ISBN 0-315-67867-4

RESUME

Cette thèse présente le dessein et la realization d'un interface de multi-processeur pour un systeme d'animation de graphique en couleur. L'acces direct a la memoire est utilisé pour le transfert de data entre les memoires des processeurs du systeme GRADS Les divers dimensions de mots de memoire, protocol de bus, et d'interruptions sonts accomodes. Une évaluation du systeme est incluse.

PREFACE

The research described in this thesis was completed several years ago, but due to unavoidable personal commitments the final presentation of this thesis was delayed until now

It is believed that despite the rapid advances in the area of microprocessors and data communication, the core of the work described is of equal relevance today To the extent possible, an effort has been made to discuss the project in light of the current state of art.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my research supervisor Dr. A S. Malowany for his guidance, advice and help throughout the course my studies at McGill His understanding, patience, and encouragement are the main reasons for the completion of this project.

The financial support provided by the Department of Electrical Engineering in the form of Research Assistantships, Teaching Assistantships, and Graduate Awards enabled me to pursue my Graduate Studies at McGill University This support is gratefully acknowledged.

Help provided by Woon Thong and Shailja with the preparation of diagrams for this thesis is appreciated

And finally, my thanks to Mom, Dad, Alka and Rakesh, without whose sacrifices my pursuit of higher studies would not have been possible.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 COMPUTER GRAPHICS

1.1 Introduction

Since the invention of the digital electronic computer in the 1940s [3], it's tremendous proliferation in almost all walks of society has as much to do with its versatility as it has to do with the continuous research into ways of making the computer practical to use by people who do not posses detailed knowledge of it's inner workings

Although there have been countless and continuous refinements made to render the computer a convivial tool, two of these stand out as major cornerstones of computing evolution The first of these was the development of high level programming languages This effectively allowed a wider circle of "computer literates", trained engineers and scientists, to use the power of the computer to solve problems

The second revolution took place when interactive computing became a reality This allowed users, who did not have any knowledge of computers, to use the machines; interacting with them to feed the pertinent input and observe the output This became doubly effective as the mode of interaction changed from being "keyboard only" to visual Computer graphics not only caused this major change, but since then has driven the design and architecture of many a computing system Swezey and Davis [36] have noted that with increased use of computer graphics in all fields of endeavour, man-machine interaction is becoming more and more important. A common example being the Macintosh series of computers by Apple Corporation

The first conceptually complete interactive graphics system was developed at MIT in the early sixties by Sutherland [35] Called "Sketchpad", it contained data structures describing the graphical representation of real or imaginary objects It provided a means of selecting them via

the use of light pens, and allowed for the rotation, scaling and translation of the image information. Although "Sketchpad" remained impractical due to the high cost of hardware required, the basic concepts are evident in most present day interactive graphics systems.

Throughout most of the 60's and the early 70's, research into interactive computer graphics continued, with large engineering institutions such as Boeing, General Motors, IBM, and Lockheed exploring the possible marriage between computer aided design and interactive computer graphics. Suddenly, in the mid-70's, the price barrier that had been constraining the widespread use of computer graphics started to collapse. With microprocessors, and high density and low cost semiconductor memories, came the possibility of lower cost graphics systems [23.]

1.2 Applications

Today there are many applications of computer graphics, but perhaps most significant are the changes it has brought about in the areas of medicine, earth sciences, engineering, aerospace, and entertainment.

In the radiology field of medical science, computer graphics is rapidly complementing techniques such as Computer Aided Tomography, adding colour as a new dimension in such imaging. The use of colour to designate different temperature zones is proving to be of particular importance in the area of cranial neurology [20.] With the advent of high speed communication networks [25], computer graphics will allow radiology information on patients to be exchanged at the push of a button, almost obviating the use of photographic films for such applications. Medical researchers in the fields of Genology are already making use of computer graphics to model and visualize complex DNA molecules and Gene structures [14].

Engineering has benefitted tremendously from computer graphics as well, mainly in the areas of CAD, CAM, CAE Most engineers today use interactive computer graphics as a routine way of capturing designs (VLSI designs, vehicle shapes), of modelling them or testing them, and for gaining rapid access to all the supplementary information. Most design workstations provide a high level of desk-top graphics support [8]

The aerospace industry has been the longest user of computer graphics, mainly in flight simulators [12.]. This provides a low risk and low cost alterr.ative to the training of pilots by being able to simulate various "situations" on the ground Computer graphics is now also invading the cockpits of modern aircraft. The Boeing-767 and the Airbus-320 make use of CRT displays to graphically present information conventionally communicated to the crew through electro-mechanical meters and displays [18] The same concept is extended to Head-Up Displays (HUD) on modern combat aircraft The HUD typically merges computer project targetting information with the target image, thereby minimizing eye movement in critical combat situations [37.].

In the Geology and Meteorology fields of earth sciences, real-time graphics is becomming an indispensible tool in the analysis of large amounts of information contained in data sets [16] Made up of billions of bits of information, these data sets are updated in real-time by a plethora of satellite and earth based remote sensing devices The application requires the raw data to be evaluated in the context of numerical simulation models One such application developed by the Space Science and Engineering Center at the University of Wisconsin allows three dimensional animation of models which make use of data on atmospheric conditions such as temperature, pressure, humidity, and wind speeds Called McIDAS [17.], it allows scientists to more accurately understand weather dynamics

Applications in the entertainment industry are also becoming apparent Computer animation has been used in such recent classics as "2001-A Space Odyssey" and "Star Wars" [7] With the

advent of High Definition Television (HDTV), the cross editing of real shots and computer generated images will take on real importance, and may create an extremely powerful medium [10.][22.].

One of the more recent developments in the areas of graphics animation is the emergence of Multi-Media Interactive Video [38.]. This approach to achieving spectacular graphics at low cost is based on the real-time merging of real video shots with computer generated images. By using digitally stored actual images, the system is able to achieve projection of complex images well beyond the modelling and animation capabilities of modest computing platforms such as desktop computers. The whole concept of multi-media graphics depends upon the ability to digitally store a vast number of complex images. Since the commercially available audio "compact discs" are the most viable low cost storage medium, the success of multi-media graphics depends on the ability to compress visual information to fit into a typical compact disc.

Major corporations like Phillips, Intel. IBM, Sony, and Matsushita are aggressively trying to define the new standards for image compression and de-compression. To date two leading proposals have emerged Phillips is championing technology called Compact Disc Interactive or CDI while Intel is proposing Digital Video Interactive or DVI [9.]. Though both promise to solve the same problem, the approach and strategy are different

The applications of multi-media video are indicative of its power and versatility. The Peugot automobile company is already using the Intel developed DVI system to train its mechanics. The system is built around a single compact disc which carries diagrams, text, and videos demonstrating repair procedures. Typically, a mechanic can watch a demonstration video on one window on the screen, and call up images of specific parts on another window on the screen. Other applications include real-estate sales, where "visits" to different sites can be arranged at the customer's location, complete with animated "walkthroughs", and the ability of the

customer to better visualize the space and proportions of the room by superimposing images of furniture on live shots of a room.

1.3 Graphics Hardware

The earliest graphics systems used a display technology called Random Scan (also referred to as vector graphics). This display required a vector generator to continuously display a series of lines and points on the screen in quick succession to form a line image or "wireframe" image. The limitation of this type of graphics system was it's inability to display solids and colours. Although the refresh requirements were no longer required with the advent of the Direct View Storage Tube [24.], the inherent limitations of a vector graphics system limited their use.

In most present day systems, raster graphics is the dominant technology [34]. This has been made possible with the low cost or colour CRTs and availability of high density memory Raster display systems require a large amount of memory and a large refresh bandwidth to provide the sequential pixel by pixel update numerous times per second, although innovative designs such as Fuch's [13.] PP5 system have succeeded in reducing them significantly

Other display technologies used for graphics displays are the Plasma panel and the Liquid Crystal Display (LCD) panel. These displays have an advantage over the CRT type raster displays due to their relatively low power consumption and ruggedness However, they are generally monochrome, and too slow for real time animation (LCD panels) [43] These displays are being increasingly used in portable equipment but have not been refined to the point of being used in state of the art graphics systems.

The increased use of computer graphics in different applications has led to the development of various designs which balance such requirements as speed, resolution, model sizes etc The availability of powerful microprocessors and Application Specific Integrated Circuits (ASICS) has

led to various architectures being explored. Increasingly, graphics systems use multiple processors Patton's article [30.] reviews the familiar Flynn classification of single/multiple data/instruction machines, as applied to multiprocessors and multiprocessing. An interesting part of the article deals with a study of system performance versus the number of processors in a highly parallel system. Rodgers further classifies systems exploiting parallelism according to the relation and specialization of the parallel computing elements [33.]. Two applications of computer graphics which require enormous computational power are real time flight simulation, and medical imaging. Both are candidates for a multi- processor architecture involving parallelism [1].

A wide variety of graphics systems are commercially available today. An excellent comparison of the Evans & Sutherland MPS-2 and Vector General VG3400 is provided in the article by Foley and Van Dam [11.]. One interesting aspect of both these systems is that they both use a general purpose host computer to generate the initial images. Another example of a multi-processor graphics system is the IRIS system made by Silicon Graphics. This uses a pipelined architecture with the key components being a Motorola 68000 and a set of "Geometry Engines" linked via an Ethernet network [6]. In an attempt to provide a more general graphics engine, Texas Instruments now markets the TMS340 Graphics Processor [40.]. When used in conjunction with a more general host computer such as the M68020 or the i80286, a TMS34070 Video Palette, and TMS4161 Multiport Video RAM, a single TMS340 can form an effective graphics system.

When real-time animation is a requirement, most systems use multi-processing either through pipelining or parallel processing. The Sellac-7 [19] uses a combined parallel/pipelined architecture, achieving a five fold increase in throughput. Another approach to increased throughput is the GODPA system [15.]. This system divides the 3D object space into 64 parts and use as many processing elements process the information. Other advanced graphics processing ar-

chitectures include the EXPERTS system [26.], the CHAP SIMD processor in the Lucasfilm Compositor System [21.], and Piper & Fournier's STINT [31.].

1.4 GRADS

A graphics system capable of animating 3D coloured objects in real-time has been developed at McGill University [28.]. This system is called GRADS (Graphics Real-time Animation Display System), and, like other advanced graphics systems, uses multiprocessing and pipelining

The objective of this thesis is to describe the design and implementation of the interface used to network the various processors in the system.

In the following chapters, the architecture, design, implementation and debugging of the Host Computer Interface are described.

2.0 GRADS ARCHITECTURE

To satisfy the processing requirements for the generation and animation of high resolution colour images, the GRADs uses multiple processors and a pipelined architecture consisting of three distinct stratas of processing elements to provide the necessary processing power for such applications Figure 1 shows the processing hierarchy of GRADS

The topmost layer consists of a VAX-11/780 Host Computer, which is responsible for performing the higher order graphics functions such as scene generation, hidden line removal, rotation, and lighting effects. This typically requires a processor with a large memory and computational throughput to support complex graphical models. To meet the requirements of real time animation, the Host Computer creates 30 static scenes or pictures every second, and then decomposes each scene into graphics primitives such as lines, points and polygons. Each primitive, along with a string of parameters describing its coordinates, colour and intensity, forms a graphics macro-instruction. These macro-instructions are then queued and passed onto the next lower strata for further processing and display. A typical macro-instruction block consisting of a number of polygons and lines is shown in Figure 2.

The Host Computer is also the master controller of the GRAD system, and is responsible for the correct initialization and operation of the various subsystems. This is achieved via a series of software routines that download firmware and microcode upon system startup, and provide centralized maintenance and diagnostic capabilities. These capabilities are either inherent in the GRADOS operating system [29.], or are provided as separate software utilities.

The second processing layer is a true Multiple Instruction Multiple Data (MIMD) processing system, and is made up of multiple processors, each with their own memory and data systems. This middle layer consists of an array of 16 & 20-bit, loosely coupled microprocessors which receive graphics macro instructions from the host computer and process them to generate pixel

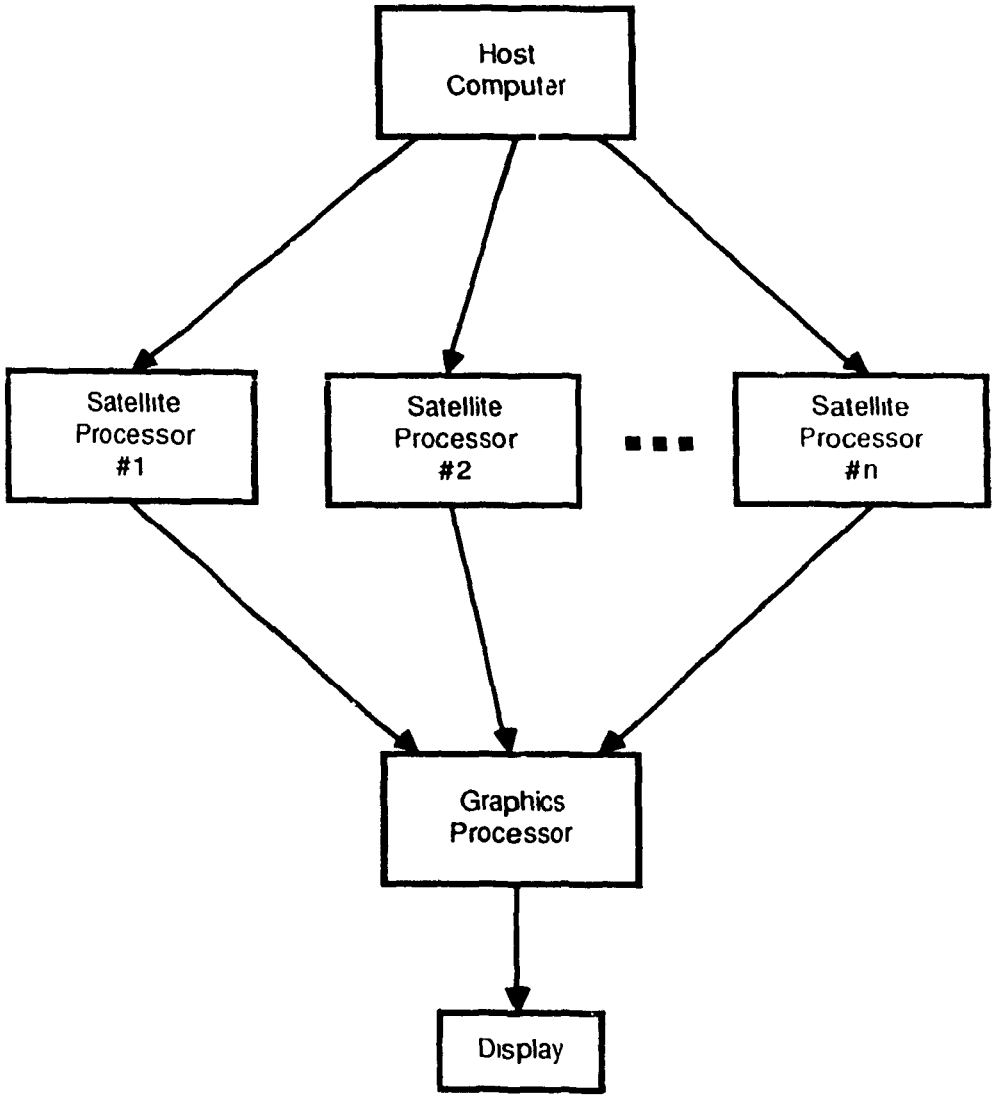


Figure 1. GRADS Processing Hierarchy

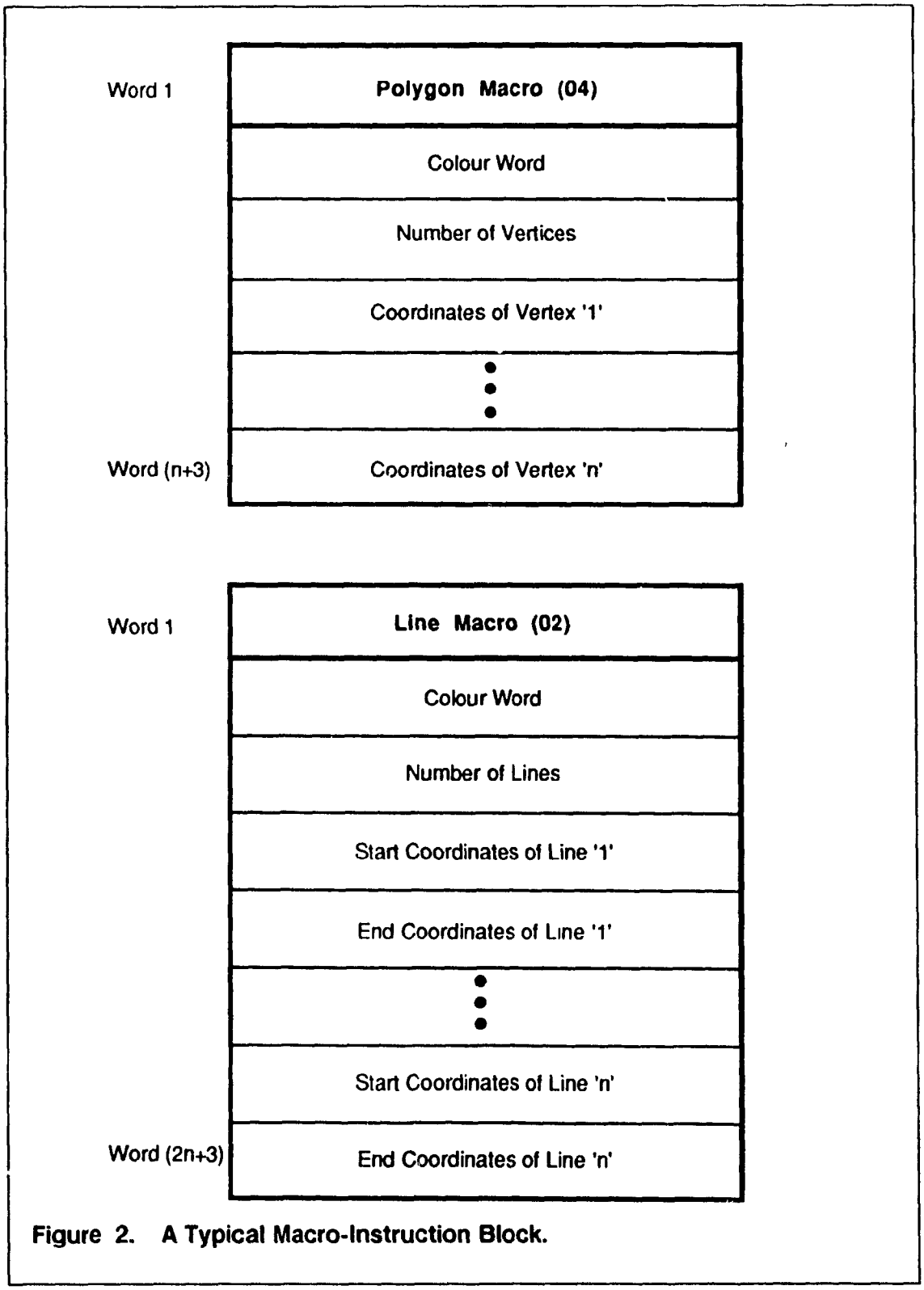


Figure 2. A Typical Macro-Instruction Block.

level information. The array concept exploits the fact that in a bit mapped, frame buffered animation system, there is virtually no constraint on the sequence in which picture elements in a frame are assembled, as long as the frames are updated at least every 33 milliseconds This means that the exact order in which the graphics primitives are processed is not important as long as all the information gets processed in the prescribed time This allows the processors to run without tight synchronization amongst themselves To ensure that all processors are equally tasked, the host computer dynamically manages the input data streams of the processors.

In the third strata, a dedicated graphics processor merges the output of all the microprocessors, and maps it onto the pixel information stored in the video frame buffer memory This frame buffer memory is periodically read out in a sequential fashion to a raster scanned colour display system.

In terms of display capabilities, the GRADs system is able to work in either a low resolution mode in which 256 X 256 pixels are displayed, each with up to 32 levels of intensity, or the high resolution mode, consisting of 512 X 512 pixels with 32 levels of intensity These two options are selectable in real time by the application software. The low resolution mode provides much higher throughput and is used during debugging of application software, or while running hardware diagnostics.

Figure 3 shows the detailed architecture of the GRAD system The main components of the system are

1. The Host Computer (on the UNIBUS or S100 Bus)
2. The Host Computer Interface (HCI)
3. The satellite microprocessors (8086, Z8000, AMD2900)
4. The Graphics Controller

5 The Video Memory

6. The TV Sequencer

2.1 Host Computer

The function of the VAX-11 780 as the Host Computer has been described above. It communicates with the Host Computer Interface of GRAD system via the UNIBUS It decomposes a picture (or a series of pictures) into a graphic primitives that will be further processed by the GRAD system For a more primitive and a less computationally intensive application than the generation of images, the GRAD system can also be hosted by an S-100 system via the S-100 bus to the HCI The S-100 system is a Cromenco Z-80 system with 64K of local memory The lesser applications include debugging, maintenance, and the display of static pictures. At run time, however, only one host computer can be active.

2.2 Host Computer Interface (HCI)

The HCI is the vehicle through which communication between the various modules take place. It carries both command and status exchanges between the various modules, and also serves as a Direct Memory Access controller module for the movement of large amounts of graphics data with little intervention between the source and destination modules. Communication (as opposed to data transfers) between various modules is carried out efficiently by means of interrupts Any module is capable of initiating an interrupt The HCI also serves to intialize the GRAD system upon startup It downloads application programs to the satellite processors and microcode for the RAM sequencer of the Graphics Controller. The HCI is designed in such a manner to accomodate a myriad of different bus structures without making any changes to the core of the machine. This is accomplished by the use of interfaces units that present a uniform

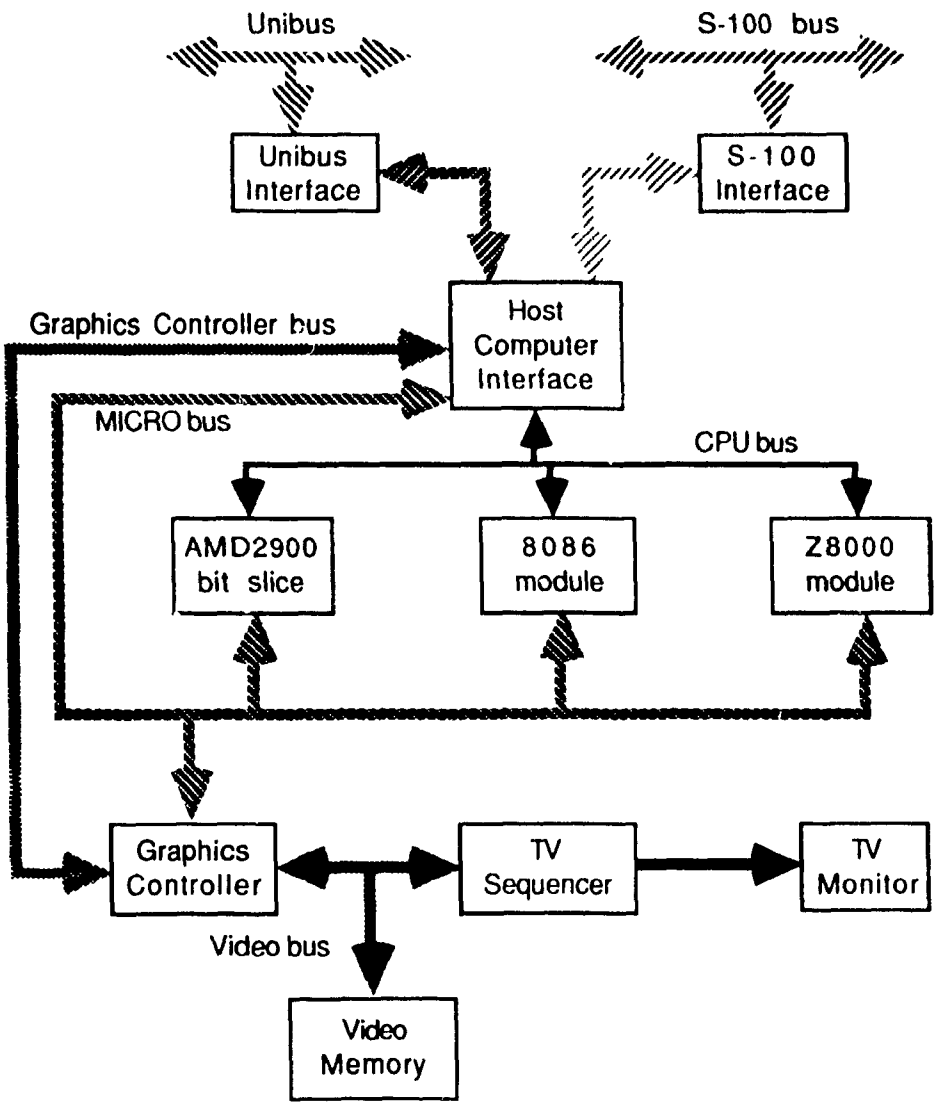


Figure 3. GRADS Architecture.

appearance to the HCI. A more detailed discussion of the architecture and working of the HCI will follow in chapter 5.

2.3 Satellite Processors

The queue of macro instructions originating from the host computer is routed via the HCI to the appropriate satellite processors The HCI is connected to the satellite processors via two busses, the CPU bus and the MICRO bus The CPU bus differs from the MICRO bus in that it allows for intelligent communication between the HCI and the satellite processor CPUs. The MICRO bus, on the other hand allows the HCI to communicate only with the memory of the satellite processors.

The satellite microprocessors include an Intel 8086, a Zilog Z8000 and an Advanced Micro Devices AMD2900 bit slice processor. Each satellite processor has its own local memory that is accessible to itself and also to the HCI via the MICRO bus. The Intel 8086 CPU is a 16-bit device and can access up to one megabyte of memory (in Real Mode) and up to 64K of I/O ports. The highest clock speed that it can be run at is 8MHz [2.]. The Zilog Z8000-based satellite processor is a 16 bit machine and is equipped with 64K of memory and operates with a 250 nanosecond cycle time [32.]. The third satellite processor is based on the AMD2900 bit-slice ALU, utilizing five of them to form a 20-bit processor [5.]. The microcode for this application is optimized for graphics processing and is stored in a fast bipolar 1K X 40 bit Random Access Memory. Its data memory is configured as a 4K X 20 bit block. Figure 4 shows the architecture of the satellite processor based on the AMD2900.

These satellite processors have the task of converting the queue of high-level macro instructions to pixel information. Exactly which macro instruction is routed to which satellite processor is dependent upon a number of factors. Certain graphic primitives, such as line drawing, are best handled by a bit slice processor such as the AMD2900 from the point of view of speed of exe-

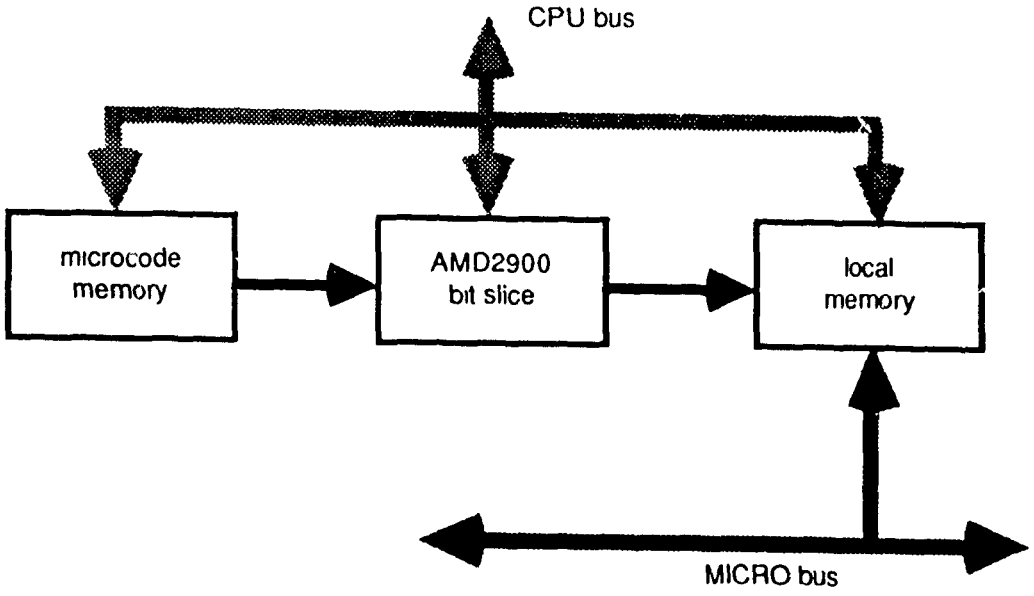


Figure 4. AMD 2900 Based Satellite Processor.

cution Other graphic primitives, on the other hand, are better handled by a general purpose processor such as the the Z8000 or the 8086 A third variable is one of availability. If two processors are already busy executing a task, it may be faster to route a new task to an idle third processor that may not execute as efficiently as one of the two that are busy

2.4 Graphics Controller

The resulting pixel information from the various satellite processors is then gathered and assembled by the Graphics Controller [4.] The assembled pixel information is then written into video memory The Graphics Controller is connected to the HCI via the Graphics Controller bus. It is via this bus that the Graphics Controller can also receive instructions and graphics data and report its status to the HCI

The Graphics Controller was implemented using a RAM-based state machine, or sequencer circuit The RAM sequencer has a microcode memory configuration of 32 X 1K A RAM sequencer design was chosen to lend flexibility to the operation of GRADS, and as an added benefit, to aid in its debugging Each satellite processor is continuously polled by the Graphics Controller for a completed packet of pixel data When ready, the satellite processor requests a DMA transaction with the Graphics Controller The terms of the DMA transaction (eg word count) are included in the DMA packet. The Graphics Controller also serves to arbitrate between the MICRO bus and the VIDEO bus

2.5 Video Memory

The Video Memory contains the pixel information of the picture that is currently being displayed. It is written into by the Graphics Controller, and is read from by the TV Sequencer. The size of the Video Memory determines the intensity and the resolution of the picture. The Video Memory is designed to be modular and incrementally expandable, with resolution and intensity

being interchangeable. The elemental module of the Video Memory is called a Single Low Resolution Memory Plane (SLRMP). It consists of 64K bits of static memory organized as 4K words of 16 bits each. To obtain a color picture, each of the three primary colors of red, blue and green must have a color plane assignment. To obtain a low resolution color picture containing 256 X 256 pixels and 32 levels of intensity each, 15 SLRMP are required A 512 X 512 pixel picture with the same intensity per color per pixel, requires four times as many planes

A variety of video memory planes were assembled as the technology improved An alternate implementation has three 4K X 16 bit static memory banks built on one board. Yet another type uses Dynamic RAM, containing 768K bits of storage. It can be configured either as 12 low resolution planes or 6 high resolution planes.

2.6 TV Sequencer

From the video memory, the pixel information is read out via the VIDEO bus continuously and sequentially by the TV sequencer and converted into an analog signal to be sent to a raster RGB monitor The image sent to the monitor is updated thirty times every second to provide flicker-free images. The TV sequencer also generates the necessary synchronization signals for the monitor. The TV sequencer merely converts the RGB information in the video memory planes into analog signal; whether an image is of high or low resolution is controlled primarily by the Host Computer, and secondarily, by the Graphics Controller.

The focus of this thesis is the Host Computer Interface. Subsequent chapters describe its design, development and integration in detail.

3.0 HCI DESIGN APPROACH

In designing the host computer interface, several data communications methodologies were considered Most of the communication options can be classified under the following three categories:

1. Using a packet switched network connecting the various processors, and using an X.25/X.75 type Open System Interconnect (OSI) networking protocol. This is widely used in wide area networks such as Telecom Canada's Datapac network, and in local area networks such as Ethernet (Figure 5). Since data packets in such networks are routed depending upon their contents, virtual "point-to point" or "point-to-multipoint" paths can be created Although attractive for systems separated by at least a few meters, it turns out to be overly expensive in terms of processing overhead for an inexpensive multiprocessor system. Such connectivity can provide burst transmission rates of only 10 M Bits/Sec. In addition, transmission delays in packet networks under heavy traffic conditions are somewhat randomized, further diminishing its desireability for such real time applications.

   At the time of researching these options, single chip X.25 chips with built in DMA capability, such as the Motorola MC68605 [42.] were not available, making this option non-practical for discrete implementation.

2. A high speed parallel backplane connecting single card microprocessors, using a central arbitration controller. This method has been used extensively in systems like the S100 TURBODOS and Northern Telecom's DVS Meridian system (Figure 6). In such a scheme, the central controller can either be in the data path, forming a true star network, or perform only arbitration on a parallel data flow system, and thus forming a bussed data system with a virtual star network for arbitration and control. Such systems can be extremely fast. The

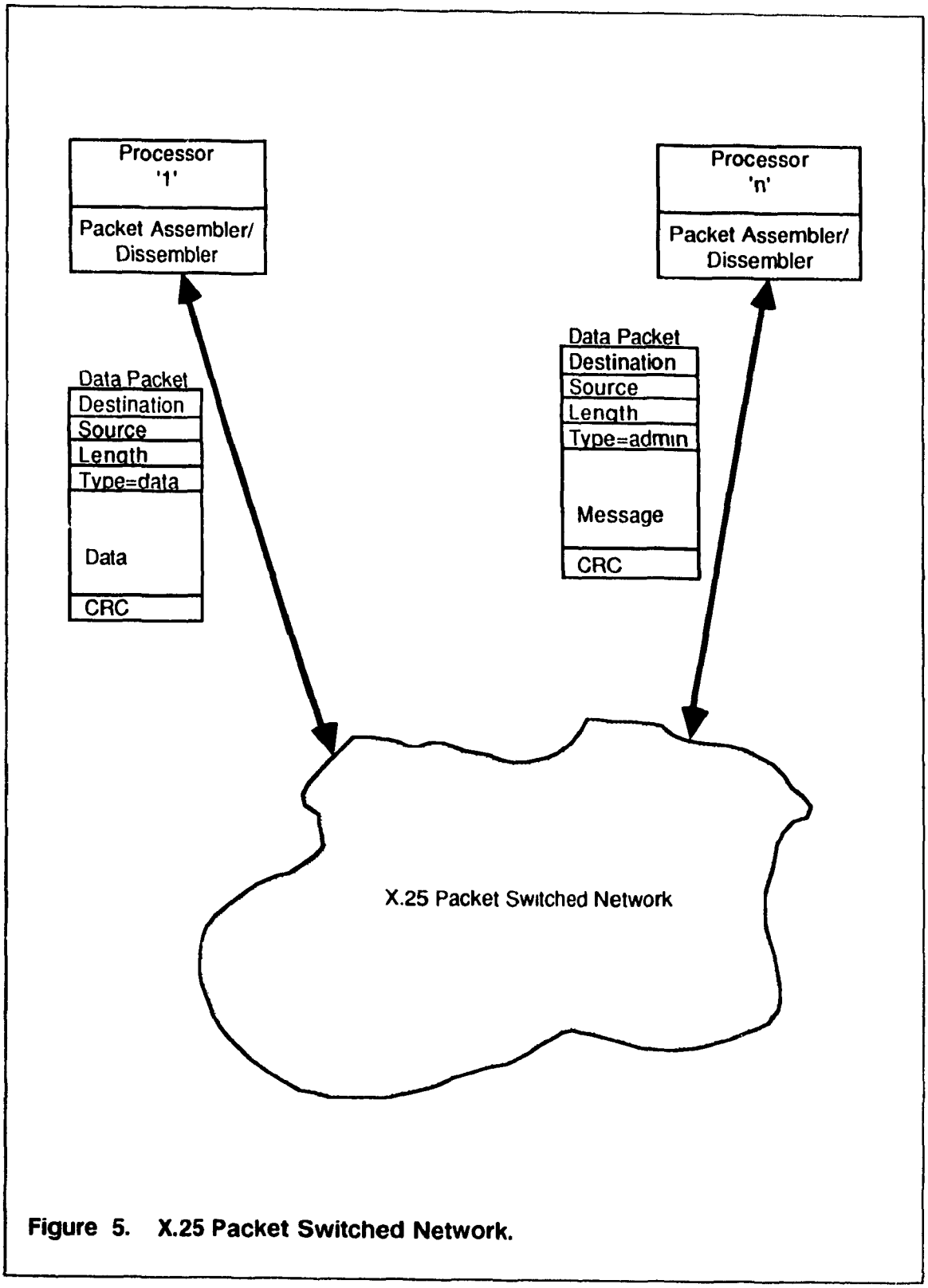


Figure 5. X.25 Packet Switched Network.

data transfer rates in such cases are only limited by memory bandwidths and system backplane lengths. In an application like GRADS, where all processors are located in a single shelf, a parallel bus of 16-20 bits can be made to operate at cycle times of better than 500 nano seconds. This translates into peak information transfer rates of 40 M Bits/second. Moreover, such systems can be implemented with fairly inexpensive technology. Other variations of this technology to emerge since 1982 are various multi-processor bus standards such as the VME [41.], which provide shared memory options using Dual Port Memories. Such systems provide for high speed data moves between processors without the need for dedicated communications arbitrator.

3. A fiber optic network linking various intelligent modules. This is an approach well suited to applications where both speed and wider separation are desirable. Typical present day fiber systems can support transmission rates in excess of 100 MB/S over several kilometers. Although a scaled down version of the same technology has been used to form "optical backplanes", the implementation remains fairly complex due to the basic "point-to point" nature of a fiber connection (Figure 7). Added to this are the mechanical constraints imposed on the system, where cards cannot be simply pulled out or replaced without having to undo cumbersome fiber connectors.

Considering the pros and cons of the three different types of data communication options, a parallel wired backplane approach was considered optimal for intermodule communication, given the speed, complexity and cost factors. Support of processors with different word lengths dictated the selection of a true star system with a capability of intercepting the data, and packing or unpacking it to match source and destination word sizes.

Having selected the topology and architecture, further design of the Host Computer Interface necessitated the definition and design of a suitable arbitration method, and the design of a high speed controller capable of supporting a variety of microprocessors using different word sizes

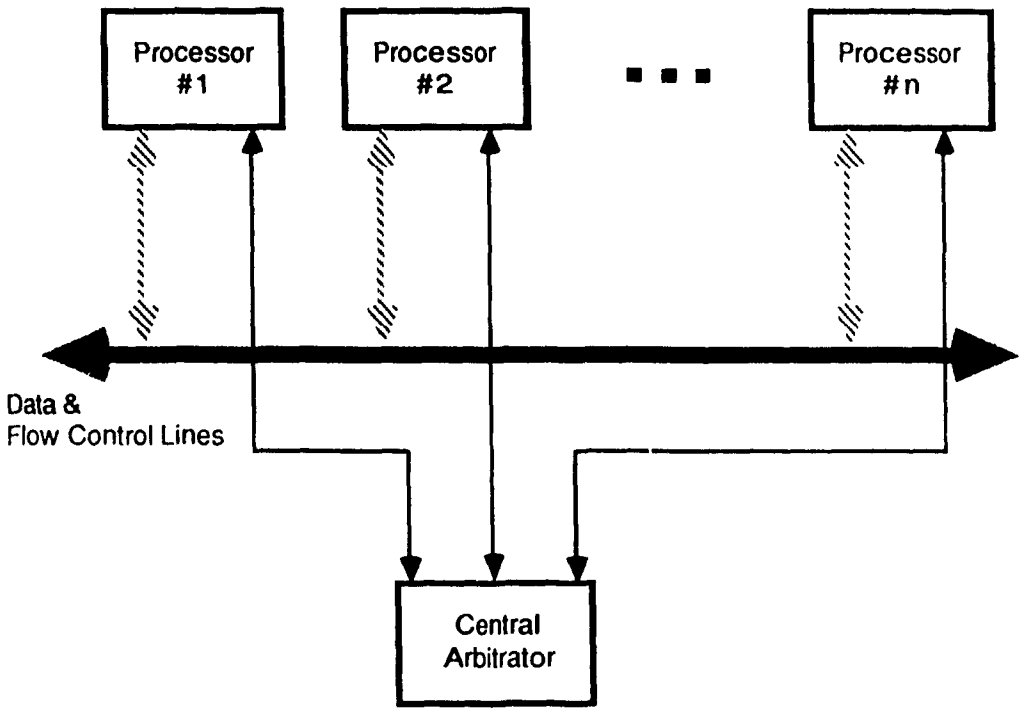
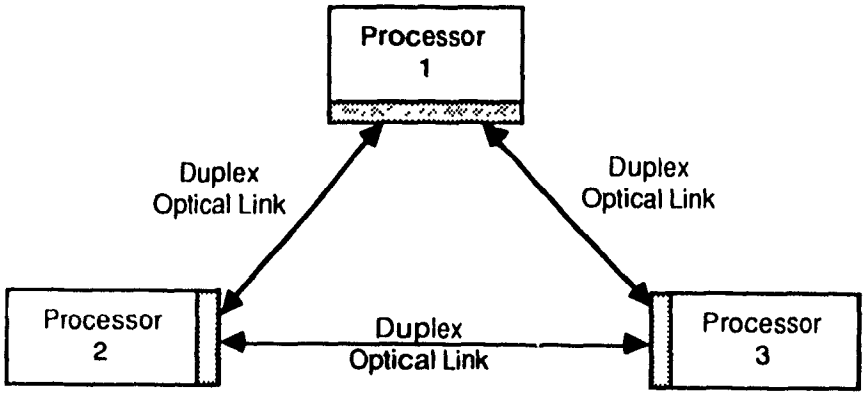
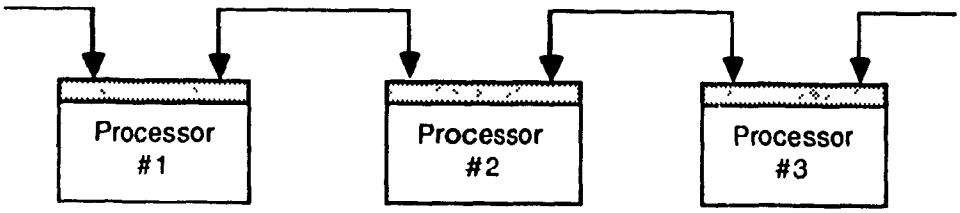


Figure 6. Parallel Bus Systems.



Point-to-Point Topology with Dedicated Links



Point-to-Point Links in "Repeater" Topology

Figure 7. Optical Point to Point Links.

and bus protocols. The detailed architecture and design of the Host Computer Interface are presented in the following chapter.

4.0 FUNCTIONAL REQUIREMENTS OF THE HCI

The Host Computer Interface was designed to support three basic communication functions between the processors in GRADS. The primary requirement was to provide a high speed data transfer capability between processors. Since typical block sizes were expected to be several kilobytes long, direct memory access to the memories of the processors was considered to be the most effective implementation alternative. Due to the different computers in GRADS and their different bus protocols, one of the key requirements was to manage block moves across different word size boundries, leading to a word packing/unpacking capability in the HCI. To accomodate the different bus protocols, the HCI was designed as a distributed system, with a central process controller and satellite "personality cards" which translate a basic data transfer handshake protocol into specific signals and sequences required by the system being interfaced.

For the purpose of moving short blocks of information, the DMA approach was not deemed efficient. Therefore a control and status word transfer capability was also required of the HCI to support system control information transfer. Since such information in a system is critical to its real time performance, a higher priority for such single word exchanges was built into the system.

In addition to the control and status word transfer capability, an interrupt routing subsystem was built into the system. This allows the processors to route interrupts to other processors as part of the interprocessor signalling sequence. To avoid system lockout due to a "runaway" processor, the interrupt subsystem has some restrictions on its usage.

The features of the HCI based on these functional requirements, and are described in the next chapter.

5.0 ARCHITECTURE OF THE HCI

The Host Computer Interface has been designed as an integral, independent workable system The Host Computer Interface idles and remains dormant till such time as a request for inter-module communication is received by it If a request is received by the HCI while it is busy, the request is ignored. It should be noted that it is the responsibility of the requesting device to maintain it's request in case the request is not granted by the HCI since the HCI does not log the requests to service them in a queue fashion In the event the HCI receives more than one request simultaneously, it grants it's resources on a pre-determined or fixed priority basis The typical service request cycle for data transfer, control/status word transfer or interrupts, is represented by the state diagram in Figure 8.

The HCI is made up of four main modules:

1. The Control Register module,
2. The DMA Controller module,
3. The Interrupt module, and
4. The Data Path module.

5.1 Control Register Module

The Control Register module is used to store the information that defines the terms and conditions of the requested data transfer The registers are loaded initially by the computer to which the HCI is granted. It is then the responsiblity of the requesting computer to provide the correct information and signal to the HCI when the process may be started. The registers are used to contain such relevant information as the source and destination of the data transfer, the source and destination addresses, the number of words of information to be transferred and the packing

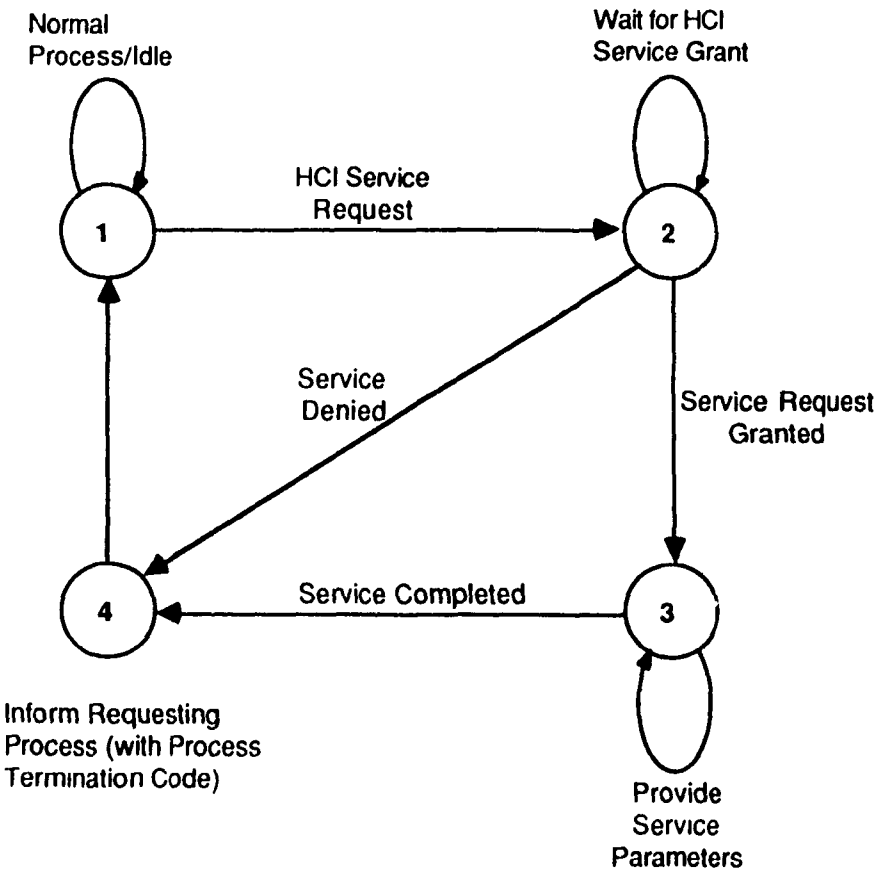


Figure 8. HCI Service Provision Cycle - State Diagram.

and unpacking of the word required Some of the registers in the Control Register module are reserved for readback only. These registers provide information to the external units regarding the status of the HCI and of any error conditions that might have been generated by the data move. A third category of registers is reserved for control signals which are used to request and start the HCI Figure 9 illustrates the partitioning of register access for a four processor system Each processor can access it's own local initialization registers and it's own control status registers. Access across processor boundaries is denied The innermost layer of DMA control registers are accessible to any processor in the system, however forced arbitration ensures that only one processor has access during the initiation of a data transfer transaction Once the requested process is underway, the Control Registers are not accessible to any external device This feature prevents accidental modification of any process that is already underway

5.2 DMA Controller Module

Closely related with the Control Register module is the DMA Controller module The function of the DMA Controller module is to generate the actual sequence of control signals for the DMA process. Inextricably linked with the DMA Controller module are the Control Registers whose contents define the parameters for the DMA process (Figure 10) The source and destination memory codes (4 bits each) point to one of the sixteen memories in GRADS from where the data is read by the DMA machine, and to which the data is written by the DMA machine The source address (24 bits) points to the address in the source computer's memory where the DMA read cycle starts Since this pointer is incremented after each successive read, it initially points to the beginning of the data block to be transferred. Similarly the destination address (24 bits) points to the address in the destination computer's memory where the DMA write cycle starts The word count (16 bits) indicates the number of data words to be transferred, or the size of the data block to be moved. Since the source and destination word counts may be different due to different bus widths, it always refers to the number of words to be written into the destination

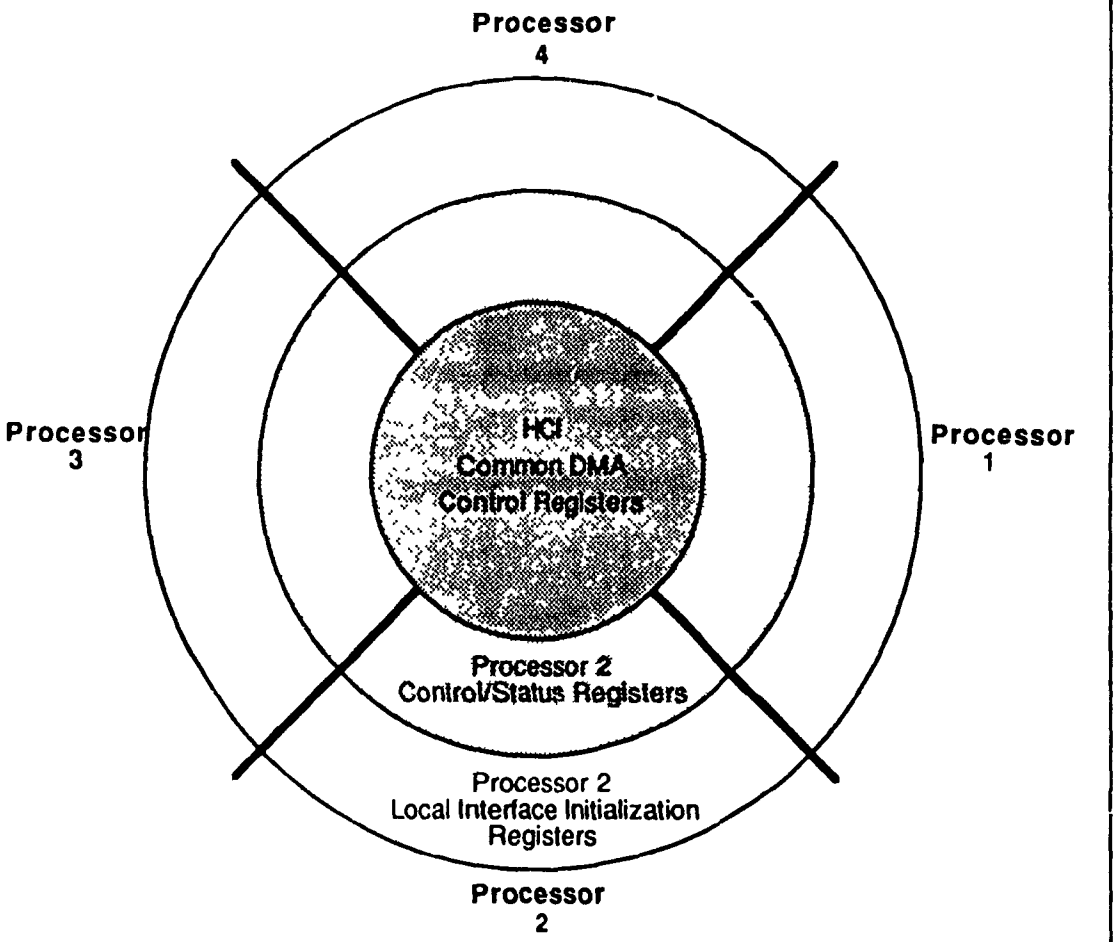


Figure 9. HCI Control/Status Register Classes.

memory. The packing unpacking code (4 bits) identifies one of the sixteen possible source destination bus width ratios This code is used to pack or unpack data words to ensure compatibility with source and destination memory widths The heart of the DMA Controller module is the DMA state machine which generates the actual signals of communication between the DMA module and the satellite interface module It has been realized as a sophisticated ROM sequencer to carry out the process in an efficient fashion with a minimum amount of hardware and a maximum degree of flexibilty The rest of the circuitry in the DMA Controller module provides the steering logic to the various processor units

The DMA Controller module regards each computer or each bus as the same regardless of the size and the protocols. The actual synchronization with a particular bus is carried out by a specific satellite system which matches the DMA Sequencer's request with the bus protocol of that particular computer The communication between the DMA Controller module and the various computers is basically asynchronous, consisting of a request-grant protocol Figure 11 shows the DMA Controller module initiating a DMA read write request with a specific target machine via it's interface card Internally the DMA Controller module has been designed as a state sequencer clocked at 10MHz.

5.3 Data Path Module

The Data Path module provides the actual path for the flow of data words during a DMA process This module provides a pipelined path to speed up data transfers between two computers The data path is 40 bits wide and can therefore accept the largest word available in the system Smaller words or bytes are sucessively strobed into various fields of this 40-bit register to optimize memory usage. Similary, the unpacking of a large word can be carried out This feature makes it possible to swiftly transfer data between various memories and processors of different word sizes.

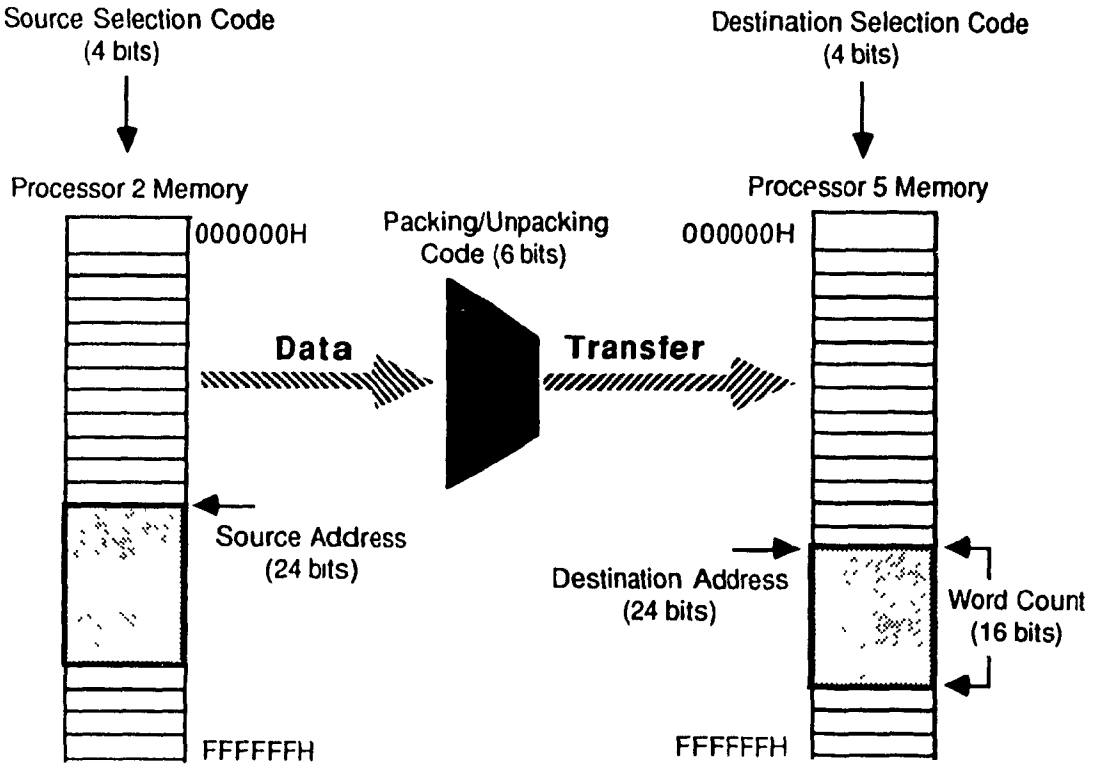


Figure 10. Data Transfer Parameters in the Control Registers.

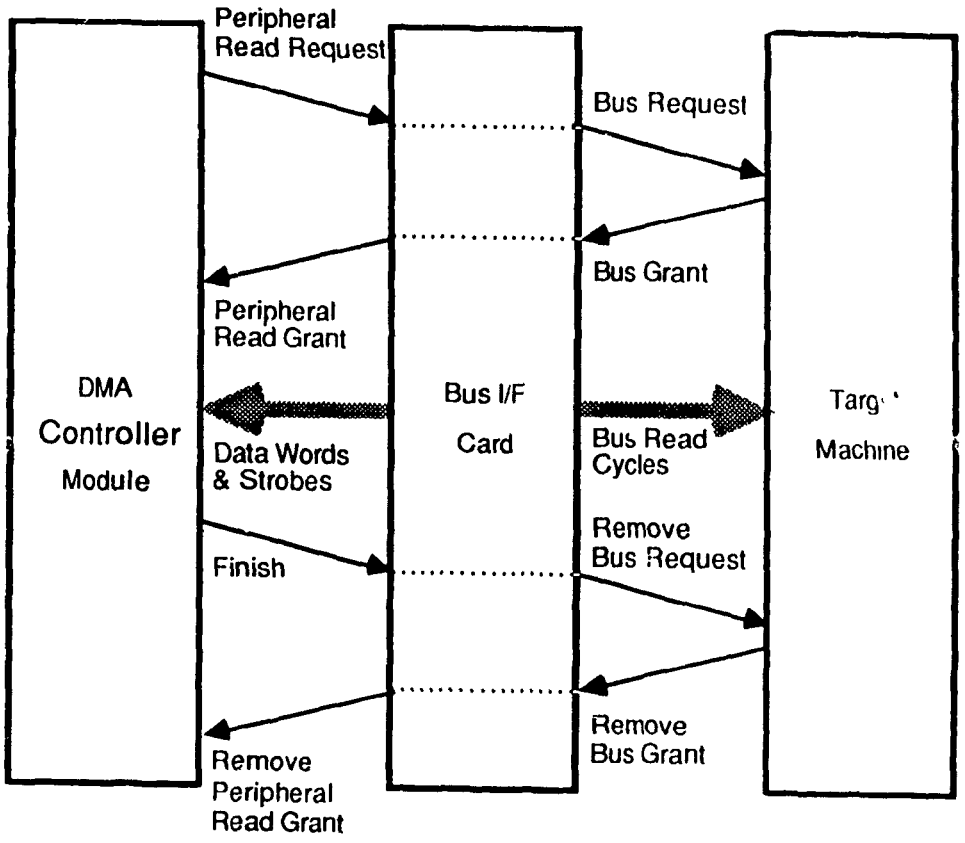


Figure 11. DMA Sequencer Request/Grant Protocol.

5.4 Interrupt Module

The Interrupt module is used to route the interrupts from one processor to another as well as from the HCI to a processor. Interrupts are also used as an efficient means of signalling the completion of a task assigned to the HCI. This prevents the processor from waiting and checking for the completion of a requested block move. The interrupts issued by the Interrupt module are routed to the appropriate satellite interface which follows up the exact interrupt procedure of the specific computer system. The interrupt facility may be overridden by the requesting device by loading an interrupt disable bit in the appropriate control register. This is an important feature for blocking out unwanted or undesirable interrupts for a certain duration of time

The overall architecture of the entire interface is given in Figure 12. The units enclosed by the shaded block are collectively called the DMA Controller module.

The HCI undergoes four distinct states when it is engaged by some requesting device till its task is brought to completion. In state one, the HCI idles. It proceeds to state two for the requesting device to load the control registers and request the HCI to start execution. Upon receipt of the "Start DMA" signal, it goes into state three where it carries out the desired process and then enters state four to await the termination of the job. It then returns to state one or the idle condition. The four states are depicted in Figure 13.

The interaction between these modules is described in the following chapter.

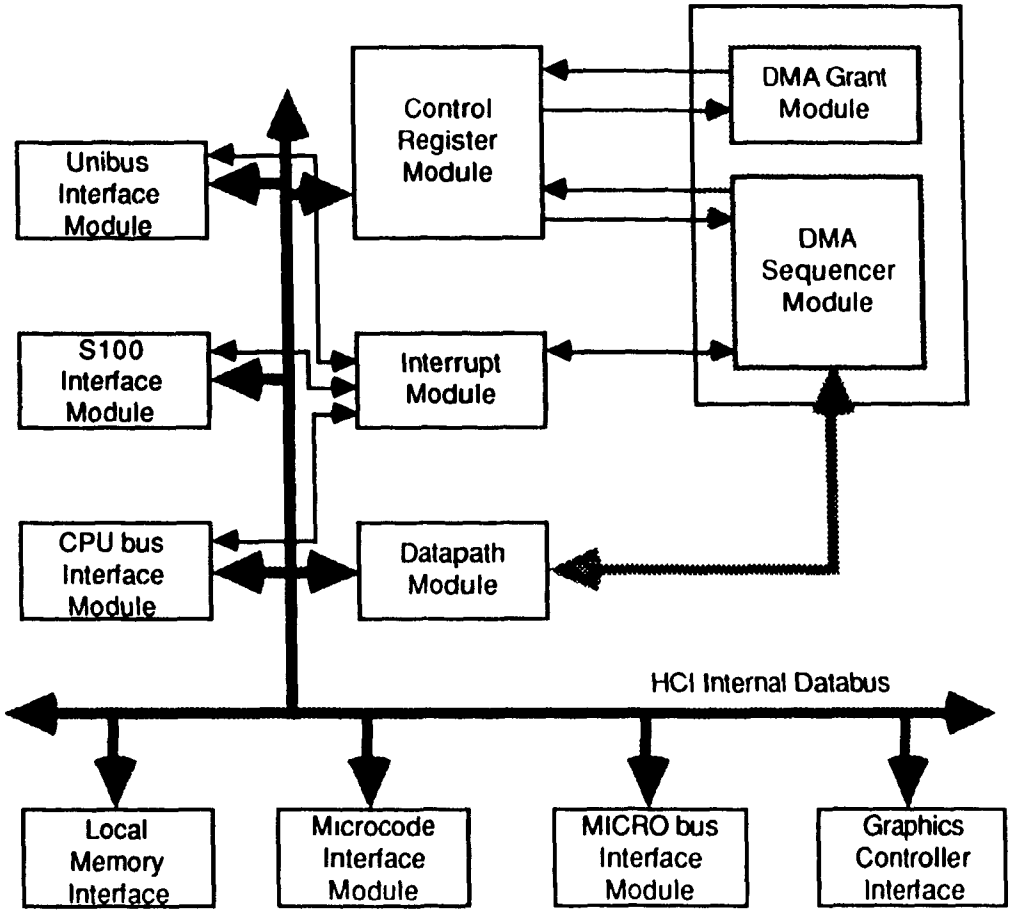


Figure 12. Overall Architecture of the HCI.

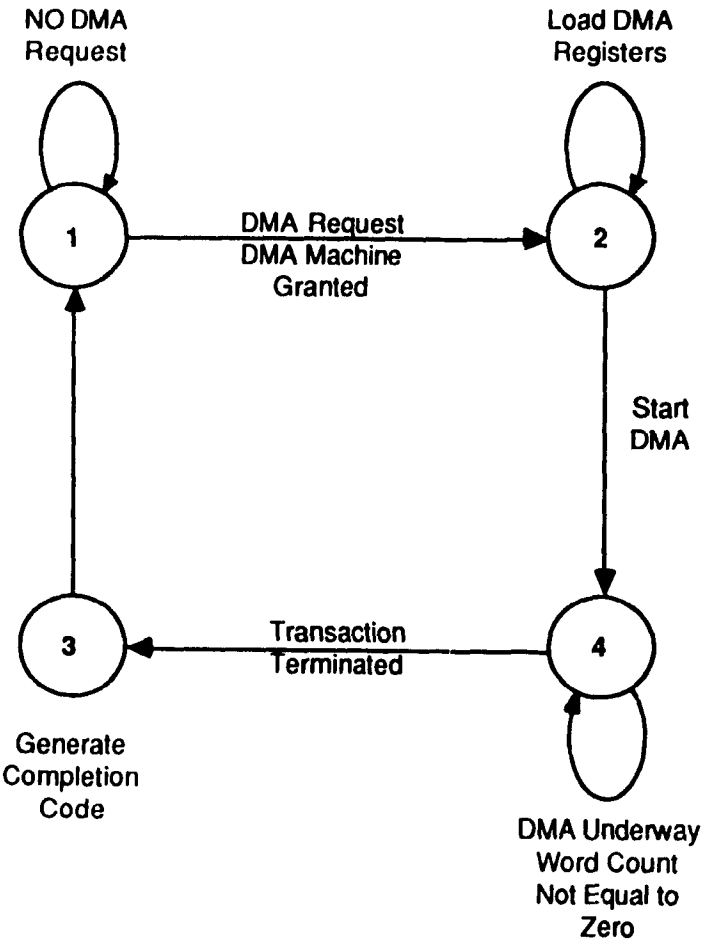


Figure 13. DMA State Transitions.

6.0 HCI INTERFACES AND PROTOCOLS

The Host Computer Interface links the following different types of systems via the appropriate bus protocols.

1. The VAX 11/780 via the Unibus
2. The Cromemco Z-80 via the S-100 or the IEEE 696 bus.
3. The array of microprocessors at layer 2 via the CPU bus.
4. The Video Memory via the MICRO bus.
5. The Graphics Controller via the GRACON bus.

Each of these interfaces is functionally similar for data interchange, but with different privileges and restrictions. This chapter describes each one of these interfaces and their respective protocols in detail.

6.1 The CPU Bus

The CPU bus is made up of 40 bidirectional data lines, 20 bidirectional address lines, and 12 control signals as shown in Figure 14. The data lines carry the data between the HCI and the designated microprocessor module during block moves, control status word transfers and HCI resource requests. All microprocessors do not use the full 40 bit width of the data bus, only the lower 16 or 20 bits depending on their word size The address lines carry the source or destination address of the data being moved across the data bus Address lines can also carry the address of the HCI control registers being loaded or queried by a microprocessor module

When the HCI wants to move a block of data to or from a specific microprocessor module, it asserts a CPU bus DMA REQUEST When the target microprocessor is in a position to grant the HCI access to it's bus, it responds with a CPU bus DMA GRANT (Figure 15) Upon receipt

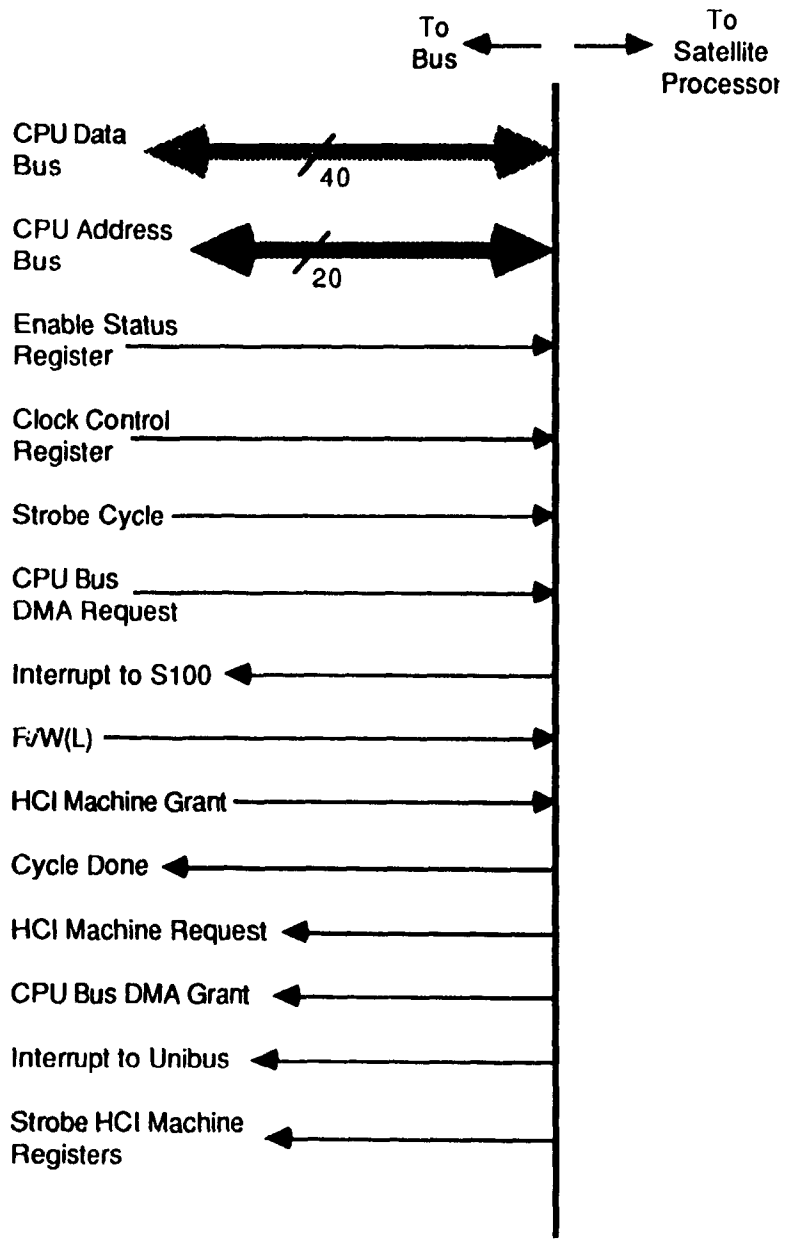


Figure 14. CPU Bus Signal Assignment.

of the CPU bus DMA GRANT signal, the HCI proceeds to generate the address and the read/write signal, and transfers each word of information with a STROBE signal Each strobe is acknowledged by a CYCLE DONE signal from the microprocessor When the designated block of information has been moved, the HCI drops the CPU bus DMA REQUEST, which in turn causes the microprocessor to drop the CPU bus DMA GRANT signal and regain mastership of the bus. In reality, a maximum of 16 words of data are transferred in each bus request grant cycle. This limit was imposed to allow higher priority tasks such as control status word transfers to be carried out in an interleaved fashion on the same bus This interleaving, however, remains totally transparent to the originating processor and is managed entirely by the HCI The implementation of this feature is described in a subsequent chapter

Two of the control signals on the CPU bus are "one hot" lines that control the strobing of the control and status registers of each microprocessor The CLOCK CONTROL REGISTER signal strobes the contents of the data lines into the control register of the microprocessor Similarly the ENABLE STATUS REGISTER signal gates the contents of the status register onto the data lines. Typical information carried in the control and status registers is given in Figure 16.

Each microprocessor can generate interrupt requests for either of the host computers, typified by the INTERRUPT TO S100 and the INTERRUPT TO UNIBUS signals When asserted, they cause an interrupt in the appropriate system if the interrupt enable bit in the HCI has been set by the target machine to receive interrupts Microproprocessor modules are not provided a means of generating lateral interrupts to other microprocessors

Microprocessors on the CPU bus may, however, initiate block data moves by requesting HCI services through the HCI MACHINE REQUEST signal. When the HCI is ready to service the request, it returns a HCI MACHINE GRANT signal at which point in time the microprocessor can access and load the control registers of the HCI with the specifics of the requested move This is done by putting the register address on the address lines, the data on the data lines, and

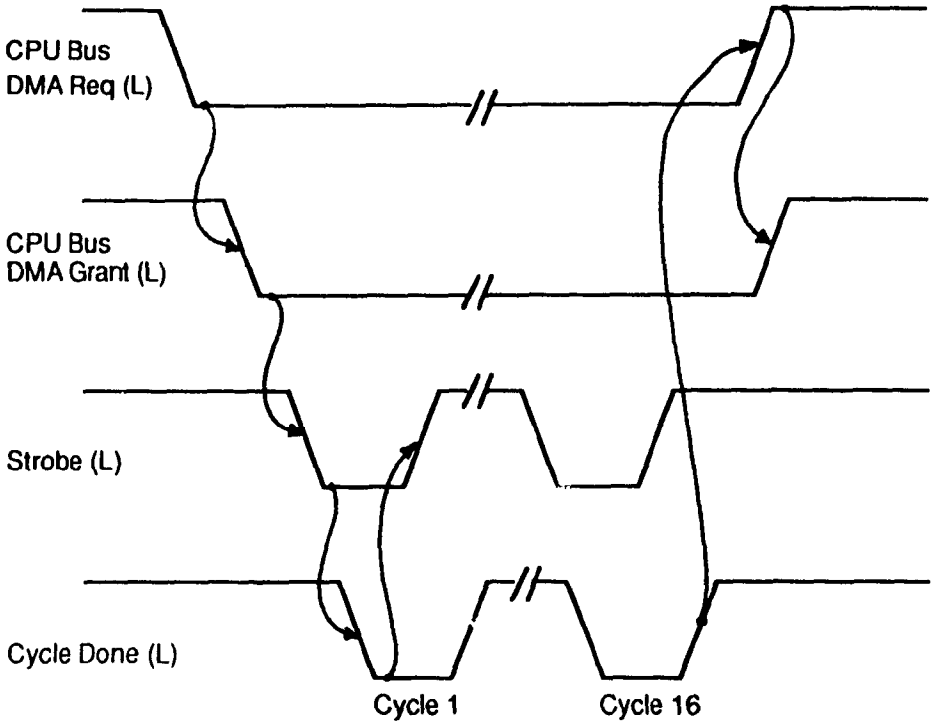


Figure 15. CPU Bus Request/Grant Signal Sequence.

Bit	Name of Signal
0	Not used
1	Interrupt to Unibus
2	HCI Machine Request
3	Not used
4	Interrupt to S100
5-12	Not used
13	MICRO Bus Error
14	Input Buffer 1 Full Flag
15	Input Buffer 2 Full Flag

CPU Status Register

Bit	Name Of Signal
0	MICRO Bus Error (GRACON)
1	MICRO Bus DMA Done (GRACON)
2	HCI DMA Done
3	Unibus Interrupt
4	S100 Interrupt
5	HCI Machine Grant Interrupt
6	Reset System
7	Reset
8	NMI
9-13	Not used
14	Input Buffer 1 Full
15	Input Buffer 2 Full

CPU Control Register

Figure 16. CPU Bus Control/Status Registers.

strobing the data via the STROBE HCI MACHINE REGISTER line. As mentioned in previous sections, 16 registers need to be loaded prior to DMA initiation. The loading of the 16th register (start DMA) causes the HCI MACHINE GRANT to be dropped and all further access to the HCI control registers to be denied (Figure 17).

6.2 The S100 Bus

The S100 or the IEEE-696 bus is the oldest standard bus for small 8 bit computer systems typified by the Z-80 processor and the CPM-80 operating system. In this application, a Cromemco Z-80 system was used as an alternate host computer, and hence the HCI was required to interface with the system. The Cromemco system adheres to the S100 bus standard and runs CDOS, a CPM-80 like operating system

The S100 bus consists of two data busses, DATAIN and DATAOUT, each 8 bits wide, and carrying data to and from the CPU respectively. The 16 bit wide address bus carries the I/O port or memory address for any transaction The remainder are control signals such as RESET, MEMORY READ & MEMORY WRITE (indicating a memory cycle), POUT & PIN (indicating an I/O port cycle), and PDBIN & PWRT (the read and write strobes). There are two further sets of signals to interrupt the operation of the CPU by an external device. The INTERRUPT REQUEST signal is generated by an external device for interrupting the normal processing. The CPU, if interrupts have been internally enabled, responds with an INTERRUPT ACKNOWLEDGE signal and begins the execution of an interrupt service routine. The Z-80 vectored interrupt mechanism allows multiple interrupt requests to be distinguished without device polling. Another signal, HOLD REQUEST, can be used by an external device to temporarily cause the CPU to stop its processing and hand over the control of the bus to the external device. To signal the relinquishing of the bus, the CPU uses a HOLD ACKNOWLEDGE signal. These signals are extensively used by the HCI to communicate with the S100 bus. A complete list of S100 bus signals is shown in Figure 18

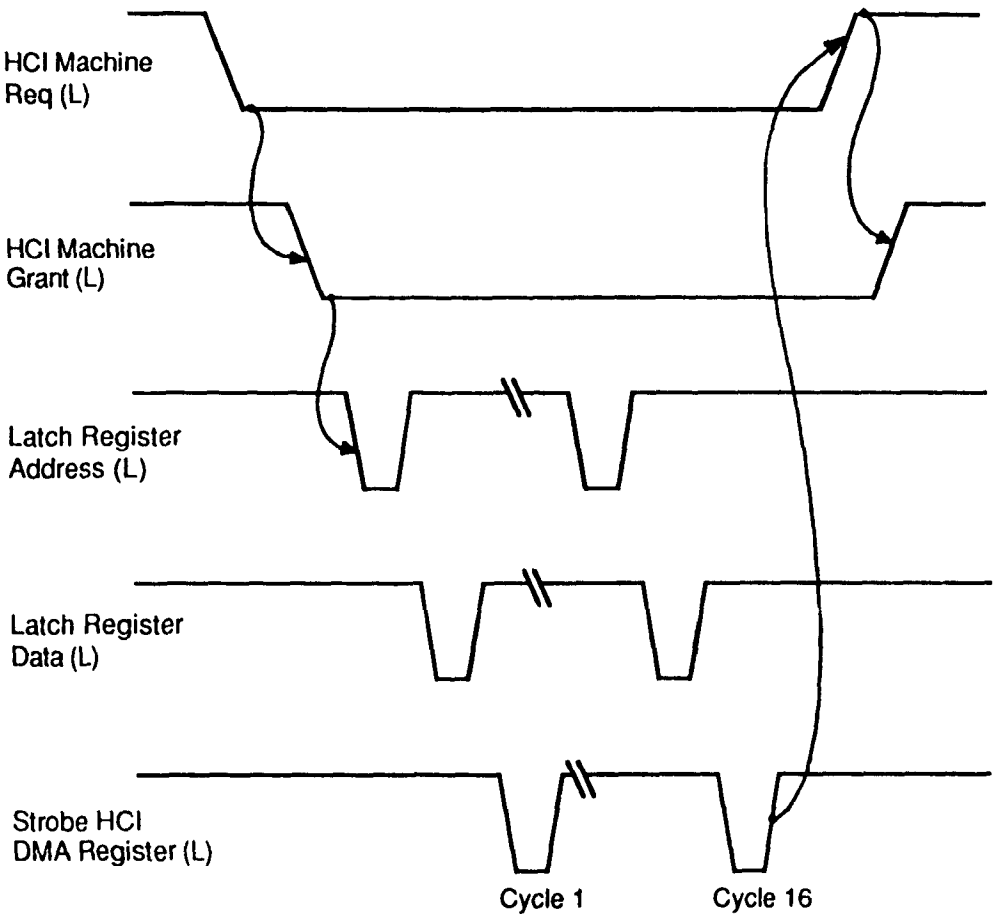


Figure 17. HCI Machine Register Loading Protocol.

Pin No	Signal Name	Pin No	Signal Name
1	+8V	51	+8V
2	+16V	52	-16V
3	XRDY	53	GND
12	NMI	68	MWRT
13	PWRFAIL	72	RDY
24	Clock B	73	INT
25	pSTVAL	74	HOLD
26	pHLDA	75	RESET
29	A5	76	pSYNC
30	A4	77	pWR
31	A3	78	pDBIN
32	A15	79	A0
33	A12	80	A1
34	A9	81	A2
35	DO1	82	A6
36	DO0	83	A7
37	A10	84	A8
38	DO4	85	A13
39	DO5	86	A14
40	DO6	87	A11
41	DI2	88	DO2
42	DI3	89	DO3
43	DI7	90	DO7
44	sM1	91	DI4
45	sOUT	92	DI5
46	sINP	93	DI6
47	sMEMR	94	DI1
48	sHLTA	95	DI0
49	CLOCK A	96	sINTA
50	GND	97	sWO
		99	POC

Figure 18. Table of S100 Bus Signals.

To map the HCI DMA module signals into S100 bus signals, a special card was designed to perform the function via a micro-programmed ROM sequencer machine Physically, this circuit was built on an S100 size card and plugged into a single card slot in the Cromemco system The ROM sequencer approach was favoured over hardwired logic due to flexibility and ease of implementation reasons. Figure 19 shows the details of the incomming and outgoing signals from the S100 card of the HCI

When the HCI wishes to read a block of data from the S100 memory, it sends a READ REQUEST to the S100 card This causes the ROM sequencer state machine to exit from the idle mode and generate a HOLD REQUEST on the S100 bus. When the S100 CPU relinquishes the bus, it acknowledges the HOLD REQUEST by a HOLD ACKNOWLEDGE The HOLD ACKNOWLEDGE cause the next state change in the ROM sequencer, which then proceeds to generate control signals disabling the CPU bus drivers and enabling its own bus drivers onto the S100 bus. In the next state, the ROM sequencer starts to read bytes of data from the memory of the S100 system by emulating the S100 memory read signals and the address (which comes from the source address regsiter in the HCI) As each byte is read, the ROM sequencer generates a strobe signal to allow it to be stored in the data path module in the HCI, and also generates a INCR SOURCE ADDR signal to advance the source address by one When the data path module in the HCI is full, it sends a FINISH signal to the ROM sequencer, which causes it to drop the control of the S100 bus by de-asserting the HOLD REQUEST signal

The write operation is essentially similar to the read operation except that the ROM sequencer receives a WRITE REQUEST, and that the ROM sequencer generates a WRITE GRANT and INCR DEST ADDR signals during the cycle. The read and write operation sequences are detailed in Figure 20.

The S100 interface card has some additional logic on board to allow the S100 system to access the HCI machine registers as I O ports The port address decode circuit maps the HCI control

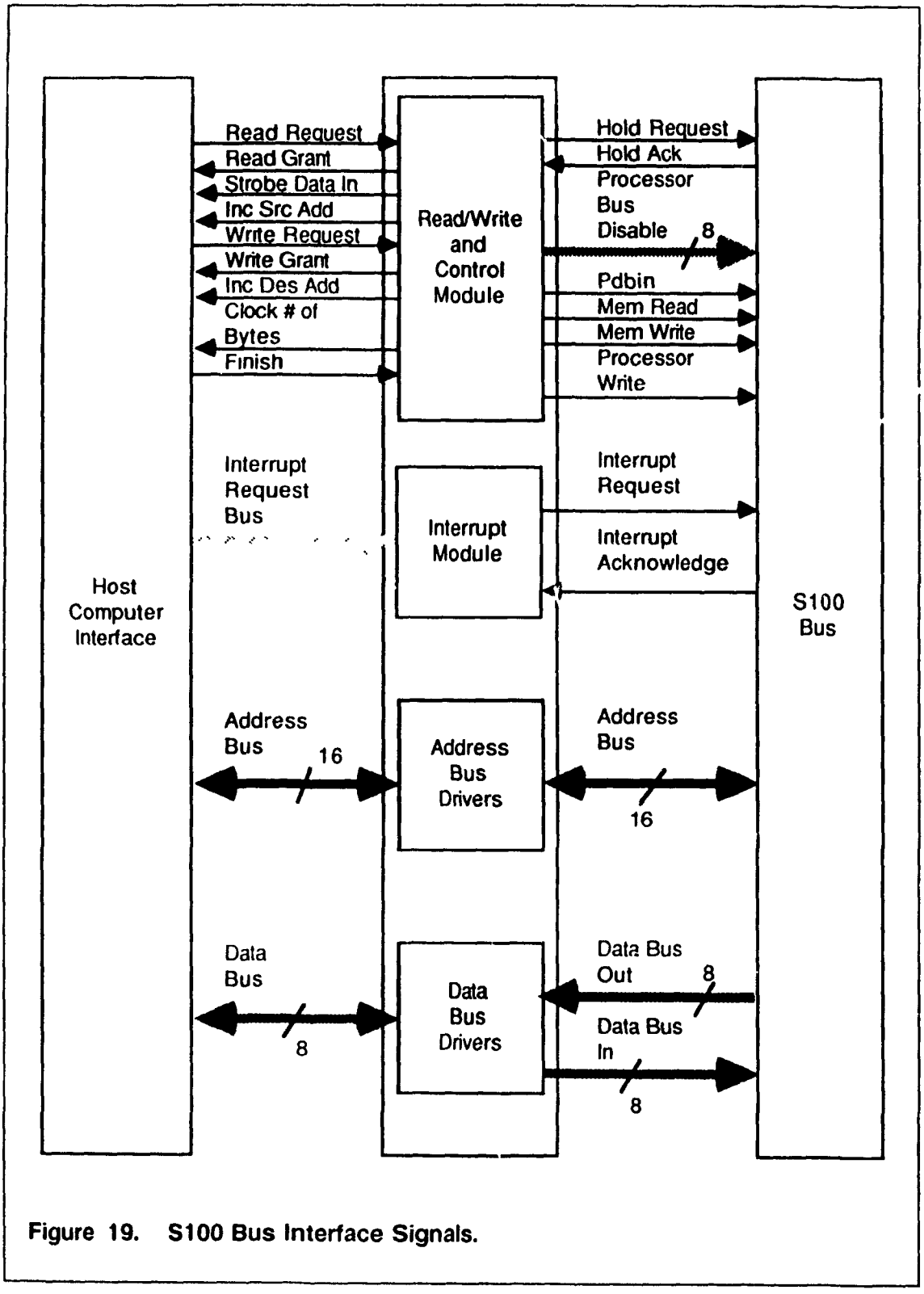


Figure 19. S100 Bus Interface Signals.

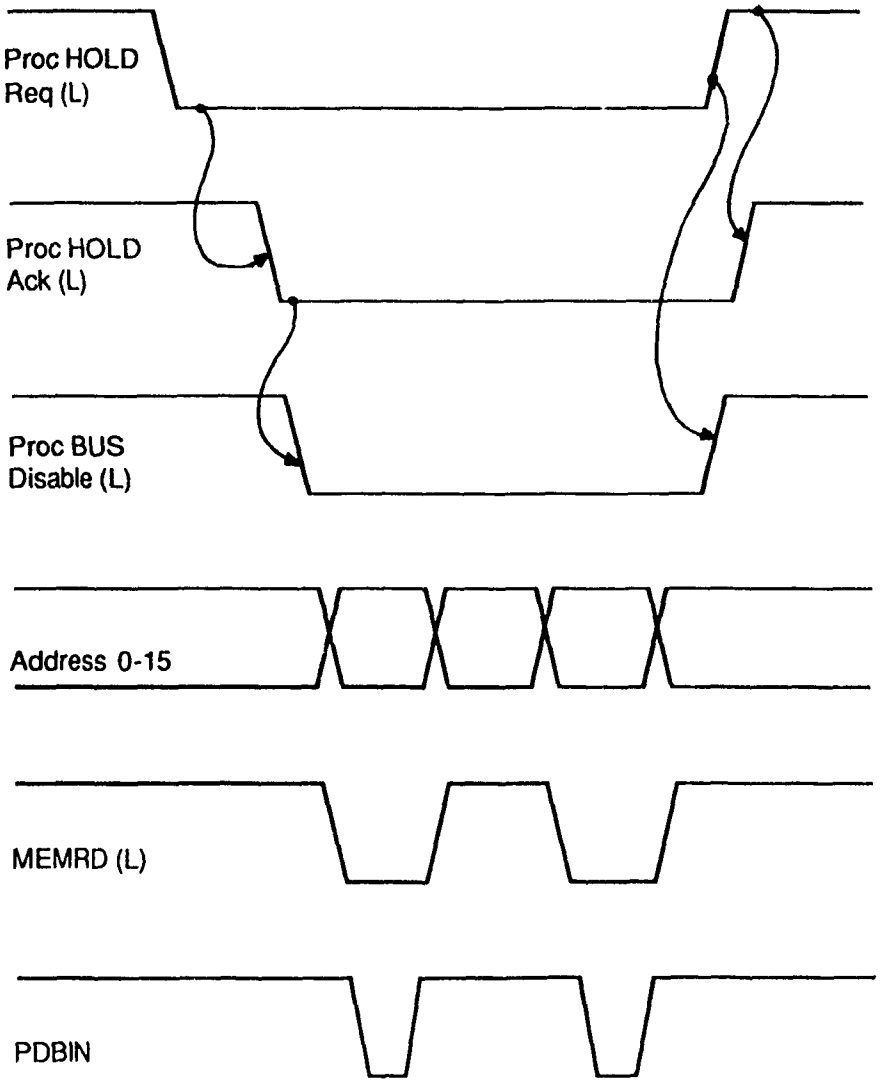


Figure 20. S100 DMA Read/Write Timing.

registers onto the Z-80's I/O space, the base address of which is selectable by a DIP switch on board the card The S100 port assignments for GRADS are given in Figure 21. Any interrupt signals from the HCI destined for the S100 bus are mapped onto the S100 INTERRUPT REQUEST line. Upon receipt of the INTERRUPT ACKNOWLEDGE signal, the logic provides an interrupt vector to the S100 system. The interrupt vector can be preloaded by the S100 system onto the interface card by S100 system upon initialization.

The implementation of the S100 interface card ROM sequencer was actually carried out using static RAMS which can be loaded by the S100 system upon initialization There were two reasons for this Firstly debugging microcode that is downloadable makes the system troubleshooting much easier than having to burn new ROMs. Secondly, the speed of the EPROMS available at the time of hardware design constrained the operation of the sequencer to less than 4 MHz. The RAM based machine can run upto 10 MHz Having a RAM sequencer requires some mechanism to prevent the state machine from executing random microcode upon power up. An on board flip-flop locks the state machine till such time as the microcode has been downloaded and the RAM sequencer has been enabled.

6.3 The UNIBUS Interface

The UNIBUS is common to DEC computers such as the PDP-11 and the VAX-11 family. Unlike the more primitive S100 bus, it is and asynchronous bus with a handshake protocol between the bus master (normally the CPU) and the memory. In addition, the UNIBUS has a "look-ahead" arbitration scheme, whereby the next bus user is selected concurrently with an ongoing cycle. A typical memory read cycle on the UNIBUS is shown in Figure 22.

Due to the edge-driven nature of most UNIBUS signals, a ROM sequencer type implementation was not considered suitable. Keeping in line with the DEC design approach, dedicated logic was used around monostable devices to generate the sequence of UNIBUS signals in response to

Base Address = C0 Hex

S100 Local Initialization Registers

00H	Address of Microcode RAM (L)
01H	Address of Microcode RAM (H)
02H	Microcode Word (L)
03H	Microcode Word (H)
04H	Microcode Enable

S100 Bus Private Registers

05H	Control Word (L)
06H	Control Word (H)
07H	Status Word (L)
08H	Status Word (H)
09H	Issue Interrups (0-7)
0AH	Request HCI Services
0BH	DMA Status

S100 Bus DMA Registers

0CH	Source/Destination Code
0DH	Source Address (0-7)
0EH	Source Address (8-15)
0FH	Source Address (16-23)
10H	Destination Address (0-7)
11H	Destination Address (8-15)
12H	Destination Address (16-23)
13H	Word Count (0-7)
14H	Word Count (8-15)
15H	Packing/Unpacking Code

Figure 21. S100 Port Assignments for GRADS.

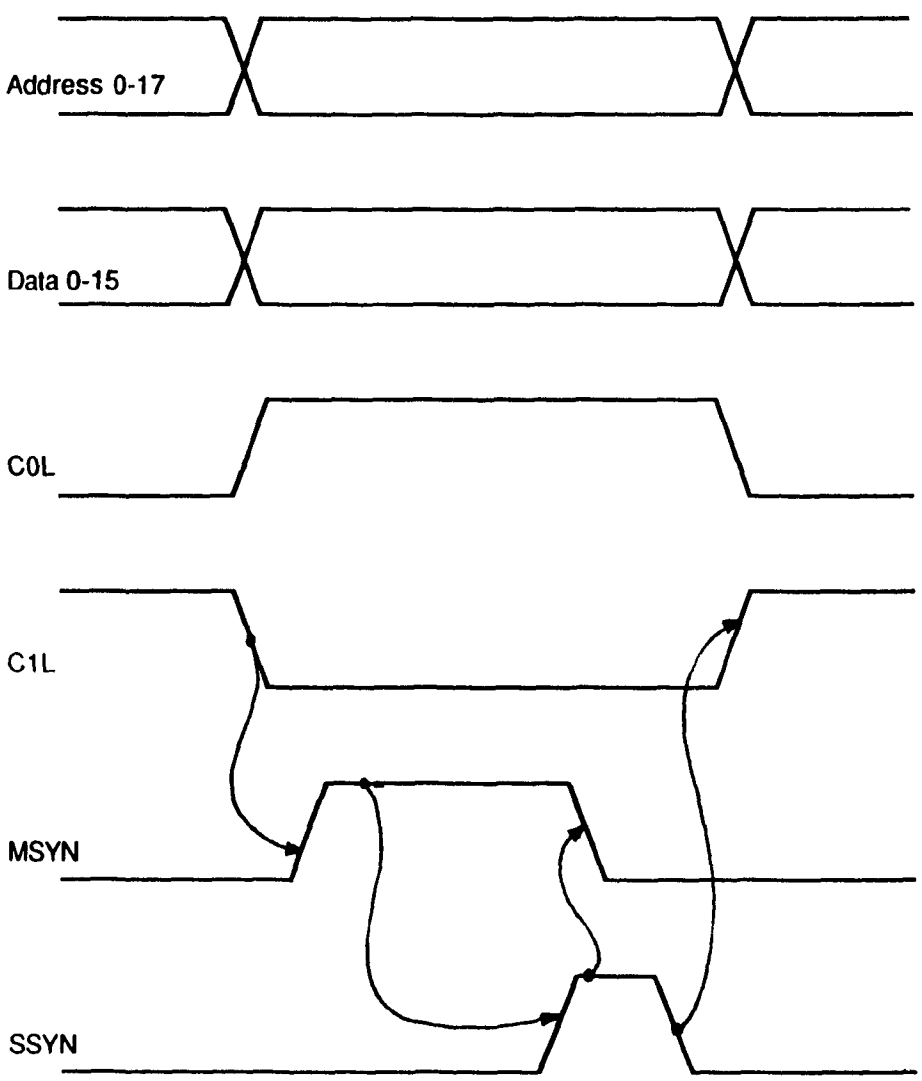


Figure 22. UNIBUS Memory Read Cycle.

the requests from the HCI. Figure 23 shows the list of the HCI and UNIBUS signals The UNIBUS interface card plugs into a vacant slot in the VAX-11 or the PDP-11, and maps some of the address space onto the HCI's control registers, in a manner similar to the S100 bus A typical DMA sequence on the UNIBUS is shown in Figure 24.

6.4 The GRACON bus Interface

This bus is used by the HCI to download the microcode into the RAM based state machine of the graphics controller. This bus is fully controlled by the HCI and consists of 12 unidirectional address lines, 8 bi-directional data lines, and 3 control lines (Figure 25).

Of the control lines, LOAD GRACON RAMS and READ BACK RAMS , are used to define the mode of operation of the state machine (Figure 26). In the normal operational mode, all three control lines must be held high. In the load mode, the third control signal, WRITE STROBE is used to strobe the data present on the data bus into the graphics controller's memory The exact timings of the read and write cycles are shown in Figure 27

6.5 The MICRO bus Interface.

The MICRO bus provides a path for all the satellite processors to communicate their processed macroinstructions to the graphics controller. The graphics controller answers requests for DMA moves from each microprocessor, and moves data from the memory of the microprocessor into the video memory planes. The HCI also connects to the MICRO bus as one of the devices requiring a DMA service into the video memory planes In this mode, the host computers can read from or write directly into the video memory of the system, a feature very useful for system debugging and diagnostics.

The MICRO bus consists of 12 unidirectional address lines and 20 bi-directional data lines, which are time shared across all the units on the bus. The HCI, as well as all microprocessors,

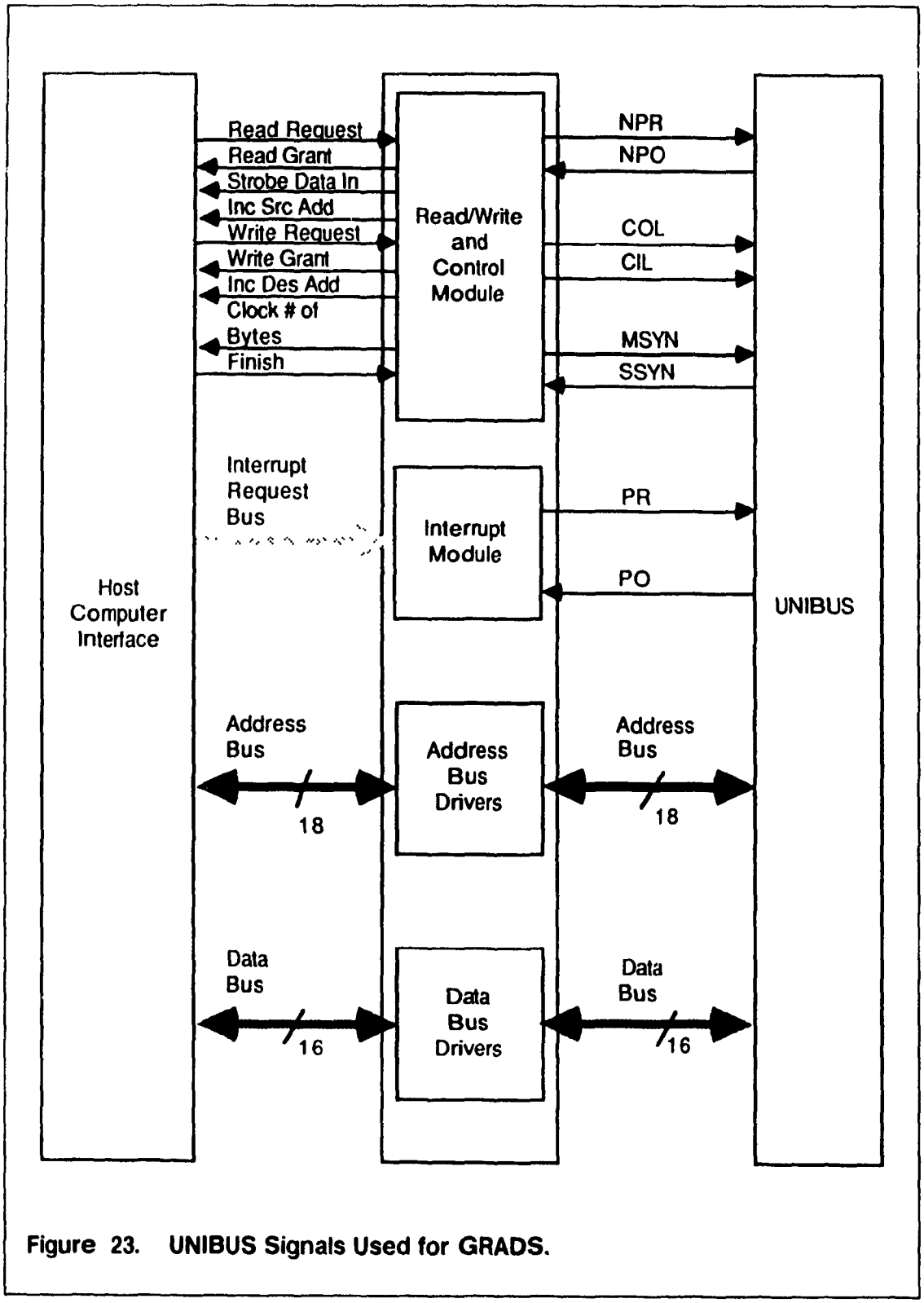


Figure 23. UNIBUS Signals Used for GRADS.

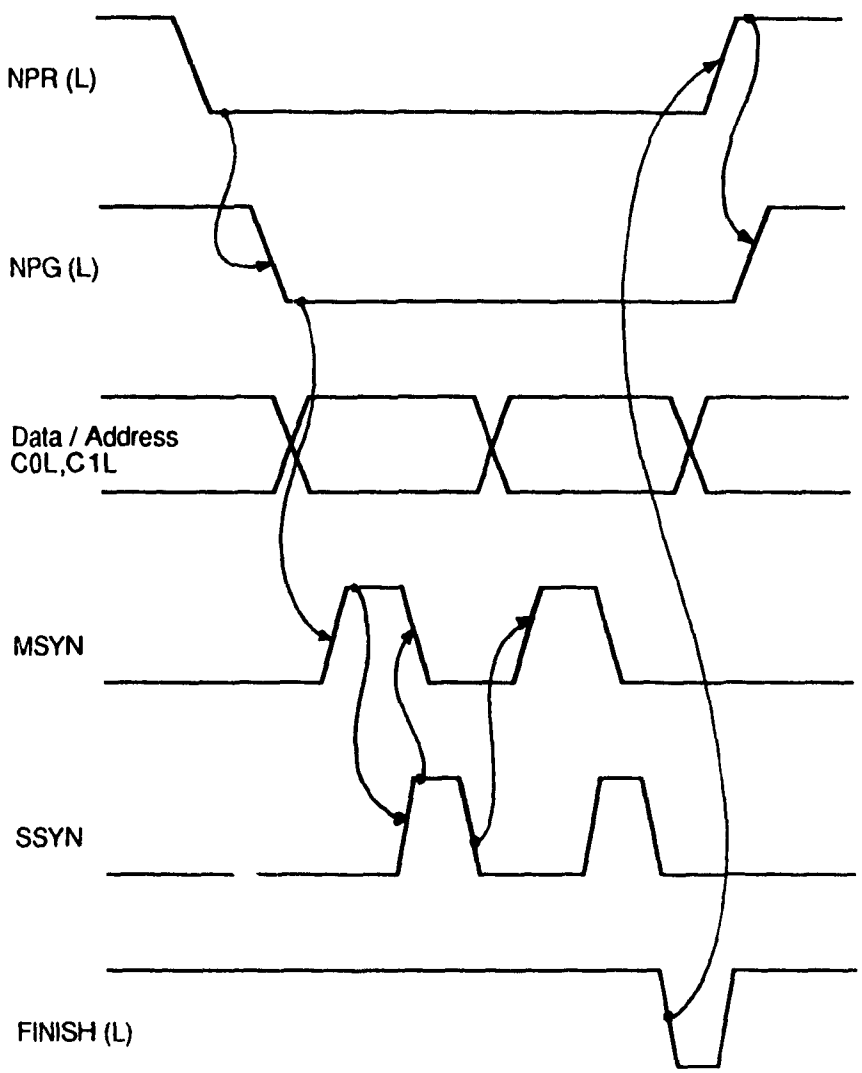


Figure 24. UNIBUS DMA Read Cycle.

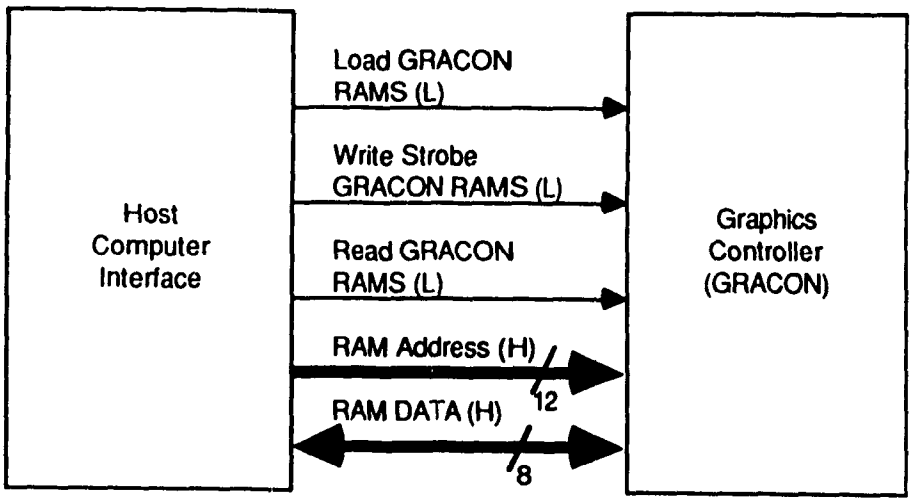


Figure 25. HCI-Graphics Controller Bus Signals.

Load GRACON RAMS	Readback RAMS	Operation Mode
Low	Low	Illegal (no activity)
Low	High	Load Mode
High	Low	Read Mode
High	High	No operation

Figure 26. Graphics Controller Bus Truth Table.

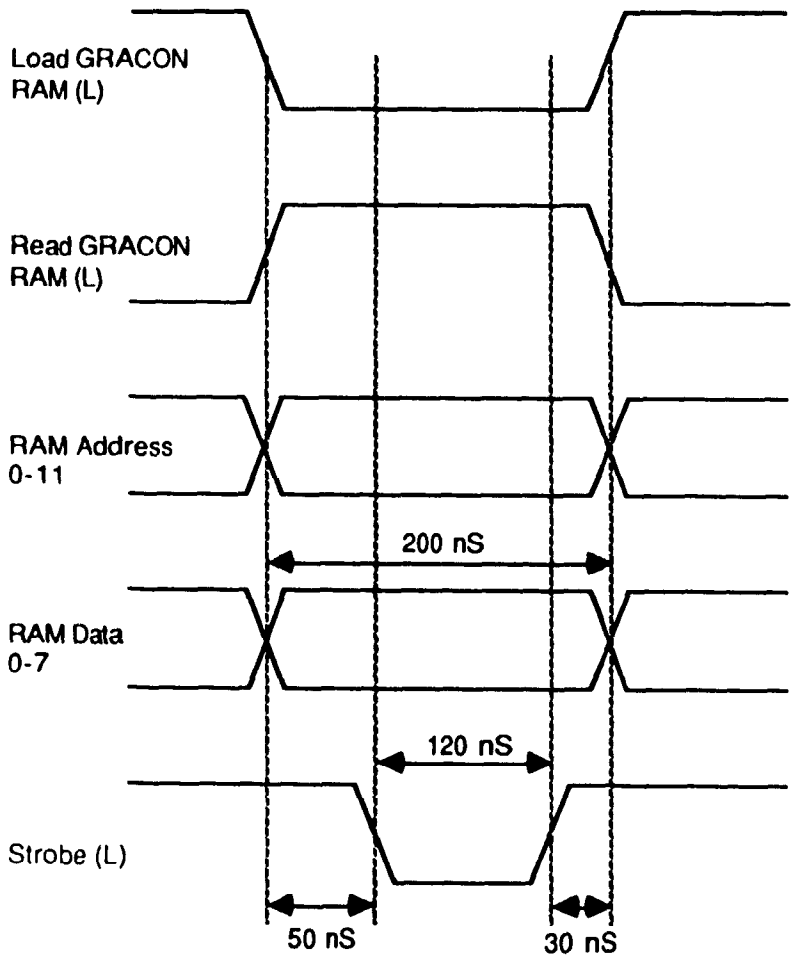


Figure 27. Graphics Controller Bus Read Write Cycle.

have individual MICRO BUS REQUEST and MICROBUS GRANT lines The request for DMA is made by asserting the request line till the grant is obtained. The end of the DMA cycle is indicated by the DMA DONE signal which causes the request signal to be dropped, which in turn causes the grant signal to be dropped The graphics controller moves data between the source and the destination on a word basis It, therefore uses two more signals to manage the flow of information within a block move The two signals are DMA CYCLE REQ and DMA CYCLE FINISH. Each word transaction is initiated by DMA CYCLE REQ assertion, along with the correct address and data. The acceptance of the information is acknowledged by a DMA CYCLE FINISH, which causes the request to be dropped, and a new cycle to be initiated Typical MICRO bus sequences are shown in Figure 28.

The bus protocols described above are essentially derived from a common DMA sequencer machine in the HCI. The detailed implementation of these interfaces, and the common HCI modules is described in the following chapter.

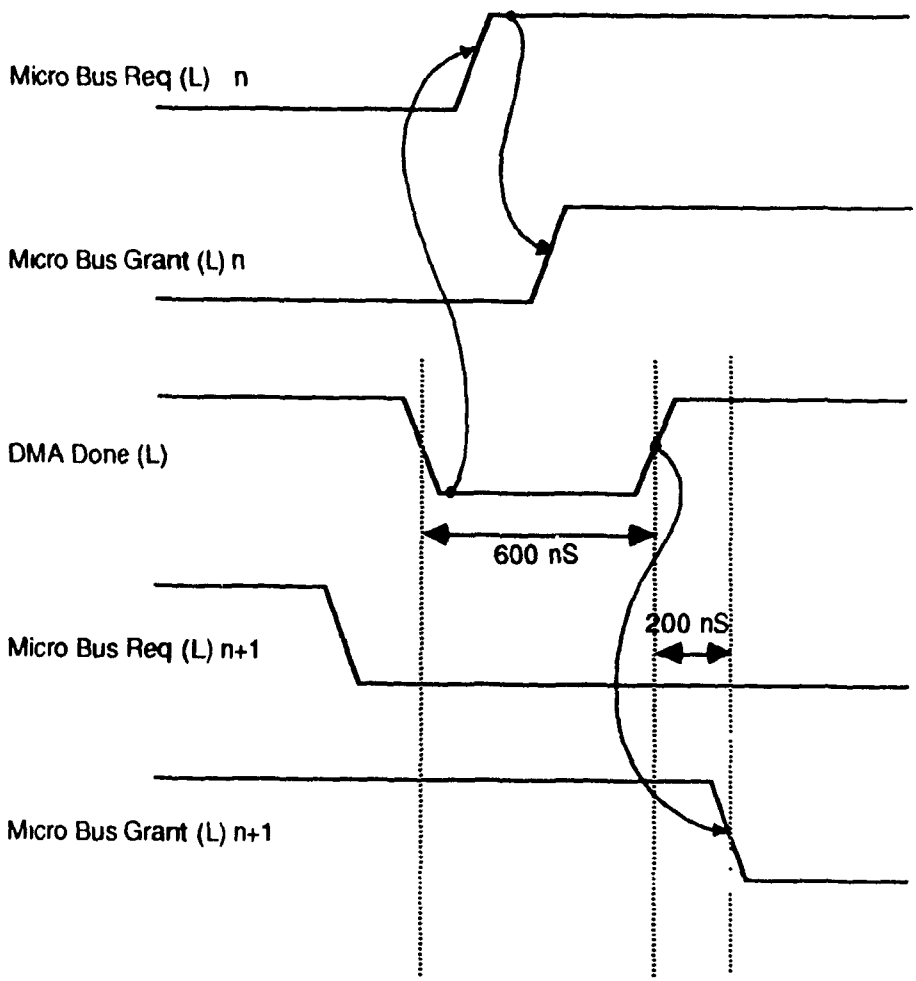


Figure 28. Micro-Bus Timing Sequence.

7.0 HCI IMPLEMENTATION

The various modules of the host computer interface were protoyped and built using SSI MSI logic devices, mainly of the low power Schottky (LS) family [39.]. Internally the design is synchronous, clocked at 10 MHz by an on-board 74S124 oscillator using a quartz crystal as the tuned element. The design contains asynchronous logic only in peripheral areas where interfacing with an asynchronously clocked bus is required.

7.1 Control Register Module

The control register module was implemented using edge-triggered octal latches 74LS374. Data from the host computers is latched into these devices asynchronously, i e they appear ~ registers in the I O space of the host computers In the case of 16 bit machines, two 8 bit registers are clocked together. The only exception to the use of latches are the DMA control registers which contain Source Address, Destination Address, and Word Count information These registers were based on a bank of 74LS193 synchronous counters with an asynchronous parallel load capability Since the Source Address and Destination Address registers contain data buffer pointers, once DMA starts they are clocked upwards with every successive source memory read strobe and the destination memory write strobe respectively The Word Count register is clocked downward with every write transaction on the destination bus Since packing unpacking of information can cause a discrepancy between number of source words read and destination words written, it is important to note that the word count specified to the HCI controls the number of words to be written into the destination machine The Borrow signal of the counter is used to generate the internal WORD COUNT ZERO signal, which triggers the DMA control module to terminate the process.

7.2 DMA Controller Module

The DMA Controller module arbitrates the requests by different GRADS resources for DMA transactions, and controls the DMA process Any of the computing resources in GRADS can request a DMA transaction for block data moves This request is initiated by setting the DMA REQUEST bit in their respective sections of the control register module All the request bits are brought into the DMA Controller module and latched in 74LS279 S R latches (Figure 29)
The output of these latches are fed into a set of 74LS278 priority encoders The highest priority in GRADS is given to the VAX11 host computer, followed by the S100 Z-80 host computer, and the 8 satellite processing units The output of the priority encoders is a single DMA GRANT signal to the highest priority requesting device The DMA GRANT signal also locks the priority encoders so that request arbitration is frozen for the duration of the DMA process. The DMA GRANT signal to the appropriate processor allows it to access the core DMA Control registers in the control register module and issue the START DMA command by setting the appropriate bit in the DMA control registers Once the START DMA command has been issued, the DMA Controller module drops the DMA GRANT signal to the requesting CPU. This essentially denies access to the DMA control registers, avoiding the possibility of any CPU possibly modifying the contents while a DMA transaction is in progress

At this point the DMA transaction is in progress, and remains so till one of the three conditions signal the DMA Controller to terminate it. Under normal conditions the termination is brought about by the WORD COUNT ZERO signal, which indicates that the requisite number of data words have been transferred. Two abnormal conditions can also cause the DMA process to be aborted, i e the READ TIMEOUT and the WRITE TIMEOUT These conditions are caused when the source bus or the destination bus does not respond to a bus request within 1 mS. This timing is non critical, and is maintained by a pair of monostable multivibrators (74LS123). The presence of either termination signal results in a single pulse being generated by the pair of 74LS74 D flip flops working as a pulse generator. This pulse is ANDed with the priority en-

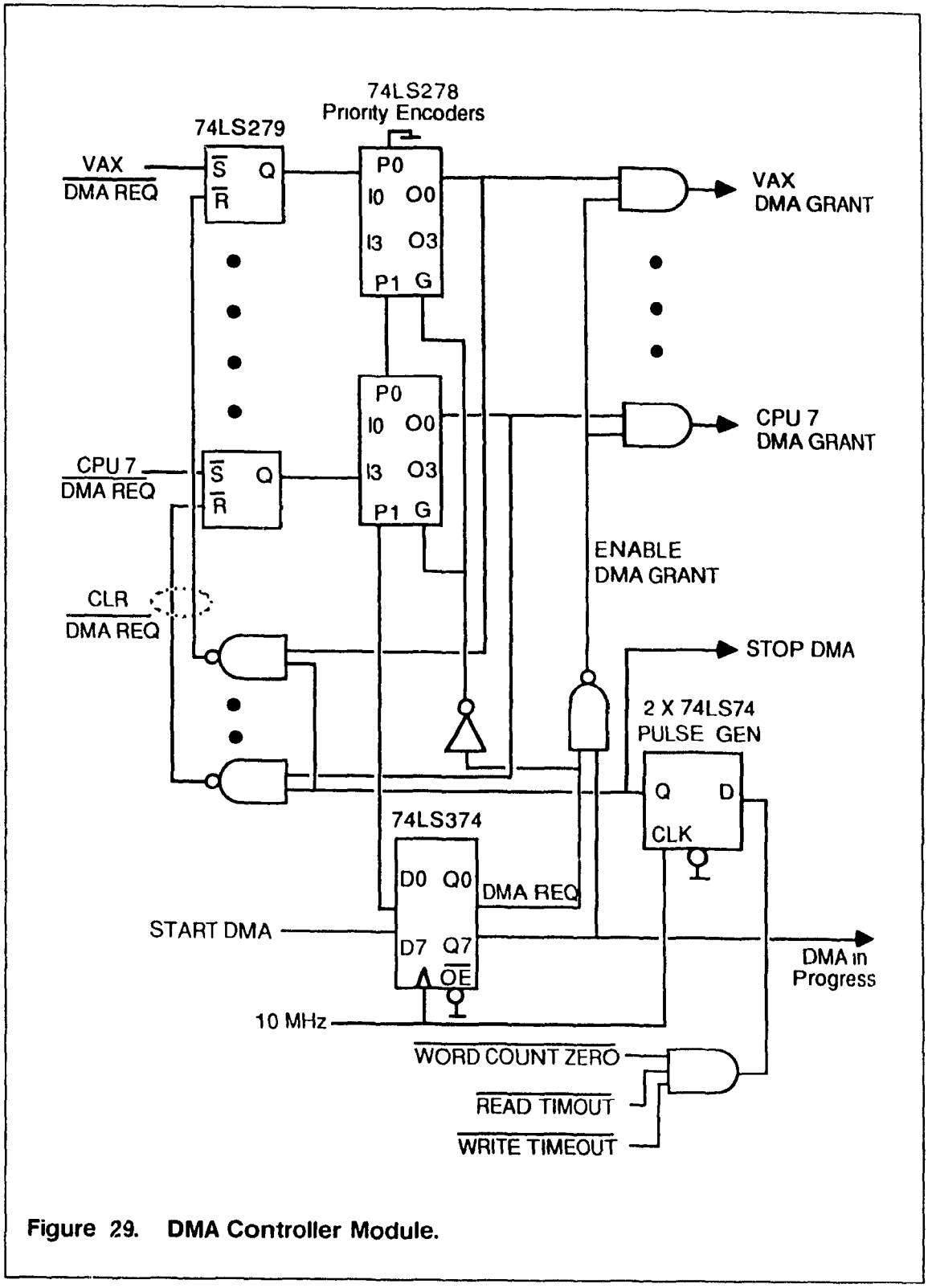


Figure 29. DMA Controller Module.

coder's outputs to clear the appropriate DMA request. All other non-serviced DMA requests remain latched, and are prioritized for the next DMA cycle.

7.3 Data Path Module

Of all the modules of the host computer interface, the Data Path module is the most complex. It controls the flow of data blocks between the source processor bus and the destination processor bus, and mediates in the flow to pack and unpack the data as requested.

The data path module initiates a series of block read requests and write requests to the source and destination processors (via their respective interface modules) by providing a READ WRITE REQUEST and the source destination address. This read or write process continues till the data path module issues a FINISH signal indicating that it's buffer capacity has been reached, or that the DMA session is over (Figure 30).

The Data Path module works under the control of the DMA Controller module, and is made up of four distinct subsystems:

- Data Input Buffer
- Data Output Buffer
- Word Packing Unpacking Logic
- Word Packing Unpacking Control

Figure 31 shows the typical flow of information through these subsystems The data is read in blocks from the source computer, each block being buffered in the Data Input Buffer. The Data Packing Unpacking logic performs the necessary word width adjustments, and the block of data to be transferred to the destination computer is buffered in the Data Output Buffer.

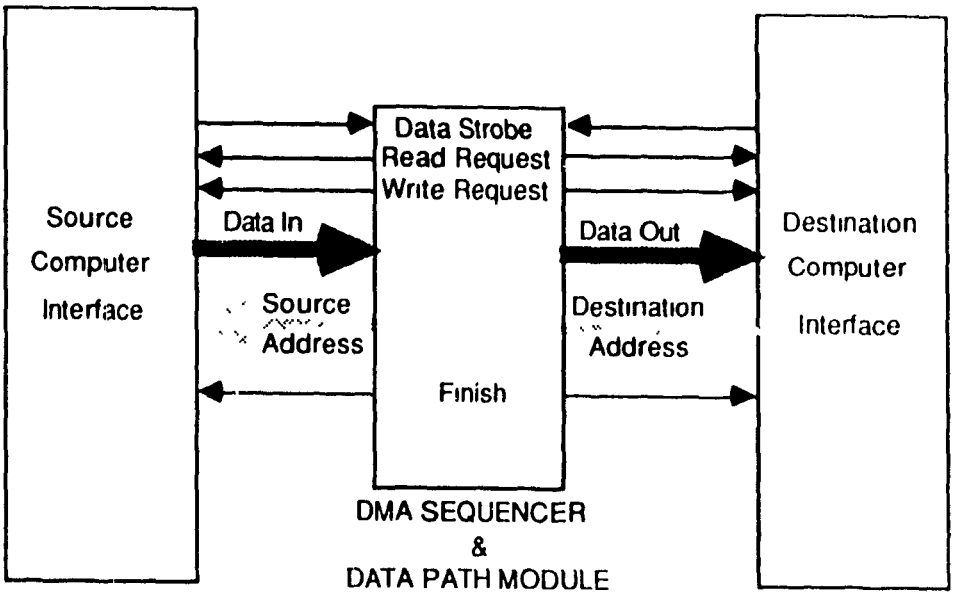


Figure 30. Data Path Module Signals & Controls.

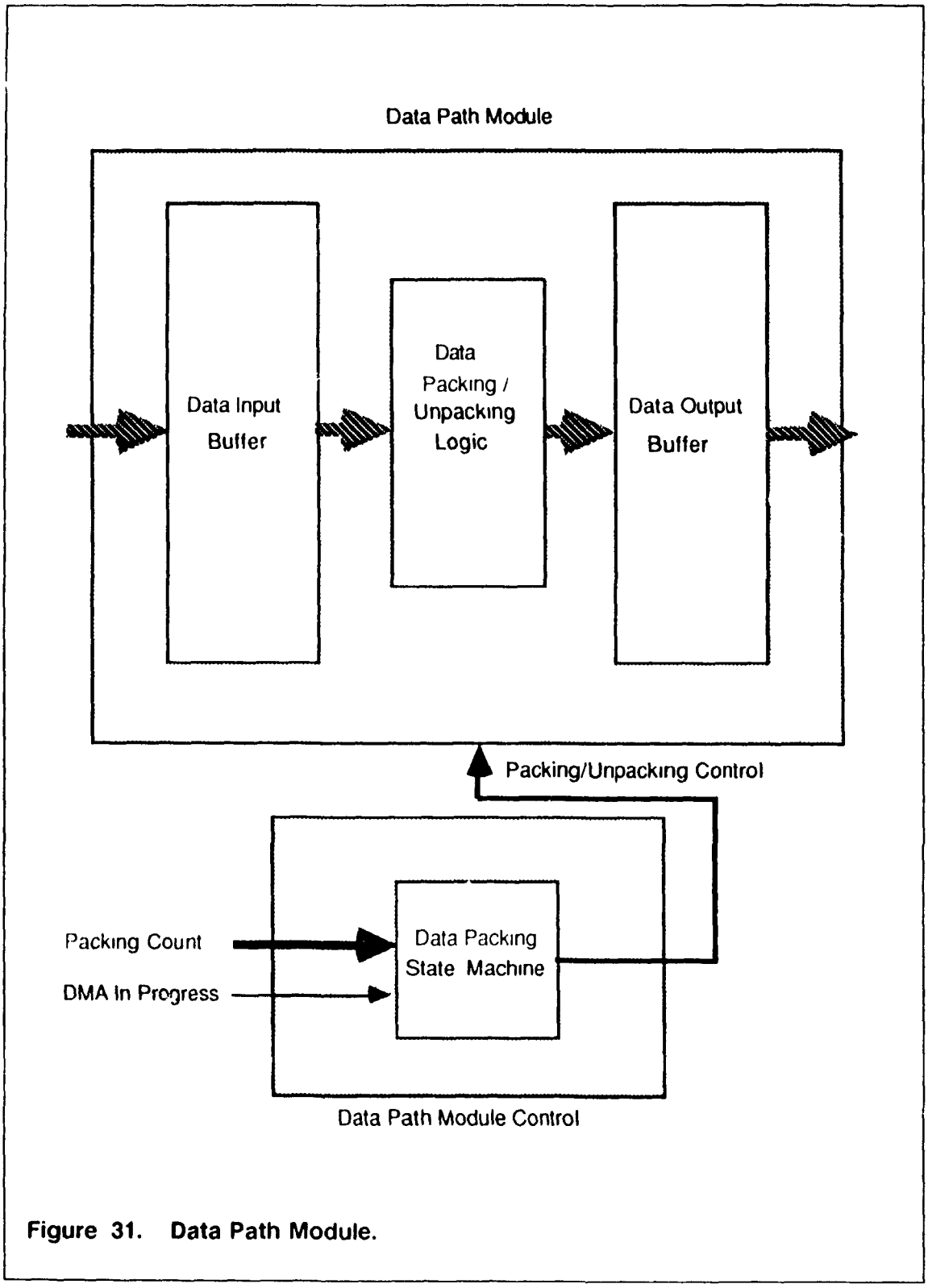


Figure 31. Data Path Module.

The Data Input Buffer is made up of 16 deep FIFO which can accomodate a 40 bit word To achieve the 40 bit word width, 8 74S225 FIFOs were used (Figure 32) If the DMA IN PROGRESS signal is high, and the input buffer is completely empty, a read request is generated and sent to the source computer interface This causes the source computer bus to be aquired by the HCI and a total of 16 words being read into the input buffer At the end of the 16th read, the input ready flag drops on the FIFO, which causes the FINISH (L) signal to be generated, resulting the source computer bus to be relinquished The Output Ready signal needs to be de-asserted before a new read request is generated This forces the FIFO to be completely emptied before initiating a fresh DMA read cycle to fill it up from the source memory The decision to read 16 words in one burst was based on a desire to optimize the data transfer process for large blocks On the other side of the input buffer, the data is extracted by the packing/unpacking logic for word width manipulation As the 16 words are clocked out of the output end of the FIFO, another read of 16 words is initiated This process continues till the DMA process is aborted by the DMA Controller module

The Packing Unpacking logic consists of a set of five octal buffers (74LS244s), each corresponding to one octet of the 40 bit input word, and a set of five octal data latches (74LS374s), each corresponding to one octet of the 40 bit output word. Linking the input word buffers and the output word latches is an 8 bit wide data bus The details of this section are shown in Figure 33. This arrangement allows any octet of the input word to be latched into any octet of the ouput word. The exact sequence is regulated by the packing unpacking state machine in accordance with the packing/unpacking code specified for a particular DMA transaction For example, when reading from a 16 bit processor and writing to an 8 bit processor, both the upper and the lower octets of the input word are mapped onto and latched into the lowest octet of the output word.

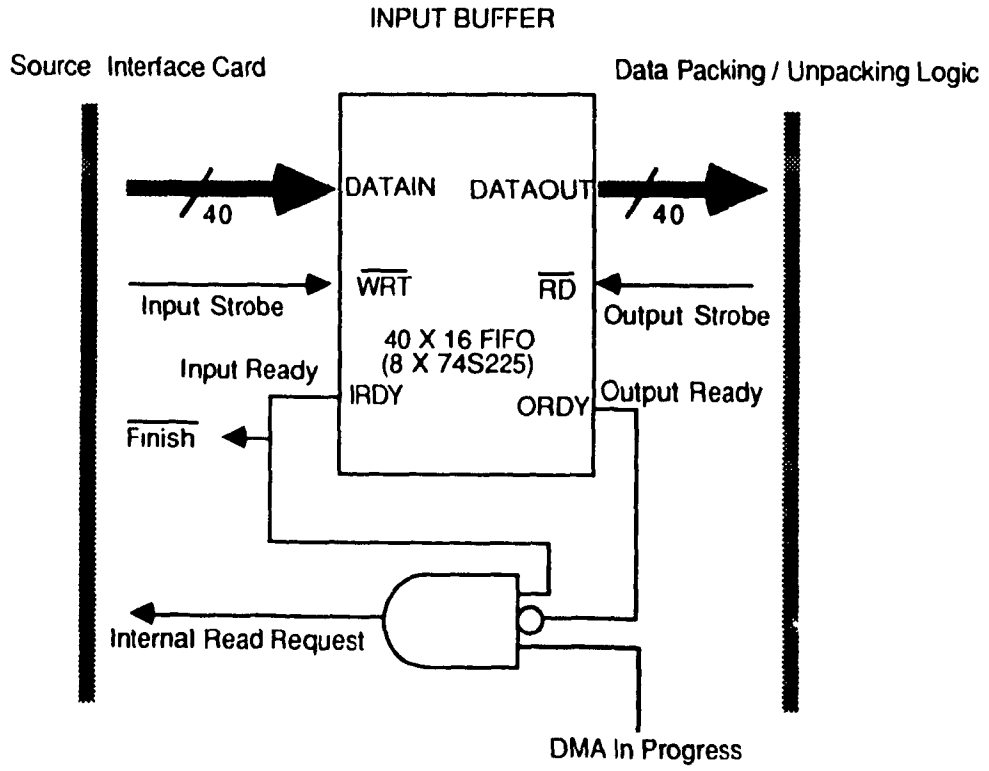


Figure 32. Data Path Module - Input Buffer.

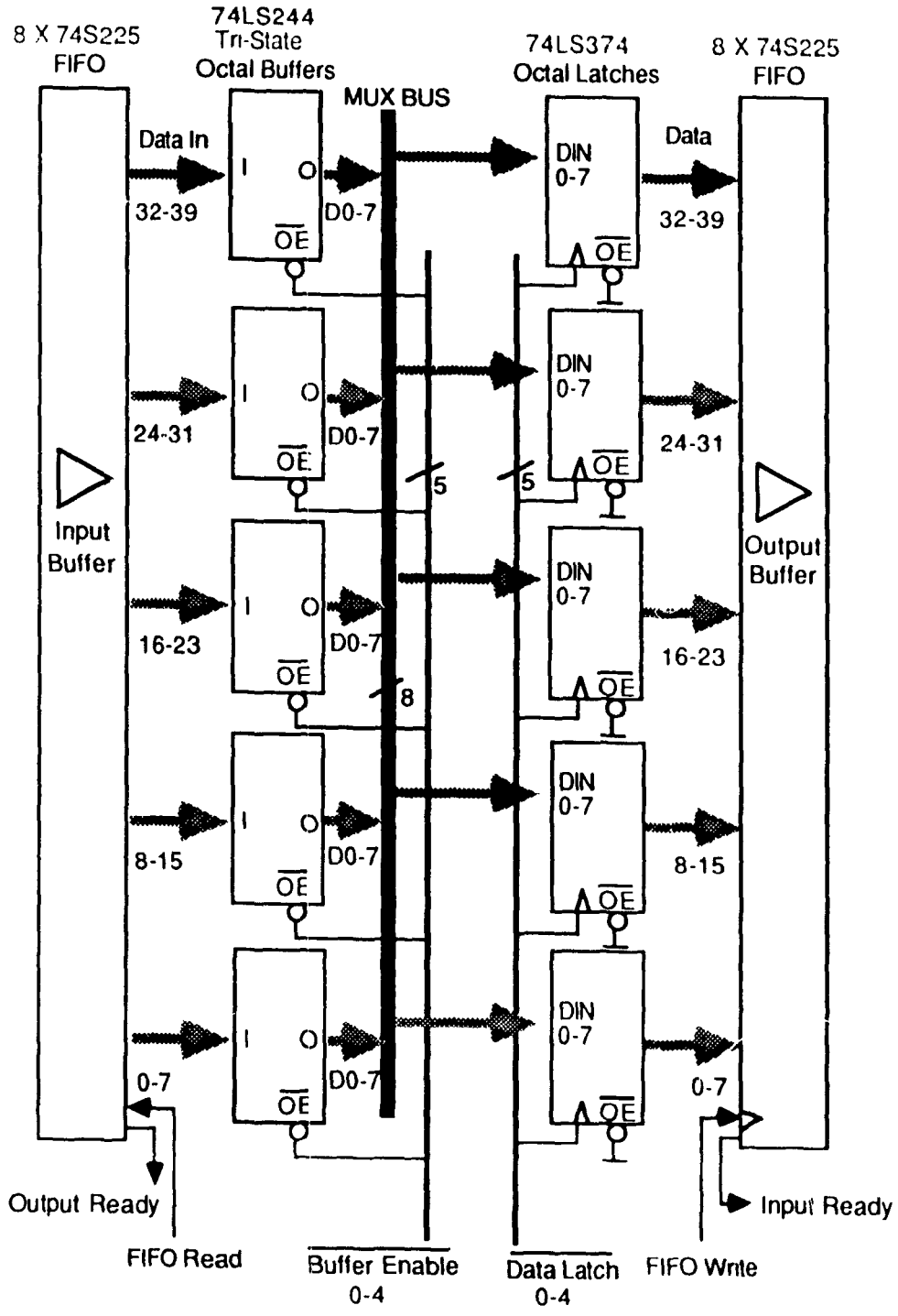


Figure 33. Data Packing/Unpacking Logic.

The packing/unpacking logic is controlled by a ROM based state machine (Figure 34). This ROM sequencer generates a predetermined sequence of signals to read from the input buffers, control the packing unpacking logic, and write into the output buffers It takes the 4 bit packing unpacking word as it's main input (Figure 35) Since the input and output buffers are clocked asynchronously by the source and destination computers, the packing unpacking state machine starts and stops are also controlled by the availability of data in the input buffer and space in the output buffer The machine resumes from its last state when these conditions are satisfied

The Data Output Buffer works in a fashion similar to the Data Input Buffer except that it initiates a write request to the destination computer when there is a block of 16 words ready to be written This is ensured by generating the write request signal when the Input Ready signal of the FIFO is de-asserted. The bus of the destination processor is released when the FIFO is empty and the FINISH (L) signal is generated. The details of the Data Output Buffer are shown in Figure 36.

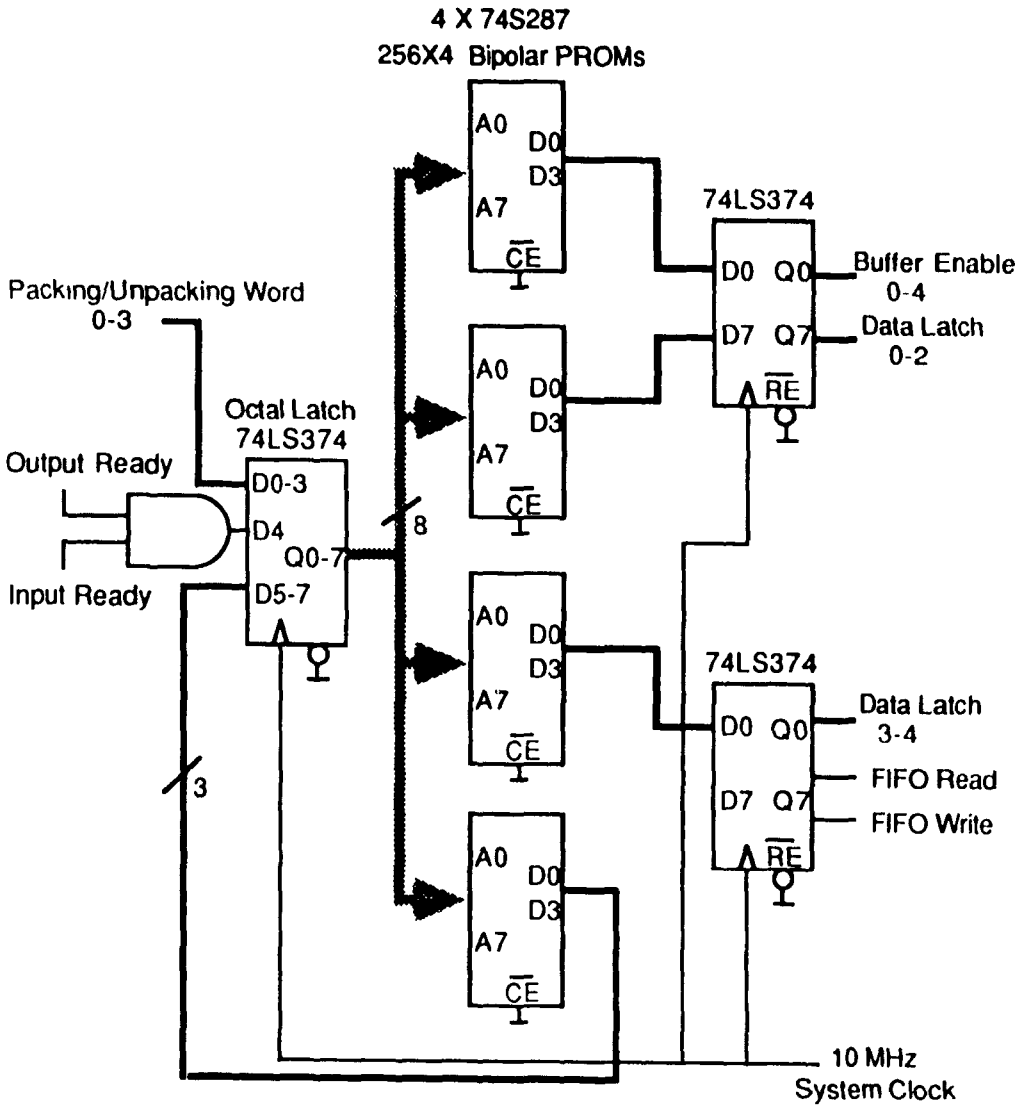


Figure 34. Data Packing Unpacking State Machine.

CODE	SOURCE WORD WIDTH	DESTINATION WORD WIDTH
0000	8	8
0001	8	16
0010	8	32
0011	8	40
0100	16	8
0101	16	16
0110	16	32
0111	16	40
1000	32	8
1001	32	16
1010	32	32
1011	40	8
1100	40	16
1101	40	40

Figure 35. Data Packing Unpacking Table.

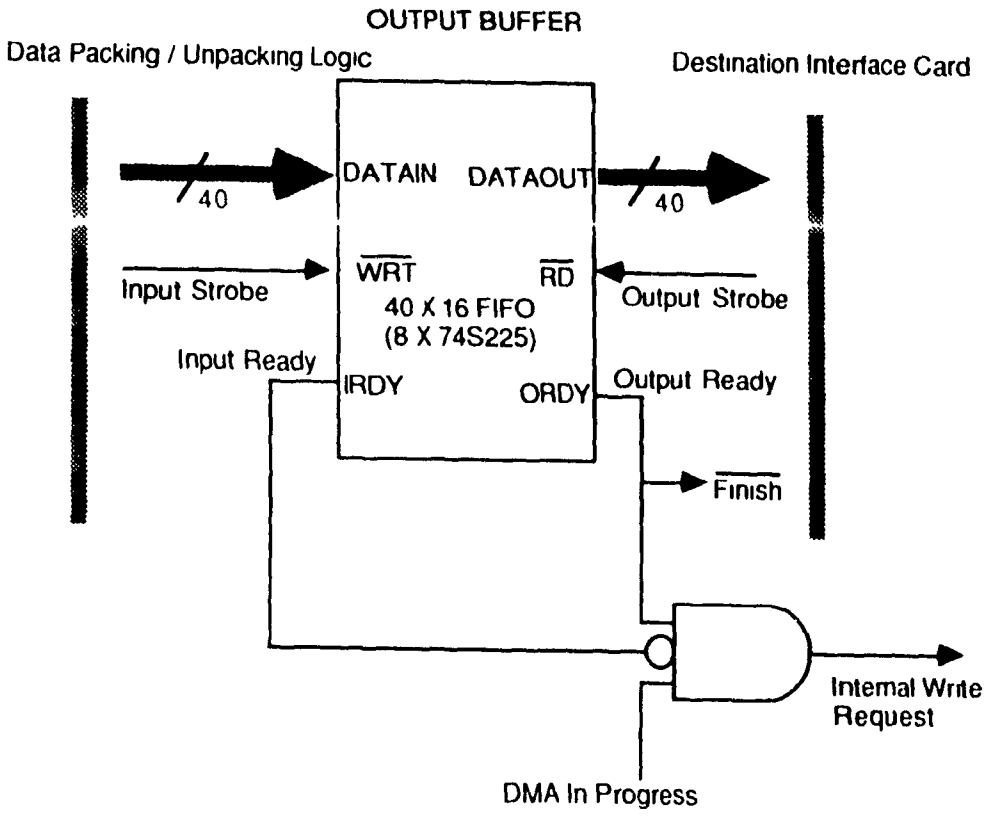


Figure 36. Data Output Buffer.

8.0 TESTING AND DEBUGGING

The testing and debugging of a system as large and as complex as the the HCI can be a challenging task indeed, the complexity of the HCI stemming from the number of signals in the system, the asynchronous nature of all external bus signals, the sheer size of the implementation (four large cards with 100 ICs each), and the distributed architecture of the HCI. To make such a task more amenable, it was decided to spend a fair amount of time on the development of a robust testing environment, which consisted of a testing and debugging strategy, and the appropriate test tools The following sections of this chapter describe the strategy, the tools and the actual process in greater detail.

8.1 The Testing Strategy

The foremost consideration in the development of a suitable test plan for the HCI was the size and complexity of the system The focus of the test plan, therefore, was to develop a methodology for the incremental testing of the system. To this end, the modular and distributed architecture of the HCI lent itself very well In addition, an attempt was made to reduce the number of variables at any given point in the testing phase so as to keep the number of monitor points low A four step process was devised for each module.

1 Verification of implementation accuracy.

2. Verification of idle state characteristics

3 Verification of functionality at low speed

4 Verification of functionality at full speed

The first involved checking the system for wiring errors. This was done by "buzzing" off each connection against a wiring list and the circuit schematic Once all corrections had been made,

the card was powered up using a current limited bench power supply The current consumed by the card, and the presence of "hot spots" on the card were used as indicators of the presence of serious errors (typically short circuits between chip outputs and ground, and signal collisions between competing devices). Such problems, if present, were resolved before further testing and debugging.

One of the design decisions was to implement all inter module signals as active low, and pull them up to their idle or inactive states on the receiving end. This feature allowed the cards to be powered up individually and checked for their idle states This test proved to be one of the most effective steps in identifying both logic and microcode errors without system integration

In the third phase, the module was exercised through tools that simulated the signals between the module under test and the module to be interfaced with The simulation tools were mostly software packages running on a Z-80 system and simulating the signalling sequence through parallel I O ports (Figure 37) As most of the protocols in the HCI are based on a handshake type sequence, turning down the basic clock of the module under test allowed the process to be viewed at slower speeds thereby reducing the contribution of timing and speed related conditions, and allowing logical errors to be isolated with greater ease

In the last phase, the clock was turned up to the full speed, mostly 10 MHz, and the functionality checked again A failure between steps three and four effectively pointed to some specific areas of the circuit Typical problems encountered in this phase were

1. Race conditions and glitches due to timing skews in coding decoding logic circuits
2. Signal collisions due to turn-on and turn-off times of drivers
3. Unreliable data transfer due to memory access time limits
4. Signal distortion on long and unterminated lines, due to noise pickup and reflections

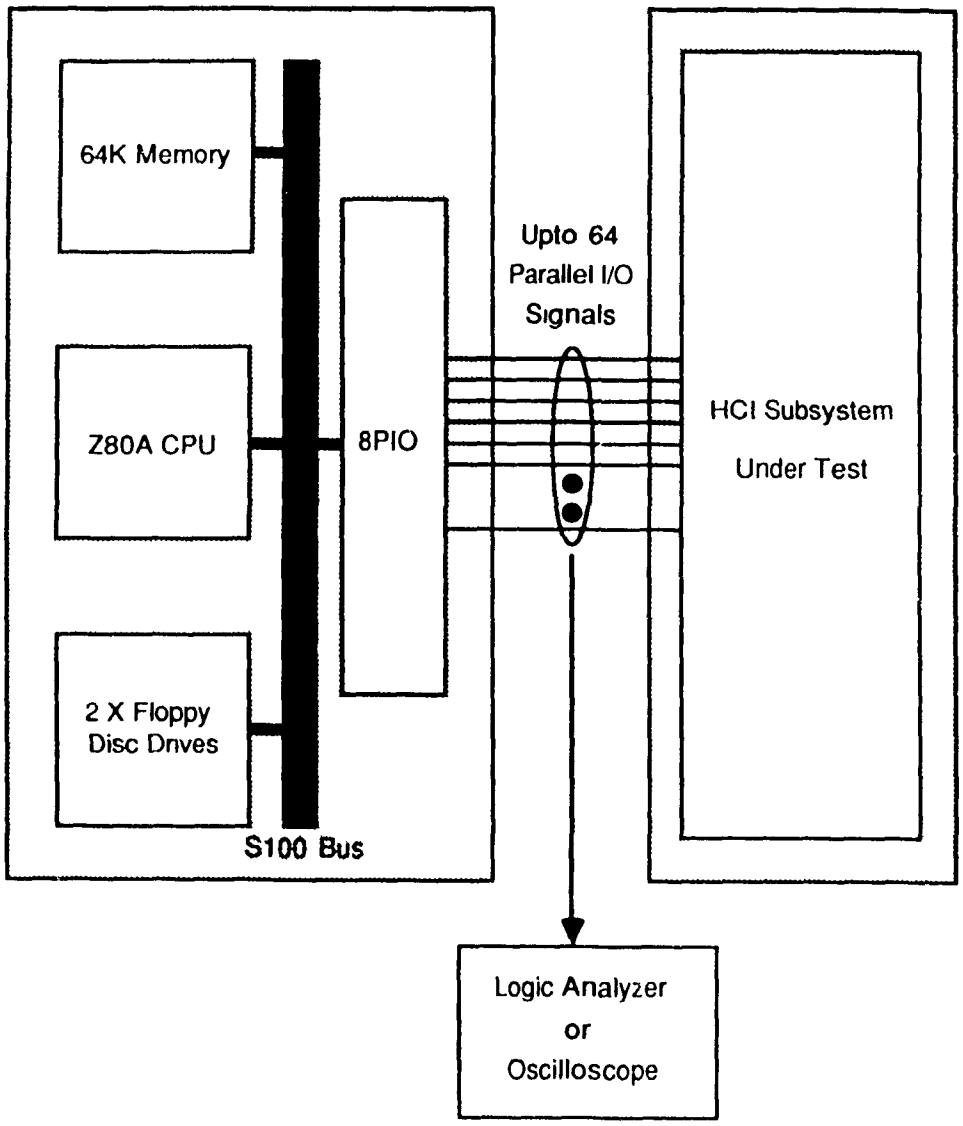


Figure 37. Signal Sequence Simulation with the Z-80 and 8PIO Cards.

Overall this process proved to be extremely succesfull in detection of a large fraction of bugs, errors and design oversights before system integration was attempted.

8.2 Test Tools

In addition to the available instruments, several test jigs and tools were developed to debug the system. The most widely used test tool was the Cromemco Z-80 microcomputer system with an 8PIO card (providing 64 parallel latched input output lines). This system, with the appropriate software load, could emulate any of the interfaces in the GRADS system. Some software modules were also developed to synthesize specific signal sequences, effectively emulating a programmable signal generator Due to the large numbers of microcode driven state machines in the HCI, and the GRAD system in general, a microcode editor and downloader was also developed on the Z-80. This tool microcode editor allows the microcode in the system to be edited, filed and then downloaded into the target machine in an interactive fashion This capability proved to be extremely effective in reducing the microcode debugging and fixing time Most modules were were debugged by repeating the entire process cycle on a continuous basis This enabled the progress to be monitored to a specific point using oscilloscopes and analyzers To prevent the systems from "hanging up" and causing the iterative process to stop, an external watchdog timer/reset generator was developed. The system was used to ensure the continuous looping of the test process Figure 38.

In addition to these tools, two existing laboratory test instruments were extensively used. the Tektronix 465B dual trace 100 MHz oscilloscope, and the Tektronix 8 bit 50 nano second resolution logic analyzer.

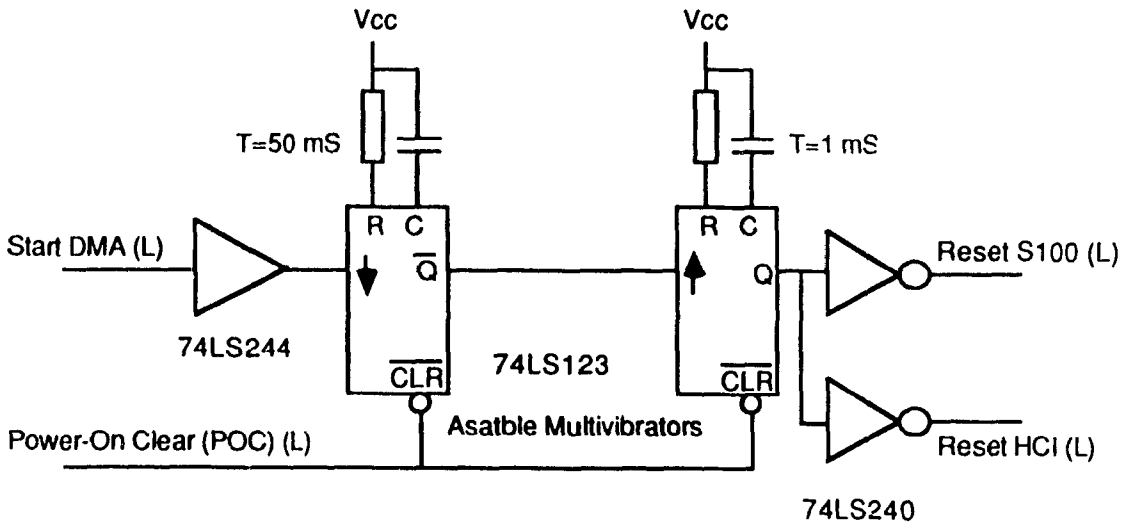


Figure 38. Watchdog Timer Arrangement for HCI Testing.

8.3 Test Process

One of the first submodules of the HCI to be debugged was the S100 interface, leading to full integration testing of the HCI by achieving full DMA transfers with the S100 host computer system.

8.4 S100 Card Testing

The S100 card was tested out in two main steps In the first step, the card was plugged into the system bus and all bugs were traced and corrected till the card resided on the bus, in its idle state, without interfering with the normal operation of the Z-80 computer system Once this was achieved, the resident monitor (RDOS) on the Z-80 host computer was used to load and read local registers on the S100 card Since these registers are used to load microcode in the RAMs of the state machine on the S100 card, a special program was developed to exercise the loading mechanism. The program moves a block of microcode from the memory of the host Z-80, and loads the microcode in the S100 card It then reads it back to compare and verify the integrity of the moved data Once the correct move has been verified, the program loads a register which enables the S100 card to enable it's drivers under microcode control and interact with the bus This is the true test of correct operation of the system while in the idle state Figure 39 shows the microcode enable circuits

The actual DMA sequence of the S100 card was debugged using an external stimulus to emulate the HCI READ REQUEST signal This signal, if correctly interpreted by the microcode on the S100 card, would cause the S100 card to request the S100 bus, obtain the grant, and begin an indefinite block read sequence from an arbitrarily chosen location in the host computer memory When a FINISH signal is sent to the S100 card, the microcode should release the S100 bus in an orderly fashion and restore the normal CPU operation This sequence is de-

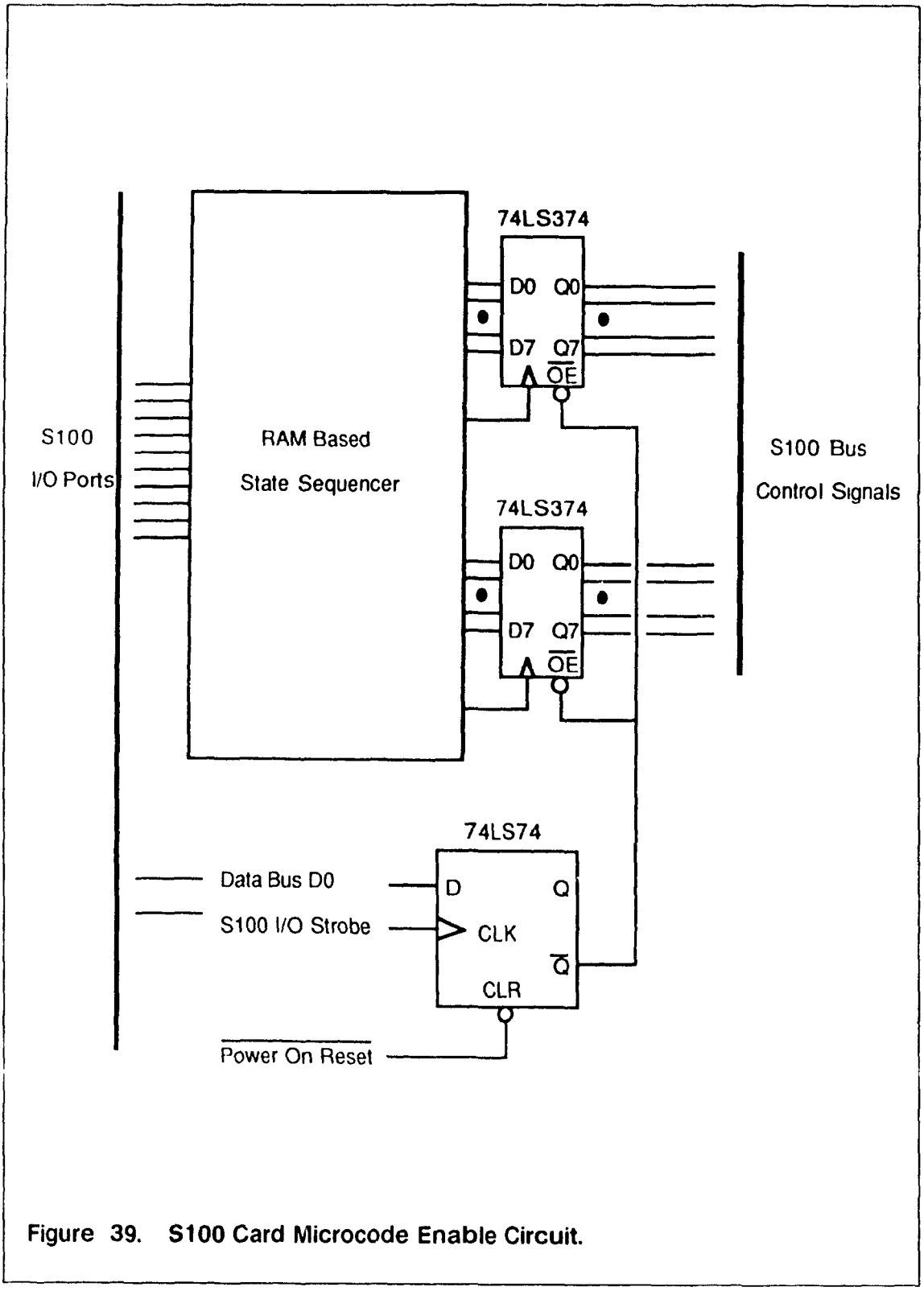


Figure 39. S100 Card Microcode Enable Circuit.

picted in Figure 40, where the shaded areas highlight the bus aquisition and bus release portions of the DMA cycle The crucial things to check in this cycle are

1. The correct sequence of bus request steps
2 The correct bus release procedure
3 The non-disruptive nature of the temporary bus mastership

The correct sequence of bus request operations was studied on an oscilloscope by generating repeated request and finish signals and effectively putting the process in a "loop" The same approach was used for the bus termination sequence The non disruptive nature of the interaction was ensured by running the RDOS monitor program on the host computer CPU during the DMA attempts When the DMA was non disruptive, the monitor ran without problems, albeit slowly due to the limited fraction of time available to the CPU A typical test was to perform a large memory dump on the terminal screen using RDOS while trying out the DMA sequence During the block read sequence, the other signals were also verified using the oscilloscope A bus state indicator card constructed for quick visual confirmation of the bus state was tremendously useful during this process The indicator card simply picked off the standard bus signals, such as HOLD REQUEST, HOLD GRANT and HALT, off the backplane and used coloured LEDs to display them on the front edge During the the debugging stages, when the system got locked in indeterminate states, these visual indicators provided a quick status update without having to laboriously check all the signals on the S100 bus

Once the S100 card had been debugged independently, it was interfaced with the central control register module and the DMA controller module After idle state sanity checks, debugging of the integrated system was initiated using an interactive software package on the Z 80 host computer. The software package allowed the DMA parameters to be defined interactively so that the HCI's services could then be requested by the host computer and the DMA block move initiated The user menu of the software package is shown in Figure 41

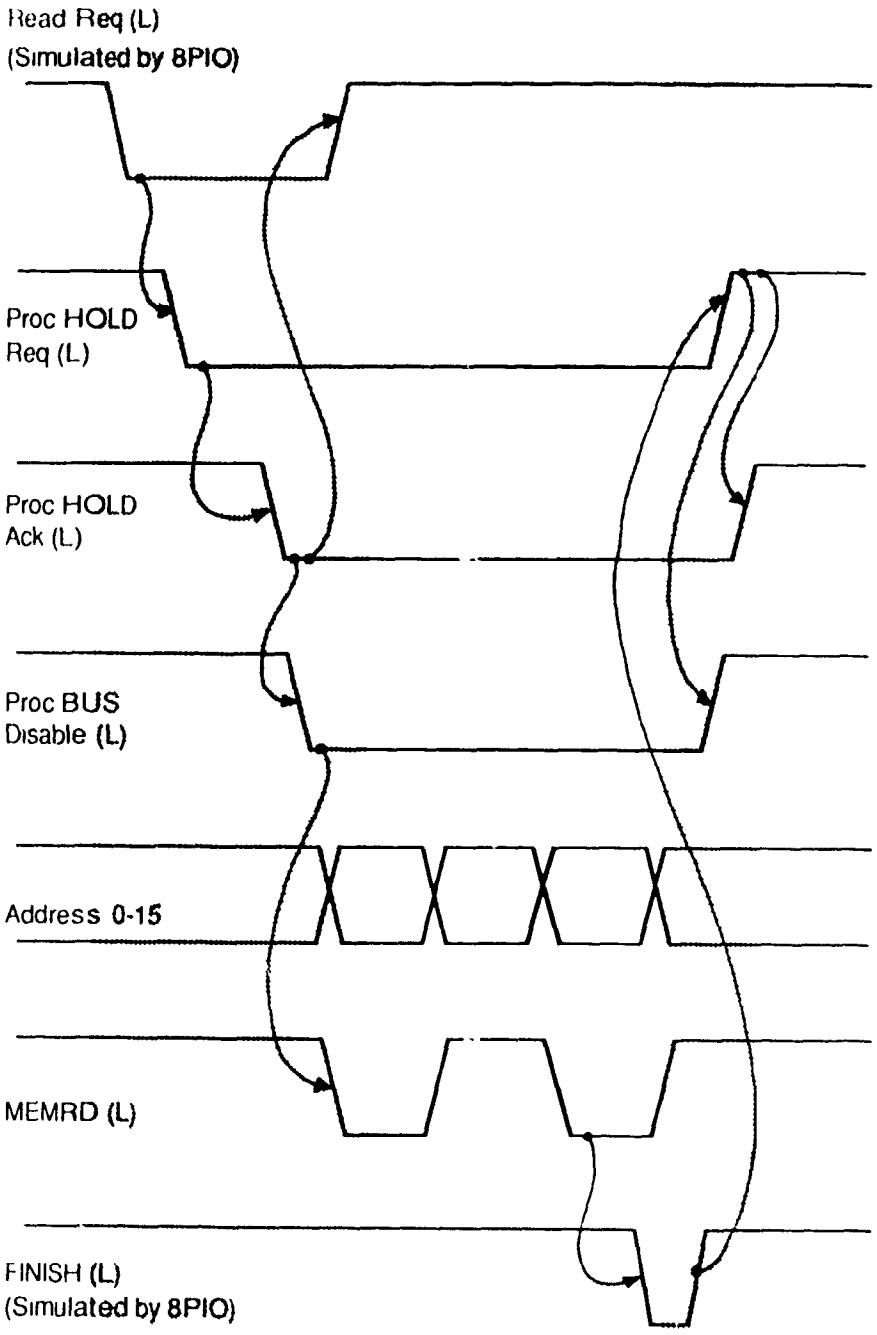


Figure 40. S100 Card Debugging Sequence.

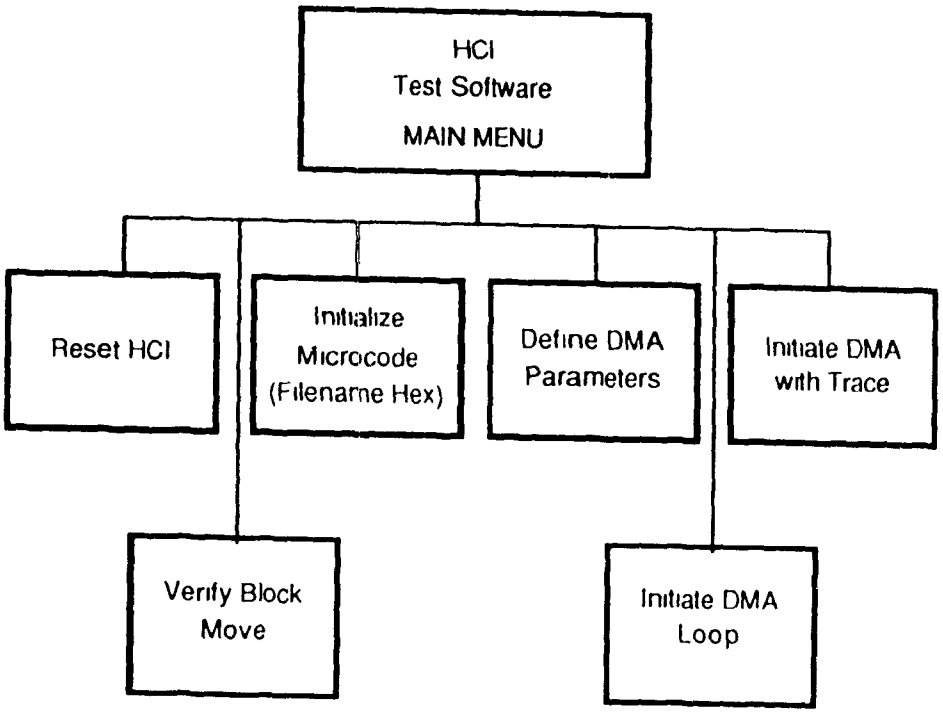


Figure 41. HCI Test Software Menu.

As in the earlier case, the effective debugging of the system was possible only by "looping" the entire process The software was designed to load the registers of the HCI, start DMA, and wait for about 10 milliseconds before restarting the process In many instances, the DMA machine 'hung-up", thereby jamming the host computer bus and preventing the program from repeating the process For this purpose, an external watchdog timer was used to reset the HCI after a certain grace period This external timer was based on re-triggerable monostable devices and was triggered via an ouput port by the software routine driving the HCI This step reset and armed the watchdog timer The algorithm of the test software package is shown in Figure 42.

A functional overview of the test setup is depicted in Figure 43, and the actual laboratory system shown in Figure 44

Once the entire DMA process had been debugged using the S100 bus as one active computer, it was extended tc other busses incrementally At the time of completion of the project, only the UNIBUS, and the CPU bus interfaces were actually excercised due to hardware availability, the rest of the modules being designed The remaining interfaces were, however, simulated using the 8PIO cards as previously mentioned, and the correct transfer of information between different busses verified

8.5 Test Results.

As the testing progressed, apart from basic logic and wiring errors, two main classes of problems became apparent The first was traceable to faulty states created by defects or deficiencies of the microcode in the various state machines. Once the faulty states had been identified, it was a question of downloading new microcode into the system and verifying the correct operation Extensive use was made of an interactive microcode editor and download utility which were specially developed for this project [27].

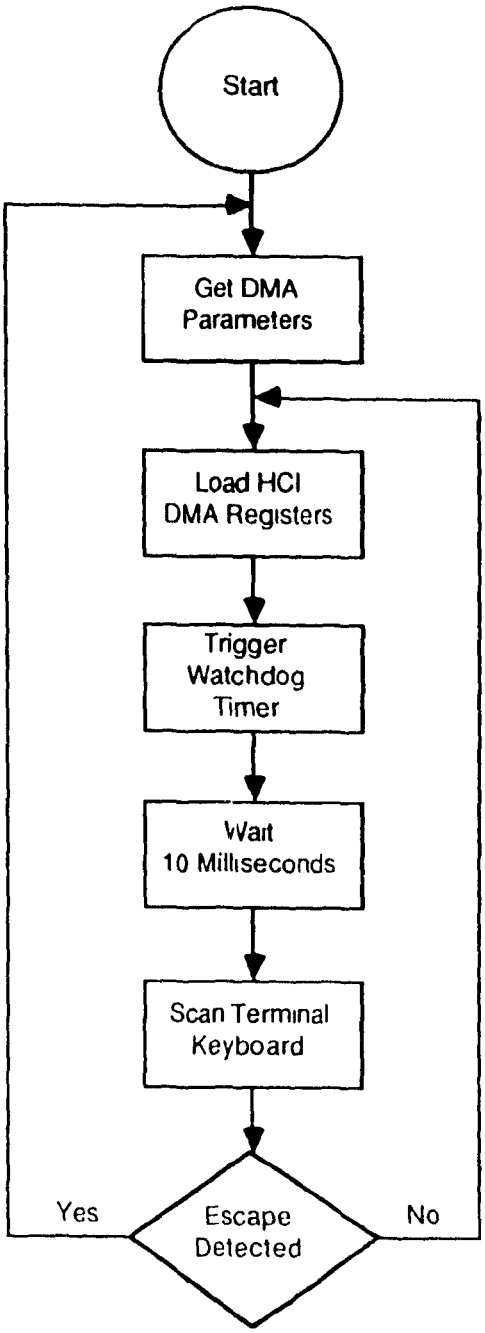


Figure 42. Test Software Flow Chart.

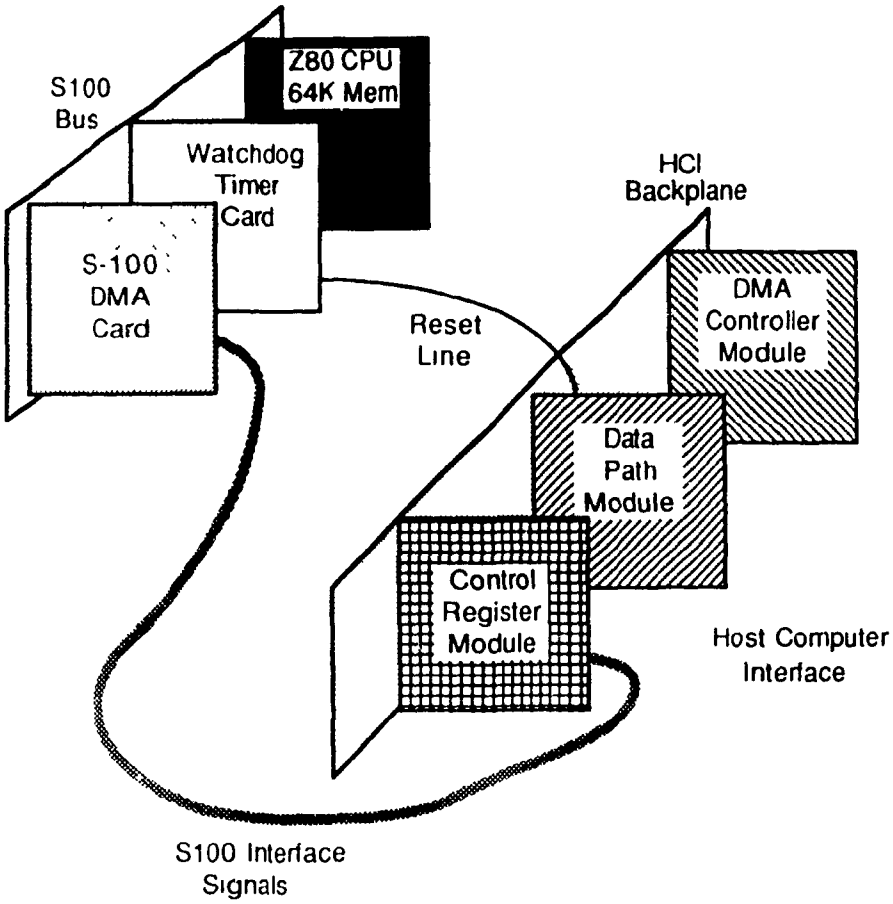


Figure 43. Functional Setup for HCI Testing.

Figure 44. Laboratory Setup for HCI Testing.

The other class of problems were related to timing on asynchronous busses supported by the HCI Once again, the microcode driven sequencers used in almost every part of the HCI proved to be of tremendous value in terms of being able to fix most timing problems via microcode updates rather than hardware patches

Although all the modules of GRADS were not completely integrated by the time this research was terminated, the HCI was tested out individually against all of the identified GRADS modules It was also observed that the debugging and verification effort of the HCI equaled all definition, design, and implementation effort of the GRADS system, re-inforcing the need for design-for-testability in large systems

8.6 Performance Evaluation

The total data transfer rate of the HCI is limited to a 40 bit word every 300 nS, or a bandwidth in excess of 120 Mb S This figure represents the instantaneous burst transfer speed without other overheads and, while it is significantly more than the 10 Mb S maximum transfer rate of Ethernet type LANs, it can never be realized in practice due to other factors. To evaluate the HCI's DMA performance, typical figures for data transfers from and to an S100 bus system are presented

One of the obvious overheads in DMA transactions is the time taken by a DMA machine to gain a master status on the bus. In the case of S100 bus machines using the Z-80 CPU, bus requests are granted by the CPU after the completion of the instruction in progress For a Z-80 processor the longest instruction can be 4 machine cycles If running at a clock frequency of 4 MHz (Z-80A) each machine cycle is 250 nS, and therefore it can take upto 1 micro second to complete the instruction prior to granting control of the bus. It takes an additional cycle to re-clock the HOLD ACKNOWLEDGE signal from the bus, leading to a maximum bus aquisition time (Tba) of 1 25 micro seconds All read write cycles on the S100 bus are two machine cycles

long, requiring 500 nS to complete each memory cycle (Tmc). Since the HCI always reads in packets of 16 words, the time to read one packet is

Tba + (16 X Tmc) = 9.25 micro seconds

The HCI reaches its maximum DMA efficiency when the data block to be transferred is an integral multiple of 16 source words For example, a block of 4 Kwords (4096 words) requires a transfer of 256 packets of 16 words each, requiring a total transfer time of 2 368 milli seconds This represents an overall transfer rate of roughly 1 7 Mwords per second or 13 8 Mb S To quantify the efficiency of the system, this figure needs to be compared with the 16 Mb S maximum bandwidth of the Z-80's memory

When block sizes are not integral multiples of 16 source words, an over-fetching of the input buffer can occur In the worst case, the data block can be 1 word more than an integral multiple of 16, causing an over-read of 15 source words to take place this adds a total of 7 5 micro seconds of over-read time (Tor) to the normal process For a block of 4097 words, the time taken is represented by.

[Tba + (16 X Tmc)] X 257 = 2 38 mS

This is equivalent to a transfer rate of 13 7 Mb S. Comparing the two figures, it is evident that the penalty of over-reads due to input data buffering is insignificant when the HCI is used to transfer large blocks of more than, say, 1K words.

If the system were to be optimized for smaller blocks of data by reducing the input buffer size to 1 word, the time to transfer a block of 4096 words would be.

(Tba + Tmc) X 4096 = 7.2 mS

This is equivalent to a data transfer rate of 4 6 Mb S. The throughput gain due to input buffering is clearly evident from these figures

Even in it's most elemental application, utilizing only 8 of the possible 40 data bits, the HCI's DMA performance exceeds that of CSMA CD type LANs (Ethernet). As tested, the performance of the HCI was in accordance with the design goals, and in fulfillment of GRADs' data transfer requirements

8.7 Tool Evolution

Some of the debugging methods described in this chapter are indicative of the level of commonly available tools at the time of the research, most of the debugging having been done with a simple dual-trace oscilloscope and an 8-channel logic analyzer. Use of the current state-of-the-art tools such as in circuit emulators, and multi channel logic analysers would have drastically changed the way the HCI was tested, just as large software based logic simulators would have helped in the identification of numerous timing problems prior to implementation.

With the advent of the IBM PC, such software tools are now commonly available at affordable prices, resulting in great improvements in efficiency of design and testing. Similarly, escalation in the complexity designs is addressed by the dramatic electronic filing capacities of hard disks and the advent of high resolution colour displays of modern engineering workstations.

One cannot overlook the tremendous impact of custom VLSI (ASICS) in modern digital design. The availability of CAD software packages such as CADENCE allow for rapid design and simulation of custom circuits which can then be manufactured at silicon founderies. Together with the breakthroughs in packaging and miniaturization, the domain of digital design remains exiting and challenging

8.8 Future Work

The most obvious extension of this work would involve a re-implementation of the system exploiting currently available technology such as the TMS340 processor, multi-port video RAMs, and video peripherals Much of the discrete logic and state machines could be built using Programmable Gate Arrays (PGAs) and custom VLSI

Although some single card systems with impressive performance are commercially available, complex real-time animation still requires an approach involving multiple concurrent processors

An even greater challenge, perhaps, would be to develop a user-friendly interactive software environment for the preparation and execution of large graphics animation models.

9.0 CONCLUSION

In this thesis the design of a colour graphics system for real-time animation was presented.

After reviewing current literature on computer graphics and computer architectures, the system requirements for real-time animation were discussed and a specific architectural solution called GRADS was proposed, with a focus on the Host Computer Interface.

The design requirements, system implementation, and test results of the Host Computer Interface were detailed, and the performance of the system discussed.

REFERENCES

1. Alexander P., "Array Processors in Medical Imaging", IEEE Computer, June 1983, pp. 17-30.

2 Ampudia R , "A Microprocessor Design Using the Intel 8086 for a Colour Graphics Animation System", M.Eng. Thesis, McGill University, 1982.

3. Bootle S K , "The History of the Digital Computer", McGraw-Hill Book Company, 1981

4 Carayannis G., "A Colour Display System for Real-Time Animation", M Eng. Thesis, McGill University, 1981.

5 Chau D.W Y., "Microcoding the AMD2900 Bit-Slice Microprocessor of the Graphics Real-Time Animation Display System", M.Eng. Thesis, McGill University, 1984.

6. Clark J.H , Davis T , "Workstation Unites Real-Time Graphics with Unix and Ethernet", Electronics, October 1983, pp 113-119.

7 Crow F.C., "Shaded Computer Graphics in the Entertainment Industry", IEEE Computer, March 1978, pp 11-22.

8 DeFanti T A , Brown M.D , McCormick B.M., "Visualization: Expanding Scientific and Engineering Reasearch Opportunities", IEEE Computer, August 1989, pp. 12-22.

9 Dixon D F , "Life Before the Chips: Simulating Digital Video Interactive Technology", Communications of the ACM, volume 32, July 1989, pp 824-831.

10 Feldman L., "High Definition Tele-Vision", Radio Electronics, February 1989, pp. 35-43.

11. Foley J D., Van Dam A , Fundamentals of Computer Graphics, Addison Wesley, 1982.

12. Forsstrom K.S , "Array Processors in Real-Time Flight Simulation", IEEE Computer, June 1983, pp. 62-70

13. Fuch H., et al., "Pixel Planes 5: A Heterogenous Multiprocessor Graphics System Using Processor Enhanced Memories.", Computer Graphics, Vol. 23, No. 4, July 1989

14 Fuchs H., Levoy M., Pizer S.M., "Interactive Visualization of 3D Medical Data", IEEE Computer, August 1989, pp. 46-51.

15 Goldwasser S.M , "A Generalized Object Display Processor Architecture", Computer Architecture Symposium Proceedings, SIGARCH Vol.12(3), June 1984, pp 38-47.

16 Hibbard W , Santek D., "Visualizing Large Data Sets in the Earth Sciences", IEEE Computer, August 1989, pp. 53-66.

17 Hibbard W., "A Next Generation McIDAS Workstation", Proceedings of the Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology., American Meteorological Society, Boston, 1988, pp. 57-61.

18 Hopkins H , "Classical Glass: Automation in the Cockpit", Flight International, September 1988, pp 87-91

19. Ikedo T., "High Speed Techniques for a 3D Colour Graphics Terminal", IEEE CG&A, May 1984, pp. 46-58.

20. Levin D.N., et al., "Integrated 3D Display of MR, CT, PET Images of the Brain", Proc NCGA-89, National Computer Graphics Association, Fairfax VA, Vol.1, 1989, pp. 179-186.

21. Levnthal A , Porter T , "CHAP- A SIMD Graphics Processor", Computer Graphics, July 1984, pp 77-82.

22. Lewis G , "High Definition Television", Electronics & Wireless World", March 1988, pp 226-229

23. Machover C., "A Brief, Personal History of Computer Graphics", IEEE Computer, November 1978, pp. 38-45.

24. Machover C., Myer W., "Interactive Computer Graphics", IEEE Computer, October 1984, pp. 145-161

25. Morris B , Tolmie D., "High Speed Channel (HSC): Mechanical, Electrical, and Signalling Protocol Requirements", Draft American National Standard X3T9 3 88-023, August 1988.

26. Niimi H., Imai Y., Murakami M., Tomitu S , Hagiwara H , "A Parallel Processor System for Three Dimensional Colour Graphics", Computer Graphics, July 1985, pp 67-76.

27. Pancholy R., Sung K., "Software Tools for HCI Debugging", Internal Report, McGill University, 1981.

28. Papapetros A., "The Design of a Colour Display System for Real-Time Animation Using Microprocessors", M.Eng Thesis, McGill University, 1977.

29. Partovi A., "The Design of An Operating System for a Real-Time 3-D Colour Animation System", M.Eng Thesis, McGill University, 1981.

30. Patton P C., "Multiprocessors: Architecture and Applications", IEEE Computer, June 1985, pp. 29-40.

31. Piper T., Fournier A., "A Hardware Stochastic Interpolator for Raster Displays", Computer Graphics, July 1984, pp. 83-94

32. Robert R., "A Z8000 Microprocessor Design for the GRADS. Graphics Real-Time Animation Display System", M Eng. Thesis, McGill University, 1983

33. Rodgers D.P., "Improvements in Multiprocessor System Design", Computer Architecture Symposium Proceedings, SIGARCH Vol.13(3), June 1985, pp 225-231.

34. Schindler M., "Better Graphics Opens New Windows on CAE Stations", Electronic Design, January 1983, pp. 77-86.

35. Sutherland I.E., "SKETCHPAD - A Man Machine Graphical Communication System", AFIPS SJCC, Vol 23, 1963, pp. 329-346.

36. Swezey R.W., Davis G., "A Case Study of Human Factors Guideline in Computer Graphics", IEEE CG&A, November 1983, pp.21-30.

37 Taylor J.W.R., Jane's All the World's Aircraft, Jane's Publishing Company Limited, 1985.

38. "The All-Singing, All-Dancing Computer", The Economist, March 1990, pp. 65-66.

39. The TTL Data Book, Volume 2, number SDLD001, Texas Instruments Incorporated, 1985.

40. TMS34010 Graphics System Processor Data Sheet SPPS011, Texas Instruments Incorporated, 1986.

41. VME System Products, Motorola Semiconductor Technical Bulletin BR308, 1985.

42. X25 Protocol Controller, Motorola Semiconductor Technical Bulletin ADI-1218, 1986.

43. Zuckerman M M "Innovative Display Technologies. Why Have a Flat Panel When You Can Have a CRT?", IEEE CG&A, April 1984, pp. 9-15.