ENTERPRISE
COMPUTERS



Technical Information

ENTERPRISE TECHNICAL INFORMATION

****

(C) 1984 INTELLIGENT SOFTWARE LTD.

CONTENTS
****

SECTION	TITLE
	The Operating System
1	Exos - Kernal
2	Exos - Function/Error Codes
3	Exos - Video Driver
4	Exos - Sound Driver
5	Exos - Cassette Driver
6	Exos - Keyboard Driver
7	Exos - Serial/Net Driver
8	Exos - Editor
	The Custom Chips
9	The Video Chip (Nick)
10	The Sound Chip (Dave)
	The BASIC Language

1 INTRODUCTION

EXOS is the ROM operating system for the ENTERPRISE micro-computer. It provides an interface between an application program and the hardware of the machine. The main feature of EXOS is a channel based input/output system. The input/output system allows device independent communication with a range of devices including:

1. The display. This may be configured in a variety of modes.

2. The keyboard. Includes the built in joystick and programable function keys.

3. A screen editor with word processing capabilities.

4. Cassette tape.

5. RS232 type serial interface.

6. Centronics compatible parallel interface.

7. Comprehensive four source stereo sound generator.

8. Integral three wire network interface.

Provision is also made for the addition of other devices together with associated driving code so that the system is simply expandable.

The input/output system only supports devices and includes no implicit file handling. Any file handling needed (such as with cassette tape) is provided by the device handler itself. A disk device handler would provide random access file handling facilities but would interface using exactly the same function calls. ..

2 EXOS SYSTEM ORGANISATION

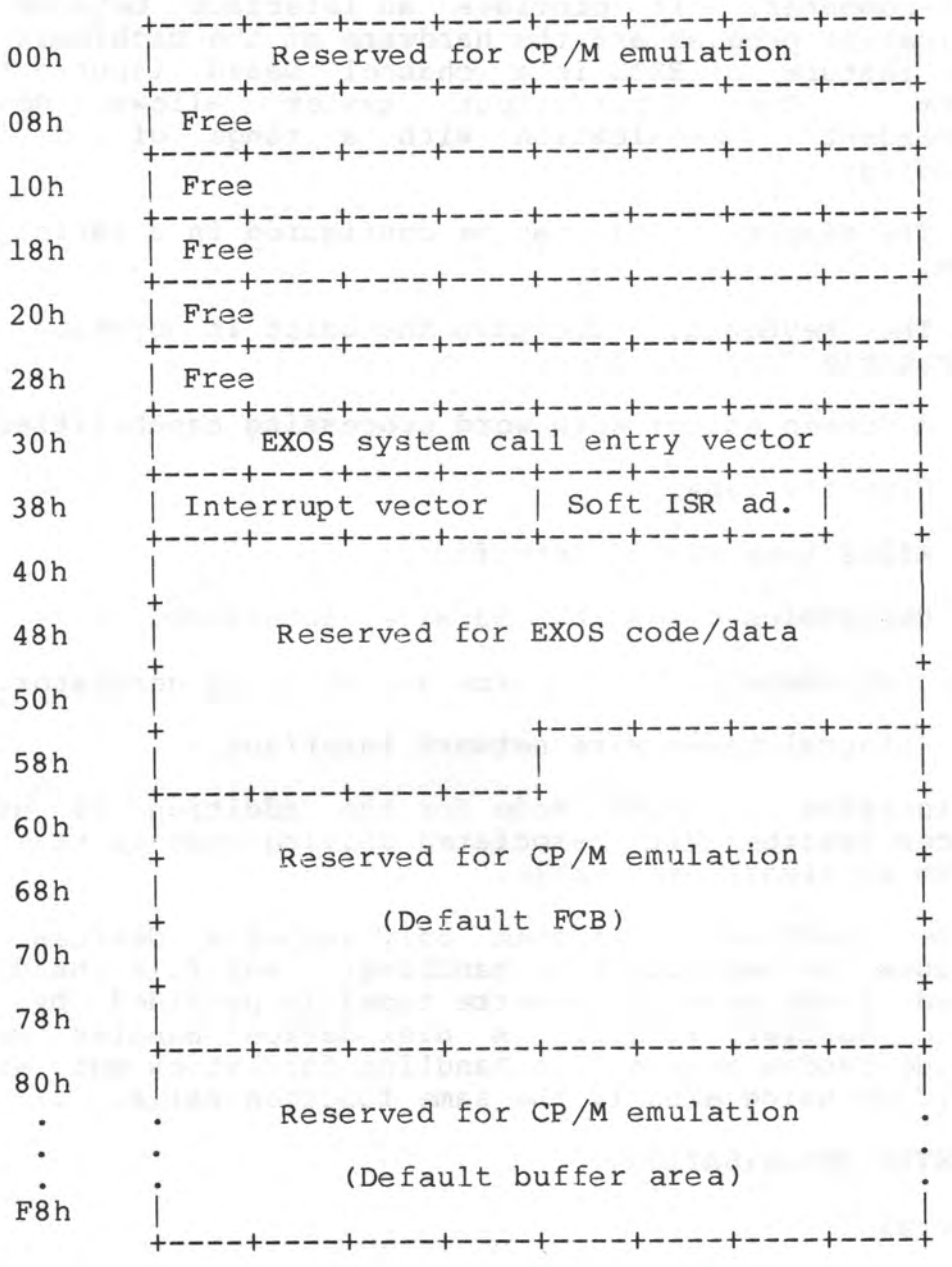
2.1 General

EXOS resides in ROM using a segment that is normally not mapped into the Z80 address space. Communication with EXOS is achieved through vectors in the first 256 bytes of RAM (in the Z80 address space). This RAM segment must always be present in the first page of the Z80 address space as it contains the interrupt vector and other system information required to allow EXOS to service interrupts and system calls.

 Copyright (C) 1984 Intelligent Software Limited

2.2 Use of zero page

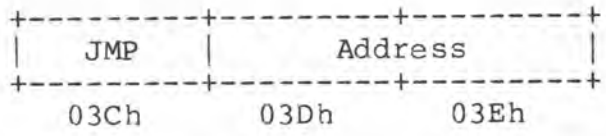
The following information is stored in the first 256 bytes of RAM:



 Copyright (C) 1984 Intelligent Software Limited

2.2 Software Interrupts

The "Soft ISR adr" entry is the address of a user supplied routine to service certain software generated interrupts. The address has the form:



Location 03Ch contains a Z-80 absolute jump instruction and must not be changed by the user. **Address** is the address of the software interrupt service routine. If the address is zero then software interrupts will be ignored. The segment containing the service routine must always be kept in the correct page of Z-80 memory space as it can be entered anytime as a result of an interrupt.

The software interrupt can respond to a variety of events selected by the software, such as STOP key pressed or data arrived on the network.

2.3 System calls

Programs may communicate with EXOS through the "RST 30h" instruction. The code at location 30h is not a simple jump instruction as this code is responsible for selecting the EXOS ROM page before EXOS can be entered. The space from 3Fh to 5Bh will be used for additional code needed for EXOS entry. This entire area from 30h to 5Bh should not be modified by the user at all, except for the SOFT ISR address at 3Dh & 3Eh.

The different EXOS calls are defined by use of a function code. This code is a single byte value and should be stored immediately following the "RST 30h" instruction. Parameters to the EXOS calls are normally passed in the Z80 registers. Buffers and strings are passed by address. Return values from EXOS are also normally passed back in registers. If no values are returned in either register BC or DE then the old contents will be overwritten anyway. However registers HL, IX, IY and the alternate register set (including AF') will remain unchanged by all EXOS calls (with an exception for the ALLOCATE CHANNEL BUFFER call which returns a pointer in IX). Register AF returns a status code.

 Copyright (C) 1984 Intelligent Software Limited

EXOS maintains a separate system stack in the EXOS workspace and therefore needs very little stack space in the program area. However, at least 8 bytes should always be available beyond the top of the stack. Even if no EXOS calls are made, this space is required for interrupt servicing. The program stack should also be managed correctly such that there is never any wanted information above the stack pointer.

The system calls available are:

Code	Function	
0	System reset	(user only function)
1	Open channel	(user only function)
2	Create channel	(user only function)
3	Close channel	(user only function)
4	Destroy channel	(user only function)
5	Read character	
6	Read block	
7	Write character	
8	Write block	
9	Channel read status	
10	Set and read channel information	
11	Perform special function on channel	
16	Read/Write/Toggle EXOS Variable	
17	Capture channel	
18	Re-direct channel	
19	Set default device name	
20	Return system status	
21	Link device	(user only function)
22	Read exos boundary	
23	Set user boundary	
24	Allocate segment	
25	Free segment	
26	Scan ROMs	(user only function)
27	Allocate channel buffer	(device only function)
28	Explain error code	

All EXOS calls return a status value in register "A". This value is zero if the call was completely successful and non-zero otherwise. The S and Z flags are set to reflect the value of A prior to the return (except for a RESET SYSTEM call which never returns with A non-zero anyway). EXOS function call 28 can be used to provide a simple explanation string for certain error codes (see later).

 Copyright (C) 1984 Intelligent Software Limited

3 Memory Organisation

3.1 The ENTERPRISE Addressing Scheme

The Z-80 has a 16-bit address bus [0000h - FFFFh] which allows direct addressing of 64k of memory. In order to increase this capacity to 4M the DAVE chip provides an interface to the ENTERPRISE's 22-bit address bus through four internal 8-bit registers (the page registers).

The top two bits of the Z-80 address define four "Z-80 pages" each of 16k and each with a corresponding page register. Thus Z-80 page-0 contains addresses 0000h - 3FFFh, page-1 contains 4000h - 7FFFh, etc. The eight bits from the appropriate page register replace the top two bits of the Z-80 address to give a 22-bit system address.

In this manner the ENTERPRISE memory is divided into 256 16k segments any one of which can be made to appear in any of the four Z-80 pages by putting its segment number into the appropriate page register.

The top four segments (FCh - FFh) provide the built in RAM in the 64k machine. These segments are also the only ones which the NICK chip can use for screen memory and because of this are referred to as the "Video RAM". The internal ROM 32k ROM is made up of segments 00h and 01h, which are reflected by the hardware in segments 02h and 03h. The cartridge occupies segments 04h to 07h.

3.2 EXOS Segment Allocation

When the system is started up, or a cold reset is done, EXOS determines the amount of available RAM and tests each segment. Faulty segments are treated as if they do not exist, leaving the remaining RAM usable. The system will not function, however, unless at least 32k of RAM, including segment FFh, is working.

EXOS maintains an allocation of segments between five states :- free, allocated to the system, allocated to a device, allocated to the user, or shared between the user and the system. There is always one segment (the system segment) allocated to the system, details of which will be given later. This will contain EXOS and device variables and data areas. This will always be the top video RAM segment (segment 0FFh).

 Copyright (C) 1984 Intelligent Software Limited

One RAM segment (the page zero segment) is kept outside this allocation scheme. This segment contains the EXOS entry points as defined above and must always be in Z-80 page zero when interrupts are enabled or EXOS is called. On a 64k machine this will be the lowest video RAM segment (segment 0FCh). On a machine with any extra RAM, it will be a non-video segment. Apart from the first 256 bytes it may be used by the user. It cannot be freed (see later).

All other RAM segments in the system will initially be free. The contents of Z-80 pages one and two when the applications program is entered will be undefined.

Two EXOS function calls are provided to control RAM allocation - allocate segment and free segment. These calls may be made by the user or by a device driver. The user will only be able to free segments allocated to him and a device driver will only be able to free segments allocated to a device (although there is no check on which device is freeing it).

A new segment may be obtained by a device driver or the user by making an "allocate segment" function call. If there is a free segment then it will be marked as allocated to the user or device as appropriate and its segment number returned. If there is no free segment but there is unused space in the last segment allocated to the system then this segment may be allocated to the user as a shared segment (see the next section). A device will never be allocated a shared segment. If there is no free space at all then a non-zero status code will be returned. Non-video RAM segments will be allocated in preference to video RAM segments if there is a choice.

Whenever EXOS requires more RAM (when a channel is opened) it may grab a free segment if one exists and allocate it to itself. It always frees segments as soon as possible when it requires less memory (when channels are closed). EXOS will always use video RAM segments if any are available.

3.3 Shared Segment

There can be at most one segment in the system which is shared between the user and the system. When the user makes an "allocate segment" EXOS call, if there are no free segments and there is currently no shared segment, then the user will be allocated a segment which EXOS is using part of. This is then the shared segment.

 Copyright (C) 1984 Intelligent Software Limited

The shared segment contains two boundaries, the USER boundary and the EXOS boundary. EXOS uses the segment from the top down as far as the EXOS boundary. The user is allowed to use from the bottom up as far as the USER boundary, which will never be higher than the EXOS boundary.

The part of the segment (if there is any) between the two boundaries is no-man's land, neither EXOS nor the user should use it. If EXOS requires more RAM to open a channel then it can move the EXOS boundary down as far as the USER boundary and similarly the user can move his boundary up as far as the EXOS boundary to get more space.

To manipulate the boundaries in the shared page two EXOS calls are provided - SET USER BOUNDARY and READ EXOS BOUNDARY.

The page may become un-shared at any time if EXOS no longer needs it, in which case a "read exos boundary" call will indicate that there is no shared segment. Also the user may give up the page entirely to EXOS with a suitable free segment EXOS call. If there is no shared segment when the user does a read system boundary EXOS call then the position that the EXOS boundary would be in a shared page if the user were to be allocated one, is returned, with a segment number of zero to indicate that there is no shared segment. This is useful for finding out how much RAM in the system is free.

Note: The pointers which mark the boundaries both point to the byte above the boundary (since a boundary should be considered as being between two bytes). They will be in the range 0000h...3FFFh, and can be equal.

3.4 Device RAM Areas and Device Descriptors

Every device in the system has a device descriptor which defines the name of the device, the address of its code and various other details. EXOS keeps a linked list of these descriptors in RAM. Details of the format will be given below.

There are three types of device in the system which differ in how they are linked into the chain and how their RAM is allocated.

 Copyright (C) 1984 Intelligent Software Limited

3.4.1 Built In Devices

Built in devices are those whose code is in the internal ROM with the EXOS kernel. At startup time their device descriptors are copied into RAM and linked into the chain. Each built in device has a fixed area of RAM in the system segment allocated to it. These areas are determined at assembly time and thus may be addressed absolutely since these devices are linked with the EXOS kernel.

3.4.2 External Devices

These are devices whose code is contained in expansion ROMs, either in the cartridge socket or on the expansion stack. They are linked into the system in the same way as, and immediately after, the built in devices. They will each be allocated an area of RAM immediately below the copy of the device descriptor in system RAM. The size of this RAM area is specified in the device descriptor in the ROM.

When the device code is called IY will point to the device descriptor (in Z-80 page-2) and so the RAM can be accessed relative to IY. If "n" bytes are allocated then they can be accessed at:

IY-4, IY-5 ... IY-4-(n-1)

This area of RAM will always be within the one segment and once allocated will never move or be de-allocated, and will be filled with zeroes before calling the devices initialisation routine. Note that in fact built in devices can also be allocated an area of RAM in this way but this is unlikely to be used as it is generally more convenient to address RAM absolutely.

The later section on external ROM scanning will explain exactly which of the 256 possible segments will be examined for external devices.

 Copyright (C) 1984 Intelligent Software Limited

3.4.3 User Devices

User devices can be linked into the system at any time by an EXOS call giving the address of a complete device descriptor in RAM. (As opposed to the incomplete ones contained in the ROMs). They can be allocated an area of device RAM in the system segment when they are linked in. IX will be pointed to this RAM when the initialisation entry point of the device is called. The device must then remember this address since it will never be told it again. If "m" bytes are allocated then they can be accessed at:

IX-1, IX-2 ... IX-m

Like the device RAM for external devices, this RAM can never move or be de-allocated, and will be filled with zeroes before calling the devices initialisation routine. Note that when the device's initialisation routine is called again (at warm reset time for example) IX will not point to this RAM area so the device will have to remember that it has already been initialised once.

3.5 Channel RAM areas and Channel Descriptors

Every open channel has an area of "CHANNEL RAM" allocated to it. This is allocated when the channel is opened and de-allocated when it is closed. The size of it is determined by the device when the channel is opened and although it cannot change in size after this time, it can be moved around. This can only occur when other channels are opened or closed or a new device is linked in. The device driver is always informed when the channel RAM for any channels open to it is moved.

3.5.1 Allocating Channel RAM

When a channel is opened to a device, EXOS will call the OPEN CHANNEL entry point of the device. At this stage no RAM is allocated. Before it finally returns to EXOS the device driver must make an "allocate channel buffer" EXOS call. This will cause EXOS to allocate the requested amount of channel RAM for this channel. It will return an error code if there is insufficient RAM available, which can then be returned to the user. It actually requires a bit more RAM than the amount specified by the device as it also creates a channel descriptor for this channel.

 Copyright (C) 1984 Intelligent Software Limited

When the channel RAM is setup the device specifies two 16 bit sizes the sum of which is the total channel RAM size. One size specifies the amount of the RAM which must be in one segment and the other specifies the size of the remainder of the RAM which may be allowed to extend into other segments. The latter of these is ignored (taken to be zero) except for a video device. A pointer to the allocated buffer is returned in IX and the segment containing the start of the buffer will be put in Z-80 page-1.

EXOS moves these channel RAM areas around to make efficient use of memory. Whenever the device driver code is entered with an EXOS channel call, IX will point to the appropriate channel RAM, which will be in Z-80 page 1. When EXOS decides to move this RAM (which can only happen when a channel is opened or closed) it will move it and then call the BUFFER MOVED entry point of the device code to allow the device to adjust any pointers which it may need to. This is particularly important for the video driver as it must update the line parameter table. Interrupts are disabled from just before moving a buffer and returning from the BUFFER MOVED routine of the device.

Note that register IX actually points at the byte after the last one in the RAM area. This is a consequence of the way that EXOS builds its RAM downwards. Thus the bytes which the device driver can use (assuming "m" bytes of channel RAM) are:

(IX-1), (IX-2) ..... (IX-m)

If the allocated buffer extends into a second segment (only allowed for a video device) then this will have a segment number of one less than the first one - and so on for further segments.

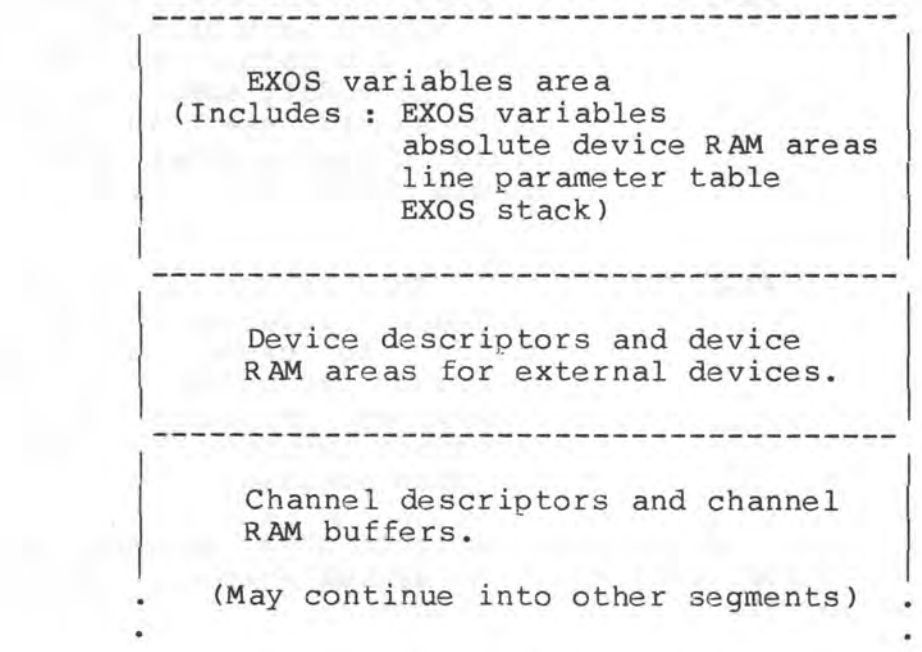
3.5.2 Channel Descriptors

As mentioned before EXOS creates a channel descriptor for each channel when it is opened. This is in fact located at IX+0 upwards, but there are no fields in it which are of interest to the device driver so its layout is not defined here.

3.6 EXOS memory usage

EXOS uses RAM from the top of system memory downwards. As far as possible it keeps the video RAM for itself and allocates non-video RAM to the user. The layout of the EXOS system segment (which is always segment 0FFh) is as below:

 Copyright (C) 1984 Intelligent Software Limited



The very top of the EXOS variable area contains a few defined values which are guaranteed not to move in future versions of EXOS. They are listed here with the address where a device will see them (in Z-80 page-2). If they are to be accessed by an applications program, then the correct segment must of course be paged in.

0BFFFh - USR_P3 \
0BFFEh - USR_P2 \ These are the contents of the four
0BFFDh - USR_P1 / paging registers when EXOS was last
0BFFCh - USR_P0 / called

0BFFA/Bh - STACK_LIMIT Devices which need more than the default 100 bytes of RAM can let their stack grow down as far as the contents of this variable.

0BFF8/9h - RST_ADDR The address of a warm reset routine which must be in the page-zero segment. If this is zero then a cold reset will be done.

0BFF6/7h - ST_POINTER The Z-80 address of the status line memory. The 42 bytes from this address onwards are the status line (see video driver spec ET11/9).

0BFF4/5h - LP_POINTER The Z-80 address of the start of the line parameter table. The first line parameter block will be the status line one.

 Copyright (C) 1984 Intelligent Software Limited

0BFF3h - PORTB5 This is the current value in the output port 0B5h which is a general I/O port. Devices which need to modify some bits of this port should use this value to avoid changing other bits, and should keep this variable up to date.

0BFF2h - FLAG_SOFT_IRQ This is set to a non-zero software interrupt code by a device to cause a software interrupt to occur. It is also tested by various devices to determine whether the stop key has been pressed.

The size of the EXOS variable area depends on the amount of RAM and ROM in the system which is determined at startup time and cannot change.

The size of the device RAM area depends on what external and user devices are linked into the system. The size of this can only change when a user device which requests some system RAM is linked into the system.

The channel buffer area is very dynamic since it can change whenever a channel is opened or closed. The channel buffers may occupy any number of segments. The system will of course ensure that channel buffers for the video device are kept in the internal video RAM.

 Copyright (C) 1984 Intelligent Software Limited

4. Applications Program Interface

4.1 Initialisation

4.1.1 Locating Expansion ROMs

Any expansion ROM which is intended to be recognised by EXOS must start with the following:

```
0000:     DB   'EXOS_ROM'
0008:     DW   DEVICE_CHAIN        ;May be zero (see later)
000A:                              ;Entry point of ROM.
```

When the machine is first switched on, or when a cold reset is performed the entire 4 Mega-byte memory space is examined to look for expansion ROMs. To avoid problems with ROMs reflecting due to incomplete decoding, not every possible 16k segment is examined. Basically each 256k boundary is examined, thus allowing one expansion ROM in each expansion unit on the stack, and then the cartridge slot is examined.

The cartridge slot occupies segment numbers 4, 5, 6 and 7. Each of these segments is tested for a ROM, but when a ROM is found the first 256 bytes of it are compared byte-for-byte with segment 4, and the new ROM ignored if it is identical. This ensures that if a single 16k cartridge which reflects four times is plugged in then it will only be found once, but still allows a mother board with decoding to support one to four 16k cartridges plugged in any of its four slots.

4.1.2 Selecting an applications ROM

The segment numbers of all expansion ROMs which are found are put into a list, which also includes the internal ROM. Under certain circumstances EXOS will call the entry point of all the ROMs at 0C00Ah in turn, passing an "action code" in register C and various other parameters in other registers. This is referred to as a "ROM SCAN" and will be discussed in more detail later with details of the various action codes.

At cold reset time a ROM scan is done with action code 1. Any ROM which contains a main applications program which wants to take control at startup (such as BASIC or LISP) should thus use this as its cue to start. To do this is simply does not return from the entry point call, but must do a reset EXOS (with flags set to 01100000b) call before it can do anything else. Having done this call it must then set up its own stack somewhere in the page-zero-segment re-enable interrupts and is then free to make any EXOS calls.

 Copyright (C) 1984 Intelligent Software Limited

4.2 Scanning the ROMs

When a ROM scan is done each ROM in the list will be called in turn, with the internal ROM being called last. In certain cases a ROM may not return from this call if it wishes to start itself up as an applications program. This should only occur in response to action codes 1 or 2 below. Normally a ROM will examine the action code and possibly other parameters and then either return immediately, or carry out some function and then return.

If a ROM does return then it must preserve the action code in C and any other parameters passed in registers B or DE, to allow the ROM scan to continue. If the ROM has carried out some service or is returning some parameters and wants to prevent any other ROMs from also trying to do this, it should return with the action code in register C set to zero which all other ROMs will ignore. If this is done then registers B and DE can be corrupted. All other registers (AF, HL, AF', BC', DE', HL', IX and IY) can be corrupted by ROM service routines.

The action codes provided are:

0. Do nothing - Must preserve BC and DE.
1. Cold reset
2. User string
3. Help string
4. Unknown EXOS variable
5. Unknown error code

Action codes above this should not occur and so may be ignored.

4.2.1 Action code 1 - Cold Reset

This was described above. No parameters are passed and any ROM which wants to take control at power on should not return.

4.2.1 Action code 2 - User string

This action code results from a user SCAN ROMs function call. It is passed a pointer to a string in register DE. This string will have a length byte first and will be stored in a buffer in Z-80 page-2 (the system segment). The first word of this string will have been uppercased and register B will contain a count of how many bytes there are in this first word.

The first word is the name of a command, service or program. If the ROM does not recognise this name then it should return from the call, preserving BC and DE. If it does recognize the name then it should respond to it, possibly interpreting the rest of the string as parameters, returning with register C=0 unless it wishes other ROMs to also respond to this command.

The ROM can interpret this action code as a cue to start up if it recognises a suitable string. For example the strings "BASIC", "LISP" and "FORTH" will be interpreted in this way by the appropriate language cartriges. In this case the ROM will not return from the call but behaves as if it received action code 1 (see earlier).

The ROM may recognise the string as a command or service in which case it will carry out some operations which may involve I/O (all normal user EXOS calls including open and close channel and even another SCAN ROM call can be made) and then return. It may be useful for such services to make use of the default channel number facility (using channel number 255 - see later) since it cannot otherwise know what channels are open because it does not know from what applications program it was called.

4.2.1 Action code 3 - Help string

This action code also results from a user SCAN ROMs function call, where the first word of the string was "HELP". The "HELP" (and any trailing spaces) will have been removed from the string and then the rest of the string treated exactly as if it was the original string received from the user. The effect of this is that the ROM sees a string with the first word of it uppercased with the length of this first word in B. If this string received is null (so the original string was just "HELP"), then the ROM should just write its name and version number to the default channel (using channel number 255) and return with BC and DE preserved. If the string is not null then if the first word is any of the action code 2 commands recognised by this ROM then any help information about that command should be printed and register C set to zero.

 Copyright (C) 1984 Intelligent Software Limited

4.2.1 Action code 4 - Unknown EXOS variable

This action code results when a read/write/toggle EXOS variable call was made with a variable number not-recognised by the internal ROM. Expansion ROMs can therefore implement additional EXOS variables which may be useful for expansion devices. The parameters are (see also the spec of the EXOS call later on):

B = 0, 1 or 2 for READ, WRITE and TOGGLE (ones complement)
E = EXOS variable number (Always >= 128)
D = New value to be written (only if B=1)

If the variable number is not recognised then the ROM should return with BC and DE preserved. If the variable number is one supported by this ROM then the function should be performed and the following parameters returned:

C = 0
D = New value of EXOS variable

4.2.1 Action code 5 - Unknown error code

This action code results from a user "explain error code" function call. The error code is passed in register B and if it is recognised by this ROM then a pointer to a string should be returned as follows:

B = Segment number containing message
C = 0
DE = Address of message string (length byte first).

Error codes in the range 0C0h to 0FFh are reserved for internal EXOS use and so should not be generated by external devices or ROMs although they can be interpreted by this function to re-define the built in error codes.

4.3 Software Interrupts

Software interrupts provide a way for the user to be alerted to various events occuring within EXOS. When a device (probably in its interrupt routine) detects an event which should cause a software interrupt, it sets a byte in the EXOS variable area (FLAG_SOFT_IRQ) to the code indicating what type of interrupt it was. This byte is also available as an EXOS variable.

 Copyright (C) 1984 Intelligent Software Limited

Nothing else occurs until EXOS is about to return to the user's code, either directly from the interrupt routine, or from an EXOS call if the program was executing in EXOS when the interrupt occurred. Devices should look at FLAG_SOFT_IRQ and cause a premature return if it contains the value ?STOP (20h) to ensure prompt response to the stop key, returning status code .STOP.

When execution is about to return to the user, if he has defined a soft ISR address in page-0 then this routine will be jumped to instead of returning to the user's program. The environment will be exactly as it would be if the return to the user had been made, with the correct paging and the user's stack active. The return address to the user will still be on the stack so the routine may return to the main program. If it does return then all registers must be preserved as it could be an effective interrupt.

It is not necessary for the software interrupt routine to return if it doesn't want to, it can cause some sort of warm re-start of the user's program.

The service routine can find out the software interrupt code by reading an EXOS variable CODE_SOFT_IRQ. This is in fact a copy of the code set up by the device since the code itself is reset to zero before jumping to the routine to avoid multiple responses to the software interrupt. If more than one software interrupt occurs before the soft ISR can be called then only one will be acknowledged.

All sources of software interrupts from built in devices can be enabled or disabled by setting appropriate EXOS variables, or making special function calls. The codes from built in devices are:

10h...1Fh	-	?FKEY....	Keyboard function key pressed.
20h	-	?STOP	Keyboard STOP key pressed
21h	-	?KEY	Keyboard any key pressed
30h	-	?NET	Network data received

4.4 EXOS Variables

There are a number of variables defined which may be set, read or toggled by an EXOS function call. These variables control many different aspects of the system, particularly in controlling options for devices. Each one is 8 bits and is identified by an 8 bit variable number. Some are provided for expansion devices and are not of interest to the user. The currently defined variables are (subject to change):

 Copyright (C) 1984 Intelligent Software Limited

0 - IRQ_ENABLE_STATE b0 - set to enable sound IRQ.
b2 - set to enable 1Hz IRQ.
b4 - set to enable video IRQ.
b6 - set to enable external IRQ.
b1,3,5 & 7 must be zero.

1 - FLAG_SOFT_IRQ. This is the byte set non-zero by a device to cause a software interrupt. It could also be set by the user to cause a software interupt directly. This variable is also available at a fixed address as specified in section 3.6.

2 - CODE_SOFT_IRQ. This is the copy of the flag set by the device and is the variable that should be inspected by a software interrupt service routine to determine the reason for the interrupt.

3 - DEF_TYPE Type of default device.
0 => non file handling device (eg. TAPE)
1 => file handling device (eg. DISK)

4 - DEF_CHAN Default channel number. This channel number will be used whenever a channel call is made with channel number 255.

5 - LOCK_KEY Current keyboard lock status.

6 - CLICK_KEY 0 => Key click enabled

7 - STOP_IRQ 0 => STOP key causes soft IRQ
<>0 => STOP key returns code

8 - KEY_IRQ 0 => Any key press causes soft IRQ, as well as returning a code.

9 - RATE_KEY Keyboard auto-repeat rate in 1/50 second

10 - DELAY_KEY Delay 'til auto-repeat starts.
0 => no auto-repeat

11 - TAPE_SND 0 => Tape sound enabled

 Copyright (C) 1984 Intelligent Software Limited

```
12 - WAIT_SND    0 => Sound driver will wait when a
                      queue is full
                 <>0 => Sound driver returns .SQFUL error ...
13 - MUTE_SND    0 => internal speaker active.
                 <>0 => internal speaker disabled.
14 - BUF_SND     Size of sound envelope storage in
                 'phases', for next sound channel
                 opened.


15 - BAUD_SER    Defines serial baud rate.
16 - FORM_SER    Defines serial word format.

17 - ADDR_NET    Network address of this machine
18 - NET_IRQ     0 => Data received on network will
                 cause a software interrupt.
19 - CHAN_NET    Channel number of network block
                 received


20 - MODE_VID    Video mode     \  These variables select
21 - COLR_VID    Colour mode     \ the characteristics of
22 - X_SIZ_VID   X page size     /  a video page when it
23 - Y_SIZ_VID   Y page size    /   is opened.

24 - ST_FLAG     0 => Status line is displayed

25 - BORD_VID    Border colour of screen
26 - BIAS_VID    Colour bias for palette colours 8...16

27 - VID_EDIT    Channel number of video page for editor
28 - KEY_EDIT    Channel number of keyboard for editor
29 - BUF_EDIT    Size of edit buffer (in 256 byte pages)

30 - FLG_EDIT    Various flags to control editor
                 function - yet to be defined.

31 - SP_TAPE     Non-zero to force slow tape saving.
32 - PROTECT     Non-zero to make cassette write out
                 protected file.
33 - REM1       \ State of cassette remote controls,
34 - REM2       / zero is off, non-zero is on.
```

 Copyright (C) 1984 Intelligent Software Limited

5. EXOS System Calls.

Below are details of all the EXOS function calls. Function codes 1 to 11 are the EXOS channel calls and cause the appropriate entry point of the appropriate device driver to be called. When the device is entered the channel number will be in register A and be in the range 1...255. The parameters in registers BC and DE will be passed direct to the device driver. Registers IY and IX will be set to point to the appropriate device RAM and channel RAM areas respectively. EXOS saves all registers and so any registers may be corrupted by devices except where parameters are returned in them.

The format of all strings is a length byte, followed by the bytes of the string. The length byte is the number of characters in the string and does not include the length byte itself. A null string is a single byte of zero. All name strings are checked for legality and uppercased before being used by the system or passed to devices. This includes filenames and device names for "open channel" function calls and the first word of name strings for the "locate ROMs" function call.

Where parameters are in fact pointers to memory care must be taken with the memory paging. The correct segment must be in the Z-80 memory page and the buffer pointed to must not go over a 16k page boundary. To access this memory the device driver must determine which segment was in the appropriate Z-80 page when EXOS was called. Four bytes in the EXOS variable area are defined to contain the values of the four paging registers when EXOS was last called so the segment can be extracted from here.

As a convenience for built in devices a common subroutine called GET_SEGMENT is provided which is passed a caller's Z-80 address in DE and will put the appropriate segment in Z-80 page 1. It will return a modified address in DE which points to the required address in Z-80 page 1. Note that this will page out the channel RAM, but the device RAM will still be available in page 2 (for built in & external devices). This routine only alters registers AF and DE.

 Copyright (C) 1984 Intelligent Software Limited

5.1 Filename syntax

For open and close channel a filename string is required. The syntax of this is:

[[device-name] [unit-number] :] [filename]

where [] denotes an optional part. The device name (if present) is a up to 28 letters, upper or lower case. The unit number (if present) is a decimal number in the range 0 to 255 and may optionally have the character "-" seperating it from the device name, or may immediately follow the device name. The COLON must be present if either the device name or unit number is present to seperate the filename. The filename itself is a string of up to 28 characters from the following:

Upper or lower case letters (not distinguished)
Digits
\/-_.

If the device name is present but no unit number then a default unit number of 1 is used. If no device name is present then the default device name ("TAPE" on the basic machine) is used either with the supplied unit number if there is one or with the default unit number set by the set default device name EXOS calls.

Function 0 - System reset

Parameters: C = Reset type flags
Returns: A = Status (always zero but flags not set)
Interrupts disabled

This call causes a reset of the operation system. The flags passed in register C control exactly what the RESET does, as below.

 Copyright (C) 1984 Intelligent Software Limited

b0 ... b3 must be zero

b4 - Set => Forcibly de-allocate all channel RAM, and re-initialise all devices. User devices will be retained and device segments will not be de-allocated.

b5 - Set => As bit-4 but also re-link in all built in and extension devices. User devices will therefore be lost. Device segments are not de-allocated.

b6 - Set => De-allocate all user RAM segments.

b7 - Set => Warm reset. This is equivalent to pressing the RESET button on the machine, bits 0 & 1 are ignored and the function will not return. It will cause a jump to the warm reset address if defined or a complete system re-start if not.

Note that the flags are not set to be consistent with the status code (which is always zero anyway) and registers BC', DE' and HL' are corrupted by this EXOS call. Also a side effect of the call is that interrupts are disabled.

Function 1 - Open channel

Parameters: A channel number (must not be 255)
DE pointer to filename string
Returns: A status

The status return in A indicates whether the open was successful. The format of the filename string is specified later. It includes device name, unit number and filename.

The filename and unit number are passed to the device driver for interpretation and many devices will just ignore them. If the device is one which supports filenames then it will return an error code if the file specified does not already exist. Some devices require options to be selected (by special function calls) before the channel can be used. Also some devices require parameters to be specified by setting EXOS variables before a channel can be opened.

If the device is one which allows multiple device drivers of the same name to exist (such as disks) then the unit number is used to determine which driver is called. For example for disks unit numbers 0 and 1 correspond to the first driver, 2 and 3 to the second and so on.

 Copyright (C) 1984 Intelligent Software Limited

Note that before this call gets through to the device routine, the filename component of the string pointed to by DE will have been copied to a new string which DE now points to, without the device name or number, uppercased and checked for illegal characters. This string will be in the system segment which is located in Z-80 page-2 when the device is called so it is not necessary to do any paging in order to access it. The unit number will be passed as a byte in register C.

For the open channel function to be successfully completed, the device must allocate itself a channel buffer before it returns.

Note also that the channel number will have been incremented by one before being passed through to the device so it will be in the range 1...255 and can never be zero as far as the device is concerned.

Function 2 - Create channel

```
Parameters:   A  channel number (must not be 255)
              DE pointer to filename
Returns:      A  status
```

The create function is similar to the open function except that if the device supports filenames then the file will be created if it doesn't exist, and an error code returned if it does. It is identical to OPEN CHANNEL for all built in devices except the cassette driver.

Function 3 - Close channel

```
Parameters:   A  channel number (must not be 255)
Returns:      A  status
```

The close function flushes any buffers and de-allocates any RAM used by the channel. Further reference to this channel will not be allowed. The devices entry point is called before the channel RAM is de-allocated.

Function 4 - Delete channel

```
Parameters:   A  channel number  (must not be 255)
Returns:      A  status
```

The destroy function is similar to the close function except that on a filename device the file is deleted. It is identical for all built in devices.

 Copyright (C) 1984 Intelligent Software Limited

Function 5 - Read character

```
Parameters:   A  channel number
Returns:      A  status
              B  character
```

The read character call allows single characters to be read from a channel without the explicit use of a buffer. If no character is ready then it waits until one is ready.

Function 6 - Read block

```
Parameters:   A  channel number
              BC byte count
              DE buffer address
Returns:      A  status
              BC bytes left to read
              DE modified buffer address
```

The read block function reads a variable sized block from a channel. The block may be from 0 to 65536 bytes in length and can cross segment boundaries. Note that the byte count returned in BC is valid even if the status code is negative, although not if it is an error such as non-existent channel. This allows a partially successful block write to be re-tried from the first character which failed.

Function 7 - Write character

```
Parameters:   A  channel number
              B  character
Returns:      A  status
```

The write character function allows single characters to be written to a channel.

Function 8 - Write block

```
Parameters:   A  channel number
              BC byte count
              DE buffer address
Returns:      A  status
              BC bytes left to write
              DE modified buffer address
```

The block write function allows a variable sized block to be written to a channel and is similar to block read. The byte count returned in BC is valid even if the status code is negative, provided that control got as far as the device driver.

 Copyright (C) 1984 Intelligent Software Limited

Function 9 - Channel read status

Parameters:	A	channel number
Returns:	A	status
	C	00h if character is ready to be read FFh if at end of file 01h otherwise.

The read channel status function call is used to allow polling of a device such as the keyboard without making the system wait until a character is ready.

Function 10 - Set and Read Channel Status

Parameters:	A	channel number
	C	Write flags
	DE	pointer to parameter block (16 bytes)
Returns:	A	status
	C	Read flags

This function is used to provide random access facilities and file protection on file devices such as disk or a RAM driver. The format of the parameter block is:

bytes:	0...3	- File pointer value (32 bits)
	4...7	- File size (32 bits)
	8	- Protection byte (yet to be defined)
	9...15	- Zero. (reserved for future expansion)

The assignment of bits in the read and write flags byte is as below. The specified action is taken if the bit is set.

	WRITE FLAGS	READ FLAGS
b0	Set new file pointer value	File pointer is valid
b1	not used (0)	File size is valid
b2	Set new protection byte	Protection byte is valid
b3...b7	not used (0)	always 0

This allows the file pointer and/or the protection byte to be set independently, or just to be read. Not all devices need to support this function. If a device doesn't support it then it should return a .NOFN error code.

 Copyright (C) 1984 Intelligent Software Limited

Function 11 - Special function

Parameters:	A	channel number
	B	sub-function number
	C	unspecified parameter
	DE	unspecified parameter
Returns:	A	status
	C	unspecified parameter
	DE	unspecified parameter

This function call allows device specific functions to be performed on a channel. Its use is discouraged as it necessarily makes the program device dependent,

Typical functions performed by this call would be:

- Display a given page on the screen
- Program a function key
- Flush the network buffer.

Further details of the special functions available can be found in the individual device driver specifications.

Function 16 - Read, Write or Toggle EXOS Variable

Parameters:	B = 0	To read value
	= 1	To write value
	= 2	To toggle value
	C =	EXOS variable number (0...255)
	D =	New value to be written (only for write)
Returns:	A =	Status
	D =	New value of EXOS variable

This function allows EXOS variables to be set or inspected. These variables control various functions of the system and specific devices. Note that the value is returned in D even for write and toggle. A list of currently defined EXOS variables was given earlier. Expansion ROMs can implement additional EXOS variables but only in the range 128...255.

Function 17 - Capture channel

Parameters:	A -	Main channel number
	C -	Secondary channel number (0FFh to cancel capture)
Returns:	A -	Status

The capture channel function causes subsequent read function calls (read character, read block and read status) to the main channel, to read data instead from the secondary channel. When the function call is made the main channel must exist but no check is made on the secondary channel number.

 Copyright (C) 1984 Intelligent Software Limited

The capture applies to all subsequent input from the main channel number until either the secondary channel is closed or gives any error (such as end of file) or the main channel is captured from somewhere else. The effect of the capture can be cancelled by giving a secondary channel number of 0FFh which is not a valid channel number.

Function 18 - Re-direct channel

Parameters:	A -	Main channel number
	C -	Secondary channel number (0FFh to cancel redirection)
Returns:	A -	Status

The re-direct function causes subsequent output sent to the main channel with write character or write block function calls to be sent to the secondary channel instead. The redirection lasts until the secondary channel is closed or returns an error or the main channel is redirected somewhere else. A secondary channel number of 0FFh will cancel any redirection of the main channel.

Function 19 - Set default device name

Parameters:	DE -	device name pointer (without a colon)
	C -	device type 0 => non file handling 1 => file handling
Returns:	A -	status

The set default device name function specifies a device name and (optionally) a unit number which will be used in an OPEN or CREATE function call if none is specified by the user. Initially the default name will be TAPE-1 but will be set to DISK-1 if a disk device is linked in. The specified device name and unit number are checked for legality (ie. no invalid characters) but not for existence in the device chain.

If a string with only a unit number, such as "45" is specified then the new number will become the default but the default name will be un-changed.

The "device type" given in register C is simply copied to the "device type" EXOS variable. This will be zero in the default machine because the default device is "TAPE" which is not a file handling device. If a disk unit is connected then the device type will be set to 1.

 Copyright (C) 1984 Intelligent Software Limited

Function 20 - Return system status

```
Parameters:   DE -> Parameter block, 8 bytes.
Returns:       A =  Status code, always 0.
               B =  Version number (currently 1)
              DE - unchanged
```

This function returns the version number of the system and various parameters which describe the RAM segment usage in the system. The parameters returned are, in order:

0. Shared segment number (0 if no shared segment)
1. Number of free segments.
2. Number of segments allocated to the user, excluding the page-zero segment and the shared segment (if there is one).
3. Number of segments allocated to devices.
4. Number of segments allocated to the system, including the shared segment if there is one.
5. Total number of working RAM segments.
6. Total number of non-working RAM segments.
7. *** Not currently used ***

Function 21 - Link Device

```
Parameters:   DE - Pointer to RAM in Z-80 space
                   containing device descriptor.
              BC - Amount of device RAM required.
Returns:       A - status
```

The link device function causes the device descriptor pointed to by DE to be linked into the descriptor chain. The descriptor will be put at the start of the chain and any existing device with the same name will be disabled. DE must point at the TYPE field of the descriptor and the descriptor must not cross a segment boundary. Once linked in the user must ensure that the device code and descriptor are not corrupted until a RESET function call with bit-5 set (to un-link user devices) has been made.

The amount of RAM requested will be allocated in the system segment. Whenever the device is called this RAM area will be pointed to by IY (although it can never move). IY will point at the byte just beyond the end of the RAM so if, for example, two bytes were requested then the two bytes would be (IY-1) and (IY-2).

 Copyright (C) 1984 Intelligent Software Limited

Function 22 - Read EXOS Boundary

```
Parameters:   none
Returns:      A  - status  (Always zero)
              C  -  Shared  segment number.  0  if
                    there is no shared segment.
              DE -  EXOS boundary in shared segment
                    (0..3FFFh)
```

The read EXOS boundary function returns the offset within the currently shared segment of the lowest byte which the system is using. If there is no shared segment then DE will point to where the EXOS boundary would be if a shared segment were allocated.

Function 23 - Set User Boundary

```
Parameters:    DE  - Offset  of new USER  boundary.
               (0...3FFFh)
Returns:       A  - Status
```

The set user boundary function allows the user to move the USER boundary within the currently shared segment. If there is no shared segment then this function is not allowed. The boundary may not be set higher than the current EXOS boundary.

Function 24 - Allocate Segment

```
Parameters:   none
Returns:      A  - status
              C  - Segment number
              DE - EXOS boundary within segment
```

The allocate segment function allows the user to obtain another 16K segment for his use. If a free segment is available then it will be allocated and status returned zero with segment number in C and DE will be 4000h.

If there are no free segments but the user can be allocated a shared segment, then the segment number will be returned in C and DE will be the initial EXOS boundary. In this case a non-zero positive status code (.SHARE) will be returned. The user boundary is initially set equal to the EXOS boundary.

If there are no free segments and there is already a shared segment then an negative status code will be returned.

If this function call is made by a device driver then the segment will be marked as allocated to a device and a shared segment can not be allocated.

 Copyright (C) 1984 Intelligent Software Limited

Function 25 - Free segment

```
Parameters:    C - Segment number
Returns:       A - status
```

The free segment function allows the user to free a 16k segment of RAM. The segment must be currently allocated to the user or be shared. The page-zero segment cannot be freed as it was never allocated explicitly with an ALLOCATE SEGMENT call.

If this function call is made by a device driver then it must be to free a segment which was allocated to a device driver with an ALLOCATE SEGMENT call. There is no checking of which device is freeing the segment - Devices are supposed to be well behaved.

Function 26 - SCAN ROM

```
Parameters:    DE = Pointer to application name string
Returns:        A = Status
```

This is the function described earlier which allows the user to cause a ROM scan to occur with action code 2 (or 3 if the first word of the string is "HELP"). This allows external ROM services to be used and also to transfer from one applications ROM to another. Device drivers are not allowed to make this call.

Function 27 - Allocate Channel Buffer

```
Parameters:     DE - Amount of buffer which must be
                        in one segment
                BC  - Amount of buffer which needn't
                        be in one seg.
Returns:        A - status
               IX -> Points newly allocated buffer
             PAGE1    contains the new buffer segment
```

The allocate channel buffer function is provided only for devices and may not be called by the applications program. It is used to provide a channel with a RAM buffer when it is opened. The "multi segment size" passed in register BC is ignored for non-video devices since they must have their channel buffer all in one segment. So for non-video devices BC need not be loaded before making the call

 Copyright (C) 1984 Intelligent Software Limited

Function 28 - Explain Error Code

```
Parameters:    A  - Error code which needs explaining
              DE  - Pointer to buffer for string (80
                    characters)
Returns:       A  = 0
              DE  - Unchanged
```

This function allows an EXOS error code to be converted into a short text message. All external ROMs are given a chance of doing the translation, and some error codes are explained by the internal ROM. If the string returned is of zero length then it is an error code which no ROM was willing to explain.

To ensure compatibility with later versions new error codes should be below 0C0h since 0C0h to 0FFh are reserved for internal EXOS use, although not all are used at present.

5 Device Driver Interface

5.1 Format of Device Descriptors

The format specified here is of a complete device descriptor in RAM. Whenever a device is entered IY is set to point to the TYPE byte of the descriptor in Z-80 page-2. However the segment containing the descriptor is not explicitly paged in here, since this page must contain the system segment. In in the case of built in and external devices, the descriptor will always be in the system segment and so it will be accesible via IY. This is not the case however for user devices since the descriptor for them could be anywhere. It is assumed that user devices know where their descriptor is (or else don't care).

```
IY - 3 :  NEXT_LOW \  24-bit address of next descriptor in
IY - 2 :  NEXT_HI   > the chain. The address will be in
IY - 1 :  NEXT_SEG /  Z-80 page-1. End of chain marked by
                      Segment number 0.

IY + 0 :  TYPE     -  Must be zero. To allow for future
                      expansions.
```

 Copyright (C) 1984 Intelligent Software Limited

IY + 1 : IRQFLAG - A byte specifying which interrupts this device should service. If it is zero then the interrupt entry point (see below) need not be valid. The layout of the byte is:

b1 - Set for servicing sound interrupts
b3 - Set for servicing 1Hz interrupts
b5 - Set to service video interrupts
b7 - Set to service external interrupts
b0,2,4 & 6 - unused, must be zero

IY + 2 : FLAGS - General flags byte. Currently only one flag defined:

b0 - Set if device is video device. Channel RAM must be in video RAM.
b1-b7 - Unused - must be zero.

IY + 3 : TAB_LOW \ 24-bit address of the start of a
IY + 4 : TAB_HI > table of entry points to the device.
IY + 5 : TAB_SEG / Must be in Z-80 page-1.

IY + 6 : UNIT_COUNT - How many unit numbers this device can serve. See text for details, zero for all built in devices.

IY + 7 : NAME - The device name. The first byte is a length byte, defining the number of characters (0..28) in the name, excluding the length byte itself. The name can include upper case letters, digits, underlines and periods.

5.2 User Devices

When a user device is linked in a pointer to a device descriptor must be given. This points to the TYPE field of the descriptor. The three bytes before this will be set up by EXOS and therefore need not be initialised before linking in. The full 24-bit entry table pointer must be set up to point to the correct segment.

 Copyright (C) 1984 Intelligent Software Limited

5.3 Built in and External Devices

The internal ROM and any expansion ROMs contain a chain of device descriptors which is scanned at startup time to link in all the devices. There is a pointer to the first descriptor at offset 0008h in the ROM. Each descriptor in the chain must give all fields except the three byte link (NEXT_LOW, NEXT_HI, NEXT_SEG) and the segment of the entry point table (TAB_SEG). The entry point table is assumed to be in the same segment as the descriptor chain and this entry will be filled in when the descriptor is copied to RAM.

The format of the chain of descriptors is:

```
        XX_NEXT_LOW \  Pointer to next descriptor.  Must
        XX_NEXT_HI  /  be in z-80 page-1.Points at SIZE field.

        XX_RAM_LOW  \  Size of RAM area required (see text)
        XX_RAM_HI   /

        TYPE        \
        IRQFLAG      \
        FLAGS         |
        TAB_LOW       |    These fields are copied to RAM.
        TAB_HI        >    They are as defined earlier
          not used    |
        UNIT_COUNT    |
        NAME          |
          .          /
          .         /

--->    SIZE           Number of bytes in descriptor
                       (see text).
```

The end of the chain is marked by a pointer of 0000h. Thus if a ROM contains no devices there should be a 0000h at offset 0008h in the ROM. The pointer points at the size field which is a single byte of the number of bytes in the descriptor from the TYPE field to the end of the NAME. This is variable since the name can be of variable length.

The XX_RAM_LOW/HI field is a 16-bit number specifying the size of device RAM area which should be allocated immediately below the descriptor in RAM. It must include two bytes to provide the link at the start of the descriptor and is given in twos complement form. Thus for the built in devices which will require no RAM here, it should have a value of 0FFFEh (-2). The RAM area can be accessed at (IY-4), (IY-5), etc.

 Copyright (C) 1984 Intelligent Software Limited

5.4 Device Entry Points

Each device descriptor contains a pointer to a table of entry points for the device. Each entry point is a 16-bit address which must be in Z-80 page-3 and be in the same segment as the entry point table itself.

The format of the entry point table is:

0. Interrupt entry point (Need only be valid if IRQFLAG is non-zero).
1. OPEN CHANNEL
2. CREATE CHANNEL
3. CLOSE CHANNEL
4. DESTROY CHANNEL
5. READ CHARACTER
6. READ BLOCK
7. WRITE CHARACTER
8. WRITE BLOCK
9. CHANNEL READ/WRITE STATUS
10. SET AND READ CHANNEL STATUS
11. SPECIAL FUNCTION
12. Initialisation entry point
13. BUFFER MOVED entry point

On entry to any of these routines, the code will be executing in Z-80 page-3. Page-2 will contain the system segment which includes the EXOS stack which will be active. The entry is made by a CALL instruction in page-2 and so the return can be an ordinary RET instruction. Page-0 will contain the page-zero segment with the interrupt and function call entry points in (interrupts will be enabled). For entry points 3 to 11 and 13, page-1 will contain the channel descriptor and start of the channel RAM buffer for the appropriate channel. For other entry points the contents of page-1 are undefined (although for OPEN and CREATE it will contain the channel descriptor after the ALLOCATE CHANNEL BUFFER call has been made).

EXOS does sufficient checking of the stack pointer to ensure that whenever a device is entered , it can use 100 bytes of stack (this number may change) and still have enough stack to make EXOS calls and service interrupts. A device's interrupt routine is assured of 50 bytes (also subject to change) with enough left to make a SET EXOS VARIABLE call.

When page-1 contains the channel descriptor, IX will point at the start of the channel RAM as defined earlier. IY will point to the devices RAM area (depending on the type of the device).

 Copyright (C) 1984 Intelligent Software Limited

Entry points 1 to 11 correspond to the EXOS channel calls and parameters are mostly passed straight from the caller. There is an exception for OPEN and CREATE in which the parameters are processed slightly (see earlier). All of them must return an EXOS status code in register A but the flags need not be set consistent with this code since this is done by EXOS.

5.5 BUFFER MOVED Entry Point

This entry point is provided mainly for the video driver which has to keep the line parameter table up to date when buffers are moved, but will also be needed by other devices. If it is not needed it can simply point to a RET instruction (no status code is returned).

It is called by EXOS immediately after a channel buffer has been moved. On entry IX will point to the new position of the channel RAM in Z-80 page-1 in the usual way and A will contain the channel number. Also BC will contain the amount that the buffer was moved by. This is only a 16-bit number and so its sign is ambiguous since the buffer could have been moved by almost 64k either way. This should not be a problem since apart from the video device only the offset (ie lower 14 bits) will ever be relevant.

5.6 Interrupts

EXOS handles all types of interrupts. When an interrupt occurs EXOS examines the hardware to determine which of the sources of interrupt caused it. It then scans through all devices calling the interrupt entry point of each one which had the appropriate bit set in the IRQFLG byte in its descriptor (see earlier).

If two different types of interrupt occur at the same time then they will both be serviced, although one of them will have to wait until the other has been done. There is no nesting of interrupts.

On entry to a device interrupt routines, all registers including the index registers and the alternate register set will have been saved so may be corrupted. On entry, register A will contain a single bit set to indicate which type of interrupt is being serviced, since an interrupt service routine may service more than one type of interrupt.

 Copyright (C) 1984 Intelligent Software Limited

Interrupts will be disabled before entering any interrupt service routine and it is anticipated that the routine will not re-enable them. Also the service routine should not attempt to reset the interrupt hardware since EXOS does this itself after all devices have been serviced.

There is an EXOS variable called IRQ_ENABLE_STATE which controls which sources of interrupts are enabled. If this is updated it will not affect the actual enable state of the interrupts until the next interrupt has been serviced.

+++++ End of Document +++++

 Copyright (C) 1984 Intelligent Software Limited

EXOS Function Codes and Error Codes

```
;*****************************************************************************
;
;
;   This file defines EXOS variables, function codes, error codes and software
; interrupt codes as externals.  It should be assembled to a .REL file and
; then linked with any code which uses EXOS.  It produces no object code, just
; defines symbols.
;
;
err_code        defl    100h
;
err             MACRO   name
err_code        defl    err_code - 1
                PUBLIC  name
name            equ     err_code
                ENDM
;
;
;
var     MACRO name,number
name    equ     number
        PUBLIC  name
        ENDM
;
;
;
;-----------------------------------------------------------------------------
;
;                       EXOS FIXED VARIABLES
;                      ======================
;
;
        var     USR_P3,         0BFFFh  ;Four segments which were in Z-80 space
        var     USR_P2,         0BFFEh  ; when EXOS was last called.
        var     USR_P1,         0BFFDh
        var     USR_P0,         0BFFCh
;
        var     STACK_LIMIT,    0BFFAh  ;Bottom limit of stack for devices.
;
        var     RST_ADDR,       0BFF8h  ;Warm reset address
;
        var     ST_POINTER,     0BFF6h  ;Address of status line RAM
        var     LP_POINTER,     0BFF4h  ;Address of start of video LPT.
;
        var     PORTB5,         0BFF3h  ;Current contents of Z-80 port 0B5h.
;
        var     FLAG_SOFT_IRQ,  0BFF2h  ;Flag <>0 to cause software interrupt.
;
;
;
;-----------------------------------------------------------------------------
```

```
;                    FUNCTION CODES
;                  ==================
;
;
        var     @RESET, 0               ; Reset system
        var     @OPEN,  1               ; Open channel
        var     @CREAT, 2               ; Create channel
        var     @CLOSE, 3               ; Close channel
        var     @DEST,  4               ; Destroy channel
        var     @RDCH,  5               ; Read character
        var     @RDBLK, 6               ; Read block
        var     @WRCH,  7               ; Write character
        var     @WRBLK, 8               ; Write block
        var     @RSTAT, 9               ; Read status
        var     @SSTAT, 10              ; Set channel status
        var     @SFUNC, 11              ; Special vartion
        var     @EVAR,  16              ; Set/read/toggle EXOS variable
        var     @CAPT,  17              ; Capture channel
        var     @REDIR, 18              ; Re-direct channel
        var     @DDEV,  19              ; Set default device
        var     @SYSS,  20              ; Return system status
        var     @LINK,  21              ; Link device
        var     @READB, 22              ; Read EXOS boundary
        var     @SETB,  23              ; Set USER boundary
        var     @ALLOC, 24              ; Allocate segment
        var     @FREE,  25              ; Free segment
        var     @ROMS,  26              ; Locate ROMs
        var     @BUFF,  27              ; Allocate channel buffer
        var     @ERRMSG,28              ; Return error message.
```

```
;*****************************************************************************
;
;
;                       ERROR CODES
;                     ================
;
;
;       General errors returned by the EXOS kernel
;
        err     .IFUNC  ;0FFh   Invalid function code
        err     .ILLFN  ;0FEh   EXOS function call not allowed
        err     .INAME  ;0FDh   Invalid name string
        err     .STACK  ;0FCh   Insufficient stack
;
        err     .ICHAN  ;0FBh   Invalid channel number (0FFh) or channel
                        ;           does not exist.
        err     .NODEV  ;0FAh   Device does not exist  (OPEN/CREATE)
        err     .CHANX  ;0F9h   Channel already exists (OPEN/CREATE)
        err     .NOBUF  ;0F8h   No ALLOCATE BUFFER call made (OPEN/CREATE)
        err     .BADAL  ;0F7h   Bad ALLOCATE BUFFER params. (OPEN/CREATE)
        err     .NORAM  ;0F6h   Insufficient RAM for buffer.
        err     .NOVID  ;0F5h   Insufficient video RAM.
;
        err     .NOSEG  ;0F4h   No free segments (ALLOCATE SEG)
        err     .ISEG   ;0F3h   Invalid segment (FREE SEG, SET BOUNDARY)
        err     .IBOUN  ;0F2h   Invalid user boundary (SET USER BOUND)
        err     .IVAR   ;0F1h   Invalid EXOS variable number
        err     .IDESC  ;0F0h   Invalid device descriptor type (LINK DEV)
;
;
;
;       General errors returned by various devices
;
        err     .ISPEC  ;0EFh   Invalid special function code
        err     .2NDCH  ;0EEh   Attempt to open second channel
        err     .NOFN   ;0EDh   Function not supported
        err     .ESC    ;0ECh   Invalid escape character
        err     .STOP   ;0EBh   Stop key pressed
;
        public  .SWI
;
;
;
;       File related errors
;
        err     .NOFIL  ;0EAh   File does not exist
        err     .EXFIL  ;0E9h   File exists (CREATE)
        err     .FOPEN  ;0E8h   File already open
        err     .EOF    ;0E7h   End of file met in read
        err     .FSIZE  ;0E6h   File is too big
        err     .FPTR   ;0E5h   Invalid file pointer value.
        err     .PROT   ;0E4h   Protection violation
;
;
;
```

```
;       Keyboard errors
;
        err     .KFKEY  ;0E3h   Invalid function key number
        err     .KFSPC  ;0E2h   Run out of function key space
;
;
;
;       Sound errors
;
        err     .SENV   ;0E1h   Envelope is too big or number 255.
        err     .SENBF  ;0E0h   Not enough room to define envelope
        err     .SENLO  ;0DFh   Envelope storage requested too small (ie. <2)
        err     .SQFUL  ;0DEh   Sound queue is full (and WAIT_SND <> 0)
;
;
;
;       Video errors
;
        err     .VROW   ;0DDh   Invalid row number to scroll
        err     .VCURS  ;0DCh   Attempt to move cursor off page
        err     .VCOLR  ;0DBh   Invalid colour passed to INK or PAPER
;
        err     .VSIZE  ;0DAh   Invalid X or Y size to OPEN
        err     .VMODE  ;0D9h   Invalid video mode to OPEN
        err     .VDISP  ;0D8h   Naff parameter to DISPLAY
        err     .VDSP2  ;0D7h   Not enough rows in page to DISPLAY
;
        err     .VBEAM  ;0D6h   Attept to move beam off page
        err     .VLSTY  ;0D5h   Line style too big
        err     .VLMOD  ;0D4h   Line mode too big
        err     .VCHAR  ;0D3h   Can't display character on graphics page
;
;
;
;       Serial errors
;
        err     .BAUD   ;0D2h   Invalid baud rate
;
;
;
;       Editor errors
;
        err     .EVID   ;0D1h   Invalid video page for OPEN.
        err     .EKEY   ;0D0h   Trouble in communicating with keyboard.
        err     .ECURS  ;0CFh   Invalid co-ordinates for positioning cursor.
;
;
;
;       Cassette errors
;
        err     .CCRC   ;0CEh   CRC error from cassette driver
;
```

```
;
;       Network errors
;
        err     .SEROP  ;0CDH   Serial device open - cannot use network
        err     .NOADR  ;0CCH   ADDR_NET not set up
;
;
;
;
printe  macro   e
          .printx       * Smallest error code is e *
        endm
;
;
        IF1
          printe        %err_code
        ENDIF
;
;
;-------------------------------------------------------------------------------
;
;                 WARNING CODES
;                ================
;
        var     .SHARE, 07Fh            ;Shared segment allocated
```

```
;*******************************************************************************
;
;
;                          EXOS VARIABLE NUMBERS
;                          =====================
;
;
;
;
        var     $IRQ_ENABLE,    0       ; Interrupt enable bits.
;
        var     $FLAG_SIRQ,     1       ; Flag to cause a software interrupt.
        var     $CODE_SIRQ,     2       ; Software Interrupt code.
;
        var     $DEF_TYPE,      3       ; Type of default device.
        var     $DEF_CHAN,      4       ; Default channel number
;
        var     $LOCK_KEY,      5       ; Keyboard lock status.
        var     $CLICK_KEY,     6       ; Key click enable/disable.
        var     $STOP_IRQ,      7       ; Software interrupt on STOP key.
        var     $KEY_IRQ,       8       ; Software interrupt on any key press.
        var     $RATE_KEY,      9       ; Keyboard auto-repeat rate.
        var     $DELAY_KEY,     10      ; Delay before auto-repeat starts.
;
        var     $TAPE_SND,      11      ; Tape sound enable/disable.
;
        var     $WAIT_SND,      12      ; Sound driver waiting if buffer full
        var     $MUTE_SND,      13      ; Sound mute enable/disable.
        var     $BUF_SND,       14      ; Sound envelope storage size.
;
        var     $BAUD_SER,      15      ; Serial baud rate.
        var     $FORM_SER,      16      ; Serial word format.
        var     $ADDR_NET       17      ; Network address of this machine
        var     $NET_IRQ,       18      ; Software interrupt on network.
        var     $CHAN_NET,      19      ; Channel of network interrupt
;
        var     $MODE_VID,      20      ; Video mode.
        var     $COLR_VID,      21      ; Video colour mode.
        var     $X_SIZ_VID,     22      ; Video X page size.
        var     $Y_SIZ_VID,     23      ; Video Y page size.
;
        var     $ST_FLAG,       24      ; Status line displayed flag.
        var     $BORD_VID,      25      ; Border colour.
        var     $BIAS_VID,      26      ; Fixed bias colour.
;
        var     $VID_EDIT,      27      ; Video channel number.
        var     $KEY_EDIT,      28      ; Keyboard channel number.
        var     $BUF_EDIT,      29      ; Size of edit buffer.
        var     $FLG_EDIT,      30      ; Editor control flags
;
        var     $SP_TAPE,       31      ; Cassette I/O speed.
        var     $PROTECT,       32      ; Cassette protection control
        var     $REM1,          33      ; Cassette remote 1
        var     $REM2,          34      ; Cassette remote 2
```

```
;*******************************************************************************
;
;
;               SOFTWARE INTERRUPT CODES
;              ============================
;
;
        var     ?FKEY,  10h             ;Function keys 10h...1Fh
        var     ?STOP,  20h             ;Stop key
        var     ?KEY,   21h             ;'any key' 21h
;
        var     ?NET,   30h             ;Network data received.
;
;
;
;
```

1 Introduction

The screen driver handles the display of any number of video "pages" in the various different display modes which the video chip provides although it does not support all the possible modes.

The display is managed in terms of video "pages", with one page corresponding to each channel. Before a channel is opened to the screen driver the user must specify various parameters, such as a video mode and page size, by setting EXOS variables. A channel can then be opened to device **"VIDEO:"**. If a filename or unit number is specified then it will be ignored. The video driver will work out how much screen RAM it needs for this video page and obtain that much RAM from EXOS, including enough for the various variables needed.

Once the channel has been set up in this way, the user can read and write characters or blocks of data. This data will be interpreted differently by pages of different modes, particularly control characters and escape sequences.

At this stage however, the video page will not be visible on the display. A special function call is required to cause a video page to be actually displayed on the screen. It is only at this time that the appropriate line-parameter blocks are set up and the text/graphics will appear. It is possible to display any vertical section of a video page at any vertical position on the screen, covering up anything which was displayed on those scan lines before. If the page width is less than the full screen width then the margins will be adjusted to display the page in the middle of the screen.

The screen driver has a 256 character font in video RAM which it uses for all character type displays. This is initialised to a standard ASCII character set repeated twice but any character may be re-defined by the user. Each character is 8 pixels wide and 9 lines deep. These values include the space between characters and between lines.

1.1 Co-ordinate Systems

The co-ordinate system used in specifying graphic positions etc is standardised so that giving the same commands to two pages of different resolutions or colour modes will produce a pattern of the same size on the screen. A graphics page of full screen size will be 972 logical pixels high and 1344 pixels wide. This corresponds to twice the maximum horrizontal and four times the vertical resolution available. All beam positions etc. are specified in these co-ordinates, and depending on the colour mode the actual position will have to be an approximation.

 Copyright (C) 1984 Intelligent Software Limited

Text pages do not use this co-ordinate system, they use a system based on character positions so the top left corner is (1,1) and the top right corner (of a full screen size low resolution text page) is (1,42).

Attribute graphics pages actually keep a beam position in graphics co-ordinates and a seperate cursor position in text co-ordinates. The use of these is explained later.

2 Basic Control of Video Pages

As mentioned before, each video page is a seperate channel. When a channel is opened to the video driver this implies that another video page is to be created. The video driver looks at EXOS variables which specify the page size, page mode and colour mode. These variables must be set up by the user before opening a video channel. From these variables the video driver determines how much video RAM it needs and obtains that much with an EXOS function call ("Allocate channel buffer").

The video driver maintains the line parameter table in a fixed place in its absolute device RAM area. The line parameter table always consists of 28 line paraneter blocks of 9 scan lines each for the display area and various other ones to generate the frame sync and borders. The first line parameter block is reserved for the status line display which is a fixed area of RAM. The other 27 line parameter blocks can display any part of any page, so display is always in vertical units of 9 pixels. All 28 line parameter blocks are initially set up to be blank (ie all border colour). The variable LP_POINTER in the EXOS variable area points to the start of the line parameter table.

2.1 Display Modes

The display mode is specified by an EXOS variable MODE_VID the allowed values of which are:

```
 0 - Hardware text mode (up to 42 chars/line).
 1 - High resolution pixel graphics.
 2 - Software text mode (up to 84 chars/line).
 5 - Low resolution pixel graphics.
15 - Attribute graphics.
```

The three graphics modes correspond to the PIXEL, LPIXEL and ATTRIBUTE modes of the Nick chip (see separate Nick chip specification).

 Copyright (C) 1984 Intelligent Software Limited

2.2 Colour Modes

As well as the display mode, each video page is of a particular colour mode. The colour mode is specified by an EXOS variable called COLR_VID. The allowed values for this variable are:

0 - Two colour mode
1 - Four colour mode
2 - Sixteen colour mode
3 - 256 colour mode

For text modes it is only useful to use two colour mode, unless the characters in the font are re-defined for doing some sort of block graphics. Also attribute mode must always be in two colour mode, although sixteen colours will actually be available.

2.3 Page Size

Two EXOS variables, X_SIZ_VID any Y_SIZ_VID, define the size of the page to be created. The vertical size is specified in character rows. It can be any value from 1 to 255 although only 27 rows can be displayed on the screen at one time. The horizontal size is specified in low resolution character widths, and can be any number from 1 to 42.

A special function call is provided to return the size of a video page. It returns the number of lines and the number of characters per line. The characters per line value returned is the actual number of characters per line so in the case of a software-text mode it will be double the value in X_SIZ_VID when the channel was opened.

The parameters for this call are:

Parameters: A = Channel number (1...255)
B = 2 (Special function code)

Returns: A = Status
B = Number of characters per row
C = Number of rows
D = Mode of page (0, 1 or 2, 5 or 15)

 Copyright (C) 1984 Intelligent Software Limited

2.4 Display Control

Video pages are not actually displayed on the screen until the user explicitly requests this. This request is done by a special function call. The parameters for this call are:

Parameters:
- A = Channel number (1...255)
- B = 1 (Special function code)
- C = First row in video page to display (1...size)
- D = Number of rows to display (1...27)
- E = row on screen where first row should display (1...27).

Returns: A - Status

The three row parameters are all given in character row units since the area of screen specified must be a whole number of line parameter blocks. The displayed page will replace anything which was displayed on that part of the screen before. If the channel is subsequently closed then any part of the screen which was displaying that channel will be made border colour (by bringing the margins in the relevent line parameter blocks right in).

A value of 1 for the position on screen parameter (given in register E) refers to the line on the screen directly below the status line. Thus it is not possible to overlay the status line since zero will not be accepted.

If a value of zero is given for the position in the page parameter (register C) then the portion of the screen defined by the other two parameters will be blanked (ie. made entirely border colour).

3. Character Output

The screen driver supports both the single character write and the block write EXOS function calls. A block write is exactly equivalent to writing all the characters individually, except that it is rather faster as it avoids the overhead of going through EXOS for every character. Block write is implemented using the general purpose WRBLOCK utility routin.

3.1 Printing Characters

All characters above 1Fh will be treated as printing characters and will be put at the appropriate place on the video page. All modes have some sort of "cursor" which moves when a charcater is printed but the details vary between different modes.

The bit maps for characters are stored in a fixed character font which is initialised to an ASCII character set. Each character is eight bits wide and nine bytes deep. The user can re-define any of these characters by an

 Copyright (C) 1984 Intelligent Software Limited

escape sequence as specified below.

3.1.1 Text Mode Character Printing

Text pages (modes 0 and 2) maintain a single text cursor which in text co-ordinates. The printing character is displayed at this position and the cursor moved to the next character slot. At the end of a line the cursor automatically moves to the start of the next line, with automatic scrolling if it is at the bottom of the screen (this automatic scrolling can be disabled).

3.1.2 Pixel Graphics Mode Character Printing

Pixel graphics pages maintain a beam pointer in graphics co-ordinates. The printing character is put at this beam position and the beam moved to the next character position, moving to the start of the next line if at the end of a line. Characters can be put at any pixel position, not just on character boundaries. There is no scrolling. If the beam is too near the bottom of the page to fit the character on then it will not print anything.

Characters are printed in the current ink colour regardless of whether the beam is on or off and the paper colour of the charcater is unaffected. The characters are plotted by reading bits out of the font and plotting the pixel in the current ink colour if the bit is set. Thus the characters will be correct in any colour mode. To improve legibility the character height is doubled for sixteen and 256 colour mode.

3.1.3 Attribute Graphics Mode Character Printing

Attribute mode character printing is rather more complex. All attribute graphics pages maintain a seperate text cursor in text co-ordinates and a graphics beam position in graphics co-ordinates. Characters are printed at the text cursor position and so will always be in exact character positions. At the end of a line it will go on to the start of the next line and at the end of the page it will go back to the top left of the page - there is no scrolling.

The character is displayed by copying the currently selected ink and paper colours into all attribute bytes corresponding to this character (nine of them). The character data itself from the font is then put into the pixel data area but only if the beam is on. If the beam is off the pixel data is unaffected, only the attrbute data is affected. This allows the colours of part of an existing display to be changed.

 Copyright (C) 1984 Intelligent Software Limited

3.2 Control Codes and Escape Sequences.

Character in the range 00h to 1Fh are control characters and are not printed. Some of these are interpreted by video pages, depending on the mode. Any which are not understood are simply ignored. A special control code is ESCAPE (ASCII 1Bh) which is used to start an escape sequence for carrying out various functions.

Here is a list of the control codes and escape sequences interpreted by the various modes. The more complex ones of these are explained further in the following sections.

3.2.1 Codes Interpreted by Any Video Page

^Z (1Ah) - Clear entire page and home cursor/beam.

^J (0Ah) - Line-feed. Move cursor down to next line (scrolls if at bottom of screen in text mode and scroll is enabled.)

^M (0Dh) - Carriage return. Returns cursor to start of current line

^^ (1Eh) - Cursor/beam home. (ASCII RS)

escK - Define character (see below)

escC - Set all palette colours \ see below for parameters.
escc - Set one palette colour /

escI<n> - Set ink colour to <n> \ See below for details of
escP<n> - Set paper colour to <n> / these in different modes

esc=<y><x> - Set cursor position (see below)

3.2.2 Codes Interpreted by Text Pages Only

^Y (19h) - Clear to end of line. Does not move cursor.

^H (08h) - Cursor left. (ASCII BS)
^I (09h) - Cursor right. (ASCII TAB)
^K (0Bh) - Cursor up. (ASCII VT)
^V (16h) - Cursor down. (ASCII SYN)

esc? - Read cursor position. Also supported in attribute mode. (see below)

esc.<n> - Set cursor character to character code <n>.

escM<n> - Set cursor to palette colour <n>

escO - Set cursor on.
esco - Set cursor off.

 Copyright (C) 1984 Intelligent Software Limited

```
escS          - Set automatic scroll on
escs          - Set automatic scroll off

escU<m><n>    - Scroll up lines (m-20h) to (n-20h).    m <= n
escD<m><n>    - Scroll down lines (m-20h) to (n-20h).  m <= n
```

3.2.3 Codes Interpreted by Graphics Pages Only

```
escA<xx><yy> - Position beam at co-ordinates (xx,yy) where
               xx & yy are each 16-bit hex numbers specified
               low byte first.

escR<xx><yy> - Relative beam movement by amount (xx,yy).

esc@         - Read beam position.  (see below)

escS         - Set beam on.
escs         - Set beam off.

esc.<n>      - Set beam to line style <n> - see below.
escM<n>      - Set beam to line mode <n> - see below.

escF         - Graphics fill - see below.
escE         - Plot ellipse  - see below.
```

3.3 Position Cursor

The escape sequence to position cursor works in all modes. In text mode it simply moves the cursor. In attribute graphics mode it moves the text cursor but leaves the graphics beam pointer alone. In pixel graphics modes it moves the graphics beam pointer to the appropriate text co-ordinates, so it will be on a character boundary.

The format of the escape sequence is:

```
esc=<y><x>
```

This sets the cursor to row (y-20h) and column (x-20h). If either <x> or <y> is 20h (thus setting row or column zero) then that co-ordinate will remain un-changed. This allows just the row or column to be set.

 Copyright (C) 1984 Intelligent Software Limited

3.4 Define Character

This escape sequence allows the user to re-define one of the 256 characters. Although it is sent to a specific channel, it actually affects a global character font and will thus affect other channels. The syntax of the escape sequence is:

escK<n><r1><r2><r3><r4><r5><r6><r7><r8><r9>

where: <n> is the character number (0...255)
<r1>...<r9> are the bytes for the nine rows of the character. <r1> is the top row.

Note: In high resolution text mode, only the middle six bits of the character bytes will actually be displayed as the other two are masked out and used to control the colour selection.

3.5 Palette Colours

Each video page has a palette of eight colours associated with it which will be initialised to some useful set of colours (such as black, red, green, yellow, blue, magenta, cyan, white). There is an escape sequence with which the user can change all these colours. the format of this is:

escC<c><c><c><c><c><c><c><c>

Each <c> is a byte specifying one of the palette colours and there must always be eight of them.

There is another escape sequence which allows just one palette colour to be changed. The format of this is:

escc<n><c>

Where <n> is the palette colour number 0...7 and <c> is the new value for this palette colour.

When new palette colours are selected any line parameter blocks which correspond to this video page will be updated.

3.6 Ink and Paper Colours

The user may specify a palette colour for both the ink and paper colour with seperate escape sequences. For all video modes the ink colour defaults to one and the paper colour to zero.

For pixel graphics pages the allowed ink and paper colours depend on the colour mode so it is 0 or 1 in two colour mode, 0...3 in four colour mode, 0...15 in sixteen colour mode and 0...255 in 256 colour mode. Pixels are always plotted in the current ink colour. The paper colour of the display is only changed when the page is cleared.

 Copyright (C) 1984 Intelligent Software Limited

For attribute graphics mode the ink and paper colours can be in the range 0...15 and they control what colours are put into the attribute bytes. When a character is plotted both the ink and paper colours of the relavent attribute bytes will be set up. When plotting lines, ellipses or filling however, only one of them is set up depending on the line mode (see later for details). The line mode also controls whether ones or zeroes are put into the pixel data area.

In four, sixteen or 256 colour text modes the ink and paper colours have no effect, the palette colours for each pixel are determined directly from bits in the character font.

In two colour hardware text mode the ink and paper colours are always (0,1) or (2,3). These interact with the top bit of the character number being printed. Basically the top bit is complemented if colour pair (2,3) is selected.

In two colour software text mode four colour pairs are available, (1,0), (2,3), (4,5) and (6,7). Characters are always printed in the current colour pair. There is no interaction with the character codes.

3.7 Graphics Line Style

For a graphics page the line-style may be set with an appropriate escape sequence. This specifies a single byte has the following meanings:

1	-	Solid line (default)
2..14	-	Various types of broken and dotted lines.

3.8 Graphics Line Mode

An escape sequence specifies the line mode byte which has the following meanings:

0	-	PUT plotting (default)
1	-	OR plotting
2	-	AND plotting
3	-	XOR plotting

For pixel graphics pages when plotting a pixel the current ink colour is combined with the old colour of the pixel according to the operation selected by the line mode and then stored.

 Copyright (C) 1984 Intelligent Software Limited

For attribute graphics pages the line mode is more complex. Line modes 4 to 7 are also defined which are equivalent to 0 to 3 but do "paper" plotting rather than "ink" plotting. When paper plotting is selected the current paper colour is put into the attribute byte, leaving the ink colour in this byte unaffected. When ink plotting is seected just the ink colour is put into the attribute byte, leaving the paper colour unaffected. In adition the plotting on the pixel map will use a one bit for ink plotting and a zero bit for paper plotting. This bit will be combined with the old bit for this pixel according to the PUT/OR/AND/XOR mode and the result stored.

3.9 Graphics Fill - Paint

The graphics fill command is a simple escape sequence which does a fill from the current beam position. It fills in the current ink colour up to any boundry which is not the same colour as the current beam position. It handles concave shapes and tests for reaching the edge of the video page. It may fail to fill the entire shape if it runs out of stack but this should only happen with extremely complex shapes since it does garbage collection on the stack when it gets full.

3.10 Graphics Ellipse Drawing

The ellipse drawing routine takes two 16-bit parameters (low byte first) specifying the x and y radii. To draw a circle these should be the same value. The centre of the ellipse will be at the current beam position. The format of the escape sequence is:

escE<xx><yy>

3.11 Border and Fixed Bias Colours

Two EXOS variables BORD_VID and BIAS_VID are provided to control the hardware border and fixed colour bias registers. The values in these variables are written out to the NICK chip on every interrupt. The border colour is written directly to the border register. The top 5 bits of the fixed bias variable are written to the bottom 5 bits of the fixed bias register, and the top bit of the register is set according to the EROS variable MUTE_SND since it is used to silence the internal speaker.

 Copyright (C) 1984 Intelligent Software Limited

4. Character Input

4.1 Simple charcater input

When a read character (or one character of a read block) is done from the video driver the result depends on the mode. In text modes the ASCII code of the character at the cursor position will be returned, without moving the cursor. With graphics mode the palette colour of the pixel at the current beam position will be returned. This will be 0 or 1 in two colour mode, 0...3 in four colour mode, 1...15 if sixteen colour or attribute modes and 0...255 in 256 colour mode.

4.1 Reading Cursor Position

The escape sequence: esc? is supported in text and attribute modes. This triggers the video channel to return the current cursor co-ordinates as the next two characters read from this channel. The co-ordinates will be returned in the same way as they are specified for cursor positioning, ie. with a 20h offset added on.

4.2 Reading beam position

This is supported by graphics pages only (including attribute mode). The escape sequence: esc@ will trigger the channel to return the current graphics beam position as the next four bytes read from the video page. The co-ordinates are returned in the same format as they would be specified for an absolute beam position.

5. Status Line

As mentioned earlier there is a special status line display which is outide the normal page/channel structure. The first line parameter block is reserved for this status line and will never be used to display any other page. The two byte variable LP_POINTER which is at a fixed address in the EXOS variable area contains the address of the start of this line parameter block. This is used by the cassette driver to modify the palette colours of the status line to provide the cassette level meter display.

An EXOS variable (called ST_FLAG) is provided which if zero will cause the status line to be displayed and if non-zero will cause it to be un-displayed (that region of the screen will be border colour). This is implemented by the video driver examining ST_FLAG every interrupt and setting the margins in the reserved line parameter block appropriately.

 Copyright (C) 1984 Intelligent Software Limited

There is a two byte pointer (ST_POINTER) defined at a fixed address in EXOS variable area which contains the address of the 40 byte area of RAM which is the status line. This pointer will point to Z-80 page-2 and the RAM will be at this address in segment 0FFh. The user or external devices can access the status line by finding its address from ST_POINTER. Built in devices can access it directly by using the public symbol ST_LINE which is the start of the status line RAM (in Z-80 page-2 segment 0FFh).

 Copyright (C) 1984 Intelligent Software Limited

1. Introduction

The sound device driver provides all the sound control in the machine. It provides an interface which allows the user to manipulate most features of the sound chip.

It only allows one EXOS channel to be open to it at a time. The sound chip has four sound sources - three tone channels and a noise channel. The sound driver maintains a queue of sounds for each of these sound channels. Each sound in the queue will be played in turn when the one before it is finished. These sound queues are of a fixed maximum size of 25 sounds in each queue and will be held in the channel RAM.

As well as the sound queues, the sound device maintains a list of envelopes each with an envelope number 0...254. Each sound in the queues refers to a specific one of the envelopes or to the "null" envelope 255. When the sound is actually played the specified envelope controls changes in pitch and left and right amplitude of the sound throughout its duration. The storage for envelopes is in the sound device's channel RAM and the size of it must be specified with an EXOS variable before opening the channel.

2. General Device Interface

The sound device is write only - it will not accept any read function calls. Printing characters are ignored. All the sound functions are controlled by various control codes and escape sequences. It should accept write character and write block function calls in the obvious way.

It will have an interrupt routine which is entered 50 times per second (once every TV frame). This scans the sound queues and processes any sounds which are waiting or are currently being played. Each 20ms period is called a 'TICK' and all timing is in terms of ticks.

There are no special function calls for the sound driver. The following EXOS calls produce a FUNCTION NOT SUPPORTED error:

READ CHARACTER
READ BLOCK
SET/RETURN CHANNEL STATUS
SPECIAL FUNCTION

 Copyright (C) 1984 Intelligent Software Limited

An EXOS variable BUF_SND is used to specify how much storage is required in channel RAM for envelopes and must be set up before a channel to the sound driver is opened. It is specified in phases and the sound driver will obtain enough RAM to guarantee that the user can define envelopes with a total of the requested number of phases. Thus if the user requests 20 phases then he will be able to define one envelope with 20 phases in it or 20 envelopes each of one phase. Because of the overhead associated with each envelope, the former case takes up rather less RAM than the latter, so if 20 phases are requested then probably more than that number can be defined before storage will be exhausted since most envelopes have more than one phase. The number of phases requested can be 2...255.

3. Envelopes

3.1 General Description of Envelopes

An envelope consists of a series of between 1 and 40 phases each of which will be executed in turn when the envelope is used. Each phase is defined by four numbers:

PD - Duration of this phase in ticks (16 bits).
CP - Change value for pitch in 1/512 semitones (16 bits).
CL - Change value for left amplitude (8 bits).
CR - Change value for right amplitude (8 bits).

The change values are signed numbers to allow a change in either direction, up or down in pitch and louder or softer in amplitude. The pitch change can be any signed 16-bit number -32768...32767. The amplitude change values must be in the range -63...+63. They specify the total change in the appropriate parameter which is required during this phase. The specified change in pitch or amplitude will be spread linearly over the specified number of ticks as will be described later.

One particular phase of the envelope may be distinguished as the start of the release phase. The effect of this will be described in detail later but basically controls the dying away of the sound after the note has really finished.

3.2 Format of Envelope Definition

Envelopes are defined by an escape sequence sent to the sound channel:

esc E <en> <ep> <er> [<cp> <cl> <cr> <pd>]*

<en> = Envelope number 0...254 (8 bits).

 Copyright (C) 1984 Intelligent Software Limited

<ep> = Total number of phases in envelope 1...40 (8 bits).

<er> = Number of phases before release (8 bits). If no release phase is required then this should be 0FFh which will result in the sound finishing as soon as the sound duration is expired.

<cp> = Pitch change (16 bits) \ For each phase as described
<cl> = Left amp. change (8 bits) \ above, repeated up to 40
<cr> = Right amp. change (8 bits) / times (as specified by EP)
<pd> = Phase duration (16 bits) /

When an envelope definition is received, if an existing definition of that envelope exists then it is deleted. The new definition is then added to the list. If there is insufficient space to store it then an error code will be returned. If this happens then the old definition of that envelope will be lost.

4. Sound Production

To actually produce a sound an escape sequence must be sent which specifies the sound. The format of this is:

esc S <env> <p> <vl> <vr> <sty> <ch> <d> <f>

The meaning of each field is:

<env> - (8 bit) Envelope to use for this sound. An envelope number of 255 will produce a "beep" type sound which is of constant amplitude and pitch for the duration of the sound.

<p> - (16 bit) Starting pitch of sound in 1/512 semitones. Only exact quartertones will necessarily be musically correct. The others are generated by linear interpolation. Ignored for noise channel.

<vl> - (8 bit) Overall left amplitude. (0...255)
<vr> - (8 bit) Overall right amplitude. (0...255)

 Copyright (C) 1984 Intelligent Software Limited

<sty> - (8 bit) Sound style byte. For the noise channel, this byte is put into the noise control register for the duration of the sound. For a tone channel the top four bits are put into the four sound control bits in the sound frequency register for that channel, they thus control filtering, distortion and ring modulation. Zero gives a pure tone or white noise.

<ch> - (8 bit) Source for this sound. 0, 1 or 2 for the appropriate tone channel and 3 for the noise channel.

<d> - (16 bit) Duration of this sound in ticks.

<f> - (8 bit) Flags byte.
b0...b1 - SYNC count for this sound. See later section on synchronisation.
b2...b6 - Not used, should be zero.
b7 - Set to force over-ride of any sound in queue for this channel. Clear to wait its turn.

When a sound is received it is added to the end of the appropriate queue (killing the queue first if bit-7 of the flags byte is set). If the queue is full then there are two courses of action which depend on the state of the EXOS variable WAIT_SND. If this is zero then the sound driver will just wait until there is space in the sound queue, testing the stop key to allow it to be interrupted. If this EXOS variable is non-zero then it will return an error code .SQFUL.

5. Processing of Sounds

5.1 Pitch and Amplitude Control

If the envelope specified in a sound definition is not defined then the appropriate channel will be silent for the duration of the sound. The same thing applies if the envelope definition vanishes during the course of processing the sound.

The production of a sound under control of an envelope requires various current values to be kept. There will be the current pitch value which is initialised to the pitch value given in the sound specification. There will also be left and right current amplitude values, which are initialised to zero at the start of the sound.

 Copyright (C) 1984 Intelligent Software Limited

With these values initialised, processing of the sound can begin. Each phase of the envelope in turn is executed, each one lasting for the number of ticks specified in the envelope specification. At each tick, the current pitch and amplitude values are modified so that the change value in the envelope specification is spread linearly over the duration of the phase. The result at the end of each phase is that the signed change value has been added to the current value at the start of the phase, for each of the three parameters.

In the case of amplitude parameters the value is limited to the range 0 to 63. If an attempt is made by an envelope to make the amplitude go above 63 then the actual amplitude will just stick at 63. For pitch values there is no checking, the value will simply wrap around from 65535 to 0 and vice versa.

At each tick the current values of pitch and amplitude must be output to the sound chip itself. The details of how the register values are calculated are different for pitch and amplitude values.

For amplitude, the current left amplitude (6 bits) must be multiplied by the overall left amplitude specified in the sound definition (8 bits). The resulting 14 bit number is the required left amplitude. The top six bits of this value are be written to the appropriate DAVE register. The right amplitude is of course treated similarly.

The 16-bit pitch value must go through a logarithmic conversion process to produce in a counter value to go into the appropriate DAVE registers. The top byte of the pitch defines a quater-tone number from 0 to 255 (range 9-10 octaves). The counter value for this quarter-tone is found from a lookup table, with appropriate shifting depending on the octave. The top four bits of the lower byte of the pitch value are used to linearly interpolate between the selected counter value and the next one up. This gives a resolution of approximately 1/64 tone which is certainly adequate to produce smooth pitch changes.

5.2 Time control

While all the above processing of phases is going on, another counter is timing the length of the note itself. This is a 16 bit counter initialised to the sound duration specified in the sound definition, and decremented at each tick. If the end of the envelope is reached before this counter reaches zero then the sound will be silenced and remain silent until the counter does reach zero.

 Copyright (C) 1984 Intelligent Software Limited

If the sound duration counter reaches zero before the envelope has finished then two things happen. Firstly if the release phase of the envelope has not yet been reached then control skips to the start of the release phase. Secondly a flag is set to allow this sound to be over-ridden by another sound which is waiting in the queue or appears in the queue at a later time.

5.3 Synchronisation

The flags byte in the sound definition has two fields defined, the OVER-RIDE bit and the SYNC count. If the over-ride bit is set then the relevant sound queue is flushed before putting the new sound in, so it will be ready to start immediately. If it is clear then the new sound is just added to the queue to wait its turn.

The SYNC count is a two bit count which is ignored when the sound is put in the queue. The sound driver maintains an internal SYNC COUNT which is normally zero. When a sound reaches the head of a queue and is thus ready to be played, the SYNC byte in the sound is examined in conjunction with the internal SYNC counter.

If the SYNC count in the sound is zero then the sound is just started in the normal way with no synchronisation. If the SYNC count in the sound is non-zero then the sound is held up and the internal SYNC counter is examined. If it is zero then the SYNC counter from the sound is copied into it. If it is non-zero then it is decremented by one and if it goes to zero then all held up sounds are released simultaneously.

The effect of all this is that if three sound are to be played simultaneously then they should be queued up on their appropriate channels each with a SYNC count of two (one less than the number of sounds). The sound driver will then ensure that they are started simultaneously even if one of them gets to the head of its queue slightly earlier. This facility can thus be used to iron out slight timing differences in multi voice tunes.

 Copyright (C) 1984 Intelligent Software Limited

6. Other Control Functions

There are various other functions which the sound device provides which are listed here.

^Z - Flush all sound queues.

Esc Z <n> - Flush an individual sound queue.
n = 0, 1 or 2 for tone channel, 3 for noise channel.

^X - Flush all of envelope storage

^G - Cause a PING, see terminal bell below.

7. Interaction With Other Devices

7.1 Other Devices Accessing Sound Registers

Ideally the sound device would have exclusive use of all the sound registers. Unfortunately the cassette device has to fiddle with some of them for output and timing purposes. The sound driver sets up all sixteen sound registers on each interrupt so it will recover very quickly if its registers are corrupted.

7.2 Keyboard Click

The keyboard device has to click when a key is pressed. This event must be triggered from the keyboard device's interrupt routine and so cannot be done with an EXOS call. It is achieved by the keyboard device calling a globally defined routine KEYCLICK in the sound driver which uses tone channel zero to produce a click without interfering with any other sound which may be on this channel. The key click sound takes about 1/100 second and the routine does not return until it has finished.

 Copyright (C) 1984 Intelligent Software Limited

5.3 Terminal Bell

When the screen driver gets an ASCII code 7 (BELL) it is supposed to make an appropriate sound. It does this by calling a globally defined routine called BEEP in the sound driver. This uses tone channel 2 to make a bell sound. If there is already some sound on this channel then the routine does nothing. If there is no sound on this tone channel then a ping sound will be started and the routine will return. If another call to BEEP is made before the first ping has finished then it will hang up until the first one finishes, and then start the second one and return. .cp 3

A ping can also be triggered by sending an ASCII BELL character to the sound driver. This will have just the same effect as if the video driver calls the routine BEEP.

 Copyright (C) 1984 Intelligent Software Limited

INTRODUCTION

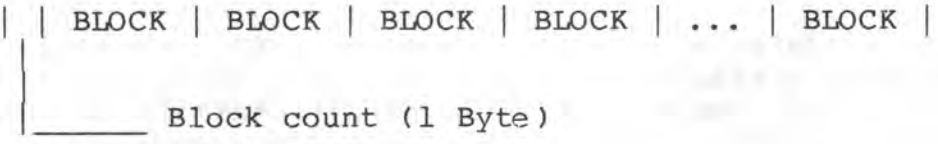
The EXOS cassette device allows bulk storage of data on cassette tapes. It operates within the EXOS channel scheme, supporting the normal character and block operations. This is accomplished by using a large (4Kbyte) RAM area to buffer characters between tape accesses. Data on cassette tape are organised into chunks of up to 4K with no restriction on the total length of data files.

The data on tape are stored as single cycles of variable frequency. The input circuitry digitises the incoming system with a zero-crossing detection network, allowing accurate timing of cycle periods. A leader tone of constant frequency precedes each 4K chunk to provide timing information. By taking its timing from the leader signal the cassette software can translate incoming data over a wide range of transmission rates, limited by the processor clock speed at high data rates and the low frequency roll-off of the input circuitry at low data rates. The optimum data rate is 2400 'baud' with an 8MHz processor clock.

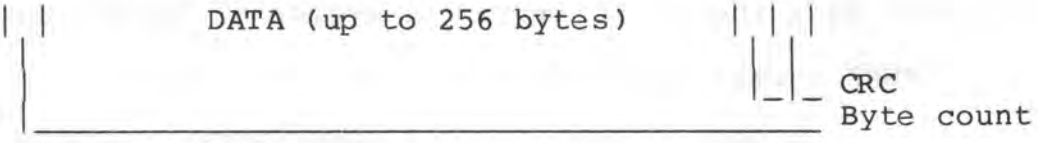
LOGICAL DATA FORMAT

The standard data chunk is organised into a string of 256 byte blocks, each block preceeded by a byte count and followed by a 2 byte cyclic redundancy check (CRC) value. The first block is preceded by a count of the number of blocks in the chunk. A chunk may contain from 0 to 16 256-byte blocks.

CHUNK format:



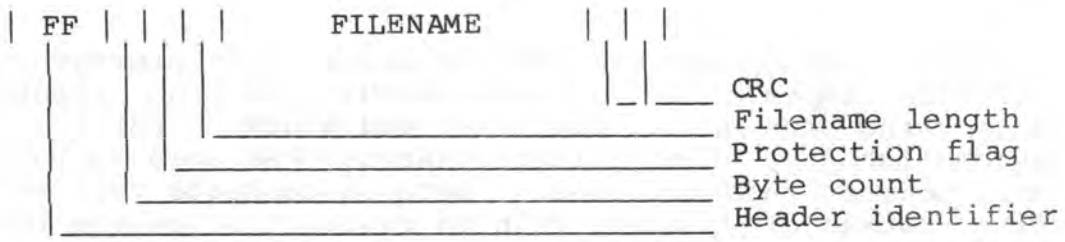
BLOCK format:



A data file comprises a sequence of data chunks preceded by a header chunk containing information identifying the file. The header contains exactly one block and is itself identified by a value of 0FFH in the block count byte. The header chunk also contains a one byte protection flag. A non-zero value for this flag inhibits the operation of BASIC's COPY function, thereby allowing a measure of data security for cassette files.

(c) 1984 Intelligent Software Ltd. (cge 10-9-84)

HEADER format:



The end of the data file (EOF) is sensed by three possible events:

1) Block count = 0
2) 0 < Block count < 16
3) Byte count < 256

Case (1) applies to a file that is an integral multiple of 4096 bytes long. In this case there would be a chunk of zero length following the last data-containing chunk. Case (2) is the normal situation where a chunk is more than 256 bytes below its 4K capacity. If a chunk contains 16 blocks the last of which is incomplete case (3) will recognise this as the EOF.

PHYSICAL DATA FORMAT

As mentioned above, data are stored as single cycles of variable frequency, the frequency relative to the leader frequency determining the logical bit value. A low frequency cycle corresponds to a logical 0, while a high frequency cycle represents a logical 1. The values of these frequencies are chosen to allow +/- 20% wow tolerance.

Each data chunk is preceeded by 2560 cycles of leader tone followed by a single low frequency cycle used to 1) identify the beginning of the data stream and 2) determine to phase of the incoming signal. There then follows a dummy byte to allow the capacitive filtering to recover from the frequency change between the leader and the synchronisation cycle. The data stream is followed by a footer of another 5 cycles of leader tone.

CHUNK format (physical):

| LEADER | SYNC CYCLE | DUMMY BYTE | DATA | FOOTER |

There are two available speeds for writing data to tape, corresponding roughly to 2400 and 1000 'baud'. The frequencies for these two speeds are listed below:

FAST RATES:	Leader	2.45 KHz
	Sync	0.99 KHz
	1	2.91 KHz
	0	1.99 KHz
SLOW RATES:	Leader	1.03 KHz

Sync	0.49 KHz
1	1.29 KHz
0	0.83 KHz

HARDWARE

The EXOS cassette driver makes extensive use of the variable rate interrupt sources provided by the DAVE chip. These are programmed to provide the time base for reading and writing. Only the lower 8 bits (out of the 12 available) are used.

Output signals are also generated from the DAVE chip using the left channel DAC (port A8). Values of 0 and 1 in this register produce an output square wave signal of +/- 21mV. The right channel is not used.

On input the cassette signal passes through a filtering network designed to remove low frequency noise from DC biased audio cassette recorders. This network tends to interfere with the incoming signal in such a way as to alter duty cycles when the frequency of the input signal changes drastically, as it does in the vicinity of the synchronisation pulse. It is to counteract this effect that a dummy byte is inserted immediately after the synchronisation pulse.

The filtered input signal then passes through a Schmidt trigger (with 25mV hysteresis) to produce a TTL output that is latched to b7 of port B6. The input circuitry provides accurate zero-crossing detection for input signals ranging from 15V peak-to-peak down to the hysteresis level of 25mV.

The optimum input level is +/- 3V. The strength of the input signal relative to this optimum is detected by a biased comparator buffered through a 10 microF capacitor to b6 of port B6.

OVERALL STRATEGY

In order to maintain compatability with the EXOS channel-based system, channel RAM is used to buffer characters on reading and writing operations. Also, since tape is not a random access device there is a necessary asymmetry between reading and writing. Therefore, the OPEN and CREATE EXOS commands, rather than doing roughly similar operations, are used for reading and writing, respectively. These operations correspond to the BASIC LOAD and SAVE commands.

```
OPEN   => READ from cassette => LOAD
CREATE => WRITE to cassette  => SAVE
```

All reading and writing operations are organised to use two large subroutines that read or write the data chunks to or from channel RAM. When a channel is CREATEd to the cassette driver the cassette driver sets up the header chunk in channel RAM and

then writes it to tape. The channel RAM is then filled with characters using either single or block write calls. When the buffer contents reach 4K the next chunk is written to tape. This continues until the channel is DESTROYed, at which time the remaining contents of the channel RAM buffer are written to tape.

Similarly, when a channel is OPENed to the cassette driver, the driver enters a searching routine to find the header corresponding to the desired file. If no filename is specified the first file encountered is used. The user then uses single or block character reads to access the contents of the channel RAM buffer. Whenever the user tries to read from the an empty buffer the cassette driver goes off to read the next chunk. Should that chunk contain one of the three EOF conditions, a flag is set in device RAM which returns an EOF error to the user when he next attempts to read from an empty RAM buffer (or immediatly in the first EOF case).

LOW LEVEL STRATEGY

Both reading and writing depend on the use of the programmable DAVE interrupt generators. The cassette driver, on entering the low level reading and writing routines, reconfigures the EXOS interrupt jump vector at 038H, having stored the previous contents in device RAM.

1) WRITING

Three parameters are used by the tape writing routine: the location of the channel RAM buffer, the number of characters in the buffer, and a flag to determine whether a header chunk is to be written. Having configured the interrupts and the DAVE interrupt timers, the data are written to tape, LSB to MSB, in 256 byte blocks. CRC values are calculated on the data bits and the two byte CRC is sent after each block, low byte first.

The CRC is calculated using the following algorithm:

1) XOR the new bit with b15 of the CRC value
2) Copy the new value of b15 into the CF
3) If CF = 1 then XOR the CRC with 0810H
4) Rotate the CRC one bit to the left, putting the CF into b0.

Having sent the data stream a footer of 5 leader cycles is tacked on the end.

2) READING

The reading routine revolves around two subroutines that start DAVE interrupt timers when transitions in the squared cassette output are detected in b7 of port B6. Both of these routines detect transitions relative to a reference level, the one looking for transitions to the reference state, the other looking for transitions away from the reference. In both cases,

when the required transition is found, the DAVE timer is enabled. Computations are done 'underneath' the delay, ending in a HALT instruction. When the DAVE interrupt occurs one of the polling loops is entered, timing the delay until the next appropriate transition, then setting up the next interrupt delay when the transition is detected.

Even though data is written to the cassette with a 50/50 duty cycle, there can be no guarantees that the duty cycle will be preserved on playback. Therefore all computations are done on full cycle timings. Even so, there can be local problems (such as the one described above for the synchronisation pulse) that affect the full cycle timing for a few cycles. The ratio of logical 1 to logical 0 cycle period is not great enough to cause such problems, so data reliablility remains high.

On entry the chunk reading subroutine goes through a process of noise exclusion by looking for 50 cycles of approximately the same frequency, which it then assumes to be the leader tone. Having found the leader (which can be assumed to have a constant duty cycle), a value is calibrated for the DAVE chip corresponding to the half-cycle time plus 25%. This value remains unchanged through the rest of the routine.

The routine then determines the time from the occurrence of the DAVE interrupt to the next transition. This value is the leader cycle reference relative to which the logical 1 and 0 times will be compared. This time is measured and averaged 256 times to ensure consistency.

Having done its calibration from the leader, the synchronisation pulse is detected by comparing cycle times until a very long cycle is found. (In this context 'very long' means more than 25% longer than the leader cycle period.) The phase of the incoming signal is determined by looking at the level of the first half-cycle of the synchronisation pulse and the phase references readjusted accordingly.

The data stream is then read by repeated applications of the same routine: interrupt delay started at leading edge; while waiting for the interrupt to occur compare the last interrupt-to-leading-edge time with that determined from the leader; if the observed time is greater then the bit is a 0, otherwise it is a 1; perform necessary byte manipulations and bufferings; HALT in expectation of the DAVE interrupt; when the interrupt occurs poll to the next leading edge and continue.

(c) 1984 Intelligent Software Ltd. (cge 10-9-84)

EXOS CALLS

1) OPEN CHANNEL

The cassette driver is configured to allow one read channel and one write channel to be open simultaneously. If the user attempts to OPEN a second read channel the driver will return with a .2NDCH error. If there is not enough RAM to hold the 4K buffer a .NORAM error is returned. Finally, if the searching routine is interrupted by the STOP key, a .STOP error will be returned.

If any of these errors occurs, the cassette device will believe that the channel has not been opened succesfully and will allow another read channel to be OPENed. However, the .STOP error condition will leave the 4K channel RAM buffer allocated to the cassette device. It is the user's responsibility to decide whether to de-allocate that RAM by CLOSEing the channel, or to proceed in some other way.

2) CREATE CHANNEL

This function responds in the same way as the OPEN function with the addition that if an OPENed file has had a non-zero protection flag, the CREATE routine returns with a .NOFN error. This condition is like the .STOP error in that the RAM is allocated but the cassette driver does not recognize the channel as being open.

3) CLOSE/DESTROY CHANNEL

CLOSE and DESTROY are identical functions. If a write channel is CLOSE/DESTROYed the remaining buffer contents are written out to tape. CLOSE/DESTROY also clears the status line and reconfigures the status line parameter table to the palette values present when channels were OPEN/CREATEed.

4) CHARACTER/BLOCK READ

Characters are read from the channel RAM buffer. If the buffer is empty and one of the three EOF conditions has been met, a .EOF error is returned to the user. If the EOF condition has not been met, the next tape chunk is read. The operation of the CRC error condition is described below in the section on status line messages.

5) CHARACTER/BLOCK WRITE

Characters are read into the channel RAM buffer. When the buffer size reaches 4K it is automatically written to tape.

N.B. Character read, block read, character write, and block write will only work on the appropriate read or write channel. If a read operation is attempted on a write channel, or vice versa, a .2NDCH error is returned.

6) CHANNEL STATUS

Returns a flag indicating whether the chunk currently in RAM is the final chunk of the file. This function should not be confused with the .EOF error flag returned by the read operations. The latter error flags are returned in case of an attempt to read a character beyond the last character in the file. Channel status reports on the status of the chunk currently residing in RAM.

```
C = 0FFH => End of file chunk
  =   0  => Not EOF
```

7) SET CHANNEL STATUS and SPECIAL FUNCTIONS

Not supported by the cassette driver. Attempts to access these functions will result in a .NOFN error.

OTHER FUNCTIONS

1) REMOTE CONTROL

The two remote control relays on the Enterprise are controlled by code in the cassette driver. EXOS maintains two variables, REM1 and REM2, which can be changed to toggle the remotes.

```
REM1/2 = 0 => OFF
       <>0 => ON
```

In the case of a direct call, the entry conditions are:

```
A = 0 => REM1
  <>0 => REM2

D = FF => ON
  <>FF => OFF
```

When reading or writing from tape, the cassette driver uses the remote control relays to switch the tape machine on and off between chunks. If a single read/write channel is open to the cassette driver, both relays are switched simultaneously. If a read and a write channel are both open, as would be the case for copying a cassette, REM1 is used for READ channel and REM2 is used for the write channel, allowing separate control of the two machines.

2) LEVEL INDICATION

A display on the status line reflects the status of the input signal level by reading b7 of port B7. If the level is higher than the optimum, a red square is displayed, otherwise a green square is displayed. The optimum tape machine output level can be found by adjusting until the display flashes rapidly between red and green.

The level indicator only functions properly when the data part of the chunk is being read, where the interrupt timing is not so critical. During the leader, a constant red or green square is displayed, from which coarse adjustments can be made.

3) MESSAGES

Various messages are displayed on the status line during operation of the cassette device.

When a file is saved the legend 'SAVING' appears with the filename (truncated to 15 characters).

On reading a file from tape, the legend 'SEARCHING' appears while the driver looks for a header chunk. When a header is found, the legend 'LOADING <filename>' appears if the correct header has been found, or 'FOUND <filename>' if the correct file header has yet to be found. In the latter case the searching routine continues until the correct file header is found.

(c) 1984 Intelligent Software Ltd. (cge 10-9-84)

If a CRC error is detected in the course of reading a chunk, a flag is set in device RAM. When the user attempts to read the from the channel buffer, a .CCRC error is returned and the message 'LOADING ERROR' appears on the status line.

The status line is blanked when channels are CLOSE/DESTROYed.

4) STOP KEY

The STOP key is tested on every input level test and every output interrupt. A .STOP error is returned and the software interrupt request flag set.

5) WRITING SPEED

The output data rate can be adjusted by setting the EXOS variable SP_TAPE.

```
SP_TAPE  = 0 => 2400 'baud' (default)
         <>0 => 1000 'baud'
```

1. Overview of Features

The keyboard device has an interrupt routine which scans the keyboard matrix every video frame (20ms). This detects key presses and buffers a single character.

It supports programming of the eight function keys. Each one can be programmed seperately for shifted and unshifted use giving effectively sixteen function keys. If the string programmed into any one of these function keys is of zero length then instead of returning characters, this function key will cause a software interrupt when it is pressed.

The eight function keys also each return a specific code if used with the CTRL or ALT keys, giving effectively another 16 functions.

The keyboard driver treats the joystick as if it were four cursor keys and provides diagonal movement by alternating two cursor codes. Autorepeat is supported on all keys. Both the delay until autorepeat begins, and the autorepeat speed can be altered.

The keyboard provides audible feedback by triggering the sound device to produce a click whenever a key is pressed. This can be disabled by the user.

There is no editing of lines since input is purely character, not line orientated. Any editing functions must be performed by the calling program or device.

2. Character Input

All input is done using the EXOS read character and read block calls. Read block is supported for compatibility with other devices although it is not very likely to be used. The keyboard is not an output device and so will not accept write function calls.

With the exception of the function keys which can be programmed with arbitrary strings, each key produces a single ASCII code. Many keys will produce different codes when used in conjunction with the CTRL, SHIFT or ALT keys.

 Copyright (C) 1984 Intelligent Software Limited

2.1 Lock Modes

The keyboard is always in one of four modes; Normal, shift-lock, caps-lock or alt-lock. The default mode is normal. The mode can be changed by various key combinations:

```
CTRL LOCK  -  Enters Caps-lock mode.
SHIFT LOCK -  Enters Shift-lock mode.
ALT LOCK   -  Enters Alt-lock mode.
LOCK       -  Returns to Normal mode.
```

When the keyboard is in any of the lock modes then it behaves as if the appropriate SHIFT, CTRL or ALT key was held down permanently. If the appropriate key actually is held down during a lock mode then it temporarily counteracts the effect of the lock. Thus for example in SHIFT LOCK mode the action of the SHIFT key is effectively reversed. In this example if the CTRL key is used while in SHIFT LOCK mode it will behave as if it was in NORMAL mode. This applies to all other combinations.

The current lock mode is indicated on the status line the first six characters of which are reserved for the keyboard. It displays the word SHIFT, CAPS or ALT as appropriate and is blank for normal mode.

There is an EXOS variable LOCK_KEY which is always set to the current lock status according to the following codes:

```
0 - Un-locked
1 - CAPS lock
2 - SHIFT lock
8 - ALT lock
```

If this EXOS variable is changed by the user then the next keyboard interrupt will update the lock mode appropriately. Any values other than the above which are put into the variable will be changed to one of the four allowed values.

 Copyright (C) 1984 Intelligent Software Limited

2.2 Key Codes

These are the ASCII codes returned by each key both normally, and with SHIFT, CTRL and ALT. (All values are in hexadecimal.)

Key	NORMAL	SHIFT	CONTROL	ALT
1	31	21	31	31
2	32	22	32	32
3	33	23	33	33
4	34	24	34	34
5	35	25	35	35
6	36	26	36	36
7	37	27	37	37
8	38	28	38	38
9	39	29	39	39
0	30	5F	1F	9F
-	2D	3D	2D	2D
^	5E	7E	1E	9E
@	40	60	00	80
[	5B	7B	1B	9B
;	3B	2B	3B	3B
:	3A	2A	3A	3A
]	5D	7D	1D	9D
\	5C	7C	1C	9C
,	2C	3C	2C	2C
.	2E	3E	2E	2E
/	2F	3F	2F	2F
space	20	20	20	20
A	61	41	01	81
B	62	42	02	82
C	63	43	03	83
D	64	44	04	84
E	65	45	05	85
F	66	46	06	86
G	67	47	07	87
H	68	48	08	88
I	69	49	09	89
J	6A	4A	0A	8A
K	6B	4B	0B	8B
L	6C	4C	0C	8C
M	6D	4D	0D	8D
N	6E	4E	0E	8E
O	6F	4F	0F	8F
P	70	50	10	90
Q	71	51	11	91
R	72	52	12	92
S	73	53	13	93
T	74	54	14	94
U	75	55	15	95
V	76	56	16	96
W	77	57	17	97

 Copyright (C) 1984 Intelligent Software Limited

X	78	58	18	98
Y	79	59	19	99
Z	7A	5A	1A	9A
ENTER	0D	0D	0D	0D
ESC	1B	1B	1B	1B
TAB	09	09	09	09
DEL	A0	A1	A2	A3
ERASE	A4	A5	A6	A7
INS	A8	A9	AA	AB
STOP	03	03	03	03
joy up	B0	B1	B2	B3
joy down	B4	B5	B6	B7
joy left	B8	B9	BA	BB
joy right	BC	BD	BE	BF
Function 1	.	.	F0	F8
Function 2	.	.	F1	F9
Function 3	.	.	F2	FA
Function 4	.	.	F3	FB
Function 5	.	.	F4	FC
Function 6	.	.	F5	FD
Function 7	.	.	F6	FE
Function 8	.	.	F7	FF

3. Special Features

3.1 Keyclick control

When the interrupt routine detects a key press it calls a routine called KEYCLICK in the sound driver which produces an audible click. This routine is only called if the EXOS variable KEY_CLICK is zero. Thus setting this variable to a non-zero value will disable key click.

3.2 Autorepeat Control

Autorepeat is controlled by two EXOS variables. DELAY_KEY is the delay until autorepeat starts in 1/50 seconds. If it is zero then autorepeat is disabled.

The other EXOS variable is RATE_KEY which is the delay between each repetition of the key, also in 1/50 seconds.

 Copyright (C) 1984 Intelligent Software Limited

3.3 Function key control

There are sixteen logical programmable function keys numbered 0 to 15. Keys 0 to 7 refer to the basic function keys, 8 to 15 are the shifted versions. Any one of these may be programmed with a string of characters (which may include control codes etc.) using a special function call. The default string for all keys is a null string.

```
Parameters:   B = 8  (Special function code)
              C = Function key number (0..15)
             DE = Pointer to string (Length byte first)
Returns:      A = Status
```

The maximum length for each programmed string is 23 characters excluding the length byte. An error will be returned if the string is too long.

If the programmed string is of zero length (the null string) then this function key will cause a software interrupt when it is pressed. The software interrupt code will be 10h for function key 0, up to 1Fh for function key 15.

3.4 Stop Key Control

There is an EXOS variable called STOP_IRQ which controls the action of the STOP key. If it is non-zero then the stop key simply returns the ASCII Ctrl-C code (03h) in the same way as all other keys. If STOP_IRQ is zero then instead of this a software interrupt is caused, with software interrupt code 20h.

3.5 Hold Key Control

When the hold key is pressed the keyboard driver will hang up in its interrupt routine until the key is pressed again. This will thus freeze any listing etc. which is being produced. When it hangs up it will call the routine SILENCE in the sound driver to ensure that the DAVE chip is not left in a noisy state.

When the hold key is pressed it displays the message "HOLD" in place of the current lock mode on the status line. This message will be replaced by the correct lock mode message (which is blank for normal mode) when interrupts start again. If the STOP key is pressed while in HOLD mode then this will force an exit from HOLD mode, and will then respond to the STOP key in the normal way.

 Copyright (C) 1984 Intelligent Software Limited

3.6 Normal Key Software Interrupts

When a normal key is pressed the character code for it is simply put in the buffer. However if the EXOS variable KEY_IRQ is non-zero then as well as returning the character code, a software interrupt will be caused with software interrupt code 21h.

3.7 Direct Joystick Reading

A special function call is provided which will directly read the joystick on the main keyboard, or one of the two external joysticks on the control ports. The parameters for this are:

```
Parameters:    B = 9  (Special function call)
               C = 0  (internal joystick)
                 = 1  (external joystick 1)
                 = 2  (external joystick 2)
Returns:       A = Status
               C = b0  -  Set if RIGHT presses
                   b1  -  Set if LEFT pressed
                   b2  -  Set if DOWN pressed
                   b3  -  Set if UP pressed
                   b4  -  Set if FIRE pressed
                b5..b6 -  Clear
```

Note that for the internal joystick the "fire" button is in fact the space bar.

 Copyright (C) 1984 Intelligent Software Limited

CONTENTS

1. Introduction

2. Hardware

3. Serial Device

3.1 Low-level Operation

3.2 Using the Serial Device
3.2.1 Read and Write Instructions
3.2.2 Baud Rate Selection
3.2.3 Word Format Selection

4. Network Device

4.1 Data Transaction Protocols
4.1.1 Broadcast Data
4.1.2 Directed Data
4.1.3 Accepting Data Blocks

4.2 Low-level Network Operation
4.2.1 Hardware Connections
4.2.2 Obtaining Control of the Network
4.2.3 Attracting Attention
4.2.4 Destination Machine Synchronisation
4.2.5 Sending the Data Bytes
4.2.6 Data Transmission Format

4.3 Using the Network
4.3.1 Address Determination
4.3.2 Opening Channels
4.3.3 Interrupts
4.3.4 Buffers
4.3.5 Read and Write Instructions

 Copyright (C) 1984 Intelligent Software Limited

1. Introduction

The serial interface and network driver are provided as two separate EXOS device drivers as far as the user is concerned, and are referred to internally as the SERIAL: and NET: devices respectively. However since they share the same hardware only one of the devices can be supported at a time. So if the user wishes to open a channel to the serial device he must first close any channels open to the network, and vice versa.

Only one serial channel may exist at a time, while any number of channels may be opened to the network device provided there is sufficient RAM for the 512 bytes of buffer for each channel. All channels opened to the network or the serial device support both input and output.

2. Hardware

The serial/network hardware consists of two outputs and two inputs. The outputs are the bottom two bits of an output port (port 0B7h) as below:

b0 - DATA OUT
b1 - STATUS OUT

Each of these outputs is connected to an open collector inverter with a pullup resistor. Thus for example setting bit 1 of this port will pull the STATUS OUT line Low. Clearing this bit will allow the STATUS OUT line to float high unless any other connected machine is pulling it Low.

The two inputs are available as bits of a general purpose input port. The other bits are used for cassette inputs and various other things. The port number is 0B6h and the relevant bits are:

b4 - DATA IN
b5 - STATUS IN.

For use as a serial interface these inputs and outputs are used separately to provide half duplex communication (data can be sent either way but only one way at a time). For use as a network the STATUS IN and STATUS OUT lines are joined together, as are DATA OUT and DATA IN.

NOTE:
Throughout this document the signal levels referred to are the actual levels on the external lines. On the Enterprise, both inputs and outputs are inverted between these external lines and the appropriate bits of the ports. Thus for the software a High level on a line corresponds to a zero, while a Low level is interpreted as a one - to 'pull a line Low' a 1 is written to the appropriate bit in the output port.

 Copyright (C) 1984 Intelligent Software Limited

3. Serial Device

3.1 Low-level Operation

The serial device uses five wires - DATA IN, DATA OUT, STATUS IN, STATUS OUT and REF (used as an offset signal reference instead of the 0 Volt GROUND line). This allows independent handshaking on input and output. The device supports both read and write EXOS calls. When it is called with a "read character" function call it sets the STATUS OUT line High (it is normally held Low). This signals the sending device that it can send a character. It then monitors the DATA IN line (which is also normally Low) until it goes High signifying a start bit. The bits of the character can then be read in, possibly with a parity bit if that is selected (see later).

The network driver handles a small buffer for incoming characters. After one character has been read, the DATA line is monitored for a short time to see if the sending machine has any more to transmit. If another character is sent, it will be read in and buffered; up to sixteen characters may be read and stored if they are immediately available. Once the buffer is filled, or if no further start bit is detected within the timeout period, the STATUS OUT line is pulled Low again preparatory to returning to the user program. However, some devices are rather slow in responding to handshaking lines, so the DATA IN line is checked for a short time afterwards to ensure that no more characters are being sent. Any spurious characters which are received can be buffered (there is an additional eight-byte overflow in case the main buffer is full). Stored characters are supplied to the user one at a time when "read character" is called, so this buffering is transparent.

"Write character" is simpler than read character since there is no problem of the other end of the connection misbehaving (ie sending extra characters). To send a character the serial driver monitors the STATUS IN line until it is High (which it may be already if the receiver is ready). Then the DATA OUT line is moved from its quiescent Low level to High for the start bit. The bits of the data are then sent, followed by a parity bit (if parity is selected) and then the required number of stop bits (DATA OUT Low).

 Copyright (C) 1984 Intelligent Software Limited

3.2 Use of the Serial Device

3.2.1 Read and Write Instructions

The serial device supports the normal EXOS read and write instructions for single characters or blocks. Data is not interpreted in any way, so machine code can be sent just as easily as ASCII text.

3.2.2 Baud Rate Selection

The EXOS variable BAUD_SER governs the baud rate, which applies both to input and output. Before opening the serial channel the user should set it to the appropriate value for the required rate, according to the following code:

```
 0 => 50 baud  1=> 75 baud 2 => 110 baud    3 => 134.5 baud
 4 => 150 baud 5 => 200 baud 6 => 300 baud    7 => 600 baud
 8 => 1200 baud 9 => 1800 baud 10 => 2400 baud 11 => 3600 baud
12 => 4800 baud 13 => 7200 baud 14 => 9600 baud 15 => 9600 baud
```

The default setting is 8 (1200 baud). Numbers greater than 15 are reduced to their lower four bits before interpretation.

3.2.3 Word Format Selection

The word format can be set by changing an EXOS variable. The options which can be set are:

number of data bits (7 or 8)
parity (off, odd, even)
number of stop bits (1 or 2).

These are specified by a single byte, of which the bits are interpreted as follows:

```
b0 - Number of data bits:  Clear => 8 bits
                           Set   => 7 bits
b1 - Parity enable.  Clear for no parity.
b2 - Parity select (Ignored if b2 is clear).
                     Clear =>  even parity
                      Set   =>  odd parity
b3 - Number of stop bits: Clear => two stop bits
                          Set   => one stop bit
b4..b7 - Not used, must be zero.
```

To set up the format, place the required value in the variable FORM_SER before opening the serial channel. The default value is zero, ie. 8 bits, no parity, two stop bits.

 Copyright (C) 1984 Intelligent Software Limited

Note that the data bits are sent least significant first, and if 7 data bits are selected then bits 0 to 6 of the byte will be sent and bit 7 will be ignored. On reception bit 7 will be cleared.

4. Network Device

4.1 Data Transaction Protocols

The network can be used in two modes: directed and broadcast. In directed mode one machine sends data to one other machine and all other machines ignore it. In broadcast mode one machine sends data to all the other machines at once. Each machine on the network is identified with a unique address in the range 1 to 32 (0 is reserved for broadcast mode).

Each block that is sent is preceded by a header which includes a synchronisation pattern, the source and destination addresses and the count of the number of data bytes in the block.

4.1.1 Broadcast Protocol

The header of a block which is being broadcast is sent with a destination address of zero. The block will be received by all machines which are listening and there is no method of determining whether it was read correctly or not.

This lack of handshaking introduces problems if some of the machines had interrupts disabled at the time of the broadcast, and thus arrive at the network interrupt handler once transmission has already started, or even after it has finished. In order to avoid possible confusion brought on by receiving only part of a block, the destination machines check for the synchronisation pattern which is supplied in the block header. The header consists of a repeating four-byte block, which continues for long enough to give the receiver time to be ready in most problem cases while obviously being kept reasonably short in order to save time.

4.1.2 Directed Data Protocol

As Directed Data is destined for just one machine, the sending machine can wait for acknowledgement in order to ensure that the destination is listening, and to confirm that the data block is received without error.

 Copyright (C) 1984 Intelligent Software Limited

When a computer reads a block header which has its own number as the destination address, and is prepared to accept the block (see later), then it sends back an acknowledgement to the source machine. This is simply a signal on the DATA line to show that it is ready to receive; the absence of the signal within a given time (of the order of 1ms) is interpreted by the source machine as an error.

Once the header has been acknowledged, the remainder of the block is sent. After the complete data block has been received, the destination machine must carry out the checksum calculation and either confirm the data by sending another acknowledgement signal, or reject it by not responding.

Error Response

When a destination machine finds an error returns immediately from the network interrupt routine without setting any interrupt flags (see later). The source machine, when it finds no acknowledgement signal after sending a header or a complete block, retries as follows:

1. Release network *

2. Loop for long delay - say one second, plus short random addition to avoid the possibility of repeated collisions.

3. Try to gain control of network and send again

* Note that it is extremely important that under no circumstances should an error occur which causes a machine to hang irretrievably while it is in control of the network, as this would also hang the network itself and thus any other machines which are trying to use it. Any time a machine is waiting indefinitely for a signal on the network lines, pressing the STOP key will regain user control.

4.1.3 Accepting Data Blocks

A machine is only prepared to receive a block from another computer on the network if there is a suitable channel open to the network driver. When a channel is opened by the user a remote address number is given as the unit number. If this is zero then blocks will be accepted from anywhere; if it is non-zero then only blocks from that specific network address will be accepted. Any number of these non-zero address channels may be opened. If a non-specific channel is also opened, it will only receive data from source machines which are not explicitly served by an individual channel.

 Copyright (C) 1984 Intelligent Software Limited

4.2 Low-level Network Operation

4.2.1 Hardware Connections

The network driver is rather more complicated than the serial interface driver because it has to include protocols to avoid collisions. It uses the same hardware as the serial driver. All machines on the network are joined by three wires: GROUND, DATA and STATUS. On each machine the DATA line is connected to both DATA OUT and DATA IN, and the STATUS line is connected to both STATUS OUT and STATUS IN. This allows the machine to pull either line Low (the outputs are open collector) and also to monitor the level on each line.

The STATUS IN line is also connected to the external interrupt input so the status line going Low can trigger an interrupt. This is how the machine can respond to data sent down the network asynchronously.

4.2.2 Obtaining Control of the Network

When a machine wishes to send data to another machine, or to broadcast it, it must first get control of the network. Only one machine can be in control of the network at a time and the protocol used for obtaining control is designed to ensure that collisions (two machines taking control at the same time) do not occur. The penalty for this is a slight loss of speed and a priority ordering of machines on the network.

There is a timing constant C defined, which is chosen to correspond to a period of about one millisecond. When a machine wants control of the network it must follow this procedure:

1. Both STATUS OUT and DATA OUT should be left High whenever this machine does not have control of the network.

2. If the STATUS line is High go to step 4

3. Wait until the STATUS line is High and remains High for a period of RND*C where RND is a random number in the range 1 to 16. If the STATUS line does not remain High for the required time then repeat this step with a new random number.

4. Pull the STATUS line Low.

 Copyright (C) 1984 Intelligent Software Limited

5. Wait for a period of C*ADDR where ADDR is the address of this machine on the network, constantly monitoring the DATA line. If DATA goes Low for any time during this interval then release the STATUS line (set it High again) and return to step 3.

5. Pull the DATA line Low and then proceed to send the block header (see later).

Throughout a network transmission, interrupts are disabled because of the timing considerations.

4.2.3 Attracting Attention - Protocol

Once a machine has secured itself control of the network, it can start the transmission of the data block.

First of all, it has to attract the attention of its intended audience. This is done by repeatedly sending a header consisting of a synchronisation byte, the destination and source addresses of this block and a type byte. The bytes are sent in the order sync/dest/source/type/sync.... and are terminated with the ones complement of the dest byte in the position of the next sync byte. The format of these bytes is as follows:

Machine address bytes

The destination address is the number of the machine on the network which this data block is being sent to; if it is zero then it indicates that it is a broadcast block, which all machines should receive. The source address is the number of the machine which is sending the block.

Byte format:

b0..b5 - machine address: 1 to 32, or 0 for broadcast data
(0 only valid as a destination address)

b6 - source/destination selection: 0 => addr=source
1 => addr=dest

b7 - complement of bit 6

Synchronisation byte 00000000B

The synchronisation byte is required because fast interrupt response is not guaranteed; the STATUS line can be brought Low at any time, perhaps while interrupts are disabled on the required destination machine. The synchronisation method is detailed below.

 Copyright (C) 1984 Intelligent Software Limited

Type byte 1xxxxxx1B

This is for future expansion, but must ALWAYS have bit 7 set in order to avoid possible confusion with the synchronisation byte. At present the device handler also assumes that bit 0 is set to 1, and will return prematurely from servicing a network interrupt if this is not true.

4.2.4 Destination Machine Synchronisation

As stated above, there is some problem in synchronising the destination machine, which stems from the uncertainty of catching the machine while its external interrupt is enabled. This is now explained in detail:

If external interrupts are disabled, a latch will record the interrupt transition. The interrupt will then take place as soon as the input is re-enabled, and the destination machine will immediately start reading the DATA line.

The source machine is not expected to wait for all machines to confirm that they are listening, but begins to output its attention-grabbing sequence as soon as it is ready - in the hope that all relevant machines will catch on before that sequence finishes. The protocol must therefore cope with a destination machine starting to read the DATA line at any time during this period.

RS232 character framing is by means of DATA line levels, so any transition from 1 to 0 can be interpreted as a start bit. This means that a computer which starts to listen to the DATA line at an arbitrary moment may pick up what it thinks is a start bit when in fact it has simply found a changing level in the middle of a character. The purpose of the synchronisation byte in the address sequence is to make sure that the character framing is correctly aligned as quickly as possible.

The method chosen for achieving this is to send a synchronisation character made up of all zero bits, and to tell the destination machine to check every start bit transition until it finds one which is followed by eight zero bits and a stop bit. The next 'start bit' is then guaranteed to be the true start of a character.

The above specifications require that any aspiring destination machine should follow this algorithm when it enters the interrupt routine:

1. If STATUS line is High, return. (This would occur if the response had been so slow that the transfer was all over, or had been abandoned due to lack of interest)

 Copyright (C) 1984 Intelligent Software Limited

2. Wait until the DATA line is Low

3. Wait for DATA line to go High (possible start bit)

4. Delay for required time to synchronise to middle of bits

5. Continue to read DATA at one-bit intervals until a Low bit is found

6. If any number other than nine High bits were found (including the start bit) go to 4

7. Read next two bytes, and store in DEST and SOURCE respectively provided that bits 6 and 7 are as expected for each address. Else return.

8. Read next byte, and store in TYPE provided that bits 7 and 0 are set. Else return.

9. Read sync byte. If not zero go to 14

10. Read destination address. If not same as DEST, or if same but invalid (ie. not this machine, nor broadcast) then return.

11. Read source address. If not same as SOURCE, or if same but invalid (ie. no channel open to it, or read buffer not available) then return.

12. Read type. If not same as TYPE then return.

13. Go to 10

14. If latest byte is not ones complement of DEST then return.

15. Read next two bytes. If they are 1's complement of each other then the latter is the byte count for the following data block. Else return.

16. If the destination address was non-zero (ie. transaction is Directed rather than Broadcast), pull DATA Low for two C time periods as acknowledgement then release it.

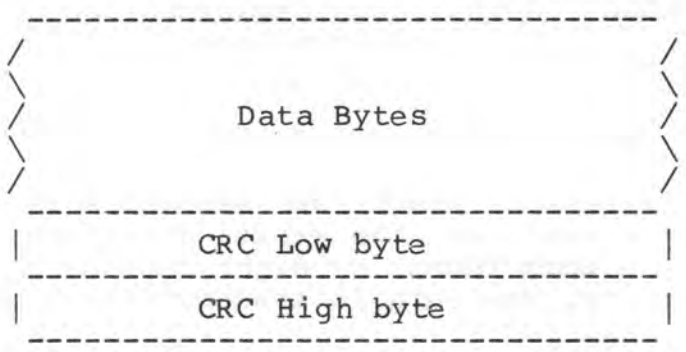
NOTE: Whenever a byte is read, STATUS should be checked for still being held Low by the source machine. If it goes High at any point, the link must be assumed broken immediately.

 Copyright (C) 1984 Intelligent Software Limited

4.2.5 Sending the Data Bytes

When the destination machine pulls the DATA line Low, and leaves it there for at least one C time period, that is the signal to the source machine to start sending the data. When DATA is released, the sender pulls it Low again immediately then gets on with sending the first start bit of the data block.

The format of the data block is:



The byte count which was received in the header sequence is a true count of the number of data bytes in the block. A value of zero means 256 bytes.

The two CRC bytes are a cyclic redundancy check calculated on all data bytes in the block (ie. excluding the addresses, type, serial number and byte count). The CRC is evaluated as a 16-bit value, which is is re-calculated when the data block is received. If the CRC calculated on reception is not the same as the value received then the block is assumed to be garbage.

The algorithm for calculating the CRC is the same as that used for the cassette driver but is explained here as well. For purposes of the CRC calculation the data block is regarded as a bit stream. Each time a data bit (not a start or stop bit) is sent, a 16 bit CRC register is updated. This will have to be done in the serialisation routine which will be shared with the serial interface driver so it will be calculated on serial data sent or received as well even although this is not required.

 Copyright (C) 1984 Intelligent Software Limited

The CRC register is a 16 bit register which is initialised to zero at the start of the data block and then when each bit is sent or received the following operations are performed:

1. XOR the new bit with the most significant bit of the CRC register and set the carry to this value.

2. If the carry is set then XOR the register with 0810h.

3. Rotate the register one bit left, rotating the carry into the least significant bit.

4.2.6 Data Transmission Format

All network blocks are sent as a series of bytes in the same format as the serial interface driver sends them, using a word format of eight data bits, two stop bits and no parity. The data is transmitted at 9600 baud.

4.3 Using the Network

4.3.1 Address Determination

The above protocols require that each machine on the network has a unique address, so the network handler refuses to open channels onto the network unless the computer has been given its address.

The address is held in the EXOS variable ADDR_NET, and must be set by the user before attempting any network operations. At power-on or cold reset it is set to zero, which is invalid as a network machine number.

4.3.2 Opening Channels to the Network

As mentioned before, each channel which is opened to the network must specify a remote machine number for which the channel is reserved, although an address of zero will allow data blocks to be received from any machine. This number is given in the call to the EXOS channel opening function as a unit number, which is enclosed in parentheses in the name string. For example, to open a network channel to computer 20 from a machine code program the DE register pair should point to the string:

```
DB   8,"NET(20):"          (see EXOS Kernel Spec ET10)
```

Note the clear distinction between the channel number and the network address of the machine which that channel serves. The two are completely independent; it is up to the user to keep track of which channel serves which machine.

 Copyright (C) 1984 Intelligent Software Limited

4.3.3 Interrupts

When the first channel is opened to the network, the external interrupts are enabled and will remain in operation until the last channel is closed. While external interrupts are enabled any transition from High to Low on the STATUS line will cause the network interrupt service routine to be called. A particular consequence of this is that opening a serial channel (for which the quiescent line levels are Low) on a machine which is connected to a network will cause any machines using the network to be interrupted, and to hang in anticipation of a data block.

4.3.4 Buffers

Each channel opened to the network sets up two buffers of 256 bytes length; a Receive buffer and a Transmit buffer.

The Receive buffer is filled with data which arrives in blocks on the network, and can hold just one block at a time. The user then reads from the buffer until it is empty, when it will be able to accept further blocks from remote machines. The Receive buffers of each channel work independently, so blocks can be received from any machine provided that its specific Receive buffer is free.

The Transmit buffer is filled by the user, and sent as complete blocks onto the network. It can be forced onto the network at any time by using a 'Flush' command, or will be sent automatically when the 256th byte is written to it. The Transmit buffer will also be flushed automatically if the channel is closed, thus enabling files of any length (from 1 byte upwards) to be sent to another machine while the buffering remains transparent.

4.3.5 Read and Write Instructions

The network device supports the usual EXOS read and write character function calls and the block read and write calls.

The reception of blocks from remote machines is done by an interrupt service routine, which is invoked when the STATUS line goes Low. The machine will remain in its interrupt routine watching for the DATA line to go Low and then read in the header. If the header is read succesfully, and the block is one which this machine wants to receive, then the data is read into the Receive buffer for the channel serving the given remote machine. Each channel can buffer one block at a time in its Receive buffer.

 Copyright (C) 1984 Intelligent Software Limited

The network driver can cause a software interrupt when a block is successfully received from the network. An EXOS variable called NET_IRQ is provided to switch this function on and off. If NET_IRQ is zero then software interrupts are enabled, and the occurence of a successful network interrupt will cause the value ?NET to be placed in the variable FLAG_SOFT_IRQ.

The EXOS variable CHAN_NET is used to pass to the user the channel number from which buffered data can be read. CHAN_NET is only updated when a character is read from the channel which it specifies, or if that channel is closed. It is then changed to the channel number of the lowest numbered machine which has caused an interrupt. So if machines 2 and 10 send a block each while data sent by machine 5 is still waiting to be read, then when CHAN_NET is changed it will point to the channel for machine 2 rather than 10 whichever interrupted first. Machine 10 will be serviced later, as soon as it becomes the lowest number awaiting attention. This allows specific machines to be given priority - a teacher operating from machine 1 will almost almost have his message received in preference to a message from another machine, while broadcast messages are given the highest priority of all. If no data is available CHAN_NET holds the value 255.

The user requests bytes by calling the EXOS RDCH or RDBLK routines, and is given bytes from the Receive buffer if they are available. If not, an end of file condition is returned - this is to prevent the machine from hanging while awaiting a block which may never get sent. The Read Channel Status facility is supported, so the user can check for end of file before trying to read a character.

Writing to the network is carried out as a series of WRCH instructions. If an error is encountered when the buffer is to be sent, the network handler will continuously retry at intervals of about one second. This can be stopped by pressing the STOP key, which will also cause the buffer contents to be cleared before the handler returns. Note that this error condition can only be detected for directed data blocks; broadcast data does not require acknowledgement and therefore cannot check that the data was received properly.

 Copyright (C) 1984 Intelligent Software Limited

0. INTRODUCTION

This is not a formal specification of the editor. It is just an outline of the editor's functions, concentrating on the interface between the editor and an applications program (eg. Lisp or BASIC). It is intended to contain sufficient information to allow an applications program to be fully interfaced to the editor. It should be read in conjunction with the specifications of the EXOS kernel (ET10/8), video driver (ET11/8) and keyboard driver (ET13/4).

1. Outline of Editor Operation

The editor is an EXOS device which the applications program can read and write to. It can support any number of channels open to it at a time, each channel corresponds to a seperate "document" which is being edited. The word document here is used loosely since for example the editor channel used by BASIC is referred to as a document although it is actually a collection of BASIC commands, program listings error messages etc.

Each editor channel has video channel and a keyboard channel associated with it. Different editor channels can share the same keyboard channel (which is useful as the keyboard driver only allows one channel to be open to it), but must have seperate video channels.

Each editor channel also has an area of channel RAM which it uses for a text buffer. This buffer can be any size from a few hundred bytes to just under 16k and will typically be a few kilobytes. Text can be entered into the editor's buffer either from the applications program or from the keyboard. The editor writes characters to the video channel in such a way that it is kept updated to form a "window" onto the text buffer, although the video channel has its own copy of the text it is displaying.

2. Opening Channels

Before opening an editor channel, three EXOS variables must be set up. These are:

VID_EDIT - Channel number of video page.

KEY_EDIT - Channel number of keyboard channel.

BUF_EDIT - Size of editor buffer in units of 256 bytes.

 Copyright (C) 1984 Intelligent Software Limited

The video and keyboard channels must be opened before opening the editor channel. The video page must be a text mode and must be at least 2 rows by 4 characters. The editor determines the size of the video page in its open channel routine and complains if it is too small. Note that the editor does not display the video page on the screen. It is up to the applications program to do a special function call to the video page to display it if this is what is wanted (see ET11/8).

The actual size of the editor's text buffer is:

256*BUF_EDIT + n

where 'n' is between zero and 255 and depends on the width of the video page (space reserved for the ruler line) and the exact size of the editor's variable area. The valid range for BUF_EDIT is thus zero to 254.

The editor will work with any size buffer but it is sensible to ensure that it is at least as big as the video page. The editor stores lines as variable length in its buffer so, since short lines are common, it generally manages to store more than the calculated minimum number of lines.

3. Writing to the Editor

Any printing characters written to the editor will be put into the text buffer at the current cursor position and displayed on the video page. Codes in the range 080h to 09Fh are also interpreted as printing characters and displayed on the video page. The video driver will display these as characters 00h to 01Fh from the font, which are normally un-obtainable because 00h to 01Fh are interpreted as control codes.

The editor will also interpret the following control codes. All codes in the range 00h to 1Fh not mentioned here are ignored.

00h (NUL) - Writes a null to video to check it is still OK.
09h (TAB) - Move to, or insert spaces to next tab stop.
0Ah (LF) - Ignored.
0Dh (CR) - Goes to start of new line (equivalent to CR-LF).
18h (^X) - Set left margin at cursor column.
19h (^Y) - Clear to end of line.
1Ah (^Z) - Clear whole buffer and screen and home cursor.
1Bh (ESC) - Starts escape sequence (see below)

The only escape sequence interpreted by the editor is to position the cursor at arbitrary co-ordinates. This is identical to the video driver escape sequence for this function (see ET11/8).

 Copyright (C) 1984 Intelligent Software Limited

Codes above 0A0h are interpreted exactly as if they had been received from the keyboard. These provide various editing functions and cursor movement, but are really provided for demonstration purposes. See below for the codes and their functions.

4. Reading From the Editor

When the editor receives a read character function call, it examines the EXOS variable "FLG_EDIT". This byte contains a series of flags which control the response of the editor to this read character call. The editor's action will first be described in general terms without reference to the individual flags. The effect of each flag will then be described afterwards.

The assignment of bits in this byte is:

```
MSB  bit-7  -  SEND NOW
     bit-6  -  SEND ALL
     bit-5  -  NO READ
     bit-4  -  NO SOFT
     bit-3  -  NO PROMPT
     bit-2  -  AUTO ERA
     bit-1  -  not used
LSB  bit-0  -  not used
```

4.1 Basic Editor Read Action

With all editor flags zero, when the editor receives a read character call, it does not return a character immediately but records the state of the flags and enters its main editing loop which provides the actual editing facility to the user.

This loop reads a character from the keyboard, responds to it and then loops back for another one. This allows the user to type in text and carry out various editing functions. The details of this are given below in the section on editing. There are two keys which are of importance here - ESCAPE and ENTER.

If ESCAPE (ASCII code 01Bh) is received from the keyboard then this character will immediately be returned to the applications program as the response to the read character call.

 Copyright (C) 1984 Intelligent Software Limited

If ENTER (ASCII code 0Dh) is pressed then this is a command to the editor to begin sending text back to the applications program. The details of how much text is sent are determined by the flags and will be described below. An internal editor flag is set to indicate that it is in the process of sending text, and the first character is returned to the applications program. When the applications program makes another read character call the internal flag is still set, so instead of entering the editing loop, it simply returns the next character immediately. This continues until all the required text has been sent at which time the internal flag is cleared so the next read character call will again enter the editing loop.

Note that the editor flags are sampled once when the first read character call is made and then not again untill all the text has been sent. Changing them in the meantime will therefore have no effect on the read which is in operation. If a write character call is mode while a read is in progress then th einternal flag is cleared so that read will be aborted.

4.2 The Editor Read Flags in Detail

4.2.1 The SEND NOW Flag (bit-7)

This flag is set to do an immediate read from the editor, rather than waiting for the user to press ENTER. The text which is returned is as defined by the other flags.

4.2.2 The SEND ALL Flag (bit-6)

This is the main flag which determines how much text is sent. If it is clear then the paragraph containing the cursor will be sent. If it is set then the whole editor buffer will be sent.

In the first case the applications program will be sent the characters of the paragraph one by one terminated with a CR and then an LF. This CR-LF can be used by the applications program to determine when to stop reading (beware if NO SOFT is clear!). The cursor will be left on the first character of the next paragraph with a new (empty) paragraph being created if that was the last one.

 Copyright (C) 1984 Intelligent Software Limited

If SEND ALL is set then the entire buffer will be sent. This will include CR-LF sequences at least between each paragraph (again see NO SOFT flag below) so this cannot be used to stop the applications program reading. Instead, after the last CR-LF has been sent (the last characters sent will always be CR-LF), the next character read will produce a .EOF (end of file) error. This error will only be received for one character.

4.2.3 The NO READ Flag (bit-5)

If this flag is set then ENTER will not in fact return any characters at all to the applications program. The only way to get back to the applications program is thus to press ESCAPE. Although no characters are returned, the routine which selects which characters to send depending on the flags, is still executed. Thus the cursor is moved in the same way as if the text was returned.

4.2.4 The NO SOFT Flag (bit-4)

This flag controls the sending of soft carriage returns and soft spaces. Soft carriage returns are those which seperate successive lines of a single paragraph (continuation lines if you like). Soft spaces are those spaces which are inserted for justification, and also the spaces before the left margin of a line.

If this flag is clear then soft spaces are returned as normal ASCII spaces (20h) and soft carriage returns are returned as normal CR-LF sequences. If the flag is set then both of these are suppressed and no characters are returned for them.

4.2.5 The NO PROMPT Flag (bit-3)

This flag is normally clear. If it is set then the cursor position when the first read character call was made is remembered. When it comes to returning text, if the cursor has not been moved out of the current paragraph, or to before the remembered prompt position, and if SEND ALL is clear, the the current paragraph will be sent back but starting from the character at the remembered cursor position.

This is used by BASIC when doing an INPUT command. It prints the prompt and the does an editor read with this flag set. When the paragraph is sent to BASIC the prompt will not be returned, just the response to it.

 Copyright (C) 1984 Intelligent Software Limited

2.2.6 The AUTO ERA Flag (bit-2)

This flag is also normally clear. If it is set then if the very first character typed at the keyboard is a printing character, as opposed to an editing function or cursor movement, then the current line will be cleared before responding to the key. This is provided specially for BASIC to allow commands such as RUN to be typed on top of an existing line after editing a program. It may just conceivably be of some use to other applications programs.

2.2.7 Typical Flag Combinations.

The use of these flags can be rather confusing so some typical combinations are given here. It is left as an exercise for the reader to understand why these combinations are used for the various purposes. (x indicates don't care.)

```
000101xx  -  BASIC reading a command.
000110xx  -  BASIC doing an INPUT command.
001xx0xx  -  WORD PROCESSOR in editing mode.
11000xxx  -  WORD PROCESSOR reading a document for
             printing.

????????  -  LISP ?
```

 Copyright (C) 1984 Intelligent Software Limited

3. Editing Functions

This is a list of all the keyboard editing functions together with the codes which trigger them. These codes can also be sent for the applications program.

Ctrl-F1	(0F0h)	-	Reform paragraph
Ctrl-F2	(0F1h)	-	Centre line
Ctrl-F3	(0F2h)	-	Toggle tab
Ctrl-F4	(0F3h)	-	Set left margin
Ctrl-F5	(0F4h)	-	Release margins
Ctrl-F6	(0F5h)	-	Move paragraph up
Ctrl-F7	(0F6h)	-	Colour line
Ctrl-F8	(0F7h)	-	not used
Alt-F1	(0F8h)	-	Justify paragraph
Alt-F2	(0F9h)	-	Clear tabs
Alt-F3	(0FAh)	-	Toggle ruler
Alt-F4	(0FBh)	-	Right margin
Alt-F5	(0FCh)	-	Reset margins and tabs
Alt-F6	(0FDh)	-	Move paragraph down
Alt-F7	(0FEh)	-	Colour paragraph
Alt-F8	(0FFh)	-	not used
INS	(0A8h)	-	Insert a space
Shift-INS	(0A9h)	-	Insert a new line.
Ctrl-INS	(0AAh)	-	Toggle insert/overwrite mode.
DEL	(0A0h)	-	Delete character right
Shift-DEL	(0A1h)	-	Delete line right
Ctrl-DEL	(0A2h)	-	Delete word right
ERA	(0A4h)	-	Erase character left
Shift-ERA	(0A5h)	-	Erase line left
Ctrl-ERA	(0A6h)	-	Erase word left
UP	(0B0h)	-	Cursor up line
Shift-UP	(0B1h)	-	Cursor up page
Ctrl-UP	(0B2h)	-	Cursor up paragraph
DOWN	(0B4h)	-	Cursor down line
Shift-DOWN	(0B5h)	-	Cursor down page
Ctrl-DOWN	(0B6h)	-	Cursor down paragraph
LEFT	(0B8h)	-	Cursor left character
Shift-LEFT	(0B9h)	-	Cursor to start of line
Ctrl-LEFT	(0BAh)	-	Cursor left word
RIGHT	(0BCh)	-	Cursor right character
Shift-RIGHT	(0BDh)	-	Cursor to end of line
Ctrl-RIGHT	(0BEh)	-	Cursor right word

 Copyright (C) 1984 Intelligent Software Limited

PROGRAMMING THE NICK CHIP
****

In contrast to the majority of video display devices which allow the user various modes for the whole display, NICK allows many different modes in the same display frame.

Principal features of NICK are:-

* Mixed mode displays
* User definable characters from fonts of 64, 128 and 256
* 8-bit colour output (256 colours)
* 2,4,16 and 256 colours per line chosen from 256
* Maximum resolution (using interlace) 672 x 512
* Cell based graphics, bitmap and characters
* Characters any hight from 1 to 256 scanlines
* Choice of 256 border colours
* User defined screen width and height
* External colour input (unlimited sprites or TV camera)
* Efficient use of RAM (can work with < 1K of RAM)
* 4 colour-pairs in 84 column text mode
* Special use of bits to increase colour options
* Pointer-based memory mapping, for flexibility and speed

CONTROL REGISTERS
****

Input-output addresses 080H to 08FH are reserved for NICK registers although only 080H to 083H are used at present. These registers are write-only.

080H	/FIXBIAS		
	d7		VC1 output used to kill external colour
	(d6,d5)		(PRIOR1,PRIOR0) external colour priority, working in conjunction with an external 4-bit (16 colour-value) input on EC0-EC3
		=00	EC0-EC3 select corresponding palette colour whenever the display is active and /EXTC is low
		=01	The external colour on EC0-EC3 selects the corresponding palette colour if /EXTC is low and the internal display is generating a palette colour in the range COL8-COL15.
		=10	The external colour on EC0-EC3 selects the corresponding palette colour if /EXTC is low and the internal display is generating a palette colour in the range COL8-COL15 OR the external colour is in the range COL0-COL7 (EC3 low)
		=11	The external colour on EC0-EC3 selects the corresponding palette colour if /EXTC is low and the internal display is generating a palette colour in the range COL8-COL15 OR the external colour is in the ranges COL0-COL3 or COL8-COL11 (EC2 low)
	(d4..d0)		colour bias for palette colours 8-15
081H	/BORDER		
	(d7..d0)		8 bit border colour
082H	/LPL		
	(d7..d0)		(a11 ..a4) of pointer to line parameter table in video RAM. The index into an entry of 16 bytes,(a3..a0), is generated by the hardware.
083H	/LPH		
	d7		/(load line parameter base) normally 1
	d6		/(clock in line parameter base)
	(d3..d0)		(a15..a12) or pointer to the line parameter table in video RAM.

The video display is controlled by values loaded into the video RAM segments (up to 64K at the top of the 4M space) by the Z80. Once this 'line parameter table' has been loaded in and the line parameter base register has been loaded the display requires no more action on the part of the Z80.

The visible display is split up into 'video mode lines'. These 'modelines' are made up of 1 to 256 scanlines. (A scanline is one scan of the electron beam across the CRT and takes about 64 microseconds.)

The following 16 registers are loaded from the line parameter table before each modeline:

SC	scanlines in this modeline (two's complement)
MB	the MODEBYTE (defines video display mode)
LM	left display margin etc.
RM	right margin etc.
LD1L	(a7..a0) of line data pointer LD1
LD1H	(a8..a15) of line data pointer LD1
LD2L	(a7..a0) of line data pointer LD2
LD2H	(a8..a15) of line data pointer LD2
COL0	8 bit value of palette colour £0
COL1	" " £1
COL2	" " £2
COL3	" " £3
COL4	" " £4
COL5	" " £5
COL6	" " £6
COL7	" " £7

BUS ACTIVITY FOR THE VARIOUS MODES
****

The possible video modes are:

VSYNC — no border colour and use margin information to control positioning of the vertical sync pulse. This gives considerable interlace flexibility.

PIXEL — use information pointed to be LD1 as a bit mapped display.

ATTR — use information pointed to by LD2 as a 2-C bitmap display and information pointed to by LD1 as cell-based graphics attributes (ie: to define paper and ink colours in the cell).

CH256 — use information pointed to by LD1 as indices of characters pointed to by LD2. These characters can by any number of lines deep (up to 256).
NB: offsets in the font pointer define which line of the character to start on.

CH128 — As above but assumes a font of 128 characters.

CH64 — As above but assumes a font of 64 characters.

LPIXEL — As for pixel mode but with half the horizontal resolution.

Note that all these modes may be mixed on the same screen and one has the choice of 2-C, 4-C, 16-C and 256-C colour modes for the PIXEL, LPIXEL, CH256, CH128 and CH64 modes. Also note the special interpretation of certain bits of display data described below.

Details of bus use during a memory cycle in the various modes:

PIXEL	/VDC1	/VDC2
Address	LD1 (15....0)	LD1 (15....0)
Data into	BUF1 (7....0)	BUF2 (7....0)

BUF1 and BUF2 are loaded sequentially into the shift register and clocked out MSB first, ie. both are display bytes. The line data pointer LD1 is incremented twice in each memory cycle. Screen data is fetched from memory and the LD1 counter is incremented only in the active part of the display, ie. between the left and right margins of a scanline. The scanline count loaded at the beginning of the PIXEL modeline determines how many scanlines in this scanline.

ATTR	/VDC1	/VDC2
Address	LD1 (15....0)	LD2 (15....0)
Data into	BUF1 (7....0)	BUF2 (7....0)

Cell based (parallel attribute) graphics. LD1 is used as a pointer to the colour array (paper and ink colours) and LD2 points at 2-C pixel data, ie. the display bytes. LD2 is incremented once every memory cycle while the display is active and keeps incrementing up for all scanlines in the modeline. LD1 restarts from the same address for each scanline so attribute data in BUF1 applies to cells which are 8 bits wide and have a depth of the number of scanlines in the modeline.

The character modes involve indirection through the character font. A different font may be defined for each modeline and line by line vertical scrolling is obtained by offsetting the original index.

CH256	/VDC1	/VDC2
Address	LD1 (15....0)	LD2(7....0),BUF1(7....0)
Data into	BUF1 (7....0)	BUF2 (7....0)

LD1 is reloaded at the start of each scanline and acts as a pointer into a section of RAM containing the indices of the characters to be displayed. It is incremented once in each memory cycle. LD2 is a pointer into the character font to be used and is incremented at the start of each scanline (it points to a row of a character in the font.) Thus the font consists of 256 bytes defining the first row of each character and then another 256 bytes for the next row of each character etc. If the characters are 9 lines deep this requires 2304 bytes of character font (256x9). The data in BUF2 is loaded into the shift register.

CH128	/VDC1	/VDC2
Address	LD1(15....0)	LD2 (8....0) BUF1(6....0)
Data into	BUF1(7....0)	BUF2(7....0)

This is basically the same as the 256 character font mode but note that the font for 128 9 line deep characters only requires 1152 bytes of memory.

CH164	/VDC1	/VDC2
Address	LD1(15....0)	LD2(9....0) BUF1(5....0)
Data into	BUF1(7....0)	BUF2(7....0)

This is basically the same as the 256 character font mode but note that the font for 64 9 line deep characters only requires 576 bytes of memory.

LPIXEL	/VDC1	/VDC2
Address	LD1(15....0)	LD1(15....0)
Data into	BUF1(7....0)	BUF2(7....0)

This is much the same as the PIXEL mode except that the LD1 pointer is only incremented once in each memory cycle and the BUF2 data is not used. This gives half the horizontal resolution of the PIXEL mode.

VSYNC

No use is made of the information loaded from memory. It is equivalent to the LPIXEL mode.

THE LINE PARAMETER REGISTERS

SC This is a two's complemented count of the number of scanlines in the modeline, ie: OFFH for one scanline in modeline.

MB d7 If set this takes the VIRQ interrupt line low for the duration of the modeline. In conjunction with the DAVE chip this causes an interrupt to be generated at the beginning of the modeline.

(d6,d5) Defines the colour mode:

00 2-C Two colour mode. If a bit in the byte of display data is 1 a pixel of palette colour £ 1 is output and if 0 a pixel of palette colour £0. The bits are output to the screen in the following order:

d7	d6	d5	d4	d3	d2	d1	d0

01 4-C Four colour mode. Pairs of bits in the byte of display data define the colour of the pixel displayed. 00 for palette colour £0, 01 for palette colour £1, 10 for palette colour £2, and 11 for palette colour £3. The pixels are displayed in the following order:

(d7,d3)	(d6,d2)	(d5,d1)	(d4,d0)

10 16-C Sixteen colour mode. Groups of 4 bits in the byte of display data define the colour of the pixel displayed. 0000 for palette colour £0 up to 1111 for palette colour £15. The pixels are displayed in the following order:

(d7,d5,d3,d1)	(d6,d4,d2,d0)

Note that palette colours £0 to £7 have 8-bit values loaded from the line parameter table at the start of each scanline but that palette colours £8 to £15 have 8-bit values defined as follows:-

```
palette colour £8 = (f4,f3,f2,f1,f0,0,0,0)

palette colour £9 = (f4,f3,f2,f1,f0,0,0,1)
     ......
palette colour£15 = (f4,f3,f2,f1,f0,1,1,1)
```

where (f4,f3,f2,f1,f0) are the low 5 bits of the FIXBIAS register.

11 256-C Two hundred and fifty six colour mode.
In this mode the byte of display data defines the colour of a single display pixel.

The actual colour produced is as follows:-

```
RED   = [b0]*(4/7) + [b3]*(2/7) + [b6]*(1/7)
GREEN = [b1]*(4/7) + [b4]*(2/7) + [b7]*(1/7)
BLUE  = [b2]*(2/3) + [b5]*(1/3)
```

(Note that this specification of red, green and blue also occurs when allocating a colour to the palette.)

d4 =0 for /VRES. The VRES mode the LD1 and LD2 data pointers are reloaded at the start of each scanline and so the same display pattern is repeated for each scanline of the modeline.

(d3,d2,d1) defines the video display mode (see above):

000 VSYNC mode

001 PIXEL mode

010 ATTR mode

011 CH256 mode

100 CH128 mode

101 CH64 mode

110 unused at present

111 LPIXEL mode

d0 If 1 this forces a reload of the line parameter base register. This will normally be programmed to occur at the end of each video frame.

LM d7 =1 for MSBALT, ie: if the top bit of the display byte is 1 this causes palette colours £2 and £3 to be selected instead of £0 and £1 in the 2-C display mode. If the top bit is 0 palette colours £0 and £1 are used as usual. In both cases the top bit seen by the shift register is forced to 0. This mode is useful in simulating an 80 column VDU in the PIXEL mode. Since the msb or LHS of any character is 0 for character spacing it can be used to highlight areas of text.

d6 =1 for LSBALT ie: if the bottom bit of the display byte is 1 this causes palette colours £4 and £5 to be selected instead of £0 and £1 in the 2-C display mode. If the top bit is 0 palette colours £0 and £1 are used as usual. In both cases the top bit seen by the shift register is forced to 0. This mode is useful in simulating an 80 column VDU in the PIXEL mode, Since the lsb or RHS of any character is 0 for character spacing it can be used to highlight areas of text.

(d5...d) define the left hand margin of the active display. In practice this value will not be below 10 for the left hand edge of the CRT. The display changes from being border colour at the left hand margin and the display data counters start being incremented. The left hand margin defines the start of the vertical sync pulse in the VSYNC video mode.

RM d7 ALTIND1 If a 2-C character mode is selected this will cause characters with an index above 080H to have a paper of palette colour £2 and an ink of palette colour £3 instead of £0 and £1.

d6 ALTINDO If a 2-C character mode is selected this will cause characters which have their next to most significant bit set to swap palette colours as follows:-

£0 - > £4
£1 - > £5
£2 - > £6
£3 - > £7

(d5...d0) These bits define the right hand side of the active display. The maximum value is normally 54 for the right hand edge of the CRT. The display returns to the border colour at the right hand margin and the line data pointers are not incremented until the next left hand margin. In the VSYNC video mode the right hand margin defines the end of the vertical sync pulse.

LD1L This 8-bit value defines the starting value of (a7..a0) of the line data pointer LD1.

LD1 This 8-bit value defines the starting value of (a8..a15) of the line data pointer LD1.
LD1 is used as a pointer to the next byte of display data in the PIXEL and LPIXEL modes. In the CH256, CH128 and CH64 modes it is the index of a character in the character font. In the ATTR mode it points to attribute information.

LD2L This 8-bit value defines the starting value of (a7..a0) of the line data pointer LD2.

LD2H This 8-bit value defines the starting value of (a8..a15) of the line data pointer LD2.
LD2 is used as a pointer for pixel information in the ATTR display mode and as a pointer to the character font in the CG256, CH128 and CH64 modes.
It is not at present used in the PIXEL and LPIXEL modes but it is hoped that it will define vertical pixel resolution at a later date.

COL0 Palette colour £0. This is the paper colour in 2C modes though note the exceptions above.

COL1 Palette colour £1. This is the ink colour in 2C modes though note the exceptions above.

COL2 Palette colour £2 (Alternate paper)

COL3 Palette colour £3 (Alternate ink)

COL4 Palette colour £4

COL5 Palette colour £5

COL6 Palette colour £6

COL7 Palette colour £7

**********END OF DOCUMENT**********

```
;   Program to demonstrate simple use of the Nick chip.
;
;   This program sets up a simple pixel mapped display and then runs a
; moving line demonstration program on it.  Note that this program is
; designed to be started from CP/M and so is origined at 100h.  Once
; started it makes no further use of CP/M.
;
;     Copyright 1984 Intelligent Software Ltd.

RAM0    EQU     0FCH            ; First 16K of video RAM
RAM1    EQU     0FDH            ; Second  ''
RAM2    EQU     0FEH            ; Third   ''
RAM3    EQU     0FFH            ; Fourth  ''


;                       Z80 SPACE ADDRESSES
;                       -------------------
screen  equ     08000h                  ;Start address for pixel screen
lpt     equ     0C000h                  ;Start address of line parameter
                                        ;   table in RAM.
stack   equ     0D000h                  ;Z-80 stack.
;
;
;-------------------------------------------------
;
;   Data areas used for the line drawing demo program
;
;
offset  equ     0D000h
colour  equ     0D002h
count   equ     0D003h
;
BLUE_TABLE      equ     0D100h
RED_TABLE       equ     0D200h
;
x1              equ     0               ;Offsets into RED and BLUE tables
y1              equ     1
x2              equ     2
y2              equ     3
delta_x1        equ     4
delta_y1        equ     5
delta_x2        equ     6
delta_y2        equ     7
table           equ     8               ;16*4 byte table
;
;
;
;
```

```
;                       MISC PARAMETERS
;                       ---------------
NULL    EQU     00              ; NO USE VALUE

BLACK   equ     00000000b       ;Some useful colours to put in the palette.
RED     equ     01001001b
GREEN   equ     10010010b
YELLOW  equ     11011011b
BLUE    equ     00100100b
MAGENTA equ     01101101b
CYAN    equ     10110110b
WHITE   equ     11111111b


;                       LINE PARAMETER MASK BITS
;                       ------------------------

; MODE BYTE  (byte 1)
; ---------

RELOAD  EQU     001H            ; Reload line parameter counters
VBLANK  EQU     000H            ; Use margin for VSYNC
PIXEL   EQU     002H            ; Normal PIXEL mode
ATTR    EQU     004H            ; ATTRIBUTE mode
CH256   EQU     006H            ; 256 font CHAR mode
CH128   EQU     008H            ; 128 font CHAR mode
CH64    EQU     00AH            ; 64 font CHAR mode
TPIXEL  EQU     00CH            ; --------future expansion--------
LPIXEL  EQU     00EH            ; half resolution pixel mode

VRES    EQU     010H            ; low vertical resolution

C2      EQU     000H            ; 2-colour mode
C4      EQU     020H            ; 4-colour mode
C16     EQU     040H            ; 16-colour mode
C256    EQU     060H            ; 256-colour mode

VINT    EQU     080H            ; Take *INT low

; LEFT MARGIN  (byte 2)
; -----------

MSBALT  EQU     080H            ; Alt colours on D7 (PIXEL mode) 0->2
LSBALT  EQU     040H            ; Alt colours on D0 (PIXEL mode) 0->4

; RIGHT MARGIN  (byte 3)
; ------------

ALTIND1 EQU     080H            ; Alt colours on MSB of character index 0->4
ALTIND0 EQU     040H            ; Alt colours on next MSB of char index 0->2
```

```
;               Nick Chip I/O Ports
;               -------------------

FIXBIAS EQU     080H            ; Fixed colour bias
BORDER  EQU     081H            ; Border colour
LPL     EQU     082H            ; Line parameter pointer (A4-A11)
LPH     EQU     083H            ; Line parameter pointer (A12-A15) (see below)

;               LPH MASK BITS
;               ------------
LPLOAD  EQU     080H            ; Force *LOAD of LP counter low
LPCLOCK EQU     040H            ; Force *CLK of LP counter high


PAGE0   EQU     0B0H            ; Page in 4M of 0000H-3FFFH in Z80 space
PAGE1   EQU     0B1H            ; Page in 4M of 4000H-7FFFH in Z80 space
PAGE2   EQU     0B2H            ; Page in 4M of 8000H-BFFFH in Z80 space
PAGE3   EQU     0B3H            ; Page in 4M of C000H-FFFFH in Z80 space


        ASEG
        ORG     100h

        DI

        LD      A,RAM2          ;Put VIDEO-RAM in pages 2 and 3
        OUT     (PAGE2),A       ; (Page 1 not used, Page 0 already has this
        INC     A               ;  code in.)
        OUT     (PAGE3),A

        LD      A,0             ;Set BORDER colour
        OUT     (BORDER),A
        LD      A,067H          ;Any old value for fixed colour bias.
        OUT     (FIXBIAS),A

        LD      SP,STACK        ;Set up stack pointer

        LD      BC,ENTAB-LPTAB  ;Copy line parameter table from ROM
        LD      HL,LPTAB        ;  into video RAM.
        LD      DE,LPT
        LDIR

        LD      A,low(LPT/16)
        OUT     (LPL),A         ;Set up LINE PARAMETER POINTER
        LD      A,high(LPT/16)
        OUT     (LPH),A         ;Ensure LP counter is loaded (clk=0, load=0)
        OR      LPCLOCK         ;(clk=1)
        OUT     (LPH),A
        OR      LPLOAD          ;(load=1)
        OUT     (LPH),A         ;Then let it run
        ld      hl,screen       ;Clear the screen
        ld      de,screen+1
        ld      bc,256*64-1
        ld      (hl),0
        ldir
```

```
;-------------------------------------------------------------------------------
;
;   The code from here on is just a moving line demo program which uses
; the RAM from SCREEN upwards as a 4 colour 256*256 pixel bit mapped display.
;
;
                ld      de,RED_TABLE
                ld      hl,red_initial
                call    setup_table
;
                ld      de,BLUE_TABLE
                ld      hl,blue_initial
                call    setup_table
;
;
;
LOOP:           ld      a,11110000b
                ld      (colour),a
                ld      hl,BLUE_TABLE
                call    ITERATE
;
                ld      a,00001111b
                ld      (colour),a
                ld      hl,RED_TABLE
                call    ITERATE
;
                ld      a,(count)
                inc     a
                and     0Fh
                ld      (count),a
;
                jr      LOOP
```

```
;
;----------------------------------------
;
ITERATE:        push    hl
                call    NEW_XY
                inc     hl
                inc     hl
                call    NEW_XY
;
                pop     hl
                push    hl
                ld      a,(count)
                add     a,a
                add     a,a
                add     a,table
                ld      e,a
                ld      d,0
                add     hl,de
;
                push    hl
                call    LINE
;
                pop     de
                pop     hl
                push    de
                ld      bc,4
                ldir
;
                pop     hl
;
                jp      LINE
;
;
;
;-----------------------------
;
;
setup_table:    ld      bc,9
                ldir
                ld      h,d
                ld      l,e
                dec     hl
                ld      bc,16*4-1
                ldir
                ret
;
;
red_initial:    db      0,0,0,0,7,5,5,3,0
blue_initial:   db      255,0,255,0,256-5,5,256-9,3,0
;
```

```
;
;-------------------------------
;
NEW_XY:         push    hl
                pop     iy
                call    NEW_X_OR_Y
                inc     iy
;
;
NEW_X_OR_Y:     ld      a,r
                and     0Fh
                ld      c,a
                or      0F0h
                ld      d,a
;
                ld      a,(iy+0)
                ld      b,(iy+4)
                bit     7,b
                jr      nz,delta_minus
;
                add     a,b
                jr      c,overflow_plus
                cp      d
                jr      c,no_overflow
                cpl
overflow_plus:  cpl
                jr      overflow
;
delta_minus:    add     a,b
                jr      nc,overflow_minus
                cp      c
                jr      nc,no_overflow
                jr      overflow
overflow_minus: neg
overflow:
                ld      (iy+0),a
                xor     a
                sub     b
                ld      (iy+4),a
                ret
;
no_overflow:    ld      (iy+0),a
                ret
;
;
;
;--------------------
```

```
;
;
LINE:           ld      e,(hl)              ;Get two co-ordinate pairs
                inc     hl                  ; into DE and HL
                ld      d,(hl)
                inc     hl
                ld      a,(hl)
                inc     hl
                ld      h,(hl)
                ld      l,a
;
                ld      a,h                 ;Start from lowest Y co-ord
                cp      d                   ; (in DE)
                jr      nc,no_reverse
                ex      de,hl
no_reverse:
;
;
                ld      ix,normal_x         ;If starting X is higher than
                ld      a,l                 ; finishing X then flag
                sub     e                   ; reflection is required and
                jr      nc,no_reflect_x     ; negate both X co-ordinates.
                ld      ix,reflect_x
                sub     l
                ld      e,a                 ;E  = Starting X
                neg
                sub     l
no_reflect_x:   exx
                ld      c,a                 ;C' = DELTA_X
                exx
;
                ld      a,h
                sub     d                   ;D = Starting Y
                exx
                ld      b,a                 ;B' = DELTA_Y
;
                ld      iy,normal_xy        ;If DELTA_Y is more than
                cp      c                   ; DELTA_X then flag XY reflec-
                jr      c,no_reflect_xy     ; tion and swap DELTA_X and
                ld      iy,reflect_xy       ; DELTA_Y.
                ld      b,c
                ld      c,a
                exx
                ld      a,e                 ;Also swap starting X and Y
                ld      e,d                 ; co-ordinates over.
                ld      d,a
                exx
no_reflect_xy:
                ld      l,c                 ;Set HL' = ERROR = DELTA_X/2
                srl     l
                ld      h,0
                ld      a,c
                exx
                add     a,e                 ;A' = Finishing X-coordinate
                ex      af,af'
```

```
;
;
                ld      b,d             ;BC := initial co-ordinates
                ld      c,e
;
line_loop:      jp      (iy)            ;Transform from virtual co-ords
normal_x:                               ; in BC to screen co-ords in DE
                xor     a
                srl     d               ;Calculate address of pixel
                rr      e               ; = Y*64 + X/4
                rla
                srl     d
                rr      e
                rla
                set     7,d             ;Screen starts at 8000h
;
                ld      hl,mask_tab     ;Find correct mask for this
                add     a,l             ; pixel in the byte.
                ld      l,a
                ld      a,(colour)      ;Find correct mask for this
                and     (hl)            ; colour.
                ex      de,hl
                xor     (hl)            ;Exclusive-OR plot the pixel
                ld      (hl),a
;
;
                ex      af,af'          ;If reached final X co-ord then
                cp      c               ; exit.
                ret     z
                ex      af,af'
;
                inc     c               ;Step X-coordinate on by one.
                exx
                xor     a
                ld      d,a             ;ERROR -:= DELTA_Y
                ld      e,b
                sbc     hl,de
                exx                     ;If still positive then do
                jp      p,line_loop     ; not need to adjust Y.
                exx
                ld      e,c             ;Must go diagonally so
                add     hl,de           ; ERROR +:= DELTA_X
                exx
                inc     b               ;Step on Y-coordinate
                jp      line_loop
;
```

```
;
reflect_x:      xor     a
                sub     e
                ld      e,a
                jp      normal_x
;
normal_xy:      ld      d,b
                ld      e,c
                jp      (ix)
;
reflect_xy:     ld      d,c
                ld      e,b
                jp      (ix)



        IF      LOW($) GT 256-4
                ds      256-low($)
        ENDIF

mask_tab:       db      10001000b
                db      00100010b
                db      01000100b
                db      00010001b
;
```

```
;
;-------------------------------------------------------
;
;
;   The line parameter table.  This defines the screen format.
;
LPTAB:

; PIXEL for 256 lines (4 colour)
        db      256-256,pixel+c4+vres,14,14+32          ;This is the actual
        dw      screen,0                                ; display area.
        db      black,red,blue,magenta,0,0,0,0


; BLANK for 5 lines                                     ;Bottom margin.
        db      256-5,lpixel,63,0
        db      0,0,0,0,0,0,0,0,0,0,0,0

; BLACK for 10 lines                                    ;Start of vertical
        db      256-10,lpixel,4,63                      ; blanking period.
        db      0,0,0,0,0,0,0,0,0,0,0,0

; Start vertical sync (4 lines)                         ;Vertical sync pulse
        db      256-4,vblank,4,63
        db      0,0,0,0,0,0,0,0,0,0,0,0

; End vertical sync (1 line)                            ;Finish vertical sync
        db      256-1,vblank,63,4                       ; pulse.
        db      0,0,0,0,0,0,0,0,0,0,0,0

; BLACK for 30 lines
        db      256-30,lpixel,4,63                      ;Rest of vertical
        db      0,0,0,0,0,0,0,0,0,0,0,0                 ; blanking period.

; BLANK for 6 lines
        db      256-6,reload+lpixel,63,0                ;Top marigin with
        db      0,0,0,0,0,0,0,0,0,0,0,0                 ; RELOAD to start again
```

****

DPC SOUND CHIP. DPC/DAVE/1 ISSUE 4

DAVE WOODFIELD 2.9.83

CONFIDENTIAL. COPYRIGHT INTELLIGENT SOFTWARE LTD. 1983

****

The DPC Sound Chip performs the following functions:-

1. Multi-function '3 tones + noise' stereo sound generator.
2. Memory paging.
3. Address decoding for on-board ram, rom and cartridge.
4. Interrupt system including 1Hz and programmable frequency timer interrupts and two external inputs.
5. Reset circuit compatible with Z80 and dynamic ram.
6. I/O strobe signals for use with external 74LS374 tri-state octal latches.
7. 1MHz system clock.
8. Z80 wait states generation.

DPC Sound chip has 22 internal registers, 17 of which are write only, 16 of these registers are associated with the sound generation, four R/W registers are for memory management, and one R/W register is used for interrupt control. The last write only register is used for setting the overall system configuration. Internal decoding is provided for a further 3 I/O registers, read and write strobes being brought out for use with external latches and tri-state buffers on the data bus. Reset clears all 22 internal registers.

The 3 tone generators produce square waves with frequency programmable from 30Hz to 125KHz which can be modified in various ways:-

a. Distortion can be introduced by using the output frequency to sample H.F. clocked polynomial counters. PN counters which can be selected are 4,5 or 7 bit. The 7 bit PN can also be exchanged for a variable length 17/15/11/9 bit PN counter.
b. A simple high pass filter is provided on each channel, clocked by the output of a different channel.
c. A ring modulator effect is provided on each channel, with the output of a different channel for its other input.

The noise channel is normally a 17 bit PN counter clocked from 31KHz, generating a pseudo white noise. The input to this counter can be changed to clock off any of the 3 tone channels, and the PN counter can be reduced in length to 15, 11 or 9 bits. This counter can also be exchanged for the 7 bit PN counter. The resulting noise is then passed through low pass and high pass filters and a ring modulator, each controlled by the output of a different tone channel.

The 3 tone generator outputs and the noise generator output are routed to 2 amplitude control circuits (left and right). Each amplitude control consists of four 6 bit write only registers (one for each sound) which are multiplexed onto an external 6 bit D/A ladder network. In its own time slot each channel outputs the value in its amplitude register if tone is high, else zero.

Either or both of the sound output channels may be turned into 6 bit D/A outputs, when they will constantly output the value in tone channel 0 amplitude registers. This is controlled by 2 bits in the write only sound configuration register. Three further bits may be used to synchronise the tone generators by holding them at a present count until sync bit goes low.

Memory management consits of 4 read/write registers which may be output onto A14-A21 pins by selecting the required register with A14', A15'. This provides 256 * 16K pages. These outputs may be tri-stated with BREQ.

Four latched interrupts are provided, a 1Hz interrupt for time clock applications, an interrupt switchable between 50Hz, 1KHz, or the outputs of tone generators 0 or 1, and 2 external negative edge triggered interrupts. Each interrupt latch has its own enable and reset controlled by a 8 bit write only register. An attempt to read this register will return the state of the four interrupt latches and two interrupt input pins, and also two flip -flops toggling off the timer interrupts. The setting of any interrupt latch will bring IRQ low (open drain). 50Hz/1KHz/tone generator interrupt selection is made by 2 bits in the sound configuration register.

Select signals are generated for rom, cartridge, extension ram, video ram and video I/O.

A Z80 reset is provided on RSTO, either on switch on by an external RC network on CAP, or a low going signal on RSTI. The latter generates a 1ms reset pulse synchronised to the falling edge of M1 to prevent loss of data stored in dynamic ram. A 1MHz clock out is also provided.

A write only system configuration register is used to set the system for 16/64K on board ram, 8/12MHz input clock, and wait states. The wait state generator can be programmed to give zero wait states, waits on opcode fetch only, or waits on all memory accesses. Note that no wait is generated for access to video ram, as this would conflict with Z80 clock stretch.

REGISTER DESCRIPTIONS

R0 W £A0

b7-b0 Low byte of number to be loaded into 12 bit down counter to set period of tone channel 0.

R1 W £A1

b3-b0 High nybble of above, f out = 125,000/(n+1) Hz.

b5,b4
00 = Pure tone
01 = Enable 4 bit polynomial counter distortion
10 = " 5 bit " " "
11 = " 7 bit " " "

b6 1 = Enable high pass filter using tone channel 1 as clock.

b7 1 = Enable ring modulator with tone channel 2

R2 W £A2

As R0 but for tone channel 1.

R3 W £A3

As R1 but for tone channel 1 except:-

H.P.F. uses tone channel 2
R.M. uses noise channel

R4 W £A4

As R0 but for tone channel 2.

R5 W £A5

As R1 but for tone channel 2 except:-

H.P.F. uses noise channel
R.M. uses tone channel 0

R6 W £A6

b1,b0 Select noise clock frequency:-

00 = 31.25KHz
01 = tone channel 0
10 = " " 1
11 = " " 2

b3,b2 Select polynomial counter length:-

00 = 17 bit
01 = 15 bit
10 = 11 bit
11 = 9 bit

b4 1 = Swop 17 bit and 7 bit polynomial counters

b5 1 = Enable low pass filter on noise using tone channel 2 as clock.

b6 1 = Enable high pass filter on noise using tone channel 0 as clock.

b7 1 = Enable ring modulator with tone channel 1

R7 W £A7

b0 Sync for tone channel 0.
(1 = hold at present, 0 = run)

b1 Sync for tone channel 1.

b2 Sync for tone channel 2.

b3 1 = Turn L.H. audio output into D/A, outputting value in R8.

b4 1 = Turn R.H. audio output into D/A, outputting value in R12

b6-b5 Select interrupt rate:-

00 = 1KHz
01 = 50Hz8
10 = Tone generator 0, f=250,000/(n+1)
11 = Tone generator 1

b7 Undefined

```
R8  W  £A8
----------
    b5-b0   Tone channel 0 L.H. amplitude
            Also value output to L.H. D/A if R7 b3 = 1

R9  W  £A9
----------
    b5-b0   Tone channel 1 L.H. amplitude

R10 W  £AA
----------
    b5-b0   Tone channel 2 L.H. amplitude

R11 W  £AB
----------
    b5-b0   Noise channel L.H. amplitude

R12 W  £AC
----------
    b5-b0   Tone channel 0 R.H. amplitude
            Also value output to R.H. D/A if R7 b4 = 1

R13 W  £AD
----------
    b5-b0   Tone channel 1 R.H. amplitude

R14 W  £AE
----------
    b5-b0   Tone channel 2 R.H. amplitude

R15 W  £AF
----------
    b5-b0   Noise channel R.H. amplitude

R16 R/W  £BO
------------
    b7-b0   Page register output to A14-A21 if A15',A14' = 00

R17 R/W  £B1
------------
    b7-b0   Page register output to A14-A21 if A15'.A14' = 01

R18 R/W  £B2
------------
    b7-b0   Page register output to A14-A21 if A15',A14' = 10

R19 R/W  £B3
------------
    b7-b0   Page register output to A14-A21 if A14',A15' = 11
```

R20 W £B4

b0	1 = Enable 1KHz/50Hz/TG interrupt
b1	1 = Reset 1KHz/50Hz/TG interrupt latch
b2	1 = Enable 1Hz interrupt
b3	1 = Reset 1Hz interrupt latch
b4	1 = Enable INT1
b5	1 = Reset INT1 latch
b6	1 = Enable INT2
b7	1 = Reset INT2 latch

R20 R £B4

b0	1 = 1KHz/50Hz/TG latch set
b1	1 = 1Hz latch set
b2	1 = INT1 latch set
b3	1 = INT2 latch set
b4	1KHz/50Hz/TG divider, (fint/2 square wave)
b5	1Hz divider, (0,5Hz square wave)
b6	INT1 input pin
b7	INT2 input pin

R21 W £B5

Active low strobe on WRO

R21 R £B5

Active low strobe on RDO

R22 W £B6

Active low strobe on WR1

R22 R £B6

Active low strobe on RD1

R23 W £B7

Active low strobe on WR2

R23 R £B7

Active low strobe on RD2

R31 W £BF
---

b0 On board RAM, 0=64k,1=16k

b1 Input clock frequency, 0=8MHz, 1=12MHz

b3,b2 00 = Wait on all memory access except video ram
01 = Wait on M1 only, except video ram
10 = No waits
11 = No waits

SELECT OUTPUTS

VIO Low for I/O access £80 to £8F. Gated with IORQ,RD,WR in video chip.

ROM Low for memory on pages 0 to 3. (0-£FFFF)

CART Low for memory on pages 4-7, (£10000-£1FFFF)

VRAM Low for any memory access on pages £FC-£FF. (£3F0000-£3FFFFF) IF R31 b0 = 0

Low for any memory access other than rom or cartridge, (£20000-£3FFFF) IF R31 b0 = 1

Gated with MREQ,RD,WR in video chip.

XRAM Low for any memory access on pages £F8-£FB. (£3E0000-£3EFFFF)