SLAC -PUB - 5184
January 1990
(T)

SCALABLE COHERENT INTERFACE

Knut Alnæs & Ernst H. Kristiansen, Dolphin Server Technology A.S., Oslo, Norway
David B. Gustavson*, Stanford Linear Accelerator Center, Stanford, California
David V. James, Apple Computer, Cupertino, California

Abstract

The Scalable Coherent Interface (IEEE P1596) is establishing an interface standard for very high performance multiprocessors, supporting a cache-coherent-memory model scalable to systems with up to 64K nodes. This Scalable Coherent Interface (SCI) will supply a peak bandwidth per node of 1 GigaByte/second. The SCI standard should facilitate assembly of processor, memory, I/O and bus bridge cards from multiple vendors into massively parallel systems with throughput far above what is possible today.

The SCI standard encompasses two levels of interface, a physical level and a logical level. The physical level specifies electrical, mechanical and thermal characteristics of connectors and cards that meet the standard. The logical level describes the address space, data transfer protocols, cache coherence mechanisms, synchronization primitives and error recovery. In this paper we address logical level issues such as packet formats, packet transmission, transaction handshake, flow control, and cache coherence.

1 INTRODUCTION

The Scalable Coherent Interface (SCI) Project started in November 1987 as a study group under the Microprocessor Standards Committee (MSC) of the Technical Committee on Mini- and Microcomputers in the IEEE Computer Society. Paul Sweazey was the chairman of the study group, which used the working name SuperBus. In July 1988 the study group became a working group, adopting the name Scalable Coherent Interface, chaired by David B. Gustavson.

The objective of the SCI working group is to define an interconnect system which scales well as the number of attached processors increases, provides a distributed cache-coherent memory system, and defines a simple interface between modules [1,4,5,7,8,11].

We quickly discovered that a traditional backplane bus could not achieve our goals. Today's buses are limited by the distance a signal must travel and the propagation delay across a backplane. In asynchronous buses, the limit is the time needed for a handshake signal to propagate from the sender to the receiver and for a response to return to the sender. In synchronous buses, it is the time difference between clock and data signals which originate in different places.

Transmission lines in backplanes are disturbed by connectors and variations in loading as the number of inserted modules varies. This makes reliable high speed signalling on a backplane bus very difficult. In addition, a backplane bus can only service one request at a time and therefore becomes a bottleneck in multiprocessor systems.

The SCI working group solves these problems by defining a radically different interconnect system. We are defining an interface standard which enables a system integrator to attach his board to an interconnect which may have many different configurations. These configurations may range from simple rings to complex multistage switching networks.

The interface standard defines a point-to-point communication between neighbor nodes, greatly reducing transmission line problems. This point-to-point link uses differential ECL signalling, allowing high speed transfers of 1 Gbyte/second though the link is only 2 bytes wide. Small packets carry data from node to node across these links, Buffering in the node interfaces accommodates many simultaneous requests, making SCI well suited to high performance multiprocessor systems. The SCI standard allows up to 64K nodes to be connected to an interconnect, and should provide the next generations of computers with sufficient interconnection bandwidth.

A bit-serial link is also under development, for use with fiber optic or coaxial cable links over longer distances (but at lower speeds). The bit serial version will support the same architecture and protocols as the 2-byte-wide version.

Cache coherence is an important part of the proposed standard. Current mechanisms prove insufficient when the number of processors increases dramatically. This calls for a new approach to the cache consistency problem. The SCI working group is defining a scalable distributed directory scheme where processors sharing cache lines are linked together by pointers stored in the caches.

High volume products using the SCI standard are expected to become available by the mid-1990's. Figure 1 gives a rough estimate of future volumes of board level products[2].

* Work supported by the Department of Energy, contract DE-AC03-76SF00515.

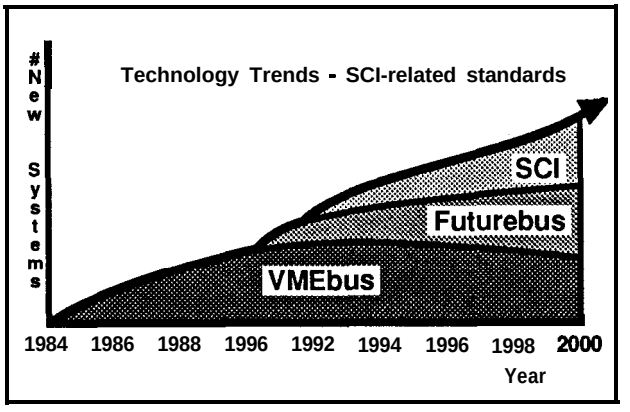

Figure 1. Technology trends.

The following sections provide more insight into the solutions which the SCI working group is currently pursuing. The next section describes various configurations of an SCI system and emphasizes interfacing via different interconnects. The packet format and packet transmission is described in section three. In section four we focus on the mechanisms for packet flow control. Section five gives a brief overview of the cache coherence model. Finally, we summarize the standardized Control and Status Register space and the status of realization in silicon.

2 CONFIGURATIONS

SCI supports multiple configurations ranging from simple low cost implementations to high performance, high cost systems. An important property of SCI is that it includes hooks to allow several different implementations to reside simultaneously in a system. This is done by separating the interfacing node from the transporting interconnect. A view of a typical system is illustrated in Figure 2.

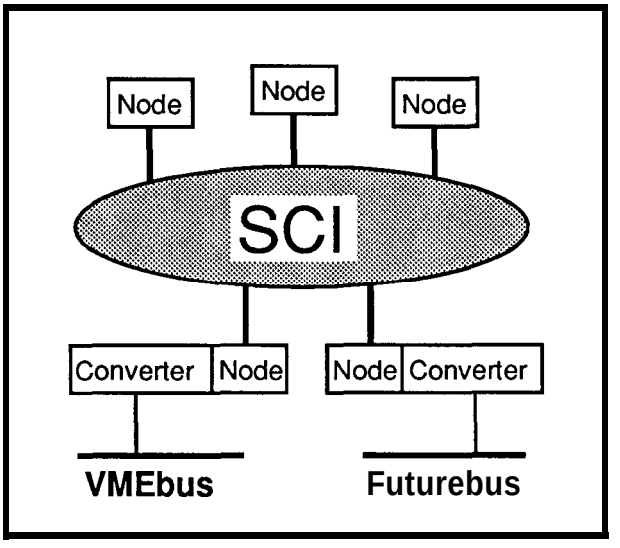

Figure 2. SCI Configuration.

2.1 SCI viewed by a node

An SCI node receives a steady stream of data and transmits another stream of data. These streams consist of SCI packets and idle symbols. A node is responsible for operating on these packets and idle symbols according to the SCI standard. To do that, a node may have the construction shown in Figure 3.

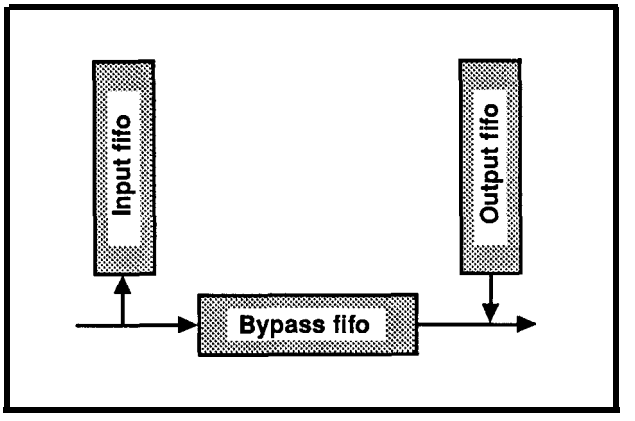

Figure 3. SCI interface.

When there is no traffic on the SCI interconnect, a node receives idle symbols. Since the utilization is zero in this case, all nodes are free to transmit. The idle symbols convey this information to the nodes. In case the node has nothing to send and the bypass fifo is empty, the output consists of idle symbols only.

When a node receives a packet, it checks the packet's destination. Packets destined for other nodes are routed to the bypass fifo and transmitted onward. The retransmitted packet accumulates flow-control information for other SCI nodes. The flow-control information is divided between the packet header and the (minimum one) idles separating the packets. The arbitration, priority and forward progress schemes are enforced this way.

When a node receives a packet which is destined for it (and it is ready to accept it), the packet is routed to the input fifo until the node has time to process it further. The packet's header information is also used to generate a short 'echo' packet, which is routed to the bypass fifo, ultimately to be received by the packet's sender. The echo is part of the arbitration, priority and forward-progress mechanisms.

A node which is granted interconnect access and which has an empty bypass fifo is allowed to transmit a packet. Since many nodes may have interconnect access simultaneously, multiple nodes may transmit at the same time. This contention is solved either by buffering in the interconnect or by filling

the bypass fifo of the transmitting node(s). The SCI system uses idles, packet headers, and echoes to selectively grant interconnect access under heavy system loading.

2.2 SCI interconnect

SCI can be configured in many ways. However, there are two basic structures-the ring and the switch. The ring implementation is the simplest. In a ring, nodes pass packets to their neighbors. In such a structure there are no active components except the nodes. This means that the nodes themselves have to control the arbitration, priority and forward progress schemes.

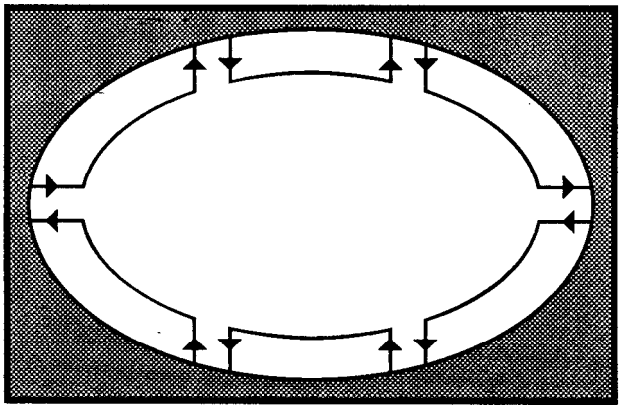

Figure 4. Ring interconnect.

A switch looks at the destination address and routes the packet directly to the destination. A switching structure can have various complexities and costs, including full crossbar switches and butterfly switches. In a switching structure, priority and forward progress schemes must be enforced by the switch. However, the node interfaces are the same in both a ring and a switch implementation.

2.3 Interconnection to other buses

Another important feature of SCI is the ability to interf'ace to other buses. Some SCI transactions and cache states are specifically defined to accommodate other buses.

A bus bridge will respond to a range of destination addresses. The bus bridge node is responsible for converting SCI transactions into native bus transactions. Two cases are handled with special care: bus locking and cache coherence.

Most backplane buses accommodate a unique read-modify-write transaction to manipulate semaphores and other critical data. During the read transaction a lock signal is asserted, inhibiting the use of the bus until the data is written. Since SCI is defined with a four-phase transaction protocol with no guaranteed delivery order, a lock must be executed as a single SCI transaction.

Some bus protocols also incorporate a cache coherence scheme. Most use a snooping scheme where bus interfaces monitor all bus activity and update their cache states accordingly. In SCI this is not possible, since no one node can observe all the relevant transactions.

2.4 Scalability

A significant aspect of SCI is scalability. It should be possible to have a simple, cheap system with the same basic properties as a high performance one. To achieve this, a large and important task of the SCI working group is to assure that enough, but not too much, functionality is included in the standard.

A simple and cheap system would be a ring, with all packets at the same priority. This results in round-robin arbitration. A requesting node is simplified by allowing only one packet outstanding at any time, but it still needs separate request and response queues. A responding node might only be able to handle a single request at a time. If it is busy, a busy echo will inform the sender to re-transmit the packet.

A more complex, but still fairly inexpensive, system could use a combination of rings and bridges. The rings would be used between nodes which require low latency and where the ring bandwidth is sufficient. The bridges would be used to connect rings. Such a system could even support a dynamic interconnect where any node can be plugged into any socket. Multiple outstanding requests and live insertion might be supported.

The most complex system would be a switching interconnect built of elements like the butterfly switch. This interconnect is hardwired, so a node can only be plugged into its addressed location. This kind of interconnect would handle more traffic, and multiple outstanding requests from a requesting node could be supported. In addition to the round-robin arbitration scheme, multiple priority levels could be

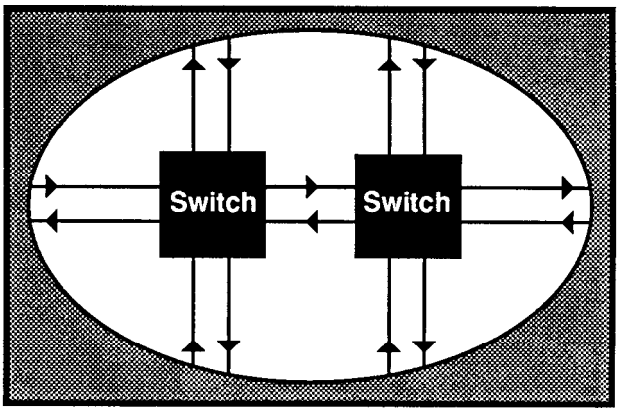

Figure 5. Switch interconnect.

provided. This interconnect also supports live insertion and withdrawal, and may be able to implement request-combining schemes to reduce the effect of congestion at hot spots.

3 PHYSICAL LAYER

SCI specifies signals at an interface to an interconnect system. All signals are unidirectional differential 100k ECL compatible signals. 18 signals are sent from a node: 16 data signals, 1 flag bit and 1 clock signal. The frequency of the clock is 250 MHz. The skew between the signals is one of the most critical items.

Power distribution is solved by distributing 48 VDC to all nodes and using on-board power converters. This reduces the number of pins-needed for power and ground, allows the vendor to select the optimal voltages for various logic families and interface needs, greatly simplifies power-on module replacement, and makes uninterruptible power supplies very simple via storage batteries.

The board size recommended is 6U (233.35mm) x 280mm.

4 PACKET FORMAT

Figure 6 shows the packet format. The width of a packet word is 16 bits. In addition, a flag indicates that a packet is being received or transmitted. Each word in the packet is clocked with a differential clock line. A node receives 2 bytes at a rate of 500MHz resulting in an interconnect bandwidth of 1 Gbyte/second.

A packet consists of three main sections: a header section, an address and data section, and an error check word. The first 16-bit word of the header contains the ID code of the final receiving node. By looking at the first word of a packet, a node can quickly determine if the packet is addressed to that node. During routing through an SCI interconnect, intermediate nodes and switches look at the target word to determine where to route the packet. The third word of the packet contains the ID code of the sender, needed to address the response back to the correct sender, as shown in Figure 7.

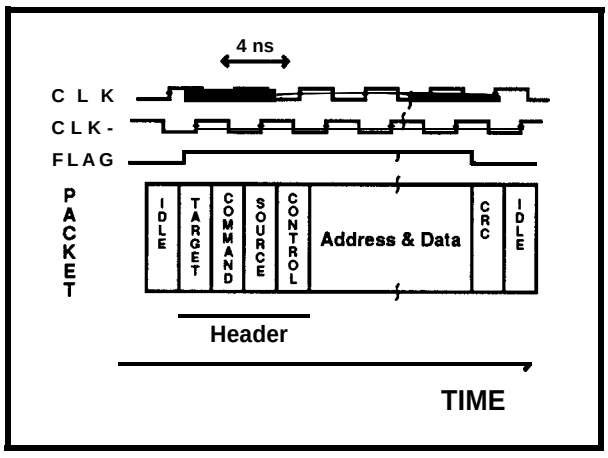

Figure 6. Packet format.

The command word of the header controls packet flow and interconnect access. Priority arbitration is supported with round robin arbitration on the lowest level. Flow control and arbitration will be discussed in more detail in section 5. The command word of the header also contains the transaction type and the packet length.

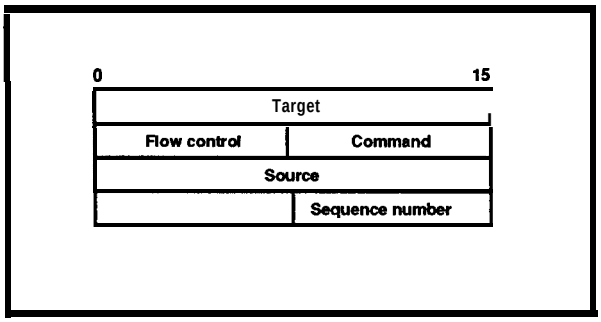

Figure 7. Header format.

The command field contains the command a responder must execute. In a multiprocessor SCI environment, a command is often applied to a cache line. The cache line size is 64 bytes, but manipulations on smaller and larger data sizes are also supported. The commands can be divided into cache coherence transactions, lock transactions, DMA transactions, and I/O register transactions. The cache coherence transactions manipulate a linked-list structure used to maintain a coherent memory image.

The sequence number in the control word is a label which identifies a packet. A node connected to an SCI interconnect may send many requests (up to 64), before a response is received. This transaction pipeline can cause responses to be returned out of order, and therefore a sequence number is needed to identify a response with the corresponding request.

The target word and the three first address words define the 64-bit SCI address. The data part may contain from 16 to 256 bytes. When a packet is transmitted, a cyclic redundancy code (CRC) for the packet is computed, and this code is attached after the last word of the packet. The CRC is a "serial-parallel" version of the 16-bit CCITT-CRC.

4.1 Packet reception

In an SCI interconnect, a node is addressed by a 16-bit identification code, which is located in the first word of the packet. This allows 64K nodes to be attached to the interconnect. This allows for easy detection, and decisions to pick up the packet can be made quickly. An input flag marks the beginning of a packet; if the target ID of the packet

matches the ID code of the node, the packet is stripped from the interconnect. While the packet is being stripped and received, a CRC for the packet is computed. The computed CRC code is compared with the CRC code at the end of the packet. If they match, the reception is completed; otherwise the packet is discarded.

A stripped packet creates a small echo packet, with interchanged target and source IDs. The echo packet is returned to the sender for flow control. If the input fifo was not empty, the busy bit in the control word of the echo is set, so that the sender knows it must retransmit the packet later. If bad CRC is received, the echo CRC is complemented so it will be discarded (it is too late to avoid its transmission).

4.2 Packet transmission

A node. may transmit if the bypass fifo is empty (see Figure 3) and the node is granted interconnect access through the flow control mechanism. Before transmission, the packet is put into the output fifo.

Transmission starts by putting the target word onto the output and setting the output flag high. The output flag is high while the packet is being transmitted. A CRC is attached to the end of the packet when the output flag goes low.

If a packet enters the node interface during transmission, and the packet is not for this node, the packet is put into the bypass fifo until the transmission is done. The size of the bypass fifo must therefore be at least as large as the maximum transmitted packet size to avoid fifo overflow.

4.3 Transaction handshake

SCI supports a transaction pipeline up to 64 transactions deep. This means that a node may send up to 64 requests without waiting for a response. A normal transaction consists of two subactions, a request subaction and a response subaction. Together with each subaction there is an echo packet returned to the sender, as shown in Figure 8.

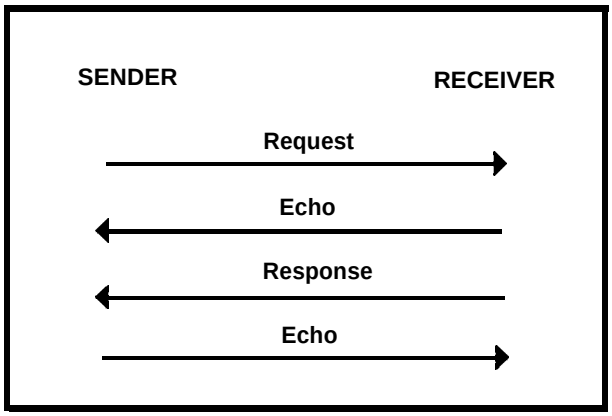

Figure 8. Split transaction handshake.

When the request is transmitted, it is labelled with a sequence number. The ID code of the sender and the sequence number uniquely identify a packet in the SCI interconnect. When a responder accepts a packet, the sequence number in the request packet is saved. The responder will add this sequence number to the response packet when the response is transmitted back to the sender.

Transmission errors could cause many kinds of problems. Fault recovery has been carefully considered, and most of the burden placed on software error handlers. The principle relied on is that transmission errors are detected by a time-out mechanism so the sender can retry a transaction if no echo or response has been received within the time-out interval.

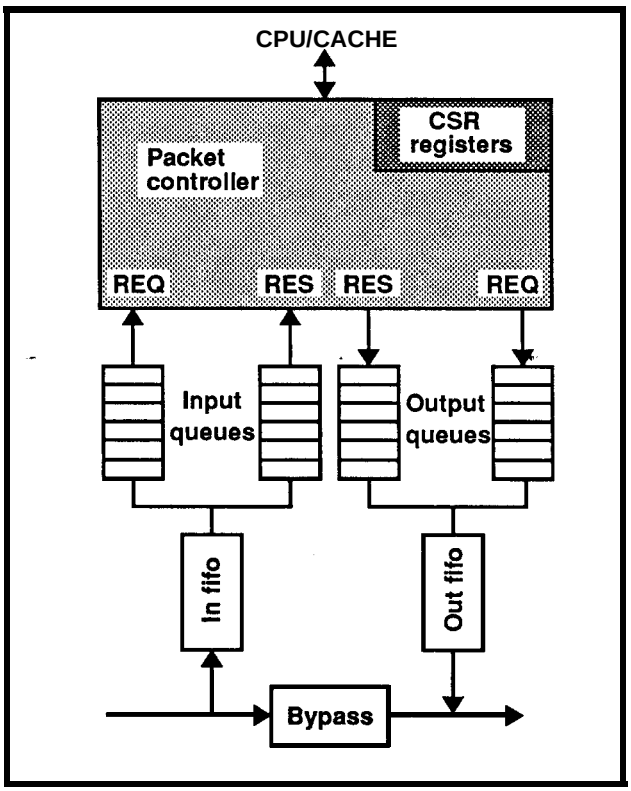

Figure 9. Node interface.

5 FLOW CONTROL

In SCI, flow control of packets is needed to maintain high throughput and fair access when many packets are sent to the interconnect at the same time. The flow control issues discussed in this section are arbitration, deadlocks, servicing, and congestion.

As explained earlier, a node may transmit when its bypass fifo is empty. This means that up to 64K nodes may start to transmit at once, allowing 64K packets to exist in the interconnect. However, nodes connected to a ring can not

retransmit until their bypass fifos are empty. To avoid starvation, an arbitration algorithm ensures that all nodes have access to the ring. Our current algorithm is based on fair and priority transactions. The arbitration mechanism is enforced by header information and idle symbols between packets. The priority level of a transaction is coded into the command word of the packet header (as shown in Figure 7).

Another node which wants to transmit and has a higher priority marks the header of a passing packet. This informs the packet's sender that another node with a higher priority wants to transmit. This flow-control information is also distributed to others, in idle symbols between the packets.

To avoid deadlocks, separate request and response queues are added to each input and output fifo as shown in Figure 9. To ensure fairness, packets are selectively accepted into these queues, based on an approximate packet aging protocol. Also, the acceptance protocol can be influenced by the incoming packet's priority.

6 CACHE COHERENCE

High performance processors need local caches to reduce the effective memory access latency. In a multiprocessor environment this- leads to potential conflicts because several processors may simultaneously want to modify locally cached copies of the same dam.

Cache coherence protocols define mechanisms that guarantee consistent data even if data is cached and modified by several processors. The SCI definition supports a hardware-based cache coherence protocol, reducing the programmer's software effort to secure consistency, and also reducing operating system complexity.

Many existing cache coherence protocols use a snooping technique and rely on transactions like broadcast and eavesdropping to guarantee data consistency. In a large high speed distributed system, the broadcast transaction is ineffective at best, and eavesdropping is impossible to implement because it requires a bus common to all processors in the system. Since a highly scalable interconnect system is one of the main objectives in defining the SCI, these and similar mechanisms are unsuitable.

We have developed a directory-based cache coherence protocol[6] with distributed properties, where all the nodes with cached copies participate in the control. The principle is that every sharable block in memory is associated with a list of processors sharing that block. A memory block is usually the size of a cache line, which is 64 bytes.

The selection of 64 bytes as the cache line size is based on many factors. The density of current state of the art ECL chips prohibits packet sizes larger than 80 bytes because of the fifo buffering. An 80-byte maximum packet size has a reasonable overhead, making cache line transfers efficient for a 64-byte line size and less efficient for a 32-byte line size. Concern about false sharing makes a 12%byte line size less attractive, and trace driven simulations [10] show that a 64-byte line size is a good choice for SCI. Futurebus+ has also selected 64 bytes, making the interface between SCI and Futurebus+ simpler and more efficient.

Every block has a tag which includes a pointer to the processor at the head of the list. Each processor cache tag has a pointer to the next node sharing that cache line. In effect, all nodes with cached copies of a memory block are linked together by these pointers. The nodes have a forward pointer and a backward pointer to connect them with the previous and next node in the list. The resulting doubly linked list is shown in Figure 10.

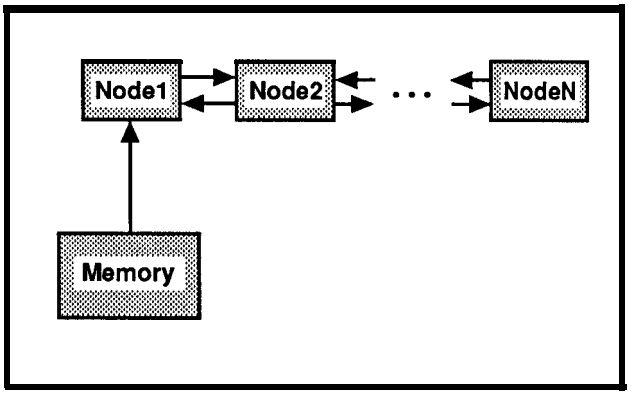

Figure 10. SCI sharing list.

This distributed list concept ensures good scaling properties. Even as the number of nodes in a list grows dramatically, the corresponding memory tag size is constant. However, two pointer locations are associated with every cached block in a node.

The list pointers are actually the interconnect addresses for the processors. When a node accesses memory to get a copy of shared data, it provides memory with its own address. If there are currently no nodes with cached copies, the requesting node is made the head of a new list and memory saves the node address in the tag for this block. If, however, there exist nodes with cached copies of data, the pointer to the head of the sharing list is returned from memory to the requesting node, and this node inserts itself at the head of the list. Currently cached dam is always returned from the old head, rather than from memory.

The nodes in a linked list typically have read access to shared data. When a node wants write access, and it is currently the list head, then it purges the rest of the list. If it is in another portion of the list, the node first deletes itself from the list, then performs another memory read to move to the

head of the list. Write access is restricted to the head node only.

All bus transactions concerning cache coherence are implemented within the standard packet format described above.

The cache coherence protocol described above has not yet been tried in real systems. We are therefore relying on several people at the University of Oslo who are using their expertise to do formal verification[9].

7 CONTROL AND STATUS REGISTERS

The Control and Status Registers (CSRs) are an important part of the proposed standard. The CSR definitions are essential for all initialization and exception handling. Some of the CSRs must be SCI specific, but the majority of the necessary definitions can be common with other standards[3]. The IEEE has approved a request for a standard project for defining CSRs. The project number is IEEE P1212, chaired by David V. James. The CSR standard is being coordinated with Futurebus+, Serial Bus and SCI. It will also try to coordinate with the ongoing CSR activity for VMEbus.

8 REALIZATION

Realization in commercial systems is important for acceptance of a defined standard. Therefore the first implementation is being done in parallel with the standardization work. So far we have done measurements that assure us that it will be possible to make implementations for the 1 Gigabyte/second transfer rate.

We have both a high level and a low level simulation model of an SCI system running. We have simulated both the arbitration and the cache coherence scheme. The length of a maximum data packet will initially be limited to 64 bytes (i.e. a cache line). For the first implementation we are using ECL gate arrays, with one chip (or perhaps two) for the SCI interface and the cache coherence protocol. This interface chip will be common for all nodes. In addition, Dolphin Server Technology is making a physically addressed cache controller which can be used as a second level cache controller, and a global memory controller chip that supports the necessary directory handling in global memory.

The first configuration will be a ring structure with high performance CPU's, large main memory and connection to standard buses like VMEbus for I/O functions. We expect to have prototypes ready for testing late this year.

9 CONCLUSION

This paper has presented an overview of the objectives of the SCI working group, and the solutions which are currently being pursued. Scalability of a system is a key aspect as many high performance computer manufacturers are moving toward large multiprocessor systems. In order to utilize these systems efficiently, a cache coherence mechanism must have good scaling properties. Also, for a system to both be cost effective and support high performance solutions, it is necessary to separate the module interface from the interconnect implementation.

We feel that our current proposals meet these objectives. The SCI project is moving rapidly and has attracted participants from many of the high performance computer companies. We already have a first draft of the standard available, and we hope to send it out for ballot late this year. The proposed architecture appears to be achievable based on technology available today.

If you would like to participate in this work, or if you would like more detailed information, please contact one of the authors or the chairman of the project:

David B. Gustavson, IEEE P1596 Chairman
Computation Research Group, bin 88
Stanford Linear Accelerator Center
Stanford, CA 94309, USA
tel: (415) 926-2863
fax: (415)961-3530 or (415)926-3329
Email: DBG@SLACVM.bitnet

10 ACKNOWLEDGMENTS

Many people have already contributed to SCI's development: though we cannot list them all, we wish to acknowledge a few contributions which seem to us to be particularly significant.

Paul Sweazey, originally of National Semiconductor and recently of Apple Computer, started the SuperBus study group, which he chaired until the SCI working group was organized. Paul has also brought a thorough understanding of the cache coherence problem, due to his work coordinating the Futurebus Cache Coherence task group.

Paul Borrill of Sun Microsystems, Futurebus+ chairman, helped push our goals to much higher bandwidths and to increased parallelism through the use of switches instead of shared buses.

John Moussouris, a co-founder of MIPS Computers, has provided critical insights into the directions we need to take in order to rendezvous with future technology, has helped put us in touch with the appropriate experts, and has helped expose problems and errors in various models.

Phil Ponting of CERN in Geneva has provided effective and vital communications and redistribution to the many European participants.

Hans Wiggers of Hewlett Packard Laboratories has helped us examine various physical layers, and is considering the implications of an optical fiber implementation of SCI.

Mark Williams of Hewlett Packard leads a joint task group to consider the interface between SCI and Futurebus+.

Stein Gjessing, Stein Krogdahl and Ellen Munthe-Kaas at the University of Oslo are working on formal specification and verification of the SCI cache coherence protocols.

11 AUTHORS

Knut Alnæs, originally of Norsk Data and now a staff member of Norsk Data subsidiary Dolphin Server Technology A.S., Oslo, Norway, is involved in SCI simulation and is responsible for the interface chip design.

David B. Gustavson works in the Computation Research Group of the Stanford Linear Accelerator Center at Stanford University, Palo Alto, California. He is chairman of the IEEE P1596 (SCI) working group. He has had experience with many standard buses, beginning with the early S-100 (IEEE 696) but especially Futurebus (IEEE P896.x) and Fastbus (IEEE 960-1986, IEC 935). He also chairs the Fastbus Software Working Group (IEEE 1177-1990).

David V. James, originally of Hewlett Packard and recently of Apple Computer, Cupertino, California, has broad experience with multiprocessor architecture, which he is applying to SCI's needs, from control register and I/O architecture to distributed cache coherence and forward progress. David is vice chairman of SCI, coordinator of the Logical task group and has written the majority of the working documents. He is also the chairman of the CSR project (IEEE P1212).

Ernst Kristiansen was responsible for development of memory systems, proprietary buses and I/O-systems at Norsk Data, and is now at Norsk Data's subsidiary, Dolphin Server Technology A.S., Oslo, Norway, where he is project manager for the SCI implementation.

12 REFERENCES

1. **D. B. Gustavson, D. V. James, J. Moussouris and P. Sweazey, "The Scalable Coherent Interface Project (SuperBus)", draft of August22, 1988.**

2 **P. L. Borrill "VMEbus—The Next 5 Years", VMEbus in Research, October 1988.**

3. **D. V. James, "Scalable I/O Architecture for Buses", COMPCON Spring 1989, pp 539-544.**

4. **D. B. Gustavson, "Scalable Coherent Interface", COMPCON Spring 1989, pp 536-538.**

5. **E. H. Kriitiansen, "IEEE SCI (P1596)", VMEbus in Research, October 1988.**

6. **A. Agarwal, R. Simoni, J. Hennessy and M. Horowiz, "An Evaluation of Directory Schemes for Cache Coherence", 15th International Symposium on Computer Architecture, June 1988.**

7. **E. H. Kristiansen, K. Alnes, B. 0. Bakka and M. Jenssen, "Scalable Coherent Interface", Eurobus Munich, May 1989.**

8. **P. Sweazey, "Cache Coherence on SCI", IEEE/ACM Computer Architecture Workshop, Eilat, Israel, June 1989.**

9. **S. Gjessing, S. Krogdahl, E. Munthe-Kaas, "Formal Specification and Verification of SCI Cache Coherence", NIK 89, Stavanger, Norway, November 1989.**

10. **H. 0. Bugge, E. H. Kristiansen, "Trace Driven Simulations for Decisionmaking on Cache Line Size and Cache Size in a Two Level Cache Design", NIK 89, Stavanger, Norway, November 1989.**

11. **D. B. Gustavson, "The Scalable Coherent Interface, IEEE P1596, Status and Possible Applications to Data Acquisition and Physics", IEEE Nuclear Science Symposium, 1989.**