protocol that provides 75% of the raw bandwidth for data transfer; efficient support for cache coherency and several other key mechanisms useful in multiprocessor designs; up to 80 MHz operation with existing technology; 64 byte block transfers for memory and IO; support for single word reads and writes with byte enable; data bandwidths from a few hundred MBytes/sec to 2.5 GBytes/sec; a synchronous, pipelined arbitration system; and a transceiver technology that consumes very little power.

There have been three separate implementations of the technology to date, and designs have been proven up to 80 MHz operation for compact systems. A standard chipset that allows a wide variety of systems to be built using XDBus is also available. It is the intent of Xerox Corporation to license XDBus for widespread use as an open industry standard. Interested parties should contact the authors.

Bibliography

[1] Gunning, B., Yuan, L., Nguyen, T., Wong, T., "A CMOS Low-Voltage-Swing Transmission-Line Transceiver" in *Proceedings of the 1992 IEEE International Solid State Circuits Conference.*

[2] McCreight, E.M., "The Dragon Computer System" in *Proceedings of the NATO Advanced Study Institute on Microarchitecture of VLSI Computers,* Urbino, July 1984.

[3] Goodman, J. R., "Using Cache Memory to Reduce Processor-Memory Traffic" *IEEE Transactions on Computers C-27,* No 12, December 1978, pp 1112-1118.

[4] National Semiconductor DS3890 octal trapezoidal driver.

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

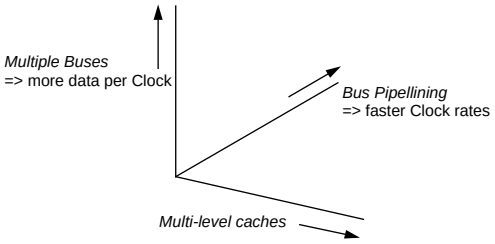
[6] Frailong, J-M., et. al., "The Next-Generation SPARC Multiprocessing System Architecture," *IEEE COMPCON Spring '93,* San Francisco, Feb. 1993.

The synchronization primitive supported directly by XDBus is Swap. The implementation of Swap is efficient in that non-local operations are done only in case the target of the Swap is marked shared. For non-shared locations Swap is performed local to the cache, and is therefore much faster. Although only Swap is supported directly, any *FetchAndOp* type of primitive could be implemented with equal efficiency.

There is also a set of two transactions Lock and Unlock that allow any sequence of bus transactions to be made atomic. Atomicity can be provided with respect to a single location or any contiguous, self-aligned region in memory that is a power of two in size.

7: Scalability

The XDBus provides scalability of performance along three orthogonal dimensions: First, the bus cycle time may be decreased through the use of pipelining, resulting in a proportional increase in performance. Second, two or four buses can be used in parallel to increase the available bandwidth by the same factor. Third, multi-level caches may be used to localize traffic, relieving bottlenecks that may have occurred otherwise, and permitting higher levels of performance.



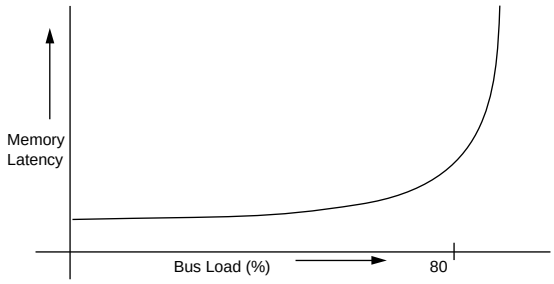
XDBus provides scalability along three orthogonal dimensions

A given system can use one or more of these techniques independently, or in combination, to tailor bus performance to application needs. In contrast, traditional buses confine a system designer to a narrow performance range, forcing a painful choice between low performance and migration to a different bus technology.

8: System-Level Performance

Measured performance of the XDBus on a SPARC Center 2000 system confirms that the bus can be run very close to 100% utilization, and at this utilization nearly 75% of the raw bus bandwidth is provided to the application that is running.

Measurement also showed the phenomenon of packet convoying, in which packets are bunched together in time rather than being spread out evenly. Convoying occurs because of the particular flow control method used: reply packets are systematically given priority over request packets. A more precise flow control mechanism, where only request packets directed to the particular queue about to overflow are stopped, would eliminate convoying.



Dependence of Memory Latency on Bus Load

XDBus exhibits stable latency behavior with increasing bus load. As shown in the figure below, memory latency degrades slowly with increasing bus load up to around 80% bus utilization. At this point latency begins to increase rapidly until the bus reaches saturation. This slow dependence of latency on load means that the bus can be run at heavy loads without individual processors seeing a noticeable performance degradation.

9: Application Areas

Although XDBus was designed specifically for cost effective general purpose multiprocessors, it can be used to advantage in a number of other applications that need high bandwidth. Key amongst these are multimedia applications where video and voice processing is required; routers and switches for Gigabit networks; and medium to high end document processing systems.

High end PC's and low end workstations are currently faced with the dilemma of how to provide the high bandwidths needed for multimedia applications. Several proposals have suggested separate video buses to solve the bandwidth problem. This adds significant cost, however, since the video bus is present in addition to the system bus connecting processor and memory. XDBus resolves this by providing a single interconnect that is suitable for both high bandwidth applications as well as for connecting processors and IO to memory.

10: Conclusions

This paper has presented a low cost, high performance, packet switched bus that is designed for high integration and wide applicability. The main features of the bus are fully synchronous operation; a 64 bit, parity protected data path expandable to 128 and 256 bits; a packet switched

impedance of the trace being driven. GTL also provides special drivers with pullup capability for situations where pullup resistors are impractical, for example within a chip.

XDBus also allows a standard hardware interface to be created, much like a VLSI macro. The design of this macro can then be leveraged across multiple chip implementations, saving design time and avoiding costly errors.

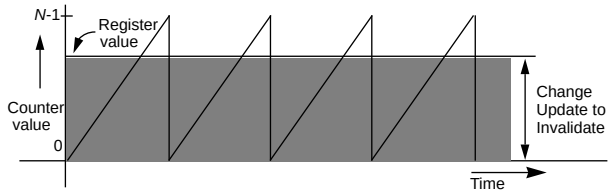
A final advantage of XDBus as a VLSI interconnect is that its signalling scheme is compatible with the coming generation of 3.3V, and even lower voltage, technologies. What makes this possible is that the GTL's signal swing is fixed by the external pull up voltage and its threshold is set by a reference voltage. Both of these voltages are independent of the supply used to power on chip logic.

6: Multiprocessing

The XDBus provides a number of carefully selected features for multiprocessor support. There is a simple but efficient hardware coherency protocol to keep cached data consistent. Support is also provided for maintaining TLB (Translation Lookaside Buffer) consistency. A dedicated interrupt transaction removes the need for the usual jumble of wires to communicate interrupts from devices to processors. Two schemes are provided for multiprocessor synchronization: a simple efficient Swap primitive, and a more general locking mechanism.

The cache coherency protocol is a generalization of the well known multi-copy write broadcast protocol [2]. The first generalization is to adapt the algorithm to a packet switched bus. The main difficulty here is that bus transactions are no longer atomic since they are broken up into request and reply packets. XDBus's scheme resolves this difficulty by conceptually treating a read as if all the work was performed on the request packet, and a write as if all the work was performed on the reply packet. Snooping information that tells whether an address that appears on the bus is present in one or more caches is collected by the arbiter and logically OR'd to give a single result. This result is then returned in a reply packet to the device that sent the corresponding request.

The second generalization enables the hardware to effectively emulate any coherency scheme between pure write update and write invalidate. The basic idea is to remove a cached copy probabilistically when a foreign write is done to it. Setting the probability close to 1 yields a write invalidate scheme, while setting it close to 0 yields write update. This scheme can be easily implemented using a free running modulo *N* counter, a register that contains a value between 0 and *N*-1, and a comparator. The counter is updated on each clock, so its value is essentially uncorrelated with the arrival of packets. When a foreign write arrives, the value of the counter is compared with that of the register and the update is turned into an invalidate if the counter's value is less than the register's. As shown in the figure below, setting the register to *N*-1 gives write invalidate; setting it to 0 gives write update; and setting it to intermediate values gives intermediate schemes. This idea can be implemented cheaply, but has the potential for significantly improving performance when an application's sharing patterns are known.



Probabilistic Conversion of WriteUpdates to WriteInvalidates

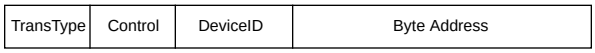
The third generalization is the support of cache coherency in a multi-level hierarchy of caches. Surprisingly, this adds little complexity to the basic single level algorithm: just one additional transaction called *KillBlock* is needed. Multi-level caches provide localization of data traffic, and have the potential for supporting hundreds of processors in a single system.

Most multiprocessor systems provide no hardware support for consistency of address mapping information. This information is typically kept in main memory tables and copies are kept in TLB's inside each processor. The copies must, however, be kept consistent and this is usually done in software at a substantial cost in system performance. XDBus supports a single operation called DeMap, which forces a given address translation to be flushed synchronously from all TLB's. Since all new translations must use main memory tables, the software can safely change an entry by first locking it and then using DeMap to flush the TLB entries. This provides a simple and efficient way to solve the TLB consistency problem.

In traditional bus designs, interrupts are communicated from IO devices to processors via dedicated wires. This scheme has the obvious problem of connectivity when multiple IO devices must communicate with multiple processors. It also has the drawback that the communication paths are fixed and interprocessor interrupts are not handled the same way as IO interrupts. XDBus provides a single transaction to transport an interrupt from an IO device or a processor to one or more processors. The transaction either specifies that a particular processor is to be interrupted or all processors are to be interrupted. Besides providing a single mechanism for transporting all interrupts, this transaction facilitates dynamic interrupt targetting: since the target processor is determined dynamically, an interrupt can be dispatched to the least loaded processor in the system.

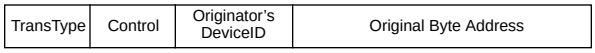
between successive packets. The arbiter also performs flow control by giving a higher priority to reply packets.

A request packet's header contains the transaction type, a small number of control bits, the requestor's DeviceID, and a physical byte address; it may also contain additional transaction dependent information.



Format of a Request Packet

The reply packet's header contains the same transaction type, the original requestor's DeviceID, the original address, some control bits, and transaction dependent data.



Format of a Reply Packet

This replication of type, DeviceID, and address allows request and reply packets to be paired unambiguously. Normally, the protocol ensures a one-to-one correspondence between request and reply packets. However, because of errors, some request packets may not get a reply. Thus, devices cannot depend on the number of request and reply packets being equal because this invariant will not be maintained in general. The protocol requires devices to provide a simple, but crucial guarantee: they must service request packets in arrival order, independent of packet priority. This guarantee forms the basis for XDBus's data consistency protocol.

The XDBus defines a complete set of transactions for data transfer between caches and memory, IO, interrupts, cache consistency, synchronization, and address mapping: The *ReadBlock* transaction reads a block of data from memory or a cache. *WriteBlock* writes new data into the memory system. *FlushBlock* allows caches to write dirty data back to memory. *NonCacheableReadBlock* allows data to be removed from the memory system. *KillBlock* deletes a block of data from all caches. *WriteSingleUpdate* and *SwapSingleUpdate* are short transactions used by caches to update multiple copies of shared data. *WriteSingleInvalidate* and *SwapSingleInvalidate* are short transactions used by a cache to update its copy but invalidate others. *IOReadSingle*, *IOWriteSingle*, and *IOSwapSingle* initiate and check IO operations, while *IOReadBlock* and *IOWriteBlock* allow block transfer of data between IO devices. The *Interrupt* transaction provides the mechanism for transporting interrupts to processors. The *Lock* and *Unlock* transactions allow arbitrary sequences of atomic operations to be implemented. Finally, the *DemapInitiate* and *DemapTerminate* transactions provide a way to remove virtual to physical address translations. There is room for twelve additional transactions in the encoding space (one of the transactions codes is used to indicate an idle bus and signal errors). With the exception of the Swap transaction, this set is processor independent.

The XDBus has a data transport efficiency of 73% when reading blocks of data and 88% when writing blocks, the remainder being consumed by protocol overhead such as DeviceID, address, and transaction type. These numbers derive from the fact that 8 out of 11 cycles in block read type transactions and 8 out of 9 cycles in block write type transactions carry data. The efficiency for short transactions is considerably lower. In practice the overall bus efficiency is close to 75% because most transactions on the bus are used to carry blocks—short transactions simply do not occur often when running realistic applications on a multiprocessor.

5: VLSI Interconnect

The XDBus's signalling scheme as well as its logical protocol are designed to promote a high level of system integration. Entire subsystems such as memory controllers, cache controllers, graphics controllers, interfaces to standard IO buses, and high speed network controllers can be implemented on a single chip that is connected directly to the XDBus. In traditional buses, this level of integration is simply not feasible, and so much more board real estate is needed for separate bus transceiver chips and glue logic.

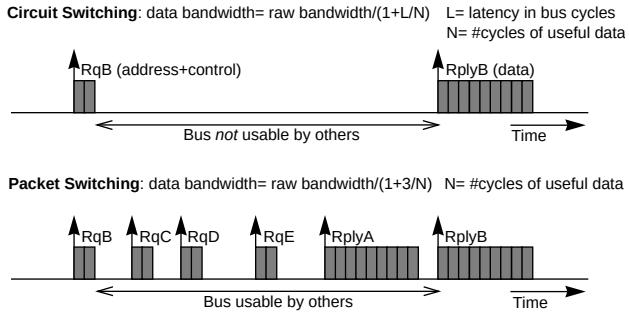
A key factor in achieving high integration is that on-chip power consumption by bus transceivers is extremely low. As the table below shows, a single transceiver consumes only 9 mW, compared with 71 mW for BTL [4] and 125 mW for ECL (BTL and ECL are alternative transceiver technologies). This low power consumption allows several hundred transceivers to be integrated onto a single chip with a power budget of less than two watts. ECL and BTL require almost 10 times as much power making it infeasible to reach this level of integration.

Device Type	Power Per Transceiver	Power for 160 Transceivers
GTL	9 mW	1.5 W
BTL	71 mW	11 W
ECL (10H123)	125 mW	20 W

The protocol also contributes to high integration because it makes efficient use of wires. Close to 75% of the raw bandwidth of the bus is available for useful data transfer once all the "overheads" such as address and control have been accounted for. This means that just 64 data wires are needed for reaching sustained data bandwidths of over 250MBytes/sec at 40 MHz. With a less efficient protocol, many more wires would be needed, and it would be difficult to integrate a complete bus interface on a single chip even with a low power technology such as GTL.

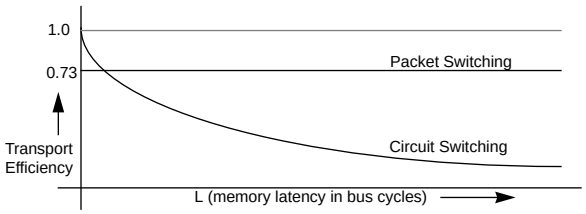
These advantages make XDBus an ideal VLSI interconnect. The bus can be used for communication within a chip, between chips on a board, as well as over a backplane by simply using drivers sized appropriately for

The figure below explains why this is the case. A circuit switched bus is not available for use in the interval between a request to memory (RqB) and the reply (RplyB) from memory. Thus the bus remains idle in this interval.



In a packet switched bus, the request and reply phases of an operation are broken up into independent request and reply packets which arbitrate separately for the bus. Other requesters may use the bus during the request-to-reply interval to send new requests (RqC, RqD, RqE) or to reply to earlier requests (RplyA). Thus a higher fraction of raw bus bandwidth is used to transfer useful data, which translates to higher transport efficiency.

The way in which transport efficiency depends on memory latency for the two schemes is interesting. As shown in the figure below, the relative advantage of a packet switched bus over a circuit switched bus increases with memory latency. The trend in computer systems in the last five to ten years has been that system level memory latencies have increased steadily when measured in number of system clocks. This trend is expected to continue with the advance of technology, which means that packet switching becomes more and more advantageous with time.



Transport Efficiency for Packet Switching and Circuit Switching

A related advantage of packet switching is that it works well with bus pipelining. The extra pipeline stages add to latency, but do not affect the useful bus bandwidth. In contrast, if pipelining is used in a circuit switched bus, the extra pipeline stages not only add to memory latency, but also subtract from useful bus bandwidth.

Packet switching also has several other desirable attributes. Since request and reply are completely dissociated, this provides a natural way to support slow devices. Also, circuit switched buses are prone to deadlock when configured in a hierarchy; there is no such problem with packet switched buses.

Packet switching does impose a penalty, however. An implementation is more costly in chip real estate, mostly due to the requirement of buffering packets. Also, a design is somewhat more complex and must face new problems such as flow control, which are absent in a circuit switched implementation. Here again, the technology trend is in favor of packet switching. The added gate count due to a packet switched implementation is already a fraction of the total number of gates on a state-of-the-art chip, and this fraction will only decrease as overall gate counts increase.

4: Bus Protocol

The XDBus's operation can be understood best in terms of three layers: *cycles*, *packets*, and *transactions*. These layers correspond to the electrical, logical, and functional levels, respectively. A bus cycle is simply one complete period of the bus clock—it forms the unit of time and information transfer on the bus (the information is typically address or data). A packet is a contiguous sequence of cycles and is the mechanism by which one-way logical information is transferred on the bus. The first cycle of a packet is called the *header*. It carries address and control information, while subsequent cycles carry data. There are two different packet sizes: 2 cycles and 9 cycles. As shown in the figure below, a transaction consists of a request packet followed later by a corresponding reply packet. Together, the request and reply packets perform some logical function such as a memory read.

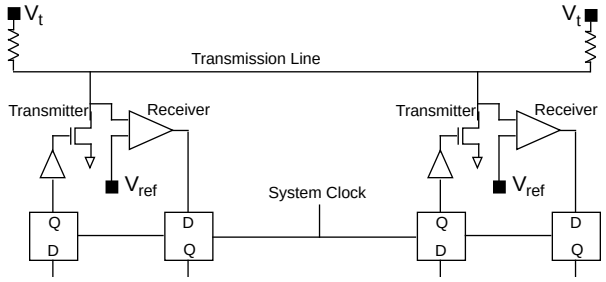
Each XDBus has an arbiter that permits the bus to be multiplexed amongst contending devices, which are identified by a unique DeviceID. Before a transaction can begin, the requesting device must get bus mastership from the arbiter. Once it has the bus, the device puts its packet on the bus once cycle at a time, and then waits for the reply packet. Packet transmission is uninterruptable in that no other device can take the bus away during this time, regardless of its priority. The transaction is completed when another device gets bus mastership and sends a reply packet. Request and reply packets may be separated by an arbitrary number of cycles. As pointed out earlier, the bus is free to be used in the interval between request and reply. The arbiter is designed to overlap processing of requests with transmission of packets such that no cycles are lost

XDBus was conceived and initially implemented in the Computer Science Laboratory at the Xerox Palo Alto Research Center. The bus technology is currently in its third generation of design and several commercial multiprocessors, including Sun Microsystem's new SPARCcenter 2000 series [5, 6], are using it as their main system interconnect.

2: Physical Characteristics

XDBus uses low voltage-swing GTL [1] transceivers connected to a terminated transmission-line to achieve both fast switching speeds and low power consumption. The speed and power advantages of GTL do not compromise noise immunity, however, and noise immunity is as good as that of ECL which is the industry benchmark.

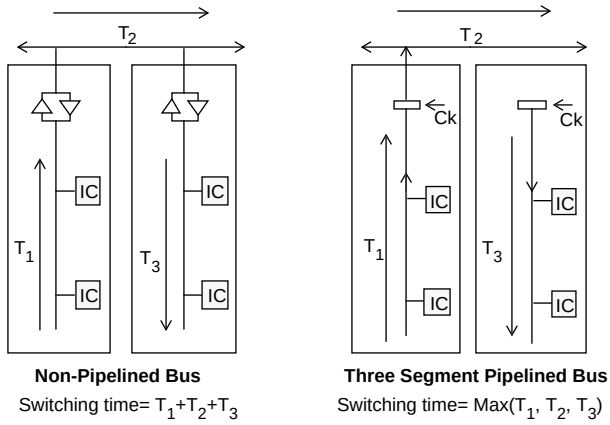
The figure below illustrates the signalling scheme. It shows two GTL transceivers connected to a single wire terminated at both ends at its characteristic impedance.



The terminated transmission line, combined with a small 800 mV voltage-swing ensures fast switching times. Furthermore, as shown in the figure, data is transferred synchronously from a flip-flop in the sender to a flip-flop in the receiver, so no time is wasted in synchronization and a complete system clock period is available for data transfer. This means that the clock rate is limited essentially only by signal transition time. Since one bit is transferred per clock for each wire, this also means that the data rate is as fast as possible under the given constraints.

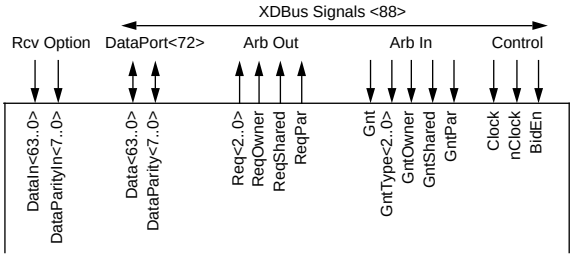
A low voltage swing also ensures low power consumption because power varies as the square of voltage-swing. An important aspect of the low power design is the use of simple open drain drivers. These drivers consume no power in the off state and very little power when on, so virtually all the power for the bus is consumed off chip in termination resistors. This low on chip power consumption is mostly what is responsible for the high levels of integration possible in XDBus based designs.

A final aspect of the signalling scheme is that it can be pipelined naturally: If the bus settling time is too long, the bus can be broken up into shorter segments connected via pipeline registers (which are present anyway for speed reasons). As shown in the figure below, each of these shorter segments can switch several times faster than the original long segment, and so the pipelined bus can be run much faster.



The following section explains how packet switching allows pipelining to be used in XDBus based systems with no loss in available bandwidth.

The figure below shows the chip level XDBus signals. There are 88 signals, 72 of which constitute a parity protected data/address path, 13 are used for communicating with the arbiter, and 3 are for clocks and miscellaneous control. An XDBus interface may be used in bidirectional mode, or optionally in unidirectional mode for pipelined bus configurations. In unidirectional mode, the Data Port is used for sending data and address, while the optional DataIn and DataParityIn wires are used for receiving data and address. In bidirectional mode, the Rcv Option port is not used and may be omitted altogether to save wires.



3: Packet Switching

A fast bus cycle time only provides the potential for high performance. The bus actually delivers high performance only if most of the cycles are used to transfer useful data. The ratio between the number of useful data cycles and total bus cycles is a measure of bus transport efficiency. XDBus uses *packet switching* to deliver a much higher transport efficiency than traditional *circuit switched* buses.

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

XDBus: A High-Performance, Consistent, Packet-Switched VLSI Bus

Pradeep Sindhu*, Jean-Marc Frailong*, Jean Gastinel*, Michel Cekleov,
Leo Yuan, Bill Gunning*, Don Curry*

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 XDBus is a low cost, synchronous, packet-switched VLSI bus designed for use in high performance multiprocessors. The bus provides an efficient coherency protocol which guarantees processors a consistent view of memory in the presence of caches and IO. Low-voltage swing (GTL) CMOS drivers connected to balanced transmission line traces ensure low power as well as high speed for chip, board, and as backplane applications.

The signalling scheme and coherency protocol work together to promote a high level of system integration, while permitting a wide variety of configurations to be realized. These configurations include small single board systems, multiple bus systems, multiboard backplane systems, and multi-level cache systems. The bus is used in several commercial systems including Sun Microsystem's new SPARCcenter 2000 series [5, 6].

1: Introduction

The XDBus is a synchronous, packet-switched bus designed to address the requirements of low cost, high bandwidth, cache coherency, and high integration in the design of an emerging class of powerful, but compact and cost-effective, general-purpose multiprocessors. While XDBus was designed as a multiprocessor interconnect, other applications including multimedia, ATM switches, and medium to high-end document systems can derive substantial benefits from using it.

Most of the advantages of XDBus stem from the synergy between an efficient packet-switched protocol layered on top of a fast, low voltage-swing signalling scheme. Implementations based on XDBus are low cost because a smaller number of wires switching at lower frequencies is needed to achieve a given level of performance; the signalling scheme uses ordinary CMOS technology and consumes little power, obviating the need for expensive cooling or exotic packaging; and finally, high integration ensures a small parts count and therefore low cost.

Implementations based on XDBus deliver high speed because the signalling scheme allows bus cycle time to be made extremely short, while protocol efficiency ensures that most of the raw bus bandwidth is delivered as useful data bandwidth to applications.

XDBus's physical and protocol layers interact to promote a high level of integration. Complex devices, including memory controllers, cache controllers, high speed network controllers, and external bus controllers that traditionally required entire boards can be integrated onto a single chip connected to the XDBus. The result is a high performance, but compact and cost effective system.

A unique advantage of XDBus is the broad range of architectural and packaging configurations it can support. Because of its low power and its ability to be pipelined, the XDBus can be used at the chip, board, and backplane levels. Its scalable performance can support systems with bandwidth needs from a few hundred Mbytes/sec to a few GBytes/sec through the use of bus pipelining and bus replication. Finally, XDBus also provides support for multilevel caches, which localizes bus traffic and enables many more processors to be combined into a single system.

XDBus provides an efficient protocol for maintaining multiprocessor cache coherency. With this protocol, the hardware ensures that multiple cached copies of data are kept consistent and that both input and output devices take cached data into account. The protocol is fundamentally write update but can emulate the spectrum of algorithms from write update to write invalidate. This flexibility enables applications to best utilize precious bus bandwidth. The coherency scheme also supports multilevel caches although no existing implementation uses this feature.

The XDBus contains 88 signals, 72 of which are accounted for by the data path. The remaining are used for control functions such as arbitration and clocking.