VLSI Design
1995, Vol. 2, No. 4, pp. 305–314
Reprints available directly from the publisher
Photocopying permitted by license only

The STC104 Packet Routing Chip

PETER W. THOMPSON and JULIAN D. LEWIS
INMOS Limited, Bristol, England

High-performance parallel systems demand a high-performance interconnect so that their component parts can exchange data and synchronise efficiently. The interconnect must be cheap, and must also scale well in both performance and cost relative to the system size. In this paper we describe the rationale, architecture and operation of the STC104, the first commercially available, general-purpose interconnect chip. The serial protocols used by the device are described, followed by an overview of the microarchitecture. The operation of the fundamental block is outlined, including the response to error conditions. Chip-wide design issues and design methodology are discussed, and finally various aspects of performance are calculated.

Key Words: *Packet routing; Wormhole routing; Interconnect; Serial interconnect; Custom VLSI; Parallel processing*

INTRODUCTION

This paper describes the STC104, a programmable single-chip VLSI device which can interconnect up to 32 other devices, including other instances of itself. In the first section the principles of operation for systems using the device are outlined. The following section describes the interconnection protocol used. Subsequent sections are devoted to the micro-architecture of the chip and constitute the core of the paper. Thereafter general design issues are discussed, and the final section deals with various aspects of performance.

A number of designs for packet routing chips have been published before, for example [1, 7, 11]. However none have been able to interconnect so many devices as the STC104, and nearly all have been designed for a specific network topology such as a torus or hypercube. All previous routing chips have either been research projects or else proprietary. The STC104 is the first general-purpose production routing chip to become available. It is also the first chip to employ interval routing [8], and the first wormhole routing chip to completely separate the flow-control protocol from the packet size, thereby making the size of routeable packets completely variable.

CONCURRENT INTERCONNECTION NETWORKS

A parallel system can provide high performance by allowing many identical components of moderate performance to work concurrently. In just the same way a parallel systems interconnect can provide a high aggregate bandwidth by allowing many separate connections with moderate individual performance to operate simultaneously. Maximum simplicity, modularity and fault-tolerance can be achieved by making each connection point-to-point, and serial [15].

Routing

To connect many devices together using point-to-point connections, it is not feasible to provide a direct, physical path between every pair, even when the connections use a minimum number of wires. Thus data must be able to travel from one device to another via one or more intermediate nodes, i.e., it must be through-routed. The most effective way to achieve this is to make the data *self-routing* so that it contains within it the information which deter-

mines which way it should go. A piece of data together with a header containing its associated routing information is called a *packet*. The STC104 is a self-contained dynamic packet-routing chip, which uses wormhole routing [12] to minimise latency. In this technique, only the header of the packet is initially read in by the routing switch. The routing decision is taken, the header is output, and the rest of the packet is sent directly from the input to the output without being stored in the switch, which means that a packet can be passing through several switches at the same time. Thus this method can be thought of as a form of dynamic circuit switching, in which the header of the packet, in passing through a sequence of switches, creates a temporary circuit (the 'wormhole') through which the data flows. As the tail of the packet is pulled through, the circuit vanishes. As well as minimising latency, wormhole routing has the further advantage that it is independent of the packet length, thereby making the chip more general purpose.

To maximise the benefit of wormhole routing, a routing switch must be able to take the routing decision very quickly, and in order for a large fan-out single-chip device to be feasible, it must do so with minimal hardware. The STC104 achieves this by using *interval routing* [8], in which the header of each arriving packet is compared with a set of ranges of values ('intervals'), one for each output of the chip. The packet is routed to the output in whose range the packet header falls. Interval routing provides a consistent 'address space', in which the same header corresponds to the same destination device regardless of the source, but places some restrictions on the relationship between packet headers and the interconnected devices, so that this mapping is no longer completely arbitrary. The STC104 extends interval routing to allow separately constructed and programmed networks to be connected, and hierarchical networks to be constructed. Each output which is connected to another network or a different level in a hierarchy can be programmed to discard the routing header as a packet is routed through it. Since the packet length is arbitrary, and the routing decision depends only on the header at the beginning of the packet, a second header can be included immediately after the first which will then become the packet header when the first header is deleted. The packet can pass through further routing switches using the second header [10].

Grouped-adaptive and Universal Routing

The STC104 supports locally adaptive routing by allowing a free choice between a programmed group (similar to a 'hunt group') of output links. Provided every link of the group is connected to the same device or to an entirely equivalent device, the choice of which link to use can be made on the basis of which link is available first. This maximises performance by ensuring that there are not several packets waiting to use one link when another equivalent link is idle. This can be used to exploit the bandwidth of 'bundles' of links for interconnection networks whose underlying topology does not require the full fan-out of the STC104, and for efficient load-balancing in multi-stage indirect networks [6]. It also enables a degree of automatic fault-tolerance to be provided [13].

The STC104 also supports a method, called two-phase, or universal, routing [14], for preventing interconnect hot-spots in large networks. Every packet entering a two-phase routing network is routed first to a randomly selected node, and from there to its original destination. By spreading the load across the interconnect this maximises the number of links which can transport it, thereby increasing bandwidth and reducing latency under high load conditions, at the extense of peak bandwidth and minimum latency under low load. To implement this algorithm the STC104 can be programmed so that some of its inputs (those along which data enters the interconnect) add a randomly generated header to the front of each packet. This header is generated in a range so that it corresponds to a label of one of the other STC104s in the network. The packet is then routed to that STC104 in the normal way, at which point the generated header is recognised and stripped off before the packet is routed further. The packet's original header is now exposed, and so it is routed to its true destination. This technique performs both phases of the algorithm with the same set of routing devices, requiring only the point-to-point links to be in separate sets for the two phases [10].

THE SERIAL INTERCONNECT PROTOCOL

Each STC104 communicates with other devices using a simple layered asynchronous serial protocol. At the lowest level, a serial bit-stream together with its clock is encoded onto two wires, one of which carries the data value, and the other (unconventionally called a 'strobe') changes state only when the data value does not. The clock can thus be recovered very simply by XORing the two signals together. This encoding has the advantages that it uses only 100 MHz signals to carry 200 MBaud data and a

clock, and has a full bit-time of skew tolerance. It is very similar to the self-timed signalling convention described in [9]. The output drivers of the STC104 are adequate for connections of up to 1m through impedance-controlled PCB tracks [5]. Distances of up to 10m can be covered by using simple differential buffers [10]. The STC104 regenerates and resynchronises its signals so that there is no loss of signal quality however large the network, and generates its own clocks so that high-frequency clock distribution is not required.

The STC104 uses the serial bit-stream to carry a sequence of 'tokens', each of which either contains a byte of data or performs a special control function. Each token contains a flag bit distinguishing data from control tokens, and a parity check bit. A byte of data thus requires ten bits to transmit, so a raw bit-rate of 200 MBaud provides a maximum data rate of 20 MBytes/s. Control tokens contain two bits to distinguish them and so are four bits long in total.

The STC104 has a total of 34 independent interfaces[1], each of which consists of two input and two output pins. Each pair of pins carries the bit and token level protocols described above. When one of these interfaces is connected to a corresponding interface on another device, the result is called a DS-Link. Logic at each end multiplexes control tokens into the data stream to manage the bi-directional flow of data so that none is lost. Each 4-bit control token transmitted allows 8 data tokens to be sent in the opposite direction, so the bandwidth overhead is approximately 5%. This flow-control protocol completely removes any possibility of internal buffer overflow. Each end of a DS-Link checks for parity errors in the received tokens and also for a break in transmission, which is signalled as a failure of the connection [5]. In electrically clean environments such as on a PCB or through a backplane the error rate is extremely low[2], so the detection of an error generally indicates a gross failure, such as a connector pulled out.

Each incoming token stream is interpreted by the STC104 as a sequence of packets. The end of each packet is indicated by one of two control tokens, called *terminators*. The start of each packet is simply taken to be the first data received following the end of the previous packet. The first one or two bytes of each packet (the choice being a programmable parameter) are interpreted as a routing header, which causes a circuit to be opened across the switch. Any further bytes simply follow through this circuit, which is closed after the packet terminator passes through it. This simple packet format imposes a very low overhead, and permits very short packets to be formed efficiently. The shortest possible packet is 14 bits (a one byte header followed immediately by a terminator), which is also the minimum overhead. Note that these protocols are being developed into an IEEE standard, P1355 [16].

MICRO-ARCHITECTURE

Classification of Architecture

The STC104 is a special purpose MIMD processor [2] with distributed control. Data flows through a 'star' network from 32 input ports to a single routing point, the crossbar switch, and then out to 32 output ports. Each DS-Link implements one input and one output port. A single DS-Link together with logic to provide extra buffering and implement the routing function forms a unit referred to as a 'linkslice', providing one complete input path and one complete output path. The chip contains 32 identical linkslices, one crossbar switch, and several miscellaneous units providing global services to the main blocks. The overall architecture is illustrated in Figure 1.

Each linkslice has exactly the same 'operating program', (in this case a set of interacting concurrent state machines), but they do not operate in lockstep as in a classical SIMD machine. The state of each linkslice at any point in time is a function of the configuration parameters and the data values that have flowed through it. Since the data flow will in general be different between linkslices, their state at any point in time will also be different, as in an MIMD machine.

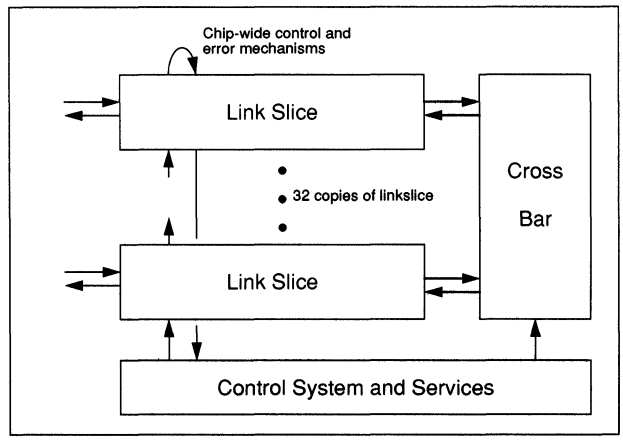

FIGURE 1 Top-Level Decomposition of the STC104

[1]Two of which are reserved for programming and error monitoring.

[2]BERs less than 10^{-14} in tests.

Global services, including initialisation and reset, are provided by one block, the 'control unit', which has a series of dedicated daisy-chain connections to all the routing path blocks, one per function. See the section "Control System and Programming" for a further discussion.

Pipelining and Parallelism

The STC104 is a highly parallel machine. It consists of an array of 32 linkslices operating concurrently, each containing a pipeline of functional blocks. Each functional block has its own locus of control and operates concurrently with all the others. All inter-block data transfers are performed with a handshaking protocol which guarantees token-level flow control as packets pass through a set of blocks, each implementing part of the overall function. Just as packets may be distributed simultaneously through multiple routers, they will in general also be distributed through the functional blocks of an individual router.

Many functional blocks are themselves internally sub-divided into multiple communicating state machines operating in parallel. The distinction is that below the level designated 'functional block', communication is special-purpose, will in general be wide (e.g. connections between a datapath and its control logic), and may not be fully handshaken, in particular because synchronised interfaces imply a minimum latency cost of 1 cycle, which may be unacceptable.

Including state machines in the 34 DS-Links, the crossbar, the control unit and various minor blocks, the STC104 contains in excess of 500 parallel state machines, containing over 1K bits of state vector. Thus the whole chip has in excess of $2^{1024} \simeq 10^{308}$ states, ignoring the values of the data being transferred.

Data Transfers

Each DS-Link module provides a pair of 2-wire serial connections, one for each direction, which are connected off-chip, and a pair of 11-bit data buses connected into the core of the chip. The 11 bits constitute 8 data bits, 1 flag bit to distinguish termination tokens from data tokens, and 2 handshake wires to implement inter-block flow-control. All data transfers within the chip are token-wide. The choice of token-wide internal communication is based on an analysis of the most parallel (and therefore fastest) implementation that will fit in the available area. It is a compromise between the performance requirement favouring wider and hence faster buses, and the cost constraint favouring narrower ones. There is some advantage in making the core of the chip faster than the DS-Link, but even if area were available for 16 or 32 bit wide data buses, they would provide diminishingly small returns for severely increasing costs, since the serial DS-Link will inevitably be the bandwidth limiting factor. The choice of an 11-bit bus is also the simplest way to interface to a DS-Link, which is fundamentally a token engine.

Crossbar Switch

The heart of the STC104 is a 32-way full crossbar switch. This has 32 11-bit input ports, and 32 11-bit output ports. Up to 32 point-to-point connections may be established at any time between any pairs of input ports and output ports. The crossbar is capable of sustaining transfer rates of 32 bytes per cycle at 50 MHz, or 1.6 GBytes/sec. Control within the crossbar is distributed, and it is capable of making or breaking up to 32 connections in parallel in one cycle.

The crossbar is passed a value from the linkslice identifying the required output port at the start of each packet. When the output port is free, the crossbar makes a connection between the corresponding pair of busses, which is maintained until a termination token marking the end of the packet is received and passed on. While the connection is made, not only the data wires but also the handshake signals of the 11-bit bus are connected, so that flow-control is preserved through the crossbar, and hence across the entire chip.

The crossbar performs arbitration between linkslices which simultaneously request the same output, connecting one and stalling the others until the first packet has been transferred. The arbitration is performed round-robin, which ensures that every waiting input will be serviced after a bounded time. The crossbar also implements the grouping mechanism, performing another level of arbitration between outputs in a group should more than one of them be available to service a requesting input.

Buffering

The STC104 has a total of 70 tokens of buffering on each of 32 paths, or 2240 tokens in total. The single path buffering is distributed as follows:

input side DS-Link	20
input linkslice FIFO	20
token queue	3
crossbar	4
output linkslice FIFO	20
output side DS-Link	3
TOTAL	70

Fast Packet Transfer

The maximum bandwidth of the DS-Link at 200 MBaud is 25 MTokens/sec, whereas the core bandwidth is 50 MTokens/sec (one token per cycle). The advantage of making the core twice as fast as the DS-Link is that there are situations when a packet is held up waiting for arbitration for an output path, which delays the packet, and also those behind it in the input path. Thus a packet contained in the Input FIFO whose output is free may nevertheless have to wait while the packet ahead of it is routed. When this is complete, however, the waiting packet can be transferred from the Input FIFO to the empty Output FIFO of its chosen output at a rate of one token per cycle, thereby making up some of the time lost.

OPERATION OF A LINKSLICE

Each 'linkslice' implements two halves of a routing path: from the DS-Link to the crossbar (input side), and from the crossbar to the DS-Link (output side). The linkslice consists of four functional blocks on the input side (the input FIFO, the random header generator, the interval comparator, and the token queue) and two on the output side (the error handler and the output FIFO), as illustrated in Figure 2.

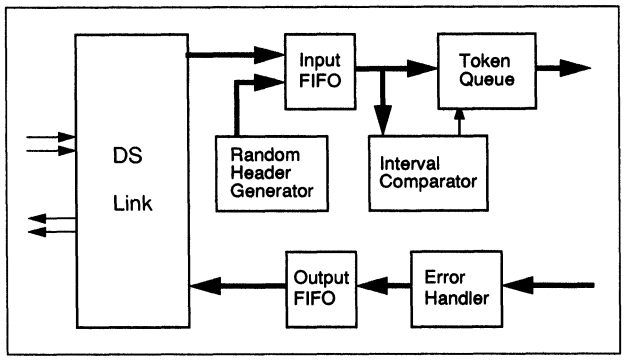

FIGURE 2 STC104 Linkslice Decomposition

Input Side

The data on the input side passes through two blocks, the input FIFO and the token queue, operating as a 2-stage pipeline. Two further blocks, the random header generator and interval comparator, are not in the main data-flow but supply extra items of information which are multiplexed into the data stream by the input FIFO and token queue respectively.

Input FIFO: Incoming tokens pass through the input FIFO, which imposes only a single cycle of latency when the device is free running. The FIFO will pass tokens on, in the order they were received, as soon as the token queue is ready to receive them. As in all inter-block data transfers this is handshaken as described in the next section to ensure that tokens will not be lost under any load conditions. When the outgoing path is stalled, e.g. because two packets have both requested the same output port, the FIFO continues to accept tokens from the DS-Link and stores them internally up to its capacity of 20 tokens. The FIFO is capable of inputting and outputting data at a rate of one token per cycle.

When a programmable flag in the linkslice is set, the input FIFO will multiplex a randomly generated header from the random header generator onto the front of each packet received from the DS-Link, to implement the first part of the Universal Routing algorithm.

Random Header Generator: The random header generator supports the two-phase Universal Routing algorithm. It provides either one or two byte headers (determined by the header_length flag) in the window $base \leq header < base + range$, where *base* and *range* are arbitrary 16-bit unsigned integers which are programmed into it.

Interval Comparator: The implementation of the interval routing scheme is divided between two blocks: the token queue and the interval comparator. The interval comparator is a relatively large datapath which provides a very fast parallel implementation of interval lookup. It compares each incoming header against the programmed routing table to produce the output link number and two flags called 'discard' and 'invalid', in a single cycle. The 'invalid' flag is used for system security purposes, to ensure that only packets with a valid destination are routed. The 'discard' flag supports the Universal Routing algorithm, by causing a header added during the first phase of Universal Routing by another STC104 to be discarded. Selecting an interval with

the discard flag set is the boundary between the first and second phases of the algorithm.

Token Queue: The token queue is responsible for interpreting the data produced by the interval comparator, and for detecting packet structure errors in the token stream. Each packet is either passed to the crossbar, proceeded by the identity of the required output port, or else consumed and an error reported. The token queue uses the following algorithm for each packet:

if there are insufficient data bytes in the packet to form a header *then* consume the whole packet and report a 'short packet' error

else if the header selects an interval whose 'invalid' flag is set *then* consume the whole packet and report an 'invalid packet' error

else if the header selects an interval whose 'discard' flag is set *then* delete the header only and start this routing algorithm again, using the next header_length bytes of the packet as the header

else supply a signal to the crossbar encoding the requested output path and then copy through the packet until the terminator token

endif

When copying the body of a packet through to the crossbar the token queue acts as a small FIFO buffer.

Output Side

There are notionally two blocks on the output side (the error handler and the output FIFO), which are optimised into a single layout box in the silicon implementation.

Error Handler: In normal operation the error handler simply copies data through, acting as a zero-cycle latency buffer. If the DS-Link of the linkslice reports an error then the error handler is activated to return the linkslice to a well-defined state. If the error handler is part-way through copying a packet from the crossbar to the output FIFO, then it consumes the remainder of the packet up to and including the termination token marking the end of the packet. Its behaviour after this point is dependent on a programmable flag called discard_on_error.

If discard_on_error is false, then the error handler refuses to accept more data from the crossbar until the DS-Link error condition clears, so the linkslice appears to the crossbar to be permanently busy. By stopping cleanly on a packet boundary, the error handler ensures that the path through the crossbar from the input linkslice will be closed down, and so the input path is free to make connections to other output paths. If the link is a member of a group, then further packets sent to that group will simply be directed by the crossbar to other members of the group which are still operating. In this way the STC104 supports a degree of automatic fault-tolerance [13].

If discard_on_error is true then the error handler will consume an indefinite number of whole packets from the crossbar until the DS-Link error condition clears. This feature guarantees the interconnect as a whole will not stall even if parts of it fail.

Output FIFO: The output FIFO is a 20 token buffer and operates in the same manner as the input FIFO, except that it also implements the header deletion function.

If the programmable header_delete flat is set for the linkslice, then the first one or two data tokens of each packet, i.e. its header, are removed before the remainder of the packet is passed through. The size of the header is set by another programmable flag (header_length). If, after removing the header, there are no data tokens remaining in the packet then the termination token marking the end of the packet is deleted (since there are no data tokens, this is all that remains) and a 'null packet' error is reported.

CHIP-WIDE DESIGN ISSUES

Implementation Constraints and Technology

A VLSI implementation of any complex function is constrained by the necessity to fit all the logic on a single die, and the restrictions on that die imposed by the fabrication process and packaging to be used. The package in particular imposes a direct upper limit on the power that can be dissipated, which in CMOS technology limits the number of transistor switching events per unit time. This is a function of the mean number of transistors switching per cycle (which is a measure of the effective parallelism of the design) and the number of cycles per second (i.e. the operating frequency). The STC104 router is inherently very parallel, which yields very high performance, but also leads to a worst-case power dissipation of about 5W, requiring a ceramic type package.

The STC104 is fabricated in the 'HCMOS4' process of SGS-Thomson which uses 3 layers of metal for routing, stacked contacts between metal layers, and has a minimum feature size of 1.0 microns.

Complex circuits are generally interconnect limited in current VLSI technology (more than by power or raw transistor area), and so stacked contacts (vertical connections between horizontal layers of metal used to carry signals) are particularly valuable in reducing the area required to implement complex functions. Multiple connections between routing layers may be stacked vertically so that the area consumed in an intermediate layer by making a contact between a lower layer and a higher one is minimised. Allowing efficient inter-layer connection is a vital factor in achieving dense layout, for example in the STC104, which has a die-size of 204.6 mm^2 and contains 1.875 million transistors.

Design Methodology

The STC104 was specified in a natural language document. Architectural modeling of the micro-architecture of one device, and of a network of many devices, was done in the occam language [4] running on a transputer network. The micro-architectural decomposition was formally checked using the CSP process algebra [3] via a commercial model checker.

The STC104 is implemented in the Cadence OPUS design environment using C for top-level functional modelling, VHDL for behavioural modelling, and Eldo for circuit modelling. The silicon area is dominated by full-custom logic, but commercial logic synthesis and place-and-route tools were used for some non-critical control logic and finite state machines.

The level of customisation was carefully managed throughout the project to minimise design time while meeting die-size constraints. A semi-custom approach (e.g. use of logic synthesis) reduces design time, but is inefficient. A full-custom approach achieves high efficiency at the cost of more design time. The goal is to pick the most semi-custom (and hence fastest) approach that meets the physical constraints. For example, the random header generator datapath is built from standard cells, but the mapping from the behavioural description to standard cells was done by hand for maximum area efficiency, and routing was done by hand to exploit the regularity of the datapath structure.

If the total area exceeded the available space at any time, then an incremental step in customisation was taken. The finished layout just fits in the available space with very little margin, which is to say that it is very close to the easiest and quickest solution to the problem of fitting the specified function in a fixed silicon area.

The total design time was approximately 10 engineer-years including all architectural, design and verification work.

Clocking

The STC104 has a single fixed 5 MHz clock input from which high speed clocks are derived internally using phase-locked loops (PLLs). Since the DS-Link bit-level protocol carries its own clock, there is no requirement for any phase relationship between the 5 MHz clocks used on connected STC104s, or other devices connected thereto, which simplifies system design.

One PLL is used to clock the core of the chip. The clock multiplication factor is selected by pins, allowing a range of clock speeds other than the nominal 50 MHz design target. Thus devices sampled at the lower end of the manufacturing speed curve may be used successfully at reduced speeds. The PLL generates a 4-phase clock which is distributed around the chip to local clock drivers. These generate the functional clocks which are distributed locally to the individual blocks.

The input side of the DS-Link is clocked by the incoming data stream, and has special synchronisation circuitry to communicate with the core of the chip. The output side of the DS-Link uses another clock, generated by a second PLL. Thus it is quite possible for the core of the chip to be running at one speed, say 50 MHz, while the output side of a DS-Link is sending data at 100 MHz, and the input side of the same link is receiving data at 10 MHz. Each of the 32 DS-Links can have its output speed individually programmed, the input speed being determined by the programmed output speed of the other end of the link.

Block Level Communication—The Valid/Hold Protocol

All inter-block data communication is handshaken using a valid/hold protocol. Each data bus is accompanied by two handshake wires: 'valid' and 'hold'. The valid line runs in the same direction as the data, from producer to consumer, and can be interpreted as meaning 'the producer is ready: the data bits are stable and valid'. The hold line runs in the opposite direction, from consumer to producer, and can be interpreted as meaning 'the consumer is not ready: hold off completing the transfer'. In all cases the lines and the data bits may only be sampled on a

cycle boundary. The four possible combinations of valid and hold are:

valid	hold	meaning	
0	0	producer not ready	-no transfer
0	1	neither ready	-no transfer
1	0	both ready	-transfer takes place
1	1	consumer not ready	-no transfer

A transfer will only take place on a cycle boundary on which valid is TRUE and hold is FALSE. When the producer becomes ready it sets valid to TRUE and waits for a cycle on which hold is sampled FALSE. When the consumer becomes ready it sets hold to FALSE and waits for a cycle on which valid is sampled TRUE. The consumer can guarantee no data will be lost if it is not ready (e.g. it has no buffer space) by setting hold to TRUE—the transfer cannot take place until hold is FALSE, so the producer is forced to wait until the consumer is ready. This ensures complete token-level flow-control throughout the device, which, combined with the flow-control protocol on the DS-Links, ensures that buffer overrun never occurs. This protocol also allows a data value to be transferred every cycle, and so maximises performance of the inter-block communication.

Control System and Programming

The STC104 is a highly programmable system, containing approximately 28 kbits of user defined data. In addition to the interval routing tables, (of which there are 32, each containing 36 23-bit words), there are many special features individually programmable by the user on a per-link basis. All this programmable state is accessible remotely via the STC104 'control unit', which also manages remote reset, error reporting, and several housekeeping functions.

The control unit has two dedicated DS-Links at least one of which is connected into a 'control network' using a special control protocol above the protocols previously described [10]. A central controller can use this network to interact with the control unit of any device in the system, using only a single DS-Link. The control unit acts as an agent for the central controller, implementing instructions received over the control network. The central controller can thus remotely reset any chip, and read or write any programmable registers in that chip, e.g. the interval routing tables in the STC104. Within the STC104, the control unit uses a 3-wire serial bus to issue commands to all the blocks in the chip. Each major block has a sub-unit which interfaces the parallel registers in the block to the serial bus.

A further dedicated bus (the 'error pipeline') is daisy-chained between all 32 linkslices. The error pipeline has 32 sections, one in each linkslice, each of which can in any cycle either pass on an error report from its upstream neighbour, or inject a new error report into the pipeline and pass it to its downstream neighbour. The end of the error pipeline is connected to the control unit. When error reports reach the control unit, a message is sent to the central controller identifying the source and nature of the error.

PERFORMANCE

In this section we consider the peak performance of the STC104. Although this peak is achievable, it is of course unlikely to be attained in realistic situations. For a more detailed analysis of average and worst-case performance, see chapters 6 and 7 of [10]. The purpose of this section is to demonstrate that the rate at which the chip can perform its various functions is never a limiting factor.

Since the shortest packet is 14 bits, a packet can be received along a link running at 200 MBaud every 70 ns. If the link is operating bi-directionally with similar data patterns, an average additional bit-time must be allowed for the occasional 4-bit flow-control tokens, so the maximum sustainable packet rate is one per 75 ns per link. Thus with all links operating bi-directionally at 200 MBaud packets can be received at a peak rate of $32/(75 \times 10^{-9}) = 4.3 \times 10^8$ per second.

Arbitration and Routing

The crossbar of the STC104 contains independent arbiters for each output, which can all operate simultaneously. Thus the instantaneous peak arbitration rate is $32 \times 50 \times 10^6 = 1.6 \times 10^9$ arbitrations per second. The continuous peak is limited by the maximum packet rate above.

Since each linkslice operates independently of the others, there are no shared resources to limit the packet processing rate. Since the arbitration rate is more than adequate, packets can in fact by routed at the maximum rate of 4.3×10^8 per second. The pipelining and absence of limiting shared resources in the STC104 allows this rate to be sustained indefinitely, albeit in the unlikely circumstance that

very short packets arrive continuously on all links, and are routed so that there are no conflicts.

Arithmetic

Each interval comparator is capable of making 36 8-bit comparisons in parallel per cycle, each of which is equivalent to an 8-bit integer addition. Therefore the raw arithmetic performance of the 32 interval comparators is $32 \times 36 \times 50 \times 10^6 = 5.76 \times 10^{10}$ 8-bit adds/second. In practice the fastest arrival rate of header bytes is one two-byte header per 25 bit-times, which at 200 MBaud is two bytes every 130 ns, i.e. approximately every 6.5 cycles.

The random header generator contains 2 16-bit adders and is capable of a raw performance of $32 \times 2 \times 50 \times 10^6 = 3.2 \times 10^9$ 16-bit adds/second. In practice the random header generator does 5 useful 16-bit adds per header, and the maximum arrival rate is one packet per 6.5 cycles[3], so the usable performance is: $32 \times 5 \times 2 \times 50 \times 10^6/6.5 = 2.5 \times 10^9$ 8-bit adds/second.

Combining the two, the total usable arithmetic power of the STC104 is approximately 2×10^{10} 8-bit adds/second.

Latency

Latency calculations for the STC104 have to account for two different clock regimes. The core is clocked at a fixed speed (50 MHz), while the link is clocked at a variable speed, from 10 to 200 MBaud. The equations below refer to 'system cycles' which are cycles of the 50 MHz clock as applied to the core of the chip, and 'link cycles' which are those of the DS-Link variable clock[4].

The DS-Link has a latency of 4 system + 17 link cycles on its input side, and 1 system and 22 link cycles on its output side. Further blocks in the datapath contribute the following latency (for double byte headers):

input linkslice FIFO	1 system cycle
token queue	4 system cycles
crossbar	3 system cycles
output linkslice FIFO	1 system cycle

The total latency is therefore 14 system + 39 link cycles. A system cycle is 20 ns, and at a link speed of 200 MBaud one link cycle is 5 ns. The total latency is therefore: $14 \times 20 + 39 \times 5 = 475$ ns. This is the latency which can be observed in low-load conditions, whereas under heavy load the latency is dominated by contention for outputs, as discussed in [10].

Bandwidth

As described in the second section, the DS-Link uses 10 bits to represent a data token (one byte) and 4 bits to represent a control token in the serial protocol. A packet contains one or more data tokens, and exactly one 4-bit termination token marking its end. Packets may have an unbounded number of data tokens before the termination token, so the density of termination tokens varies from 0% to 50%. Thus the packet protocol imposes a fixed cost of one termination token (4 bits) per packet and a continuing cost of 0.5 bits per token (data or termination) to pay for flow control, as previously described.

The best case, for token bandwidth, is that of successive short packets (1 data + 1 termination token), as that case has the highest proportion of termination tokens (50%), which require fewer bits to transmit. Such a packet stream uses an average of $(10 + 4)/2 = 7$ bits/token; the overhead of implementing flow control costs another 0.5 bits/token, on the assumption that the links all operate bi-directionally. Since each bit requires a certain time to transmit, the lowest ratio of bits/token gives the highest rate of tokens/second. However, counting only data tokens this is in fact the *worst* case as all 15 bits (including flow control overhead) only contribute one byte. The byte rate is then the raw bit rate / 15.

The best case for the data byte rate is that of very long packets. As the packet length tends to infinity, the average cost falls towards 10.5 bits/byte (the half bit is the flow control overhead). Figure 3 shows two bandwidth lines, the upper (marked 'long packets') showing a cost of 10.5 bits/byte, the lower (marked 'short packets') showing a cost of 15 bits/byte. All real packet transfers will fall between these bounds, depending on the packet size, which is the number of bytes over which the fixed packet overhead cost can be shared. The theoretical maximum token bandwidth is exactly twice the rate of the lower line.

With all its links running at 200 MBaud, the STC104 can sustain a data bandwidth (allowing for all protocol overheads) of 19 MBytes/s on 32 paths simultaneously, or 610 MBytes/s in total.

[3]In fact packets with single-byte headers can arrive every 4 cycles.

[4]We assume here that all links operate at the same speed.

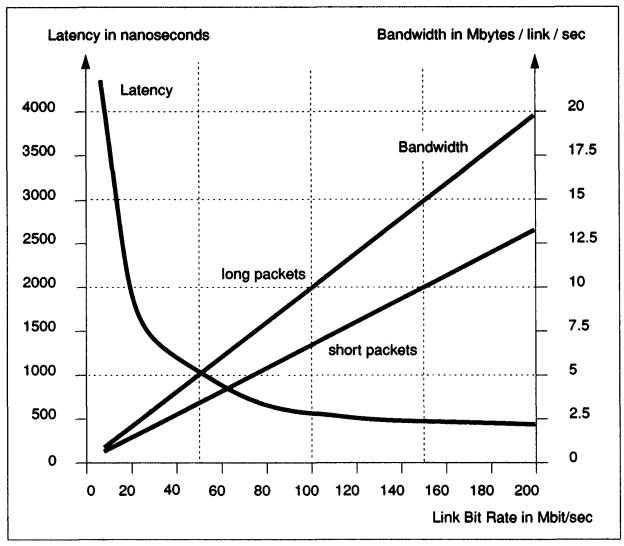

FIGURE 3 STC104 Peak Bandwidth/Latency

CONCLUSION

The STC104 is a landmark in the construction of cost-effective high-performance interconnection networks. The combination of simple protocols on high-speed serial connections, wormhole routing, and interval routing, enables a high fan-out device to be constructed on a single chip, with room left over to implement sophisticated features such as grouped-adaptive routing and two-phase Universal Routing. The highly concurrent and pipelined architecture of the chip gives it the power to switch over 400 million packets per second, while careful attention to inter-block and inter-chip handshaking ensures that data is never lost except in the very rare event of a serial communication error.

The STC104 is a commercial device in production, with documentation and software support. It is being designed into a wide variety of systems, from telecommunications subsystems to parallel computers. It allows a variety of interconnection networks to be constructed and configured to meet the particular needs of a wide range of systems. The size and topology of a network can be varied to satisfy requirements for latency, bandwidth, fault-tolerance and cost. All barriers to the widespread use of point-to-point interconnection networks are thereby removed.

References

[1] W.J. Dally and C.L. Seitz, "The Torus Routing Chip" *J. Distributed Computing,* 1, no. 3, pp. 187–196, 1986.

[2] M.J. Flynn, "Very High Speed Computing Systems," *Proc IEEE,* 54 (1966) pp. 1901–1909.

[3] C.A.R. Hoare, *Notes on Communicating Sequential Processes (PRG-33),* Oxford University Computing Laboratory, 1983.

[4] INMOS Limited, *occam 2 Reference Manual,* Prentice Hall, 1988.

[5] INMOS Limited, *The T9000 Transputer Hardware Reference Manual,* INMOS Limited, 1993.

[6] A. Klein, "Interconnection Networks for Universal Message-Passing Systems," *Proc ESPRIT Conference 91,* pp. 336–351, Nov. 1991.

[7] S. Konstantinidou and L. Snyder, "Chaos router: a practical application of randomisation in network routing," *Proc. Symp. on Parallel Algorithms and Architectures,* pp. 21–30, 1990.

[8] J. van Leeuwen and R.B. Tan, "Interval Routing," *The Computer Journal,* 30(4), 298–307, 1987.

[9] A.J. McAuley, "Four State Asynchronous Architectures," *IEEE Transactions on Computers,* 41, No. 2, 1992, pp. 129–142.

[10] M.D. May, P.W. Thompson and P.H. Welch (Eds.), *Networks, Routers and Transputers: Function, Performance and Applications,* IOS Press, 1993.

[11] L.M. Ni, Y. Lan and A-H. Esfahanian, "A VLSI router design for hypercube multiprocessors," *Integration,* 7 (1989), pp. 103–125.

[12] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer,* 26, no. 2, February 1993.

[13] P.W. Thompson, "Globally Connected Fault-Tolerant Systems," in Kerridge, J. (ed.), *Transputer and occam Research: New Directions,* IOS Press, 1993.

[14] L.G. Valiant, "A scheme for fast parallel communication," *SIAM J. on Computing,* 11 (1982), pp. 350–361.

[15] C.P.H. Walker, "The Bus-less Computing Environment," *Proceedings of BUSCON/92-West,* CMS, 1992.

[16] C. Whitby-Strevens, *Standard for Heterogeneous Interconnect (Draft),* INMOS Limited, 1994.

Biographies

JULIAN D. LEWIS graduated in Computer Science from the Department of Computer Science at the University of Manchester, England, in 1989 and has worked for the transputer group at INMOS since; first at Bristol, England and then at the Catania Design Centre in Italy. He has worked on the IMS T9000 superscalar microprocessor, and the IMS C103 (ASIC) and STC104 (full-custom) packet router chips in both silicon design and design verification. He is currently working on the architecture and design verification strategy for the latest INMOS transputer family. He is a member of the BCS and IEEE.

PETER W. THOMPSON obtained a degree in Mathematics and Physics from the University of Warwick, England, in 1980, and studied mathematical physics at the Universities of Cambridge and Oxford until 1985. He is now a Principal Design Engineer at INMOS Limited, where his main rôle is the specification and top-level design of new VLSI devices. He has worked on parts of the IMS T9000 transputer, the IMS C103 and STC104 packet router chips, and the IMS C100 protocol converter and IMS C101 parallel link adapter chips. He is an Associate Fellow of the Institute of Mathematics and its Applications and a member of the IEEE.