What is Futurebus + ?

National Semiconductor
Application Note 1036
Paul Borrill
January 1996

ABSTRACT

Futurebus + is a specification for a scalable (32/64/128 or 256-bit wide) bus architecture. Arbitration is provided by a fully distributed, one or two pass, parallel contention arbiter with allocation rules to suit the needs of both real-time (priority based), and fairness (equal opportunity access based) configurations. Two transmission methods are provided: (1) a technology-independent, compelled protocol, supporting broadcast, broadcall and transfer intervention (the minimum requirement for all Futurebus + Systems), and (2) a configurable transfer-rate source-synchronized protocol supporting only block transfers and broadcast for the maximum performance systems. Futurebus + takes its name from its goal of being capable of the highest possible transfer rate consistent with the technology available at the time modules are designed, while ensuring compatibility with all other modules designed to this standard before *and after.* The plus sign (+) refers to the extensible nature of the specification, and the hooks provided to allow further evolution to meet unanticipated needs of specific application architectures. This paper describes the history, structure and applications of the Futurebus + architecture.

HISTORICAL PERSPECTIVE

Futurebus + is a revised and substantially extended version of the original IEEE 896.1—1987 Futurebus standard, where the basic protocols and facilities of the new Futurebus + were developed.

From 1983 to 1987 the IEEE P896 Futurebus Working Group, provided a forum for leading experts to develop innovative technology and protocols for a scalable performance multiprocessor system bus. The most recent Futurebus + efforts, from 1988 onwards, represented a commercial consolidation of all the basic Futurebus philosophies into a realizable and practical standard as an industry consensus developed between the major organizations who became interested in developing products based on Futurebus + .

During early 1988, the VME International Trade Association (VITA) saw the need to develop a strategy which would lead to the definition of a Next Generation Architecture Bus Standard, to follow the widely successful IEEE 1014: VMEBus standard. They developed a set of requirements which included openness, performance goals, and system facilities and flexibility which would not hinder systems using this bus for many generations of computer systems. In December 1988, VITA formally announced its intention to base its ''VME Futurebus + Extended Architecture'' (VFEA), on a revised and extended IEEE 896.1—1987 Standard, in conjunction with the IEEE Futurebus + Working Group.

Around the same time, the US Navy's Next Generation Computer Resources Program (NGCR) chartered a Backplane Working Group to develop a set of requirements for use in future Mission Critical Computers. A principal requirement was that the chosen standard be likely to become a major commercial success also, in order that the US Navy could capitalize upon the systems expertise developed from R & D Funding provided by the commercial and industrial sectors. All known existing and proposed bus architectures were evaluated against a set of criteria.

Futurebus scored substantially higher than the other contenders in almost all categories. After significant internal discussion at the highest levels of the DOD, the Pentagon announced in December 1988, their selection of the IEEE 896.1—1987 Futurebus as the basis for all future US Navy mission critical computers.

A third major influence on the specification came from the IEEE P1496 Rugged Bus Working Group, who had, in 1986, decided to develop a ruggedized specification for a backplane bus. Rather than develop an entirely new bus from scratch, the Rugged Bus Working Group chose to use the IEEE P896.1—1987 Standard as a base from which to develop their application needs. In November 1988, at a Joint Rugged Bus/Futurebus Working Group meeting, the two groups agreed to merge their efforts for a single bus standard, to be called Futurebus + . The Rugged Bus Working Group brought significant additional skills to the joint activity: real-time, fault-tolerance, maintainability and environmental conditions, in addition to the dedication and hard work of their members who became an integrated part of the team which defined and reviewed the latest Futurebus + specification.

An additional strong influence on the specification came from the Multibus Manufacturer's group who, in February 1989, announced their intention to pair IEEE 1296 (Multibus II) with the developing Futurebus + Specifications, into a common ''Futurebus + Extended Architecture'' (FBX). From that point onwards, member companies of the Multibus Manufacturers Group provided technical support for the Working Group activities, and contributed greatly to the cooperative spirit among what were previously competing factions in the bus industry, to develop a *single* cache coherent bus architecture for the industry, which would now compete as a truly open standard against the myriad of proprietary and semi-proprietary architectures which had plagued the industry in previous generations of computers.

MAJOR FEATURES

Futurebus + represents a major paradigm shift for the bus industry. It is the first comprehensive bus architecture designed *a-priori* to be an *OPEN standard (An interface specification for which there are* ***no*** *restrictions for who may use it),* and which was explicitly designed to support multiple generations of computer technology.

Reprinted with permission of IEEE.

Futurebus+ will benefit end users immeasurably, by allowing vendors to build systems which not only have a much greater degree of compatibility, but which are ''gracefully upgradable'', preserving investments already made in the boards, enclosures, power systems, peripherals and other hardware (and software) as new improvements in processor architecture, levels of integration, and clock speed, are implemented.

Futurebus+ derives its name from its lack of built-in obsolescence parameters, and its upwardly compatible architecture and protocol extensions. Futurebus+ represents a significant departure from the philosophy of other standard or proprietary buses. The most important objectives of the Futurebus+ project were: (1) to create a bus standard that would provide a *significant* step forward in the facilities and performance available to the designers of future multiprocessor systems, and (2) to provide a stable platform upon which manufacturers can base several generations of computer systems.

In order to meet this objective, the following requirements were set:

- Architecture, processor and technology independent (attributes of a truely *OPEN* Standard).
- A basic asynchronous (compelled) transfer protocol for simple, higher reliability operation with a handshake flow control over each word of data transferred.
- An optional source-synchronized (packet) protocol for the highest possible performance, with flow control over each block of data transferred.
- No technology-based upper limit to the performance of the bus in both modes (i.e., limited only by the laws of physics—not by the technology).
- Fully distributed parallel and arbitration protocols to provide the minimum possible number of single point failure mechanisms.
- Parity protection on all lines, and feedback checking where possible (e.g., modules may write to themselves to facilitate self-testing).
- Multi-level mechanisms for locking of modules, *and* the avoidance of deadlock or livelock.
- Circuit switched and split transaction protocols. Plus support for memory controller commands to implement remote lock and other SIMD-like operations.
- Support of real-time mission critical computations. (Multiple priority levels with which to arbitrate, and the consistent treatment of priorities throughout the arbitration, message and DMA protocols. Plus support for the distributed clock synchronization protocol defined in IEEE P896.3.)
- Support for fault-tolerant and high availability systems (Dual bus operation, fault detection and isolation mechanisms and live insertion and withdrawal of modules).
- Direct support for snoopy-cache based shared-memory systems, with recursive protocols to support single or unlimited size systems consisting of arbitrarily connected buses.
- Recognition and support of strong and weak sequential consistency (time order assumptions).
- Compatible message transport definition supporting a number of message passing protocols.

TECHNOLOGY INDEPENDENCE

A unique attribute of the design of the Futurebus+ protocols is their technology independence: achieved through basing the protocols on fundamental protocol and physics principles and optimizing them for maximum communication efficiency (and hence throughput) rather than for a particular generation or type of processor. Timing and handshake protocols are thereby governed by ''law of nature'' types of constraints rather than limitations of current and projected technology such as device propagation delays, and capture windows.

The benefits of technology independence are reflected in the principle of no technology-based upper limit to the performance of the bus. The configuration or transaction capability modes guarantee interoperability when two boards of a different speed, or different generation, communicate on the same bus segment; thus providing the Futurebus+ with an unprecidented ability to support multiple generations of computers, well into the 21st century.

This principle of design longevity also provides a self-adapting performance span which allows graceful upgrades in computer equipment. This leads to less customer annoyance, greater confidence in their supplier and the equipment they sell. Manufacturers are also likely to find significant advantages in their manufacturing capability, the learning curve of their design engineers, and a strong strategic advantage in credibility to their customers.

Fundamental to the compatibility of slow, fast, old or new modules attached to the same Futurebus+ segment, is the notion of *broadcast*. Each connection phase (address cycle) is broadcast in a way which guarantees that each module on the same bus segment is able to decide whether or not to participate in the forthcoming data transfer phase. This turns out to be a basic requirement of a snooping cache coherence scheme, which allows multiple caches to search their directories to see whether or not they ought to be interested in the transaction. Even then, the protocols provide for the performance enhancements possible when a master knows that other modules will not be interested, by indicating whether other caches should even bother to search their directories or not (for example, when a cache block is replaced and ownership is transferred back to the main memory, or when the transactions are using the message passing mode).

The P896.1 specification may be implemented using *any* logic family (e.g., TTL, BTL, CMOS, ECL, GaAs) providing the physical implementation is constrained such that it meets the Futurebus+ signaling requirements (incident wave switching, and relative skews between information and synchronization lines).

Futurebus+ protocols may be used at *any* level in the system hierarchy. Device to device, board to board, or system to system. These protocols are particularly powerful when used at two or more levels in a hierarchical bus system.

ARCHITECTURE INDEPENDENCE

Futurebus+ is architecture independent, providing a flexible and elegant general purpose solution to cache consistency, within which other cache protocols will operate compatibly, while at the same time providing an elegant unification with a message passing transport protocol.

There are many exceptionally powerful features implicit within the Futurebus+ specifications which may not be obvious on a first reading of the specification. The Working Group chose to provide a flexible set of tools within which an implementation can be crafted to suit a wide spectrum of applications. Many of the protocol features may be used not only for the immediately obvious purpose, but are capable of being utilized for other purposes, many of which were recognized by the Working Group, but which were omitted from the descriptions in order not to over-complicate them.

Implicit in the philosophy of the Futurebus+ specification, is the concept that large systems may be built from smaller subsystems, and those small subsystems may in turn be built from yet smaller subsystems. Examples of which include the choice of split or interlocked transactions and a fully recursive, truly hierarchical cacheing paradigm. This is in anticipation of future generations of systems, which will inevitably utilize many levels of caches and internal/external buses. The Futurebus+ protocols will work well at *any* or *all* levels in this hierarchy; since for each generation of computer systems, each bus is absorbed into the packaging technology one level lower than the previous generation: traffic flowing on backplane buses today will flow on the on-board buses of tomorrow; and on chip-level buses soon after that!

Futurebus+ has been designed with large scale integration of the bus interface in mind. A severe restriction on the number of signals has allowed the bus to steer clear of being intrinsically expensive. Hence, while initial implementations of the Futurebus+ may require multiple interface devices, the learning curve, and volume, will allow these to be further integrated and for the market forces to rapidly bring down the cost of the interface to become consistent with even the lowest cost workstations, industrial and personal computers.

PARALLEL PROTOCOL

The principal objective behind the design of the parallel protocol, was to achieve the highest possible transfer rate consistent with the implementation technology and manufacturing constraints at the time the modules were designed, allowing both backwards and forwards compatibility. This ''technology-independence'' aspect of the Futurebus+ protocols provides the following advantages:

- A long design longevity for the bus, consistent with the attributes of a widely implemented standard for commercial, industrial and military requirements. Futurebus+ has been designed in anticipation of its continued use well into the 21st century.
- Graceful upgrade of system components, rather than the more usual ''throw it all away and start again'' scenario that occurs with traditional computers when higher clock-rate processors become available.
- Flexible tradeoff between cost and performance at any one point in time. Boards may be designed with exotic technologies to achieve the highest possible performance, or with more cost-effective technologies to achieve excellent cost/performance tradeoffs, and both can operate within the same system (providing they conform to the same profile).

The Futurebus+ parallel protocols support *both* connected and split transactions. Simplistically speaking, connected transactions are cheaper, easier to implement, but are slowed by the access time of the memory. Split transactions are more complex, expensive, but provide higher multi-stream system bandwidth since the bus need not be idle during the memory access time. Connected transfers provide lower latency for access to memory (hence single-stream performance is optimized), split transactions provide more concurrency in the protocols, hence they optimize multi-stream performance. Both are needed in a system architecture. Connected transfers provide a cost-effective performance for I/O operations, while split transactions provide the maximum performance required by MPU/Cache-memory operation.

Current RISC or CISC Microprocessors do not easily support a command structure for memory controllers to perform remote lock operations. Futurebus+ provides a simple workaround for this: use the split transactions for everything with the exception of lockvariables, and automatically revert to using connected transactions to perform RMW type operations. This allows the performance enhancements of split transactions to be obtained without having to redesign the processor, or utilize ugly software workarounds which require existing code to be modified and adversely affect the performance of the system.

SYNCHRONIZATION DOMAINS

The most frequently misunderstood aspect of bus design and the one which invokes the most religious fervor among protagonists, is the synchronization protocol.

One of the basic arguments for the asynchronous (source synchronized) protocols in Futurebus+ is that it allows the synchronization domain of the sender to extend along the bus segment, presenting only one synchronization interface between the bus and the receiver. This contrasts with centrally synchronized buses which require three separate synchronization domains (sender, bus, receiver). Synchronization interfaces mean performance is lost due to resynchronization delays and MTBF is reduced due to metastability. Since Futurebus+ has only one synchronization interface, it will naturally yield a higher performance than any centrally synchronized bus.

SCALABILITY AND PERFORMANCE

Scalability of cost and performance were early requirements in the design of the Futurebus+ protocols. Although Futurebus+ is basically a 64-bit address architecture, employing a standard 64-bit multiplexed address/data highway, a 32-bit Address/Data *subset* , and 128-bit or 256-bit data *superset* is also provided. Perhaps more importantly, the performance scales over time with a fixed width highway, allowing for example, systems to be built with 20 ns transfers in 1991 and 10 ns transfers by 1995.

Note: While 10 ns per transfer is often regarded as an upper limit due to the physics of the bussed transmission line environment, this is a conservative estimate based on measurements with a full-length ''backplane'' implementation of Futurebus+ protocols and an electrical environment, supporting upwards of 20 boards. Systems with fewer modules, shorter stubs, and shorter backplanes or no backplanes at all (i.e., on-board implementations of the Futurebus+ protocols), will be able to significantly exceed this limit.

Multiplying these numbers by the data highway width options (32, 64, 128, and 256 bits), provides a protocol throughput (with an appropriate electrical environment), of approximately 100 Mbytes/second at the low-end for systems using the 32-bit subset in 1990, while systems using the 256-bit superset in 1995 will peak at 3.2 GBytes/second—a dramatic demonstration of the benefits of a scalable design.

There are two principal reasons why Futurebus+ is able to attain this level of performance, while other buses cannot: (1) All Futurebus+ protocols are *source synchronized,* meaning that the synchronization signal is always emitted by the same module which emits the information signals, thereby eliminating spatial skews, and (2) substantial effort has been put into the understanding the signaling environment to guarantee that receivers are triggered by the *incident* wave from the driver as it passes by along the bus lines.

Note: Spatial Skews result when there is a space (and hence time) difference between the transmitting and receiving modules, and the passage of data between them is synchronized by a signal transmitted by a 3rd party. For example, a bus with a central clock source must account for an additional uncertainty in the arrival of data at a receiver equal to the space (time) difference between the sender and receiver of the data, independently of the mechanism used to distribute the central clock.

Another dimension of scalability for the Futurebus, which lends itself also to fault-tolerant applications, is the use of dual or multiple parallel Futurebus's. The locking mechanisms, cache coherence mechanisms, and the general architectural facilities are designed such that they can be readily applied to such applications. Moreover, it may well be possible that implementations of dual 64-bit or 128-bit buses (as opposed to single 128-bit or 256-bit buses) are likely to lead to better multi-stream system throughput (with a fixed number of bytes per block), due to the smaller fraction of the overall transaction time used as overhead in the narrower width implementations of the bus.

A NOTE ON PERFORMANCE

Since Futurebus+ is a technology-independent specification, there is very little in the protocol specification which would indicate what their actual performance would be in a specific implementation, without building and testing such a system.

Although there is, in principle, no upper limit to the performance of the Futurebus, there are some basic issues regarding the achievable performance which are not widely understood. The actual performance of any bus in any particular system will be a function of the following three critical elements:

- *The theoretical cleanliness of the protocols.* e.g., do the protocols carry any unnecessary overhead; do they carry any unnecessary constraints of synchronism to a central clock, or too many round-trip delays in the handshakes; do they incorporate fixed technology constraints such as arbitrary set-up and hold times?
- *The architecture of the system in which the bus is used.* The most theoretically perfect bus protocol will grind to a snail's pace in a poorly architected system. Very often, it is the memory system bandwidth which is being measured rather than the intrinsic protocol bandwidth of the bus. Systems which perform random transfers can never sustain as high a transfer rate over the main highway as systems specifically designed to take advantage of the intrinsic efficiencies of the memories and bus protocols, such as block transfers.
- *The implementation.* Only if full advantage is taken of available semiconductor technologies, will the protocols perform at their full potential. Since the Futurebus+ is a truly *open* standard, its benefits are available to any and all who choose to adopt the bus. Competitiveness, and added-value, will now focus on the quality and thoroughness of the implementations. Since Futurebus+ protocols provide significant room for learning-curve effects in the industry, one can expect a rapid evolution in the performance of systems which use Futurebus+, through a steady but definite improvement in the implementation.

Data from various simulation results and reports on anticipated semiconductor performance have been used to predict the expected peak performance figures for the Futurebus+ over time as shown in *Figure 1*. The packet protocols are shown from a 1991 baseline because of the anticipated wide availability of devices which can implement this mode during that time-frame. Conservative assumptions were used in the development of this model; it is quite possible for some commercial organizations to be well ahead of these performance curves when they introduce their products.

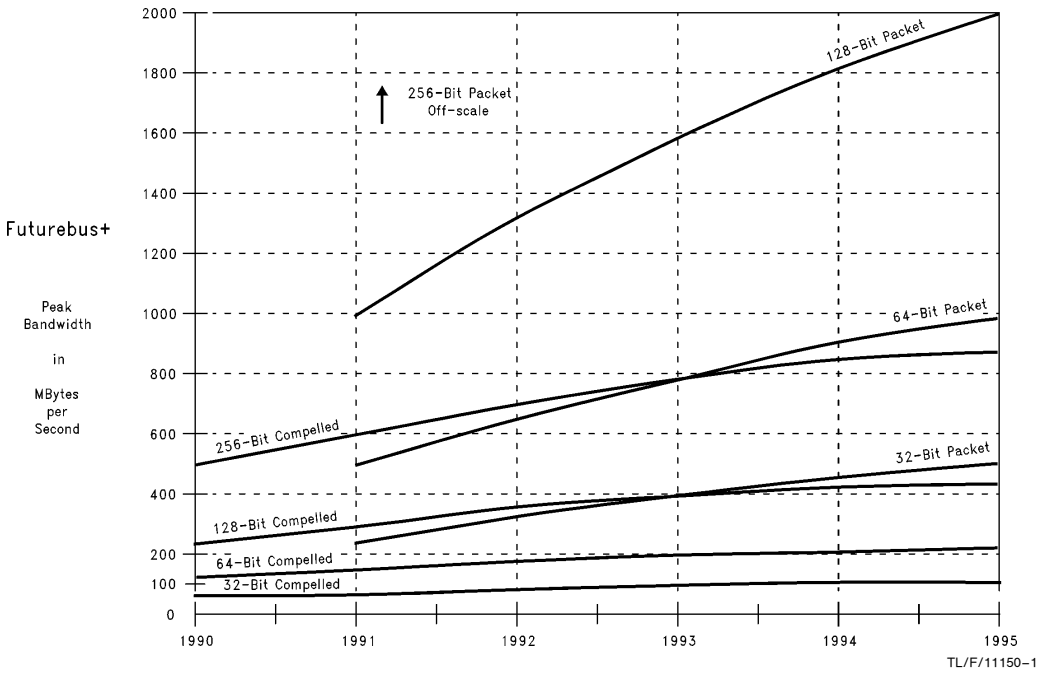


FIGURE 1. Futurebus+ Protocol Bandwidth (Peak)

ARBITRATION

Futurebus+ uses a highly evolved version of the Parallel Contention Arbiter. This mechanism arbitrates among active contenders through a combinatorial logic network distributed among all possible contenders using a set of simple bussed lines connecting them. By applying a unique arbitration vector to this logic, one contender only (with the highest arbitration vector) will be uniquely selected as the winner. The Futurebus+ implementation additionally distributes the responsibility for synchronization of this mechanism among all the contenders, such that no central logic whatsoever is required on the bus segment.

Although not as familiar as a central request/grant scheme, the parallel contention arbiter used in Futurebus+ has a number of significant advantages:

- The current master can observe any requests (and their priority). This allows the current master to make a reasonable decision as to whether it should give up the bus immediately (high priority request), at the next ''convenient'' breakpoint (low priority request), or to continue until the end of a potentially long block transfer or series of transactions (no requests pending).
- Multiple priority levels. Allows real-time systems to deterministically allocate bus bandwidth to those processors running the most critical tasks in situations of high system load, i.e., it allows one to determine if a process is schedulable (guaranteed to meet its deadlines even under conditions of high system load), and improves the probability that a process is schedulable within a system with fixed available bandwidth.
- A master-elect can be pre-empted by a higher priority contender which arrives after the arbitration has taken place, but before the current master has relinquished the bus. This allows the higher priority contender a significantly shorter average latency to access the bus.
- Arbitration Messages (or events) can be broadcast on the arbitration bus without the need to disturb the traffic currently underway on the parallel bus. This mechanism can be used to implement, for example, interrupts, targeted events and parallel processing rendezvous at synchronization points before proceeding; all without the need for dedicated lines on the bus for those miscellaneous system control functions.

Implemented in a purely text-book fashion, and with all the parameters of the arbitration network set to their maximums, this arbitration mechanism will perform approximately similar to a daisy chained mechanism, but without the inherent disadvantages of such a scheme. Significantly faster implementations may be designed, which still fully conform, by noting that the arbitration parameters (arbitration vector, arbitration contest settling time, Wired-OR Glitch filters, etc.) can be traded off against the number of active masters in the system, the total number of modules attached to any one bus segment, and the length of that bus segment: all parameters which can be ascertained during the system initialization phase. Also, in order to take advantage of the performance of higher speed modules (i.e., modules with faster logic), two arbitration ''speeds'' are dynamically selectable. This ensures that the compatibility is maintained among old and new boards attached to any bus segment.

A unique aspect about the Futurebus+ arbitration scheme, is the optional ''idle-bus'' arbitration method, used when there is only one requesting master. This method, using the Address/Data lines to signal a request when the bus is idle, provides the fastest possible latency for a module to access a memory subsystem. In the event that two or more masters are detected during this request phase, the scheme automatically reverts to using the parallel contention arbiter method, already started concurrently with the fast idle bus method. The result is a significantly faster average latency for a module to access the memory subsystem. Used in combination with the connected bus protocols. This can result in a significant reduction in the miss penalty of modern RISC processors, as compared to all other bus designs.

SYSTEM ARCHITECTURE SUPPORT

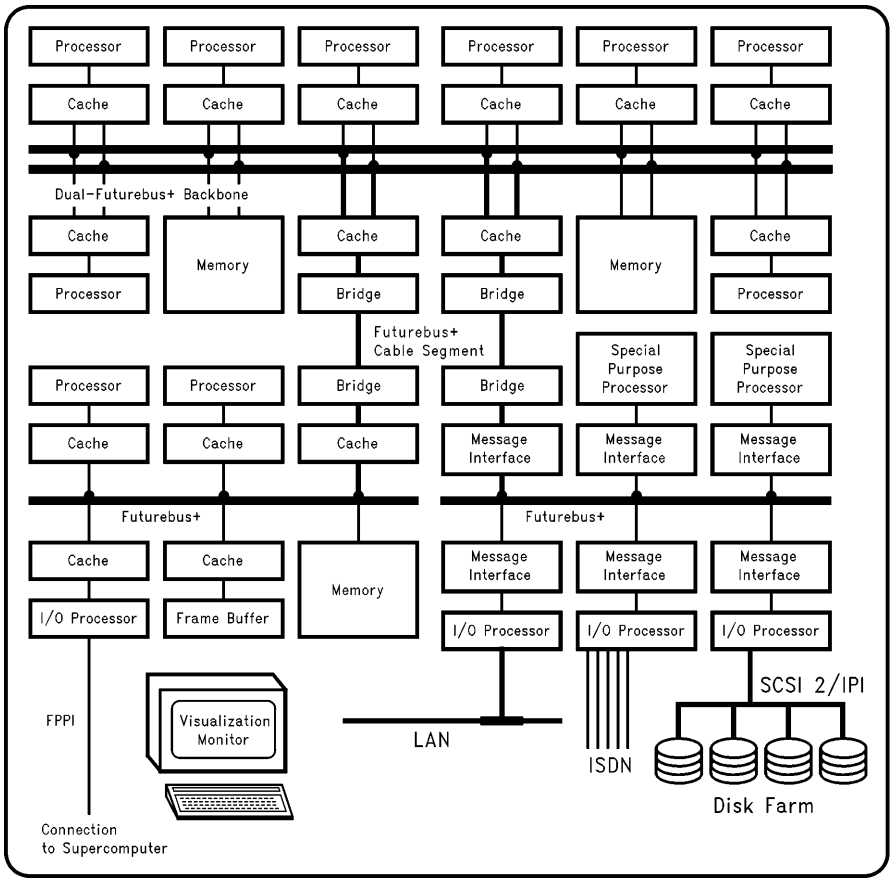
The Futurebus+ specification has been written to provide support for a wide variety of system architectures. Within a single bus segment, both functionally distributed (message passing) and shared memory (cache coherent) systems may coexist. For multi-crate systems, Futurebus+ has a number of companion standards offering a bridging service to other bus standards. This wide range of options will allow a smooth transition from existing buses to Futurebus+ and beyond to other advanced interconnection architectures such as the Scalable Coherent Interface (SCI): a ring-based interconnection for large scale parallel processing system implementations. It also allows the needs of secure data and tightly-coupled parallel processing regimes to be balanced.

A message passing architecture is, in its purest form, a write-only system. In such a system, a module requiring access to data sends a request message to the module owning the data; the responding module will reply at some time later by writing the data to the requestor. It is easy to see that access restrictions may be readily implemented in such a system without cutting down on the available bus bandwidth; the responder simply checks the authorization of the requestor before replying. What is not immediately obvious is that such a system can make very efficient use of the bus. First, only write transactions are being used, and write transactions are inherently more efficient: requiring only the presentation of the data and a response, rather than a request, an access time for the data and a response, as is required for read transactions. Second, messages use fixed size blocks, allowing burst transactions to amortize the cost of transaction setup over the block. Message passing systems also have the advantage of being simple to design and analyze since nothing is shared and all transactions are inherently split.

Futurebus+ describes an optional message passing architecture in Chapter 9. The P896.1 model uses a default message frame of 64 bytes (note the compatibility with the buffer sizes needed by the cache coherence protocols), a broadcast mailbox with a message filter and point to point request and response mailboxes. Messages are transmitted without the use of any special bus level protocols. Only two of the Futurebus+ transaction types are used: Write-unlocked and write-no-acknowledge. This makes Futurebus+ message passing easy to implement and easy to add to an existing interface design.

Shared memory architectures represent a different approach to system data flow. In this model, all data needed for use by more than one resource within the system is held in a global, shared memory subsystem, accessible to all bus masters within the system. This method of sharing data would be costly in terms of bus bandwidth and access latency if all modules had to compete for access to the same physical memory module(s), so caching and split transactions are provided to enable much higher performances to be obtained by eliminating unnecessary contention. A shared memory model is extremely useful when multiple processors are operating on the same data and is the preferred architecture for use in tightly-coupled multiprocessing. The use of constructs such as caches and split transactions requires that the hardware implement a well-thought out series of protocols to ensure that all processors always operate on valid data only.

The shared memory and message passing models are not mutually exclusive within the Futurebus+ environment: they can readily co-exist, allowing the system integrator to meet seemingly orthogonal requirements.



TL/F/11150–2

FIGURE 2. Typical Futurebus+ Application

SEQUENTIAL CONSISTENCY

Computer Architecture has been evolving for many years, and many problems which now seem familiar, were far from obvious when they were initially recognized. Several years ago, the Working Group solved some fundamental physical and conceptual problems concerning the physics of the backplane environment, and the design of the bus protocols to provide general mechanisms to support cache coherency. Recognizing that there are likely to be many more problems which have not yet been either widely recognized, or well articulated, the Futurebus+ Working Group has incorporated many (mostly hidden) hooks into the specifications to allow its evolution, in a compatible way, to meet both anticipated and unforeseen future requirements.

For example, the Futurebus+ protocols have been carefully designed to be fully consistent with the physical model of the universe implied by special relativity. The model dispels the myth of the concept of simultaneity, because all laws of physics, including electrodynamics, optics, mechanics, *and data location and movement* are invariant with respect to all coordinate systems, and that spatial distribution is equivalent to time (in this case, latency). It is the lack of recognition of this principle in computer science which gave rise to the conceptual difficulties called sequential consistency.

Sequential consistency was first recognized in early 1988 by the Futurebus+ Working Group when trying to reconcile the view of the universe from the different perspectives of the programmer and hardware engineer. It is the assumption, often made without explict reference by programmers, that the order of events observed by any one device is the same as any other device. Unfortunately, when the transmission time of an event or protocol packet (due to the finite speed of light) is significant relative to the period between events occurring in the system, this assumption is no longer valid, since it violates the principle of relatively (that there is no absolute frame of reference, and the order of events depends on both the space and time coordinates of the observer relative to the rest of the system).

Futurebus+ provides support for dynamically choosing between strong and weak sequential consistency. One way this is supported is through the split transaction protocol, by either posting the writes and continuing without waiting for a response (weak consistency), or by requiring the processor/cache to wait for all responses from other blocks before continuing (strong consistency).

The effect of strong sequential consistency is to sequentialize several variable updates through one place in space-time (the current owner). Since in the MOESI model there is by definition only *one* current owner (although there may be several surrogate owners in the bridges), the effect is to create a single point in space through which the synchronization can take place.

It was important to provide support for strong consistency since many experts believe that sequential ordering of events must be guaranteed in order to ensure the correct operation of certain programs (this is currently an outstanding research question in the academic world). However, strong sequential consistency significantly reduces the amount of concurrency allowed in the system, and can therefore have a highly detrimental effect on multiprocessor system performance. Weak sequential consistency is provided for those *new* programs wishing to take maximum advantage of the concurrency available in a system, but which are cognizant of the programming constraints needed to ensure correct operation when strong sequential consistency is not guaranteed.

Mechanisms to support synchronization barriers for the rendezvous of multiple threads in a task executed in parallel are also provided by the broadcast and broadcall facilities in the parallel protocol, and by the arbitration messages provided by the bus allocation scheme.

BRIDGES

The Futurebus+ Working Group has been firmly committed to the concept of bridges for many years, and believe that bridges are a requirement for any new bus architecture to succeed. There is presently an enormous investment made in existing bus products. This investment is two dimensional, representing a great depth in quantity of systems sold and breadth in terms of available functionality within a given bus architecture. Since a new bus will not be able to achieve this penetration for some time, and furthermore does not wish to require previous investment to be written-off, bridges provide a pragmatic solution to this problem.

A bridge represents a concept more than a physical implementation. The idea is to allow transactions begun on one bus to be completed on another with minimal impact on the software and hardware of the standard modules on either bus. This means more than simply converting protocols. A bridge must provide a nearly seamless means of communication between what are basically incompatible architectures. This includes methodologies for defining how interrupts are handled on a bus which does not recognize the single-processor concept of interrupts, for example.

For Futurebus+, this has been addressed for a number of existing and future bus standards. Futurebus+ provides its own specifications within the basic protocols to bridge to itself. Bridges have also been defined between Futurebus+ and VMEBus (IEEE P1014.2), Multibus II (IEEE P1296.2), and the Scalable Coherent Interface (IEEE P1596). Those will allow Futurebus+ to be used with existing or future hardware and software from any of those bus architectures. A smooth upgrade path from lower performance buses to higher performance buses is assured, as is the usefulness of the newer bus standard even without a broad product line in its early days.

TESTABILITY AND MAINTAINABILITY

Futurebus+ has been designed from the outset to make it straightforward to build testable implementations. Facilities to support error detection go hand in hand with hooks to stimulate those mechanisms to deliberately cause an error, so that the mechanism may be tested.

Much discussion took place on the need for a boundary scan standard (such as IEEE P1149) to be included within the Futurebus+ signal set. Since the bandwidth requirements for such a bus are so low, the relative cost of bus signal lines is not negligible, it was decided to support these facilities through the use of a boundary scan controller on the modules themselves (logic is cheap inside LSI devices), and to stimulate and control this controller through commands and telemetry transmitted through the IEEE P1394 High-Serial Bus mechanism provided on two of the pins of the Futurebus+ signal set.

In addition, IEEE P896.3 may specify a test access port as a requirement on the access panel of modules built to a profile which includes requirements for Maintainability and Testability. See IEEE P896.3 for further details.

REAL-TIME APPLICATIONS

Futurebus+ is also designed to support real-time applications where the correctness of computation depends upon not only the results of the computation but also the time at which outputs are generated. Examples of real-time applications include air-traffic control, avionics, process control and mission critical computations. The measures of merit in a real-time system include:

- Fast response to urgent events and accurate timing information.
- High degree of schedulability. Schedulability is the degree of resource utilization at or below which the deadlines of tasks can be ensured. It can be thought of as ''real-time throughput'', i.e., the number of timely transactions per second.
- Stability under transient overload. When the system is overloaded by events and it is impossible to meet all the deadlines, we must still guarantee the deadlines of selected critical tasks.

The distributed clock synchronization protocol defined in P896.3 provides users with accurate and reliable timing information. The prioritized arbitration message mechanism provides the basis of fast response to urgent events. The multiple priority levels in the arbitration protocol and the consistent handling of priority among arbitration, message and DMA protocols provides the user with a consistent and powerful scheduling tool. As a result, users can employ analytical scheduling algorithms, for example, the rate monotonic algorithm, to ensure a high degree of schedulability and stability. In short, Futurebus+ architecture facilitates the development of real-time systems, whose timing behavior can be analyzed and predicated *a-priori*.

FAULT-TOLERANT APPLICATIONS

Futurebus+ is also designed to be suitable for use in high integrity systems: the address/data, command and arbitration lines are all protected by a parity bit per byte, there are *no* central elements to adversely affect the survivability of the system, and modules can be inserted or removed while the system is operating to facilitate high availability applications.

Included in the arbitration protocols and initialization facilities are the mechanisms to support the live insertion and removal of modules. Although Futurebus+ includes the necessary hooks. Full specification of how they are used in an application system can be found in P896.3.

PROFILES AND INTEROPERABILITY

Older bus standards often suffered from interoperability problems. Boards from different manufacturers would often not communicate with one another. Since this defeats the purpose of open standard architectures, the Futurebus+ specifications have been laid out to maximize interoperability. This has been done in three ways: First, the specifications have been written in a form which is unambiguous (using formal equations where possible rather than purely English statements). Second, an adequate amount of background description has been provided, to encourage the implementor to understand and share in the instinctive elegance of the protocols with the original designers of the specifications. Third, by providing a hierarchical system of constraints to allow a tradeoff between application specific (e.g., low cost versus high performance), and application generality without compromising interoperability. This latter point has been implemented through the concept of profiles.

The Futurebus+ family contains a number of specifications which do not individually constitute a bus specification, but provide a rich set of tools with which to build optimal implementations for a wide spectrum of applications. When called up within profiles, however, these specifications, and the options within them, are much more tightly constrained. This is to provide the implementor with clear guidelines, which if followed, will maximize interoperability with any other implementation which follows the same guidelines.

As an example, the specifications which may be treated as a family for use within a typical Futurebus+ profile are: P896.1 Futurebus+ Logical Layer Specifications, P896.2 Futurebus+ Physical Layer and Profiles, P1194.1 Backplane Transceiver Logic, P1212 CSR Architecture, P1394 (High speed serial bus), and P896.3: Futurebus+ Systems Configuration Guide.

A profile consists of a mapping of which sections of which specifications and which options within those sections are appropriate for use together within an implementation. Profiles sanctioned by the Working Group will be contained within the P896.2 document, and higher level profiles suitable for specialized system application environments will be included within P896.3. This explicit discussion of what is required and what is not within a given profile addresses the interoperability issues brought about by arbitrary assignment of optionality by manufacturers. Profiles also allow the buyer of Futurebus+ based products to know exactly what features come with each product. If a manufacturer follows the requirements laid out within a profile, that product is much more likely to be interoperable with the products of any other manufacturer meeting that profile.

ACKNOWLEDGEMENTS

Futurebus+ would never have been possible without the visionary outlook of several key individuals: Lyman Hevle, Executive Director of VITA, Shlomo PriTal of Motorola, LCDR Harrison Beasley, of the US Navy, Paul Cook, Chair of the P1496 Rugged Bus Working Group, and Vice Chair of the Futurebus+ Working Group. John Hyde, Chair of the P1296.2 Committee and representative of the Multibus Manufacturers Group, and Wayne Fischer of Force, and Chair of the P1014.2 Working Group.

REFERENCES

1. IEEE P896.1: Futurebus+ Draft 8.2, Published by the IEEE Computer Society, 1730 Massachusetts Avenue, N.W., Washington, D.C. 20036-1903.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

http://www.national.com

National Semiconductor Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 180-530 85 85
English Tel: +49 (0) 180-532 78 32
Français Tel: +49 (0) 180-532 93 58
Italiano Tel: +49 (0) 180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2308
Fax: 81-043-299-2408