JBus Architecture Overview

Technical Whitepaper **Version 1.0, April 2003**

This paper provides an overview of the JBus architecture. The Jbus interconnect features a 128-bit packet-switched, split-transaction request and data bus that delivers the high bandwidth and low latency needed for network communications and high performance applications. The bus features a flexible, extensible and easily implemented structure that supports multiple processors and multiple I/O bus bridges.

Sun microsystems

4150 Network Circle
Santa Clara, CA 95054 USA
http://www.sun.com

CHAPTER 1

Introduction

In a multiprocessor system, system interconnect is a key determiner of the cost, performance, and reliability of shared-memory, cache-coherent multiprocessors. The bandwidth of the interconnect is an important factor in determining the system performance. The architecture and implementation of the interconnect is also critical to the performance and scalability of multiprocessor systems.

Because system interconnect is so important, Sun has architected a new 128-bit wide, high speed interconnect, called “JBus”, that provides packet-switched, split-transaction features. JBus enables a board or multi-chip module to be configured as a 1-4 ways system. It also enables shared memory with low latency and fast cache coherency.

Sun has developed a powerful new low cost chip set that implementing JBus for 1 to 4 way multiprocessor systems. The high density chip set synthesizes a number of design innovations which combine to deliver new levels of performance and reliability for existing and newly ported software applications.

The system is completely compatible with the Sun’s robust Solaris operating system, which supports multiple processors.

The JBus takes additional advantage of the on-chip density to achieve high bandwidth interconnect. Thus the JBus can optimize the performance of a small symmetric multiprocessing system without hard latency or protocol throughput constraints. The dual bus structure supports both the high speed inter-processor system bus, and the lower speed peripheral interfaces.

Sun intends the JBus interface to accommodate a wide range of peripherals, so that designers can quickly and easily configure a system.

This paper focuses on Sun’s latest bus-based interconnect architecture - JBus.

CHAPTER 2

Shared Memory Multiprocessors

Small-to-medium-sized shared memory multiprocessors are the dominant form of parallel architecture seen today. This architecture provides a global physical address space and symmetric access to all of main memory from any processor, often called a symmetric multiprocessor (SMP). Every processor has its own cache, and all the processors and memory modules attach to the same interconnect, which is usually a shared bus. SMPs dominate the server market and are becoming more common on the desktop. They are also important building block for large-scale multiprocessor systems.

In a computer system, the various subsystems interface to one another. The memory and processor need to communicate, as do the processor and the I/O devices. This is commonly done through either a bus or a general interconnection network which serves as a shared communication link between the subsystems. The following sections examines the bus-based multiprocessor systems in detail.

2.1. Bus-based Multiprocessor Systems[1],[2],[3]

In the bus-based interconnect**,** the interconnect is a shared bus located between the processors' private caches and the shared memory subsystem. This bus-based shared memory approach is used in most small scale (1-4 processors) parallel multiprocessor systems sold today.

A bus is a convenient device for ensuring cache coherence because it allows all processors in the system to observe ongoing memory transactions. If a bus transaction threatens the consistent state of a locally cached object, the cache controller can take appropriate actions to invalidate the local copy. Protocols using this mechanism to ensure coherence are called snoopy protocols because each cache snoops on the transactions of other caches. Cache coherence will be discussed in more detail in section 2.2.

2.2. Cache Coherency[1]

In a shared memory multiprocessor system with a separate cache memory for each processor, it is possible to have many copies of shared data; one copy in the main memory and others in local cache memories. With multiple copies of data in the system, any local modification of the data can result in a globally inconsistent view of the data. This is known as the cache coherence problem.

A cache coherence scheme is the discipline that ensures that changes in the values of shared data are propagated throughout the system in a timely fashion. The choice of a cache coherence scheme is the most important design decision for a coherent shared-memory interconnect system.

The cache coherence scheme keeps each processor's view of memory consistent. Coherency is maintained on aligned blocks of memory, called cache lines, which are typically between 32 and 128 bytes wide. Sun currently uses a 64-byte cache block.

2.2.1. Snoop-based Protocol[1]

The interconnect is a shared bus located between the processor's private cache and the shared main memory subsystem. The protocol relies on each processor monitoring all requests to memory. This monitoring, or snooping, allows each cache to independently determine whether accesses made by another processor require it to update its caching state. There are several key properties of a bus which support coherence.

First, all transactions that appear on the bus are visible to all cache controllers. Second, they are visible to all controllers in the same order (the order in which they appear on the bus).
A coherence protocol must guarantee that, for all transactions that appear on the bus, the controllers take appropriate action.

CHAPTER 3

JBus Features

JBus is designed to be a powerful, and reliable interconnect to meet the needs of many applications. The following presents the detail features of this interconnect.

3.1. JBus Interconnect Flexibility

JBus delivers the high bandwidth and low latency needed for networking, communications and other embedded applications. Built for the bandwidth hungry Internet infrastructure, JBus targets multiprocessor 64-bit servers. It features a 128-bit packet-switched split-transaction request and data bus It is a flexible, extensible and easily implemented bus structure. Processors can attach to a coherent shared bus without any glue logic. JBus is capable of providing scalable (1-4 ways) Symmetric Multi-Processor (SMP)[1] or Chip Multi- Processor (CMP) configurations with a selection of data bus widths, power, space and cost options.

JBus may be used in a variety of platforms, though this paper uses UltraSPARC® IIIi as an example.

3.2. JBus Coherency

JBus supports a snoop-based (broadcast) cache coherence protocol. In this protocol, all addresses are sent to all system devices. Each device examines (snoops) the state of the requested cache line in its local cache, and the system determines the global snoop result a few cycles later. Coherency is maintained on aligned blocks of memory, called cache lines, which are typically between 32 and 128 bytes wide. Sun currently uses a 64-byte wide cache block.

3.3. Details of the Protocol

The protocol is a snoop-based, write-invalidate protocol. Like most protocols, cache misses are satisfied from memory, unless a system device (processor or I/O controller) has modified the cache line. To do a write, a processor has to become the owner of the cache line. All other system devices invalidate any shared copies they have cached, and the current owner supplies the data. Henceforth, when other processors request to share an owned or modified copy of the data, the owning processor, not memory, will supply the data. The following sections describe the cache state and valid state transitions.

3.3.1. Cache States

The cache has five states:

- **cI**: Cache Invalid. Line is invalid in cache
- **cS**: Cache Shared. Line is valid, (potentially) shared, and clean in cache
- **cE**: Cache Exclusive. Line is valid, exclusive, and clean in cache.
- **cO**: Cache Owned. Line is valid, (potentially) dirty and (potentially) shared.
- **cM**: Cache Modified. Line is valid, exclusive and (potentially) dirty in cache

Valid means that the line contains useful data. *Exclusive* means that no other cache has a copy. *Dirty* means that the data has been modified.

3.3.2. Coherency State Transaction

The state transition diagram in Figure 3-1 shows the legal transitions for the MOESI cache state for a block. When the block is first read by a processor, if a valid copy exists in another cache, then it enters the processor's cache in the S state. If no other cache has the copy at the time, it enters the cache in the E state. When a block is in the E state and is written by the same processor, it can directly transition from the E to M state without generating another bus transaction, since no other cache has a copy. If another cache had obtained a copy in the meantime, the state of the block would have been demoted from E to S by the snooping protocol. A write to a block in any state will promote the block to M state. A read request will demote the block from M to S state and also cause the block to be written back to main memory. The owned state indicates that even though other shared copies of the block may exist, this cache (instead of main memory) is responsible for supplying the data when it observes a relevant bus transaction.

3.4. JBus Characteristics [7],[8],[9]

- **Simple SMP protocol implementation**

The protocol is SPARC®-V9 compliant. It provides high performance for 1-4 ways, and supports high bandwidth UPA64S / PCI interfaces[13,14].

- **150-200 MHz JBus operation**

Bus is designed for maximum clock frequency for up to 3 JBus loads. A load could be a CPU, a JIO, or a bridge to anything else.

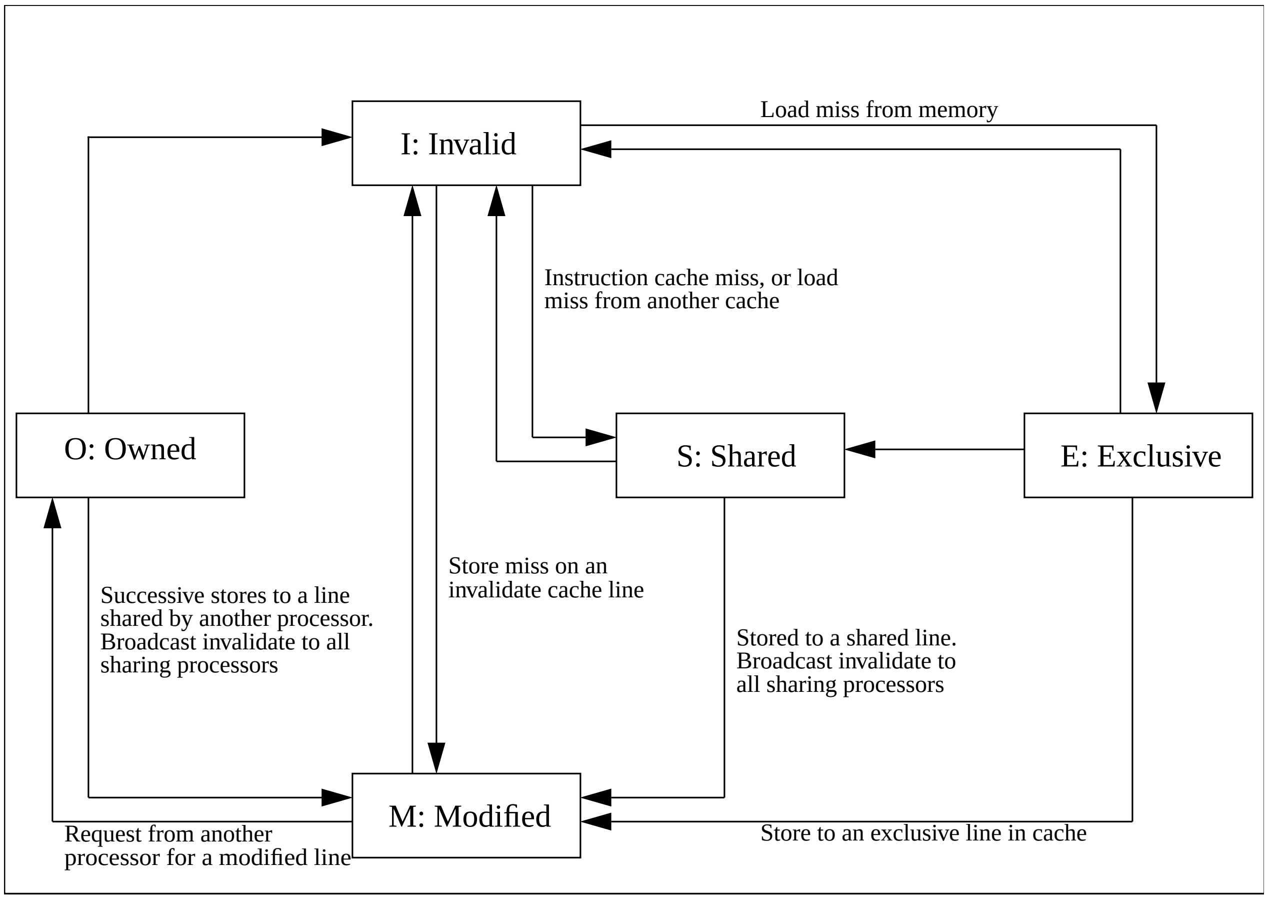
I: Invalid
Load miss from memory
Instruction cache miss, or load miss from another cache
O: Owned
S: Shared
E: Exclusive
Store miss on an invalidate cache line
Successive stores to a line shared by another processor. Broadcast invalidate to all sharing processors
Stored to a shared line. Broadcast invalidate to all sharing processors
M: Modified
Request from another processor for a modified line
Store to an exclusive line in cache

Figure 3-1 Legal state transitions for JBus MOESI protocol

- **Multiple JBus segment topologies**

To reach the 200 MHz clock rate, JBus may be divided electrically into multiple 3-load segments, with bus signals pipelined through a one clock delay bus repeater.

- **16-byte Shared address/data bus**

The 128 bit multiplexed address/data bus has a peak off-chip bandwidth of 3.2 GB/sec.

- **UPA64S & PCI I/O interfaces available**

JIO chip is a companion to JBus. It supports both Sun's high-performance graphics interface (UPA64S) and the PCI interface. The JIO was designed to minimize I/O traffic on JBus, thus providing most of the JBus bandwidth for the shared processors.

- **Bus arbitration latency minimized by “unfair” round robin arbitration**

A distributed arbitration protocol allows for the JBus to provide the lowest possible latency for bus ownership. The round robin protocol is unfair by design, favoring the last owner. The last owner can drive the bus without being dependent on possible simultaneous asserted requests, saving one cycle of arbitration latency.

- **Globally synchronous clocking**

The JBus is synchronous with a centrally distributed clock (system clock).

- **Low pin count**

The JBus pin count is 171 with 1.5V DTL (Dynamic Termination Logic).

- **Simple 360-pin edge connector for module**

Besides signal pins, there are power pins and ground pins. The total pin count is 360.

- **On-chip MOESI cache coherence protocol**

JBus supports a simple, low latency on-chip MOESI cache coherence protocol. The protocol provides high performance, high bandwidth for small (1-4 ways) multiprocessor systems.

- **All coherency is based on physical addresses (SPARC® V9 memory ordering model)**

JBus supports write-invalidate MOESI cache coherence protocol which meets the sun4u and SPARC V9 memory ordering model. For example, sun4u requires to report read data and time-out errors with the read transaction to the requesting JBus master port.

- **Strong memory ordering (Sequential consistency)**

The protocol supports a strong memory consistency model. It is the most straightforward model to maintain memory consistency. The simplest way to implement the strong order is to require a processor to delay the completion of any memory access until the invalidations caused by that access are completed.

- **Out-of-order data return for different cacheable addresses**

This will happen due to randomness in arbitrating for data returns on JBus, and different latencies in ports when seeing snoop results and sending read data back. To keep the design simple, data return from a single non-cacheable port is in order. Data return for the same cacheable address is in order. Order is determined by the address bus order.

- **All coherent events are queued when an address is issued on the bus**

Processors queue coherent transactions which need snooping that show up on the address bus, including their own. (Their own transactions do not require full snooping of the caches.)

- **All later coherency events (outstanding JBus reads) to the same address are blocked**

There are two ways to handle the address conflicts. One is to retry the request, the other is to block the request. Retry may cause a starvation problem if arbitration is not fair. Blocking does not have the starvation problem but needs extra memory space to keep track of requests.

- **Address blocking is distributed -- each port tracks its own reads**

The blocking is done in a distributed fashion, with each port tracking it's own read. The maximum number of outstanding reads is eight per port.

- **Cache-to-cache transfers when data is owned (modified) in a cache**

The protocol supports cache-to-cache transfer if the line is owned or modified in a cache. The caching master is responsible for returning dirty data from its cache, and can do this as soon as possible.

- **Snooping is fully decoupled from memory access**

This is to reduce the memory latency. Without this feature, the memory latency would increase, because all the dynamic delays would be serialized instead of in parallel.

- **Point-to-point snoop results and flow control**

JBus supports cache coherence by sending a snoop to all system devices. Each device examines (snoops) the state of the requested cache line in its local cache, and sends its acknowledgement back to the requester via a point-to-point network. The point-to-point network facilitates an extremely fast snoop response. The design also allows the latency to fluctuate, rather than requiring a fixed latency. This makes the implementation easier.

- **Variable snoop latencies (typically three cycles)**

Snoops are initiated at the processors and I/O cache when the request address is presented on the shared bus. The snoop latency needs to be less than the memory latency, since the data returned from memory will be cancelled if the snoop result indicates a dirty line exists in the cache.

- **Separate address and data flow control**

JBus maintains a simple Address / Data flow algorithm. The Address / Data state is maintained by each requestor, to decide if there is a room for address and write data at the targets of this transaction. The transaction is not driven on the bus if there is no room at the destination. The read plus writeback is an atomic transaction, so both the read and writeback request are held up by the initiator if there is not room available for the writeback data.

- **Error protection**

Errors on JBus are detected as a result of cacheable / non-cacheable read access to JBus, and cacheable / non-cacheable write access to JBus.   Errors are flagged by setting appropriate AFSR (Asynchronous Fault Status Register) and JBus parity bits. Interrupts may be generated to signal different types of errors.

CHAPTER 4

Summary

The JBus architecture is an ideal solution for today’s small-scale, high-density SMP. It delivers up to 3.2 GB/s. bandwidth for high performance applications in many areas, this bandwidth is exactly what the next generation of MP systems will require. It also provides a scalable solution so that precise cost and space specifications can be matched to application requirements. It can be used for a wide range of applications from a desktop system to small-scale (4-way) multiprocessor systems.

The combination of performance, flexibility, and low cost implementation makes the JBus technology and its system an attractive real-world option for the developer of high performance embedded systems, including networking, communications, packet processing, and information appliances.

CHAPTER 5

References

A. Background

1. D. E. Culler, J.P. Singh and A. Gupta, Parallel Computer Architecture, Morgan Kaumann Publisher, Inc., San Franciso, California, 1999.

2. J. Duato, S. Yalmanchili and L. Ni, Interconnection Networks: An Engineering Approach, IEEE Computer Society Press, 1997.

3. K. Hwang and Z. Xu, Scalable Parallel Computing, McGraw-Hill, 1998.

4. K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, programmability, McGraw-Hill, 1993.

5. D. E. Lenoski and W.D. Weber, Scalable Shared-Memory Multiprocessing, Morgan Kaumann Publisher, Inc., San Franciso, California, 1995.

6. D. Patterson and J. L. Hennessy, Computer Organization and Design: The Hardware / Software Interface, 2nd Edition, Morgan Kaumann Publisher, Inc., San Franciso, California, 1997.

B. Architecture of JBus

7. R. N. Raman, “JBus: A Bus Architecture for Embedded On-chip Multiprocessing”, Embedded Microprocessor Forum, 2001.

C. UltraSPARC IIIi

8. K. Normoyle, “A Dash of Jalapeno: Introducing the UltraSPARC IIIi Microprocessor”, Microprocessor Forum, 2001.

9. G. Konstadinids et al., “Implementation of a Third-Generation 1.1-GHz 64-bit Microprocessor,” IEEE Journal of Solid-State Circuits, Nov. 2002, pp. 1461-1469.