SLAC - PUB - 5656
September 1991
(I)

The Scalable Coherent Interface and Related Standards Projects*

David B. Gustavson

Stanford Linear Accelerator Center
P.O. Box 4349, MS 88
Stanford, CA 94309, USA

Abstract

The Scalable Coherent Interface (SCI) project (IEEE P1596) found a way to avoid the limits that are inherent in bus technology. SCI provides bus-like services by transmitting packets on a collection of point-to-point unidirectional links. The SCI protocols support cache coherence in a distributed-shared-memory multiprocessor model, message passing, I/O, and local-area-network-like communication over fiber optic or wire links. VLSI circuits that operate parallel links at 1000 MByte/s and serial links at 1000 Mbit/s will be available early in 1992.

Several ongoing SCI-related projects are applying the SCI technology to new areas or extending it to more difficult problems. P1596.1 defines the architecture of a bridge between SCI and VME; P1596.2 compatibly extends the cache coherence mechanism for efficient operation with kiloprocessor systems; P1596.3 defines new low-voltage (about 0.25 V) differential signals suitable for low power interfaces for CMOS or GaAs VLSI implementations of SCI; P1596.4 defines a high performance memory chip interface using these signals; P1596.5 defines data transfer formats for efficient interprocessor communication in heterogeneous multiprocessor systems.

This paper reports the current status of SCI, related standards, and new projects.

1 Introduction

The Scalable Coherent Interface was developed by a number of bus designers and system architects that had come to understand the fundamental limits to bus technology during their work on Fastbus (IEEE 960) and Futurebus+ (IEEE 896.x). These modern buses push bus signalling technology to its limits, and provide various architectural features that support the use of multiple processors.

However, these bus limits are rapidly becoming a serious problem as the demand for computing power continues to grow. The economic reality is that this demand can only be met by using a large number of fast microprocessors. But buses are inherently a bottleneck (only one transfer at a time) and their signalling speed is limited by the imperfect transmission lines that result from bus-style connections.

Therefore, buses can't support a large number of processors, especially not fast ones. While their useful life can be extended a bit by cleverness and brute force, the potential gains are relatively small and the costs become very high. For example, doubling the width of a bus does not double its speed because there are fixed overheads associated with arbitration and addressing. Lengthening block transfers to reduce the effect of these overheads is of little use once the blocks exceed the size of cache lines.

Signalling speeds can be increased by shortening the bus, but that makes it less useful. Reducing the signal voltage helps, but eventually encounters noise problems. Using multiple buses to get more than one transfer at a time results in a complex (expensive) bus-bridge mechanism to maintain cache consistency (coherence) in shared-memory systems that use bus-snooping technology.

The Superbus Study Group was started by Paul Sweazey (Futurebus+ cache-coherence task group coordinator) in November of 1987 to see if there were potential solutions to these problems. In July 1988 the outline of the solutions had become clear and the study group was replaced by the P1596 SCI working group. The work was essentially completed by January 1991, when specification draft D1.00 went out for ballot, and has subsequently been undergoing minor improvements, polishing, and debugging of the specification C code during the review process. (Most of the SCI specification is executable, to reduce ambiguity, simplify testing and enable accurate simulation.)

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

Paper presented at Open Bus Systems '91, November 26–27, 1991, Paris France

2 SCI Goals

The SCI design goals include:

- **Scalability,** so that the same mechanisms can be used in high-volume single-processor (or few-processor) systems such as one might find in desktop machines, as well as in large highly parallel multiprocessors (next generation supercomputers).
- **Coherence,** to support the efficient use of cache memories in the most general and easiest-to-use multiprocessor model, distributed shared memory.
- An **Interface,** a standardized open communication architecture that allows products from multiple vendors to be incorporated into a single system and interoperate smoothly.

Scalability is important to get the costs down, not only through increased volume of production but through the simplicity of having to learn only one new paradigm, one that will work over several generations of machines.

A standard SCI module defines signals, connector and power for operation at a link speed of 1000 MByte/s. A standard cable and connector defines the use of these signals for applications where the module form factor is not appropriate, over short distances (meters). A standard fiber-optic serial interface solves interface problems over longer distances (kilometers) at a link speed of 1000 Mbit/s. The same bit stream may be sent over coaxial cable at lower cost for medium distances (tens of meters). Other speeds and signalling technologies will be standardized in the future as appropriate.

2.1 Solving the Bus Signalling and Bottleneck Problems

SCI needed two fundamental changes from the way a bus transmits information. First, to make signalling speed independent of the size of the system, interfaces don't wait for each signal to propagate (i.e., a bus cycle) before sending the next. This means that each communication is performed by sending packets. Packets include an address, command and data as needed. While the propagation velocity of a packet is still limited by the speed of light, its rate of information transfer is not. As technology advances, the transfer rate can increase indefinitely. Computer scientists know techniques that can compensate for the bad effects of delay (latency), but little can be done to compensate for a transfer rate (bandwidth) that is too small.

Secondly, SCI uses multiple signal paths (links) so that multiple independent transfers can take place concurrently. For high performance one can use separate links for each processor, memory, or I/O device.

The SCI link design was further refined by applying lessons learned in practical multiprocessor systems. The link signals are differential, because differential signals produce the least system ground noise and are least sensitive to noise from other sources. The links are unidirectional because bidirectional links create noise when the drivers are turned off or on to reverse the direction of the link, and because turn-around delays increase with cable length, a scalability problem. The links are fast and narrow because interface chips are expected to be pin-limited.

Reverse-direction flow control signals are not used, because such mechanisms make the amount of buffer storage needed in the interfaces dependent on cable length. In order to reach high speeds, small-voltage-level differential signals were used. SCI initially uses 16-bit-wide ECL-compatible signals because of the industry experience with and support for that standard. Future links will probably use even lower voltages that are chosen for compatibility with VLSI CMOS or GaAs circuitry.

Thus each SCI interface (node) has (at least) two links, one incoming and one outgoing. The links run continuously, sending idle symbols when no packets are being transmitted, so that the receiver can remain perfectly synchronized at all times, ready for action. SCI packets do not need the prologue that is essential for Ethernet or similar networks.

In a high performance SCI system, a vendor-dependent switch accepts packets from nodes and routes them to the appropriate other nodes as specified by the address in the packet. SCI does not specify the details of such a switch, because many cost/performance tradeoffs are possible.

Lowest-cost SCI systems, such as desktop systems, typically will use a ring connection instead of a switch, connecting one node's output link to its neighbor's input. link The interface circuit was made more complex by the decision to support rings, because a node may receive a packet intended for some other node and have to pass it along. That requires some buffer memory and some address recognition logic. However, the result is that a single interface definition works over a very wide range of applications, increasing the production volume and lowering the costs for everyone.

Point-to-point signalling is much easier than bus signalling. This, in combination with SCI's low-current single-voltage (48 V) power distribution system, should make ring-connected backplanes very inexpensive. A few printed-circuit-board layers will generally be enough.

Intermediate-level SCI systems may use a combination of switch and ring, using paired SCI interfaces as a simple bridge that connects two rings. Combining many rings one can build a kind of distributed switch fabric. There is active research in progress to discover optimal configurations[15].

SCI defines efficient packet-based protocols that provide the kinds of services one expects from a computer bus. The main differences seen by the user are related to the separation of the request for service from the response. A simple bus waits during a memory read access time, until it gets the data. No other parties can use the bus during that wait. That makes it conceptually easy to handle error conditions or to perform complex mutual-exclusion operations (like read-modify-write).

More-sophisticated buses split the operation into a request and a response phase, just as SCI does. Then the interface has to keep track of pending requests in order to match the responses to them, and a different style of mutual exclusion becomes necessary. Engineers who have already used split-response buses will find SCI to be clean and simple. Those who haven't may face a learning curve.

2.2 Solving the Cache Coherence Problem

Cache memories are important for keeping processors running at full speed, but they introduce some system management complexity. The various caches often contain duplicate copies of data that must be kept consistent with one another, the cache coherence problem. When two processors read the same data (perhaps a synchronization flag) from memory, and cache it (perhaps on-chip), and then one changes the data (perhaps to free a resource that the other is waiting for), somehow the other one has to discover that its cached copy is invalid so that it can be updated with current information.

Buses traditionally solve this problem by taking advantage of their bottleneck: all cache controllers observe all transactions. This is called "snooping," and it works because all transactions are visible to everyone on the bus, and only one can happen at a time. When any controller sees a transaction that changes the validity of cached data, it takes appropriate action to make the data consistent (coherent) again.

SCI solves the coherence problem by using a directory to keep track of which caches have copies of which data. The directory consists of a distributed doubly-linked list with its head pointer kept by the memory controller and the list pointers kept by each cache controller. This structure scales well because there is always sufficient storage available for the directory, no matter what level of sharing occurs, with no pre-allocation required. Performance is good because the maintenance of the list is distributed. Even though these data structures are shared and concurrently updated by multiple processors, the protocols are designed to support concurrent updates without locks.

The cache coherence problem can be avoided if memory is not shared. This approach is used in most of the early multiprocessors, which rely on message passing and explicit interprocessor communication. However, it is often difficult to move computer programs from single-processor machines to message-passing multiprocessors. Note that shared-memory machines can easily do message passing, so they are more versatile. The goal then is to make the coherence protocols efficient and economical.

2.3 Multiprocessor Issues

SCI designers placed a high priority on eliminating several architectural problems that have plagued past multiprocessors, such as deadlocks and livelocks. Deadlocks can occur when two processors request the same two resources in the opposite order, for example. Each processor may get one resource (blocking the other), then become blocked by failure to get the other. Livelock, or starvation, occurs when one processor repeatedly gets access to a shared resource without letting another have its turn for an arbitrarily long time. Generically, these are referred to as "forward progress" issues. In order to guarantee forward progress, each operation should result in some useful work done, so that the system will not consume all its resources performing useless retries.

Experience with multiprocessor systems has shown that they tend to synchronize themselves around problems like these. Even though the designer has estimated the probability of livelock to be extremely small, its effect on the system once it occurs is such that it tends to cause frequent repetitions.

Therefore, the SCI protocol was designed to eliminate dependencies that can cause deadlocks, and to include a simple mechanism that assures fair allocation of resources to avoid livelocks. In a few cases, deadlock avoidance seems to make the protocol less efficient (until one considers the indirect consequences!) but in most cases there was no penalty for a clean protocol.

Of course, software that is outside SCI's scope can still create deadlocks. The SCI designers could only eliminate deadlocks from the underlying protocols.

Efficient mechanisms are included for supporting shared resources, shared data structures and mutual exclusion. Since the old standby read-modify-write is impractical in a parallel-processing environment, SCI took a fresh look at the problem and incorporated several mechanisms that will be helpful.

In a cache-coherent environment, mutual exclusion can be handled by the mechanism that guarantees exclusive use of a cache line by a writer. I.e., the processor can temporarily lock an exclusive cache line while it does any operations it desires, then release it to other readers or writers. However, coherence may not be available in all cases. For example, when accessing (through a bridge) a bus that does not support cache coherence, one may need to tell the bridge to do a read-modify-write.

Therefore SCI defines a set of lock primitives that experience shows to be useful for a variety of purposes: masked swap, compare and swap, and fetch and add. Each of these primitives sends the command and necessary data to a destination device (perhaps a bridge or a memory controller). That device performs the operation indivisibly, then returns the result to the requester. This mechanism works well through switches or other interconnects. (It even works well on buses.)

A particularly interesting use of the swap operations is in the maintenance of shared lists that hold information for an interrupt-servicing processor or commands for a DMA controller (for example). If these lists are properly structured, multiple processors can add items to them while one processor takes items off, without any need for lock variables.

For example, a clean way to handle interrupts would be to associate a particular list with one priority of interrupt service and a specific bit in an interrupt-triggering control register. To request interrupt service, add a work item to the list describing what needs to be done, then set the appropriate bit in the interrupt register. The item in the list corresponds to the interrupt vector in single-processor systems. When the processor is ready to service that priority of interrupts, it takes work items off the list and services them.

This mechanism is clean because all storage allocation is done by the service requester, so there is no danger of the server running out of storage when the work piles up (a possible cause of deadlock). The interrupt bit is simple to implement because it is only a latch, with no FIFO storage or critical timing implied. Setting the bit merely alerts the processor that the list should be checked; the processor can clear the bit as soon as it commits to look at the list.

A common use of mutual exclusion is to grant sequential use of one resource to a series of requesters. This process can generate a large amount of useless interconnect traffic as processors keep updating their cached copies of the shared exclusion variable. SCI defines a Queue on Lock Bit mechanism, QOLB, that uses the linked lists of the cache coherence system to pass the resource efficiently from one processor to the next.

Statistics on memory sharing are controversial, because there are few relevant machines in existence and because the usage patterns on any real machine will evolve to optimize performance on that architecture. However, most researchers agree that unshared data is the most common case, followed by pair-wise shared, followed by multiply-shared. Thus the pair-wise sharing case may be an important one to optimize. SCI defines optional pair-wise sharing optimizations that allow two processors to pass data back and forth without interacting with memory, thus distributing system traffic and reducing activity at the memory controllers.

2.4 Current SCI Status

The IEEE P1596 Scalable Coherent Interface Draft 1.00 officially passed its sponsor ballot in April 1991, with 92% approvals. Revisions to that draft were made subsequently in response to suggestions from the balloting body. These are primarily clarifications, but significant changes were made in the initialization model and in the C code that embodies the detailed specifications. The bit-serial link specifications were significantly improved using material supplied by the Serial HIPPI working group. The revised draft will be recirculated to the balloting body to make sure these revisions have not created new problems. It will also be made available to others at that time. Meanwhile, Draft 1.00 remains conceptually correct in most areas and contains much tutorial information to introduce the SCI concepts and make them understandable.

Final approval of SCI as an IEEE standard is anticipated for December 1991. Supporting chips should be available within a few months. Dolphin Server Technology of Oslo Norway will offer single-chip SCI interfaces that incorporate transceivers, FIFOs and cache coherence support. Several versions are planned, for use as processor interface, memory interface or I/O interface. Hewlett Packard is preparing parallel/serial converter chips to interface the Dolphin chips (parallel) to the 1000 Mbit/s encoding used on optical fiber or coaxial cable, and these should be available by year's end.

3 Related Standards Projects

The following projects are used by SCI or are part of ongoing work related to future SCI developments.

3.1 P1212: Control and Status Register Architecture.

This specification defines the I/O architecture for SCI, Futurebus+ (P896.x) and SerialBus (P1394). Chaired by David V. James, Apple Computer, Cupertino, CA, dvj@apple.com, phone 408-974-1321, fax 408-974-9793. The draft standard was approved and then modified in response to ballot comments, and is now undergoing its second and presumably final recirculation to the balloting body. Final approval by the IEEE Standards Board is expected in December.

3.2 IEEE Std. 1301: Standard for a Metric Equipment Practice for Microcomputers — Coordination Document

An approved standard (June 1991). This specification defines the generic metric modular packaging family used by the SCI module. Chaired by Hans Karlsson, Ericsson Telecom AB, Stockholm, Sweden, phone +46-8-719-6037, fax +46-8 719 8282.

3.3 IEEE Std. 1301.1: Detailed Standard for a Metric Equipment Practice for Microcomputers Using 2 mm Connectors and Convection Cooling

An approved standard (June 1991). This specification details the specific subset of IEEE Std. 1301 that is used by the SCI module. Chaired by Hans Karlsson (see above). The 2 mm connector is presently defined by EIA IS – 64, February 1991, but will become an IEC standard soon. (This connector family is sometimes called Metral, which is DuPont's trademark.)

3.4 P1394: SerialBus

This working group is developing a high speed (10–20 MByte/s) serial bus that can be used for low-cost diagnostics (supporting JTAG in large systems) and I/O. The goal is very low cost ($15 per connection, including cable, connector and interface), for use in consumer products. SCI includes a SerialBus connection in the module power connector. Chaired by Michael Teener, Apple Computer, Santa Clara, CA, phone 408-974-3521, fax 408-985-9893, teener@apple.com. This standard should complete in 1992.

3.5 P1596.1: SCI/VME Bridge

This standard will define a bridge architecture for interfacing VME buses to an SCI node. This will provide I/O support for early SCI systems via VME. Products are likely to be available in 1992. Chaired by Ernst Kristiansen, Dolphin Server Technology, Oslo, Norway, ehk@ifi.uio.no, phone +47-2-627000, fax +47-2-627313.

Work is well underway, but the draft is a bit behind schedule due to other high-priority work that has required our attention. The main decisions involve the mechanism for mapping addresses between VME and SCI, which versions of VME to support, and how to handle interrupts, mutual exclusion, and cache coherence.

Current thinking would place the SCI bridge in the controller positions of four half-length VME backplanes, arranged back to back. This should give excellent performance; dedicating the controller positions seems a reasonable compromise for an I/O system. The bridge may also support transfers from one VME bus to another. We expect to support most of the VME sizes.

3.6 P1596.2: Cache Optimizations for Large Numbers of SCI Processors

This working group is developing request combining, tree-structured coherence directories and fast data distribution mechanisms that may be important for systems with thousands of processors, compatible with the base SCI coherence mechanism. Chaired by Ross Johnson, University of Wisconsin, Madison, WI, ross@cs.wisc.edu, phone 608-262-6617, fax 608-262-9777.

This project is developing and extending ideas that came up during the development of the base SCI standard, but which we felt could be postponed to avoid delaying SCI's introduction. That is, the protocols defined by P1596 seem adequate for systems with perhaps hundreds of processors (enough for a year or so), and include hooks for adding these optimizations later.

Until August, 1991, we thought it would be impractical to maintain binary trees in the cache coherence distributed directory because the overhead would be too high. The schemes we had seen others use were not acceptable for SCI because they required setting lock variables to get mutual exclusion while tree maintenance was done, a violation of an SCI design principle. (Lock variables introduce a variety of complications, such as what to do in case of failure of the processor that should release a lock.)

Thus we considered using approximate or temporary pointers that would form shortcuts along the linear directory lists, but which would gradually become inaccurate as processors rolled out cache lines etc. Whenever a temporary pointer was used it would be checked for validity, and the algorithm would drop back to following the (always valid) linear list when necessary.

But at the August meeting, Ross Johnson presented a method for maintaining correct trees at all times without using lock variables and without adding intolerable performance burdens. Though details need to be worked out and some corner cases need more study, we have the impression that the remaining questions can be resolved. Other issues concern the worst-case scenarios (when things happen in the worst possible sequence) and how to reduce the likelihood of having these scenarios occur.

We are optimistic about request combining. Previous systems that have tried to implement request combining had severe time constraints, trying to combine two requests at the moment they meet in the interconnect. That made the hardware burden large and the system constraints serious. But with SCI's queue structures, it becomes possible to do combining when two combinable requests are found waiting in one queue, a much less time-critical problem. Then combining only takes place when traffic is backed up, i.e. just on those occasions when it is needed.

3.7 P1596.3: Low-Voltage Differential Signals for SCI

This project will specify low-voltage differential signals suitable for high speed communication between CMOS, GaAs and BiCMOS logic arrays used to implement SCI. The object is to enable low-cost CMOS chips to be used for SCI implementations in workstations and personal computers, at speeds of at least 200 MBytes/s. The chairman is Gary Murdock, National Semiconductor, Santa Clara, CA, phone 408-721-7269, fax 408-721-7218.

Faster signalling requires smaller signals, if edge rates and currents are to be kept reasonable. Smaller signals require differential signalling (or at least their own reference independent of system ground). At first glance, differential signalling seems to cost a factor of two in signal traces and pins, but the real cost is much smaller because far fewer ground pins are needed, far less system noise is created (or picked up), and the higher signalling speeds reduce the number of parallel signals needed.

SPICE modelling has been done that shows we can signal at SCI speeds with contemporary CMOS technology. In fact, the hardest part of the problem seems to be how to provide or accept the data at the signalling rate!

Present modelling is based on a 250 mV voltage swing, centered on 1 V. This provides a little headroom for common-mode rejection at the receiver, while allowing use of 5 V, 3.3 V and eventually 2 V technologies.

The working group will choose certain signal levels and rates to be supported as signal interchange (link) standards for CMOS implementations of SCI, and will also define an 8-bit and a 4-bit link to complement the 1596-defined 16-bit and 1-bit links. This work should complete in 1992.

3.8 P1596.4: High-Bandwidth Memory Chip Interface

This project is defining an interface that will permit access to the large internal bandwidth available inside dynamic memory chips. The goal is to increase the performance and reduce the complexity of memory systems by using SCI signalling technology and a subset of the SCI protocols. The chairman is Hans Wiggers, Hewlett Packard Laboratories, Palo Alto, CA, wiggers@hplabs.hp.com, phone 415-857-2433, fax 415-857-3991.

This working group has met several times, with good attendance (including several experienced memory-system designers) and high industry interest. A serious problem with present memory systems is the need to use a large number of memory chips in parallel banks to get the bandwidth needed for today's powerful microprocessors. As the capacity per chip increases, the smallest memory configuration with adequate bandwidth reaches a point where it has an unreasonably large capacity (and needlessly high cost). Furthermore, the increments for expansion are too large.

We hope to get much higher bandwidth from far fewer chips by using SCI-like signalling technology. This will lower the entry cost for low-end systems while raising the performance of high-end systems.

Several models are being considered. One of the most promising uses several RAM chip ringlets attached to a single controller by 8-bit-wide point-to-point links. The name RamLink is becoming popular for this approach. Details of the signalling are still being worked out, but there seems to be general agreement to use small signal voltages. Whether the signals will be single-ended or differential will depend on the link lengths, packages and connectors to be supported. SPICE modelling is underway.

Other issues include whether to perform ECC correction in each RAM chip or in the controller. Doing it in the RAM chip leaves the details up to the vendor, with possible future technology improvements being incorporated transparently. Doing it in the controller is probably the lowest initial cost and gives the system designer the most control, but has a performance penalty for transmitting the extra information on the links.

Another open question is the link protocol to be used. If an SCI-like bypass FIFO can be incorporated on the RAM chips, we can use a simplified version of the present SCI protocols. If not, we must define a scheduling mechanism that prevents two packets from being transmitted at once. Scheduling is complicated by predictability issues, such as the effects of refresh or ECC soft-error correction on the RAM chip.

3.9 P1596.5: Shared-Data Formats Optimized for SCI

This project will specify data formats for efficiently exchanging data between byte-addressable processors on SCI. The specification will define integer and floating-point sizes, formats, and address-alignment constraints. Bit fields will be supported as subcomponents of a larger byte-addressable integer datum. Bit fields may occupy between 1 and N bits of contiguous memory storage where N is the size of the containing data object.

The SCI standard supports efficient data transfers between heterogeneous workstations within a distributed computing environment. Current systems require conversions among large numbers of vendor- or language-dependent data formats; specifying a single transfer format greatly reduces the complexity of this conversion problem. In addition to simplifying the data-interchange problem, standard data formats provide a framework for the design of future processor instruction sets and language data types. Chaired by David V. James, Apple Computer, Cupertino, CA, dvj@apple.com, phone 408-974-1321, fax 408-974-9793.

Work on this project has just begun, but it should not take long to complete, since much of the groundwork has been done earlier in conjunction with P1212 and other projects.

4 Future Plans

As technology advances, SCI will need to define new link standards. For example, present fiber optic technology is currently expensive at 1000 Mbit/s and prohibitive at higher rates. Yet this situation is changing rapidly, and SCI applications in HDTV would be greatly helped by a bit rate at least double this. We will monitor progress in this area.

Similarly, there are enormous application possibilities for slower links. An eight-bit-wide link operating at 250 MByte/s or 500 MByte/s might be about right for the next generation personal computers. Perhaps it will be appropriate soon to define a personal-computer form factor and signal standard for SCI, based on new CMOS chips or processors with integrated SCI interfaces.

While designing SCI we learned a lot about how the processor should interact with the interconnect. We are considering how best to spread this information to processor designers, to make multiprocessor systems more efficient. Possibly this could be a recommended practice, or even a standard clean 64-bit RISC architecture optimized for use with SCI.

We have in mind two projects for bridges to Futurebus+ that are currently waiting for the right time to start. One is a very simple bridge to Profile B, an I/O bus with no cache coherence. The other is a general symmetric bridge that includes cache coherence.

5 Conclusion

The base SCI standard covering the physical signalling, logical protocols, and cache coherence mechanism should be approved by the IEEE Standards Board in December 1991. The first commercial implementor (Dolphin Server

Technology, Oslo, Norway) expects to have working prototypes within a few months of the standard's approval, and has promised to make the interface chips available to others.

SCI's performance seems such a large step ahead of the current state of the art in computer buses that it has some difficulty in appearing credible. The best answer to these doubts will be the existence of working silicon, available at reasonable prices, being used in working systems.

The complexity of SCI is approximately the same as that of a split-cycle bus system (like the VAX BI or Futurebus+ or Fastbus with Buffered Interconnects) in small applications, and is much less than that of a bus system for large applications. Nevertheless, relatively few designers have experience with split-cycle bus design issues and therefore there will be some need for training as they move to SCI.

SCI has no evident competition in terms of open systems that could hope to deal with the massive computation needs for the next generation of data acquisition, analysis, and general computation.

For details, or to participate in this work, please contact the author:

David B. Gustavson, IEEE P1596 Chairman
Computation Research Group, MS 88
Stanford Linear Accelerator Center
Stanford, CA 94309 U. S. A.
tel: (415) 926-2863 fax: (415) 926-3329
Email: dbg@slacvm.slac.stanford.edu

6 Bibliography and References

1. D. B. Gustavson, "IEEE P1596, A Scalable Coherent Interface for Gigabyte/s Multiprocessor Applications", Nuclear Science Symposium, Orlando, Florida, November 9–11, 1988.
2. D. V. James, "Scalable I/O Architecture for Buses", COMPCON Spring 1989, San Francisco, CA, Feb. 27–March 3, 1989.
3. D. B. Gustavson, "Scalable Coherent Interface", COMPCON Spring 1989, San Francisco, CA, February 27–March 3, 1989.
4. E. H. Kristiansen, K. Alnes, B. O. Bakka and M. Jenssen, "Scalable Coherent Interface", Eurobus Munich, May 1989.
5. P. Sweazey, "Cache Coherence on SCI", IEEE/ACM Computer Architecture Workshop, Eilat, Israel, June 1989.
6. S. Gjessing, S. Krogdahl, E. Munthe-Kaas, "Formal Specification and Verification of SCI Cache Coherence", NIK89, Stavenger, Norway, November 1989. Also available as Informatics Research Report No. 142, University of Oslo, 1990.
7. E. H. Kristiansen, "Scalable Coherent Interface", New Backplane Bus Architectures, pp 67–75, CERN CN/90/4, March 22–23, 1990.
8. J. E. Smith and J. R. Goodman, "Restricted Fetch&Φ Operations for Parallel Processing, by Gurindar S. Sohi," Computer Sciences Technical Report #922, University of Wisconsin – Madison, March 1990.
9. K. Alnes, E. H. Kristiansen, D. B. Gustavson, D. V. James, "Scalable Coherent Interface", CompEuro 90, Tel Aviv, Israel, May 1990.
10. D. V. James, A. T. Laundrie, S. Gjessing, G. Sohi "New Directions in Scalable Shared-Memory Multiprocessor Architectures: Scalable Coherent Interface," *Computer* Vol. 23, No. 6, June 1990, pp. 74-77.
11. S. Gjessing, S. Krogdahl, E. Munthe-Kaas, "Approaching Verification of the SCI Cache Coherence Protocol", Informatics Research Report No. 145, University of Oslo, August 1990.
12. "SCI – Scalable Coherent Interface, P1596/D1.00 23Jan91," Draft for Sponsor Ballot Review. Prepared by the P1596 Working Group of the Microprocessor Standards Committee. IEEE, New York, N.Y., 1991.
13. S. L. Scott, "A Cache Coherence Mechanism for Scalable, Shared-Memory Multiprocessors," Computer Sciences Technical Report #1002, University of Wisconsin – Madison, February 1991.
14. P. J. Woest and J. R. Goodman, "An Analysis of Synchronization Mechanisms in Shared-Memory Multiprocessors," Computer Sciences Technical Report #1005, University of Wisconsin – Madison, February 1991.
15. S. L. Scott and J. R. Goodman, "Performance of Pipelined K-ary N-cube Networks," Computer Sciences Technical Report #1010, University of Wisconsin – Madison, February 1991. This paper shows that when pipelining is permitted, as is the case for SCI, one gains performance rapidly relative to synchronous systems by increasing the dimensionality of the interconnect, and the resulting performance is much higher than for synchronous systems.
16. S. Gjessing, E. Munthe-Kaas, "Formal Specification of Cache Coherence in a Shared Memory Multiprocessor," Informatics Research Report, University of Oslo, November 1991.