Palo Alto Research Center

Papers from the SunDragon Project

XEROX

Papers from the SunDragon Project

A CMOS Low Voltage Swing Transmission Line Transceiver, by Bill Gunning, Leo Yuan, Trung Nguyen, and Tony Wong.

XDBus: A High-Performance, Consistent, Packet-Switched VLSI Bus, by Pradeep Sindhu, Jean-Marc Frailong, Jean Gastinel, Michel Cekleov, Leo Yuan, Bill Gunning, and Don Curry.

The Next-Generation SPARC Multiprocessing System Architecture, by Jean-Marc Frailong, Michel Cekleov, Pradeep Sindhu, Jean Gastinel, Mike Splain, Jeff Price, and Ashok Singhal.

SPARCcenter 2000: Multiprocessing for the 90's! by Michel Cekleov, David Yen, Pradeep Sindhu, Jean-Marc Frailong, Jean Gastinel, Mike Splain, Jeff Price, Gary Beck, Bjorn Liencres, Fred Cerauskis, Chip Coffin, Dave Bassett, David Broniarczyk, Steve Fosth, Tim Nguyen, Raymond Ng, Jeff Hoel, Don Curry, Leo Yuan, Roland Lee, Alex Kwok, Ashok Singhal, Chris Cheng, Greg Dykema, Steve York, Bill Gunning, Bill Jackson, Atsushi Kasuya, Dean Angelico, Marc Levitt, Medhi Mothashemi, David Lemenski, Lissy Bland, and Tung Pham.

The Years of the Dragon reprinted from "Benchmark" magazine, Spring 1993.

CSL-93-17 December 1993 [93-00139]

CR Categories and Subject Descriptors: B.3.2 **[Design Styles]:** cache memory, shared memory; C.1.2 **[Multiple Data Stream Architectures]:** MIMP processors, C.0 **[General]:** system architectures; B.4.3 **[Interconnection]:** VLSI bus interconnect, backplanes; B.7.1 **[Types and Design Styles]:** Advanced technologies, input/output circuits

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Introduction

From 1988 to 1992 a remarkable cooperation took place between the Computer Science Lab at Xerox PARC and Sun Microsystems. The CSL "Dragon" multiprocessor research project moved in with a development team at Sun to create a series of Sun products based on the Dragon team's inventions in bus and memory architecture. The business details are beyond the scope of this tech report -- but now that products are shipping, accountants on both sides seem to be very happy.

Here are five internal reports on aspects of the SunDragon project, as it came to be known. All are also scheduled to be published elsewhere, as indicated with each paper. "A CMOS Low Voltage Swing..." describes the basic logic design, known as GTL (Gunning-Transistor-Logic) for the SunDragon bus. "XDBus: ..." describes the logic for the bus itself, including its support for cache consistency and its packet-switched protocol. "The Next Generation..." describes the fuller system architecture of the Sparcenter 2000-'s use of the XDBus as the heart of a commercial scalable multiprocessor. "SPARCenter 2000: ..." briefly describes the full SPARCenter 2000 system from a user perspective. Finally, "The Year of..." offers some commentary and interviews from Sun and Xerox on the process of this unusual cooperation.

In closing, let me honor and acknowledge Jean Gastinel, leader of the Dragon project, whose vision saw the opportunity with Sun and whose technical and managerial leadership made it a success.

Mark Weiser
Head, Computer Science Lab

A CMOS Low Voltage Swing Transmission Line Transceiver

Paper WP3.7

Authors:

Bill Gunning 415-336-1194 (tel) 415-964-0706 (fax)

Xerox PARC
3333 Coyote Hill Road
Palo Alto, Ca 94304

Leo Yuan 415-336-3841 (tel) 415-964-0706 (fax)

Sun Microsystems MTV 16-10
1501 Salado Drive
Mountain View, Ca 94043

Trung Nguyen 408-433-7546 (tel) 408-434-6457 (fax)

LSI Logic
1551 McCarthy Blvd.
Milpitas, Ca 95035

Tony Wong 408-433-7539 (tel) 408-434-6457 (fax)

LSI Logic
1551 McCarthy Blvd.
Milpitas, Ca 95035

Abstract

CMOS I/O circuits designed for terminated transmission line inter-chip communication are described. The nominal signal swing (800 mV) and signal quality are comparable with ECL systems. Typical on-chip power is 15 mW per I/O. Several ASICs with 160 I/Os have been built.

High performance CMOS ASIC designs with over one hundred I/Os on a single chip are becoming common. Wide (e.g. 72 bit) high-speed signal busses are often used to interconnect VLSI components. Conventional unterminated interconnects for CMOS level signals usually have poor signal quality with severe overshoot and ringing, accompanied by EMI and a tendency to trigger latch-up. ECL based high performance systems have used terminated transmission line interconnect to avoid ringing and reflections. An extensive body of experience has been established in building high performance ECL systems [1]. The disadvantage that is cited most often is the relatively high power dissipation inherent in ECL implementations.

CMOS can also be used in a terminated transmission line environment. Fig. 1 shows a simplified representation of a bidirectional transmission line with I/O transceiver cells. The drivers are open drain N-channel devices and the receivers are differential comparators. When all drivers are inactive, the high level signal (Voh) is established by the terminator supply voltage Vt.

The loaded characteristic impedance of stripline signal traces on a printed circuit board can be about 50 ohms. As shown in Fig. 1, when there is more than one driver, the transmission line must be terminated at both ends to prevent reflections. System arbitration allows only one driver access to a line at any time. The load seen by each driver is about 25 ohms. The power dissipated is reduced if the signal voltage swing and the Vol level are small. The minimum voltage swing must be large enough to assure acceptable noise margin.

In this CMOS I/O design, referred to as GTL, the output levels are:

Vol_{max} = 0.4 volts, Voh_{min} = Vt = 1.2 volts and Vref = 0.8 volts.

The signal amplitude is close to that of ECL. The function of Vref corresponds to that of Vbb in ECL.

The maximum on-chip power dissipated by an output transistor driving a doubly terminated 50 ohms transmission line load is $(0.8 / 25) \times 0.4 = 12.8$ mW. The *typical* power for this size active driver is less, because Vol is smaller. The power dissipated in an inactive driver is essentially zero. Similar drivers are used for backplane traces with a loaded characteristic impedance of about 35 ohms.

We have designed several large CMOS ASICs each of which has about 160 transmission line I/O cells. Table 1 shows the estimated nominal power dissipation using ECL, BTL and GTL technology and logic levels. All drivers are assumed to be active and use 0 and +5 volt power. BTL refers to a design used by the proposed IEEE -896 Futurebus [2]. Assumptions are in the appendix.

Fig. 2 shows a simplified schematic diagram of a driver implemented with components available in the existing I/O cells of a standard CMOS gate array technology [3]. When a driver pulls low, damping is provided by the load resistance and the on resistance of the driver. When a driver turns off, an underdamped overshoot caused by package inductance can occur. This driver design example includes an arrangement to reduce overshoot and the turn off di/dt.

When Vin is low, M4 and M3 are conducting and M1 and M2 are not conducting. When Vin goes from low to high, the turn off transition at the drain of M4 is controlled by the temporary path through M2 and M3 which ties the gate of M4 to its drain (M1 is weak). This causes M4 to conduct when its Vsd is larger than the N-channel threshold. M1 will pull the gate of M4 to ground when M3 is turned off. The signal at the gate of M3 is delayed by the two inverters for about 1 ns. Typical propagation delay from Vin to Out is 1 ns.

The capacitance of an inactive driver is 2.5 pF at the die pad. Low capacitance is important to reduce refections when a signal generated by another driver passes an inactive driver.

Fig. 3 shows a differential receiver that has also been designed using components in the standard gate array technology [3]. The DC gain and offset of the receiver, combined with its input register, guarantee that the register will reset for Vin - Vref > 50 mV, and the register will set for Vref - Vin > 50 mV over process, power and junction temperature variations. This small uncertainty band improves noise margin. Typical propagation delay is 1.5 ns. Typical average power dissipation is 5.5 mW.

The SSO measurements shown in Fig. 4 were made with a 20 GHz bandwidth oscilloscope using a 1 GHz bandwidth active probe. The chip was programmed to switch 142 I/O drivers simultaneously. The load is a doubly terminated 50Ω stripline.

The measurements show Vol ≈ 0.23 volts. The total simultaneous switched current for 142 drivers is about 142*(1.2 - 0.23)/25 = 5.5 A. The ground bounce shown by an unswitched driver is about 130 mV. The package is a custom 383 pin PGA using glass BT resin, on-package surface mount bypass capacitors and power and ground planes.

Acknowledgments:

GTL technology was first developed and implemented at Xerox PARC. Thanks are given to our many colleagues for their support and technical contributions. The following people were particularly helpful.

David Yen, Christopher Cheng - Sun Microsystems; Dan Wong - LSI Logic; Richard Bruce, Jean Gastinel, Jeffrey Hoel, Alfred Permuy, Ed Richley - Xerox PARC

References:

[1] W. R. Blood, "MECL System design handbook" - Motorola semiconductor products.

[2] National Semiconductor DS3890 octal trapezoidal driver.

[3] LSI Logic LCA100K.

Appendix:

The MC10H123 triple bus driver is designed to drive a 25 ohm load in a multiple drive point (party line) bus. It uses 56 mA. Subtract 5 mA for the shared bias circuit. Assume that each pre-driver uses $(56 - 5) \times 5/3 = 85$ mW. The output driver power is $1.1/25 \times 0.9 = 39.6$ mW. The total power is 125 mW. 160 drivers $\approx$ 20 watts.

For BTL, the NSC DS3890 octal driver uses 50 mA typical. Assume that this is $5 \times 50/8 = 31.25$ mW for each pre-driver. The nominal output stage power (with Vol = 1V) is $1/25 \times 1 = 40$ mW. 160 drivers $\approx$ 11 watts.

A worst case GTL driver Ron is $0.4/(0.8/25) = 12.5\ \Omega$. Assume typical Ron is $6.25\ \Omega$. Typical driver current is $1.2/(25 + 6.25) = 38.4$ mA. Vol = 0.24 V. Active driver power = 9.2 mW. 160 drivers $\approx$ 1.5 watts.

Figure captions

Fig.1 Bidirectional transmission line bus with I/O transceivers

Table1 Estimated nominal power for 160 active I/O drivers.

Fig.2 CMOS I/O driver with turn-off damping.

Fig.3 CMOS I/O receiver with uncertainty band < 100 mV.

Fig.4 SSO waveforms with142 drivers switching 5.5A in 383 pin PPGA.

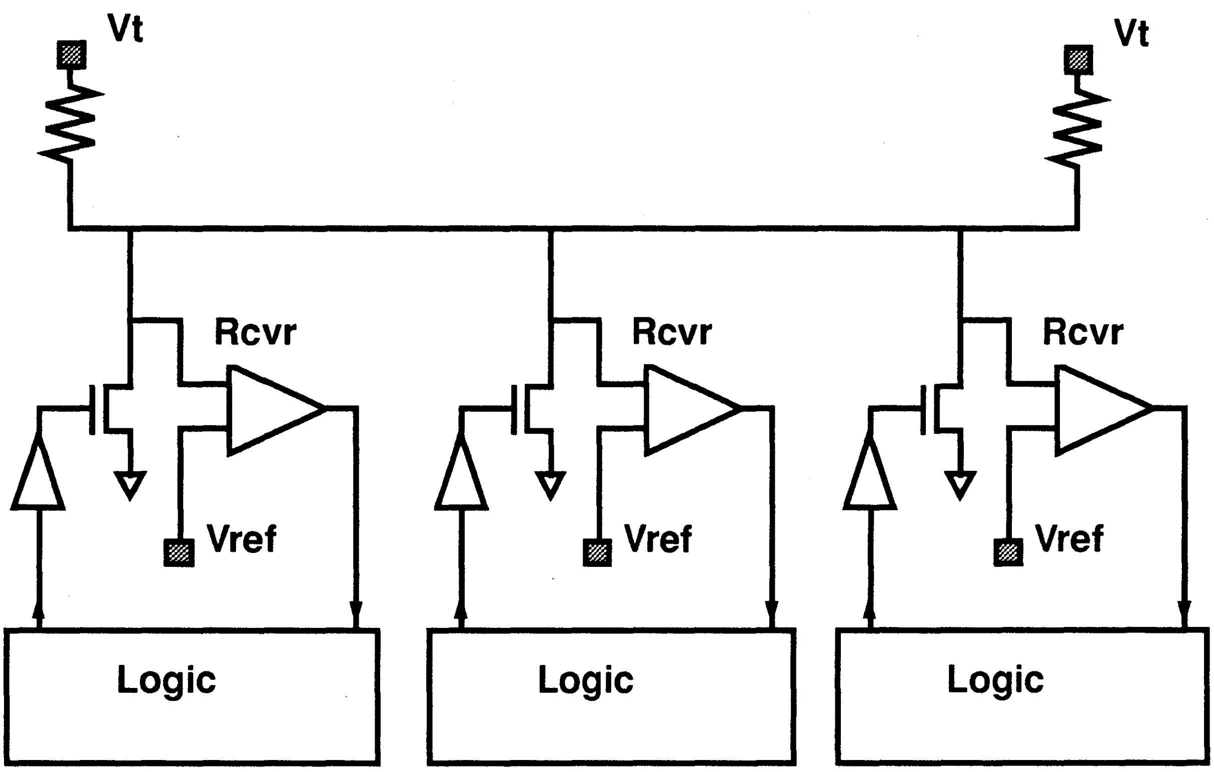


Fig.1Bidirectional transmission line bus with I/O transceivers

Logic level	Power (watts)	Termination (both ends)
ECL	20	50 ohms to 3.0 V.
BTL	11	50 ohms to 2.0 V.
GTL	1.5	50 ohms to 1.2 V.

Table1Estimated nominal power for 160 active I/O drivers.

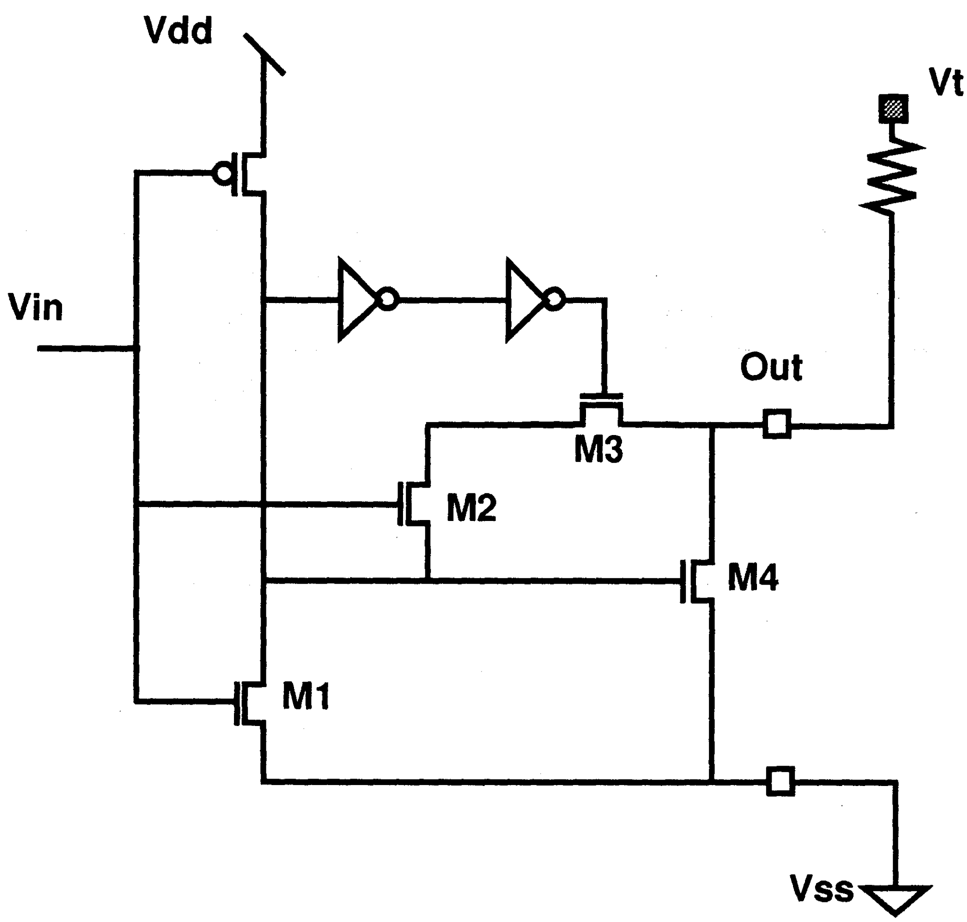


Fig.2CMOS I/O driver with turn-off damping.

Paper WP3.7 Session: High performance Circuits
Authors: Gunning, Yuan, Nguyen & Wong

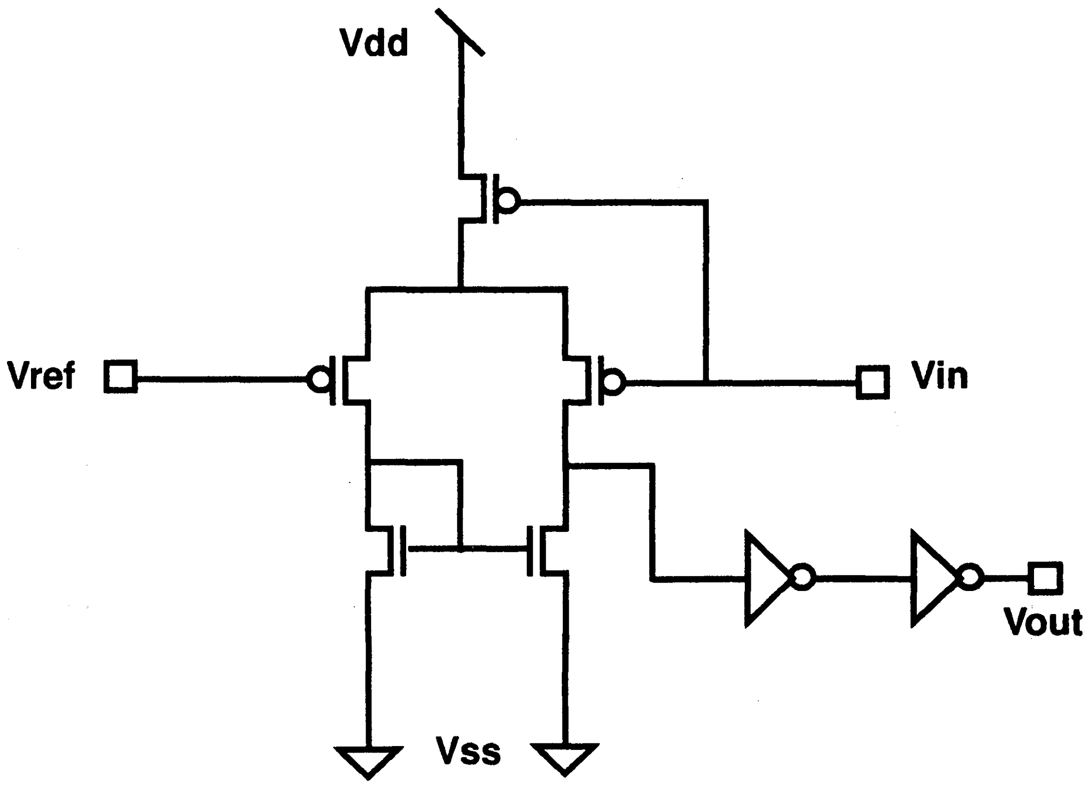


Fig.3CMOS I/O receiver with uncertainty band < 100 mV.

Paper WP3.7 Session: High performance Circuits
Authors: Gunning, Yuan, Nguyen & Wong

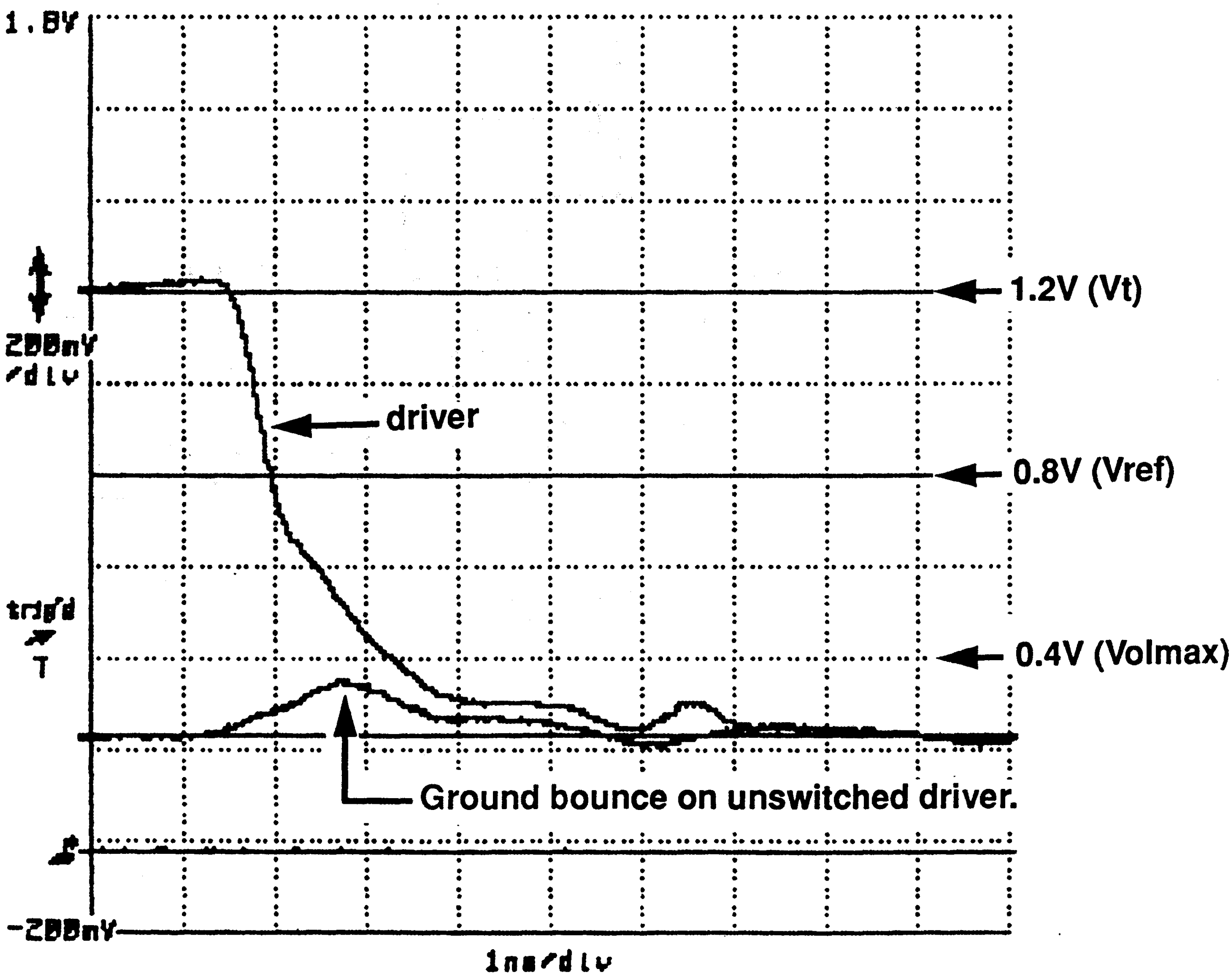


Fig.4 SSO waveforms with142 drivers switching 5.5A in 383 pin PPGA.

Paper WP3.7 Session: High performance Circuits
Authors: Gunning, Yuan, Nguyen & Wong

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.

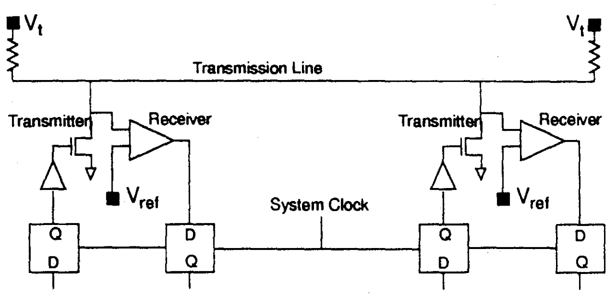
The XDBus contains 88 signals, 72 of which are accounted for by the data path. The remaining are used for control functions such as arbitration and clocking.

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.

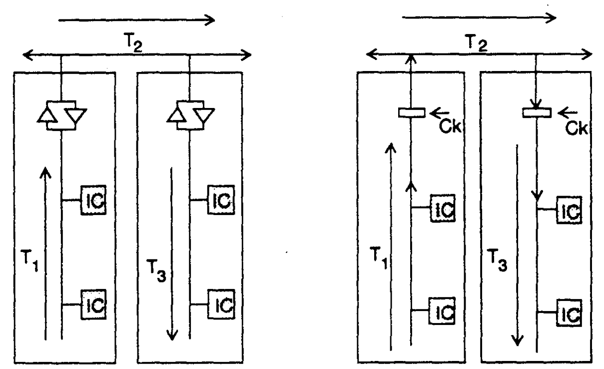
The figure below illustrates the signalling scheme. It shows two GTL transceivers connected to a single wire terminated at both ends at its characteristic impedance.



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.

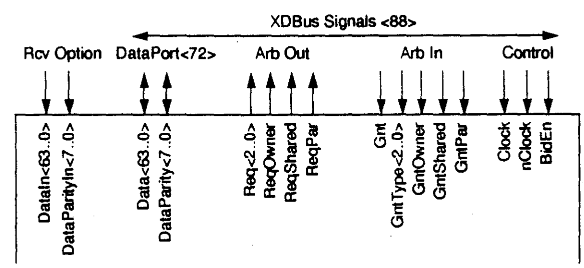


Non-Pipelined Bus
Switching time= $T_1+T_2+T_3$

Three Segment Pipelined Bus
Switching time= $\mathrm{Max}(T_1, T_2, T_3)$

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

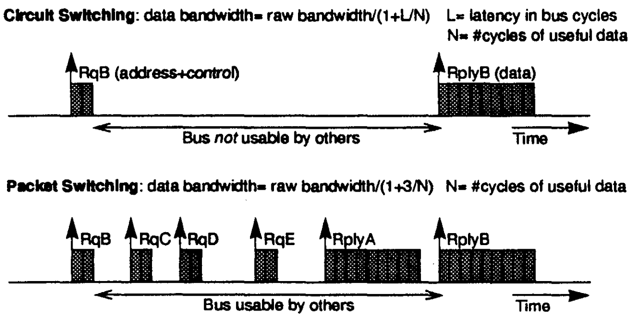
The figure below shows the chip level XDBus signals. There are 88 signals, 72 of which constitute a parity protected data/address path, 13 are used for communicating with the arbiter, and 3 are for clocks and miscellaneous control. An XDBus interface may be used in bidirectional mode, or optionally in 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

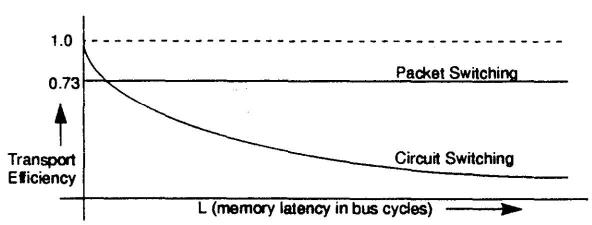
A fast bus cycle time only provides the potential for high performance. The bus actually delivers high performance only if most of the cycles are used to transfer useful data. The ratio between the number of useful data cycles and total bus cycles is a measure of bus transport efficiency. XDBus uses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

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

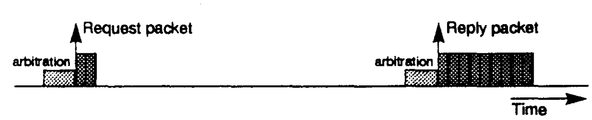
A related advantage of packet switching is that it works well with bus pipelining. The extra pipeline stages add to latency, but do not affect the useful bus bandwidth. In contrast, if pipelining is used in a circuit switched bus, the extra pipeline stages not only add to memory latency, but also subtract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

TransType	Control	DeviceID	Byte Address

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.

TransType	Control	Originator's DeviceID	Original Byte Address

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 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.

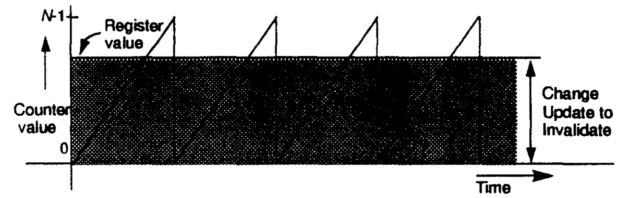
A final advantage of XDBus as a VLSI interconnect is that its signalling scheme is compatible with the coming generation of 3.3V, and even lower voltage, technologies. What makes this possible is that the GTL's signal swing is fixed by the external pull up voltage and its threshold is set by a reference 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.

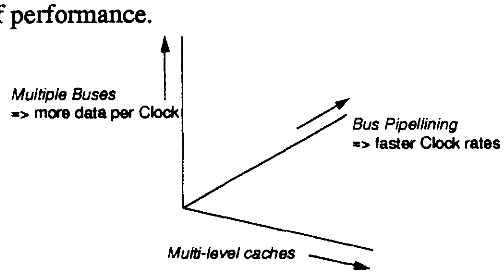
In traditional bus designs, interrupts are communicated from IO devices to processors via dedicated wires. This scheme has the obvious problem of connectivity when multiple IO devices must communicate with multiple processors. It also has the drawback that the communication paths are fixed and interprocessor 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.

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.



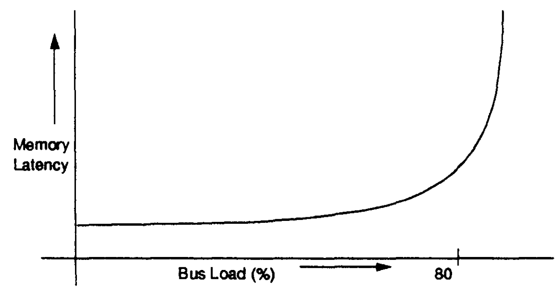
XDBus provides scalability along three orthogonal dimensions

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 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 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

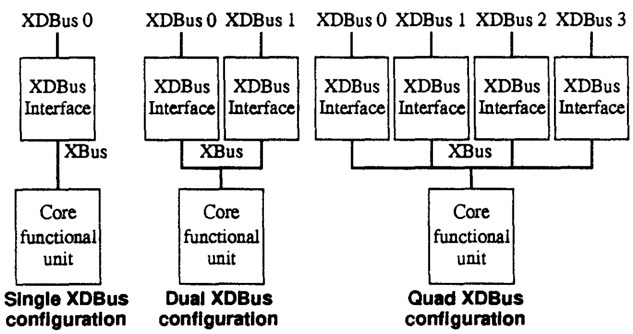
When multiple XDBuses are used in a system, they are interleaved on 256-byte boundaries, based on the physical address being referenced. Packets which do not carry a physical address (interrupts and TLB maintenance) always use the first interleave (interleave 0).

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

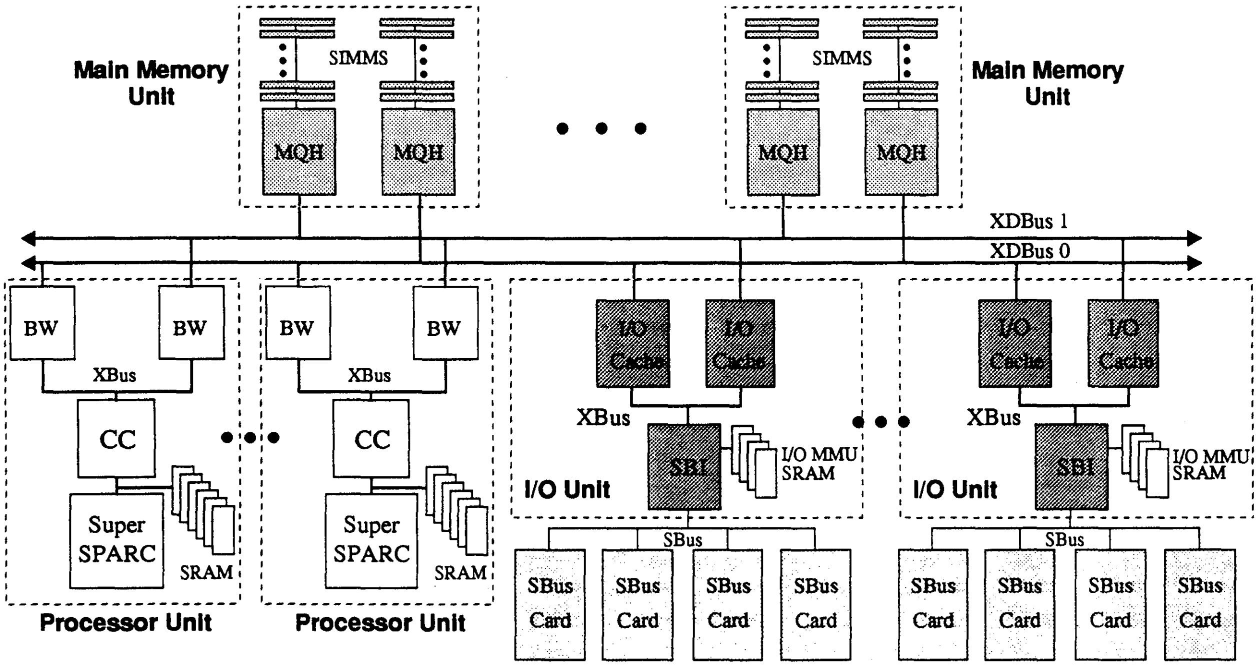
The processor unit is based on the TI SuperSPARC processor [4] (TMS390Z50) with a 1 MB parity-protected second level cache controlled by a TI SuperSPARC MultiCache Controller [5] (CC, TMS390Z55). The XDBus interface function is performed by the Bus Watcher (BW).

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.

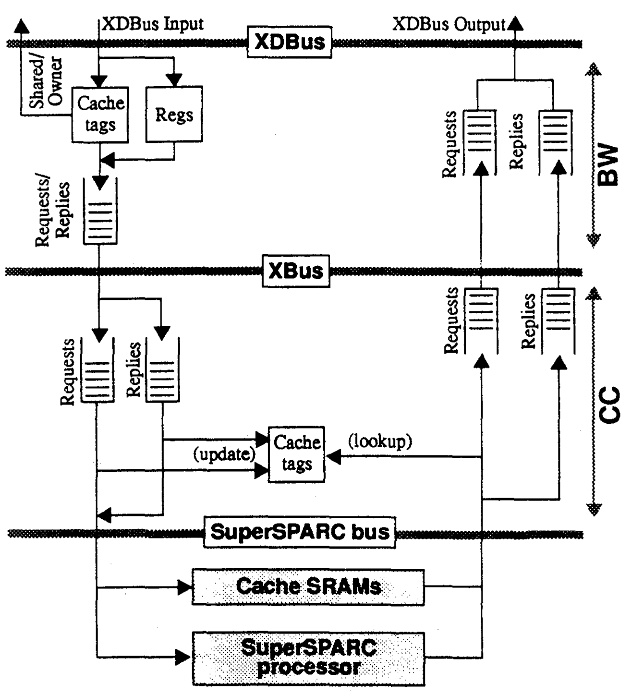
The two first-level caches are entirely internal to the SuperSPARC processor, which supports external invalidation based on physical addressing. Both are interleaved in such a way that they are indexed by address bits below the



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



Processor Unit block diagram

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.

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

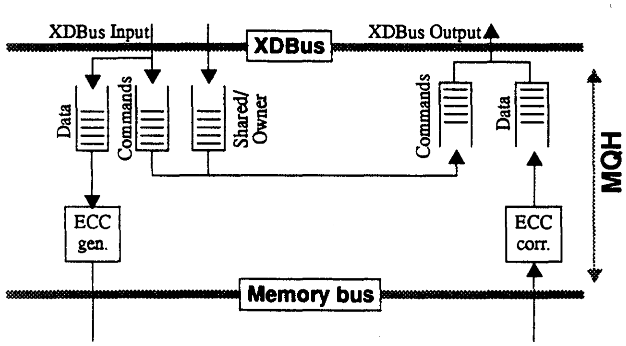
BW while a cache miss is outstanding, providing a faster return path for the reply.

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

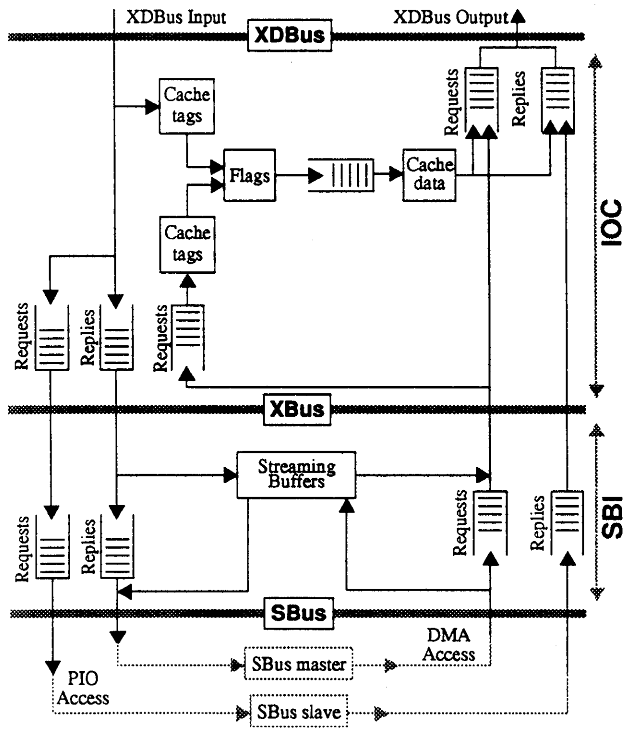
The I/O unit provides an SBus [6] with up to four masters. The XDBus interface consists of an I/O Cache (IOC), while the core functional unit consists of an SBus interface controller (SBI) and an external SRAM which provides translation tables for DMA accesses.

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



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 '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.

SPARCcenter 2000: Multiprocessing for the 90's!

Michel Cekleov, David Yen, Pradeep Sindhu*, Jean-Marc Frailong*, Jean Gastinel*, Mike Splain, Jeff Price, Gary Beck, Bjorn Liencres, Fred Cerauskis, Chip Coffin, Dave Bassett, David Broniarczyk, Steve Fosth, Tim Nguyen, Raymond Ng, Jeff Hoel*, Don Curry*, Leo Yuan, Roland Lee, Alex Kwok, Ashok Singhal, Chris Cheng, Greg Dykema, Steve York, Bill Gunning*, Bill Jackson*, Atsushi Kasuya*, Dean Angelico, Marc Levitt, Medhi Mothashemi, David Lemenski, Lissy Bland*, Tung Pham*.

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

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

Abstract

The SPARCcenter 2000 is the first implementation of a new generation of symmetric high-performance, highly configurable SPARC multiprocessor systems. With up to 20 processors, extensive main memory, expansibility and large IO capacity, the SPARCcenter 2000 is designed to meet the computing needs of most corporate data centers. A high throughput system interconnect, composed of two interleaved XDBuses, combined with multiprocessor scalability, make the SPARCcenter 2000 the right platform for compute intensive tasks. Reliability and availability features, full SPARC binary compatibility, IO scalability and Solaris 2.X MP capability also make the SPARCcenter 2000 the ideal rightsizing platform for commercial and RDBMS applications.

Introduction

The SPARCcenter 2000 defines a new breed of shared-memory multiprocessor computers designed to accommodate the needs of most organizations, from large department to medium-scale enterprises. It uses a modular architecture composed of three types of units: the Processor Unit, the Memory Unit and the I/O Unit. All these units are interconnected through two XDBuses. The XDBus is a high-speed consistent packet-switched bus [1].

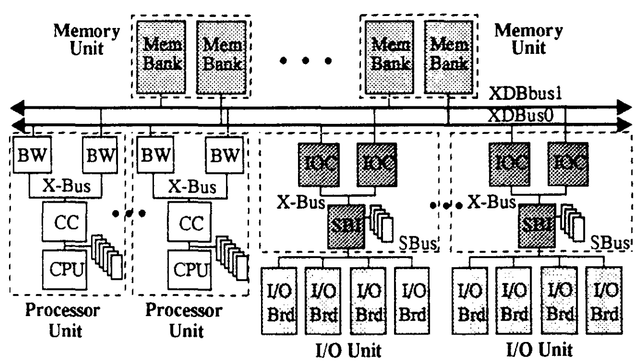
The SPARCcenter 2000 provides unparalleled expansion capability in combination with excellent scalability in three dimensions: compute power, main memory capacity and I/O bandwidth. The XDBus is flexible enough and fast enough to deliver outstanding performance and maintain scalability to levels which are unheard-of in today's marketplace.

This paper describes the implementation of the SPARCcenter 2000. It begins with an overview of the system architecture, followed by a description of the implementation of the System Board, and its various components: the Processor Unit, the Memory Unit, the I/O Unit, the BootBus. These sections focus on the salient details of the implementation. For a description of the architecture the reader can refer to [2]. The following sections describe the implementation of the Control board, the Arbitration, and the support of the JTAG scanning logic. Finally the paper closes by discussing some less visible but potentially important features of the SPARCcenter 2000: reliability, availability, and serviceability. A few performance numbers are quoted in the conclusion.

System Overview

System Architecture

The system architecture of the SPARCcenter 2000 is depicted by the following picture:



SPARCcenter 2000 system Architecture

The heart of the SPARCcenter 2000 is a high-speed packet-switched bus complex which provides a very high data bandwidth. The backplane consists of two XDBuses each providing 320 MB/s of data throughput with at a 40 Mhz clock rate. The XDBuses operate in parallel and the system can be rebooted with a single XDBus in case of a permanent failure of one of them. The system's functional units are connected to both XDBuses. Memory banks are attached to individual XDBuses and a memory unit is composed of two interleaved memory banks.

The SPARCcenter 2000 can use up to 20 SuperSPARC processors [3]. The SPARCcenter 2000 uses processor modules compatible with the SPARCserver 600MP and

SPARCstation 10 systems.

The main memory is configured in multiple Memory Units. All these units have the same access time from every processor and I/O device regardless of their physical locations in the system. Physical memory addresses are interleaved between the two XDBuses on a 256-byte boundary and memory banks attached to the same bus can also be interleaved to avoid bottlenecks. A Memory Unit can have a memory capacity between 64 MB with 4 Mbit DRAM chips and 512 MB with 16 Mbit DRAM chips. A fully configured system can support 5 GB of main memory.

The SPARCcenter 2000 offers incrementally expandable I/O with up to 10 SBuses. Each SBus supports 4 SBus slots for a maximum configuration of 40 SBus peripheral boards. Each SBus is connected to the XDBuses through an I/O Unit. Like memory, all SBuses are accessed with the same latency from every processor. Each SBus delivers 50 MB/s of sustainable data throughput. A SPARCcenter 2000 can be configured with up to 18 2.1 GB DSCSI-2 disks in the system rack for a maximum of 38 GB internal capacity. Expansion racks with 48 drives for a capacity of 100 GB are also available. The SPARCcenter 2000 can also be configured with a multitude of independent network interfaces. This exceptional I/O capacity and configurability makes the SPARCcenter 2000 suitable for very large applications.

Although the architecture of the SPARCcenter 2000 is similar to other large-scale symmetric multiprocessing systems, the expansion capability, the overall system balance with commensurate memory bus bandwidth makes this system unique in the industry.

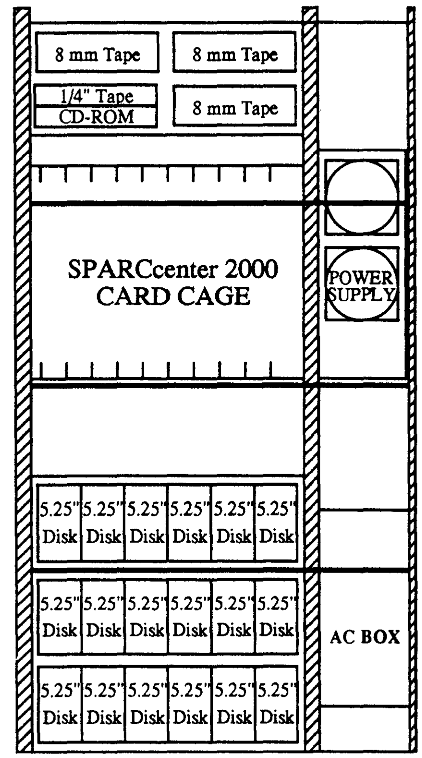
Packaging and Power

The SPARCcenter 2000 consists of 10 System Boards configured in a 10 slot XDBus backplane and a Control Board mounted on the other side of the backplane. The SPARCcenter 2000 uses a 9U format for the System Board. The basic system is packaged in the standard 56" SunRack with up to 18 5.25" disk drives, a CDROM, 1/4" tape and up to three 8mm 5 GB tape drives for backup.

The power requirements do not exceed 180 W per slot, including 10 W per SBus slot. Most power is drawn from the +5V supply. There is also a +1.2V supply for XDBus termination and a +/-12V for the SBus boards. The power supply is resistant to most kind of power fluctuations. The system can continue functioning during a brownout of 160 VAC for at least 15 minutes. It can also tolerate complete AC brownout if they are limited to a single cycle.

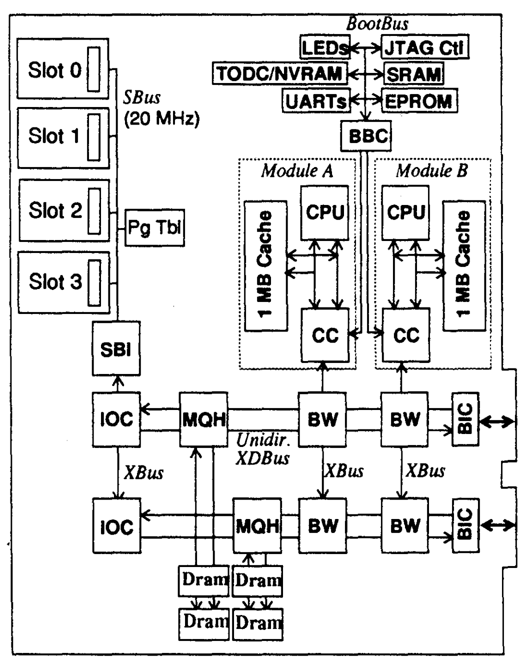
System Board

The System Board is the primary component in the implementation. A system may contain up to 10 System Boards. Each board contains a backplane interface connection to dual XDBuses, two sockets for the SPARCmodules, sockets for 16 SIMMs, an SBus with four slots, the local



SPARCcenter 2000 system cabinet configuration

XDBus segments and a JTAG interface for diagnostics and configuration. Each System Board contains two Processor Units, a Memory Unit and an I/O Unit.

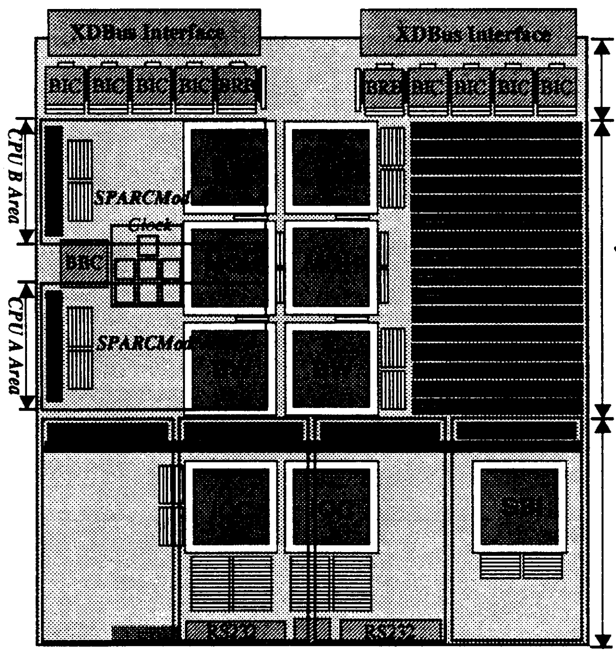


System Board Logic Diagram

The actual implementation is highly integrated: the System Board consists primarily of nine large 100K gate CMOS ASICs: four Bus Watchers (BW), two Memory

Queue Handlers (MQH), an SBI Interface (SBI) and two I/O Caches (IOC) and ten smaller ASICs: two Board Bus Arbiters (BARB) and eight bit-slice pipeline registers to connect the on-board XDBus segments to the backplane (BIC). The System Board also supports two SPARCmodules consisting of a 3 Million transistor SuperSPARC processor and a 1MB direct-map cache memory. The cache memory is composed of a 2 Million transistor Cache Controller (CC) and the SRAM array. The SPARC modules are not actually part of the System Board. Instead, the SPARCmodules are mounted on the System Board through dedicated connectors. Similarly, the memory DRAM memory chips are not on the System Board but on SIMMs which plug vertically.

The following block diagram shows the location of the major components.



SPARCcenter 2000 System Board Layout

This highly integrated design is key to system reliability. It also simplifies design without compromising performance and configuration flexibility.

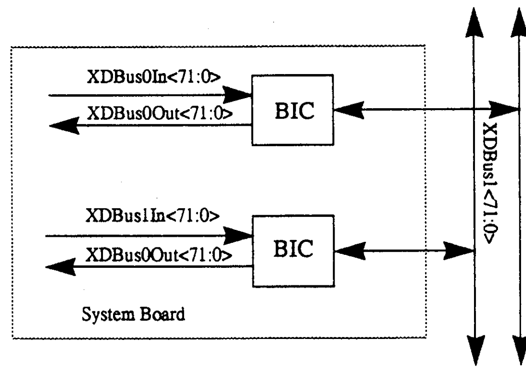
The System Board has fourteen layers, eight for signals and six for power and ground.

XDBus Backplane

Pipelined Implementation

The backplane connects up to 10 System Boards. It consists of two bidirectional segments of the XDBuses.The System Board supports unidirectional segments for each of the XDBuses. The figure below shows the architecture of the XDBus complex.

The on-board XDBuses are each composed of the unidi-



Bidirectional Backplane XDBuses
and Unidirectional on-board XDBuses

rectional segments XDBusIn and XDBusOut. Each of these segments consists of 64 bits for address and data and 8 bits for parity. The Bus Interface chips (BIC) separate the unidirectional on-board XDBuses and the bidirectional backplane XDBuses. The BIC is composed of two pipeline registers, one between the outbound on-board segment and the backplane segment and the other between the backplane segment and the inbound on-board segment. Each BIC has an 18 bit wide data path including 16 bits of data/address and 2 bits of parity. Four BICs are necessary to provide the interface between the on-board and backplane segments of a single XDBus.

Packet-Switched Protocol

The XDBus uses a packet-switched protocol (also known as a split transaction protocol) to transfer data between clients. A packet-switched protocol offers a larger overall throughput than the more conventional circuit-switched protocol. In a packet-switched protocol, the requestor arbitrates for the control of the bus and as soon as it is granted it sends a request packet and immediately releases the bus. The bus is free to be used by other clients while the request is being processed. When the requested data is available a reply packet is issued. Reply packets are tagged so that they can be matched with the corresponding request. A packet-switched protocol permits an optimal utilization of the raw bandwidth.

Low-Power Electrical Power Implementation

The XDBus runs at 40 MHz and uses low voltage-swing technology called GTL (Gunning Transceiver Logic) [4]. GTL uses a 0.8V voltage swing between 0.4 and 1.2V. This technology is specially designed for high-speed, high density CMOS gate arrays. GTL permits CMOS to be used in a terminated transmission line environment. Because the power dissipation is very low, a wide bus like the XDBus can be driven directly from an ASIC without having to use costly external drivers.

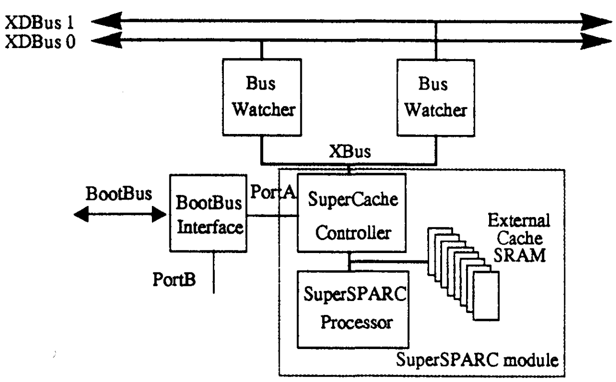
Configuration

The backplane also provides identifiers (BoardIDs) to uniquely identify each System Board. A System Board can be plugged in any slot of the cardcage and the BoardID can be accessed at system configuration time through a pseudo register. There are no jumper or switch settings required for any configuration. This avoids the likelihood of installation errors.

A minimum system consists of at least one System Board with a single processor and one Memory Unit. Because all units are equally accessible, the specific locations of the memory, processors, and I/O devices are not fixed. If an application requires only 4 processors but 3 GB of memory, the system can be configured with 6 System Boards with fully populated Memory Units but only two SPARCmodules. The boards may be plugged in any slot.

Processor Unit

The Processor Unit consists of a SuperSPARC processor, an external cache and system support devices connected to the BootBus. The figure below illustrates the main components of the Processor Unit and their interconnections:



Processor Unit

The external cache includes the Cache Controller and two Bus Watcher ASICs and 1 MB of parity protected SRAM. The SuperSPARC processor, the Cache Controller and the SRAM are located on the SuperSPARC module. The module is a small daughter card which plugs through a 100 pin connector onto the System Board. The Bus Watchers are located directly on the System Board.

The SuperSPARC processor is a highly integrated chip which includes two integer units, a floating point unit, a branch processor, a SPARC reference MMU and a 36 KB on-chip set-associative cache [3]. This processor is capable of executing up to 3 instructions per cycle if they are fetched from the internal cache and scheduled properly. The processor is a 3 Million transistor custom BiCMOS chip.

The Cache Controller controls 1 MB of combined instruction and data external cache. The cache is physically accessed and is managed as a write-back cache. The current SRAM technology limits the size to 1 MB but the Cache Controller can support up to 2 MB of cache. The Cache Controller is a 2.2 Million transistor BiCMOS chip [5].

The Cache Controller and the two Bus Watchers implement the cache consistency protocol. The Bus Watcher chips contain a copy of the cache tags to minimize contention between the processor and the XDBus accesses. The cache consistency protocol relies on snooping the XDBus traffic. The Bus Watcher basically filters out almost all bus transactions leaving the processor free to access the cache most of the time. When sharing is detected, the Bus Watcher updates some state bits or retrieves the requested data from the cache. Each Bus Watcher contains half of the duplicated tags and are interleaved on a 256 byte boundary.

The Bus Watchers and the Cache Controller are interconnected by a local packet-switched bus known as the XBus. The XBus is very similar to the XDBus. Transactions supported by the XBus are similar to XDBus transactions. The difference is in the use of dedicated commands to maintain the two copies of cache tags consistently. The use of a packet-switched protocol on the XBus is also key to the performance of the processor in a multiprocessor environment. Although the external cache handles only a single miss at a time, multiple requests may be outstanding. For instance multiple requests for block invalidation or update can be issued from the same Processor Unit. This guarantees that the processor does not stall even when there is lots of data sharing with other processors.

The XBus is also uses GTL transceivers which allow a direct pin to pin connection between the Cache Controller and the Bus Watcher. The XBus is also clocked at 40 MHz.

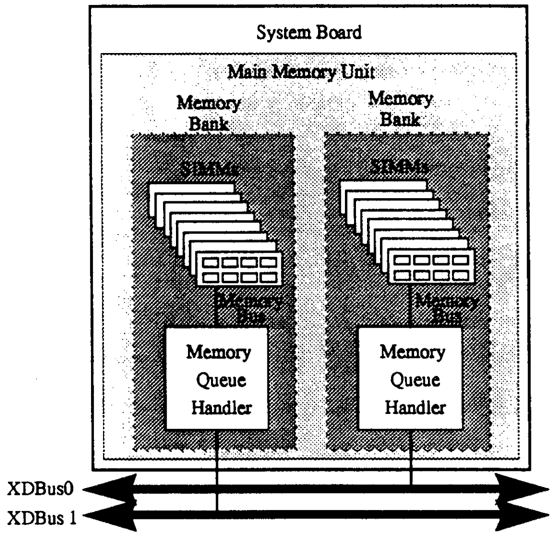
Although the first generation of the SPARCcenter 2000 utilizes 40 MHz SuperSPARC processors, faster SuperSPARC processor modules can be used to upgrade existing SPARCcenter 2000 systems. Most of the Cache Controller operates synchronously to the processor clock. The XBus interface operates synchronously to the system clock and there is an asynchronous boundary inside the Cache Controller.

The Cache Controller is also connected to a local bus called the BootBus. Support devices like a time-of-day-clock, UART, scratch-pad memory, EPROM are attached to the BootBus. The BootBus is shared by the two processors on the same System Board.

Memory Unit

Overview

The SPARCcenter 2000 Memory Unit consists of two memory banks, one per XDBus. A memory bank is defined as a memory array controlled by a Memory Queue Handler (MQH) ASIC. The MQH interfaces directly to the XDBus and controls the memory array. The MQH supports read and write operations on 64-byte blocks only (unit of block



Main Memory Unit

transfer on the XDBus). The MQH does not support writes on smaller quantities and the main memory in the SPARCcenter 2000 is always cached.

A memory bank is the unit of interleaving. A memory bank consists of one or two groups of 4 custom SIMMs. These are the two possible configurations (in addition, of course, to the case where the bank is not populated at all).

The MQH can handle DRAM densities from 1 Mbit to 256 Mbit. The first generation of SPARCcenter 2000 uses 4 Mbit and 16 Mbit DRAMs.

The MQH is connected to the SIMM through a Memory Bus. This is a 72 bit wide TTL bus also clocked at 40 MHz. However, the timing access to the SIMM is fully programmable allowing DRAM with different timing to be used.

The memory is protected by an Error Correcting code which detects and corrects single-bit errors and detects all double-bit errors. It can also detect triple and quadruple-bit errors if the erroneous bits are in the same nibble.

The MQH is implemented as a 100K CMOS gate array.

Interleaving

Because of XDBus interleaving, for each memory group on a given XDBus there must be an identical group on the other XDBus. The memory size increment is the memory capacity of two groups of 4 SIMMs.

The physical memory address space is entirely programmable and each memory group is controlled by a distinct address decode register. The memory bank on the same XDBus can be configured for no interleave, 2-way interleave or 4-way interleave by programming the address decoding registers of the MQHs.

In a shared-memory symmetric multiprocessor system the motivation for an interleaved main memory is to allow multiple independent accesses. With the XDBus packet-switched protocol, multiple memory transactions issued by different processors and I/O devices can be pending at the same time. In large configurations, memory bank interleaving reduces the probability of having "hot spots" developing on a given MQH.

SIMM

The SPARCcenter 2000 uses custom SIMMs. The SIMM is composed of Crossbar ASIC and memory chips on a small board. Each SIMM is organized as 18 bits wide (16 bits of data and 2 bits for ECC), so 4 SIMMs operating in parallel are necessary to interface to the memory bus.

Each SIMM contains 18 x4 DRAM chips and four crossbar ASICs. Since the DRAM access time is nominally four system clock cycles, the DRAM data path on the SIMM was designed to be four times the memory bus width. The Crossbar's function is to time multiplex the data and transform a single DRAM access into four double-word transfers on the memory bus. Since the memory is always accessed in 64 byte blocks, two sequential accesses using page mode are made on the memory bus.

With 16 Mbit DRAM chips a SIMM has a capacity of 32 MB, and a memory group a capacity of 128 MB. The minimum memory configuration for the SPARCcenter 2000 is 256 MB. A fully populated backplane SPARCcenter 2000 system with 10 System Boards provides a maximum main memory size of 5 GB.

NVRAM

The SPARCcenter 2000 also supports a battery-backed non-volative RAM (NVRAM). The main purpose of using non-volatile memory is to accelerate synchronous disk writes. Write accesses to disk can take place in two phases. First the data is copied into non volatile memory and the write is acknowledged. Later, in a second phase the actual write to the disk takes place when a scheduling algorithm determines it is time to do so. The data stored in the NVRAM has the same property as data stored on disk drives, it can survive system crashes and power failures.

Synchronous disk writes can be done much faster because accessing the NVRAM is much faster than accessing current technology disks. In the SPARCcenter 2000, the NVRAM bandwidth is comparable to the regular main memory bandwidth.

Applications like NFS and DBMS, where data integrity is crucial, can easily take advantage of NVRAM and provide a better response time.

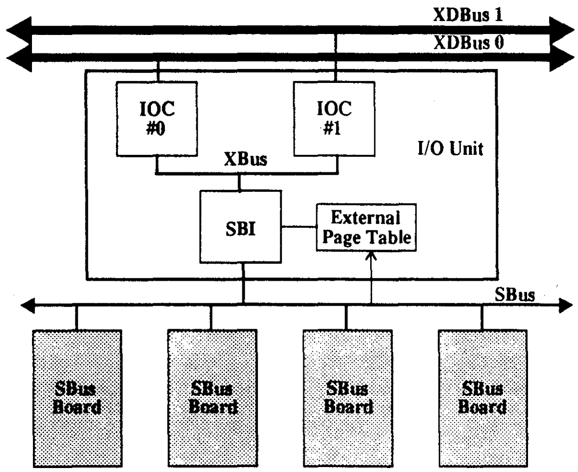
The NVRAM is implemented with non-volatile SIMMs (NVSIMM). The NVSIMM is composed of SRAM chips, Crossbar ASICs and a small lithium coin battery. Each NVSIMM has a capacity of 1 MB and the minimum configuration is two group of 4 NVSIMM, i.e. 8 MB.

I/O Unit

Overview

The SPARCcenter 2000 System Board supports a complete SBus which is used as I/O bus. Each SBus has four slots and is clocked at 20 MHz. All peripheral devices are connected to the SBus.

The I/O Unit provides the bridge between the SBus and the XDBus complex. The I/O Unit is composed of two I/O Cache chip (IOC), an SBus Interface chip and an External Page Table (XPT). The IOCs and the SBI are interconnected with an XBus. The following figure depicts the architecture of the I/O Unit.



I/O Unit

I/O Model

The I/O Unit provides three different I/O models:

- Programmed I/O where the processor directly reads and writes the I/O devices.

- Consistent DVMA I/O. DVMA stands for Direct Virtual Memory Access. In this mode, the data is moved directly between the SBus and the XDBus complex and the SBus address is translated into a physical XDBus address by the XPT. This mode is called consistent because the data is moved into the "Shared Memory Image" in the I/O Cache. In this mode, there can be only one pending transaction between an SBus board and the memory system.

- Stream Mode DVMA. As for the previous mode, SBus addresses are translated by the XPT but the data is not moved directly between the SBus and the memory system. Instead it go through pairs of buffers. These double buffers are not part of the shared memory image and are not kept consistent until they are flushed or invalidated by software.

Implementation

The IOC contains a small fully associative write-back cache which is kept consistent. Data is read from this cache or written into this cache when I/O transfers are done in consistent DVMA mode. The IOC also provides simultaneous I/O accesses to XDBus for each SBus slot. Even if an SBus board has an XDBus transaction pending, the other SBus boards can still access the XDBus.The IOC provides the interface between XDBus and the local XBus.

The SBI contains the read and write buffers used in Stream Mode DVMA. Each SBus slot has its own pair of double buffers and they are managed under software. The Stream Mode is the most efficient of the two DVMA modes. In stream mode, each SBus can sustain 50 MB/s when using 64-byte bursts and 30 MB/s when using 16-byte bursts The peak bandwidth is 80 MB/s. The transfer mode is selected on a slot basis.

The External Page Table implements a single-level page table through a set of SRAM chips. It can map up to 64 MB of DVMA address space. Each entry maps a 4 KB page. The XPT is controlled by the SBI and is maintained consistent by the kernel.

The SPARCcenter 2000 supports the Revision B.0 of SBus. All SBus transfer sizes (2,4, 8, 16, 32 and 64 bytes) are supported in both DVMA I/O modes. All SPARC addressable quantities are supported in programmed I/O mode. Each slot supports the full 28-bit address space. The SBus clock is independent of the system clock. An asynchronous boundary is implemented inside the SBI. An important feature of the SPARCcenter 2000 implementation of SBus is the parity extension support for data integrity. Parity can be enabled on a slot basis, so that devices which are not supporting parity can still be used.

Peripherals

The principal advantage of using a standard bus is the wealth of available peripheral boards. This is specially true for SBus since a large number of systems use it as an I/O bus and there is a large market for independent board manufacturers.

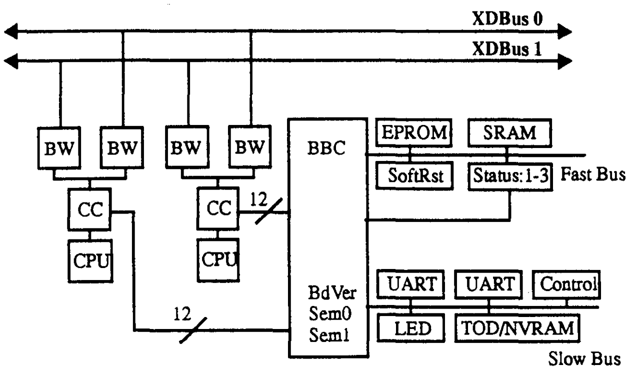
The following common interfaces are available: FDDI, Token Ring, HSI (4 synchronous lines), DSBE (Differential SCSI with buffered Ethernet controller), ISDN, GX frame buffer.

BootBus

Each SPARCcenter 2000 System Board also supports an 8-bit local bus called the BootBus. This bus is shared by the two Processor Units and is used to access system support devices. The BootBus is controlled by the BootBus Controller chip (BBC). The Cache Controllers are connected to the BBC through a 12 signals interface.

There are two types of devices connected to the BootBus: fast and slow devices. The SuperSPARC processors can access the fast devices simultaneously. However, the slow bus can only be accessed by one processor at a time. The following figure illustrates the connection between the Processor Units and the BootBus devices.

The devices connected to the fast bus are: 512 KB of



BootBus

EPROM, 16 KB of SRAM, three Status registers, two Semaphore registers, the System Software Reset register and the System Board version register. The EPROM contains the boot code and the SRAM is used as a scratch pad and for the stack.

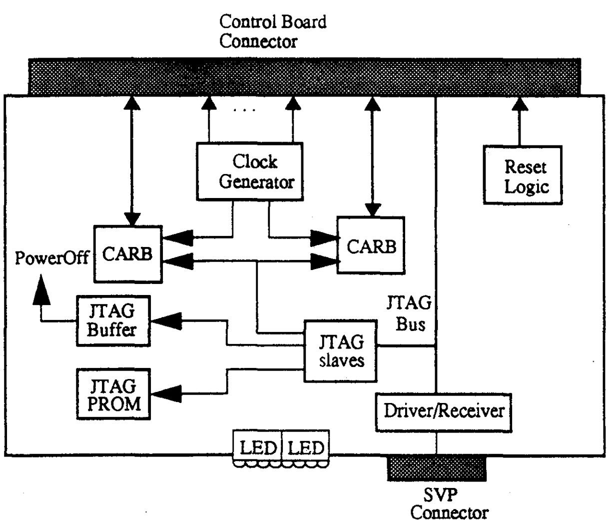
The devices located on the slow bus are: the LED diagnostic register, a Control register, a UART which provides two RS232 ports, a UART for a keyboard/mouse interface, the JTAG Master Interface register and a Time-Of-Day/Non Volatile RAM chip.

The slow devices are shared using one of the semaphore register.

Control Board

The Control Board provides the System Clock Generation, Central Arbitration, Power-on reset generation and the JTAG port for a Service Processor connection. It also has some LEDs to indicate the status of power and some system signals.

The figure below details the major functional units of the Control Board.



Control Board

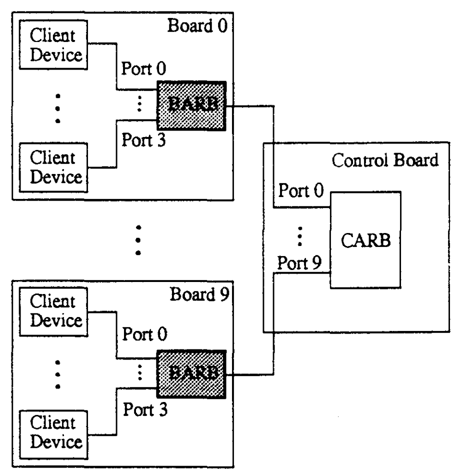
The Control Board supports the clock generation logic. The 40 MHz system clock is generated as a PECL signal. All clock traces on the distribution path have the same length and an equal number of loads to minimize the skew. The clock received by the Central Arbiters (CARB) has the same clock path as the XDBus client chips on the System Board.

The reset logic generates a general system reset signal which is forwarded to all System Boards. A reset is generated when either the reset switch on the front panel is activated, a power-on condition is detected, the optional Service Processor requests a system reset, a fatal error is detected in the system or one of the processor requests a reset by setting a bit in one of the BootBus register.

The Control Board also includes a JTAG PROM which contains the system ID and the Ethernet address.

Arbitration

The SPARCcenter 2000 uses a two level arbitration scheme to grant access to the XDBus. Because the two XDBuses operate in parallel, the arbitration logic is duplicated for each of them. The arbitration is implemented with two types of ASIC: the Board Arbiter (BARB) and the Central Arbiter (CARB). Arbitration requests generated by XDBus client devices are collected by BARBs and forwarded to the CARB. The CARB then selects a client device as XDBus master by issuing a grant signal to that client's BARB. This selection is made according to specific requirements of priority and fairness. Finally the BARB forwards the grant signal to the device. The architecture of this hierarchical arbitration scheme is depicted below.



Two Level Arbitration Architecture

The BARB and the CARB also participate in the data consistency protocol by merging the "Shared" and "Owner" signals issued by the BWs when they respectively detect that a block is present in the cache and modified.

The arbitration algorithm uses multiple level of priorities to provide an implicit flow control mechanism. It also supports an explicit flow control mechanism which is activated when one of the XDBus client detects that one of its incoming queue is becoming congested.

JTAG

JTAG (acronym for Joint Test Action Group) is a boundary-scan test standard adopted by IEEE [6]. In the SPARCcenter 2000, JTAG is implemented on the System Board, Control Board, backplane and all the ASICs. There is a JTAG control port accessible from each processor on the BootBus. An optional Service Processor may also be connected to the Control Board. In this case, the System Board JTAG ports are overridden.

JTAG provides a serial path to shift data and commands from the JTAG control port to a particular chip in the system. Rather than a single long serial path on the System Board, JTAG provides a way for branching to a particular scan path which contains a limited number of ASICs. There are six such paths per System Board which are called scan rings. It is also possible from a processor to scan devices on another System Board through the backplane JTAGs scan bus.

Within each ASIC there are also multiple scan paths. Dedicated component ID, status and control registers as well as all the flip-flops in the chip can be accessed through JTAG. This is a very important feature of the SPARCcenter 2000 which has multiple applications.

During self-test and debugging it is possible to read and write all registers inside a chip through full scan. This allows testing of all the logic in the ASICs by using Automatic Test Pattern Generation. In all the SPARCcenter 2000 ASICs the test coverage is more than 95%.

A second important use of JTAG is to test board connectivity. A data pattern is scanned in the output register of one of the chips and then the contents of the input register of another chip, connected to the former one, is read and checked.

The SPARCcenter 2000 ASICs also provides Shadow scan chains which can be read without perturbing the chip's behavior. This may be used to monitor hardware statistics and check for errors while the system is running.

JTAG is also used to initialize and configure the SPARCcenter 2000 after a power-on reset. By reading the identifier of the ASICs, the current system physical configuration can be easily determined. The configuration software then loads in various control registers basic information necessary to uniquely identify each device in the system.

Automatic System Configuration

The SPARCcenter 2000 uses an extensive set of diagnostics to determine which devices are fully functional. These diagnostics are part of the Power-On Self Test (POST) firmware which is stored in the BootBus EPROM. POST runs before the kernel is booted and configures automatically the system. POST also runs before each reboot and when recovering from fatal errors.

POST runs in several phases. First each processor tests itself and then the rest of the Processor Unit (CC and BWs) is also tested using JTAG. At that time the System Board are in loopback mode and the access to the backplane segment of the XDBuses is disabled. The other devices are also tested while the System Board is in loopback mode. The processors then elect one of them as master processor through JTAG. Next, all XDBus interface are checked and the master processor configures the system, including mapping the memory banks.

In case of failure there are multiple alternatives. A series of heuristics are used to decide which configuration to be used. There is often more than one way to reconfigure the system and it is generally a tradeoff between performance and functionality. This is discussed in the next section.

Reliability, Availability and Serviceability

The SPARCcenter 2000 incorporates many features that improve the reliability, availability and serviceability of the system.

Reliability

By using a very high level of integration and a conservative design methodology the probability of errors is greatly reduced. The extensive diagnostics performed by POST also ensure that only fully functional devices are enabled so that errors cannot occur because of failing components.

The SPARCcenter 2000 can also tolerate and recover from some errors like memory errors. The SIMM implementation guarantees that a Memory Unit will continue to operate even in case of the permanent failure of a DRAM chip. The failing memory group can then be deconfigured.

All data paths are parity protected in the SPARCcenter 2000 to ensure data integrity. XDBus, XBus, SBus, the arbitration signals and the SuperSPARC processor bus are parity protected. The external cache is also parity protected. There is also extensive checking of the XDBus transactions consistency by all the client devices.

The SPARCcenter 2000 is equipped with a number of environmental sensors to protect the system from hazardous conditions. A sensor checks the temperature and another one detects a failure from the fan. Abnormal AC or DC conditions are also detected.

Availability

The SPARCcenter 2000 is not a fault tolerant system in the sense of being capable of "non-stop" operation. However, it has extensive capabilities to detect and identify errors and the system can be reconfigured automatically without human intervention.

When a fatal error is detected while the system is running, it is logged and the system automatically resets. POST is invoked and tests all the hardware resources to determine if the error is permanent or transient. POST will then make the judgment on the resulting configuration.

In case of an XDBus failure, the system can be rebooted with a single bus without losing any functionality but the overall performance will be less as half the memory bandwidth is lost.

In the event of a processor failure, the failed Processor Unit is isolated from the rest of the system. This does not affect functionality as long as there are other processors in the system.

If an MQH fails, the Memory Unit is disabled. When a DRAM chip fails the corresponding memory group can be disabled and the Memory Unit remains functional if the second group is also populated.

Failure in the I/O subsystem are more difficult to handle as in general it is important to avoid loss of connectivity to I/O resources. If an IOC fails, POST will reconfigure the system with a single XDBus if no alternate path exists to critical I/O devices.

Serviceability

The SPARCcenter has numerous features designed to enhance the serviceability of the system.

There is an extensive failure/error logging and reporting logic in all the ASICs.

There are no jumpers for configuration, no slot dependency in the backplane and all connectors are keyed which makes installation less prone to human errors.

There is a very small number of FRUs which contributes to a very low maintenance cost.

The front panel, the System Board and the Control Board all have status/error lights.

Solaris

The SPARCcenter 2000 is compatible with all other SPARC platforms and runs the Solaris 2.X software environment. Solaris offers the largest Unix application base in the industry.

The latest version of Solaris 2.X offers several new features to improve performance and makes system management and security easy and cost-effective.

Solaris 2.X has full symmetric multiprocessing and multithreading capabilities to improve the performance of both commercial and technical applications.

It features installation tools to configure and upgrade systems over the network.

System administration is made easy through: automatic backup over the network, graphical tools to set-up new accounts and print servers, a tool to configure client systems and an icon based tool to install third-party software.

All these new features make the SPARCcenter 2000/Solaris combination the ideal rightsizing solution.

Conclusion

With 20 SuperSPARC processor at 40 MHz, the SPARCcenter 2000 delivers 2.19 GIPS and 269 MFLOPS[1]. In an 8 way configuration, the system delivers 8,047 SPECrate_int92 and 10,600 SPECrate_fp92[2].

The entry price with two SuperSPARC processors is below $100,000 which gives an unprecedented price performance ratio for this type of system.

The combination of large scale multiprocessing, extensive main memory and I/O capacity, reliability and availability and a very attractive price/performance make the SPARCcenter 2000 unique.

The multidimensional scalability and the ability to accommodate future processor upgrades will protect customers' investment and make the SPARCcenter 2000 the computing platform of the 90s.

References

[1] "XDBus: A High-Performance, Consistent, Packet-Switched Bus". Pradeep Sindhu, Jean-Marc Frailong, Jean-Gastinel, Michel Cekleov, Leo Yuan, Bill Gunning and Don Curry. Proceedings COMPCON 1993.

[2] "The Next-generation SPARC Multiprocessing System Architecture". Jean-Marc Frailong, Michel Cekleov, Pradeep Sindhu, Jean gastinel, Mike Splain, Jeff Price and Ashok Singhal. Proceedings COMPCON 1993.

[3] "A Three-Million-Transistor Microprocessor". Fuad Abu-Nofal, Rick Avra, Kanti Bhabuthmal, Rom Bhamidipaty, Greg Blanck, Andy Charnas, Peter DelVecchio, Joe Grass, Joel Grinberg, Norm Hayes, George Haber, Jim Hunt, Govind Kizhepat, Adam Malamy, Al Marston, Kaushal Mehta, Sunil Nanda, Hoa Van Nguyen, Rajiv Patel, Andy Ray, Jim Reaves, Alan Rogers, Stefan Rusu, Tom Shay, Irwan Sidharta, Terry Tham, Peter Tong, Richard Trauben, Anthony Wong, David Yee, Naeem Maan, Don Steiss, Lynn Youngs. Proceedings IEEE International Solid-State Circuits Conference 1992.

[4] "A CMOS Low-Voltage Swing Transmission-Line Receiver". Bill Gunning, Leo Yuan, Trung Nguyen, Tony Wong, Proceedimgs IEEE International Solid-State Circuits Conference, 1992.

[5] "A BiCMOS 50 MHz Cache Controller for a SuperScalar Microprocessor". Balakrishna Joshi, R. K. Anand, Curt Berg, Jorge Cruz-Rios, Ashok Krishnamurthi, Nyles Nettleton, Sophie Nguyen, Jim Reaves, John Reed, Alan Rogers, Stefan Rusu, Chuck Tucker, Chung Wang, Ming Wong, David Yee, Jung-Herng Chang. Proceedings IEEE International Solid-State Circuits Conference 1992.

[6] "IEEE Standard Test Access Port and Boundary-Scn Architecture, P1149.1". IEEE Computer Society, Test technology technical Committee, January 1989.

1. With the Apogee 0.82 compiler and the SUN Fortran compiler 3.0α respectively

2. With the SUN C and Fortran compilers 3.0α

Reprinted with permission from Spring 1993 issue of "Benchmark."

THE YEARS OF THE DRAGON

A new technology platform that was jointly developed by Xerox Corporation and Sun Microsystems will give new speed, power and efficiency to many types of computing devices. The successful collaboration that made this possible is described here.

When Xerox Corporation and Sun Microsystems teamed up in 1988 to co-develop a new technology, they did so without the fanfare that often accompanies announcements of high-tech partnerships. Yet members of the so called "Dragon" team achieved the ultimate goal of a Silicon Valley project: They came up with a new way to harness more computing power, more quickly and less expensively. The technology platform they designed condenses the electronics of an entire board onto a single chip, harnesses the power of multiprocessors and offers unprecedented expansion capability in three dimensions--memory, processing and input/output (see Dragon Technology section at end of this article).

The first products to use the Dragon technology platform were announced by Sun last November. The occasion gave those close to the project pause for reflection on what Wayne Rosing, president of Sun Labs, calls "one of the most successful collaborations I've seen."

Adds John Seely Brown, director of the Xerox Palo Alto Research Center (PARC), "What we have here is a quiet, very successful strategic technological alliance that was a total win-win situation."

The project began in 1986, when a team of researchers in PARC's Computer Science Lab (CSL) began working on a shared-memory multiprocessor architecture that could be used to drive a range of Xerox imaging products. Ron Rider, vice president, Corporate Architecture, recalls the project: "We wanted to build a very powerful computer system that depended on multiple processors cooperating with each other and working in parallel. We knew it was a technology that was going to be vital to Xerox in the future."

But developing the technology would require a major investment of tools and people. "You'd have to amortize that cost over the Xerox product family alone, and it basically wasn't affordable," says Brown.

Leveraging the World

Besides, Xerox senior managers had realized they couldn't do it all by themselves, Brown adds. "We wanted to connect to the world, to leverage the world. We decided to do those things that we could do best and try to build partnerships for those things others could do as well as or better than we could."

Adds Rider, "We wanted to take advantage of the technology both so that Xerox could get some financial benefit and so that we could have access to using it."

In 1987, Rosing, then Sun's vice president of Advanced Development, went to PARC "fishing" for interesting technologies.

"I had heard about this great bus technology for multiprocessors," he recalls. "They gave me a briefing on it and it was evident to me that PARC had solved a problem that Sun hadn't even grokked the magnitude of--how to build a multiprocessor with a small number of pieces of silicon that could be replicated cheaply. They were dozens of person-years ahead of where we were at the time."

Rosing made an on-the-spot decision to use the technology as the basis of Sun's multiprocessor product line. The next several months were spent hammering out an agreement that would distribute the cost of development between the two companies.

"It was to be a three-year development project to build a multiprocessor server that would be the highest performance server in the world in its class," says Rider.

The technology alliance was a departure for both partners, according to Jean Gastinel, head of the PARC Dragon team. "Until now, Sun had never imported technology from the outside--they had done everything themselves. And PARC had a reputation for doing research and not doing products," he explains.

Another departure was the way the team was organized. Beginning in September, 1988, 10 members of the PARC Dragon team began reporting for work at Sun Microsystems instead of at Xerox.

A Virtual Corporation

"It was kind of rough for the first six months, but once people got used to the idea, they worked as a single team," says Rider. "They weren't Xerox, they weren't Sun--they were working on this project."

"You almost have to create the concept of a virtual corporation for a project like this," adds Rosing. "Ron and I never had to sit down privately and resolve any issues. Whatever face-saving happened, happened within the team because the people felt the project was more important than their individual issues."

Rosing attributes the teams cohesiveness to "the secret of staffing," which begins, he says, with a recognition that the NIH (not invented here) syndrome exists. "The desire for an engineer to build and create new things is almost untamable, and for engineers to be willing to subordinate their egos and their creativity to the common good is not a natural thing," he says.

"On the management level, too, there are egos involved. It's important not to inflame the situation through incorrect staffing.

"For example," Rosing continues, "the PARC people were the 'theologians' of the bus technology. If you had two engineers of fundamentally different theological persuasions on the same team, there will be a religious war--so I made sure we didn't have any Sun bus theologians on the team."

Another important ingredient for success, according to Rosing, is common objectives. "I think the biggest single reason why so many partnerships fail is that the two companies don't have an aligned and shared goal," he says.

Gastinel points to the egalitarian nature of the team as another important difference. "We had co-ownership of the results, so our management structure was that of equals," he explains. He also cites the availability of technology and equipment as another factor that helped the team succeed. "Sun was incredibly good about giving us whatever we needed," he says.

The Computer Farm

"Whatever they needed" turned out to be a roomful of processors and workstations, nicknamed the computer farm. The engineers jury-rigged the 30 machines together to create 1,000 MIPS (million instructions per second) of computing power--the amount they need to run an unprecedented volume of simulations. Their innovative use of simulations enabled them to design a set of chips that "ran on first silicon," working perfectly the first time they were manufactured--an unusual feat in Silicon Valley.

"They showed that by using massive amounts of simulation it is possible to simulate degrees and levels of complexity previously unthinkable," says Brown.

Gastinel and his team pioneered the use of distributed simulation, inventing a new methodology for design and testing. "We simulated the design in a variety of different ways," he says. "For example, we would test a chip in isolation, connected to a subsystem, with all subsystems connected together, with a processor, without a processor and so on.

"In a lot of projects, bugs are considered bad, but we took the opposite attitude. We wanted to find all the bugs we could during the simulation phase. For six months, we had about 20 people trying to break the machine."

Gastinel estimates that the use of simulations to design and validate the system saved the team a year--the time that might have been required to redesign and reproduce faulty chips.

He and his team, now back at Xerox, will duplicate their intensive use of simulations in their next project, which will require 10 times the number of MIPS that were needed to design Dragon. Fortunately, instead of a 300-machine computer farm, it will take only five Dragon-based servers to generate the 10,000 MIPS.

"What has come out of this project is a new way of doing design and a new set of tools," says Rider. "Dragon has provided the keys that will enable us to build things that are much smaller and much larger than we ever thought this technology would allow."

Rosing points to products in the Sun pipeline that "just dropped out" of the Dragon effort. "There has been a lot of mileage out of this investment, and I feel that both parties will continue to get benefits from it."

One way they will benefit is by jointly licensing the technology to any company that wants to use it. Both partners will get royalties for know-how and the use of the Dragon chips. Xerox will additionally receive royalties for intellectual property--from licensees as well as from Sun.

At Sun, the company that has "always done everything themselves," paying royalties may not sit well with everyone. Rosing defends the arrangement: "The royalties will never add up to what it would have cost us to do it on our own. If you only consider the profits that would have been lost if the machine had come out a year later-- there's no comparison.

-Debra Feinstein

DRAGON TECHNOLOGY: A CLOSER LOOK

The Sun SPARCcenter 2000 is the first implementation of Dragon technology to reach the market. Announced in November 1992, the 2000 is an open, multiprocessor system powerful enough to run mainframe-class applications. Thanks to its superb cost/performance ratio, Sun believes the 2000 is poised to displace mini- and mainframe computers in many settings. An entry-level system costs under $100,000--less than one-tenth the price of a typical mainframe.

Yet the 2000 has tremendous growth potential. A fully expanded version could interface with 1,000 or more users and provide backbone computing support for an entire enterprise.

Sun describes its system as the first affordable, high-performance Unix server specifically designed for client-server computing. Client-server refers to the relationship between a host computer and desktop workstations. In mainframe environments, the host-desktop relationship is master-slave.

Client-server computing assumes greater host-workstation cooperation in executing programs and sharing and moving data. This strategy harnesses desktop power but puts heavy input-output (I/O) demand on the host.

The speed of the 2000--a 20-processor model can handle 2 billion instructions per second--makes it ideal for demanding applications like the kind of computer simulations used in electronics and automotive design. Sun has made a concerted effort to help database vendors tune their packages to take advantage of the 2000's multiple processors and of an architecture that uses high-speed "cache"memory to good advantage.

How does Dragon technology contribute to the 2000's performance and cost advantages? What building blocks does Dragon provide for future products?

BENCHMARK asked Dragon team member Jean Gastinel of Xerox PARC and David Yen of Sun Microsystems to explain.

Would you describe the system architecture? What makes it unique?

The architecture features a Unix operating system and follows industry network standards to ensure connectivity in an open systems environment. The Dragon architecture dovetails with Xerox' embrace of open systems and its corporate strategy of using architectural building blocks.

Modularity is the system's hallmark. Plug-in modules make it easy to expand. The strategy also allows for backup of key components to improve reliability.

A shared-memory, multiprocessor machine poses many design challenges. Dragon architecture relies on a state-of-the-art cache coherence scheme to solve one of them. To speed operation, each processor has its own reserve (or cache) of "fast" memory to store data.

However, when multiple processors are teamed, problems can arise if the system stores different data in various caches. To make sure the right data is selected for processing, Dragon architecture relies on a unique and extremely flexible algorithm. This lets the system operate at high speed and eliminates the possibility that data can be corrupted.

What's so special about the Dragon bus?

A bus is a highway for moving data. Bus limitations create traffic bottlenecks--especially in multiprocessor systems. Ordinary buses don't have the bandwidth to move huge amounts of data quickly and smoothly. The Dragon bus solves this problem with physical and logical protocol advances.

On the physical side, the news is GTL--Gunning Transceiver Logic, named for team member Bill Gunning. Most system voltage levels require at least a 3-volt swing to transmit data. With GTL, that drops to 0.8 volts. GTL payoffs include faster transmission and less noise and interference.

This low-power characteristic lets designers build bus interfaces into high-density, application-specific integrated chips (ASICs) instead of creating more costly and bulky external alternatives.

One of the most complicated aspects of computer design relates to low-voltage signals. The GTL approach permits lines to be inexpensively terminated in a manner that lets chip-to-chip data transmission occur at the speed of light.

A very efficient packet-switching protocol is used to send bursts of data to memory devices on the bus. Packet-switching can move two to four times more data than traditional circuit-switching. The Dragon bus' transaction sets can support a large number of multiprocessors, too. This overall design expands bandwidth to 640 megabytes per second and provides tremendous I/O capacity to support processors and peripherals. The 2000 system's twin buses are configured to handle up to 20 processors without any performance loss.

How much is power consumption reduced? Why is that important?

An entry-level 2000 system uses less than half the power of a comparable minicomputer. Less power means less heat, and heat dissipation has long been a system design headache. When semiconductors overheat, they don't work properly and performance degrades. Less power consumption reduces the need for cooling hardware and boosts reliability--and of course, less energy is consumed.

What technology factors translate into cost savings?

The system's modular architecture delivers major cost savings. The system is partitioned to require a small number of highly integrated building blocks. The system costs less because there are fewer components. The compact design extends to the housing, too, since only a small cooling system and power supply are needed.

One integration secret is the use of high-density CMOS gate arrays to encapsulate all control logic. If you were to compare a 2000 board with boards from a competitive machine, you would see that the 2000 board has consolidated its logic into relatively large chunks of silicon, while the competitive boards are dotted with smaller packages of random logic.

Does the Dragon technology give the 2000 a reliability edge?

Definitely. In the movie, "Terminator II," Arnold Schwarzenegger repeatedly fails to kill his robotic rival because the villain can restructure himself and repair any damage it sustains. Shotgun shells may slow it, but they don't kill it.

The 2000 has similar resilience. While it can't repair itself, it can reconfigure itself to bypass faulty parts. There are very few single points of failure on this machine. Redundancy is built into crucial areas like the backplane bus, the processor, and the I/O and memory subsystems. Even if there are multiple hardware problems, the system can reconfigure itself without human intervention. A failure prompts an E-mail message and activates a yellow LED to alert operators that service is needed and should be scheduled.

You say the system is scalable in three dimensions. What does that mean?

System requirements tend to grow in three areas--number of processors, amount of available memory and I/O capacity. While added memory requirements are relatively easy to address, many systems bump up against processor or I/O expansion ceilings.

Dragon technology makes it easy to scale systems in all three dimensions. For instance, the 2000 system can be expanded to include up to 20 processors and up to 10 I/O buses. What's more, the expanded unit will deliver incremental performance on a par with the added resources. In some expanded systems, performance deteriorates when resources are forced to compete with each other. The 2000's memory-sharing architecture eliminates this problem. This scalability permits a true "pay as you grow" approach to computer power acquisition. Users are no longer forced to project and budget today for the system they will need three years from now.

What future Dragon implementations in Xerox products are in store?

Currently, 10 boards are needed to hold the electronics for a Xerox DocuTech Publishing System. Dragon technology could reduce that board count to one. The Potential reduction in cost and improvement in performance has put DocuTech high on the Xerox list for Dragon implementation.

Dragon technology, in fact, is destined to influence the design of many new products. As an example, Dragon's ability to connect multiple processors and access wider bandwidths could push development of a number of interesting new image processing applications.

-Linda Lovely