

IEEE Standard for High-Bandwidth Memory Interface Based on Scalable Coherent Interface (SCI) Signaling Technology (RamLink)

Sponsor

**Microprocessor and Microcomputer Standards Committee
of the
IEEE Computer Society**

Approved 19 March 1996

IEEE Standards Board

Abstract: A high-bandwidth interface optimized for interchanging data between a memory controller and one or more dynamic RAMs is specified. RamLink is an applicable interface for other RAM-like devices as well.

Keywords: dynamic RAMs, high-bandwidth, interface, RAM, RamLink

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1996 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1996. Printed in the United States of America.

ISBN 1-55937-745-3

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying all patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (508) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

[This introduction is not a part of IEEE Std 1596.4-1996, IEEE Standard for High-Bandwidth Memory Interface Based on Scalable Coherent Interface (SCI) Signaling Technology (RamLink).]

RamLink began as an application of the Scalable Coherent Interface (SCI) Local Area Multiprocessor (LAMP) technology that was intended merely to increase the bandwidth of a memory IC's package and reduce the layout constraints of bused memory arrays.

However, it soon became clear that the same bus physics problems that engendered LAMP and guided RamLink were becoming serious obstacles in microprocessor I/O systems as well.

This provided motivation to generalize RamLink beyond mere memory applications, so that it could also serve as a very high performance, low pin-count, and physically extended microprocessor I/O bus.

Although SCI was itself designed for such tasks, among others, its simplicity was obscured by defining its cache coherence protocols in the same standard as its packet and signaling protocols, causing its acceptance to be delayed. Furthermore, RamLink is even simpler than SCI, because it does not attempt to handle multiprocessor issues such as efficient locks, deadlocks, and starvation.

It proved straightforward to provide a minimal set of features in RamLink that allow it to serve as a link from the microprocessor to an SCI/LAMP bridge, providing an attractive future upgrade path. Though the performance of such an indirect connection to the LAMP will be well below that of a well-integrated microprocessor/SCI/LAMP design, it could provide several times the performance of I/O buses such as today's popular PCI, and at lower cost (assuming similar production volumes).

As microprocessor speeds increase, the constraints on bus speeds, lengths, and layout become increasingly painful. Thus an attractive incremental growth path would be the addition of multiple RamLink ringlets to the microprocessor. In addition to memory, these could support high performance bridges to the usual I/O buses, and low-latency connections to a Local Area MultiProcessor network based on SCI.

Later, as users and software become accustomed to low-latency networking and clustering, performance could be increased another large step by adding a direct SCI connection to the processor/cache controller. This upgrade path allows users to adapt in stages to the radically falling network latency: from today's usual 1000 microseconds to tens of microseconds, using an indirect RamLink connection to LAMP; and then to the few-or-sub-microsecond range with a direct connection.

RamLink ringlets can bridge SyncLink memory-array buses as well as to the usual I/O buses, so the designer is free to choose the hybrid mixture of bused connections and point-to-point connections that seems most appropriate to the intended market and application.

These considerations have resulted in introducing the generalized RamLink earlier than necessary for memory applications, but at about the right time for processor I/O applications.

Committee membership

The following is a list of participants in the IEEE Std 1596.4-1996 Working Group. Voting members at the time of publication are marked with an asterisk (*).

Hans A. M. Wiggers, Chair*
David B. Gustavson, Vice Chair*
David V. James, Technical Editor*
Glen D. Stone, Secretary*

Romain Agostini*
Duane Anderson*
David Barrett*
Louis Berube*
Kyle Black
Gene Bowman
Haakon Bugge
Vince Cavanna
Del Cecchi*
Kurt Chan
Mike Chastain*
Wei Chen
Peter Cheng*
Jeff Childs
James Chow
Diane Coon
Bob Cooper
Adrian Cosoroaba
Richard Crisp*
Ron Daniel
Martin Davis
Tony DeGroot
Frank Denap
Vic Desai
Ira Deyhimy
Martin Freeman*
Richard Fryer*
Parviz Ghaffaripour
Stein Gjessing*
Jim Goodman*
Hartesh Grewal
Ronald Haigh
Jeff Haight
Del Hanson
William Hardell
Dave Harper
Mats Hedberg
Lyndon Herzberg
William Hiles
Syed Huq
Bret Indrelee*
Suren Irukulla
Ed Jacques
David James*
Sverre Johansen*

Eric Johnson
Ross Johnson*
Anatol Kaganovich*
Sam Karunanithi
Prabhas Kejriwal
Stephen Kempainen*
Khan Kibria*
Todd Kline*
Mike Koster
Raj Kumar
Jean-Pierre Lachance
Ralph Lachenmaier*
Mark Lane
Larry Larsen*
Oystein Larsen
Gary Lesmeister
Michael LeVine
Lloyd Lewins*
Qiang Li*
Volker Lindenstruth
B. Mitchell Loebel*
Tom Lovett
Roland Marbot*
Don May
Dave McMullen
Rusty Meier
Ben Milander
Resa Mirkhani
Jeff Mitchell
Pete Moore
Jim Morgan
Hans Mueller*
Gary Murdock
Wayne Nation
James Nelson
Larry Nevin
Ron Nordin
Shigeo Oshima
Farrel Ostler
C. N. Patel
Goeffrey Peck
Mike Polacek
Kendall Pope
Betty Prince*
Steven Przybylski*

Narayan Purohit
John Radford
Frank Ramsay
Alasdair Rawsthorne
Livio Ricciulli
Bill Rivard
Knut Roberg
Michael Rodbell
Edward Rodriguez
Luchi Roth*
Gil Russell
Robert Safranek
Wayne Sanderson
Kotobu Sato
Gene Schramm
Tim Scott*
Ed Seim*
John Shaw
Fred Shirley
Ken Smits
Bjorn Solberg
Joanne Spiller*
Vinnie Squitieri
Brad Stewart*
Glen D. Stone*
Tim Strayer
Paul Sweazey
Jay Swenson
Farhad Tabrizi
Richard Talbott
Bill Terrell
Manu Thapar
Henry Tominaga
Jim Tong
K. C. Tsai
Russ Tuck
Bill Vogley*
Phil Vukovic
Roger Weiss
Jeff West
Gary Wray
Bin Wu
Danny Yeung*
Frederick Zeise
Mike Ziegler*

The following persons were on the balloting committee that approved this standard for submission to the IEEE Standards Board:

Duane L. Anderson	John Hyde	L. M. Patnaik
Keith D. Anthony	David V. James	Mira Pauker
David Bartek	Ernst H. Kristiansen	Betty Prince
Harrison A. Beasley	Thomas M. Kurihara	Steven Przybylski
John C. Brightwell	Gerry Laws	Steve Quinton
Charles Brill	Linser Rollins	Brian Ramelson
Håkon Ording Bugge	Stephen Mabbs	Fred U. Rosenberger
Dante Del Corso	Thanos Mentzelopoulos	John Rynearson
Sourav K. Dutta	Bruce Millard	Donald Senzig
Gordon Force	Klaus Dieter Müller	Manu Thapar
Martin Freeman	Mitsunori Nakata	Thomas W. Vrankar
Stein Gjessing	Wayne Nation	Michael H. Wenzel
Julio Gonzalez Sanz	J. D. Nicoud	Hans A. Wiggers
James Goodman	Tadahiko Nishimukai	Robert J. Wood
John Griffith	Gregory C. Novak	David L. Wright
David B. Gustavson	Elwood Parsons	Bin Wu

When the IEEE Standards Board approved this standard on 19 March 1996, it had the following membership:

Donald C. Loughry, *Chair*
Richard J. Holleman, *Vice Chair*
Andrew G. Salem, *Secretary*

Gilles A. Baril	Ben C. Johnson	Arthur K. Reilly
Clyde R. Camp	E. G. “Al” Kiener	Ronald H. Reimer
Joseph A. Cannatelli	Joseph L. Koepfinger*	Gary S. Robinson
Stephen L. Diamond	Lawrence V. McCall	Ingo Rüsçh
Harold E. Epstein	L. Bruce McClung	John S. Ryan
Donald C. Fleckenstein	Marco W. Migliaro	Chee Kiow Tan
Jay Forster*	Mary Lou Padgett	Leonard L. Tripp
Donald N. Heirman	John W. Pope	Howard L. Wolfman
	Jose R. Ramos	

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal	Alan H. Cookson	Chester C. Taylor
	Lisa S. Young	
	<i>IEEE Standards Project Editor</i>	

CLAUSE	PAGE
1. Overview	1
1.1 Scope	1
1.2 Purpose	2
1.3 Document structure	2
1.4 Objectives	3
1.5 Expected applications	4
2. References	7
3. Definitions	7
3.1 Conformance levels	7
3.2 Definitions of RAM and interconnect-related terms	7
3.3 Bit and byte ordering within packets	10
3.4 Bit and byte ordering within CSRs	11
3.5 Field notation (italics and bold usage)	11
3.6 Numerical values	11
3.7 Signaling layers	12
4. RamLink configurations	13
4.1 Simple topologies	13
4.2 Hierarchical topologies	13
4.3 Hybrid signaling	14
4.4 Wide (9-bit) data transfers	15
4.5 DRAM error-checking options	16
4.6 Link error-checking	18
5. RamLink operation	19
5.1 Split-response transactions	19
5.2 Response scheduling	20
5.3 Retried transactions	20
5.4 RamLink address space	20
5.5 Address errors	21
5.6 Request queue sizes	22
5.7 Request ordering	22
5.8 Refresh operations	23
5.9 Rate adjustments	24
6. Packet formats	25
6.1 Packet components	25
6.2 Request packet formats	29
6.3 Copy packet	31
6.4 Event packets	32
6.5 Response packets	35
6.6 Retry packet format	37
6.7 Idle packets	37
6.8 Special initialization packets	38

CLAUSE	PAGE
7. RamLink initialization	39
7.1 standBy mode.....	39
7.2 shutDown mode	39
7.3 shutOff recovery	40
7.4 Self test and initialize	41
Annex A (informative) Bibliography.....	42
Annex B (normative) RingLink signaling	43
Annex C (informative) SyncLink signals	51
Annex D (normative) Control and status	56
Annex E (normative) I/O extensions	67

IEEE Standard for High-Bandwidth Memory Interface Based on Scalable Coherent Interface (SCI) Signaling Technology (RamLink)

1. Overview

1.1 Scope

This document represents a simplification of the concepts developed within IEEE Std 1596-1992, Scalable Coherent Interface (SCI) [B4].¹ It specifies an SCI-like interface for cost-effective communication between a memory controller and one or more dynamic memory chips. Rather than using the formal document title, *High-Bandwidth Memory Interface Based on Scalable Coherent Interface (SCI) Signaling Technology*, the informal term *RamLink* is used to refer to this standard.

This standard defines a high-bandwidth interface that permits access to the large internal bandwidth already available in dynamic memory chips. The goal is to increase the performance and reduce the complexity of memory systems by using a subset of the SCI protocols. Hierarchical memory systems will be considered, from multilevel caches to main-memory systems. The interface specification will apply to individual memory chips as well as their controllers. The interface should be applicable to commodity parts that will fulfill the requirements of near-future and subsequent generations of computer systems.

The main scope of the work is aimed at establishing the RamLink logical protocols. Distinct signal-layer (physical and electrical specification) options allow the RamLink protocols to be optimally utilized within distinct applications. A signal-layer specification called RingLink supports point-to-point ringlet-based topologies. An alternative signaling model, called SyncLink, supports localized bus-based connections; the functionality of SyncLink signals is also described, but the detailed SyncLink electrical specifications have been deferred for future extensions of this standard.

Although optimized for memory systems, RamLink recognizes that processor-memory buses may also interface to the I/O system. RamLink supports such I/O interfaces, and it provides a minimal, yet sufficient, set of operations to support I/O devices as well as DRAMs.

¹Numbers in brackets preceded by the letter B correspond to those of the bibliography in annex A.

1.2 Purpose

The performance of microprocessors doubles every two to three years. The density of main-memory components used within these systems also doubles, but memory-access bandwidth has lagged behind.

To meet their memory-access bandwidth requirements, designers currently build complex interleaved memory systems that use many additional components. Application of SCI concepts will break through this barrier, which already impacts the performance of high-end workstations, as well as the entry cost for low-end computers.

When lower-density DRAM chips were 1 bit wide, additional memory-access bandwidth was achieved by connecting multiple DRAMs in parallel, so that a 16-bit, 32-bit, or even 64-bit word could be obtained in a single memory-access cycle. As memory densities have increased, the need to access a video image every refresh cycle has been met by increasing the memory-chip interface widths to 4-bit, 8-bit, 16-bit, and even 32-bit widths. A single chip will soon provide sufficient storage for holding a video image or an operating system, if the access-bandwidth requirements could be met.

By increasing the memory bandwidths, RamLink is expected to decrease the costs of small systems, where the minimum number of memories is driven by the system's bandwidth (not storage size) requirements. The RamLink protocols are intended to be technology independent, in that the assumed memory controller could be integrated with the processor on small systems, or (on large systems) could efficiently interface to a variety of system buses.

1.3 Document structure

The RamLink specification is partitioned into clauses that serve several distinct purposes.

Clause 1: Overview provides background for understanding the goals and objectives of this document.

Clause 2: References lists the references used in the remainder of this document.

Clause 3: Definitions and notation describes the terminology and notation used in the remainder of this document.

Clause 4: RamLink configurations describes various RamLink topologies, including hierarchical topologies and complementary RamLink signaling standards.

Clause 5: RamLink operation describes the various RamLink operations, including split-response transactions and addressing errors.

Clause 6: Packet formats specifies the formats of RamLink packets.

Clause 7: RamLink initialization specifies protocols and timing for power-on and reduced-power modes of operation (shutdown and standby).

The annexes are applicable to specific subclasses of RamLink devices and include the following:

Annex A: Bibliography provides an informative listing of standards that complement the RamLink document, IEEE Std 1596.4-1996.

Annex B: RingLink signaling specifies a RamLink physical signaling standard, consisting of point-to-point connections between nodes in a ring, which is optimized for longer-distance or through-connector communications.

Annex C: SyncLink signaling describes a RamLink physical signaling standard, consisting of shared input and output links, which is optimized for shorter-distance PC-board-local communications.

Annex D: Control and status specifies the RamLink addressing model, the supported transaction set, the (optional) control-and-status registers, and configuration-ROM formats.

Annex E: I/O extensions specifies how simple RamLink I/O devices can be supported. The (optional) interrupt mechanism and globally-synchronized time-of-day registers are specified. A simple bus bridge architecture is specified as well.

1.4 Objectives

RamLink is intended to be a primary system-level interconnect standard that is well suited to memories. The following objectives were assumed or developed during development of this standard, and have heavily influenced the design options that were selected.

- *Commodity parts.* The RamLink interface should be applicable to a majority of future DRAM applications.
- *Predictable timing.* The RamLink interface should allow memory controllers to schedule the responses for multiple concurrently active requests.
- *Controller flexibility.* The RamLink interface should support simple controllers (which are unaware of the slave access-time parameters), as well as high-performance controllers (which have detailed knowledge of slave access-time parameters).
- *Device flexibility.* In addition to supporting DRAMs, the RamLink interface should support other RAM-like components, including (but not limited to) ROM and FLASH memories. RamLink should also support high-bandwidth I/O devices, if the I/O is designed to emulate the characteristics of RAM-chip components.
- *Bridge feasibility.* RamLink should be capable of supporting bridges to other interconnect systems, such as SCI.

It should be emphasized that the primary goal of RamLink is to support low-cost commodity RAM parts. The support of other devices should be consistent with this goal; thus, device and bridge interfaces should be sufficient, but may be suboptimal.

To meet these objectives, the following design strategies were used:

- *Slave simplicity.* Whenever possible, complexity should be located in the controller rather than in the RamLink slaves.
- *Hierarchical structures.* For simplicity, RamLink protocols limit the number of directly attached devices. To support larger numbers of attached devices, a large extendable address space allows each of these devices to be a controller for other, indirectly attached, RAMs.
- *Physical layer independence.* The RamLink standard defines logical protocols that can be readily implemented on either RingLink or SyncLink signaling technologies. RingLink's point-to-point signaling is robust and layout insensitive; SyncLink's bused signaling has higher performance but severe physical-layout constraints.

The scope of the RamLink definition is consistent with these strategies, leveraging physical layers and register architectures developed by others, described as follows:

- *Protocols.* The RamLink interface specifies protocols used to communicate between a controller and one or more memory devices, in terms of the packets transferred between these components.
- *Slave-provided CSRs.* The RamLink interface defines a limited set of control commands. Other special capabilities (autoconfiguration ROM and specialized control) are supported by control-and-status registers.
- *Electrical signals.* The RamLink interface specifies signals used to communicate between the controller and one or more memory slaves. However, other standards are referenced for detailed signal levels and timing characteristics.
- *Physical packaging.* The RamLink interface does not specify physical packaging requirements. Other standards groups, such as JEDEC, the semiconductor engineering standardization body of the Electronic

Industries Association (EIA), are expected to define physical-packaging standards based on market requirements.

1.5 Expected applications

To broaden the application base, the RamLink design is primarily intended to be a cost-effective high-bandwidth generic RAM interface. Specific configurations that were considered to test the applicability and performance of the protocols are discussed in 1.5.1, 1.5.2, 1.5.3, and 1.5.4.

1.5.1 Bulk memory controller

One application of RamLink technology is to connect the memory controller to large numbers of inexpensive RAMs, as illustrated in figure 1. The attachment points for RamLink connections are illustrated; these depend on the signaling standard (RamLink or SyncLink) that is used. In both cases, each slave has one input link and one output link and the data flows in one direction (counter-rotating rings, as might be inferred from the illustration, are not used). See annex B and annex C for details.

Within such applications, multiple ringlets are expected to improve the total available bandwidth (for large transfers), to reduce the average latency (for small, interleaved transfers), or to improve the system reliability.

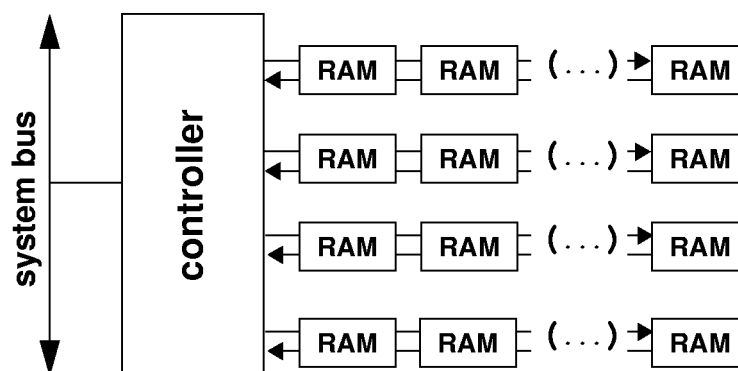


Figure 1— Multi-link controller

Reliable systems are expected to store redundant data copies, so that the data can be recovered after one of the ringlets fails. Standard redundant arrays of independent disks (RAID) techniques can be applied to such memory topologies. For example, a memory controller might support five ringlets: four would be used for saving interleaved copies of the data, and the fifth would be used to store a parity-check copy.

Traffic patterns on such systems are hard to predict. However, several likely processor-to-memory access patterns were considered. An uncached processor is expected to generate word accesses (a word is either four or eight bytes), with approximately three times as many reads as writes.

A cached processor is expected to generate cache-line accesses, which are here presumed to be 64 bytes. With a write-back cache, there are approximately three times as many reads as writes. All transactions transfer a cache line of data (64 bytes).

For a write-through cache, there are approximately three times as many writes as reads. The read transactions fetch a cache line of data (64 bytes), but the write transactions are the size of the accessed data (typically four or eight bytes). These memory-traffic models are illustrated in figure 2.

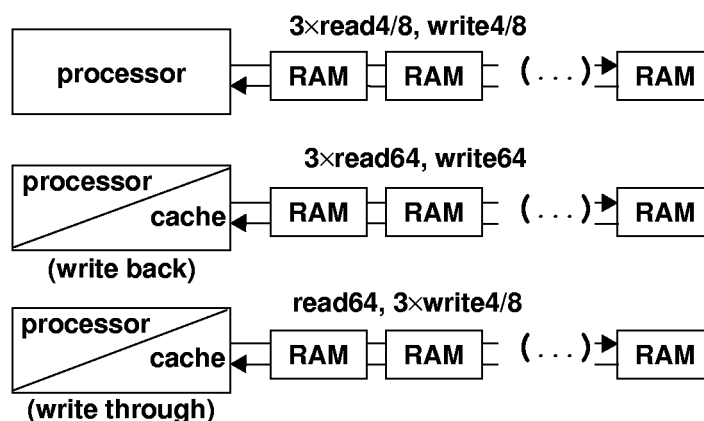


Figure 2— Memory-access traffic models

1.5.2 Preconfigured uniprocessor

RamLink technology could also be useful for connecting a processor to its RAM and ROM. The high bandwidth of RamLink, coupled with the constantly increasing capacities of RAM, makes it possible to construct computers with few components, as illustrated in figure 3. For example, a single 64 Mbit RAM will soon be sufficient to hold a $1\text{ K} \times 1\text{ K}$ 32 bits/pixel image with 4 Mbytes remaining for the operating-system and application code.

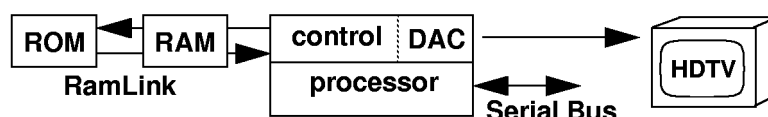


Figure 3— A preconfigured uniprocessor

Such applications show the need to support more than DRAM slaves. Other memory technologies, such as ROM or FLASH, are needed on low-cost systems; it is desirable to attach these to the same RamLink interface.

1.5.3 Extendable uniprocessor

An extendable system would include memory, disk and an I/O interface device, as illustrated in figure 4. The memory device may control several memory components such as synchronous DRAMs. The disk and the I/O can request attention from the processor by using RamLink's optional interrupt mechanism. The processor performs transfers from the disk (or I/O) buffers to the DRAM storage using the RamLink read and write transactions. The interrupt mechanism is described E.2.2, E.3.4, and E.3.5.

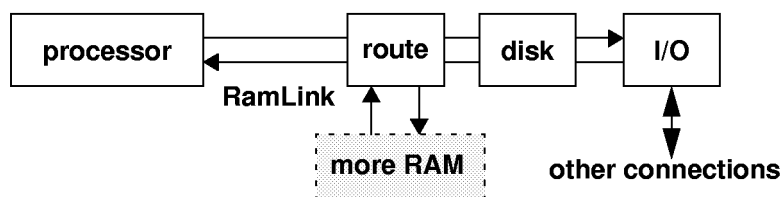


Figure 4— Extendable uniprocessor system

The intent is to support a wide range of I/O devices (when supported) without impacting the costs of memory-only configurations. Data transfers between direct memory access (DMA) capable I/O devices are performed by activating the controller (an alarm is used for this purpose), which performs the data transfer on the I/O device's behalf. This provides the appearance, but not the complexity, of multiple-master bus arbitration capabilities.

The costs of supporting I/O protocols are borne by the I/O device (which sends the alarm and provides split-response transaction queues) and supportive controllers (which copy subactions between transaction queues). The designs of memory chips and simple controllers (which support only memory-chip slaves) are not affected (see annex EE).

1.5.4 Extendable multiprocessor

As the cost of processing continues to decline, the processor's memory interface becomes a significant portion of the overall system costs. The trend toward high-pinout packages (exceeding 400 pins) is incompatible with the primary objectives of components in multiprocessor systems, which are low-cost and low-power.

Thus, RamLink may be a useful interface for connecting the processor to local RAM chips. Some of these local RAM chips could be mapped to node-local memory addresses and others could cache remotely resident data, as illustrated in figure 5.

The bridge chip is of particular interest in this application. An SCI bridge chip would contain all the SCI-specific protocols and the request and response queues. Actual transfers of data blocks from SCI to the memory would be routed through the processor/memory controller, allowing remote accesses to be consistent with the processor's cache.

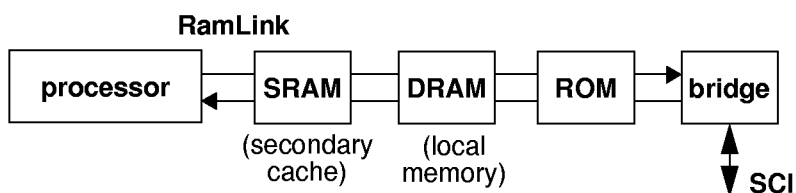


Figure 5— Extendable multiprocessor

Interfaces to uncached I/O buses, such as Peripheral Component Interconnect (PCI), have I/O-device-like queues. Data transfers from the bridge into processor memory are performed by activating the controller, which performs the data transfer on the bridge's behalf. Additional split-response queues allow the processor to access the (potentially higher latency) I/O bus addresses without complicating the scheduling of (lower latency) RamLink-local transactions (see E.4.3 for details).

It is also possible to support the SCI multiprocessor coherence protocols. To do this, some of the local RAM would be allocated for extra tag-storage, and concurrent tag-update protocols would be supported by the processor and bridge. The support of coherent multiprocessor bridges is beyond the scope of this standard.

2. References

The following documents shall be used in conjunction with this standard:

IEEE Std 1596.3-1996, IEEE Standard for Low-Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI).²

ISO/IEC 9899: 1990: Programming Languages—C. ³

ISO/IEC 13213: 1994 [ANSI/IEEE Std 1212, 1994 Edition] Information technology—Microprocessor systems—Control and Status Register (CSR) Architecture for microcomputer buses.⁴

3. Definitions

3.1 Conformance levels

Several key words are used to differentiate among different levels of requirements and options, as follows:

3.1.1 expected: A keyword used to describe the behavior of the hardware or software in the design models assumed by this standard. Other hardware and software design models may also be implemented.

3.1.2 may: A keyword that indicates flexibility of choice with no implied preference.

3.1.3 shall: A keyword indicating a mandatory requirement. Designers are *required* to implement all such mandatory requirements to ensure interoperability with other standard-conformant products.

3.1.4 should: A keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “it is recommended.”

3.2 Definitions of RAM and interconnect-related terms

A large number of RAM and interconnect-related technical terms are used in this document. These terms are described below:

3.2.1 addressed refresh: A RAM-refresh protocol, in which the controller-provided read0-requests schedule the timing and specify addresses for RAM refresh cycles.

3.2.2 agent: A switch or switch-like component between a RamLink controller and a RamLink slave. During normal operation, the agent has two behaviors: from the higher-level controller's perspective, the agent behaves like a RamLink slave, and from the lower-level slave's perspective, the agent behaves like a controller.

²IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

³This standard is a revision and redesignation of ANSI X3.159-1989.

⁴This standard is available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA, or from the International Electrotechnical Commission, 3 rue de Varembé, Case Postale 131, CH 1211, Genève 20, Switzerland/Suisse.

3.2.3 alarm: A signal generated by a slave and received by the controller, which is typically used to interrupt the processor, or to activate processing of the slave's request/response packet queues.

3.2.4 autorefresh: A RAM-refresh protocol in which controller-provided *refreshNow* signals schedule the timing for RAM refresh cycles and RAM-local hardware specifies refresh-cycle addresses. *See: refresh.*

3.2.5 bounded scheduling: A scheduling algorithm used by a simple controller. Rather than specifying the exact time for a slave to return its response, a time window is provided where the time window is longer than the expected request-processing delay.

3.2.6 bridge: An interface between heterogeneous memory-mapped buses. Due to the distinct capabilities of the attached buses, a bridge is expected to perform address translations, fragmentation (one transaction is decomposed into several smaller transactions), concatenation (multiple writes are combined into one), or prefetching (one read initiates speculative reads of likely-to-be-used addresses).

3.2.7 byte: A set of eight signals or bits taken as a unit.

3.2.8 cache line: Often simply called a “line,” the block of memory (sometimes called a sector) that is managed as a unit for coherence purposes; i.e., cache tags are maintained on a per-line basis. Although the SCI line size influenced the RamLink packet sizes, coherence protocols are beyond the scope of this standard.

3.2.9 controller: The entity that initiates RamLink transactions. There is exactly one controller on each RamLink ringlet.

3.2.10 CSR Architecture: ISO/IEC 13213: 1994 [ANSI/IEEE Std 1212, 1994 Edition] Information technology—Microprocessor systems—Control and Status Register (CSR) Architecture for microcomputer buses.⁵

3.2.11 directed transaction: A transaction that is processed by one and only one responder. The read and write transactions are always directed. The event transactions may be directed or broadcast.

3.2.12 doublet: Two bytes of data.

3.2.13 ECC: *See: error-correction coding.*

3.2.14 error-correction coding (ECC): An encoding of data and redundant check bits that enables decoding hardware to reconstruct the original data in the presence of a data-bit or check-bit error.

3.2.15 EDC: *See: error-detection coding.*

3.2.16 error-detection coding (EDC): An encoding of data and redundant check bits, such that in the presence of a data-bit or check-bit error decoding hardware can detect the error, but cannot reconstruct the original data.

3.2.17 event, event command: A command contained within an *event packet*.

3.2.18 event packet: A short, four-byte packet containing an event command that is directed to one slave or broadcast to all. Device state is affected by the event command, but no response is returned to the controller.

3.2.19 exact scheduling: A scheduling algorithm used by the controller to predict the time delay needed by a slave to generate response packets. The controller reserves a time slot for the response, where the time slot is the exact size needed and is at the precise time for the expected response packet.

3.2.20 extended packet: A version of a command *packet* that is generated by a controller to initiate a directed transaction with a specified slave.

3.2.21 idle packet: Four consecutive null bytes, which usually contain no information. Idle packets are used to fill the space between *RamLink* packets. They may be used to convey an interrupt request.

3.2.22 initialization packet: Special packets that are only generated by the controller during the RamLink initialization process. There are three types of initialization packets: *sync*, *wait*, and *wake*.

3.2.23 JTAG: An abbreviation for Joint Test Activity Group that is used to describe the serial diagnostic signals that have been defined by this group.

⁵Information on references can be found in clause 2.

3.2.24 line: See: **cache line**.

3.2.25 Mbit: A synonym for 2^{20} bits.

3.2.26 Mbyte: A synonym for 2^{20} bytes.

3.2.27 node: A term used to describe a RamLink slave within the context of the CSR Architecture. The entity associated with a particular set of control-register addresses (including identification ROM and reset-command registers). In normal operation each node can be accessed independently (a control-register update on one node has no effect on the control registers of another node).

3.2.28 octlet: Eight bytes of data. Not to be confused with an octet, which has been commonly used to describe eight bits of data. In this document, the term byte, rather than octet, is used to describe eight bits of data.

3.2.29 packet: A block of information that is transmitted within a single transfer operation.

3.2.30 quadlet: Four bytes of data.

3.2.31 RamLink: The packet-transfer architecture portion of this standard, which describes how packets are transferred between the master and slaves. Several signal-layer specifications can be used, including *RingLink* and *SyncLink*.

3.2.32 refresh: A periodic referencing of all storage locations in a DRAM, which typically recharges data-storage capacitors in order to maintain data integrity.

3.2.33 refresh period: Applicable to RAM that is in the autorefresh mode. The maximum elapsed time between refresh commands that is sufficient to ensure that RAM contents remain defined.

3.2.34 request packet: A packet that is generated by a controller to initiate a directed transaction with a selected slave.

3.2.35 requester-capable: A term used to describe RamLink slaves that behave as DMA masters in the sense that they generate request packets and receive response packets. The delivery of these request and response packets is done by the controller.

3.2.36 response packet: A packet that is generated by a slave to return data or status from an address specified by a previous request packet.

3.2.37 ringlet: The concept of RamLink is based on a point-to-point connection of devices circularly connected, starting and ending at a controller, thus forming a ring. Rings are more effective when small, hence the diminutive *ringlet* is emphasized in this standard.

3.2.38 RingLink: A physical signaling model, consisting of point-to-point connections between nodes in a ring, that supports the RamLink logical protocols. RingLink is optimized for robust longer-distance single-board as well as cross-board communications.

3.2.39 SCI: An abbreviation for IEEE Std 1596-1992, IEEE Standard for Scalable Coherent Interface [B4].

3.2.40 SCI standard: Refers to IEEE Std 1596-1992.

3.2.41 self-refresh: A RAM-refresh protocol, where RAM-local hardware schedules the timing and specifies addresses for RAM refresh cycles.

3.2.42 shutDown: The lowest-power operating mode of RamLink slaves, in which all signals, except for a bused *linkOn* signal, may be ignored.

3.2.43 shutOff: An unpowered state of RamLink slaves, in which a drop in the supply voltage may have caused a loss of volatile memory state. The contents of DRAM storage become undefined; storage on special RAM-devices (such as FLASH memory) may be unaffected by the shutOff state.

3.2.44 slave: The entity that responds to RamLink transactions (the transaction addressing is sufficient to support up to 63 slaves on each RamLink ringlet).

3.2.45 standBy: A lower-power operating mode of RamLink slaves, in which a change in the flag line is sufficient to quickly reactivate attached chips.

3.2.46 standBy packet: An event packet that initiates the transition into the standBy state.

3.2.47 SyncLink: A physical interconnect model, consisting of shared input and output links, that supports the RamLink logical protocols. SyncLink is optimized for short-distance single-board communications using a bused connection. The signal-levels for this interconnect model have been left for a future extension to this standard.

3.2.48 sync packet: A packet consisting of four high and four low flags, as well as four data bytes whose bits are the complement of the flag values, typically generated when leaving shutdown (to synchronize the device and controller receiver circuits).

3.2.49 tick: A time interval that is equal to the transmission of one RamLink data byte.

3.2.50 tick-pair: A time interval that is equal to the transmission time of two RamLink data bytes.

3.2.51 wait packet: A packet with four high flags and four undefined data bytes, typically output by the controller after a *standBy* packet has been output.

3.2.52 wake packet: A packet with two low and two high flags as well as four undefined data bytes, typically generated by the processing of idle packets when leaving the standBy mode.

3.3 Bit and byte ordering within packets

RamLink communication is performed by sending packets, and (to simplify hardware) each packet consists of a multiple of four bytes. Packet format illustrations specify the content of the data bytes as well as the values of the associated flag and clock lines, as shown in figure 6.

The first-through-last bytes within a packet (as transmitted in time) are always the top-through-bottom bytes, as shown in figure 6. The partitioning of the fields within data bytes is specified by the number of bits within each field, which are specified at the bottom of the packet. When a data object consists of multiple bytes, the first-through-last bytes are the most- through least-significant, respectively.

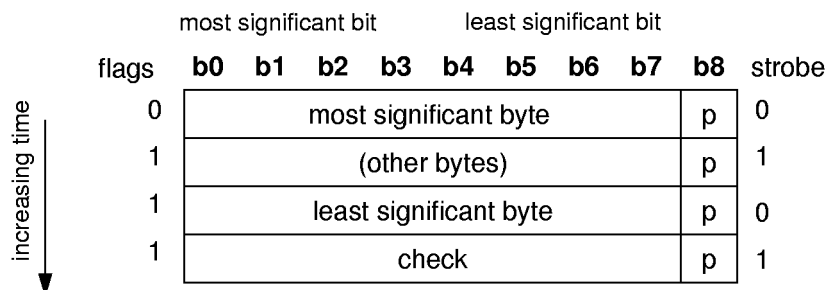


Figure 6— Byte ordering within packets

Within any data byte, the most-significant through least-significant bits are always the left-most through right-most bits. Although bit labels are not included in most illustrations, the left-most through right-most bits are to be labeled bit 0 through 7, respectively.

This is a consistent big-endian data representation. Some designers are more comfortable using mixed notations (0 is the most-significant byte, but the least-significant bit). However, the use of consistent notation minimizes confusion when specifying the addresses of bit fields and multi-byte components that cross byte-address boundaries.

Because all packets are multiples of 2 bytes, 2-byte-wide data-path extensions (although not currently defined) can readily be supported within future signaling-layer standards. The nine data bits and the flag line would have to be

replicated, but one strobe signal would be sufficient. However, the details of 2-byte-wide signaling-layer definitions are beyond the scope of this standard.

3.4 Bit and byte ordering within CSRs

Annex C of this standard defines (optional) control-and-status registers (CSRs) that are 4 or 8 bytes. To ensure interoperability between controllers and slaves, the ordering of the bytes within these registers is defined by their relative addresses. RamLink specifies the mapping of data-byte addresses to bytes within the multi-byte CSRs, which indirectly specifies the RamLink byte-transmission order. For a quadlet CSR access, the most-significant data byte of a quadlet at address *A* has the smallest address (and is transmitted first), as illustrated in figure 7.

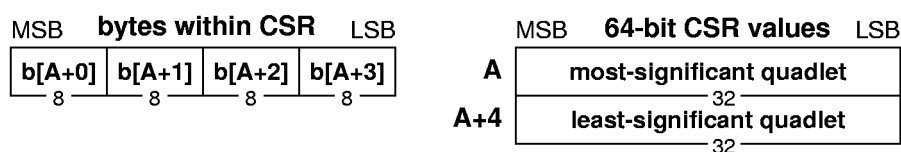


Figure 7— Byte ordering within CSRs

Since 64-bit addressing is supported throughout this standard, some values are stored as quadlet-register pairs. For consistency, the most-significant quadlet register also has the smallest address, as previously illustrated in figure 7.

Note that different byte ordering conventions may be specified for vendor-defined registers. For example, a RamLink-accessible graphics frame buffer could place data-byte-zero in the least-significant portion of a pixel-depth parameter. These vendor-dependent formats are beyond the scope of this standard.

For most of the CSRs, the sizes of all fields within the quadlet are specified. The bit position of each field is implied by the size of fields to its right or left. This labeling convention is more compact than bit-position labels, and avoids the question of whether 0 should be used to label the most- or least-significant bit.

3.5 Field notation (*italics and bold usage*)

Fields within a packet or fields within a register are often illustrated in figures and described in text. Within the figures, a nonitalic Helvetica font is used. When described within text, an italic font is used to describe a field, such as ***id***, to avoid confusion with surrounding English words. A bold font is used to emphasize the first use of the term; subsequent use of the term is presented in a regular italic font.

3.6 Numerical values

Decimal, hexadecimal, and binary numbers are used within this document. For clarity, decimal numbers are generally used to represent counts, hexadecimal numbers are used to represent addresses, and binary numbers are used to describe bit patterns within binary fields.

Decimal numbers are represented in their standard 0, 1, 2, ... format. Hexadecimal numbers are represented by a string of one or more hexadecimal digits (0-9, A-F) followed by the subscript 16, except in C-code contexts, where they are written as 0x123EF2, etc. Binary numbers are represented by a string of one or more binary digits (0, 1), followed by the subscript 2. Thus, the decimal number 26 may also be represented as 1A₁₆ or 11010₂.

When describing bit or byte quantities, the abbreviations of K, M, and G are used to represent powers of two, rather than powers of 10, as is the normal metric practice. Thus, 1 Kbyte refers to 2¹⁰ (1024) bytes, 1 MBit refers to 2²⁰ bits, and 1 Gbyte refers to 2³⁰ bytes.

3.7 Signaling layers

RamLink defines a common set of communication protocols that can be supported on a wide-range of signal-layer technologies.

3.7.1 RingLink signaling

The most robust and extendable signal-layer is RingLink. A set of point-to-point links is used to connect the controller to multiple slaves (typically memory chips), as illustrated in figure 8. The disadvantage of using RingLink signaling is latency; the cumulative through-chip delays add to the effective access time of any RAM (see annex B for details).

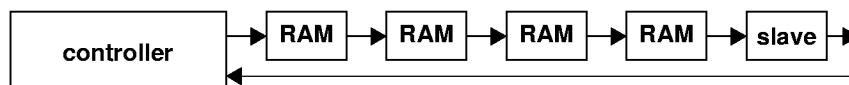


Figure 8— A small RingLink layout

3.7.2 SyncLink signaling

A lower-latency signal layer is SyncLink. Two links, send and sink, are used to connect the controller to multiple slaves (typically memory chips), as illustrated in figure 9. The disadvantage of using SyncLink signaling is extensibility. High-speed buses are only effective when the stubs are short and the overall length is short (see annex C for details).

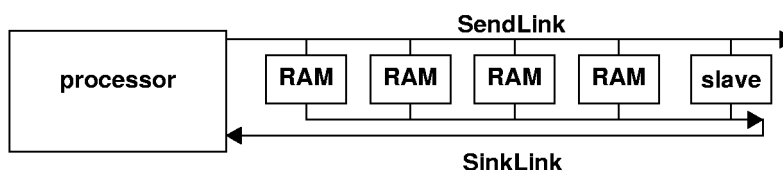


Figure 9— A small SyncLink layout

3.7.3 RamLink illustrations

When the descriptions of RamLink protocols are applicable to any of these signaling layers (3.7.1 and 3.7.2), signaling-layer-independent illustrations are used, as illustrated in figure 10. Two links from the controller (outgoing and incoming) are illustrated, but their connections to the chips (which are signaling layer dependent) are hidden by routing the links under the attached chips.

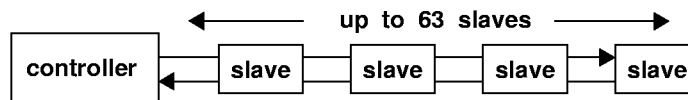


Figure 10— Signaling-layer independent illustrations

4. RamLink configurations

4.1 Simple topologies

RamLink uses point-to-point communication to achieve a simple *high-bandwidth* data-transfer path between a memory controller and one or more memory slaves. A RamLink ringlet consists of one controller and up to 63 slaves, as illustrated in figure 11.

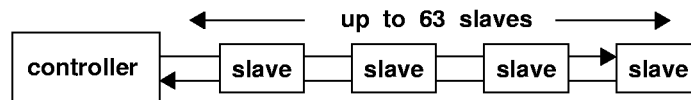


Figure 11— Logical RamLink topology

Having only one controller on each ringlet dramatically simplifies the initialization and arbitration protocols. Limiting each ringlet to 63 slaves simplifies the packet encoding, because the slave address can be contained in the first byte of every packet. The limit is 63, rather than 64, because slave address 63 is used for broadcast and narrowcast addresses.

The links between RamLink nodes are unidirectional and the signal values can change at every clock transition. The nominal clock period is physical-layer dependent (typically 4 ns), but RamLink changes data values on both edges of the clock (i.e., the data signals may change every 2 ns). For a byte-wide implementation, this corresponds to a raw data bandwidth of 500 MBytes/s.

The basic links contain 13 signals: *linkOn* (an initialization signal), a clock reference signal, a strobe signal, a flag signal, and nine data signals. The link is source-synchronous, in that the incoming strobe signal indicates when the other input signals are valid. The flag signal marks the beginning and end of transmitted packets. The ninth bit is provided to support 9-bit-wide RAMs.

Note that 8-bit and 9-bit chips are interoperable. When receiving, 8-bit chips ignore the ninth bit; when driving, the ninth bit is driven to a high state; the value of the ninth bit is maintained when packets pass through 8-bit slaves. RamLink supports both 8-bit and 9-bit-wide DRAMs, allowing the marketplace to determine which is the most applicable.

The data signals are used to transmit bytes within packets. Depending on the location within a packet, the bytes provide address, command, status, or data values. The final byte in all packets is a checksum byte that validates the packet's contents.

4.2 Hierarchical topologies

Although each ringlet is limited to 63 slaves, a memory system may contain larger numbers of memory chips. A simple two-level hierarchy supports up to 3969 memory chips, as illustrated in figure 12. This hierarchical memory system has hybrid agent nodes, because at the higher levels of the hierarchy, agents appear to be slaves, while at the lower levels of the hierarchy these agents appear to be controllers.

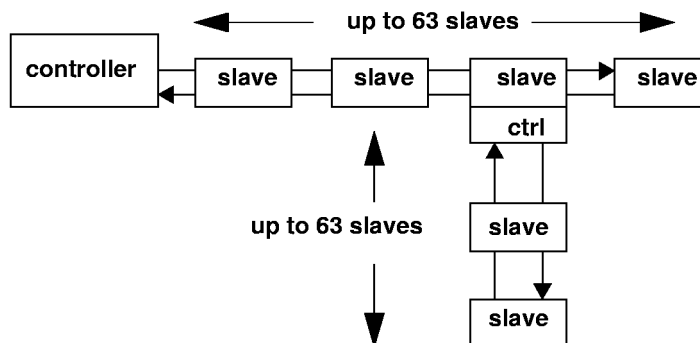


Figure 12— Hierarchical RamLink topology

It is also possible to create a hierarchy in which RamLink is used at the higher levels of the hierarchy, and another communication protocol (such as the Synchronous DRAM interface) is used at the lower levels.

As a slave, the RamLink agent behaves like a large memory slave (whose size is the sum of the RAM sizes below it in the hierarchy). Agents are required to perform address translations on packets passing through them, since the RamLink addresses contain only a six-bit slave identifier (*slaveId*). For requests, address translations are expected to involve moving some of the more-significant address-offset bits into the 6-bit *slaveId* field, as illustrated in figure 13. For responses (returning from the slave to the controller), the address translations involve substitution of the agent's *slaveId* identifier.

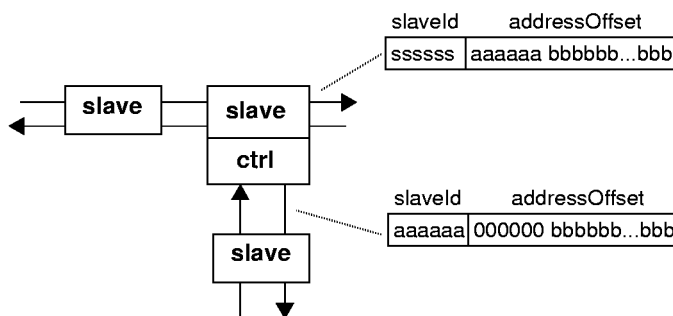


Figure 13— Hierarchical address translation

To support such hierarchical configurations, RamLink supports extended 64-bit address offsets. However, the detailed mappings of hierarchical topologies are beyond the scope of this standard.

An agent is also required to modify the *time* field within request packets (the through-delay is subtracted) and retry packets (the through-delay is added).

4.3 Hybrid signaling

In a hierarchical topology, different signaling technologies may be utilized at different levels of the hierarchy. On large systems, for example, RingLink signaling (which is more robust) may be used at the higher levels, and SyncLink

(which supports RamLink protocols but has lower latencies) may be used on the lower levels of the hierarchy. Thus, RingLink signaling could be used for interconnecting SIMM packages, and SyncLink signaling could be used for SIMM-local communications, as illustrated in figure 14.

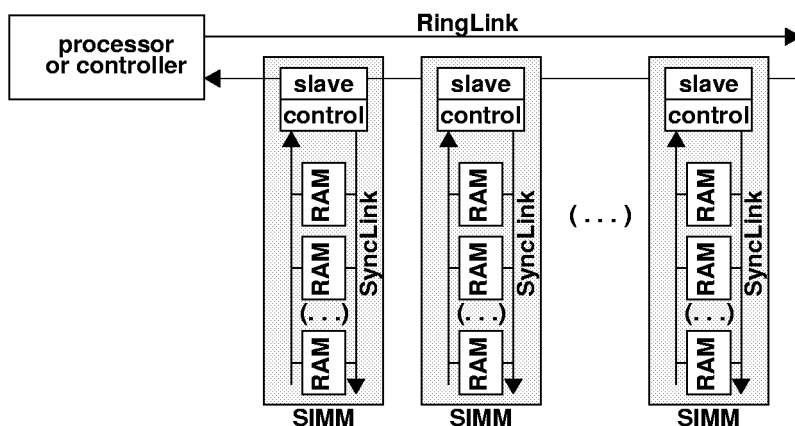


Figure 14— A hybrid large-system design

On small systems, SyncLink may be more appropriate for minimizing the latencies when connecting a few memory chips to the controller. However, RingLink signaling (which is more robust) may be used to connect the I/O devices. Thus, SyncLink signaling could be used for directly attaching a few memories, and RingLink could be used for I/O connections, as illustrated in figure 15.

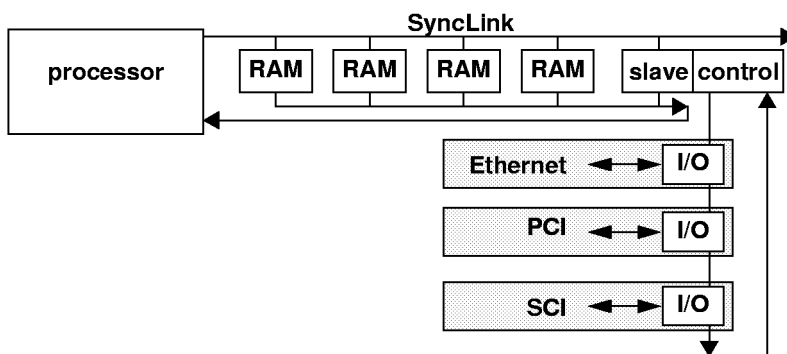


Figure 15— A hybrid small-system design

4.4 Wide (9-bit) data transfers

RamLink slaves that are intended to support parity or ECC applications are expected to be 9-bit-wide RAMs. To support these 9-bit-wide slaves, a ninth signal is provided; this is used as a parity bit during the non-data (head and tail) portions of packets, as a data bit when data bytes are sent, and as an extension bit for the 9-bit check cycle, as illustrated in figure 16.

extra bit	p	p..p	d0	d1	d2	d3	d4	d5	d6	d7	p	check
8 data bits	head		data0	data1	data2	data3	data4	data5	data6	data7	timelo	

Figure 16— Wide (9-bit) data-byte formats

Because the ninth bit has no impact on the state sequencers, 9-bit RamLink slaves can be used within 8-bit systems with minimal impact on the cost of 8-bit-optimized controllers, as illustrated in figure 17. On writes, an 8-bit slave shall discard the ninth data bit; on reads, an 8-bit slave shall provide a 1-valued ninth data bit. The ninth data bit is included in the calculation of the packet's check value.

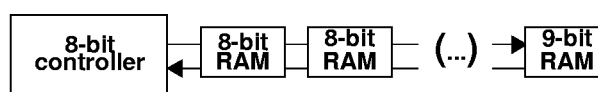


Figure 17— Mixed (8/9-bit) ringlet topology

4.5 DRAM error-checking options

Some implementations are expected to provide error-correction circuitry (ECC) to correct single-bit soft and/or hard errors. RamLink supports a variety of ECC implementation options, as described in 4.5.1, 4.5.2, 4.5.3, and 4.5.4.

4.5.1 Nonexistent ECC

Although less robust, not implementing ECC is the most efficient and the lowest-cost option, since 8-bit-wide RAMs are cheaper than 9-bit-wide RAMs. Individual-byte writes are fast (special read/modify/write ECC cycles are not needed), and either 8-bit-wide or 9-bit-wide RAMs can be used, as illustrated in figure 18.

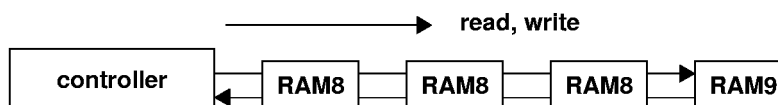


Figure 18— Controller ECC circuitry, 9-bit-wide RAM

4.5.2 Controller-based ECC

RamLink allows the controller to implement ECC by using slightly larger RAMs and a 9-bit-wide data-transfer path. To read a 72-bit word, the controller generates a read8 transaction, which fetches eight items (an item consists of one byte and its associated ECC bit), as illustrated in figure 19.

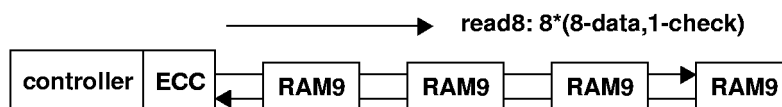


Figure 19— Controller ECC circuitry, 9-bit-wide RAM

For small write transactions, controllers are expected to generate a read8 transaction, insert the modified data and regenerate the ECC, and generate a write8 transaction to return the updated eight bytes of data.

Controller-based ECC protocols are less efficient for small (less than 8-byte) transfers and 9-bit-wide RAMs are expected to be relatively more expensive than their 8-bit counterparts. However, (when compared to on-chip ECC) the RAM-chip complexity is reduced by performing the ECC computations in the controller.

4.5.3 Controller-based parity

RamLink also allows the controller to implement simple byte-parity error-detection circuitry (EDC), using the same data-transfer protocols previously discussed. To read an 8-bit byte, the controller generates a read4 transaction, which fetches 4 bytes of data (including the desired data byte and its associated parity bit), as illustrated in figure 20. Small writes are done in a similar fashion, using writeSb to update the 1-, 2-, or 4-bytes and their affiliated parity bits.

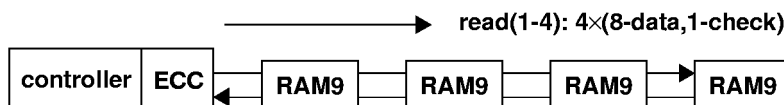


Figure 20— Controller parity circuitry, 9-bit-wide

Controller-based parity protocols are more efficient for small (less than 8-byte) transfers. However, (when compared to on-chip ECC) the error-detection protocols are less robust; double-bit errors are not detected and single-bit errors cannot be corrected.

4.5.4 Transparent ECC

RamLink is also intended to support slave components that support transparent ECC or that do not support ECC at all, such as ROM, whose checksum can be verified when the system is booted. Slaves with transparent on-board ECC and slaves without ECC behave similarly—only eight bits of data are returned.

The advantage of providing slave-resident ECC is that large numbers of bits can be accessed simultaneously and the ECC code can be optimized for larger word sizes (like 256 bits). These ECC codes are more efficient (reducing the storage overhead for ECC) and they can be optimized for vendor-dependent RAM bit-error statistics. On-chip ECC also reduces the use of link bandwidth, because the ECC data are never observed on RamLink.

However, on-chip ECC circuitry may increase the development time (and hence the cost) of RAM chips, particularly when RAM processes are optimized for high-density DRAM, not digital logic. Another disadvantage is that vendors prefer to customize their ECC algorithms based on their product requirements, and their RAM-chip and RAM-board failure models.

RamLink supports both types of 8-bit slaves: basic slaves (which have only 8 bits of data), and slaves with on-board ECC (which only return 8 bits of data), as illustrated in figure 21.

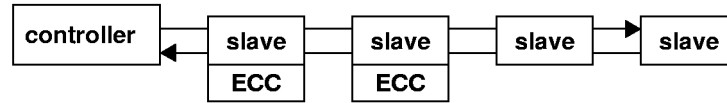


Figure 21— Transparent ECC

The slave memories that provide on-chip ECC quickly detect and correct recoverable (typically single-bit) errors before their response is returned. Because the data are verified before transmission, the controller's response-processing protocol is not affected by recoverable errors.

The detection of unrecoverable errors may be delayed and may occur during packet transmission. These unrecoverable errors are reported in the status byte (the next-to-last byte) of the response packet. Depending on the controller design, the unrecoverable error may be contained or may propagate to the user of the data.

On an SCI memory controller, for example, the CRC on the initial response packet could be “stomped,” and the bad status code could be included within a subsequent repeat transmission of the response packet.

Although on-chip ECC is largely transparent to the controller, it is partially visible to the controller in the following ways:

- a) *Correctable Status.* When error correction occurs, an advisory-error status code (RESP_CORRECTED or AGENT_CORRECTED) is returned.
- b) *Uncorrectable Status.* When an uncorrectable error is detected, a fatal transaction-status code (RESP_CORRUPTED or AGENT_CORRUPTED) is returned.

4.6 Link error-checking

The per-byte parity and per-packet check byte provide protection against link-transmission errors. To ensure interoperability between devices, these values shall be correctly generated by the controller and all slaves. Slaves should check for correct parity and check values on incoming packets, but are not required to.

Since parity and check-byte errors are not expected during normal operation, slaves and controllers are not expected to defer packet processing until the packet integrity has been verified. These transmission errors are expected to be fatal, since slaves and controllers may erroneously update storage (based on incorrect address or command values) before the error is detected.

4.6.1 Controller-generated parity

The controller is responsible for generating a ninth data bit. For the initial non-data cycles, the ninth bit is a parity bit. During the packet's data cycles, the ninth bit is either a data bit (if 9 bits of data are being stored), or (if only 8 bits of data are available) shall be driven to a 1-value. The controller is responsible for generation of the final check value within request and event packets.

4.6.2 Slave parity checking

The slave should check for parity and check-byte errors when packets are received. When an error is detected, additional storage updates should be inhibited. If error checking is supported, and a request-packet error is detected, a response shall not be returned.

4.6.3 Slave-generated parity

The slave is also responsible for generation of the final check-byte when generating its response and event packets, to ensure interoperability between devices. The cost of supporting the check-byte is minor when compared to providing a byte-parity indication, since only one level (rather than three) of exclusive-or logic is required.

The slave is also responsible for generating a ninth data bit. For the first (*slaveId* cycle), the flag/data/parity fields can be copied from the request. For the second (commands) cycle, the flag can be copied from the request; the flag and parity bits are complements of the request-provided values. For the final status cycle, a fixed value is typically returned, so the flag and parity-bit values can be precomputed. Thus, parity protection can be supported without implementing 8-bit parity-generation logic.

During the response packet's data cycles, the ninth bit is either a data bit (if a ninth bit is available) or shall be driven to a 1-value (if only 8 bits are available). In the final check cycle, the ninth bit is part of the check value.

5. RamLink operation

5.1 Split-response transactions

To access RAM, the controller initiates read or write transactions addressed to one of the slaves on the attached ringlet. These split-response transactions are composed of two components, called request and response packets. These are called split-response transactions because the response packet is not returned immediately, so other (possibly unrelated) packets may be transmitted on RamLink while the request is being processed by the slave.

For a write transaction, the request packet transfers command, address, time, and data from the controller to the slave. The response packet normally returns the transaction-completion status, as illustrated in 2a) in figure 22. In some cases, the slave may be unable to complete the transaction within the requested time, and a retry packet (instead of a response) is returned at the specified time, as illustrated in 2b) in figure 22.

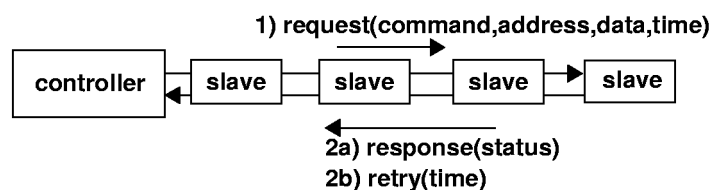


Figure 22— Split-response write transaction

For a read transaction, the request packet transfers the command, address, and time from the controller to the slave. The response packet normally returns the requested data and status.

The completion status in the write response confirms the write completion. The controller is expected to use the completion status to detect write-request transmission errors, or to provide flow control; another request is not sent until a previously queued request completes.

Request packets circulate past their addressed slaves, and are eventually stripped by the controller. When a request packet has passed its addressed slave, the protocols allow a returning response packet to be inserted over part or all of the now-stale request packet. The design of the packet's flag coding avoids misinterpretation of the partial-request-packet fragments generated by these insertions.

An event packet is a specialized packet that is similar to a request, but an event is always accepted and no response is returned. An event can be directed to one slave or broadcast to all. Events transfer only one byte of control information from the controller to the slave(s), and no address-offset is provided. Because of their limited information content, events are only used for special control purposes (such as synchronizing RAM refresh operations).

5.2 Response scheduling

The controller is responsible for timing the generation of its own request packets and scheduling the return of slave-generated response packets to ensure that slave-response packets don't corrupt other request or response packets on RamLink. The controller is expected to inhibit the generation of requests that would collide with previously scheduled responses and to schedule new responses to avoid collisions with previously scheduled responses.

Within hardwired configurations, the controller is expected to be preprogrammed with knowledge of the attached devices, particularly memory (possibly restricting the number of supported device types). In field-configurable applications, the slave devices may provide ROM to identify their capabilities, as defined in the CSR Architecture. Although the number of (potentially) outstanding transactions is large (4×63 , since four may be active for each slave), the controller complexity is typically based on the number of outstanding transactions (for which a request has been sent, but a response hasn't been returned), which is expected to be a much smaller number (typically four).

To support such scheduling, a *sch* (exactly scheduled) bit and a 10-bit *time* field are included within each request packet. The *time* field specifies the slave response time in multiples of 2-byte intervals, called tick-pairs.

If the *sch* bit is 1, the *time* field specifies the time when the slave's response shall be returned. Controllers using exact scheduling typically set the *sch* bit to 1. Exact scheduling is expected to be used when deterministic RamLink behavior is desired, or under heavy loading conditions (exact scheduling maximizes the effective bandwidth).

If the *sch* bit is 0, the *time* field specifies a time limit within which the slave's response shall be returned; the response may also be returned earlier. Controllers using bounded scheduling typically set the *sch* bit to 0. Bounded scheduling is typically used by simple memory controllers, under light loading conditions, or when the slave's response time cannot be predicted by the controller (typically due to asynchronous on-chip refresh operations).

For all request packets, *time* specifies the number of tick-pairs between the last (*checksum*) byte of the request and the first (*slaveId*) byte of the returned response, as measured at the slave's output. If the request is overwritten (by a previously scheduled response from the same chip), the time is referenced from when the request would otherwise have been output from that chip.

5.3 Retried transactions

A *retry* packet may be sent to the controller in place of an expected response. It indicates that the original request packet should be resent at a later time. The retry packet includes a distinct *time* field; if this value is included within the resent request (and the slave is idle), this *time* value is sufficient to ensure that (in the absence of refresh operation conflicts) the next request will be successfully completed. An exception is a concurrent refresh operation, which may cause the next request to fail, even though it contains the *time* value from the previously returned *retry* packet.

5.4 RamLink address space

The RamLink address space is similar to the 64-bit fixed address space described within the CSR Architecture and implemented within the SCI standard. Only 64 node offsets (rather than the specified 64 K) are supported and the range of address offsets is either 32 or 48 bits (rather than the specified 48 bits), as illustrated in figure 23.

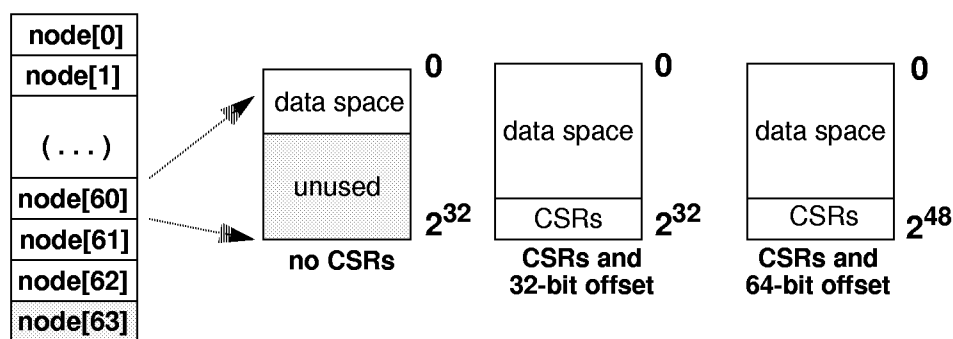


Figure 23— RamLink address space partitioning

Note that the definitions of node (as used within the CSR Architecture) and slave (as used within RamLink) are the same. Thus, there are 64 node/slave addresses within the RamLink address space. Each node/slave shall respond to a 32-bit address-offset space, and may (optionally) respond to a 48-bit address-offset space as well.

Node 63 has special meanings. It is used to uniquely distinguish specialized sync, broadcast, and narrowcast packets. The broadcast address space is more restrictive than the CSR Architecture, which allows write4 transactions to be broadcast—on RamLink, only events can be broadcast. Although node 62 is the initial (post-reset) node address, its operational properties are the same as those of nodes 0 through 61.

A node with no CSRs responds to 32-bit address offsets, but a RESP_ADDRESS error is returned when an unimplemented portion of its address space is accessed. Thus, a read of a potential CSR address is sufficient to determine the absence of these CSR resources. Similarly, reads of the data space are sufficient to determine how much of this address is supported. For example, on a simple 64 Mbit RAM chip, all addresses other than the first 8 Mbytes of the data space would result in a RESP_ADDRESS error.

For 32-bit- or 32/48-bit-capable nodes with CSRs, a read of a potential CSR within the 32-bit address-offset space will succeed. Note that 48-bit-capable nodes support 32-bit address offsets by aliasing a portion of their data space and all of their CSR space into 32-bit address-offset space. Thus, a read of a potential CSR within the 48-bit address-offset space (not the 32-bit address-offset space) is used to differentiate between 32-bit and 32/48-bit options: a RESP_ADDRESS error is returned if the node is only 32-bit capable.

Accesses of (optional) CSR locations can be used to support extensive testing or RAM-initialization protocols (see annex D for additional details).

5.5 Address errors

In normal operation, the controller is expected to know the addresses of the attached slaves, and request packets containing invalid addresses are not expected to be generated. However, such request packets are generated during system initialization, and could be generated by unpredictable error conditions.

Request packets with invalid *slaveId* addresses circulate around the ringlet and are ignored by the controller. The *slaveId* address error is detected by the controller when the response packet is not returned within the scheduled time, as illustrated in figure 24.

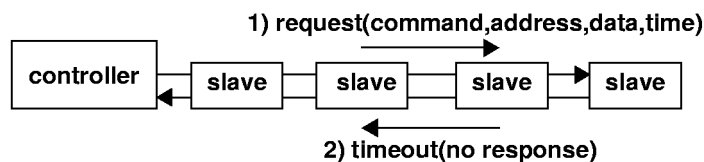


Figure 24— SlaveId address error

5.6 Request queue sizes

The RamLink protocols allow up to four requests to be concurrently queued within each slave; distinct 2-bit *tId* (transaction identifier) values are used to uniquely identify the queued requests and their returned responses. However, since buffer space is required to queue active requests or delayed responses, slaves may support only a smaller range of *tId* values. Any request with an unsupported *tId* value is ignored, and (to the controller) behaves like an address error.

The request-queuing capabilities of a RamLink slave shall be characterized by the values of *writeReq* and *totalReq*. The *writeReq* value specifies the number of *tId* values (starting from zero) that can be used for either write or read requests; the value of *totalReq* specifies the number of *tId* values (starting from zero) that can be used for reads. These values shall be constrained as specified in equation 1:

$$1 \leq \text{writeReq} \leq \text{totalReq} \leq 4 \quad (1)$$

For example, consider a slave that can queue, at most, one write request, but that can also queue three read requests, or one write request and two read requests. For this slave, the values of *writeReq* and *totalReq* would be one and three.

When *writeReq* or *totalReq* are less than four, only the lower *tId* values shall be used. In this example, only write requests with *tId* values of 0 are processed. Similarly, only read requests with *tId* values of 0, 1, and 2 are processed. Requests with other *tId* values are ignored and (to the controller) appear to be *slaveId*-address errors.

For each slave, controllers are expected to determine the values of *writeReq* and *totalReq* by generating read requests with all possible *tId* values and write requests with all possible *tId* values. By observing which of these transactions generate a *slaveId*-address error, the controller can determine the slave's *writeReq* and *totalReq* values.

There is no minimum specification for delays between the slave's request packets; the slave shall be capable of handling the worst-case, back-to-back, request-arrival rate. Requests need not be processed at their arrival rate; if an insufficient request/response processing delay is provided, several (or all) of these requests may be busied.

5.7 Request ordering

High-performance controllers are expected to have two or more transactions concurrently queued. For example, a write transaction and a following read transaction (to the same address) could be concurrently queued in the same slave. If the write transaction is not retried, the controller expects the read to return previously-written data.

To support these controller expectations, RamLink slaves shall ensure that transactions, when not retried, are processed in an apparent first-in, first-out (FIFO), request ordering. RamLink controllers shall ensure that the response-time values specified within concurrently-queued requests are monotonically increasing, which means that reordering of requests within the slave (to meet the desired response deadlines) is not required. The controller's ordering constraints are expected to have a minimal performance impact while dramatically reducing the complexity of multiple-*tId*-capable RamLink slaves.

To reflect these ordering assumptions at a higher level, the controller is either expected to wait for the returned response, or to use autorefresh, and to schedule dependent requests so that retry packets are never returned.

5.8 Refresh operations

RAMs based on dynamic memory technologies have to be refreshed. To avoid data corruption, refresh operations are typically performed periodically, which bounds the maximum time between successive refresh operations on any sets of storage locations. Depending on the application, refresh may be performed transparently, or may be explicitly scheduled by the controller.

Other queued dependent requests are delayed while a refresh operation is being performed. Thus, the refresh operation may cause other queued requests to be retried. To avoid such retries, simple controllers are expected to use bounded scheduling, and to provide additional time for the requests to be processed. High-performance controllers are expected to use exact scheduling and avoid these retries by explicitly scheduling their DRAM refresh operations.

5.8.1 Self-refresh

All refresh-dependent RamLink chips shall provide a self-refresh capability. Although primarily intended to be used when the slave is in *shutDown* mode, self-refresh can be used during normal system operation. This simplifies the controller (which need not worry about RAM refresh) at the cost of introducing uncertainty in controller scheduling algorithms.

A slave enables its selfrefresh circuitry when a *refreshSelfOn* event is received, or when entering shutdown mode. While selfrefresh is enabled, broadcast and directed *refreshNow* events are ignored.

5.8.2 Autorefresh

To support more-efficient controller scheduling algorithms, refresh-dependent slaves shall also support autorefresh. The prefix *auto* refers to the addressing that is automatically updated by the slave, not to the timing, which is explicitly set by the controller.

By using the autorefresh capability, the controller can accurately predict RAM response times and can schedule refresh activity during idle periods. Accurately specified response times simplify diagnosis of system failures, since memory-access sequences are repeatable. Accurately specified response times also help synchronize RAID-like memory arrays or array-element processing in vector-processing applications.

A slave enables its autorefresh circuitry when a *refreshAutoOn* event is received. While autorefresh is enabled, broadcast and directed *refreshNow* events initiate the slave's refresh operations. The actual refresh operation is deferred until the currently queued requests have been processed.

To simplify controllers and ensure interoperability between memory chips, the refresh requirements of slaves are constrained. Slaves in autorefresh mode shall be designed to operate correctly if the controller issues the appropriate *refreshNow0*, *refreshNow1*, *refreshNow2*, and *refreshNow3* events. The controller is only expected to issue one type, but is expected to select the one corresponding to the most-frequent refreshing requirements.

When generating *refreshNow0* events, the controller is responsible for generating at least 4096 *refreshNow0* events in every 64 ms interval. If evenly spaced, the average interval between *refreshNow* events would be 15.6 μ s, but the controller has the flexibility of spacing the *refreshNow* events at regular intervals, or of hatching them within 64 ms intervals. The *refreshNow1*, *refreshNow2*, and *refreshNow3* events (when used) are sent more frequently, as specified in table 1.