The Alpha Distributed Computer System Testbed

J. Duane Northcutt

TR #88033

Archons Project Technical Report

March 25, 1988

The Alpha Distributed Computer System Testbed

J. Duane Northcutt

March 25, 1988

Archons Project Technical Report #88033

This research was supported in part by the USAF Rome Air Development Center under contract number F30602-85-C-0274. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of RADC or the U.S. Government.

Table of Contents

List of Figures

List of Tables

Abstract

The Alpha Distributed Computer System Testbed is the hardware base on which the Alpha decentralized operating system is constructed. This facility was created following a study of the needs of the Archons project and an extensive survey of commercially available hardware. This testbed is composed of a collection of hardware that meets the project's minimum functional specifications, conforms to the project's cost constraints, and requires a minimum amount of custom hardware fabrication to put it in place.

The testbed has been in place for some time now, and despite the fact that it has served the project well, a number of potential improvements to the testbed have become evident in the course of its use. Some modifications have been made to the initially chosen, off-the-shelf hardware used in the construction of the testbed. Furthermore, plans exist for a series of future upgrades to the testbed, including the use of: next generation processors; multiple application processors; semiconductor-based secondary storage devices; and redundant, high-performance, real-time-oriented communication networks.

This document includes a description of the testbed configuration as it exists today. To date, there are two (identical) instances of this testbed—one at Carnegie-Mellon University and the one at General Dynamics Corporation, Fort Worth Division—and this document applies equally well to either instance.

The intent of this document is to describe the requirements that the testbed was designed to meet, the structure of the testbed, and a summary of the experience gained in using the testbed. This document also describes a collection of improvements that could be made to the testbed, and the modifications that were made to the testbed's various components. In addition, this document includes technical descriptions and schematic diagrams for the boards which make up the testbed nodes.

1 Introduction

The Archons project at Carnegie-Mellon University began in 1978, immediately following a largely hardware-centered project (HXDP at Honeywell [Jensen 78]), and was directed toward the development of fundamental technology for decentralized real-time operating systems. In 1982, following an extended period of conceptual exploration, the project became interested in validating some of its research concepts through the implementation of a system, and the search began for a testbed on which these experiments could be performed. A study was performed to determine the project's needs for a testbed and a survey of existing hardware was carried out [Clark 83].

The original objectives and rationale for the testbed are provided here, despite the fact that this system was specified and constructed quite a long time ago (as judged by the rate at which computer technology is changing), and in retrospect a number of choices might have been made differently. In particular, there were fewer reasonable processing element options then—the Sun-1 was a new (and relatively expensive) product and used only an MC68000 CPU; IBM-PCs were not such a clear-cut standard and were not a particularly attractive platform at the time.

To provide the proper context for describing the testbed, the following sections describe the testbed's initial requirements and objectives, the system hardware considered as candidates for the testbed, and the hardware ultimately chosen for the testbed.

1.1 Requirements and Objectives

This facility was constructed to meet the needs of the Archons project (primarily the Alpha decentralized operating system [Northcutt 87]). The testbed was created following a survey of commercially available hardware. One of the initial constraints placed on the testbed design was that it be constructed from off-the-shelf hardware; the project did not have the time or resources to build hardware for what is a system software research effort. An additional constraint had to do with the total cost of the testbed. In order to make the testbed more cost-effective, the machines that make up the testbed should be multi-purpose—i.e., they should be able to be used to execute the experimental operating system's code and also to develop the code and to perform other project functions (e.g., word-processing and electronic mail).

A major requirement was that the testbed support the empirical portion of the project's research into operating systems for distributed computers. This requires multi-computer hardware, consisting of a collection of processing nodes interconnected by a "thin-wire" communications network (ideally, this would be composed of a set of replicated, bit-serial buses, that use a deterministic bus-allocation scheme). Based on the needs of multiple, simultaneous system developers and the requirements of realistic applications, the testbed must consist of from five to ten processing nodes.

Another requirement was that the testbed provide hardware concurrency within each processing node. This is based on the project's experience with previous distributed systems that indicated the need for actual concurrency in order to support such decentralized operating system functions as time-driven scheduling and inter-node coordination and communications. Therefore, it is a basic assumption that hardware concurrency is required within the testbed's processing nodes to support the project's continued research into the exploitation of concurrency within decentralized operating systems.

An additional requirement is that the testbed's processing nodes be representative of modern processor technology. This is because the project's research is directed towards developing technology for future operating systems and the viability of these concepts cannot be adequately judged by implementations that are heavily biased by the nature of technologically obsolete hardware. The application processors should have representative instruction execution times, reasonably sophisticated memory translation hardware (e.g., enforce memory access restrictions, permit system notification on different types of memory accesses, and allow memory moves by mapping instead of copying), and should not have overly restrictive memory addressing or memory size limitations.

An objective of the testbed was that it support multiple simultaneous users, at any phase of program development, testing, debugging, or evaluation. The type, and number, of processors required for this testbed tend to be too costly to be dedicated single-user machines, where system developers must queue up for serial access to the testbed. This would not only be a non-cost-effective use of hardware, but it would create a bottleneck in the project's development cycle.

Another objective was the need for system program development to be performed remotely. It is typically the case that low level system software development must be performed within close physical proximity to the target machine (in order to have the necessary control over the hardware). However, the physical arrangement of project resources calls for the ability for system programmers to work in their own offices, while the testbed's target execution machines are in the laboratory.

The initial intent was to begin with a cross-development environment, in order to make use of a suite of sophisticated system program development utilities that support the generation of code for, and the interactive interface to, the target machines in the execution environment. The project could not afford to develop native development tools (e.g., compilers, linkers, editors, and debuggers). Therefore, a further requirement was that a high degree of integration should exist between the target machines and the development machines in order to maximize the leverage obtainable from an existing development environment.

The testbed was also required to be highly extensible—i.e., easily expandable through the addition of processing nodes, at a low cost. Also, it was desired that the number of different types of components be kept to a minimum, in order to simplify the process of supporting the testbed by allowing for a high degree of interchangability within the testbed. These requirements led to the desire for a system based on standard hardware interconnection structures, which allows more choices and greater flexibility in configuration than systems based on non-standard buses, and a system for which detailed technical documentation is available. Therefore, proprietary products were ruled out early in the testbed selection process.

Finally, it was desired that the testbed also provide support for fault insertion capabilities in order to simulate the failure of processing nodes as well as the communications interconnect.

1.2 Alternatives Considered and Rejected

In order to meet the defined constraints a number of different options were considered for the testbed hardware. The following is a summary of some of the machines that were examined for use as processing nodes within the testbed, their salient characteristics, and the reasons for their consideration.

- **IBM PC with 68000 Co-Processor**

Considered as a candidate testbed node was the IBM PC-XT with a Sri-Tech MC68000 co-processor add-in card. With this configuration, the 68000 co-processor would be used as an application processor, while the PC would be used to support the peripheral subsystems. An Ethernet interface card was commercially available at that time, as well as a limited number of other local network interfaces (e.g., ARCnet).

At the time of this evaluation, the project had a relationship with IBM that made it possible to obtain IBM PCs at a significantly lower cost than was widely available.

This configuration did not support a great deal of hardware concurrency—there would only be two points of (programmable) hardware control. Also, this system suffered from the fact that two different families of processors would have to be supported within a node (i.e., Motorola MC68000 and Intel i8088). At the time, the IBM PC programming environment was not sufficiently well developed to make it a proper program development host. Therefore, program development would have to be done either on the MC68000 co-processor under Xenix, or on another type of machine altogether. It was also unclear if it was possible for the project to obtain the necessary technical documentation for this machine configuration.

- **IBM CS-9000**

Also considered was a machine made by IBM Instruments Division, known as the CS-9000. This machine was intended as a laboratory computer and the basic system consisted of an 8MHz MC68000 CPU with 128KB of read/write memory, 128KB of PROM, three 16-bit timers, four DMA channels, three serial and two parallel ports, and an integral keyboard. At the time, the available options included an array of floppy and hard disks, an Ethernet controller, and several types of instrument I/O controllers.

IBM had agreed to make the necessary technical documentation available to the project, and promised to make this equipment available to the project at an attractive price.

This machine was based on a variation of the Versabus and so there were few add-in cards available (and they were high priced), and it provided no viable alternatives for supporting hardware concurrency within a processing node. Furthermore, the software development environment was not adequate to meet the project's requirements. At the time, the system ran OS 1.1, with Xenix promised for some time in the future.

- **Charles River Data System MULTIBUS System**

Representative of the large number of commercially available systems was a MC68000/MULTIBUS-based system from Charles River Data Systems.

This machine had the advantage of being based on the MULTIBUS, which allowed multiple processor boards to be placed into a single backplane, and supported a great deal of flexibility in the choice of peripheral (e.g., network, disk, I/O) controllers that could be used.

The software development environment available for these machines was not sufficient to meet the project's needs and thus additional software development machines would have to be purchased. Furthermore, these machines were quite expensive and required that a great deal of effort be put into the task of system integration to obtain the kind of testbed that was needed.

- **Homebrew MULTIBUS System**

Among the options considered was the in-house construction of testbed nodes based on commercially available MULTIBUS boards.

There existed a great range of different types of processors and I/O controllers, and by integrating the system in-house the cost of each node would be less than similar systems purchased from OEMs.

The cost, in terms of dollars and manpower, of such an approach was considered prohibitive, because of the difficulties associated with even such simple hardware integration tasks coupled with the creation of the necessary software development environment.

- **CM***

In addition to the hardware survey performed as a part of this effort, a pair of graduate students associated with this project were interested in using the CM* system as a testbed for their thesis work, and suggested that the rest of the project consider it as well. Based largely on advice from previous users and the engineering staff, and the obvious incompatibility between this multiprocessor's architecture and the project's testbed requirements, the project as a whole never pursued this option with any degree of seriousness.

1.3 Ultimate Choice and Rationale

Following the survey of potential testbed systems, the decision was made to choose Sun Microsystems workstations as the development and execution platforms for the testbed. At the time that the decision was made, only the Sun-1 was available—this machine used the version 1.0 processor board (which was MC68000-based) and ran the UNOS operating system. However, Sun was about to begin delivery of the Sun-1U—i.e., the machine that used the version 1.5 processor (which was MC68010-based) and ran 4.1BSD UNIX.

The Sun workstations were MC68000/MULTIBUS-based (which allowed for multiple-processor configurations and flexibility in the choice of I/O controllers), they ran UNIX (which provided a superior program development environment), and the basic workstations had a large amount of primary memory, a high-performance custom MMU, and a reasonable suite of on-board devices (e.g., timers and USARTs). Sun had indicated to us that Lucasfilm had successfully placed multiple version 1.0 processor boards in the same backplane.

The project obtained a special pricing arrangement from Sun to make the purchase of a collection of these machines affordable. Especially attractive was the fact that these machines could be used both as testbed execution nodes and development workstations. Furthermore, Bill Joy and Andy Bechtolsheim of Sun Microsystems met with project members, discussed the project's plans, and as a result, made it possible for the project to obtain a large number of version 1.5 processor boards from Sun. These boards could be used to create multiple-processor testbed nodes in support of the hardware concurrency requirements.

Ultimately, the project also purchased a collection of (at that time, newly announced) Sun-2 workstations. These machines were MC68010-based, had a number of relatively minor enhancements over the Sun-1 systems, and ran the SunOS version of the UNIX operating system. In addition, a collection of version 1.5 processor boards, along with several MULTIBUS enclosures (which included power-supplies and backplanes), were purchased.

Joy and Bechtolsheim also enabled the project to obtain the necessary technical documentation for the version 1.5 processors (including the PROM source code and the schematic diagrams for the hardware)†. Without this detailed technical information it would not have been possible for the project to develop an operating system on bare-hardware (this project did not intend to make minor modifications to some currently running operating system, such as UNIX).

This is how the project began on its path of using the Sun Microsystems products, both as turn-key development machines running UNIX and as the execution targets within the testbed's processing nodes. This path began with the testbed nodes based on Sun-1 machines and Sun-2 machines as development workstations. Later, the testbed was upgraded to use hybrid Sun-2/Sun-1-based testbed nodes and Sun-3 workstations as software development machines. The project is currently contemplating another upgrade that would have the testbed nodes be Sun-3-based and have all of the software development work be performed on Sun-4 workstations.

---

†*This task was not accomplished without the expenditure of considerable effort on the part of various project members. Sun Microsystems has proved to be quite concerned about the proprietary nature of the detailed technical information about their most recent machine offerings. However, once a new model has been announced, they have been much more compliant with such requests.*

2 Testbed Structure

The Alpha testbed can be examined at three different levels of abstraction—the *system* level, the *node* level, and the *processor* level. The system level encompasses the hardware used for application program control and development work, the distributed computer on which the project's decentralized operating system is to execute, and the physical application processes being controlled by the distributed computer. The node level is concerned with the architecture of the individual processing nodes in the distributed computer system, while the processor level defines the components and structure of the processing elements in each node.

2.1 System Level

At the system hardware level of abstraction, the testbed consists of three components—the *development and control system*, the *distributed computer system*, and the *application system* (see Figure 1).

The control and development system is used for the development of application programs, the control of applications running on the distributed computer system, and the monitoring of the operating system as well as application experiments. The distributed computer system component of the testbed is the host on which the Alpha operating system executes. The application system is comprised of a set of application devices, and is the interface between the distributed computer on which the Alpha kernel executes and the physical system it is monitoring and controlling.

2.1.1 Development and Control System

Although it is possible that an application system itself may contain all of the devices necessary for interacting with and controlling this experimental system, this is not presumed to be the case for the Alpha testbed. Therefore, a set of *control stations* is integrated into the testbed system for the purpose of software development, experimental control, and monitoring of the testbed's distributed computer system. The workstations that make up the testbed's control and development system can be connected to the distributed computer system through its nodes or its communications network (either directly or through gateway machines).

In the current testbed the control stations are standard Sun Microsystems workstations, running the UNIX operating system. The Sun/UNIX software environment provides a wide range of useful programming utilities (e.g., window interface, compilers, editors, and debuggers), and the Sun Microsystems workstations serve as excellent cross-development platforms for writing code for the testbed nodes. To support the remote development of stand-alone operating system code and to allow remote control of system experimentation, a number of new software tools were created, and existing tools had to be modified (for further details, see [Northcutt 88d]).

These control stations provide users of the testbed with a sufficiently high degree of control over the testbed nodes to permit the development of low-level code in a totally remote fashion. Testbed users have the ability to communicate directly with each of the processing elements in each of the testbed nodes. The Sun Microsystems window management tools are used to allow the developers to simultaneously control multiple testbed

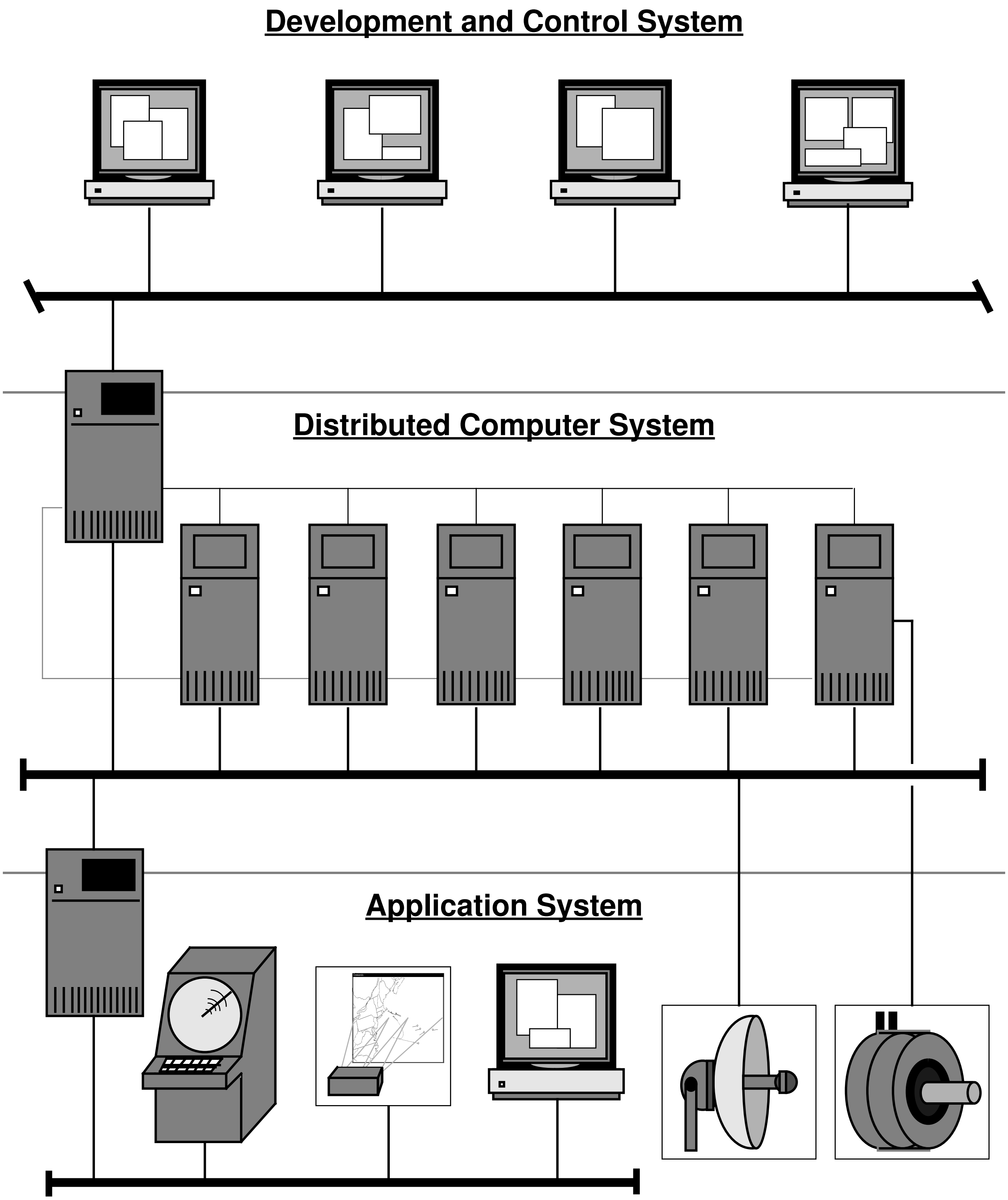


Figure 1: System Level Structure

nodes. A window can be created at a workstation for the console serial-lines associated with each processing element in each node being used. Programs can be down-loaded from the Sun workstations to the individual processing elements in the testbed nodes, and information can be up-loaded to the workstations from the memory of any processing element in the testbed. Furthermore, remote users have the ability to control the AC power to the individual testbed nodes (to clear hung nodes and for failure recovery experiments).

In the current testbed, the control workstations interface with the testbed nodes via both the Ethernet and the 9600 baud serial-lines provided by a gateway machine. One of the control and development workstations (running UNIX) has two Ethernet controllers and acts as a gateway between the testbed's distributed computer and the outside world. With this configuration, software running on the testbed nodes is able to communicate with programs running under UNIX on the control workstations via standard protocols.

2.1.2 Distributed Computer System

The distributed computer system provided by the Alpha testbed consists of a set of processing *nodes* interconnected by a global communications network. The nodes in the system are the individual hardware computational units on which the individual portions of a decentralized operating system may execute. Logically, the distributed computer can be thought of as a single, conventional uniprocessor, with the nodes representing the boards and the communications network representing the backplane.

Nodes in the current testbed are homogeneous in architecture as well as in their interconnection to the global communications network†. The testbed consists of a set of (currently six) custom-built, dedicated-function, shared-memory multiprocessor nodes, based on Sun Microsystems Model 2/120 workstations. Each node consists of a collection of (3 or more) processors with local memory, shared memory, and various I/O-device controllers (e.g., serial-line, network, disk), all in a common MULTIBUS backplane (having 9 slots with specialized P2-bus connections).

The nodes of the distributed computer are interconnected by a private Ethernet cable that is connected to the Computer Science Department's Ethernet via a gateway. The gateway machine is a standard Sun Microsystems workstation, running the UNIX operating system, with an Ethernet controller connected to each of the Ethernets, and a 16 channel asynchronous serial-line multiplexer. The gateway machine runs software that allows it to selectively connect or disconnect the two networks, and to communicate with the serial-line channels connected to the console ports of the processors in each node. Furthermore, the distributed computer system itself is partionable into separate computers by severing the interconnection network between nodes.

The Ethernet is not an ideal choice for the testbed's communications interconnection structure, however it was easily available, well-documented and has reasonably high bandwidth (i.e., sufficient bandwidth to either ignore the effects of the Ethernet's CSMA/CD bus allocation scheme, or allow the emulation of more deterministic bus allocation schemes).

†*This is not because we are not interested in the problem of heterogeneity, but rather as a convenience to allow us to focus on other issues.*

2.1.3 Application System

The application system is the interface between the distributed computer system and the physical system that it is monitoring and controlling. From the point of view of the system software executing on the distributed computer, the application system consists of a collection of *application devices*. Application devices can act as sources of data to the computer (e.g., sensors) or data sinks (e.g., actuators). Because the intended application domain of the Alpha operating system is real-time command and control, these devices are assumed to be intelligent, in the sense that all highly repetitive, fast response-time (i.e., sampled-data loop) processing is expected to be handled by the device interface controllers. Low-level signal data is therefore expected to be pre-processed and not passed around in bulk form on the distributed computer's communications network. While this may not be the case in low-level process control or signal processing applications, this is typical of supervisory command and control systems [Northcutt 88a].

Application devices may be connected to the distributed computer system through nodes, the network, or gateway machines. Connecting an application device to a distributed computer's communications network can be thought of as being equivalent to placing a peripheral controller into a conventional uniprocessor's backplane. Devices that are connected in this fashion must be capable of either simulating the necessary software interface, or executing the communication protocols necessary to interact with system software on a processing node. Alternatively, the devices attached to a testbed node are associated with the node, and an object at that node is responsible for performing the necessary software encapsulation of the device.

It should be noted that the application devices need not be real, but could instead be simulated. In fact, the application system in the current testbed is composed of a collection of processing elements that execute software to simulate the application devices and actual physical process(es) being controlled. In the current configuration, these processing elements are the same workstations that make up the control and development system, and are attached directly to the testbed's communications network.

2.2 Node Level

The Alpha testbed provides a distributed computer system, composed of a collection of processing nodes interconnected by a global communications network. These testbed nodes were designed to support the project's goal of experimentation with raising the semantic level of traditional system device interfaces. The objectives here are to reduce the semantic gap between the operating system abstractions and the underlying hardware, to achieve concurrency within the operating system itself, and to explore the possibilities of achieving increases in system performance through the downward migration of functionality. The nodes of the testbed support this goal by providing the system-builder with a collection of general-purpose processing elements that can be used to provide the desired subsystem functionality without having to construct custom hardware.

In support of these objectives, each node in the testbed is a symmetrical, shared-memory multiprocessor that is used to emulate a uniprocessor with intelligent device controllers by having one processing element (the *application processor*) be dedicated primarily to application programs, while other processing elements execute as much of the system software as possible in the guise of peripheral control subsystems. A node's peripheral subsystems are, conceptually, intelligent device controllers, that are implemented by general-purpose processing elements (similar to the application processing element), accompanied by standard device controllers. Each processing element in a node shares local memory via the node's common backplane, and has its own, on-board devices and local memory. Processing elements in a node typically execute out of their common memory, and interact with each other via shared memory and an interprocessor interrupt mechanism†.

Ideally, the application software executes alone in the application processor. However, because of the current limitations of the testbed, the application processing elements each execute a copy of the Alpha kernel in addition to the application code. Ultimately, the hope is to migrate all of the system functionality into operating system processing elements and intelligent peripheral controllers. In such a system configuration, there might be multiple application processing elements at each node, functioning in a manner similar to floating point co-processors—the application processing elements execute only the application code given to them by the system processing elements.

Logically, a node in the testbed is composed of three major types of components: application processor(s), peripheral co-processors, and interprocessor interface mechanisms. An application processor is a processing element (i.e., a process/memory pair) where the application programs and the kernel proper execute. A peripheral co-processor is a processing element that is used in conjunction with a simple device controller to emulate a programmable device controller—primarily for the communications and secondary storage subsystems, but also used to support scheduling. The interprocessor interface part of a node consists of common memory, an interprocessor interrupt generation mechanism, and is accessible to all processors and co-processors in the node.

A representative testbed node (as depicted in Figure 2) includes an application processor, a scheduling co-processor, a communications co-processor, a secondary storage co-processor, and a common MULTIBUS memory board. The application processing element is a Sun Microsystems version 2.0 processor, with (a minimum of) 1MB of local memory and an interprocessor interrupt module. The peripheral processing elements are (for reasons of cost and availability) Sun Microsystems version 1.5 processor boards, with custom hardware and firmware modifications. The Sun 1.5 processor board is a discontinued product of Sun Microsystems that was based on the original Stanford University MC68000 single board computer design, and modified to use the MC68010 microprocessor. The communications processing element has associated with it a Sun Microsystems Ethernet controller. This board consists of an Intel i82586 Ethernet controller chip, 256KB of local buffer memory, and an address translation unit. Likewise, the secondary storage processing element has associated with it a Xylogics xy450 high-performance disk controller and 84MB disk (with an SMD interface).

Although, the standard disk drives found in Sun-2/120 workstations have SCSI/ST506 interfaces, the Sun 2/120 enclosures used in the testbed were modified to accept 8-inch Fujitsu (SMD) disk drives. This was done because it was felt that the performance and reliability penalties of using SCSI/ST506 interface disk drives (which were significant at the time, and have not improved much since) were unacceptable.

†*The interprocessor communication mechanism is a custom designed and implemented hardware module that is added to the application processor and is equally accessible to all of the processing elements in a node.*

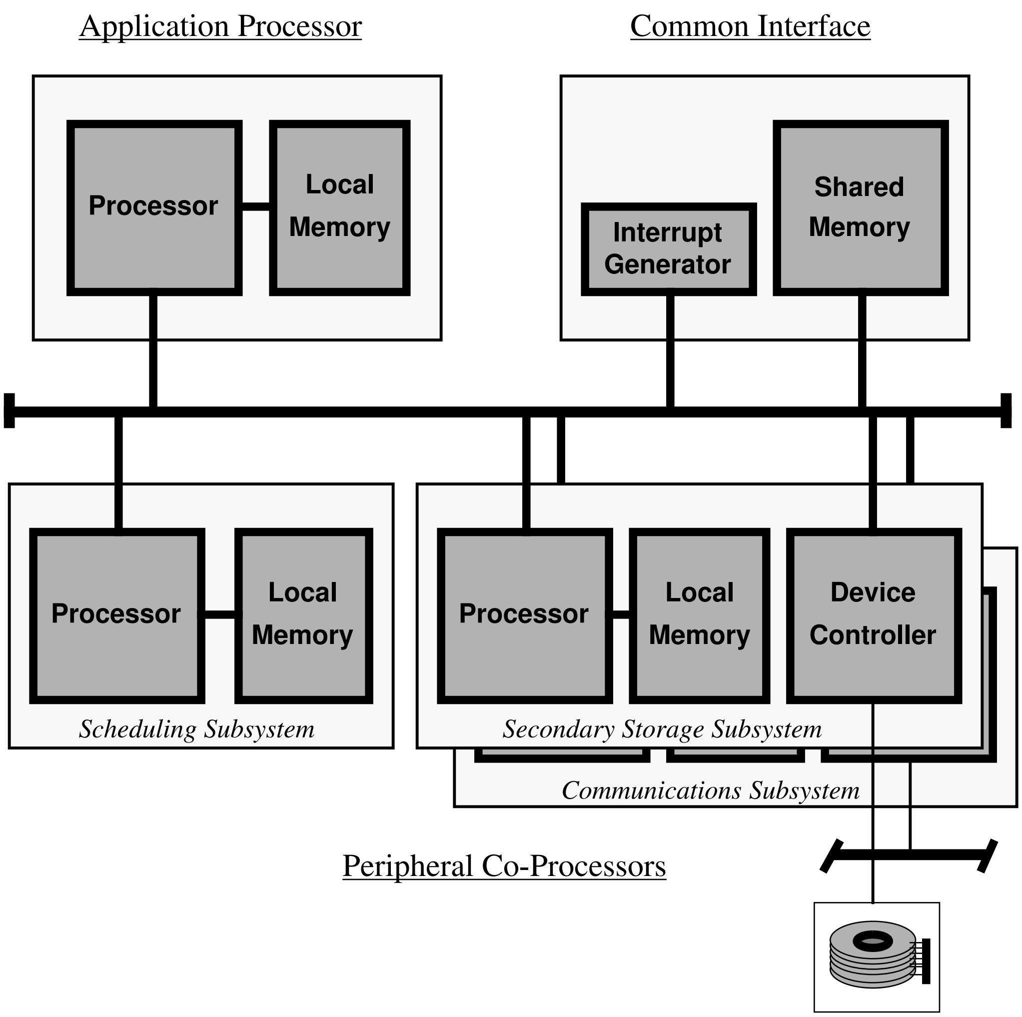


Figure 2: Node Level Structure

2.3 Processing Element Level

Initially, all of the processing elements in a node were Sun version 1.5 processor boards. However in the current testbed the Sun version 2.0 processor board is used as the application processing element, and the Sun version 1.5 boards are used as the peripheral co-processors. The following is a brief description of the major features of each of the two types of processor boards.

2.3.1 Sun 2.0 Processor

The Sun Microsystems version 2.0 processor board is a MULTIBUS single-board computer based on a 10MHz Motorola MC68010 microprocessor, with zero wait-state access to as much as 8MB of off-board read/write memory with byte-parity, and 64KB of programmable read-only memory. This processor also contains a Memory Management Unit (MMU) that supports a segmentation-with-paging virtual memory management scheme. Also included on the processor board is a dual, independently programmable USART, five programmable 16-bit timers, a 16-bit parallel input port, and a battery backed-up time-of-day clock. Furthermore, the processor supports MULTIBUS multi-master arbitration and allows the expansion of on-board memory via a proprietary high-speed bus, using the MULTIBUS P2 connector.

The structure of a Sun Microsystems version 2.0 processor board is depicted in Figure 3. Most of the features of the CPU are standard and require little explanation. The Motorola MC68010 is a multi-register, stack-oriented microprocessor, supporting a linear virtual address space of up to 16MB and having both client and supervisor protection modes, with a separate stack pointer for each mode.

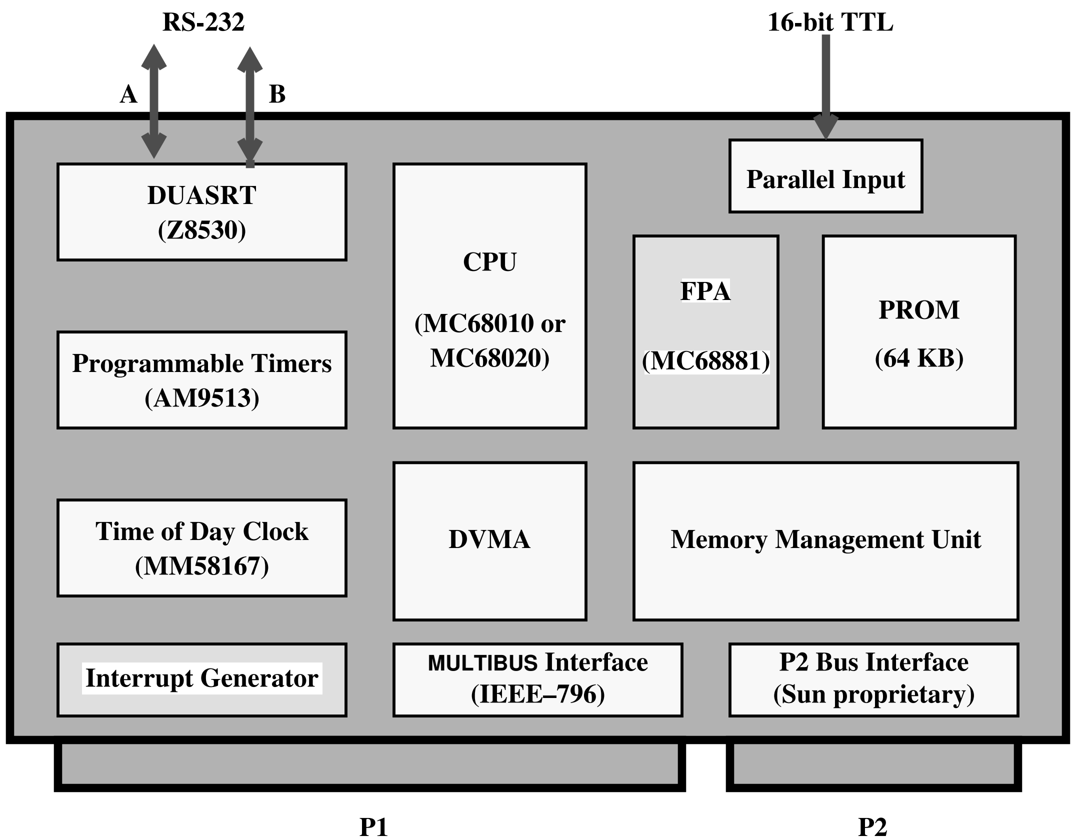


Figure 3: Modified Sun Version 2.0 Processor Board

While the MC68010 CPU does not provide memory management directly, it does support instruction restart to permit the implementation of demand-paged virtual memory, and the Sun Microsystems version 2.0 processor has a custom MMU on-board. The version 2.0 MMU supports a linear virtual address space of 16MB, which is divided into 512 segments (each consisting of a maximum of 32KB of memory), which are further divided into pages of 2KB each. The MMU is implemented as a two-level translation table and therefore has fixed limitations on the number of segment and page table entries. A system register (implemented as a memory location) known as the *context register* defines the current address mappings for the processor†. This is done by selecting one of the sets of segment table entries, known as *segment maps*. In the 2.0 processor, there are eight segment maps, each consisting of 512 segment map entries known as *segment descriptors.* Each segment descriptor consists of an (eight-bit) index to a

†*The context register consists of two parts—one is used when the processor is executing in supervisor mode, and the other is used when the processor is executing in user mode.*

one of the set of page table entries, known as *page maps.* The MMU's page table contains 256 page maps, and each page map consists of 16 page map entries, known as *page descriptors.* Each page descriptor contains validity, protection, and usage information, as well as a field for the physical page index of a single (2KB) page of memory. Figure 4 provides an illustration of the Sun version 2.0 processor's MMU.

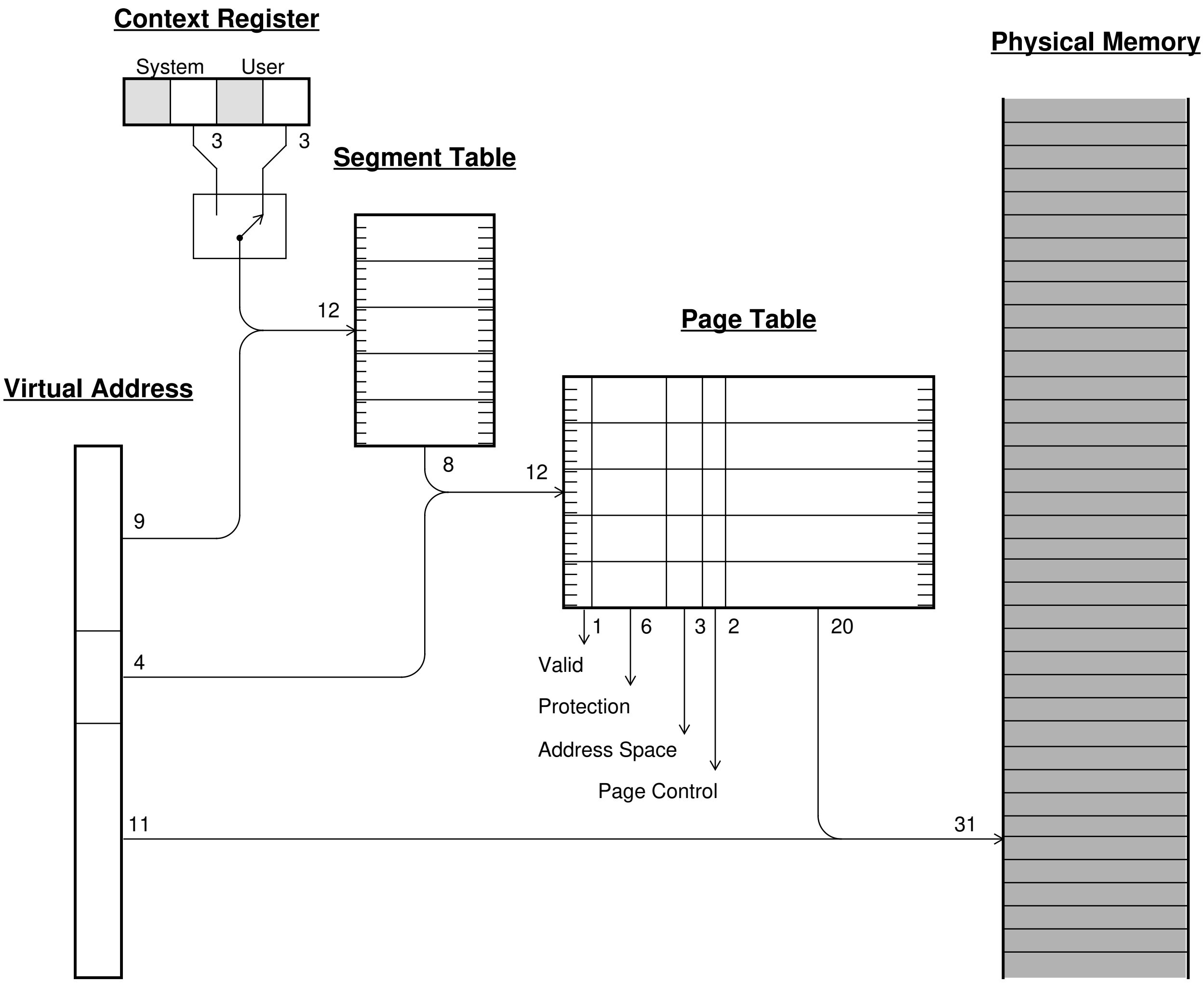


Figure 4: Sun 2.0 Memory Management Unit

A virtual address is translated to a physical address in the following way. First, the context register is used to determine the appropriate segment map to use, and the most significant nine bits of the virtual address emitted by the CPU are then used to index into this map. The segment table lookup yields an index into the page map table, and the next four bits of the virtual address are then used to select the appropriate page entry within the page map table. This yields a physical page index, and the least significant 11 bits of the virtual address are used to select the correct byte within the given physical page.

It is possible that a *page fault* may occur during translation of a virtual address. This happens when the referenced page descriptor is marked as invalid, the reference is in violation of the protection bits specified for the page, or no physical memory page can be found at a desired off-board physical address. In any of these cases, system software must determine the proper course of action when a page fault occurs (e.g., whether demand allocation of a physical page is called for and the faulting instruction is to be continued from the point of the fault, or whether the fault signals a system error and the instruction should be aborted).

In the Sun 2.0 processor, the page descriptor's protection field is unencoded—allowing all combinations of user/supervisor read/write/execute protections to be applied. Furthermore, the CPU's function codes are used to allow multiple, full 16MB address spaces to be used, as opposed to using physical address bits for selecting among address spaces. The address spaces provided in the Sun 2.0 processor are: local memory space, on-board I/O space, MULTIBUS memory space, and MULTIBUS I/O space. Table 1 indicates the assignment of processor function codes to address spaces. Note that code must be executing in supervisor mode in order to change the CPU's function code or to move data between address spaces. See [Motorola 84] for further details on the processor's generation of function codes and refer to [Sun 84] for additional details on their use on the Sun 2.0 processor board.

FC_2	FC_1	FC_0	Function Code Reference Classification
0	0	0	undefined/reserved
0	0	1	user data
0	1	0	user program
0	1	1	access to MMU
1	0	0	undefined/reserved
1	0	1	supervisor data
1	1	0	supervisor program
1	1	1	interrupt acknowledge

Table 1: Sun 2.0 Function Code to Address Space Mappings

The Sun 2.0 processor board can map a region of its virtual address space onto the MULTIBUS—i.e., MULTIBUS masters can make memory references that are translated by the processor's MMU, just as if the CPU had emitted the address. This feature is known as Direct Virtual Memory Access (DVMA) and makes it possible for the processor to make its local memory (or I/O devices) available to any MULTIBUS master. This function also serves to avoid the problems associated with typical multi-ported memory implementations. In typical systems, there is additional complexity associated with maintaining dual address mappings of physical memory locations—i.e., one mapping for the CPU that accesses memory through virtual addresses and another for the DMA channels that access local memory through physical addresses. All DVMA accesses are performed under the supervisor data function code, and all addresses from the MULTIBUS have the value 0xF00000 or'ed to them before being presented to the processor's MMU (i.e., MULTIBUS address 0 translates to processor virtual address 0xF00000). Also, the same protection rules apply to DVMA references as to CPU memory accesses. Should either parity errors or circular references (i.e., DVMA accesses that refer to the MULTIBUS) occur on DVMA references, the responsible MULTIBUS transaction is aborted.

In addition to the on-board I/O devices, the Sun 2.0 processor has a pair of memory-mapped system status and control registers—known as the *Bus Error Register* and the *System Enable*

Register. The Bus Error Register latches low-level indications of hardware errors for examination by software. This register indicates low and high byte parity errors, time-out errors, protection errors, if the reference was a P1-bus access, and if the requested virtual memory page was valid. The System Enable Register allows software control of low level processor board functions. This register allows software to enable and disable primary memory parity generation, interrupts on levels one through three, parity error reporting, DVMA accesses, all interrupts, and boot state. On power-up, the processor board is placed into boot state, with parity generation and reporting disabled, DVMA accesses disabled, and all (internally generated and externally generated) interrupts disabled. When the processor is in boot state, all supervisor program fetches are directed to on-board PROM, regardless of the state of the MMU.

Table 2 indicates the hardware-defined assignment of physical addresses in the Sun 2.0 processor board.

2.3.2 Sun 1.5 Processor

In the current testbed the processors used for peripheral processing elements in each node are modified versions of the Sun Microsystems version 1.5 processor board. (The initial version of the testbed used 1.5 processors for the application processor as well.) The general structure of the version 1.5 processor (as shown in Figure 5) is quite similar to that of the version 2.0 processor, and so only the differences are outlined here.

The version 1.5 processor:

- has 256KB of on-board memory,
- requires one wait-state to access local (primary) memory,
- can have only 2MB of physical (local) memory,
- provides only 2MB of virtual address space per context,
- performs refresh of local memory by software,
- does not allow access to its primary memory from the MULTIBUS.
- uses the Intel i8274 as its DUSART,
- has no battery-backed-up time-of-day clock,
- and has only 32KB of on-board PROM.

Address Space	Size	Address	Device	Wait States Read	Wait States Write
0 (Local Memory)	23-bit	0x000000	Physical Memory (1-8MB)	0	0
		0x700000	Monochrome Frame Buffer	0	4 - 8
		0x780000	Keyboard and Mouse UART	0	4 - 8
		0x781800	Video Control Register	0	4 - 8
1 (On-Board I/O)	14-bit	0x000000	EPROM	2	—
		0x000800	*reserved*	—	—
		0x001000	Encryption Processor	4 - 8	4 - 8
		0x001800	Parallel Port	2	—
		0x002000	Serial Port	2	2
		0x002800	Counter/Timer	2	2
		0x003800	Time-of-Day Clock	12	2
2 (MULTIBUS Memory)	20-bit	0x000000	MULTIBUS Memory (0-1MB)	2 + *device access time*	
3 (MULTIBUS I/O)	20-bit	0x000000	MULTIBUS I/O (0-1MB)	2 + *device access time*	

Table 2: Sun 2.0 Physical Address Assignments

Perhaps the most significant difference between the 1.5 and 2.0 processor boards has to do with the MMU. The version 1.5 MMU supports a linear virtual address space of 2MB, which is divided into 64 segments (each consisting of a maximum of 32KB of memory), which are further divided into pages of 2KB each. A maximum of 16 contexts can be loaded in the segment map table at any one time. In the Sun Microsystems 1.5 processor, the MMU provides protection only on the segment level, and the protection bits are not fully decoded (i.e., not all combinations of read/write/execute permissions can be assigned both supervisor and user modes). Furthermore, the validity indicator is encoded into the protection field making it necessary for system software to save and restore the current protection when pages are temporarily invalidated. The version 1.5 MMU does, however, provide per-page statistics for both page reference and page modification. With the Sun 1.5 processor board, a virtual memory fault can happen because a page map table has not been allocated to the currently executing context (i.e., a *segment* fault), or because no physical page is allocated at the referenced virtual address (i.e., a *page* fault). There is but a single context register that is used in both supervisor and user modes, and the virtual address space usable for memory is limited to 2MB per context because of the fact that virtual address bits from the processor are used to memory map the on-board I/O and MMU state into a common address space. The Sun version 1.5 MMU is illustrated in Figure 6.

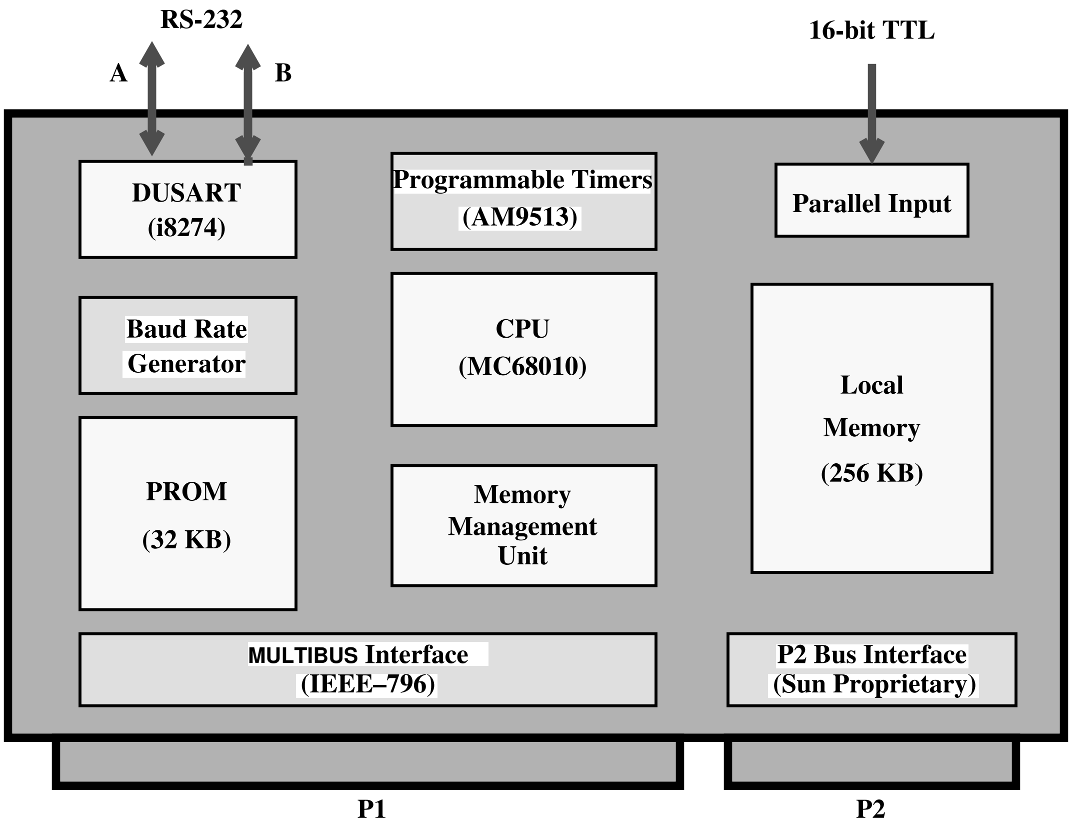


Figure 5: Modified Sun Version 1.5 Processor Board

In summary, the Sun 2.0 MMU has a larger number of both segment and page table entries, it performs protection on a per-page, as opposed to a per-segment level, and the protection field in the page descriptor is fully decoded. In the version 2.0 board, the CPU's function codes are used to support multiple, 16MB address spaces. Also, in the 2.0 CPU, segment descriptors always refer to a valid page map, so segment faults to not occur, but the same behavior may be obtained by having segment descriptors refer to a special page map that has all of its page descriptors marked as invalid.

Table 3 provides a description of the Sun 1.5 MMU address space codes, and Table 4 describes the meaning of its protection field.

It is interesting to note that the Sun Microsystems 1.0 processor board used the MC68000 microprocessor, and its MMU was implemented in such a manner as not to introduce any wait-states to the local memory access time. However, timing changes in the MC68010 microprocessor introduced the need for one wait-state in each memory access on the version 1.5 processor board.

Further details of these Sun Microsystems processors and their MMUs can be found in [Bechtolsheim 82], [Rattner 82], and [Sun 84].

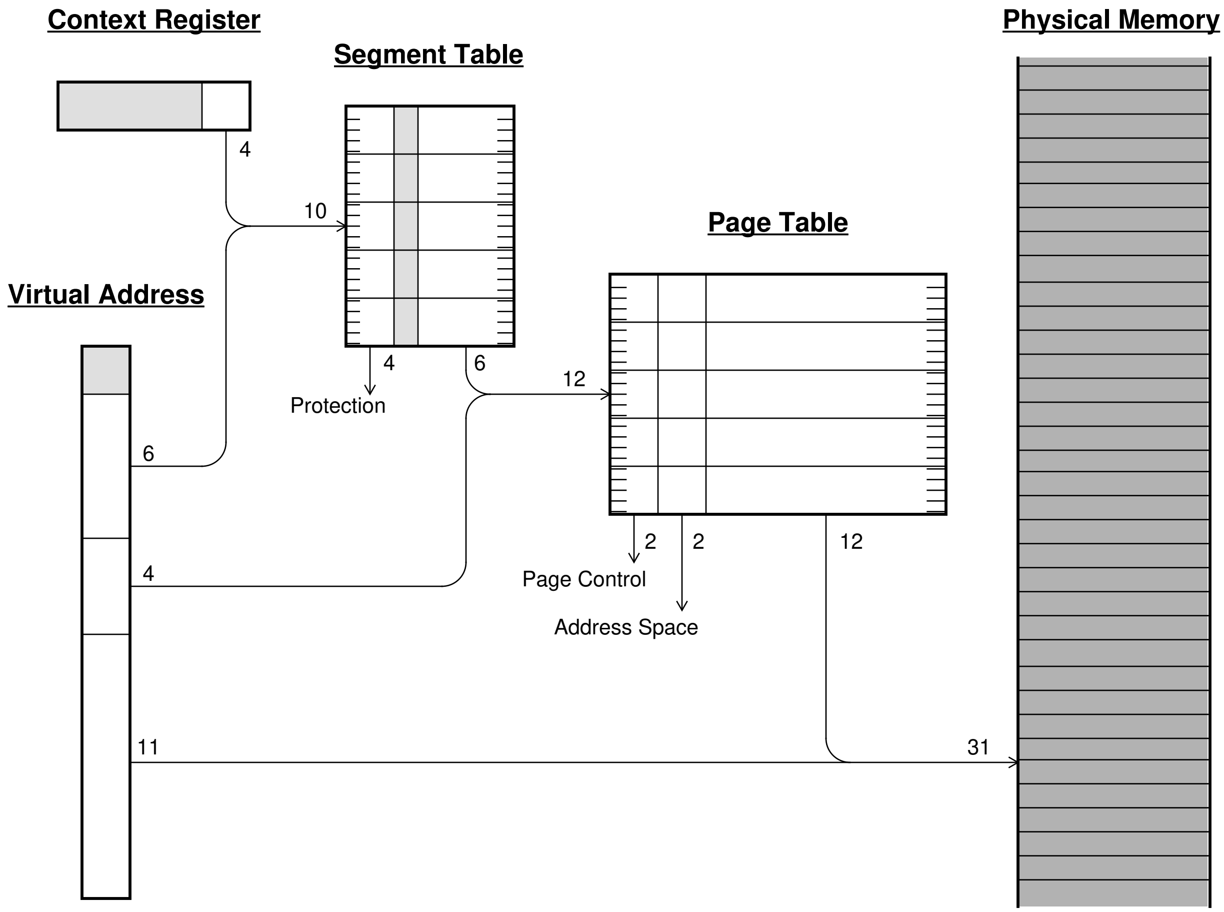


Figure 6: Sun 1.5 Memory Management Unit

Address Start	Address End	Memory Device Description
0x000000	0x1FFFFF	Primary Memory Logical Address Space
0x200000	0x23FFFF	On-Board PROM (Read Only)
0x400000	0x43FFFF	On-Board PROM (Read Only)
0x600000	0x600006	On-Board DUSART
0x800000	0x800002	On-Board Timer
0xA00000	0xBFF800	Page Map Table
0xC00000	0xDFF800	Segment Map Table
0xE00000	0xE00000	Input Port (Read) / Context Register (Write)

Table 3: Sun 1.5 Physical Address Assignments

Code	Memory Access Rights	
	Supervisor	User
0	none	none
1	execute	none
2	read	none
3	read/execute	none
4	read/write	none
5	read/write/execute	none
6	read	read
7	read/write	read
8	read	read/write
9	read/write	read/write
10	read/write	read/execute
11	read/write	read/write/execute
12	read/execute	read/execute
13	read/write/execute	read/execute
14	read/write/execute	execute
15	read/write/execute	read/write/execute

Table 4: Sun 1.5 MMU Protection Codes

3 Experience

In the course of using this testbed a number of (both positive and negative) characteristics of the system have become apparent. This chapter is an attempt at describing some of the more significant discoveries concerning the use of the testbed.

To a very large extent, the development and control portion of the testbed has more than adequately met the project's requirements. The standard Sun Microsystems UNIX-based workstations provide an excellent development environment for the Sun processor-based stand-alone execution environment. The Sun/UNIX operating system is well suited for low-level operating system development work—the system provides a wide range of generally useful tools and provides much in support of the construction of more project-specific software tools.

Neither the development and control, nor the application systems, provided any unexpected results in their use. However, the same is not true of the testbed's distributed computer system component. The following focuses first on the beneficial features of the distributed computer system, and then on its less desirable features.

3.1 Positive Features

Despite the shortcomings of the initial hardware (i.e., the Sun-1 based system), there are a number of hardware features of both processors that proved to be quite useful in the design and implementation of the system, and some of these features contributed significantly to the success of the research being done by the project. In general, the Sun processors, with a 10Mhz Motorola MC68010 microprocessor, custom MMU, and on-board USART and programmable timers, provide an outstanding execution target for doing operating system development work.

The processor's power-on initialization and diagnostic code (contained in the on-board PROMs) help to make the processor more easily usable in a stand-alone context and to speed the development of the low-level kernel code. Furthermore, by providing sufficient PROM space, the processor allows the addition of a number of very large, highly capable programs to the processor's power-up state (e.g., the TFTP program, as described in [Northcutt 88d]).

Furthermore, by taking advantage of the fact that the Sun Microsystems processors have a local memory bus that is separate from the general-purpose, multi-master MULTIBUS, it is possible to construct multiprocessor nodes of Sun processors and other vendors' device controllers. The goals of the testbed would have been severely compromised had the application processor not been equipped with a separate bus for local memory accesses, or if parallelism could not be achieved between the application processor and the peripheral functions.

In order to perform the research work desired by the project, it was required that the testbed application processors provide a highly functional address translation unit. In the case of the Sun processor boards, this need is met by a high performance MMU, supporting demand-paged virtual memory. By making use of the application processor board's virtual memory support, experiments can be performed with the use of separate virtual address spaces to meet the reliability goal of fault containment. Also, the processor's virtual memory hardware provides the opportunity for using optimizations (such as copy-on-write) to increase overall system performance by minimizing the copying of information within a system.

3.2 Shortcomings

The process of developing the Alpha operating system [Northcutt 88a] has revealed a number of problems (ranging from minor to fairly significant) with both versions of Sun processor boards used in the testbed.

Initially, the Sun 1.5 processor was used as the testbed's execution target. From the point of view of this testbed, there were quite a few deficiencies in the version 1.5 processor board. Many of these "problems" were rectified in the version 2.0 Sun processor, however some less than desirable features still remain.

While the bulk of these undesirable features were experienced with the processor boards, a few problems were also encountered with the Sun Ethernet controller board (most of which were due to features of the Intel i82586 Ethernet controller chip).

The following subsections describe the shortcomings that were experienced in the course of using the testbed. Included are problems that were encountered, both large and small, with the Sun 1.5 processor board, the Sun 2.0 processor board, and the Sun Ethernet controller board.

3.2.1 Sun 1.5 Processor

Primarily, the undesirable features of the version 1.5 processor are related to its small virtual address space, its MMU design, its lack of adequate support for multiprocessor implementations, its use of software memory refresh, and its inadequate support for timer functions.

In addition, a few (more minor) problems were encountered in the initial use of these processor boards—e.g., there were errors in some early versions of the processor's printed circuit board that required extensive cut-and-jumper work to enable the processor's multi-master features; and the lack of any provision for per-processor, unique identification bits made this processor difficult to use in a multi-computer/multiprocessor system†.

3.2.1.1 Limited Virtual Address Space Size

The 1.5 processor's restricted virtual address space had a great impact on the initial version of the Alpha operating system's kernel. In fact, the most deeply regretted design decision made in the initial version of Alpha resulted from the (perceived) virtual address space restrictions of the version 1.5 processor. The fact that a major portion of the virtual address space is used to provide memory-mapped access to the MMU and peripheral devices on the application processor board results in a usable virtual address space of only 2MB per context. While this size might be considered adequate for most client contexts, it was thought that such an addressing limitation would not properly allow the kernel to coexist within a context with the client programming units (e.g., *threads* and *objects*, as defined in [Northcutt 88b]).

The decision was therefore made to have the Alpha kernel exist in a context separate from that of the client programming units. Each client context was designed to share a common segment (known as the *context exchange region*) through which all access to the kernel context is mediated. This constraint imposes a cost that recurs each time the kernel is entered or exited (which occurs four different times in the course of a normal operation invocation). Furthermore, some of the kernel's state information was required to be accessible in both the client and kernel contexts. This introduced the additional overhead of adjusting references to parameter pages and the kernel stacks of threads on entering and exiting the kernel.

Thus, as a result of using the processor's physical address lines to address the on-board de-

---

†*The placement of processor identification numbers within the on-board PROMs is considered to impose too great of a maintenance cost to be practical (i.e., it requires a different binary pattern for each set of PROMs, it disallows the use of gang-programming, etc.).*

vices, a significant performance penalty was incurred upon each instance of inter-programming unit interaction (i.e., operation invocation) performed by the client, and the entire structure of the kernel was affected.

3.2.1.2 Memory Management Unit Design

Another feature of the version 1.5 processor that had a significant effect on the initial design of the Alpha kernel is the design of the processor's MMU. The version 1.5 processor's MMU was designed to provide protection only at the segment level. This does not allow the fine degree of protection control that is needed for doing various per-page virtual memory operations (e.g., copy-on-write pages). The fact that protection in this processor is on a segment (as opposed to a page) level makes for a number of difficulties, such as: inhibiting the ability to protect code and data that may be combined into a single segment, being unable to adequately protect low-core, and not being able to do efficient, page-level protection of such structures as client stacks within thread contexts.

The segment descriptor protection field in the 1.5 processor's MMU is encoded, limiting the possible protection modes and requiring the explicit saving and restoring of the protection field if the segment should be temporarily invalidated. Another difficulty with the 1.5 MMU has to do with the fact that segment descriptors are not validated on page map cache accesses. This means that additional kernel complexity is introduced to deal with the fact that a segment fault will not occur if a segment descriptor is invalidated between validating a segment descriptor and accessing one of its page descriptors. Furthermore, the small number of page descriptors that can be loaded into the 1.5 MMU's page map table introduces another limited resource whose efficient management is critical to the overall system performance.

The version 1.5 MMU also does not allow the processor to access page map or segment map entries that belong to another context. This means that all such manipulations must be performed from within the target context, requiring that the cost of entering and exiting another context be paid each time such a manipulation is to be performed. In the case where large portions of a context are to be remapped from within another context, the cost of performing context switches is considerable. Also, because the version 1.5 processor board periodically asserts the MC68010's non-maskable interrupt, the kernel must ensure that the system stack pointer refers to a valid page at all times. Thus, each context swap must be accompanied by the movement of the system stack into the destination context.

3.2.1.3 Support for Multiprocessor Configurations

Some complications in the initial design of Alpha's kernel resulted from the fact that the testbed processors were not designed to function in a multiprocessor context. The version 1.5 processor does not allow direct access to its memory, MMU, or on-board devices from the MULTIBUS. With the Sun version 1.5 processor, information to be exchanged with the other hardware functional units in a node must be placed in shared MULTIBUS memory or explicitly moved by the application processor. This introduced the need for additional copying of information between the processors in a node, incurring performance degradation with each access to shared MULTIBUS memory, as well as the additional complexity of having to jointly manage a limited amount of non-local memory among the processors in the node.

A further effect of the fact that the application processor was not designed for multiprocessor use is the lack of a means for the generation of MULTIBUS interrupts. While the processor does respond to bus-vectored interrupts on the MULTIBUS, it does not include a mechanism for generating them. Without this function, the processors in a node must communicate via the polling of shared memory locations—a technique that is more costly and less responsive than an interrupt-based interface.

3.2.1.4 Software Memory Refresh

The version 1.5 processor implements the refresh of its local memory in software, as opposed to the more traditional use of hardware to transparently refresh dynamic memory. While this function in itself did not consume many processor cycles, the indirect costs of software memory refresh were significant. The software memory refresh function required that a non-maskable interrupt be generated periodically, and the ever-present nature of this interrupt incurred a cost in terms of both complexity and performance.

Because non-maskable interrupts are guaranteed to be generated periodically, a number of precautions had to be taken in the initial Alpha kernel code. These precautions imposed a fixed overhead cost in order to deal with an infrequent event. Each time the processor's context was changed, special care had to be taken to ensure that the system stack pointer would be valid in the destination context. This was because interrupts required a valid system stack, or the processor board would enter a *hung* state (upon the occurrence of the non-maskable interrupt) that could only be terminated by power-cycling the processor. Thus, the ability to disable all interrupts for a short period of time would have proved to be a very desirable function.

Despite the fact that the software refresh of local memory costs very little in terms of processor cycles, the potentially large hidden costs associated with this technique make it a poor design decision. While the cost of doing the actual refresh is small (i.e., 128 NOP instructions every 4msec), it incurs a significant cost as a result of synchronization demands imposed on the kernel (i.e., approximately 200usec on each context switch or operation invocation).

With a small amount of additional hardware it would be possible to provide hardware refresh of the local dynamic memory, limit the use of the processor's non-maskable interrupt to such functions as power-fail or external reset, and simplify the internal structure of the Alpha kernel.

3.2.1.5 Programmable Timers

The programmable timer used on the version 1.5 processor also had a number of features that made it difficult to deal with it. The design of this timer is such that reading the state of a counter within 70nsec of a clock pulse caused undefined data to be read. Given that the processor and the Alpha kernel both clock these timers at high speeds (i.e., 200nsec), this may occur with great frequency. This requires that all timer read operations must disable the clock, read its value, and then re-enable, which both adds to the cost of timer manipulations and lessens the accuracy of the timer by allowing clock pulses to be missed. Also, the internal registers of this timer are multiplexed into a two word window by selecting the desired registers in one command and manipulating the selected register in another command. However, because interrupts may occur between the selection and access operations, great care must be taken in the manipulation of the timer, thus adding further to the overhead of performing timer operations, while reducing the responsiveness of the system.

The version 1.5 processor used a separate timer for each of the two USART channels, thus leaving only one timer (out of five) for the user. This does not provide adequate hardware timer support for the timed event and performance monitoring functions in Alpha. This required that the user timer be shared among all of the diverse functions in the kernel, at costs in complexity and performance.

Additionally, the lack of an external reset pin on the timer makes it impossible to perform a complete external hardware reset of the application processor. If a hardware reset is issued while the dynamic memory refresh timer is running, the processor enters a hung state and must be power-cycled to be reset.

3.2.2 Sun 2.0 Processor

With the release of their Sun-3 system, Sun Microsystems made available to the project the detailed technical information needed to use the Sun-2 processors within the testbed. At that point, the testbed was modified to include a single Sun version 2.0 processor in each processing node. The Alpha kernel was then modified to use the 2.0 processor as the application processor, with the 1.5 processors being used as peripheral co-processors. The fact that there was little difference in the fundamental structure of the two processors, coupled with the fact that the initial design of Alpha took into account the eventual move to Sun 2.0 processors, made the transition a fairly easy one.

Most of the problems that were encountered with the version 1.5 hardware were rectified in the version 2.0 processor board, including: the elimination of timing problems, allowing zero-wait state access to physical memory (maximum 8MB); the use of function codes and an increased number of segments per context (i.e., 512), to provide a full (i.e., 16MB) virtual address space for each context; the use of hardware memory refresh, thereby eliminating the need for periodic non-maskable interrupts; the inclusion of fully decoded, page-level protection; and the support of DVMA, allowing arbitrary segments of the processor's virtual address space to be made available to MULTIBUS masters.

Among the drawbacks that the project discovered with the Sun Microsystems 2.0 processors were that there is no on-board memory (additional boards must be connected via the processor's MULTIBUS P2 connector) and only 8 contexts can be mapped in the 2.0 MMU at any one time (as opposed to 16 in the 1.5 MMU). The Sun 2.0 processor still does not go very far in its support of multiprocessor configurations. In particular, the Sun 2.0 processor board generates bus faults whenever the read-modify-write cycles it issues on MULTIBUS locations coincide with the board's memory refresh cycles. This causes unnecessary complexity and overhead to be added to the process of synchronizing access to shared storage by multiple processors in a node. While the 2.0 processor does include a system register that allows certain MULTIBUS interrupt lines to be asserted, this function is inadequate for the more general demands of the Alpha testbed—the assignment of the interrupts cannot be modified, and the register is not accessible by the other processors in the node.

3.2.3 Sun Ethernet Controller Board

Along with the change to Sun version 2.0 application processors, the node's (quite simple) 3-Com Ethernet controller boards were replaced with the (somewhat more sophisticated) Sun Microsystem's Ethernet controller. A few (more or less, minor) problems were encountered with the use of the Sun Microsystem's Ethernet controller. Many of the problems with this board stem from the Intel i82586 Ethernet controller. Among the most troublesome of this part's (rather substantial) list of problems is the fact that DMA under-run errors frequently occur when short buffers are encountered in a packet's transmit list. The result of this feature is that unnecessary packet retries are performed whenever small transmit buffers are found (which occurs commonly in Alpha where the encapsulation of packets for various protocols is implemented by way of separate transmit buffers).

Furthermore, the nature of the i82586's multicast mechanism makes it unsuitable for use with the type of dynamic, logical addressing used in Alpha. The multicast mechanism was intended to be used with a small, static set of addresses and acts only as a filter, relying on software to ultimately determine whether a multicast packet should be received. When the set of multicast addresses recognized by a controller is to be changed, the controller effectively disables multicast address recognition for period of time (in excess of several minimum packet length times). The low level use of logical addressing in Alpha requires that the set of multicast addresses recognized by a controller be changed dynamically, and so the frequent occurrence of disabled packet reception significantly degrades the communication's processor's ability to function properly.

A problem with the design of the controller board itself lies in the inability of the controller to directly access the application processor's memory. This requires memory copies be performed between the application processor and the controller's buffer memory, despite the fact that the i82586 is capable of scatter/gather memory accesses. The performance of the Alpha kernel's communications subsystem is made to suffer because of the need to copy all information to and from the Ethernet controller.

Another problem with the board caused a slight disruption of interrupt level assignments. A misguided optimization in the printed circuit board etch resulted in making it impossible to use the board's interrupt jumper block to make the controller generate interrupts on any level other than three. Furthermore, the fact that the offending signal is on an internal layer of the board makes it quite difficult to make standard cut-and-jumper modifications. As a result, the controller was left at interrupt level three and other interrupts were assigned as needed to other interrupt levels.

4 Potential Improvements

Based on the project's experience with using the testbed, there are a number of changes that could be made to the development/execution environment interface software, the processing node structure, and the individual processors, in order to have the testbed better support the project's needs.

Improvements could be made at all levels of the testbed system. The system level of the testbed environment (as described in [Northcutt 88d]), could be substantially improved by the inclusion of a native, stand-alone, source-level debugger, along with more sophisticated performance evaluation and monitoring tools. At the node level, the single Ethernet currently used as the inter-node interconnection network could be replaced with a replicated set of higher bandwidth buses that use a priority-based arbitration scheme. Similarly, the inclusion of higher bandwidth intra-node backplane would represent a significant improvement to the testbed. Despite all of this, the area where the greatest improvement could be made is with the application processors.

One way in which the testbed might be improved for use by the Alpha operating system effort is by adding more than one application processor to each node. The Alpha programming model was designed explicitly to cope with the problems of multiprocessing. However, by having only a single application processor in each node, the testbed has limited the project's ability to explore this area. While it might be possible to perform such an upgrade in the future, there are no plans for this because the bulk of the multiprocessor work is being performed in conjunction with Alpha release 2.0 (and beyond), targeted to a symmetrical multiprocessor [Northcutt 88c]. For future releases of the Alpha operating system, an execution environment consisting of more symmetrical multiprocessors is assumed. The discussion here focuses on the implications of the testbed hardware on the research performed to date on Alpha.

4.1 Application Processors

One major addition required by the Alpha operating system, for both types of processors in the testbed, is hardware support for floating point operations. This function was not included in the testbed's original requirements because it was thought that operating system's do not need floating point support. However, as work with the Alpha system progressed, it was discovered that the algorithms used to perform time-driven resource management required floating point for dynamic range (i.e., to support a wide range of time constraints, from very short to very long), for computing probability distributions, and for the scaling of families of curves to fit the requirements of time-value functions [Northcutt 88b]. Furthermore, in the course of implementing the project's command and control demonstration [Maynard 88], the application's need for floating point support became apparent.

Although many of the negative aspects of the Sun Microsystems version 1.5 processor board have been corrected in the Sun Microsystems version 2.0 processor board, not all of the desired features have been incorporated into that design. The testbed currently uses Sun version 2.0 processor boards as the nodes' application processors. The following subsections describe modifications that could be made to the Sun 2.0 processor to better meet the needs exhibited by the Alpha operating system. The Archons project is currently involved in research efforts to provide hardware support mechanisms in each of the areas described below.

4.1.1 Memory Management Unit

Both types of MMU in the Sun Microsystems processors used in the testbed are less than ideal for Alpha in that they do not provide support for the Alpha programming model. In particular, the Sun MMUs are not hardware address translation caches, but rather a set of hardware lookup tables that must be made to function as a cache by the system software. Greater system performance and a reduction in the complexity of the virtual memory portion of the Alpha kernel could be attained through the use of a hardware address translation cache. The Alpha operating system would benefit from a modification to the Sun MMUs that would allow the thread and object portions of a context to be switched independently, in order to better support the type of partial context swaps performed by the kernel on operation invocation.

An example of an MMU that is both a real hardware cache and supports the kind of partial context swapping that would be useful in Alpha can be found in the Digital Equipment Corporation's VAX processors (with a translation lookaside buffer and the separate *System*, *P0* and *P1* regions). This type of MMU would incur a significantly greater cost in terms of hardware complexity, but would greatly reduce the complexity of the kernel software in the area of virtual memory management and would increase the performance of the system by decreasing the cost of operation invocations.

Furthermore, the Sun MMUs use a two-level paged scheme that is less desirable, in the case of Alpha, than a base/limit segmented scheme which provides greater flexibility in manipulating portions of address spaces. An implication of the Alpha system's robustness requirements is that objects exist in private address spaces (see [Northcutt 88a] for further details). Given this design decision, it is incumbent on the implementer to provide an efficient means of passing parameters between objects in separate address spaces. In addition to this, the Alpha system's reliability requirements have resulted in a design that includes mechanisms for the support of kernel-level atomic transactions. Specifically, a write-ahead-log mechanism is associated with the kernel's data locking primitives in order to control the visibility of changes to objects made by threads executing within atomic transactions. Both the inter-address space parameter passing and the lock/write-ahead-log mechanisms can best be supported by modifications in the processor's hardware that permit the system software to indicate that a trap to the kernel should occur on certain forms of accesses to the locked data.

Hardware support for these functions could be provided by an MMU that allows arbitrary portions of an object's address space to be defined as segments. In this way, the parameters of local operation invocations could be multiply mapped (in both the invoking and the invoked objects' contexts) and marked to be copied when the data is modified from either context (i.e., the segments are marked copy-on-write). Similarly, the data defined in lock operations could be marked so as to cause access faults or blocking when a thread accesses a region of locked data, or to create a write-ahead-log (on demand) when the locked data is written by the thread holding the lock.

Note that while this functionality can be supported by a true segment-based MMU, it can be thought of as orthogonal to the memory translation function, and implemented as a separate hardware unit from the MMU. This is the approach currently being explored by the Archons project.

4.1.2 Multiprocessor Interface

A very useful modification would be to make the local memories of all of the processing elements within a node be dual ported—i.e., accessible both by its local processor and via the MULTIBUS. With such a modification, the amount of information that is currently passed between the Alpha's kernel proper and its subsystems would be drastically reduced. Several performance restricting features of the current testbed could be eliminated if modifications were made to allow the application processor and the peripheral device processors within a node to share memory in a more symmetrical (and reasonably high-performance) manner. Duplication of data structures and unnecessary information exchanges could be eliminated, and information to be exchanged among the various components within a node could instead be accessed directly and shared by way of manipulating memory mappings (as opposed to physically copying).

With these modifications, it is important that all of the external memory references to the processor's local memory are made through the processor's MMU. Otherwise, a great deal of logical complexity would be introduced in order to deal with the fact that external references are made to physical locations, while internal references are made to virtual locations. Also, much of the benefit of sharing and moving blocks of memory by address mapping would be lost if the MMU were not in the memory addressing path for external references. The problem of ensuring the coherence of the local memories in such a multiprocessor system cannot be overlooked—the suggested modifications, in effect, call for a true symmetric, shared-memory multiprocessor, with all of the attendant functionality and complexity. In addition, it is important that the multi-porting of local memories be done in such a fashion as to allow external access to the memory's full address range and that the impact on the local memory access time be minimized (i.e., either have sufficient bandwidth to allow zero wait-state access or resolve access contention in favor of the local processor). Note that the discussion here is directed towards an approach similar to that on which the current testbed is based (i.e., where the processors within a node have local memory and share common memory). However, this discussion applies equally well to more traditional multiprocessor architectures where the local memories are in fact caches, and the common memory forms the bulk of the node's primary memory resource.

At the very least, both versions of the Sun Microsystems processor boards could be redesigned to perform better in the type of shared-memory multiprocessor nodes used in the current testbed. For example, the use of an intra-node bus with an interprocessor mailbox mechanism (e.g., MULTIBUS II or VAX-BI) would be a useful addition to the processors in a node. In lieu of this, the least that is needed is for all of the processors in a node to be able to generate, clear, and (locally) mask MULTIBUS interrupts, all under software control.

4.1.3 Hardware-Supported Timers

In Alpha, the overhead associated with the manipulation of the processor's on-board timers was found to be rather significant. The kernel requires hardware supported timers for many reasons. For example, in Alpha a *system timer* was used to: create timestamps for logged events; provide the kernel with a node-local time-of-day reference for use in performing time-driven resource management functions, providing client-level time services, and creating a globally consistent time reference among the nodes in the system; and as a non-reusable, monotonically increasing counter for use in the distributed, unique identifier generation algorithm (i.e., concatenating the time with a unique node identifier).

There are two inter-related issues involved with system timers, namely *granularity* (i.e., the accuracy that can be achieved by the timer with respect to continuous real-time) and *range* (i.e., the span of time that a timer can cover without repeating counts). In a physical implementation, these characteristics of timers translate to the speed at which the timer is clocked and the size of the timer (in number of bits).

While the Sun processor boards provide a greater degree of timer support than the more traditional, periodic timer (e.g., a line-time clock interrupt), still more is called for in Alpha. The

Sun version 1.5 processor provides only a single 16-bit counter that could be clocked at a rate from 200nsec to 64msec, while the version 2.0 processor provides a total of three usable 16-bit counters that can be clocked at the same rate. The 200nsec clock rate provides sufficient granularity to meet the needs of Alpha. However, in order to keep the system timer from rolling over during the node's expected up-time it is necessary to cascade multiple of the 16-bit timers to accommodate the necessary time range (with three cascaded timers a maximum node up-time of approximately two years can be achieved at the highest level of granularity).

In Alpha there is a need for a continuously running, long-term, high-resolution system timer that runs without the need for software intervention, even when the processor's power is interrupted (e.g., a battery backed-up, 64-bit counter, clocked at 200nsec). This need is approximately met in the Sun version 2.0 processor by a battery backed-up time of day clock, however the integrated circuit is difficult for the software to use, its counters are structured in such a fashion as to make it difficult to extract desired components of the counter, and the circuitry suffers rather severely from temperature-induced drift. It should be noted here that it is not sufficient to obtain extended range from a hardware timer with software (i.e., by having the hardware timer generate an overflow interrupt and cascade the count into a memory location). Such an approach either requires that the overflow interrupt be non-maskable by the processor, or that the system timer be non-monotonically increasing in value (i.e., the timer will be invalid for the period between the overflow interrupt and the incrementing of its software component). Both of these problems stem from the fact that the input clock-derived increment operation on a hybrid hardware/software timer is not atomic with respect to the execution of the system software, and both of these problems are unacceptable in systems such as Alpha.

In addition to the need for such a system timer, various subsystems in Alpha require hardware support for *interval timers*. The communications subsystem uses timer-based protocols and so there is a need for setting a timed event (i.e., a timer that will generate an interrupt after a given period of time has elapsed), and for cancelling previously set timed events prior to their deadlines. To indicate at what future point in time thread dispatch operations are to be performed, the scheduling subsystem must set timed events and then clear them when circumstances cause the computed schedules to change. The same need for high granularity and wide range that was indicated for system timers exists for the interval timers as well.

In Alpha it was determined that the cost for adding and removing timed events and maintaining the interval timer in general (i.e., scaling, setting, clearing, and handling interrupts from the timer) was rather significant, and a hardware-based solution would be valuable. A hardware structure that may serve this purpose could be constructed in conjunction with the system timer described above. The interval timer adjunct could be based on an associative memory array into which the (node-local) absolute times of the desired events could be written (perhaps with a small, user-defined, key value). The associative array is continuously presented with the current value of the system clock, and when one or more matches occur, an interrupt would be generated to the processor and a status indication would be provided to indicate the timed events whose deadlines have passed. Because of the inevitable size limitations of such a structure, software optimizations would have to be applied to cache the events with the nearest deadlines in the associative array, keeping the others in standard memory until their window approaches. Such an optimization would also allow the simplification of the associative timer array, in that it would only have to evaluate a fixed-size, sliding subrange of the system time values.

4.1.4 Additional Processor States

The design of the Alpha kernel incorporates a feature that permits the structured extension of the basic functionality of the kernel through the use of kernel threads and kernel objects [Northcutt 88b] within the operating system itself. This design, plus the fact that the kernel level of the Alpha operating system provides a high degree of functionality, results in a great deal of logical structure being applied at the lowest levels of the Alpha operating system. In order to carry this logical structure through into the implementation, while retaining a high degree of efficiency, it is necessary that the processor hardware provide some degree of support.

In order to properly support Alpha's kernel thread and kernel object scheme, it is necessary to have more than the two states (i.e., *user* and *supervisor*) that the MC68010 microprocessor provides. This is because the kernel objects and threads execute within the kernel's context, in supervisor mode, and need to provide stack protection and dynamic stack extension (similar to what is needed by client threads and objects). However, with just the two machines states currently available, should a fault occur on a system stack access, there is no other machine state (with an associated stack) that the machine can switch to in order to handle the fault.

While it is not necessary that the hardware support an arbitrary number of protection levels (such as the protection rings in the Honeywell Multics machines), it is important that the processor hardware support some degree of structure within the protected level. For example, the four machine states provided by the Digital Equipment Corporation's VAX processor would do well in supporting Alpha's needs in this area.

4.1.5 Protected Code Entry Points

Another example of where the Alpha kernel could be supported by modifications to the testbed hardware is to be found when attempting to carry the notion of object operations down to the hardware level. It is possible to enforce in hardware the Alpha programming model's concept that operations provide the only execution entry points for an object, and even validate that the invoking entity has the proper privileges to do the invocation.

The protection of code entry points is currently provided by the Alpha kernel's software, and could be supported more effectively in hardware. The processor hardware could be modified to provide protection to the entry points of objects (similar to the *gate* concept found in the Intel i80286), which would enforce the programming model at a low level within the system and could provide a minor performance increase for operation invocations in Alpha.

4.2 Peripheral Subsystem Controllers

The anticipated need for hardware-supported distributed system components was validated in the Alpha operating system work done on the testbed. It is clear that practical distributed systems depend on hardware support for concurrent system functionality. The peripheral system functions that called out for the greatest degree of hardware assist in Alpha were related to the communications and secondary storage subsystems. While the testbed provided full processing elements (i.e., processor/memory pairs) for each of these subsystems, in addition to their basic device controllers, what is really called for are instances of the currently available class of intelligent peripheral controllers. For example, what is needed for the communications subsystem is an Ethernet controller board with on-board CPU, with local memory and full access to the application processor's memory, as well as the state of the scheduling co-processor.

In the current implementation of Alpha, outgoing remote invocation parameters (and scheduling information) are mapped onto the MULTIBUS via the application processor's DVMA mechanism, and then copied into the Sun Ethernet controller board's buffer memory. On incoming remote invocations, the parameter information must be copied from the Ethernet controller's buffer into a page in the application processor's local memory. In both of these cases, the Alpha kernel must manage the use of the Sun 2.0 processor's limited size DVMA window, and a substantial amount of memory copying must be done. Ideally, the Ethernet controller would be able to perform direct scatter/gather operations on the application processor's local memory—i.e., read and write memory in place, without copying. An enhanced communications controller would also support logical addressing on the network, instead of the single hardwired station address (and an essentially static, multicast filter) that is provided by the current, dumb Ethernet controller. The ideal solution would be to have the communications controller provide an associative name table that is used to accept packets from the network, and can be accessed directly by the kernel executing in the application processor. This would require true, multiported access to the name table so that there is no interruption of network service while names are added and deleted, and so that the kernel can perform associative name lookup operations in order to determine if a given entity is local to the node.

Similarly, the storage management subsystem requires an intelligent disk controller that can scatter/gather pages from the application processor's local memory. This controller requires the same degree of local autonomy and buffering that was indicated for the communications subsystem; neither controller should require that a processor (and the node's bus) be in use throughout an entire I/O transaction. Also, local buffering is needed to smooth the flow of data, because once the data starts coming off either a disk or a network, it must be stored somewhere or be lost. In either case, it is ideal to store the data where it can be placed into its ultimate location (typically within the application processor's local memory) via memory mapping operations (as opposed to copying).

The requirements described here are for the system's peripheral subsystems, but in general they apply as well to any I/O controller that an application may require be added to a node.

4.3 Scheduling Subsystem Co-Processor

In Alpha, one processing element in each testbed node is dedicated to performing the time-driven management of application processor cycles. This is known as the scheduling subsystem co-processor and it imposes a number of significant requirements on the hardware.

The current scheduling algorithm requires the functionality of a general-purpose, programmable processing element for determining the execution sequence and timing of threads, concurrently with thread execution on the application processor. The work being done in Alpha on higher-level, semantic-driven scheduling also requires concurrent, general-purpose computing capabilities.

In addition, the algorithms being experimented with in Alpha for time-driven resource management demand that this subsystem provide hardware support for floating point operations. This is necessary in order to cope with the wide dynamic range of time-value functions that may be specified as attributes of threads executing on Alpha. Measurements of the Sun 1.5 processor indicated that the software implementation of a simple "A = B * C" statement takes 250msec to execute, and many such floating point operations are required for the evaluation of the time constraints that is performed following each resource management event.

The *best-effort* scheduling algorithm in Alpha is quite computationally complex (in its naive implementation), therefore a functional unit is called for (similar to one described in [Jensen 75]), that provides highly parallel hardware augmentation for the system's time-value function evaluations. While this functional unit is described as being an adjunct to the scheduling subsystem co-processor, it would also be useful to the other system resource management facilities within Alpha, and so it should be made available (much like a conventional floating point co-processor) to other parts of the kernel software. In this manner, any kernel subsystem (e.g., the virtual memory or secondary storage) could use this functional unit to evaluate the time constraint attributes of a set of threads to determine if the demand for a resource by a given set of threads (with time constraints) exceeds its supply, and if so, how the contention should be resolved (i.e., according to some application-specified policy)

Currently in Alpha, the scheduling co-processor interacts with the kernel proper (running on the application processor) via message passing, and must maintain local replicas of the kernel's internal data structures (i.e., thread control blocks). The duplication of state information, the amount of information exchanged, and the number of interactions would be greatly reduced if the scheduling co-processor could directly access the application processor's local memory. A superior approach would involve a modification to the testbed hardware to allow the scheduling co-processor concurrent access to the application processor's primary memory (via its MMU). In this way, the scheduling co-processor could directly access the kernel's data structures and it could maintain the ordered scheduling list directly in the application processor's memory. This requires a more general and symmetrical processor/co-processor relationship than is found in typical co-processor interface schemes (such as that found in the Motorola MC68020 family).

In Alpha, the scheduling subsystem co-processor handles all blocking/unblocking requests from the application processor, in order to perform higher-level semantic scheduling functions. Also, the scheduling co-processor is currently able to interrupt the application processor in order to force a context swap. A hardware modification could be made to the system so that all of the application processor's interrupts are handled by the scheduling co-processor. If this were the case, interrupts would be immediately converted into thread unblock operations so that the Alpha system's global time-driven resource management facilities could determine the proper time to handle the requests associated with hardware interrupts. In this way, it is possible to manage all of the system's resources in a timely and consistent fashion.

Figure 7 provides a diagram of the structure of an idealized node, that incorporates all of the modifications suggested in the preceding text.

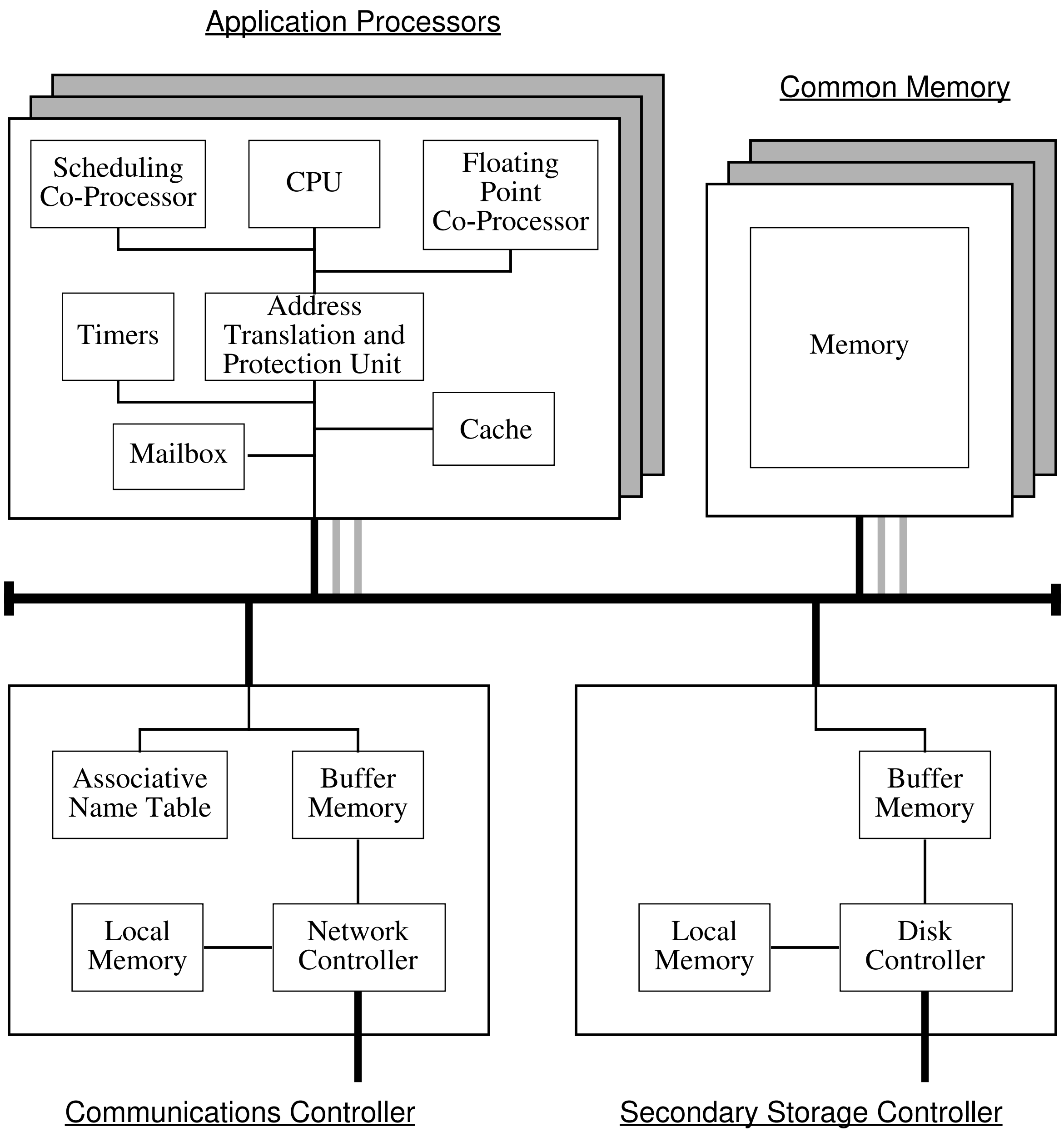


Figure 7: Idealized Testbed Node Structure

5 Hardware Modifications

In the process of constructing and using the current testbed, modifications have been made to virtually all of the major components in the testbed's distributed computer system. The following sections describe the changes that were made in the testbed's application processors and the peripheral co-processors, the daughter-board created to provide the processing elements with hardware support for floating-point, and the various signal interconnections and reassignments performed within the testbed's nodes. Detailed descriptions of these modifications are given in Appendix II.

5.1 Application Processors

A major constraint on all modifications made to the Sun version 2.0 processor boards was that the changes must be completely (and simply) reversible, and that none of the changes should interfere with the board's ability to run standard UNIX.

The only change made to the hardware of version 2.0 processor boards was related to the addition of an interprocessor interrupt generation unit. It was clear very early in the design of the testbed that a need existed for all processing elements to be able to generate and clear the global MULTIBUS interrupt lines (preferably, in a uniform fashion). While the Sun version 2.0 processor board contains a register that permits a subset of the MULTIBUS interrupts to be generated, this function cannot be made directly available to the other processors within a node.

Therefore, the interprocessor interrupt (IPI) generator was constructed as an add-in module that occupies an unused 40-pin socket on the processor board. This device acts as a memory-mapped register that permits the setting and clearing of any of the MULTIBUS interrupt lines. Furthermore, a modification was made to allow this device to be mapped onto the MULTIBUS via DVMA, so that all processing elements within a node can have access to it. A modification was necessary because the 2.0 processor only permits local memory to be mapped via DVMA, due to Sun's inability (and lack of desire) to cope with the problem of the asynchronous acknowledgment of accesses. To overcome this limitation, one of the processor's (socketed) PALs was modified to generate the proper data transfer acknowledge (DTACK) signal for the CPU.

This entire modification involves changing a socketed PAL, plugging the IPI module into an empty socket, and inserting interrupt lines into the desired slots. This change in no way affects the ability of the processor to run standard Sun distributions of UNIX.

In addition to these hardware changes, the processor's firmware was modified to provide the functionality described in [Northcutt 88d], and the boards were configured (via jumpers) for multiple bus master operation (see Appendix I).

5.2 Peripheral Co-Processors

The modifications made to the Sun version 1.5 processor (used by Alpha as a peripheral co-processor) include: configuring the processor to function properly in a multi-master environment, adding a baud-rate generator for the on-board USART, reassigning the on-board timers, upgrading the on-board memory from 256KB to 1MB, and modifying the PROMs to enhance the monitor's functionality.

While Sun had designed the version 1.5 processor boards to work in a multi-master environment, configuring the boards for multiple bus master operation involved the cutting of printed circuit traces and the addition of plug headers and jumpers, and in some cases additional modifications had to be made to compensate for errors in certain etch revisions.

The major change made to the 1.5 boards had to do with the reassignment of the on-board timers. In their original configuration, the 1.5 processor boards had only one 16-bit timer available for the programmer to use (i.e., the *user timer*). The other four timers were used by the system for such functions as watchdog timers and baud-rate generators for the serial interfaces. To meet Alpha's timer needs, a pair of timers were reassigned so that the timer generating clock signals for the USART's B port could be used by the Alpha software. A later change was made to free the two timers used for the USART's clock generation. This was done by adding a baud-rate generator integrated circuit to the processor board. In addition, the terminal count output (which indicates the expiration of a timed interval) of one of these newly freed timers was wire-or'ed with the user timer's output in order to allow two separate 16-bit timers to generate interrupts to the CPU (which could then be demultiplexed in software by interrogating the timer's status registers).

Another modification to the 1.5 processors was the upgrade of its on-board memory from 256KB to 1MB. This was accomplished by replacing the 64Kb memory components with 256Kb ones, adding drivers for two additional memory address lines, and changing the firmware to refresh the memory array at a different rate (i.e., 4msec, as opposed to 2msec). This memory upgrade was performed to provide additional on-board local memory for the subsystem co-processors, without incurring the expense (in terms of power, heat, backplane slots, and P2 connectors) of additional memory boards.

Also, the processor's MULTIBUS interrupt lines were remapped (via jumpers) to permit each co-processor to receive a different interrupt line from the MULTIBUS on its processor level 4 line (see Table 5).

Finally, a custom-constructed plug was added to each processor in a testbed node to provide a unique node and processor identifier. This was done because the version 1.5 processor board does not have any means of identifying the board (unlike the version 2.0 processors, which have identification PROMs). In order to provide a means of obtaining a unique node and processor identifier for each processor, a shorting plug was placed into the input port connector on each processor. The input port has resistor pull-ups on each bit so it is possible to create the desired bit pattern by either shorting an input line to ground or allowing it to be pulled high. For Alpha, an encoding scheme is used that allows one nibble to represent the node and the other nibble to represent the processor (where the application processor is 0, the scheduling co-processor is 1, the communications co-processor is 2, and the storage co-processor is 3). Additionally, the PROM monitor was modified to print the contents of the input port as part of the processor's reset banner.

5.3 Floating Point Accelerators

In the course of developing the Alpha kernel, and in developing the initial command and control application, it was determined that some form of hardware augmentation for floating point operations was required. Based on this need, a daughter-board was designed and implemented that provides hardware floating point support in a drop-in package that replaces the MC68010 (in either the 1.5 or 2.0 versions of the processor) and executesthe standard code generated by the Sun Microsystems' C compiler.

This add-in board (transparently) replaces the processor's MC68010 CPU with a MC68020/ MC68881 (CPU/FPC) pair (as represented in Figure 8). The MC68020/ MC68881 combination was used because this is the same component set used in the Sun-3 workstations that are used in the testbed's development system, and so the hardware could be made to properly execute the normal and floating point co-processor opcodes emitted by the Sun-3's C compiler.

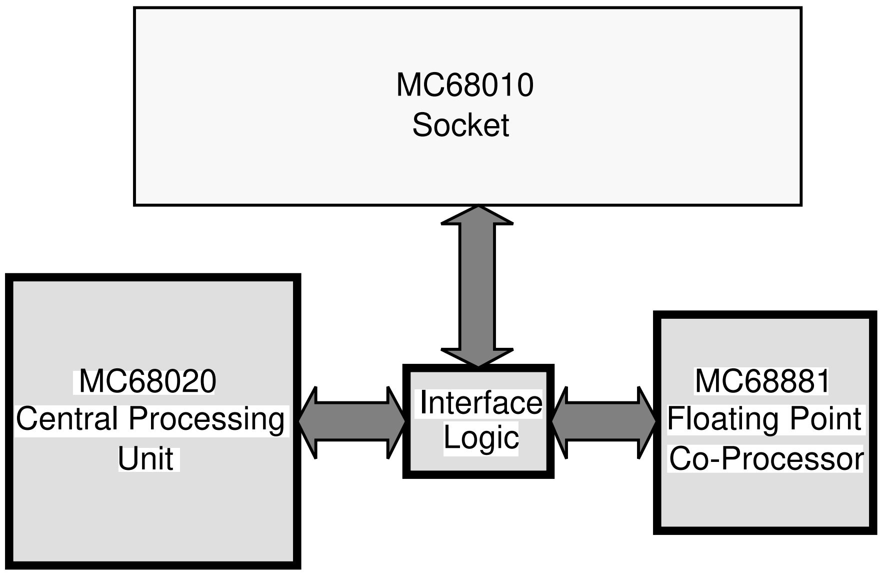


Figure 8: Floating Point Accelerator

This solution applies to both version 1.5 and version 2.0 Sun processor boards (requiring only a change of compiler switches), and only minor modifications had to be made to the math function library to allow the more complex functions to work in stand-alone mode. By and large, the existing binary files created for the original, MC68010-based processors are able to execute on the MC68020/MC68881 board without modification.

5.4 MULTIBUS Interrupt Assignments

To provide the maximum flexibility with the minimum number of modifications and the greatest amount of uniformity, the MULTIBUS interrupt signals are assigned in a particular fashion for each board in a node.

Among the considerations for choosing these assignments was the desire to permit all processors within a node to receive interrupts from the Ethernet controller on the same interrupt level. However, because of printed circuit board etch problems with the Sun Ethernet controller board, this interrupt was constrained to be at MULTIBUS level three.

Each processor in the node receives interrupts from the Ethernet at CPU level three, interrupts from the interprocessor interrupt generator at CPU level four, and their on-board device interrupts at CPU levels five through seven. In this way, each processor can monitor its own interprocessor interrupt line, as well as the interrupt line from an associated device controller, and mask out all unwanted signals. This was done by using hardware masking of interrupts for co-processors, and performing no hardware masking of interrupts for the application processor (which can mask unwanted interrupts in software). Since the Sun 2.0 processor board places its on-board interrupts (i.e., from the timer and USART) onto the MULTIBUS, it was also necessary to mask these interrupts from the peripheral processor boards.

Table 5 illustrates the hardware mapping of MULTIBUS interrupts to CPU interrupts for each of the processors in a testbed node.

Processor Interrupt Destinations

Level	CPU #1 (AP)	CPU #2 (SP)	CPU #3 (CP)	CPU #4 (SSP)
0	——	——	——	——
1	——	——	——	——
2	——	——	——	Disk Controller
3	Ethernet Cntlr.	Ethernet Cntlr.	Ethernet Cntlr.	Ethernet Cntlr.
4	Intr. Generator 3	Intr. Generator 0	Intr. Generator 1	Intr. Generator 2
5	On-Board Timer	On-Board USART	On-Board USART	On-Board USART
6	On-Board USART	On-Board Timer	On-Board Timer	On-Board Timer
7	NMI	NMI	NMI	NMI

a)

MULTIBUS Interrupt Sources

Level	Interrupt Source	Function
1	Interrupt Generator 0	Processor #1 to Processor #2
2	Interrupt Generator 1	Processor #1 to Processor #3
3	Ethernet Controller	Ethernet to All Processors
4	Interrupt Generator 2	Processor #1 to Processor #4
5	Interrupt Generator 3	All Processors to Processor #1
6	Disk Controller	Disk to Processor #4
7	Unused	I/O Device to Processor #3
8	Unused	I/O Device to Processor #2

b)

Table 5: Inter-Node Interrupt Assignments

5.5 Serial-Line Connections

The standard Sun-2/120 enclosure has a pair of 25-pin D-sub connectors on its rear bulkhead which are connected to the processor board's two RS-422 serial channels (i.e., Port A and Port B). The testbed adds up to three additional processors to each enclosure and, in order to assist in the process of operating system development and debugging, serial-line console access is needed to each of the processors.

To have connections to all processors in a node, without having to modify the enclosure's bulkhead, a special cable harness was created that allows the daisy-chained interconnection of consoles (as shown in Figure 9). The node's A port is connected directly to the console line of the application processor, while the B port is connected to the console line of the first co-processor. In addition, modifications were made to the PROM monitor's original transparent-mode serial-line feature to permit the nested connection to each of the peripheral co-processors in a node via the node's B port (for details of this function, see [Northcutt 88d]).

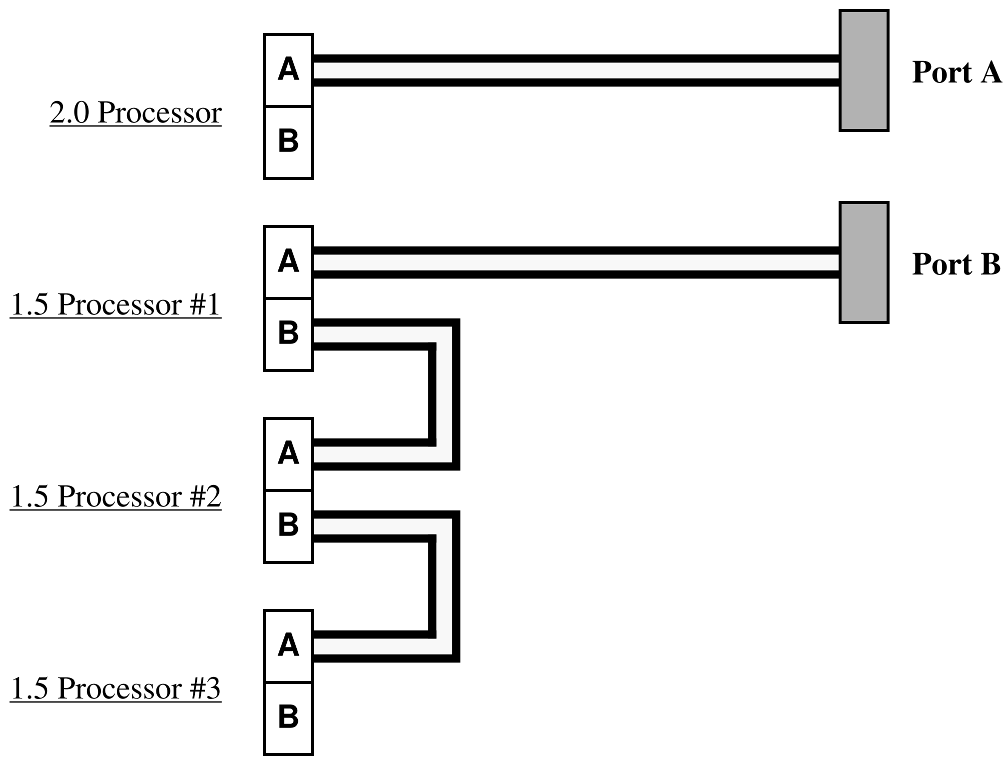


Figure 9: Serial-Line Cable Harness

5.6 MULTIBUS Backplane Configurations

In support of the needs of multiple processor node configurations, the Sun 2/120 workstation includes a MULTIBUS backplane, with integral parallel bus arbitration logic, and a segmented P2 connector section. A standard Sun 2/120 backplane has three separate P2 segments, of six, two, and one connector. However, in support of the testbed's multiple processor nodes, the 2/120's P2 backplane has been subdivided even further. In the current testbed, the P2 backplanes are now divided into a pattern of 4, 1, 1, 2, and 1 connectors.

The 2.0 processor board and its local memory must exist on the same P2 segment, and the Sun Ethernet controller can be placed on any P2 segment. Each of the Sun version 1.5 processor boards must be on its own, separate P2 segment. The shared MULTIBUS memory and the Xylogics disk controller have standard Intel P2 interfaces, and therefore must be on compatible P2 segments. The card slot assignment of the testbed nodes is shown in Table 6 (slot number 1 is the left-most slot as viewed from the front of the 120 box).

Slot	MULTIBUS Board	System Function
1	Sun 1MB Memory	Local Memory #1
2	Sun 1 MB Memory	Local Memory #2
3	Sun 2.0 Processor	Application Processor
4	Sun Ethernet Interface	Communications Controller
5	Sun 1.5 Processor	Secondary Storage Co-Processor
6	Sun 1.5 Processor	Communications Co-Processor
7	SMD Disk Controller	Secondary Storage Controller
8	256KB Memory	Shared MULTIBUS Memory
9	Sun 1.5 Processor	Scheduling Co-Processor

Table 6: Configuration of Boards in a Testbed Node

6 Acknowledgments

Most members of the Alpha team contributed in one way or another to the testbed. In particular, Ray Clark, Sam Shipman, David Maynard, Huay-Yong Wang, Chuck Kollar, Dan Reiner, and Bruce Taylor all worked on the testbed. They made suggestions, wrote PROM code, repaired boards, modified boards, wrote support software, burned PROMS, made and installed wiring harnesses, etc.

Ray Clark deserves special thanks for his assistance in creating this report.

We would all like to thank John Zsarnay, Lawrence Butcher, and Larry Slomer of the CMU CSD engineering lab for their assistance in creating and maintaining the testbed.

We would also like to thank Bill Joy and Andy Bechtolsheim of Sun Microsystems for suggesting the use of, and making available for us, the version 1.5 processor boards, for providing PROM source code and prints for the processor boards and for generally supporting our efforts.

Funding for the purchase, upgrade, and maintenance of the Alpha testbed came primarily from our sponsors at the Rome Air Development Center and General Dynamics.

Many of the technical details provided in this document (particularly, the Tables) were derived from information found in [Sun 84].

References

[Bechtolsheim 82] Bechtolsheim, A., Baskett, F. and Vaughan, P.
The SUN Workstation Architecture.
Technical Report 229, Computer System Laboratory, Stanford University, March 1982.

[Clark 83] Clark, R. K. and Shipman, S. E.
The Archons Testbed—A Requirements Study.
Archons Project Technical Report #83051, Department of Computer Science, Carnegie-Mellon University, May, 1983.

[Jensen 78] Jensen, E. D.
The Honeywell Experimental Distributed Processor—An Overview.
Computer 11(1):137-147, January, 1978.

[Jensen 75] Jensen, E. D.
Time-Value Functions for BMD Radar Scheduling.
Technical Report, Honeywell System and Research Center, June, 1975.

[Maynard 88] Maynard, D. P., Clark, R. K., Kegley, R. B., Keleher, P. J., Northcutt, J. D., Shipman, S. E. and Zimmerman, B. A.
An Example Real-Time Command and Control Application.
Archons Project Technical Report #88032, Department of Computer Science, Carnegie-Mellon University, March, 1988.

[Motorola 84] Motorola
M68000: 16/32-bit Microprocessor Programmer's Reference Manual.
Prentice-Hall, Englewood Cliffs, 1984.

[Northcutt 87] Northcutt, J. D.
Mechanisms for Reliable Distributed Real-Time Operating Systems: The Alpha Kernel.
Academic Press, Boston, 1987.

[Northcutt 88a] Northcutt, J. D.
The Alpha Operating System: Requirements and Rationale.
Archons Project Technical Report #88011, Department of Computer Science, Carnegie-Mellon University, January, 1988.

[Northcutt 88b] Northcutt, J. D.
The Alpha Operating System: Programming Model.
Archons Project Technical Report #88021, Department of Computer Science, Carnegie-Mellon University, February, 1988.

[Northcutt 88c] Northcutt, J. D., Clark, R. K., Shipman, S. E., Maynard, D. P., Lindsay, D. C., Jensen, E. D., Smith, J. M., Kegley, R. B., Keleher and Zimmerman, B. A.
Alpha Preview: A Briefing and Technology Demonstration for DoD.
Archons Project Technical Report #88031, Department of Computer Science, Carnegie-Mellon University, March, 1988.

[Northcutt 88d] Northcutt, J. D. and Shipman, S. E.
The Alpha Operating System: Programming Utilities.
Archons Project Technical Report #88041, Department of Computer Science, Carnegie-Mellon University, April, 1988.

[Rattner 82] Rattner, M., Bechtolsheim, A., Gilmore J., Joy B., Lyon T., McGilton, H. and Shannon, B.
Programmer's Reference Manual for the Sun Workstation.
Technical Report 800-0345, Sun Microsystems, Inc., October, 1982.

[Retsky 84] Retsky, J. E.
MC68020 & MC68881 Platform Boards for Evaluation in a 16-bit System.
Application Note, Motorola Semiconductor Products, Inc., November, 1984.

[Sun 84] Sun Microsystems.
Engineering Manual for the Sun-2/120 CPU Board.
Technical Report 800-1185-01, Sun Microsystems, Inc., September, 1984.

[Sun 85] Sun Microsystems.
Hardware Installation Manual for the Sun-2/120.
Technical Report 800-1170-01, Sun Microsystems, Inc., May, 1985.

Appendix I: Testbed Board Configurations

This appendix provides a description of the jumper and switch settings necessary to configure each of the boards that make up a testbed node. All user-definable options for a board that are not explicitly referenced in this appendix are left in their default (factory configured) settings.

The boards covered in this appendix are the:

- Sun 2.0 processor
- Sun 1.5 processor
- MULTIBUS memory
  - Chrislin
  - Texas Instruments
- Sun Ethernet controller
- Xylogics disk controller

See [Sun 85] for further configuration information on these boards.

Throughout this appendix, the following numbering conventions are used to describe jumper and switch assignments:

Jumper Pin Assignments:
(Component side, edge connector down)

1	3	•	•	•
2	4	•	•	•

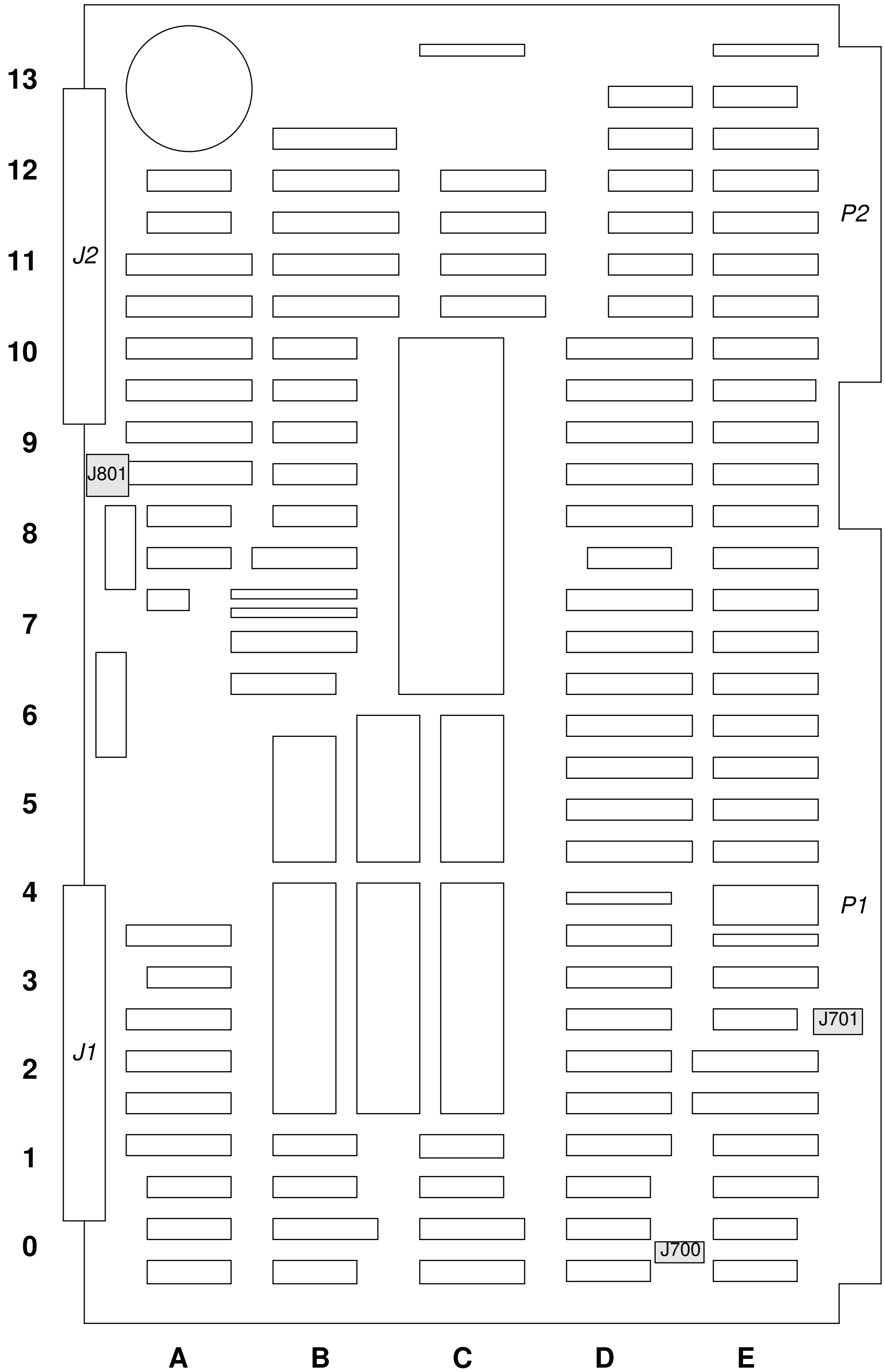
Switch Assignments:
(Component side, edge connector down)

	On	Off
1		
2		
•		
•		
•		

	1	2	•	•	•
On					
Off					

Sun 2.0 Processor

<u>Board Layout</u> — Component Side**:**



Configuration:

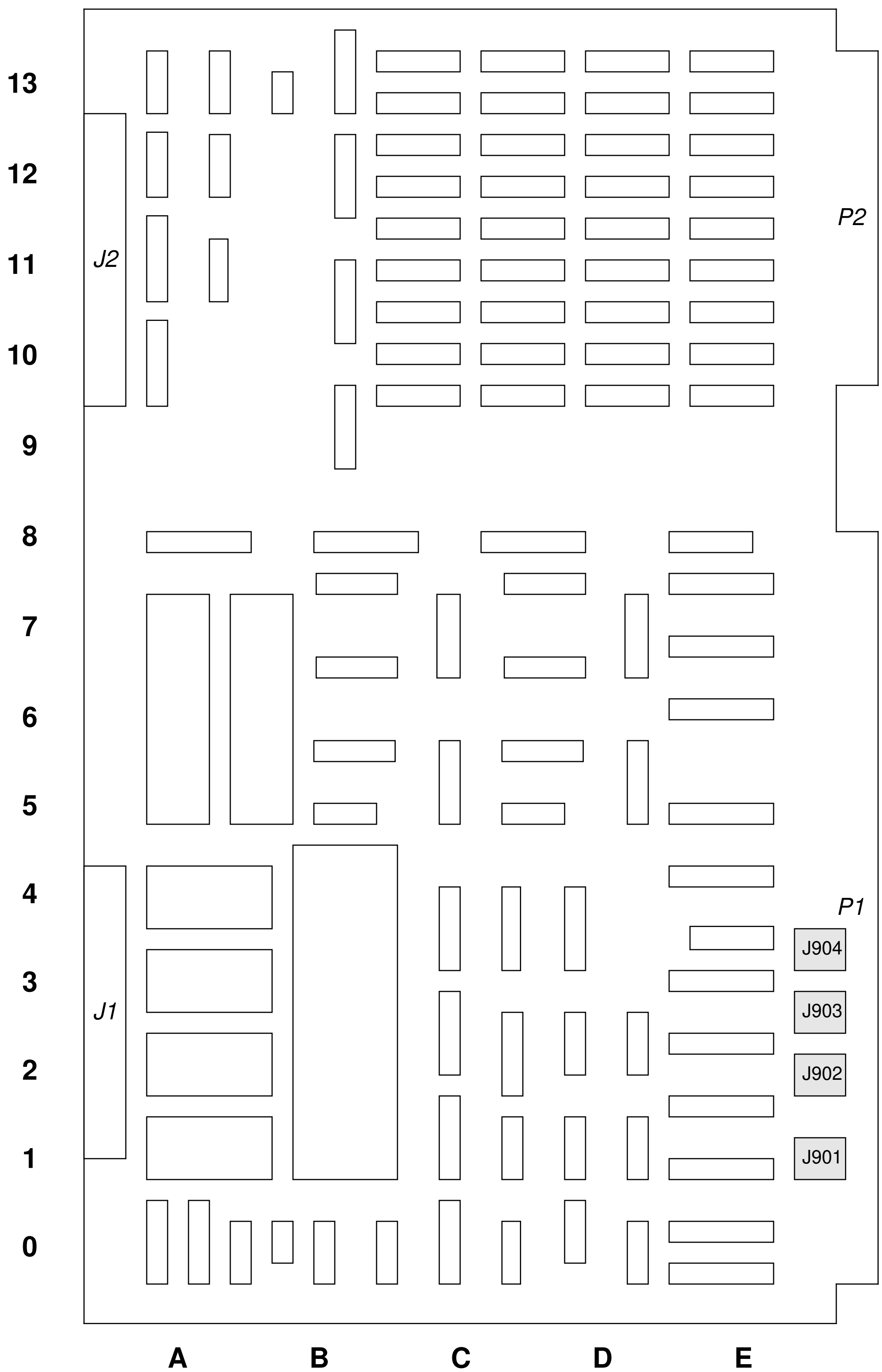
The Sun version 2.0 CPU board must be configured for multi-master operation—i.e., the CBREQ and BPRN signals are taken from the MULTIBUS. This processor serves as the "primary" processor within a node. This means that it generates the clock signals for the bus (BCLK and CCLK) and it generates the node's initialization signal (INIT). The +5V supply that is fed to the mouse via the J2 connector must be disconnected, otherwise the shorting plug which is used to supply the processor's node/processor number can cause a direct short from Vcc to GND on the board (with quite serious consequences).

Jumper Settings:

Jumper Number	Board Location	Function	Configuration
J700	D0	MULTIBUS Control (INIT, CBREQ, and BRPN)	1 :: 2
J701	E3	MULTIBUS Clocks (BCLK and CCLK)	1 :: 2, 3 :: 4
J801	A9	Mouse +5V Supply	—

Sun 1.5 Processor

Board Layout **—** Component Side**:**



Configuration:

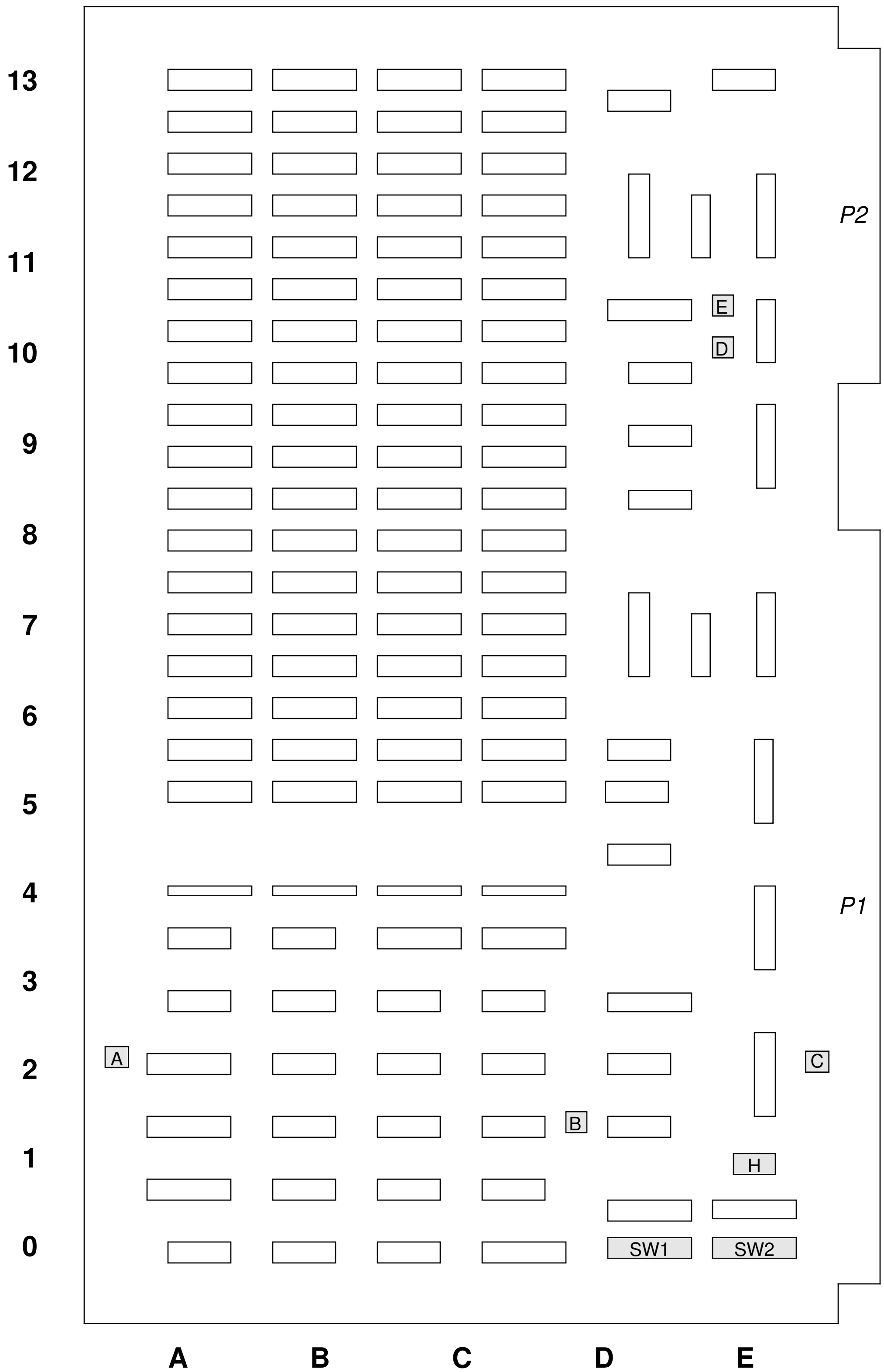
The Sun version 1.5 CPU board must be configured for multi-master operation—i.e., the CBREQ and BPRN signals must be taken from the MULTIBUS. Also, the 1.5 processors are "secondary" processors, in that they taken the initialization (INIT) and clock signals (BCLK and CCLK) from the MULTIBUS. Furthermore, the interrupt configuration for 1.5 processors are dependent on their function. Only two interrupt lines are brought on board from the MULTIBUS—i.e., level 3 (from the Ethernet controller), and level 4 (from the interprocessor interrupt generator). While each 1.5 processor receives its interprocessor interrupt on level 4, the MULTIBUS interrupt signal which is used for this interrupt line varies for each 1.5 processor in a node. The first 1.5 processor uses MULTIBUS level 0, the second uses level 1, and the third, level 2.

Jumper Settings:

Jumper Number	Board Location	Function	Configuration
J901	E1	MULTIBUS Control (INIT, BCLK, & BPRN)	3 :: 4
J902	E2	MULTIBUS Common Bus Request (CBREQ)	3 :: 4
J903	E3	MULTIBUS Constant Clock (CCLK)	—
J904	E4	MULTIBUS Interrupts (co-processor #1)	9 :: 10, 7 :: 16
		(co-processor #2)	9 :: 10, 7 :: 14
		(co-processor #3)	9 :: 10, 7 :: 12

MULTIBUS Memory — Chrislin

Board Layout — Component Side:



Configuration:

The Chrislin MULTIBUS memory board is configured to provide 256KB of memory on the (P1 portion of the) MULTIBUS. All of the banks of memory on the board are enabled and the starting address of the memory array is set to 0x90000. No interrupts (e.g., parity errors) are enabled and extended (i.e., P2) addressing is disabled.

Jumper Settings:

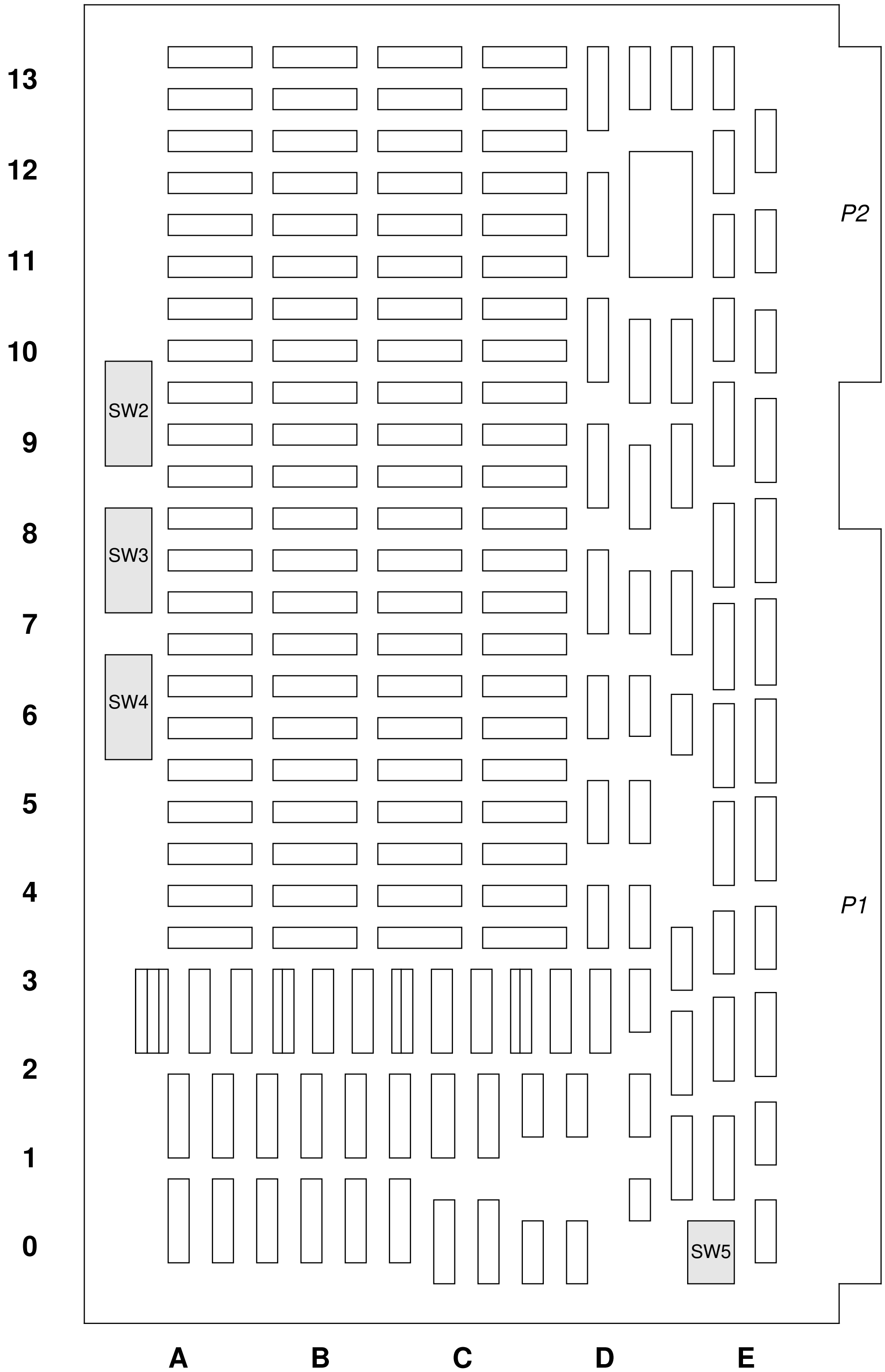
Jumper Number	Board Location	Function	Configuration
A	A2	Advance Write Enable	3 :: 5
B	D1	16KB Bank Select	1 :: 3
C	E2	MULTIBUS Ack (AACK)	1 :: 2
D	E9	Extended Memory Selection	1 :: 3
E	E10	1MB Bank Selection	1 :: 3

Switch Settings:

Switch Number	Board Location	Function	Configuration	
			on	off
SW1	D0	Starting Address - High Part	7, 8	1 - 6
SW2	E0	Starting Address - Low Part	1 - 8	—

MULTIBUS Memory — Texas Instruments

Board Layout — Component Side:



Configuration:

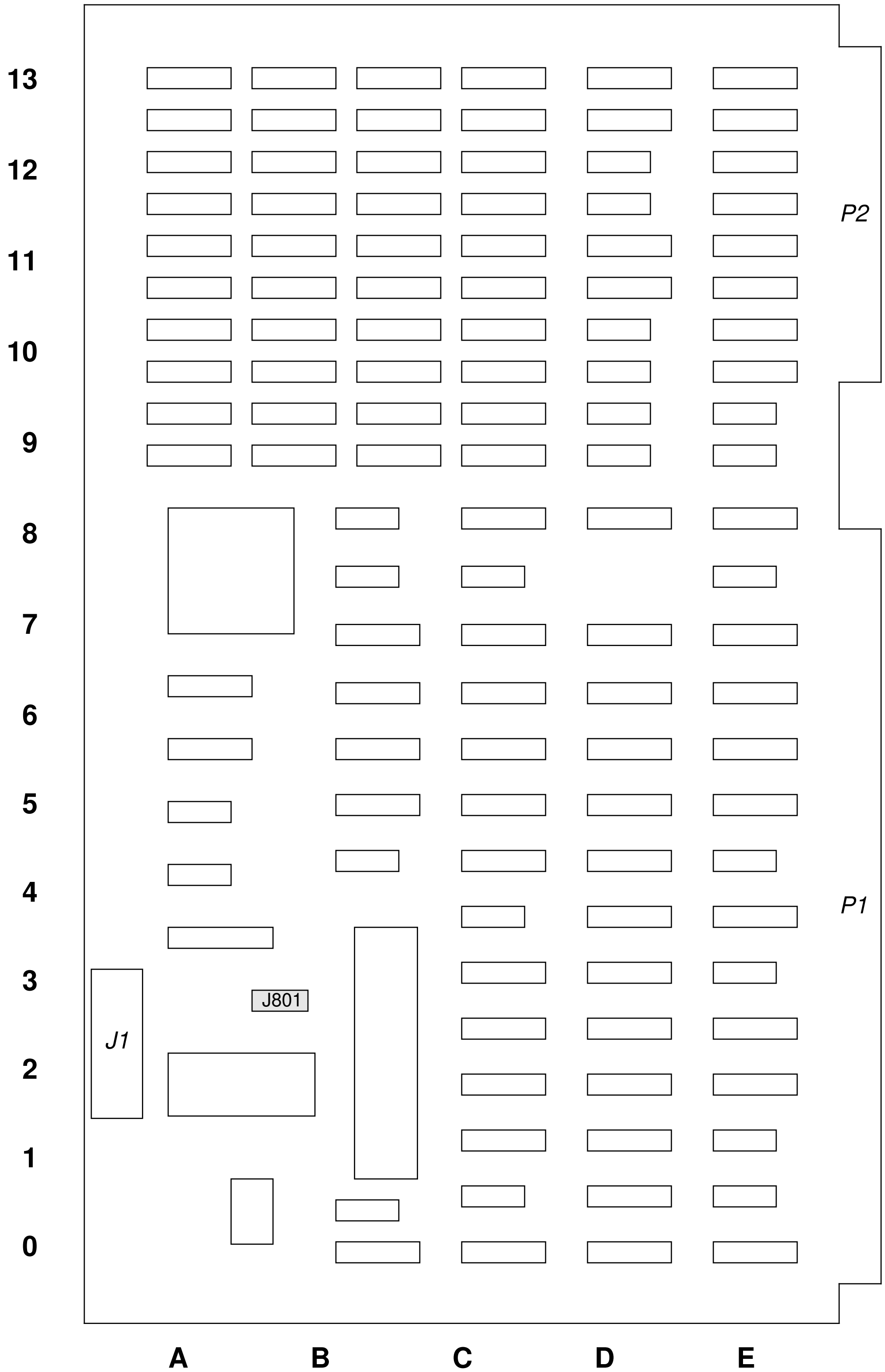
The Texas Instruments MULTIBUS memory board is configured to provide 256KB of memory on the (P1 portion of the) MULTIBUS. All of the banks of memory on the board are enabled and the starting address of the memory array is set to 0x90000. No interrupts (e.g., ECC errors) are enabled and extended (i.e., P2) addressing is disabled. The I/O port is set to an innocuous address in the P1 I/O space.

Switch Settings:

Switch Number	Board Location	Function	Configuration on	Configuration off
SW2	A9	Memory Starting Address	2, 3, 5 - 8	1, 4
SW3	A8	Memory End Address	2	1, 3 - 8
SW4	A6	I/O Port Address	1, 3, 5, 7	2, 4, 6, 8
SW5	E0	Extended Address	—	1 - 5

Sun Ethernet Controller

Board Layout — Component Side:



Configuration:

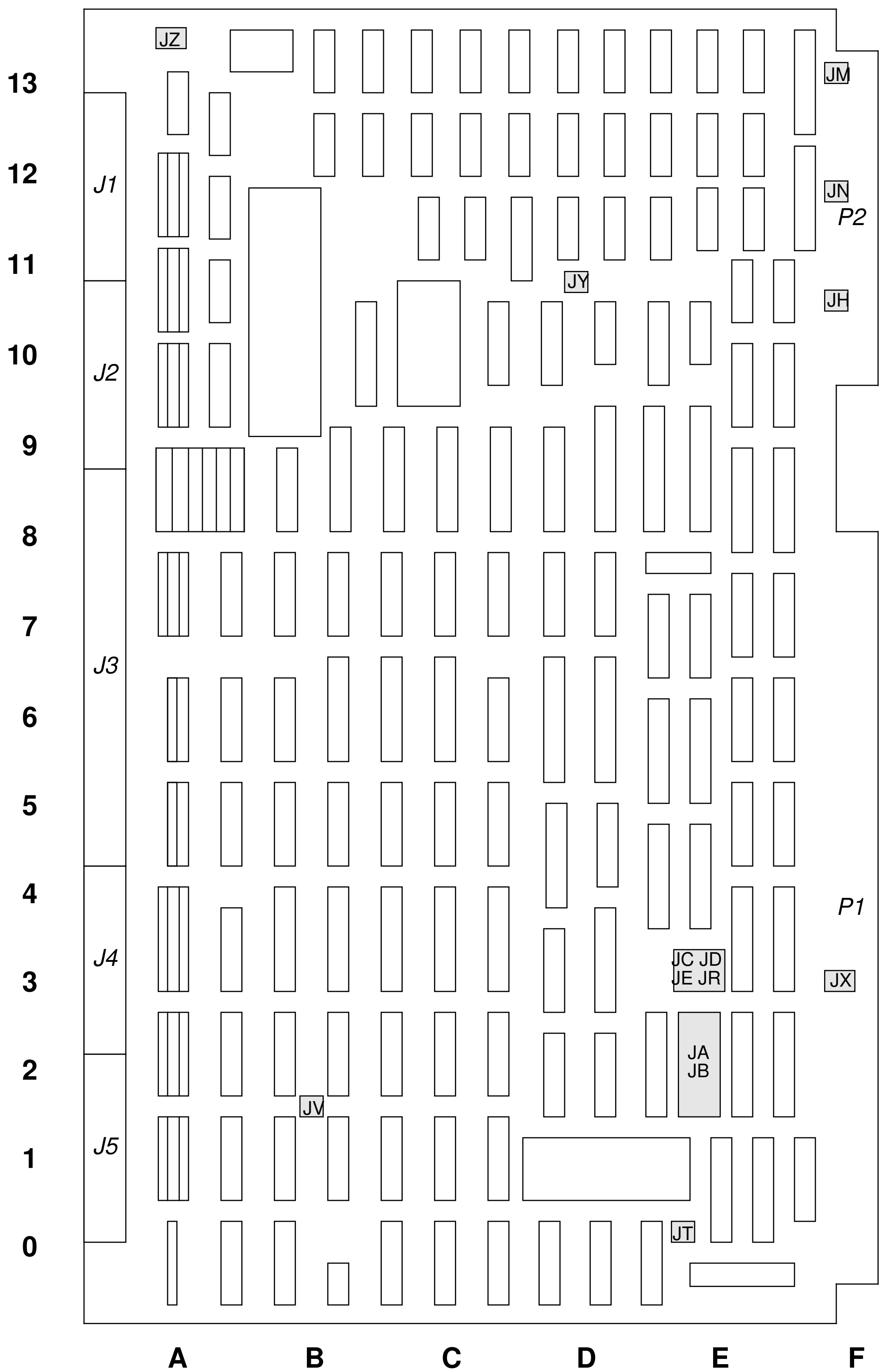
The Sun Ethernet controller is set up to work without additional P2 memory and is set to work with the type of tap that it is connected to (i.e., 3-Com or DEC). The board must generate interrupts at MULTIBUS level 3 (because of an etch defect).

Jumper Settings:

Jumper Number	Board Location	Function	Configuration 3-Com	Configuration DELNI
J801	B3	Ethernet Tap Type	1 :: 3	—

Xylogics Disk Controller

Board Layout — Component Side**:**



Configuration:

The Xylogics SMD disk controller is set up to work with parallel bus arbitration, to have a base address at 0xEE40, with extended addressing disabled, and to interrupt at MULTIBUS level 2.

Jumper Settings:

Jumper Number	Board Location	Function	Configuration
JA, JB	E2	MULTIBUS Base Address	5 :: 6, 15 :: 16
JC, JD JE, JR	E3	MULTIBUS Base Address, MULTIBUS Arbitration	3 :: 4, 6 :: 7, 10 :: 11, 14 :: 15, 13 :: 17
JX	F3	MULTIBUS Interrupt Level	1 :: 5
JM	F13	Extended Addressing	5 :: 7

Appendix II: Alterations to the Testbed Boards

This appendix describes the changes made and the hardware functions that were added to the boards that go into a testbed node. This appendix describes the modifications made to the Sun 1.5 processor boards, the interprocessor interrupt generator module that was added to the Sun 2.0 processor board, and the floating point accelerator unit that was added to both the Sun 2.0 and 1.5 processor boards.

Sun 1.5 Processor Modifications

Description of Modifications

The primary reason why modifications were made to the 1.5 processor boards was to deal with the on-board timers. The initial change involved swapping the assignment of a pair of timers, while a subsequent modification freed the pair of timers used to generate clocks for the on-board DUSART. In addition to these changes, the MULTIBUS configuration signals must be modified, and some of the 1.5 processors were modified to provide 1MB of on-board read/write memory.

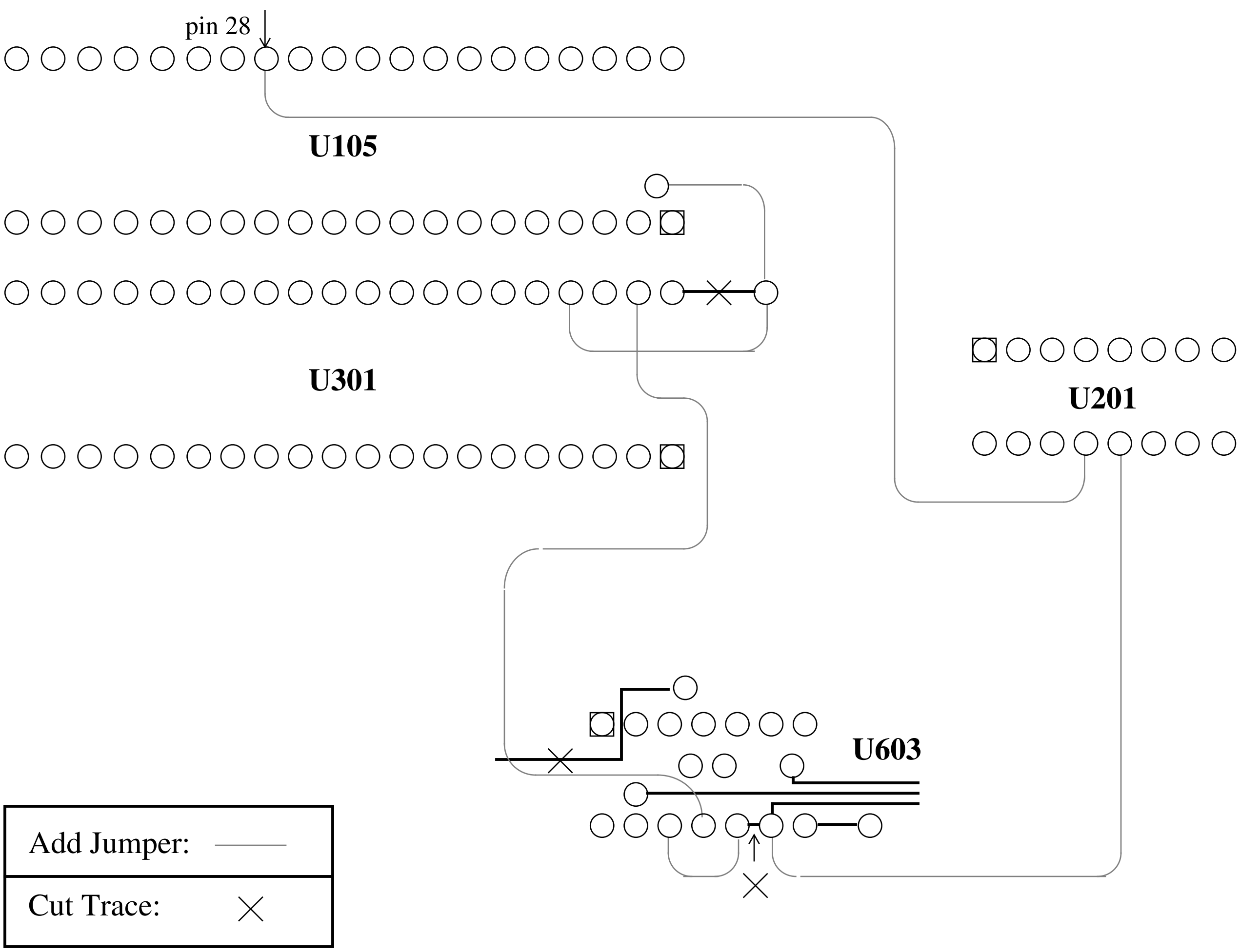
On-Board Timer Changes

In order to permit the cascading of hardware timers, the functions of timer number three (i.e., the memory refresh timer), and timer number five (i.e., the UART B baud rate generator) were swapped. This allowed the user timer (number two) to be cascaded with the B port's timer, in order to create a 32-bit system timer.

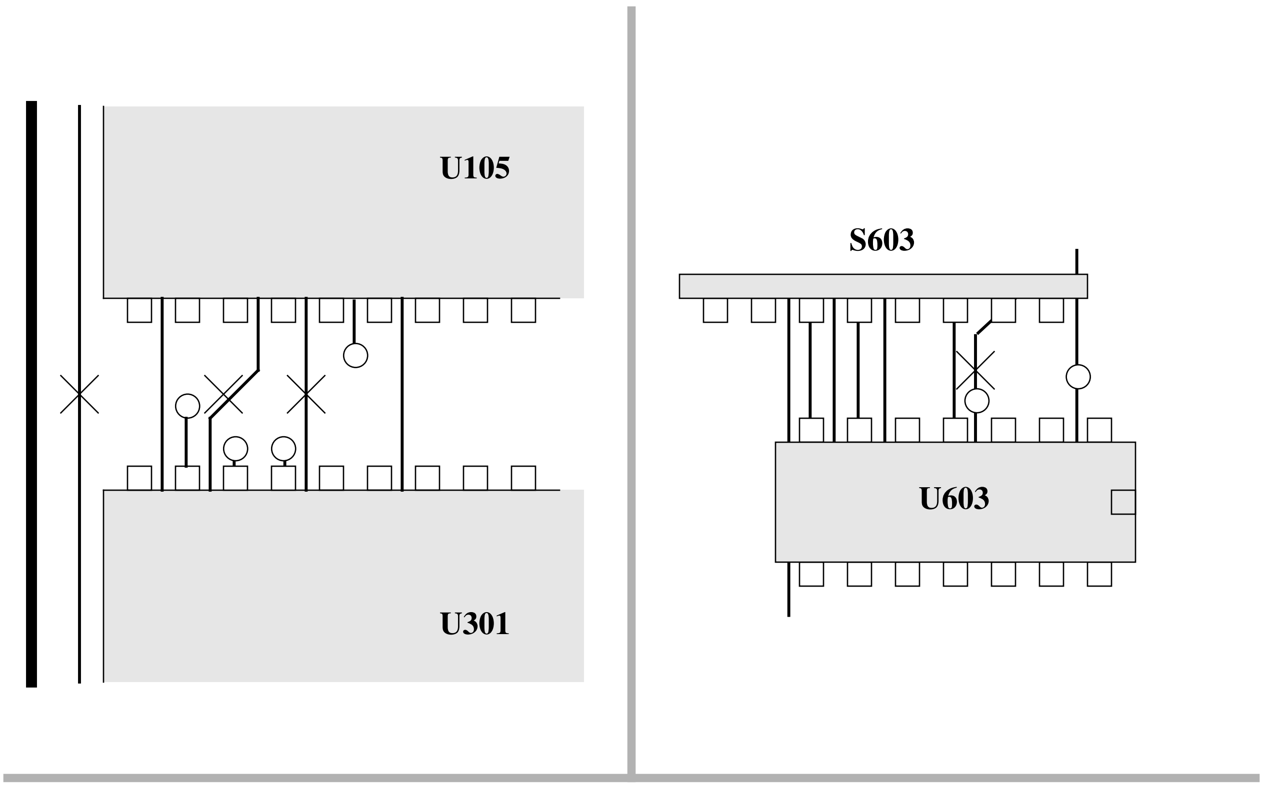
This change required that the B port be unused—a condition that cannot be met in a system with multiple co-processors, connected via daisy-chained serial-lines. To deal with this shortcoming, a further modification was made that involved the addition of a dual baud rate generator to the board in order to free two of the hardware timers for use by the system software (i.e., timers number three and four).

An additional modification involves the addition of the ability to generate interrupts from one of the newly freed timers. This is accomplished by wire-or'ing the output of timer number four with the output of timer number two. This modification involved swapping an unused inverter (U201) with an open collector output inverter that is used where its open collector output is not needed (U603).

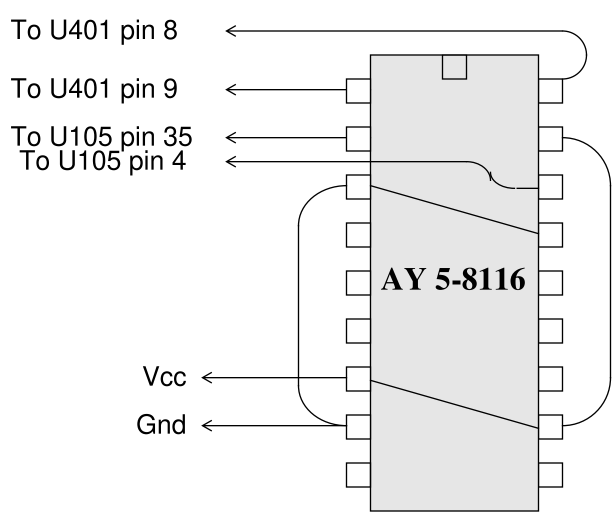
PC Board Changes: (solder side, edge connectors down)



PC Board Changes: (component side, edge connectors down)



Glue chip 'dead bug' style to board, connected with wire-wrap wire.

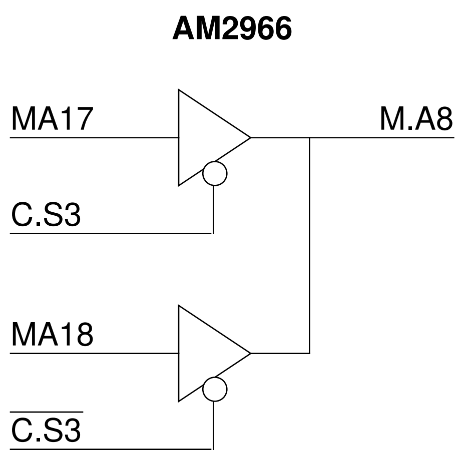


On-Board Memory Upgrade

In order to carry out the upgrade of the 256KB of on-board memory to 1MB, an additional (multiplexed) address line must be provided to the memory array. This is accomplished by the addition of a memory driver (i.e., an AM2966) and the replacement of the 64Kb memories with 256Kb components (i.e., 41256 equivalents).

The AM2966 is affixed to the board in a "dead-bug" fashion, and derives its power from another component on the board. The on-board memory array already has traces for the new address line, and so the addition of the address line is accomplished simply by connecting the output of the driver to pin 1 of the memory array.

Circuit Diagram:



Other Changes to Boards

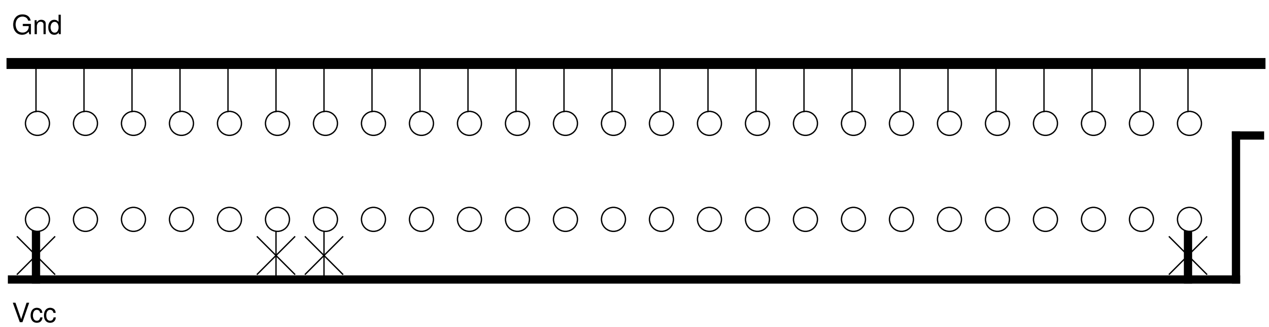
Input Port Change:

In order to permit the input port be used with the node/processor address plug, the Vcc connections (originally used to provide power to the keyboard and mouse) must be removed.

Note: *If these power connections are not severed, a direct short-circuit between power and ground may exist, resulting in a fire hazard.*

PC Board Changes: (solder side, edge connectors down)

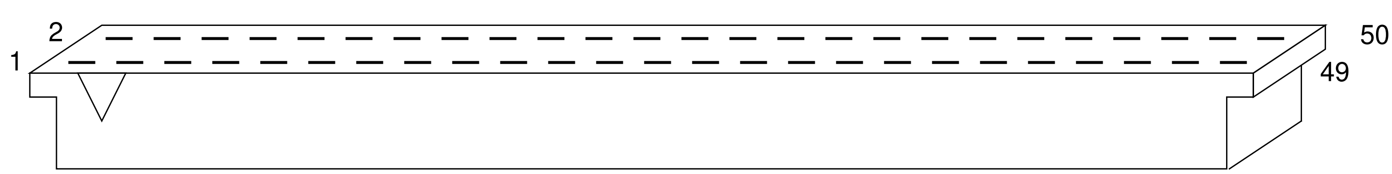
Node/Processor Address Plug:



Insert jumper between odd and even pins to generate 0.

By current convention—the least significant nibble is the number of the processor within the node, the next nibble is the number of the node within the system, and the high two nibbles are unused (and left as `0xFF`).

For example, the second processor within node three includes a plug with the following jumper configuration (i.e., `0xFF21`): 3 :: 4, 5 :: 6, 7 :: 8, 9 :: 10, 13 :: 14, and 15 :: 16.



Interprocessor Interrupt Generator

Theory of Operation

The PAL's registers contain the state information which is used to generate four levels of MULTIBUS interrupt. The registers can be read and written as a group, or individually, and the on-board I/O location of this unit can be mapped to the MULTIBUS via DVMA.

The 7438 NAND gates provide open-collector drivers to the MULTIBUS.

Design Notes

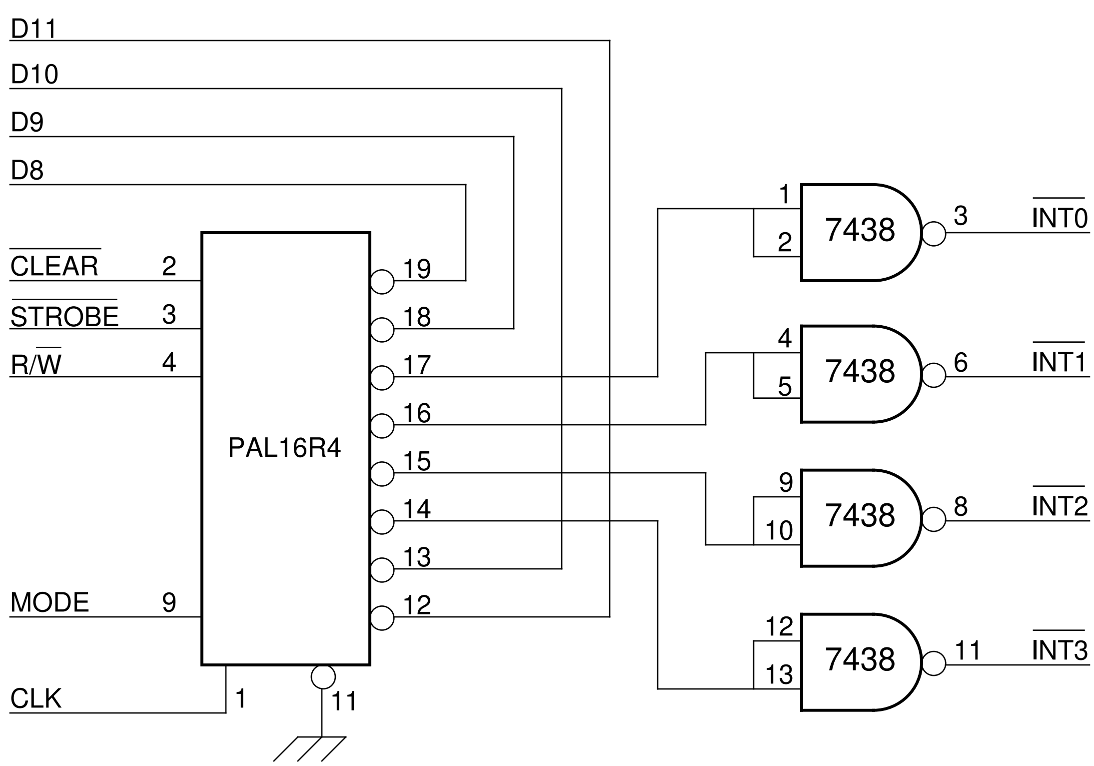
The changes to this processor are minor since the goal was to minimize changes to the 2.0 processor and to make no changes that would keep it from running UNIX. The only modification is an addition of the interrupt generator, which (incidentally) required the recoding of a PAL on the processor board. This configuration of the interrupt generator module is compatible with Sun 2.0 processor PROM revision AJ (and beyond).

Construction Details

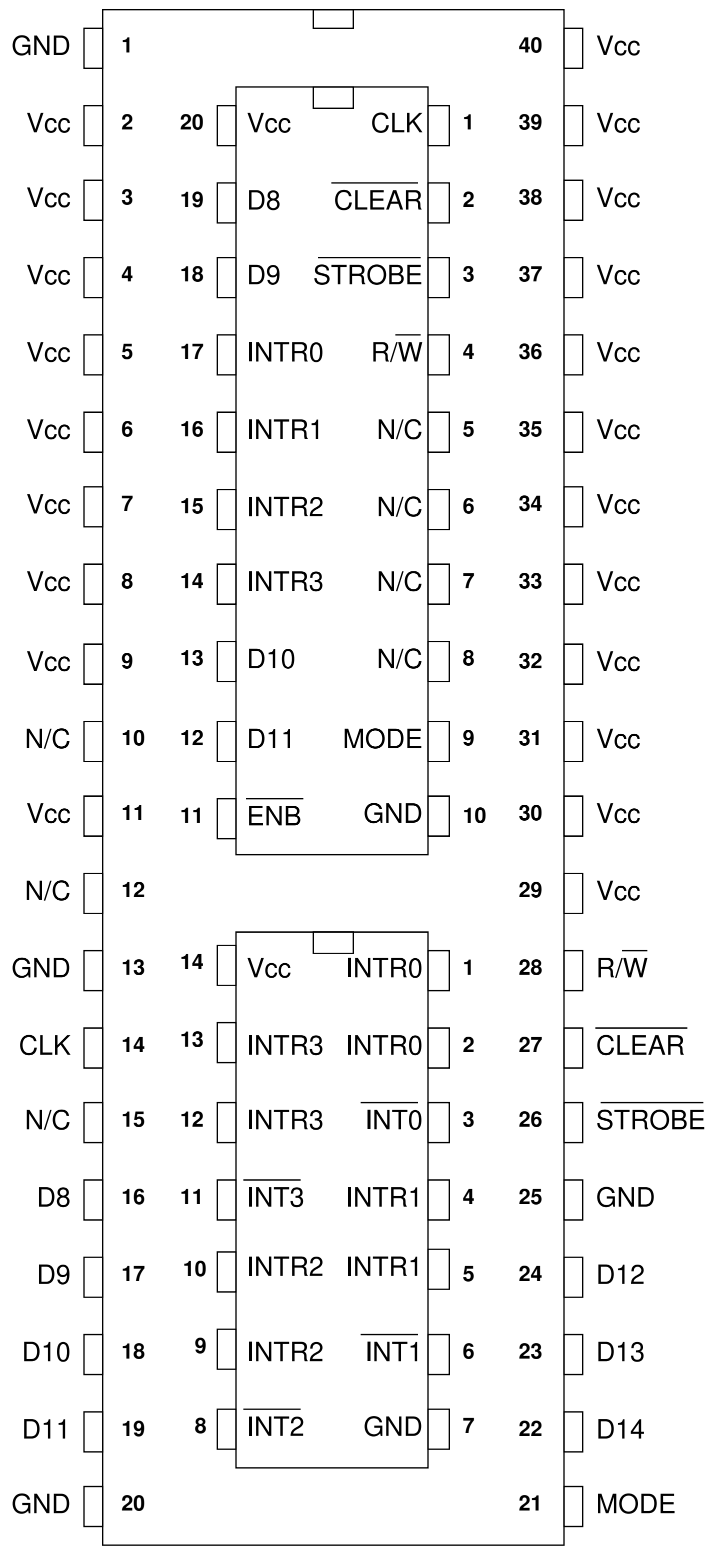
The following includes a schematic diagram of the interprocessor interrupt circuit, a figure describing the manner in which the circuit is packaged into a module, and a diagram indicating the manner in which the module is attached to the Sun 2.0 processor board.

Schematic Diagram

Module Packaging



Installation Details



The interprocessor interrupt generator module is inserted into the Sun 2.0 processor's 40 pin socket that is set aside for an AM9513 DES encryption processor.

In order to permit the I/O device in the DES processor socket to be accessed from the MULTIBUS (via DVMA), the PAL (numbered U602) that generates its control signals must be replaced with a modified (U602a) PAL.

The interrupt lines from the interprocessor interrupt module are connected to the MULTIBUS in the following fashion:

U602 PAL Coding

Interprocessor Interrupt Level	**MULTIBUS Interrupt Levels**
$\overline{\text{INT0}}$	$\overline{\text{INTR0}}$
$\overline{\text{INT1}}$	$\overline{\text{INTR1}}$
$\overline{\text{INT2}}$	$\overline{\text{INTR2}}$
$\overline{\text{INT3}}$	$\overline{\text{INTR4}}$

```
PAL16R4
IGEN
DCP CONTROL PAL (MODIFIED FOR IGEN USE) VERSION 0.1
JDN@CMU
CLK  /RESET  VCC  VCC  /WRDCP  /RDDCP  VCC  LA1  VCC  GND
/OE  /CLEAR  /STROBE  /Q0  /X400  /X200  /ACK  /Q1  /Q2  VCC

IF (VCC) CLEAR =     RESET                      ; INTERRUPT GENERATOR MODULE CLEAR

IF (VCC) STROBE =    WRDCP * /RESET +           ; INTERRUPT GENERATOR MODULE STROBE
                     RDDCP * /RESET

Q0 :=          VCC                              ; UNUSED

X400 :=        X400 * X200 * /RESET +           ; 4X CLOCK
               /X400 * /X200 * /RESET

X200 :=        /X200 * /RESET                   ; 2X CLOCK

ACK :=         STROBE * /RESET                  ; CPU DTACK

IF (VCC)       Q1 = VCC                         ; UNUSED

IF (VCC)       Q2 = VCC                         ; UNUSED


FUNCTION TABLE

CLK  /OE  /RESET
               /RDDCP  /WRDCP  LA1
                               /X400  /X200
                                      /CLEAR  /STROBE  /ACK
                                                    Q0  Q1  Q2

;C  C  I      I  I  I    S  S    O  O  S    S  O  O
------------------------------------------------------------------------------------------
;
; CLEAR
;
 C  L  L      X  X  X    H  H    L  H  H    X  X  X        ; CLEAR
 L  H  X      X  X  X    Z  Z    X  X  Z    Z  X  X        ; TRISTATE OUTPUTS
 C  L  H      H  H  X    X  X    H  H  H    X  X  X        ; IDLE
;
; READ CONTROL REGISTER
;
 C  L  H      L  H  X    X  X    H  L  L    X  X  X        ; STROBE CONTROL REG
;
; WRITE CONTROL REGISTER
;
 C  L  H      H  L  X    X  X    H  L  L    X  X  X        ; STROBE CONTROL REG
;
; CLOCK DIVIDER
;
 C  L  H      X  X  X    L  H    H  X  X    X  X  X        ;
 C  L  H      X  X  X    H  L    H  X  X    X  X  X        ;
 C  L  H      X  X  X    H  H    H  X  X    X  X  X        ;
 C  L  H      X  X  X    L  L    H  X  X    X  X  X        ;
;
------------------------------------------------------------------------------------------
;C  C  I      I  I  I    S  S    O  O  S    S  O  O

;LL -> LH -> HL -> HH
```

DESCRIPTION:

THIS PAL IS USED TO GENERATE CONTROL SIGNALS FOR THE INTERRUPT GENERATOR MODULE WHICH IS PLACED IN THE DCP SOCKET.

IT PASSES THE "/RESET" LINE THROUGH FROM THE SYSTEM TO THE IGEN MODULE IN THE FORM OF THE "/CLEAR" SIGNAL. THIS SIGNAL IS INTENDED TO RESET THE IGEN MODULE AND TRISTATE ITS OUTPUTS. THE "/RESET" LINE ALSO DISABLES ALL OF THE OUTPUT LINES OF THIS PART AND RESETS THE DIVIDER TO ITS INITIAL STATE (I.E., "H H").

THE CLOCK SIGNAL FOR THIS PART IS "/C100" THE NEGATIVE SENSE OF THE 100NS CLOCK.

THE "/WRDCP" AND "/RDDCP" SIGNALS ARE THE READ AND WRITE STROBES FOR THE DCP. THESE SIGNALS ARE OR'ED TOGETHER TO GENERATE THE "/STROBE" SIGNAL WHICH IS USED TO INDICATE THAT THE IGEN MODULE HAS BEEN SELECTED (FOR A READ OR WRITE OPERATION).

THE "LA1" SIGNAL IS THE LATCHED ADDRESS BIT #1 AND IS NOT USED BY THIS PAL.

THE "/X200" AND "/X400" SIGNALS ARE CLOCK DIVIDER OUTPUTS, AND THE "Q0", "Q1", AND "Q2" SIGNALS ARE UNUSED.

THE "/OE" SIGNAL IS STRAPPED TO GENERATE VALID OUTPUTS AT ALL TIMES.

IGEN PAL Coding

```
PAL16R4
IGEN
INTERRUPT GENERATOR MODULE VERSION 2.0
JDN@CMU
CLK /CLEAR /STROBE RNW VCC VCC VCC VCC MODE GND
/ENB D0 D1 INT0 INT1 INT2 INT3 D2 D3 VCC

/INT0 :=   CLEAR +                              ; INTERRUPT 0
           STROBE * /RNW * /MODE * D0 +
           /STROBE * /INT0 +
           RNW * /INT0 +
           /MODE * /INT0 +
           /D0 * /INT0

/INT1 :=   CLEAR +                              ; INTERRUPT 1
           STROBE * /RNW * /MODE * D1 +
           /STROBE * /INT1 +
           RNW * /INT1 +
           /MODE * /INT1 +
           /D1 * /INT1

/INT2 :=   CLEAR +                              ; INTERRUPT 2
           STROBE * /RNW * /MODE * D2 +
           /STROBE * /INT2 +
           RNW * /INT2 +
           /MODE * /INT2 +
           /D2 * /INT2

/INT3 :=   CLEAR +                              ; INTERRUPT 3
           STROBE * /RNW * /MODE * D3 +
           /STROBE * /INT3 +
           RNW * /INT3 +
           /MODE * /INT3 +
           /D3 * /INT3

IF (RNW * STROBE * /CLEAR)    /D0 = /INT0       ; READ 0

IF (RNW * STROBE * /CLEAR)    /D1 = /INT1       ; READ 1

IF (RNW * STROBE * /CLEAR)    /D2 = /INT2       ; READ 2

IF (RNW * STROBE * /CLEAR)    /D3 = /INT3       ; READ 3

FUNCTION TABLE
```

```
CLK /CLEAR /STROBE RNW MODE
/ENB D0 D1 INT0 INT1 INT2 INT3 D2 D3

;       /
;   /   S
;   C   T
;   L   R       M  /            I  I  I  I
;C  E   O   R   O  E            N  N  N  N
;L  A   B   N   D  N    D  D    T  T  T  T    D  D
;K  R   E   W   E  B    0  1    0  1  2  3    2  3
;
;C  I   I   I   I  I    Q  Q    S  S  S  S    Q  Q
-------------------------------------------------------------------------------
 X  X   X   X   X  H    X  X    Z  Z  Z  Z    X  X    ; OUTPUT DISABLED
;
 C  L   X   X   X  H    Z  Z    Z  Z  Z  Z    Z  Z    ; CLEAR
;
 C  H   H   X   X  H    Z  Z    Z  Z  Z  Z    Z  Z    ; IDLE
;
 C  L   L   X   X  X    Z  Z    X  X  X  X    Z  Z    ; INVALID STATE
;
; MODE = 1
;
 C  L   H   X   X  L    Z  Z    L  L  L  L    Z  Z    ; CLEAR
 C  L   H   H   X  L    Z  Z    L  L  L  L    Z  Z    ; IDLE
 C  H   L   L   H  L    L  L    L  L  L  L    L  L    ; WRITE 0000 TO CTL. REG.
 C  H   L   L   H  L    H  L    H  L  L  L    L  L    ; WRITE 0001 TO CTL. REG.
 C  H   L   L   H  L    L  H    H  H  L  L    L  L    ; WRITE 0010 TO CTL. REG.
 C  H   L   L   H  L    L  L    H  H  H  L    H  L    ; WRITE 0100 TO CTL. REG.
 C  H   L   L   H  L    L  L    H  H  H  H    L  H    ; WRITE 1000 TO CTL. REG.
;
; MODE = 0
;
 C  H   L   L   L  L    L  L    H  H  H  H    L  L    ; WRITE 0000 TO CTL. REG.
 C  H   L   L   L  L    H  L    L  H  H  H    L  L    ; WRITE 0001 TO CTL. REG.
 C  H   L   L   L  L    L  H    L  L  H  H    L  L    ; WRITE 0010 TO CTL. REG.
 C  H   L   L   L  L    L  L    L  L  L  H    H  L    ; WRITE 0100 TO CTL. REG.
 C  H   L   L   L  L    L  L    L  L  L  L    L  H    ; WRITE 1000 TO CTL. REG.
;
 C  H   L   H   X  L    L  L    L  L  L  L    L  L    ; READ CTL. REG.
 C  H   L   L   H  L    L  L    L  L  L  H    L  H    ; WRITE 1000 TO CTL. REG.
                                                      ;  MODE = 1
 C  H   L   H   X  L    L  L    L  L  L  H    L  H    ; READ CTL. REG.
 C  H   L   L   H  L    L  L    L  L  H  H    H  L    ; WRITE 0100 TO CTL. REG.
                                                      ;  MODE = 1
 C  H   L   H   X  L    L  L    L  L  H  H    H  H    ; READ CTL. REG.
 C  H   L   L   H  L    L  H    L  H  H  H    L  L    ; WRITE 0010 TO CTL. REG.
                                                      ;  MODE = 1
 C  H   L   H   X  L    L  H    L  H  H  H    H  H    ; READ CTL. REG.
 C  H   L   L   H  L    H  L    H  H  H  H    L  L    ; WRITE 1 TO CTL. REG.
                                                      ;  MODE = 1
 C  H   L   H   X  L    H  H    H  H  H  H    H  H    ; READ CTL. REG.
-------------------------------------------------------------------------------
```

DESCRIPTION:

THIS PAL IS USED TO GENERATE INTER-PROCESSOR INTERRUPTS AMONG SUN MICROSYSTEMS PROCESSORS IN A COMMON BACKPLANE.
ALONG WITH A QUAD OPEN COLLECTOR DRIVER, THIS PAL USES THE 40 PIN SOCKET PROVIDED FOR THE AMD9513 DATA ENCRYPTION PROCESSOR ON SUN 2.0 CPU'S.
THE DCP CONTROL PAL (U602) IS ALSO MODIFIED IN ORDER TO PROVIDE THE NEEDED CONTROL SIGNALS.

THE "/ENB" LINE IS USED TO TRISTATE THE INTERRUPT OUTPUTS.
THE "/CLEAR" LINE CLEARS ALL OF THE INTERRUPT FLIP-FLOPS.
THE "RNW" LINE INDICATES IF A READ OR A WRITE OPERATION IS TO BE PERFORMED. (READ = 1, WRITE = 0).
THE "/STROBE" LINE IS USED TO SELECT THIS PAL.
THE "MODE" LINE IS BIT 15 OF THE CONTROL REGISTER AND INDICATES IF THE MASK (IN BITS 10-7 OF THE CONTROL REGISTER) IS TO BE USED FOR SETTING OR CLEARING THE INTERRUPT FLIP-FLOPS.
THE "D0-D3" LINES ARE BITS 10-7 OF THE PROCESSOR DATA BUS TO BE USED AS MASKS FOR THE SET AND CLEAR OPERATIONS. THESE LINES ARE BIDIRECTIONAL AND THE OUTPUTS ARE TRISTATED EXCEPT WHEN READING FROM THE INTERRUPT FLIP-FLOPS.

THE INTERRUPT FLIP-FLOPS ARE SET BY SETTING THE "MODE" BIT OF CONTROL REGISTER AND SETTING THE DESIRED BITS IN THE INTERRUPT MASK.
TO CLEAR INTERRUPT LINES, THE APPROPRIATE BITS ARE SET IN THE LOW NIBBLE OF THE CONTROL REGISTER, AND THE "MODE" BIT IS CLEARED.

THE CURRENT STATE OF THE INTERRUPT FLIP-FLOPS CAN BE DETERMINED BY READING THE CONTROL REGISTER.

CONTROL REGISTER FORMAT:

Bit 15:	*MODE*
Bit 14 - 12:	*UNUSED*
Bit 11:	*INT3*
Bit 10:	*INT2*
Bit 9:	*INT1*
Bit 8:	*INT0*

MODE:
BIT USED TO INDICATE THE TYPE OF OPERATION
1 = SET INTERRUPT(S), 0 = CLEAR INTERRUPT(S)

INT3-INT0:
MASK USED TO SET/CLEAR INDIVIDUAL INTERRUPTS
1 = CHANGE STATE (SET/CLEAR DEPENDING ON MODE BIT), 0 = NOP

Floating Point Accelerator

Theory of Operation

This unit behaves similar to that described in a Motorola application note [Retsky 84]. The main differences result from the changes made necessary by the fact that the Sun-2 processor board is designed in such a fashion as to generate the DTACK signal for the CPU at a fixed time during a MULTIBUS cycle, independent of the acknowledgement signal that is returned from the MULTIBUS slave device. This design detail has no effect on the MC68010 CPU, but violates the signal sequencing requirements for the MC68020. Therefore, it was necessary to "stretch" the DTACK signal on the floating point accelerator board.

Design Notes

The objective of this daughter board was to provide hardware support for the floating point needs of the Alpha operating system, in a fashion that requires a minimum amount of hardware modification and a minimum degree of software support. This was accomplished by creating a drop-in replacement for the Sun processor boards' MC68010 processor that uses a MC68020/MC68881 pair that is identical to that in a Sun-3 workstation. This permits the use of the standard Sun compiler and programming support tools and minimize the software effort needed to make use of the new hardware.

There are a number of commercial units of this type available (all of which are based on the same Motorola application note), but none of these are capable of functioning in a Sun-2 processor, and it was therefore necessary to construct a custom unit.

To deal with the problems posed by the Sun-2 processor board, additional logic is needed beyond that provided by the three PALs in the Motorola design. In order to maintain the commonality of components used in the design and to permit future modifications, another 16R4 PAL was added to the design.

It should be noted that it is necessary to use B speed PALs and a 16MHz MC68020 to ensure that the unit's set-up and hold times are met. If PALs with faster set-up times are used, it may be possible to use a 12.5MHz processor. In any event, the MC68881 can be either a 10MHz or 12.5MHz speed part.

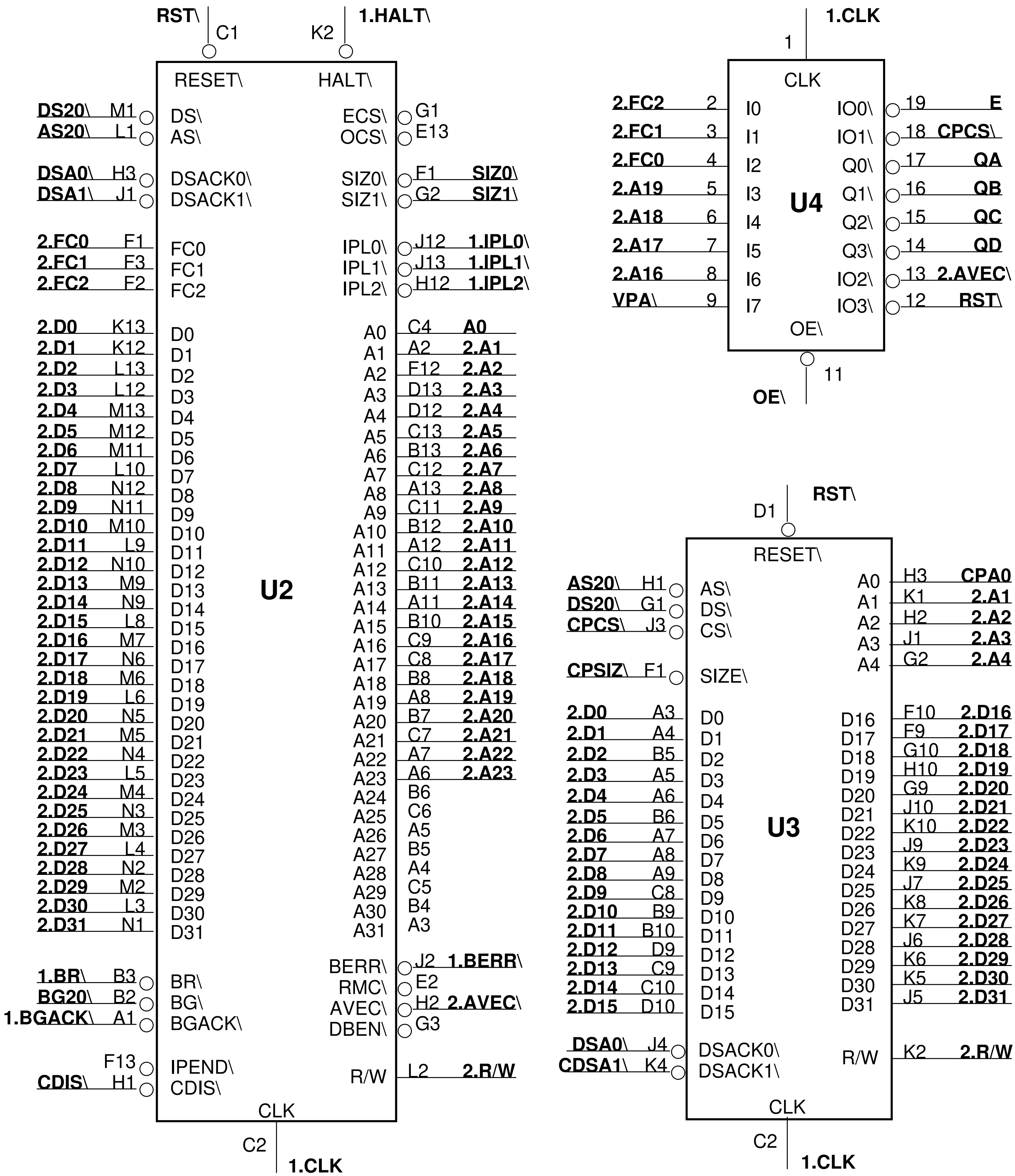
Construction Details

The following includes a schematic diagram for the floating point hardware, a description of the layout of the unit's (four-layer) printed circuit board, and the listings for the unit's PALs.

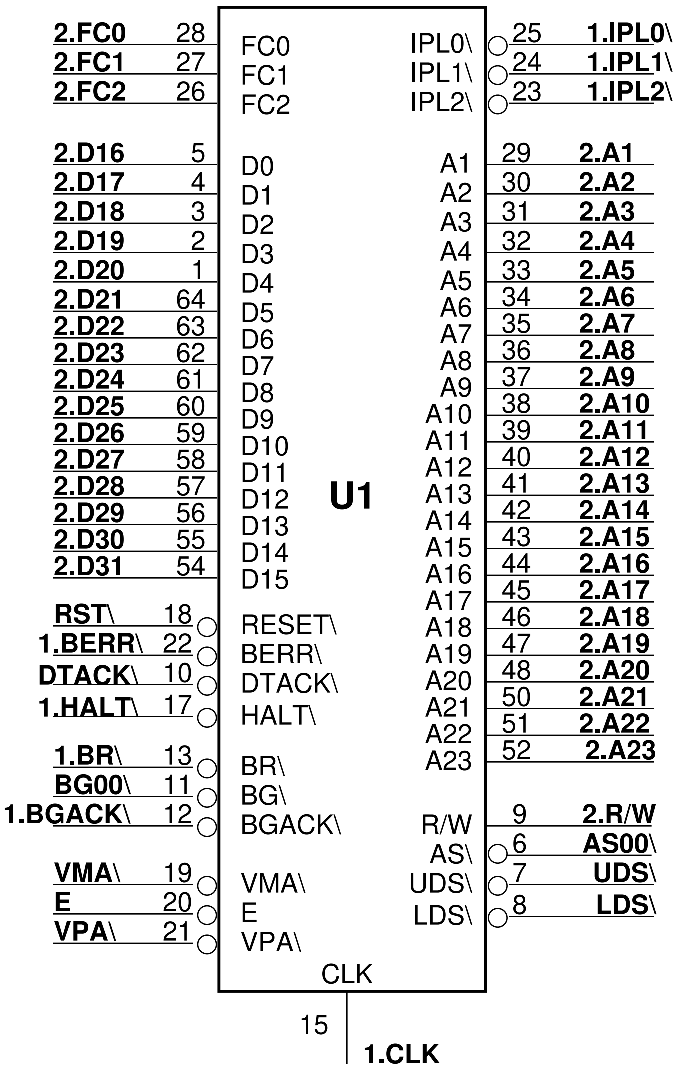
Schematic Diagram

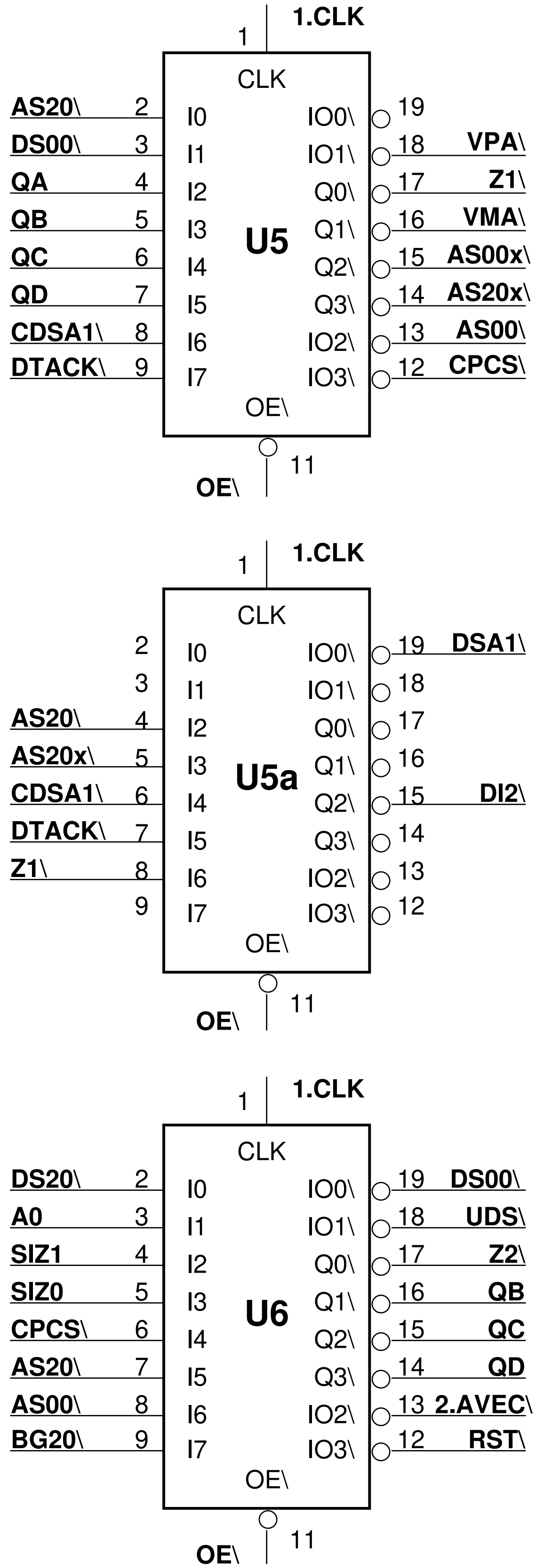
PCB Layout — Component Side

FPA PAL #4 Coding



Schematic Diagram — continued

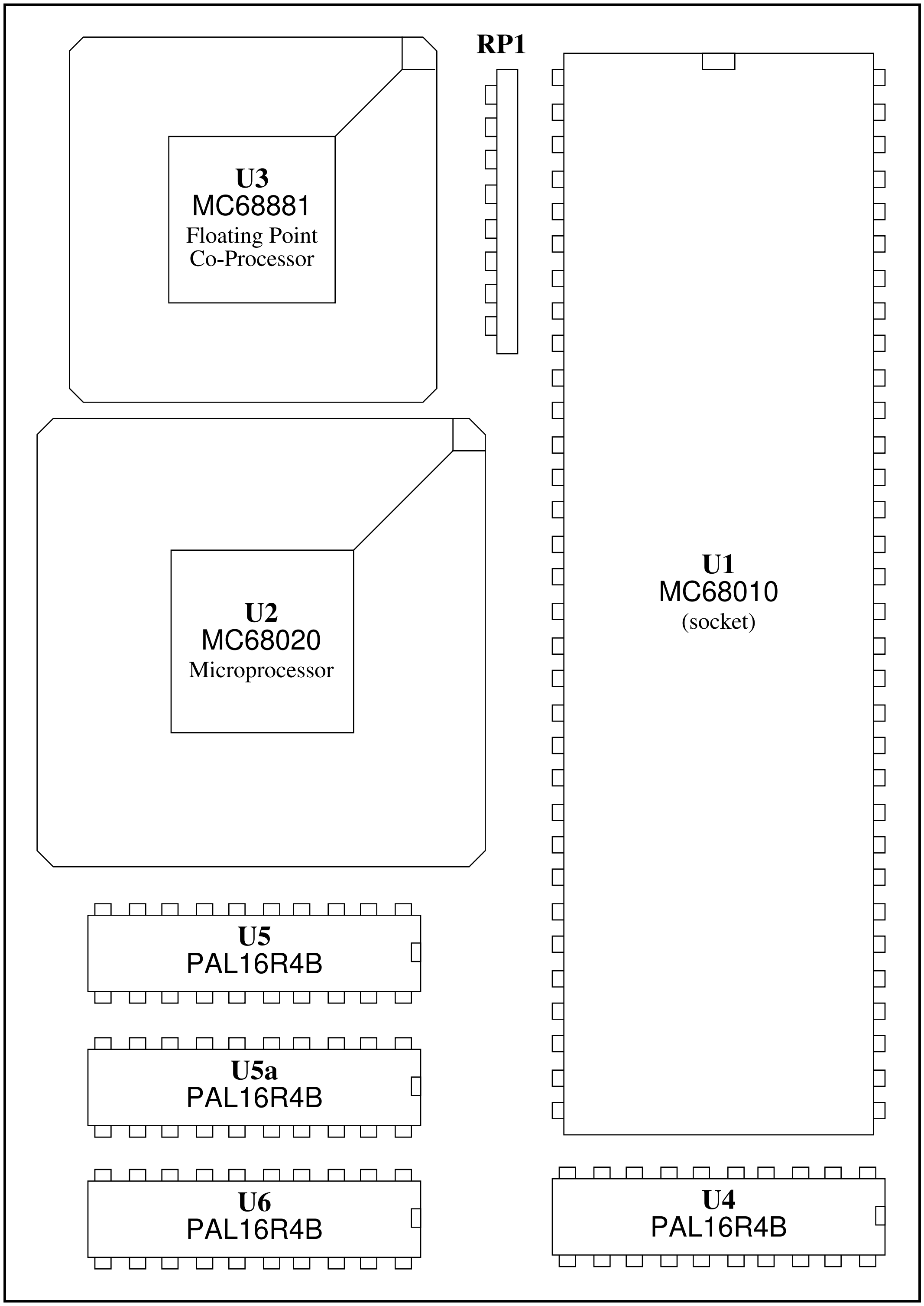




Power and Ground Pins

IC	Vcc	Gnd
U1	14, 49	16, 53
U2	A9, D1, D2, D3, E3, G11, G13, M8, N8, N13	A10, B1, B9, C3, F12, G12, H13, J3, K1, K3, L7, L11, N7
U3	A1, B1, B8, E2, E9, H8, J2	A2, A10, B2, B3, B4, B7, C1, C3, D2, E10, F2, H9, J8, K3
U4, U5, U5a, U6	20	10

PAL16R4B


RP1
U3
MC68881
Floating Point
Co-Processor
U2
MC68020
Microprocessor
U1
MC68010
(socket)
U5
PAL16R4B
U5a
PAL16R4B
U6
PAL16R4B
U4
PAL16R4B

```
FPA4
FLOATING POINT ACCELERATOR VERSION 1.0
JDN@CMU (FROM MOTOROLA)
CLK FC2 FC1 FC0 A19 A18 A17 A16 /VPA GND
/OE /RESET /AVEC QD QC QB QA /CPCS E VCC

/QA :=      QA * /RESET                       ; DECADE COUNTER

/QB :=      /QB * /QA * /RESET +
            QB * QA * /RESET +
            QD * /RESET

/QC :=      /QC * /QA * /RESET +
            /QC * /QB * /RESET +
            QC * QB * QA * /RESET

/QD :=      /QD * /QC * /RESET +
            /QB * QA * /RESET +
            /QD * /QA * /RESET

IF (VCC)    /E = /QD * /QC +                  ; 6800 PERIPHERAL CLOCK
            /QD * /QB

IF (VCC)    CPCS = FC2 * FC1 * FC0 *          ; CO-PROCESSOR SELECT
            /A19 * /A18 *
            A17 * /A16

IF (VCC)    AVEC = FC2 * FC1 * FC0 *          ; AUTOVEC RQST TO 020
         A19 * A18 * A17 *
         A16 * VPA

FUNCTION TABLE

;INPUTS
FC2 FC1 FC0 A19 A18 A17 A16 /VPA /OE /RESET
```

```
;OUTPUTS
QA QB QC QD E /CPCS /AVEC

;
;
;                                 /
;                                 R                            /  /
;                           /     E                            C  A
;F  F  F    A  A  A  A      V  /  S                            P  V
;C  C  C    1  1  1  1      P  O  E     Q  Q  Q  Q             C  E
;2  1  0    9  8  7  6      A  E  T     A  B  C  D       E     S  C
;
;I  I  I    I  I  I  I      I  I  I     Q  Q  Q  Q       S     S  S
------------------------------------------------------------------------
 L  L  L    L  L  L  L      H  H  L     Z  Z  Z  Z       X  X  X   ;1 TRI-STATE TEST
 L  L  L    L  L  L  L      H  L  L     H  H  H  H       H  H  H   ;2 RESET CTR.
 L  L  L    L  L  L  L      H  L  H     L  L  L  H       H  H  H   ;3 DECADE CTR.
 L  L  L    L  L  L  L      H  L  H     H  L  L  H       H  H  H   ;4 & "E-CLK" TEST
 L  L  L    L  L  L  L      H  L  H     L  L  L  L       L  H  H   ;5
 L  L  L    L  L  L  L      H  L  H     H  L  L  L       L  H  H   ;6
 L  L  L    L  L  L  L      H  L  H     L  H  L  L       L  H  H   ;7
 L  L  L    L  L  L  L      H  L  H     H  H  L  L       L  H  H   ;8
 L  L  L    L  L  L  L      H  L  H     L  L  H  L       L  H  H   ;9
 L  L  L    L  L  L  L      H  L  H     H  L  H  L       L  H  H   ;10
 L  L  L    L  L  L  L      H  L  H     L  H  H  L       H  H  H   ;11
 L  L  L    L  L  L  L      H  L  H     H  H  H  L       H  H  H   ;12
 L  L  H    H  H  H  H      L  L  H     L  L  L  H       H  H  H   ;13 /AVEC TEST
 L  H  L    H  H  H  H      L  L  H     H  L  L  H       H  H  H   ;14
 L  H  H    H  H  H  H      L  L  H     L  L  L  L       L  H  H   ;15
 H  L  L    H  H  H  H      L  L  H     H  L  L  L       L  H  H   ;16
 H  L  H    H  H  H  H      L  L  H     L  H  L  L       L  H  H   ;17
 H  H  L    H  H  H  H      L  L  H     H  H  L  L       L  H  H   ;18
 H  H  H    H  H  H  H      L  L  H     L  L  H  L       L  H  L   ;19
 H  H  H    H  H  H  H      H  L  H     H  L  H  L       L  H  H   ;20
 H  H  H    H  H  H  L      L  L  H     L  H  H  L       H  H  H   ;21
 H  H  H    H  H  L  H      H  L  H     H  H  H  L       H  H  H   ;22
 H  H  H    H  L  H  H      H  L  H     L  L  L  H       H  H  H   ;23
 H  H  H    L  H  H  H      H  L  H     H  L  L  H       H  H  H   ;24
 H  H  H    L  L  H  L      X  L  H     L  L  L  L       L  L  H   ;25
 H  H  H    L  L  H  H      X  L  H     H  L  L  L       L  H  H   ;26
 H  H  H    L  L  L  L      X  L  H     L  H  L  L       L  H  H   ;27
 H  H  H    L  H  H  L      X  L  H     H  H  L  L       L  H  H   ;28
 H  H  H    H  L  H  L      X  L  H     L  L  H  L       L  H  H   ;29
 H  H  L    L  L  H  L      X  L  H     H  L  H  L       L  H  H   ;30
 H  L  H    L  L  H  L      X  L  H     L  H  H  L       H  H  H   ;31
 L  H  H    L  L  H  L      X  L  H     H  H  H  L       H  H  H   ;32
------------------------------------------------------------------------
```

DESCRIPTION:

THIS PAL IS USED TO GENERATE THE 68010'S "E" CLOCK (USED FOR

INTERFACING TO 6800 PERIPHERALS), THE AUTOVECTOR SIGNAL, AND THE CO-PROCESSOR CHIP SELECT SIGNAL.

INPUTS:

THE "CLK" LINE IS THE FUNDAMENTAL CLOCK FROM THE 68010 AND SHARED WITH THE OTHER PALS, THE 68020, AND THE 68881.

THE "FC0-FC2" LINES ARE THE 68020'S FUNCTION CODE BITS. A PATTERN OF "111" INDICATES "CPU SPACE" -- THIS IS WHERE INTERRUPTS ARE ACK'D AND CO-PROCESSES ARE DEALT WITH.

THE "A16-A19" LINES ARE 68020 ADDRESS LINES. THESE BITS ARE DECODED ALONG WITH THE FUNCTION CODE BITS TO DETERMINE WHAT AREA OF CPU SPACE IS BEING ACCESSED.

THE "/VPA" LINE IS THE VALID PERIPHERAL ADDRESS INDICATION FROM THE 68000 SYSTEM AND MEANS THAT A 6800 PERIPHERAL HAS BEEN SELECTED.

THE "/OE" LINE IS THE OUTPUT ENABLE SIGNAL FOR THE PAL AND IS PERMENENTLY ENABLED.

THE "/RESET" LINE IS USED TO RESET THE FLIP/FLOPS IN THE PAL. THIS SIGNAL IS GENERATED BY THE 68000 SYSTEM.

OUTPUTS:

THE "QA-QD" LINES ARE THE PAL'S DECADE COUNTER OUTPUTS. THIS IS A FREE-RUNNING COUNTER USED TO GENERATE THE 60/40 DUTY-CYCLE "E" SIGNAL FOR 6800 PERIPHERALS.

THE "/CPCS" LINE IS THE CO-PROCESSOR CHIP SELECT LINE USED TO SELECT THE 68881.

THE "/AVEC" LINE IS THE 68020'S AUTOVECTOR INTERRUPT ACKNOWLEDGE REQUEST LINE. THIS REQUESTS THAT THE 68020 DO AN AUTOVECTOR INTERRUPT ACKNOWLEDGE CYCLE.

FPA PAL #5 Coding

PAL16R4B
FPA5

```
FLOATING POINT ACCELERATOR VERSION 1.0
JDN@CMU (FROM MOTOROLA)
CLK /AS20 /DS00 QA QB QC QD /CPDSA1 /DTACK GND
/OE /CPCS /AS00 /AS20X /AS00X /VMA /Z1 /VPA /DSA1 VCC

AS20X :=    AS00X                                ; ADDR STRB DELAYED 2 CLKS

AS00X :=    AS20                                 ; ADDR STRB DELAYED 1 CLK

VMA :=      /QD * QC * /QB * /QA * VPA +         ; VALID MEMORY ADDRESS
            VMA * QC +
            VMA * QD

Z1 :=       DS00 * QD * /QC * /QB * /QA * VMA +  ; /DSA1 INTERMEDIATE
            Z1 * DS00

IF (VCC) DSA1 = CPDSA1 +                         ; 68020 DATA STRB ACK #1
                Z1 +
                DTACK * AS20X

IF (VCC) AS00 = /CPCS * AS20 * AS00X             ; 68000 SYSTEM ADDR STRB
```

FUNCTION TABLE

```
;INPUTS
/AS20 /DS00 QA QB QC QD /CPDSA1 /DTACK /OE /CPCS /VPA
```

```
;OUTPUTS
/AS00 /AS20X /AS00X /VMA /Z1 /DSA1

;
;              /
;              C /           / /
;/ /           P D   /    /  A A      /
;A D           D T   C /  A  S S /    D
;S S           S A / P V  S  2 0 V /  S
;2 0  Q Q Q Q  A C O C P  0  0 0 M Z  A
;0 0  A B C D  1 K E S A  0  X X A 1  1
;
;I I  I I I I  I I I I I  S  Q Q Q Q  S
--------------------------------------------------------------------------------
 H H  H H H H  H H H H H  X  Z Z Z Z  X  ;1TRI-STATE TEST
 H H  H H H H  H H H H H  X  Z Z Z Z  X  ;2
 H H  H H H H  H H L H H  H  X H X H  H  ;3IDLE AFTER RST
 L H  H H H H  H H L H H  L  H L X H  H  ;4 TEST /AS00X
 L H  H H H H  H H L L H  H  L L X H  H  ;5  AND /AS20X
 L H  H H H H  H L L H H  L  L L X H  L  ;6
 H H  H H H H  H H L H H  H  L H X H  H  ;7
 H H  L L L H  H H L H L  X  X X X H  H  ;8 TEST /VMA C=1
 H H  H L L H  H H L H L  X  X X X H  H  ;9               2
 H H  L L L L  H H L H L  X  X X H H  H  ;10              3
 H H  H L L L  H H L H L  X  X X H H  H  ;11              4
 H H  L H L L  H H L H L  X  X X H H  H  ;12              5
 H H  H H L L  H H L H L  X  X X H H  H  ;13              6
 H H  L L H L  H H L H L  X  X X L H  H  ;14              7
 H H  H L H L  H H L H L  X  X X L H  H  ;15              8
 H H  L H H L  H H L H H  X  X X L H  H  ;16              9
 H H  H H H L  H H L H H  X  X X L H  H  ;17             10
 H L  L L L H  H H L H H  X  X X L L  L  ;18              1
 H L  H L L H  H H L H H  X  X X L L  L  ;19 TEST /Z1     2
 H L  L L L L  H H L H H  X  X X H L  L  ;20              3
 H H  H L L L  H H L H H  X  X X H H  H  ;21              4
 H H  L H L L  H H L H H  X  X X H H  H  ;22              5
 H H  H H L L  L H L H H  X  X X H H  L  ;23 TEST /DSA1   6
 H H  L L H L  H H L H H  X  X X H H  H  ;24              7
--------------------------------------------------------------------------------
```

DESCRIPTION:

THIS PAL IS USED TO GENERATE ADDRESSING STROBES AND THE VMA SIGNAL.

INPUTS:

THE "CLK" LINE IS THE FUNDAMENTAL CLOCK FROM THE 68010 AND SHARED WITH THE OTHER PALS, THE 68020, AND THE 68881.

THE "/AS20" LINE IS THE 68020'S ADDRESS STROBE SIGNAL.

THE "/DS20" LINE IS THE 68000'S DATA STROBE SIGNAL. THIS SIGNAL IS GENERATED BY PALU6, AND IS USED IN THE GENERATION OF THE "/DSA1" OUTPUT.

THE "QA-QD" LINES ARE FROM THE PALU4'S DECADE COUNTER. THESE SIGNALS ARE USED TO GENERATE THE "/VMA" OUTPUT.

THE "/CPDSA1" LINE IS THE CO-PROCESSOR DATA STROBE ACKNOWLEDGE SIGNAL NUMBER 1 FROM THE 68881.

THE "/DTACK" LINE IS THE DATA TRANSFER ACKNOWLEDGE SIGNAL FROM THE 68000 SYSTEM. IT IS USED TO GENERATE THE 68020'S "/DSK1" SIGNAL.

THE "/CPCS" LINE IS THE CO-PROCESSOR CHIP SELECT USED TO ENABLE THE 68881 AND USED IN THIS PAL TO INHIBIT SENDING THE "/AS" SIGNAL TO THE 68000 SYSTEM WHEN THE 68020 IS TALKING TO THE 68881.

THE "/VPA" LINE IS THE VALID PERIPHERAL ADDRESS SIGNAL THAT INDICATES THAT A 6800 PERIPHERAL HAS BEEN SELECTED, SO THE 68020 CAN GET A DELAYED "/DSA" SIGNAL, THIS IS ALSO USED TO GENERATE THE "/VMA" SIGNAL TO THE 68000 SYSTEM.

THE "/OE" LINE IS THE OUTPUT ENABLE SIGNAL FOR THE PAL AND IS PERMENENTLY ENABLED.

OUTPUTS:

THE "/VMA" LINE IS THE VALID MEMORY ADDRESS SIGNAL SENT TO THE 68000 SYSTEM, FOR USE BY 6800 PERIPERALS.

THE "/AS00X" LINE IS A DELAYED ADDRESS STROBE SIGNAL FROM THE 68020.

THE "/AS20X" LINE IS THE DELAYED ADDRESS STROBE SIGNAL FROM THE 68020.

THE "/Z1" LINE IS AN INTERMEDIATE SIGNAL USED TO GENERATE THE "/DSA1" SIGNAL.

THE "/AS00" LINE IS THE ADDRESS STROBE SIGNAL FOR THE 68000 SYSTEM.

THE "/DSA1" LINE IS THE DATA STROBE ACKNOWLEDGE #1 SIGNAL FOR THE 68020. THIS INDICATES THE COMPLETION OF A 16BIT DATA TRANSFER. (THIS UNIT INDICATES THAT ALL XFERS ARE 16BIT)

FPA PAL #5a Coding

PAL16R4B

```
FPA5A
FLOATING POINT ACCELERATOR VERSION 1.0
JDN@CMU
CLK NC1 NC2 /AS20 /AS20X /CPDSA1 /DTACK /DI1 NC3 GND
/OE NC4 NC5 NC6 /DI2 NC7 NC8 NC9 /DSA1 VCC

DI2 :=          DTACK * AS20 +                    ; /DSA1 INTERMEDIATE #2
                DI2 * AS20

IF (VCC) DSA1 = CPDSA1 +                          ; 68020 DATA STRB ACK #1
                DI1 +
                DTACK * AS20X +
                DI2

FUNCTION TABLE

;INPUTS
```

```
/AS20 /AS20X /CPDSA1 /DTACK /DI1 /OE
;OUTPUTS
/DI2 /DSA1

;
;       /
;  /    C    /
;/ A    P    D                       /
;A S    D    T    /    /             D
;S 2    S    A    D    /    D        S
;2 0    A    C    I    O    I        A
;0 X    1    K    1    E    2        1
;
;I I    I    I    I    I    Q        S
-----------------------------------------------------------------------------------------------------------
 H H    H    H    X    H    Z    X    ;1 TRI-STATE TEST
 H H    H    H    X    H    Z    X    ;2
 H X    H    H    H    L    H    H    ;3 IDLE AFTER RESET
 L H    H    H    H    L    H    H    ;4 TEST /AS00X
 L L    H    H    H    L    H    H    ;5  AND /AS20X
 L L    H    H    H    L    H    H    ;6
 H L    H    H    H    L    H    H    ;7
 H X    H    H    H    L    H    H    ;8 TEST /VMA    C=1
 H X    H    H    H    L    H    H    ;9              2
 H X    H    H    H    L    H    H    ;10             3
 H X    H    H    H    L    H    H    ;11             4
 H X    H    H    H    L    H    H    ;12             5
 H X    H    H    H    L    H    H    ;13             6
 H X    H    H    H    L    H    H    ;14             7
 H X    H    H    H    L    H    H    ;15             8
 H X    H    H    H    L    H    H    ;16             9
 H X    H    H    H    L    H    H    ;17             10
 H X    H    H    L    L    H    L    ;18             1
 H X    H    H    L    L    H    L    ;19 TEST /DI1     2
 H X    H    H    L    L    H    L    ;20             3
 H X    H    H    H    L    H    H    ;21             4
 H X    H    H    H    L    H    H    ;22             5
 H X    L    H    H    L    H    L    ;23 TEST /DSA1    6
 H L    H    H    H    L    H    H    ;24             7
 H H    H    H    H    L    H    H    ;25             8
 H H    H    H    H    L    H    H    ;26 BAD /DSA1
 L H    H    H    H    L    H    H    ;27  (AS20)
 L L    H    H    H    L    H    H    ;28  (AS20X)
 L L    H    L    H    L    L    L    ;29  (DTACK)
 L L    H    H    H    L    L    L    ;30  (AS20-)
 H L    H    H    H    L    H    H    ;31  (AS20X)
 H H    H    H    H    L    H    H    ;32  (IDLE)
 H H    H    H    H    L    H    H    ;33 NORMAL /DSA1
 H H    H    H    H    L    H    H    ;34  (AS20)
 L L    H    H    H    L    H    H    ;35  (AS20X)
 L L    H    L    H    L    L    L    ;36  (DTACK)
 H L    H    H    H    L    H    H    ;37  (DSA1)
 H H    H    H    H    L    H    H    ;38  (IDLE)
 H H    H    H    H    L    H    H    ;39 /DSA1 TESTS
 H H    L    H    H    L    H    L    ;40  (CPDSA1)
 H H    H    H    H    L    H    H    ;41  (IDLE)
 H H    H    H    L    L    H    L    ;42  (DI1)
 H H    H    H    H    L    H    H    ;43  (IDLE)
 L H    H    L    H    L    L    L    ;44  (DI2)
 H L    H    H    H    L    H    H    ;45
 H H    H    H    H    L    H    H    ;46  (IDLE)
-----------------------------------------------------------------------------------------------------------
```

DESCRIPTION:

THIS PAL IS USED TO GENERATE ADDRESSING STROBES AND THE VMA SIGNAL.

INPUTS:

THE "CLK" LINE IS THE FUNDAMENTAL CLOCK FROM THE 68010 AND SHARED WITH THE OTHER PALS, THE 68020, AND THE 68881.

THE "/AS20" LINE IS THE 68020'S ADDRESS STROBE SIGNAL.

THE "/DS20" LINE IS THE 68000'S DATA STROBE SIGNAL. THIS SIGNAL IS GENERATED BY PALU6, AND IS USED IN THE GENERATION OF THE "/DSA1" OUTPUT.

THE "QA-QD" LINES ARE FROM THE PALU4'S DECADE COUNTER. THESE SIGNALS ARE USED TO GENERATE THE "/VMA" OUTPUT.

THE "/CPDSA1" LINE IS THE CO-PROCESSOR DATA STROBE ACKNOWLEDGE SIGNAL NUMBER 1 FROM THE 68881.

THE "/DTACK" LINE IS THE DATA TRANSFER ACKNOWLEDGE SIGNAL FROM THE 68000 SYSTEM. IT IS USED TO GENERATE THE 68020'S "/DSK1" SIGNAL.

THE "/CPCS" LINE IS THE CO-PROCESSOR CHIP SELECT USED TO ENABLE THE 68881 AND USED IN THIS PAL TO INHIBIT SENDING THE "/AS" SIGNAL TO THE 68000 SYSTEM WHEN THE 68020 IS TALKING TO THE 68881.

THE "/VPA" LINE IS THE VALID PERIPHERAL ADDRESS SIGNAL THAT INDICATES THAT A 6800 PERIPHERAL HAS BEEN SELECTED, SO THE 68020 CAN GET A DELAYED "/DSA" SIGNAL, THIS IS ALSO USED TO GENERATE THE "/VMA" SIGNAL TO THE 68000 SYSTEM.

THE "/OE" LINE IS THE OUTPUT ENABLE SIGNAL FOR THE PAL AND IS PERMENENTLY ENABLED.

OUTPUTS:

THE "/VMA" LINE IS THE VALID MEMORY ADDRESS SIGNAL SENT TO THE 68000 SYSTEM, FOR USE BY 6800 PERIPERALS.

THE "/AS00X" LINE IS A DELAYED ADDRESS STROBE SIGNAL FROM THE 68020.

THE "/AS20X" LINE IS THE DELAYED ADDRESS STROBE SIGNAL FROM THE 68020.

THE "/DI1" AND "/Z3" LINES ARE INTERMEDIATE SIGNALS USED TO GENERATE THE "/DSA1" SIGNAL.

THE "/AS00" LINE IS THE ADDRESS STROBE SIGNAL FOR THE 68000 SYSTEM.

THE "/DSA1" LINE IS THE DATA STROBE ACKNOWLEDGE #1 SIGNAL FOR THE 68020. THIS INDICATES THE COMPLETION OF A 16BIT DATA TRANSFER. (THIS UNIT INDICATES THAT ALL XFERS ARE 16BIT)

FPA PAL #6 Coding

PAL16R4B
FPA6
FLOATING POINT ACCELERATOR VERSION 1.0
JDN@CMU (FROM MOTOROLA)
CLK /DS20 A0 SIZ1 SIZ0 /CPCS /AS20 /AS00 /BG20 **GND**
/OE /BG00 /LDS NC NCX /DS20X /Z2 /UDS /DS00 **VCC**

DS20X := DS20 *; 68020 DATA STRB DELAYED*

Z2 := AS20 * /AS00 *; /BG00 INTERMEDIATE*

IF (**VCC**) DS00 = DS20 * DS20X *; 68000 DATA STRB*

IF (**VCC**) UDS = DS00 * /A0 *; 68000 UPPER DATA STROBE*

IF (**VCC**) LDS = DS00 * SIZ1 + *; 68000 LOWER DATA STROBE*
DS00 * /SIZ0 +
DS00 * A0

IF (**VCC**) BG00 = BG20 * /CPCS * /AS20 * /Z2 + *; 68000 BUS GRANT*
BG20 * /CPCS * AS00 * /Z2 +
BG20 * /AS20 * /Z2 +
BG20 * /AS20 * AS00 * /Z2

FUNCTION TABLE

;INPUTS
/DS20 A0 SIZ1 SIZ0 /CPCS /AS20 /AS00 /BG20 /OE
;OUTPUTS
/BG00 /LDS NC NCX /DS20X /Z2 /UDS /DS00

```
;
;                                  /
;/           / / / /     /         D       /
;D     S S   C A A B     B /       S     / D
;S     I I   P S S G /   G L     N 2 /   U S
;2 A   Z Z   C 2 0 2 O   0 D   N C 0 Z   D 0
;0 0   1 0   S 0 0 0 E   0 S   C X X 2   S 0
;
;I I   I I   I I I I I   S S   Q Q Q Q   S S
--------------------------------------------------------------
 H H   H H   H H H H H   X X   Z Z Z Z   X X   ;1 TRI-STATE
 H H   H H   H H H H L   X X   X X X X   X X   ;2 RESET
 H H   H H   H H H H L   H H   X X H H   H H   ;3 RESET
 L H   H H   H H H H L   H L   X X L H   H L   ;4 TEST /DS20X
 L H   H H   H H H H L   H L   X X L H   H L   ;5 TEST /DS00
 L L   H H   H H H H L   H L   X X L H   L L   ;6 TEST /UDS
 L H   H H   H H H H L   H L   X X L H   X L   ;7 TEST /LDS
 L L   H L   H H H H L   H L   X X L H   X L   ;8
 L L   L H   H H H H L   H H   X X L H   X L   ;9
 L L   L L   H H H H L   H L   X X L H   X L   ;10
 H H   H H   H L H H L   H X   X X H L   X H   ;11 TEST /Z2
 H H   H H   H H H H L   H X   X X X H   X X   ;12 TEST /BG00
 H H   H H   H L H L L   H X   X X X L   X X   ;13
 H H   H H   H L L L L   L X   X X X H   X X   ;14
 H H   H H   H H H L L   L X   X X X H   X X   ;15
 H H   H H   L L H L L   H X   X X X L   X X   ;16
--------------------------------------------------------------
```

DESCRIPTION:

THIS PAL IS USED TO GENERATE SYSTEM DATA STROBES, BUS GRANT, AND INTERNAL

LOGIC SIGNALS.

INPUTS:

THE "CLK" LINE IS THE FUNDAMENTAL CLOCK FROM THE 68010 AND SHARED WITH THE OTHER PALS, THE 68020, AND THE 68881.

THE "/DS20" LINE IS THE 68020'S DATA STROBE SIGNAL.

THE "A0" LINE IS THE 68020'S ADDRESS BIT 0. THIS IS USED IN THE GENERATION OF THE "UDS" AND "LDS" SIGNALS FOR THE 68000 SYSTEM.

THE "SIZ1" AND "SIZ0" LINES ARE USED TO SELECT THE SIZE OF THE 68020'S DATA TRANSFERS.

THE "/CPCS" LINE IS THE 68881'S CHIP SELECT LINE.

THE "/AS20" LINE IS THE 68020'S ADDRESS STROBE SIGNAL.

THE "/AS00" LINE IS THE ADDRESS STROBE SIGNAL BEING SENT TO THE 68000 SYSTEM.

THE "/BG20" LINE IS THE 68020'S BUS GRANT SIGNAL (I.E., "/BG").

THE "/OE" LINE IS THE OUTPUT ENABLE SIGNAL FOR THE PAL AND IS PERMENENTLY ENABLED.

OUTPUTS:

THE "/BG00" LINE IS THE BUS GRANT SIGNAL THAT IS SENT TO THE 68000 SYSTEM.

THE "/LDS" LINE IS THE LOWER DATA STROBE LINE GENERATED FROM 68020 SIGNALS AND SENT TO THE 68000 SYSTEM.

THE "/UDS" LINE IS THE UPPER DATA STROBE LINE GENERATED FROM 68020 SIGNALS AND SENT TO THE 68000 SYSTEM.

THE "NC" LINES ARE NOT CONNECTED.

THE "/DS20" LINE IS THE 68020'S "/DS" OUTPUT SIGNAL, DELAYED BY ONE CLOCK PERIOD AND THEN SENT TO THE 68000 SYSTEM.

THE "/Z2" LINE IS AN INTERMEDIATE VALUE USED IN THE GENERATION OF THE "/BG00" SIGNAL.

THE "/DS00" LINE IS THE STRETCHED DATA STROBE SIGNAL FOR THE 68000 SYSTEM.