SGS-THOMSON MICROELECTRONICS

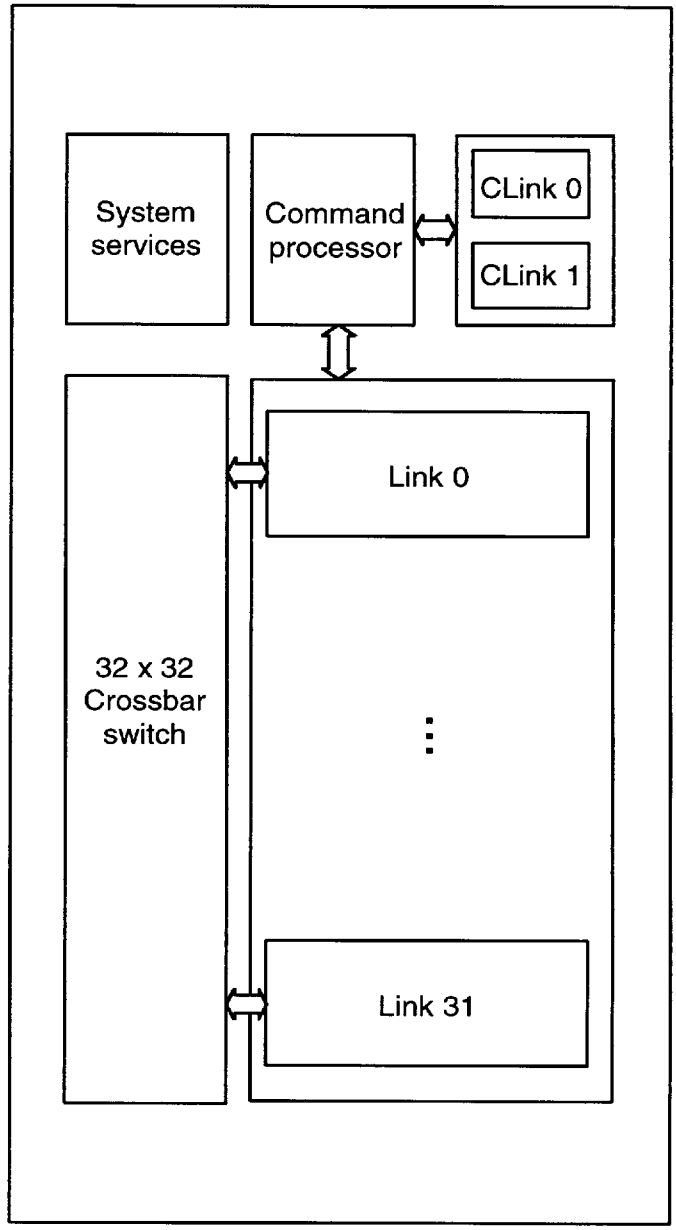
STC104

ASYNCHRONOUS PACKET SWITCH

ENGINEERING DATA

FEATURES

- 32 way Asynchronous Packet Switch (APS)
- 32 x 100 Mbits/s serial bi-directional links
- 300 Mbytes/s bandwidth
- High rate of packet processing
  – up to 200 Mpackets/s
- Less than 1 µs packet latency
- Wormhole interval routing algorithm:
  routes packets of any length
- Grouped adaptive routing
  (support for fault tolerant networks)
- Non-blocking crossbar
- Concurrent processing of packets
- Partitionable architecture
- Cascadable to any depth
- Support for hierarchical routing
- Supports 2-phase routing: hot spot avoidance
- Separate control system
- No loss of signal integrity
- Full flow-control
- Bit and packet level error handling
- Uses 4 wire per link Data-Strobe encoding for simple clock extraction
  – eliminates need for high speed clocks
- On-board phase locked loop requires single 5 MHz crystal input
- On-chip buffering of 70 tokens per link
- Highly configurable: 28 Kbits user-programmed data
- Boundary scan support
- Available in 208 CLCC package



APPLICATIONS

- Core switching element for packet switched communications networks e.g. ATM, Fibre Channel, switched Ethernet and Token Ring.
- Switched interconnect mechanism for heterogeneous processor systems using the STC101 Parallel DS-Link® Adaptor.

June 1996

Errata sheet

This errata sheet lists the behavior of the STC104 which differs from that stated in the following datasheet. Subsequent revisions of the device will behave as detailed in the datasheet.

Control Link 0 reporting of errors

Double error messages are sent upon a single valid data link error being detected, instead of only one error message sent from control link 0. Each error message is valid and identical, and the second error message is sent immediately following the first.

Spurious errors during reset

Reset 1 and 2 via the control link can cause spurious error messages to be generated. These errors should be ignored.

Spurious generation of InvalidHeader errors

Spurious invalid header error messages are output from control link 0 when data is being routed from links in which there are invalid bits set in the interval tables even though the flag may not be set for the interval selected. The data is routed correctly for those links without the invalid flag set, the bug being in the error reporting mechanism. The only way to guarantee that these messages are not generated is not to mark any interval registers as invalid.

DiscardIfInactive bit of the PacketMode0–31 registers

The **DiscardIfInactive** bit of the **PacketMode0-31** registers (see table 5.2, page 34) has a different effect than specified. The **DiscardIfInactive** bit is only effective if the link has been started and has subsequently detected an error. If the link never becomes active (either because this end of the link is not started, or the other end is not) then packets addressed to the link are not discarded regardless of the setting of the bit.

Implementation of grouping

There is a bug in the implementation of grouping in the STC104. The effect of this is that the implementation does not support fault–tolerant operation as was originally intended. The grouping mechanism as implemented may direct a packet addressed to a group to a link of the group which is inactive, with the result that the packet will stall indefinitely. Groups containing links which are not active (either because they are not connected, not turned on or because they have detected an error) are thus not usable.

Control unit – zero length packet

Zero length packets (single terminators) forwarded from clink1 to clink0 will cause outgoing messages from the STC104 to be stalled until a further terminator is forwarded to the uplink. This may happen if clink1 detects a parity error while it is not in the middle of a packet and the localize error bit is set. No further acknowledges, handshakes or error reports will be output on clink0 until a terminator is output on clink1.

Control unit – RecoverError unwanted protocol error

When a *RecoverError* command is sent to a device while a command is in progress the control unit will report a protocol error, in addition to processing the *RecoverError* in the expected way.

Control unit decodes MSB header only

Control link daisy-chains which rely on the MSB to distinguish devices will not work. This limits daisy-chains to 256 devices.

DSlink response to error

After a C104 link has detected a parity error, it must have its **ParityError** status bit cleared (by a *Start* or *Reset*) before any tokens are received on that link.

Any attempt to send tokens to a C104 link which is idle following a parity error will cause the C104 to send a short burst of tokens and then fall silent, invoking a disconnect error in the sender.

DS-Link flow control deadlock

DS-Link data transmissions deadlock under certain conditions. The deadlock occurs when the 20 token DSLink token FIFO fills completely. When the FIFO is completely full it reports that it is empty, thus no more tokens leave the FIFO and the result is deadlock. It should be noted that this is a very rare occurrence.

Contents

1 STC104 introduction 5

2 Communication on an STC104 system 7

3 Operation of STC104 networks 8

3.1 Wormhole routing 9
3.1.1 Buffering 10
3.2 Interval labelling 10
3.3 Multiple networks 13
3.4 Security in networks 14
3.4.1 Use of the Invalid flag 14
3.4.2 Partitioning of STC104s for use in parallel networks 14
3.5 Modular composition of networks using header deletion 16
3.6 Hot spot avoidance – universal routing 17
3.7 Grouping of output links 19

4 STC104 functional description 22

5 Control of the STC104 25

5.1 Control links 25
5.1.1 Commands 26
5.1.2 Control links used to provide fan-out in a control network 31
5.1.3 Control link speeds 32
5.2 Programmable configuration register functionality 33
5.2.1 Packet processor registers 33
5.2.2 Data DS-Link registers 36
5.2.3 Control link registers 38
5.2.4 System services registers 38
5.3 Initialization of the STC104 40

6 Data/Strobe links 41

6.1 Low-level flow control 42
6.2 Link speeds 42
6.3 Errors on links 43
6.3.1 Reliable links 44
6.3.2 More reliable links 44
6.4 Link state on start up 44
6.5 Resetting DS-Links 45
6.6 Link connections 45

7 Levels of reset 47

7.1 Level 0 – hardware reset 47
7.2 Level 1 – labelled control network 47

7.3 Level 2 – configured network 47
7.4 Level 3 47
7.5 Per link reset 47
7.6 Effects of different levels of Reset 48

8 Configuration register addresses 49

8.1 Subsystem addresses 49
8.2 Register addresses 51
8.2.1 Packet processor configuration registers 51
8.2.2 Link configuration registers 51
8.2.3 Control link configuration registers 51
8.2.4 System services configuration registers 51

9 Clocks 52

9.1 Clock input 52
9.2 Phase locked loop decoupling 52
9.3 Speed selection 53

10 Electrical specifications 54

10.1 Absolute maximum ratings 54
10.2 Operating conditions 55
10.3 DC characteristics 55
10.4 Power rating 55

11 Timing specifications 56

11.1 ClockIn timings 56
11.2 DS-Link timings 57
11.2.1 Link Input and Output relative skews 58
11.2.2 Skew budget 59

12 Pin designations 60

13 Package specifications 62

13.1 STC104 208 pin CLCC cavity-down package pinout 62
13.2 STC104 208 pin CLCC cavity-down package dimensions 63
13.3 STC104 208 pin CLCC cavity-down package thermal data 64

14 Ordering information 65

1 STC104 introduction

This document contains preliminary information for the STC104 Asynchronous Packet Switch (APS). The STC104 is part of the product family based around high speed asynchronous serial communication between various parts of a system.

The STC104 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 packets to be routed from any of its links to any other link. The links operate concurrently and the transfer of a packet between one pair of links does not affect the data rate or latency for another packet passing between a second pair of links. Each link can operate at up to 100 Mbits/s, providing a bidirectional bandwidth of 19 Mbytes/s. The STC104 supports a rate of packet processing of up to 200 Mpackets/s.

The STC104 allows communication between devices, such as microprocessors, that are not directly connected. A single STC104 can be used to connect up to 32 microprocessors. The STC104 can also be connected to other STC104s to make larger and more complex switching networks, linking any number of microprocessors, link adaptors, and any other devices that use the link protocol. Another member of the product family, the STC101 Parallel DS-Link Adaptor, will allow links to be interfaced to peripheral buses and devices, refer to the *STC101 datasheet (document number 42 1593 xx)* for details.

The STC104 enables networks to be built which effectively emulate a direct connection between each of the devices in the system, and removes the need for through-routing software. In the absence of any contention for a link output, the packet latency[1] will be less than 1μ second through each STC104.

Data in an STC104 communication system is transmitted in packets. To enable packets to be routed, each packet has a header at the front which contains routing information. The STC104 uses the header of each incoming packet 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 STC104 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 STC104 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, thereby pipelining the transmission of the packet. 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.

1. Latency here means the time between the first bit of the packet being received on one link and being re–transmitted on another.

The algorithm which makes the routing decision is called *interval labelling*, which is complete, deadlock free, inexpensive and fast. Each destination in a network is labelled with a number, and this number is used as the destination address in a packet header. Effectively, each of the 32 links on a routing switch is labelled with an interval of possible header values, and only packets whose header value falls within that interval are output via that link. Thus the header specifies a particular link along which to transmit the packet. Consecutive links may be programmed to be 'grouped', so if a packet is routed to an output link which is busy it will automatically be routed along any other link in the group which is available. In this way performance can be optimized by allowing packets to be routed to any one of several outputs, depending on which link in the group is the first to become available. Grouping also provides fault tolerance.

The STC104 can be programmed so that the output link selected by the router is independent of the input link on which the packet arrives. Alternatively the STC104 can be programmed so that some link inputs are mapped to a specific set of link outputs. This can be used to enable independent networks to be implemented with the same ST C104 with complete security.

The STC104 can be programmed so that certain header ranges are marked as invalid in which case packets whose headers fall within this range are discarded. This can be used to enforce security in multi-user networks.

To eliminate network hot spots, the STC104 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.

Usually packets are routed through the STC104 unchanged. However a flag can be set in the specified output link, in which case the header of the packet is discarded. Each link output of the STC104 can be programmed to delete the header of a packet, revealing a second header to route the remainder of the packet to the next destination device. This assists in the modular and hierarchical composition of routing networks and simplifies the labelling of networks.

The STC104 is controlled and programmed via a control link. The STC104 has two separate control links, one for receiving commands and one to provide daisy chaining. The control links enable networks of STC104s 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.

2 Communication on an STC104 system

The STC104 can be connected to a range of microprocessors or devices via an STC101 Parallel DS-Link Adaptor. In this datasheet the combination of an STC101 and connected device (or a device with one or more integrated DS-Links, for example an IMS T9000 transputer) is referred to as a processing node, as shown in figure 2.1.

System wide communication can be provided by connecting different microprocessors to a single routing network via one or more STC101 Parallel DS-Link Adaptor devices. By using several STC101s, a microprocessor can be connected to several different networks.

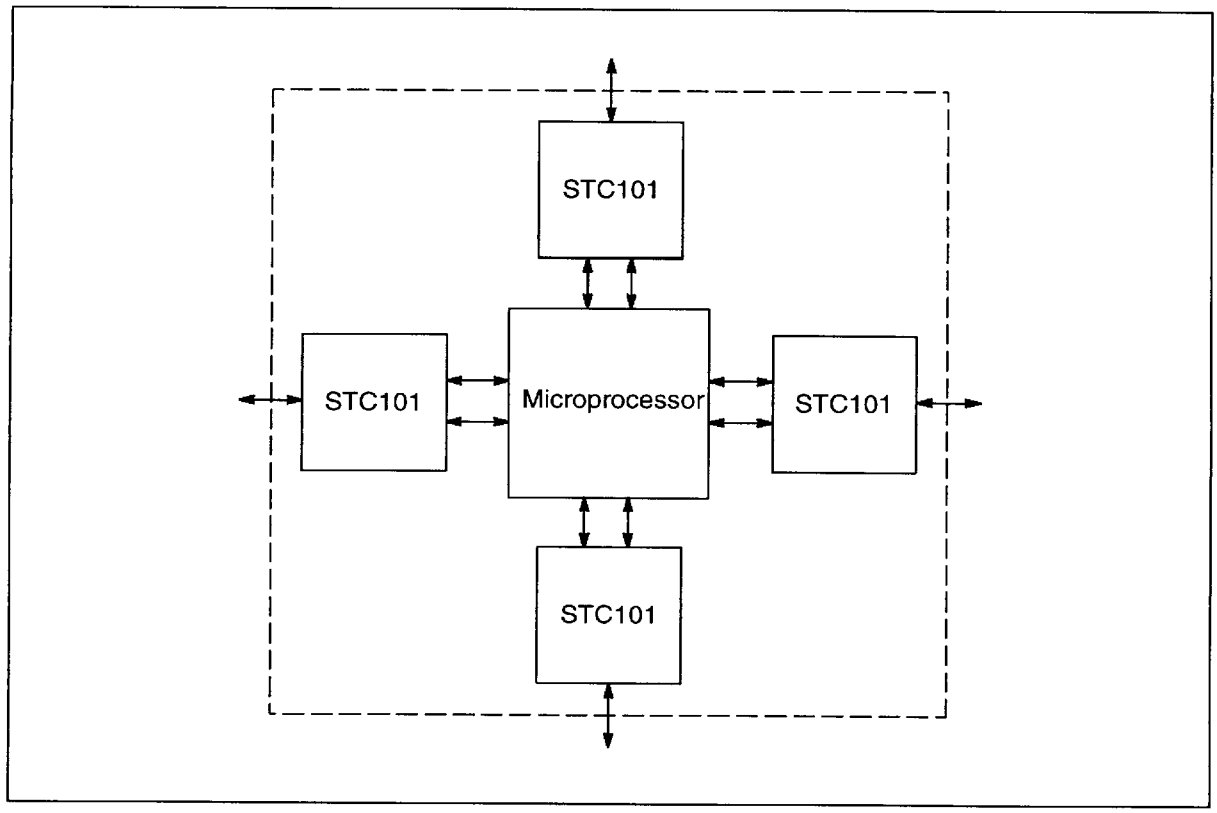


Figure 2.1 Processing node

Communications between different processes on a microprocessor usually take place over software channels. Communication between processes on different processors often take place over a message passing system based around an arbitrated bus. This can lead to high latency within a system and a large software and hardware overhead for the system as a whole and lower data bandwidth. By implementing software virtual channels and using the *dynamic message routing* capability of the STC104, overall system bandwidth can be improved significantly while reducing the need for hardware arbitration within the system. The STC104 and STC101 use a protocol which supports both virtual channels and dynamic message routing, and provide a high data bandwidth and very low latency.

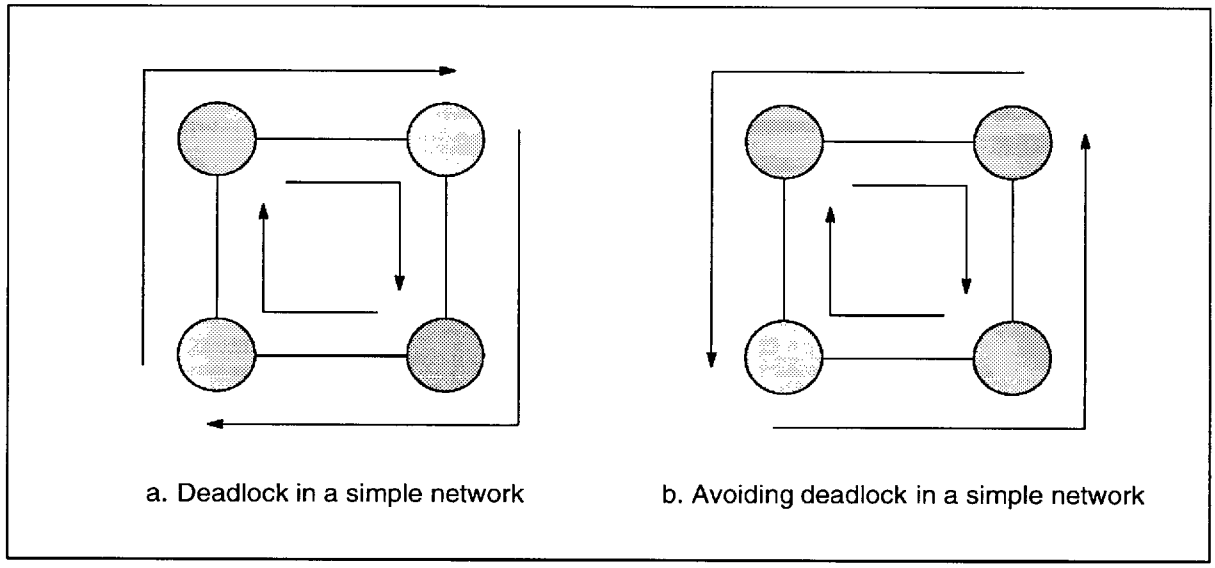
3 Operation of STC104 networks

A single STC104 can be used to connect up to 32 subsystems that are not directly connected to each other. The STC104 can also be connected to other STC104s to make larger and more complex switching networks, linking any number of microprocessors, link adaptors, and any other devices that can utilize the link protocol.

An STC104 network consists of one or more STC104 routing devices connected together by bi-directional 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 nodes or to another network. These links are called terminal links.

The purpose of a communication network is to support efficient and reliable communication. Consequently, an essential property of a communications network is that it should not deadlock[2]. Deadlock can occur in most networks unless the routing algorithm is designed to prevent it. For example, consider the square of four nodes shown in figure 3.1a. If every node attempts to send a packet to the opposite corner at the same time, and the routing algorithm routes packets in a clockwise direction, then each link becomes busy sending a packet to the adjacent corner and the network becomes deadlocked.

Deadlock is a property of the network topology and the routing algorithm used and can be avoided by choosing networks for which deadlock-free wormhole routing algorithms exist. Instead of routing packets in a clockwise direction, the deadlock-free algorithm routes two of the packets anti-clockwise. Since the links are bi-directional this allows all of the packets to be routed without deadlock, as illustrated in figure 3.1b.



a. Deadlock in a simple network

b. Avoiding deadlock in a simple network

Figure 3.1 Deadlock in networks

In order to support the efficient routing of packets through a network the STC104 implements a complete deadlock-free routing algorithm in hardware. The component parts of the algorithm are described in the following sections.

2. Deadlock is a state where further progress is impossible due to a cycle of resource dependencies.

3.1 Wormhole routing

The STC104 interprets the signals on its inputs as sequences of packets. It takes the first one or two bytes of data (the choice being a configurable parameter) as the header of the packet, which determines what it will do with the whole packet. The length and contents of the remainder of the packet are arbitrary. The end of the packet is indicated by one of two distinguished termination tokens, called EOP (end of packet) and EOM (end of message).

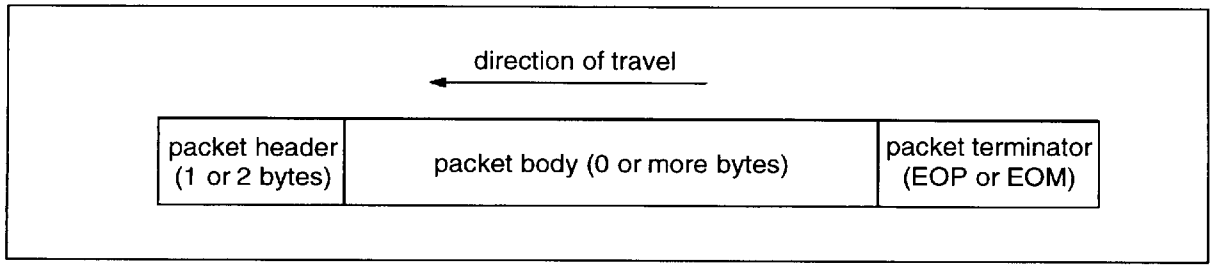


Figure 3.2 Packet structure

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 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 STC104 uses *wormhole routing* (figure 3.3) 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 STC104s, 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.

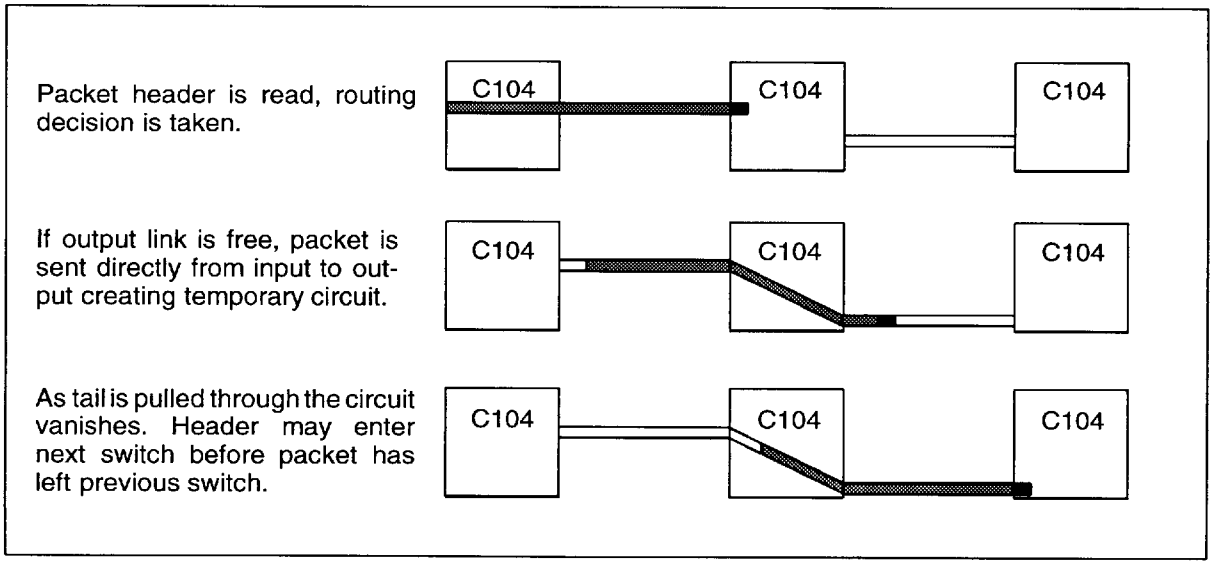


Figure 3.3 Wormhole routing

The implications of wormhole routing are that a packet can be passing through several STC104s 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.

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.

Note that if a packet is transmitted from a link running at a higher speed than the link on which it is received, there will be a loss of efficiency because the higher speed link will have to wait for data from the slower link. In most cases all the links in a network should be run at the same speed.

3.1.1 Buffering

To exploit the full bandwidth of the internal pathways on the STC104 there is buffering on each path through the device. The buffering is fully handshaken FIFO buffering with minimal latency.

3.2 Interval labelling

Wormhole routing requires an efficient routing strategy to decide which link a packet should be output from. The STC104 uses a routing scheme called *interval labelling*, whereby each output link of an STC104 is assigned a range, or interval, of labels. This interval contains the number of all the terminal nodes (i.e. microprocessors, gateway to another network, peripheral chip, etc) which are accessible via that link. As a packet arrives at an STC104 the selection of the outgoing link is made by comparing the header label with the set of intervals. This is illustrated in figure 3.4. The intervals are contiguous and non-overlapping and assigned so that each header label can only belong to one of the intervals. The output link associated with the interval in which the header label 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.

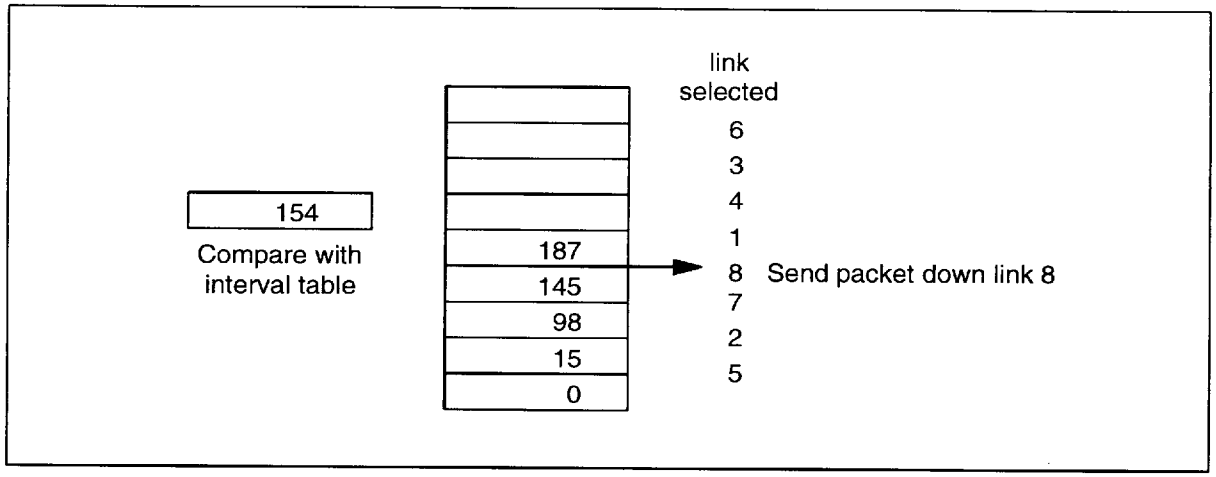


Figure 3.4 Interval labelling

Figure 3.5 gives an example of interval routing for a network of two STC104's and six processing nodes. The example shows six links, one to each processing node, labelled

0 to 5. The interval contains the labels of all nodes accessible via that link. The interval notation [3,6) is read as meaning that the header label must be greater than or equal to 3 and less than 6. If the progress of a packet with the header label 4 is followed from $node_1$ then it is evident that it passes through both STC104s before leaving on the link to $node_4$.

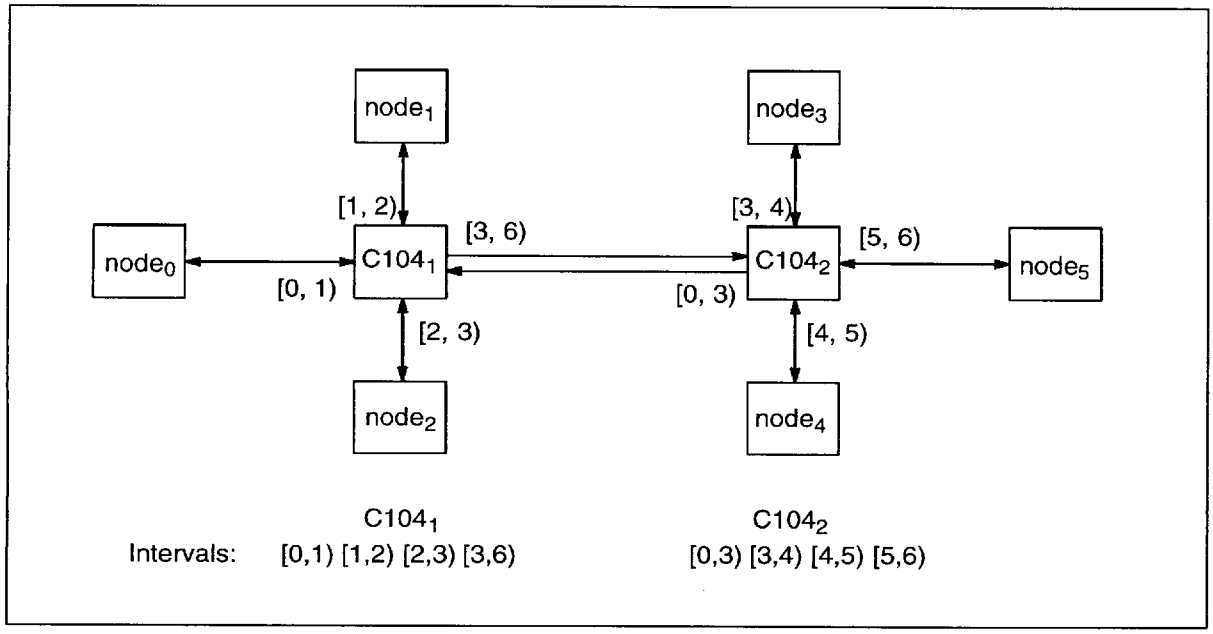


Figure 3.5 Interval routing

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 labellings, which will be provided to customers, are available for grids, hypercubes, trees and various multi-stage networks. A few topologies, such as rings, cannot be labelled in an optimal deadlock free manner. Although they can be labelled so that they are deadlock free, this is at the expense of not using one or more of the links, so that the labelling is not optimal. Optimal deadlock free labellings 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. 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.

Interval routing is implemented on the STC104 by interval selector units. There is one interval selector unit per input link, which performs the routing decision for each packet arriving on the link. An interval selector unit effectively consists of 36 base and limit comparators (see figure 3.6).

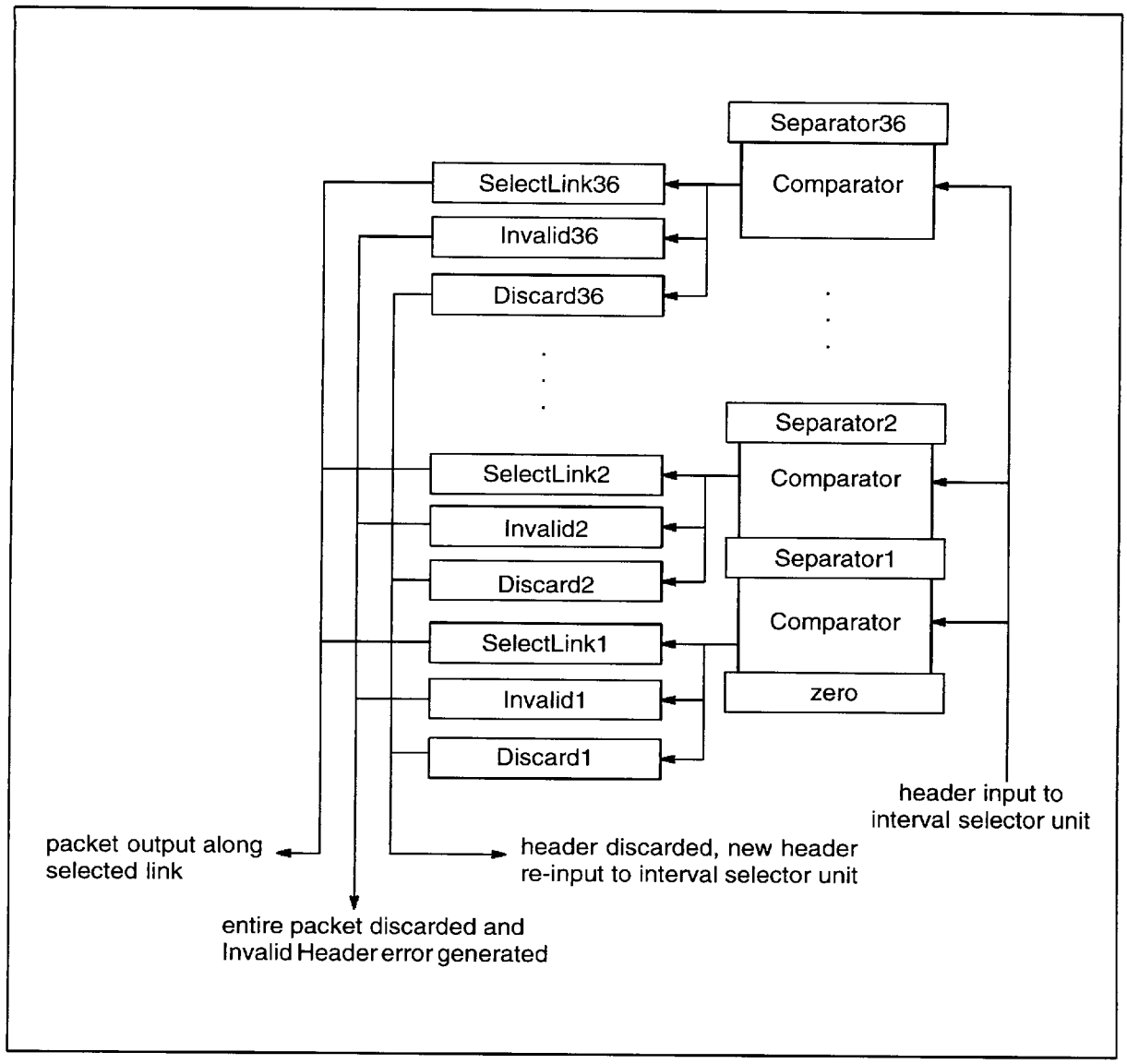


Figure 3.6 Interval selector registers

Each comparator is connected to a pair of programmable interval separators, except the lowest whose base is fixed at zero. Each interval separator (**Separator1-36**) is connected to the limit of one comparator and the base of the next comparator, except the top one (**Separator36**) which is connected to the limit of the top comparator only. The **Separator1-36** register bit fields 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 bit field (**SelectLink**). The **SelectLink** bit field contains the number of the associated output link. The link is selected for output if the packet header is greater than or equal to the base and less than the limit value of the adjoining comparator. Once the path through the crossbar is set the tokens are passed through until an EOP or EOM terminator token is detected.

Each **Interval** register has two flags: **Discard** and **Invalid**. The **Invalid** bit designates whether packets whose headers fall into the interval below the separator should be discarded (with the generation of an 'Invalid Header' error). This is used to ensure security in multi-user networks, see section 3.4.1. If the **Invalid** bit is set, the **Discard** bit should not be set. If the **Invalid** bit is not set, the **Discard** bit designates whether headers falling into the interval below the separator should be discarded. This is used in the implementation of Universal Routing; see section 3.6. Note that if the **Discard** bit or the **Invalid** bit is set, the value of the corresponding **SelectLink** field is arbitrary, since it will never be used.

Note that if two successive separator values are the same, this forms a *null interval.* If one or more null intervals occur below a non-null interval (note that the topmost interval **Interval36** can never be null), they must all have the same values in their **SelectLink**, **Discard** and **Invalid** fields.

If the **HeaderLength** flag is 0 (i.e. the STC104 is set to input 1 byte headers), all **Separator** fields must contain values in the range 0 to 255 inclusive. If it is required to use the header value 255 (the maximum possible with one byte) then the **Interval36** register must be used, rather than programming any of the **Separator** fields of the **Interval1** to **Interval35** registers with a value in excess of 255.

Note: more than one **Discard** or **Invalid** bit may legitimately be set, and two or more **SelectLink** fields may have the same value.

Further details on the Interval registers are given in section 5.2.1.

3.3 Multiple networks

System wide communication can be provided by connecting different microprocessors to a single routing network via one or more STC101 Link Adaptor devices. By using several STC101s a microprocessor can be connected to several different networks, or simply logical sub-networks of one network of STC104s. The use of multiple networks can provide the following:

- 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.
- Separate networks for data and control messages.

3.4 Security in networks

The STC104 can provide a mapping between the value of the incoming packet header and the output link on which it will be forwarded, which is independent of the link on which the packet is received. This can be achieved by programming the **Interval** registers identically for each link. The STC104 is then logically a single entity. However, the **Interval** registers can be programmed and devices labelled to ensure that every packet arriving on a particular link takes a set route through the network.

3.4.1 Use of the Invalid flag

The STC104 can be programmed so that certain header ranges are marked as 'invalid' in which case packets whose headers fall within this range are discarded. This can be used to forbid routing of packets with headers in certain ranges. Associated with each interval is an **Invalid** flag. If a header of an incoming packet falls within an interval which has its **Invalid** flag set, the packet is discarded and an 'Invalid Header' error is generated.

Note that, if the number of destination labels in the system is less than the range of headers being used (256 or 64k) then at least one interval should have its **Invalid** flag set to trap illegal headers, whether the system is multi-user or not.

3.4.2 Partitioning of STC104s for use in parallel networks

In some circumstances, where the STC104 is to be connected to two or more different networks, it is advantageous for the STC104 to be treated as two or more independent devices. For example, a single STC104 could be used for access to both a data network and a control network (see figure 3.7). This can be implemented by *partitioning* the STC104. The links of the STC104 can be divided into disjoint sets, called partitions, with the **Interval** registers of every link in each partition programmed identically.

Complete security is achieved provided that, in each partition, no **SelectLink** field of any **Interval** register contains the number of a link in another partition, and no link group crosses any partition boundary. Within each partition, all the **HeaderLength** flags (which set the header length to 1 or 2 bytes for each link, see section 3.5) must be the same, and the **RandomBase** and **RandomRange** registers should be the same for all links in the partition which are set to random header generation mode (see section 3.6).

Partitioning provides economy in small systems, where using an STC104 solely for the control network is not desired.

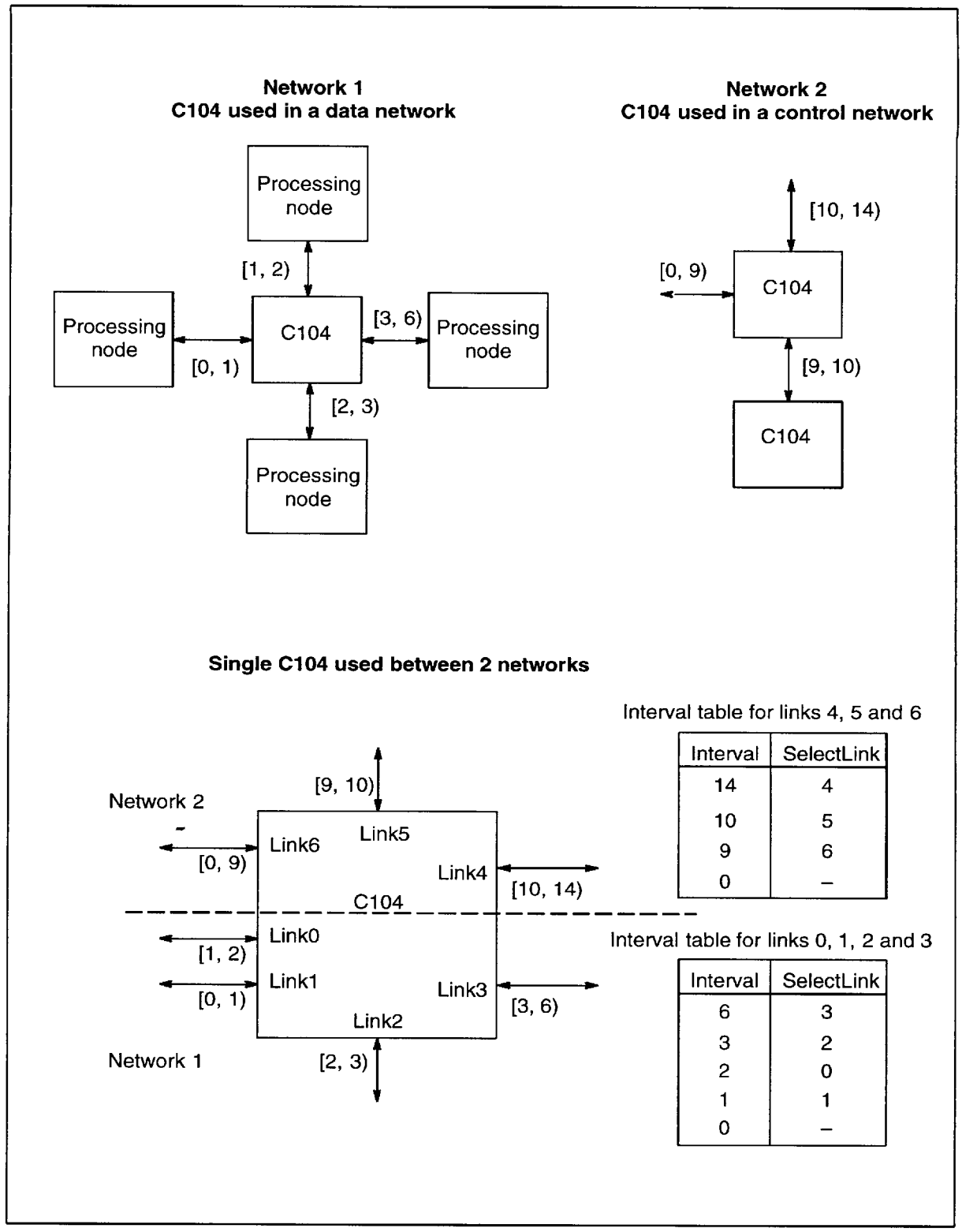


Figure 3.7 Using partitioning to enable one STC104 to be used by two different networks

3.5 Modular composition of networks using header deletion

To assist in the modular composition of routing networks the STC104 contains a hardware mechanism to implement *header deletion.* Each link output of the STC104 can be programmed to delete the header of a packet before transmitting the remainder of the packet. This exposes a further header which is used by the destination device.

Associated with each link output is a **HeaderDeletion** flag (contained in the **PacketMode** register, see section 5.2.1). When the **HeaderDeletion** flag is set to 1, the header of the packet which is being output through the link is discarded and the remainder of the packet forwarded to its destination. The number of bytes which are deleted depends on the setting of the **HeaderLength** flag: if this is 0, one byte is deleted; if it is 1, two bytes are deleted.

If there are no data bytes following the deletion of the header (i.e. only a termination token), the termination token is also discarded and a 'null packet' error is signalled. Note that this applies even if the link is inactive and the associated **DiscardIfInactive** flag is set, in which case the packet is discarded anyway. The **DiscardIfInactive** flag is used when an error has occurred on the link, see section on Errors page 30 for further details.

Header deletion allows networks to be connected together, as shown in figure 3.8. In this example a packet is routed through two networks and then to a processing node. All of the terminal links of the two networks are set to header deletion mode. Figure 3.8 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 processing node. This can be applied to hierarchically constructed networks, in which case the sub-headers are similar to the local/national/international hierarchy of telephone numbers.

In the case in which each STC104 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 STC104 traversed.

A major advantage of extending the capabilities of the STC104, 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.

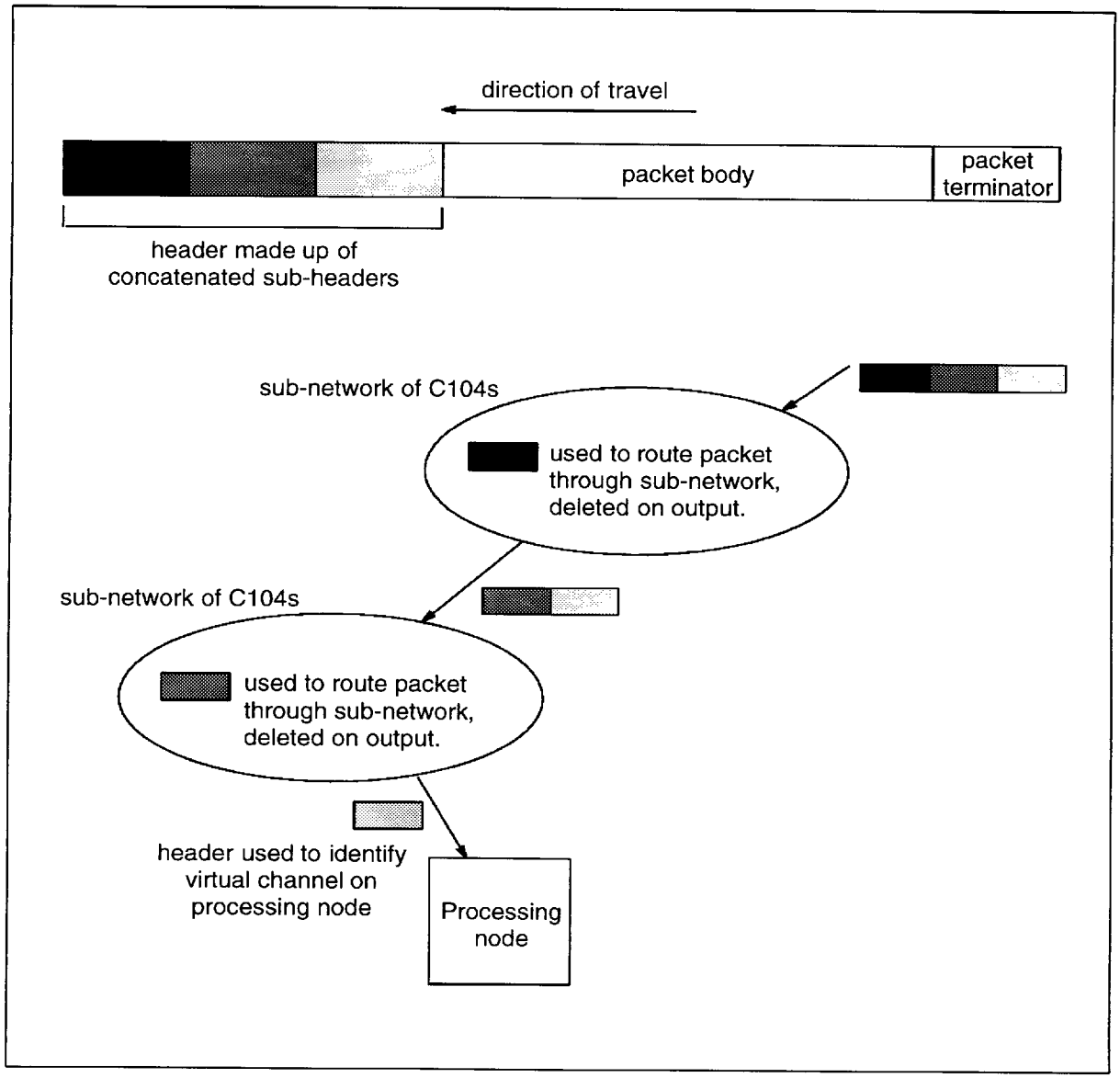


Figure 3.8 Hierarchical composition of networks using header deletion

3.6 Hot spot avoidance – universal routing

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 STC104 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.

Each input link of an STC104 can be set to random header generation mode by setting the **Randomize** flag in the **PacketMode0-31** registers. If this flag is 1 each arriving packet is routed depending on a pseudo-randomly generated header of length one or two bytes (depending on the **HeaderLength** flag of the link). The header is generated within a range determined for each link by two 16-bit unsigned programmable registers, **RandomBase** and **RandomRange**. Headers are generated in the range **RandomBase** to (**RandomBase** + **RandomRange** −1) inclusive. The seed of the pseudo-random sequence for each link is loaded into the register **RandomSeed**. Note that these registers must be loaded with known values in order to ensure repeatable behavior. Also, no two **RandomSeed** registers should be loaded with the same value, nor should any be loaded with a zero value.

Note that no random headers will be generated until a configuration write is performed to the **ConfigComplete** register (see section 5.2.4). Note also that all links for which the **Randomize** flag is set must have their **RandomBase**, **RandomRange** and **RandomSeed** correctly set before a write is made to the **ConfigComplete** register, otherwise random headers may be generated from the wrong range.

Note that it is usual for all links of the STC104 which have their **Randomize** flag set to have the same values in their **RandomBase** and **RandomRange** registers. Different links may have different values in these registers if the STC104 is partitioned into two or more logical devices; see section 3.4.2.

Associated with each interval is a **Discard** flag in the **Interval1-36** registers. The **Discard** flag is set to indicate that the randomly generated header has reached its intermediate 'random' destination. The interval with its **Discard** flag set is called the 'portal' interval. If the input header is indicated as belonging to a portal interval the header is discarded, revealing the final destination header.

It is the combination of the random header generation mechanism and the **Discard** flags which enables the Universal Routing algorithm to be implemented in a single network of STC104s. The **Randomize** flag is set for each link entering the network. The random header effectively designates one of the STC104s of the network, to which the packet is routed. At the intermediate STC104 the randomly generated header will correspond to the portal interval in which the **Discard** flag is set, and therefore it will be discarded. This reveals its original destination header which is used to route the packet out of the network.

If none of the **Discard** flags are set, the portal mechanism is disabled.

Note that it is possible that the randomly generated header will fall into a 'portal' interval immediately, in which case it is discarded at once and not transmitted. This corresponds to the randomly chosen intermediate STC104 happening to be the one through which the packet enters the network.

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 3.5 above. Header deletion mode deletes headers as the packet is forwarded along an output link, whereas header deletion associated with universal routing occurs when the random header of a packet entering the STC104 on an input link is determined 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 an STC104 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.

3.7 Grouping of output links

The STC104 implements *grouped adaptive routing*, whereby consecutive output links can be grouped so that packets routed to the first link of the group will be sent down any free link in the group, depending on which is the first link to become available. This achieves improved network performance in terms of both latency and throughput.

Figure 3.9 gives an example of grouped adaptive routing. Consider a message routed from $C104_1$, via $C104_2$, to processing $node_1$. On entering $C104_2$ the header specifies that the message is to be output down **Link5** to processing $node_1$. If **Link5** is already in use, the message will automatically be routed down **Link6**, **Link7** or **Link8**, dependent on which link is available first.

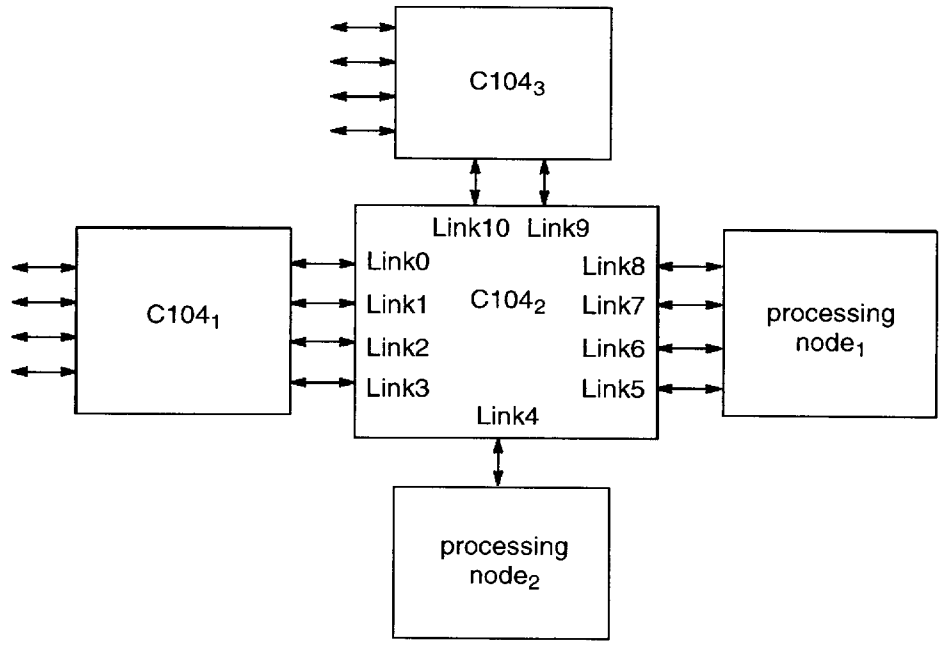
The links can be configured in groups by setting the **ContinueGroup** bit in each of the **PacketMode0-31** registers (see section 5.2.1). Each **ContinueGroup** bit corresponds to a link and can be set to 0 (Start) to begin a group or 1 (Continue) to be included in a group, as shown in figure 3.9. The group wraps around from 31 to 0, therefore the **ContinueGroup** bit can be set to 1 for **Link0**. Note that setting the **ContinueGroup** flag of every link to 0 effectively disables the grouping feature, and that for meaningful routing to take place at least two links must have their **ContinueGroup** flag set to 0, otherwise all links are in the same group.

Note that the information in the **ContinueGroup** bits is not used to form output groups until a configuration write to the **ConfigComplete** register occurs (see section 5.2.4).

When an incoming packet is directed out of a link which is the start of a group it is in fact being directed to any link in the group. Any output of that group whose link is active may

respond to the request and transmit the packet. If two or more outputs are available or become available at the same time, only one of them will respond to the request. Note that it is illegal to select a link for output which is not the start of a group. Each link in a group will output one packet before any link in the group outputs a second packet.

If there are more input packets directed to an output group than there are output links in the group, the 'excess' inputs requesting an access are stalled. As soon as one of the output links in the group becomes free, one of the stalled inputs is granted access to that output. The arbitration is fair, such that if several inputs are stalled waiting for the same output group, each waiting input will transmit one packet before any of the inputs transmits a second packet.



Settings of the **ContinueGroup** bit in the **PacketMode0-31** registers for C104$_2$

0	Start (0)	Grouped
1	Continue (1)	
2	Continue (1)	
3	Continue (1)	
4	Start (0)	
5	Start (0)	Grouped
6	Continue (1)	
7	Continue (1)	
8	Continue (1)	
9	Start (0)	Grouped
10	Continue (1)	
11	Start (0)	
.	.	
.	.	
.	.	
31	Start (0)	

Figure 3.9 Grouped adaptive routing

4 STC104 functional description

The STC104 consists of the main functional blocks shown in figure 4.1. The STC104 has thirty-two data DS-Links and two control links. Each of the thirty-two data DS-Links have their own packet processing hardware.

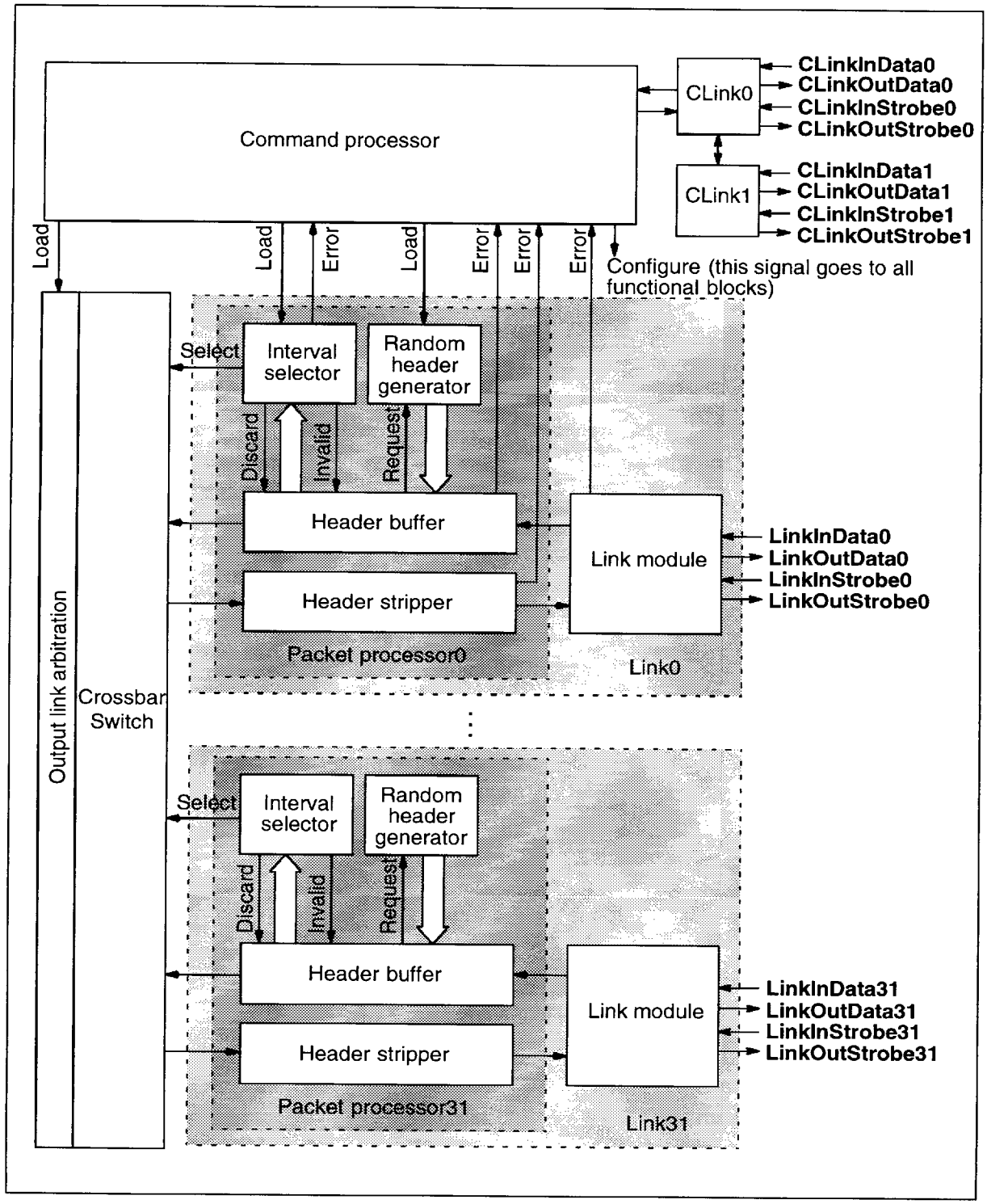


Figure 4.1 STC104 functional block diagram

Each data link is connected to a packet processor. The token stream received on the link is passed to the packet processor and interpreted as a sequence of packets. Each packet processor consists of the following blocks: interval selector; random header generator; header buffer and header stripper.

The stream of tokens received on each link is interpreted as a stream of independent packets. Each packet is either output through one of the thirty-two links or discarded. The header determines which link the packet is to be transmitted from. If the specified link is not busy the packet is transmitted immediately without being buffered. If the link is busy as much data as possible will be buffered before data flow is stalled until the output link becomes available.

The interval selector contains the interval registers and comparators. The header of each packet arriving on the link is forwarded to the interval selector. Dependent on the setting of the interval selector registers and the label of the header, the packet is processed in one of the following ways:

- The header is compared to the intervals and the output link from which the packet is to be forwarded is selected (from the **SelectLink** bit field of the **Interval** register). The entire packet is routed unchanged out of the selected link.
- The link to which the packet is routed has its associated **HeaderDeletion** flag set. The header of the packet is discarded and the remainder of the packet forwarded to its destination.
- The header is compared to the intervals, an error has occurred on the output link selected (or the selected link has not been started) and the associated **DiscardIfInactive** flag is set in the **PacketMode** register. The entire packet is discarded.
- The interval into which the header falls has its associated **Invalid** flag set. The entire packet is then discarded and an 'Invalid Header' error is generated.
- The link input on which the packet arrives has its **Randomize** flag set. A 'Request' signal is sent to the random header generator, which produces a random header which is added to the front of the existing header. The random header is then forwarded to the interval selector and an output link is selected to route the packet to a random node.
- The interval into which the header falls has its associated **Discard** flag set. This indicates that the header falls within the 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 labelling algorithm.

The random number generator generates a pseudo-random sequence of headers from a programmed range for implementing the Universal Routing algorithm.

The header stripper can delete the first header of each packet routed out through the link. This is dependent on whether the **HeaderDeletion** bit of the **PacketMode0-31** register is set.

The link modules accept requests for data from the header buffer and make requests for data from the header stripper. The streams of data into the header buffer and out of the header stripper are handshaken so there is no chance of buffer overflow or overwriting of data.

The crossbar is an array of switches which connect datapaths in one direction and the corresponding control signals in the other direction. It contains arbitration circuitry which permits one of each configured group of outputs to grant a request made by an input to the start of the group.

The control unit contains two control links (**CLink0** and **CLink1**). Commands can be received on **CLink0** and responses and error messages returned. These commands can be used to reset the device and to read and write the configuration registers in the subsystems. **CLink1** is provided to allow a series of devices to be daisy-chained and thereby all controlled over one link.

5 Control of the STC104

The STC104 is controlled and programmed via the control links. Messages sent to the STC104 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 selector, the random number generator and the links.

This chapter describes the control links and the control commands which can be sent and received. It then describes the functionality to be controlled by the configuration registers.

5.1 Control links

The control links on the STC104 allow a separate control network to be used to assist in configuring, error handling and resetting of components connected in a system, even in the presence of errors on the data communications links in the network.

The STC104 has two bidirectional control links; **CLink0** and **CLink1**. They use the same electrical and packet level protocols as the communication DS-Links, (**Link0-31**, see chapter 6). Thus, an STC104 can be connected by its control link to a data DS-Link of a controlling processing node and the node can issue commands to the STC104.

All communications with the controlling processor are via **CLink0.** The STC104 is programmed via commands along **Clink0**. **CLink1** provides a daisy-chain link, allowing a simple physical connectivity to be used for controlling networks.

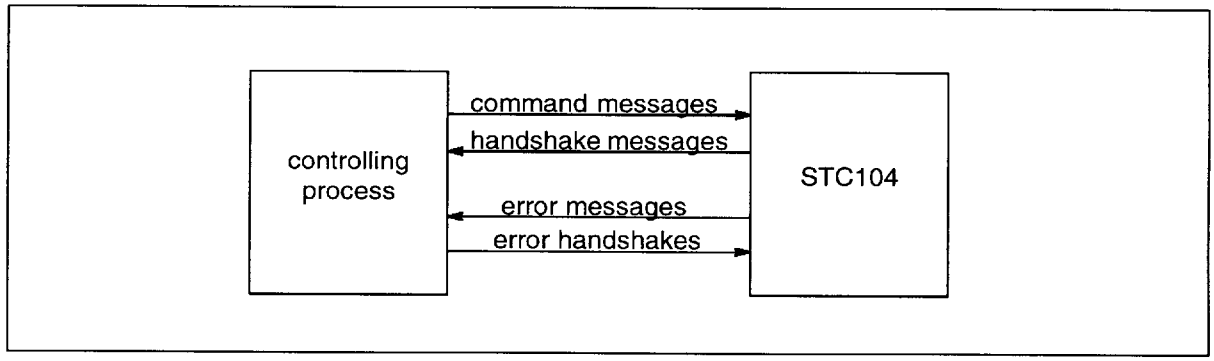


Figure 5.1 Communication between the controlling process and the STC104

The control links can be connected into a daisy chain or tree, with a controlling processor at the root. Figure 5.2 shows daisy-chained STC104's connected to one of the data DS-Links of a controlling processor, each STC104 has thirty-two data DS-Links but is shown as having just five data links for clarity.

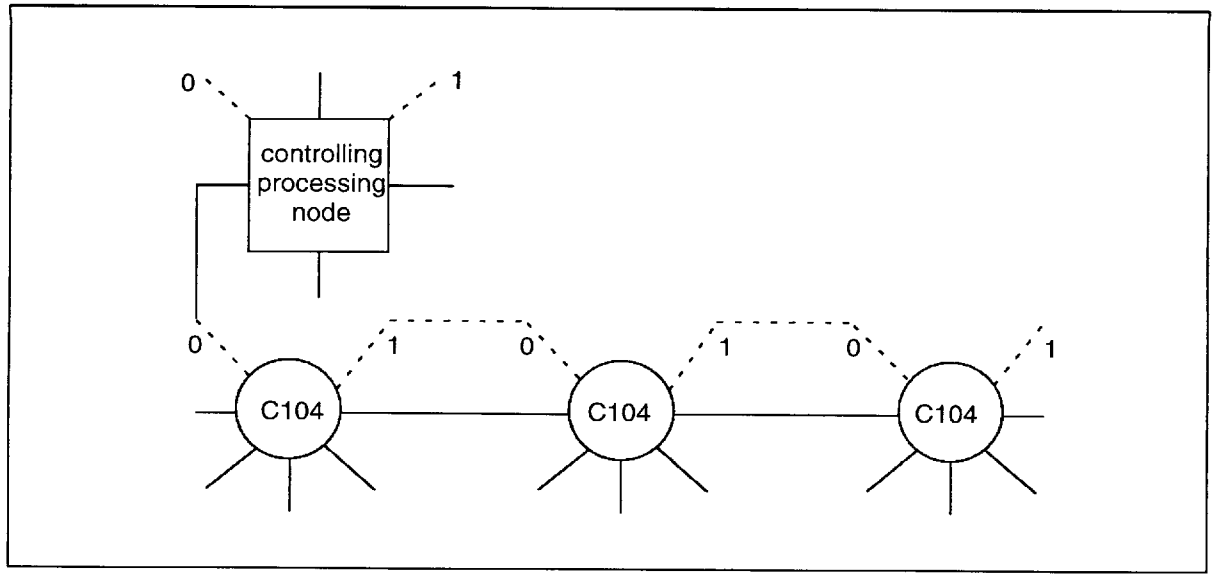


Figure 5.2 A daisy-chained control link network

In order to establish a connection between the controlling processor and each node, a label and return address must be given to each node. The label is used to establish whether or not a packet arriving on **CLink0** is for that node and if not the packet is forwarded down **CLink1**, if active, until it reaches its destination. If **CLink1** is not active the packet is discarded and a 'Control Protocol Error' message is sent (see page 30 for details on error messages). Any output must be prefixed by the return header in order to identify the node of origin to the controlling process, and to route the message through any STC104s used in the control network.

This provides a distinct connection between the controlling processor and each individual node of the application network.

5.1.1 Commands

A high level protocol is defined for the controlling network to allow the devices to issue commands to, and receive responses from, other devices in the network. Commands are sent as packets with the first byte after the header containing a command code, which may be followed by additional data.

In order that packets which are from different devices can be distinguished by the microprocessor which receives them, each packet contains a header which identifies the originating device. The packet header is also used to route the packet through a network. Bytes following the header are treated as the data section of the packet until a packet termination token is received. A packet termination token is either an EOP (end of packet) token or an EOM (end of message) token.

To avoid possible problems of buffer overrun or deadlock, a two level protocol is used on top of the basic packet exchange mechanism between the controlling processor and each device. This protocol applies independently to the interactions between each device and the controlling processor, and applies symmetrically in both directions. This protocol makes use of distinguished packets called *acknowledge packets*. An acknowl-

edge packet is a packet consisting only of a header and an EOP termination token. All other packets are referred to as *data packets*. At the lower level of the protocol, each data packet sent in either direction must be acknowledged by the transmission of an acknowledge packet in the opposite direction before another data packet may be sent. This ensures that data packets are not lost even if very limited buffering is provided. The only exception to this rule is the *RecoverError* command data packet which may be sent by the controlling processor even if no acknowledge packet has been received for a previously sent data packet.

The upper level of the protocol distinguishes two classes of data packets called *command packets* and *handshake packets*. Each command packet sent in either direction must be handshaken by the transmission of a handshake packet in the opposite direction before another command packet may be sent. The symmetrical exchange of a handshake for each command ensures that deadlock does not occur even if both the controlling processor and the device send a command packet at the same time. The only exception to this rule is *RecoverError* and *Reset* command packets which may be sent by the controlling processor even if no handshake packet has been received for a previously sent command packet.

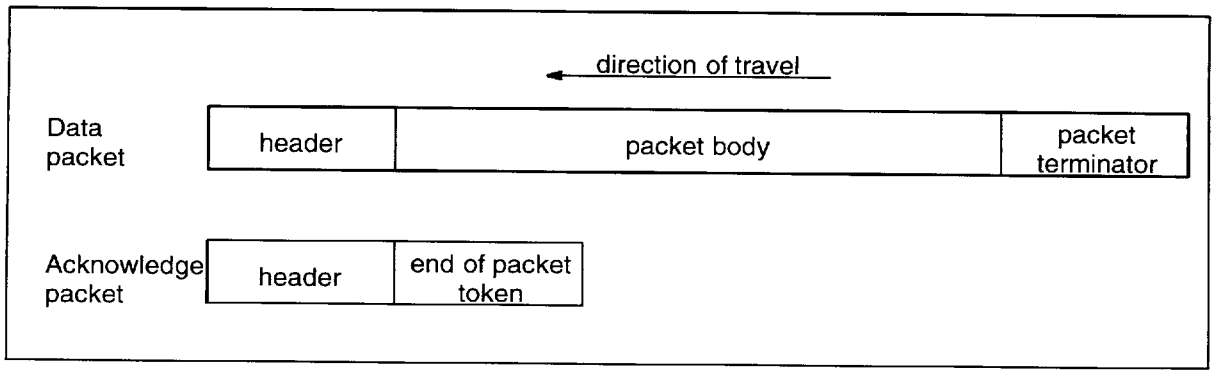


Figure 5.3 Structure of data and acknowledge packets

The handshake message can contain the result of a *CPeek* or *Identify* command, or it may be simply a handshake code corresponding to the command message. Each such message is preceded by the return header and followed by an EOM token. Command handshake codes are the same as the command codes except with the top bit inverted. Some of the handshake messages include a status byte which indicates whether the received command was valid as defined below.

- Status byte has value **0** if command is valid.
- Status byte has value **1** if command is invalid or has failed for some reason.

Each handshake message must be acknowledged by the controlling processing node by sending an acknowledge packet to the STC104.

The following section details the command messages which can be sent from a controlling processing node to the STC104.

Figures 5.4 and 5.5 show the command packets received by the STC104 and the handshake packets returned to the controlling process respectively. Note that the STC104

data links can receive headers of 1 or 2 bytes. Header labels of command packets are always 2 bytes, therefore an STC104 which is through routing command packets must be configured to accept 2 byte headers.

Start

This command programs the label and the return header of the STC104 in order to establish the virtual link between the controlling process and the STC104. The header of the first command received after power-up is taken by the control system as the 'label' for the STC104 and all subsequent messages with the same header are interpreted by the STC104. Therefore the first command sent should always be a *Start* command. Messages with a different header are forwarded via **CLink1** to the next device in the daisy-chain, if possible, otherwise they are discarded and a 'Control Protocol Error' message is sent (see page 30 for details on error messages).

The return header is 2 bytes long, with byte 0 being the first byte transmitted following the command code. Note that if this command is used to re-program the return header, the acknowledge for the command message packet will be sent with the *old* header, whilst the handshake will be sent with the *new* header.

The *Start* command must be the first command received. If an error occurs before the first *Start* command is received, the *StartHandshake* will be returned before the *Error* message is sent.

Reset

This command resets the STC104. The *Reset* command message includes a 'level' parameter. The level of reset is encoded in the 'level' byte of the command message. A *ResetHandshake* with a success status indicator (0) is sent on completion if the reset level is valid.

Note that a *Reset* command may cause a handshake for a previously transmitted command to be: terminated prematurely (with an EOM token); completed with a failure status; or suppressed entirely.

See chapter 7 for more information on reset levels.

Identify

The *Identify* command message causes the STC104 to respond with a handshake containing an identifier unique to the device type. This can be used to check the contents of a network. The lower 16 bits of the identifier are the same as the contents of the **DeviceID** register (see section 5.2.4); the upper 16 bits are zero.

RecoverError

This command is used in error recovery on the control links (see table 5.1). It restores the protocol after a link error in the control link system. Note that if there is an unhandshaken error, the *RecoverError* handshake will be returned before the error message is sent.

CPeek

The *CPeek* command includes a 2 byte address which points to a register in the configuration address space. The handshake message returns the value stored at the given address. If the address is invalid the handshake message returns an invalid status.

CPoke

The *CPoke* command includes a 2 byte address and 4 bytes of data. It is used to set the value of a configuration register. It writes the data to the configuration space register at the given address. If the address contained in the command message was invalid the status byte of the handshake message indicates failure.

Note, some configuration registers do not have a value, but writing to them causes some action to occur.

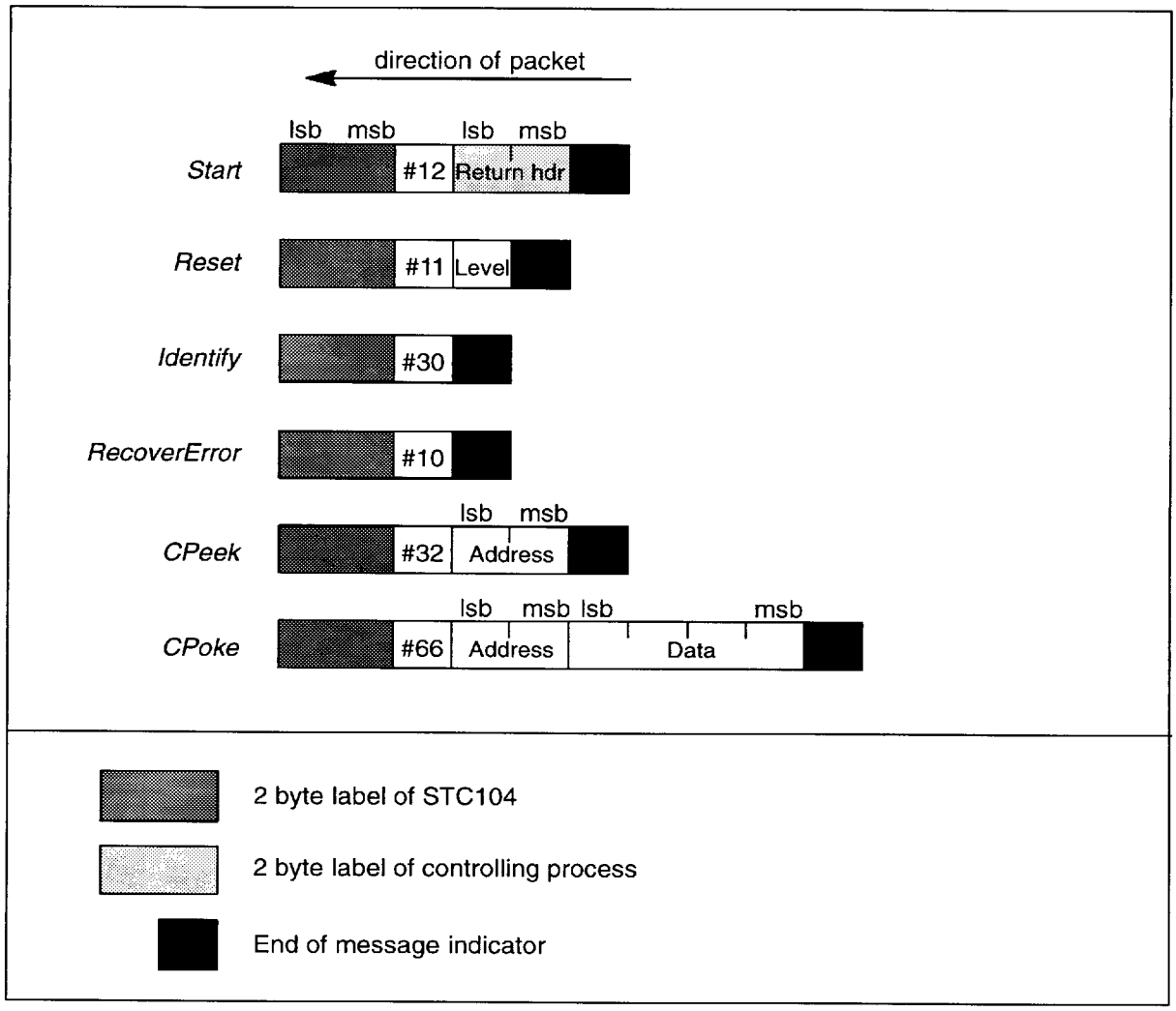


Figure 5.4 Control link command packets received by the STC104 on **CLink0**

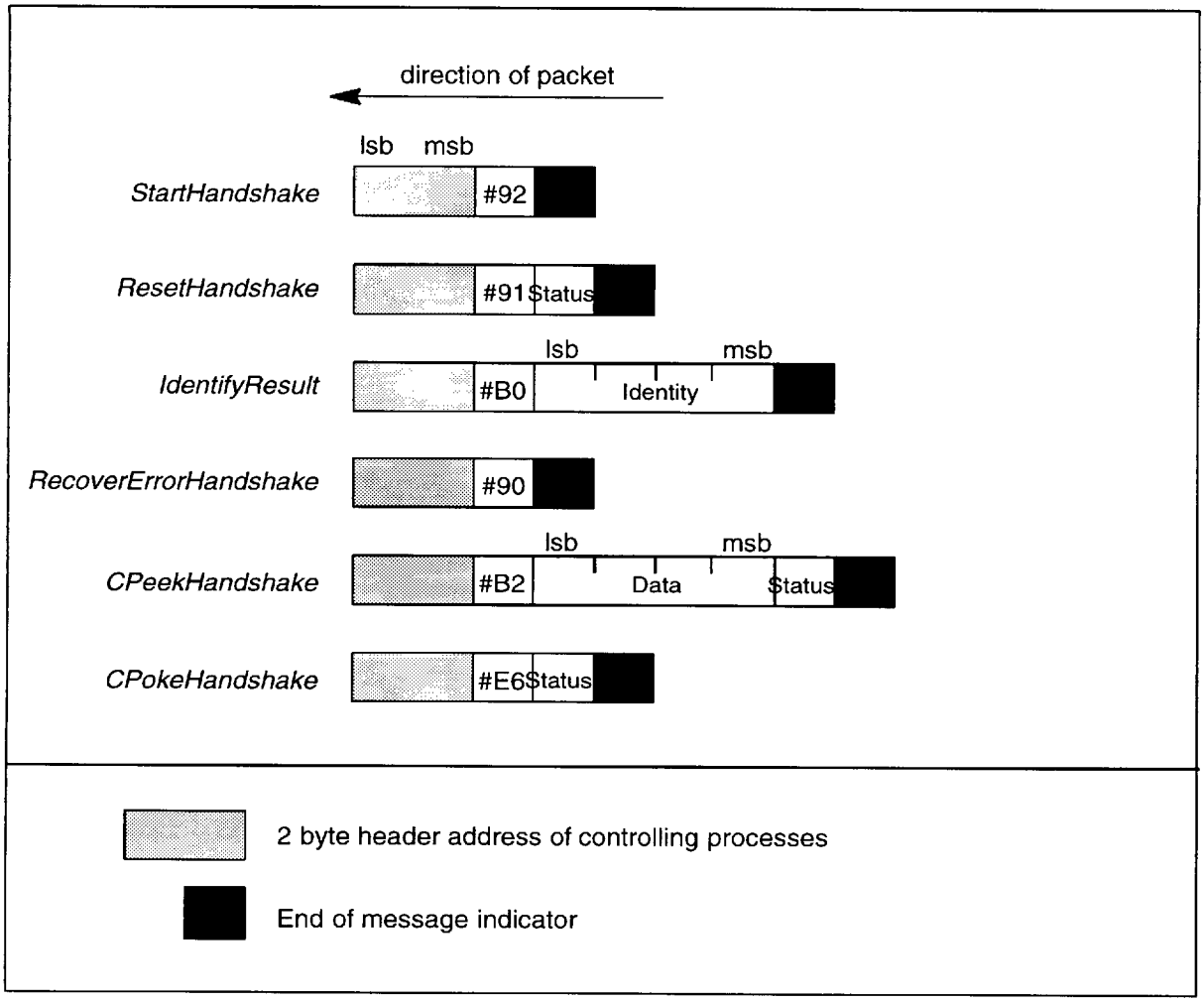


Figure 5.5 Handshake packets sent by the STC104 via **CLink0**

Errors

The STC104 can send an *Error* message to the controlling process to indicate that an error has occurred. The *Error* message contains an error code which determines the cause of error, as given in table 5.1. The error codes are contained in the first byte which accompanies the error message. The second byte records the number of the link on which the error was detected. For control link errors this second byte has no meaning, and is zero. Note that an error detected on one link has no direct effect on the operation of other links.

The *Error* message is a command packet as described above. All error messages must be handshaken by the control processor with the *ErrorHandshake* command.

The behavior of the links themselves in response to errors is described in section 6.3. The occurrence of a link error makes the link inactive, and so the input half of such a link will not make any requests for output links until the link is restarted. Neither will the output half grant any requests by input links, unless the **DiscardIfInactive** flag is set for that link, in which case requests will be granted and the corresponding packets consumed until the link becomes active again. This enables systems which are robust

against the loss of packets to avoid errors causing network blockages. The other types of error (Invalid header, Short packet, and Null packet) cause the offending packet to be discarded, but do not prevent the processing of subsequent packets.

The **ErrorCode** register contains the value of the most recent error.

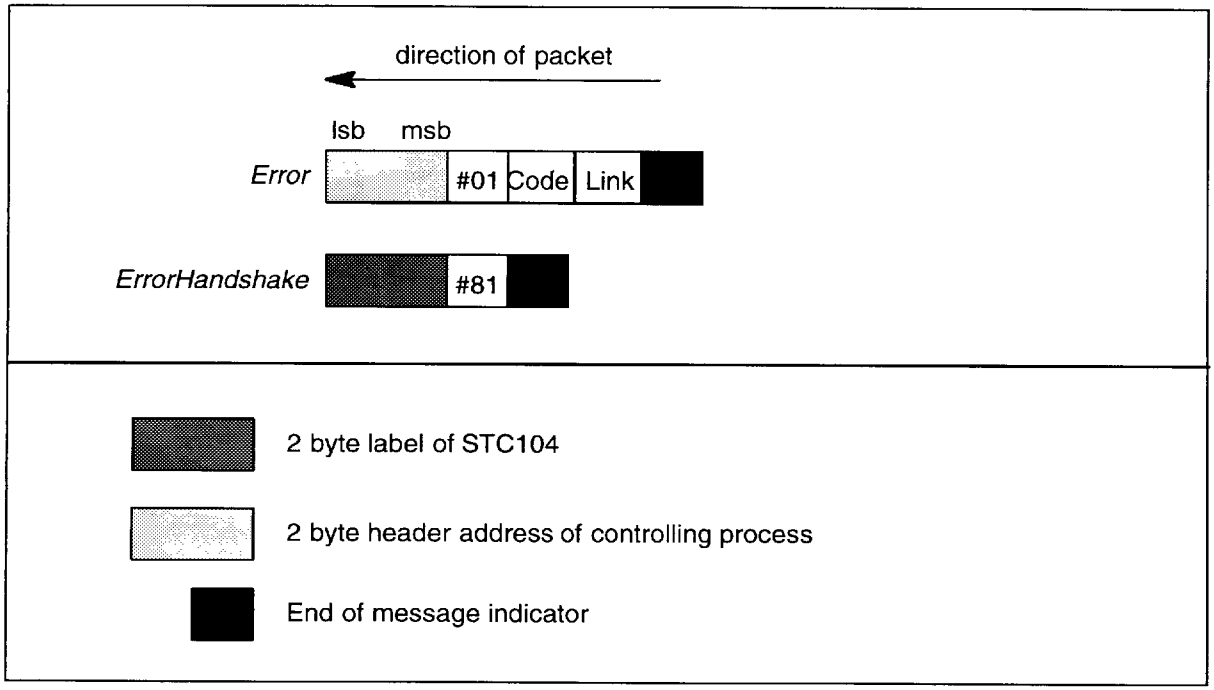


Figure 5.6 Error message

Code	Error type
#C0	Control command code error, i.e. unrecognized command.
#C1	Control protocol error, i.e. unsolicited acknowledge.
#C2	Control link 1 parity or disconnect error.
#80	Link parity or disconnect error.
#04	Invalid header – i.e. the header value received was not within the range of the interval selector.
#05	Short packet – i.e. packet is terminated before the header is received.
#06	Null packet – no bytes left after header deletion.

Table 5.1 Error codes

5.1.2 Control links used to provide fan-out in a control network

The same electrical and packet protocols are used for system control as for data transfer allowing large concurrent systems to be programmed, monitored and debugged in a simple way using virtual links. Large systems can use STC104 routers in the control network to improve fan-out.

Figure 5.7 gives an example of a daisy-chained control link network in which $STC104_1$ is used to route control link packets from the control processor to the application

network. In this example the controlled application network consists of a number of STC104s, and two data DS-Links of $STC104_1$ are connected to the control links of the application network to provide fan-out of the controlling system. **CLink1** is connected back to $STC104_1$ by data DS-Link (**Link0**), and used to route messages back to the control processor. Note that header labels of command packets are always 2 bytes, therefore $STC104_1$ must be configured to accept 2 byte headers. Note also that $STC104_1$ must be configured, and its links started, before any control messages are sent to the application network.

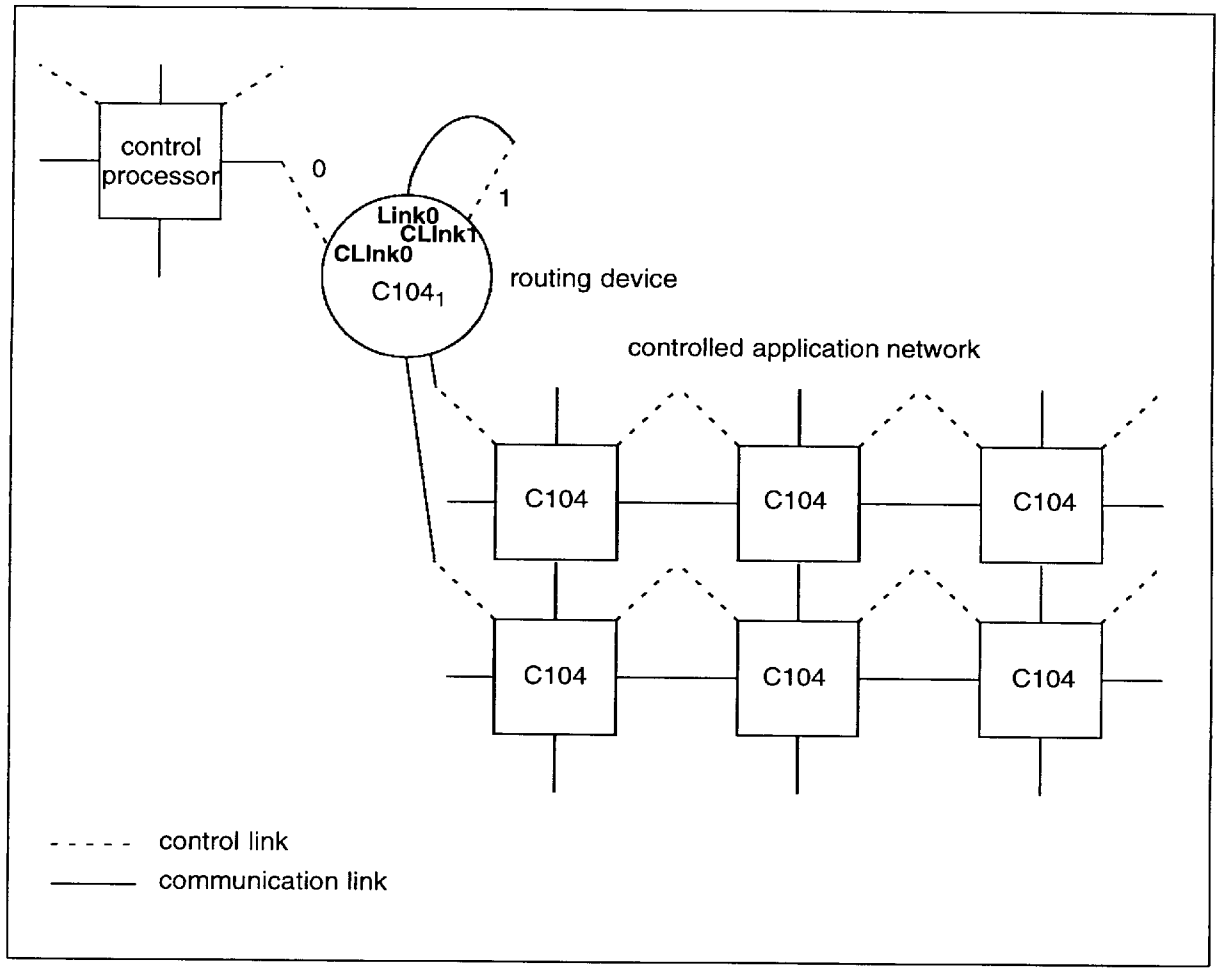


Figure 5.7 An STC104 providing fan-out.

5.1.3 Control link speeds

After power-on the control links run at a default speed of 10 MHz; this can be changed by means of *CPokes*. The speed selection for control links is identical to that of the data DS-Links (see section 6.2), and the control links share the same master clock.

5.2 Programmable configuration register functionality

This section gives the format and function of all the configuration registers in the STC104. The registers enable the STC104 to be programmed and are loaded by means of *CPoke* commands received on **CLink0**.

The tables below detail the bit fields of each of the registers and give the addresses, they also include whether each register is read only, write only, or read and writable. A complete listing of the register addresses is given in chapter 8.

All registers are 32 bits long, and 32 bits are always read or written. When writing to the registers, all reserved/undefined bits must always be zero, unless otherwise stated. The values returned from these bits on a configuration read is undefined.

Note that in the following bit field descriptions the lowest numbered bit is the least significant bit.

5.2.1 Packet processor registers

The functionality to be controlled by the packet processor configuration registers, and the associated bit fields are described below.

PacketMode0-31

The **PacketMode0-31** registers (one for each of the thirty-two links) must be programmed before the corresponding link is started. They may be re-programmed before or after the link has been started.

PacketMode0-31	#8080 to #9F80	Read/Write
Bit	**Bit field**	**Function**
0	**Randomize**	Sets a given link input to random header generation mode. When set to 1, a random header is added to the front of each incoming packet.
1	**ContinueGroup**	Each bit can be set to 'start of group' or 'continuation of group'. 0 start 1 continue
2	**HeaderDeletion**	Sets a given link output to delete header mode. When set to 1, the first byte of each packet is deleted.
3	**DiscardIfInactive**	When set to 1, all packets directed to this link will be discarded in their entirety if the link is inactive, for example if the link has not been started or an error has been detected. When set to 0, packets directed to this link are stalled if the link is not active.
4	**HeaderLength**	Sets the link to input headers 1 or 2 bytes long. 0 1 byte header 1 2 byte header If 2 byte headers are expected, the least significant 8 bits are equal to the first byte received, and the most significant 8 bits are equal to the second byte received (little-endian convention). If 1 byte headers are expected, for purposes of comparison with the **Separator** fields of the **Interval1–36** registers, the most significant 8 bits of the header value are 0, and the least significant 8 bits are equal to the received byte. Note that if a two byte header is expected and only one byte is received before the packet terminates, an error is flagged and the packet is discarded.
31:5		Reserved, write 0.

Table 5.2 Bit fields in the packet mode (**PacketMode0-31**) register – one register per link

PacketCommand0-31

The **PacketCommand0-31** registers (one for each of the thirty-two links) contain a bit which when set causes a reset of the associated packet processor and link.

PacketCommand0-31	#80A8 to #9FA8	Read/Write
Bit	**Bit field**	**Function**
0	**Reset**	When set to 1 forces the link and packet processing logic into a reset state.
31:1		Reserved, write 0.

Table 5.3 Bit fields in the packet command (**PacketCommand0-31**) register – one register per link

Interval1-36

There are 36 programmable interval registers per link, called **Interval1** to **Interval36**. There is a nominal register **Interval0**, which is not programmable and always holds the value zero.

Interval1-35	#80x to #9Fx where x is #80+(#1 to #23)	Read/Write
Bit	**Bit field**	**Function**
15:0	**Separator**	Sets the interval separator for each link.
16	**Discard**	Designates the portal interval. When set discards the header of the packet.
17	**Invalid**	Designates an invalid interval.
22:18	**SelectLink**	Indicates the associated link from which the packet is to be output.
31:23		Reserved, write 0.

Table 5.4 Bit fields in each of the interval (**Interval1-35**) registers for each of the 32 links

The **Interval1-35** registers for each link must be programmed before the link is started.

The register **Interval36** has a nominal **Separator** value which is greater than any header representable in 16 bits. Its **Separator** field is not programmable. Note that whenever it is not required to use the maximum possible header value (this will be the case for all but the very largest systems), the **Interval36** register should have its **Invalid** bit set.

Interval36	#80A4 to #9FA4	Read/Write
Bit	**Bit field**	**Function**
16	**Discard**	When set discards the header of the packet.
17	**Invalid**	Designates an invalid interval.
22:18	**SelectLink**	Indicates the associated link from which the packet is to be output.
15:0, 31:23		Reserved, write 0.

Table 5.5 Bit fields in the maximum interval (**Interval36**) register for each of the 32 links

RandomBase0-31, RandomRange0-31 and RandomSeed0-31

When the **Randomize** flag in the **PacketMode0-31** register is set, the link is in random header mode and each arriving packet is routed depending on a pseudo-randomly generated header. The header is generated within a range determined for each link by two 16-bit unsigned programmable registers, **RandomBase** and **RandomRange**. **RandomRange** must be 1. Headers are generated in the range **RandomBase** to (**RandomBase** + **RandomRange** –1) inclusive. Note that this sum is modulo 2^{16} and may 'wrap around' zero. The seed of the pseudo-random sequence for each link is loaded into the register **RandomSeed** and must not be zero. Note that these registers must be loaded with known values in order to ensure repeatable behavior. Also, no two **RandomSeed** registers should be loaded with the same value.

RandomBase0-31	**#80A5 to #9FA5**	**Read/Write**
Bit	**Bit field**	**Function**
15:0	**RandomBase**	16 bit unsigned value of random header base level.
31:16		Reserved, write 0.

Table 5.6 Bit fields in the **RandomBase0-31** register – one register per link

RandomRange0-31	**#80A6 to #9FA6**	**Read/Write**
Bit	**Bit field**	**Function**
15:0	**RandomRange**	16 bit unsigned value of random header range.
31:16		Reserved, write 0.

Table 5.7 Bit fields in the **RandomRange0-31** register – one register per link

RandomSeed0-31	**#80A7 to #9FA7**	**Read/Write**
Bit	**Bit field**	**Function**
15:0	**RandomSeed**	Start of 16 bit pseudo-random sequence.
31:16		Reserved, write 0.

Table 5.8 Bit fields in the **RandomSeed0-31** register – one register per link

Note that if the **RandomSeed** value is changed after the link has been started, the new value may not be used until a write to the **ConfigComplete** register (see table 5.16) has been performed.

5.2.2 Data DS-Link registers

Each of the 32 links has three registers, the **LinkMode** register, **LinkCommand** register and **LinkStatus** register. In addition the configuration space contains the **DSLinkPLL** register which contains the **SpeedMultiply** bit field (see section 5.2.4). This takes the 5 MHz input clock and multiplies it by a programmable value to provide the root clock for all the DS-Links.

The tables below describe the functionality of the DS-Links to be controlled, and the associated bit fields in the configuration registers. For more information on the meaning of these bit fields refer to the Data/Strobe links chapter 6.

Link0-31Mode

The **Link0-31Mode** registers may be re-programmed before or after the link has been started.

Link0-31Mode	#8001 to #9F01	Read/Write
Bit	**Bit field**	**Function**
1:0	**SpeedDivide**	Sets transmit speed of the **Link0-31** (see table 6.2). 00 = /1, 01 = /2, 10 = /4, 11 = /8
2	**SpeedSelect**	Sets the **Link0-31** to transmit at the speed determined by the **Speed-Divide** bits as opposed to the base speed of 10 Mbits/s.
3	**LocalizeError**	Packets in transit at the time of an error will be discarded or truncated. When set false communication on the link stops until the link is reset.
4	**1 (RESERVED)**	This bit should be written as 1.
31:5		Reserved, write 0.

Table 5.9 Bit fields in the **Link0-31Mode** registers – one register per link

Link0-31Command

The **Link0-31Command** registers contain four bits which when set cause a specific action to be taken by the DS-Link.

Link0-31Command	#8002 to #9F02	Write only
Bit	**Bit field**	**Function**
0	**ResetLink**	Resets the link engine of the **Link0-31**. The token state is reset, the flow control credit is set to zero, the buffers are marked as empty, the parity state is reset, and the link stops sending tokens.
1	**StartLink**	When a transition from 0 to 1 occurs **Link0-31** will be initialized and commence operation.
2	**ResetOutput**	Sets both outputs of **Link0-31** low.
3	**WrongParity**	The **Link0-31** output will generate incorrect parity. This may be used to force a parity error on the device at the other end of the **Link0-31**.
31:4		Reserved, write 0.

Table 5.10 Bit fields in the **Link0-31Command** registers – one register per link

Link0-31Status

The **Link0-31Status** registers contain information about the state of the DS-Link.

Link0-31Status	#8003 to #9F03	Read only
Bit	**Bit field**	**Function**
0	**LinkError**	Flags that an error has occurred on the **Link0-31**.
1	**LinkStarted**	Flags that the output **Link0-31** has been started and no errors have been detected.
2	**ResetOutputComplete**	Flags that **ResetOutput** has completed on the **Link0-31**.
3	**ParityError**	Flags that a parity error has occurred on the **Link0-31**.
4	**DiscError**	Flags that a disconnect error has occurred on the **Link0-31**.
5	**TokenReceived**	Flags that a token has been seen on the **Link0-31** since **ResetLink**.
31:6		Reserved, write 0.

Table 5.11 Bit fields in the **Link0-31Status** registers – one register per link

5.2.3 Control link registers

The link module hardware in each control link is identical to that in each data DS-Link. An equivalent set of configuration bit fields is provided in the **CLink0-1Mode**, **CLink0-1Command** and **CLink0-1Status** registers for the control links, as for the data DS-Links (see section 5.2.2).

5.2.4 System services registers

System services consists of a block of 5 configuration registers which contain control information and general information. The functionality to be controlled by the system services configuration registers, and the associated bit fields are described below.

DeviceID

The **DeviceID** register contains a 16 bit device identification code unique to the device. The value of the device identification code for the STC104 is 384. The device identification code can also be read using the *Identify* command.

DeviceID	#1001	Read only
Bit	**Bit field**	**Function**
15:0	**DeviceID**	Device identification code.
31:16		Reserved, write 0.

Table 5.12 Bit fields in the **DeviceID** register

DeviceRevision

The **DeviceRevision** register contains the revision of the device.

DeviceRevision	#1002	Read only
Bit	**Bit field**	**Function**
15:0	**DeviceRev**	Device revision.
31:16		Reserved, write 0.

Table 5.13 Bit fields in the **DeviceRevision** register

ErrorCode

The **ErrorCode** register is a 13 bit register used for debugging. It contains the value of the error code representing the most recently occurring error and the number of the link on which the error occurred.

ErrorCode	#1004	Read only
Bit	**Bit field**	**Function**
7:0	**ErrorCode**	Contains an error code which can be used for debugging after a crash. Refer to table 5.1, page 31 for the error code definitions.
12:8	**LinkNumber**	Number of the link on which the error occurred.
31:13		Reserved, write 0.

Table 5.14 Bit fields in the **ErrorCode** register

DSLinkPLL

The **DSLinkPLL** register contains the **SpeedMultiply** bit field and is used to program the DS-Link speeds. This takes an internally generated 10 MHz clock and multiplies it by a programmable value to provide the root clock for all the DS-Links. Refer to section 6.2 in the Data/Strobe links chapter for further details.

Note that this register should not be loaded with any value less than eight.

DSLinkPLL	#1005	Read/Write
Bit	**Bit field**	**Function**
5:0	**SpeedMultiply**	Sets link master clock to required value (see Data/Strobe links chapter).
31:6		Reserved, write 0.

Table 5.15 Bit fields in the **DSLinkPLL** register

ConfigComplete

Once the configuration registers have been set up a write to the **ConfigComplete** register initializes the STC104. The output groups are then set up corresponding to the current values of the **ContinueGroup** flags, and the random header generators are started for all links whose **Randomize** flags are set. If the configuration is subsequently changed a write should be made to this register when the set of changes is complete.

A write must also be performed to this register to restart the random number generators after any type or level of reset.

Note that a write to this register may cause a temporary stall of packets flowing through the STC104.

ConfigComplete	#1003	Write only
Bit	**Bit field**	**Function**
0	**ConfigComplete**	A write to this register sets up the output groups for the links and starts the random header generators.

Table 5.16 Bit fields in the **ConfigComplete** register

5.3 Initialization of the STC104

The value of the **DSLinkPLL** register must be set before the STC104 can operate. For each link in use, the following parameters must be supplied:

- Link **SpeedSelect** flag, and if set to 1, the value of the **SpeedDivide** field.
- Link **LocalizeError** flag.
- **HeaderLength** flag.
- Values of the **Interval** registers.
- **Randomize** flag, and if set to 1, the values of the **RandomBase**, **RandomRange** and **RandomSeed** registers.
- **ContinueGroup** flag.
- **HeaderDeletion** flag.
- **DiscardIfInactive** flag.

Once the configuration registers have been set up a write to the **ConfigComplete** register initializes the STC104. If the configuration is subsequently changed a write should be made to the **ConfigComplete** register when the set of changes is complete.

Note that none of the data links should be started until a write has been performed to the **ConfigComplete** register. If the configuration is changed while packets are being routed by the device, the results may be non-deterministic.

6 Data/Strobe links

The STC104 has 32 links used for routing, and two control links which are used for monitoring and control purposes only. All of these links use a protocol with two wires in each direction, one for data and one to carry a strobe signal and are referred to as data/strobe (DS-Links). The DS-Links are capable of:

- Up to 100 Mbits/s per link.
- Unidirectional peak bandwidth of 10 Mbytes/s per link.
- Support for virtual channels and through routing.

Each DS pair carries tokens and an encoded clock. The tokens can be data or control tokens. Figure 6.1 shows the format of data and control tokens on the data and strobe wires. Data tokens are 10 bits long and consist of a parity bit, a flag which is set to 0 to indicate a data token, and 8 bits of data. Control tokens are 4 bits long and consist of a parity bit, a flag which is set to 1 to indicate a control token, and 2 bits to indicate the type of control token.

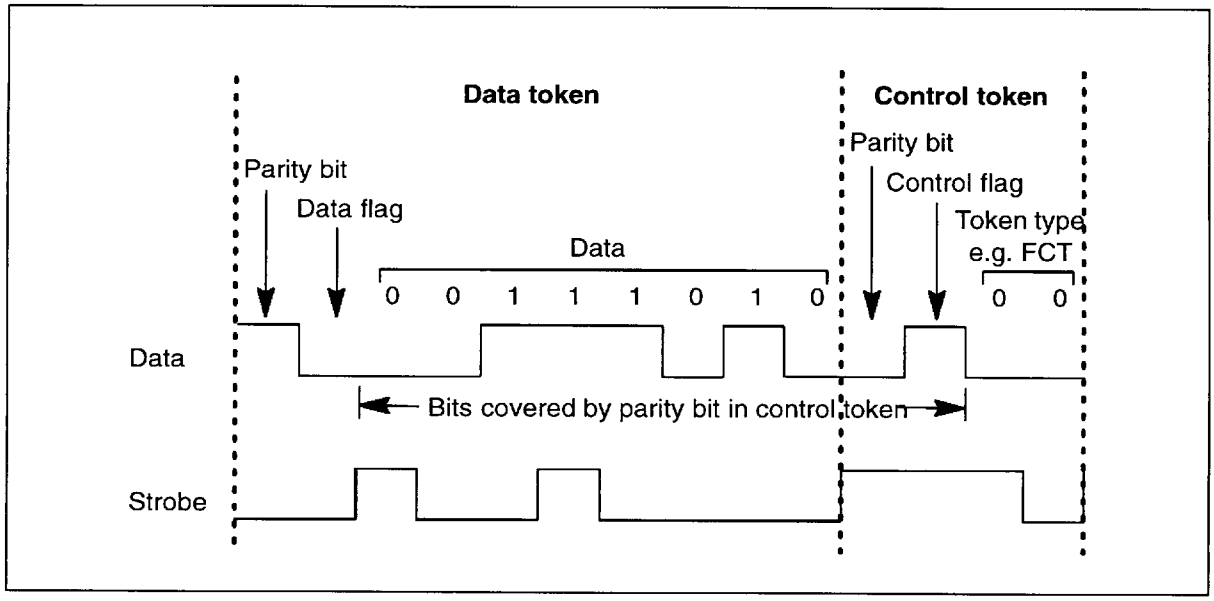


Figure 6.1 Link data and strobe formats

The DS-Link protocol ensures that only one of the two wires of the data strobe pair has an edge in each bit time. The levels on the data wire give the data bits transmitted. The strobe signal changes whenever the data signal does not. These two signals encode a clock together with the data bits, permitting asynchronous detection of the data at the receiving end.

The data and control tokens are of different lengths, for this reason 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 6.1. This allows single bit errors in the token type flag to be detected. **Odd** parity checking is used. Thus the parity bit is set/unset to ensure that the bits covered, inclusive of the parity bit (see figure 6.1), always

contain an odd number of 1's. The coding of the tokens is shown in table 6.1. To ensure the immediate detection of parity errors and to enable link disconnection to be detected null tokens are sent in the absence of other tokens.

Token type	Abbreviation	Coding
Data token	–	P0DDDDDDDD
Flow control token	FCT	P100
End of packet	EOP	P101
End of message	EOM	P110
Escape token	ESC	P111
Null token	NUL	ESC P100

P = parity bit
D = data bit

Table 6.1 Token codings

6.1 Low-level flow control

Token-level flow control is performed in each DS-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 (20 tokens of buffering is in fact provided). Whenever the link input has sufficient buffering available to consume a further 8 tokens a 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.

6.2 Link speeds

The STC104 links can support a range of communication speeds, which are programmed by writing to registers using the *CPoke* command via control link **CLink0**. At reset all links are configured to run at the **BaseSpeed** of 10 Mbits/sec.

Only the transmission speed of a link is programmed as reception is asynchronous. This means that links running at different speeds can be connected, provided that each device is capable of receiving at the speed of the connected transmitter.

The transmission speeds of all of the links on a given device are related to the speed of a single on-chip clock. The frequency of this master clock is programmed through the **SpeedMultiply** bit field described in section 5.2.2. The master frequency is divided down to obtain the transmission frequency for each link. The division factor can be programmed separately for each link via the **SpeedDivide** bit field described in section 5.2.2. For a given device, with a given programmed master clock frequency, this

arrangement allows each link to be run at one of four transmission speeds, as shown in table 6.2.

SpeedMultiply	SpeedDivide				BaseSpeed
	/1	/2	/4	/8	
8	40	20	Reserved	Reserved	10
10	50	25	Reserved	Reserved	10
12	60	30	Reserved	Reserved	10
14	70	35	Reserved	Reserved	10
16	80	40	Reserved	Reserved	10
18	90	45	Reserved	Reserved	10
20	100	50	Reserved	Reserved	10

Table 6.2 Link transmission speed in Mbits/s

6.3 Errors on links

Link inputs can detect parity and disconnection conditions as errors. A single bit odd parity system will detect single bit errors at the link token level. The protocol to transmit NUL tokens in the absence of other tokens enables disconnection of a link to be detected. A disconnection error indicates one of two things:

- the link has been physically disconnected;
- an error has occurred at the other end of the link, which has then stopped transmitting.

The **LinkError** bit in the **Link0-31Status** registers flags that a parity and/or disconnection error has occurred on the **Link0-31**. The bit fields **ParityError** and **DiscError** indicate when parity and disconnect errors occur respectively.

When a DS-Link detects a parity error on its input it halts its output. This is detected as a disconnect error at the other end of the link, causing this to halt its output also. Detection of an error causes the link to be reset. Thus, the disconnect behavior ensures that both ends are reset. Each end can then be restarted.

Note that a disconnect error is only flagged once a token has been received on a link and transmission is subsequently interrupted. Therefore when one end of a link is started up before the other end of a link, a disconnect error does not occur as no tokens have yet been received. As soon as the other end of the link is started communication can begin immediately.

DS-Links are designed to be highly reliable within a single subsystem and can be operated in one of two environments, dependent on the level of reliability required. A DS-Link can be set to an environment in which any link errors are localized to the link. This is set by the **LocalizeError** bit in the **Link0-31Mode** register. The **LocalizeError** bit is set on a per link basis, therefore it is possible to have some links in a system set to localize link errors and other links which are not. The consequence of a link error depends on which environment the link is in, as described below.

6.3.1 Reliable links

In the majority of applications, the communications system should 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. This environment is achieved by setting the **LocalizeError** bit in the **Link0-31Mode** registers to 0. Normal practice will then be to reset the subsystem in which the error has occurred and to restart the application.

6.3.2 More reliable links

For some applications, for instance when a disconnect or parity error may be expected during normal operation, an even higher level of reliability is required. This level of fault tolerance is supported by localizing errors to the link on which they occur. This is achieved by setting the **LocalizeError** bit in the **Link0-31Mode** register to 1. A link error in this mode results in packets in transit at the time of the error being discarded or truncated.

If the failed link is not grouped with any other links, the **DiscardIfInactive** bit in the **PacketMode0-31** register should be set to 1, so that the link discards any packets routed to it. This prevents the network being blocked by packets routed via that link.

If the failed link is grouped with one or more links, and the **DiscardIfInactive** bit is set to 0, packets will automatically be directed to other links in the same group.

Note that these mechanisms apply at any time the link is inactive, not just after the occurrence of an error.

6.4 Link state on start up

After power-on all **LinkData** and **LinkStrobe** signals are low, without clocks. Following power-on reset an initialization sequence sets the speed of the link clock. The DS-Links are initially inactive. They are configured and started by configuration writes. Their status can be determined by configuration reads.

Each DS-Link (**Link0-31**) must be explicitly started by writing to the **StartLink** bit in its **LinkCommand** register, with the exception of **CLink0** which starts as soon as it receives a token. When a DS-Link is started up it transmits control tokens.

Data may not be transferred over the link until the receiving link has sent a FCT, which it will do as soon as it has been started. The data/strobe outputs are held low until the first FCT is sent.

The receiving link receives and correctly decodes the tokens. However, only when the receiving link has been explicitly started by writing across the (internal) configuration bus can it send tokens back. NUL tokens are then sent until data is required.

6.5 Resetting DS-Links

If one end of a running DS-Link is reset, that end of the link stops transmitting tokens on a token boundary and any buffered data is discarded. The other end of the link detects a disconnection and also stops transmission. The reset end then also detects disconnection and clears its flow-control state and error status bits, and the link becomes insensitive to transitions on its input for 3.2 μs. In order to ensure that both ends of the link have completed reset and are sensitive to transitions before either end is started there is a further delay of 12.8 μs. Note that the Data and Strobe outputs are simply held at the values they have at the end of the last transmitted token, since forcing them to zero could be decoded as a bit by the other end of the link.

Since the disconnection protocol between the two ends of a DS-Link ensures that both ends become reset automatically if an error is detected, there is normally no reason to explicitly reset either end. However, one end may be reset as a consequence of a reset of a device or subsystem. In this case it is important to ensure that either: both ends of the link have been started before the reset occurs; or that both ends are quiet (by resetting if necessary). This is because if one end of a DS-Link is already running before the other end comes out of reset, the initial transmission of FCTs will be lost, and so the reset end will never receive permission to transmit data. Also, unless the reset end is brought out of reset precisely on a null token boundary (for which there is 1 chance in 8), it will misinterpret the bit-stream and consequently detect a parity error.

6.6 Link connections

DS-Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 200 millimeters. For longer distances a matched 100 ohm transmission line should be used, see figure 6.2.

The inputs and outputs have been designed to have minimum skew at the 1.5 V TTL threshold.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

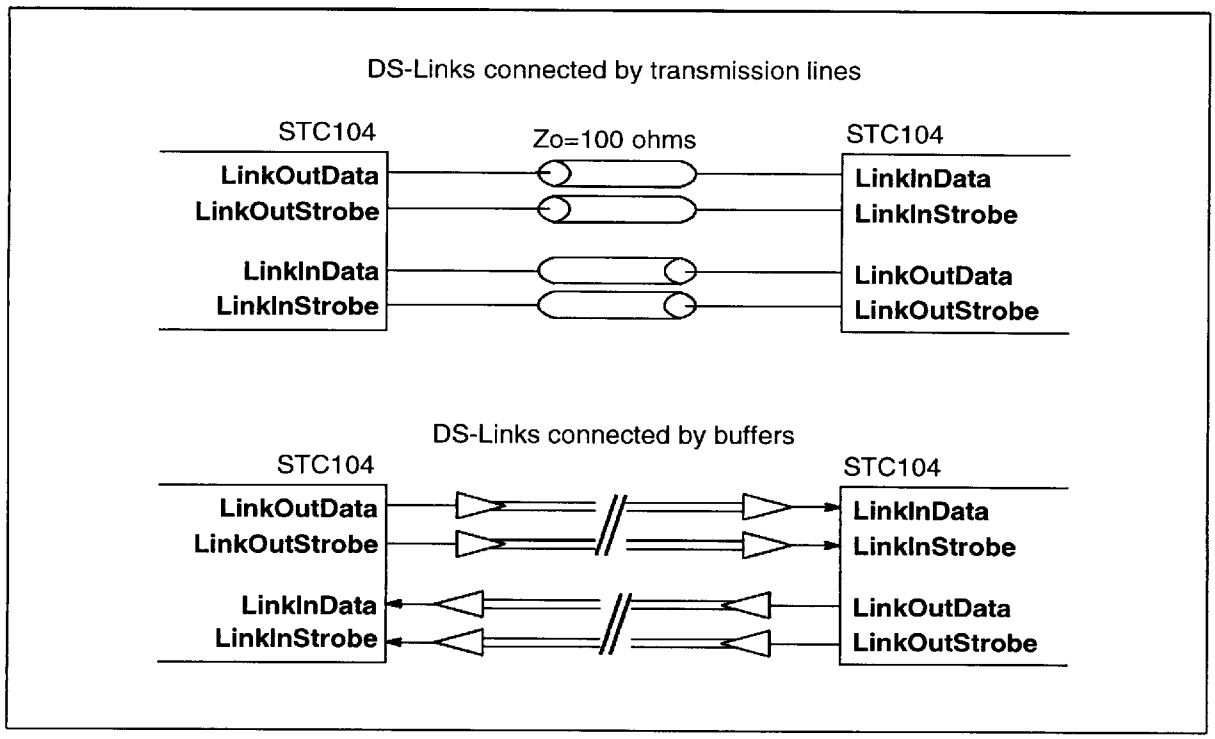


Figure 6.2 DS-Link connections

7 Levels of reset

The STC104 can be reset to a given level using the *Reset* command or **Reset** pin. Setting the **Reset** pin high for 1 cycle of **ClockIn**, resets the chip. The *Reset* command received along **CLink0** also resets the STC104, but in this case the control links and any stored label and return header values are not reset. Any reset, except a *Reset3* command, results in any packets currently being routed within the STC104 being lost. The different levels of reset are described below.

Note that any level of reset may abort the command which was executing when the *Reset* command was applied. An illegal level of *Reset* will also result in a handshake with a failure status being returned.

7.1 Level 0 – hardware reset

The network can be returned to level 0 by taking all the **Reset** pins in the network high for a number of cycles.

After a hardware reset each STC104 is in the following state:

All writeable configuration registers are undefined, all state machines are in their initial state and all links (data and control) are inactive with their output pins low and set to run at the default speed of 10 MHz. The label and return headers for the control links are undefined. All buffers and latched error conditions are cleared.

7.2 Level 1 – labelled control network

The network can be reset to level 1 by sending a *Reset1* command message to each STC104.

This level of reset leaves the identity and return headers unaltered and all connected control links remain operational. All the data DS-Links are inactive with their output pins low and set to run at the default speed of 10 MHz. All data in the STC104 is lost.

7.3 Level 2 – configured network

The network can be reset to level 2 by sending a *Reset2* command message to each STC104.

At this level of reset the identity and return headers are unaltered and register contents are unaffected. All data in the STC104 is lost. The data DS-Links are reset and returned to their inactive state. The control links are not affected.

7.4 Level 3

Reset levels above 2 are not applicable to the STC104. If a reset level above 2 is received it is handshaken with status set to false.

7.5 Per link reset

In order to preserve the logical partitioning of the STC104, associated with each link is a **ResetLink** bit (in the **LinkCommand** register) to perform reset on that link only.

Setting the **ResetLink** bit to 1 resets the link and puts all associated logic into a reset state. The logic remains in this state until the bit is set back to 0.

Since the resetting logic may cause, for example, a request from a packet processor for an output to be withdrawn just as the output grants the request, it is essential for correct operation to ensure that all links in a partition are put into the reset state before any of them are taken out of it. When all of the packet processing logic has been restarted, the links themselves may be restarted in the usual way.

7.6 Effects of different levels of Reset

The *Reset* command is accompanied by a 'level' parameter. The effect of reset levels 1 and 2 on various aspects of the STC104 state is summarized in table 7.1; a *ResetHandshake* with a status indicator of True (0) is sent on completion. Any other value of the level parameter causes the status of the reset handshake to be False (1): no other action is taken.

The handshake state indicates whether the control unit expects a handshake message; the acknowledge state indicates whether it expects to receive an acknowledge packet; and the error state is the latched error signals which would otherwise cause *Error* messages to be sent. When the handshake/acknowledge state is cleared any outstanding handshakes/acknowledges will be ignored. The corresponding effects of the *RecoverError* command are also shown. The *RecoverError* command resets the acknowledge state so that acknowledges are neither expected nor pending, and causes the re-transmission of any unhandshaken error message.

State	Reset level		Recover Error
	1	2	
Data DS-Links	Re-configured	Cleared	no effect
Packet processors	Re-configured	Cleared	no effect
Handshake state	Cleared	Cleared	Cleared
Acknowledge state	no effect	no effect	Cleared
Error state	Cleared	Cleared	no effect

Table 7.1 Effects of the different levels of reset on various aspects of the STC104 state

8 Configuration register addresses

The complete bit format of each of the configuration registers is given in section 5.2.

8.1 Subsystem addresses

The registers in the configuration space are accessed via *CPeek* and *CPoke* command messages received along **CLink0**. A 2 byte 16 bit address is issued, the most significant byte refers to the subsystem, the least significant byte refers to the local register within the subsystem.

There are 35 subsystems connected to the configuration bus:

- 32 data links and packet processors
- 2 control links
- system services

Table 8.1 gives the addresses of each of the subsystems.

The subsystem set of all 32 data DS-Links has a unique address #FF, as does the subsystem set of all 32 packet processors. Note that the packet processors have the same subsystem addresses as the data DS-Links. This address referring to the set of DS-Links and packet processors should be used when writing (poking) to the associated register for each of the 32 DS-Links or packet processors. For example, to simultaneously write to all 32 DS-Link command registers (**Link0-31Command**), address #FF02 should be used.

Subsystem	Hex address
System services	#10
Link0	#80
Link1	#81
Link2	#82
Link3	#83
Link4	#84
Link5	#85
Link6	#86
Link7	#87
Link8	#88
Link9	#89
Link10	#8A
Link11	#8B
Link12	#8C
Link13	#8D
Link14	#8E
Link15	#8F
Link16	#90
Link17	#91
Link18	#92
Link19	#93
Link20	#94
Link21	#95
Link22	#96
Link23	#97
Link24	#98
Link25	#99
Link26	#9A
Link27	#9B
Link28	#9C
Link29	#9D
Link30	#9E
Link31	#9F
Control link0	#FD
Control link1	#FE
All data links (Link0-31)	#FF

Note: The packet processor subsystems have the same subsystem addresses as the data links.

Table 8.1 Subsystem addresses

8.2 Register addresses

The configuration registers are given local addresses within each subsystem, these are given in the tables below.

8.2.1 Packet processor configuration registers

Register	Local address	Bit size	Read/Write
PacketMode	#80	5	R/W
IntervalN (N = 1 to 36)	#80 + (#1 to #24)	23	R/W
RandomBase	#A5	16	R/W
RandomRange	#A6	16	R/W
RandomSeed	#A7	16	R/W
PacketCommand	#A8	1	R/W

Table 8.2 **PacketProcessor0-31** configuration registers

8.2.2 Link configuration registers

Register	Local address	Bit size	Read/Write
LinkMode	#01	8	R/W
LinkCommand	#02	4	W
LinkStatus	#03	6	R
LinkWriteLock*	#04	1	R/W

* This register is not used on the STC104.

Table 8.3 **Link0-31** configuration registers

8.2.3 Control link configuration registers

Register	Local address	Bit size	Read/Write
CLinkMode	#01	8	R/W
CLinkCommand	#02	4	W
CLinkStatus	#03	6	R
CLinkWriteLock*	#04	1	R/W

* This register is not used on the STC104.

Table 8.4 **CLink0-1** configuration registers

8.2.4 System services configuration registers

Register	Local address	Bit size	Read/Write
DeviceID	#01	16	R
DeviceRevision	#02	16	R
ConfigComplete	#03	0	W
ErrorCode	#04	16	R
DSLinkPLL	#05	5	R/W

Table 8.5 System services configuration registers

9 Clocks

Two on-chip phase locked loops (PLL) generate all the internal high frequency clocks from a single clock input, simplifying system design and avoiding problems of distributing high speed clocks externally. This chapter details the PLL input specifications and decoupling requirements. There is one PLL for the system clocks and one for the link clocks.

9.1 Clock input

The high frequency internal clocks are derived from the clock frequency supplied by the user. The user supplies the clock frequency for input to the PLL's via the **ClockIn** input. The nominal frequency of this clock is 5 MHz.

A number of STC104s may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. **ClockIn** must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. **ClockIn** must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

The timing requirements for **ClockIn** are given in section 11.1.

9.2 Phase locked loop decoupling

The internally derived power supply for internal clocks requires an external low leakage, low inductance 2 µF capacitor to be connected between **CapPlus** and **CapMinus**. A surface mounted ceramic capacitor should be used. In order to keep stray inductances low, the total PCB track length should be less than 20 mm, thus the capacitor should be no more than 10 mm from the chip. The connections must not touch power supplies or other noise sources.

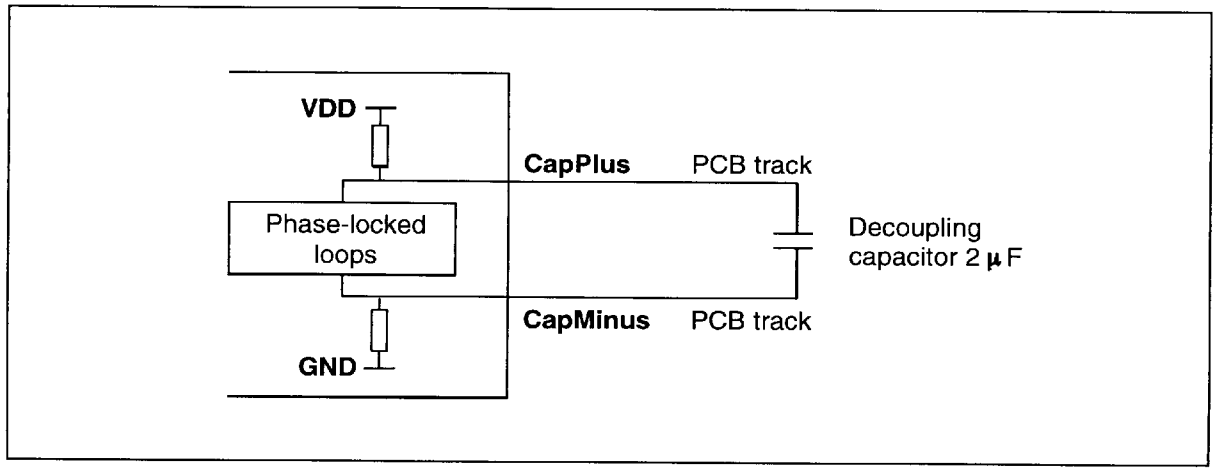


Figure 9.1 Recommended PLL decoupling

Note: CapPlus and **CapMinus** lie between **GND** and **VDD**, and **CapPlus** is greater than **CapMinus**. However, **CapPlus** and **CapMinus** are not at a guaranteed voltage level. Therefore **CapPlus** and **CapMinus** must be connected only to a decoupling capacitor and the decoupling capacitor must not be shared between devices.

9.3 Speed selection

The internal clock rate is variable in discrete steps. The clock rate at which the STC104 runs at is determined by the logic levels applied on the speed select lines **CoreSpeedSelect0-1** as detailed in table 9.1.

CoreSpeedSelect1	CoreSpeedSelect0	Core clock speed (MHz)	Core cycle time (ns)
0	0	30	33.3
0	1	40	25.0
1	0	50	20.0
1	1	Reserved	

Note: Inclusion of a speed selection in this table does not imply immediate availability.

Table 9.1 Core speed selection

10 Electrical specifications

Inputs and outputs are TTL compatible.

10.1 Absolute maximum ratings

Symbol	Parameter	Min	Max	Units	Notes
VDD	DC supply voltage	0	7.0	V	1,2,3,4,5
VI, VO	Voltage on input and output pins	−0.5	VDD+0.5	V	1,3,4,5
II	Input current		10	µA	6
tOSC	Output short circuit time (one pin)		1	s	4
TS	Storage temperature	−65	150	°C	4

Table 10.1 Absolute maximum ratings

Notes

1 All voltages are with respect to **GND**.

2 Power is supplied to the device via the **VDD** and **GND** pins. Several of each are provided to minimize inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between **VDD** and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

3 Input voltages must not exceed specification with respect to **VDD** and **GND**, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

4 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

5 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as **VDD** or **GND**.

6 The input current applies to any input or output pin and applies when the voltage on the pin is between **GND** and **VDD**.

10.2 Operating conditions

Symbol	Parameter	Min	Max	Units	Notes
VDD	DC supply voltage	4.75	5.25	V	1
VI, VO	Input or output voltage	0	VDD	V	1,2
TA	Operating temperature range	0	TAMAX	°C	3

Table 10.2 Operating conditions

Notes

1 All voltages are with respect to **GND**.

2 Excursions beyond the supplies are permitted but not recommended.

3 For details of TAMAX, refer to section 13.3 on thermal data.

10.3 DC characteristics

Symbol	Parameter		Min	Max	Units	Notes
VIH	High level input voltage		2.0	VDD+0.5	V	1,2,3
VIL	Low level input voltage		–0.5	0.8	V	1,2,3
II	Input current	@ GND<VI<VDD		10	µA	1,2
VOH	Output high voltage	@ IOH=2mA	VDD–2		V	1,2,3,4
VOL	Output low voltage	@ IOL=4mA		0.4	V	1,2,3,4
IOZ	Tristate output current	@ GND<V0<VDD		10	µA	1,2,3
CIN	Input capacitance	@ f=1MHz		7	pF	3
COZ	Output capacitance	@ f=1MHz		10	pF	3

Table 10.3 DC characteristics

Notes

1 All voltages are with respect to **GND**.

2 Parameters for STC104 measured at 4.75V<**VDD**<5.25V and 0°C<**TA**<TAMAX. Input clock frequency = 5 MHz.

3 Characterized on a sample of devices, not tested.

4 For link outputs, IOH=1mA, IOL=1mA.

10.4 Power rating

Maximum power dissipation for the STC104 with 32 links operating at 100 Mbits/s is 10W.

11 Timing specifications

11.1 ClockIn timings

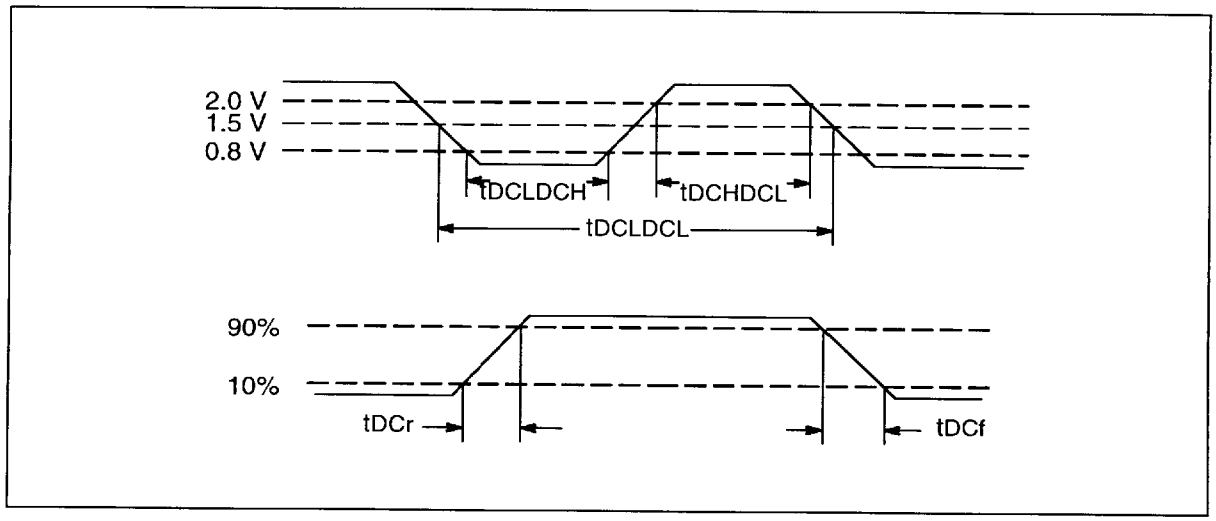


Figure 11.1 **ClockIn** timing

Symbol	Parameter	Min	Nom	Max	Units	Notes
tDCLDCH	**ClockIn** pulse width low	40			ns	
tDCHDCL	**ClockIn** pulse width high	40			ns	
tDCLDCL	**ClockIn** period		200		ns	1, 2
tDCr	**ClockIn** rise time			10	ns	3
tDCf	**ClockIn** fall time			8	ns	3

Table 11.1 **ClockIn** timings

Notes

1 Measured between corresponding points on consecutive falling edges.

2 This value allows the use of 200 ppm crystal oscillators for two devices connected together by a link.

3 Clock transitions must be monotonic within the range V_{IH} to V_{IL} (refer to Electrical specifications chapter 10).

11.2 DS-Link timings

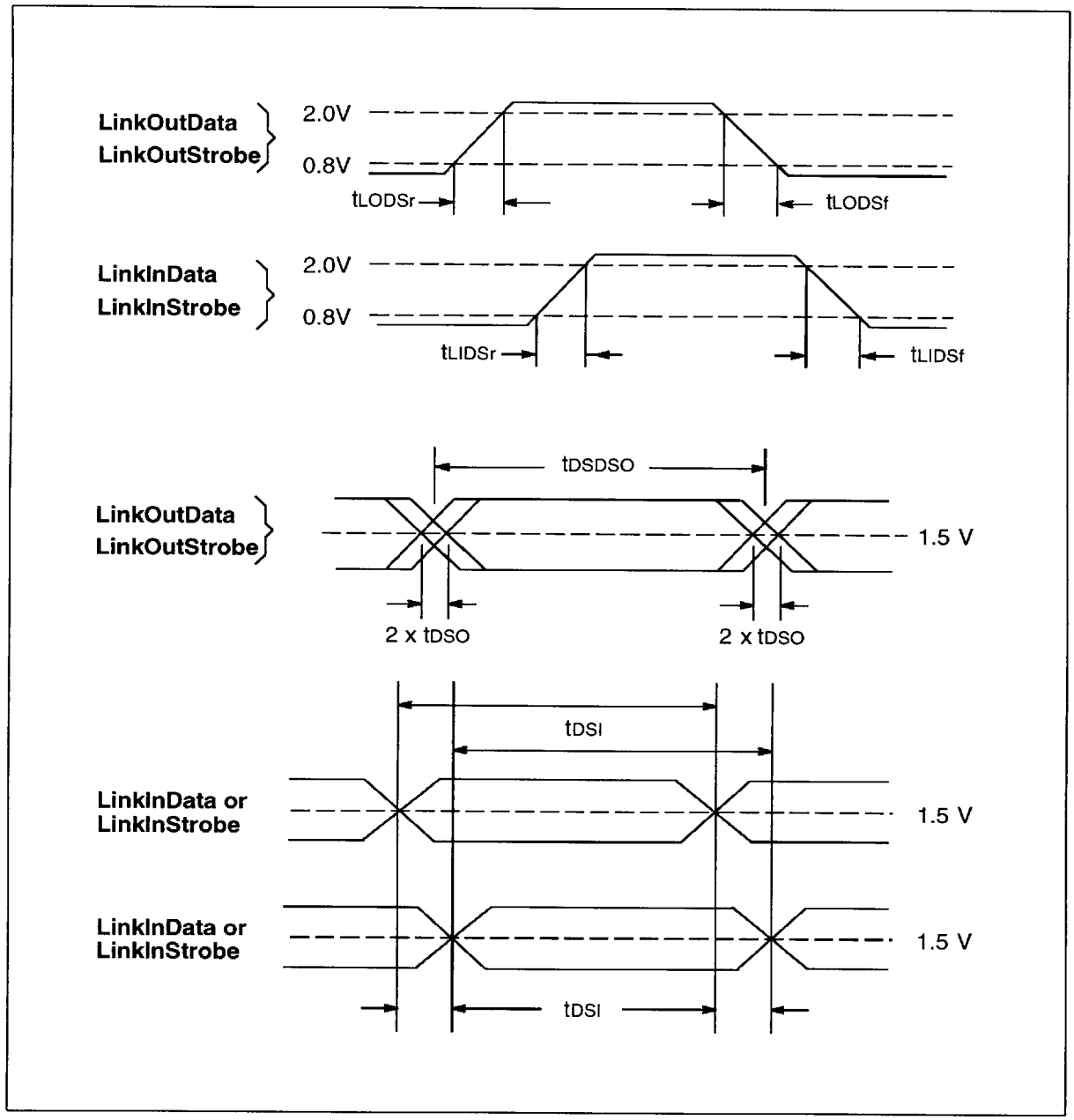


Figure 11.2 DS-Link timing

Symbol	Parameter	Min	Nom	Max	Units	Notes
CLIZ	LinkIn capacitance		7		pF	1
tDSDSI	Sustainable averaged input bit period	9		110	ns	
tDSDSO	Output bit period	10		100	ns	2
tDSI	Data/strobe input edge minimum separation		2.5		ns	3,4,5
tDSO	Data/strobe output skew			1	ns	6
tLIDSf	LinkIn fall time (2.0–0.8V)			100	ns	7
tLIDSr	LinkIn rise time (0.8–2.0V)			100	ns	7
tLODSf	LinkOut fall time (2.0–0.8V)			7	ns	8
tLODSr	LinkOut rise time (0.8–2.0V)			7	ns	8
ROH	Output impedance (output driving high)	110		247	Ω	9
ROL	Output impedance (output driving low)	63		104	Ω	9

Table 11.2 DS-Link timings

Notes

1 Sampled, not 100% tested.

2 tDSDSO represents the minimum and maximum programmable bit periods.

3 tDSI is the shortest permissible spacing of 2 consecutive edges on the Data and Strobe wires (1 edge of either sense on each wire). If arriving Data and Strobe edges are skewed to the extent that this parameter is exceeded then the order of the edges becomes ambiguous and a parity error is likely to result.

4 Edge separation includes consecutive edges of a data input or a strobe input.

5 Based on a slew rate of 1.5 V/s, monotonic across the transition region. For other values of slew rate, use the following formula:
1.0 + (k * slew rate) where k=1.0

6 tDSO is the maximum discrepancy between the time when a DS Output edge (either sense) starts a transition and the theoretical ideal (i.e. all consecutive DS edges spaced by tDSDSO).

7 Edges must be monotonic, hence faster edges are recommended unless the link is to be used in a noise free environment.

8 Measurement based on a loading of 25pF.

9 The link output drivers have been optimized for driving a 100Ω transmission line.

11.2.1 Link Input and Output relative skews

For the skew parameters to be valid for a wide range of operating speeds (10 – 100 Mbits/s) certain parameters must be made relative to edge rates, as the interaction of

edge rates and logic threshold have significant impact on the skew. Note that skew is measured relative to the edges crossing a nominal 1.5V logic threshold.

t_{DSI} = Fixed skew + k * (the larger of t_{LIDSr} and t_{LIDSf})

Where *Fixed Skew* is related to the worst case DSDecoder input skew rejection and internal input path mismatch, and *k* is found by characterization and related to minimum variation in input threshold and input pad propagation delay.

t_{DSO} = Fixed skew

Where *Fixed Skew* is related to the worst case Link Output PLL jitter and internal output path mismatch.

11.2.2 Skew budget

The concept here is that in order to eliminate the risk of DSLink parity errors due to the relative skew between Data and Strobe inputs a system designer must ensure that the sum of $2t_{DSO}$ and the relative skew between Data and Strobe induced by all system interconnect and buffering must be less than $t_{DSDSO} - t_{DSI}$.

Note that an edge rate dependent calculation must be performed for external buffers with variable thresholds in order to calculate worst case $t_{EXTSkew}$ for both Data and Strobe.

i.e. $2t_{DSO} + t_{EXTSkew} < t_{DSDSO} - t_{DSI}$

The parameter t_{DSO} on the left hand side of the expression is multiplied by two to allow for the worst case situation of Data and Strobe undergoing maximum skew in opposite directions.

12 Pin designations

This section details the function of the pins on the STC104. Pinout details are given in chapter 13.

Supplies

Pin	In/Out	Function
VDD		Power supply
GND		Return

Table 12.1 STC104 supplies

Clocking

Pin	In/Out	Function
ClockIn	in	5 MHz input clock
CapPlus, CapMinus		External capacitor for internal clock power supply
CoreSpeedSelect0-1	in	Speed selectors

Table 12.2 STC104 clocks

Control system

Pin	In/Out	Function
Reset	in	System reset
CLinkInData0-1	in	Control link input data channel
CLinkInStrobe0-1	in	Control link input strobe
CLinkOutData0-1	out	Control link output data channel
CLinkOutStrobe0-1	out	Control link output strobe

Table 12.3 STC104 control system

Communications

Pin	In/Out	Function
LinkInData0-31	in	Link input data channels
LinkInStrobe0-31	in	Link input strobes
LinkOutData0-31	out	Link output data channels
LinkOutStrobe0-31	out	Link output strobes

Table 12.4 STC104 communications links

Test Access Port (TAP)

Pin	In/Out	Function
TDI	in	Test data input
TDO	out	Test data output
TMS	in	Test mode select
TCK	in	Test clock
notTRST	in	Test logic reset

Table 12.5 STC104 TAP pins

Miscellaneous

Pin	In/Out	Function
HoldToGND		Must be connected to **GND**
HoldToVDD		Must be connected to **VDD**
DoNotWire		Must not be wired
NotUsedForRevA		This pin is not used on current revisions of the STC104. It must be connected to **GND**.

Table 12.6 STC104 miscellaneous pins

13 Package specifications

The STC104 is available in a 208 pin formed CLCC cavity-down package.

13.1 STC104 208 pin CLCC cavity-down package pinout

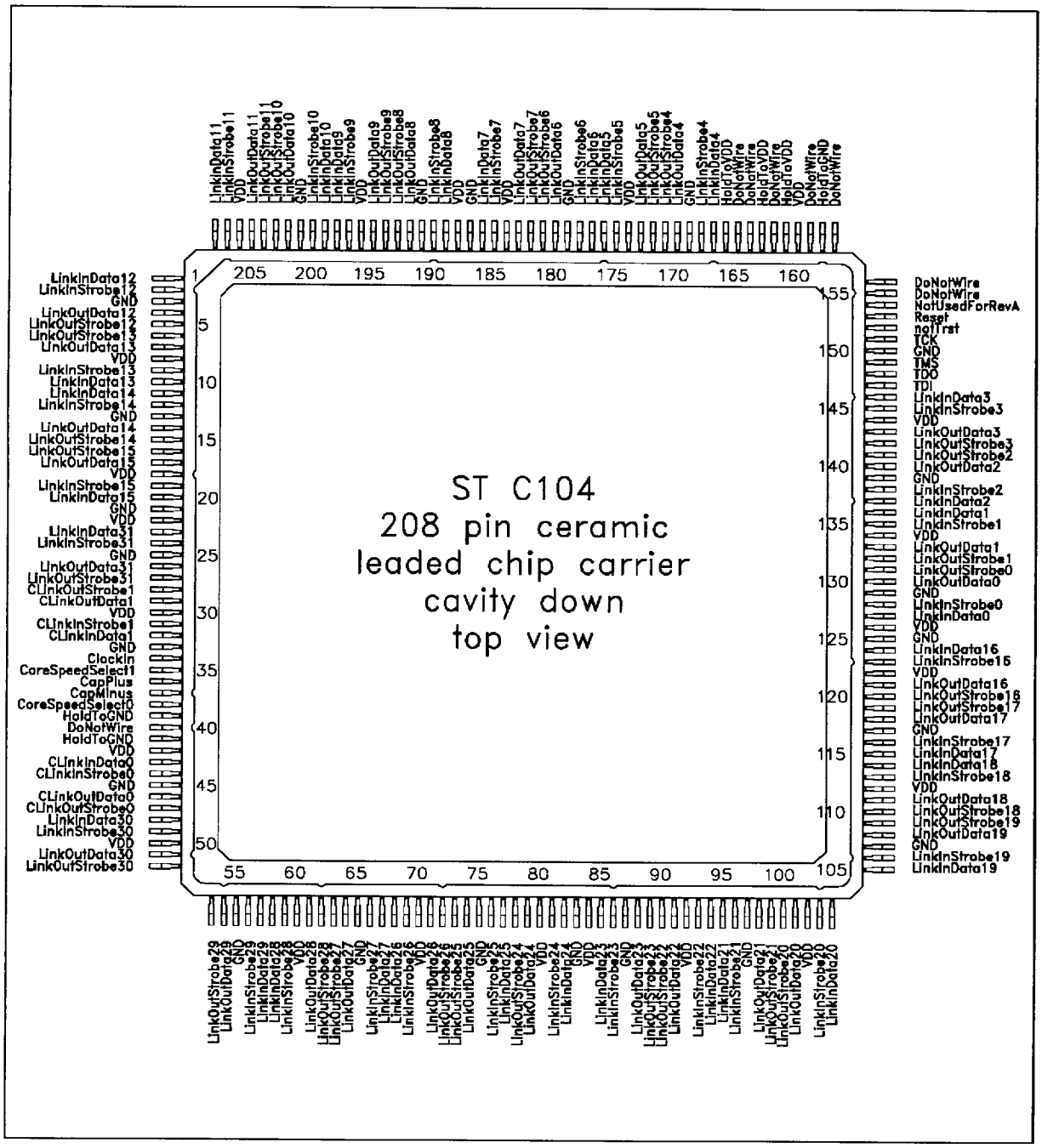


Figure 13.1 STC104 208 pin CLCC cavity-down package pinout

13.2 STC104 208 pin CLCC cavity-down package dimensions

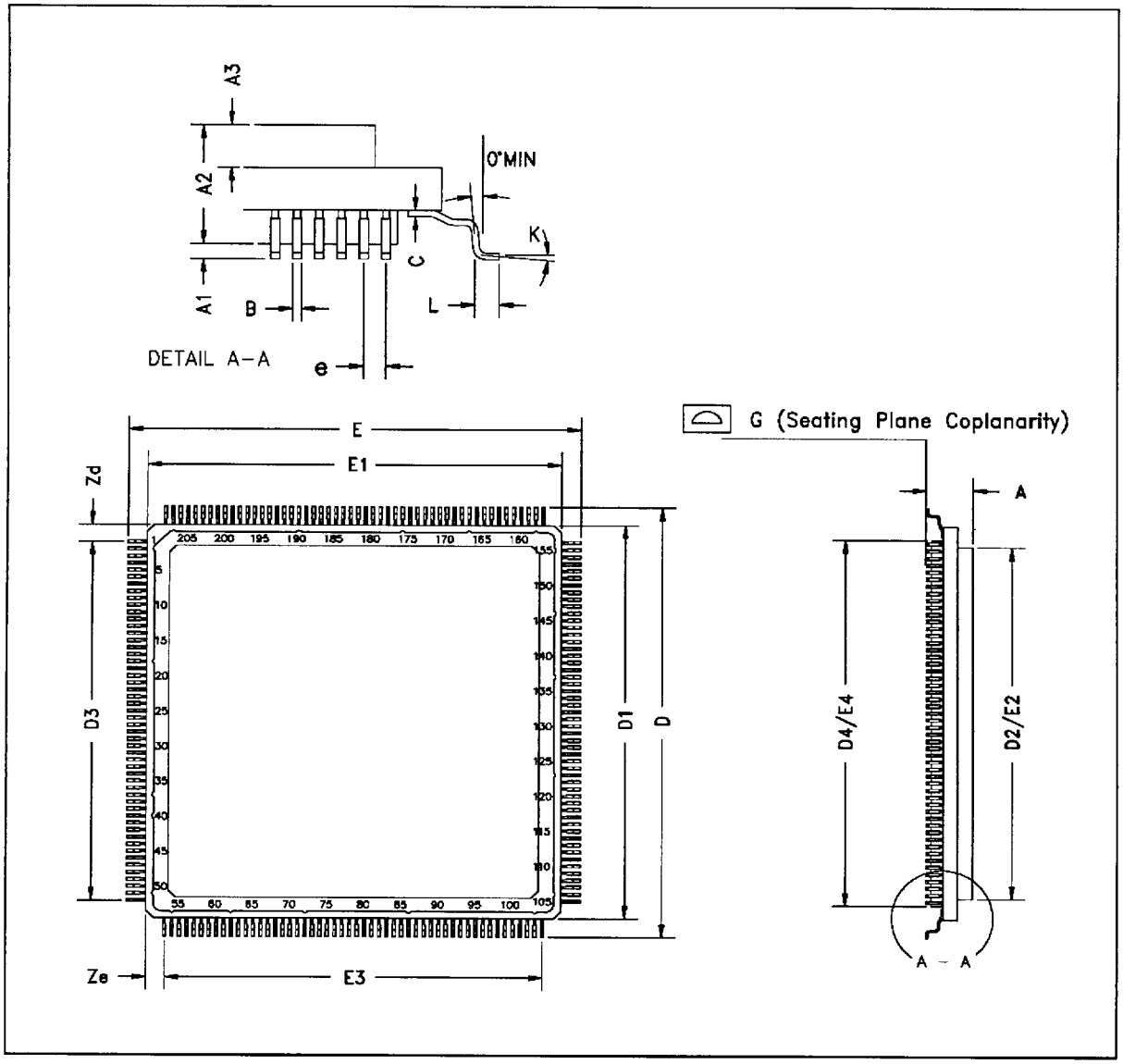


Figure 13.2 STC104 208 pin CLCC cavity-down package dimensions

REF.	CONTROL DIM. mm			ALTERNATIVE DIM. INCHES			NOTES
	MIN	NOM	MAX	MIN	NOM	MAX	
A	–	–	3.500	–	–	0.138	
A1	0.25	–	–	0.010	–	–	
A2	2.33	2.63	2.93	0.092	0.104	0.115	
A3	–	–	1.00	–	–	0.039	
B	0.180	–	0.280	0.007	–	0.011	
C	0.100	–	0.200	0.004	–	0.008	
D	30.300	30.600	30.900	1.193	1.205	1.217	
D1	27.700	28.000	28.300	1.091	1.102	1.114	
D2	24.750	–	22.250	0.974	–	0.994	
D3	–	25.500	–	–	1.004	–	REF
D4	25.150	–	26.250	0.990	–	1.033	
E	30.300	30.600	30.900	1.193	1.205	1.217	
E1	27.700	28.000	28.300	1.091	1.102	1.114	
E2	24.750	–	22.250	0.974	–	0.994	
E3	–	25.500	–	–	1.004	–	REF
E4	25.150	–	26.250	0.990	–	1.033	
e	–	0.500	–	–	0.020	–	BSC
G	–	–	0.100	–	–	0.004	
K	0°	–	7°	0°	–	7°	
L	0.300	0.500	0.700	0.012	0.020	0.028	
Zd	–	1.250	–	–	0.049	–	REF
Ze	–	1.250	–	–	0.049	–	REF

Notes:

1 Lead finish to be gold plated.

2 Maximum lead displacement from the notional center line will be no greater than ±0.1mm.

Table 13.1 STC104 208 pin CLCC cavity-down package dimensions

13.3 STC104 208 pin CLCC cavity-down package thermal data

The STC104 is tested to a maximum silicon junction temperature of 100°C. For operation within the given specifications, the case temperature should not exceed 95°C.

Given a maximum operating junction temperature of 100°C, the following maximum power conditions apply:

Conditions	Maximum power (Watts)
Still air at 305C	3.41
Case held at 955C	15.0

For actual maximum power dissipation see section 10.4.

For temperatures above 100°C the operation of the device cannot be guaranteed and reliability may be impaired.

For further information on reliability refer to the SGS-THOMSON Microelectronics Quality and Reliability Program.

External thermal management must be used to ensure optimum performance and reliability.

14 Ordering information

Device	Package
STC1041-F10S	208 pin CLCC cavity-down package

For further information contact your local SGS-THOMSON sales office.