# Futurebus+ Profiles Support Diverse Applications

## Profiles Build on Menu of Protocols and Capabilities

**By John Theus**
**TheUs Group, Sherwood, OR**

*Last issue, we began our Futurebus+ series with an overview of the goals and the structure of the standards. This article presents the major tools that make up the Futurebus+ logical layer—starting at the bottom of the protocol stack with synchronization. This is followed by an introduction to command and control protocols, and then on to the higher layer topics such as split transactions and cache coherency. An application profile chooses from among these tools and those of its own invention to specify a solution for a specific application area. In the final part next issue, we will explore the cache coherency functions and other advanced features in more detail.*

### Synchronization

Futurebus+ uses asynchronous protocols for all information transfer except during packet mode, which uses a source-synchronous protocol. The asynchronous protocols use a handshake system that is fully compelled—each and every action requires a reaction before the protocol can proceed (see Figure 1). The protocols are also designed so that no timing specifications are necessary. (A sender guarantees 0 or positive set-up time between the information and strobe signals.) Therefore, a module accounts for its own delays, and it has no need to know about delays in the other modules.

This approach to synchronization was picked because it is independent of the state of technology. As technology improves, devices become smaller and faster, and speed-of-light delays at both the micro (chip) and macro (system) levels become shorter, and the protocols run faster. This technology independence is very important in backplane-based systems due to their design longevity. Implementations built with last year's technology continue to work properly in systems built with next year's technology, and systems can be upgraded gracefully.

The protocols use both edges of the handshake signals to mark events. For example, the rising edge of the data strobe marks valid data and the next falling edge marks the next valid data. Since the handshake signals require the same signaling bandwidth as the information signals, the protocols make optimum use of the available electrical environment.

### Allocation and Arbitration

Bus allocation is the process of assigning an order to the active bus requests. Bus arbitration is the process of unambiguously issuing a single bus grant. Examples of bus allocation are first-come-first-served, fairness (each module is given equal access), and priority. Futurebus+ specifies two methods for doing bus arbitration and allows several algorithms for bus allocation.

Arbitration uses either a central system or a distributed system. The central arbiter requires a backplane with non-bused request and grant signals from each module's position to the arbiter's position. This arbiter is logically simple and has low propagation delays. This arbiter has been built using asynchronous logic, and it produces metastable-free outputs in under 10 ns. Each module is limited to two request signals, so real-time priority changes are difficult. The bus allocation policy is not specified since it is implemented in the central arbiter (module hardware doesn't care). An application profile would specify an appropriate allocation policy.

The distributed arbiter uses bused signals to imple-

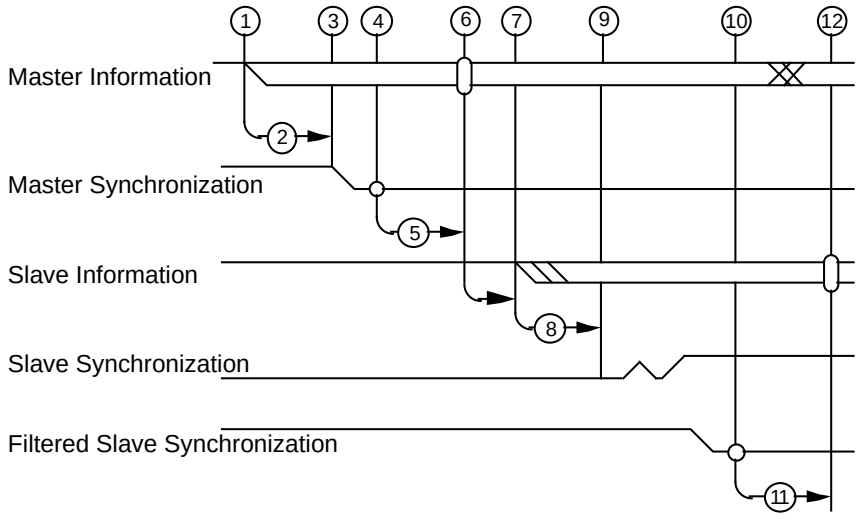


Figure 1. An example of the fully-compelled, step-by-step handshake process. Multiple modules drive the slave information signals and the slave synchronization signal. The disturbance after step 9 is called a wire-OR glitch which is an artifact of turning off one voltage-mode open-collector driver while one is still turned on. The filtered signal has had all pulse widths less than the round-trip bus propagation delay removed.

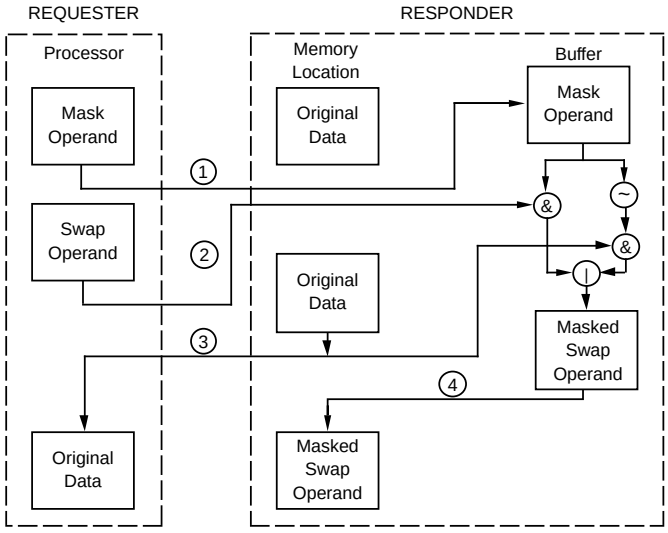


Figure 2. A master requests a mask-and-swap-lock command and transfers the operands in steps 1 and 2. The response occurs in step 3. The result of the operation is written to memory in step 4.

ment a distributed, asynchronous state machine and has no central facilities. This arbiter is logically complex and significantly slower (at least 10×) than the central arbiter. Its allocation mechanism provides fairness and up to 256 levels of priority, which can be changed in real-time. A byte-wide data path is used during the arbitration phase to choose a winner. This data path can also be used as a message facility that is completely independent of the main parallel data bus, so it is useful for fault recovery and for signaling events.

## Parallel Bus

The parallel data bus is controlled by an 8-bit (plus parity) command, an 8-bit status, and a 3-bit capability field—each of which occupies its own set of signals. The command field specifies address (32 or 64) and data (32,64,128, or 256) widths, data length (up to 64 words), and a transaction type command. The transaction type includes simple commands, such as read and write, plus more complex commands, such as cache line invalidate, read shared, modified response, and fetch-and-add locked.

Each independent status signal contains a wire-OR result of each individual module's status. These bits are used for signaling exceptions (errors and deadlock avoidance) and cache line shared and modified status.

The capability field is used to negotiate protocol options on a transaction-by-transaction basis. The participating modules decide whether to use a split or connected transaction (defined below), whether to use a compelled or packet data phase, and if packet is used, which one of two packet frequencies.

The data bus is a time-multiplexed path with byte parity that carries destination address, source address, data, and byte masks. A master transmits all of these information types, while slaves can only transmit data during a connected read transaction. The 32- or 64-bit destination address selects the first word to be accessed. The size of a word is defined dynamically in the command field. One or more bytes within a word can be accessed by using a partial-word access. When this transaction type is selected, a byte mask is transmitted following the destination address.

For split transactions, the source address is used for routing a response back to the requester. This address contains a 16-bit field that describes the requester's physical location, an 8-bit transaction priority field, and a 6-bit field that may be used for a user-defined tag.

Data are transmitted using either the compelled or packet transfer modes in blocks of length $2^n$ up to 64 words long. The compelled mode has an additional feature for transferring blocks of arbitrary length.

## Split Transactions

Futurebus+ supports both connected and split transactions. The selection is made on a transaction-by-transaction basis—usually based upon the address. A connected transaction is the traditional bus action of supplying an address and waiting on the bus for the result. When processors, buses, and memory systems were all well matched in performance, sitting on the bus waiting for a read request to be fulfilled made sense. With today's faster processors and buses, a great deal of system performance can be lost using this approach.

To better utilize the bus resources, a read or write operation can be divided into two separate bus transactions—a request and a response. A read request consists of a starting address (destination address), the length of the block to read, and an address to return the data (source address). When the module that accepted the request has compiled a response, it becomes a bus master and transmits a read response with the data and the source and destination addresses. Although write operations can be split—the response says the final destination received the request and the accompanying data—most applications only split read operations.

## Locks

A set of locking protocols is available for applications that need synchronization or semaphore instructions. These facilities are separate from the cache-based locks that require the lock variable to live in coherent address space, since many system architectures need semaphores in non-coherent space.

The lock protocols allow one or more bus transactions to be grouped together to form an atomic operation. A simple case is an exchange instruction that re-

sults in an atomic read and write set of bus transactions. The bus protocols guarantee that the read and write transactions are indivisible, and the protocols also signal the destination module enforce this as well. The Futurebus+ protocols for doing these connected locks allow any number of read or write transactions—to one or more slaves—to be locked together. All connected locks are released when the master retires from the bus.

In a system using split transactions, these connected locks do not work since the bus is not held by the master between requests and responses. A protocol called a lock command is used to move the lock operation from the bus onto the destination module. The exchange instruction used above would be translated into a mask-and-swap lock command. As shown in Figure 2, the requester would issue a write transaction that contains both mask and swap operands, along with an opcode for the command. The responder would atomically read and save the contents of the operand's address, and then execute the mask-and-swap operation and store the results at the operand's address. The responder would then become a bus master and transmit the saved original contents back to the requester. Opcodes are provided for mask-and-swap, compare-and-swap, fetch-and-add little-endian, and fetch-and-add big-endian.

## Cache Coherency

A hierarchical cache coherency protocol is provided that allows a system of interconnected modules—on one or more physical buses—to work within a single coherency domain. The logical model of this system has memory at the root, processors as leaves and bus bridges as branches. Physical implementations allow memory and processors to exist anywhere in the system. The coherency protocol implements a MESI subset of the MOESI model. (The work that led to these acronyms and the Futurebus+ coherency protocol will be discussed in part 3 of this series.)

## Message Passing

Many applications—heterogeneous, secure, and fault-tolerant systems to name a few—use message passing as their principal form of communication. Futurebus+ message passing is built on top of the compelled and packet transfer protocols. Messages can be addressed to specific modules, broadcast to all modules on a bus, or broadcast to all modules on all buses in a system.

Message passing is specified at two levels—a frame level that transports a single frame of data, and a message level that transports one or more frames that make up an entire message. The 64-byte frame contains a header and a body with data.

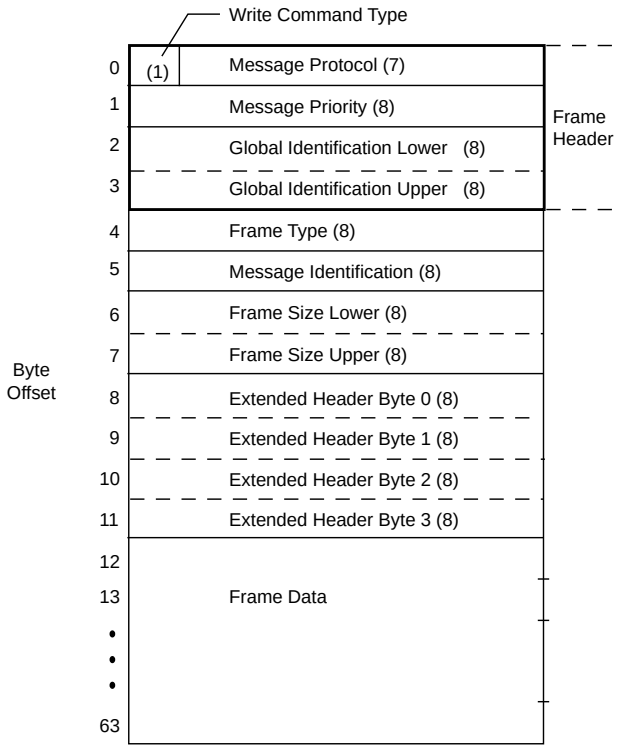


Figure 3. The structure of the Futurebus+ defined message level format is shown here as a string of bytes. The first four bytes are the frame level format that is common for all message level formats.

Of the 128 possible message-level formats, one is completely specified, and 64 formats are available for user definition. The specified message format further defines the header with eight additional bytes as shown in Figure 3. The information stored here and the protocols that use it specify a complete messaging system. Protocols are provided for single and multiple frame messages. In the latter case, a protocol exists that allows frames to arrive at the destination out-of-order.

## Live Insertion

Three levels of live insertion (inserting boards with system power on) are defined, which vary on their degree of transparency to bus operations and their difficulty of implementation. The first level is the simplest to implement, but it requires that operations on the bus be quiesced during insertion and withdrawal. The duration of suspended operation can be relatively short if physical position sensing hardware is used. The standard Futurebus+ protocols have all the hooks necessary to suspend and continue bus operations.

The second level allows insertion and withdrawal while bus operations continue, but some restrictions apply. Glitches may cause errors, but they must be detectable and correctable. Some types of protocols may be restricted—packet mode for example—in their operational characteristics. Transceivers, power control,

and connectors must all be designed to permit reliable bus operation during the transitory conditions.

The third level requires that the above-named components guarantee error-free bus operation during insertion and withdrawal. No restrictions are placed on the protocols while the module is being moved. The most interesting aspect of these procedures is in the technologies used by the transceivers to prevent glitches. This topic will be discussed in part 3.

## Fault Tolerance

The basic protocol set provides for fault detection by using byte parity in the areas with the highest probability of error. These are the data bus, the command bus and the distributed arbitration bus. No attempt was made with the basic protocols to detect all single-bit and single-point errors.

A separate set of protocols is being developed that will detect all single-bit and single-point errors across the entire protocol set. Several new signals are required to make this possible. For example, parity generation is not possible with the wire-OR status signals. When the master receives the status information, it will echo what it's received so that the status senders can check it.

## Control and Status Registers

One of the main goals for Futurebus+ was to eliminate the need for jumpers to configure a module. A standard set of registers is defined that performs three main tasks. The first are the capability registers—implemented in ROM—that describe the feature set of the module. All the various protocol options have corresponding register definitions. The control registers are used to turn features on and off and set operational parameters. The status registers report current or previous state in areas such as self-test results and error logs.

These registers are in the top one-sixteenth of the 32 bit address space. Each module has two 4K-byte areas for these control functions. A module's address compare logic uses five uniquely hardwired bits from the backplane—called its geographical address—to differentiate itself from the other modules in the backplane.

## Profiles

As described in part 1 (see µPR 5/27/92 p. 17), profiles are the mechanism for selecting tools for a specific application. Profiles are named using a single letter, initially assigned starting with A and incrementing from there. After the first couple of assignments, letters have been picked to match the name of the application.

The three profiles that are specified in 896.2 are A, general-purpose bus; B, I/O bus; and F (Fast), system bus. The A, B, and F profiles are almost the same except in their logical areas. All are meant for the commercial world, use a 12 SU (~265 mm height) by 300 mm board size; use a 2 mm connector with up to 504 pins, and use BTL transceivers. In hindsight, these three profiles probably should have been a single profile with proper subsets and supersets. (Profile F has a superset called F+.) Module designers can easily make subsets and supersets with these profiles, but the specifications don't make it obvious.

The A, B, and F profiles were completed first because they address the most mature applications. Profile A was driven by the VMEbus board-vendor community and is therefore similar to VMEbus in its lack of specificity. The board vendors wanted a profile that allowed them to build products under a single profile banner that can be used for several different applications. Profile A modules will physically intermate and communicate at some level, but it's up to the system integrator educated about the Futurebus+ protocols to know if any given set of modules can do useful work together.

Profile A modules must support 32-bit address and data paths and perform full-word read and write transactions using the connected and compelled protocols. All other facilities are optional, including the style of arbiter. This flexibility allows a very broad range of features to be implemented. Two likely configuration problems are incompatible arbiters and incompatible split-response handling.

Profiles B and F were driven by system designers with specific application requirements in mind. Modules built to either of these profiles are guaranteed to work within their respective applications. Both profiles require central arbitration, full- and partial-word access, and the handling of split transactions.

Profile B, in its role as an I/O bus, is designed to move blocks of data exclusively. No higher-level protocols (locks, cache coherency, or message passing) are allowed. Designs built to this specification are likely to be simpler than those for any other profile. Modules with 32-bit address and data are allowed, but most designs will be 64 bits wide. The only other permitted option is packet mode. All modules will work together running the baseline configuration. The options described here allow supersets to be implemented, all of which are controlled on a transaction-by-transaction basis.

Profile F—the fast system bus—builds on Profile B by adding the higher-level protocols. F has a specified superset called F+ that requires the implementation of packet mode. F also differs from B by specifying protocol parameters that insure good citizenship and allow the calculation of minimum performance levels. For example, 32-bit modules are allowed but are not encouraged

by limiting their bus usage. If a proposed 32-bit design needs more bandwidth than the specified amount, it must implement a 64-bit data path. Similarly, if a module requires more than the specified amount of time to satisfy a read request, then it is required to split the transaction. The majority of these specifications are simple maximum-allowed timing requirements that place a performance floor on the implementation.

The two profiles that are nearing completion are P896.5, Profile M (Military), and P896.6, Profile T (Telecom). Profile M will become both an IEEE and military standard, and it has been driven by the U.S. Navy's Space and Naval Warfare Systems Command. Profile M is intended to address the needs of most Navy mission-critical computing applications in ships, submarines, aircraft, large missile, torpedo, and shore facilities.

To address the large range of physical environments, Profile M has three classes which use a common logical layer. The convection-cooled commercial class uses the same mechanical packaging as Profiles A, B and F. The intermediate class uses a smaller conduction-cooled board in severe environments where the ultimate in packaging density is not required. The final class is the most demanding and is based on the Standard Electronic Module–Format E (SEM-E). Due to its expense, this class is usually limited to tactical aircraft.

The Profile M logical layer has all the tools of a system bus with support for locks, cache coherency, and message passing. In addition, this profile has provisions for the detection of all single-bit and single-point errors and for dual redundant buses. The IEEE P1394 SerialBus is also used as an additional communication path. Additional backplane signals are defined that are unique to this application, such as nuclear event detect (!).

Profile T has been driven by several of the U.S., European, and Japanese telecommunications giants. Their application requires the protocols of a system bus in an environment that features continuous operation and existing equipment packaging standards. Single-bit and single-point error detection, redundant modules and buses, and live insertion and withdrawal are all provided for in the specification.

Finally, two new profiles have recently become official IEEE projects. P896.7, Profile C (Cable) will specify mechanisms for interconnecting Futurebus+ backplanes. P896.8, Profile D (Desktop) will specify a high performance parallel bus for the desktop and mezzanine environments. New applications continue to come forward. Space-based systems look promising as the next profile project. Clearly, the application profile approach has allowed and will continue to allow Futurebus+ to grow with the times and technologies. ♦

## SMM Explained

*Continued from page 16*

All of these pins were no-connects, except for the IIBEN pin on the Am386DXLV, which is a $V_{CC}$ pin on Intel's 386DX. Note that although AMD offers their Am386DX in a PGA package, this packaging option is not available for the Am386DXLV.

C&T's Super386 is offered in four forms, the 38600DX and 38600SX which are pin-compatible with Intel's parts, and the 38605DX and 38605SX which have additional signals for cache control and SMM. The DX version uses a larger package than the original 386DX, so only the SX version (which uses the same package as an Intel 386SX) is shown in the table.

Unlike the other vendors, AMD has a separate ready line for cycles to the SMM space, and an input to enable a mode which allows I/O instruction trapping. The IIBEN pin is checked on every I/O cycle, and if it is asserted the processor will not pipeline the execution of instructions immediately following an I/O instruction. This guarantees that no instructions will be executed before the SMM interrupt is called, although it is possible that instruction prefetches will occur.

### Conclusion

Each vendor has implemented its SMM in a slightly different way, but all provide the basic functions needed to implement advanced power-management strategies. Intel's solution has the least flexibility (fixed addresses, fixed SMM width, and few SMM memory configuration options), but that is not surprising in a design which integrates the processor with the chip set logic. The 82360 could be replaced with proprietary logic, in which case the SMM space could be any size and width.

AMD's solution is notable for its inflexible SMM memory width and large processor save/restore context. Cyrix offers a simple, flexible, and efficient solution, which is supplemented by a separate suspend/resume mechanism that can be used with external clock stop logic to reduce power consumption. C&T's solution is the most sophisticated, with its on-chip interrupt and I/O trapping logic, wealth of conditions which can invoke SMM, and instruction backup facility.

The SMM in the processor chip will require some amount of support in the system logic chip set. Sophisticated system and peripheral power management will require external activity monitors to assert SMI# when a peripheral or the system has been idle for a certain length of time. External I/O trapping logic will be required to decode I/O cycles and selectively activate SMI# on cycles to devices which have been powered-down in a restartable state. Intel already has these features in their 82360; C&T has I/O trapping.♦