in performance of the *ROM filing system.

The service routines are able to determine which operating system has called them by the value of the Y register passed with this service call. On operating systems supporting the OSRDRM call the Y register contains a negative value while other versions of the operating system make this call with a positive value in the Y register.

The example given at the end of this section shows how the service routine at the head of a *ROM filing system ROM detects the operating system type and responds appropriately. This example will function on both types of operating system but will take advantage of OSRDRM routine if available. *ROM filing system ROMs designed for use on the earlier operating systems will still work with later versions.

11.5 *ROM data format

The format in which data should be stored in *ROM filing system ROMs is very similar to the tape data format. The data is divided into blocks which may be up to 255 bytes long. Each block of data is preceded by a header which contains information about the block. Both the block of data itself and the header are followed by a 16 bit cyclic redundancy check (CRC) value, The filing system calculates its own values for these CRCs during the loading process and compares them. If the filing sysem's value differs from the stored value then the filing system flags an error and rejects the data. (A routine for calculating CRCs is included in the example at the end of this section.)

The *ROM filing system data format is as follows:

offset	**description**	**length**
	Block Header	
0	&2A, a synchronisation byte	1
1	a file name (ito 10 chars.)	n
1+n	&00, a file name terminator	1
2+n	load address (low byte first)	4
6+n	execution address	4
10+n	block number (low byte first)	2
12+n	block length (in, in bytes)	2
14+n	block flag (see below)	1
15+n	address of next file	2
17+n	header CRC(1 to n + l6 incl.)	2
	Block Data	
19+n	data	m
19+n+m	data block CRC	2
	(next blocks)	
z	&2B, end of ROM marker	1

The block flag:

bit 0	Protection bit (file only allowed to be *RUN)
bit 6	Set if block contains no data
bit 7	Set if this is the last block of the file

For the *ROM filing system the headers for all but the first and last blocks may be replaced by a single byte header of value &23 (‘#’) with no CRC. This is implemented to reduce the memory overheads inherent with the tape style data format.

By convention the first file in a *ROM filing system ROM should be a title file. This is a file of zero length which serves to identify the ROM. The name of this file will appear on catalogue listings of the ROM. The file name of this title file should consist of a name and a version number preceded and followed by an asterisk e.g.‘*Mon00*’ or ‘*GAMESO5*’.

11.6 An example of a *ROM filing system ROM

The program below is written in BASIC 2 to assemble a ROM image which can be ‘blown’ into an EPROM and placed in a BBC microcomputer paged ROM socket or into a ROM cartridge slot on the Electron Plus 1 expansion.

Included in the program below is a routine for calculating CRC values (FNdo_crc). The actual CRC values required for this ROM image are included in the comments so that the actual values may be inserted directly if someone wanted to reduce the typing load when trying out this example.

```
 10 REM ************************************
 20 REM *                                  *
 30 REM *   *ROM filing system ROM example  *
 40 REM *                                  *
 50 REM ************************************
 60 REM Assemble CRC caLcuLating routine

 70 DIM MC% &100:PROCassm

 80 REM Set up constants required for ROM assembLy

 90 serROM=&F5
100 ROMid=&F4
110 ROMptr=&F6
120 OSRDRM=&FFB9
130 version=0

140 REM Reserve space for ROM image and prepare to assemble

150 DIM code% &4000
160 FORI=4 TO 7 STEP3
170 P%=&8000:O%=code%
180 [
190 OPT I
200 .ROMstart EQUB 0               \ null language entry
210           EQUB 0
220           EQUB 0
230           JMP service          \ service entry point
240           EQUB &82             \ ROM type, service ROM
250           EQUB copyr-ROMstart  \ offset to copyrights
260           EQUB version         \ binary version number
270           EQUS "Serial Rom"    \ ROM titLe string
280           EQUB 0
290           EQUS "0"             \ ROM version string
300 .copyr    EQUB 0
310           EQUS "(C) 1982 Acorn Computers" \ copyright$
320           EQUB 0               \ end of paged ROM header
330 .service  CMP #&D              \ service routine
340           BEQ initsp           \ initialise call?
350           CMP #&E
360           BEQ rdbyte           \ read byte call?
370           RTS                  \ not my call

380 \ Routine for paged ROM service call &D

390 .initsp   PHA                  \ save accumulator
400           JSR invsno           \ invert *ROM number
410           CMP ROMid            \ compare with ROM id
420           BCC exit             \ if *ROM > me, not my
430           LDA #data AND 255    \ low byte of data address
440           STA ROMptr           \ store in pointer
450           LDA #data DIV &100   \ high byte of data
460           STA ROMptr+1         \ store in pointer
470           LDA ROMid            \ get my paged ROM number
480           JSR invert           \ invert it
490           STA serROM           \ make me current *ROM
500 .claim    PLA                  \ restore
510           LDA #0               \ service call claimed
520           RTS                  \ finished
530 .exit     PLA                  \ call not claimed restore
540           RTS                  \ accumulator and return

550 \ Routine for paged ROM service call &E

560 .rdbyte   PHA                  \ save accumulator
570           TYA                  \ copy Y to A
580           BMI os120            \ if Y -ye OS has OSRDRM

590 \ this part for OS with no OSRDRM

600           JSR invsno           \ invert *ROM number
610           CMP ROMid            \ is it my paged ROM no.
620           BNE exit             \ if not do not claim call
630           LDY #0               \ Y=0
```

```
 640           LDA (ROMptr),Y       \ load A with byte
 650           TAY                  \ copy A to Y
 660 .claim1   INC ROMptr           \ increment ptr low byte
 670           BNE claim            \ no overflow
 680           INC ROMptr+1         \ increment ptr high byte
 690           JMP claim            \ claim call and return

 700 \ this part for OS with OSRDRM

 710 .os120    JSR invsno           \ A=current *ROM number
 720                                \ not necessarily me
 730           TAY                  \ copy A to Y
 740           JSR OSRDRM           \ OS will select ROM
 750           TAY                  \ byte returned in A
 760           JMP claim1           \ incremnt ptr & claim

 770 \ Subroutine for inverting *ROM numbers

 780 .invsno   LDA serROM           \ A=*ROM number
 790 .invert   EOR #&FF             \ invert bits
 800           AND #&F              \ mask out unwanted bits
 810           RTS                  \ finished

 820 \ End of header code/beginning of data

 830 .data     EQUB &2A             \ synchronisation byte
 840 .hdstrt   EQUS "*EXAMPLE*"     \ *R0M title
 850           EQUB 0               \ name terminator
 860           EQUD 0               \ Load address0
 870           EQUD 0               \ execution address=0
 880           EQUW 0               \ block number0
 890           EQUW 0               \ block length=0
 900           EQUB &CO             \ block flag
 910           EQUD eof             \ pointer to next file
 920 .hdcrc    EQUW FNdo_crc(hdstrt,hdcrc) \ CRC (&246F)
 930 .eof

 940 \ No data block for this file

 950           EQUB &2A             \ synchronisation byte
 960 .filel    EQUS "TEXT"          \ file title
 970           EQUB 0
 980           EQUD 0               \ null load address
 990           EQUD 0               \ null execution address
1000           EQUW 0               \ first block
1010           EQUW dat2-datl       \ length of data
1020           EQUB &80             \ first & last block
1030           EQUD eofl            \ pointer to end of file
1040 .hdcrcl   EQUW FNdo_crc(filel,hdcrcl) \ CRC (&E893)
1050 .datl     EQUS "REM This is a very short text file."
1060           EQUB &D              \ The file contents
```

```
1070 .dat2     EQUW FNdo_crc(datl,dat2) \ Block CRC (&655D)
1080 .eofl
1090           EQUB &2B             \ end of ROM marker
1100 .eor
1110 ]
1120           NEXT
1130           PRINT" *S.ROM        ";~code%;" ";~O%
1140           END

1150 REM Define function which calculates CRC
1160 REM Requires start and end of block up to 255 bytes

1170 DEF FNdo_crc(start,end)
1180 ?&82=(start-&8000+code%) AND &FF
1190 ?&83=(start-&8000+code%) DIV &100
1200 ?&84=end-start
1210 CALL crc
1220 =(!&80) AND &FFFF

1230 REM Define procedure which assembles CRC routine

1240 DEF PROCassm
1250 startaddr=&82
1260 Lo_crc=&81
1270 Hi_crc=&80
1280 len=&84
1290 FORI=0 TO 3 STEP3
1300 P%=MC%
1310 [
1320           OPT I
1330 .crc      LDA #0
1340           STA Hi_crc
1350           STA Lo_crc
1360           TAY
1370 .label1   LDA Hi_crc
1380           EOR (startaddr),Y
1390           STA Hi_crc
1400           LDX #8
1410 .label2   LDA Hi_crc
1420           ROL A
1430           BCC label3
1440           LDA Hi_crc
1450           EOR #8
1460           STA Hi_crc
1470           LDA Lo_crc
1480           EOR #&10
1490           STA Lo_crc
1500 .label3   ROL Lo_crc
1510           ROL Hi_crc
1520           DEX
1530           BNE label2
```

```
1540           INY
1550           CPY len
1560           BNE label1
1570           RTS
1580 ]
1590 NEXT
1600 CALL crc:ENDPROC
```

When the resultant ROM is installed in the machine the following dialogue may ensue.

```
>*ROM
>*CAT

*EXAMPLE*
TEXT
>*EXEC TEXT
>REM This is a very short text file.
```