

Design of a Large Scale MIMD Computer

by

Eleni Kapogiannis

Submitted to the Department of Electrical Engineering and
Computer Science

in partial fulfillment of the requirements for the degrees of
Bachelor of Science in Electrical Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer
Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

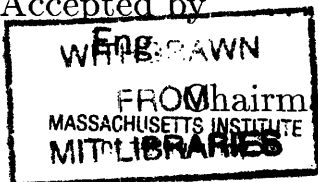
May 1994

© Massachusetts Institute of Technology 1994

Signature of Author
Department of Electrical Engineering and Computer Science
May 13, 1994

Certified by
Rodney A. Brooks
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Students



JUN 15 1994

Design of a Large Scale MIMD Computer

by

Eleni Kapogiannis

Submitted to the Department of Electrical Engineering and Computer Science
on May 13, 1994, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Electrical Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

With the increase in availability of low cost, high performance computing technology has come the ability to develop significant complex systems. A research group at the MIT Artificial Intelligence Laboratory is currently capitalizing on this computational technology in developing an almost humanoid system called Cog. This system is to include vision, sound input and output, and dextrous movement. The enabling technology for Cog is a large scale MIMD parallel computer named $\pi\beta$. This thesis focuses on the design and development of $\pi\beta$, including a detailed description of the design, reasons for the particular choice of architecture, and an evaluation of the computer's performance.

Many thanks are due to all the people who have supported me throughout my research work. I would especially like to thank my thesis supervisor, Professor Rodney Brooks, for the help, advice, and confidence which he gave me over the course of my research. This work would not have been possible without his constant support.

I would also like to thank the members of both the Cog and Mobot research groups at the MIT Artificial Intelligence Laboratory. The enormous amount of help and advice which I received from them is greatly appreciated.

Finally, I wish to thank my family and friends for their support during my studies at MIT. I would not have reached this point without them. Special thanks to my parents, Argie, and Christos who have provided encouragement through all of my endeavors in life.

Contents

1	Introduction	1
2	Background	4
2.1	Reasons for Parallelism	4
2.2	Types of Parallel Architectures	6
2.3	MIMD systems	8
2.3.1	Organization of Memory	8
2.3.2	Granularity	10
2.4	Analysis of and Comparison to Past Parallel Processor Systems . . .	11
2.4.1	Thinking Machines Corporation CM-5	11
2.4.2	BBN Butterfly Parallel Processor	12
2.4.3	PASM - Purdue University	13
2.4.4	Intel iPSC/860	14
3	Description of Hardware	16
3.1	Design of $\pi\beta$	16
3.1.1	Vesta SBC332	19
3.1.1.1	Motorola MC68332 microcontroller	19
3.1.2	Dual Port RAMs	21
3.1.3	Motor Board Port Connection	24
3.2	Interface from FEP to $\pi\beta$	25
3.2.1	SCSI Interface From FEP to InterFEP	26
3.2.2	Interface from Inter-Front End Processor to $\pi\beta$	30

4	Performance Evaluation and Future Work	35
5	Conclusion	38
A	Schematics of System	40

List of Figures

2-1	Hierarchy of Parallel Processor Architectures	7
3-1	Diagram of Backplane	17
3-2	Block Diagram of MC68332	20
3-3	Interface from FEP to $\pi\beta$	25
3-4	SCSI Hardware Interface From FEP to InterFEP	27
3-5	Logic for Communicating With Selected Processor	31
A-1	Detailed Schematic of a $\pi\beta$ Node	41
A-2	Schematic of InterFEP	42

List of Tables

3.1	Decoding of Data Transfer Sizes	24
3.2	Am53C80A Registers	28

Chapter 1

Introduction

Research in the design and development of parallel computer architectures has increased drastically in recent years, sparked by the availability of low cost microprocessors. Parallel computers avoid the bottleneck found on single processor computers of only being able to execute one instruction at a time. Thus, much of the research into parallel architectures has been focused on increasing the performance of serial computational problems through the use of parallel machines.

There has also been a noticeable increase in the use of parallel computer architectures for various applications within the field of artificial intelligence. Whereas classical approaches to artificial intelligence were based on centrally controlled systems, recent applications have tended toward the use of behavior based systems. The behavior based approach offers a large potential for parallel and distributed processing, which can be easily exploited by parallel hardware.

Though really complex behavior based systems have not yet been built, with the recent technological advances, the present seems to be an opportune time for the construction of a complex, intelligent system. With this in mind, a research group at the MIT Artificial Intelligence Laboratory, under the direction of Professors Rodney Brooks and Lynn Andrea Stein [Brooks and Stein, 1993], is currently developing an almost humanoid system named Cog. This system will include vision, sound input and output, and dextrous movement. The goal of this project is to increase our understanding of human cognition.

The enabling technology for Cog is a large scale, MIMD parallel computer which is called $\pi\beta$. This thesis is about the design and evaluation of $\pi\beta$.

Many different types of parallel computer architectures have been developed, each having its own strengths and weaknesses. The architectures differ in several ways, such as the number of instructions issued per cycle, the number of data streams operated on per cycle, the organization of memory, the choice of processor granularity, and the choice of interconnection topology. $\pi\beta$ is a loosely coupled, MIMD parallel computer, with a fixed topology network. This design was chosen as the one best suited for the control of the humanoid system.

Some may argue that $\pi\beta$ is not an optimal choice for the control of Cog. It would perform poorly on typical benchmark programs used in the evaluation of conventional parallel computers. Also, a very high performance single processor solution would yield a greater throughput and probably be more cost effective than $\pi\beta$. However, such an argument is made without a thorough understanding of the goal of the entire system.

Realizing that the overall goal of the project is to better understanding human cognition, we wish to find solutions which match what is already understood about biological systems. Since the architecture of an enabling technology affects the way in which solutions to the problem are implemented, putting various constraints on the design of $\pi\beta$ will guarantee some degree of biological relevance in the solutions. The constraints which this research group feels are important include a system which is totally asynchronous, has no global control, has no globally shared memory, and cannot transmit pointers between processors.

Furthermore, $\pi\beta$ was designed as a scalable system. This allowed subsystems to be implemented and left operating while other subsystems were still being developed. This corresponds to the layered approach [Brooks, 1986] to building intelligent systems, which has been used at the lab for several years. Such a scalable design, eventually reaching the required control for human level of cognition, is simply not possible using a single processor solution.

Therefore, $\pi\beta$ was designed with the above constraints in mind. It is not designed

to be the highest performance parallel processor, but it is designed to be well suited for the control of the humanoid robot. The reasons for $\pi\beta$'s choice of architecture, a detailed description of the design, and an evaluation of the computer's performance are each addressed in this thesis.

Chapter 2

Background

The constraints faced in designing $\pi\beta$ are somewhat different than those usually confronted in the design of large scale parallel machines. Most general purpose parallel computers are designed to act like fast serial computers, and they are typically intended to increase the performance and throughput of sequential numeric computations [Hwang and Degroot, 1989]. Many benchmark programs have been developed to measure the increase in performance yielded by these computers. Unlike these designs, however, $\pi\beta$ is constructed for a very specialized application. It is specifically designed to act as the enabling technology for Cog, which is an inherently parallel system. The idea behind the development of $\pi\beta$ is to take advantage of the available parallel hardware in order to simplify the task of controlling the humanoid robot. Due to the fundamental differences constraining the design of $\pi\beta$ with regards to other large scale parallel processors, there are certain architectural distinctions in the resulting systems.

2.1 Reasons for Parallelism

A parallel structure was chosen for the control of the system for numerous reasons. Since the main goal of the overall project is to construct a humanoid system that is capable of learning and interacting with its surroundings, $\pi\beta$ must be able to respond to the inputs from the environment on a real-time scale. A great amount

of data will be continuously sensed by Cog, and will need to be processed by the computer. This sensory data will come from the cameras providing for vision, the microphones allowing for hearing, the 'skin' used for the detection of touch, motor data which assists in detecting position, as well as various other sensors on the body of Cog. The controlling computer must be able to quickly process and respond to this incoming data. A serial computer is simply unable to handle the vast amounts of inputs to the system and respond on a real-time scale. Parallelism offers the ability to simultaneously receive and process the data at various processor nodes of the system, and thus react to the environment on a human time scale. In addition to processing the incoming data quickly, parallelism enables us to servo the actuators on the humanoid at high rates, ensuring a quick response from the humanoid to its environment. Furthermore, through the use of parallelism, one is able to coordinate the many actuation systems on Cog which need to be synchronized. For example, the movement of the eye and neck motors must be coordinated so that stable images can be received [Brooks and Stein, 1993]. Hence, parallelism provides for the simultaneous processing of various inputs to the system, as well as the quick response of the many outputs of the system, all resulting in the ability to control a real-time application.

A second reason for using a parallel architecture is that it allows us to add functionality to different processor nodes, without needing to recompile, reload, and restart the entire system. Thus, lower level functions which have been implemented and are operating as expected may be left running continuously on the system, while new functionality is being added or debugged at other processor nodes. This permits us to get entire subsystems of the humanoid up and running, while other subsystems are still being developed or debugged.

One of the most important reasons for selecting a parallel architecture is that it allows us to take advantage of the inherent parallelism in the humanoid system. Recently, development in artificial intelligence has seen a switch from the use of centralized systems to that of distributed systems. Just as the human body is a highly parallel system, so is the control of the humanoid system. $\pi\beta$ is proposing to model the brain slightly above the neural level. Thus, at one level, groups of

processors will be responsible for the control of such functions as lower level motor control (cerebellum) and facial recognition (cortex). At a slightly different level of parallelism, various processors will be dedicated to controlling complete subsystems such as the visual system or the hearing system. Therefore, the control of these obviously parallel systems will be simplified through the use of parallel hardware.

2.2 Types of Parallel Architectures

There are several classifications of parallel processors which are in use today. Figure 2-1 shows a hierarchy of the various types of architectures which are commonly used [DeCegama, 1989]. The majority of parallel processors are made up of two or more von Neumann-based processors. The two other types of parallel processors which have been developed use either the dataflow or reduction approach, both of which differ from the von Neumann-based machines in that their instructions are not issued in a sequential order. I will focus only on the issues involved in designing von Neumann-based machines, since $\pi\beta$ is included under this classification.

Von Neumann-based parallel computers are further classified according to the number of instructions issued and the number of data streams active during a single cycle [Flynn, 1972]. Thus, there are four major categories of von Neumann-based parallel architectures: a single instruction working on a single data stream (SISD), a single instruction working on multiple data streams (SIMD), multiple instructions working on a single data stream (MISD), and multiple instructions working on multiple data streams (MIMD). The SISD architecture corresponds to the traditional von Neumann processor. SIMD architectures add the capability of working on multiple data streams, and they are widely used in modeling low levels of parallelism in a system. Typical examples of SIMD architectures include vector and array processors. The MISD category does not seem to offer any benefits for the control of various applications which have been studied, so this classification has not been studied in any detail. Finally, the MIMD category of parallel computers seems to offer the most flexibility in designing truly parallel systems. A great amount of time has been spent

Parallel Computers

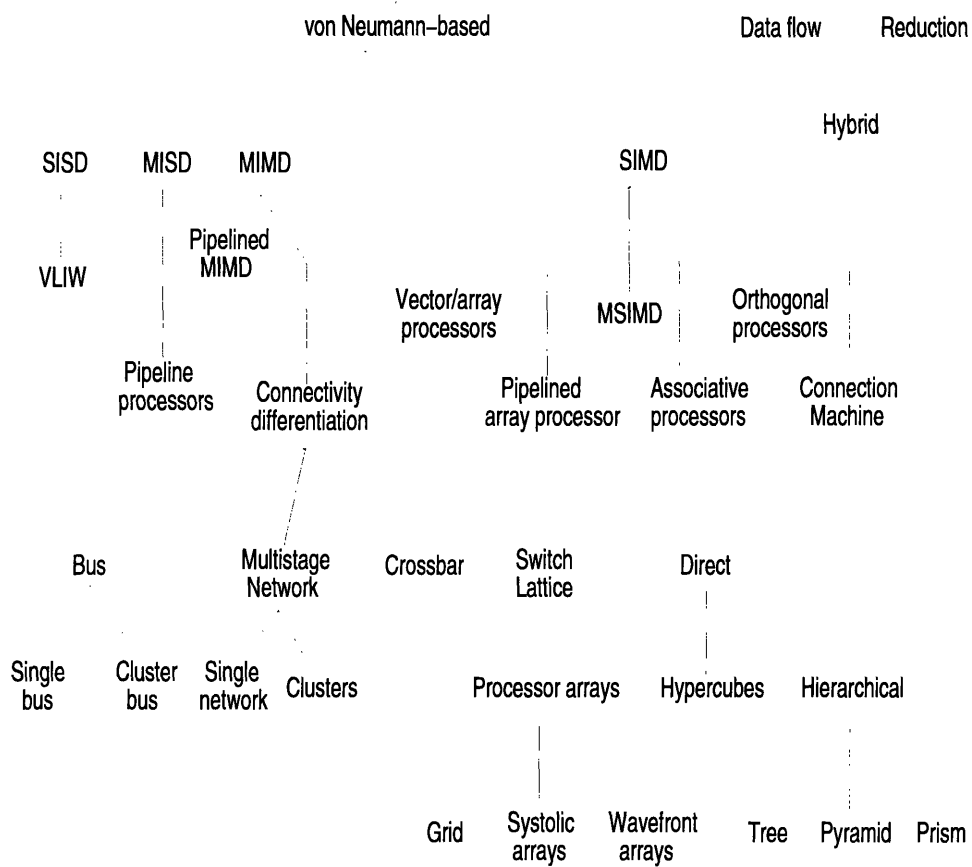


Figure 2-1: Hierarchy of Parallel Processor Architectures

developing MIMD architectures, and this category seems to offer the most hope in further improving the capabilities of parallel systems.

2.3 MIMD systems

$\pi\beta$ is classified as a MIMD parallel computer. The category of MIMD systems includes a wide variety of network architectures, each one geared towards improving a different performance specification. They differ in the way that memory is organized in the system, the interconnection topology, and the granularity of the processors. Certain types of MIMD architectures offer great performance for use as general parallel processors, while others are well suited for specialized applications. Thus, one needs to select the best architecture for the application in which it will be used.

2.3.1 Organization of Memory

The category of MIMD parallel processors can be further categorized in the way that memory is organized in the system. The two basic types of memory organization in MIMD computers are tightly coupled MIMD systems and loosely coupled MIMD systems. Tightly coupled systems consist of a global memory that is accessible to all the processors in the system. The bus, multistage networks, and crossbar topologies are examples of tightly coupled systems. These systems are often used as general purpose computers, since they easily allow the sharing of memory between any processors in the system. However, in order to keep the memory access time of these systems low, and thus keep the performance high, some sort of local cacheing scheme must be used [DeCegama, 1989]. This implementation can get rather expensive in hardware, since the problems of cache coherency and maintaining a consistent global memory must be addressed. Also, the performance of the entire system declines as more processors are added. Hence, though the cost of implementing a tightly coupled memory system is often worth the performance yield for general purpose computers, this does not carry over to many specialized parallel systems where the communication between the various processors is relatively fixed.

Alternately, loosely coupled systems are designed with each processor having its own local memory where the bulk of the processing is done. Communication among the processors is achieved via some sort of message passing scheme. Both dynamic and static interconnection implementations have been used in loosely coupled systems. The switch lattice is an example of a dynamic implementation, in which the connections allowing communication between processors are run-time flexible. This involves the use of some sort of high speed switching network, which can again get rather costly in hardware implementation. On the other hand, static interconnections are fixed at run-time. Certain designs have been made directly connecting each processor in the system to every other processor, but this is very impractical and costly for large systems. Thus, fixed topology networks are usually successful when dealing with applications that are easily broken up into sections, and where the communication between sections is relatively light and predictable. Examples of these types of networks are the processor array, hypercube, and tree topologies. These topologies work well with applications where communication between processors depends on locality. In this case, the processors are most likely to communicate with their nearest neighbors. The loosely coupled systems seem to offer a greater performance for the given cost when applied to certain tasks, and they are successful designs for more specialized applications.

The level of parallelism which is being modeled by $\pi\beta$ for the control of the humanoid system is slightly above the neural level. Thus, small groups of processors will be in charge of various subfunctions such as low level motor control (cerebellum) and facial recognition (cortex), as well as the control of entire subsystems such as the vision and hearing systems [Brooks and Stein, 1993]. Since the number of connections between the various parallel functions is rather small and static, a fixed topology network was chosen for $\pi\beta$. The connections between processors are flexible during the development stages of the system, and they can be made by the placement of a dual port RAM (DPRAM) board between any two processors which need to communicate. Cables are run from one port of each processor node to a port of the DPRAM, and this enables both processors to access the memory of the DPRAM. These connections

remain fixed at run-time. Since a fixed topology network was implemented, the issues of maintaining cache coherency and designing high speed switching networks were not involved. Therefore, a high performance system was able to be designed at a relatively low cost.

2.3.2 Granularity

Another classification of large scale MIMD computers involves the granularity of the parallel processing system. This reflects the size of the various processor nodes in the system. Task granularity is broken up into three main sizes: fine-grain, medium-grain, and coarse-grain parallelism.

Fine-grain parallel systems involve the use of a large number of simple, single bit processors connected together. This is the approach taken by neural networks and connectionist systems. These types of systems are trying to model parallelism at the neural level, where the basic processor element may be rather simple and slow, but the interconnection topologies are extremely complex [Almasi and Gottlieb, 1994]. This work has shown success in controlling smaller, local problems. However, for the complex application proposed for $\pi\beta$, the fine-grain processors simply do not have the capability of processing and reacting to the input data on a real time scale.

The complete opposite approach would be to use coarse-grain parallelism. This involves the use of a small number of very high performance computers. In designing $\pi\beta$ for the control of Cog, the group is trying to find parallel solutions to the control problem. In keeping with the level of parallelism that is being modeled, such high-performance computers would be underused.

Therefore, medium grain processors were chosen to be used in $\pi\beta$. They appear to have the right amount of performance capabilities to successfully control the humanoid system. The choice of medium granularity also enabled $\pi\beta$ to make use of available, low cost, moderate performance microprocessors. The processors which were actually used are the Motorola MC68332 microcontrollers. By capitalizing on their availability, a high performance parallel computer was able to be developed at a relatively low cost.

2.4 Analysis of and Comparison to Past Parallel Processor Systems

Several MIMD parallel systems have been developed over the last decade. These systems are all similar to a certain degree, but contain specific architectural differences from each other and from $\pi\beta$. Several of these systems are discussed in this section, with an emphasis on the reasons for their architectural differences.

2.4.1 Thinking Machines Corporation CM-5

The CM-5 connection machine available from Thinking Machines Corporation is an extremely powerful distributed memory MIMD machine. It consists of a fat tree interconnection topology, where links higher up in the tree structure have greater bandwidth. Thus, one is able to keep the clock constant and use wider busses near the root of the tree. This increases performance by assuming that more communication will occur at levels higher up in the tree. Each processor is connected to its four nearest neighbors, again using the principle of locality in assuming processors near each other will communicate most often. Thus, local communication is achieved at the rate of 20 Mbyte/sec, while random communication is executed at a rate of 5 Mbyte/sec. The CM-5 is a coarse-grain parallel processor, with each processor node containing a Sparc microprocessor and 32 Mbytes of local memory. As of last year, the machine was able to include up to 16K nodes because of restrictions on cable lengths, but it is designed to include up to 256K nodes [Almasi and Gottlieb, 1994].

The CM-5 offers excellent performance capabilities for a large variety of applications. $\pi\beta$ can in no way compare with the performance results of the CM-5. There are some very strong reasons, however, that favor the use of $\pi\beta$ for the control of Cog. One main reason is that the Sparc microprocessors would be greatly underused in the way that parallelism is implemented in the humanoid system. The processing power of each node in $\pi\beta$ is purposely limited so that parallel solutions to the task can be found, ensuring that the solutions retain some sort of biological relevance.

The biggest reason against using a design such as the CM-5 is simply the cost of the system. As of last year, a full 16K node system would cost about one-half billion dollars [Almasi and Gottlieb, 1994]. This corresponds to a cost of over thirty thousand dollars per processor node in the CM5. In comparison, each node of $\pi\beta$ costs only a couple of hundred dollars. Thus, $\pi\beta$ is clearly a much cheaper system. Since the CM-5 is obviously much more powerful than necessary and is not geared toward the grain of parallelism which is being modeled in Cog, it would make a poor choice of an enabling technology. $\pi\beta$ admittedly has a much lower performance, but it still has the performance necessary to control the real-time humanoid system. Therefore, though the CM-5 is admittedly a powerful computer, it is not necessarily the best choice for a given application.

2.4.2 BBN Butterfly Parallel Processor

The Butterfly Parallel Processor developed by BBN Labs in Cambridge, MA is a tightly coupled, shared memory multiprocessor system. This processor was used in computer vision applications. The system can include up to 256 processors, with each processor node consisting of a Motorola 68020 and 68881 floating-point coprocessor, and 4 Mbytes of local memory [Hwang and Degroot, 1989]. The granularity of this system is very similar to that of $\pi\beta$, with similar Motorola processors used in both designs, and the BBN Butterfly having slightly more local memory than that used in $\pi\beta$.

The major difference between the BBN Butterfly processor and $\pi\beta$ is in the way in which communication between processors is achieved. The BBN Butterfly machine has a multistage butterfly switching network that allows all of the local memories to form a collective global memory. A local memory access takes about 2 microseconds, whereas accessing a remote memory location through the network takes about 6 microseconds [Hwang and Degroot, 1989]. $\pi\beta$ has approximately the same performance capabilities as the BBN Butterfly Parallel Processor, though it does not need to make use of the high speed switching networks. Since inter-processor communications are fixed at run-time in the control of the humanoid, not all of the processors need to

communicate. Thus, the BBN would be better at handling more general purpose parallel processor applications, but it does not yield any better performance over that of $\pi\beta$ for the control of Cog. Thus, by examining the specific features of the application to be controlled, $\pi\beta$ was designed to yield a higher performance per cost in controlling the humanoid robot than the BBN Butterfly Processor.

2.4.3 PASM - Purdue University

The PASM system of Purdue University is a parallel processing system similar to $\pi\beta$. The PASM system can be configured as either a SIMD or MIMD system, and it was used as a way of studying the use of parallelism in computer vision, speech processing, and expert systems [DeCegama, 1989]. PASM included numerous processing elements, with each processing element consisting of a general purpose microprocessor where the bulk of the processing was done, and memory modules for data and instruction storage. This is similar to the structure of $\pi\beta$, where, as will become clear later on in this thesis, the Vesta SBC332 board is essentially the same as a PASM processing element.

The PASM system was expandable to include over 1000 processing elements, each with its own physical address. The processing elements were connected in a general cube network [DeCegama, 1989]. Supposing the system contained 'N' processing elements, the cube supports up to 'N' simultaneous transfers, where microcontrollers were used to direct the activities of the processing elements in MIMD mode [DeCegama, 1989]. This type of network offers diverse implementation options, and because of the connections between local processors, many different subsystems can be used simultaneously. The PASM architecture also has the advantage of a certain amount of reliability. If one processing element fails, only those processing elements which rely on the failed processor must be disabled. The other functional processing elements can continue operating as usual, and thus the entire system does not need to be shut down.

$\pi\beta$ is very similar to the PASM system. The major difference lies in the interconnection structure of the two systems. Instead of the cube interconnection system,

$\pi\beta$ makes use of a point-to-point interconnection scheme, with sparsely connected nodes. Each processor in $\pi\beta$ can be connected to any other processor in the system via a dual port RAM. This gives $\pi\beta$ a little more flexibility over PASM, which was restricted by the cube configuration. Furthermore, $\pi\beta$ does not need the use of microcontrollers to direct the communication between processors, since the dual port RAMs are directly connected to both processors which are sharing information. The only circumstance in which the cube interconnection system would be better than the point-to-point system, is if the processors communicating together changed rapidly, and many processors needed to directly communicate with each other. Since Cog did not require such a high number of processors to communicate with each other, and the processors which need to communicate are relatively fixed, the point-to-point system worked well. $\pi\beta$ offers the same advantages as PASM, while allowing a simpler interconnection scheme. Therefore, the overall performance per cost of $\pi\beta$ was greater than that of PASM.

2.4.4 Intel iPSC/860

The Intel iPSC/860 is a MIMD parallel machine with a hypercube interconnection topology. The computer is designed to support anywhere from 8 to 128 nodes, with an 'N' node hypercube having each processor connected to its $\log_2 N$ nearest neighbors [Hwang and Degroot, 1989]. The Intel iPSC/860 had the Intel i860 processor at each node of the system, which is a 64-bit RISC processor and runs at 40 MHz. Internode communication was achieved at 5.6 Mbytes/sec [Almasi and Gottlieb, 1994].

The hypercube structure has many similar properties to the fixed topology network of $\pi\beta$. The Intel iPSC has a slightly more powerful processor at each node, and thus inter-processor communications are slightly faster on the iPSC. Also, the iPSC uses the principle of locality, where nearest neighbor communication is much faster than general communication. Hence, it is again clear that the iPSC is a much better general purpose parallel processor than $\pi\beta$. However, the flexibility which $\pi\beta$ offers us during the development phase in selecting which processors will communicate help make the addition of new functionality to the system a much simpler task. Therefore,

while $\pi\beta$ does have a slightly lower performance than the Intel iPSC/860, it offers us a greater flexibility during the development phase, while still maintaining the required performance capabilities.

In the comparison of the above systems, it is clear that $\pi\beta$ is not designed to be used for conventional parallel processing applications. There are much more powerful computers available which are designed for that purpose. However, it should also be clear that given the constraints of the humanoid system, $\pi\beta$ meets the necessary requirements while offering a greater performance per cost over other available parallel machines. Therefore, $\pi\beta$ was successfully designed to meet the requirements for the control of the humanoid, while maintaining a relatively low overall cost for the system.

Chapter 3

Description of Hardware

The hardware design of the massively parallel system consisted of two main parts. The first of these involved the design of the actual parallel processing system, $\pi\beta$, while the second involved the design of an interface between a MacIntosh computer acting as a front end processor (FEP) and $\pi\beta$. This interface to the FEP was provided for both file serving and debugging purposes.

3.1 Design of $\pi\beta$

The design of $\pi\beta$ was made modular by breaking the overall parallel processing system into identical sections, which will be referred to as backplanes. $\pi\beta$ is expandable to contain up to sixteen backplanes, with each backplane consisting of sixteen processing elements. Figure 3-1 shows a diagram of the backplane, where the major components include Vesta SBC332 boards, ports allowing connections to IDT7025 Dual Port Static RAMs, connections enabling communication to numerous motor controller boards, and an interface to the front end processor.

The Vesta SBC332 Board was used as the basic processing element of the system, with each board containing a processor paired with its own local memory. Thus, $\pi\beta$ contains a large number of these processing elements, with each one dedicated to the control of a specific subsystem of Cog. For example, one might be dedicated to the control of the cameras used for the vision system, while another may be dedicated to

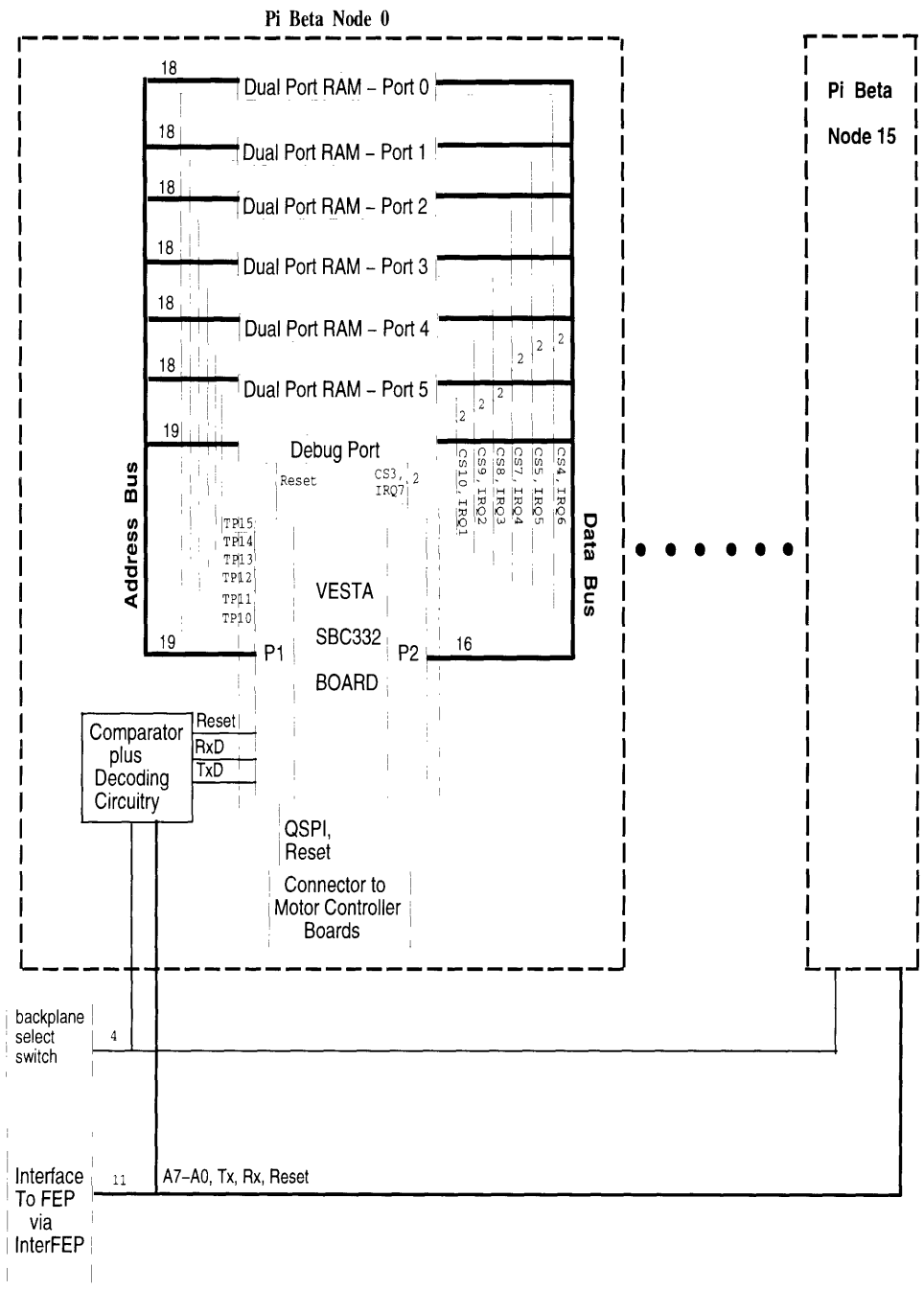


Figure 3-1: Diagram of Backplane

the control of the arm motors. On a higher level of parallelism, a group of these processing elements may be dedicated to the control of the vision system, while another group may be dedicated to the control of the hearing system, and still a third group may take on the role of motor control usually associated with the cerebellum.

Though each of the processing elements operates independently, communication between certain processors is required for the proper control of the humanoid system. This is the case since many of the processors rely on information obtained from other processors in order to perform their function. One system in which processor communication is required is the vision system. The processor dedicated to determining the position of an object in sight would require information from both the eye and neck motors in order to identify the precise location of the object relative to Cog. If the eye motor was at a position of thirty degrees relative to the head, and the neck motor was at a position of thirty degrees relative to the body, the object would be at a position of sixty degrees relative to the humanoid. Thus, it is crucial that the processor responsible for determining the position of objects obtain all of the necessary information. If the eye and neck motors are controlled by different processors, the processor determining the position must be able to communicate with these other processors and acquire the necessary information.

In $\pi\beta$, communication between processors is accomplished through the use of dual port RAMs (DPRAMs). Two processors are able to share the memory of a DPRAM and communicate by passing information over this shared memory. There are six ports available per processor, each of which may be connected to a DPRAM. Thus, each processor node on $\pi\beta$ is able to communicate with up to six other processors.

Each processor node on $\pi\beta$ also has an interface allowing for communication to various motor controller boards. This interface allows for the exchange of motor control information between the processor node on $\pi\beta$ and the processor on the motor controller board. Thus, the $\pi\beta$ processor is able to monitor the operation of the specified motor through the information it receives, as well as send information changing the position and/or velocity of that particular motor.

3.1.1 Vesta SBC332

The main component of $\pi\beta$ is the Vesta SBC332 board, which is a commercially available board manufactured by Vesta Technologies, Inc [Ves, 1993]. Each Vesta board is a complete standalone unit similar to the Motorola Business Card Computer. The Vesta SBC332 board contains a Motorola MC68332 microcontroller, sockets for on-board RAM and EPROM up to 1 Mbyte each, a continuous-cycling external watchdog timer, a power supply monitor, and an on-board reset switch. Thus, each Vesta board functions as a self-contained computer with its own local memory. There are two sixty four pin connectors on the Vesta board, labeled P1 and P2, which allow access to the signals coming off the Motorola MC68332 microcontroller. Each Vesta SBC332 board is mounted onto the backplane via the connectors P1 and P2, and the backplane is thus able to access the MC68332 processor as needed.

3.1.1.1 Motorola MC68332 microcontroller

The Motorola MC68332 is a 32-bit microcontroller which runs at 16.78 MHz. It is upwardly compatible with the Motorola M68000 family of processors. A block diagram of the MC68332 is shown in Figure 3-2 [Mot, 1990], where the major components include the CPU32, the timer processor unit (TPU), and the queued serial module (QSM). Both the TPU and the QSM modules are stand alone subsystems which, after having been configured by the host CPU, require little or no CPU intervention to operate.

The TPU is an intelligent peripheral module which provides sixteen different channels, each associated with a specific pin on the MC68332, to be used in time-related functions. The TPU can be used for simple activities such as discrete input/output, or more complicated functions such as pulse width modulation. Complicated timing functions were not necessary for the control of Cog, so, in $\pi\beta$, the TPU lines are simply used as discrete outputs. These outputs are completely controlled by the CPU32, and they can be set high or low only upon request from the CPU. The TPU lines were used in the interface to the dual port RAMs, where they controlled the semaphore inputs to the dual port RAMs. This is further discussed in the section describing the

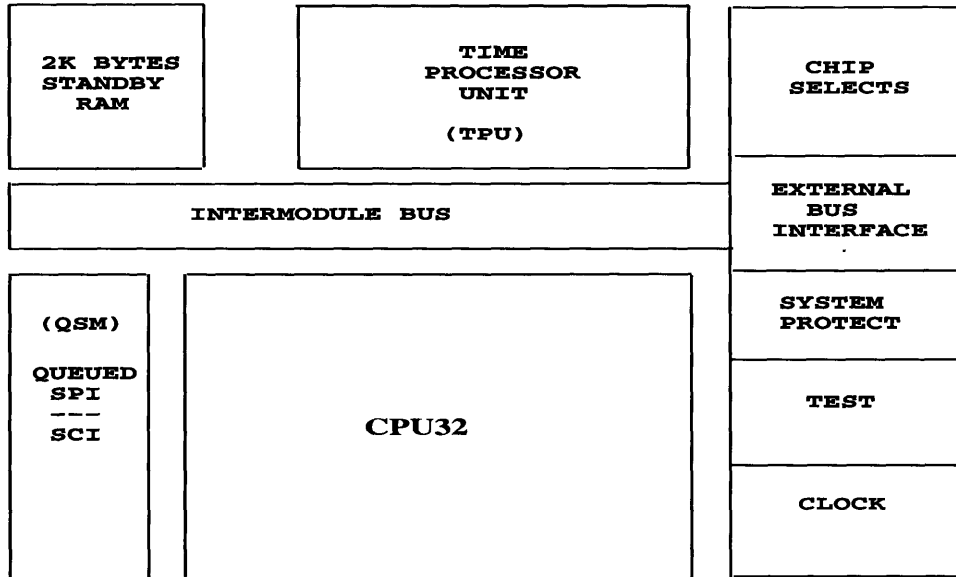


Figure 3-2: Block Diagram of MC68332

functionality of the dual port RAMs.

The queued serial module provides two serial communication interfaces: the queued serial peripheral interface (QSPI) and the serial communications interface (SCI). The QSPI is a synchronous serial communications interface intended for use in communicating with peripherals and other MCUs. It consists of seven signals: master in slave out (MISO), master out slave in (MOSI), the serial clock (SCK), and four select lines (PCS3-PCS0). Thus, the processor is able to select one of sixteen peripherals by asserting the correct address on the lines PCS3-PCS0, and then sending serial information out over the MOSI line or reading serial information in over the MISO line. In $\pi\beta$, the QSPI is used to interface to various motor controller boards. This interface is discussed in the section describing the communication to the motor boards.

The SCI on the MC68332 is used to communicate to a MacIntosh computer acting as a front end processor (FEP). This connection is needed for file serving purposes as well as for the debugging of the various processor nodes. This interface is discussed in detail in section 3.2.

3.1.2 Dual Port RAMs

The dual port RAM (DPRAM) used in $\pi\beta$ is the IDT7025 high speed 8K X 16 dual port static RAM [Int, 1990]. The two ports on the IDT7025, referred to as 'left' and 'right', are completely independent, with separate control, address, and data lines. Thus, each port has asynchronous access to any memory location, regardless of whether the other port is also accessing memory or not. The memory access time of the DPRAM is 55ns. This is fast enough to respond to the MC68332 memory accesses, since the MC68332 runs at approximately 60ns. The DPRAM currently provides for inter-processor communications as well as an I/O interface for the vision system. Hardware for capturing sound input is currently under development, and this system will also interface to the MC68332 via the DPRAM.

There are several features of the IDT7025 DPRAM which make it an appealing choice for the communications interface. One feature is that it has on-chip power down circuitry controlled by the \overline{CE} line which puts the chip in standby mode when it is not selected. The standby mode allows the chip to consume very little power when it is not being used.

Another feature of the IDT7025 is that each port has an interrupt line. On the left port, this line ($\overline{INT_L}$) is set when the right port writes to memory location 0x1FFE. Similarly, on the right port, this line ($\overline{INT_R}$) is set when the left port writes to memory location 0x1FFF. To clear the interrupt, the respective ports need to read the memory location that triggered the interrupt. Thus, the left port needs to read address 0x1FFE, and the right port needs to read address 0x1FFF. The interrupt lines from this chip are connected to the interrupt lines on the MC68332 and are used for the vision system of Cog. The lines IRQ1 through IRQ6 on each MC68332 processor are connected to the interrupt outputs from the DPRAMs on ports five through zero, respectively. A frame grabber board is interfaced to a MC68332 processor node on $\pi\beta$ through the DPRAM. The frame grabber continuously writes frames to the DPRAM, and the interrupt is set when it finishes writing the last byte of a 128 X 128 Kbyte frame. Thus, the processor receives an interrupt when a complete frame has been written, and it may access any portion of that frame by reading the memory of

the DPRAM. Information is obtained from the cameras of the vision system in this manner.

One extremely useful feature on the IDT7025 DPRAM is the on-chip hardware support of semaphore signalling between ports. Problems may arise in the system if the two ports on the DPRAM both try to access the same location in memory at the same time. Though simultaneous reads provide no problems, simultaneous writes or a simultaneous read from one port and write from another may result in faulty data being passed between processors. The IDT7025 provides one semaphore line for each port ($\overline{SEM_R}$ and $\overline{SEM_L}$). The on-chip semaphore logic on the IDT7025 is a set of eight latches. These latches are independent of the DPRAM, and may be accessed by asserting the semaphore line of the corresponding port. The address lines A2-A0 are used to select the latch, and data pin D0 is used to set or reset the latch. Thus, if two processors are sharing a portion of memory on the DPRAM, when a processor wishes to access the shared memory, it requests a token by writing a zero to the agreed latch in the semaphore logic. The processor then reads back this value. If it is a zero, the semaphore was successfully set, and the processor has gained control and may modify the memory as needed. If, however, when the latch value is read back it is a one, this means that the other port currently has access to the shared memory. In that case, the processor can continue to wait until the other side relinquishes the token, or it may go on to perform another function and later return to try and gain control of the token. The IDT7025 also provides semaphore logic to take care of the case of both ports attempting to gain control of the semaphore token. If simultaneous token requests are made from both ports, the chip will arbitrarily select one side to receive the token. Therefore, by using the semaphore logic, one is able to ensure that valid data is consistently transmitted between the two ports.

As was mentioned previously, the semaphore line inputs are controlled by the CPU of the MC68332 microcontroller. TPU lines ten through fifteen from the Vesta board connector P1 are connected to the semaphore line input on the DPRAMs in ports zero through five, respectively. Thus, a processor can request a token from a specific DPRAM by asserting the semaphore line input and selecting the correct DPRAM

port. In this manner, the message passing technique between two processors was guaranteed.

Appendix A-1 contains a detailed schematic of a $\pi\beta$ processing node where the connections to the DPRAM boards are clearly shown. The address and data lines of the DPRAMs are connected via a cable to those on the MC68332, and chip selects CS4, CS5, and CS7 through CS10 are connected to the chip enable inputs of the DPRAMs on ports zero through five, respectively. The MC68332 may assert the chip select output corresponding to the selected DPRAM, and may thus exchange data with the processor connected to the other port of the DPRAM. The read/write output (R/\overline{W}) from the MC68332 is connected to the input/output write input (\overline{IOW}) of the DPRAM, enabling the transfer of information from the selected dual port RAM to the MC68332. Likewise, the input/output read input (\overline{IOR}) of the DPRAM is connected to the inverted R/\overline{W} signal from the MC68332, enabling the transfer of information from the MC68332 to the selected DPRAM.

The first version of $\pi\beta$ provided for only word addressing of the DPRAMs, with the upper byte (\overline{UB}) and lower byte (\overline{LB}) inputs to the DPRAMs tied low. This required the use of different code to access the DPRAM than that used to access other parts of memory. Hence, it had the result of immensely complicating the system. For this reason, byte addressing functionality is being added to the DPRAM interface in version two of the backplane, which is currently under development. The logic for the control of the \overline{UB} and \overline{LB} inputs is as follows:

$$\overline{UB} = \overline{SIZ1} * SIZ0 * \overline{A0}$$

$$\overline{LB} = \overline{SIZ1} * SIZ0 * A0$$

The SIZ1 and SIZ0 outputs of the MC68332 indicate the length of data being transferred as is shown in table 3.1 below. $\pi\beta$ currently does not use transfer sizes greater than a word, but such transfer sizes could be used in the future if desired. The A0 address bit is asserted when addressing the upper byte of a word, and is otherwise not active. Thus, by using the above decoding logic, a word as well as a particular

SIZ1	SIZ0	Data Transfer Size
0	1	byte
1	0	word
1	1	3 byte
0	0	long word

Table 3.1: Decoding of Data Transfer Sizes

byte of a word are able to be transferred between the MC68332 processor and the DPRAM.

The second version of $\pi\beta$ will also include drivers on the address and data signals sent to the DPRAMs. The transfer of information to and from the DPRAMs in version one was not robust when long cables were used in the connection from the backplane to the DPRAM board. Thus, processors relatively far from each other would sometimes lose some information that should have been passed between them. By buffering these signals, the second version of the backplane should provide for stable message passing via relatively long cables.

3.1.3 Motor Board Port Connection

There are several motors on Cog which are used to control its body movements as well as its eye movements. The actuators on the various motors are controlled by a motor controller board whose main component is the Motorola MC6811 micro-controller. The MC6811 exchanges control information with an MC68332 processor node on $\pi\beta$. The interface between the MC68332 and the MC6811 is achieved using the QSPI port, which was described in the section dealing with the MC68332. The MC68332 functions as the master, with the MC6811 taking on the role of the slave. Each processor node on $\pi\beta$ is able to communicate with up to sixteen of these motor controller boards, with each board controlling one of the motors on Cog. Each MC6811 processor has a specific identity associated with it, and the MC6811 that the MC68332 processor wishes to communicate with is selected by placing the correct address on the PCS3-PCS0 lines. Information is sent to the MC6811 over the MOSI line, while information is sent back to the MC68332 over the MISO line.

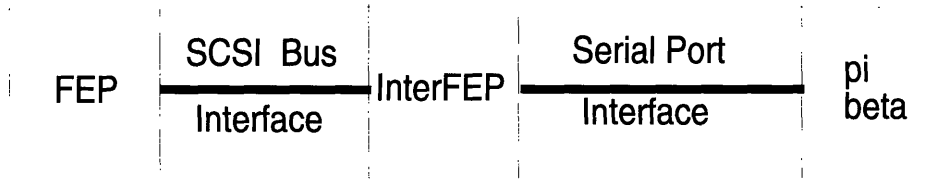


Figure 3-3: Interface from FEP to $\pi\beta$

The design of $\pi\beta$ currently has packets being exchanged with the motor controller boards at a rate of 32 Hertz as long as less than eleven motor controller boards are attached to the single processor. This rate drops down to 16 Hertz if this condition is not met. The MC6811 receives information for the velocity and position of the motor. It sends back to the MC68332 information about the encoder reading, the motor and driver temperature, and the motor current. From this information, the $\pi\beta$ node communicating with this MC6811 is able to ensure the correct operation of the motor and detect if any problems have occurred.

3.2 Interface from FEP to $\pi\beta$

Though the processors on the backplane all have the information necessary to control the specific function which they are in charge of, there is still a need to communicate with these processors. Users may need to communicate with a certain processor in order to pass various instructions to that processor. Also, users will need to know if a processor is functioning as expected. If not, the user may want to restart that processor and ignore any faulty results which it may have computed. In order to achieve this communication, an interface was set up between a MacIntosh computer dedicated as a front end processor (FEP) and $\pi\beta$. This interface entailed the use of a Motorola MC68332 similar to those used on the backplane to act as an intermediate front end processor (interFEP), with this interFEP connected between the FEP and $\pi\beta$ as shown in Figure 3-3.

3.2.1 SCSI Interface From FEP to InterFEP

The FEP is connected to the interFEP over the small computer system interface (SCSI) bus of the MacIntosh computer. SCSI provides a specification for connecting small computers with intelligent peripheral devices [Ame, 1986]. It is a local I/O bus which can sustain data transfer rates up to 4 Megabytes per second. The SCSI standard provides for two types of SCSI devices: an initiator and a target. As is implied by the name, the initiator is capable of beginning an operation, whereas the target is capable of responding to a request to perform a specific operation. The SCSI allows for multiple initiators as well as multiple targets. At the moment, $\pi\beta$ only requires the use of a single initiator (the FEP) and a single target (the Am53C80A SCSI interface controller connected to the interFEP). As newer versions of Cog are created and multiple existing versions will want to be used, this situation may change. One may want to have a different interFEP for each version of Cog, and the FEP would be able to select which version to communicate to through the SCSI bus. This functionality would require only minor software additions to the system, and is thus easily added to the overall system.

The hardware interface between the FEP and the interFEP is shown in Figure 3-4. The 25 pin connection coming out of the FEP is converted to a 50 pin interface with the ground signal alternated between the previous 25 signals through a standard MacIntosh converter cable. This interface, having alternating grounds between signals, allows for better transmission of the signals over the cables. The alternating ground wires help to eliminate any crosstalk between the signals being transmitted which could otherwise interfere with the proper transmission of these signals. The SCSI standard allows for cable lengths of up to six feet when using the single-ended fifty pin connection. Since the distance between the FEP and the backplane is actually about 10 feet, the interFEP board was mounted close to the FEP, keeping the cable length from the SCSI interface to about 2 feet. Next, the SCSI bus signals are brought into an Am53C80A SCSI interface controller. This chip provides an interface between a SCSI bus and a microprocessor. Thus, the Am53C80A connects directly with the SCSI bus signals and the interFEP, allowing the interFEP to control certain

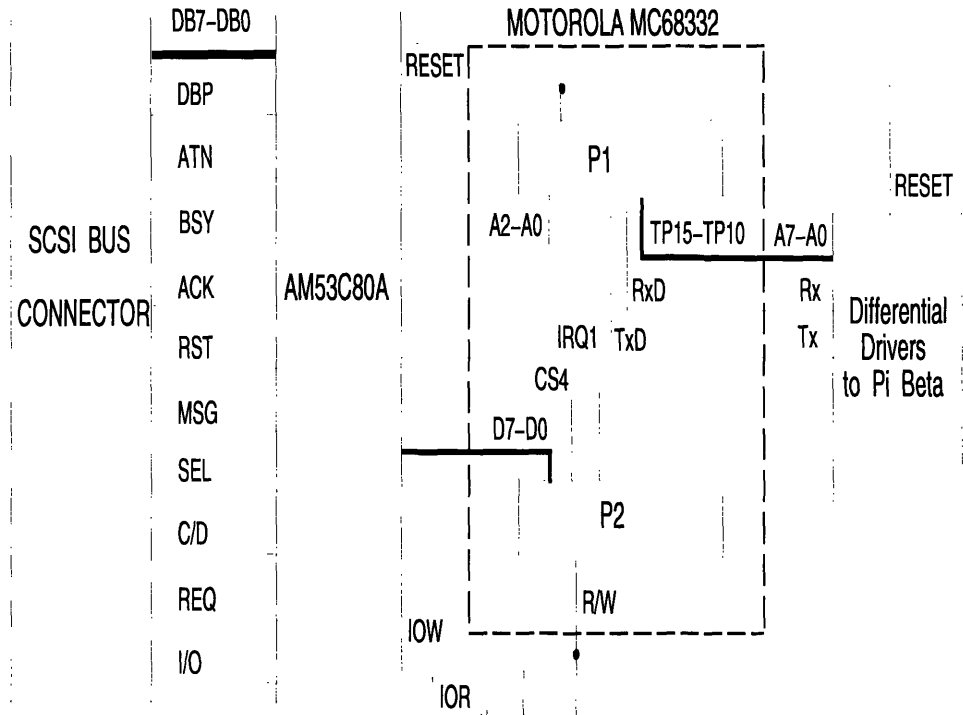


Figure 3-4: SCSI Hardware Interface From FEP to InterFEP

SCSI bus operations.

The SCSI standard calls for a Macintosh computer to always take on the role of the initiator. Accordingly, in $\pi\beta$, the FEP acts as the initiator, while the Am53C80A assumes the role of the target. The interFEP is able to communicate with the Am53C80A as a peripheral device. The SCSI controller chip contains thirteen registers, each holding eight bits of information, which may be accessed by the interFEP. The interFEP may select a register by asserting the chip select input (\overline{CS}) to the Am53C80A and addressing the specific register through various combinations of the inputs A2-A0, I/O read (\overline{IOR}), and I/O write (\overline{IOW}). A summary of the registers along with their address is shown in Table 3.2 [Adv, 1992]. Only three of these registers are both readable and writeable, while the others are either read only or write only. Information is transferred to or from the selected register via the data bus connecting the interFEP to the Am53C80A. By reading and writing these registers, the interFEP is able to control the SCSI controller chip's operation, and thus monitor or assert the various signals on the SCSI bus. Since the programmed I/O input mode

Register	Address (A2-A0)	<i>IOR</i>	<i>IOW</i>
Current SCSI Data Register	000	0	1
Output Data Register	000	1	0
Initiator Command Register	001	X	\overline{X}
Mode Register	010	X	\overline{X}
Target Command Register	011	X	\overline{X}
Current SCSI Bus Status Register	100	0	1
Select Enable Register	100	1	0
Bus and Status Register	101	0	1
Start DMA Send Register	101	1	0
Input Data Register	110	0	1
Start DMA Target Receive Register	110	1	0
Reset Parity/Interrupts Register	111	0	1
Start DMA Initiator Receive Register	111	1	0

Table 3.2: Am53C80A Registers

was used to transfer data over the SCSI bus, the registers dealing with DMA mode transfers were not needed. The programmed I/O mode offered sufficient data transfer rates, so that the DMA mode of operation was not necessary. Also, some of the SCSI controller chip's registers are only used when the chip is operating as an initiator. Since the chip is functioning as a target in $\pi\beta$, these registers were also not necessary.

The SCSI protocol defines eight bus phases which a SCSI device can be in. These phases are as follows:

bus free phase
arbitration phase
selection phase
reselection phase
command phase
data phase
status phase
message phase

Both the FEP and the interFEP commence in the bus free phase. The FEP is able to control the SCSI bus via the SCSI manager trap routines implemented in MacIntosh Common Lisp [App, 1985]. In order to initiate a data transfer, the FEP first needs to arbitrate for control of the SCSI bus by asserting the SCSIGet command.

The FEP thus enters the arbitration phase, and this command returns successfully if the FEP manages to gain control of the SCSI bus.

The FEP may now enter the selection phase, where it selects the target that it wishes to communicate with. The interFEP code would have already written the device ID into the SCSI controller chip by writing the specific ID value to the Select Enable Register of the Am53C80A. When the SCSI controller chip senses that a selection attempt is being made and its ID is on the bus, it sends an interrupt to the interFEP. The interFEP thus realizes that it has been selected, and it then begins to take control of the SCSI bus.

Next, the interFEP needs to write the Target Command Register of the SCSI controller chip in order to assert the $\overline{C/D}$ signal and enter the command phase. Meanwhile, the FEP would have already called the SCSCmd routine, passing it a pointer to the specified command, as well as the size of the command in bytes. The FEP and interFEP then go through a handshaking routine, where the interFEP requests the next data byte, and the FEP acknowledges when it has been sent. The interFEP knows how long the command is, and when all the command information has been received, it deasserts the $\overline{C/D}$ signal and enters the data transfer phase.

In the data transfer phase, the FEP is able to send information to the interFEP by calling the SCSIWrite function and passing it a transfer instruction block as an argument. Likewise, it is able to receive information by calling the SCSIRead function and again passing it a pointer to a transfer instruction block. The transfer instruction block contains information regarding how many bytes to transfer, as well as the location in memory from or to which these bytes should be sent. The interFEP knows which way the data is to be transferred from the information received during the command phase, and it places the corresponding signal on the $\overline{I/O}$ line to direct the exchange of communication. The FEP and interFEP will then again go through the handshaking routine of the interFEP requesting a data transfer and the FEP acknowledging that the requested information is on the bus. This will continue until the data transfer is complete.

The interFEP then either enters the message phase by asserting the \overline{MSG} signal,

or it completes its routine and waits for another selection attempt. The FEP calls the SCSIComplete function, passing along a message as an argument if the interFEP is expecting one. When this call completes, the SCSI bus again enters the bus free phase, and the FEP may again select a device, while the interFEP monitors the bus to see if it is being selected.

Thus, the FEP and interFEP are able to exchange data in the above manner. This is useful for file serving purposes as well as aiding in the debugging of the system. The SCSI bus provides for a fast and reliable exchange of data.

3.2.2 Interface from Inter-Front End Processor to $\pi\beta$

The interface from the interFEP to $\pi\beta$ takes place over the serial peripheral interface on the MC68332 microprocessors. There are eleven signals which are sent between the interFEP and $\pi\beta$. These signals consist of A7-A0, Rx, Tx, and Reset. The signals A7-A0 are used to select which processor to communicate to. The upper four bits, A7-A4, select the backplane that the processor is on, while the lower four bits, A3-A0, select which processor on the particular backplane to communicate to. The Rx, Tx, and Reset lines are then decoded properly so that they are sent to the selected processor. Data is received by the processor over the RxD line, and transmitted from the processor over the TxD line. Also, the Reset signal from the FEP is decoded so that it is received by the Reset input on the selected MC68332 microcontroller.

The logic ensuring that the signals are decoded properly and sent or received from the selected processor is the same for each processing node on $\pi\beta$. Figure 3-5 shows the schematic for this decoding logic. First of all, each processor node has a HC688 eight bit comparator associated with it. The HC688 has connections for power and ground, an active low cascade in signal, sixteen input signals (A7-A0 and B7-B0), and one active low output signal. When the cascade low input is asserted, the HC688 compares the values A7-A0 with B7-B0. If there is a perfect match, the chip asserts its output.

In $\pi\beta$ the 'B' inputs to the chip are fixed. Each $\pi\beta$ node has a particular physical address associated with it. The high order bits, B7-B4, are the same for all nodes on a

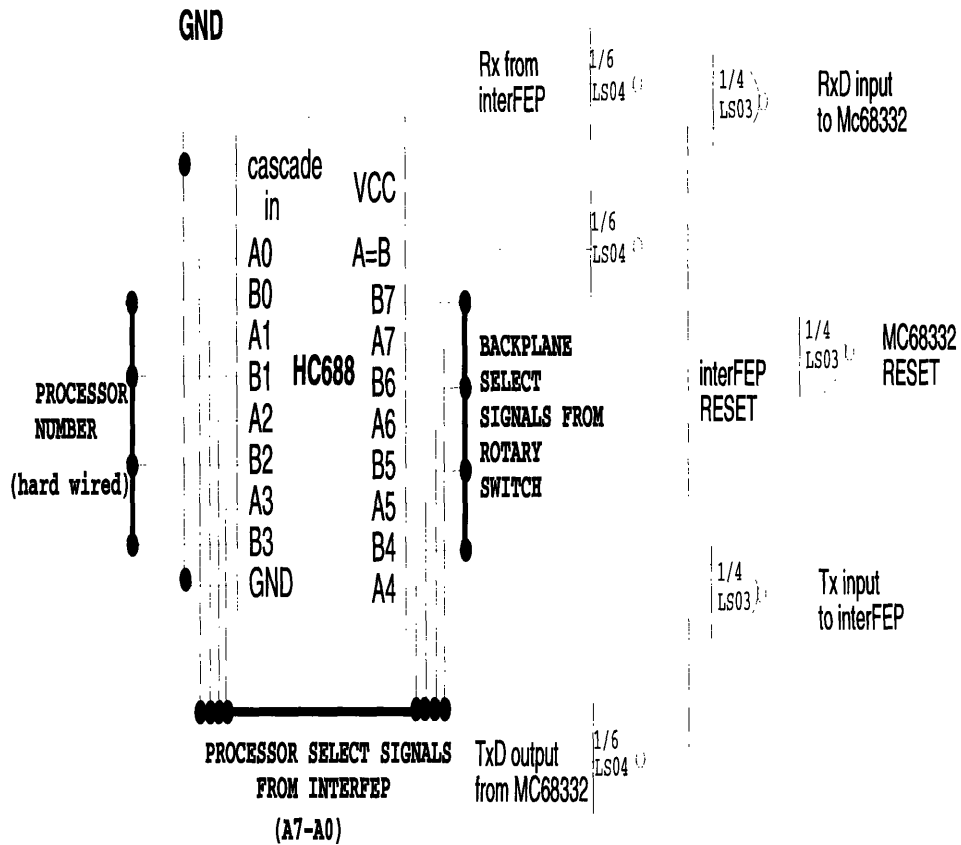


Figure 3-5: Logic for Communicating With Selected Processor

backplane. These bits are set by a rotary switch which is mounted on the backplane. Since the backplane is quite big and the outputs from the switch need to be brought to each node on the backplane, the lines are buffered, with one buffered line attached to the top eight processor nodes, and the second buffered line attached to the bottom eight processor nodes. This assures that the signals from the rotary switch are sent reliably to the input of the comparator. The LS244 non-inverting buffer is used to buffer the signals in the current version of the backplane. In version two, however, a current driver will instead be used to buffer these signals. This will provide stronger signals to the processor nodes on the backplane.

Each backplane has a unique ID set by the rotary switch. With the rotary switch outputting four lines connected to the upper four 'B' inputs to the comparator (B7-B4), there are up to sixteen unique ID's, and thus, up to sixteen backplanes in $\pi\beta$. The bottom four 'B' inputs to the comparator (B3-B0) are directly set for each processor on the backplane. From left to right, starting on the first row and proceeding to the second, each processor is numbered sequentially with the bottom four bits beginning at '0000' and ending at '1111'.

The 'A' inputs to the comparator are connected to the signals A7-A0 coming from the interFEP. These signals are brought in through a fourteen pin connector on the backplane, and then buffered, just as the rotary switch outputs are buffered, to the sixteen various nodes on the backplane. Finally, the cascade in input to the comparator is tied to ground, so that the comparator is always activated. Thus, one of the processors on $\pi\beta$ may be selected, and the comparator associated with that processor will have its output signal asserted.

This output signal is then used to direct the Rx, Tx, and Reset lines to the correct processor node. The following equations were used to decode the serial and reset lines to the selected processor, with the $\overline{A = B}$ signal being the active low output of the comparator determining if a node is being selected or not:

$$Rx\bar{D} = \overline{(A = B) * Rx}$$

$$Tx = \overline{(A = B) * Tx\bar{D}}$$

$$\overline{RESET}_{MC68332} = A = B * \overline{RESET}_{FEP}$$

A LS03 four channel NAND gate and a LS04 six channel inverter were used to implement this logic. First, the active low output signal from the HC688 is brought into an input of the LS04 inverter, so that the signal is now active high. For the case of the Reset signal, which is also active low, this signal is brought into a second input of the inverter making the output of the inverter active high. The active high Reset output of the inverter is then sent into one input of the LS03 nand gate, with the active high processor select output of the inverter being the other input to the nand gate. Thus, when both the reset and processor select signals are asserted, their inputs to the nand gate will be high, resulting in a low output. Since an open-drain nand gate is used, this output is connected directly to the active low Reset input of the MC68332 through a pull-up resistor. Thus, only when both the Reset and processor select signals are asserted will the processor node have its reset signal asserted.

The logic for directing Rx to the MC68332 input RxD is similar to that for the Reset signal. The active high Rx signal is first run through an inverter, and then brought through the nand gate with the other input being the active high processor select signal from the inverter. The output of the nand gate is then connected directly to the MC68332 RxD input, again using a pull-up resistor. Thus, the input to RxD will be a zero only if the processor select signal is asserted and a zero is being received. Otherwise, the input to RxD will be a one. This functions properly, since the processor will know when data is actually being sent between the interFEP and a processor node by looking for the valid stop and start bits for data transmission.

Finally, the Tx signal is also similar to the others, except that it goes in the opposite direction, from the processor node to the interFEP. Thus, the TxD signal from the MC68332 is brought to the one input of the LS03 nand gate, with the other input being the active high processor select signal from the LS04 inverter. This is done for each of the processing nodes, and all the nand gate outputs from these nodes are tied together, run through a pull-up resistor, and connected to the Tx line sent to the interFEP. Thus, if any processor node is trying to transmit a zero, the Tx line will

be held low, and otherwise it will be tied high. Again, this system works because the various combination of start and stop bits will trigger a valid transmission, and then the valid bits will get to the interFEP. Thus, communication is properly achieved between the interFEP and the selected processor node.

As was explained in the previous section, the interFEP was mounted near the FEP in order to keep the SCSI cable length small, and thus have reliable data transfers over the SCSI bus. Unfortunately, this required that the connection from the interFEP to $\pi\beta$ take place over a relatively long cable (about ten feet). In order to ensure that the communication over this longer cable is also reliable, differential drivers were used to transmit and receive the signals. The differentially driven signals, with the true and complement signals driven on properly terminated twisted pair cables, offers very good common-mode rejection and generates clean logic levels. Also, properly terminating the lines gets rid of any transmission line effects from reflections due to impedance mismatching, which may alter the signal.

The 96172 quad differential drivers are used to take in the single-ended signal and output the differential signal. On the receiving end, the 96173 quad differential line receivers are used. They take in the differential signals, have the necessary termination resistors to get rid of transmission line effects, and then output the single-ended signal. Therefore, information was sent reliably between the interFEP and $\pi\beta$.

The interFEP enabled communication between the front end MacIntosh computer and $\pi\beta$. It is able to select any of the processor nodes on $\pi\beta$ to communicate with, and thus check on the status of each of these nodes. It also will send any information to the FEP about problems which it has discovered. Thus, this is helpful in debugging the system. The interFEP also provided for file serving purposes by receiving or sending bytes from or to the FEP, and directing any data to or from the correct processor node.

Chapter 4

Performance Evaluation and Future Work

$\pi\beta$ is currently in use at the MIT Artificial Intelligence Laboratory for the control of the partially developed humanoid robot. However, since Cog is still in the early stages of development, $\pi\beta$ has yet to be used for an extended period of time so that conclusive performance evaluations can be made. Nevertheless, $\pi\beta$ is being extensively used in developing early control for various subsystems of the humanoid, and has met the real time requirements which have thus far been encountered.

One major subsystem which is currently under development and has used $\pi\beta$ for an extended period of time is the vision system. A great amount of work has been done in developing this system. Images from the frame grabbers are able to be obtained and processed by $\pi\beta$ on a real time scale. These images are also successfully output through monitor boards to a display within the same time frame. $\pi\beta$ has the capability of reading in about ten frames per second, which is adequate for the amount of capability expected from Cog.

Also, the beginnings of a visual tracking system has been implemented using $\pi\beta$. This system enables Cog to 'track' a specific object, keeping it in its field of sight. $\pi\beta$'s architecture has been successful in enabling such functionality on the humanoid robot. These early results suggest that $\pi\beta$ does indeed have the capability to control the humanoid.

Another major subsystem which has been under development and made use of $\pi\beta$ involves the lower level motor control of the humanoid. Communication with the MC6811 motor controller boards has been achieved at a rate of 32 Hz. Thus, a $\pi\beta$ node is able to successfully send control information to the MC6811, and change the position and/or velocity of the specified motor. The actuators on Cog are able to be servoed at high rates, and the system is thus able to respond to its environment in a human time scale.

As for the interface to the front end processor, some work still needs to be done in completing this implementation. The hardware for the system is complete, and communication between the FEP and the interFEP has been achieved. However, the entire interface has not been tested as of yet, and we expect the need for some amount of debugging of this interface. We are confident, though, that the interface will run as expected from the results of early tests on certain aspects of the interface. We expect to be able to reach communication rates of about 1.5 Mbytes/sec between the FEP and interFEP.

As more functionality is added to the humanoid robot, and more tests are done with the system, we expect to make better performance evaluations of $\pi\beta$. Since this was a first attempt at the control of such a complex, intelligent system, we expect to find areas which need improvement in the current architecture, and iterations of the design will then be made. Furthermore, once a clear picture of the necessary requirements for the control of the system is obtained, an intelligent selection of a system with higher performance processors which is well suited to the specific application may be made.

A second revision of the backplane is already under way with some minor changes from the first. It will provide for byte addressing of the dual port RAMs, more reliable communication to the dual port RAMs, and better power distribution to the system. This new revision should be completed shortly.

In future revisions of $\pi\beta$, there are several factors which may be improved if necessary. The frame size read in from the vision system may be increased to larger than 128 X 128 Kbyte frames to facilitate processing this data. This should not be very

difficult since the interface to the dual port RAMs provides several unused address lines. Furthermore, the communication rate between the FEP and the interFEP may be increased by using either blind SCSI data transfers or using the DMA mode of operation on the Am53C80A SCSI controller chip. Thus, there is the capability of easily adding in certain improvements, should the performance of $\pi\beta$ turn out to be slightly less than what is expected.

Chapter 5

Conclusion

The design of $\pi\beta$ is unique from the design of conventional parallel processor computers. Since it was developed for a very specialized purpose, the constraints of the specified application were able to be considered in the architectural design of $\pi\beta$. Not only was there a need for a high performance enabling technology for Cog, but there was also the constraint of guaranteeing that the implemented solutions for the control of the humanoid robot would be relevant to what is currently understood about biological systems.

Thus, the major constraints which were considered in the design of $\pi\beta$ are that it respond to its environment in real-time, the subsystems operate asynchronously, there is no global control, and there is no globally shared memory. These constraints differed from those faced in the design of most large scale parallel machines, since these are generally aimed at speeding up serial computations. Contrary to these designs, $\pi\beta$'s purpose was to simplify the control of the inherently parallel system by making use of available parallel hardware.

These constraints simplified the required parallel architecture needed to control the humanoid robot. Since memory was local to each processor, there was no need to maintain a consistent global memory or worry about cache coherency. This simplified the architecture, resulting in a higher performance system at a lower cost.

Though $\pi\beta$ may not offer the best performance and throughput as compared to other parallel computers, it does offer the high performance necessary for the control

of Cog, as well as meet the specified constraints of the system. The scalability of the design enables the developers of Cog to use the layered approach to designing an intelligent system, and therefore not need to change the subsystems which have already been implemented.

Therefore, $\pi\beta$ does successfully meet the requirements for which it was designed. It does encourage the finding of parallel solutions to the control of Cog, and it does allow for the control of a real-time system. It is a good choice of the control for an application which is expected to go through many different iterations and will thus change often. Therefore, $\pi\beta$ functions well as the enabling technology for Cog.

Appendix A

Schematics of System

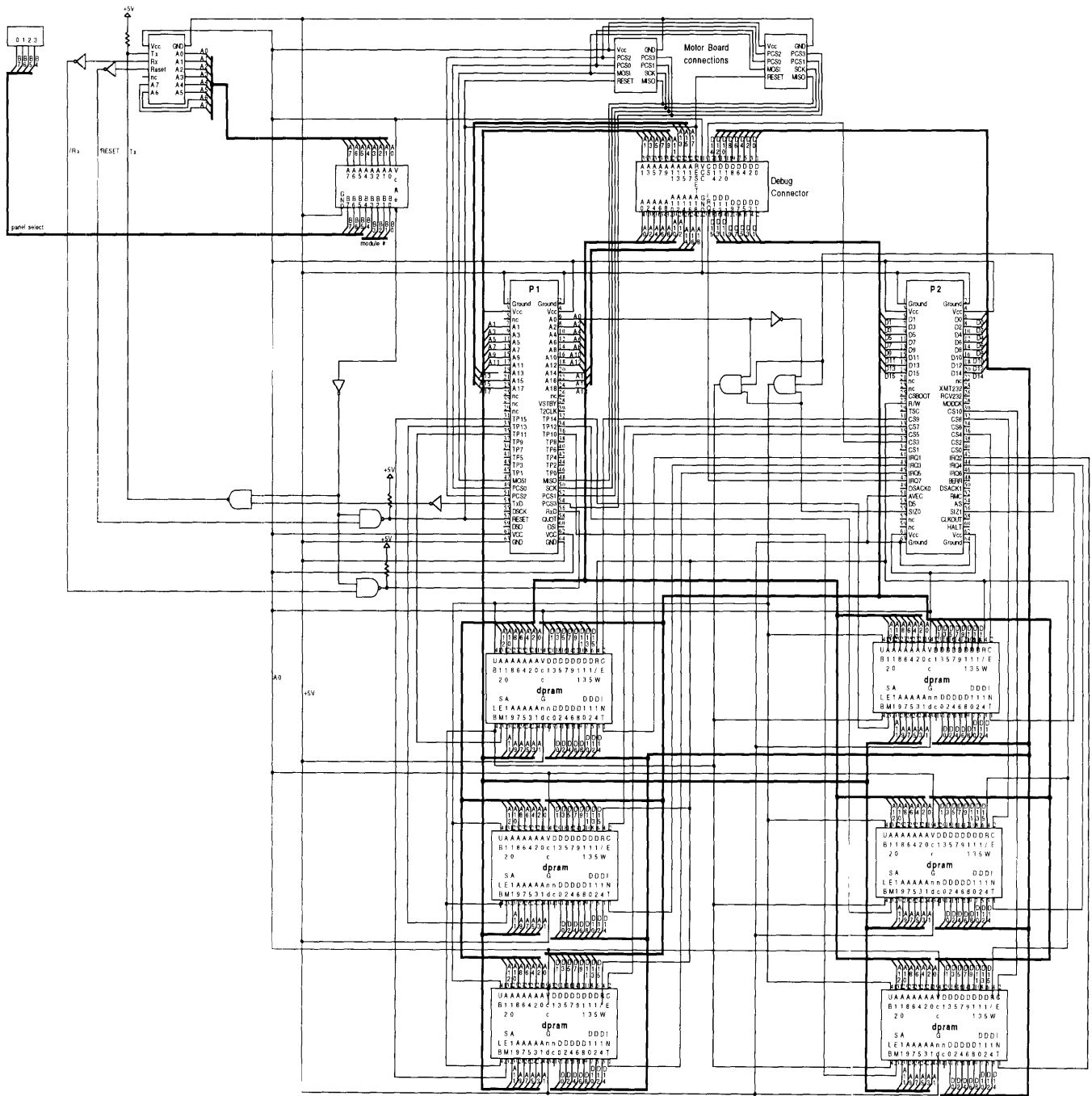


Figure A-1: Detailed Schematic of a $\pi\beta$ Node

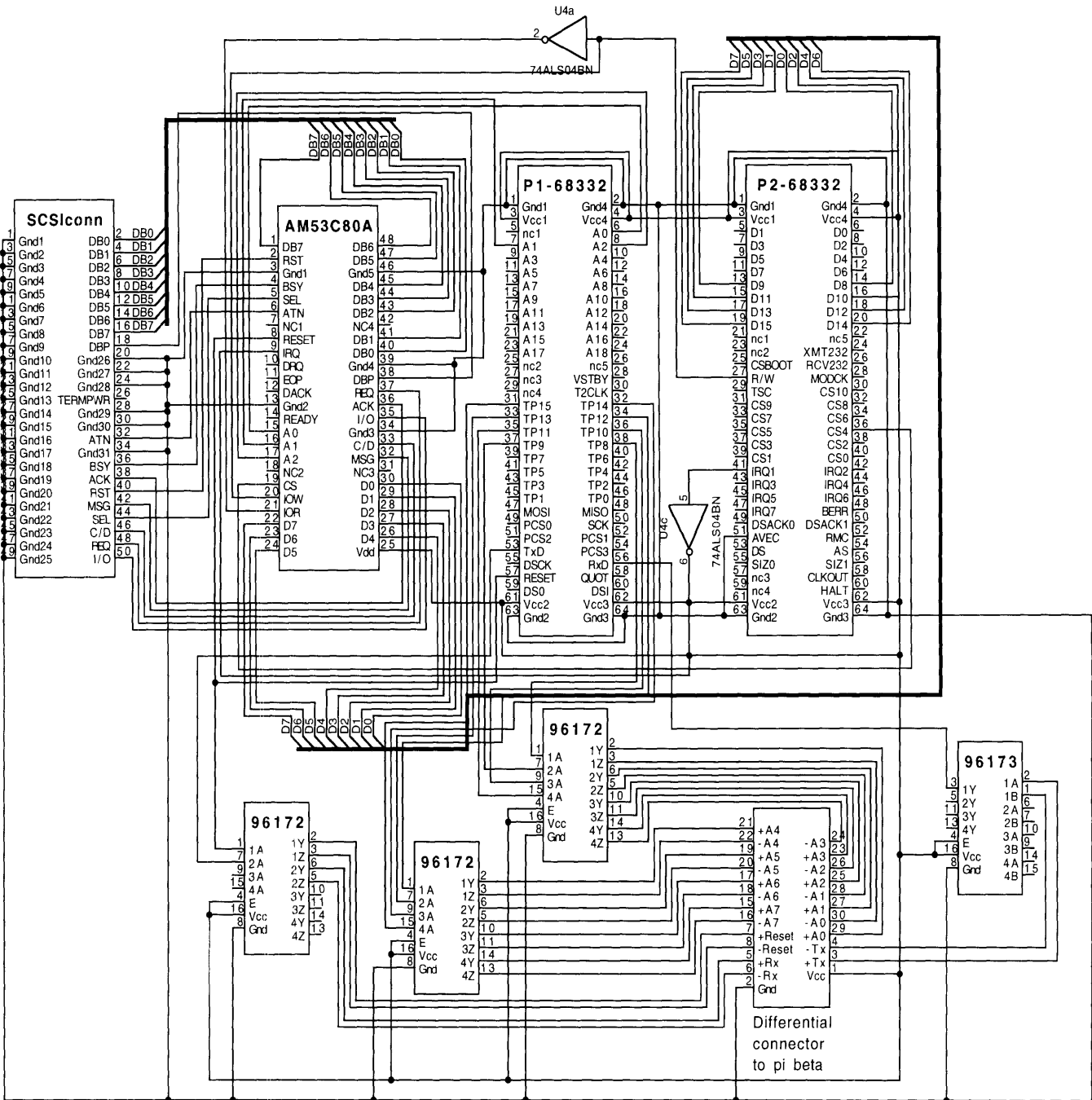


Figure A-2: Schematic of InterFEP

Bibliography

- [Adv, 1992] Advanced Micro Devices. *Data Communication Products*, 1992.
- [Almasi and Gottlieb, 1994] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing*. The Benjamin/Cummings Publishing Co., Inc., 1994.
- [Ame, 1986] American National Standards Institute, Inc. *American National Standard For Information Systems - Small Computer System Interface (SCSI)*, 1986.
- [App, 1985] Apple Computer, Inc. *Inside MacIntosh Volume IV*, 1985.
- [Brooks and Stein, 1993] Rodney A. Brooks and Lynn Andrea Stein. Building brains for bodies. Memo 1439, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, Massachusetts, 1993.
- [Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14-23, 1986.
- [DeCegama, 1989] Angel L. DeCegama. *The Technology of Parallel Processing*. Prentice Hall, 1989.
- [Flynn, 1972] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21:948-960, 1972.
- [Hwang and Degroot, 1989] Kai Hwang and Douglas Degroot, editors. *Parallel Processing for Supercomputers and Artificial Intelligence*. McGraw-Hill Publishing Company, 1989.
- [Int, 1990] Integrated Device Technology, Inc. *Specialized Memories Databook*, 1990.

[Mot, 1990] Motorola, Inc. *MC68332 User's Manual*, 1990.

[Ves, 1993] Vesta Technology, Inc. *Vesta SBC332 Family Hardware Manual*, revision 1.3 edition, 1993.