'smart' DMA devices that share main memory control and status blocks with the driver. DMA accesses which use streaming buffers have no guaranteed write ordering. Software must explicitly drain the read or write buffers at the start or end, respectively, of a DMA transfer. This operation is provided as part of the interface between the operating system and device drivers in a way that is not machine specific. Other architectures require a similar resynchronization step at the boundary of DMA transfers, for example to flush a processor's virtual cache. It is important to note that this difference in ordering models does not affect cache coherency, but only the exact order in which writes are perceived. As such, there is never any need to flush any cache; in fact, there is no hardware support for cache flushing.

4.3: TLB coherency

The processor unit provides hardware support for multiprocessor TLB coherency. This is an extension to the SPARC Reference MMU (SRMMU) described in [7].

When a processor requires a TLB flush for a range of entries, the flush command is broadcast to all other processors and the issuing processor is stalled until the other processors have completed the operation.

This mechanism is much more efficient than software schemes that use inter-CPU interrupts to implement TLB coherency.

4.4: Interrupt management

XDBus provides a generic interrupt transport mechanism which indicates a target unit for the interrupt (possibly broadcast), an interrupt level and an interrupt source identification.

When a processor unit receives an interrupt packet, it sets the corresponding interrupt source bit and interrupt level in internal registers. This source identification allows the amount of interrupt polling performed by the processor to be reduced.

Individual processors can issue arbitrary interrupt packets, providing a general mechanism for interprocessor interrupts.

The I/O unit transforms the level-sensitive interrupt scheme of SBus to the packet-oriented transport provided by XDBus. It also provides a mutual exclusion mechanism which prevent interrupt service race conditions by multiple processors and identify exactly which SBus device asserted a given interrupt level. This mechanism further reduces the amount of SBus device polling.

I/O units can be individually programmed at any time to direct interrupts to a specific processor. This allows static or dynamic interrupt load balancing by the kernel.

4.5: Performance measurement tools

Most of the system components provide event counters to measure various aspects of system activity. They include count of bus transactions, by type of transaction, global or from/to a specific functional unit, second-level cache miss rate and miss latency, instruction counters.

In addition to the counter/timer used for normal kernel activities ('tick timer'), each processor has a high resolution (1 μs) timer which can be used for kernel profiling.

5: Conclusion

The modularity of this architecture has been important in the implementation of the SPARCcenter 2000. It has allowed us to make modifications to the system-level design late in the project, and has served as the basis for multiple system designs.

Another interesting feature in that the architecture lends itself well to the definition of new functional units for more specialized purposes, such as high-speed network interfaces and graphics operators.

6: Acknowledgments

This project was the result of a joint development between Sun Microsystems, Inc. and Xerox Corp. It is the outcome of the Dragon research project conducted at the Xerox Palo Alto Research Center (PARC). The authors want to thank both companies for their support during the research and the development phases of this project.

7: References

[1] P. Sindhu, et al.,"The XDBus: A High Performance, Consistent, Packet Switched Bus," IEEE COMPCON Spring '93, San Francisco, Feb. 1993.

[2] B. Gunning, et al., "A CMOS Low-Voltage-Swing Transmission-Line Transceiver," ISSCC DIGEST OF TECHNICAL PAPERS, pp. 58-9, Feb. 1992.

[3] "The XBus Specification," Texas Instruments, 1992.

[4] F. Abu-Nofal, et al., "A Three-Million-Transistor Microprocessor," ISSCC DIGEST OF TECHNICAL PAPERS, pp. 108-9, Feb. 1992.

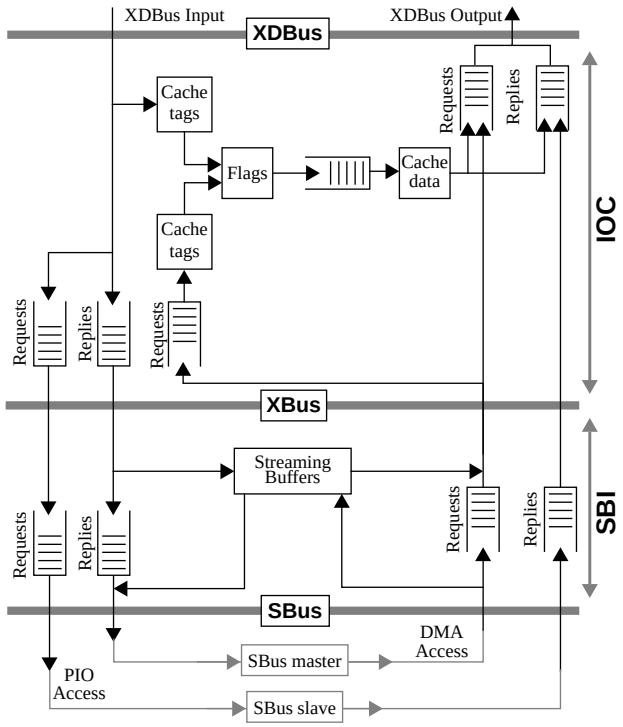
[5] B. Joshi, et al., "A BiCMOS 50MHz Cache Controller for a Superscalar Microprocessor," ISSCC DIGEST OF TECHNICAL PAPERS, pp. 110-111, Feb. 1992.

[6] "SBus Specification B.0", Sun Microsystems #800-5922-10, December 1990

[7] "The SPARC Architecture Manual (version 8)", Prentice Hall,1992.

[8] M. Cekleov, et al., "SPARCcenter 2000: Multiprocessing for the 90's," IEEE COMPCON Spring '93, San Francisco, Feb. 1993.

[9] "IEEE Standard Test Access Port and Boundary-Scan Architecture, P1149.1," IEEE Computer Society Test Technology Technical Committee, New York, Jan. 1989.



I/O Unit block diagram

available. As soon as the data is available, the SBus device gets the highest priority for SBus arbitration, in order to compensate for the additional latency it incurred.

The I/O unit allows up to three outstanding operations on XDBus for each SBus master: a stream buffer fetch, a stream buffer write, and a non-stream read or write operation. This high degree of concurrency provides good SBus DMA performance.

3.4: Bus interface unit and arbitration

The bus interface unit is not visible to the programmer. It is used to segment the XDBus across packaging levels, for example at the boundary between a board and the backplane. It contains two major components, a bus driver/receiver (BIC) and a local arbiter (BARB).

BIC provides a bidirectional XDBus on the backplane side and a unidirectional XDBus on the on-board side, with a pipeline stage in between. This arrangement allows the bandwidth of a segmented XDBus to be utilized fully. If both sides were bidirectional, each packet would require two clock cycles of turnaround time, resulting in a bandwidth loss of over 30%.

XDBus arbitration is performed in two tiers to provide packaging flexibility and good electrical behavior. The bottom tier is composed of BARB, which arbitrates between functional units on a board. The top tier is a central arbiter (CARB) which arbitrates between BARBs. BARB uses round-robin arbitration. CARB uses a modified round-robin scheme to provide fairness between functional units instead of between BARBs.

The arbitration logic is responsible for enforcing flow control for packets on the XDBus. When a functional unit detects that one of its incoming XDBus queues (or a queue indirectly filled by XDBus inputs) has reached a high water-mark, it notifies the arbitration system to deny grants at priority levels which could be used to send data to this queue. Once the queue has reached low water-mark again, the unit allows the arbiter to grant devices at a lower priority. To help the flow control mechanism and provide better latency, the arbitration system uses 4 priority levels, 2 for requests and 2 for replies. All replies are at a higher priority than requests.

The arbitration system is also responsible for merging consistency information (shared/owner) from the caches in the processor and I/O units without wire-oring.

4: Software and performance tuning features

4.1: Cache behavior tuning

The processor's second-level cache supports update as well as invalidate mode for writes to shared data, including atomic operations. The choice is under software control and is made by the processor unit which issues the shared write. In addition, the processor unit provides a statistical invalid/update feature, which transforms write-updates into write-invalidates with a programmable probability. Given the large cache sizes, there may be a large proportion of data which is artificially shared (i.e. the data resides in multiple caches, but is in the active set of only one processor). The randomized invalidate/update allows in effect a user-tunable 'cache laundering' mechanism.

4.2: Processor write ordering

In the presence of store buffers, multiple paths to target devices (multiple XDBuses) and multiple memory controllers, strong ordering cannot be implemented efficiently.

For processor accesses, the architecture supports the TSO/PSO memory models described in the SPARC V8 architecture [7]. Software can switch dynamically, for example on a per-process basis, between TSO and PSO mode for each processor. Note that a single-thread process cannot perceive the difference between TSO and PSO modes and can thus always run in the weaker model for higher performance. In addition, accesses from processors to I/O devices (programmed I/O) always follow the TSO model to ease the task of writing device drivers.

DMA accesses which do not use streaming buffers follow the TSO model, which allows easy implementation of

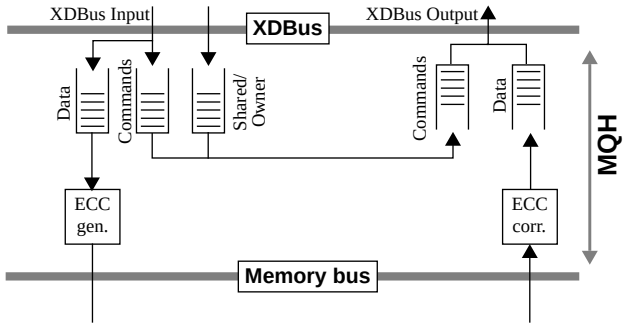
BW while a cache miss is outstanding, providing a faster return path for the reply.

The processor unit also provides a special path, the BootBus, for initialization. When the processor is reset, it starts fetching code from the BootBus, which contains firmware for power-on self-test. This allows system testing to be initiated with a minimum of assumptions on which parts are functional, providing good fault isolation at system level.

3.2: Memory unit

A memory unit consists of a memory controller per XDBus, with one to four memory banks per memory controller. Memory banks are implemented with custom SIMMs.

The memory controller (MQH) includes refresh logic for the memory banks, error correction (SECDED over 64 bits), and provides programmable timing for the DRAMs to customize its behavior for each type and speed grade of memory, including static memory. The ECC code is arranged in such a way that a single DRAM contributes only 1 bit per ECC-corrected word, which protects against the failure of an entire DRAM. In addition to its role as a memory controller, MQH also acts as the reflection point for shared writes on XDBus.



Memory Unit block diagram

The range of memory addresses assigned to each bank is programmable and permits up to four-way memory interleaving per XDBus. Memory interleaving is on a 64 byte basis. This allows a single cache line to be interleaved among multiple MQHs, which is especially important when a cache needs to write back an entire 256 byte line (4 64 byte blocks).

The memory unit performs memory operations sequentially (i.e.memory banks are not interleaved within a memory unit), but has a high degree on internal concurrency. As a result, a single memory unit comes close to being able to fill the whole bus bandwidth. In addition, the latency is decreased by fully overlapping arbitration with memory access time.

Each memory bank is implemented with four custom SIMMs. Each memory SIMM provides 18 bits of data per clock period. It implements a 72-bit data path internally, using 18 x4 DRAMs, with one word every 4 clocks. The 72 bits are multiplexed on the SIMM using a small custom multiplexor/demultiplexor chip (CBS) over the 18 bit interface. This arrangement allows a wide (288 bits) memory access path while maintaining a narrow physical interface and minimizing on-board logic.

3.3: I/O unit

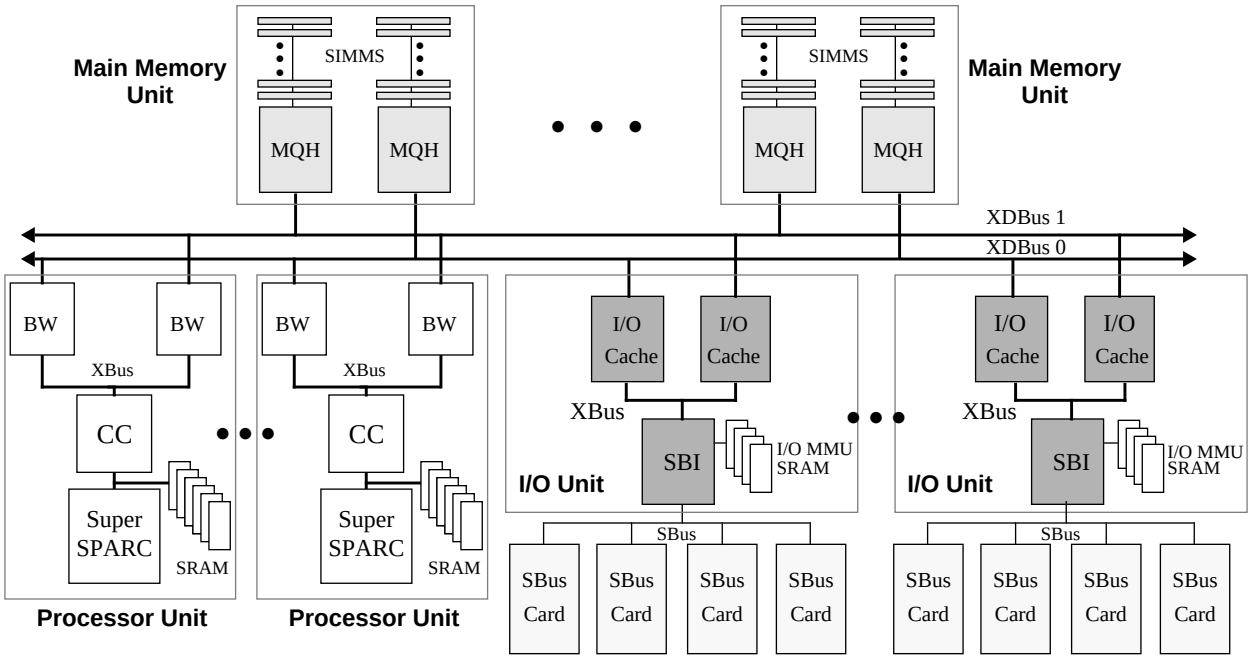
The I/O unit provides an SBus [6] with up to four masters. The XDBus interface consists of an I/O Cache (IOC), while the core functional unit consists of an SBus interface controller (SBI) and an external SRAM which provides translation tables for DMA accesses.

In addition to the normal XDBus interface functions, IOC includes a small cache for DMA operations. The IOC cache replacement algorithm may be LRU or based on the SBus slot which requires the operation. The cache uses 64 byte lines without subblocking, and has a dual directory structure to provide snooping without interfering with SBus accesses. There is a single copy of the shared and owner flags.

SBI acts as the SBus controller (including translation of virtual DMA addresses), as an SBus master for programmed I/O operations and as an SBus slave for DMA operations to and from memory. Address translation is performed using an external page table (SRAM) which allows up to 64 MB of mapped virtual DMA space. SBI also contains the asynchronous boundary between the system clock domain and the SBus clock domain. SBI provides two major features to improve SBus performance: streaming buffers and rerun management.

The streaming buffers provide read-ahead and write-behind buffering for DMA operations. Each SBus slot has its own set of buffers to avoid interference between slots. The streaming buffers assume a sequential DMA access pattern. The device driver is responsible for the decision to use or not to use the streaming buffers for each DMA transfer. The decision is encoded as a flag bit in the DMA virtual address translation tables. When a DMA transfer does not use the streaming buffers, the operation is passed to the IOC, which either uses its internal cache to satisfy it, or bypasses the cache altogether for block-sized (64 byte) transfers.

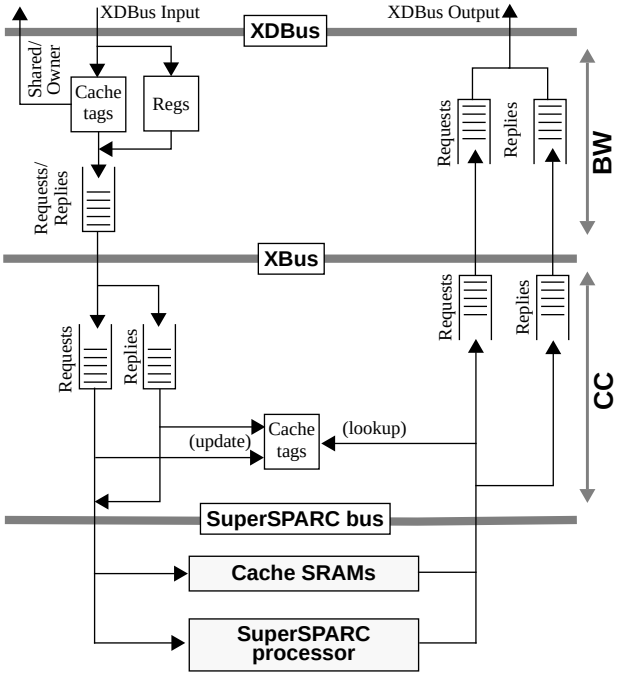
SBI uses SBus reruns to force an SBus master off the SBus whenever the expected latency is large, for example on a streaming buffer miss. When an SBus device is rerun, its arbitration is blocked until the data it required becomes



SPARCcenter2000 logical system architecture

page size (4K byte pages). This allows the cache access to be done concurrently to the TLB lookup, while removing the aliasing problems which occur with virtual caches.

The second level cache has a direct-map organization and is physically indexed. It uses an external SRAM array for data storage, while the tags are stored internally. The cache directory is replicated between CC and BW. This dual directory architecture allows BW to snoop the XDBus traffic and provide shared/owner information to other caches without disturbing the processor's access to its second-level cache. It also allows BW to 'filter out' most XDBus traffic and keep the utilization (and thus the latency) of the XBus low.



Processor Unit block diagram

To keep the dual directories synchronized, all modifications to the cache state occur logically from the XDBus side, and are then propagated to the cache controller. For example, on a cache read miss, the new line tag is entered in the BW directory when it detects its own ReadRequest packet on the XDBus, while it is entered in the CC directory only once the ReadReply packet is forwarded to the CC via the XBus. This permits the write-back of up to four blocks on the old line to be performed while the read of the new block is being done. The write-back is actually accomplished in two stages, first from cache array to BW, and then from BW to memory. This allows the read reply to be forwarded to the processor as early as possible.

In addition to normal processor and cache support features, the processor unit also provides special hardware for fast memory to memory copy (block copy), interrupt management and a set of counters/timers.

Up to three read operations may be simultaneously outstanding from the processor side: a cache miss, a cache prefetch operation and a block copy operation. In addition, there can be as many as 16 outstanding writes to shared data. To improve cache miss latency, the XBus arbitration mechanism provides bus parking for the corresponding

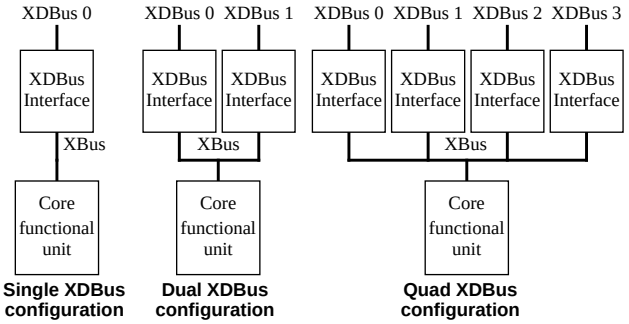
physical address (interrupts and TLB maintenance) always use the first interleave (interleave 0).

The interleaving size is chosen to allow a cache line size of up to 256 bytes (see section 3.1).

2.3: Interfacing to multiple XDBus

Supporting multiple interleaved XDBuses requires most functional units to provide a 'convergence bus' to connect the unit to the XDBuses.

Functional units which require convergence are split in two parts, the XDBus interface and the core functional unit. The XDBus interface contains the logic which needs to be replicated when multiple XDBus are used, as well as logic to pass onto the core functional unit only those XDBus packets that are 'relevant'. The core functional unit itself does not depend on the exact number of XDBuses. The XDBus interface logic and the core functional units are connected together using a different bus, the XBus [3], as outlined in the following figure:



XBus connection

The XBus protocol is extremely similar to the XDBus protocol, but has additional control information which permits the exchange of private data between the core functional unit and the XDBus interface. For example tag manipulation information in the CPU unit is passed on the XBus. The XBus also uses an arbitration mechanism which respects the order of incoming packets from the various XDBus in order to guarantee proper serialization of events, especially for broadcast operations such as shared write updates. This is achieved by conceptually timestamping XBus arbitration requests originating from the XDBus interface.

It is important to note that the XBus implementation details are slightly different for each functional unit type, based on private communication requirements between the core functional unit and the XDBus interface.

2.4: Internal structure of functional units

The packet-switched protocol used by XDBus implies a strong reliance on queuing mechanisms. A typical XDBus interface has at least four logical queues, corresponding to request and reply packets, incoming (XDBus → XBus) and outgoing (XBus → XDBus). Additional queues may be needed for operations which use only one of the buses. This queue structure may also extend into the core functional unit.

Each functional unit has a set of loadable parameters that allows it to be configured for a particular system setup. These parameters include for example the address ranges to which the unit responds, the minimal arbitration latency, and the latency for shared and owner information. Parameters are normally setup by power-on firmware. This allows a large amount of flexibility in system configuration, while providing a uniform view of the hardware architecture to the operating system. Each of the functional units can also be 'frozen', i.e. forced into a reset state under software control. This allows a defective unit to be configured out of the system.

Functional units perform continuous checking for unrecoverable error conditions, such as a cache coherency failure. When an error is detected, the functional unit issues an error signal and logs the error in registers which are accesible via JTAG [9].

3: Functional units

The figure at the top of the next page provides a general block diagram encompassing the three main types of functional units: processor, I/O and memory.

3.1: Processor unit

The processor unit is based on the TI SuperSPARC processor [4] (TMS390Z50) with a 1 MB parity-protected second level cache controlled by a TI SuperSPARC MultiCache Controller [5] (CC, TMS390Z55). The XDBus interface function is performed by the Bus Watcher (BW).

The following table gives the cache configuration.

Cache Type	Size/ Org.	Protocol	Block size/ Line size
Instruction, 1st level	20 KB 5-way	invalidate, invalidate	32B/64B
Data, 1st level	16 KB 4-way	write-through, invalidate	32B/32B
Combined, 2nd level	1 MB direct-map	write-back, update/invalidate	64B/256B

The first line in the protocol column refers to the effect of writes from the processor, while the second refers to the effect of writes from the XDBus side.

The two first-level caches are entirely internal to the SuperSPARC processor, which supports external invalidation based on physical addressing. Both are interleaved in such a way that they are indexed by address bits below the

TO APPEAR IN COMPCON SPRING '93, San Francisco, Feb. 1993

The Next-Generation SPARC Multiprocessing System Architecture

Jean-Marc Frailong*, Michel Cekleov, Pradeep Sindhu*, Jean Gastinel*,
Mike Splain, Jeff Price, Ashok Singhal

Sun Microsystems Computer Corporation
2550 Garcia Avenue
Mountain View, CA 94043-1100

* Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Abstract

The multiprocessor architecture described in this paper defines a set of functional building blocks that share a common hardware interface, the XDBus. This modular approach permits the implementation of multiprocessors covering a wide range in performance and cost. It allows the ratio of processing power, memory capacity, and I/O bandwidth to be varied within a given machine while permitting system designers to address different points on the overall performance spectrum. Each functional block (processor, memory, I/O) consists of a small number of highly integrated chips.

The architecture provides a number of features to support high performance symmetric multiprocessor software. These include hardware caches, I/O, and TLB coherency, dynamic interrupt dispatching with source identification, weak write ordering, block copy hardware, and hardware performance monitoring

1: Introduction

The SPARCcenter 2000 system is an implementation of the next generation multiprocessor system architecture based on the SPARC processor architecture. This architecture provides a shared memory model with full hardware support for cache coherency.

One of the major goals for the architecture was to provide cost-effective scalability of a single implementation over the largest possible range of systems and configurations. To achieve this goal, the architecture is designed as a set of independent functional units which communicate over a common hardware interface, XDBus [1]. XDBus is both a chip-level bus and a backplane bus. The scalability is achieved in two ways.

First, a system implementation may use one, two or four XDBuses in parallel to customize the system bandwidth for its maximum configuration.

Second, since the functional units are completely independent, a system implementation may choose how to package functional units into boards based on entry system configurations and expansion requirements. The functional units currently implemented are the processor unit, the I/O unit and the memory unit, as well as bus interface logic and arbitration.

As one example, Sun Microsystem's SPARCcenter 2000 uses 2 XDBus in a backplane configuration and provides two processor units, one memory unit, and one I/O unit on a single board.

2: Overview

2.1: XDBus main features

XDBus is a 64-bit wide high-performance packet switched bus which supports memory coherency using a generalized write-broadcast protocol. It also provides transactions for non-memory references (programmed I/O), interrupt handling and TLB consistency. The basic transfer unit sizes are 64 bytes (a block) and 1, 2, 4 or 8 bytes. The arbitration interface is fully pipelined and allows a single requestor to have multiple arbitration requests pending.

The physical interface uses low-swing GTL drivers [2] and can be either bidirectional or unidirectional. In the unidirectional mode, the bus can be segmented and pipelined across multiple packaging levels, which allows a single logical bus to be used both as a backplane bus and as an on-board bus connecting multiple devices on a single board.

2.2: XDBus interleaving

When multiple XDBuses are used in a system, they are interleaved on 256-byte boundaries, based on the physical address being referenced. Packets which do not carry a