Support Group Application Note
Number: 004
Issue: 1
Author:

Tube Application Note

Applicable
Hardware :

BBC B
BBC B+
BBC Master 128

Related
Application
Notes:

Support Group
Acorn Computers Limited
Acorn House
Vision Park
Histon
Cambridge CB4 4AE

Overview

One of the BBC Microcomputer's strengths lies in its sophisticated Operating System, the MOS. This operating system has a very fast and flexible response to Interrupts, which allows the machine to take a wide range of peripherals and handle them with ease. The TUBE is a fast bus interface through which additional Co-processors (also called second processors) can be added. when a co-processor is connected to the TUBE interface, the BBC Micro continues to look after all of the I/O processing, whilst the additional co-processor now carries out the task of running the Language Application.

The Co-Processor

The co-processor can be based on any microprocessor chip, and can have any memory size that this chip can address. Units already in existence include 64K RAM 6502, 1MBit 32016, 64K Z80 etc. You may wonder why there is any point in adding a unit using a 6502 chip, the same series as the BBC Micro itself, and with a RaM size the same as the Model B+? The reasons are twofold:

1) Speed - whilst the co-processor is carrying out the computational tasks of the Application program, the BBC Micro itself can look after the screen display, printer, disc drive etc.

2) Memory - In the Model B or B+ with a Disc Filing System, the position of PAGE is typically &1900. The application has to fit above this and below HIMEM. In the Model B, HIMEM moves down when a high resolution screen display is selected further reducing the memory available. In the 6502 co-processor, PAGE is typically at &800 and HIMEM is at &8000 regardless of Filing System and screen display mode.

The TUBE interface

The co-processor is connected to the TUBE connector on the BBC Micro via the TUBE cable. In the system, the BBC Micro is referred to as the "host", and the co-processor as the "parasite". Within any co-processor there is a custom chip, the TUBE ULA, which provides the actual communicating interface between both processors. The processors are not synchronised with one another, and the ULA chip forms an effective set of pigeon holes through which each processor can leave information for the other to read when it is ready.

For the TUBE interface to work, there has to be a set of machine code in both processors which looks after the communicating protocols on both sides of the ULA. In the BBC Micro, this "host" code resides typically in one of the sideways ROM's ie NFS 3.34, DNFS, 1770 DFS etc. In the co-processor, the "parasite" code resides in a small ROM commonly called the "boot" ROM.

Software Compatibility

Applications software which has been written for the BBC Micro itself may not run on a 6502 co-processor if it has not followed the rules for using the Tube interface. applications writers may become over familiar with the memory map in the BBC Micro and it's operating system interface and "bypass" the standard routines. This application will then almost certainly fail when it is run in the different memory map of a cop-processor and where not all of the assumed interfaces are available. In the following sections of the Application Note, the various "legal" way of interfacing with the operating system will be described. if properly implemented, the Application Program will then run in either the BBC Micro or the 6502 co-processor, with no modification. If the communication techniques across the Tube interface are also properly implemented then maximum advantage of the overall system speed can be obtained without one processor waiting an unreasonable length of time fo the other processor to respond.

The Load address used by filing system file names contains a "high order" address which indicates which processor memory map it is to be loaded into. for example, a load address of &FFFFxxxx indicates that the file should be loaded into the I/O processor area. A load address of &0000xxxx indicates that the parasite is the destination. In practice, filing systems shorten the high order bytes to two ie &FFxxxx or &00xxxx. A file loaded under BASIC control will load into the current language processor regardless of the file load address. This ability to force *LOADing of files into a particular processor can be used to set up user machine code from a filing system into the I/O processor when the co-processor is in use.

The TUBE ULA

As stated previously, the ULA acts as a parallel interface between two asynchronous processor systems. It consists of four byte-wide read-only registers and four byte-wide write-only registers. eight bytes of memory mapped I/O space are used to address these registers, four for the data registers and four for the associated status registers.

<u>Register number 1</u>

	<u>I/O address</u>	<u>Co-proc address</u>	
status	&FEE0	&FEF8	write/read (clears IRQ)
data	&FEE1	&FEF9	bit 7 - data available/IRQ
			bit 6 - not full

Parasite to Host: Carries the OSWRCH call. Data register is a FIFO that can handle a VDU command length (10 bytes).

Host to Parasite: There is a 1 byte buffer. It is used to generate IRQ's in the parasite from events in the host.

<u>Register number 2</u>

	<u>I/O address</u>	<u>Co-proc address</u>	
status	&FEE2	&FEFA	write/read
data	&FEE3	&FEFB	bit 7 - data available
			bit 6 - not full

Used to implement OS calls that take a long time or that cannot interrupt Host tasks. The parasite passes a byte describing the required task. The two processors then exchange data until the task is complete. OS calls handled through this register include: OSRDCH, OSCLI, OSBYTE, OSWORD, OSBPUT, OSBGET, OSFIND, OSARGS, OSFILE, OSGBPB.

<u>Register number 3</u>

	<u>I/O address</u>	<u>Co-proc address</u>	
status	&FEE4	&FEFC	write/read
data	&FEE5	&FEFD	bit 7 - data available/NMI
			bit 6 - not full

Used for the background task of fast data transfer between the two processors.

Register number 4

	I/O address	Co-proc address	
status	&FEE6	&FEFE	write (sets IRQ)/read (clears IRQ)
data	&FEE7	&FEFF	bit 7 - data available/IRQ
			bit 6 - not full/IRQ

Used as the control channel for block transfers going through Register 3, and also the transfer register for error strings from host to parasite. In both cases, the host interrupts the parasite by placing a byte into the Register. In the former case it is a byte describing the required action, in the latter it is an error code.

Writing for compatibility

The applications software writer needs to know which operating system interfaces will work "across the tube", and which will not. He will also need to know how to implement other techniques for those that don't work.

Valid calls across the Tube: OSWRCH, OSBYTE (Y only returned for A>= &80), OSRDCH, OSCLI, OSWORD, OSBPUT, OSBGET, OSFIND, OSFILE, OSARGS, OSGBPB.

Note:
a) OSBYTE calls have some restrictions in the co-processor ie:
 &00-&7F - Only X (not Y) is sent and returned.
 &82 - Always returns 00 in both X and Y (ie the parasite high order address).
 &83 - Always returns 00 in X and 08 in Y (ie OSHWM in parasite).
 &84 - Always returns &8000 or &B800 (ie the position of HIMEM in the parasite).

b) OSWORD call parameters sent and received across the Tube can be ascertained from the following table

OSWORD No.	Params sent	Params received
1 (&01)	0	5
2 (&02)	5	0
3 (&03)	0	5
4 (&04)	5	0
5 (&05)	2	5
6 (&06)	5	0
7 (&07)	8	0
8 (&08)	14	0
9 (&09)	4	5
10 (&0A)	1	9
11 (&0B)	1	5
12 (&0C)	5	0
13 (&0D)	0	8
14 (&0E)	16	16
15 (&0F)	16	16
16 (&10)	16	13
17 (&11)	13	13
18 (&12)	0	128
19 (&13)	8	8
20 (&14)	128	128
<128 (&80)	16	16
>127 (&7F)	XY offset 0	XY offset 1 (ie in param block)

Invalid calls across the Tube: OSRDRM/OSRDSC, OSWRSC

Other invalid actions include:
Peeking/Poking memory unless the location is guaranteed to be the same in both memory maps.
Directly addressing memory mapped I/O locations such as VIA.
Directly address &F4, the ROM latch variable.
Keeping data areas in &A00, &B00 and &C00. These are above PAGE in the parasite.
Using pages 4, 5, 6 and 7 in the I/O processor, normally reserved for the current language. these are used by the Tube code.
Using the two pages above normal PAGE position in the I/O processor. These are used for font explosion when a parasite processor is being used. Therefore stay above &2100, if PAGE is normally &1900, for user machine code.

Valid communicating techniques:
OSBYTE 146 - Read from FRED (1MHz bus).
OSBYTE 147 - Write to FRED (1MHz bus).
OSBYTE 148 - Read from JIM (IMHz bus).
OSBYTE 149 - Write to JIM (1MHz bus).
OSBYTE 150 - Read from SHEILA (memory mapped I/O).
OSBYTE 151 - Write to SHEILA (memory mapped I/O).
OSBYTE 157 - Fast Tube BPUT.
OSBYTE 234 - Read Tube presence.
OSWORD 5 - Read from I/O processor (transfers 1 byte).
OSWORD 6 - Write to I/O processor (transfers 1 byte).
OSWORD 224 to 225 - Passed through USERV vector (Can transfer up to 128 bytes either way across the Tube).

Some typical examples:

a) Moving a few bytes: Use OSWORD 5 or 6 from the co-processor the appropriate number of times.

b) Moving up to 128 bytes: OSWORD 5 or 6 is typically too slow for more than a few bytes, hence the following is a recommended approach for the transfers to or from the co-processor and initiated by the co-processor:

Use OSWORDS in the range &E0(224) to &FF(255). These "unknown" OSWORDS are passed through the USERV vector at &200 in the I/O processor. Thus to transfer 64 bytes from the co-processor to the I/O processor we select, say OSWORD &E0, with the following parameter block:

XY + 0	68	bytes to transmit
1	0	bytes to receive
2	LO	Lo byte of destination in I/O processor
3	HI	Hi byte of destination in I/O processor
4	x	1st byte of 64
|	|	|
68	x	64th byte of 64

To transmit 64 bytes from the I/O processor to the co-processor, we select, say OSWORD &E1, with the following parameter block:

XY + 0	4	bytes to transmit
1	68	bytes to receive
2	LO	Lo byte of source location in I/O processor
3	HI	Hi byte of source location in I/O processor

In practice you would intercept the USERV vector in the I/O processor (at &200,1) for the first run of the program. When &E0 is intercepted, the data from the parameter block already set up is ready to be moved across the Tube by the co-processor Tube code into the I/O processor location given in the parameter block. Control is returned to the program. When &E1 is intercepted, the data is taken from the address given in the parameter block and placed in the parameter block starting at address offset 4. Following this, the Tube code will copy the data across the Tube into the co-processor into the original parameter block location.

c) Moving large quantities of data: To move large quantities of data, for example a complete screen update, the fast Tube BPUT OSBYTE call is used. Prior to the call, some machine code is placed in the I/O processor ready to handle the data as it arrives across the Tube from the co-processor. When data is ready to be moved across the Tube, an OSBYTE &9D (157) is made to initiate the users code to in the I/O processor. The data is moved through Tube register 1 and handled by the user code on the other side. Note that all OSBYTE calls are vectored through the BYTEV vector at &20A,B in the I/O processor, and this vector has to be intercepted to detect an OSBYTE &9D call and pick up the X and Y parameters to be used by the user's code. The advantage of OSBYTE &9D over other OSBYTE calls is that after the X and Y parameters have been sent across the Tube, control is immediately returned to the parasite. no parameters are returned.

The Correct Use of Tubes

The Tube (c)Acorn Computers etc. is both a custom chip and a set of protocols. The protocols control the flow of control and data between a second processor and the BBC machine to which it is attached. Clearly the implementation of these protocols in the second processor is different with different processors and operating environments, so this document is not concerned with second processor code. This document does however please constraints on the performance of second processors, as the description of the Tube use on the BBC machine side includes expected response times so that the BBC machine user need not poll the hardware.

Conventions.

We assume you know about the 6502, and a bit about BBC machines. Hexadecimal numbers are written &<Hex digit> (<Hex digit>^) eg &FEE5, &0406. Decimal numbers are just written. Host means BBC computer, parasite means second processor.

1. Claiming the Tube

Before you can use the Tube, you must claim it successfully. This is to prevent reentrancy problems with background and foreground tasks trying to use the system at the same time, for instance during an Econet (c) peek. Of course, before attempting to use the Tube system you must be certain that the Tube is present, by using OSBYTE call &EA with Y=&FF, X=0. The answer, in X, is 0 if there is no Tube or &FF if the Tube system is present. Only if the Tube is present may you call the Tube code entry point, as otherwise it is language workspace eg BASIC's variables.

To claim the Tube you must call the Tube code entry point at &0406 with a reason code of &C0+x in the accumulator. x is an ID code which should identify you uniquely. this call return with the carry set if the claim was successful, carry clear if it failed. Failure implies that some background task is using the Tube, so the usual course of action is to keep trying until yu succeed. For example, the DFS uses the following subroutine .... (MASM format)

```
CLATUB PHA              ; Save A, as it happens
CLATBO LDAIM &C1        ; My magic number
       JSR &0406        ; Tube code entry point
       BCC CLATBO       ; If it failed try again.
       PLA              ; Recover A
       RTS
```

Some other magic numbers which have been allocated are:

&C0 - Cassette filing system
&C1 - Disc filing system - DFS
&C2 - Econet - Low level primitives
&C3 - Econet filing system
&C4 - ADFS
&C5 - Teletext
&C6 - Acorn in-house Terminal - HOSTFS (?)
&C7 - VFS - video discs
&C8 - 64/128k beebs sideways RAM utils
&C9 - Z80 chaps, CP/M

::
&CF - Acacia RAM FS (not allocated by us!) - also user applications

The ID code is in fact a six bit quantity, so you should use:
LDAIM &CO+MYID
JSR &0406
in your code.

When you have finished using the Tube you must release it so that other users may claim it. So that another user cannot release the Tube when you claimed it, you must call the entry point with &80+x in the accumulator, where x is the same magic number you used when claiming. For example DFS uses the following subroutines.... (MASM format):

```
RELTUB PHA              ; Save A
       LDAIM &81        ; Magic number
       JSR &0406        ; Tube entry point
       PLA              ; Get A
       RTS
```

2) Data Transfers/Execution

The same entry point is used to initiate a data transfer through the Tube. The type of transfer is selected by means of a reason code in the accumulator. You must also tell the system where in the second processor's memory space to start the transfer. You must place the address of te first byte to the moved (source or destination, depending on the transfer type, or the Execute address if you are forcing execution in the parasite) in four bytes of memory in the BBC machine, low byte first as usual, and put the low byte of the address of these four bytes in X and the high byte of the address of the four bytes in Y. So . .

```
              Four byte address in BBC m/c
   YX ---> :   Low byte     :  \
           :   MidLo byte   :   > - - - - - -> data byte in second processor.
           :   MidHi byte   :  /
           :   Hi byte      : /
```

Reason codes for data transfers are as follows:

RC	Description	Initial delay	Delay per byte
0	Multi byte transfer, parasite to host	24 uS	24 uS
1	Multi byte transfer, host to parasite These transfer any number of bytes in the appropriate direction - terminate by releasing the Tube or recommanding for another protocol.	0	24 uS
2	Multi pairs of bytes transfer, parasite to host	26 uS	26 uS/pair
3	Multi paris of bytes transfer, host to parasite These transfer an even number of bytes in the appropriate direction, faster than protocols 0 and 1 - terminate by	0	26 uS/pair

	releasing the Tube or recommanding for another protocol.		
4	Execute - Execution starts in the parasite at the address pointed to by YX. This call contains an implied release and does not return to you.		
5	Reserved - this call is used in handling OS calls which are passed across the Tube.		
6	256 byte transfer, parasite to host	19 uS	10 uS/byte
7	256 byte transfer, host to parasite These transfer exactly and only 256 bytes only after 256 bytes are transferred may the system be recommanded or released.	0	10 uS/byte

Having commanded the system, you may now transfer the data. The initial delay is the time you must wait after control returns to you before you transfer the first byte of data. The port used to transfer the data is memory mapped into the BBC machine at location &FEE5 (another magic number !), so either do an LDA &FEE5 for parasite to host or STA &FEE5 for host to parasite to transfer the data.
eg To transfer 256 bytes of data into an arbitrary page in the BBC machine memory from the parasite.

```
; Set up page zero locations &80, &81 to point to the destination page
; Set up locations &3000, &3001, &3002, &3003 to contain the source address in the parasite

CLAIM  LDAIM  &CO+&10          ; Say my ID is 16
       JSR &0406               ; Claim the Tube
       BCC CLAIM

       LDXIM &00               ; Lo byte of &3000
       LDYIM &30               ; Hi byte of &3000
       LDAIM 6                 ; P -> H, 256 bytes
       JSR &0406

       JSR ANRTS               ; 6 uS delay
       JSR ANRTS               ; 12 uS delay
       JSR ANRTS               ; 18 uS delay

       LDYIM 0                 ; so the initial delay in 19 uS

LOOP   LDA &FEE5               ; Get it from the port ( 2 uS = 2)
       STAIY &80               ; Put the data byte     (+3 uS = 5)
       NOP                     ;                       (+1 uS = 6)
       NOP                     ;                       (+1 uS = 7)
       NOP                     ;                       (+1 uS = 8)
       INY                     ;                       (+1 uS = 9)
       BNE LOOP                ; Next data byte        (+1.5 uS = 10.5 uS/byte)

       LDAIM &80+&10           ; Release code + My ID
       JSR &0406               ; Release the Tube
```

```
            :
            :
ANRTS RTS
```

Execute:
Calling the Tube code with A=4 is used to force execution to start in the second processor at a location defined by the parasite. This is used by filing systems when *RUNning a file. The filing system claims the tube, loads the program code into the second processor RAM and then uses call 4 with YX pointing to the exec address of the file to start execution of the code.

3) OSWORD calls

These are a few things you should know if you are going to use your own OSWORD calls to pass control/ data back and forth across the Tube.

Low numbered OSWORD calls:
These have variable numbers of parameters both in and out. In practice, for all OSWORD calls with A<128, 16 bytes are passed each way. This covers all the Acorn assigned calls and allows them to be made transparently from either side of the Tube.

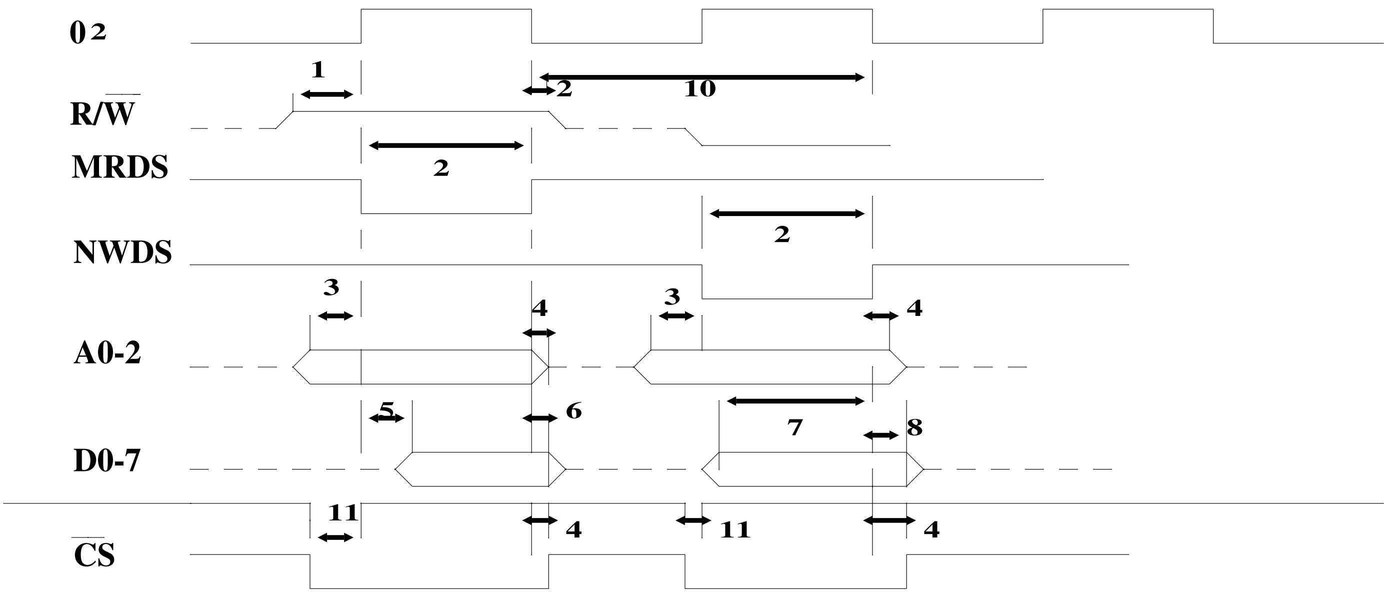
High numbered OSWORD calls:
OSWORD calls with A>128 have a special format to allow a variable number of parameters to be passed each way.

```
YX ----> n  (byte)
YX+1     m  (byte)
YX+2     data
```

when the call is made in the second processor, n bytes (inclusive of the bytes containing n and m) are copied into the I/O processor, then the call is made there with YX pointing to the copy. When the call returns, m bytes (inclusive again) are copied back into the second processor. 2 <= n,m <= 128, so you can pass at most 126 bytes of data back and forth. eg if you wish to pass a 4 byte record, the first byte of which is a status return, use:

```
YX  ->  6       ; 6 bytes to the I/O processor
        3       ; 3 bytes returned
        data0
        data1
        data2
        data3
```

TUBE AC Electrical Specification



NB On the host side, 0_2 is the timing reference and R/W gives the direction of transfer. On the parasite side, the timing and direction are implied by MRDS or NWDS.

	MIN	MAX
1) R/W set up to 0_2	35 ns	
2) timing strobe pulse width	110 ns	
3) address set up time	35 ns	
4) address and chip select hold times	10 ns	
5) data out delay time		70 ns
6) data out hold time	20 ns	
7) data in set up time	50 ns	
8) data in hold time	20 ns	
9) R/W hold time	10 ns	
10) cycle time	250 ns	
11) CS set up time	20 ns	

The chip must operate within this specification, as this meets 4MHz 6502 requirements.

All other timings, such as across tube transfer times, are non critical, but are expected to be at most 1 or 2 microseconds

Interrupt Operation

The tube has three processor interrupt outputs, two to the parasite (PIRQ & PNMI) and one to the host (HIRQ). Each line has an enable bit, and PIRQ has two, one for each possible interrupt source. The interrupt lines go active (low) under the following conditions:

HIRQ Q = 1 and register 4 has data available in the parasite to host latch

PIRQ either: I = 1 and register 1 has data available in the host to parasite latch
or: J = 1 and register 4 has data available in the host to parasite latch
(or both)

PNMI either: M = 1 V = 0 1 or 2 bytes in host to parasite register 3 FIFO or 0 bytes in parasite to host register 3 FIFO (this allows single byte transfers across register 3)
or: M = 1 V = 1 2 bytes in host to parasite register 3 FIFO or 0 bytes in parasite to host register 3 FIFO. (this allows two byte transfers across register 3)
(or both)

In all cases the interrupt condition is cleared by removing the cause; in the case of HIRQ or PIRQ reading the data from the appropriate register, in the case of PNMI reading or writing data to or from register 3 as appropriate.

Reset Operation

An active (low) signal on HRST initialises the tube to a known state, and automatically produces a PRST active output to reset the parasite system.

The state is T, P, V, M, J, I, Q are all set to zero.

All the registers are purged except that register 3 has one valid but insignificant byte in the parasite to host FIFO (this is to prevent an immediate PNMI state after PRST).

The T control bit allows the processor to reset the Tube to the above state with the exception that P, V, M, J, I & Q are unaffected, and P allows separate reset of the parasite processor by the host under software control. These resets are activated by setting the respective flags, and they must be cleared before the reset device will operate again, unless of course HRST is activated in the mean time.

DMA Operation

The DRQ pin (active state = 1) may be used to request a DMA transfer - when M = 1 DRQ will have the opposite value to PNMI, and depends on V in exactly the sam way (see description of interrupt operation). DACK then selects register 3 independently of PAo-2 and PCS, and forces a read cycle if PNWDS is active or a write cycle if PNRDS is active (not inverse sense of PNWDS and PNRDS so that the DMA system can read the data from memory an write it into the Tube in one cycle).

Control and Status Flags

In the above table the positions in the memory map of the various status and control bits are shown, and their significance is explained below.

A1, A2, A3, A4 = 1	data available in register 1, 2, 3, 4
F1, F2, F3, F4 = 1	register 1, 2, 3, 4 not full
N =1	register 3 action required (if M = 1 then PNMI active)

The data available flag signifies data available <u>to</u> the processor reading the flag, whereas the not full flag shows that the register <u>from</u> the processor reading the flag has space for more data.

In the case of a simple latch such as register 2, A2 on the host side and F2 on the parasite side will always have the opposite value. In the case of the FIFO registers, data is available when there is one of more valid byte in the register, but the not full flag only becomes inactive when the entire register is loaded.

All registers are simple latches except register 3, which has 2 byte FIFOs in each direction, and register 1, which has a ten or more byte FIFO from the parasite to the host. The host to parasite part of register 1 is a simple latch.

Q = 1	enable HIRQ from register 4
I = 1	enable PIRQ from register 1
J = 1	enable PIRQ from register 4
M = 1	enable PNMI from register 3
V = 1	two byte operation of register 3
P = 1	activate PRST
T = 1	clear all Tube registers
S = 1	set control flag(s) indicated by mask

These flags are set or cleared according to the value of S, eg writing 92 (hex) to address 0 will set V and I to 1 but not affect the other flags, whereas 12 (hex) would clear V and I without changing the other flags. All flags except T are read out directly as the least significant 6 bits from address 0.

Register 3 Operation

Register 3 is intended to enable high speed transfers of large blocks of data across the tube. It can operate in one or two byte mode, depending on the V flag. In one byte mode the status bits make each FIFO appear to be a single byte latch - after one byte is written the register appears to be full. In two byte mode the data available flag will only be asserted when two bytes have been entered, and the not full flag will only be asserted when both bytes have been removed. Thus data available going active means that two bytes are available, but it will remain active until both bytes have been removed. Not full going active means that the register is empty, but it will remain active until both bytes have been entered. PNMI, N and DRQ also remain active until the full two byte operation is completed.

General Description

The Tube is a completely asynchronous parallel interface between two processor systems. To each system it resembles a conventional peripheral device, occupying 8 bytes of memory or I/O space. within that space are four byte wide read only latches, and four byte wide write only latches, plus associated control flags. Some of the latches are just that - data written in one side is read out of the other on the next read to that address, but some are in fact FIFO buffers, which store two or more bytes to be read out in the order they were put in. Information is stored in the Tube until removed by the receiving processor, thus allowing completely asynchronous operation of the two systems. messages and data are passed to and fro through the various registers according to carefully designed software protocols, and proper allocation of the registers to specific tasks allows both systems to operate with the minimum waiting time.

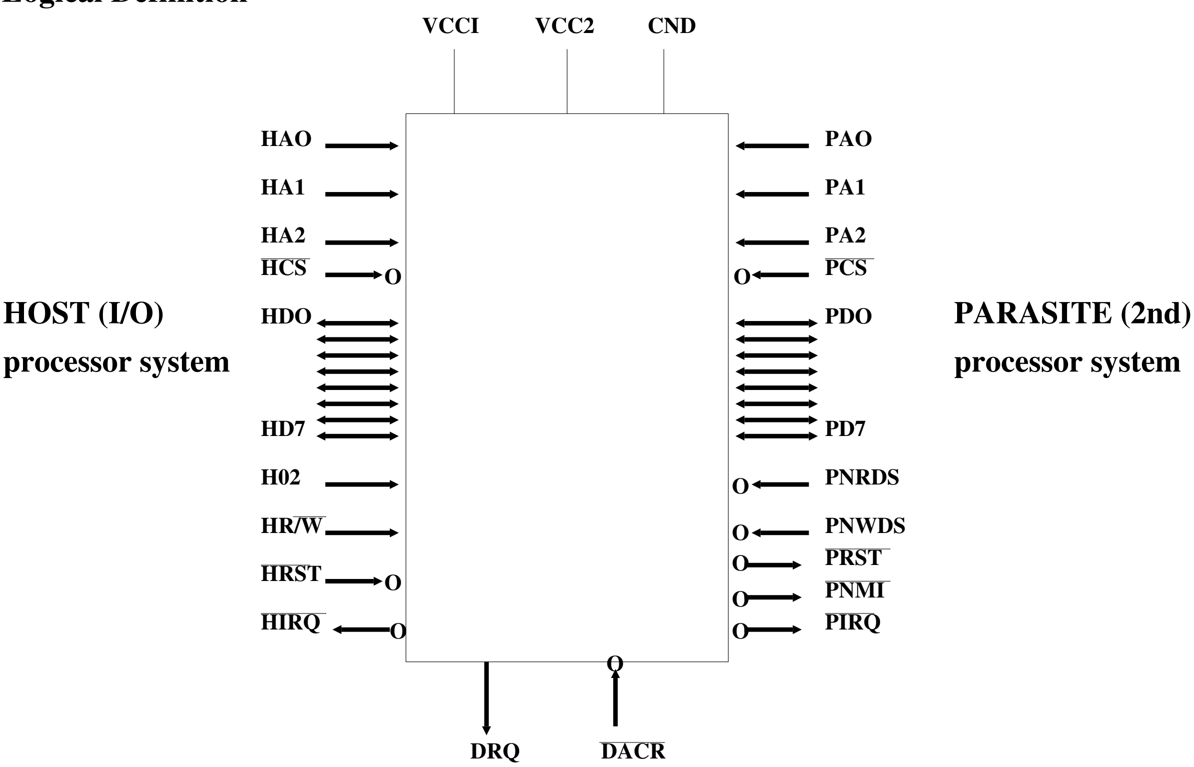
Register Organisation

A2	A1	A0	R/W or NWDS	D7	Host DO		D7	Parasite DO	
0	0	0	1	A1	F1 P V M J I Q		A1	F1 P V M J I Q	
0	0	1	1		read reg 1			read reg 1	(1)
0	1	0	1	A2	F2 x x x x x x		A2	F2 x x x x x x	
0	1	1	1		read reg 2			read reg 2	
1	0	0	1	A3	F3 x x x x x x		N	F3 x x x x x x	
1	0	1	1		read reg 3	(2)		read reg 3	(3)
1	1	0	1	A4	F4 x x x x x x		A4	F4 x x x x x x	
1	1	1	1		read reg 4	(4)		read reg 4	(5)
0	0	0	0	S	T P V M K I Q		x	x x x x x x x	
0	0	1	0		write reg 1	(6)		write reg 1	
0	1	0	0	x	x x x x x x x		x	x x x x x x x	
0	1	1	0		write reg 2			write reg 2	
1	0	0	0	x	x x x x x x x		x	x x x x x x x	
1	0	1	0		write reg 3	(7)		write reg 3	
1	1	0	0	x	x x x x x x x		x	x x x x x x x	
1	1	1	0		write reg 4	(9)		write reg 4	(10)

Notes:

1) Will clear PIRQ if register 1 was the source
2) May activate PNMI depending on M and V flags
3) May clear PNMI (see description of interrupt operation)
4) Will clear HIRQ if it was active
5) Will clear PIRQ if register 4 was the source
6) Will activate PIRQ if I = 1
7) May activate PNMI depending on M and V flags
8) May clear PNMI
9) Will activate PIRQ is J = 1
10) Will activate HIRQ if Q = 1
11) All bits marked x are insignificant and will read out as 1

TUBE Logical Definition



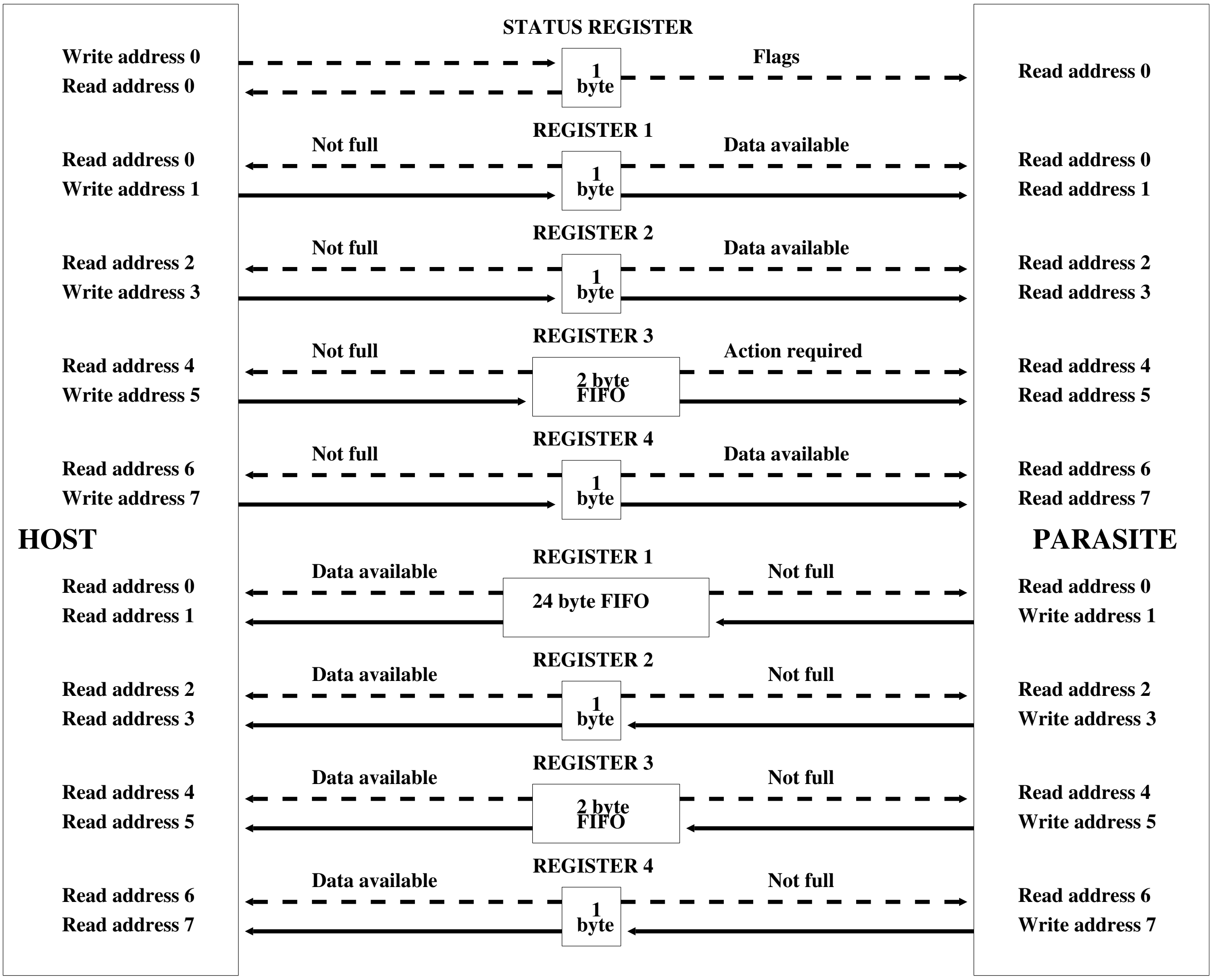
DMA controller on 2nd processor system

Description of pins:

Power supply	GND	0v reference
	VCC1	Main +5v supply
	VCC2	Secondary supply - may not be used. - must be derived from +5v through dropping resistor.
Data buses	HDo-7	8 bit data bus to host processor
	PD0-7	parasite processor
Address signals	HAo-2/PAo-2	3 register select lines from host/parasite address bus
	HCS/PCS	chip select line from host/parasite address decoding
Timing signals	H02	Host processor clock-high level signifies valid address bus
	HR/W	Host read-write line-controls direction of information flow on HDo-7
	PNRDS	Parasite read strobe (low level active)
	PNWDS	Parasite write strobe (low level active)
Interrupt lines	HRST	Clears all internal latches and initialises tube to known state - also generates PRST
	HIRQ	Interrupt to host processor
	PRST	Reset line to parasite processor
	PNMI	Non-maskable interrupt to parasite
	PIRQ	Interrupt to parasite

DMA lines	DRQ	Request for DMA transfer
	DACK	DMA acknowledge from DMA controller

Schematic diagram of Tube registers



The following tables show the relative address and type of each register in the Tube, firstly for the Host system, and secondly for the parasite system (second processor).

Table 1 Host System Registers

Address	Read
000	Status flags and Register 1 flags
001	Register 1 (24 byte FIFO read only)
010	Register 2 flags
011	Register 2 (1 byte read only)
100	Register 3 flags
101	Register 3 (2 byte FIFO only)
110	Register 4 flags
111	Register 4 (1byte read only)

Address	Write
000	Status flags
001	Register 1 (1 byte write only)
010	- - - - - -
011	Register 2 (1 byte write only)
100	- - - - - -
101	Register 3 (2 byte FIFO write only)
110	- - - - - -
111	Register 4 (1 byte write only)

Table 2 Parasite System Registers

Address	Read
000	Status flags and Register 1 flags A1 F1 P V M J I Q
001	Register 1 (1 byte read only)
010	Register 2 flags
011	Register 2 (1 byte read only)
100	Register 3 flags
101	Register 3 (2 byte FIFO read only)
110	Register 4 flags
111	Register 4 (1 byte read only)

Address	Write
000	- - - - - -
001	Register 1 (24 byte FIFO write only)
010	- - - - - -
011	Register 2 (1 byte write only)
100	- - - - - -
101	Register 3 (2 byte FIFO write only)
110	- - - - - -
111	Register 4 (1 byte write only)

INTERFACE SPECIFICATION

Title:	**Acorn Tube Software Protocol Specification**
Reference:	**SP0, 989, 900I01**
Issue No:	**08**
Author:	
Date:	**27. 05. 1986**
Design Authority:	

NOTE: All enquiries and requests for changes to this specification must be directed to the Design Authority.

DISTRIBUTION:	ACORN	H Fisher	Business Systems Dept
		A Hinchley	Communication Systems Dept
		J B Tansley	Engineering Systems Dept
		M Jenkin	Engineering Systems Dept
		A McKernan	Personal Systems Dept
		C B Turner	R & D Services Dept
	EXTERNAL	F Cockshott	Glasgow Univ.
		C Hall	TDI
		J Wein	Digital Research

Interface Specification Ref: SOtube8

1 Change History

31 Jan 1984 : Initial issue
9 Feb 1984 : Correct R4 protocols, Minor clarifications
24 Feb 1984 : Correct R4 type 0-3 protocols
6 Aug 1984 : OSWORD Parameter Counts corrected
12 Oct 1984 : Document restructure
25 Oct 1984 : Distribution List change
20 Nov 1984 : Correction of typographical errors
27 May 1986 : Addition of overview and timing details

2 Introduction

The Tube allows two processors to communicate using software protocols, it provides single or multibyte buffering and the handshake signals required.

This document describes the Tube protocols from the point of view of the second processor. In this document the following conventions are observed:

Hexadecimal numbers are preceded by &
When bit are numbered within a byte, bit 0 is the least significant bit
'Host' refers to the BBC computer
'Parasite' refers to the second processor

3 Overview

The objective of a host - parasite tube configuration is to allow th parasite to execute user programs, using the host system as a servant to take care of low level I/O tasks. In order to use the host system as a servant, the parasite must be able to control suitable routines in the host system. The standard host tube code allows the parasite to make use of the standard BBC operating system calls. Thus any parasite system, on initialisation, must be capable of using these standard OS routines through the tube, and may or may not load new routines into the host at a later stage of start up. This document therefore defines the protocols necessary for making use of these standard OS routines and coping with the results of issuing commands - for example loading files from disc across the tube into the parasite system. a parasite system also has to be able to cope with the needs of external systems such as a network - which may want to peek or poke memory in the parasite system.

The system calls form three main categories:
a) Calls on which the host system acts and makes no return of information to the parasite.
b) Calls on which the host acts and then returns a few parameters to the parasite in a non time critical fashion.
c) Calls on which the host acts and then returns or reads parameters or blocks of data in time critical fashions.
An example of category c is a load of a file from disc. The process is started by the parasite issuing an appropriate filing system call. The filing system response will result in bytes physically read from the disc having to be passed straight across the tube to the parasite.

Categories a and b are dealt with by th host and parasite using registers 1 in the parasite to host direction and 2 in both directions. register 1 in the parasite to host direction is used for OSWRCH calls (fitting category a). These types of transfer are made by each processor polling the relevant register status until it reaches the correct state for reading or writing. When it is not busy performing tasks, the host processor polls registers R1STAT and R2STAT, waiting for commands. Filing system calls (category c) are made with this type of transfer through register 2, but any time critical resulting transfer occurs under interrupt. The 'Non Interrupt Protocols' section of this document describes in detail the protocols for these types of transfer,

Time critical transfers use registers 1 or 4 in the host to parasite direction to generate IRQ's in the parasite and reading or writing of register 3 by the host to generate NMI's in the parasite. The tube does not cause interrupts in the host system, nor does the host system poll tube registers during time critical block transfers such as file loading/saving. During time critical block transfers, the host processor simply reads from or writes to register R3DATA at defined rates. Each read or write generates an NMI in the parasite system, and the parasite must service this interrupt appropriately in time for the next interrupt. Some transfers also occur without NMI's through register 3. The 'Interrupt Driven Operations" section of this document describes in detail the protocols for these types of transfer.

4 Hardware Structure

The Tube hardware provides 4 independent bi-directional communication paths. Each consists of a one byte control register and a one byte data register (which may have multibyte buffering). The characteristics of each register set are described below, in the descriptions RnSTAT refers to status register n and RnDATA refers to its corresponding data register.

4.1 Register set 1

R1DATA
- write
- read (reading clears IRQ)

R1STAT
- bit 7 data available/IRQ
- bit 6 not full

In the parasite to host direction this register set is used for the OSWRCH operating system call. The data register is a FIFO big enough to enable the longest VDU command to reside within it - thus increasing the chance that the host and parasite will achieve parallel execution.

In the host to parasite direction the data register provides a 1 byte buffer. When the host writes to it an IRQ is generated to the parasite. It is used to pass on event interrupts, such as a keypress interrupt, and the escape operation.

4.2 Register set 2

R2DATA
- write
- read

R2STAT
- bit 7 data available
- bit 6 not full

This register set is used to implement long (in machine time used) OS calls, or those which (eg RDCH) cannot interrupt the WRCH host background task - in fact, any call apart from OSWRCH. The parasite passes a byte to describe the required action. The two machines then co-operate in passing data across R2DATA until the job is done.

4.3 Register set 3

R3DATA
- write
- read

R3STAT
- bit 7 data available/nmi
- bit 6 not full

R3DATA is programmable (from the Host) to be either a 1 or 2 byte FIFO. If set as a 2 byte FIFO, both bytes have to be written to or read from to cause or clear a parasite NMI. register 3 can be programmed by the host not to cause parasite NMI's.

This register set is used fro the background task of block data transfer between the two machines (of register set 4). For higher performance applications this register may actually interface to a DMA controller.

4.4 Register set 4

R4DATA
- write (writing sets IRQ)
- read (reading clears IRQ)

R4STAT
- bit 7 data available/IRQ
- bit 6 not full/IRQ

This register set is used as a control channel for block transfers carried out across R3. The host interrupts the second processor by writing a byte describing the required action into R4DATA. The two machines then co-operate in passing data across register 4 until removal of a synchronisation byte by the parasite signals starting of transfers through register 3.

The register set is also used to initiate the passing of an error string from host to parasite. the host interrupts the parasite by writing an error code into R4DATA, the two machines then cooperate in passing the error string across R2DATA.

5 Software Protocols

Notes:
The BBC machine operates by polling the tube register for work.
In all the transactions which may generate errors it is important to realist that if the error is reported by the BBC machine under interrupt (ie it was generated by a 6502 BRK sequence), the protocol which generated the error is abandoned.

5.1 Non Interrupt Protocols

OSWRCH - Wait until R1DATA not full, write character into R1DATA.

OSRDCH - Wait until R2DATA not full, write RDCHNO (=&00) to R2DATA.
Wait for data in R2DATA, top bit of R2DATA is 6502 C@bit (validity bit).
Wait for data in R2DATA, R2DATA is 6502 A register (character read).

OSCLI - Wait until R2DATA not full, write CLINO (=&02) to R2DATA.
FOR all characters in the command string (including terminating <cr>)
DO [
    Wait until R2DATA not full, write character to R2DATA
    ]
Wait for data in R2DATA and read it.

IF this byte =&80 then code has been loaded into the second processor store as a result of the command and it should be entered at the address given by the last R4 protocol type 4 address.

OSBYTE - IF osbyteno , &80
THEN [
    Wait until R2DATA not full, write SBYTNo (=&04) to R2DATA
    Wait until R2DATA not full, write parameter for 6502@X to R2DATA
    Wait until R2DATA not full, write osbyteno to R2DATA
    Wait for data in R2DATA, read R2DATA which is 6502@X register
    ]

ELSEIF osbyteno = &82 THEN [result is machine high order address]
ELSEIF osbyteno = &83 THEN [result is low memory value]
ELSEIF osbyteno = &84 THEN [result is high memory value]
ELSE [
    Wait until R2DATA not full, write BYTENO (=&06) to R2DATA
    Wait until R2DATA not full, write parameter for 6502@X to R2DATA
    Wait until R2DATA not full, write parameter for 6502@Y to R2DATA
    Wait until R2DATA not full, write osbyteno to R2DATA
    IF osbyteno=&9D THEN RETURN from protocol (no reply)
    Wait for data in R2DATA, bit 7 of byte read is from 6502@C
    Wait for data in R2DATA, byte read is 6502@Y
    Wait for data in R2DATA, byte read is 6502@X
    ]

OSWORD - IF oswordno = &00
THEN [; Doing readline
    Wait until R2DATA not full, write RDLNNO (=&0A) to R2DATA
    Wait until R2DATA not full, write upper bound char to R2DATA
    Wait until R2DATA not full, write lower bound char to R2DATA
    Wait until R2DATA not full, write length allowed to R2DATA
    Wait until R2DATA not full, write &07 to R2DATA
    Wait until R2DATA not full, write &00 to R2DATA
    Wait for data in register2 -> response
    IF response .&7F
    THEN [

```
                        ; escape was pressed on input
                          RETURN from protocol
                        ]
                Read a <cr> terminated string from R2DATA
                  ]
        ELSE [
              Wait until R2DATA not full, write WORDNO (=&08) to R2DATA
              Wait until R2DATA not full, write oswordno to R2DATA
              Wait until R2DATA not full, write #params to send to R2DATA
              Write parameter block to R2DATA, last byte first
              Wait until R2DATA not full, write #params to recv to R2DATA
        Read bytes back from R2DATA into parameter block, last byte first
          ]
```

The number of parameters to send/receive is determined by:
IF oswordno<&14
THEN [Determine number of parameters from following tables

OSWORD number	Parameters to send	Parameters to receive
1	0	5
2	5	0
3	0	5
4	5	0
5	2	5
6	5	0
7	8	0
8	14	0
9	4	5
10	1	9
11	1	5
12	5	0
13	0	8
14	16	16
15	16	16
16	16	13
17	13	13
18	0	128
19	8	8
20	128	128

```
    ]
ELSEIF oswordno <&80
THEN  [
    # parameters to send=16
    # parameters to receive=16
      ]
ELSE  [
     # parameters determined in call specific manner
       (eg by embedding in transfer block)
      ]
```

```
OSBPUT -    Wait until R2DATA not full, write BPUTNO (=&10) to R2DATA
            Wait until R2DATA not full, Y to R2DATA (file handle)
            Wait until R2DATA not full, A to R2DATA (byte to write)
            Wait for data from R2DATA, discard it

OSBGET -    Wait until R2DATA not full, write BGETNO (=&0E) to R2DATA
            Wait until R2DATA not full, write file handle to R2DATA
            Wait for data in R2DATA, top bit of byte is 6502@C (validity bit)
            Wait for data in R2DATA, read R2DATA which is byte read from file

OSFIND -    Wait until R2DATA not full, write FINDNO (=&12) to R2DATA
            Wait until R2DATA not full, write type of open to R2DATA
            IF type=0
            THEN [
                 Wait until R2DATA not full, write file handle to R2DATA
                 Wait for data in R2DATA, read result
                   ]
            ELSE [
                 Wait until R2DATA not full, write file name string to R2DATA
                                                        (including terminating <cr>)
                 Wait for data in R2DATA, read handle from R2DATA
                   ]

OSARGS -    Wait until R2DATA not full, write ARGSNO (=&0C) to R2DATA
            Wait until R2DATA not full, write file handle to R2DATA
            Waiting for R2DATA not full,
                        [ write 4 bytes orsarg@data to R2DATA (ms byte first) ]
            Wait until R2DATA not full, write operation code to R2DATA
            Wait for data in R2DATA, read fs type from R2DATA
            Waiting for R2DATA data,
                        [ read 4 bytes osarg@data from R2DATA (ms byte first) ]
```

Note: osarg@data is the file sequential pointer or length depending on the type of OSARG call.

```
OSFILE -    Wait until R2DATA not full, write FILENO (=&14) to R2DATA
            Waiting for R2DATA not full,
                        [ write 16 byte OSFILE control block to R2DATA ]
                                    (last byte of block is written first)
            Waiting for R2DATA not full, write filename to R2DATA including terminating <cr>
            Wait until R2DATA not full, write type of transfer to R2DATA
                        (Any transfer is completed under interrupt using R3, R4)
            Wait for data in R2DATA, read R2DATA AND &7F = filing system type
            Waiting for data in R2DATA,
                        [ read back 16 byte control block from R2DATA ]
                                    (last byte of block is read first)
```

Note: The 16 byte control block has format:

0	**Load address**
4	**Execution address**
8	**Data start address or Length ***
12	**End address or attributes ***

* The contents of these fields depends on the call type

```
OSGBPB -  Wait until R2DATA not full, write GBPBNO (=&16) to R2DATA
          Wait until R2DATA not full,
                    [ write 13 byte OSGBPB control block to R2DATA ]
                              (last byte of block is written first)
          Wait until R2DATA not full, write type of transfer to R2DATA
          Waiting for data in R2DATA
                    [ read back 13 byte control block from R2DATA ]
                              (last byte of clock is read first)
          Wait for data in R2DATA, read R2DATA bit 7 is 6502@C bit
          Waiting for data in R2DATA, read 6502@A from R2DATA
```

5.2 Interrupt driven operations

In addition to these parasite initiated activities the parasite is also required to respond to interrupts from registers 1, 3 and 4.

To determine the source of an interrupt it is important to follow the following order:

1. check for register 4 interrupt
2. check for register 1 interrupt

Register 1 interrupts:

Register 1 interrupts occur only in the host to parasite direction. The interrupt service sequence is:

```
Read type byte from R1DATA IF type <0 THEN
    [; escape flag update
      Replace the escape flag with bit 6 of type
      RETURN from servicing interrupt
    ] ELSE
    [; Event signal
      Interrupt@R1@read 6502@Y event parameter
      Interrupt@R1@read 6502@X event parameter
      Interrupt@R1@read 6502@A event parameter
     ; BBC machine will now continue processing
```

```
                     ; any other actions to service event can be take
                    ]

Where Interrupt@R1@read is:
   UNTIL data@ready@in@R1
   DO  [
           IF data@ready@in@R4 THEN CALL R4@interrupt@service
          ]
    RETURN read R1DATA


Register 4 Interrupts:

Read type byte from R4DATA IF type<0
   THEN [; BBC machine is reporting an error
             Wait for data in R2DATA, read and discard it
             Wait for data in R2DATA, read error number form R2DATA
             Read a zero byte terminated string from R2DATA
            ]
    ELSE
           [; Type is a command to initialise for register 3 block transfer
             Wait for data in register 4, read Claimer@identity* from R4DATA

* For details of the identity numbers see Appendix A

             CASE type OF
             [
               0  :  ; Single byte transfer parasite to host.
                    Read 4 byte base address for transfer from R4DATA msb first
                    Set NMI routine for this transfer type
                    Wait for and remove synchronising byte from R4DATA

               1  :  ; Single byte transfer host to parasite
                     Read 4 byte base address for transfer from R4DATA msb first
                     Set NMI routine for this transfer type
                    Wait for and remove synchronising byte from R4DATA

                2  :  ; Double byte transfer parasite to host
                      Read 4 byte base address for transfer from R4DATA msb first
                      Set NMI routine for this transfer type
                      Wait for and remove synchronising byte from R4DATA

                3  :  ; Double byte transfer host to parasite
                      Read 4 byte base address for transfer from R4DATA msb first
                      Set NMI routine for this transfer type
'                     Wait for and remove synchronising byte from R4DATA

                4  :  ; No transfer (pass address host to parasite only)
                     Read 4 byte address from R4DATA msb first
                     Wait for data in R4DATA, discard it
```

```
    5  :  ; No transfer (filing system release)

    6  :  ; 256 type transfer parasite to host without interrupt
          Read 4 byte base address for transfer from R4DATA msb first
          Wait for data in Register 4, discard it
          Transfer 256 bytes to host, via R3DATA
          Write a byte into R4DATA; To stop unwanted ints on host

    7  :  ; 256 byte transfer host to parasite without interrupt
          Read 4 byte base address for transfer from R4DATA msb first
          Wait for data in Register 4, discard it
          Transfer 256 bytes from host via R3DATA
]
RETURN ; From the interrupt
```

Notes:

For types 0-3: As soon as the synchronising byte is remove register 3 transfer requests (NMI's) will start to occur. When the interrupt occurs 1 or 2 bytes are transferred (depending on the current mode).

A release (type 5) is a guarantee that no more register 3 NMI's will occur for the current transfer.

Register 3 Transfer Timings

During such an operation as loading a file into the parasite system from disc, data has to be rapidly passed through the tube at instants dictated by a physical process - in this case, reading from a disc. for this reason such transfers are made via register 3, which may be programmed to cause parasite NMI's. Because these NMI's occur very rapidly, there are constraints on the timings with which the parasite must respond to such NMI's.

After the parasite processor has read the synchronisation byte from register 4, the host processor will wait for at least the length of the initial delay below (zero for host to parasite direction) before transferring the first byte of data. Thus for transfers in the parasite to host direction, this initial delay is the time within which the parasite must place the fist byte (or pair of bytes) of data in register 3 after removing the synchronising byte. For NMI transfers in the host to parasite direction, this delay is zero - the parasite processor must be ready to cope with register 3 NMI's as soon as it has removed the synchronisation byte from register 4. For type 7 transfer, not using NMi's the host will write the first byte without delay, but the parasite cannot receive a byte immediately from R3DATA as it has to test the data available bit in R3STAT. The parasite has to read this first byte in R3DATA before it is overwritten by the next host write. Assuming the host code is as below, this allows 15.5 uS to read the first byte:

```
        JSR     &406        ; Initialise tube -routine returning when parasite reads
                            ; synchronising byte
        LDY     0           ; 1 uS
.loop   LDA     (host), Y   ; 2.5 uS
        STA     TPORT       ; 2 uS
        NOP                 ; 1 uS
        NOP                 ; 1 uS
        NOP                 ; 1 uS
        INY                 ; 1 uS
```

```
        BNE     loop        ;  1.5 uS
```

(1 + 2.5 + 2 + 1 + 1 + 1 + 1 + 1.5 + 2.5 + 2 = 15.5)

The service time is the maximum time that the parasite has to process each subsequent transfer. The host code is a simple loop reading or writing data, and so the parasite must be capable of inputting or outputting data fast enough for this loop. For transfers of type 1 to 3, the host writing to or reading from register 3 a byte (or pair of bytes) causes an NMI in the parasite. thus the service time is an upper bound on the allowed time for te NMI service routine. Similarly, for type 6 and 7 transfers, the service time is the maximum time for the parasite to transfer each byte to or from R3DATA.

Transfer type	Transfer direction	Initial delay	Service time
0	P to H	24 uS	24 uS
1	H to P	0 uS	24 uS
2	P to H	26 uS	26 uS/pair
3	H to P	0 uS	25 uS/pair
6	P to H	19 uS	10 uS
7	H to P	0 uS	10 uS

6 Startup Protocol

The Startup sequence for the second processor is:

```
Use the OSWRCH mechanism to write out a startup message
Send a zero byte to host via R1DATA to terminate it
Wait for data in R2DATA
;  during this wait a load may occur from the host
;  using R4/R3 block transfer protocols
IF data=&80 THEN execute from the address given in the R4 type 4 transfer
```

APPENDIX A: Filing System Claimer Identities

When a filing system claims the R3/R4 resource in the host its identity is passed to the second processor as part of te R4 startup protocol. The identity codes are not related to filing system numbers.

Filing System	Claim Identity Used
Tape	0
DFS	1
NFS	2
NFS	3
ADFS	4