1. Introduction

The serial interface and network are provided as two separate EXOS device drivers as far as the user is concerned, with device names "SERIAL:" and "NET:" 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. The other bits of this output port are not used.

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, so the actual values of the bits will be clear for a high line and set for a low line.

3. Serial Device

3.1 Low-level Operation

The serial device uses five wires - DATA IN, DATA OUT, STATUS IN, STATUS OUT and REF (which may be 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 serial 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 changed 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).

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 codes:

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 15 (9600 baud). Numbers greater than 15 are reduced modulo 16 before interpretation.

3.2.3 Word Format Selection

The EXOS variable FORM_SER, defines the word format which is used for both input and output. Certain 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.
```

The default setting is zero which selects 8 data bits, no parity and two stop bits.

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 a second 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.

Each block that is sent consists of a header which may or may not be followed by a block of data bytes. The header includes a synchronisation pattern, the source and destination addresses and a type byte, of which the latter contains a flag determining whether any data bytes are to follow. In every header there is also a count of the number of data bytes in the block, although this is ignored if the type byte specifies that no data at all will be sent.

4.1.1 Broadcast Protocol

The header of a block which is being broadcast contains 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 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.

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 (about 4 bit periods at the selected baud rate) is interpreted by the source machine as an error.

If any data bytes are to be sent, these are transmitted once the header has been acknowledged. 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 it 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 data block, retries as follows:

1. Release network *
2. Wait for long random delay, of the order of a quarter of a second.
3. Try to gain control of network and send again

* Note: 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 transactions 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 on this channel, any number of such non-zero address channels may be opened provided they all have different addresses.

If a non-specific channel is opened it will only receive data from machines which are not explicitly served by an individual channel. Only one non-specific channel may be open at a time.

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 corresponds to a delay of the order of 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.

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 to which this block is being sent; 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
(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.

Type byte

The type byte contains information defining what kind of transaction this is, including whether or not any data bytes will be sent after the header. Its format is as follows:

b0 - data flag: 0 => no data to follow header
1 => data block to follow

b1..b4 - not defined - must be zero

b5 - end-of-record flag:
1 => end of record
If the block was sent as a result of a Flush or Close command, this bit is set to force the network Read Character routine to return the 'End of file' status, thus identifying the end of a given message or file. The next Read Character or Read Status call will revert to the 'Character not available' state, unless the end-of-file flag is also set...

b6 - end-of-file flag:
1 => end of file
Only ever set in conjunction with the end-of-record flag, it implies that the sending machine has closed its channel to this computer. All subsequent requests for channel status or for further characters will return the 'End of file' condition, until another block of data is received on this channel.

b7 - Type byte flag - must ALWAYS be set to 1, so that the synchronisation byte is guaranteed to be the only string of eight zero bits in the header.

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 looking at the DATA line.

The source machine is not expected to wait for long enough to ensure that all machines will be listening, but instead 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 header 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)

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. Store the synchronisation byte, and the next three bytes which are received, in scratch memory. These four values are the header pattern, and should be sent repeatedly by the source machine.

8. Check that the destination byte is valid, and that it holds either this machine number or the value zero. Return if this is not the case.

9. Check that the source byte is valid, and that there is a channel open which can serve the specified machine - ie. either a specific channel for that address, or a non-specific channel. Otherwise return.

10. Check that the type byte is valid.

11. Continue to read bytes. Check that each group of four matches the values stored in step 7, and continue to loop until a value does not tally.

12. The value which disagrees should be the sync byte, and it should contain instead the ones complement of the dest byte. If this is not the case, return.

13. Read the 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.

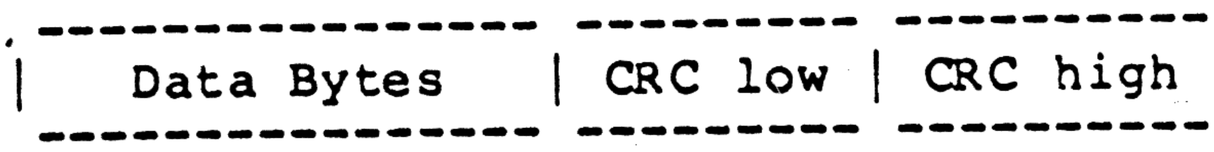
14. If the destination address was non-zero (ie. transaction is directed rather than broadcast), pull DATA low for about 1ms as acknowledgement, then release it.

NOTE: Whenever a byte is read, STATUS is checked for still being held low by the source machine. If it goes high at any time, the link is immediately assumed broken.

4.2.5 Sending the Data Bytes

When the destination machine pulls the DATA line low, and leaves it there for at least half a millisecond, that is the signal to the source machine to start sending any data which may be included in this transaction. When DATA is released, the sender pulls it low again immediately then gets on with sending 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 (but not the header). The CRC is evaluated as a 16-bit value, which is re-calculated when the data block is received. If the CRC calculated on reception is not the same as the value which was sent 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 is done in the serialisation routine which is shared with the serial interface driver, so the CRC is also calculated on data sent or received through the serial handler even though this is not required.

The CRC register is a 16 bit value 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, moving 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 characters are transmitted at the currently selected baud rate, which defaults to 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 the unit number (see EXOS kernel specification). Any filename given is ignored.

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 when he wishes to output to the network. For input, both values are made available to him by the interrupt handling code (see below).

4.3.3 Interrupts

When the first channel is opened to the network, the external interrupts are enabled and will remain so 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 bytes which arrive in blocks on the network, and can hold just one block at a time (no matter how short that block is; even a block of just one byte effectively 'fills' the receive buffer until it has been read or cleared). The user reads from the buffer until it is empty, when it will be able to accept further blocks from the network. The buffers of each channel are independent, so blocks can be received from any machine provided that the receive buffer which serves it 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" special function call, 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 Functions

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.

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 always 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.

Whenever CHAN_NET is updated, the EXOS variable MACH_NET is also altered to hold the number of the machine which caused the interrupt. This is vital when a block is received on the non-specific channel, since there would otherwise be no way of telling which machine sent it. In other cases it is simply useful - as stated above, the user would normally be expected to keep his own records of which channel serves which machine.

An application program requests bytes by calling the EXOS RDCH or RDBLK routines, and is given bytes from the receive buffer if they are available. If not, there are two possible responses: If an End-of-record flag was received in a header for this channel, an 'End of file' condition is returned. Otherwise, the Read routine will halt until this channel receives data from the network.

The "read channel status" function call is supported, so the user can check for the 'End of file' or 'Character not available' conditions before trying to read a character.

Writing to the network is carried out as a series of WRCH or WRBLK function calls. If an error is encountered when the buffer is to be sent, the network handler will continuously retry at intervals of between quarter and half a 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.

5. Quick Reference Summary

5.1 SERIAL - EXOS calls

OPEN/CREATE CHANNEL	- Treated identically. Only one channel, and no network channel. Device name "SERIAL:". Filename and unit number ignored. No EXOS variables to be set before open.
CLOSE/DESTROY CHANNEL	- Treated identically.
READ CHARACTER/BLOCK	- Reads characters from serial input, some buffering.
WRITE CHARACTER/BLOCK	- Writes bytes without interpretation.
READ STATUS	- Always character ready (C=0).
SET STATUS	- Not supported.
SPECIAL FUNCTION	- No special functions.

5.2 NETWORK - EXOS calls

OPEN/CREATE CHANNEL - Treated identically. Multiple channels allowed, but no serial channel. Unit number defines destination machine number, filename ignored. Device name "NET:". EXOS variable ADDR_NET must be set before open.

CLOSE/DESTROY CHANNEL - Treated identically. Will try to send any buffered data.

READ CHARACTER/BLOCK - Reads characters from buffer if avaialable, else returns EOF or waits.

WRITE CHARACTER/BLOCK - Puts bytes in buffer. Written to network when FLUSH special function call is done, or channel is closed, or buffer is full.

READ STATUS	-	C=0 if character in buffer. C=0FFh if end of record. C=1 if buffer empty.
SET STATUS	-	Not supported.
SPECIAL FUNCTION	-	@@FLSH = 16 Send buffered data with end-of-record flag set. @@CLR = 17 Clear send and receive buffers.

5.3 EXOS Variables

FORM_SET	-	Serial format. Set to zero by network.
BAUD_SER	-	Serial and network baud rate.
ADDR_NET	-	Network address of this machine.
CHAN_NET	-	Channel on which network block received.
MACH_NET	-	Machine from which network block received.

NET_IRQ - Zero to enable network software interrupts

5.4 Software Interrupt Codes

?NET = 30h Occurs when a data block is received by a network channel if NET_IRQ is zero.

++++++++++ END OF DOCUMENT ++++++++++