Specification for Implementation of Dragon/Tandy 6809 Based Network

Written by Jon Bird
Contributions by Oliver Broad, Jim Hart

Issue 3 - August 1996
Version 4 (5V CMOS)

Specification for the Implementation of 6809 Based Network V4.0

1. Introduction/Overview

This document describes the hardware and software protocol for a Dragon based network. In all, this is the fourth attempt at networking Dragons/CoCos attempted by us and this document has been up-issued/re-issued to reflect the continuing progress of the project.

1.1 Revision History

Issue 1, August 1991: Initial Issue prior to hardware and software design.
Issue 2, October 1992: Issued upon completion of DNOS V1.1 DragonDOS based network operating system.

1.2 Network Description

The network is designed to implement a single server/multiple station configuration. The current solutions provide the following scenarios:

1.2.1 DragonDOS based network - October 1992

A single network server exists running a DragonDOS based disk operating system (or equivalent) sharing out it's four floppy disks, and printer. Each machine on the network (including the server) has a dedicated network card fitted. The server program (DNOS SV1.2) is bootable from disk and provides a dedicated server machine. Each network station contains the network operating system on EPROM (DNOS V1.1) which provides a sub-set of DragonDOS type commands to interface to the network. A number of other services are available such as print sharing, network messeging etc.

1.2.2 OS9 Extension - August 1996

The existing network platform is extended to allow OS9 to run across the network. This is achieved by the use of an extra Dragon providing the OS9 functionality. The network file server and OS9 server are tied together via a fast parallel link. An OS9 Network Overlay (NETOS9) running on the DNOS server intercepts OS9 network calls and passes them via the parallel link to the OS9 machine for execution. All 64K capable machines can BOOT to OS9 using the normal method, once into the system the new network drivers provide the functionality. ALL devices available on the server will be accessable to the network if required. The OS9 server machine is not dedicated to the network and can still be used if required.

1.2.3 DragonDOS OS9 Services - August 1996

A third option is available with the OS9 extensions - removal of disks and the DragonDOS interface from the DNOS server. Via the use of a DOS extender (DOSPlus 4 only at present) all DragonDOS disk accesses can be directed to the OS9 machine therefore negating the need for additional drives and interfaces.

2. Hardware Definition

2.1 Network Cards

This network is to be implemented via the use of dedicated network cards for

each machine on the network. A 'standard' network card design has evolved which interfaces to the Dragon computer via two 16 way IDC connectors, however it is feasable to embed a network card into a larger design layout as required. For the purpose of this document however, the standard plug in card is assumed.

Each station on the system will have one of these cards fitted. During development, the network cards are built on stripboard. Each card is constructionally identical, apart from the server which has no software memory fitted. The interface to the Dragon is via 2 16 way IDC sockets (1 on server cards), comprising bus & control signals. The card can be connected directly to the Dragon's cartridge port through a suitable connector. No other logic is required, and the card can use the IO memory mapped via the P2~ decode signal. On the file server, additional hardware is required to map the card in with the DOS and as such the IO address is configurable in software. The network interface is a 2 wire cable connectable via a 3.5mm socket.

Each card will consist of the following:

* MC6850 ACIA - serial chip to drive the network

* Direction logic - direction select for the card TX/RX.

* Network ID - 8 DIL switches to provide a network ID for the card

* Address decoders & network buffers

On network stations:

* 8K Program Memory - 8K EPROM containing the network software
(during development uses 8K SRAM or E^2PROM)

The network card appears to the Dragon as 4 bytes of IO, comprising the 6850's Control/Status register, Data register and the Network Direction bit & network station ID. On network stations, the 8K of Program Memory appears at $C000:

IO Base#	
0	MC6850 Command/Status Register
1	MC6850 Data Register
2	WRITE: Bit 0: Network Direction 0 = RX 1 = TX READ: Network ID
3	As byte 2.

The IO Base is $FF40 on network stations, and is software configurable on the server.

2.2 OS9 Parallel Link

The parallel link required to tie the DNOS server with the OS9 server is achieved by either a dedicated 8-bit parallel link (usually via a MC6821 PIA chip) or through the parallel printer ports (back to back) or a combination of both. The standard STROBE/ACK transfers are used. Overlay software exists for using the printer port on the DNOS server and a dedicated PIA on the OS9 server.

2. Software

The following section defines the software requirements for both server and network station.

2.1 Software Layer 1 - Network Packet Drivers

Two key operations directly interface to the network I/O area (defined in section 1) implementing the transmission and receipt of a network packet. These operations are used in all communications to and from the network and are essentially the same for both DragonDOS network stations/server and OS9.
The two operations are:

NETOUT - Write a data packet
NETIN - Read a data packet

The packet structure is defined below:

$00 * 2 HEADER BYTES
$FF SYNC BYTE
1 MACHINE NUMBER BYTE
1 COMMAND REQUEST BYTE
1 ERROR BYTE
1 STREAM NUMBER BYTE
1 BLOCK LENGTH BYTE
3 BYTE CURRENT FILE POINTER
1-256 DATA BYTES
1 CHECKSUM BYTE

The calling software supplies all the information to be sent apart from the header/sync bytes, machine number (which is obtained from the network card hardware) and checksum byte via a data structure which is passed to/from the packet drivers. For the most part, the only mandatory part of a packet is the Command Request byte which is used by the network server to identify the operation that is being requested - the other information may be unused or re-defined depending on the purpose of the network packet.

2.1.1 NETWORK BLOCK IN COMMAND (NETIN)

The NETIN command's operation is to pull in a network data block (defined above). In addition, a 24 bit number is to be passed to the command specifying a time out delay, and where 0 represents no timeout. NETIN will wait for the required time period for data arriving on the ACIA. The delay is purely based on a decrementing counter loop and as such will vary dependent upon the clock speed of the processor. If none arrives within the specified time interval, an error code indicating timeout is placed in the B register and the command aborted. Once the first byte of the block ($FF) is received, a shorter timeout is used on all subsequent blocks, allowing a faster return to the calling routine should the serial stream stop in mid flow. NETIN will also check the station ID within the network block, and if it fails to match the hardware ID, then the recieve is aborted, and the command looks for the first sync byte again. When all the incoming data has been received, the 256 byte data buffer is checksummed and compared against the transmitted checksum. If any serial corruption occurs, or a checksum failure occurs, an appropriate error code is returned in the B register. The Zero bit of the CC register is clear if no

error occured.

2.1.2 NETWORK BLOCK OUT COMMAND (NETOUT)

The NETOUT command waits until the ACIA's TX register is empty, and sends the network block out byte by byte. NETOUT will also perform the checksum on the data block.

2.2 Software Layer 2 - Basic Network Protocol

The following section defines the basic protocol used to implement the network.

2.2.1 Network Server

The network server will operate on a polling operation. The server will use NETOUT to send to each station in turn a request for a command. It will then call NETIN with a small timeout delay, awaiting a station's response to a request. If NETIN times out, or returns an error the polling loop continues. If a valid request is received, the Network Command Byte is extracted and used as an index into a jump table of valid commands. The operation will then perform the required command, set the top bit of the Command byte and return a data packet via NETOUT. Control is then passed back to the network loop.

2.2.2 Network Station Command Processor

On the network stations, a software interface is provided to perform the network handshake, therefore negating the need for the user to call NETIN and NETOUT directly. The calling procedure will provide a network packet to be sent to the Command Procesor. The processor will then call NETIN to look for the attention request packet sent from the server targetted to this station. When one is detected, the supplied packet is then passed to NETOUT for transmission. The NETIN operation is then called again to await the response from the server. If a corrupted or invalid response is received (for example another request packet possibly indicating the server has discarded our request) the entire process is repeated a number of times. Atfer the maximum allowed attempts has been exceeded, the command processor will exit with the last NETIN error code in the CPU's B register.
If a valid response packet is received, it is then used to overwrite the calling operation's supplied packet. Prior to returning, the Error Byte in the packet is copied into the CPU's B register setting the Zero bit appropriatly.

2.3 Software Layer 3 - User Interface

Sections 2.1 & 2.2 defined the basic operation of the network. This section goes on to describe the developed software both for DragonDOS and OS9 which implements this protocol and provides links into the operating system to make use of it. This will start off with the basic DragonDOS implementation and then move onto the OS9 layer which has been added. Although the DragonDOS network is not strictly relevent for implementing the OS9 network, since it is built on top of the DragonDOS one (in fact, there is no existing OS9 network server as such) this section should not simply be ignored.

2.3.1 The Dragon Network Server

The basic operation of the server has already been defined. The software to

implement it runs under the DragonDOS (or compatible) system and consists of aproximately 1K of position dependent 6809 assembler. The source is for the DSAM assembler. The server code itself comprises the NETIN, NETOUT and primary server code. It also comprises a background print manager whereby print requests from the network are spooled onto (DragonDOS) disk files then printed in small sections so as not to halt the network. The remainder of the server code is dedicated to implementing the first 16 entries in the jump table which provide the network interface into DragonDOS and provides full network access to DragonDOS floppy disks. The remainder of the vector table is free to be used for other network operations. The initial 16 entries can also be used if DragonDOS support is not required with one exception - command request $0C is used as the Network Attention packet id and cannot be used.
The full list of DragonDOS supported calls is listed in section 2.3.2.

The network server software primarily utilises graphics page 2 starting at $0C00 in memory through to $1200 as it's workspace which comprises most of the user defined settings which affect the operation of the server. A supporting BASIC program is utilised to start the server up, this includes setting the ACIA address in the workspace area, defining the available floppy drives (this information is read from a disk configuration file) and loading up any server extenders that may be implemented. Finally, it executes the server assembler code.

Server extenders are separate assembler modules, which add to the capabilities of the main server program. They utilise the server workspace to determine the location of the network vector table and add new function calls to it (for example a network 'chat' system). The main server program will jump to these vectors which will then perform the operation requested of them before jumping back into the main server code via a predefined vector to send the response network packet back. All the network packet information and buffers is held in the predefined workspace of the main program.

2.3.2 DragonDOS Network Station

The code for a network station consists of aproximatly 5K of 6809 assembler held in 3 separate source code files for the DREAM assembler called DNOS V1.1. This code is designed to support access to DragonDOS disks across the network. It is held in EPROM in the cartridge memory area of a Dragon as part of the network card, normally where DragonDOS resides starting at $C000.

This code provides two levels of software: an interface to the Network Command Processor to issue all the required DragonDOS disk calls to the network and a set of BASIC commands identical to those used under DragonDOS to provide BASIC disk IO functions across the network.

2.3.2.1 Network Interface Level

As well as comprising the standard packet drivers and network command interface, it also provides the ability to perform the basic DragonDOS commands normally held in the DOS indirect jump table at [$C004] across a network. This will enable standard DOS commands to be implemented easily on the stations and allow any machine code programs to work correctly. In order to achieve this, each command has a valid Network Command number, and the relevent paramaters to the command passed within the data packet. The Network Command Processor operation will be used to issue the command.
The following table details each command, its appropriate jump table address

and the Command byte number.

Command	Jump Table	Net ID No.	Comments
Network Command Proc.	[$C004]	-	replaces disk op. proc
Ptr. to Network Pkt	[$C006]	-	replaces disk op. blk
Copy file details	[$C008]	-	local command
Get dir entry/cpy Ctrl	[$C00A]	1	
Create directory entry	[$C00C]	2	
Get file length	[$C00E]	3	
Close all files drive	[$C010]	-	Not applicable
Close a file	[$C012]	4	
Load a file block	[$C014]	5	Multiple calls
Write buffer to file	[$C016]	6	Multiple calls
Count free sectors	[$C018]	7	
Kill a file	[$C01A]	8	
Set file protection	[$C01C]	9	not return Y register
Rename a file	[$C01E]	A	not return Y register
Get directory record	[$C020]	B	not return U register
Find free buff/read sec	[$C022]	-	not applicable
Copy dir sects 20 to 16	[$C024]	-	not applicable
Network Attention	-	C	issued by server
Retry block	-	D	no longer used
Read absolute sector	[$C026]	E	additional features
Write absolute sector	[$C028]	-	not applicable
Verify absolute sector	[$C02A]	-	not applicable
Format disk	[$C02C]	-	not applicable
Base of error table	[$C02E]	-	
COPY command	-	F	needed for server

Notes

Commands marked as 'not applicable' will return a code 8 (?FC error) in the B register when called. These are commands which will not be implemented due to possible serious data loss of other users dta. The read sector call has been implemented to allow the BASIC BOOT command, and SREAD which allows for S/W directories in many programs. A variant on the command exists, to allow access to the direct track/sector reads on the disk command processor on the server.

The exact interface to these calls is defined in any DragonDOS or compatible user guide.

The Network Command Processor & command block structures supercede the DOS command processor at $C004-7.

A DNOS User's guide is available which defines the exact interface to these routines and gives examples of using the Command Processor.

2.3.2.2 DragonDOS BASIC Interface

The final layer of software on the network stations is the BASIC Interface implemented by providing a DragonDOS set of commands. These commands make use of the jump table at $C004 to perform network commands, and essentially involve a re-write of the DOS BASIC commands to sit on the network software. The process involves setting up a new BASIC token & vector jump table for the

commands, providing the patches into the BASIC RAM hooks and writing the commands. The BASIC interface will emulate a DOS ROM on boot up, providing a title screen & performing required initialisation of data and the serial chip.

The commands are/will be implemented according to the following schedule:

Network BASIC Commands Schedule

SRAM versions:

PRELIM = Minimal Network orientated commands (eg. LOAD, SAVE, KILL etc.)
(Train development begins)
V0.0 = Support of simple DOS commands (eg. FRE$, HIMEM etc.)
V1.0 = Filing commands implemented. Additional DOS commands.

EPROM versions:

FULL = First EPROM version. Should be a cut down DOS minus FREAD type comms.
PLUS = Future development to bring NOS fully compatible with DOSPLUS.

STATUS CODES:

0 = Not implemented.
A = Partly implemented.
B = Provides some DOS compatibilty.
C = Fully DOSPLUS compliant.
? = Uncertain at this time.

UPDATE HISTORY

14.02.92: First edition

22.03.92: PRELIM version completion date.
Updation of command status codes.

25.05.92: V0.0 completion date.
Updation of command status codes.
Enable sector read calls on server & net stations.

31.08.92: V1.0 completion held up by technical problems on file input. Updated to show present command status. Lower level re-design will occur before FULL version commences. Source code in DREAM format.

13.11.92: FULL version complete (includes under V1.0 not implemented last issue
& lower level re-design). Testing to continue to remove any bugs.
EPROM target for pre '93.
No development for PLUS version planned.

COMMAND	Status	PRELIM	V0.0	V1.0	FULL	PLUS
AUTO	Yes	0	0	0	C	C
BACKUP	Not applicable	0	0	0	0	0
BACKUP DIR	Not applicable	0	0	0	0	0
BEEP	Yes	0	C	C	C	C
BOOT	Yes, through ABS sect	0	0	C	C	C

CHAIN	Pushed out for PLUS	0	0	0	0	C
CLOSE	Yes	0	0	B	B	C
COPY	Under V1.0, not tested	0	0	A	A	C
CREATE	No current plans	0	0	0	0	?
DIR	Yes	B	B	B	C	C
DRIVE	Yes	C	C	C	C	C
DSKINIT	Not applicable	0	0	0	0	0
EOF	Yes, complete for FULL	0	0	0	B	C
ERL	Yes	0	C	C	C	C
ERR	Yes	0	C	C	C	C
ERROR GOTO	Yes	0	C	C	C	C
FLREAD	No current plans	0	0	0	0	?
FREAD	No current plans	0	0	0	0	?
FRE$	Yes	0	C	C	C	C
FREE	Yes	C	C	C	C	C
FROM	No current plans	0	0	0	0	A
FWRITE	No current plans	0	0	0	0	?
HIMEM	Yes	0	C	C	C	C
INPUT	Yes, complete for FULL	0	0	0	B	C
KILL	Yes	B	B	B	C	C
LINE INPUT	Yes, complete for FULL	0	0	0	B	C
LOAD	Yes	B	B	B	B	C
LOC	Yes	0	0	A	B	C
LOF	Yes	0	C	C	C	C
MERGE	Yes for FULL	0	0	0	B	
OPEN	Yes	0	0	A	B	C
PRINT	Yes	0	0	A	B	C
PROTECT	Yes	C	C	C	C	C
RENAME	Yes	B	B	B	B	B
RESTORE	Yes	0	C	C	C	C
RUN	Yes	0	C	C	C	C
SAVE	Yes	B	B	B	B	C
SREAD	Yes (S/W DIRs)	0	0	C	C	C
SWAP	Yes	0	C	C	C	C
SWRITE	Not applicable	0	0	0	0	0
VERIFY	Yes, dummy command	0	0	C	C	C
WAIT	Yes	0	C	C	C	C
LPRN	Extra:set to local PRN	0	0	C	C	C
NPRN	Extra:set to net PRN	0	0	C	C	C

Last Updated: 13.11.92

PRELIM complete:Current status reflected.
V0.0 complete
V1.0 complete: Minus data file input support
FULL complete: 1. Data file input fixed
2. Network low level re-design complete - retry available on all
commands.
3. NO FUTURE DEVELOPMENT PLANNED IN THE NEAR FUTURE.

Notes

Certain commands will not be implemented, since they will involve unauthorised access by network users. These commands include DSKINIT, BACKUP and sector

write calls. The indirect jump table also does not support these commands. A final EPROM version, targetted for the end of 1992, should support a majority of DOSPLUS commands, with the exception of FREAD type commands & possibly MERGE, CHAIN & COPY. Addition of these commands is an overall long term aim in order to produce a fully DOSPLUS compatible network operating system. All commands, will communicate to the Server calls to the indirect jump table. The exception to this is the COPY command.

Two new commands are added to the BASIC command set to provide network printing facilities - LPRN (set to local [parallel port] printer) and NPRN (set to network printer). In network print mode calls to the printer device (#-2) are intercepted by the file IO related BASIC RAM hooks to pass the required data across the network using a reserved filename. The file server software will then spool the data to disk and use it's background print mode to print the file during network inactivity.

3. The OS9 Network

One of the aims of adding OS9 network capability was that the DragonDOS system could be run in parallel with it and also that the OS9 Network Server could continue to be used by another user if desired (ie. non-dedicated). During 1995 over several months various ideas and attempts were made to do this, with the resulting system just about completed mid 1996. Since the two systems are combined, it is worth reading up on the DragonDOS side of things prior to referring to this section alone.

3.1 Network Station

The key components of the network station, once OS9 is up and running on it are the Network File Manager (NBF), Device Driver (NET6850) and device descriptors. All the device descriptors are technically the same - they are all NBF classes of descriptor and contain the same information apart from the device name. A descriptor needs to be present on the network station for each device you wish to access that is present on the OS9 network server. For example, if the network server supports a hard disk with a descriptor H0 then in order to access this device a network descriptor also called H0 must exist on the network station.

The Network File Manager provides the packaging up of the 12 standard OS9 IO requests into standard Network packets (defined in section 2.1). For OS9, Network Command bytes 128 ($80) onwards have been used:

OS9 Requests:

```
NetI$Cre  equ $80 OS9 I$Create
NetI$Opn  equ $81 OS9 I$Open
NetI$Mdr  equ $82 OS9 I$MakDir
NetI$Cdr  equ $83 OS9 I$ChgDir
NetI$Del  equ $84 OS9 I$Delete
NetI$Sk   equ $85 OS9 I$Seek
NetI$Rd   equ $86 OS9 I$Read
NetI$Wt   equ $87 OS9 I$Write
NetI$Rln  equ $88 OS9 I$Readln
NetI$Wln  equ $89 OS9 I$Wtln
NetI$Gst  equ $8A OS9 I$GetSta
NetI$SSt  equ $8B OS9 I$SetSta
```

```
NetI$Cls  equ $8C OS9 I$Close
```

A new definitions file (NBFDEFS) has been created which contains these equates and other network definitions, including the packet structure.

For the most cases, NBF simply passes the request off across the network for processing by the appropriate device on the OS9 server, collects the response and returns to the caller.

The NET6850 device driver implements the packet drivers NETIN and NETOUT and provides an interface to the Network Command Processor which NBF usses to issue all network requests. The Command Processor is accessable via either the READ or WRITE entries into the device driver. The device driver also implements a new GetStat call for issuing User Defined packets across the network. This allows normal OS9 programs to gain direct access to the network for specific functions. One utility already exists which makes use of this function - the NETIME program which obtains the system time from the OS9 server and updates the station's own clock with it.

Along with these, new SYSGO and INIT modules exist to set default network devices, and automatically call the OS9 Login utility. There is also a replacement OS9 BOOT module.

3.1.1 OS9 Network BOOT

The OS9 BOOT module is responsible for reading and loading the OS9Boot file from disk which contains the necessary modules (apart from the OS9 Kernal) with which to start OS9 on a workstation (eg. IOMAN, RBFMAN, KBVDIO, SHELL etc).

On a Dragon, under DragonDOS the BASIC command BOOT is used to start this procedure. This operation reads sectors 3 to 18 from track 0 off a floppy disk into memory starting at location 9728 and then calls the code starting at location 9730. On an OS9 boot disk, the OS9 Kernel (OS9p1 and OS9p2), INIT and BOOT modules are placed into these sectors. After a piece of memory relocation code, the BOOT module is called by the OS9 Kernel to load the bootstrap file from disk. This is then accomplished by talking directly to the disk controller to read the necessary sectors from disk. The structure, and format of this boot file is defined in the OS9 System Programmers Guide.

The network OS9 is started in a similar manner. DNOS supports a BOOT command which performs an identical function to the DragonDOS one. It issues DNOS Read Logical Sector requests across the network to get the boot sectors in. The Network OS9 boot module contains cut down versions of NETIN, NETOUT packet drivers and Network Command Processor which it usses to continue to issue DNOS Read Logical Sector requests to read in the OS9Boot file. Note that all these requests are part of the DragonDOS network and are made to disks held on the DragonDOS machine (command byte $0E) - they do not form part of the OS9 network at all.

Once the bootstrap is completed, the SYSGO module is called and the system is started.

3.2 OS9 Network Server

There is no OS9 Network server as such, the existing DNOS Network Server is

utilised attached to the OS9 machine via a fast transfer link. Software exists to use the parallel printer port of a Dragon Network Server to an MC6821 PIA device driver on an OS9 machine.

The DNOS Server usses two Server extenders (NETOS9 and NETUDF). NETOS9 maps the OS9 IO vectors issued by NBF into the server's jump table and provides operations for handling them. NETUDF supports any User Defined calls implemented by the system, in this case only handling the NETIME call. These two modules exist as DragonDOS binary files and are loaded by the Server's BASIC program.

Another piece of software is invoked either by the Server's BASIC program or is booted into higher memory by a dedicated bootstrap piece of software. This also is a DragonDOS binary which is called DOSSERV. Since this is not directly concerned with the network this code is not covered in detail. At a very raw level though it provides:

* A packet transfer mechanism similar to the network packets across the 8-bit parallel printer port.

* Jump vectors for the 12 OS9 IO calls taking the same input/output parameters as the OS9I$ calls. These are passed as parallel packets across the parallel link for processing and the results retrieved and passed back to the caller.

In the longer term DOSSERV may turn into a fully fledged DOS extender allowing DragonDOS files to be stored as OS9 files on the OS9 server. The DOSSERV program normally runs from high RAM on a D64 at $E000.

The NETOS9 and NETUDF extenders simply translate the network packets into calls to the DOSSERV program, translates the results back into network packets and jumps back into the main server program.

3.3 OS9 Parallel Link Server

The OS9 Server machine is linked to the network server through a dedicated MC6821 PIA and provides a new file manager (SBF), device driver (PIA21) and descriptor (P1) with which to access it. This setup provides fairly 'raw' data transfers allowing blocks of data to be read/written to/from the port and GetSta/SetSta calls for configuring the PIA's port direction.

The OS9 server is controlled by the Net_Serv program, usually invoked as a background task to allow the machine to continue functioning normally. Net_Serv connects to the parallel link and awaits a parallel link packet. Upon receipt of a packet, it identifies the type of packet then forks (or wakes up) a module to process it (either OS9NetIO for IO requests or OS9NetUdf for User Defined Requests). These service modules retrieve the packet, process it (normally by calling an appropriate OS9 call with information extracted from the packet) and return the information across the parallel link. They then exit (or go to sleep) prior to sending a wakup message to the Net_Serv program.

As it stands, a separate OS9NetIO module is forked for every station on the network. After performing an IO request it goes to sleep. This ensures that each station has it's own current data/execution directory because at present NBF cannot keep track of these directories. Problems can arise because of this, such as child processes changing directory on the network station also

affecting the parent's directory. A good example of this is the PWD command which recursivly changes directory to the root of the disk always leaving the user's current directory on the root. This is a definate problem which will need to be resolved at some point.

The OS9NetUdf program only performs the NetTime operation and is forked every time requested and terminates after use.

Dragon Network Card - Description

The Dragon network cards were developed on a 58*29 stripboard matrix. Connection is made via 2*16w IDC connectors which contain the processor's data bus, control lines, and the required address lines. The boards consist of some address decode logic, MC6850 serial chip, clock, buffers and an 8K EPROM. The card present on the file server has one of the IDC connectors omitted, and no EPROM as the software is on disk. Note: a ~ symbol denotes an active low signal.

The main IDC connector caries the processor data bus (D0-D7), clock (E), read/write (R/W~), address lines (A0 & A1), power lines(+5V, 0V) and address decode line (AD~). On network stations AD~ should be connected to the P2~ line on the cartridge port, on the server this will need to be derived by some external address decoder. Each network card requires 4 bytes in the Dragon's I/O map. A dual 2 to 4 line decoder (74HC139) provides the decoding for the board. The 'A' side of the chip is enabled by the AD~ signal, and depending on the state of the A1 address line, will either enable the 6850 (if A1 is low=lower 2 bytes) or the 'B' side of the decoder (if A1 is high=upper 2 bytes). The 'B' side will enable an octal buffer if 1 of the upper 2 bytes is read, outputing the state of a bank of 8 DIP switches on the data bus, or a quad latch chip for data direction.

The data bus and associated control lines are routed to the required chips on the board (6850, 2732 EPROM & 74HC541 Quad buffer for network ID). The MC6850 ACIA is used as the driving output for the network. The chip interfaces to the processor via the databus, R/W~, E, and A0. The chip is enabled via the 74HC139 on the CS2~ input, the remaining CS lines being tied to ground. The chip is clocked by a crystal oscillator, connected to a 74HC4060 divider. On existing boards an 8Mhz crystal is used, with the 2nd lowest divider used to feed into the TX CLK & RX CLK on the 6850 giving a speed of aproximatly 15Kbps. Data to be output to the network is fed out of the TX line into a tri-state buffer (74HC126). This is controlled via the output of a latch chip (4042). When set high, data is sent through the buffer to the network cable. This data is also routed through another buffer on the 126 (permanently enabled) into the RX line for diagnostic purposes (loop-back). To receive data from the network, the TX output buffer is disabled via the quad latch and data is sent through the RX buffer. The 126 has a diode in series with the power rail to prevent power scavenging when the board is powered down. The remaining buffers are unused.

A 4042 provides the quad buffer. This is connected to the lower half of the databus, but only 1 latch is used (on D0), the remaining are spare. The chip is enabled by the 139 and the output is used to select network direction, a low being input and a high output.

On network stations there is an additional IDC connector carrying the remaining address lines required for the EPROM (A2-A12), and the chip select (R2~ line). The remaining required lines are taken from the first IDC connector (E, R/W~, D0-D7). The EPROM maps into $C000-$E000, and then network card port addresses at $FF40-$FF43 ($FF42 & $FF43 are the same).

On the server, the network card will be mapped into wherever the decoder controlling the AD line puts it. A simple decoder can be composed of another 74HC139. Connect the P2~ line from the cartridge port to pin 1 of the decoder chip (EA~), and the A4 address line to pin 2 (A0a). Pin 3 (A1a) needs to be tied to 0V. Now pin 4 (0a~) provides the decode line for the DOS cartridge (the P2~ line connecting directly to the DOS cartridge must be disconnected at replaced with this output) and pin 5 (1a~) provides the network decode and connects to AD~. The disk controller will now be activated whenever an address in the range $FF40-$FF4F is present on the address bus, but when an address in the range $FF50-$FF5F is on the address bus the network card will be activated. Therefore, the card's IO will be mapped at $FF50-$FF53 ($FF52 & $FF53 are the same).