X3T9.2/92-080R0

DS-Links

Technical information presented to X3T9.2, 27 April 1992
SGS-THOMSON Microelectronics

With a number of standards activities within ANSI and IEEE on different forms of Serial Interconnect, SGS-THOMSON has been invited to present information on the DS-Link serial interconnect. DS-Links have been developed initially for processor to processor communication, but are equally appropriate for some of the specialised applications such as disk drives, disk arrays, and communications systems. Particular benefits of DS-Links result from the possibility of building low-cost packet-routing switches, with many ports and with minimal transmission delay.

This document provides technical information on DS-Links.

Questions about this document should be addressed to:

Forrest Crowell
SGS-THOMSON Microelectronics Inc.
200 East Sandpointe, Suite 120
Santa Ana
California
92707
USA

Phone: (714) 957 6018
Fax: (714) 957 3281
email: crowellf@isnet.inmos.com

or

Pete Moore
INMOS Limited
1000 Aztec West
Almondsbury
Bristol
BS12 4SQ
England

Phone: (+44) 454 616616
Fax: (+44) 454 617910
email: moorep@inmos.co.uk

Contents

X3T9.2/92–080R0

DS–Links

Technical information presented to X3T9.2, 27 April 1992
SGS–THOMSON Microelectronics

1 Introduction

SGS–THOMSON has been developing inter–processor serial interfaces since 1979 and has shipped millions of serial communication 'links' as an integral part of the transputer family of microprocessor devices. This 'OS–Link', as it is known, provides a physical point–to–point connection for a software channel between two processes, each process running in a separate processor; they are full–duplex, have an exceptionally low implementation cost and an excellent record for reliability and fault–tolerance. Indeed, the OS–link has been used in almost all sectors of the computer, telecomms and electronics markets. Many of these links have been used without transputers, or with a transputer simply serving as an intelligent DMA controller for other processors. However, they are now considered to be a mature technology, and have, by todays standards, a relatively low speed, running at 20 Mbits/s.

As part of the continuous development program with the SGS–THOMSON organisation, a new type of serial interconnect, known as the DS–link, has evolved, with the full benefit from the experience gained with OS–links, both within the corporation, and by our customers. A major feature of the DS–Link is that it provides a physical link over which any number of software (or 'virtual') channels may be multiplexed; these can either be between two directly connected devices, or can be in any number of different devices, if the links are connected via (packet) routing switches. Indeed, the inherent routing capability of the DS–link, coupled with SGS–THOMSON's experience in router design, makes it particularly good for communication both within a system and between systems. Other features include error detection, which has been added to detect and locate the most likely errors, while transmission speed has been increased to 100Mbits/s, with 200Mbits/s planned and further enhancement possible.

Although DS–Links have evolved from processor to processor communication, they are equally appropriate for some of the specialised applications such as disk drives, disk arrays, or communication systems. Indeed, almost all of the sub–systems within those equipments contain processors, and therefore the model of processor to processor communication is a natural fit, and may be more appropriate than some alternative interfaces.

The benefits of the process to process model of communication used by the DS–Links are in terms of performance (particularly of the system rather than each device), fault–tolerance, and cost.

This document contains detailed descriptions of the DS–link and the integral routing technology, plus some initial suggestions about how these would fit the requirements of the SCSI community. This release of the document concentrates on these aspects of links, rather than on physical aspects, because in general the physical aspects are covered by standard, commercially available, components. One exception is the proposed inter–equipment cable connector, which is shown in the final Appendix. An outline of the distances that can be covered by DS–Links is shown in the figure below.

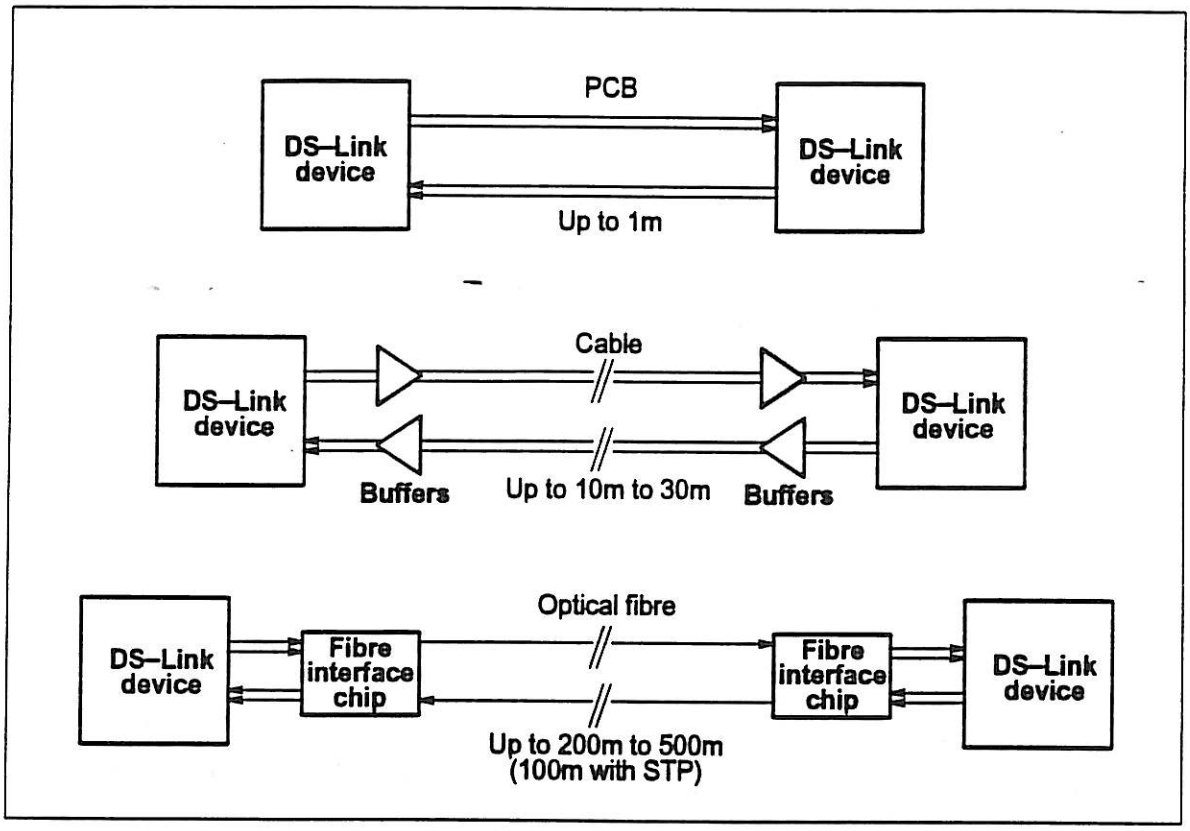


Figure 1.1

2 Communication links

DS–Links provide a simple and regular way of interfacing many devices. Routing devices enable this communication to take place across a network, even between devices that are not directly connected.

2.1 Using links between devices

DS–Links can be used to implement point to point communication between devices. This allows networks of arbitrary size and topology to be constructed. Point to point links have many advantages over bus based communications in a system with many devices:

- There is no contention for the communication mechanism, regardless of the number of devices in the system.
- There is no capacitive load penalty as more devices are added to the system.
- The communications bandwidth does not saturate as more communicating devices are added to the system. Rather, the larger the number of devices, the greater the total communications bandwidth of the system.
- Removing the bus as a single point of failure improves the fault–tolerance of the system.

For small systems, a number of DS–Links on each device can provide complete connection between a few devices. By using additional message routing devices, networks of any size can be built with complete connection between all devices.

Each connected pair of DS–Links provides a full–duplex, flow–controlled connection operating at a programmable speed of upto 100MBits/s or more.

2.2 Channel communication

A model of communication which can be implemented by DS–Links is based on the ideas of communicating sequential processes. The notion of 'process' is very general, and applies equally to pieces of hardware and pieces of software. Each process can be regarded as a black box with internal state, which can communicate with other processes using communication channels. Each channel is a point to point connection between two processes. One process always inputs from the channel and the other always outputs to it. Communication is synchronized: the first process ready to communicate waits until the second is also ready, then the data is copied from the outputting process to the inputting process and both processes continue.

2.2.1 Virtual channels

Each OS–Link of the original transputers implemented only two channels, one in each direction. To map a particular piece of software onto a given hardware configuration the programmer had to map processes to processors within the constraints of available connectivity. The problem is illustrated in figure 2.1 where 3 channels are required between two processors, but only a single link connection is available.

One possible solution is the addition of more links. However this does not really solve the problems, since the number of extra links that can be added is limited by VLSI technology. This 'solution' does not address the more general communication problems in networks, such as communication between non-adjacent processors, or combining networks in a simple and regular way.

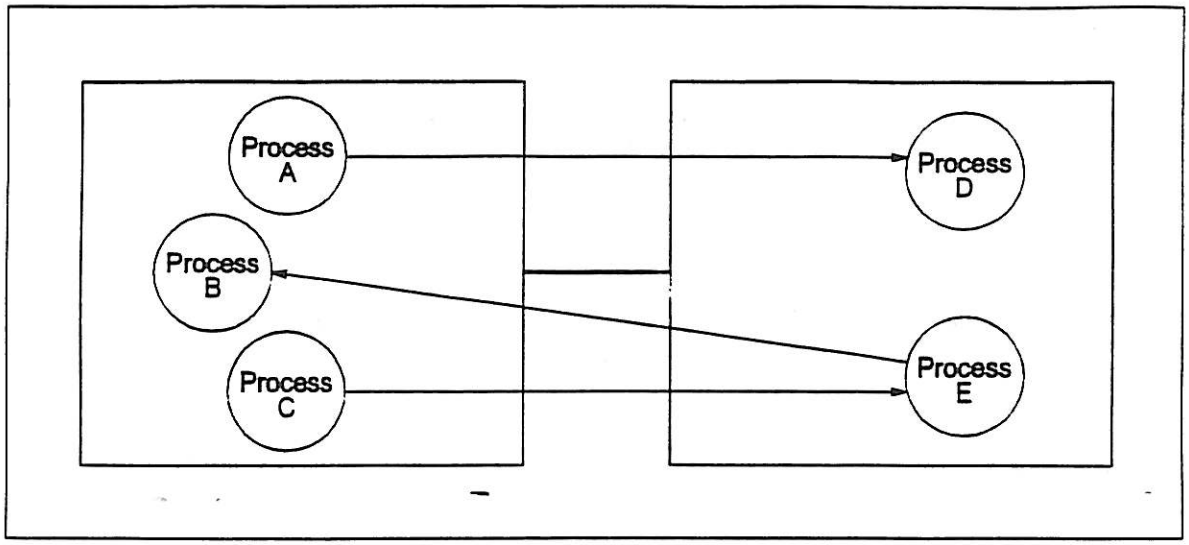


Figure 2.1 Multiple communication channels required between devices

A better solution is to add multiplexing hardware to allow any number of proces s to use each link, so that physical links can be shared transparently. These channels which share a link are known as 'virtual channels'.

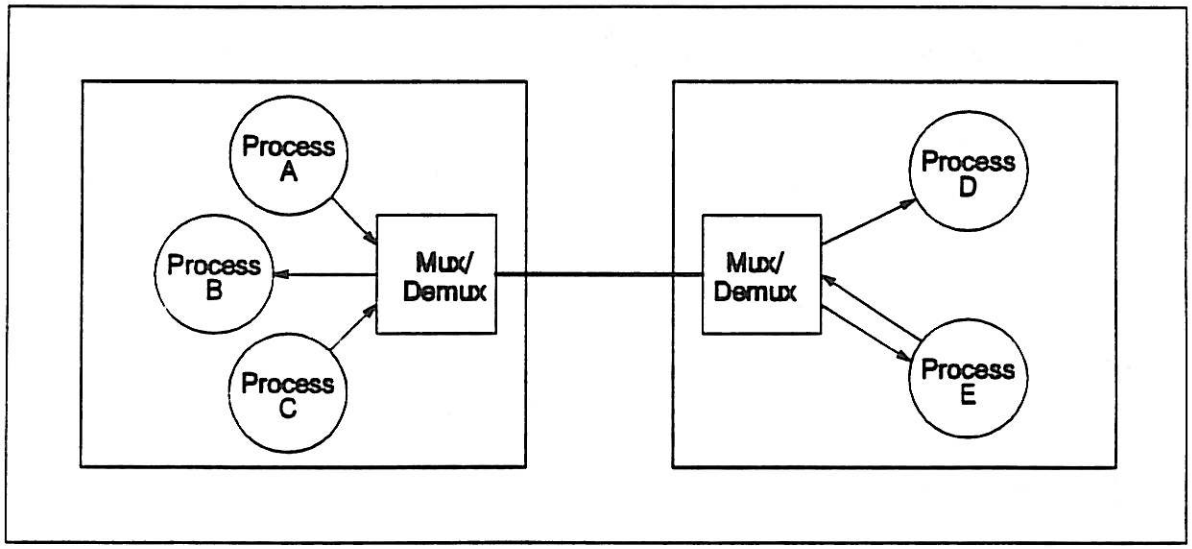


Figure 2.2 Shared DS–Links between devices

Virtual links

Each message sent across a link is divided into packets. Every packet requires a header to identify its destination process. Packets from different messages are interleaved on the link. There are a number of advantages to this, as detailed below.

- Channels are, generally, not busy all the time therefore the multiplexing can make better use of hardware resource by keeping the links busy with messages from different channels.
- Messages from different channels can effectively be sent concurrently – the device does not have to wait for a long message to complete before sending another.

90

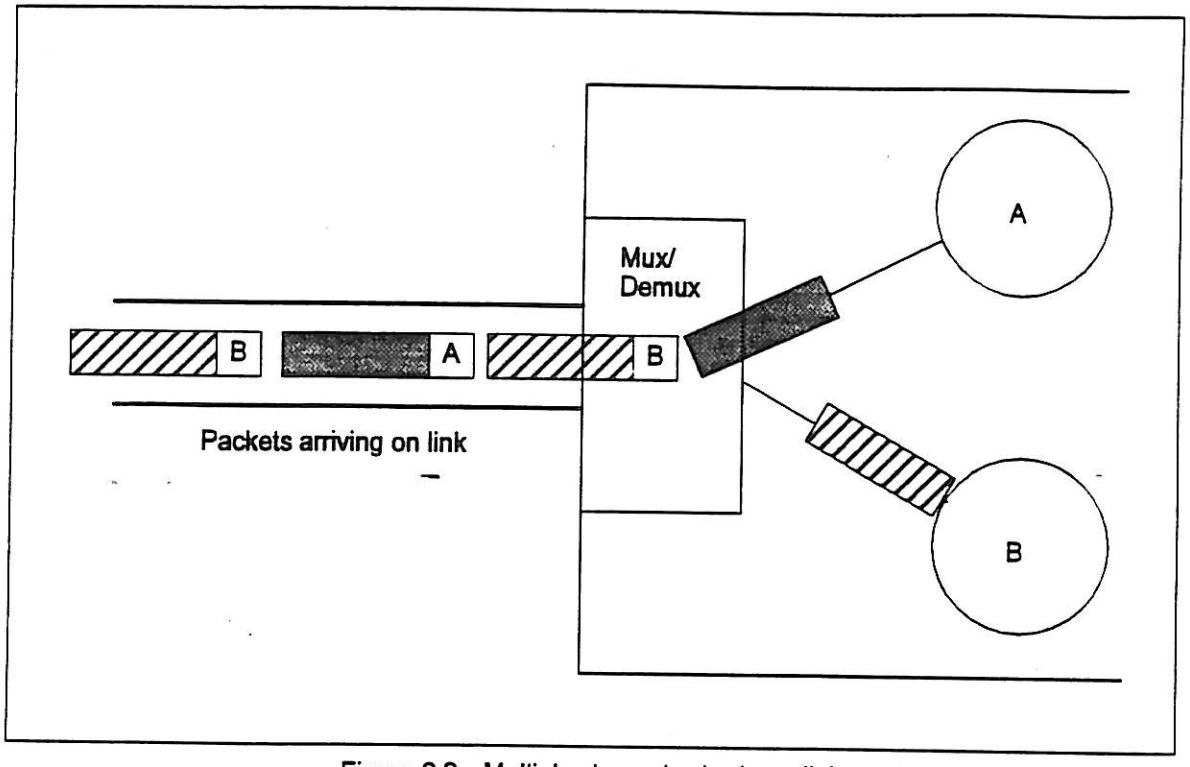


Figure 2.3 Multiple channels sharing a link

Virtual channels are always created in pairs to form a 'virtual link'. This means it is not necessary to include a return address in packets, since acknowledgements are simply sent back along the other channel of the virtual link.

2.3 Levels of link protocol

As with any communications system, the links can be be described at a number of levels with a hierarchy of protocols. At the highest level a message consists of the data sent down a channel from one process to another. Any type of data or message can be sent in this way. This communication is synchronized; it will not take place until both processes are ready and the two processes will not continue until the message transfer is complete.

2.3.1 Packet level protocol

In order to transfer a message from one device to another, it is sent as one or more packets. This allows packets from a number of different channels to be interleaved on the same link. Each packet is acknowledged by the receiving device, to maintain synchronized communication and to limit the amount of buffering required.

Every packet has a header defining the destination address followed by the data bytes and, finally, an 'end of packet' or 'end of message' token. See figure 2.4.This simple protocol supports messages of any length; the receiving device knows when each packet and message ends without needing to keep track of the number of bytes received. It also maintains synchronization at the message level.

A packet can contain up to 32 data bytes. If a message is longer than 32 bytes then it is split up into a number of packets all, except the last, terminated by an 'end of packet' token. The last packet of the message, which may contain less than a full 32 bytes, is terminated by an 'end of message' token.

Shorter messages can be sent in a single packet, containing 0 to 32 bytes of data, terminated by the 'end of message' token. With this protocol zero length messages can be sent, allowing efficient synchronization between devices.

Packet acknowledgements are sent as zero length packets terminated with an 'end of packet' token. This type of packet can never occur as part of a message because a zero length data packet must always be the last, and only, packet of a message, and will therefore be terminated by an 'end of message' token.

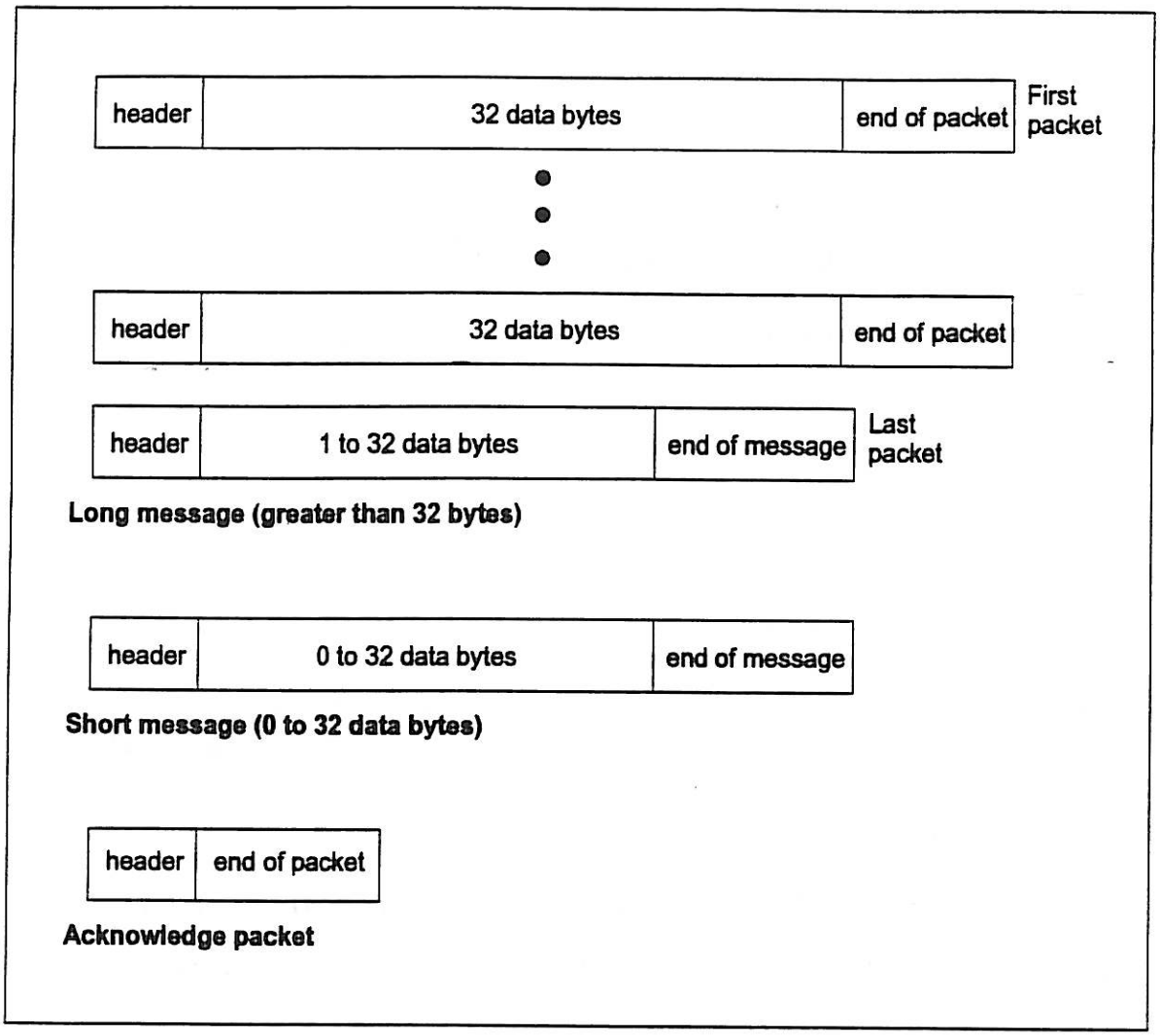


Figure 2.4 High Level protocol

2.3.2 Token level protocol

In order to support the packet level protocol described above, a lower level protocol is needed for encoding tokens which may contain a data byte or control information. Each token has a parity bit plus a control bit which is used to distinguish between data and control tokens. In addition to the parity and control bits, data tokens contain 8 bits of data and control tokens have two bits to indicate the token type (e.g. 'end of message').

Flow control token	FCT	P100
End of packet	EOP	P101
End of message	EOM	P110
Escape token	ESC	P111
Null token	NUL	ESC P100

Table 2.1 Control token codings

The parity bit in any token covers the parity of the data or control bits in the previous token, and the data/control flag in the same token, as shown in figure 2.6. This allows single bit errors in the token type flag to be detected. **Odd** parity checking is used. To ensure the immediate detection of errors null tokens are sent in the absence of other tokens. The coding of the control tokens is shown in table 2.1.

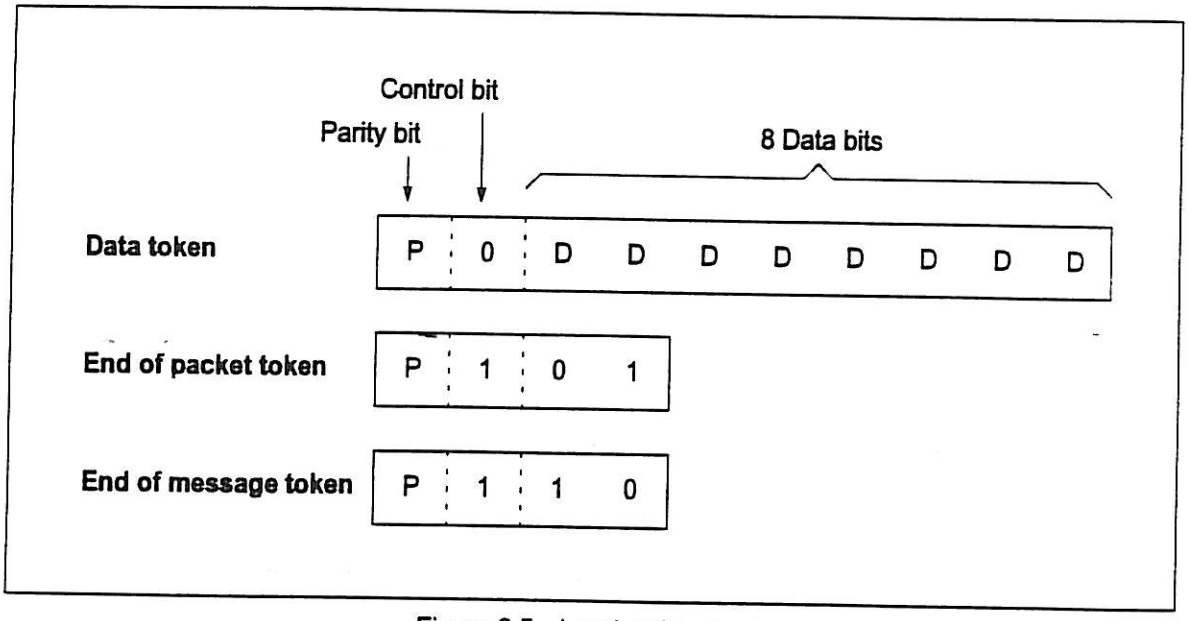


Figure 2.5 Low level protocol

Token level flow–control

The DS–Link protocol separates the functions of flow control and process synchronization. Flow control is done entirely within the link module and process synchronization is built into a higher–level packet system.

Token–level flow control is performed in each link module, and the additional flow control tokens used are not visible to the higher–level packet protocol. The token–level flow control mechanism prevents a sender from overrunning the input buffer of a receiving link. Each receiving link input contains a buffer for at least 8 tokens (more buffering than this is in fact provided). Whenever the link input has sufficient buffering available to consume a further 8 tokens a flow control token (FCT) is transmitted on the associated link output, and this FCT gives the sender permission to transmit a further 8 tokens. Once the sender has transmitted a further 8 tokens it waits until it receives another FCT before transmitting any more tokens. The provision of more than 8 tokens of buffering on each link input ensures that in practice the next FCT is received before the previous block of 8 tokens has been fully transmitted, so the token–level flow control does not restrict the maximum bandwidth of the link.

Note that token–level flow control is imposed on a device–to–device basis across each physical link, whereas packet–level flow control is performed end–to–end, and message synchronization is performed process–to–process.

Note that the link module regulates the flow of data items without regard to the packets that they may constitute. At any instant the data items buffered by a link module may form part or all of one or more consecutive packets.

Token level flow–control greatly simplifies the higher levels of the protocol, since it prevents data from being lost due to buffer overflow and so removes the need for re–transmission of packets unless errors occur.

2.3.3 Bit level protocol

To achieve the speed required, and to support the virtual channel protocol, a new, simple link standard has been implemented. .The new links have four wires for each link (a data and strobe line in each direc-

tion) and are known as DS–Links. Each DS pair carries tokens and an encoded clock. Figure 2.6 shows the format of data and control tokens on the data and strobe wires. All signals are TTL compatible.

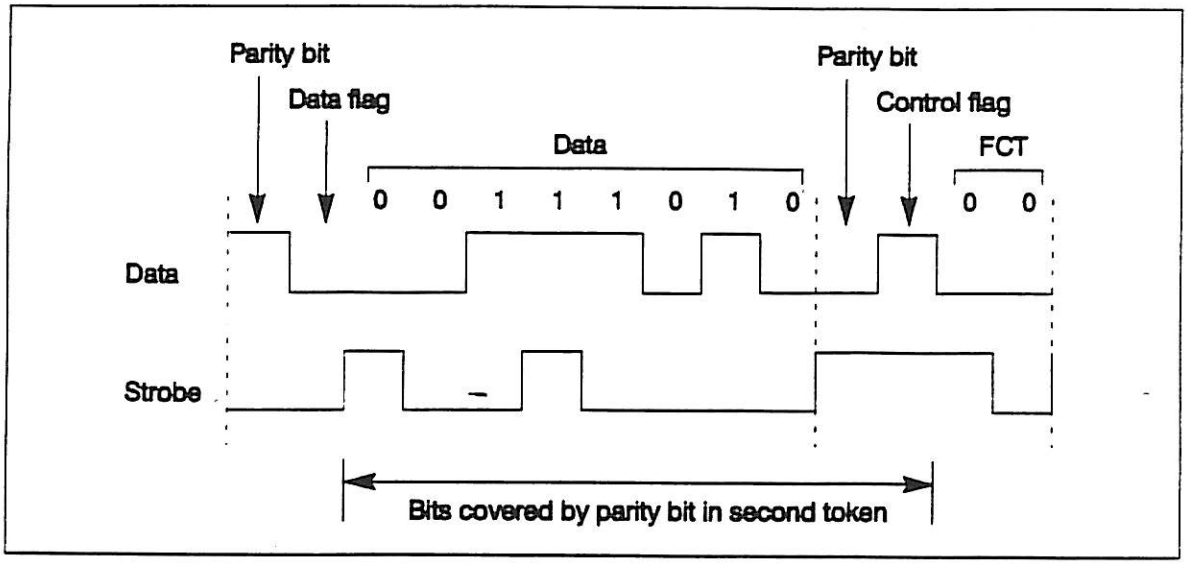


Figure 2.6 Link data format

The links are asynchronous; the receiving device synchronizes to the incoming data. This means that DS–Links 'autobaud'; the only restriction on the transmission rate is that it does not exceed the maximum speed of the receiver. It also simplifies clock distribution within a system, since the exact phase or frequency of the clock on a pair of communicating devices is not critical.

2.4 Errors on links

The DS–Links are designed to be highly reliable within a single subsystem and can be operated in one of two environments, determined by the **LocalizeError** bit in each link.

In applications where all connections are on a single board or within a single box, the communications system can be regarded as being totally reliable. In this environment errors are considered to be very rare, but are treated as being catastrophic if they do occur. If an error occurs it will be detected and reported. Normal practice will then be to reset the subsystem in which the error has occurred and to restart the application.

For other applications, for instance when a disconnect or parity error may be expected during normal operation, an higher level of fault–tolerance is required. This is supported by localizing errors to the link on which they occur, by setting the **LocalizeError** bit of the link to 1. If an error occurs, packets in transit at the time of the error will be discarded or truncated, and the link will be reset automatically.

2.4.1 Errors detected

The DS–Link token protocol allows two common types of error to be detected. Firstly the parity system will detect all single bit errors at the DS–Link token level, and secondly, because each output link, once started, continues to transmit an uninterrupted stream of tokens, the physical disconnection of a link can be detected.

Disconnection errors

If the links are disconnected for any reason whilst they are running then flow control and token synchronization may be lost. In order to restart the link it is therefore necessary to reset both ends to a known flow control and token synchronization point.

Disconnection is detected if, after a token has been received, no tokens are seen on the input link in any 1.6 microsecond window. Once a disconnection error has been detected the link halts its output. This will

subsequently be detected as a disconnect error at the other end, and will cause that link to halt its output also. It then resets itself, and waits 12.8 microseconds before allowing communication to restart. This time is sufficient to ensure that both ends of the link have observed disconnection and cycled through reset back into the waiting state. The connection may now be restarted.

Parity errors

Following a parity error, both bit–level token synchronization and flow control status are no longer valid, therefore both ends of the link must be reset. This is done autonomously by the DS–Link using an exchange–of–silence protocol.

When a DS–Link detects a parity error on its input it halts its output. This will subsequently be detected as a disconnect error at the other end, and will cause that link to halt its output also, causing a disconnect to be detected at the first end. The normal disconnect behavior described above will then ensure that both ends are reset (irrespective of line delay) before either is allowed to restart.

3 Network communications

The use of DS–Links for directly connecting devices has already been described. The new link protocol not only simplifies the use of links between devices but also provides hardware support for routing messages across a network.

3.1 Message routing

The system described previously packetizes messages to be sent over a link and adds a header to each packet to identify the virtual channel. These headers can also be used for routing packets through a communication system connecting a number of devices together. This extends the idea of multiple channels on a single hardware link to multiple channels through a communications system; a communications channel can be established between any two devices even if they are not directly connected.

Routers

The routing components in a network can be separated from the processing elements. Messages can be passed from one device, through any number of routing devices, to the destination device. This creates a temporary path through the routing system for that message so there still appears to be a single channel directly connecting one device with another.

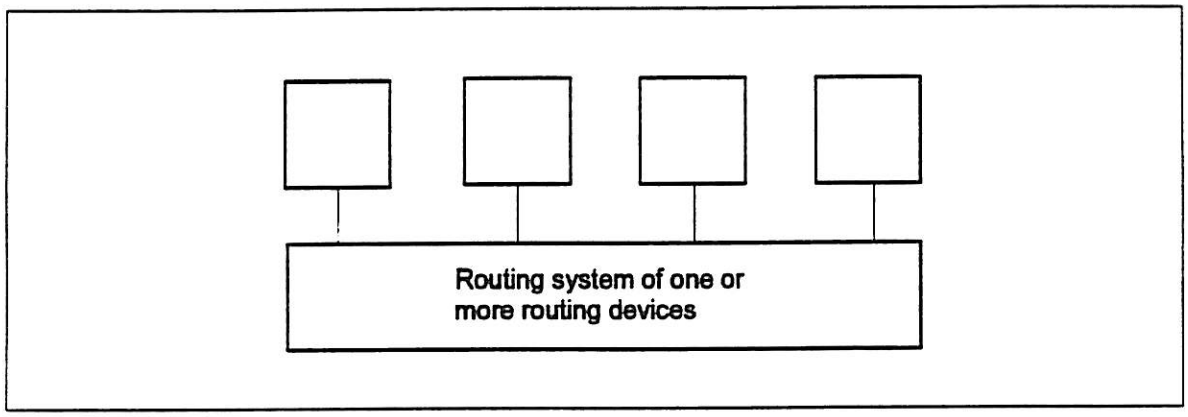


Figure 3.1 A routing system

As a packet arrives on a link, the destination address must be inspected before the outgoing link can be determined. The time before the output link can be determined is therefore proportional to the address length. Further, the address itself must be transmitted through the network and consumes network bandwidth. It is therefore important that this address be as short as possible, both to minimize latency and maximize bandwidth.

The router needs to arbitrate between packets which arrive at the same time and have to be sent out of the same link. Ideally, it should start to output the packet as soon as possible; i.e. immediately after the output link is determined, provided that the link is not already in use by another packet. This keeps the latency through the network small, in contrast to a typical packet switching network which uses a 'store and forward' algorithm in which each packet is read into a buffer, the address information is decoded and then the packet is sent out. The delay that would be introduced by this is unacceptable in a high–performance network. Also the amount of buffering needed would make a single chip implementation of a large routing switch impractical.

Separating routers and processors

There are a number of advantages to keeping the communications devices and processing elements separate in a system. Processing devices can be directly connected where appropriate, which avoids the silicon costs and extra routing delays (however small) in a system that does not require routers. Also, the design of the routing devices and processing elements can be optimized for their different roles. For exam-

ple, the routing component can have a larger number of links than would be possible if the two devices were integrated. Having a routing device with many links means that large networks can be built with a small number of routers, minimizing cost and latency and maximizing bandwidth. This approach also allows the construction of scaleable architectures where the communications throughput and processing power can be balanced.

Note, however, that the architecture allows low–valency routers to be integrated with other devices if required.

Parallel networks

Because the new link architecture allows all the virtual channels of a device to use a single link, complete, system-wide connectivity can be provided by connecting just one link from each device to the routing network. This can be exploited in a number of ways. For example, two or more networks can be used in parallel to increase bandwidth, to provide fault–tolerance, or as a 'user' network running in parallel with a physically separate 'system' network.

3.2 An example routing device: the IMS C104

An important benefit of using serial links is that it is easy to implement a full crossbar in VLSI, even with a large number of links. The use of a crossbar allows packets to be passing through all links at the same time, making the best possible use of the available bandwidth.

If the routing logic can be kept simple it can be provided for all the input links in the router. This avoids the need to share the hardware, which would cause extra delays when several packets arrive at the same time. It is also desirable to avoid the need for the large number of packet buffers commonly used in routing systems. The use of small buffers and simple routing hardware allows a single VLSI chip to provide efficient routing between a large number of links.

Wormhole routing

The IMS C104 (Appendix D) includes a full 32 x 32 non-blocking crossbar switch, enabling messages to be routed from any of its links to any other link. In order to minimize latency, the switch uses 'wormhole routing', in which the connection through the crossbar is set up as soon as the header has been read. The header and the rest of the packet can start being transmitted from the output link immediately. The path through the switch disappears after the 'end of packet/message' token has passed through. This is illustrated in figure 3.2. This method is simple to implement and provides very low latency as the entire packet doesn't have to be read in before the connection is made.

Minimizing routing delays

The ability to start outputting a packet while it is still being input can significantly reduce delay, especially in lightly loaded networks. The delay can be further minimized by keeping the headers short and by using fast, simple hardware to determine the link to be used for output. The IMS C104 uses a simple routing algorithm based on interval routing (described in section 3.3.1).

Because the route through each IMS C104 disappears as soon as the packet has passed through and the packets from all the channels that pass through a particular link are interleaved, no single virtual channel can monopolize a route through a network. Messages will not be blocked waiting for another message to pass through the system, they will only have to wait for one packet.

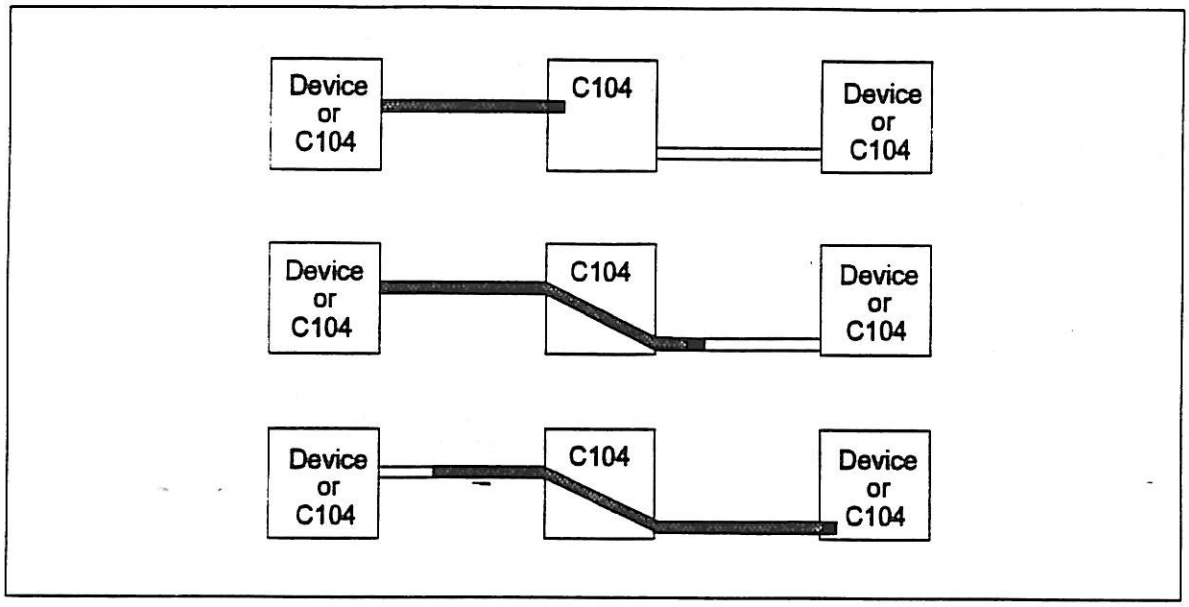


Figure 3.2 Packet passing through IMS C104

3.2.1 Using IMS C104s

A single IMS C104 can be used to provide full connectivity between 32 devices. IMS C104s can also be connected together to build larger switches connecting any number of devices.

The IMS C104s that the packets pass through do not need to have information about the complete route to the destination, only which link each packet should be sent out of at each point. Each of the IMS C104s in the network is programmed with information that determines which output link should be used for each header value. In this way, each IMS C104 can route packets out of whichever link will send it towards its destination.

The IMS C104 can implement *grouped adaptive routing*. Sets of consecutive numbered links which are connected in parallel to the same next device can be configured to be equivalent, so that a packet routed to any link in the set will be sent from any free link of the set. This achieves improved network performance in terms of both latency and throughput. This can be done because subsequent stages of routing or reception of the packet depend only on its header, not on the link along which it happens to arrive.

Header deletion

An approach that simplifies the construction of networks is to provide two levels of header on each packet. The first header specifies the destination device (actually, the output link from the routing network), this header is removed as the packet leaves the routing system. This exposes the second header which tells the destination device which process (actually, which virtual channel) this packet is for. To support this, the IMS C104 can route packets of any length. Any information after the initial header bytes used by the IMS C104 is just treated as part of the packet, even if it is going to be interpreted as a header elsewhere in the system. The IMS C104 can set any output link to do header deletion, i.e. to remove the routing header from the front of a packet after it been used to make the routing decision. The first part of the remaining data is then treated as a header by the next device that receives the packet.

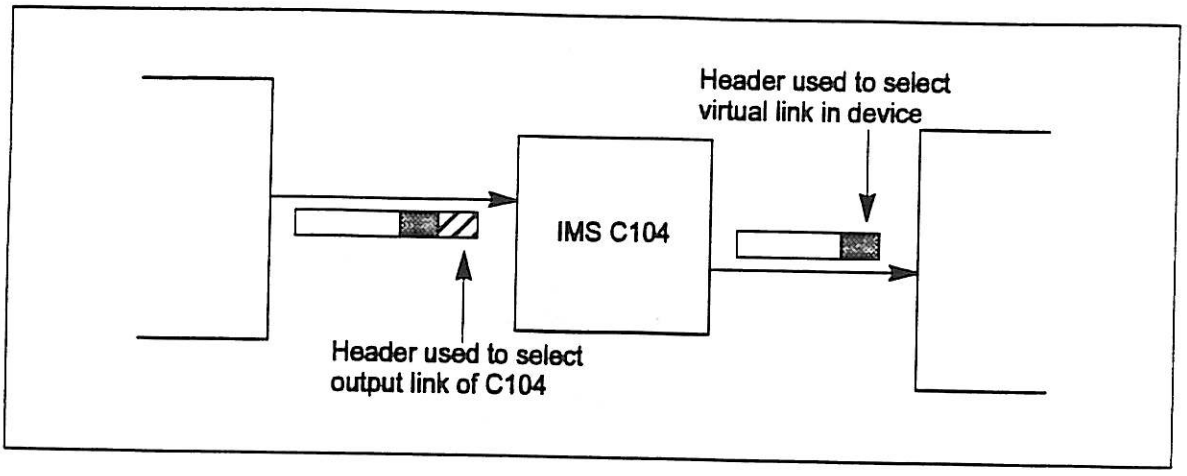


Figure 3.3 Header deletion

As can be seen from figure 3.4, by using separate headers to identify the destination device and a channel within that device, the labelling of links in a routing network is separated from the labelling of virtual channels within each device. For instance, if the same 2 byte header were used to do all the routing in a network, then the virtual channels in all the devices would have to be uniquely labelled with a value in the range 0 to 64K. However, by using two 1 byte headers, all the devices can use virtual channel numbers in the range 0 to 255. The first byte of the header will be used by the routing system to ensure that the packets reach the appropriate device before the virtual channel number is decoded.

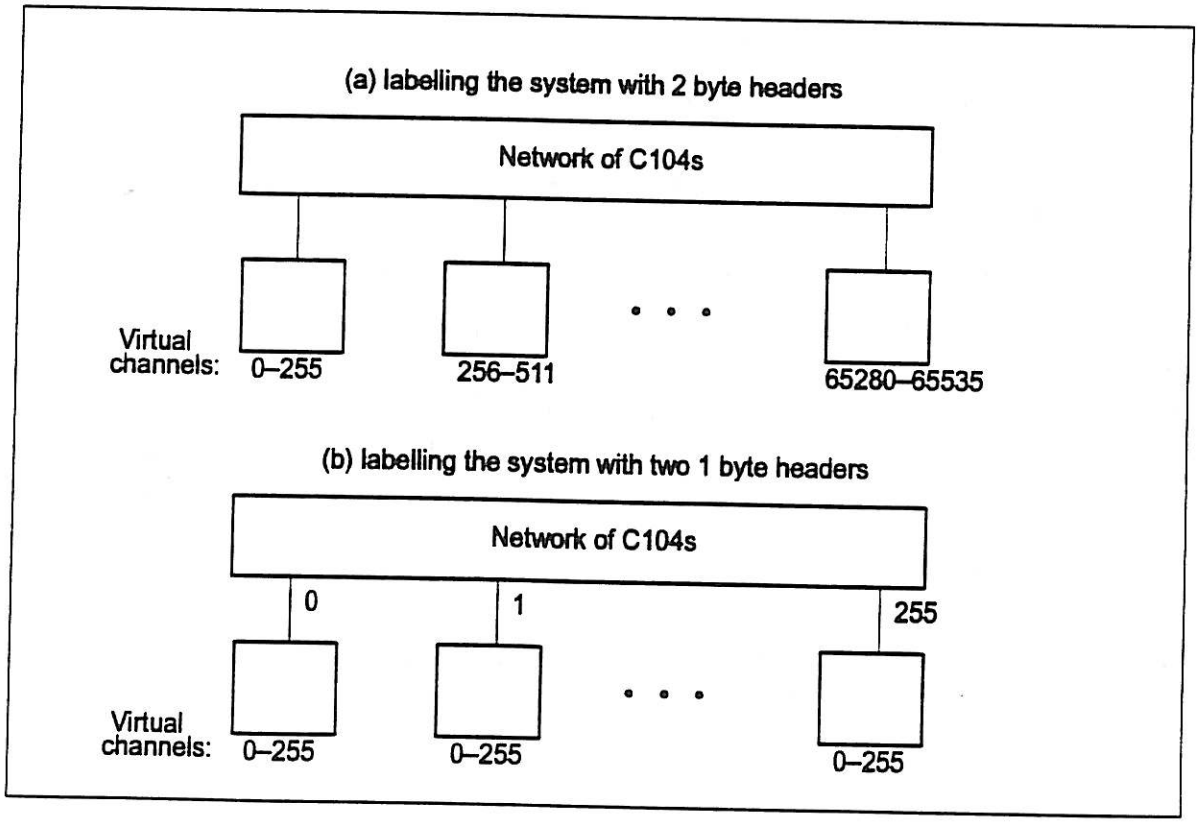


Figure 3.4 Using header deletion to label a network

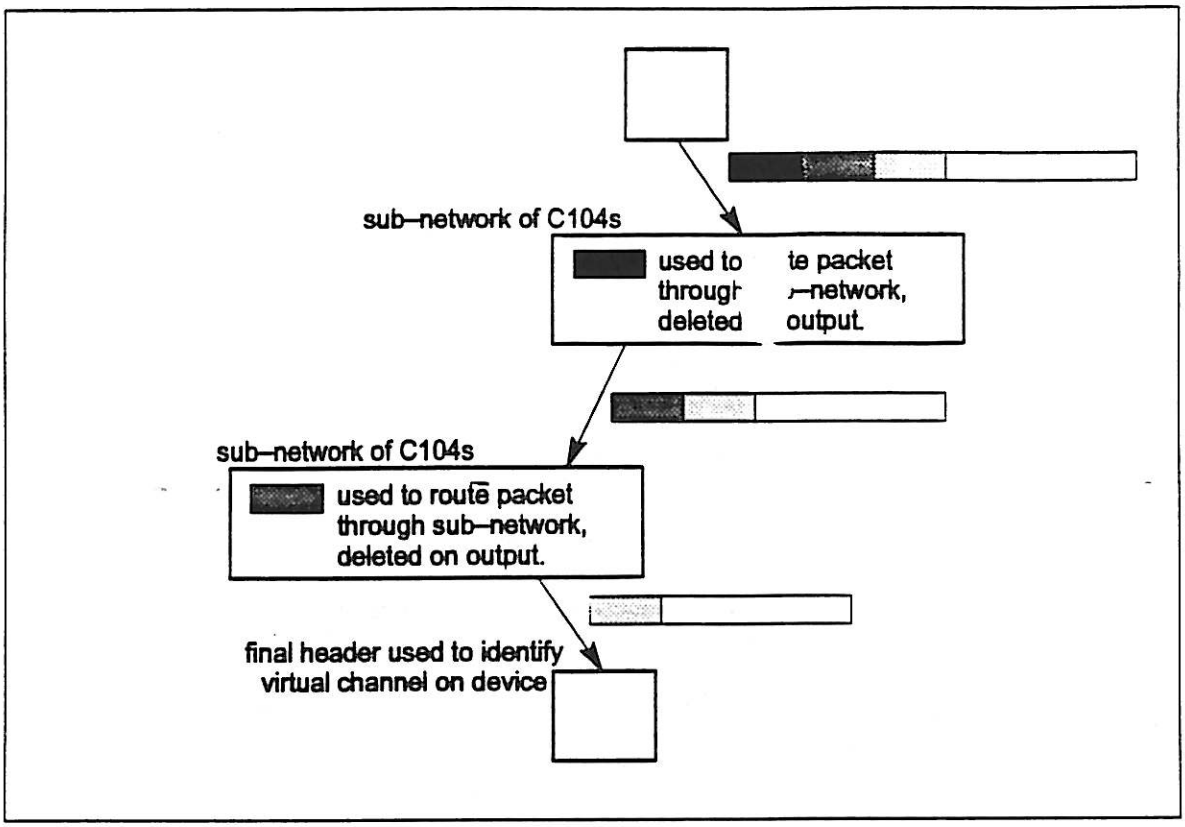


Figure 3.5 Using header deletion to route through sub-networks

The advantages of using header deletion in a network are:

- It separates the headers, and therefore the routing information, for virtual channels from those for the routing network.
- The labelling of the network can be done independently of the application software running on the network.
- There is no limit to the number of virtual channels that can be handled by a system.
- By keeping the header for routing short, routing latency is minimized.

Any number of headers can be added to the beginning of a packet so that header deletion can also be used to combine hierarchies of networks as shown in figure 3.5. An extra header is added to route the message through each network. The header at the front of each packet is deleted as it leaves each network to enter a sub-network.

3.3 Routing algorithms

In order to route a message through a network, an algorithm is required which is: complete (ensures that all messages arrive); deadlock free; optimal (packets take the shortest route); scaleable (networks of any size can be built) and simple to implement.

3.3.1 Labelling networks

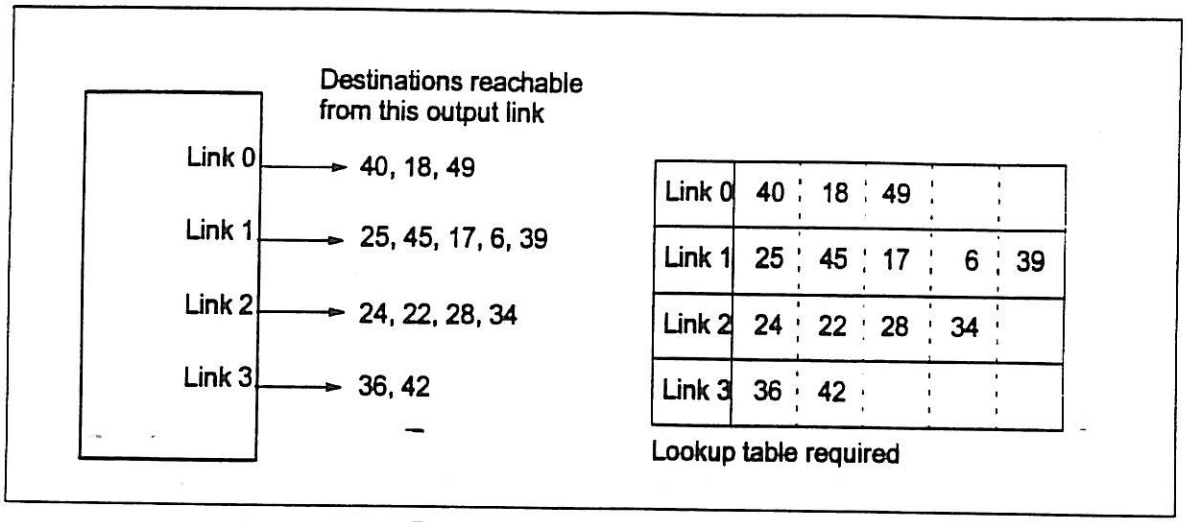


Figure 3.6 Labelling a network

For each routing component there will be a number of destinations which can be reached via each of its output links. Therefore, there needs to be a method of deciding which output link to use for each packet that arrives. The addresses that can be reached through any link will depend on the way the network is labelled. An obvious way of determining which destinations are accessible from each link, is to have a lookup table associated with all the outputs (see figure 3.6). In practice, this is difficult to implement. There must be an upper bound on the lookup table size and it may require a large number of comparisons between the header value and the contents of the table. This is inefficient in silicon area and also potentially slow.

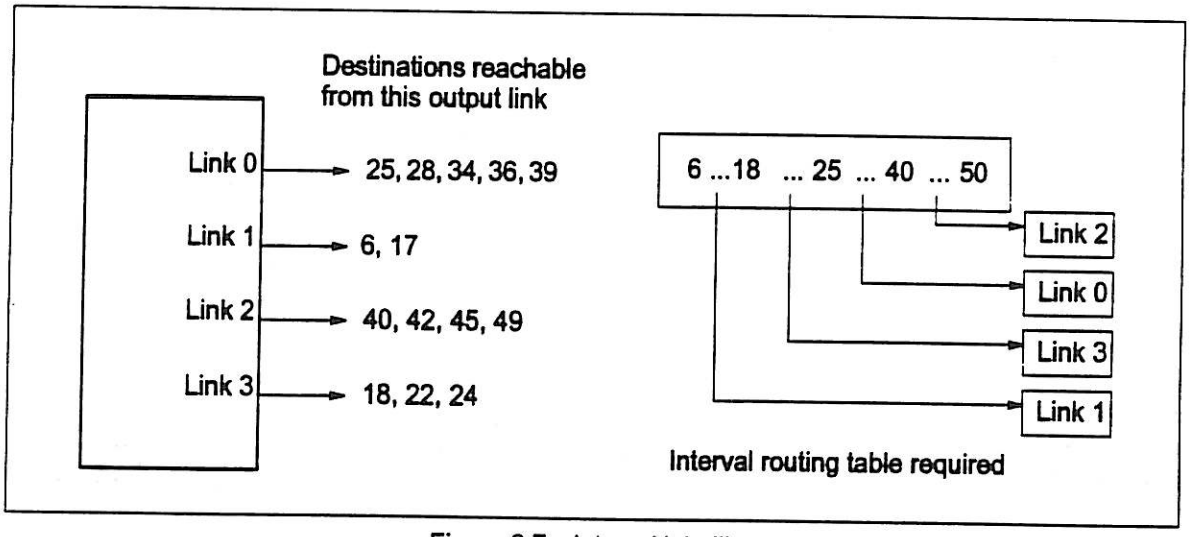


Figure 3.7 Interval labelling

However, a labelling scheme can be chosen for the network such that each output link has a *range* of node addresses that can be reached through it. As long as the ranges for each link are non-overlapping, a very simple test is possible. The header just has to be tested to see into which range, or interval, it falls and, hence, which output link to use. For example, in figure 3.7, a header with address *n* would be tested against each of the four intervals shown below:

Interval		Output link
$6 \leq n < 18$	⇒	1
$18 \leq n < 25$	⇒	3

$25 \leq n < 40$ ⇾ 0

$40 \leq n < 50$ ⇾ 2

The advantages of interval labelling are that:

- It is 'complete' – any network can be labelled.
- It provides an absolute address for each device in a network, so keeping the calculation of headers simple.
- It is simple to implement in hardware – it requires little silicon area which means it can be provided for a large number of links as well as keeping costs and power dissipation down.
- Because it is simple, it is also very fast, keeping routing delays to a minimum.

3.3.2 Avoiding deadlock

Deadlock can occur in a flow–controlled network unless the routing algorithm is designed to avoid it.As a simple example consider a network of four nodes (see figure 3.8) with one link in each direction between each node. If the routing algorithm sends all messages clockwise and all nodes start sending to the opposite corner at the same time, every link will become busy and the network will deadlock. It is possible to add buffers to the network, but this will only delay the point at which deadlock occurs. The amount of buffering needed to avoid deadlock is dependent on the network size and the application program running on the network.

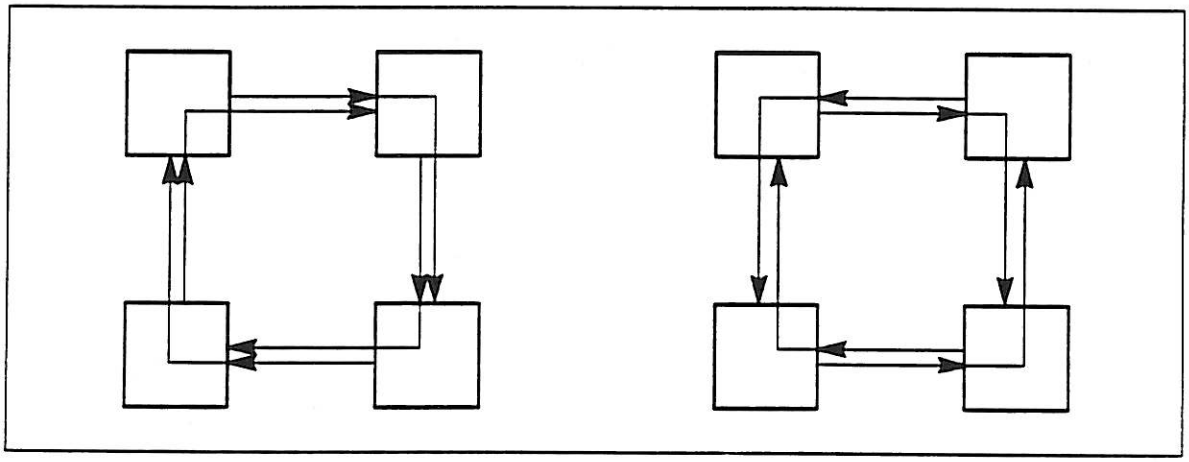


Figure 3.8 Deadlock in a network

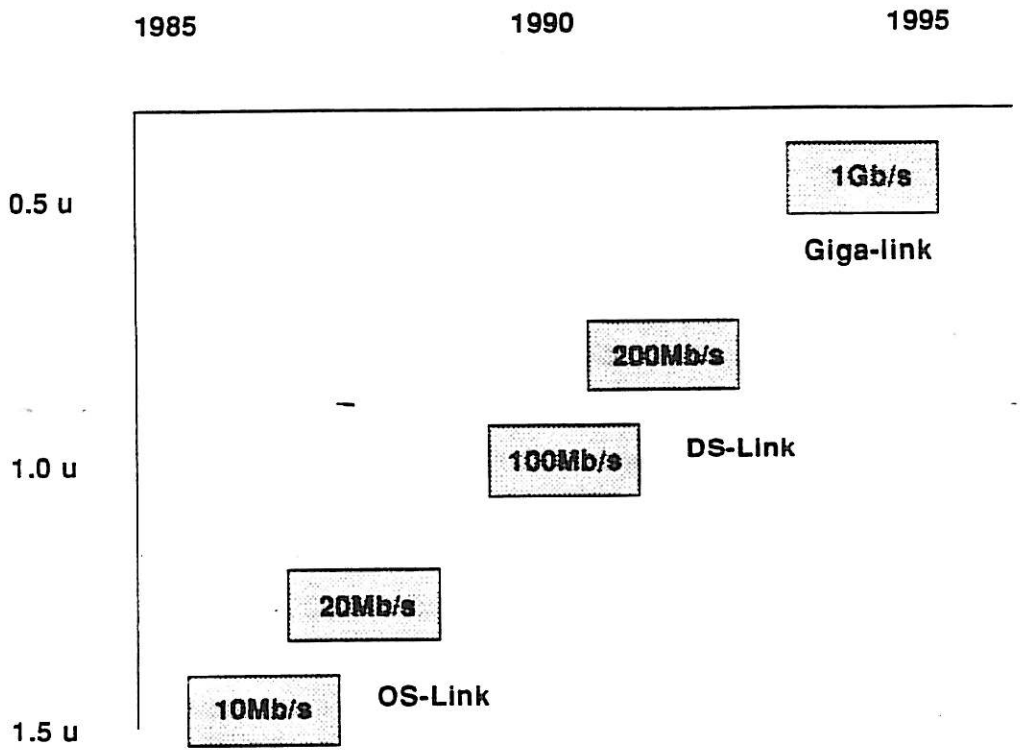
In this example, deadlock can easily be avoided by modifying the routing algorithm to send messages in opposite directions from alternate nodes. In this case, each node will only need to send one message in each direction at any time. In this network, buffering can be added just to smooth the flow of data (i.e. to prevent a process having to wait to send a message when the network is busy) but it is not needed to prevent deadlock.

It is possible to use interval labelling to label any network in a deadlock free way. Many regular networks have optimal, deadlock free routing algorithms. Examples are trees, hypercubes, multi–stage networks and grids. These networks can then be combined, so that any network can be optimally labelled as if constructed from these sub-networks.

4 Conclusion

DS–Link technology provides reliable, high–speed serial communications at low cost, in a simple form which is suitable for a wide range of applications. A simple protocol, implemented in hardware, keeps overheads down whilst allowing more complex functions to be layered on top of it. It also permits high–performance routing devices to be constructed, from which efficient systems of any size can be built to provide very high system bandwidth and fault–tolerance.

Appendix A: Link technology roadmap



Appendix B: DS-links in the context of other serial interfaces

It may be useful to those unfamiliar with DS-links to compare them with other serial interfaces which may be better known. These include P1394 Serial Bus, Serial Storage Architecture (SSA), Fibre Channel, all from the computer/disk industry, and Asynchronous Transfer Mode (ATM) from the telecommunications industry.

P1394 Serial bus recognises that a cabled bus with a total length of cable up to several tens of metres would be severely limited by the bus capacitance. It therefore uses point-to-point cables, and performs the actual bus function in silicon, where the lower bus capacitance offers an order of magitude faster performance than would be provided by a cabled bus. Serial Bus has a low implementation cost because it avoids the need for complicated coding/decoding, and also does not need analog components for clock recovery.

SSA recognises that the data-rate for short packets can be severely limited by the overheads of the header and tail of a packet; SSA consequently limits the packet overhead to five bytes. The payload of the packet is limited to 64 bytes, so that a short control packet is not blocked by a long data packet. SSA also recognises the advantages, both for performance and fault-tolerance, of improvements in network topology. By using a full-duplex ring, SSA achieves four times the system bandwidth available from a half-duplex bus or token-ring; because the ring can be accessed from either end, the system can tolerate the loss of a node or of a cable on the ring.

Fibre Channel takes the topology improvements a stage further by introducing the concept of routing "fabric." The fabric is rather like a telephone exchange, carrying many independent messages between different places at the same time. The Fibre Channel exchange is packet switched, depending on the header, rather than circuit switched as is the normal telephone exchange. Fibre Channel is designed to go both further and faster than either Serial Bus or SSA, with a distance of the order of 10 kilometres rather than 10 metres, a low speed of 133MBaud (10 MBytes/s) and a high speed exceeding 1GBaud (100 MBytes/s).

ATM appears different from the others in that it has been designed for telecommunications, but it has a number of similarities and may also be appropriate for networks within buildings. It has a fixed packet (called "cell") size of a five byte header and 48 byte data or payload. Within the header are two address fields, one to identify the path that the packet should take to its destination, and the other to identify where the packet should go at the destination. The header has a hamming-code check, but the payload field does not have a check — the check on the data is left to a higher level of the protocol and can therefore be dependent on what sort of data is being transmitted. As can be imagined with a telecommunications protocol, there is a wide variety of network topologies that can be used, and fault-tolerance is inherent in these topologies.

DS-links were designed before some of these other interfaces, but in some respects can be seen to build on the benefits of each of them.

As with all of the interfaces described, DS-links are point-to-point connections, which can be much better controlled than bussed connections.

Like P1394, DS-links have a very simple bit-level encoding, and do not require analog components for clock recovery. The DS-link bit-level encoding has the further advantages of allowing much more skew tolerance, and of being fully "autobaud" so that the receiver can receive data at any transmitted speed that it can keep up with. The simple header of the DS-link, combined with the encoding, allow low cost implementation of the DS-link circuit. The message latency of DS-Links is particularly low as a result of this simple header.

Like SSA, DS-links have small overheads on each packet and limit the length of the data field of the packet. Also like SSA, the DS-links are full-duplex, and can benefit from the bandwidth and fault tolerance improvements of a full-duplex ring.

DS-links do not attempt to cover the distance covered by Fibre Channel. They are primarily designed for connections within a subrack or cabinet, but, like P1394 and SSA, are limited to a cable distance of the order of 10 metres. Developments are in hand to extend the speed and distance of DS-links: these longer distance connections will cost more than the standard DS-Link, but the user need only pay for the performance and distance required.

Like Fibre Channel, however, DS-links can either be used directly point-to-point between nodes or can use a "fabric" of routing switches. The 32-valent routing switch has a through-switch bandwidth of 320MBytes/s

with 100MBaud DS-links, and in the absence of contention for a link output, the packet latency will be less than 1 microsecond. Contention is minimised by the routing switch having 32 buses, each with 32 ports, all operating in parallel. Contention is further minimised by the "Group Adaptive Routing" and "Universal Routing" techniques for hot-spot avoidance (described in the routing switch specification).

Like ATM as well as SSA, the DS-link header is simple and short: it can be as short as a single byte, two bytes if it goes through a router. Like ATM, the header is comprised of two parts, one of which defines the routing path, and the other the channel of the process or task at the destination. Different error handling mechanisms are appropriate for different types of data and equipments, so although a simple parity check is performed to locate the most commonly occurring errors, any further error handling is , as in ATM, left to a higher level of the protocol. As with a telephone exchange, there is a wide variety of topologies which can be used, which provide scalable performance as well as fault-tolerance.

There are many factors which contribute to the overall throughput, cost/effectiveness, and fault tolerance. Many of these, however, boil down to the complexity of the header and other overheads of the packets: the simpler these are, the faster the link can run, the less silicon area is consumed, the lower the manufacturing cost, the higher the utilization for short packets, the higher is the possible valency of the routing switch and so the fewer routing switches are needed to build a large network, and so the fewer hops any message needs to to go between any two nodes. In any network, the more different paths there are between any two nodes, the more faults the network can tolerate, and the high-valency routers very easily give such multiple routes.

Some of these parameters are tabulated below for DS-links:

Metric (for DS-Link)	**Value (for DS-Link)**
Shortest header (Direct connection)	1 byte
Shortest header (Routed)	2 bytes
Shortest data packet	header alone
Longest data packet	header plus 32 bytes
Delay of Router (unblocked)	$<1\mu s$
Hot-spot avoidance	Universal routing plus adaptive routing
Valency of single-chip routing switch currently achievable	32
Through-switch bandwidth of 32-way Router	320Mbytes/s
Maximum network size	> 1024 nodes
Number of logical channels per node	limited only by memory
Need for clock recovery analog circuitry	none
Implementation cost	32 links possible on $1cm^2$ chip
Dual redundancy possible (mirroring)?	yes
Triple modular redundancy possible (TMR)?	yes
n-Modular redundancy possible (nMR)?	yes
n+1 redundancy possible (as in RAIDs, Power supplies)?	yes
Programming model	Process-to-process, synchronized

The fact that DS-Links share attractive features with all of the interfaces mentioned suggests that DS-Links would be an ideal glue for interfacing between the various interfaces, and particularly for implementing routing between them.

In some systems, it may be seen that the combination of features in a single interface that can be used throughout the system might remove the need for other interfaces, at least within the system.

Appendix C: Mapping the SCSI protocol onto links.

1.1 Introduction

The concepts used for this mapping of SCSI onto the DS link architecture takes into account fully the two and half years of experience gained running SCSI over OS links. This proposed mapping is intended to demonstrate how to exploit the features of the DS link architecture to provide for both high performance and fault tolerance. It is not intended at this point in time to be interpreted as a definitive mapping for SCSI on DS. Such a mapping could be used on a single DS link in conjunction with any of the other alternative protocols such as GPP or FC–4S by allocating each protocol a different virtual channel.

1.2 Strategy

Making the transition from a Bus based architecture to a point to point based architecture, does in general simplify the software architecture of the system. A great deal of SCSI's functionally is provided primarily to maximise the utilisation of the Bus. Examples of such functionality are :– selection, identify, disconnect–reconnect and synchronous transfer mode.

The core aspects of SCSI that are desirable for mapping onto a serial architecture are:

- The Logical Block abstraction of the media
- The Defect Handling capabilities
- The Command Sets for different classes of devices

The following aspects of SCSI can also be desirable, but depend upon the intended application :–

- Queue Tagged I/O
- Cacheing

The approach taken in this document has been to map as literally as possible the parallel bus architecture of SCSI onto the serial point–to–point architecture of DS links. Whilst this will provide a perfectly adequate implementation, extra performance will be achieved by defining a fully coherent specification for a protocol, that provides for both the core aspects of SCSI and in addition the serial point–to–point communications aspects of DS links.

The concept of Initiator and Targets has been preserved in this mapping. However if moving to a true *peer–to–peer* architecture, it is worth considering whether such a concept be preserved.

To take advantage of the virtual channel capability of DS–Links a communication between an Initiator and a Target is implemented by 2 pairs of virtual channels (1 pair from the Initiator to the Target and 1 pair from the Target to the Initiator). Each pair consists of a control channel and a data channel. Thus in each direction, there are 2 separate channels for control and data. This has the advantage that the virtual channel carrying the data can be connected directly up to either the track buffer or the DMA controller for Targets and Initiators respectively. Also the provision of separate command and data channels ensures that long Data Out phases do not block additional SCSI Command requests from being transferred from the Initiator to the Target as would otherwise be the case for a single combined control and data channel. Using this scheme, control channels will always carry control information and non media access data.

This approach does not in any way preclude using a single pair of virtual channels to transfer both command and data instead. It is merely described in order to demonstrate how the DS link architecture can be more fully exploited to maximise performance of the system as a whole.

It may also be worth considering whether to include padding fields within packets to ensure that all fields begin on a 32 bit word boundary to simplify firmware implementations for both Targets and Initiators. This has not been included for in the current mapping.

1.3 Nomenclature

In order to avoid confusion in terminology between SCSI and DS, the following rules will be adhered to within this section of the document :–

- A **DS_SCSI_Transaction** is a transfer of information between 2 **DS_SCSI** devices.
- An **Initiator** is a device that initiates a **DS_SCSI_Transaction.**
- A **Target** is a device that provides a service to an **Initiator** via a **DS_SCSI_Transaction.**
- A **DS_Packet** is a 0 to 32 byte sequence of bytes headed by a **DS_Message_Header** token and terminated by a **DS_End_of_Packet** token.
- A **DS_Message** is a DS link message. i.e. a sequence of information delivered in 32 byte **DS_Packets.**
- A **DS_SCSI_Protocol** is the protocol used to transfer information between **DS_Devices.**
- A **DS_SCSI_Packet** is a protocol frame within the **DS_SCSI_Protocol.**
- **DS_Media_Data** is data transferred either to or from a devices media whether it be magnetic media, optical_media or host memory. It does not include interface management media such as Inquiry data for example.
- **DS_Interface_Management_Data** is data that is transferred solely for the purpose of either obtaining device capability information or defining modal operations of devices. It does not include **DS_Media_Data.** SCSI Inquiry data is an example of **DS_Interface_Management_Data**
- A **DS_Media_Data_Virtual_Channel** is a virtual channel allocated specifically to transfer Media Data to and from the media.
- A **DS_Command_Virtual_Channel** is a virtual channel allocated specifically to transferring **DS_Command_Packets** and **DS_Interface_Management_Data.**
- A **DS_Media_Data_Packet** is a **DS_SCSI_Packet** transmitted on the **DS_Media_Data_Virtual_Channel.**
- A **DS_Command_Packet** is a **DS_SCSI_Packet** transmitted on the **DS_Command_Virtual_Channel.**
- A **DS_Command_Virtual_Channel_Pair** consists of a pair of input and output **DS_Command_Virtual_Channels** on a single bi–directional **DS_Link.**
- A **DS_Media_Data_Virtual_Channel_Pair** consists of a pair of input and output **DS_Media_Data_Virtual_Channels** on a single bi–directional **DS_Link.**

1.4 Mapping SCSI onto the DS link architecture.

Essentially all that is required to map any protocol onto DS links is the availability of interfaces that superimpose the desired protocol on top of DS links packets. This will be completely transparent to the sender and receiver who will simply see a number of virtual channels that are connected as far as they are concerned directly to other devices in the network. Thus the desired protocol packet equates to a DS link message which embodies both the original packet and the routing information to deliver the original packet to its destination. The Transmitter adds the DS protocol, the Receiver subtracts the protocol. The whole scheme is basically a Time Division Multiplexor, or put another way, a Packet Switched Network. The DS link architecture provides the fabric to transfer information between any 2 devices in the network. The routing of the information is transparent to both the sender and receiver.

Using DS links and routers, routing of SCSI requests to a particular target from an initiator is completely transparent. There may be many possible routes to any particular device in the network. Thus if one link does not respond, other links are tried until contact is made. Also multiple links may be used in tandem to provide multiple command paths and also to increase performance for devices capable of transferring information faster than the bandwidth of a single DS link.

Each device is allocated a unique **Device_ID.** This is not used at all to identify a nexus to a particular device. It is included only for *end–to–end checking* purposes. Thus in the *extremely unlikely* event of a request arriving

to a device with a **Destination_ID** conflicting with the devices **Device_ID**, then recovery steps can be taken. Such occurrences could stem from :–

- Incorrectly programmed router devices
- Operating system error
- A faulty router
- A Host Adaptor error

The **Destination_ID** and **Source_ID** is included within a CRC of a **DS_SCSI_Packet.** It is not necessary to implicitly specify them within a **DS_SCSI_Packet** since both the sender and the receiver have been configured such that they both know their own **Device_ID** and the **Device_ID** of the device that is connected to the other end of a particular **Virtual_Channel.**

1.5 Protocol Description

The **DS_SCSI_Protocol** is split up across a **DS_Command_Virtual_Channel_Pair** and a **DS_Media_Data_Virtual_Channel_Pair.**

As previously mentioned, the **DS_Command_Virtual_Channel_Pair** is specifically allocated to transferring **DS_Interface_Management_Data** and the **DS_Media_Data_Virtual_Channel_Pair** is specifically allocated to transfer **DS_Media_Data.**

The protocol involves the transfer of packets on both the command and data links.

1.5.1 DS_Media_Data_Virtual_Channel Packets.

On a **DS_Media_Data_Virtual_Channel_Pair** there are 2 types of packets transmitted in either direction; **DS_Media_Data_Header_Packets** and **DS_Media_Data_Packets.**

DS_Media_Data_Header_Packet

The general format for a **DS_Media_Data_Header_Packet** is as described in table 1.1.

DS_Media_Data_Header_Packets are used to specify information about the contents of the following **DS_Media_Data_Packet.**

DS_Media_Header_Packets are always transmitted by the sender prior to every **DS_Media_Data_Packet.** Consequently, the receiver always expects to receive a **DS_Media_Data_Header_Packet** prior to every **DS_Media_Data_Packet.**

Field	Number of Bytes
Packet_Length	4
Packet_Type	1
Logical_Block_Address	4
CRC	4

Table 1.1 DS_Media_Data_Header_Packet

The **Packet_Length** field specifies the total length of the packet. It does not include the bytes occupied by either itself or the **CRC** field. Although the flow control low level protocol of DS links permits the receiving device to detect the end of the a DS message, it is quite often useful to know the precise quantity of **Media_Data** that will be provided within the **DS_Media_Data_Packet** at the start of reception. This permits for example, the receiver to organise how to buffer the data.

The **Logical_Block_Address** field provides sequencing information. During a **Media_Data_In_Phase**, segments of the overall transfer may arrive in any order and on any DS link that has a virtual channel connected to the pertaining target. The size of the transfer

DS_Media_Data_Packet

Field	Number of Bytes
Media_Data	Packet_Length
CRC	4

Table 1.2 DS_Media_Data_Packet

1.5.2 DS_Command_Virtual_Channel_Packets

The general format of a **DS_SCSI_Command_Packet** is as described in table 1.3.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1
Packet specific data	*
CRC	4

Table 1.3 DS_SCSI_Command_Packet template

The **Packet_Length** field specifies the total length of the packet specific data. It does not include the bytes occupied by either itself or the **CRC** field. In order to differentiate packets, each packet type is allocated the unique tag **Packet_Type.** The significance of each tag value for **Packet_Type** is as defined in table 1.4.

Packet Type	Tag value
DS_Configuration	0
DS_Interrogation_Request	1
DS_Interrogation_Reply	2
_Message_Out	3
DS_SCSI_Command	4
DS_Media_Data_Header	5
DS_Non_Media_Data_In	6
DS_Media_Data_Out_Phase_Request	7
DS_Non_Media_Data_Out_Phase_Request	8
DS_Non_Media_Data_Out	9
DS_Message_In	10
DS_Status	11
DS_Exception	12
DS_Reset	13
DS_Terminate_IO	14
DS_Spindle_Synchronization	15

Table 1.4 Packet Types

The **CRC** field provides a Cyclic Redundant Checksum pertaining to the complete packet of information from and including the Packet_Length field to the byte prior to the **CRC** field. The **CRC** also embodies the **Destina-**

tion_ID and the **Source_ID**. When checking the **CRC** a receiving device should include both its **Device_ID** (which is the **Destination_ID**) and the **Source_ID**. It knows the **Source_ID** because at configuration time it has been informed that all control and data packets arriving on a particular pair of input virtual channels pertain to the a particular **Source_ID**. The advantage of this scheme is that :

- a) The source and destination ID's are transparent during normal operating conditions.
- b) Full *end–to–end checking* is possible which also includes checking of routing errors.

1.5.3 Configuration

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 0
Destination_ID	2
Source_ID	2
Device_Type	1
Configuration_Data_Length	4
Configuration_Data	Configuration_Data_Length
CRC	4

Table 1.5 Configuration

Where:–

Device_Type	Tag
Initiator	0
Target	1

Table 1.6 Device_Types

Field	Number of Bytes
To be Defined	?

Table 1.7 Configuration_Data

1.5.4 Interrogation

In order that an Initiator may Interrogate devices connected to it, provision is made in the protocol to request that **Device_Properties** information be returned by the device connected to the Initiator. This information pertains only to the serial communications aspect of the interface and does not duplicate information that could otherwise be obtained via the **SCSI Inquiry** command. The request is made by issuing a **DS_Interrogation_Request** described in table 1.8.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 1
Device_Properties_Length	2
CRC	4

Table 1.8 DS_Interrogation_Request

On receipt of such a message, the device whether it be an Initiator,Target or Recovery Unit will respond with a **DS_Interrogation_Reply** as described in table 1.9.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 2
Device_Properties	Packet_Length
CRC	4

Table 1.9 DS_Interrogation_Reply

Field	Number of Bytes
Device_Type	1
Device_Type_Data	Device_Data_Length – 5

Table 1.10 Device_Properties

Field	Number of Bytes
To Be Defined	Device_Properties_Length – 5

Table 1.11 Device_Type_Data : Initiator

Field	Number of Bytes
To Be Defined	Device_Properties_Length – 5

Table 1.12 Device_Type_Data : Target

1.5.5 DS_Message_Out

The **DS_Message_Out** phase provides a mechanism for the Initiator to either inform the Target of some exception, or to request the target take some action. It is debatable whether this should be supported in the same way as Parallel SCSI, but for the sake of completeness it is defined in **DS_SCSI** as described in table 1.13.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 3
Destination_ID	2
Source_ID	2
Message_Length	1
Message	Message_Length
CRC	4

Table 1.13 DS_Message_Out

1.5.6 DS_SCSI_Command

The **DS_SCSI_Command** phase may transfer either a single **SCSI_CDB** or may be any length if a series of linked **SCSI_CDB** commands are to be executed by the target without being interspersed by commands from other competing initiators connected to the target (*Read–Modify–Write–Cycles*).

The format of the **DS_SCSI_Command** is as shown in table 1.14.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 4
Queue_Tag	1
Total_Transfer_Size	4
Transfer_Unit_Size	4
SCSI_CDB_Length	1
SCSI_CDB	SCSI_CDB_Length
CRC	4

Table 1.14 DS_SCSI_Command template

The **Queue_Tag** field is as specified in the SCSI–2 message system except that it is embodied within the Command Data Block Message. An alternative method to using **Queue_Tags** is to configure multiple **DS_SCSI_Virtual_Connections** between a particular Initiator and the Target. Multiple threads can thus be maintained across multiple **DS_SCSI_Virtual_Connections**. A combination of both techniques may also be used where there is a desire to limit the number of virtual channels used between a particular Initiator and the Target.

The **SCSI_CDB_Length** is usually equal to 6 or 10 though may be any length up to 255 if the **SCSI_CDB** contains a number of linked commands.

The **SCSI_CDB** field is either a single or linked series of SCSI–2 Command Descriptor Blocks.

The **Total_Transfer_Size** specifies the total number of bytes to be transferred during a data phase. This may be zero even for say a READ command. On receipt of such a READ command, the drive will simply seek to the specified command and transfer no data.

A Test Unit Ready command would always have the **Total_Transfer_Size** equal to zero.

The **Transfer_Unit_Size** defines unit of transfer for media access data phase type commands. For a data in phase type command, the target shall break up the request into a number of Data_In_Phase messages with Data_Transfer_Sizes of **Transfer_Unit_Size**. The **Transfer_Unit_Size** should divide exactly into the

Total_Transfer_Size. Where this is not possible, the **Transfer_Unit_Size** should equal the **Total_Transfer_Size**. The **Transfer_Unit_Size** must always be divisible by 4 to that DMA controllers interfacing to 32 bit wide memory systems can transfer the data to memory, without needing to cater for special remainder situations.

A Test Unit Ready command would always have the Transfer_Unit_Size equal to zero.

The provision of **Transfer_Unit_Size** simplifies the Initiator firmware when configuring DMA controllers connected to the pertaining virtual channels. For example, if the firmware knows that a 1 Mbyte READ command will be split up into 1000 1kByte messages, then after receipt of the first 1K Byte message into a LAN packet frame for transmitting onto a LAN, it can set the DMA controller to input the next 1k Byte into another LAN packet frame and so on. Thus SCSI transfers and LAN activity can be pipelined. This minimises latency of the system in a file server application.

If the Target is unable to accommodate the **Transfer_Unit_Size** specified (even though it can satisfy the complete **Total_Transfer_Size** so long as it is broken up into small enough segments) , it must generate an **Exception** packet (see 1.5.15) and forward it to the Initiator. The Initiator will then use the smaller **Transfer_Unit_Size** specified in the **Exception** packet.

1.5.7 DS_Data_In_Phase

A **DS_Data_In_Phase** occurs by the Target sending a **DS_Media_Data_Header_Packet** followed by a **DS_Media_Data_Packet**, on the **DS_Media_Virtual_Channel** from the Target to the Initiator. The Target may use multiple DS Links to transfer multiple **Data_In_Phases** in order to increase throughput for devices capable of transferring information faster than a single DS link bandwidth. The sequencing information is preserved by the inclusion of the Logical_Block_Address within the accompanying **DS_Media_Data_Header_Packet**. Also in the event of a failure of one of the links of a dual ported device, the information may be transferred on the other link. See also section 1.5.1.

1.5.8 DS_Non_Media_Data

The **DS_Non_Media_Data_In** phase transfers previously requested information from the Target to the Initiator. The **Queue_Tag** supplied pertains to the original initiated **DS_SCSI_Command** phase.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 6
Queue_Tag	1
Data	Packet_Length – 2
CRC	4

Table 1.15 Non_Media_Access_Data_In_Phase

1.5.9 DS_Media_Data_Out_Request

The **DS_Media_Data_Out_Request** requests that the Initiator transfer the data to be written to the media starting at the position of the specified **Logical_Block_Address**. The quantity of the data supplied will be **Transfer_Unit_Size** in length. The Initiator should verify that this value corresponds to its originally specified **Transfer_Unit_Size** within the **Command_Phase**. The **Queue_Tag** supplied pertains to the applicable initiated **DS_SCSI_Command** phase.

The Data Out phase needs a request to be sent from the Target to the Initiator so that the Target can define the I/O, the quantity and the positional information of the data. The Data In phase on the other hand needs no such request since it is the Target that is always in the position of deciding which I/O to service and the manner in which it should be implemented.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 7
Queue_Tag	1
Transfer_Unit_Size	4
CRC	4

Table 1.16 DS_Media_Data_Out_Request

1.5.10 DS_Non_Media_Data_Out_Request

The **DS_Non_Media_Data_Out_Request** requests that the Initiator transfer the non media data pertaining to the **Queue_Tag** supplied by the applicable initiated **DS_SCSI_Command** phase. The quantity of the data supplied will be **Transfer_Unit_Size** in length. The Initiator should verify that this value corresponds to its originally specified **Transfer_Unit_Size** within the **Command_Phase.**

The Data Out phase needs a request to be sent from the Target to the Initiator so that the Target can define the I/O, the quantity and the positional information of the data. The Data In phase on the other hand needs no such request since it is the Target that is always in the position of deciding which I/O to service and the manner in which it should be implemented.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 8
Queue_Tag	1
Logical_Block_Address	4
Transfer_Unit_Size	4
CRC	4

Table 1.17 DS_Media_Data_Out_Request

1.5.11 DS_Non_Media_Data_Out

The **DS_Non_Media_Data_Out** phase transfers a quantity of data as previously requested by the target by a **DS_Non_Media_Data_Out_Request**, from the Initiator to the Target. The quantity of the data supplied will be **Transfer_Unit_Size** in length and should correspond to the **Transfer_Unit_Size** specified in the original **Command_Phase**. The **Queue_Tag** supplied pertains to the applicable initiated **DS_SCSI_Command.**

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 9
Queue_Tag	1
Data	Packet_Length – 6
CRC	4

Table 1.18 DS_Non_Media_Data_Out packet.

1.5.12 DS_Media_Data_Out

The **DS_Media_Data_Out** phase transfers a quantity of data as previously requested by the target by a **DS_Media_Data_Out_Request**, from the Initiator to the Target. The quantity of the data supplied will be **Transfer_Unit_Size** in length and should correspond to the **Transfer_Unit_Size** specified in the original **Command_Phase**. The Initiator may use multiple DS Links to transfer multiple **DS_Media_Data_Out** packets in order to increase throughput for devices capable of storing information faster than a single DS link bandwidth. The sequencing information is preserved by the inclusion of the Logical_Block_Address within the accompanying **DS_Media_Data_Header_Packet**. Also in the event of a failure of one of the links of a dual ported device, the information may be transferred on the other link. See also section 1.5.1.The **Queue_Tag** supplied pertains to the applicable initiated **DS_SCSI_Command.**

1.5.13 DS_Message_In

The **DS_Message_In** phase provides a mechanism for the Target to either inform the Initiator of some exception, or to request a service from the Initiator. It is debatable whether this should be supported in the same way as Parallel SCSI, but for the sake of completeness it is defined in DS_SCSI as described in table 1.19.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 10
Message	Packet_Length – 1
CRC	4

Table 1.19 DS_Message_In packet

1.5.14 DS_SCSI_Status

The **DS_SCSI_Status** phase provides a means for the Target device to inform the Initiator the success of a SCSI command.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 11
Queue_Tag	1
SCSI_Status	1
Message_In	1
CRC	4

Table 1.20 DS_SCSI_Status packet

The **Queue_Tag** field is as specified in the SCSI–2 message system except that it is embodied within the **DS_SCSI_Status** packet.

The **SCSI_Status** field is as specified by the SCSI–2 specification and indicates the completion status of the **SCSI_CDB** issued by the initiator with the specified **Queue_Tag.**

The **Message_In** field provides for a **Command_Complete** Message–In–Phase to be embodied within the status phase. In normal SCSI–2 operation, a **Command_Complete** message usually accompanies a SCSI–2 Status–Phase.

1.5.15 DS_Exception

Exceptions that are detected by a devices are communicated to other devices via **DS_Exception** packets.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 12
Queue_Tag	1
Exception_Data	Packet_Length – 6
CRC	4

Table 1.21 DS_Exception packet

Field	Number of Bytes
To be Defined	?

Table 1.22 Exception_Data

1.5.16 Reset

Devices may be Reset into a known state via **DS_Reset_Packets.**

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 13
Queue_Tag	1
Reset_Data	Packet_Length – 6
CRC	4

Table 1.23 DS_Reset packet

Field	Number of Bytes
To be Defined	?

Table 1.24 Reset_Data

1.5.17 DS_Terminate_IO

Devices may be Reset into a known state via **DS_Terminate_IO** packets.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 14
Queue_Tag	1

Terminate_IO_Data	Packet_Length – 6
CRC	4

Table 1.25 DS_Terminate_IO packet

Field	Number of Bytes
To be Defined	?

Table 1.26 Terminate_IO_Data

1.5.18 DS_Spindle_Synchronization.

Devices may be spindle synchronized by means of DS_Spindle_Synchronization packets.

Field	Number of Bytes
Packet_Length	4
Packet_Type	1 = 15
Queue_Tag	1
Spindle_Synchronization_Data	Packet_Length – 6
CRC	4

Field	Number of Bytes
To be Defined	?

Table 1.27 Spindle_Synchronization_Data

Appendix D: IMS C104 packet routing switch

1 Introduction

This document contains preliminary information for the IMS C104 packet routing switch.

The IMS C104 (figure 1.1) is a complete, low latency, packet routing switch on a single chip. It connects 32 high bandwidth serial communication links to each other via a 32 by 32 way non–blocking crossbar switch, enabling messages to be routed from any of its links to any other link. The links operate and packets are processed concurrently, and the transfer of a packet between one pair of links does not affect the latency or data rate for another packet passing between a second pair of links.

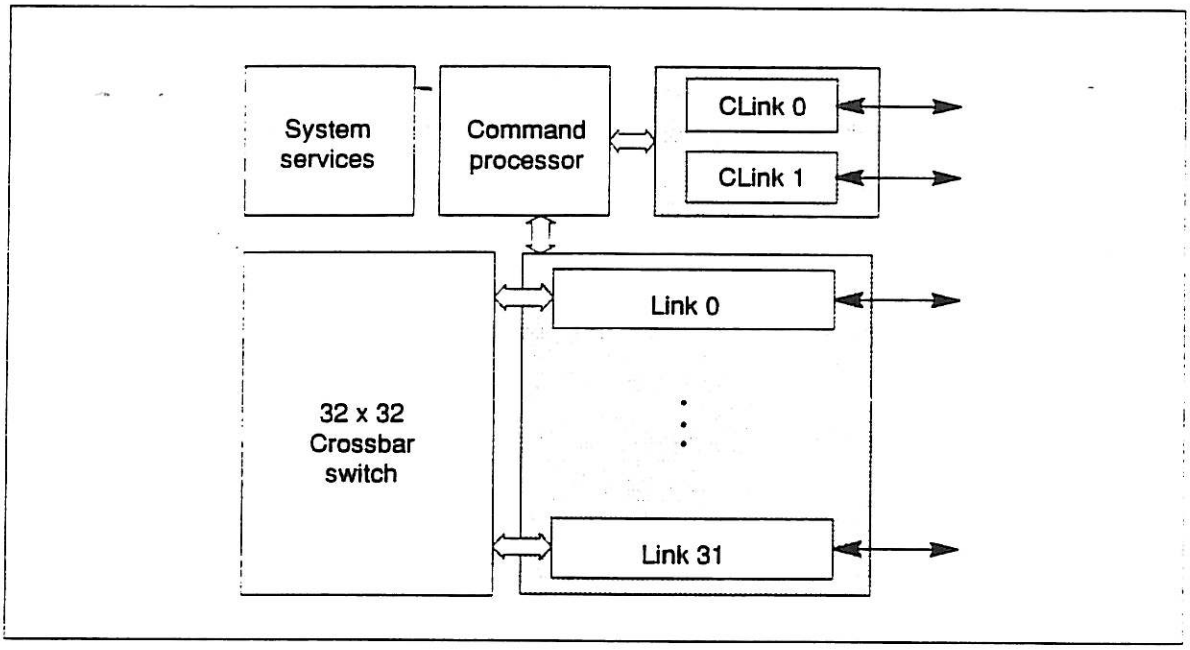


Figure 1.1 Block diagram of IMS C104

The IMS C104 allows communication between devices that are not directly connected. A single IMS C104 can be used to connect up to 32 devices. The IMS C104 can also be connected to other IMS C104s to make larger and more complex switching networks, linking any number of devices that use the link protocol. Any number of IMS C104s can be cascaded to any depth.

The IMS C104 enables networks to be built which effectively emulate a direct connection between each of the devices in the system. In the absence of any contention for a link output, the packet latency will be less than 0.5μ second.

A message on a IMS C104 communication system is transmitted as a sequence of packets. To ensure that packets which are parts of different messages can be routed, each packet contains a header. The IMS C104 uses the header of each packet arriving to determine the link to be used to output the packet. Anything after the header is treated as the packet body until the packet terminator is received. This enables the IMS C104 to transmit packets of arbitrary length.

In most packet switching networks complete packets are stored internally, decoded, and then routed to the destination node. This causes relatively long delays due to high latency at each node. To overcome this limitation, the IMS C104 uses *wormhole routing*, in which the routing decision is taken as soon as the routing information, which is contained in the packet header, has been input. Therefore the packet header can be received, and the routing decision taken, before the whole packet has been transmitted by the source. A packet may be passing through several nodes at any one time. Thus, latency is minimized and transmission can be continuous.

The term *wormhole routing* comes from the analogy of a worm crawling through soil, creating a hole that closes again behind its tail. Wormhole routing is invisible as far as the senders and receivers of packets are concerned, its only effect is to minimize the latency in message transmission.

The routing algorithm which makes the routing decision is called *interval labeling*, which is complete, deadlock free, inexpensive and fast. Each destination in a network is labeled with a number, and this number is used as the destination address in a packet header. Each link in a routing switch is labeled with an interval of possible header values, and only packets whose header value falls within that interval are output via that link.

The IMS C104 is controlled and programmed via a control link. The IMS C104 has two separate control links, one for receiving commands and one to provide daisy chaining. The control links enable networks of IMS C104s to be controlled and monitored for errors. The control links can be connected into a daisy chain or tree, with a controlling processor at the root.

The IMS C104 contains a hardware mechanism to allow independently programmed networks to be connected together. It also has additional circuitry to maximize the utilization of links operating in parallel, and to reduce the impact of message congestion on worst–case latency and bandwidth in heavily loaded networks.

The IMS C104 can be configured as a number of separate logical devices, whose attributes can be set independently.

A set of tools will be available to support the configuration of IMS C104 systems. The tools will provide support in the configuration and initialization of networks consisting of IMS C104 routing switches.

2 Operation of IMS C104 networks

A single IMS C104 can be used to connect up to 32 devices. The IMS C104 can also be connected to other IMS C104s to make larger and more complex switching networks, linking any number of devices that use the link protocol.

The IMS C104 uses a 1 or 2 byte header of each packet arriving, to determine the link to be used to output the packet. Bytes following the header are treated as the data section of the packet until a packet termination token is received. This enables the IMS C104 to transmit packets of **arbitrary** length.

An IMS C104 network consists of one or more IMS C104 routing devices connected together by bi–directional DS links. Each device is called a node of the network. Some links of the network are connected to the exterior of the network, to processing devices or to another network. These links are called terminal links.

In order to support the efficient routing of packets through a network the IMS C104 implements a complete routing algorithm in hardware. The component parts of the algorithm are described in the following sections.

2.1 Wormhole routing

In most packet–switching networks each routing switch inputs the whole of a packet, decodes the routing information, and then forwards the packet to the next node. This is undesirable in device networks because it requires storage for packets in each routing switch and it causes long delays between the output of a packet and its reception.

The IMS C104 uses *wormhole routing* (figure 2.1) in which the routing decision is taken as soon as the header of the packet has been input. If the output link is free, the header is output and the rest of the packet is sent directly from input to output without being stored. If the output link is not free the packet is buffered. The packet header, in passing through a network of IMS C104s, creates a temporary circuit through which the data flows. As the end of the packet is pulled through, the circuit vanishes. The wormhole analogy is based on the comparison with a worm crawling through sandy soil, which creates a hole that closes again behind its tail.

The implications of wormhole routing are that a packet can be passing through several IMS C104s at the same time, and the head of the packet may be received by the destination before the whole packet has been transmitted by the source. Thus latency is minimized and transmission can be continuous.

Wormhole routing is invisible as far as the senders and receivers of packets are concerned. Its major effect is to minimize the latency in the message transmission.

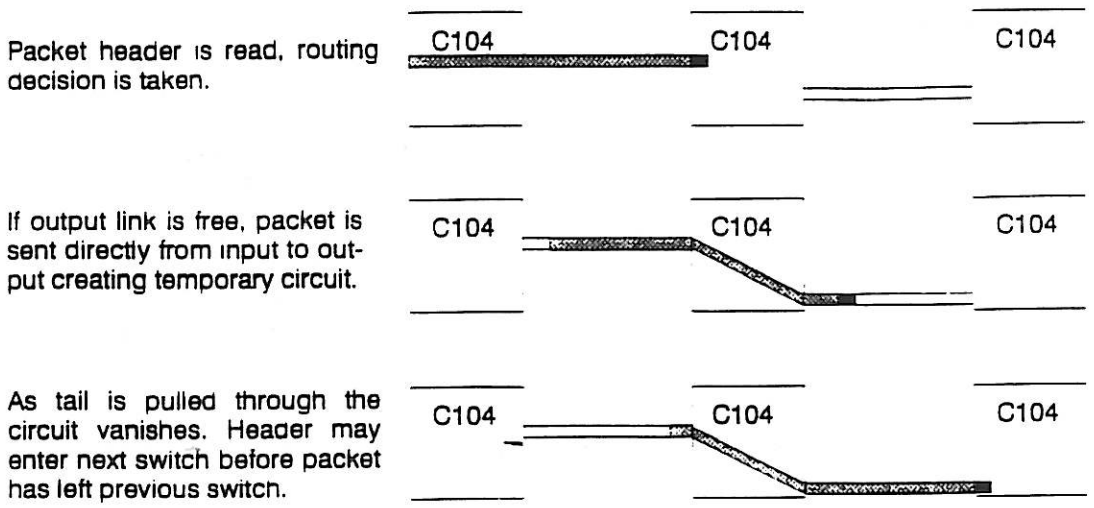


Figure 2.1 Wormhole routing

2.2 Interval labeling

Wormhole routing requires a routing strategy to decide which link a packet should be output from. The IMS C104 uses a routing scheme called *interval labeling*, whereby each output link of an IMS C104 is assigned a range, or interval, of labels. This interval contains the number of all the terminal nodes (i.e. device, gateway to another network, peripheral chip, etc) which are accessible via that link. Each terminal link of a network has an associated interval of labels. On entering a network the packet header contains a label. The label determines which link the packet is to be output to and hence must occur within the interval associated with the destination link.

As the packet arrives at an IMS C104 the selection of the outgoing link is made by comparing the header value with the set of intervals, as in the example shown in figure 2.2. The intervals are contiguous and non–overlapping and assigned so that each header value can only belong to one of the intervals. The output link associated with the interval in which the header value lies is the one selected. In the example the incoming header contains the value 154, which lies between 145 and 186, so the packet is output along link 8.

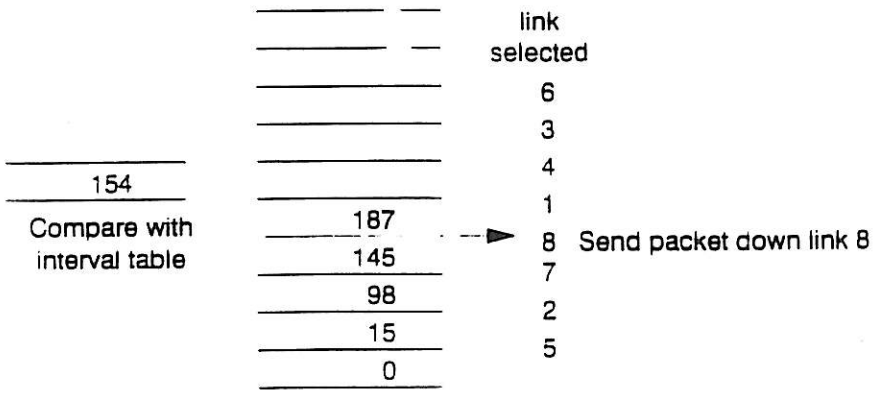


Figure 2.2 Interval labeling

Figure 2.3 gives an example of interval routing for a network of two IMS C104's and six devices showing one virtual link per device. The example shows six virtual channels, one to each device, labeled 0 to 5.

The interval contains the labels of all virtual channels accessible via that link. The interval notation [3,6) is read as meaning that the header value must be greater than or equal to 3 and less than 6. If the progress of a packet with the header value 4 is followed from $Device_1$ then it is evident that it passes through both IMS C104s before leaving on the link to $Device_4$.

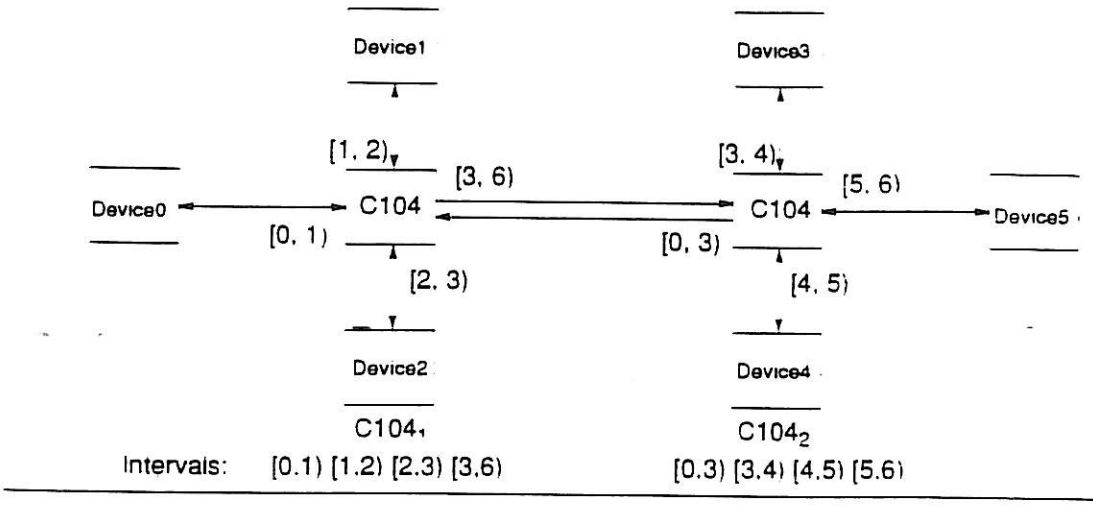


Figure 2.3 Interval routing

It is possible to label all the major network topologies such that packets follow an optimal route through the network, and such that the network is deadlock free. Optimal, deadlock free labelings are available for grids, hypercubes, trees and various multi–stage networks. A few topologies, such as rings, cannot be labeled in an optimal deadlock free manner. Although they can be labeled so that they are deadlock free, this is at the expense of not using one or more of the links, so that the labeling is not optimal. Optimal deadlock free labelings exist if one or more additional links are used.

Interval routing ensures that each packet takes the shortest route with low control overhead, and that all packets reach their destinations. It is independent of network topology and the output link selected is independent of the input link used. The transfer of a packet between one pair of links does not affect the data rate for another packet passing between a second pair of links. The hardware required to implement interval routing is simple, enabling many routing decisions to be made concurrently, thus providing a high rate of packet processing.

2.3 Modular composition of networks

To assist in the modular composition of routing networks the IMS C104 contains a hardware mechanism to implement *header deletion*. Header deletion mode is where each link output of the IMS C104 can be programmed to delete the header of a packet before transmitting the remainder of the packet.

The benefits achieved by header deletion are:

1 Simplified labeling of systems, by separating out the task of labeling networks from that of identifying virtual channels on devices.

2 Removal of the limit of a maximum of 64K virtual channels per system.

3 Hierarchical composition of networks.

Figure 2.4 illustrates how header deletion is used to simplify the labeling of systems by separating out the task of labeling networks from that of identifying virtual channels on devices. Figure 2.4(a) shows a system of 256 devices connected by a network of IMS C104s. All of the link inputs in the system are programmed to receive 2 byte headers. The IMS C104 interval routing tables and virtual channel headers are programmed to support 256 virtual channels connected to each device, with the header values allocated as shown in figure 2.4(a).

Figure 2.4(b) shows the same system but with all the link inputs in the system programmed to receive 1 byte headers, and with the terminal links of the IMS C104 network programmed to delete headers. A packet is now transmitted with a header consisting of two 1 byte sub–headers. It should be noted that as far as the IMS C104 is concerned the packet has just one header, any subsequent sub–headers are treated as part of the data body of the packet. The first 1 byte sub–header routes the packet across the network to the terminal link which the packet is to be sent out of: the terminal links being numbered from 0 to 255 as shown. This header is deleted as the packet leaves a terminal link of the network. The second 1 byte sub–header is then exposed, and is interpreted by the destination device to identify the target virtual channel.

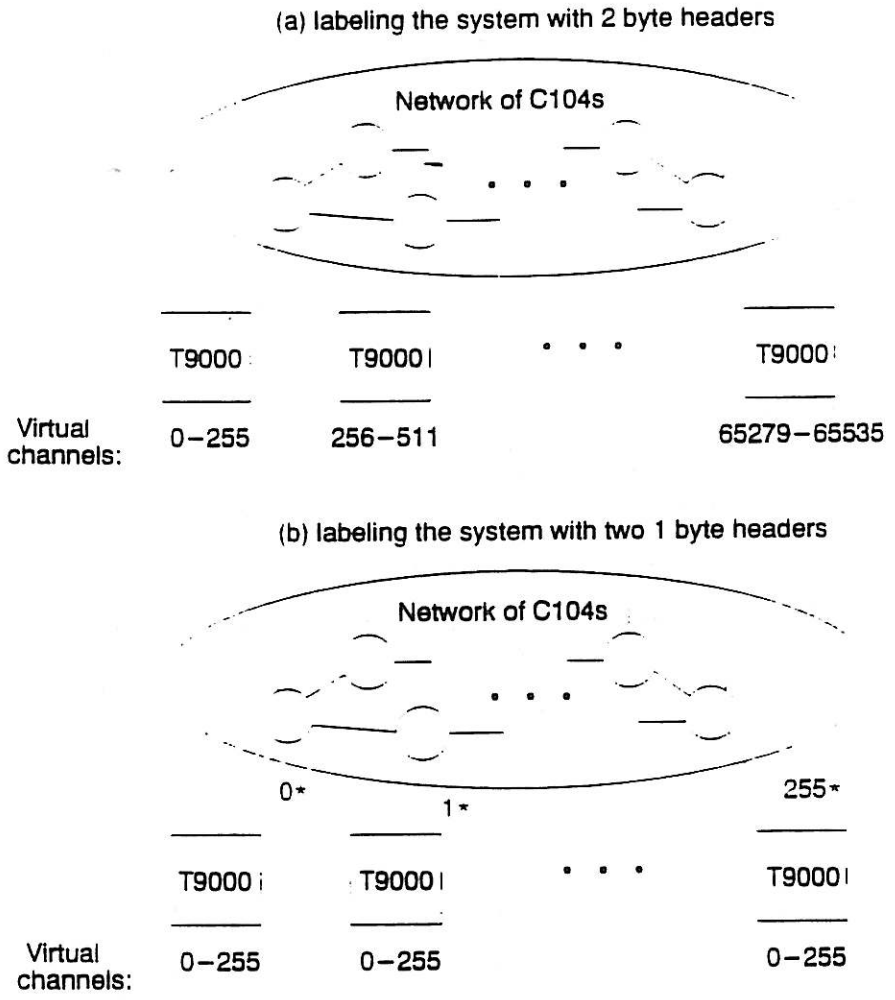


Figure 2.4 Header deletion used to separate network labeling and virtual channel identification.

In this manner header deletion allows network routing information to be separated out from the identification of virtual channels on devices. A first header is used to route the packet across a network to a terminal link, and a second header is used to identify a virtual channel within the destination device. The use of two 1 byte headers also decreases latency.

The total number of virtual channels in the system shown in figure 2.4 has not been increased, as headers are still 2 bytes long in total. However, the total number of virtual channels in the system can now be increased by programming the links on the devices to accept 2 byte headers (whilst the IMS C104s still accept 1 byte headers).

In this case a packet is transmitted with a header consisting of a 1 byte sub–header and a second 2 byte sub–header. As before, the first 1 byte sub–header routes the packet across the network and is deleted

as the packet leaves a terminal link of the network. Thus exposing the second 2 byte sub–header which allows 64K separate virtual channels to be identified on the destination device. Header deletion thereby removes the limit of 64K virtual channels in a total system, and replaces it with the less constraining limit of 64K virtual channels on each device.

Header deletion also allows networks to be connected together, as shown in figure 2.5. In this example a packet is routed through two networks and then to a virtual channel on an device. All of the terminal links of the two networks are set to header deletion mode. Figure 2.5 shows the header as it is routed through the network. The header of the packet in this case is made up of three concatenated sub–headers. The first sub–header routes the packet across the first network and is deleted as the packet leaves the terminal link of the network. The second sub–header routes the packet across the second network in the same way. Finally the third header is exposed to identify the destination virtual channel on the device.

In the case in which each IMS C104 is treated as a separate network and has its link outputs set to header deletion mode, packets can be explicitly steered across a network. This is at the expense of having 1 byte of header for each IMS C104 traversed.

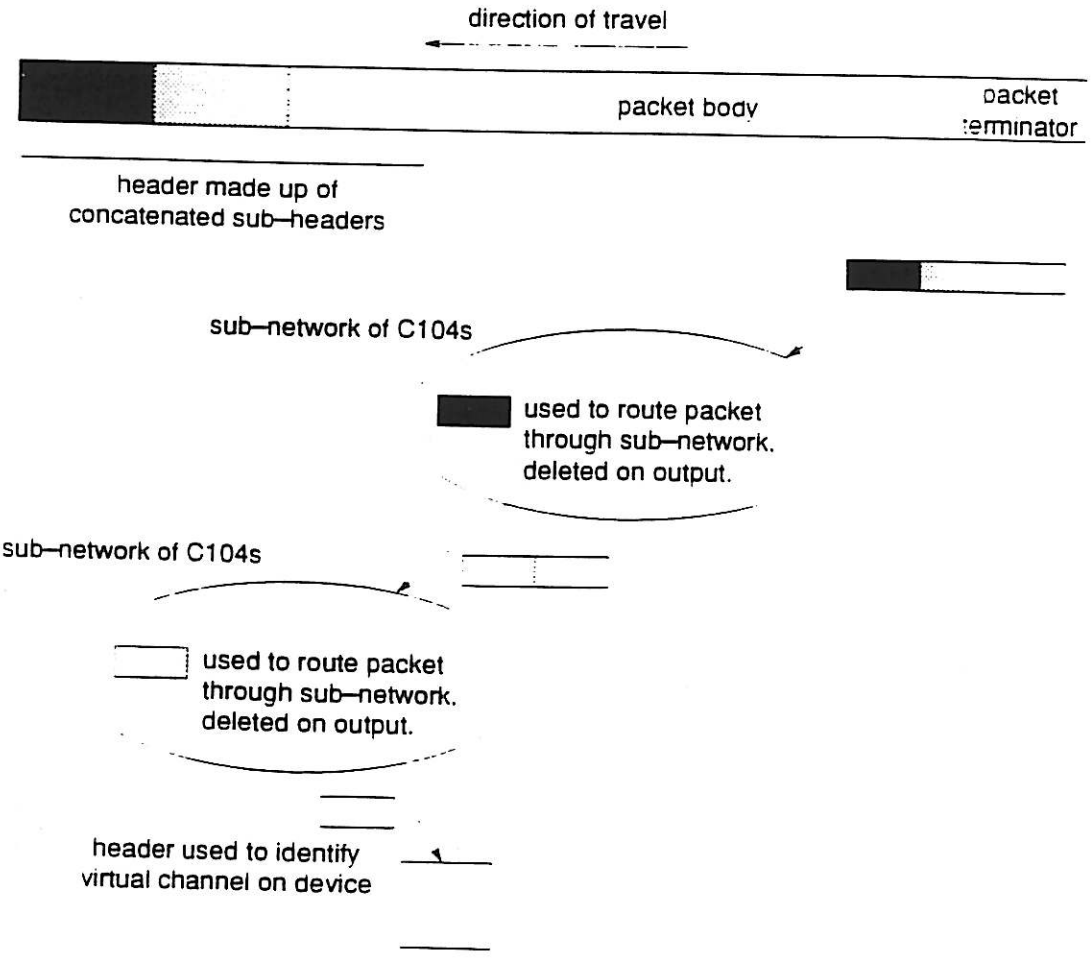


Figure 2.5 Hierarchical composition of networks using header deletion

A major advantage of extending the capabilities of the IMS C104, through header deletion, is that headers can be minimized for small systems, thus optimizing network latency and network bandwidth, whilst still enabling more complex, larger, systems to be constructed efficiently.

2.4 Use of parallel networks

System wide communication can be provided by connecting each device to a single routing network via one or more of its links. However, as each device has several links it can be connected to several different

networks. These can be completely distinct networks, or simply logical sub–networks of one network of IMS C104s. The use of multiple networks can provide the following:

- Higher available processor to processor bandwidth.
- Separate networks for different priority messages. The link protocol does not provide any support for associating a priority with a packet. This can be supported by providing a separate network for each required message priority.
- Separate networks for identified concurrent data streams in a system designed for a specific application.

2.4.1 Grouped adaptive routing

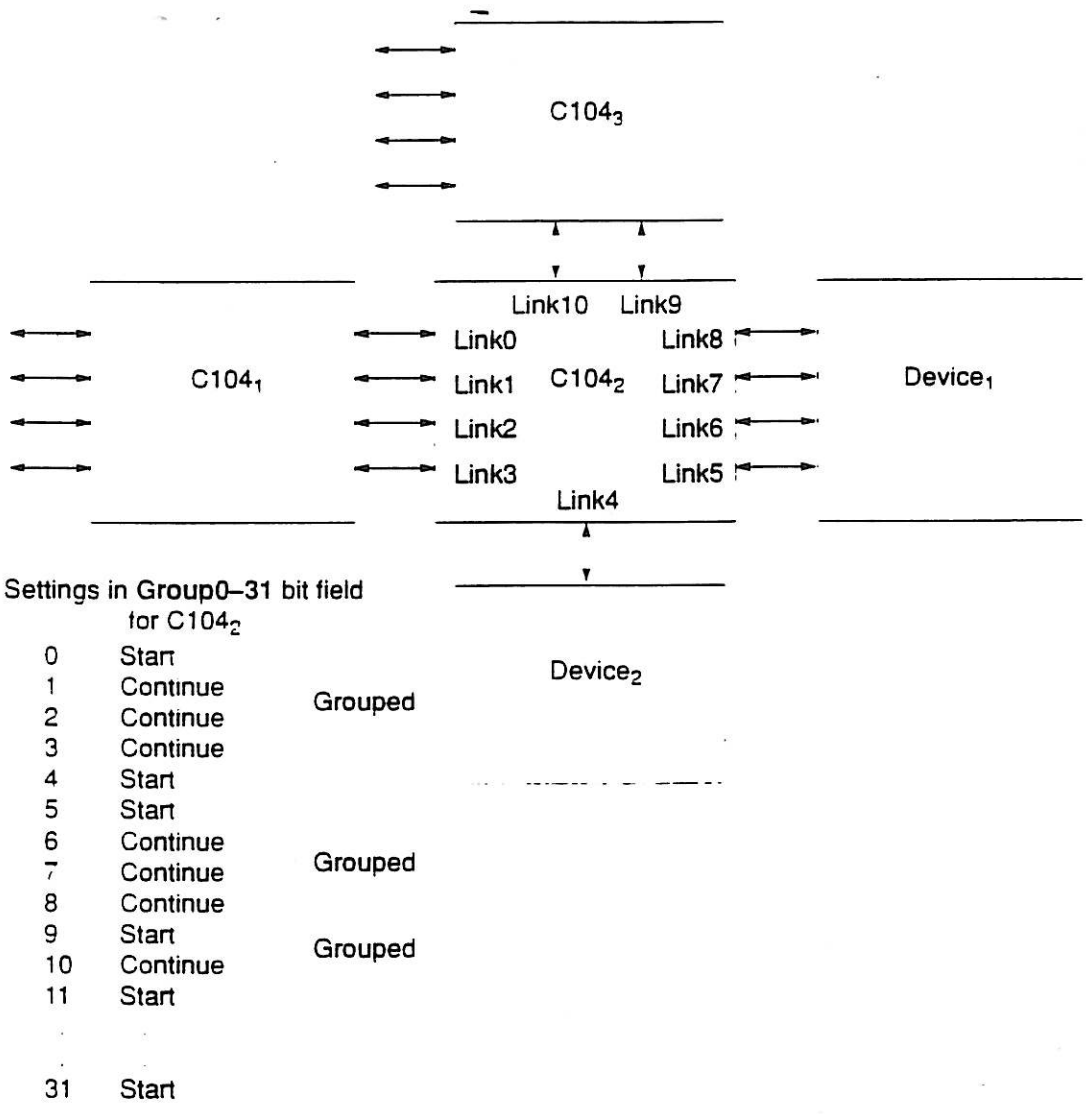


Figure 2.6 Grouped adaptive routing

The IMS C104 can implement *grouped adaptive routing*. Sets of consecutive numbered links can be configured to be grouped, so that a packet routed to any link in the set would be sent down any free link of the set. This achieves improved network performance in terms of both latency and throughput.

Figure 2.6 gives an example of grouped adaptive routing. Consider a message routed from $C104_1$, via $C104_2$, to $Device_1$. On entering $C104_2$ the header specifies that the message is to be output down Link6 to $Device_1$. If Link6 is already in use, the message will automatically be routed down Link5, Link7 or Link8, dependent on which link is available first. The links can be configured in groups by setting the Group0–31 bit fields. Each bit corresponds to a link and can be set to 'Start' to begin a group and 'Continue' to be included in a group, as shown in figure 2.6.

2.5 Hot spot avoidance

The routing algorithms described so far provide efficient deadlock free communications and allow a wide range of networks to be constructed from a standard router. Packets are delivered at high speed and low latency provided that there are no collisions between packets travelling through any single link.

Unfortunately, in any sparse communication network, some communications patterns cannot be realized without collisions. A link over which an excessive amount of communication is required to take place at any instant is referred to as a *hot spot* in the network, and results in packets being stalled for an unpredictable length of time.

To eliminate network hot spots, the IMS C104 can optionally implement a two phase routing algorithm. This involves every packet being first sent to a randomly chosen intermediate destination: from the intermediate destination it is forwarded to its final destination. This algorithm, referred to as *Universal Routing*, is designed to maximize capacity and minimize delay under conditions of heavy load. (This has been proven by simulations and theory. Refer to *'A scheme for fast parallel communication' SIAM J. of Computing, 11 (1982) 350–361).* It trades this off against best case performance in an empty network.

To implement two phase routing each packet must have a 'random' header prepended to it as it enters the randomizing network, which indicates its intermediate destination. This is implemented on the IMS C104 by enabling each input link to be programmed into a *random header* generation mode. In this mode the input link adds a random header to the front of each packet that it receives. The random header is generated from within a programmed range. The IMS C104 then treats this random header as the header of the packet, (the destination header is now treated as part of the data body of the packet), and routes the packet accordingly. The packet is routed on through the network until it reaches its random intermediate destination where the first phase of routing terminates.

Each IMS C104 link recognizes a range of *portal* values. The portal values set the random phase routing interval. This interval is compared with each arriving header. Any packet with a header within this interval will be recognized by the IMS C104; the random header will be deleted; and the header that is exposed is used to route the packet through the network to its final destination.

Note that the deletion of the random header associated with universal routing is different to that of the operation of header deletion mode, as described in section 2.3 above. Header deletion mode deletes headers as the packet is sent along a link output, whereas header deletion associated with universal routing occurs when the random header of the packet input into the IMS C104 is recognized to be within the portal range.

In order to ensure that deadlock does not occur the two phases of routing must use completely separate links. This is achieved by assigning destination headers and random headers from distinct intervals. All links in the network must be considered to be either *destination* or *random* links. The intervals associated with a given link on a IMS C104 must be a sub–interval of the destination or random header range as appropriate.

Effectively this scheme provides two separate networks: one for the randomizing phase and one for the destination phase. The combination will be deadlock free if the separate networks are deadlock free.

Universal routing can be beneficially applied to a wide variety of network topologies, including hypercubes and arrays. There are a small number of network topologies where universal routing is not always beneficial, as it can prevent highly optimal routings through the network being utilized.

3 Control of the IMS C104

The IMS C104 is controlled and programmed via the control links. Messages sent to the IMS C104 allow its configuration registers to be set and read. The registers can be accessed via *CPeek* and *CPoke* command messages sent along the control links and control the interval selectors, the random number generators and the DS links.

3.1 Programmable parameters

Interval routing is achieved in the IMS C104 by interval selector units. An interval selector performs the routing decision for each packet. It consists of 35 base and limit comparators (see figure 3.1). Each comparator is connected to a pair of registers, except the lowest whose base is fixed at zero. Each register is connected to the limit of one comparator and the base of the next comparator, except the top register which is connected to the limit of the top comparator only. These registers must be programmed with a set of unsigned 16 bit values ascending from zero, thus the intervals are non–overlapping and each header value can only belong to one of the intervals. This sets the interval for each link. Any link can be assigned to any interval. The output of each comparator is connected to a register (**SelectLinkn**). The **SelectLinkn** register contains the number of the associated output link. The contents are sent to the address gate if the packet header is greater than or equal to the base and less than the limit value of the adjoining comparator.

The interval selector reads in the value of the header and the pre–programmed comparators determine the corresponding link address for output. Once the path through the crossbar is set the tokens are passed through until an EOP or EOM terminator token is detected.

Each link input of a IMS C104 can be set to random header generation mode by the **Randomize** flag. In random header generation mode the random header generator produces a header which is added in front of the existing header and is used to route the packet to a random node, thus implementing the universal routing algorithm.

The lower limit and range of the random number generator must be programmed into the **RandomBase** and **RandomRange** registers.

Associated with each interval is a flag, held in the **Discard0–34** bit field, which indicates which of the intervals is the portal. If the input header is indicated as belonging to a portal interval (i.e. the random header has reached its random intermediate destination) the 'Discard' signal is sent to the header buffer telling it to discard the header. In this case the output of the ladder of comparators is not sent to the crossbar and the next 1 or 2 bytes of data (dependent on the **HeaderLength** flag) is taken as the new header and is again processed using the interval labeling algorithm.

If the header is not flagged as the portal by the **Discard0–34** bits the 'No' signal is sent to the address gate, which then allows the address which is produced from the ladder of comparators to be sent out to the crossbar. If none of the flags **Discard0–34** is set, the portal mechanism is disabled.

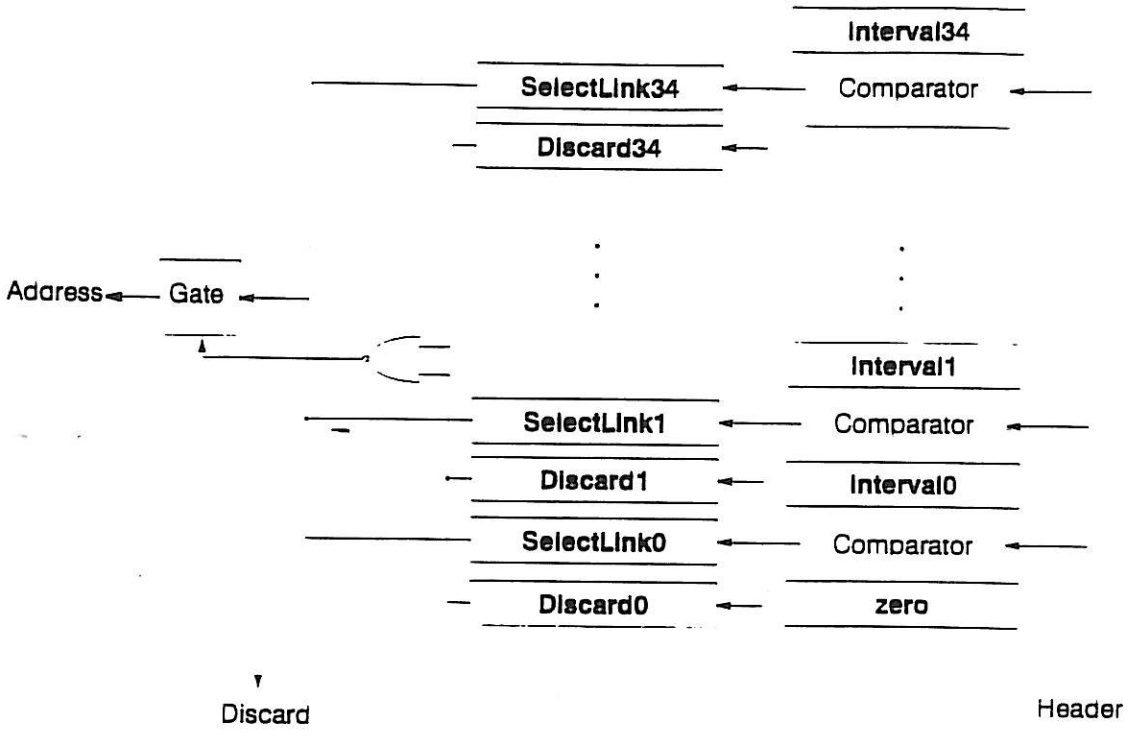
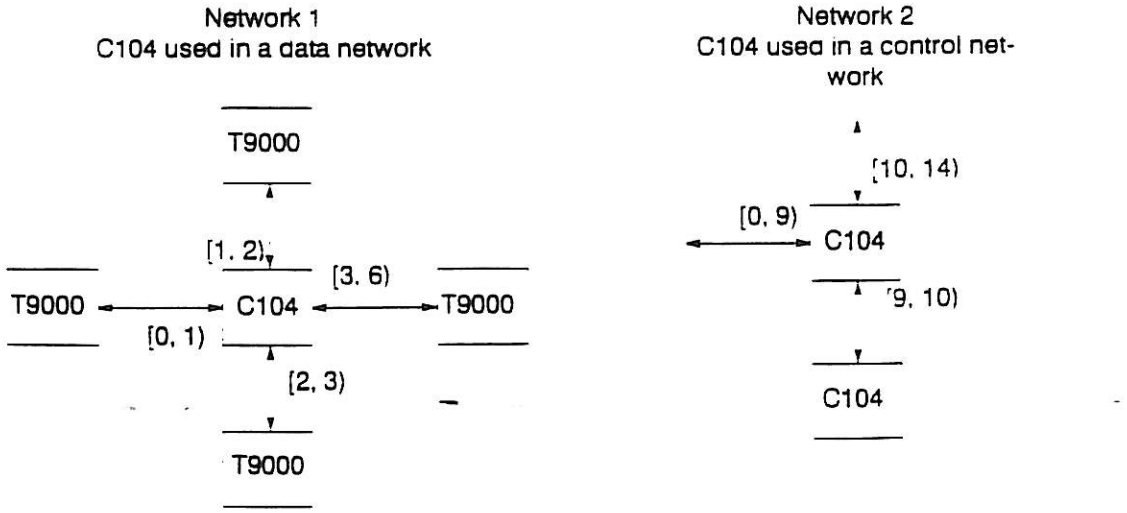


Figure 3.1 Interval selector registers

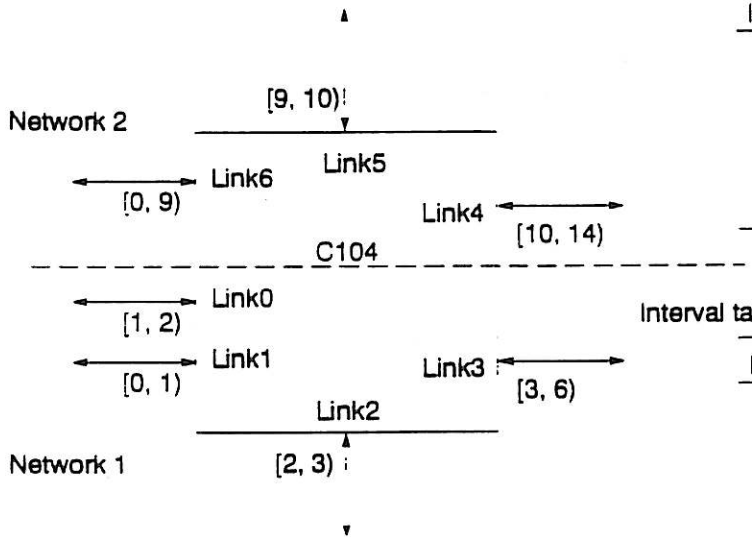
Each link can be set to input 1 or 2 byte headers. This is determined by the **HeaderLength0–31** flag in the configuration registers which are set after power on. It allows headers to be minimized for small systems, thus optimizing network latency and network bandwidth, whilst also enabling large homogeneous systems to be constructed. Heterogeneous and hierarchical systems can be implemented using hierarchical labeling and header deletion (which is implemented by setting the **DeleteHeader0–31** flag for a given link).

3.1.1 Partitioning

All the parameters described above are programmable on a per link basis. This enables an IMS C104 to be used as part of two or more different networks. For example, a single IMS C104 could be used for access to both a data network and a control network (see figure 3.2). Thus partitioning provides economy in small systems, where using an IMS C104 solely for the control network is not desired. It can also be used to enhance security in larger systems, if it is desired to prevent packets from one part of a system from being routed to another part.



Single C104 used between 2 networks



Interval table for links 4, 5 and 6

Interval	SelectLink
14	4
10	5
9	6
0	–

Interval table for links 0, 1, 2 and 3

Interval	SelectLink
6	3
3	2
2	0
1	1
0	–

Figure 3.2 Using partitioning to enable one IMS C104 to be used by two different networks

3.2 Registers

All the parameters described above are loaded into the appropriate registers by the command processor in response to commands received on the control link. The parameters must be supplied before the device can operate.

The functionality controlled by these registers is described below. The complete bit format of each register and the addresses of the registers are **not** included in this preliminary information.

Bit field	Function
HeaderLength	Sets the header length to 1 or 2 bytes
Randomize	Sets a given link input to random header generation mode
DeleteHeader	Sets a given link output to delete header mode

Table 3.1 Bit fields in the link configuration registers per link

Bit field	Function
Interval0–34	Sets the intervals for each link
SelectLink0–34	Indicates the associated link from which the packet is to be output
Discard0–34	Indicates which of the intervals is the portal

Table 3.2 Interval selector registers per link

Bit field	Function
RandomSeed	Start of 16 bit pseudo–random sequence
RandomBase	Base level of random number
RandomRange	Range of random number

Table 3.3 Bit fields in the random number generator registers per link

Bit field	Function
Group	Each bit can be set to 'start of group' or 'continuation of group'.

Table 3.4 Bit field to set grouped adaptive routing per link

4 Software

4.1 Configuration tools

A set of tools is available to support the configuration of IMS C104 systems. The tools will, among other things, provide support for the configuration and initialization of networks consisting of IMS C104 routing switches.

The tools will be able to set the attributes of each device in the network by sending initialization data down the control link, and will set the processors into a state ready to receive an application down the data links.

A Network Description Language (NDL) is used to describe networks of devices and the labeling of IMS C104s, and will allow the specification of values for all the attributes of a device.

The Network Description Language will support the following:

- declaration of processors, IMS C104 routing chips and their interconnections.
- specification of attributes for IMS C104 routing chips; including interval settings, header deletion and randomization characteristics.
- the construction of the control system, including chains of devices plus a predefined method of using the IMS C104 as a fan–out. It is possible to calculate the IMS C104 attributes (including interval values) for such devices used in the control system.
- desired message routing paths.

From the NDL file the initialization tools produce a file containing the network initialization data. This data is sent down the control link to the network.

Appendix E: DS–link macrocell, outline information

The INMOS DS–link macrocell is part of a family of communication products that support the INMOS data/strobe serial link (DS–link) protocol.

The INMOS DS–link consists of four wires, two in each direction, one carrying data and one carrying a strobe. Each link can operate up to 100 MBits/s, giving a bidirectional bandwidth of 20 MBytes/s. The DS–link protocol supports high bandwidth serial communications, virtual channels and dynamic message routing.

INMOS supplies a range of products based on the DS–link , including the IMS C104 packet routing switch, and the T9000 transputer, a 32–bit microprocessor that integrates four DS–links for senai communications. DS–link products are used to provide high bandwidth, low latency interconnect within computer systems, in applications as diverse as telecomms switching systems, local area networks, computer file servers and supercomputers.

The DS–link macrocell converts between an INMOS DS–link and two unidirectional parallel bus interfaces. It will be available in the SGS–THOMSON Semi–Custom Group macrocell library, enabling the user to implement the macrocell standalone as a general purpose link adaptor, or to incorporate the DS–link in a system ASIC. This gives flexibility in design allowing the user to optimise the system communication interface to suit the end application.

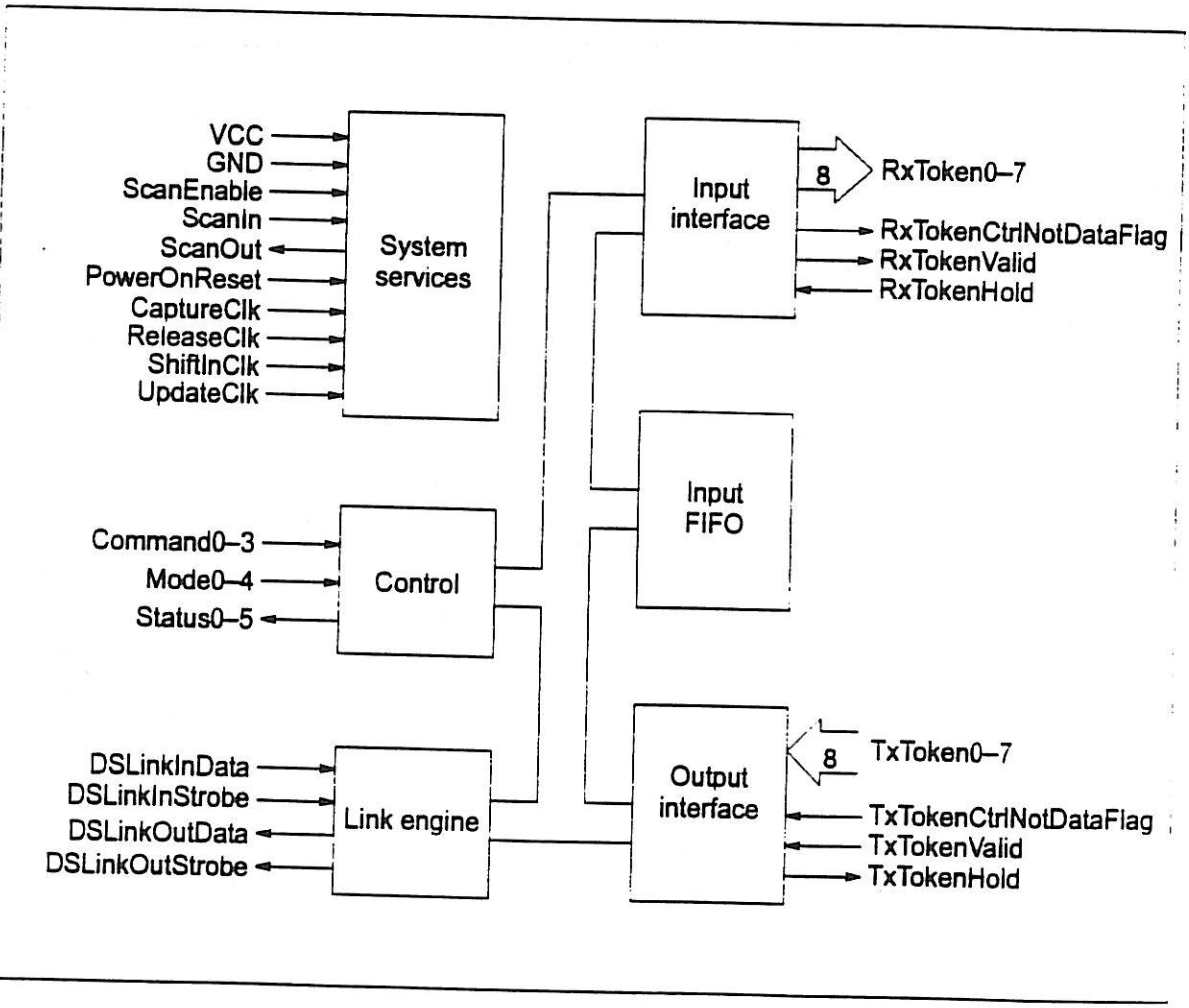


Figure 1.1 Macrocell block diagram

Preliminary pin designations

The following tables outline the function of each of the pins.

Pin	In/Out	Function
VCC, GND		Power supply and return
PowerOnReset	in	Power–on reset
CaptureClk	in	System master clock
ReleaseClk	in	System slave clock

Table 1.1 Macrocell system services

Pin	In/Out	Function
Command0–3	in	Command bus
Mode0–4	in	Mode bus
Status0–5	out	Status bus

Table 1.2 Macrocell control

Pin	In/Out	Function
TxToken0–7	in	8 bit parallel input bus
TxTokenCtrlNotDataFlag	in	Signals control tokens not data on bus TxToken0–7
TxTokenValid	in	Signals valid data on bus TxToken0–7
TxTokenHold	out	Goes low when the current valid data has been read by the Macrocell

Table 1.3 Macrocell output interface

Pin	In/Out	Function
RxToken0–7	out	8 bit parallel output bus
RxTokenCtrlNotDataFlag	out	Signals control tokens not data on bus RxToken0–7
RxTokenValid	out	Signals valid data on bus RxToken0–7
RxTokenHold	in	Goes low when the current valid data has been read by the host ASIC core

Table 1.4 Macrocell input interface

Pin	In/Out	Function
DSLinkInData	in	Link input data channel
DSLinkInStrobe	in	Link input strobe
DSLinkOutData	out	Link output data channel
DSLinkOutStrobe	out	Link output strobe

Table 1.5 Macrocell link

Appendix F: Example systems with DS-Links

Using DS-Links and the routers discussed it is possible to build a large range of systems both in terms of size (number of disks and processors), performance, and fault tolerance.

Five examples are discussed below to illustrate some of the advantages of DS-Link and router technology.

1 Single processor, five disks in a loop

In this first example the disks have two DS-Link interfaces and a router which will route between the disk and the two DS-Links. This is illustrated in figure 1. The DS-Link DMA interface and the three port router can be easily integrated into one low cost device.

This combination of DS-Link interfaces and routing on the disks allows a low cost loop disk array to be constructed. This is shown in figure 2. Although the network is a loop the network is tolerant to failures of single links or disks as the links are bi-directional and the other direction round the loop can be used. The two DS-Links coming from the processor can either be implemented using the same device that is used on the disks or by providing two complete Link DMA interfaces (again this could be one low cost device). Using 100 MBit/s links this system gives a bandwidth between the processor and the disks of either 20 or 40 MBytes/s depending on the processor DS-Link interface used.

The virtual channels between the processor and the disks allow the processor to communicate with all of the disks concurrently, with the network bandwidth being shared between them dynamically.

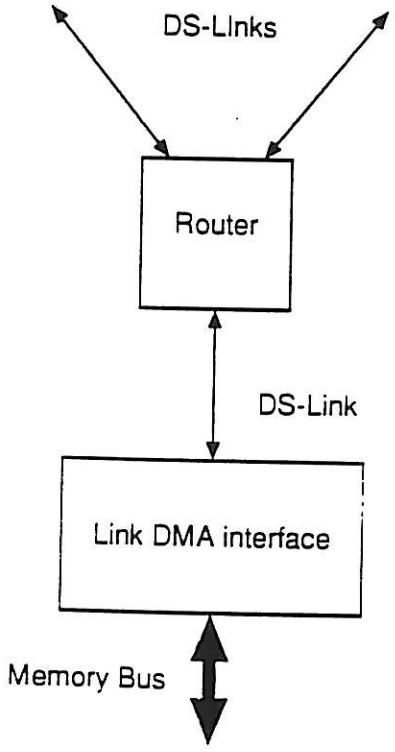


Figure 1 Three link router and link interface

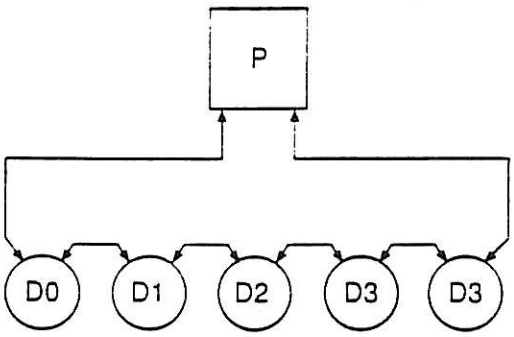


Figure 2 Example 1: Single processor, five disks in a loop

2 Single processor, four disks

In this example with a processor with four DS links and four drives with DS link interfaces the disks can be connected directly to the links from the computer without the use of a router. This system gives a bandwidth between the computer and the disks of 80 MBytes/s. The computer can communicate with all of the disks concurrently at 10 MBytes/s in each direction.

It is interesting to compare this with a parallel SCSI system design. Assuming the use of 16 bit fast SCSI on the drives the total system bandwidth would be only 20 MBytes/s, half-duplex and would require 64 wires. To achieve the same bandwidth would require four SCSI controllers in the processor and would need 256 wires to connect to the drives.

In contrast the DS-Link system uses a single interface device at the computer end and requires four 8-wire, cables to connect to the drives.

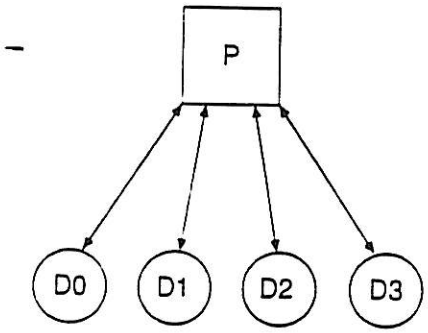


Figure 3 Example 2: Single processor, four disks

3 Single processor, sixteen disks

This example illustrates a simple use of the router, to provide fanout to a larger number of disks than the processor has links.

In this case the bandwidth to and from the processor is again 80 MBytes/s, 10 Mbytes/s in each direction on each link. However this bandwidth can be shared between any number of virtual channels going between the processor and the disks over the four links from the processor. It is possible for the processor to be transferring data to and from the all the disks concurrently. The low overheads imposed by the virtual channel communication and the concurrency possible mean that the performance in terms of I/Os per second can also be very high. The bandwidth across the links from the disks to the router is 320 MBytes/s. Third-party copies (disk to disk copies) over virtual channels set up between the disks could be performed without significant impact on the data rates to and from the processor. These disk to disk virtual channels would share the bandwidth of the disk to router links with any other virtual channels to and from the processor.

Note that the router itself is very high performance being capable of routing 200 Mpackets/s. less than 1 us packet latency, and a through-switch bandwidth of 320 Mbytes/s.

Not shown in the diagram is the low speed DS link to the router to allow the configuration of the router to be controlled. This link could come from processor in this example.

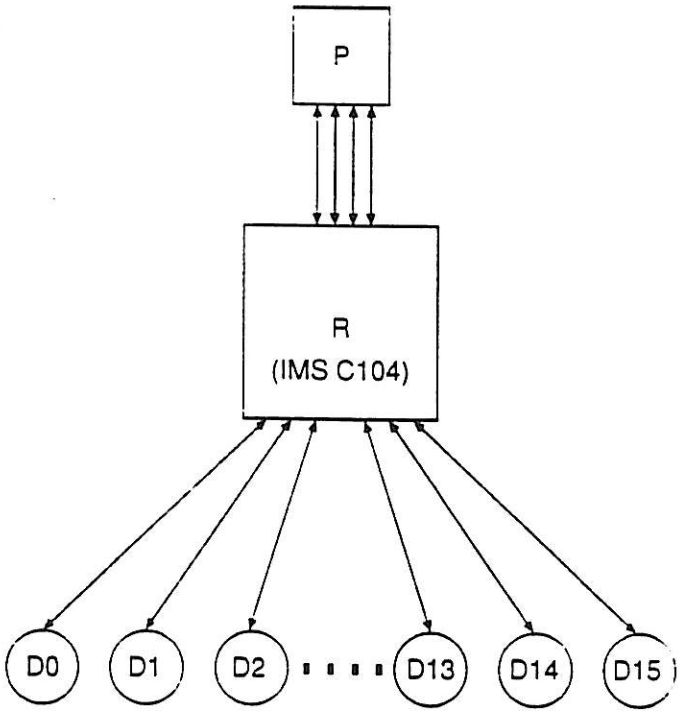


Figure 4 Example 3: Single processor, sixteen disks

4 Four processors, twenty four disks, fault tolerance

The cost of providing dual porting on disks using a DS-Link interface is low since the number of device pins used and the silicon area is low. Adding dual porting to the disks allows a fault tolerant network of disks and processors to be constructed.

In the example shown two routers are used to allow the network to be tolerant to failure of one of the routers. Two links from each of the processors are taken to each router and one link from each drive. The network is tolerant to link failures on both the processor to router links and the disk to router links as there are alternative paths. These alternative paths can either be set up statically at system initialisation or alternative paths can be created by reconfiguring the routers. Note due to the point-to-point nature of links any hard failure of one link cannot affect any other link (unlike a bus structure which is prone to single point failure).

Failures of drives and processors can also be tolerated using RAID architectures and similar redundancy techniques for the processors. Again the point to point nature of the links and the use of routers allows failed units and interfaces to be isolated and facilitates live insertion and withdrawal of units.

The performance of the network illustrates the scalability of the DS-Link and router technology. Bandwidth to and from the processors is 320 MBytes/s, to and from the disks 960 Mbytes/s. and the through switch-bandwidth 640 MBytes/s.

Again the configuration links to the routers are not shown. The method of configuring the routers would depend on the degree of fault tolerance required and the strategy employed to cope with link, router, or processor failures.

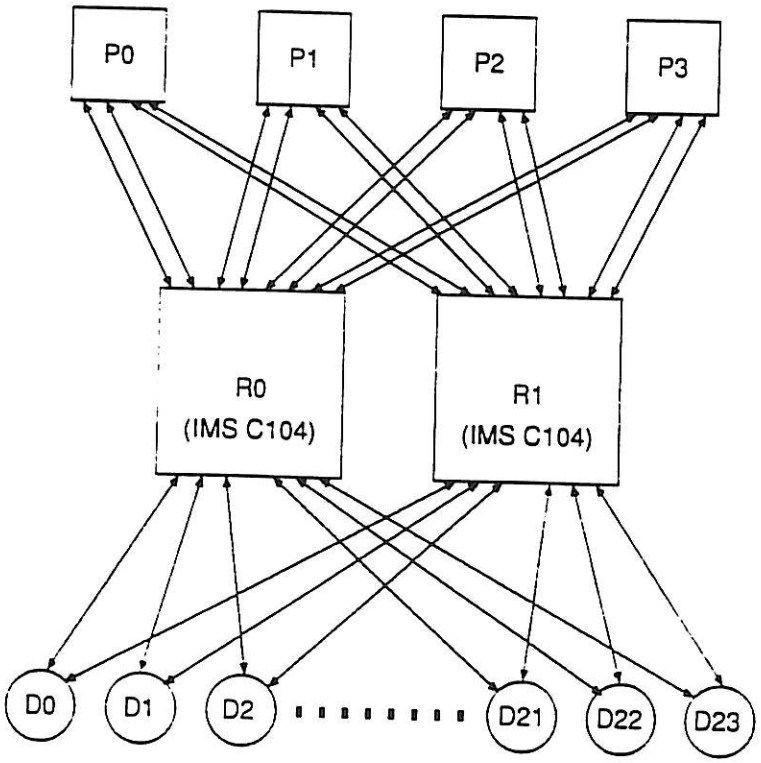


Figure 5 Example 4: Four processors, twenty four disks, fault tolerance

5 Large system, 1024 nodes (processors, disks, or other services)

This example shows that it is possible to construct very large systems using the DS-Link and router technology. The network of routers and nodes is not drawn for obvious reasons and because the network employed will depend on the application. One possible network that has been studied at INMOS for large networks is the hypercube or binary n-cube. Studies of this network have shown the performance scalability and low latency available. In this size of network the worm-hole routing and universal (random) routing features of the IMS C104 router were key to the performance of the network.

Some of the headline figures for a 10-cube network (1024 nodes) are a total network bandwidth of 1024 million, 32 byte packets per second (32768 MBytes/s) with a mean latency of 46 us for a packet and a maximum latency of 303 us.

Appendix G: Experience and benefits of using links

This appendix describes SGS-THOMSON Microelectronics' experience with high speed serial interconnect, as a result of its INMOS Division having developed the transputer family, with their inter-processor communications links. Experience of both INMOS and users is described, showing how links are used to achieve benefits in terms of performance, fault-tolerance, breadth of application, and cost.

Over-sampled links (OS-Links)

INMOS has been developing inter-processor serial interfaces since 1979 and has shipped millions of OS-Links with the T400 and T800 family of transputers. OS-Links have been used in almost all sectors of the computer, telecomms and electronics markets. Many of these links have been used without transputers, or with a transputer serving as an intelligent DMA controller for other processors.

Each OS-Link provides a physical point-to-point connection for a software channel between two processes, each process running in a separate processor. The links are full-duplex, run at 10 or 20Mbits/s, have an exceptionally low implementation cost and an excellent record for reliability and fault-tolerance.

Data-Strobe links (DS-Links)

DS-Links have been evolved from the OS-Links and benefit from the experience gained with OS-Links, both within the corporation and by our customers. Speed is increased to 100Mbits/s, with 200Mbits/s planned and substantial enhancement possible. The DS-Links provide a physical link for any number of multiplexed software channels; these can either be in the same two processors, if the link is directly connected between the two processors, or can be in any number of different processors, if the links are connected via (packet) routing switches. Error detection is added to detect and locate the most likely errors.

The fact that DS-Links are targetted towards processor to processor communication may imply that they are less appropriate for some of the specialised applications such as disk drives, disk arrays, or communication systems. On the other hand, as almost all equipments and PCBs within those equipments contain processors, it is possible that the model of processor to processor communication is actually more appropriate than some alternative interfaces.

The benefits of the processor to processor model of communication used by the DS-Links are in terms of performance (particularly of the system rather than each device), fault-tolerance, and cost, as well as the remarkable breadth of use shown by the use of OS-Links.

System Performance:

Performance benefits are particularly felt in terms of system performance, resulting especially from the use of routing switches.

> A small system with, say, four disks and one processor can have system bandwidth of 400Mbits/s (all units with two links connected in a full-duplex ring) or 800Mbits/s (processor with four links, disk drives with one link each).
>
> A medium system with sixteen processors and 32 disks, with all units dual-ported and connected by a pair of 32-way routing switches (C104s) provides a system bandwidth of 480MBytes/s to and from the processors.
>
> A large system with 256 or 1024 nodes, each node being a processor, disk, or communications port, and each node having four DS-Links: such a system would achieve a high utilization of each link at each node, even if every link was trying to operate at the same time. (Systems have been already built with 1000 or more transputers, connected by OS-Links; the virtual channels and routing switches of DS-Links make such huge systems easier to build and much higher performance)

Factors which contribute to this scalable performance, from very small systems to huge systems, are:

Having many communications taking place at the same time, not having to share access to a single bus or token ring;

Having (packet) routing switches, so that routing can be performed by hardware rather than by software;

Using Worm-hole routing, which forwards a packet as soon as the header has been received, rather than store-and-forward routing which implies much longer delays in each routing switch than worm-hole routing;

Avoiding hot-spots with grouped adaptive routing, so that if one path through a router is blocked, an alternative path may be used;

Further avoiding hot-spots with universal routing, which avoids hot-spots even when all the traffic wants to take a single route.

Having sufficiently low overheads on packets and messages that high throughput is achieved even with very short messages such as might be used for control; throughput is not dependent on all messages being many kBytes.

Fault-Tolerance

Fault-tolerance with DS-Links comes from the prevention of many possible fault mechanisms, from an end-to-end check appropriate for the application, from isolating faults to local areas for logging and maintenance, and from means of curing systems where faults may have occurred.

Error Prevention:

The Gray-coding of the DS signals has been reported by Bellcore to give substantially faster throughput and reduced design complexity, as well as greater reliability, than conventional clocked systems. (Anthony J McAuley 'Four State Asynchronous Architectures', IEEE Trans on Computers, V41 N0 2, Feb 92.).

This improved reliability derives, in part, from the fact that a whole bit-time is available for skew tolerance between the regenerated clock and the data signal.

The signals are designed, like logic signals, to operate correctly, without errors, in a clean environment where the electrical specifications are met.

If a particular link is more susceptible to noise for any reason, the routing switches can be arranged to construct a firewall around that link, so that whatever routing header is received, the packet is sent to a node which includes hardware or software to check for the more likely errors from the suspect link.

The routing algorithm used can guarantee the avoidance of network deadlock, without relying on the amount of buffering, the size of network, or the application program.

Many users have reported that OS-Links are very conservatively specified; INMOS experience with a network of OS-Links, buffered by RS422 buffers and covering close to 100 metres, is that software errors are far more likely to cause errors, although the 100 metre connection far exceeds the OS-Links specifications or recommended use.

Associated with both OS and DS links is the process model of communication which was derived from the structured programming methodologies designed to prevent a large proportion of common software errors. The particular process model is that of CSP (Communicating Sequential Processes) which was used as the basis for communication in the Ada language.

A Technical Note from SGS-THOMSON, (Roger Shepherd, 'Security Aspects of occam

Programming, INMOS Limited) describes some of these issues, which are becoming more important as corporations become more dependent on their software.

The process model bears much resemblance to that used in the Total Quality approach, which is also aimed at eliminating errors. (Don Jackson Company, Quality Service: a seminar prepared for SGS-THOMSON Microelectronics', p27.).

Error Detection:

Error Detection is often thought to imply a CRC, particularly on serial links. If detection is important, however, there must be no hazards where an error can occur between one check being checked and another being generated. (Such errors have been known to occur in communications bridges.)

There are some types of data where errors are unimportant, as long as they do not occur too frequently; for such data there should not be a CRC, because it would be an unnecessary expenditure. DS-Links therefore take the same approach to data CRC as is taken by ATM, and defer it to a higher level of protocol — which may choose to implement a CRC in either hardware or software.

It is clear that if a CRC is regarded as important for a particular application, the check should be fully end-to-end, it should cover the data for a whole message rather than an individual packet. it should cover the destination device ID, the destination channel ID, (possibly source IDs also) and the message length.

A check such as described is rigorous in that it detects missing packets or additional packets resulting from any routing errors, and is efficient because the device IDs and channel IDs do not need to be carried in the message payload.

If memory at either node of the communication is parity checked, the end-to-end check should overlap this parity check — there should not be an error hazard where the parity is stripped and the CRC generated or vice-versa.

Whether the check is a conventional CRC, a longitudinal parity check, a Reed-Solomon code or a Fire code, is dependant on the application and the cost and performance overhead that can be tolerated. SGS-THOMSON is willing to work with experts in particular application areas to define these checks.

Fault Isolation

The purpose of low-level checks is to reduce the probability that an error will need to be detected by the higher level end-to-end check, and to localise where the error occurred for maintenance purposes. The DS-Links include a number of such low-level checks.

Parity is included on the tokens transmitted on the links, to detect all single-bit errors.

All invalid tokens are detected.

There are range checks on the Routing Headers and on the Virtual Channel Headers.

There are length checks (under and over) on packets and messages.

The most likely hardware error of all, the cable being unplugged and the link therefore disconnected, is detected.

Cure of faulty systems

OS-Links are used in existing fault-tolerant systems in spite of the absence of any error checking on the OS-Links. The reason for this is that each transputer has at least two links and most have four links; networks can be constructed so that if one link fails, another can take its place. DS-Links build on this principle.

Dual-port nodes can have each link taken to a different Routing Switch: if one of the links fails, or one of the routing switches, the other link or routing switch is used instead. This is

analogous to disk mirrorring.

Charles Stark Draper Lab describe how the multiple links can be used to provide Triple Modular Redundancy (TMR) or higher levels of redundancy (nMR). (Neil Brock, 'Transputers and Fault-Tolerance', Transputing '91 proceedings, P Welch et al. Eds., IOS Press, 1991, pp462-475.)

Roke Manor Research describe how multiple links can be used to provide a system with 'n+1' redundancy, as in RAIDs and in some power supplies, and further to provide graceful degradation of performance in the event of multiple faults. (Richard Beton, James Kingdon, Colin Upstill, 'High Availability Transputing Systems', Transputing '91 proceedings, as above. pp 497-507.)

When a node or a switch is suspect, or needs to be replaced, it is possible for a system built with DS-Links to ostracise the suspect board so that it can not corrupt the rest of the system; to repair the fault, the board or equipment can be unplugged ('hot swapped') without any risk of causing glitches that affect the rest of the system. (Paul Walker, 'The Bus-less Computing Environment', BUSCON/West '92 proceedings, CMC, pp 549-558.)

Breadth of use of links

As in the previous section, there is considerable experience of using OS-Links to implement a wide range of higher level system protocols. The mapping of these protocols onto DS-Links can in many cases be identical to the mapping onto OS-Links, but in some circumstances it may be possible to simplify the protocol when using the multiplexed virtual channels of DS-Links, with their inherent addressing capability.

There is growing evidence that a wide range of mappings may be needed on interfaces within systems: examples are file-servers with multiple network interfaces (such as the Auspex NetServer) and the need to migrate intelligence closer to the disk system (John Wilkes (HP Labs), 'The Datamesh Research Project' Transputing '91, as above, pp547-553.)

ST506 has been implemented on the IMS M212 chip.

SCSI has been implemented by a number of companies, to provide dual-port or four-port access to a SCSI string, via two or four OS-Links.

Unix sockets have been implemented, also by a number of companies.

TCP-IP has been implemented.

Details of all the above are contained in product manuals of INMOS products available from SGS-THOMSON, as well as in documentation from other companies.

Database enquiries have been implemented on several systems. One is described by Sheffield University in (Richard J Oates, Jon M Kerridge, 'Adding Fault-Tolerance to a transputer-based Parallel Database Machine', Transputing '91, as above, pp449-461).

The similarity of DS-Link packets and ATM cells has prompted a paper (Catherine Barnaby and Neil Richards, 'A Generic Architecture for Private ATM Systems', to be presented at ISS, Tokyo, October 1992).

Cost

The implementation cost of links has to be low to allow four links to occupy a small fraction of the area of a transputer chip. This benefit is retained when the links are used independently of transputers. Further benefits accrue to the system costs.

The absence of coding, the absence of analog components for clock-recovery, the absence of engineering for long distances unless they are required, the very simple headers and checks, all contribute to the low implementation cost.

A less obvious contribution is from the bit-level signalling, which allows 100Mbits/s perfor-

mance with technology capable of a 50MHz clock. To achieve similar data rate with an embedded clock and clock-recovery would need a 100MHz clock to be precisely phase aligned with the incoming signal: in many respects the easiest way to do this phase alignment would be to use a 200MHz clock — compared with the 50MHz clock actually used.

Several of the system cost benefits result from the topologies made possible by routing switches. A bus or token ring, requiring a system bandwidth of 1Gbit/s would need that performance — and cost — at every node. Each node would need to be dual-port in order to support the bandwidth. A system with twenty nodes would offer each node an average bandwidth of 100Mbits/s, half-duplex. A corresponding system of twenty single-port nodes and a routing switch and each DS-Link 100Mbits/s full-duplex would offer a total system bandwidth of 2Gbits/s. Making the nodes dual-port and adding another router doubles the performance again and offers dual-redundancy of the links and routers.

The RAID paper and a note on power supplies by AT&T detail the cost (and performance) benefits of 'n+1' redundancy over dual redundancy or mirrorring. (David A Patterson, Garth Gibson, Randy Katz, 'A case for redundant arrays of inexpensive disks (RAID)', proceedings of SIGMOD (Chicago IL) 1-3 June 1988.) and (Kerry C Glover, 'A Standardized Module Approach to Power Conversion', Internal memorandum, AT&T Proprietary, 6/7/91.) As the paper from Roke Manor Research (referred to above) describes, these n+1 redundant systems are simply implemented with links.

It is possible that, as with the power supplies, substantial cost benefits may be available from a modular approach to the construction of link-based systems. Users of OS-Links have substantial experience of modular systems from the use of TRAMs (transputer modules). This experience has been used in the definition of HTRAMs for DS-Links (Paul Walker, 'Hard-Metric Mezzanine Board Standard' BUSCON/West '92 proceedings, CMC).

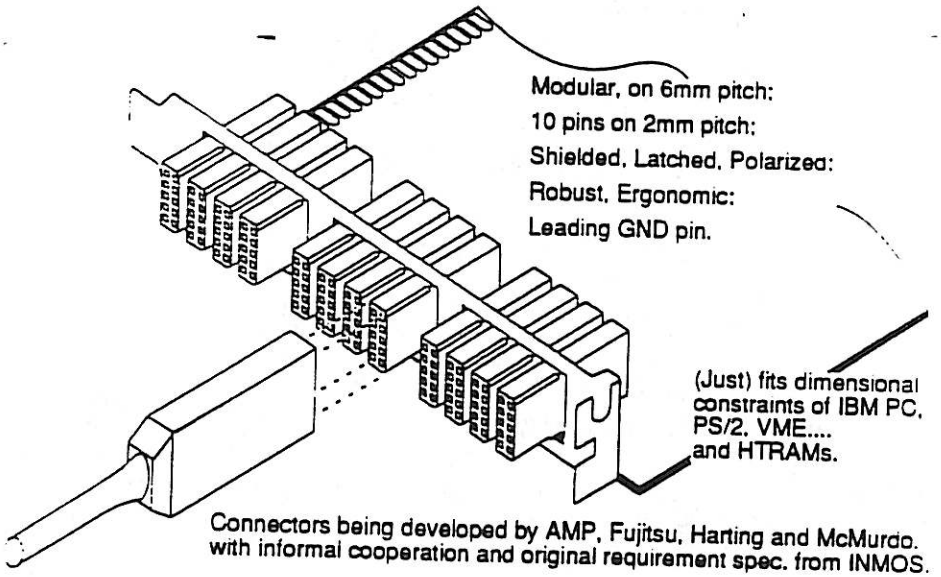
A final cost benefit comes from the process model used by links, which allows the same program to be mapped onto a variety of hardware configurations, sometimes a single processor, sometimes several processors. In terms of disk drives and systems, this would allow say a 1.3" drive with a single link and as much control program and electronics removed from the drive as possible; a 1.8" drive or 2.5" drive might be interfaced by two links, giving higher performance and dual redundancy of the interface. more of the control program would reside on the drive, but perhaps part of the cache and the filing system would remain in an external controller; a 3.5" drive might then have four links, giving yet more performance and more redundancy, and have all the control program and perhaps part of the filing system on the drive. Each of these drives (and their controllers) could be running the same software program, and all would have the same hardware interface, but there is a wide variety of cost and performance between them.

Appendix H: DS-link connectors

INMOS has been working with a number of users of transputers and links on proposing standards for 100MBaud DS-link connections between equipments. It was evident very early in this work that the standard was heavily dependent on the connector used; it needed a connector that would be low-cost, small, modular, robust, and ergonomic.

A requirement specification was produced for a connector, to which four manufacturers responded and are developing the connector.

INMOS has received samples of the male cable end, and has been told to expect samples of the female to fit on the PCB during May. A specification is being produced, along the lines of an IEC standard specification, to ensure inter-mateability between the manufacturers. Where there is no conflict, the connectors will follow the existing IEC proposal IEC SC48B (Sec).



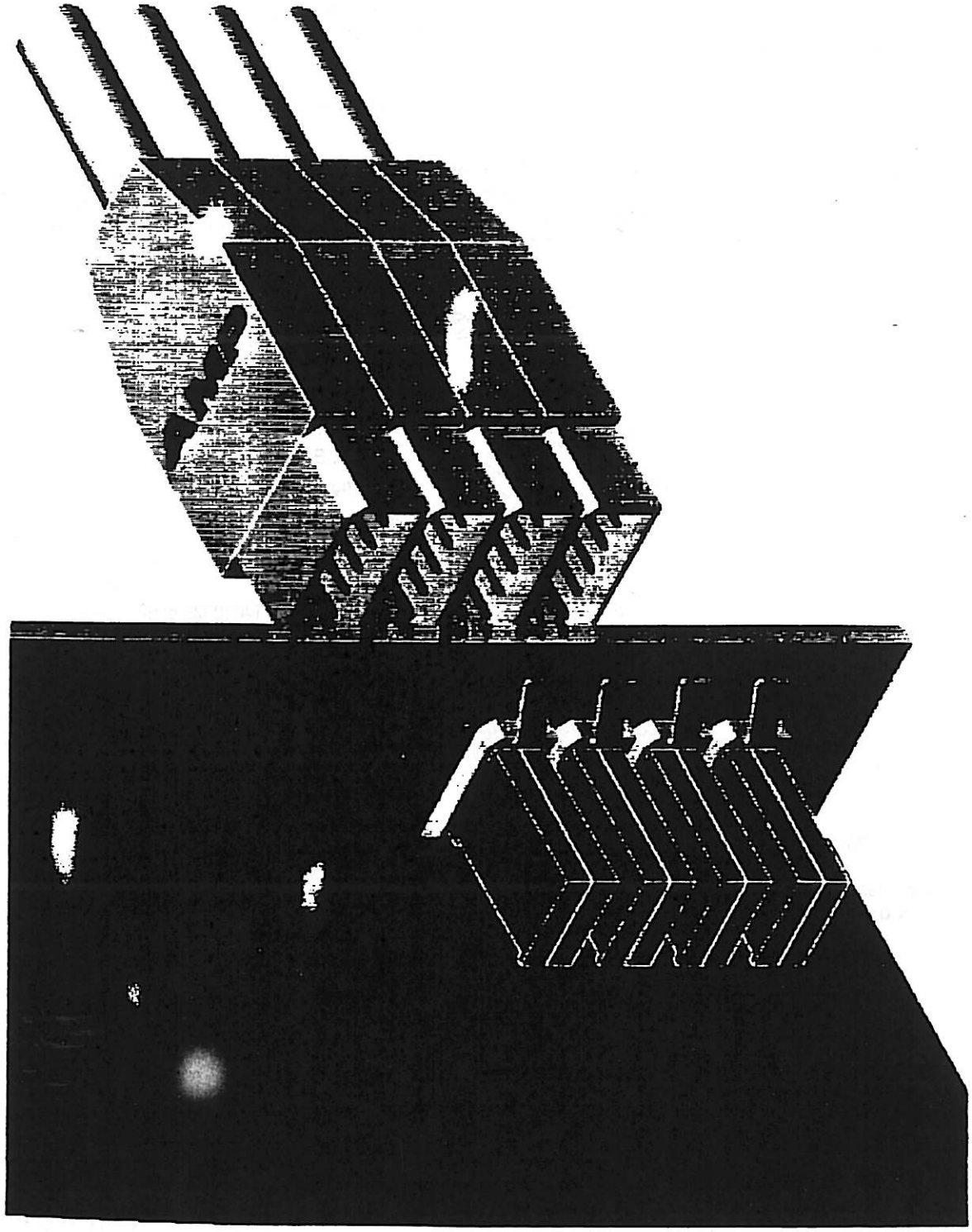
Connectors being developed by AMP, Fujitsu, Harting and McMurdo, with informal cooperation and original requirement spec. from INMOS.

The following three pages show manufacturers' drawings of different aspects of the connectors.

Enquiries about the connector — whether for DS-Links, for OS-Links, or for any other application — should be addressed to the connector companies, whose contacts concerning this connector are:

Company	Country	Contact	Phone	Fax
AMP	All (UK)	Terry Kingham	+44 81 954 2356	+44 81 954 6234
Fujitsu	Europe	Komei Yamaguchi	+49 6103 6900	+49 6103 690122
	Japan	Hajime Hasegawa	(0262) 45 3333	(0262) 48 2840
	UK	Linda Brewer	0628 76100	0628 781484
	USA	Bob Thornton	(408) 922 9000	(408) 428 0640
Harting	France	Michel Maillet	+33 1 49 38 34 00	+33 1 48 63 23 06
	Germany	Ralf Bokämper	+49 5772 47285	+49 5772 47403
	Italy	M Pinio	225 05 248	226 50 543
	UK	David Franklin	0604 766686	0604 706777
	USA	David Robak	(708) 519 7700	(708) 519 9771
McMurdo	All (UK)	Geoff Willingham	+44 705 735361	+44 705 755020

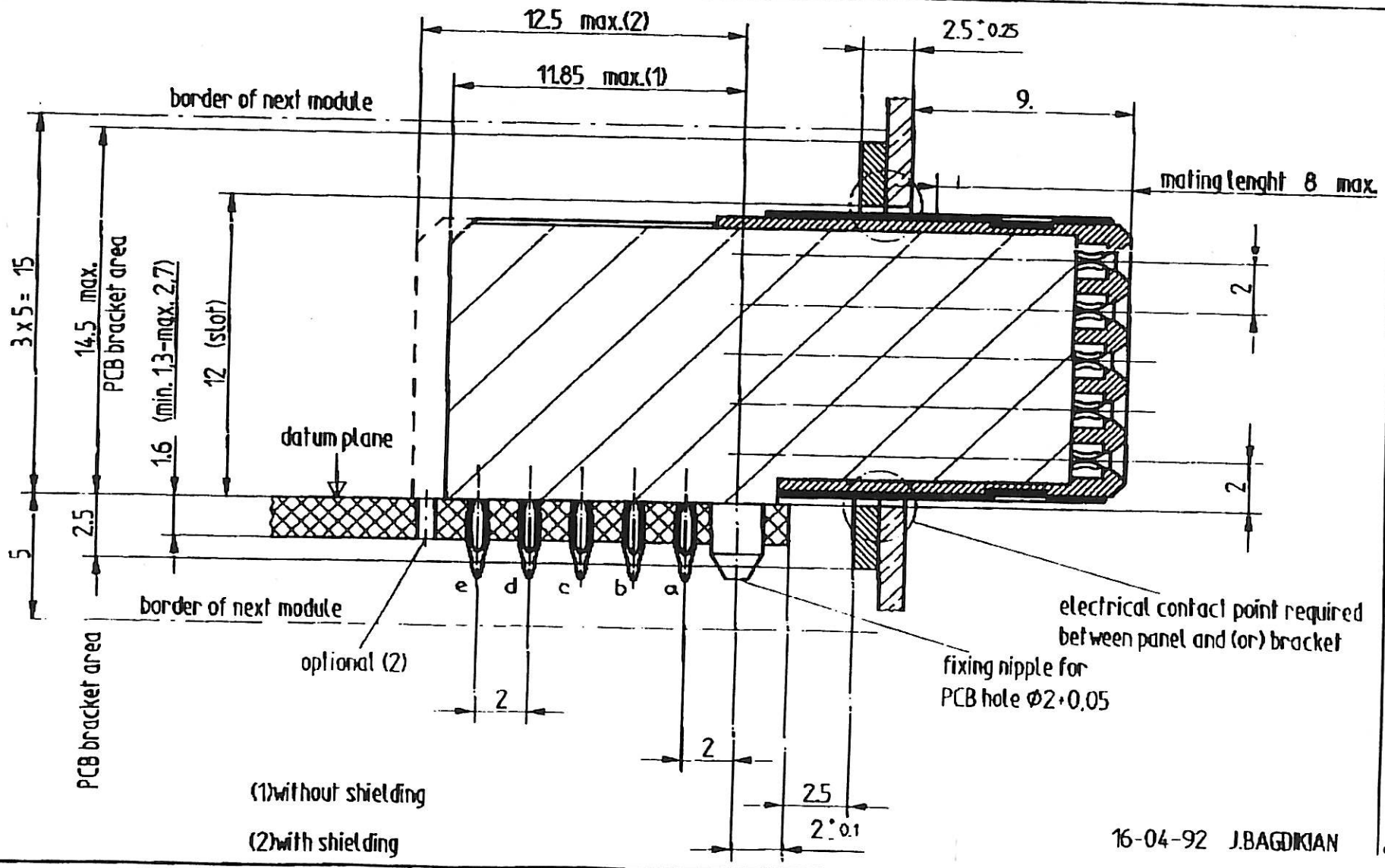
In case of difficulty, or to send INMOS a copy of your request, contact Paul Walker at INMOS Bristol, fax +44 494 617910, email paul@inmos.co.uk

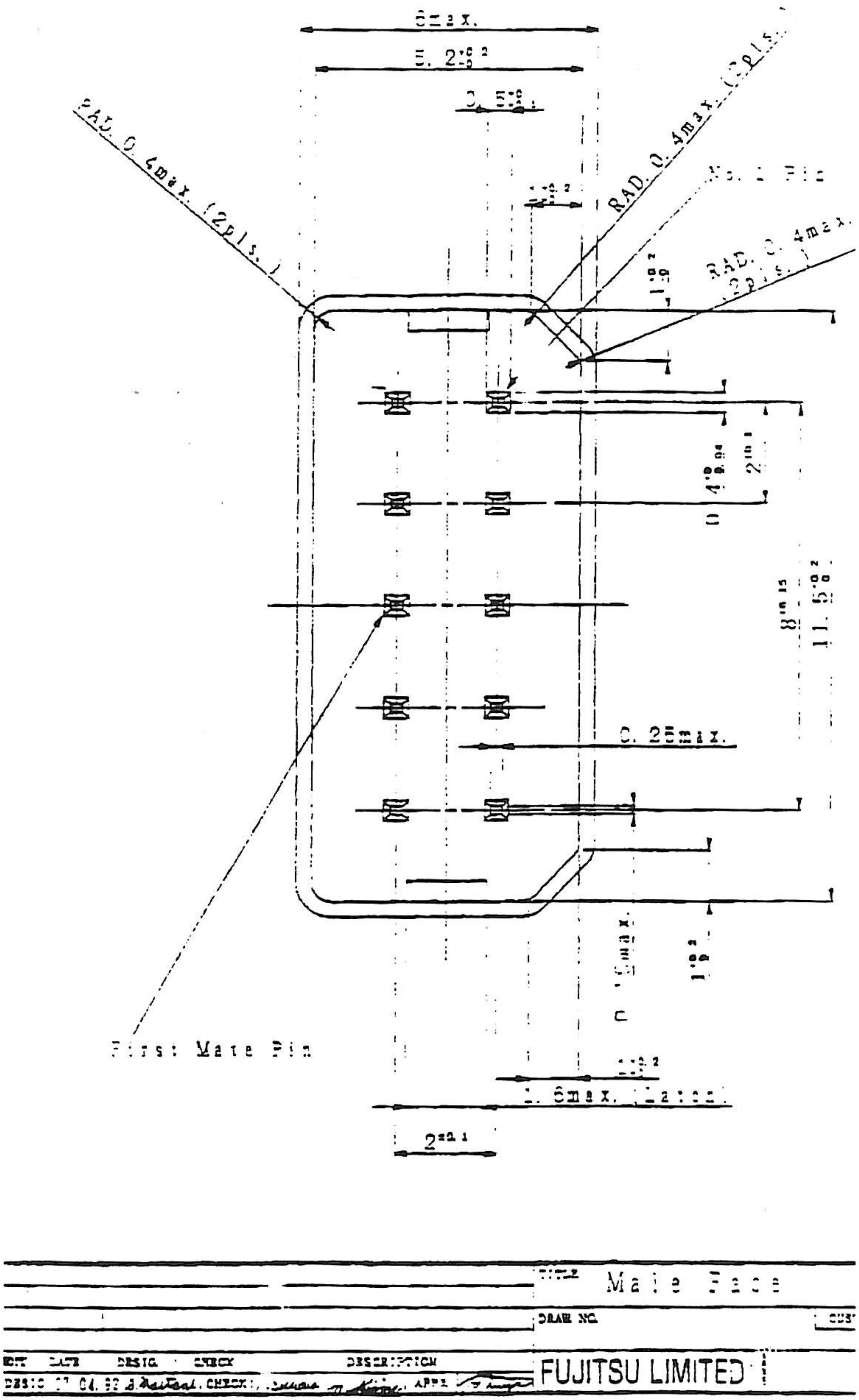


10 WAY MODULAR I/O CONNECTOR SYSTEM

Female Connector

Cross section / Side view




6max.
5.2
0.5
RAD. 0.4max.(2pls.)
RAD. 0.4max.(2pls.)
No. 1 Pin
RAD. 0.4max.(2pls.)
0.25max.
First Mate Pin
1.6max.(Latch)
Male Face
DRAW NO.
FUJITSU LIMITED