SLAC-PUB-5017
June 1989
(N)

SECTION III—USER ISSUES

CHAPTER 2—BUS GUIDELINES AND TRENDS

DAVID B. GUSTAVSON

Stanford Linear Accelerator Center
Stanford University, Stanford, Ca.94309

To be Published in a McGraw–Hill book: Digital Bus Handbook
Editor: Joseph di Giacomo

*Work supported by the Department of Energy, contracts DE-AC03-76SF00515

SECTION III—USER ISSUES

CHAPTER 2—BUS GUIDELINES AND TRENDS

2.1 Evolution of Buses—What is Coming Next?

2.2 Choosing a Bus

2.1 EVOLUTION OF BUSES—WHAT IS COMING NEXT?

Early computer buses were introduced to provide for incremental expansion of memory and I/O. As the level of integration increased, the amount of memory which would fit on a board increased from 1K bytes through 4K, 16K, 64K—suddenly the entire address space of a microprocessor could be filled by the memory on one board! Tricks had to be introduced to make holes in the memory address space so that small EPROMs could temporarily appear in place of part of the RAM to provide the bootstrap program for startup. Soon 64K-bit chips appeared, making it easy to put an entire microprocessor and its memory system on the same board. I/O devices became more compact too. A serial port which once occupied a board by itself soon became a few chips and then one, allowing many ports on one I/O board, or one or two ports right on the processor board. Disk controllers shrank. The next step was to expand processor address space through bank switching, segmentation or other memory management schemes so that larger memories could be used. Memory density continued to increase, so that soon after each processor enhancement it again became possible to put all the memory the processor could address on the same board with the processor.

Today the norm is 32-bit microprocessors (68030, 80386), and 1-Megabit memory chips. For the moment, the address space is not limiting the memory size; however, it is easy to fit memories which are appropriately matched to present processor power on the same board with the processor. As processor power increases, memory sizes should increase too. 4-Megabit chips are nearing production, 16-Megabit chips are being demonstrated, 64-Megabit chips are under development (estimated arrival quoted as 1994). For a 32-bit-wide memory, that would provide 256 Megabytes of on-board memory, not a bad match for a 100 million instruction per second processor.

However, as processor power increases, memory speeds must increase as well as sizes. This generally implies one or more levels of cache memory, relatively small but very fast

memory which is matched to the processor speed requirements. The top-level cache is probably located on the processor chip. It tries to keep the processor busy by prefetching a few instructions and by holding recently used instructions which might be part of a short loop. The second level of cache is much larger, and tries to keep whole pages of instructions and perhaps even whole arrays of data close at hand. To keep the processor running efficiently, this cache should contain the desired item almost all the time and only rarely should the processor wait for an access to main (slow) memory. Even rarer should the processor need to wait for an off-board access which uses the system bus, because bus access potentially has a long delay associated with it: there may be conflicting traffic on the bus, and perhaps even other processors of higher priority waiting to use it first. In some systems all the memory on the processor board is handled like a large cache, with a central shared main memory accessed via the bus, or a virtual memory system may simulate a large main memory by swapping blocks to and from disk storage as needed.

As the need for processing power increases, it becomes increasingly attractive to provide it by adding processors. Because of the delays associated with using the system bus, these processors will have their own on-board memory and cache. The system bus will be used for loading the memories initially from disk storage, for inter-processor communication, and for some kinds of I/O. Most traffic will be block transfers, but a significant number of accesses will be short transfers for process synchronization.

With multiple processors and caches, it becomes important to avoid inconsistency. Suppose an item is entered into two caches, and then one processor changes its copy. The other would continue to see the original value, unless a cache coherence mechanism is used to force all copies to be updated whenever one of them changes. With a coherence mechanism in place, process synchronization activity (looping while testing a lock variable) does not cause bus traffic until the lock variable changes, a significant gain in efficiency.

The buses described in this handbook range in performance from tens to hundreds of Megabytes per second. Why can't they go faster?

Synchronous buses, such as NuBus or MultiBus II, are limited for all time by their standard clock frequency. This frequency is chosen as high as practical at the beginning of the bus's life, and cannot be changed compatibly even though technological advances might make it easy to increase it later. In practice, there is not as much room for growth as one might have thought, because of the time it takes for signals to travel from one end of the bus to the other. Typical propagation speeds are a fraction of the speed of light because of capacitive loading of the bus transmission lines, and the imperfections of the transmission system require extra time for signal reflections to settle out.

It is impossible for buses based on TTL or CMOS signalling to be driven and terminated as proper transmission lines: the loaded bus impedance is so low because of distributed capacitance (typically under 40 ohms) and the signal voltage swing is so large (3 volts or more) that the currents needed are unrealistic (150 milliamperes or more per signal in the given example—note that the driver sees two transmission lines in parallel, so the current is $2 \times \frac{3}{40}$ amperes). To reduce the current to tolerable levels, the termination resistance is typically increased to something over 120 ohms; but this causes the signals to be too small at first, because the current change is too small, and then to reflect back when they reach the ends of the bus. Only after several reflections do the signals reach their desired 3-volt height, and thus the system must provide some delay before using the signals so that they have time to reach near full size before they are used[1].

Older asynchronous buses, such as VME, are limited by the propagation delay for signals from transmitter to receiver, and again for the acknowledging signal back to the transmitter,

and these delays are further increased because of the imperfections of the transmission system in the same way as for the synchronous buses described above.

Modern asynchronous buses, such as Futurebus+ or Fastbus, gain much of their speed advantage by careful attention to the physics of signal transmission. Low-capacitance transmitters and receivers, combined with small signal voltages, allow better termination of the bus, which speeds the transmission system and generally eliminates the need to wait for reflections to settle. Nevertheless, the round-trip propagation delay is still a fundamental limitation.

Pipelined transfer modes can be used in order to beat this problem. Once the connection between sender and receiver has been established, the sender can clock out a block of data at any mutually-agreed rate, without waiting for the receiver to return its acknowledging handshakes. This is a kind of source-synchronous transmission, the clock flowing with the data instead of centrally generated, and is thus well suited to long transmission paths like cable buses. Throughput is independent of distance (approximately).

Even pipelined transfers have limits, however. The obvious limits are the transceiver maximum frequency of operation and the high-frequency attenuation of the bus transmission lines, but in practice a less obvious limit, skew, is more important.

Skew is the difference in propagation delay from one bit to another across the transmitted word. Slight differences in the transmission lines or loading may change propagation velocities, differences in transceivers may change delays, or differences in switching threshholds may change apparent delays due to the finite rise and fall times of signal waveform edges.

Somehow, when a sequence of words is transmitted along the bus, it must be possible to tell which bits belong to which word when they arrive; if the skew is too large compared to the

signalling rate, some bits from one word may arrive after the first bits of the next word, confusing the receiving logic and mixing the data. With sufficient effort skews can be made very small, but in practical worst-case designs the skew allowances must be amazingly large, and significantly limit the throughput of pipelined transfers.

Throughput can be increased by widening the bus, and though this usually results in some increase in skew it is certainly a net gain. System considerations such as power dissipation, number of pins available on reasonably-priced connectors, and transient-current magnitude (consider switching 256 signals each 50+ milliamps at the same time!) tend to set practical limits.

Note that even the top-performing buses multiplex the address and data on the same set of lines. Although it would be slightly faster to have separate lines for the address, the gain is so small compared to the cost that separate lines are impractical. This is especially true in systems which use block transfers (one address cycle followed by many data cycles), like systems containing high-performance cache memories near the processor. In fact, it is easy to see that given the extra lines for addresses, you gain by widening the bus correspondingly and multiplexing the new wider path.

In short, it is hard to imagine improving bus systems even a factor of four beyond present Fastbus or Futurebus+. Buses are not going to improve at the same rate as processor performance or system demands.

So what can be done?

The first step is to abandon cycle-by-cycle handshakes, carrying the pipeline idea a step further: transmit a packet, containing address, data, and command information. Expect a packet in return containing an acknowledgment and perhaps requested data.

The next step is to abandon the bus structure, using a collection of unidirectional differential signalling links instead. Differential signalling eliminates transient currents, especially if unidirectional and continuously operating. (Turning off one set of drivers and turning on another, which happens on a bus while changing from an address or write cycle to a read cycle, or while changing from one master to another, does cause a major redistribution of current, i.e. transients, even when using differential signalling.)

To increase multiprocessor system communication throughput, connect the links through a switch network.

To reduce wasted intercommunication for synchronization etc., use caches and provide a mechanism for keeping all the caches consistent.

A new IEEE Computer Society Microprocessor Standards Committee project, P1596 "Scalable Coherent Interface," (SCI) was started in October 1988 to pursue this approach. Its goal is to support up to 64K processors with 1 GigaByte/sec per processor, using distributed cache directories to maintain coherence. It also has a "small & cheap" mode, which connects multiple boards as a ring (output link of one connected to the input link of the next) rather than through a switch, dividing the 1 GigaByte/sec among them.

The 1 GigaByte/sec speed was chosen as a barely-practical number for current technology, balancing the difficulty of transmitting signals and the difficulty of making fast integrated circuits large enough to handle the interface. There is no fundamental limit to future versions' throughput, though of course the speed of light always limits the latency, the time between a request for information and the correponding response.

---

[1] 'Computer Buses—A Tutorial', D. Gustavson, *IEEE Micro*, Vol. 4, No. 4, August 1984, pp 7–22

2.2 CHOOSING A BUS

After seeing the variety and complexity of the available buses as described in this handbook, you may wonder how to select the right one.

Of course, the vast majority of computer users need not concern themselves with choosing a bus at all, and should simply buy the available system which best meets their needs. The fact that you are reading this handbook, however, suggests that your life may not be that simple.

Perhaps you are a system architect, planning for your company's successful future. You want to choose the optimum bus, considering breadth of market, competition, reliability, performance, convenience, economics, smooth growth path.

Perhaps you are designing your own interface board. You want to choose a bus with the functionality your board demands, but which also has good software and bus monitoring hardware available to support debugging. If you plan to manufacture the board for sale, you care about the market associated with your bus. If you need to run your business with the same machine, or want to standardize on one machine for all your needs to gain reliability through redundancy (often a good idea) you may need to consider the availability of software and tools in fields far different from that of your own board design and use.

Sometimes the best solution is to use multiple buses, with interfaces between them. For example, you might use VME, STD, or STE for an expandable simple I/O system, and interface to Futurebus+ which you use to build your multiprocessor. For very large I/O requirements (multiple crates) or for large "farms" of processors (which are independent enough not to need a cache coherence mechanism) you might use Fastbus, with an interface to Futurebus+ for tape or disk I/O or for smaller coherent multiprocessor subsystems (even Futurebus+ cannot easily maintain cache coherence beyond crate boundaries).

Bus interfaces are simple if the I/O side never has any masters and has a small address space compared to the processor side. If both sides have masters (even DMA controllers) things get much more difficult, because there is then the possibility that traffic may try to go through the interface in both directions at once. Unless at least one of the buses supports a back-off mechanism, this can result in deadlocks. Fastbus and Futurebus+ were designed to work with bus interfaces, but VME and many of the simpler buses have fundamental problems. If the I/O side has too large an address space, some kind of address translation mechanism will be needed to map part of the processor address space onto part of the I/O space. It will be inconvenient to manage this translation during system operation.

Another problem area for bus interfaces is the handling of a multiprocessor synchronization semaphore or lock variable. There must be some way to prevent any interference from other processors while a lock variable is being tested and set. On a single bus, this may be accomplished with a read-modify-write operation, but if any of the boards are multiply-ported (e.g. a two-port memory with another processor or bus on the other side) the other ports must be locked out too during the critical sequence. At first, this seems beyond the domain of the bus specification, but if this kind of appropriate lock behavior is not provided it may be impossible to operate multiprocessor systems reliably.

Some buses provide a much better growth path than others. Unless expandability was designed in at the start, it is likely that you will hit significant barriers when you overflow a single crate. Address space is the most difficult resource to expand, though we *have* had a lot of experience with that problem. A bus which does not support bus interfaces will cause a lot of expansion trouble (which usually does not show up until the end of the development cycle, when bus activity gets high and the multiprocessor OS starts running). Some buses claim to offer multiprocessor support, but begin to fail when bus loading and the number of processors is high.

Some features which receive prominent treatment in the promotional literature may not be very important to you in practice. For example, you probably don't need special hardware support for a message-passing facility as long as you have some shared memory and a directed interrupt mechanism which permits one processor to signal another. On the other hand, message-passing hardware may provide an inexpensive protection mechanism which limits one processor's access to another to reduce the system's vulnerability to program errors.

Significant features which may help you select a particular bus:

(By I/O in the following I mean simple parallel registers, A/D/A converters, etc. High performance I/O such as disk controllers probably need DMA and bus mastership capabilities, and thus should be treated like a processor and included on the processor bus).

Fastbus (ANSI/IEEE 960, IEC 935): Support for many crates allows for large systems with lots of parallel non-interfering activity. A cable version operates reliably over many-meter distances. Excellent heat handling, power distribution, and ground distribution including a "clean earth" quiet analog ground. Very high performance. Large boards. Very economical for large systems (about half the price of VME per unit of usable board area, powered and cooled, according to the CERN laboratory which is a large user of both). Supports bus interfaces and some lock mechanisms. Expandable protocol and addressing. Transfers full words; byte addressing and partial-word transfers must be handled by the processor/bus interface. An associated software interface standard provides application portability.

Futurebus+ (P896.1): Cache coherence mechanism (for a single crate—coherence across a few crates can be achieved with complex interfaces, or perhaps more easily by interfacing each crate to SCI, converting to SCI's directory-based coherence mechanisms). Medium-size boards. Supports bus interfaces and lock mechanisms. Performance in practice about

the same as Fastbus; 64- and 128-bit-wide versions can outrun Fastbus in a single-crate system.

Futurebus+ probably will have wider commercial support than Fastbus—the U. S. Navy has endorsed Futurebus+ as its Next Generation architecture; VITA (the VME manufacturers organization) has declared it to be their next step after VME; MMG (the Multibus II manufacturers organization) has also declared its intention to support Futurebus+, at least for cache-coherent subsystems.

A standard bus-independent Control and Status Register and I/O Architecture is being developed (IEEE Computer Society Microprocessor Standards Committee project P1212) to make migration easier. This was begun as part of the SCI project, then made independent so it would be equally applicable to Futurebus+ and the P1394 Serial Bus, and also to the older buses as new products are developed for them.

The recommended strategy is for users of the older buses to move gradually toward Futurebus+ (via bus interfaces or converters and via CSR and I/O architecture), and then on toward SCI, providing a smooth growth path which provides practically unlimited expansion and power.

MULTIBUS II (ANSI/IEEE 1296): Moderate performance for a few processors. No cache coherence mechanism, so expansion into a highly parallel coherent shared-memory implementation is constrained. A rich and complex architecture, including hardware support for message passing.

NuBus (ANSI/IEEE 1196): Moderate performance for a few processors. No cache coherence mechanism. A very simple and elegant system, efficient to implement. Used in the Apple Macintosh II family at the standard 10 MHz speed. The protocols are used in the Next machine in a 4-slot 25 MHz CMOS version (not compatible with standard NuBus boards).

VME (ANSI/IEEE 1014): Major commercial support, many I/O cards available. Suitable for single or few-processor systems, or for an I/O system used with another bus.

STD (ANSI/IEEE 961) or STE (ANSI/IEEE 1000): Many I/O cards, small format, many vendors; good simple I/O buses for use with another bus.

SCI[1] (P1596): Under development as IEEE Computer Society Microprocessor Standards Committee project P1596, Scalable Coherent Interface, it may be worth considering by the time you read this. It is designed to support bus interfaces, lock primitives, and cache coherence over a distributed system containing large numbers of processors, providing 1 GigaByte/sec per processor. For small systems, several processors can be connected in a ring to divide 1 GigaByte/sec among them, at low cost. Large systems require a more costly switch system for interconnecting the processors. The architecture is based on 64 bits, but the physical hardware uses a narrower path (typically 16 bits). A serial fiber-optic (1-bit-wide) version is also planned, running at lower speed but over longer distances.

---

[1] SCI should become a finished IEEE standard sometime in 1990. As of this writing (June 1989), however, the best introduction in print is 'Scalable Coherent Interface' by Ernst H. Kristiansen, Knut Alnæs, Bjørn O. Bakka, and Marit Jenssen, Eurobus Conference Proceedings, Munich, 9–10 May 1989. Because SCI is still actively evolving, only the principles are likely to remain constant—details have already changed slightly.