OpenSPARC™ T1 Processor External Interface Specification

Sun Microsystems, Inc.
www.sun.com

Part No. 819-5014-10
March 2006, Revision A

Submit comments about this document at: http://www.sun.com/hwdocs/feedback

Copyright 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.sun.com/patents and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Java, AnswerBook2, docs.sun.com, Sun4U, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and in other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and in other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights—Commercial use. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à http://www.sun.com/patents et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Java, AnswerBook2, docs.sun.com, Sun4U, et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une license non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciées de Sun qui mettent en place l'interface d 'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Please
Recycle

Contents

Figures

Tables

Preface

This document describes the external interfaces for the OpenSPARC™ T1 processor from Sun™ Microsystems, Inc. This processor is the first chip multiprocessor that fully implements the Sun Throughput Computing Initiative.

How This Document Is Organized

This document is organized as described in the following paragraphs.

- Chapter 1 provides a complete description of the J-Bus system.
- Chapter 2 describes the Peripheral Component Interconnect (PCI) model for J-Bus, including data flow, flow control, and the master interface.
- Chapter 3 provides details of the J-Bus transaction set, memory transactions, noncached data transactions, interrupt control transactions and the J_ADTYPE signal.
- Chapter 4 describes J-Bus reset sequence guidelines and RST_PIN_EN exceptions.
- Chapter 5 specifies the distributed arbitration protocol for driving any address or data onto J-Bus.
- Chapter 6 provides configuration descriptions and restrictions, along with power implications.
- Chapter 7 describes the required physical address map for J-Bus devices, along with how to handle address mapping errors.
- Chapter 8 provides details of the Serial System Interface (SSI).

Using UNIX Commands

This document might not contain information about basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. Refer to the following for this information:

- Software documentation that you received with your system
- Solaris™ Operating System documentation, which is at:

  http://docs.sun.com

Shell Prompts

Shell	Prompt
C shell	*machine-name*%
C shell superuser	*machine-name*#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Typographic Conventions

Typeface*	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`% su` `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename*.

* The settings on your browser might differ from these settings.

Related Documentation

The documents listed as online are available at:

`http://www.opensparc.net`

Application	Title	Part Number	Format	Location
OpenSPARC T1 instruction set	*UltraSPARC Architecture 2005 Specification*	950-4895-03	PDF	Online
OpenSPARC T1 processor internal registers	*UltraSPARC T1 Supplement to the UltraSPARC Architecture 2005*	819-3404-02	PDF	Online
OpenSPARC T1 signal pin list	*OpenSPARC T1 Processor Datasheet*	819-5015-10	PDF	Download
OpenSPARC T1 megacells	*OpenSPARC T1 Processor Megacell Specification*	819-5016-10	PDF	Download
Running simulations and synthesis on the OpenSPARC T1 processor	*OpenSPARC T1 Processor Design and Verification User's Guide*	819-5019-10	PDF	Download

Documentation, Support, and Training

Sun Function	URL
Documentation	http://www.sun.com/documentation/
Support	http://www.sun.com/support/
Training	http://www.sun.com/training/

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

http://www.sun.com/hwdocs/feedback

Please include the title and part number of your document with your feedback:

OpenSPARC T1 Processor External Interface Specification, part number 819-5014-10

CHAPTER 1

J-Bus: Overview and Philosophy

This chapter describes the following topics:

1.1 Introduction

The J-Bus system shown in FIGURE 1-1 is composed of a central processing unit (CPU); memory; Peripheral Component Interconnect (PCI) mode; and an interconnect model that specifies architectural queues, caches, translation lookaside buffers (TLB), and overlapped and pipelined operations.

This specification provides the following level of detail:

- Specifies all behaviors that are visible between major functional areas
- Identifies major implementation decisions that are necessary to meet performance goals

J-Bus meets the following requirements:

- Correctness with respect to SPARC-V9 memory models
- Compatibility with Sun4U™ system architecture, including PCI extensions

J-Bus reflects ideas from a variety of bus implementations and is optimized to keep performance up for the 1–4 way CPU design point.

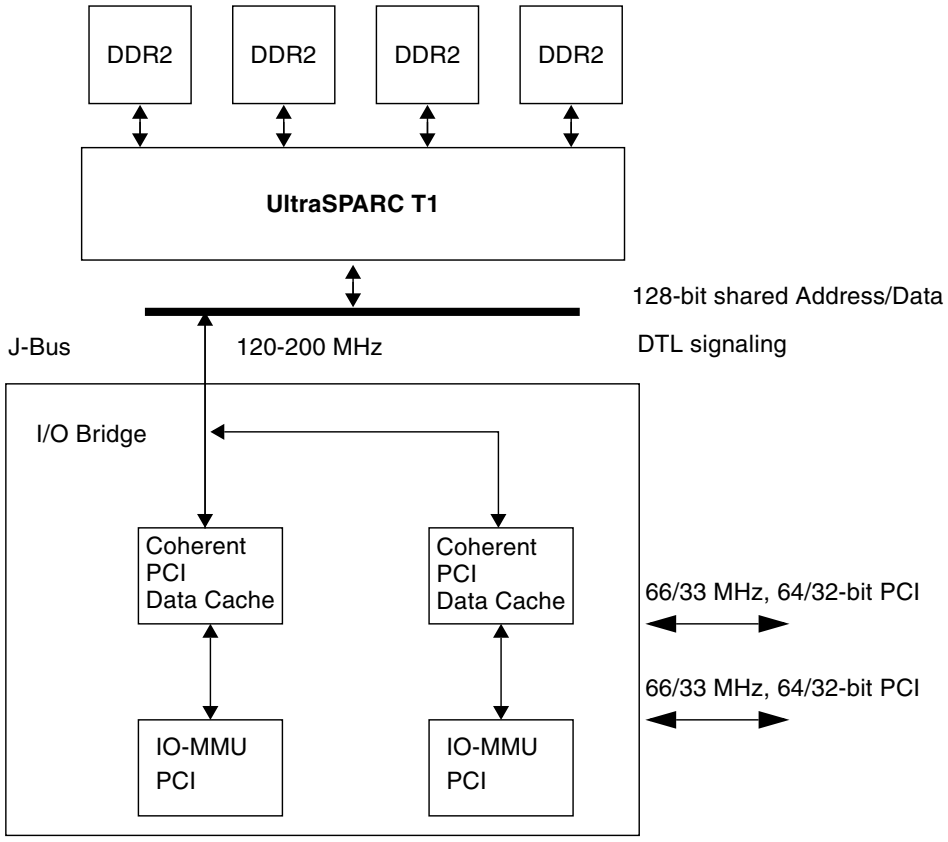

FIGURE 1-1 Non-pipelined J-Bus System

1.2 Goals

J-Bus architectural goals include the following:

- **Simple flow control.** Distributed variable latency for all producers and consumers of information.
- **Shared address and data bus**.
- **I/O bridge (IOB) provides the PCI interface**.
- **150–200 MHz J-Bus operation**, with or without dead cycle, programmable for three J-Bus loads (one CPU, two I/O bridges). Signals always driven, except for bus turn-around cycle during dead cycle mode.
- **No hard latency or throughput requirements in the protocols.** This enables some simplification of the design, as opposed to a fixed-latency, fixed-throughput design. Some low-latency behavior is noted. Average-latency and throughput requirements do exist for performance reasons.
- **Out-of-order data return for different cacheable addresses.** In-order data return from a single noncacheable port. In-order data return for the same cacheable address. Order is determined by address bus order.

  Blocking is done in a distributed fashion, with each port tracking its own reads. There is no system definition for a maximum number of outstanding reads. Any maximum would depend on the port designs.

  There is a maximum number of eight outstanding reads per port ID due to the fixed width J_ADTYPE signal, which includes read transaction ID information. Stores are not constrained.
- **Minimize parallel control flow paths to eliminate sources of race conditions.** This stipulation eliminates corner cases and makes the chip easier to design and debug.
- **Interrupt delivery model matches Sun4U at the mondo vector and software visible points**.
- **PCI subsystems have queued input packets and delayed data return**, so their activity is fully decoupled from J-Bus activity.
- **Snooping is variable latency, but fully pipelined and overlapped.** Snoops are initiated at the processors (and IO-cache) when the request address is presented on the shared J_AD bus. The snoop latency must be less than the memory latency. Start the DRAM read and cancel the return data J-Bus drive if the snoop indicates a dirty line elsewhere.

1.3 Signal List

J-Bus control signals consist of the following:

- **J_REQ_OUT_L[1:0], J_REQ_IN_L[5:4]**

  Used for distributed arbitration requests for J_ADTYPE/J_AD/J_ADP. Typically, a maximum of one CPU plus two I/O bridges. Used to request bus for addresses, write data, and returning read data.

  J_REQ_OUT_L[1:0] is output only, two copies of this port's request.

  J_REQ_IN_L[5:4] are up to two possible other requests. The hookup order depends on agent ID and must be exactly to specification; otherwise, the arbitration will not work correctly.

  Dead-cycle mode variants are used to reserve the global J-Bus for one more cycle than the ownership of a local bus segment to account for the J-Bus repeater delay, if used.

- **J_AD[127:0]**

  Used for all address and data packets.

- **J_ADP[3:0]**

  Used for word parity for all J_AD transfers. J_ADP[3] should also cover the eight extra bits of J_ADTYPE, that is, ~(^AD[127:96] ^ ^ADTYPE[7:0]). ("^" represents Exclusive OR.) Ports receiving a data packet should check incoming parity. All ports should check parity on all address cycles.

  All 1's on J_AD/J_ADP/J_ADTYPE is correct parity (the not-driven dynamic termination logic (DTL) state).

- **J_ADTYPE[7:0]**

  Identifies the packet type on J_AD/J_ADP and signals the destination for the returning read data. Contains the read transaction ID for out-of-order read data return. The undriven J_ADTYPE/J_AD value of all 1's corresponds to an idle cycle.

- **J_PACK0,1,4,5[2:0]**

  Bused to all, but single driver (enable depends on agent ID). Contains encoded snoop information and flow control, plus read data flow control (consumption notice).

  As input, only use J_PACKs enabled by a control status register (CSR) to match system configuration. J_PACK will still be 1's at reset deassertion if it should not be used.

  Agent ID is used to decide which of the bidirectional J_PACKs are driven by a port.

- **J_PAR**

  Driven by one I/O bridge every cycle.

  XNOR of J_REQ[5:4], J_PACK0,1,4,5[2:0] from N cycles ago. See Section 1.12, "J_PAR: Parity for J_REQ_L and J_PACK0-6" on page 1-16 for details.

  Used to check parity on these control signals. If I/O bridge is the source of a signal, it should pipe its output forward for use in the XNOR to avoid sampling the signals that it drives.

  This J-PAR signal is driven N cycles after the relevant signals are active on J-Bus because I/O bridge must register inputs, compute, then register output.

  Each device on the J-Bus should register this J_PAR and compare it to the J_PAR devices computed in the previous cycle (but these J_PAR devices did not drive their J_PAR value).

  Each port should detect a local error, log, and report the error, if it detects a problem.

Note – Conditionally, a CSR can specify that a delayed version of a device's own J_PACK and other devices be used to mimic the delayed propagation of J_PACKs through the repeater, which depends on the system configuration.

Note – Another CSR bit will control whether, for this port, J_PAR must be propagated through a repeater, thus requiring a further pipeline delay in the internal parity check. If this port is on the same J-Bus segment as the port driving J_PAR, there is no additional delay.

- **J_RST_L**

  Non-power-up reset, active low.

- **J_POR_L**

  Power-up reset, active low. Actual name is PWRON_RST_L.

- **J_CLK[+-]**

  Differential pseudo emitter-coupled logic (PECL) clock (120–200 MHz).

- **J_ERR**

  Not used functionally but driven by any requestor that detects an error. This pin is used for scope and logic analyzer work only. Preferably not bussed.

All J-Bus address and control signals are DTL drivers and receivers. Unused J_REQ_L or J_PACK pins should be connected so that a termination will pull them to the inactive state (1).

1.4 Address and Data Flow Control

Global address OK (AOK) and port and address-range specific data OK (DOK) state is maintained by each requestor to decide if there is room for address and write data at the targets for this transaction. The transaction is not driven on the bus if there is no room at the destination.

The start and stop window is symmetric.

- A port is not allowed to drive affected transactions starting in the second cycle after the J_PACK that changed internal AOK/DOK state.
- A port can restart on the second cycle *after* a J_PACK that re-enables the internal AOK/DOK state.

All masters track AOK and DOK states.

A simpler definition of the requirement for AOK/DOK affecting outputs is "as soon as physically possible" (assuming inputs and outputs are registered).

The same rule applies to turning AOK/DOK back on. New outputs are allowed "as soon as possible" (assuming inputs and outputs are registered).

For a port's own AOK/DOK state, the port should look at its own J_PACKs as if it were another port for the correct timing. All ports should react at the same time. This helps fairness because all ports see the blocking removed at the same time.

1.4.1 Address OK

Address OK (AOK) state assumes one address per cycle (worst case) and DOK state assumes 16 bytes of data per cycle (worst case) can be delivered to a port.

AOK state is global. Any port can disable AOK by asserting an encoded single-cycle "AOK off" on its J_PACK signals. "AOK on" from the same port is required before addresses can continue.

Note – It's important that a bit of AOK state be maintained for each port. The global AOK state is the AND of all individual AOK states. This prevents race conditions in different ports setting AOK on and off at the same time.

Note – It is important to avoid livelock due to favoritism towards the last owner and all requests being forced off during AOK_OFF (or DOK_OFF).

After a port turns AOK off, it should not turn AOK back on until it can accept at least one address transaction without turning AOK back off. This means AOK should not go back on until you have free space equal to current overrun definition + 1 (worst case: handle read + writeback), rather than turning on at the current overrun definition. Use the following guidelines:

- Turn AOK off when the space available == overrun spec (4).
- Turn AOK back on when the space available == overrun spec + 2 (6).

These guidelines should solve the livelock issue of one port constantly and solely getting the bus after AOK off/on transitions and blocking out forever. The DOK case does not need special handling.

To avoid oscillation of AOK behavior, the turn on should be delayed until more room than the minimum is available.

1.4.2 Data OK

Data OK (DOK) state is tracked on an address-range basis. Address ranges are assigned to ports at power up based on their port-ID.

An encoded "DOK off" from a port will cause writes to that address range to be suspended (by the initiator) until an encoded "DOK on" by that port is broadcast (single-cycle events on the J_PACK).

The granularity of address-range assignments is kept coarse so that the address matching, before a write is generated, is relatively easy.

The address space has a fixed partitioning to ports. A port's DOK status is used before initiating a write to an address belonging to that port.

Read plus writeback is an atomic transaction, so both the read and writeback request are held up by the initiator if room is not available for the writeback data (indicated by DOK state).

1.4.3 Interrupt Flow Control

Interrupt send is flow controlled with a separate protocol that has a one-outstanding, token-based scheme.

Each CPU thread can queue one incoming interrupt packet. Threads receiving interrupts send an ACK back if the packet is accepted and NACK back if it is refused (because one has been received already, but not digested and unloaded by software).

Note – The J-Bus masters do not look at the AOK/DOK state for restricting interrupt delivery. The interrupt receivers must remember the state for receiving an interrupt address and data that is independent of any other address or data queues.

1.5 Other Caching Issues

Peripheral Component Interconnect (PCI) interfaces hang off the I/O bridge chip.

1.5.1 I/O Data Cache

The PCI input/output caches (IO-cache) and input/output Memory Management Units (IO-MMU) are the initiators of all coherent activity from I/O bridge.

1.5.2 No-Snoop Pages

The Solaris Operating System software guarantees that a page is flushed from all caches and is only used by one port at any time, so a page can be marked no-snoop in the IO-MMU, the CPU IO-MMU, or the data Memory Management Units (D-MMU).

No-snoop references will still cause J_PACKs, but the caches do not need to be interrogated before J_PACKing. Regard no-snoop J_PACKs just like a normal operation.

Note – This is the UltraSPARC T1® processor's normal operating mode (Posted Coherency Mode).

Because both normal and no-snoop pages will have the early memory access start, and no-snoop addresses will still go out on J-Bus, there is *no performance advantage to marking pages as no-snoop*.

1.5.3 Encaching No-Snoop Data

If a non-snooped location is encached and modified, it must be flushed to memory when another port wants to see that data. Also, if another port modifies the data, other ports want to eventually see the update.

This arrangement works if the no-snoop data is installed as E (or M) in the caches. IO-caches will not have a direct flush mechanism. Software will handle any flushing necessary due to remapping.

1.6 Interrupts

An INT transaction is sent to CPUs with data as a multi-cycle packet that includes the Sun4U-specified mondo vector data. This transaction method is used for I/O interrupts sent by I/O bridge.

Note – The OpenSPARC T1 processor only looks at the first cycle (16 bytes) of mondo data because that is the only cycle of meaningful data driven by I/O bridge.

Each thread has queue for one interrupt receive address and data. The thread sends an InterruptAck (INT_ACK) transaction out when the address and data is accepted. The thread sends an InterruptNack (INT_NACK) if the interrupt is refused because the CPU is busy with one already.

Note – The OpenSPARC T1 processor has queues for receiving up to 16 interrupts (2x the 8 outstanding interrupts possible from I/O bridge). For more information, see Section 3.6, “Interrupt Control Transactions” on page 3-10.

The OpenSPARC T1 processor has dedicated registers for receiving mondo vector data.

Power-up and non-power-up reset are individually signaled with two dedicated pins.

The interrupt packet contains source and destination IDs so the target knows to accept the packet.

Because there is a dedicated mondo receive register at the processors to receive the interrupt data, the interrupt request and associated data does not go into the queue affected by AOK or DOK.

AOK flow control at the J-Bus CPU ports is only for outstanding memory request activity. They can fill only due to read and WriteInvalidate (WRI)/WriteMerge (WRM) requests.

AOK, however, can also fill due to I/O bridge input queue being full due to noncacheable activity.

1.7 OpenSPARC T1 PCI Ordering Rules

The PCI ordering rules supported by the OpenSPARC T1 Processor processor is a subset of the PCI ordering rules used by the Solaris Operating System.

1.7.1 DMA Write Ordering

PCI specifies that Posted Memory Writes (PMW) must be processed in order, which means that PMWs must become coherently visible in order. Since the OpenSPARC T1 Processor processor has an interleaved L2-cache, with four separate controllers for the four banks, this ordering requirement might not be met if the DMA writes were indicriminantly forwarded to the different banks without waiting for ACKs from the L2-cache.

Each L2 bank guarantees that it processes incoming requests in order for individual lines. Therefore, the J-Bus interface (JBI) in the OpenSPARC T1 Processor processor can issue multiple WR* transactions to a single L2 bank, as long as they were generated from the same WRM. However, JBI will stall younger WR*/WRIS transactions to a different bank, or to the same bank if from a different bus transaction, until all previous WRM/WriteInvalidateSelf (WRIS) transactions have been ACKed. Once all previous WRM/WRIS transactions have been ACKed, the oldest stalled WRM/WRIS transaction is sent to the L2 bank for which it is targeted.

1.7.2 PIO Read Returns Compared to DMA Writes

PIO Read Returns are guaranteed to force all older DMA writes to be complete before the PIO read data is returned to the processor. To implement this, the JBI stalls NonCachedRead (NCRD) return data until all previous WRM/WRIS transactions have been ACKed by the L2-cache, then forwards that return data to the requesting thread.

1.7.3 Interrupts Compared to DMA Writes

Interrupts are guaranteed to force all older DMA writes to be complete before the interrupt is delivered to the processor. The JBI stalls the INT transaction until all previous WRM/WRIS transactions have been ACKed by the L2-cache, then forwards that interrupt to the target thread. If multiple INTs are received, each one has state specifying how many WRM/WRISs must be processed and ACKed by the L2 before the INT can be forwarded to its target thread, so each INT waits "only as long as it has to."

1.7.4 DMA Reads Compared to DMA Writes – Address Consistency

The OpenSPARC T1 Processor processor does not guarantee that DMA writes and reads are kept in order, unless they are to the same cache line. The processor implements this by guaranteeing that all DMA writes and reads to the same L2 bank are issued in order, and the L2 bank processes writes and reads to the same line in order. DMA writes to one line can be processed out of order relative to DMA reads to a different line, and vice versa.

The OpenSPARC T1 Processor processor implements a mode bit that will force ordering of DMA reads in relationship to previous DMA writes regardless of the addresses. However, the mode bit is not intended to be turned on for normal system use. If an I/O driver/device pair does require read versus write ordering, the mode bit can be turned on.

1.7.5 PIO Request Ordering

All outgoing PIO requests, both PIO Reads and PIO Writes, are maintained in order for all threads, at least until they are issued on J-Bus. There is no bypass capability to enable younger writes to pass up older reads. PIOs are always issued strictly in order.

1.7.6 PIO Writes Compared to DMA Read Returns – Not Supported

PCI has a rule specifying that DMA read returns must not be able to bypass older PIO writes. This rule is only interesting if a driver issues a PIO write to a device, updates memory, and expects that the device cannot see the updated memory before

it sees the PIO write. Because the Solaris Operating System requires a PIO read to ensure visibility of previous PIO writes and I/O bridge does not support this rule, the OpenSPARC T1 processor does not support this rule either.

Instead, the OpenSPARC T1 processor has separate queues for outgoing PIO requests compared to DMA read returns, so the two classes are in no particular order relative to each other.

1.7.7 DMA Reads – No Ordering Rules

PCI inflicts no ordering rules on one read compared to another read. The effective order of reads is defined by the order that the data crosses the bus. On the OpenSPARC T1 processor, read returns will be in an arbitrary order relative to the order in which they were issued on J-Bus with one exception – two reads to the same cache line will get their returns in address order.

Note – No ordering rules means that if there is a large DMA on PCI, which is translated into multiple ReadToDiscards (RDD) on J-Bus, there is no ordering guarantee based on address order. So, it would not work if software initialized a payload, immediately preceded it with a DMA_READY flag, then interacted with a device that was reading the flag with a large transaction that also prefetched following lines of payload.

1.8 Data Transfer Detail

J_ADTYPE[7:0] identifies address and data packets on J-Bus and provides additional information about coherent state and data packet size for data returns.

Flow control for data is explained in Section 1.4, “Address and Data Flow Control” on page 1-6. Flow control for both address and data is separate from the signaling of actual transfers.

Data is pushed by the J-Bus master on writes.

Error correction code (ECC) is checked and corrected inside the CPU at the memory controller. Parity is generated and checked for all data driven on J-Bus.

1.8.1 DOK Flow Control for Writes

All data is pushed by the transaction initiator.

I/O bridge and the memory controllers should unload the initial 64 bytes fast enough so DOK is kept asserted for back-to-back writes.

There is a four J-Bus cycle latency in detecting a write that will cross the high water mark in the data queues, thus making DOK==0 and requiring the processor to react to DOK==0.

All signals between chips must be registered.

Four cycles of race latency necessitate the following requirements:

- Worst case 4*16 bytes of data overrun area (versus DOK high water mark)
- Four address input queue entries for overrun (at each processor and I/O bridge)

Because the CPUs do not support any noncacheable destinations, they do not need to enqueue incoming noncacheable addresses.

Note – The OpenSPARC T1 processor supports the 64-Gbyte noncacheable destinations and aliases them to the 64-Gbyte cacheable space.

1.8.2 Read Data Flow Control

Read data flow control supports enough read buffering to hold data for all of its outstanding reads.

1.8.3 Memory Controller Reordering

Memory ordering is arbitrary for outstanding transactions, except to maintain data consistency. Reads can pass reads. Reads can pass writes only if the addresses do not match. All reads must return data consistent with the results of all prior writes.

1.9 Posted Coherency

J-Bus was originally designed to be a central snoop bus, where all devices saw all coherent transactions. To meet OpenSPARC T1 requirements, the model is modified so that J-Bus becomes a coherent I/O bus, but processor traffic is kept strictly

internal to the processor. To do this, the system is designed so all I/O caches are effectively turned off, and the I/O bridge chips issue only "posted" coherent write transactions and only use "snapshot" coherent read transactions. This means that DMA is performed using the following transactions:

- RDD – Read and Discard Line
- WRIS – Write Invalidate Line
- WRM – (New transaction) Write Merge

All coherent operations are performed at the OpenSPARC T1 L2-cache. For posted writes, the OpenSPARC T1 processor accepts the posted coherent writes (WRIS and WRM) from J-Bus and performs the writes in strict bus order in the L2- cache. Full line WRIS transactions will snoop-invalidate in the L2, but write only into main memory (to avoid cache pollution). ReadToDiscard (RDD) transactions will read coherent data from the L2 or memory (if not present in L2). The data can then be used once for DMA.

This strategy enables the OpenSPARC T1 processor to support a coherent I/O subsystem without the complexity of duplicate tags (of the I/O caches) or recalls. Performance is the same or a slightly better because the subline DMA writes will pipeline better and have reduced bandwidth impact on J-Bus. DMA read performance for I/O bridge is identical.

1.9.1 J_PACK Snoop Results

J_PACK*n*[2:0] is the bused snoop result from each J-Bus caching port to all other caching ports. The following information is returned:

- P_IDLE: 0
- P_COHACK: 1 Snoop completed, cache does not have to return dirty data. Cache state may have been changed as a side effect.
- P_COHACKS: 2 Only used by ReadToShare (RDS) snoop. Snoop completed. This cache is keeping the line in S (was S or E). Not used if cache is returning data.
- P_COHACKD: 3 Dirty line. Cache will return data to J-Bus. Memory controller will inhibit its response. This cache may change to I or O state, depending on the transaction.

Note – The OpenSPARC T1 processor will respond to all snoops immediately (without snooping any caches) with a P_COHACK.

The following four encodings are used for address and data flow control:

- P_AOK_OFF: 4
- P_AOK_ON: 5
- P_DOK_OFF: 6

- P_DOK_ON: 7

Note – The OpenSPARC T1 processor will never issue P_DOK_OFF.

The address and data flow control J_PACKs should have higher priority than the snooping J_PACKs; otherwise, the flow control model will be broken.

Specifically, the following priority should be used:

1. P_AOK_OFF (AOK off also blocks anything DOK off would block)
2. P_DOK_OFF
3. P_AOK_ON
4. P_DOK_ON
5. P_COHACK*
6. P_IDLE

1.10 DTL Mode Programming

J_PAR may need its own DTL termination mode control.

J_AD/J_ADP/J_ADTYPE can share a mode control since they should be routed the same way on the motherboard.

J_PACK0-6 has similar connectivity, so it makes sense to force it to have the same mode control as J_AD/J_ADP/J_ADTYPE.

J_REQ_IN_L[5:0] needs its own mode control, as does J_REQ_OUT_L[5:0].

1.11 Error Code Correction and Parity

All error code correction (ECC) is handled inside the CPU/memory controller ports.

Word parity is used for all J_AD transfers.

All detected uncorrectable errors (UE) and correctable errors (CE) errors are reported using interrupts. They were asynchronous traps for CPU-initiated errors in Sun4U.

Correct word parity is always driven on J_AD.

UE/CE information is passed along with the data using J_ADTYPE. This way J-Bus errors are uniquely identified with respect to memory errors. The syndrome of the memory error is not passed with the data and is logged at the memory controller that detected the error. Software should poll the memory controller after receiving an ECC trap to get the correct error syndrome.

Note – The OpenSPARC T1 processor will never indicate CE and ignores CE if signaled by I/O bridge.

1.12 J_PAR: Parity for J_REQ_L and J_PACK0-6

Most of the J-Bus signals do not have any latency requirements so complicated timing diagrams are not needed. Arbitration (J_REQ_L) does have specific timing descriptions. J_PAR is the most complicated because of the varying propagation delay for J_PACK0-6 and J_PAR in a system, which depends on which port is driving the signals and whether the port is on a local or non-local J-Bus segment.

In addition, like all bidirectional signals, we do not want to sample as input any signals that we drive, so there must be bypasses that are specific to whether or not a port is driving a particular J_PACK0-6 or J_PAR.

FIGURE 1-2 and FIGURE 1-3 specify the J_PAR generate and check pipeline. All ports should implement the pipeline as shown.

A pipeline stage is added to enable the parity generation to be distributed over two cycles to ensure that it does not become a long timing path from I/O pad input register to I/O pad output register.

I/O bridge needs 10 control status register (CSR) bits:

- Seven bits to delay JPACK0-6 by one cycle, if bit is set (0 after reset)
- One bit to delay generated parity by one cycle, before checking (0 after reset) This does not delay the output of J_PAR by the driver.
- One bit to enable J_PAR drive (disabled after reset)
- One bit to enable J_PAR checking (disabled after reset)

Depending on which J_PACK the port drives, do not clock the corresponding input register and bypass a copy of what the port drove out.

Note – The OpenSPARC T1 processor assumes unimplemented J_PACK and J_REQ signals are 0 for calculating J_PAR.

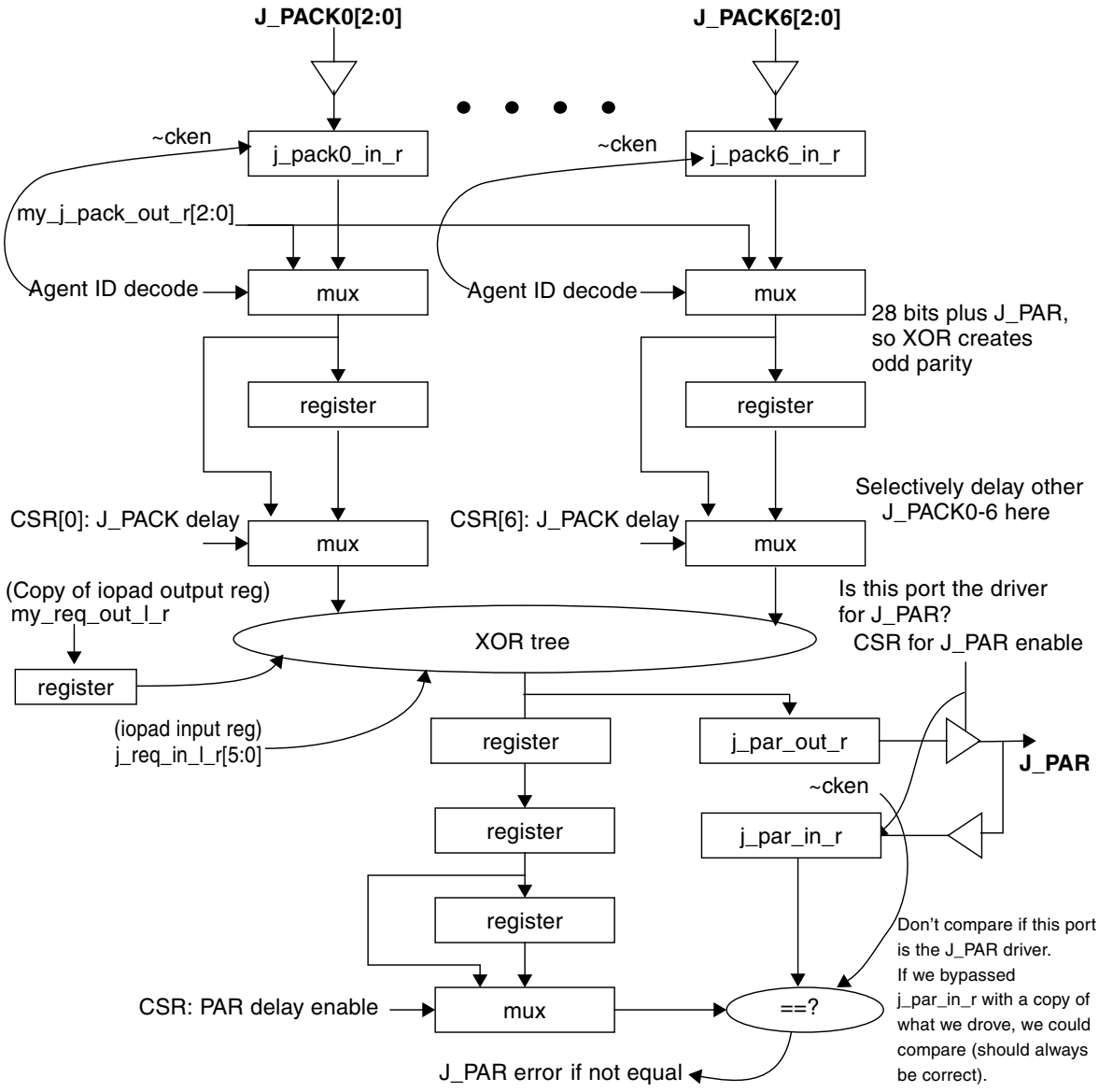

FIGURE 1-2 J_PAR Generate and Check Pipeline

This logic should be implemented on both the OpenSPARC T1 processor and I/O bridge with CSRs. It enables any of the CPUs or I/O bridges to be the J_PAR driver with arbitrary organization of which port is on which shared J-Bus segment.

At reset, no device should drive J_PAR, and J_PAR-related error checking should be disabled.

The following figure shows J_PAR generate and check timing.

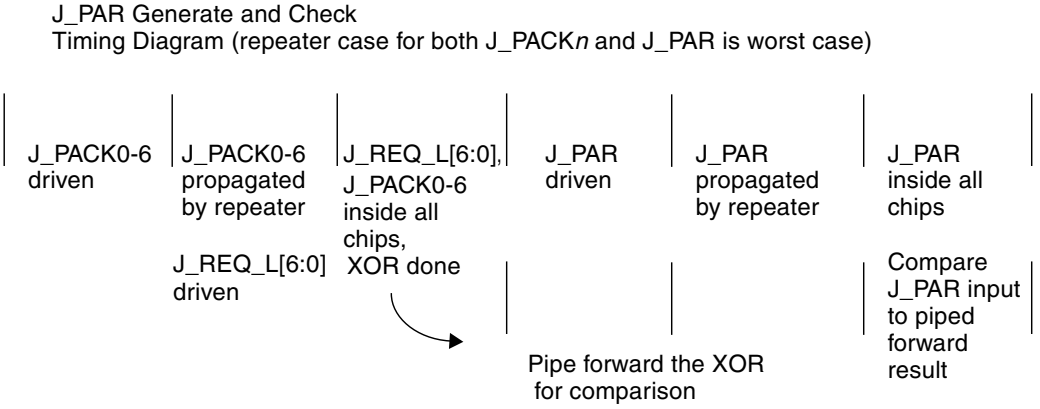

FIGURE 1-3 J_PAR Generate and Check Timing Diagram

There is CSR programmable delay of information to selectively match the repeater-created delay for chips that capture some information on a local J-Bus segment without a repeater delay. This CSR programmable delay is for J_PACK0-6 and J_PAR, but not for J_REQ_L[6:0], which are never driven by repeaters.

If all ports are on a single J-Bus segment (no repeaters), the CSRs must not delay J_PAR. The J_PACK*n* should be delayed to have uniformity of J_PAR timing with repeater-based topologies. I/O bridge can assume J_PACK*n* and J_PAR are always delayed by repeater to avoid the need for CSRs.

1.13 Fatal Errors

Address cycle parity errors, arbitration timeout, JBI to L2-cache interface timeout, and J_PAR parity errors (request and J_PACK) are considered to be fatal.

We can log the error, but we cannot just say “continue” because some ports might see one thing, and the other ports another. This is typical for intermittent errors. A static (broken wire) error might or might not be visible to all ports, but the system would stop running in that case anyway.

These fatal “not possible to continue” errors will cause a system reset.

Any error logs that are set by these fatal errors are preserved.

A separate transaction cannot be used to signal a fatal error because the bus might be hung.

Because there are no free J_PACK encodings, I/O bridge will look for four consecutive cycles of the DOK_ON (7) J_PACK encoding from any active J_PACK.

To signal I/O bridge this way for fatal errors requires that ports only assert DOK_ON for one cycle and assert a DOK_OFF before asserting another DOK_ON.

If any port detects a fatal error (address cycle parity error or J_PAR parity error), the error should be logged, J_ERR asserted (used only for debug), and DOK_ON driven as soon as possible for four consecutive cycles. This assertion of DOK_ON will cause I/O bridge to do a J_RST_L as soon as possible.

CHAPTER 2

Interconnection Model

This chapter describes the following topics:

2.1 Overview

The Peripheral Component Interconnect (PCI) model for J-Bus includes data flow, flow control, and the master interface. A J-Bus port is the interface to the single-bus interconnect.

The port is either a CPU or an I/O bridge.

A J-Bus port has a single address and data bus for the packet-switched read protocol. J-Bus supports up to seven ports, which are limited by the arbitration, and snoop and flow control pins. The OpenSPARC T1 processor uses four ports on J-Bus and only supports connecting to two other I/O bridge ports.

Because slave-only ports also must arbitrate for returning read data, this limit is not changed for slave-only ports.

Each J-Bus port should also be a slave port.

I/O bridge supports two logical J-Bus ports. There are two caching master ports and two slave ports for PCI. More than one I/O bridge is permitted in a system.

J-Bus is synchronous with a centrally distributed clock (system clock).

J_AD[127:0] is a 128-bit bidirectional packet-switched request and data bus. This bus carries address bits PA[42:0] of a 43-bit physical address space. It also carries data for reads, writes, and interrupt packets.

Note – Internally, the OpenSPARC T1 processor only supports a 40-bit physical address.

J_ADP[3:0] is word parity for J_AD[127:0] (odd: = J_ADP[0] = ~^J_AD[31:0]). J_ADP[3] should also cover the eight extra bits of J_ADTYPE. Ports receiving a data packet should check incoming parity. All ports should check parity on all address cycles.

A valid packet on the J_AD/J_ADP is identified by the driver asserting the encoded J_ADTYPE[7:0] signal. A J_ADTYPE assertion may indicate multiple cycles of data drive.

A J-Bus caching master port snoops coherent address packets on the J_AD using a write-invalidate cache-coherence protocol.

The transaction set supports block transfers of 64 bytes, cacheable writes of 0 to 64 bytes qualified with a 64-bit bytemask, and single-quadword, noncached transfers of 0 to 16 bytes qualified with a 16-bit bytemask.

Data is normally transferred in units of 16 bytes/clock cycle.

Slave ports signal read data return on the J_AD with the encoded J_ADTYPE[7:0] signal.

Byte ordering is big-endian. Bits [7:0] are bytes 15, 31, 47, and 63. Bits [127:120] are bytes 0, 16, 32, and 48.

For block read transactions of 64-byte data, the addressed 32 bytes specified by physical address bit PA[42:5] is delivered first. The successive quadwords are delivered in the wrap order shown in FIGURE 2-1.

Note that 16-byte alignment is *not* supported. Only 32-byte alignment is supported.

Address PA[5:4]	First Qword on Databus	Second Qword on Databus	Third Qword on Databus	Fourth Qword on Databus
0x0	Qword 0	Qword 1	Qword 2	Qword 3
0x2	Qword 2	Qword 3	Qword 0	Qword 1

FIGURE 2-1 Quadword Wrap Order for Block Reads on J_AD.

2.2 Data Flow

Typical data flows for read/write, read sequence for dirty encached data, noncached slave read sequence, and noncached slave write sequence are described in this section.

2.2.1 Read/Write

A typical read/write data flow to and from memory is as follows:

1. The J-Bus master port issues a read/write transaction request on J-Bus.
2. If it is a cacheable request, all J-Bus masters do a snoop on their local caches. At the same time, the addressed memory controller starts the memory cycle if the address is for main memory.
3. **Read** – If the snoop (J_PACK acknowledgements) determines that the data should come from memory, the memory controller arbitrates for J-Bus. Data is delivered to the requesting J-Bus port with an encoded J_ADTYPE[7:0] during the first cycle of the data packet.

   Write – Writeback transaction and data is sent atomically with the read. If a block store is initiating the coherent write, then snooping is done for possible invalidate. The address and data are queued at the memory control in order behind the victimizing read.

2.2.2 Noncached Slave Read Sequence

A typical noncached slave read sequence by a J-Bus port is as follows:

1. The J-Bus master port issues a read request on J-Bus.
2. After decoding the address, the addressed slave port forwards the request to the appropriate noncacheable domain controller. Typically I/O bridge will be responding to noncacheable transactions.
3. The J-Bus slave port proceeds with other work while waiting for read data to return. When data returns, the J-Bus slave port drives the data onto J-Bus to the requesting J-Bus port using an encoded J_ADTYPE[7:0].

2.2.3 Noncached Slave Write Sequence

A typical noncached slave write sequence by a J-Bus port is as follows:

1. The J-Bus master port issues a write request that includes the write data on J-Bus.
2. After decoding the address, the addressed slave port forwards the request and data to the appropriate noncacheable domain controller.

2.3 Flow Control Model

Two logical states, AOK and DOK, are used to flow control all address and data packets:

- AOK throttles address delivery for both reads and writes.
- DOK throttles data delivery for writes. DOK is tracked on a per-port basis. DOK is not used for throttling read data or interrupts.
- Some special transactions ignore AOK and DOK.

Masters should not initiate read activity without having room for returning read data.

There is a latency in seeing AOK or DOK state transition for both seeing the transition and stopping any new data. The repeater chips *add* to the latency.

The input queues at all J-Bus ports are sized to accommodate between four and eight cycles worth of overflow. Address and data queues can be sized differently. The minimum overflow requirement for the data queue depends on the target writes a port can accept (64-byte only or mixed 64-byte and 16-byte writes). See Section 1.8.1, "DOK Flow Control for Writes" on page 1-13.

The J-Bus masters are required to have enough room here for the maximum number of outstanding caching requests they support.

The address space-to-agent ID mapping is fixed to make this port-dependent DOK check easy for a master. Having fully programmable base and bounds for all addresses would require that each master track all separate base and bound registers for each port.

2.4 J-Bus Port Model

A J-Bus port is identified by a 5-bit field called the *agent ID*. There are, however, a maximum of seven ports, restricted by the number of arbitration requests and J_PACK wires.

I/O bridge can use more than one agent ID. The additional agent IDs may be useful for J-Bus to J-Bus bridge situations because the arbitration request and J_PACK limitations may apply just to a local J-Bus.

The address map and interrupt forwarding decode assumes a 5-bit agent ID. The extra bits are software programmable with a reset state that depends on the particular chip. For example, I/O bridge only has two possible agent ID pairs that it can use.

A J-Bus CPU port has two functional interface properties (master and interrupt handler). A J-Bus I/O port has the master and interrupter properties.

The CPU ports include memory controller slave ports.

2.4.1 Master Interface

The master interface property has the following characteristics:

- A J-Bus master has one outgoing request queue.
- Transactions within any destination domain are strongly ordered by the interconnect.
- A J-Bus master port is solely responsible for the ordering of its internal memory events based on its memory model.

CHAPTER 3

J-Bus Transaction Set

This chapter describes the following topics:

3.1 Transaction Set Requirements

Transaction set requirements follow these guidelines:

- Cacheable transactions are supported on 64-byte sized datums.
- Posted coherency protocol eliminates the need for atomic locks on main memory datums.
- Sun4U architecture interrupt semantics requires delivering some software-defined “opaque” state information from the interrupting source, requiring an interrupt packet with a non-blocking retry for reliable delivery.
- Sun4U system requirement of no atomic operations (CAS, SWAP) on noncached address space. Thus there is no transaction support for these atomic operations or any read-modify-writes because these operations must be performed in the cache on the J-Bus module.
- No systems requirement to support cacheable address space except main memory.
- No requirement for global cache flushing or TLB demap operations in hardware, hence no special transactions to support them. No requirement for posting interrupts to your own port or device.

- No requirement for synchronous reporting of an error on a write to a non-existent or illegal address. The writes are always accepted from the J-Bus master but are thrown away. Logging and reporting errors on writes is implementation specific and not specified by the architecture. Fatal errors, such as address parity errors, cause system reset.
- Sun4U requirement for reporting read data and time-out errors synchronously with the read transaction to the requesting J-Bus master port.

3.2 Transaction Set Terminology

The transaction set has four main categories. The asterisk (*) indicates a wildcard for all transactions having the indicated initial letters.

- **RD* (read)**
  Transaction generated by a master J-Bus port. These transactions initiate all cacheable read activity.
- **WR* (write)**
  Transaction generated by a master J-Bus port. These transactions initiate all memory write activity. A memory controller always services the transaction.
- **NC* (noncacheable)**
  Transactions used for noncacheable read/writes to PCI and I/O bridge CSRs. Although this could be inferred from the address space mapping, special transactions are used to avoid propagating all of the address space definition to all ports (NC* transactions are not snooped).
- **INT* (interrupt)**
  Generated by a master J-Bus port. Used for direct-to-CPU interrupt requests and handshakes indicating acceptance or refusal of an interrupt packet.

J_PACK
Acknowledgment generated by a J-Bus port on bused unidirectional wires from the J-Bus port to all other J-Bus ports. Generated in response to a previous RD* or WRI* transaction.

3.3 Transaction Set Summary

Cache-coherent transactions are summarized in TABLE 3-1.

TABLE 3-1 Transaction Set Summary

Transaction	TRANS[4:0]	Description
Reserved	0x00-01	
RDD	0x02	ReadToDiscard
RDD	0x03	ReadToDiscard
RDS	0x04	ReadToShare
RDO	0x06	Unsupported
OWN	0x07	Unsupported
INV	0x08	Unsupported
Reserved	0x09	
NCWRC	0x0A	NonCachedWriteCompressible
WRM	0x0B	WriteMerge
WRB	0x0C	Unsupported
WRBC	0x0D	WritebackCancelled ignored
WRI	0x0E	WriteInvalidate
WRI	0x0F	WriteInvalidate
NCRD	0x10	NonCachedRead
NCBRD	0x11	NonCachedBlockRead
NCWR	0x12	NonCachedWrite
NCBWR	0x13	NonCachedBlockWrite
INT	0x14	Interrupt Request
INTACK	0x15	InterruptAck
INTNACK	0x16	InterruptNack
XIR	0x17	Unsupported
Reserved	0x18	
Reserved	0x19	

TABLE 3-1 Transaction Set Summary *(Continued)*

Transaction	TRANS[4:0]	Description
CHANGE	0x1A	Unsupported
Reserved	0x1B-1E	
IDLE	0x1F	Idle

3.4 Memory Transactions

The section describes memory transactions.

3.4.1 ReadToShare

ReadToShare (RDS) is a request for a cache line for read access (data or instructions). The data is returned by the current owner, if one exists, or the home for the memory location.

If another cache also has this datum, Ctag transitions to S. This S versus E information is provided by J_ADTYPE during the data transfer, so this cache does not have to monitor J_PACKs to determine the cache install state.

Note – The OpenSPARC T1 Processor processor will always return shared (J_ADTYPE[2:0]
== S).

The sourcing J-Bus port must provide the addressed quadword first and wrap modulo 32 bytes based on the physical address bit AD[5] for successive quadwords as shown in FIGURE 2-1 on page 2-2.

AD[4:0] are ignored, but may be used to output diagnostic information.

If this transaction displaces a dirty victim block in the cache (Ctag state is M or O), a Writeback transaction must be paired atomically with this transaction.

The same error checking applies as for an RD transaction.

Data packet errors detected by the initiating port are the following:

- AD/ADTYPE parity error.
- Timeout.
- Wrong/illegal coherency state for install.

- Illegal ADTYPE driven in any of the data packet cycles. Not necessary to check cycles when ADTYPE provides no additional information.
- Any memory detecting a case of bad J_PACKs, where no other cache is returning data because of J_COHACKD, should return a read data error.

3.4.2 ReadToDiscard

ReadToDiscard (RDD) is a memory read with intent to discard after first use. RDD is generated by a J-Bus master when that master does not keep the line in its cacheable domain.

The data is wrapped modulo 32 bytes on AD[5] such that the addressed quadword is delivered first.

AD[4:0] is ignored, but may be used to output diagnostic information.

The same error checking applies as for an RD transaction.

Address packet errors detected by the target port are the following:

- AD/ADTYPE parity error.
- Address map violation. (PA[42]!=1), or address not in range supported by this device.
- Unused bits in AD not checked.

3.4.3 WriteInvalidate

WriteInvalidate (WRI) is a coherent write and invalidate request generated by a J-Bus master to write a data block coherently to its home location. It is used for coherent DMA writes. This transaction is used to inject new data into the coherent domain.

The data is written to memory after invalidating any copies in the L1 and L2 caches.

The write to memory can be started before all J_PACKs for the invalidates have been received, but because J_PACKs must be synchronized to maintain the in-order correspondence between them and transactions, a subsequent read will not return data until the invalidates have propagated.

In any case, this is not required, since all ports are required to complete all prior invalidates before enabling read return data to be used.

The same error checking applies as for a WRB transaction.

Address packet errors detected by the target port are the following:

- AD/ADTYPE parity error.
- Address map violation. (PA[42]!=1), or address not in range supported by this device.
- AD[5:0] nonzero.
- Unused bits in AD not checked.

Data packet errors detected by the target port are the following:

- AD/ADTYPE parity error.
- Illegal ADTYPE driven in any of the data packet cycles. Not necessary to check cycles when ADTYPE provides no additional information.

3.4.4 WriteMerge

Note – This transaction is currently only supported by the OpenSPARC T1 Processor processor and I/O bridge, although the processor never masters a WRM.

WriteMerge (WRM) writes 0–64 bytes to cacheable memory. This transaction is the same as WriteInvalidateSelf, but with 64 bits of arbitrary byte enables.

WRM has a format similar to WRI, but with a different Transaction Type and a 64-bit byte-enable mask. WRM writes 0-64 bytes in an aligned line, using a byte-enable mask, to cacheable memory.

Format is:

Address cycle:

J_AD[127:64] - Byte enables[63:0].

J_AD[63:48] -- Reserved

J_AD[47:43] -- Transaction Type == 0x0B (WRM)

J_AD[42:00] -- Address (AD[5:0] effectively ignored)

Data Cycles:

J_AD[127:0] -- Data payload, bytes 0-15.

J_AD[127:0] -- Data payload, bytes 16-31.

J_AD[127:0] -- Data payload, bytes 32-47.

J_AD[127:0] -- Data payload, bytes 48-63.

Since the WRM is a coherent transaction, it is processed in the snoop queue, and produces COHACK responses. Arbitrary encodings on byte enables (BE[63:0]) are supported. The BE[63:0] field is in the same order as on NCRD/NCWR, which appears to be that BE[0] (on the right end of the field) corresponds to Byte[0] which is on the left end of the data (J_AD[127:120]).

3.5 Noncached Data Transactions

No snooping is performed on these transactions, and data from these transactions is not cached by the requester.

3.5.1 NonCachedRead

NonCachedRead (NCRD) reads non-DRAM locations within a 16-byte region. The bytes read within the 16-byte region are specified by the byte-enable field. Devices do not need to support all combinations of byte enables.

The CPU will read 1, 2, 4, 8, and 16 bytes with this transaction. The byte location is specified with a byte enable. The address is either byte, halfword, word, doubleword, or 16-byte aligned (points to the first byte enable).

PCI can initiate reads with random byte enables. Still, the address should point to the first valid byte enable.

No snoop lookup operation is done.

Main memory cannot be read with this transaction.

Note – The 64-Gbyte noncached spaces allocated for the OpenSPARC T1 Processor processor have been aliased to the 64-Gbyte cacheable spaces allocated for the OpenSPARC T1 Processor processor. As a result, OpenSPARC T1 Processor main memory *can* be read with this transaction using the aliased address space. This aliasing is intended only for design bring-up, and not for normal operation.

Address packet errors detected by the target port are the following:

- AD/ADTYPE parity error.
- Address map violation. (PA[42]!=0) or address not in range supported by this device.
- Not checked – unused bits in AD.
- Not checked – byte-enable and PA[3:0] mismatch, or illegal byte enable.

Data packet errors detected by the initiating port are the following:

- AD/ADTYPE parity error.
- Timeout.
- Illegal ADTYPE driven in any of the data packet cycles. Not necessary to check cycles when ADTYPE provides no additional information.

3.5.2 NonCachedBlockRead

With a NonCachedBlockRead (NCBRD) request transaction, 64 bytes of non-DRAM data is read by the master J-Bus port.

This transaction is similar to NCRD, except for the following:

- No byte enable. The data is aligned on a 64-byte boundary (AD[5:0]== 0x0).
- AD[5:0] must be 0x0.

The same error checking applies as for an NCRD.

3.5.3 NonCachedWrite

A NonCachedWrite (NCWR) writes non-DRAM locations within an aligned 16-byte region. The bytes written within the 16-byte region are specified by the byte-enable field. Devices do not need to support all combinations of byte enable, but they must complete the data transfer.

NCWR is generated by a J-Bus master port to write a noncached address space, including system registers and slave address space of other slave J-Bus ports.

NCWR is written as specified by a 16-bit byte enable to slave devices that support writes with arbitrary byte enables (mainly graphics devices).

An arbitrary number of 0–16 bytes are not allowed in the byte enables. All J-Bus masters that generate NCWR must comply with the following rule:

If both byte enable[15:8] and byte enable[7:0] are non-zero, they must be equal.

This requirement enables the CPU to compress successive 8-byte stores to 16 bytes, for example.

PA[3:0] should point to the byte indicated by the first non-zero byte enable, starting with byte 0. NCRDs also follow this rule.

Exception – PCI-initiated NCWRs from I/O bridge can just forward the same address that originally appeared on PCI. For some PCI transactions, the address is 8-byte or 4-byte aligned, but the byte enables are not. This means that the first byte enable, and PA[2:0], are accurate to the word or doubleword, depending on the width of the PCI bus source.

The implication is that SPARC VIS partial-store instructions, which use 8-byte aligned addresses with random byte enables, will show up on J-Bus with addresses that are no longer 8-byte aligned.

PCI will use [3:0] for its initial address and data beat.

Interpretation of arbitrary byte enables (that is, those that are not aligned on 1, 2, 4, 8, and 16-byte boundaries) by a slave J-Bus port that does not support that interpretation is implementation specific.

No snoop lookup operation is done.

Main memory cannot be written with this transaction.

Note – The 64-Gbyte noncached spaces allocated for the OpenSPARC T1 Processor processor have been aliased to the 64-Gbyte cacheable spaces allocated for the OpenSPARC T1 Processor processor. As a result, OpenSPARC T1 main memory *can* be read with this transaction using the aliased address space. This aliasing is intended only for design bring-up, and not for normal operation.

Error Handling:

Writes fail silently and errors are reported asynchronously. NCWR is dropped by the J-Bus slave port on any type of bus or timeout error. If the J-Bus slave port drops the transaction, the J-Bus slave port may log and report the transaction using an interrupt.

Address packet errors detected by the target port are the following:

- AD/ADTYPE parity error.
- Address map violation. (PA[42]!=0), or address not in range supported by this device.
- Not checked – unused bits in AD.
- Not checked – byte-enable/PA[3:0] mismatch.

Data packet errors detected by the target port are the following:

- AD/ADTYPE parity error.
- Illegal ADTYPE driven in any of the data packet cycles. Not necessary to check cycles when ADTYPE provides no additional information.

3.5.4 NonCachedWriteCompressible

Note – The OpenSPARC T1 Processor processor does not support special handling of this transaction. If the processor is the target of an NCWRC transaction, it will be handled like an NCWR.

The NonCachedWriteCompressible (NCWRC) transaction is the same as NCWR, except downstream bus bridges are allowed to compress multiple NCWRCs to form fewer transactions with larger bursts. All NCWRCs are compressible. However, the compression logic should track J-Bus address order, so an intervening non-compressible operation (read or write) to that target space should break the compression.

3.5.5 NonCachedBlockWrite

With the NonCachedBlockWriteRequest (NCBWR) transaction, 64 bytes of noncached data are written by the master J-Bus port. The transaction is generated by a J-Bus master port for block write to a noncached address space.

NCBWR is similar to NCWR except with no byte enables. The data is aligned on a 64-byte boundary (AD[5:0] = 0x0).

The same error checking applies as for NCWR.

3.6 Interrupt Control Transactions

Interrupt control transactions are unlike a data transaction in that the 64-byte datum does not have any address space associated with it. No snooping is performed, and the datum is not cached by the requestor.

3.6.1 Interrupt Request

Interrupt (INT) request delivery is not stalled by AOK or DOK.

INT sends an interrupt. The target of the interrupt is specified in the address.

The CPUs are the only interrupt receivers and have a one-deep pipeline between acceptance and complete processing of the interrupt.

No J_PACKs are associated with this transaction.

INT is generated by an interrupter master J-Bus port for delivering packetized interrupts consisting of a 16-byte block of data to a destination J-Bus CPU port. INT is used for sending interrupts from I/O devices, reporting asynchronous events and errors to interrupt handler J-Bus ports. An INT *cannot* be sent to itself.

The interrupt transaction packet does not contain a physical address. Instead, it carries an interrupt target ID, which is the same as the target's CPU thread ID.

In the OpenSPARC T1 Processor processor, there are 32 threads which effectively act as 32 software-visible processors. To be able to target interrupts to any processor or thread, the 5-bit interrupt target ID is defined as the CPU ID (thread ID) and the OpenSPARC T1 Processor processor becomes the target of all interrupts issued onto the J-Bus.

Address Cycle:

J_AD[127:64] is a copy of J_AD[63:0].

J_AD[63:48] is reserved.

J_AD[47:43] is 0x14 (INT transaction).

J_AD[42:41] is reserved (zero).

J_AD[40:36] is the CPU ID of the target port.

J_AD[35:31] is the agent ID of the source port.

J_AD[30:0] is reserved (zero).

Data cycle 1: J_AD[127:64] has mondo data 0. J_AD[63:0] has mondo data 1.

Data cycle 2: All zeros.

Data cycle 3: All zeros.

Data cycle 4: All zeros.

The OpenSPARC T1 CPU has mondo receive registers for receipt of 16 bytes of the interrupt write packet for each thread destination.

The following rules apply at the mondo receive interface of the destination interrupt handler J-Bus port:

- If an InterruptAck (INTACK) transaction is subsequently sent from the target, the INT request has been ACKed by the destination. The sender can assume the interrupt has been sent, but complete processing by the target is not guaranteed. The only way ensure complete processing is to send another interrupt and wait for another InterruptAck. An interruptee will not InterruptAck a second interrupt, unless that interruptee has completed processing of the first interrupt.

- If an InterruptNack (INTNACK) transaction is subsequently sent from the target, the INT request has been NACKed by the destination. *NACKed* means the INT packet has been ignored because the CPU software is still processing a prior INT packet.
- The sender may try again by issuing another INT request. The retry is stateless. The target retains no information about the NACKed transaction. It is not required that the INT packet get sent again later, like PCI retry rules, but typically the INT packet will be sent again under software or hardware retry mechanisms.

  This means the sender must retain enough state to possibly retry if the INT gets NACKed.

Note – The OpenSPARC T1 Processor processor (as an interrupt target) does retain information about the NACKed transaction. Specifically, the processor will log an INTACK timeout error whenever an INT has been sent but not ACKed in a timely fashion. This is done to indicate that the OS may be having problems servicing interrupts or that a processor (thread) has become non-communicative.

The following rules apply at the master interface of the interrupter J-Bus port:

- The J-Bus port must retry later after some backoff period, if it receives an INTNACK transaction in response to the INT transaction.
- During the backoff period, the interrupt transaction *cannot* block any other transaction behind it in the master J-Bus ports class queues.

Address packet errors detected by the target port are the following**:**

- AD/ADTYPE parity error
- Unused bits in AD not checked

3.6.2 InterruptAck

InterruptAck (INTACK) delivery is not stalled by AOK or DOK.

Interrupt acknowledge means the INT-targeted CPU has accepted the interrupt and will process it as soon as possible if PSTATE.IE==1.

There is no requirement on how soon the INTACK is sent, although it should be done as soon as possible. There are no ordering requirements with respect to any other activity.

Because the interrupt is a synchronization event with respect to prior coherent write activity, the CPU synchronizes on the completion of all prior write transactions to its caches.

The CPU can do this by putting the incoming INT in its snoop queue and not setting the "mondo-receive busy" state that triggers the mondo-receive trap until the INT has reached the head of the snoop queue.

No J_PACKs are associated with this transaction.

Address Cycle:

J_AD[127:64] is a copy of J_AD[63:0] except when driven by the OpenSPARC T1 Processor processor. See Section 3.7.2, "Address Cycle" for details.

J_AD[63:48] is reserved.

J_AD[47:43] is 0x15 (INTACK transaction).

J_AD[42:41] is reserved.

J_AD[40:36] is used for the target agent ID and should be the same as the source agent ID that was initially used in the interrupt transaction that caused this InterruptAck.

J_AD[35:31] is the agent ID or CPU ID of the source port.

J_AD[30:0] is reserved.

No data packet is associated with this transaction.

3.6.3 InterruptNack

InterruptNack (INTNACK) is sent by an INT-targeted CPU if "mondo-receive-busy" state indicates the CPU already has one interrupt received that it has not processed yet.

There are no ordering requirements with respect to any other activity. InterruptNack should be sent as soon as possible.

Address Cycle:

J_AD[127:64] is a copy of J_AD[63:0] except when driven by the OpenSPARC T1 Processor processor. See Section 3.7.2, "Address Cycle" for details.

J_AD[63:48] is reserved.

J_AD[47:43] is 0x16 (INTNACK transaction).

J_AD[42:41] is reserved.

J_AD[40:36] is used for the target agent ID and should be the same as the source agent ID that was initially used in the interrupt transaction that caused this InterruptNack.

J_AD[35:31] is the agent ID or CPU ID of the source port.

J_AD[30:0] is reserved.

There is no data packet associated with this transaction.

3.7 J_ADTYPE[7:0], J_AD[127:0], J_ADP[3:0]

The J_ADTYPE bus is driven every time J_AD is driven. This bus provides additional information about the address or data on J_AD. Specifically, the J_ADTYPE bus has enough bits to identify the following:

- Address compared to data drive
- J-Bus port that is getting read return data
- Transaction ID to enable out-of-order data returns to a J-Bus port
- Ctag state to use during the install of the read data into cache

J_ADP[3:0] is driven to create odd word parity every time J_AD/J_ADTYPE is driven.

- J_ADP[0] is for J_AD[31:0]
- J_ADP[1] is for J_AD[63:32]
- J_ADP[2] is for J_AD[95:64]
- J_ADP[3] is for J_AD[127:96] and J_ADTYPE[7:0].

TABLE 3-2 J_ADTYPE[7:6] During First Cycle of Address or Data Read

J_ADTYPE[7:6]	Description
3	Address. J_AD has an address/transaction type packet, or indicates IDLE.
2	Read16 Data Return. J_AD is being used for a 1-cycle read data return.
1	Read64 Data Return. J_AD is being used for a 4-cycle read data return. The MOESI state to use on the cache install is identified in the next cycle.
0	Read Error Return. J_AD is being used for a 1-cycle read error indication.

3.7.1 Idle Cycle

J_AD is not driven if a dead cycle is inserted during bus turnaround. All receivers should only sample data when the arbiter says the bus is being driven.

During idle cycles, the current master should indicate an address cycle and drive an IDLE transaction type encoding.

- J_ADTYPE[7:0] is 0xFF.
- J_AD[63:48] is undefined (not required to be 0 or 1s).
- J_AD[47:43] is 0x1F.

- J_AD[42:0] is undefined (not required to be 0 or 1s).

3.7.2 Address Cycle

- J_ADTYPE[7:6] is 0x3.
- J_ADTYPE[5:2] is the AgentID[3:0] of the J-Bus port that is the source of the transaction. Used to return read data later and is only a debug aid for write transactions.
- J_ADTYPE[1:0] If a transaction is a read address, indicates outstanding read transaction 0 through 3. If the address is for a write transaction, should always be 0x0, with 0x1-3 encodings reserved.
- J_AD[42:0] is the physical address. Accurate down to bit 0 for noncacheables. J_AD[40:36] is used for the target agent ID during INT, INTACK, and INTNACK address transactions.

  Block read requests support 32-byte wrap around on address PA[5]. The addressed 16-byte quadword is delivered first, then the subsequent quadwords are delivered as specified by the wrap algorithm in FIGURE 2-1 on page 2-2.

  Block writes are always 64-byte aligned and PA[5:0] = 0x0.

  J_AD[4,2:0] may be used to pass diagnostic information during cacheable reads and should be ignored.

During WRI/WRIS/WRM: J_AD[4:0] should be zero.

- J_AD[47:43] is the transaction type.
- J_AD[63:48] are 16-bit byte enables, which are 0 except during NCWR and NCRD transactions.
- J_AD[127:64] are 64-bit byte enables during WRM transactions. Otherwise, J_AD[127:64] is an exact copy of J_AD[63:0].

Note – When enabled, the OpenSPARC T1 Processor processor will drive debug information on J_AD[127:64] during address cycles. For more information, refer to the chapter on Hardware Debug Support in the *OpenSPARC T1 Processor Supplement Document*.

3.7.2.1 BYTE_EN<15:0>

BYTE_EN[15:0] is only available for NCRD and NCWR transaction address cycles. This 16-bit field indicates valid bytes on the J_AD during the following data cycle.

For CPU-initiated transfers, the byte enable indicates 1, 2, 4, 8, and 16-byte noncached read requests to PCI or I/O bridge internal CSRs. It is not necessary to check that the CPU follows this rule, although it would be a good error check.

For PCI-initiated transfers, the byte enables can be arbitrary.

Arbitrary byte enables are allowed for slave writes, including the byte enable of all zeros to indicate a NO-OP at the slave. See P_NCWR_REQ in Section 3.5.3 on page 3-8.

The address should always point to the first valid byte enable for both reads and writes. Assume this is always true, although checking it and reporting an error is okay.

The byte enable is big-endian, thus ByteEnable<0> corresponds to byte 0 (bits <127:120>.

3.7.2.2 BYTE_EN<63:0>

BYTE_EN[63:0] is only available for WRM transaction address cycles. This 64-bit field indicates valid bytes on J_AD during the following four data cycles.

Arbitrary byte enables are allowed and the byte enable is big-endian. BYTE_EN[0] corresponds to byte 0 (J_AD[127:120], data cycle 1) and BYTE_EN[63] corresponds to byte 63 (J_AD[7:0], data cycle 4).

3.7.3 Read16 Data Return Cycle

There is no uncorrectable/correctable error information on these returns because Read16s are never used to return memory data.

During the 16-byte data return cycle, the following conditions apply:

- J_ADTYPE[7:6] is 0x2.
- J_ADTYPE[5:2] is the AgentID[3:0] of the J-Bus port that should receive the read data.
- J_ADTYPE[1:0] indicates return of outstanding read transaction 0 through 3.
- J_AD[127:0] has read data during read data return. During the 16-byte read return, only the data indicated by the byte enables in the read address packet must be valid. Other bytes are undefined and are not required to be 0. Correct parity is required.

3.7.4 Read64 Data Return Cycle

The four cycles of the data return cycle are described in this section.

3.7.4.1 First Cycle Read Data Return

During the first cycle of the 64-byte read data return, the following conditions apply:

- J_ADTYPE[7:6] is 0x1.
- J_ADTYPE[5:2] is the AgentID[3:0] of the J-Bus port that should receive the read data.
- J_ADTYPE[1:0] indicates return of outstanding read transaction 0 through 3.
- J_AD[127:0] has read data during read data return.

3.7.4.2 Second Cycle Read Data Return

During the second cycle of the 64-byte read data return, the following conditions apply:

- J_ADTYPE[7] is 0x0. 0x1 is reserved.
- J_ADTYPE[6:5] is 0x1 if a correctable error (in ECC) is on the first 16 bytes; 0x2, if an uncorrectable error (in ECC) is on first 16 bytes; 0x0, if there are no errors. All other encodings are illegal.
- J_ADTYPE[4:3] is 0x1 if a correctable error is on the second 16 bytes; 0x2, if an uncorrectable error is on the second 16 bytes; 0x0, if there are no errors. All other encodings are illegal.
- J_ADTYPE[2:0] indicates the MOESI state in which to install line.
  - 0x0 Invalid (NCBRD, RDD)
  - 0x1 Shared (RDS, RDSA)
  - 0x3 Unused
  - 0x5 Unused
  - 0x7 Unused
  - 0x2, 0x4, 0x6 are reserved.
- J_AD[127:0] has read data.

3.7.4.3 Third Cycle of Read Data Return

During the third cycle of the 64-byte read data return, the following conditions apply:

- J_ADTYPE[7:5] is 0. All other encodings are reserved.

- J_ADTYPE[4:3] is 0x1 if a correctable error (in ECC) is on the third 16 bytes; 0x2, if an uncorrectable error (in ECC) is on the third 16 bytes; 0x0, if there are no errors. All other encodings are illegal.
- J_ADTYPE[2:0] is 0. All other encodings are reserved.
- J_AD[127:0] has read data.

3.7.4.4 Fourth Cycle of Read Data Return

During the fourth cycle of the 64-byte read data return, the following conditions apply:

- J_ADTYPE[7:5] is 0. All other encodings are reserved.
- J_ADTYPE[4:3] is 0x1 if a correctable error (in ECC) is on the fourth 16 bytes; 0x2, if an uncorrectable error (in ECC) is on the fourth 16 bytes; 0x0, if there are no errors. All other encodings are illegal.
- J_ADTYPE[2:0] is 0. All other encodings are reserved.
- J_AD[127:0] has read data.

3.7.5 Read Data Error Cycle

The read data error cycle is one-cycle only and can be due to a 16-byte or 64-byte read request.

During the read data error cycle, the following conditions apply:

- J_ADTYPE[7:6] is 0x0.
- J_ADTYPE[5:2] is the AgentID[3:0] of the J-Bus port that should receive the read data.
- J_ADTYPE[1:0] indicates return of outstanding read transaction 0 through 3.
- J_AD[127:3] is undefined, although parity must be correct.
- J_AD[2:0] 0x0 indicates error code. Encodings are implementation dependent.

If the port does not exist for a target address, there will be no read response. The master is responsible for timing out and completing the read with error on its own.

3.7.6 Write Data Cycle

The write data cycle is always preceded by an address cycle with a transaction type that has one or four cycles of write data. The amount is implied by the transaction type. No interrupted or stalled bursts are possible. The ability to deliver the packet is only determined at the time the address cycle is driven. Changes in the DOK state after the address cycle do not affect the burst delivery.

During the write data cycle, the following conditions apply:

- J_ADTYPE[7:5] is 0x0.
- J_ADTYPE[4] is 0x1 if an uncorrectable error occurred on this 16-byte read from L2-cache, or any other error occurred in a subsystem providing the write data (I/O bridge). The memory controller will force an uncorrectable error to memory.
- J_ADTYPE[3] is reserved.
- J_ADTYPE[2:0] is 0x0.
- J_AD[127:0] has write data.

CHAPTER 4

Reset

This chapter describes the following topics:

4.1 Reset Sequence

Refer to the figures at the end of this section for definitions of M and N cycles.

The J-Bus reset sequence follows these guidelines:

- J_POR_L is asserted active low, asynchronously, and deasserted asynchronously. J_POR_L is generated asynchronously by I/O bridge, with an unclocked path from the external power supply pwr_ok signal to J_POR_L.
- J_POR_L will cause all I/Os to asynchronously disable driving, as well as protect any one-hot mux decodes.
- J_RST_L will be driven synchronously (assert and deassert) along with J_POR_L if the clock is ticking. It deasserts N cycles after the J_POR_L assertion. If this is not a power up reset (soft reset), only J_RST_L is asserted.
- After the J_POR_L deassertion, J_ID=0 will drive J_AD/J_ADP/J_ADTYPE with an IDLE transaction, so all J-Bus devices will see J_AD==IDLE when J_RST_L deasserts.

Note – The OpenSPARC T1 processor starts its phase-locked loop (PLL) lock sequence on the deassertion of J_POR_L and starts propagating clocks to internal blocks after PLL lock is achieved. This means that most of the OpenSPARC T1 processor will not get any clocks for microseconds after J_POR_L deassertion. Therefore, the processor's J-Bus output will be undefined for those microseconds after J_POR_L deassertion until PLL lock. Once clocks are being distributed, the processor will correctly drive IDLE transactions until the end of J_RST_L assertion.

- Most state is reset by J_RST_L. Additionally, J_POR_L resets the CPU memory controller.
- J_POR_L must be asserted a minimum of M J-Bus cycles after PLL lock. This timing is to allow state that is initialized by J_POR_L to get clocks. As noted previously, the OpenSPARC T1 processor does not start its PLL lock sequence until after J_POR_L is deasserted.
- Minimum assertion time of J_RST_L or J_POR_L is N J-Bus cycles.
- During J_RST_L assertion and for eight cycles after, J_PACK0-6[2:0] should be asserted (0x0) by all ports that are present, signifying their ability to receive addresses. The port ID to address decode is fixed. The termination on non-present ports should cause their J_PACKs to be 0x7.
- Every master port should set a CSR that identifies which ports are available. During reset, any non-existent J-Bus port will have the J_PACKs pulled up to 1 so this should be used to identify exists versus does not exist. Existing ports will drive J_PACKs to IDLE (0) during reset.

  This port identification method should eliminate the need for probing all ports and waiting for a timeout.

  The CSR identifying port presence is readable for verification purposes.
- Any attempts to send a transaction to an unavailable port should be inhibited and reported using a trap or interrupt and logged. Alternatively, the J-Bus master timeout times out due to no target response.
- In general, no error logs associated with J-Bus activity should be cleared by reset. This register will be cleared by explicit stores from a CPU.
- J_PACKs must have 0 or 1 at the J_RST_L deassertion to indicate presence or non-presence of a port. However, because of electrical issues, most J-Bus signals should be driven to 1 (unasserted) during J_RST_L. Because J_PACK drive is enabled by J_ID, which is a static signal, the problem is resolved by having J_PACKs only be disabled by J_POR_L, not J_RST_L.

  Remember, J_POR_L deasserts before J_RST_L. J_REQ_L could also only be disabled by J_POR_L, but since the requests are all deasserted right after J_RST_L, J_RST_L could be used to disable them, like all other J-Bus signals.

Note – J_POR_L and J_RST_L must be driven during J_RST_L by one of the I/O bridges and all ports must drive their J_PACK during J_RST_L. This prohibits using J_RST_L to tri-state those pins. J_POR_L can be used if desired, since it goes away earlier than J_RST_L. Ideally, the J_ID information is used to asynchronously control the enable of these BIDIR signals so that there should be no tri-state conflict, even when clocks are not present.

The following figures define the M cycle during system power up.

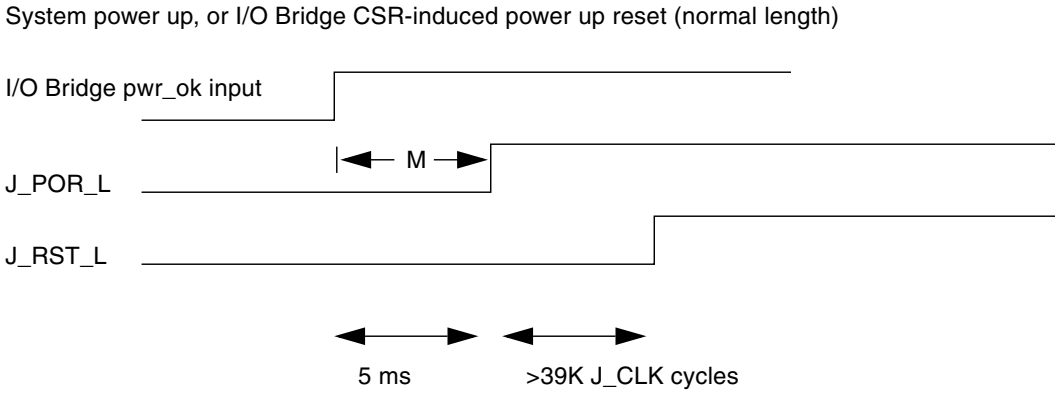

FIGURE 4-1 System Power Up (Normal Length)

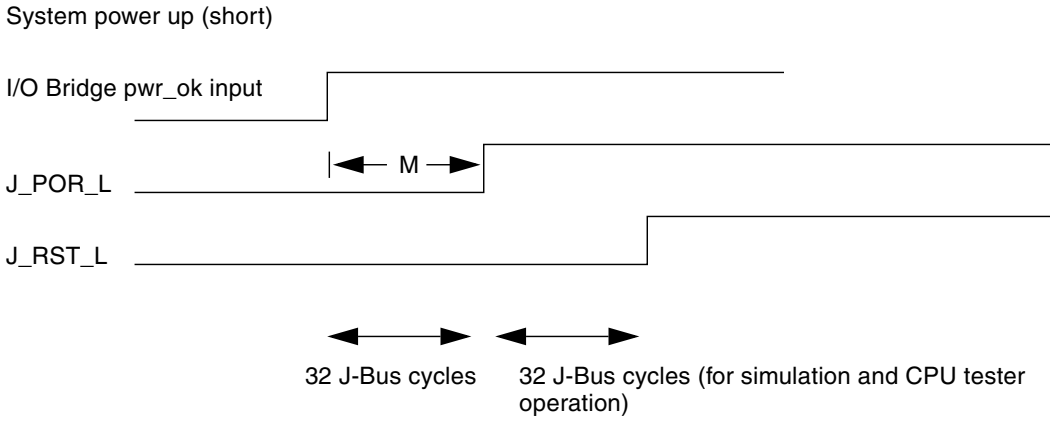

FIGURE 4-2 System Power Up (Short)

The following figures define the N cycle during reset.

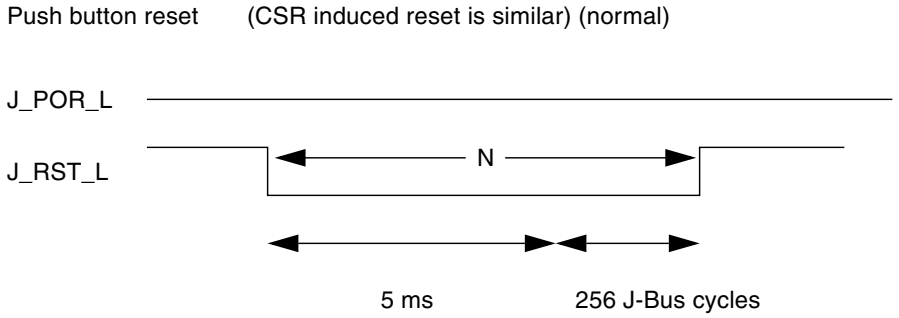

FIGURE 4-3 Push Button Reset (Normal)

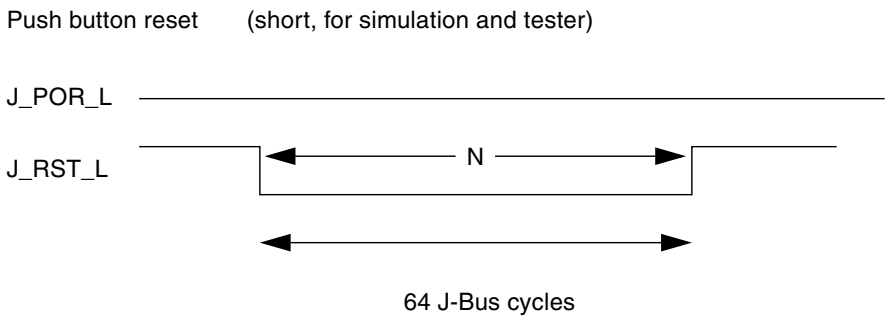

FIGURE 4-4 Push Button Reset (Short, for Simulation and Tester)

4.2 Exceptions to RST_PIN_EN

I/O bridge has a configurable J_POR_L/J_RST_L drive that depends on its J_ID, just like the J_PACKs.

J_POR_L/J_RST_L cannot be tri-stated during reset. RST_PIN_EN is not used even though the pins have tri-state drivers on a shared wire with the output enable controlled by J_ID.

On the CPU, the only bidirectional affected is J_PACK0-6[2:0]. At reset, the port drives 0 on one of them, depending on the port's J_ID (static input pins).

CHAPTER 5

J-Bus Arbitration Protocol

This chapter describes the following topics:

This chapter specifies the distributed arbitration protocol for driving any address or data onto J-Bus.

Note – Avoid livelock because it favors the last owner and all requests are forced off during AOK_OFF or DOK_OFF.

After a port causes AOK off, it should not turn AOK back on until it can accept one address transaction without turning AOK back off. This says AOK should not go back on until you have free space equal to the current overrun definition + 1 (worst case: read + writeback), rather than turning on at the current overrun definition. Use the following guidelines:

- Turn AOK off when the space available == overrun spec (4)
- Turn AOK back on when the space available == overrun spec + 2 (6).

These guidelines should solve the issue of one port constantly and solely getting the bus after AOK off/on transitions, then blocking out forever. The DOK case does not need special handling.

5.1 Overview

J-Bus can accommodate a maximum of seven J-Bus ports, limited by the number of J_REQ_IN_L signals at each port (no central request or grant arbiter).

These ports can share a common J-Bus segment or be connected with a single-cycle latency repeater. The arbitration protocol has a mode that accounts for the extra cycle of propagation latency and provides a global arbitration of all J-Bus segments.

While a turnaround (dead) cycle is available for a single-segment system, the behavior of a system with a repeater requires the DTL interfaces to allow back-to-back drives by different ports. It also requires non-driven cycles to reliably be sampled at a value of 1 by all receivers (1's have been used for all idle behaviors in the command encodings).

A distributed arbitration protocol is used for lowest latency to drive the bus. There are typically no dead cycles, unless the special dead-cycle mode is enabled using the control status register (CSR).

The arbitration is tailored to provide atomic multi-cycle access based on the assertion time of a request. It is not optimal for switching between single-cycle uses by multiple drivers (maximum of one address every two cycles, if constantly alternating drivers). Improvements to arbitration would require additional signals or compromising the multi-cycle (possibly address + data or more addresses) capabilities.

The *Last Port Driver* always drives J-Bus with an IDLE transaction, if there are no other requests.

The J-Bus arbiter will have output only signals for a port's request and N input signals for all other requests.

The other requests are hooked up in a unique way to each port depending on its agent ID. This relationship eliminates ID-dependent logic in the arbiter-critical path and is important for the I/O bridge design, which provides another level of three-way arbitration between its on-chip sources.

Agent ID values, J_REQ_IN/OUT_L and J_PACK names, and hookups are all related. The J_PACKs are related to AOK/DOK state also.

Undriven J_REQ_IN_L pins should still be hooked up to all ports as specified so that the termination can pull the pin to an unasserted value.

5.2 Possible IDs Used in a System

There are seven possible AOK/DOK states tracked by every port. Each port must track its own state.

The following table shows the agent ID values used by type of ports.

TABLE 5-1 J-Bus Agent ID Assignments

AgentID[4:0]	Possible Device	J_PACK*n*[2:0] bidir driven	J_REQ_OUT_L[5:0] (6 copies, all same)
x00yy	OpenSPARC T1 processor	J_PACK0	J_REQ0_L, J_REQ1_L
x110y	I/O Bridge 0	J_PACK4	J_REQ4_L
x111y	I/O Bridge 1	J_PACK5	J_REQ5_L
x011x	Illegal (undefined if used)		

There are 32 values in the 5-bit agent ID field of a transaction's address. The "x" don't care values cause aliasing to the seven tracked AOK/DOK states. For every ID value, there is a mapping to an AOK/DOK state.

An I/O bridge chip can generate transactions with two possible AgentID[4:0] values.

All ports should use the following table for decoding from a transaction's agent ID to AOK/DOK state.

TABLE 5-2 J-Bus Requests Assignments

AgentID[4:0]	Possible Device	J_REQ_OUT*n*_L[m] to J_REQ_IN_L[5:0] connects n specified. m is selected to distribute signals evenly for timing.
x0000	CPU 0	6 5 4 3 2 1
x0001	CPU 1	0 6 5 4 3 2
x0010	CPU 2	1 0 6 5 4 3
x0011	CPU 3	2 1 0 6 5 4
x110y	I/O Bridge 0	3 2 1 0 6 5
x111y	I/O Bridge 1	4 3 2 1 0 6
x10yy	I/O Bridge 2	5 4 3 2 1 0
x0100	CPU 4 if no I/O Bridge 0	3 2 1 0 6 5
x0101	CPU 5 if no I/O Bridge 2	5 4 3 2 1 0

"x" or "y" means the bit should not be looked at during the decode. The bit can be 0 or 1 in a transaction.

- The bits marked "x" in the AgentID[4:0] for each chip can be programmable using a software CSR or using a pin on the chip. Bit 4 is only used when decoding addresses to decide if this port is the target for the address.

- The bits marked “y” in the AgentID[4:0] can vary so a port can generate more than four outstanding read requests. Multiple agent IDs are reserved for a chip.
- Whether the port responds to all of its allocated agent ID values when decoding target addresses is implementation dependent. The recommendation is that the various agent IDs alias to a single target address space (for example, I/O bridge’s two address spaces).

CPUs can reprogram their AgentID[4:3], if necessary, with a software-accessible CSR.

There is no need for more pins on the CPU other than J_ID[2:0].

5.2.1 OpenSPARC T1 Implementation

The OpenSPARC T1 implementation of agent IDs is shown in TABLE 5-3.

TABLE 5-3 OpenSPARC T1 Implementation of Agent IDs

AID[4:0]	OpenSPARC T1 Implementation	OpenSPARC T1 Device	J_PACK	J_REQ_OUT_L
x0000	00000	OpenSPARC T1 (PIO/DMA)	J_PACK0	J_REQ0_L (2 copies)
x0001	00001	OpenSPARC T1 (PIO/DMA)	J_PACK1	N/A
x0010	00010	OpenSPARC T1 (PIO)	N/A	N/A
x0011	00011	OpenSPARC T1 (PIO)	N/A	N/A
x110y	1110y	I/O Bridge 0	J_PACK4	J_REQ4_L (6 copies)
x111y	1111y	I/O Bridge 1	J_PACK5	J_REQ5_L (6 copies)
x10yy	Not supported		N/A	N/A
x0100	Not supported		N/A	N/A
x0101	Not supported		N/A	N/A
x011x	Illegal		N/A	N/A

The OpenSPARC T1 processor will master transactions using agent IDs 0–3 to enable 16 outstanding PIO reads. The processor will respond favorably to DMA requests targeting agent IDs 0–1 (and drive J_PACK0-1 with identical information), while giving error responses to DMA requests targeting agent IDs 2–3 (and not drive J_PACK2-3). The processor will drive duplicate copies of J_REQ0_L, one for port 4 and one for port 5.

5.3 J_PACK Connectivity

Each port has seven three-bit J_PACK bidirectional ports.

Note – The OpenSPARC T1 processor has four J_PACK ports, one each for port 0, 1, 4, and 5.

A port must know which port's J_PACK is connected to which pins, so it can correctly steer AOK/DOK information to the right internal state. The connectivity must be hardwired.

Unlike the arbitration requests, whose connectivity changes for each port, the J_PACK*n* are connected the same way to all ports. A port selectively drives one of the J_PACK*n* depending on its ID.

All ports are connected identically to J_PACK6[2:0] through J_PACK0[2:0] using the bidirectional I/O pins. A port should drive a particular J_PACK*n*[2:0] based on the ID decode in TABLE 5-3.

5.4 Distributed Arbitration

A distributed arbitration protocol is specified for J-Bus to provide the lowest possible latency for bus ownership.

Requests tell others you want the bus, or that you are keeping the bus after you get it.

All requests come from output registers, and all incoming requests are registered before use. All output enables for J-Bus are registered also, so the arbitration algorithm is pipelined and uses the registered requests for the logic feeding the input of the output-enable registers.

The timing of requests is the same as any other J-Bus signal and should not be loaded more heavily than any other signal.

The arbitration protocol has the following features:

- Fully synchronous arbitration.
- Distributed protocol. All contenders simultaneously calculate the next allowed driver internally.

- Unfair round-robin among the J-Bus ports. This enables the last master to save a cycle of arbitration latency.
- The arbitration protocol enables no-dead-cycle between switching drivers and an always-driven requirement on J-Bus.
- All request signals are registered before use inside the J-Bus ports. All tri-state output enables for J-Bus are registered. This requires the protocol to be described as a pipeline, where only the state of the registered request signals can affect the driver for the next cycle, except for some special cases involved with the *Last Port Driver*.
- CPUs can see zero latency for single-cycle drives of J-Bus (read packets), if they were the last master.
- Multi-cycle packets must be driven without interruption or stalls. The duration of the request assertion controls the number of consecutive cycles.

5.4.1 Arbitration Signals

The arbitration protocol uses these signals for each J-Bus master port:

- J_REQ_OUT_L[m] output is used for the J-Bus's own request.
- J_REQ_IN_L [1:0] inputs are used for requests from other equal priority masters.
- J_RST_L. Upon reset of J_RST_L, each master J-Bus port initializes an internal state variable called *Last Port Driver* to zero. This kicks off the distributed arbitration protocol in a known state by making J-Bus port 0 the *Last Port Driver*.

Note – The OpenSPARC T1 processor supports only two J_REQ_IN_L inputs used for requests from two other equal priority masters (ports 4 and 5).

5.4.2 Arbitration Rules

The arbitration protocol has the following rules:

1. The interface currently driving J-Bus is called the *Current Driver*. Because of dead cycles, we distinguish between the *Current Driver* and the round-robin state: *Last Port Driver*.
2. After reset, the J-Bus port with agent ID == 0 is the initial *Last Port Driver*.

Note – The OpenSPARC T1 processor requires that it drives the first J-Bus transaction. All non-PLL initialization of the OpenSPARC T1 processor occurs after J_RST_L deassertion. The OpenSPARC T1 processor will not be ready to receive a transaction until that initialization is completed.

3. If no registered requests are present in this cycle, the next cycle's value for *Last Port Driver* remains the same as this cycle's value.
4. If there are no registered J_REQ_IN_L<5:0> requests, the J-Bus port that was *Last Port Driver* is the next driver for J-Bus and should drive an IDLE packet if it has no transactions.
   - If this *Last Port Driver* has a single-cycle packet, it can drive the packet without asserting its request.
   - If the *Last Port Driver* must drive a multi-cycle packet, it must assert its request and can drive in the cycle after the request. The arbitration protocol will guarantee it retains the bus in this case.
5. Otherwise, the J-Bus port will minimally see a request, wait, then drive latency.
6. The *Current Driver must* relinquish ownership of the wires by deasserting its request (if asserted) for one cycle in the presence of another J-Bus request.

   When the other requestors see this request deassertion, they will arbitrate for a new driver.

   This is a performance requirement. The *Current Driver* may be driving a single-cycle packet, or an IDLE packet, without having asserted its request. In this case, the request deassertion requirement is already met.

   The *Current Driver* should deliver one, and only one, packet before it relinquishes the bus. It should begin that packet within one cycle of the new request assertion or give up the bus. A read miss with writeback and the writeback data is considered one packet. J_ADTYPE is used to identify the valid address packet and data packets.

   Situations where more than one packet might be delivered are allowed because the latency for an external signal to affect the output flow may be longer than one cycle. However the number of packets must be finite to prevent deadlock possibilities. A request must be deasserted if it is being stalled due to flow control.

   J-Bus ports must not rely on this rule for controlling the number of transactions they receive after asserting their respective requests. That is, this rule cannot be used to flow-control incoming addresses.

   If the port has the bus, but is unable to send out the packet it wants to send (AOK off or DOK off), the port still must release its request in the presence of other requests (otherwise the AOK/ DOK situation will not get fixed).

7. If the registered *Last Port Driver*'s request is not asserted, and one or more other registered requests are asserted, arbitration happens during this cycle to decide who can drive next cycle.

   In this case, the *Current Driver* still drives J-Bus until the new *Current Driver* drives. This would be one cycle after a request deassertion. This cycle should be used and counted when determining which cycle to deassert for multi-cycle packets.

8. During an arbitration cycle, the registered request with the highest priority is determined as shown in TABLE 5-4. The value of *Current Driver* changes the next cycle to match the highest priority request.

TABLE 5-4 Arbitration Priority

Last Port Driver	Arbitration Priority Highest to Lowest
J-Bus Requestor 0	0 6 5 4 3 2 1
J-Bus Requestor 1	1 0 6 5 4 3 2
J-Bus Requestor 2	2 1 0 6 5 4 3
J-Bus Requestor 3	3 2 1 0 6 5 4
J-Bus Requestor 4	4 3 2 1 0 6 5
J-Bus Requestor 5	6 5 4 3 2 1 0
J-Bus Requestor 6	6 5 4 3 2 1 0

Last Port Driver will change the next cycle to the value of *Current Driver*.

The round-robin protocol is unfair by design, favoring the *Last Port Driver*. This feature is required because it enables the request-then-drive capability for the *Last Port Driver*. The *Last Port Driver* can drive without being dependent on possible simultaneously asserted requests.

Fairness is provided by the "release request in presence of another" rule.

9. Depending on the mode, a non-driven cycle can be enforced when switching *Current Driver*. Additionally, all ports can be conditionally disabled from looking at the switched (bidirectional) J-Bus signals during this enforced dead cycle.

TABLE 5-5 Arbitration Mode

Arbitration Mode	Effect
00	Look at J-Bus during all cycles. Dead cycle in arbiter. Only mode for Arthur system. Power-up reset mode.
01	Do not look at J_ADTYPE/J_AD/J_ADP during "turnaround" cycles. Dead cycle in arbiter. Used only for the highest frequency single-bus systems. Used if you're operating the bus at such a high frequency that dead cycles do not meet timing.
10	Look at J_ADTYPE/J_AD/J_ADP during "turnaround" cycles. No dead cycle in arbiter. Can only be used on a single-bus system.
11	Reserved

5.4.3 Timing Diagrams

The timing diagrams in this section provide examples of the arbitration protocol. The figures show how the J-Bus ownership changes from requestor to requestor.

The figures show the minimal arbitration latencies, which are as follows:

- 0 cycles if J-Bus port is *Current Driver* or the J-Bus port is *Last Port Driver* and a single-cycle packet is driven (clean read request).
- 1 cycle if J-Bus port is the *Last Port Driver* and a multi-cycle packet must be driven.
- 2 cycles if not the *Last Port Driver.*
- 3 cycles if *Current Driver* must be forced off.

The following figures show a single J-Bus segment's arbitration. Dead-cycle mode is not shown, so the case of address and data being propagated by the repeater one cycle late is not shown.

When the repeater is used, the arbitration requests arrive at all ports with the same timing as for a single J-Bus segment (no repeater for arbitration requests).

When the repeater is used, the following differences apply:

- Use of the repeater requires a dead cycle to be inserted in the arbitration control when switching from one owner to another. This enables the first owner to have global ownership of all J-Bus segments for one additional cycle, which covers the time the repeater requires to propagate any drive from the first owner's J-Bus segment to all other J-Bus segments.
- Information (address, data, and control) may or may not take an extra cycle to propagate to all ports.
- The "undriven" state of the bus is all 1's, which is interpreted as an idle cycle, so it's okay to have some indeterminism in the packet propagation with respect to ownership of the wires controlled by arbitration.

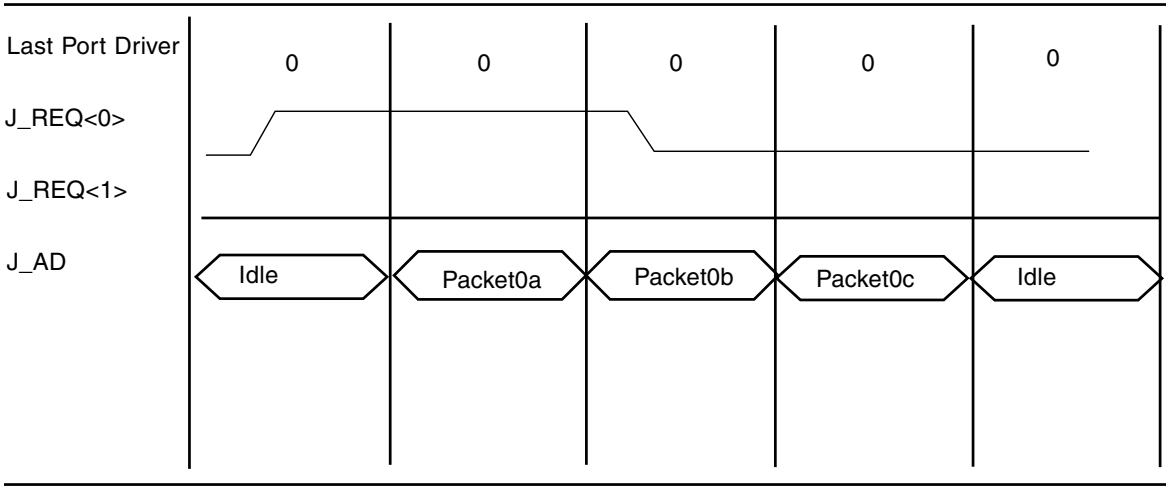

FIGURE 5-1 Arbitration: J-Bus Port 0 Drives Multi-cycle Packet in the Absence of Another Request

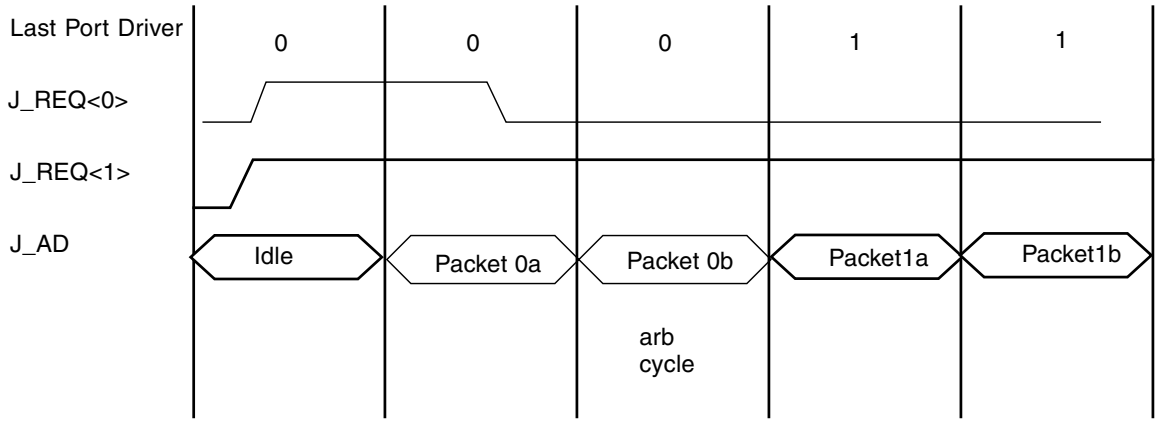

FIGURE 5-2 Arbitration: *Current Driver* Retains Ownership at the Latest Possible Cycle

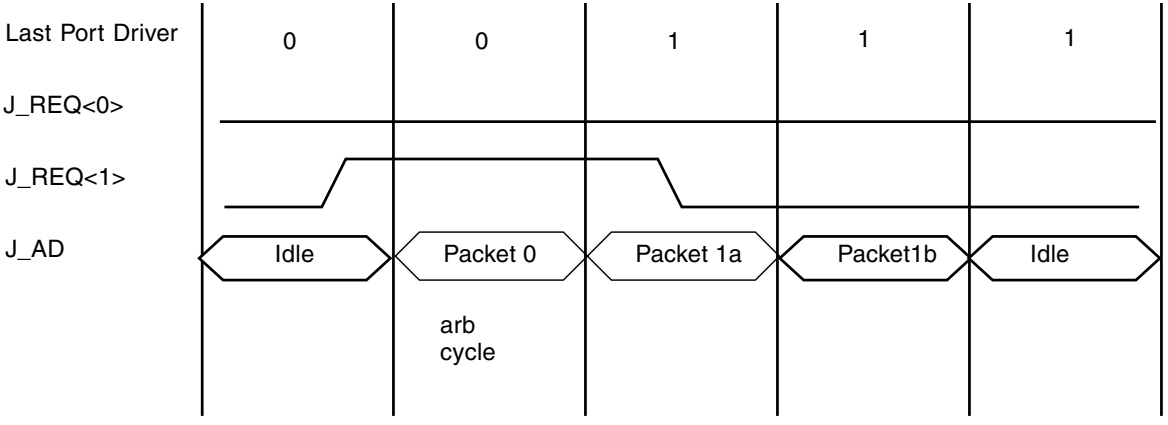

FIGURE 5-3 J-Bus Port 0 Drives Single-Cycle Packet Without Asserting Request, Right Before J-Bus Port 1 Drives a Multi-cycle Packet.

CHAPTER 6

J-Bus Electrical Specification

This chapter describes the following topics:

6.1 Overview

J-Bus is a shared multi-master bus with up to seven ports, built up from individual bus segments with at most three loads, running at 120–200 MHz with 0 or 1 dead cycles when switching drivers.

Two chips will have DTL driver and receivers – the CPU and the I/O bridge application-specific integrated circuit (ASIC).

The approximately 170 J-Bus signals are equally loaded, except for the arbitration signals. J_RST_L is propagated asynchronously and may have different loading, so it is not a DTL signal.

Minimally, a CPU connected to an I/O bridge will be present. Multiple I/O bridges could be present.

TABLE 6-1 Other Non-DTL signals

Signal	Description
J_ID[1:0]	Static signal
J_CLK[+-]	PECL clocks

6.2 Configurations

As shown in FIGURE 6-1, the CPU will always be at one end of the J-Bus and the I/O bridge ASIC will always be at the other end. Zero through three other connections will be between the two endpoints, with matched 2.5 inch stub lengths. DTL Scheme 1 drivers are used. All signals are active low. VRef = 1.125V.

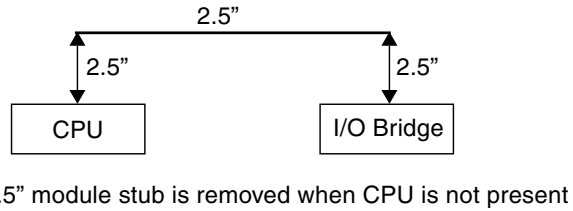

FIGURE 6-1 J-Bus Top-Only Motherboard

6.2.1 Topology Restrictions

The following topology restrictions apply:

- Middle drivers are configured in Mid-driver mode, 25-ohm pull-down, 50-ohm pull-up, tri-state during receive.
- End drivers are configured in End-driver mode, 50-ohm pull-up/pull-down, 50-ohm pull-up during receive.
- Maximum length is 15 inches (does not include intermediate stubs).
- Via loading <= 1.0 pF.
- Minimum via spacing is 1.5 inches.
- Trace impedance range is 50 ohms +- 5 ohms.
- Connector: pin-impedance controlled +- 10 ohms. Delay must be included in the 15-inch delay.

6.2.2 Noise Margins

All DTL HSPICE modeling will have a minimum margin of 200 mV relative to the receiver's voltage reference (Vref) and include the biggest noise contributors – simultaneous switching output (SSO), reflections, and calibration. The only AC noise contributors not included are crosstalk and Vref noise. Crosstalk should be no more than 8% of the signal swing. Vref noise is limited to 50 mV.

6.2.3 DTL Scheme 1

DTL Scheme 1 uses nominal 50-ohm pull-up drivers. Pull-down drivers are nominally 50 ohms at the ends and nominally 25 ohms in the middle.

On the CPU, two bits (down_25 and up_open) are available to program the devices per system configuration (hardwired to J_ID[2:0], so initial instruction fetch works).

6.2.4 Output Driver Topology

Output drivers are push-pull, with primarily NMOS pull-down and PMOS pull-up transistors. An NMOS pull-up may be used if the technology permits the gate of the driver to be overdriven with respect to VDDO.

It may be necessary to augment the primary pull-up and pull-down to improve the linearity of the switching current over the voltage range.

6.2.4.1 Pull-up Characteristics

Nominally 50-ohm drivers to VDDO are used. These drivers are present on every device on the bus. The line should be terminated to 50 ohms nominal at the two ends.

An exception occurs when an end device drives low. In this case, single-parallel termination is provided by the pull-up at the other end. DC high level is 1.5 V nominal.

6.2.4.2 Pull-down Characteristics

Pull-down drivers are nominally 50-ohm or 25-ohm drivers to VSSO.

Pull-down drivers are configurable based on J_ID[2:0]. 50-ohm pull-downs are restricted to the ends of the J-Bus (CPU 0 and I/O bridge).

When a device at the end of the J-Bus pulls low, its corresponding pull-up driver is disabled. However, the pull-up at the other end is enabled and presents a nominal 50-ohm termination to the signal. Current drive is 15 mA nominal.

Devices in the middle have pull-down drivers with nominal 25 ohm impedance. When a middle driver is enabled, all pull-up drivers are disabled, except for those at the ends. The current drive is 30 mA nominal. The DC low level is 750 mV nominal.

6.2.4.3 Characteristics When Inactive

Drivers on the two ends pull up the J-Bus to VDDO with 50-ohm nominal impedance, when not in drive mode. All other devices are tri-state.

6.2.4.4 Driver Impedance Control

Compensation is used to stay close to nominal value over all process, supply voltage, and temperature variations. Impedance variations are controlled within 6.6% of nominal, for both 50-ohm drivers and 25-ohm pull-downs.

The variation of driver/termination impedance over the operating voltage range on the bus must be kept to within 10% of its value when Vbus = VDDO/2 at all values of bus voltage within the operating range.

6.2.4.5 Driver Impedance Calibration

Driver impedance is calibrated by an on-chip reference during normal operation of the chip with minimal impact on signal integrity. A maximum calibration noise of 50 mV is specified.

6.2.4.6 Termination Characteristics

The pull-up drivers on the end devices can function as terminators, as described prevously. It is possible to have separate driver and terminator circuits, as long as they are not enabled simultaneously.

6.2.4.7 Input Receiver Characteristics

Input receivers must be designed as an analog comparator with Vref supplied from an external source.

6.3 Power Implications

This section provides power supply details.

6.3.1 1.5 V Supply

DTL logic uses 1.5 V supply delivered by a regulator from +5V.

Tolerance is +-5%, inclusive of DC and dynamic regulation specs. To achieve this, the converter will must meet a +-3% tolerance with significant bulk capacitance used on the board.

A helper linear regulator might be needed to improve the converter response time.

CHAPTER 7

Address Map

This chapter describes the following topics:

7.1 Address Map Table

The OpenSPARC T1 processor supports 40-bit physical addresses, whereas J-Bus is defined to support 43-bit physical addresses. Thus, the OpenSPARC T1 processor must perform some address mapping so that software will be able to access registers on I/O bridge and PCI addresses.

TABLE 7-1 OpenSPARC T1 Processor Address Map

Min	Max	J-Bus Address	Usage
00_0000_0000	1F_FFFF_FFFF	same	Memory
00_0000_0000	0F_FFFF_FFFF	60x_xxxx_xxxx	NC J-Bus range that aliases to memory
10_0000_0000	1F_FFFF_FFFF	61x_xxxx_xxxx	NC J-Bus range that aliases to memory
80_0000_0000	80_00FF_FFFF	Not applicable	JBI internal CSRs; maps to addr range for AID=0x00 and AID=0x01.
80_0E00_0000	80_0E7F_FFFF	400_0Exx_xxxx	J-Bus AID=0x1c
80_0E80_0000	80_0EFF_FFFF	400_0Exx_xxxx	J-Bus AID=0x1d
80_0F00_0000	80_0F7F_FFFF	400_0Fxx_xxxx	J-Bus AID=0x1e
80_0F80_0000	80_0FFF_FFFF	400_0Fxx_xxxx	J-Bus AID=0x1f

TABLE 7-1 OpenSPARC T1 Processor Address Map *(Continued)*

Min	Max	J-Bus Address	Usage
80_1000_0000	80_FFFF_FFFF	600_xxxx_xxxx	Fake DMA range; aliases back to memory, using NC-memory range for I/O bridge.
81_0000_0000	BF_FFFF_FFFF	n.a.	OpenSPARC T1 Processor internal CSRs
C0_0000_0000	CF_FFFF_FFFF	7Cx_xxxx_xxxx	J-Bus AID=0x1c
D0_0000_0000	DF_FFFF_FFFF	7Dx_xxxx_xxxx	J-Bus AID=0x1d
E0_0000_0000	EF_FFFF_FFFF	7Ex_xxxx_xxxx	J-Bus AID=0x1e
F0_0000_0000	FE_FFFF_FFFF	7Fx_xxxx_xxxx	J-Bus AID=0x1f
FF_0000_0000	FF_EFFF_FFFF	n.a.	SSI Internal registers
FF_F000_0000	FF_FFFF_FFFF	n.a.	Boot ROM, SSI-addressable locations

7.2 Three Reserved Address Spaces Per Port

All J-Bus devices must support an 8-Mbyte noncacheable address space located at physical address PA{42:41] = 2'b10, PA[40:28]==0, PA[27:23] = AgentID[4:0], PA[22:0]==<the 8-Mbyte range>.

All J-Bus devices have a reserved 64-Gbyte noncacheable address space located at physical address PA{42:41] = 2'b11, PA[40:36] = AgentID[4:0] PA[35:0]=<the 64-Gbyte range>. It is preferred that all ports accept transactions to their reserved space, and log and report errors, although not responding is okay.

All J-Bus devices have a reserved 64-Gbyte cacheable address space located at physical address PA[42:41] = 2'b00, PA[40:36] = AgentID[4:0] PA[35:0]=<the 64-Gbyte range>. It is preferred that all ports accept transactions to their reserved space, and log and report errors, although not responding is okay

Note – The OpenSPARC T1 processor aliases its 64-Gbyte noncacheable space to its 64-Gbyte cacheable space.

PA[42:41] == 2'b01 indicates an undefined address space, but it is okay if the cacheable address spaces alias to it. Cacheable address decode can just look at PA[42]==0.

Additional mapping registers should exist on each port to subdivide regions of noncacheable space as well as cacheable (memory) space. These registers are necessary to steer transactions to different I/O buses on the I/O bridge, for example.

Note – By convention, noncacheable space has PA[42] = 1, and cacheable space has PA[42] = 0.

7.3 Address Mapping Errors

If ports receive a transaction or address they do not support, they are responsible for returning a read error packet on reads.

For errors on writes, I/O bridge will log and send an error interrupt to a CPU.

CPUs will cause a deferred error trap if they receive an unsupported store (like a copyback tag parity error). So one CPU sending a bad address, mapped to another CPU, will cause that second CPU to trap. The store completes silently otherwise.

CPUs will return a read error packet for unsupported read addresses.

If PA[42] usage does not match the convention for cacheable/noncacheable, when compared to the transaction type, that is also treated as an error.

The addressed port will always respond for a read and log/report at the first bad store.

There should be no bad side-effects or address alias effects for ports receiving bad load or store addresses.

Note – If the agent ID does not map to a port that is present in the system, the transaction ideally should not be issued on J-Bus. A CSR at reset is loaded with information about all present ports (ports which assert J_PACK==0x0 during reset). This error should be reported using a trap or interrupt. Alternatively, J-Bus master timeout can occur.

CHAPTER 8

Serial System Interface

This chapter describes the following topics:

8.1 Overview

The Serial System Interface (SSI) is defined to allow microprocessors to access peripherals in a low pin count method. The OpenSPARC T1 Processor processor will not directly interface to peripherals but instead provides an interface that can be easily converted to peripheral protocols by an external programmable logic device (PLD). Isolating the OpenSPARC T1 Processor chip from these peripherals enables the devices to use higher voltage signaling and provides a mechanism for protocol conversion.

The OpenSPARC T1 processor's SSI interface can be used to interface to a complex programmable logic device (CPLD) or to a complex field programmable gate array (FPGA). CPLD can be as simple as SSI interface on one side and flash PROM interface on other side. FPGA can include more functions such as an RS232 UART or a system management microprocessor with a dedicated parallel interface to flash PROM. All of these peripherals would be memory mapped into the 256 Mbyte SSI addressable location area (FF_F000_0000 - FF_FFFF_FFFF). All devices accessible off the SSI interface will be only targets. The OpenSPARC T1 processor will always be the master of the bus.

8.2 Functional Interface

The SSI interface includes three pins – SSI_SCK (clock), SSI_MOSI (master out/slave in), and SSI_MISO (master in/slave out). SSI_CLK and SSI_MOSI are outputs of the OpenSPARC T1 Processor processor. SSI_MISO is an input. The SSI_SCK is a free running clock, toggling whenever the on-chip J-Bus clock is toggling. It is assumed to be nominally 50 MHz, but is always a divide by 4 of the J-Bus clock. No other divide ratios are possible.

8.2.1 SSI Request

An SSI request is transmitted on the SSI_MOSI line. The request can be either a read command or a write command. The format of all these requests is one start bit, a three-bit command (CMD[2:0]), a 28-bit address, 0–64 bits of data, and a parity bit. The high order (most significant) bit within the command, address, and data are always transmitted first, with the low order bit transferred last. Zeros are transmitted as a low voltage value and ones are transmitted as a high value. A start bit is a high value.

CMD[2] is 0 for write; 1 for read.

CMD[1:0] encodes the transaction size as follows:

- 2'b00-1 byte
- 2'b01-2 byte
- 2'b10-4 byte
- 2'b11-8 byte

For every SSI request, an SSI response is expected. A succeeding request cannot be sent until the preceding request has received a response. (No command pipelining is supported.)

When the OpenSPARC T1 Processor processor has no request to transfer or is waiting for a response, the SSI_MOSI line is held in the low voltage state.

The parity bit is set so that the number of ones in the start bit, the command, the address, any data bits, and the parity bit is an even number.

8.2.2 SSI Response

An SSI response is received on the SSI_MISO line. The response can be either a read response, which must contain data, or a write response, which must not contain any data. The format of a read response is one start bit, 8–64 data bits, and one parity bit. The format of a write response is one start bit and one parity bit. The high order (most significant) bit within the data is always transmitted first, with the low order bit transferred last. Zeros are transmitted as a low voltage value and ones are transmitted as a high value. A start bit is a high value.

The parity bit is set so that the number of ones in the start bit, any data bits, and the parity bit is an even number. This means a write response is two ones in consecutive cycles.

When the target has no response to transfer or is processing a request, the SSI_MISO line is held in the low voltage state.

8.3 SSI Software Interface

Addresses within the SSI address range (0xFF_F000_0000 to 0xFF_FFFF_FFFF) are issued to the off-chip SSI interface bus. The only transactions that are supported directly to the SSI interface are the following:

- 1, 2, 4, 8-byte aligned reads
- 1, 2, 4, 8-byte aligned writes

Because the Boot ROM is predominantly used for instructions, which is explicitly always big-endian, all accesses to the SSI interface bus are treated as big-endian.

8.3.1 SSI Register Interface

The SSI registers all deal with error handling and are described in the "Error Handling" chapter of the *UltraSPARC T1 Supplement to UltraSPARC Architecture 2005*.

8.3.2 SSI Error Handling

The following table describes the SSI's handling of errors. The error indication on read returns is delivered regardless of the ERREN bit, where it is up to the processor to ignore the error or receive it. Logging the error and sending an error interrupt are controlled by the ERREN bit. Note that returning zeros on an I-fetch timeout will tend to cause an illegal instruction trap.

TABLE 8-1 SSI Error Handling

Error	TType	Severity	Logs	Returns	ERREN
SSI Parity Error	Read	Uncorrectable	Just the bit	Data, with error indication	Asynchronous Interrupt
SSI Parity Error	Write	Uncorrectable	Just the bit	Not applicable	Asynchronous Interrupt
SSI Timeout	Read	Uncorrectable	Just the bit	All zeros, with error indication	Asynchronous Interrupt
SSI Timeout	Write	Uncorrectable	Just the bit	Not applicable	Asynchronous Interrupt

8.3.3 SSI Interrupts

SSI generates interrupts for two reasons – either the EXT_INT_L pin was asserted, or an error was detected.

The external interrupt pin is intended to be used by the FPGA and has *no* ordering protection, meaning when EXT_INT_L is asserted, an interrupt is issued to the IOB, without checking any transactions in flight. The interrupt is delivered to the IOB using the SSI device ID, that is, (device ID == 2).

EXT_INT_L is treated as an asynchronous input, meaning the JBI must synchronize it to its internal clock before using it. Also, EXT_INT_L is treated as an edge-triggered interrupt, meaning that JBI will detect a rising edge on the synchronized signal, and issue an interrupt to the IOB on those rising edges. If the actual use is level-sensitive, software is responsible for querying the FPGA device (or whatever is driving EXT_INT_L) to see if the interrupt is still asserted at the end of the interrupt handler.

To guarantee being seen, EXT_INT_L must be asserted for at least 4.5 J-Bus cycles.

Error interrupts, when enabled, are delivered to the IOB using the error device ID, (device ID == 1).

Glossary

agent ID 5-bit encoding that uniquely identifies a specific J-Bus port in a system. Arbitration and other point-to-point signals limit local J-Bus connectivity to seven ports.

AOK Global address OK. State maintained by each requestor port to decide if there is room for address data at the targets for a transaction. The transaction is not driven on the bus if there is no room at the destination.

ASI address space identifier

ASIC application specific integrated circuit

ASR ancillary state register

atomic A load and store pair with the guarantee that no other memory transaction will alter the state of the memory between the load and the store.

big-endian An addressing convention. Within a multiple-byte integer, the byte with the smallest address is the most significant. A byte's significance decreases as its address increases.

CAM content-addressable memory

CE correctable error

CMP chip-level multiprocessor or chip-level multiprocessing

copyback The act of a J-Bus caching master sourcing its cache block to another J-Bus master requesting it.

CREG control register

CSR control status register

Ctag The cache tags of the coherent cache in a master J-Bus port. The Ctags maintain the five MOESI cache states and participate in the J-Bus cache coherence protocol.

CTU clock and test unit

data block 64 bytes on the 128-bit data bus. Four quadwords are transferred, one quadword per clock cycle. The byte order is big-endian. In WRM transactions, valid bytes are identified with a 64-bit bytemask.

dead-cycle mode Used to reserve the global J-Bus for one more cycle than the ownership of a local bus segment to account for the J-Bus repeater delay.

DDR2 SDRAM double data rate-synchronous dynamic random access memory

DFT design for testability

DIMM dual inline memory module

dirty victim A dirty cache block which is victimized (displaced) by a cache miss.

DMA direct memory access

DOK Port and address-range specific data OK. State maintained by each requestor port to decide if there is room for write data at the targets for a transaction. The transaction is not driven on the bus if there is no room at the destination.

DTL dynamic termination logic

ECC error correction code

e-Fuse electronic fuse

FPU floating-point unit

HSTL high-speed transistor logic

IOB I/O bridge

ISI intersymbol interference

invalidate To nullify a cache state.

J_ADTYPE Signal identifies the packet type on J_AD/J_ADP and signals the destination for the returning read data. Passes UE/CE information along with the data. Includes read transaction ID information.

J_AD Signal used for all address and data packets.

J_ADP Signal used for word parity for all J_AD transfers.

JBI J-Bus interface

J_PACK Acknowledgment generated by a J-Bus port on bused unidirectional wires from the J-Bus port to all other J-Bus ports. Generated in response to a previous RD* or WRI* transaction.

J-Bus master port A J-Bus port that can initiate transactions on the interconnect.

little-endian An addressing convention. Within a multiple-byte integer, the byte with the smallest address is the least significant. A byte's significance increases as its address increases.

latency The amount of time (often measured in clock cycles) between a request to read memory, and when it (the request or the target data in memory?) is actually output.

livelock The condition when one port constantly and solely gets the bus after AOK off/on transitions and blocks out forever.

MMU Memory Management Unit

MOESI Refers to cache states – exclusive modified (M), shared modified (O), exclusive clean (E), shared clean (S), and invalid (I).

mondo vector A large collection of encoded interrupts (64x8).

ODT on-die termination

PA physical address, as in PA{35:32}.

PCI peripheral component interconnect

PECL pseudo emitter-coupled logic

PIO programmable input/output

PLL phase-locked loop

POR power-on reset

quadword 16 bytes. The byte order is big-endian. In noncached read/write transactions, valid bytes within the quadword are identified with a 16-bit bytemask.

RAS reliability, availability, serviceability

SMP symmetric multiprocessor

snooping The act of looking up the Ctags to determine the state of a cache block.

SSI Serial System Interface

SSI_MOSI SSI master out/slave in pin

SSO simultaneous switching output

system clock The J-Bus interconnect clock that is centrally distributed to all J-Bus ports and is within the interconnect.

TAP test access port

TLB translation lookaside buffer

UE uncorrectable error

Index

O

P

Q

R

S

T

W