THE DESIGN AND IMPLEMENTATION OF A VIRTUAL REALITY SYSTEM

BY

JOHN V. BELMONTE

B.S., University of Illinois, 1991

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1994

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

JUNE 1994

WE HEREBY RECOMMEND THAT THE THESIS BY

JOHN V. BELMONTE

ENTITLED THE DESIGN AND IMPLEMENTATION OF A VIRTUAL REALITY SYSTEM

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

Director of Thesis Research

Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

O-517

ACKNOWLEDGMENTS

The virtual reality system described in this thesis was conceived, designed, and implemented in the educational environment of the Advanced Digital Systems Laboratory (ADSL). I believe that there is no other university environment where this project would have prospered as it has in ADSL. My deepest gratitude goes to Doctor Ricardo Uribe for creating ADSL and tirelessly supporting and encouraging all students who have walked through its doors.

I also thank the following people: Kevin Lee for his help in the design of the V-Node and for handing down to me his knowledge of computer graphics, digital hardware design, and prototyping; Matthew Klapman for introducing me to virtual reality and for allowing me to model my head-mounted display from his own design; James Kuch for introducing me to LED tracking and for his help with the tracking algorithms; Raymund Angara for implementing the sound spatialization software; and Jennifer Anderson for her assistance in the design of the objects in the virtual world.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

"The environment as we perceive it is our invention."
—H. von Foerster, *Observing Systems* (1981)

Everything that we perceive must, at some point, have been translated into electrochemical impulses for our nervous system. Surprisingly, these impulses contain no data revealing their type or origin. As H. von Foerster [1] states in his "Principle of Undifferentiated Encoding":

> The response of a nerve cell does not encode the physical nature of the agents that caused its response. Encoded is only "how much" at this point on my body, but not "what."

This principle has some interesting repercussions. One is that our nervous system has no way of discerning between signals originating from "out in the real world" and signals originating internally. Another is that the "what" von Foerster refers to, the mapping from electrochemical impulses to physical things, is something that each of us has invented during our development. These points suggest that reality is not a well-defined and tangible thing. They also suggest that it may be possible to present stimuli, not corresponding to physical things in the "real world," which would cause a person to perceive an alternate environment which is just as real as any other.

Virtual reality, as it is commonly called, involves intercepting a user's stimuli from the "real world" and replacing them with *believable* computer-synthesized stimuli. The "believable" part is related more to interactivity than to realism. For example, we could sit a person in front of a large projection screen and show him a videotape of a beautiful valley with a running stream nearby. This visually realistic scene might fool our subject until he decides to look up at the sky and only sees the ceiling, or becomes thirsty and bumps into the screen as he attempts to walk over to the stream for a drink. If he had some way of doing such tasks, maybe he would not mind the grass appearing like indoor carpeting and the trees not casting shadows (two common signs of a limited graphics system).

Virtual reality (VR) systems often attempt to stimulate a user through the senses of sight and hearing, but only because computers have evolved to stimulate these two senses more than the others. It is possible for a VR system to work without video and sound, just as it is possible for a blind and deaf person to experience reality. The only thing necessary is for the user to develop a mapping from the impulses received by her nervous system to the things in her (virtual) environment. An excellent example of a limited VR is a text-based virtual community, many of which exist on the Internet. Users interact with an environment, which includes other people, using only a keyboard and a text terminal. Just as we can read a good book and feel as if we were "there," descriptions in a text-based virtual environment serve to replace our senses of sight, hearing, touch, etc. Our nervous system has the incredible ability to fill in the missing electrochemical impulses internally (that is, we imagine). The text-based VR experience is much more "real" than simply reading a book because the user can interact with her environment.

Some applications require a more sophisticated VR system. Because humans have evolved an exceptional spatial ability, we often need a "hands-on" interactivity in order to accomplish tasks. An exciting future application of virtual reality will be to design machines at the atomic scale. Advancements in the field of molecular nanotechnology will soon allow us to manipulate matter to the extent of combining individual atoms to form mechanical devices [2]. Atoms have many characteristics such as size, weight, attraction and repulsion, and "slipperiness," which will make building such devices using current computer-aided design methods nonintuitive. Virtual reality will be the only means to shrink ourselves to the atomic level and slow down time so that we can build nanoscale systems as if we were playing with Tinker-Toys.

This thesis describes the design and implementation of a virtual reality system, hereby referred to as "the VRS." In capability, the VRS falls somewhere between a text-based VR system and the full-fledged system that would be required for our atom-assembling application. It generates images and sounds which are tightly linked to the actions of the user, providing him with the ability to explore and interact with a spatial environment. The VRS is a working project which, to date, has successfully introduced hundreds of people to the concept virtual reality.

The following chapter presents an overview of the VRS, describing the system from both an engineer's and a user's point of view. Chapters 3-7 give detailed presentations of the various hardware and software components of the VRS: the graphics system, the head tracking system, the sound spatialization system, and the head-mounted display. Chapter 8 makes some suggestions for future work. The appendices provide technical documentation of the system's hardware and software, including schematic diagrams and program listings.

2. SYSTEM OVERVIEW

The VRS is a complex system consisting of a number of hardware and software components. These components interact with each other and the user to provide the user with a virtual environment of sight and sound.

The VRS is based around a desktop personal computer, which is considered to be the *host* of the system. A number of expansion cards, which plug into the bus of the host, have been designed specifically for this project. Most of the expansion cards contain their own processor and memory so that they may perform tasks without requiring assistance from the host's central processing unit (CPU). Another advantage to having a processor on each expansion card is that the processors can work in parallel. In all, there are five processors in the VRS.

The main functions of the host machine in the operation of the VRS are to provide:

- a communication link between the other components of the VRS
- an interface to basic input devices (that is, keyboard and joystick)
- a means of file access

For readers who have not had the opportunity to see and use the VRS, this chapter provides a description of the **physical components** and **operation** of the VRS.

2.1. Physical Components

The physical components of the VRS are shown in Figure 2.1. The host machine is an Intel 80486-based personal computer running an MS-DOS operating system. There are four expansion cards in the host machine that were designed for the VRS. One card is the G-Node graphics synthesizer card, which outputs images to the head-mounted display (HMD). Two identical V-Node video digitizer cards process video inputs from black-and-white cameras. The cameras view the head movements of the user. The final expansion card interfaces the host computer to an external digital signal processor (DSP) board. The DSP board performs sound spatialization processing on a sound source, such as a compact disc player, and drives four audio speakers. An analog joystick, connected to the host computer, provides another input from the user.

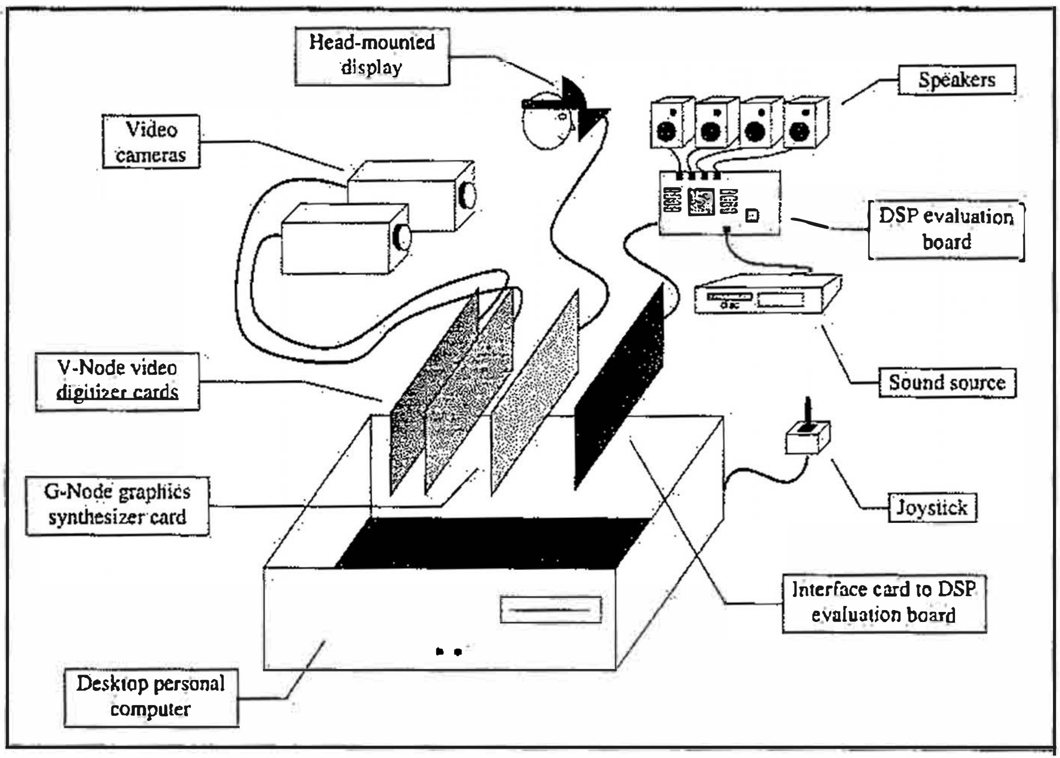


Figure 2.1. Physical components of the VRS.

2.2. Operation

The user sits on a stool facing a workbench on which the VRS resides. A steel frame attached to the bench supports two video cameras. One camera is above the user, pointing towards the floor. The other is to the user's right, pointing towards the left. The user must position herself so that her head is in the center of both camera views. A video monitor on the bench can display the output of either camera, and is used to check the alignment of the user. Before beginning, the user puts her favorite compact disc into a player that is mounted in a rack on the bench.

The user now places the HMD on her head and adjusts the headband for a secure fit. Looking ahead, she sees a colorful landscape filled with animated objects. She looks around by moving her head left, right, down, and up. Noticing a pinwheel twirling in the sky, she takes hold of the joystick in front of her and begins to maneuver herself towards the spinning object. Pushing forward on the joystick allows her to move ahead. The more she pushes, the faster she moves. Pushing the joystick to the left or right causes her to pivot. Once directly under the pinwheel, the user looks up at it. After a few seconds she begins to feel dizzy.

At this point, the user hears a familiar song that seems to be emanating from a distance off to her left. Looking in that direction, she sees a blue musical note that appears to be bouncing up and down. As she moves towards the note, the music grows in intensity. The user rushes past the note, which at close range she finds to be many times taller than herself. She continues onward to do more exploring, as the music decays off in the distance behind her.

3. GRAPHICS SYSTEM

A key quality of a virtual reality system is the ability of its graphics system to respond quickly to inputs from the user. This chapter looks at the three tightly linked elements of the VRS that affect the graphics performance. The first element is the **G-Node graphics synthesizer card**, designed specifically for the VRS. The second element is the method in which the virtual world is visually represented, in this case by **sphere-based graphics**. The final element is the **graphics pipeline** implementation which determines how a database of objects is translated into a visual scene.

3.1. G-Node Graphics Synthesizer Card

The G-Node graphics synthesizer card was designed in order to significantly improve the graphics performance of a desktop computer. Although it plugs into an IBM-AT compatible machine as an expansion card, the G-Node can be thought of as a stand-alone graphics computer. For example, one can write a program in C that displays a 3-D object rotating in space, compile it, and transfer it to the G-Node's on-board processor. The G-Node runs the program, and the rotating object appears on a display screen connected to the card. At this point, the host computer is free to run other programs since the graphics program is running independently on the G-Node.

The G-Node design is based around a Texas Instruments TMS34020 graphics system processor (GSP), which is a 32-bit microprocessor optimized for use in graphic display systems. In addition to a general-purpose processing unit, the GSP has an on-board graphics controller and hardware support for graphical data types such as pixels and 2-D pixel arrays. As shown in Figure 3.1, there are four major devices connected to the GSP:

- *video random access memory (VRAM)*, for storing the display screen, program code and data

- *a random access memory-digital to analog converter (RAM-DAC)*, which converts a digital pixel stream into an analog red-green-blue (RGB) signal

- *the host CPU*, via the Industry Standard Architecture (ISA) bus

- *a floating-point coprocessor*

The memory on the G-Node consists of one megabyte of VRAM organized as 256K x 32-bits. A VRAM chip has two access ports to its memory cells. One is a standard dynamic RAM (DRAM) port and the other is a serial access memory (SAM) port which is used for transferring pixels to the display screen. The GSP manipulates pixels in the display buffer through the DRAM port. In addition, the GSP's video controller uses the DRAM port to send commands that transfer one or more lines of the display buffer to the SAM port.

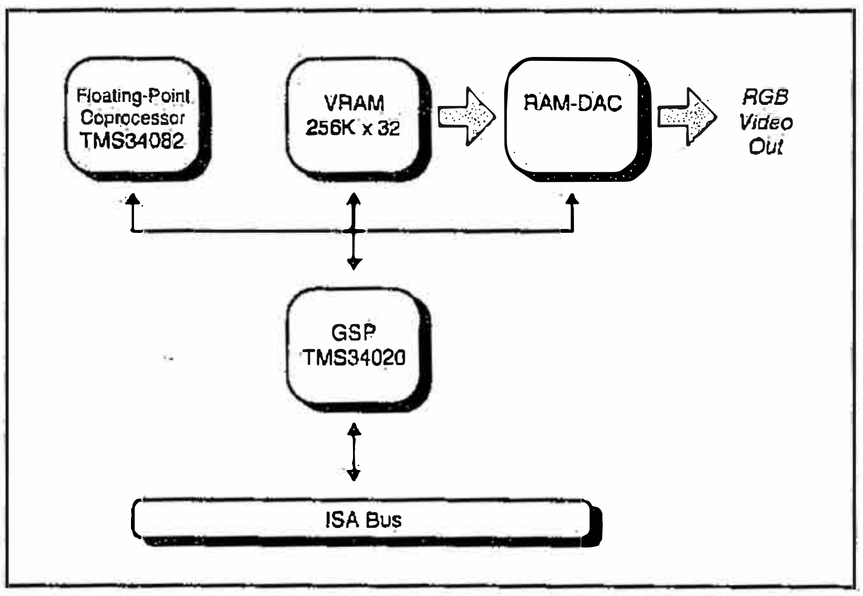


Figure 3.1. G-Node block diagram.

VRAM is normally used to store display buffers exclusively, but in the G-Node design it is also used to store program code and data. The result is that the available code and data space are dependent on the number of display buffers and the display screen resolution. When configured for double-buffered, 640 x 480 output, the G-Node has 64K of available RAM.

For double-buffered, 160 x 240 output (used to drive the HMD), the G-Node has 904K of RAM available.

The GSP has a host port that allows another processor to access its local bus. In the G-Node design, the GSP's host port is connected to the ISA bus. This allows the main CPU of the VRS to read and write to the RAM on the G-Node. Such accesses are transparent to the program running on the GSP. When the VRS system is being initialized, the host port is used to transfer the GSP's code into the on-board RAM. During normal operation of the VRS, the host port is used as a means of communication between the programs running on the GSP and the host CPU. This is accomplished by setting aside a portion of the GSP's RAM as a communication buffer. The host CPU loads data into the buffer and then sets a flag which lets the GSP's program know that the buffer holds valid data. Since host accesses are transparent to the GSP, it is free to perform other tasks while the buffer is being filled by the host. Once the GSP finishes operating on the data, it clears the flag and the process is repeated.

Also connected to the GSP is a Texas Instruments TMS34082 graphics floating-point coprocessor, which can quickly perform floating-point operations for the GSP. In addition, the TMS34082 has its own instruction sequencer. Built-in read-only memory contains routines to perform calculations that are critical to a 3-D graphics pipeline such as vector operations and window clipping. The coprocessor also has the ability to run custom routines from external RAM, but this feature is not supported in the G-Node design.

Although the GSP can support color resolutions of 1, 2, 4, 8, 16, or 32 bits per pixel, the G-Node design is fixed in hardware at 8 bits per pixel. Eight bits per pixel provides 256 simultaneous colors from a 16.8 million-color palette (the RAM-DAC has triple 8-bit video digital to analog converters, yielding 2^{24} possible colors). This seems to be an adequate number of simultaneous colors for a system that uses constant shaded (that is, single color) graphics primitives, as the VRS does. Supporting only one color resolution eliminates the need for pixel multiplexing hardware between the VRAM SAM ports and the RAMDAC.

The choice of color resolution obviously affects the amount of frame buffer memory required, but it also affects the speed at which the GSP can write pixels to memory. Since the GSP's data bus is 32 bits wide, it can access four 8-bit pixels simultaneously. The GSP's page mode access to VRAM runs at a speed of 8 MHz, so the resulting pixel bandwidth in the G-Node design peaks at 32 million pixels per second.

3.2. Sphere-based Graphics

There are a number of ways to represent three-dimensional solid objects on a graphics display. When rendering speed is critical, a common method is to approximate the shape of an object using polygon faces [3]. An arbitrary three-dimensional polygon in space will always project onto the viewing plane as a two-dimensional polygon. Such 2-D polygons can be easily rendered on the display screen.

Polygon representations work well for boxy shapes but are not well suited for shapes with curved surfaces. To accurately approximate a sphere, for example, many polygon faces are required (Figure 3.2). For each polygon face that is sent through the graphics pipeline, a matrix transformation must be applied to each of its vertices. The result is poor real-time performance in rendering rounded objects. Smooth polygon shading methods, such as Gouraud shading, have been designed to reduce the number of faces needed to approximate curved surfaces. Unfortunately, such techniques significantly increase the time it takes to draw a polygon. (Smooth-shading prohibits the use of VRAM block pixel-writes.)

The desire to produce computer animation with a limited system can often result in creative solutions. One such example is work done by a group of Dutch programmers to produce impressive real-time graphics from limited Intel 80286/386-based personal computers. (The program is called *VectorDemo*, by UltraForce Development, 1991.) Such machines have a very low processor-to-display bandwidth and slow (if any) floating-point computational hardware. One technique, employed by the *VectorDemo* program to overcome these limitations, is to represent 3-D solid objects with spheres. An arbitrary sphere in space will always project onto the viewing plane as a two-dimensional circle. The result is that rounded, three-dimensional shapes can be rendered entirely by drawing circles (see Figure 3.3). I call this type of representation *sphere-based graphics.*

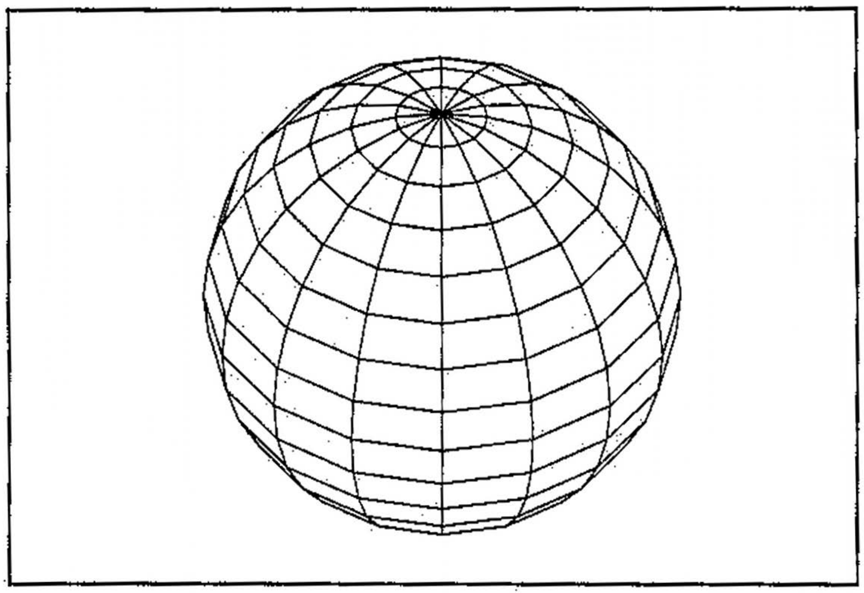


Figure 3.2. A polygon-based sphere.

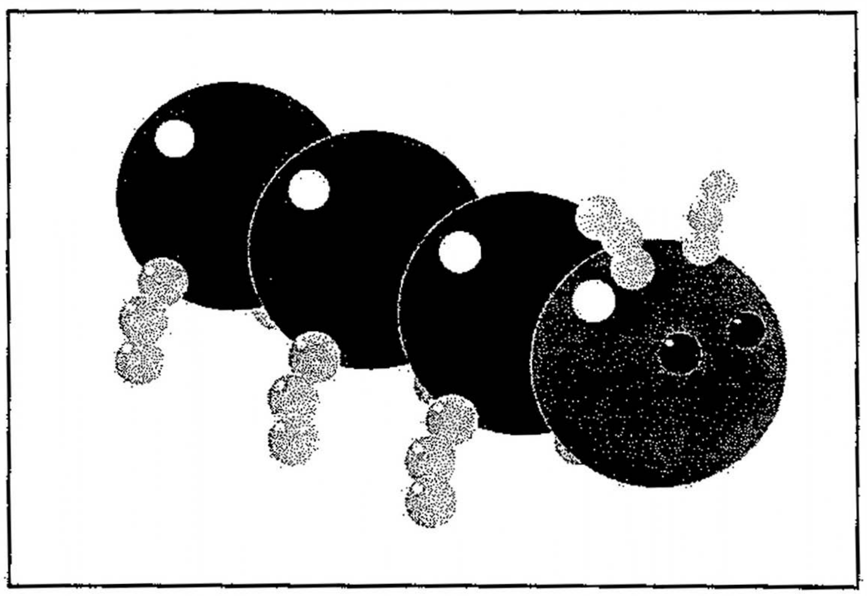


Figure 3.3. A "cootie bug" rendered with sphere-based graphics.

Sphere-based graphics have a number of advantages over polygon representations. The first is that only one point per sphere, the center point, has to pass through the graphics pipeline. (Recall that, for polygons, each vertex has to be transformed.) A small calculation is also necessary to determine the radius of the circle projected onto the viewing plane:

$$r_{projected} = \frac{r_{sphere} \times f}{d}$$

where r_{sphere} is the actual radius of the sphere, d is the distance between the viewer and the sphere, and f is the focal length (that is, the distance between the viewer and the view plane).

Sphere-based rendering takes place one sphere at a time. To make the spheres overlap properly, they are drawn in order from farthest (relative to the viewer) to nearest. For each sphere, a two-dimensional circle is drawn which corresponds to the sphere's color and projected diameter. In addition, a smaller white circle is drawn to simulate a highlight spot. The highlight has the effect of making the spheres look more three-dimensional. (The *VectorDemo* program used prestored images of ray-traced spheres for a higher degree of realism.) The highlight spots also produce an "automatic" lighting effect at the object level. When an object is oriented so that the highlight spot of each of its spheres is visible, the object appears to be facing the light source (Figure 3.4(a)). When the spheres are positioned so that most of the highlights are occluded, the object appears to be facing away from the light source (Figure 3.4(b)).

In the VRS graphics pipeline, the highlight spots are always drawn in the upper-left corner of each sphere. This models a light source which is infinitely far away (that is, it casts parallel rays of light) and is in a fixed position with respect to the viewer (for example, the light always comes from over the viewer's left shoulder).

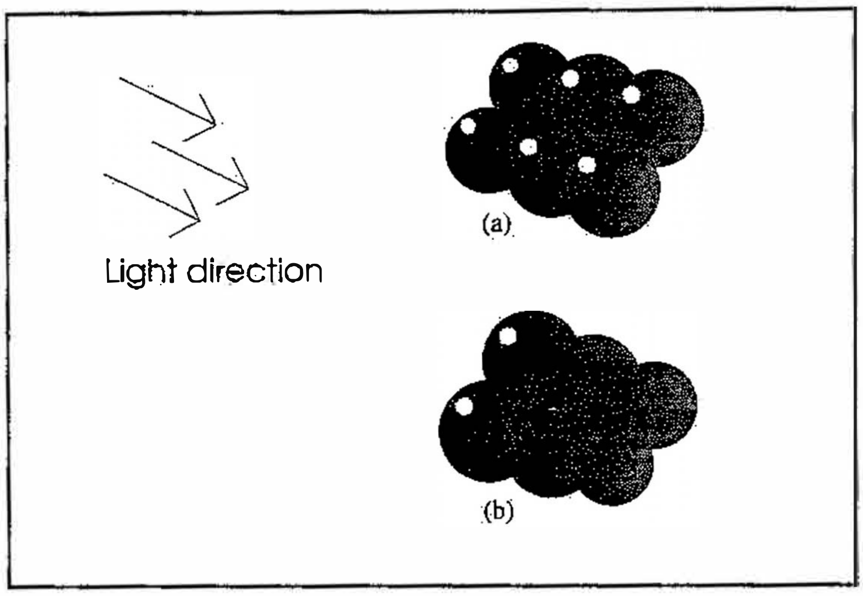


Figure 3.4. Automatic lighting effect of sphere-based graphics. An object facing the light (a) and another not facing the light (b) are shown.

There are some limitations to sphere-based graphics. Just as polygon-based graphics are not well suited for representing curved surfaces, sphere-based graphics are not well suited for representing flat surfaces. For example, a wall could be accurately represented with only six polygons (Figure 3.5(a)), but many spheres would be required to make a bumpy approximation of a wall (Figure 3.5(b)). This suggests the creation of a graphics system that could generate both sphere-based and polygon-based representations, using each where appropriate.

Another limitation of sphere-based graphics becomes apparent in the case of intersecting spheres. When two spheres intersect, the resulting shape is obviously not a sphere. Therefore, the projection of such a shape onto the viewing plane cannot be rendered using circles. As an example, Figure 3.6(a) shows the correct rendering of two intersecting spheres. By rendering the spheres one at a time, only the renderings in Figure 3.6(b) and Figure 3.6(c) are attainable. Furthermore, an object may change quickly between these two incorrect renderings as the distance between the viewer and each sphere changes. This causes a noticeable "popping" effect.

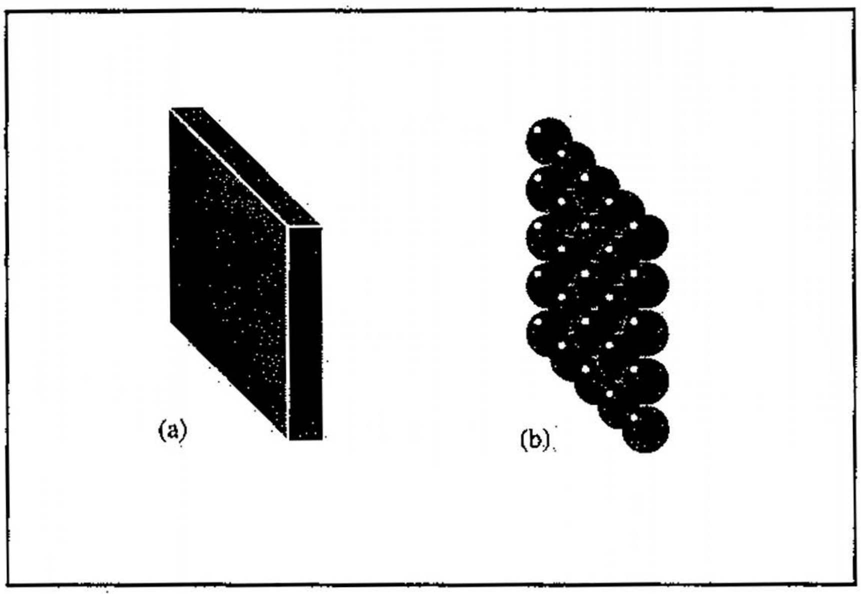


Figure 3.5. Flat surfaces: polygon (a) vs. sphere (b) representations.

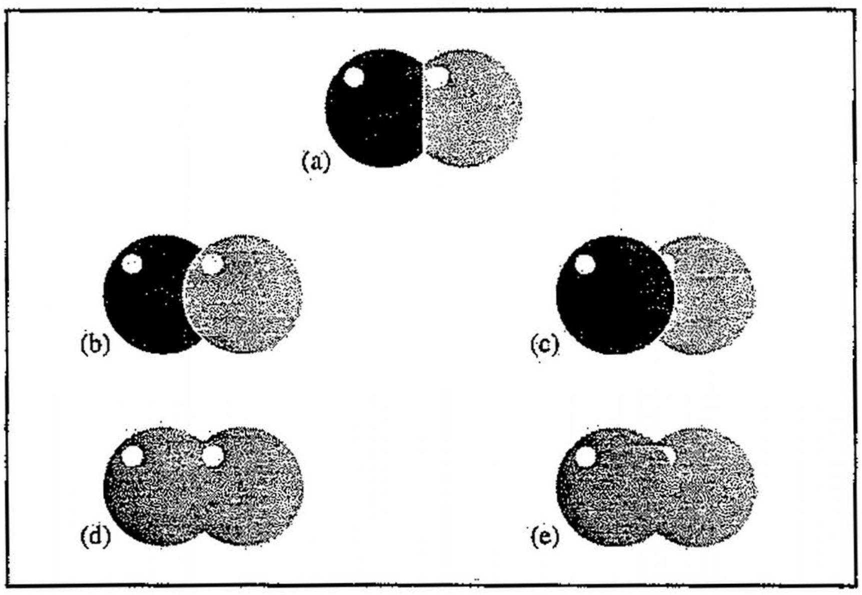


Figure 3.6. The intersecting-spheres problem. A correct rendering (a), two incorrect renderings (b) and (c), and a proposed solution (d) and (e) are shown.

Some methods can be employed to sidestep the intersecting-spheres problem. One method is to avoid intersecting spheres altogether when designing objects. This solution is rather prohibitive. (Imagine trying to build a model car out of marbles.) A more reasonable method is to limit objects to having only slightly intersecting spheres. The frequency and degree of popping are proportional to the depth of the intersection. This solution would minimize popping while providing more flexibility in object design. Still another solution requires two changes: (1) design objects such that intersecting spheres are the same color, and (2) remove any border from the rendered circles. The result is shown in Figure 3.6(d). Note that the highlight spots can foil this method, as shown in Figure 3.6(e).

3.3. Graphics Pipeline

A *graphics pipeline* defines the sequence of operations necessary to convert a database of objects into a graphics image. Each visual frame generated by the system requires a single pass through the pipeline. In the VRS, the workload of the graphics pipeline is divided between the host CPU and the G-Node board's processor. An important decision to be made is where to split the pipeline between the two processors. It is desirable to balance the workload so that neither processor has to wait for the other to finish its share of the work. Another factor that must be considered is the amount of data flowing through the pipeline at the point where it is split. This directly affects the amount of time that the processors must spend transferring data.

The stages of a graphics pipeline are dependent on both the graphics hardware and the method of object representation. For example, the G-Node board does not have hardware depth-buffering, so a depth-sorting stage is necessary in the VRS graphics pipeline. Another example is that, since sphere-based graphics have an "automatic lighting" attribute, a lighting stage is not needed. A block diagram of the VRS graphics pipeline is shown in Figure 3.7.

The VRS graphics pipeline is divided into eleven stages. The first eight stages are performed by the host CPU and the final three stages by the processor on the G-Node board. In the first stage, the host samples the user's input devices, consisting of the joystick, the two V-Node boards, and the keyboard. The joystick determines the movement of the user's view reference point, and the V-Node boards provide the user's view direction. Together these data are used to calculate the view vector in the second stage of the pipeline. The third pipeline stage uses the view vector to form a viewing transformation matrix. This matrix is used to transform three-dimensional points from world coordinates into viewing coordinates.

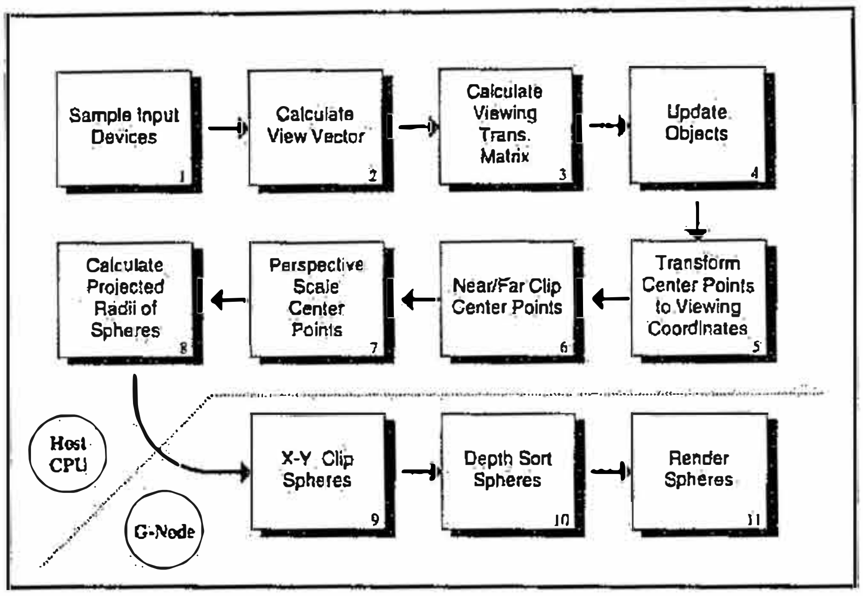


Figure 3.7. The VRS graphics pipeline.

Up to this point, the data passing through each pipeline stage have been a fixed size. This means that the time spent in the first three stages will be relatively constant. In contrast, the amount of data flowing through the remaining eight stages may vary, affecting the time needed to complete each stage. For example, the amount of data (in this case spheres) traveling through the sorting and rendering stages is dependent on how many spheres are currently in the user's view.

Stage four is responsible for updating the objects in the virtual world. An object consists of one or more spheres. Attributes that may be updated include position, size, and color.

Although the G-Node board is better suited than the host CPU for handling pipeline stages five through eight, these stages still reside on the host CPU in the current implementation. In stage five, the center point of each sphere in the world is transformed from world to viewing coordinates using the viewing transformation matrix. In stage six, all spheres whose center points are outside the bounds of the near and far planes are removed from the pipeline. Stage seven involves a perspective scaling of the center points. In stage eight, the projected radius of each sphere is calculated using the equation presented in Section 3.2.

After completing stage eight, the host CPU transfers the resulting data to the G-Node card, which will use the data in the final three stages of the graphics pipeline. The data consist of a list of spheres. Each sphere is defined by an *x-y-z* position, a diameter, and a color. At this point in the pipeline, the *x*, *y*, and diameter values are in screen coordinates. All values are transferred as 16-bit integers. Note that the host CPU and G-Node card operate on their respective parts of the graphics pipeline in parallel. After the host CPU completes its final stage and transfers the sphere data to the G-Node, the host CPU can immediately return to the beginning of the pipeline to work on the next visual frame.

In stage nine, the G-Node's processor removes spheres from the pipeline that are completely outside the bounds of the screen. (Spheres that are partially clipped by the screen's borders will be handled in hardware by the TMS34020 chip.) In stage ten, the spheres are sorted by the *z*-coordinate of their center point. Finally, in stage eleven, the G-Node card renders the spheres to the display buffer. This rendering includes drawing the highlight spot on each sphere. The spheres are drawn, in order, from farthest to nearest.

4. HEAD TRACKING SYSTEM

An important attribute of a virtual reality system is the existence of input devices that are natural to the user. When first born, humans do not stimulate their environment through keyboards and joysticks. Instead, we use our voice and body motions. This suggests that a computer with sound and sight inputs will be easier for us to use. A limited form of "sight," often used in virtual reality, is achieved when the computer can determine the position and orientation of the user's body. This method of input is known as body tracking.

Body tracking is currently one of the most difficult problems in the implementation of virtual reality. Typical simplifications of this problem are achieved by tracking only a few key parts of the body, and by tracking with less than the full six degrees of freedom (*x-y-z* position and *roll-pitch-yaw* orientation).

The VRS incorporates a head tracking system that determines the *pitch-yaw* orientation of the user's head. This vital data allows the VRS to respond to the user's action of "looking around." The *roll* orientation, which corresponds to the tilting of the head right or left, is not tracked by the system. This omission is not very noticeable since we normally keep our eyes level with the horizon (assuming that the environment has a horizon).

The VRS tracking is accomplished by attaching light emitting diodes (LEDs) to the user which are sensed by video cameras. The remainder of this chapter examines the **V-Node video digitizer card**, designed specifically for the VRS, and the details of the **LED tracking** method employed.

4.1. V-Node Video Digitizer Card

The V-Node video digitizer is an expansion card designed for the ISA bus. Independent of the host CPU, the V-Node can capture images from a video camera and perform image processing using its on-board graphics processor. This allows live video to be used as a form of input that places very little load on the host CPU, similar to a keyboard or a mouse.

Looking at Figure 4.1, it is apparent that the V-Node design is similar to that of the G-Node at the block level. With respect to the flow of data, the V-Node is identical to the G-Node, except that the V-Node is running in reverse. Recall the function of the G-Node card. It processes a small amount of data (such as sphere geometries) into a large amount of data (many thousands of pixels) which are stored in VRAM. Once an entire image is generated, the pixels are transferred out of the VRAM through the SAM port and converted to analog video. In the complementary V-Node design, analog video is converted into digital pixels, which are then transferred into VRAM by way of the SAM port. Once an entire image is captured, the V-Node processes this large amount of pixel data to extract a small amount of visual data.

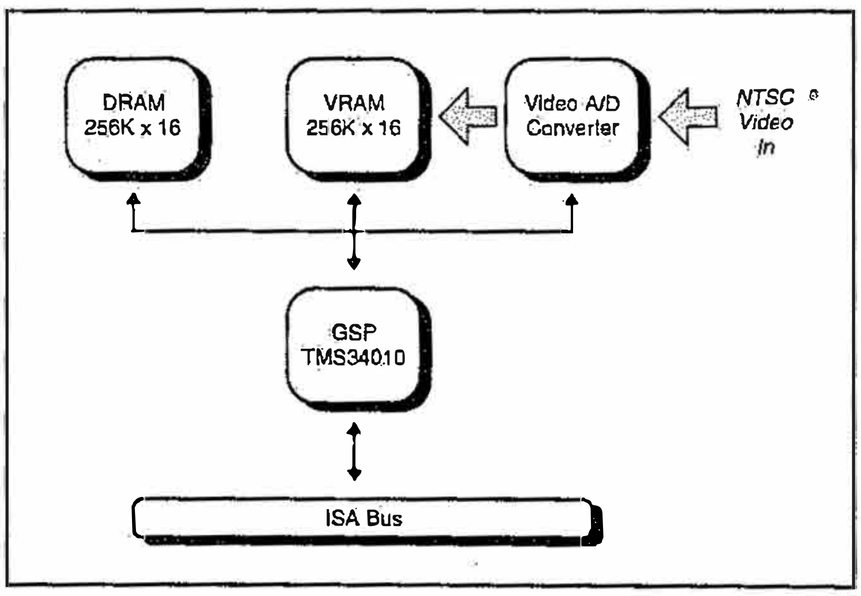


Figure 4.1. V-Node block diagram.

Like the G-Node design, the V-Node is based around a GSP: in this case the Texas Instruments TMS34010 processor. The TMS34010 is the predecessor to the TMS34020, having a sixteen-bit data path to memory (versus the TMS34020's thirty-two bit path), a slower operating speed, and a less powerful instruction set. The GSP is connected to 512K bytes of DRAM which store programs and data, and 512K bytes of VRAM which store captured images. The video A/D converter used in the design can sample at eight bits per pixel and includes a genlock and a pixel lookup table. The genlock circuit extracts the synchronization signals, which are required for coordinating the SAM port transfers, from the video source. The programmable pixel lookup table is useful for tasks such as on-the-fly image thresholding.

The V-Node card has the ability to digitize a 512 x 512 pixel, 8-bit grayscale image in 1/30th of a second (only 256 x 256 pixel images are used for the VRS head tracking). The image capture occurs in the background of the GSP's program execution, so the program running on the GSP could be processing one image while the next image is being digitized.

4.2. LED Tracking

Judging by the amount of research that is conducted in the area every year, computer vision is a very difficult problem. The head tracking subsystem of the VRS required a practical implementation of computer vision that was within the limits of the V-Node's image processing capability. Since the user must already wear an HMD to use the system, it makes sense to attach visual indicators to simplify the computer vision task. LEDs were used as visual indicators with the idea that it would be relatively easy to track a small point of light within an image.

Although it is possible to extract three-dimensional position data from a single camera view [4], the calculations involved could not be executed by the V-Node's GSP in real time. Instead, the head tracking subsystem employs a simpler two-camera setup, as shown in Figure 4.2. One camera is positioned directly above the user, and a second camera is positioned at the user's side. Two LEDs are attached to the top of the HMD. The LEDs are positioned so that the imaginary line between them points in the direction that the user is looking. Both cameras are in the plane of the user's torso and are oriented orthogonally in space.

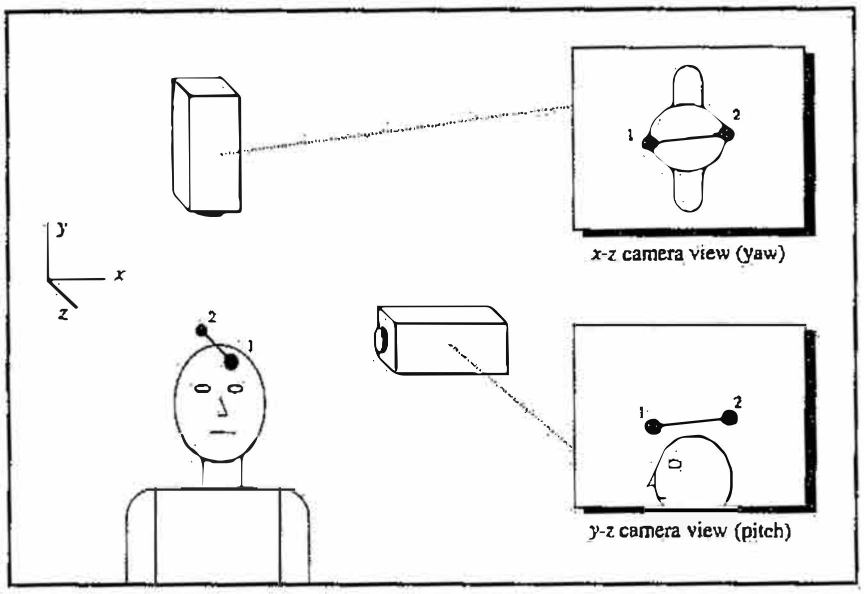


Figure 4.2. Two-camera LED tracking.

The top camera provides a view of the x-z plane, while the side camera shows the y-z plane. Once the positions of both LEDs within each camera's view are determined, the data can be combined to form a three-dimensional ray which indicates the user's direction of view.

Two V-Node cards, one for each camera, are used in the head tracking subsystem. Their task is to digitize an image and determine the position of each LED in as short a time as possible. To simplify this task, a number of assumptions are made. The first assumption is that the red LEDs that appear in the image will have a greater light intensity than the rest of the scene. To help make this assumption valid, red gel filters are placed over the camera lenses. The gels have the effect of allowing red light to pass while attenuating light at other wavelengths.

With the assumption that the LEDs will be the brightest objects in the camera's view, they can be located by a simple thresholding of the digitized image. This thresholding is performed "on the fly" by the video A/D converter. As each pixel is digitized, its value is looked up in a RAM conversion table that resides on the A/D chip. The RAM table must be programmed according to the desired threshold intensity. As an example, if we wanted to threshold at a value of 128 (zero corresponds to black and 255 to bright white), we would fill locations 0-127 of the table with 0's and locations 128-255 with 1's. The result will be a monochrome image, with pixels below the threshold set "off" and pixels equal to or above the threshold set "on."

The tracking is accomplished by scanning the thresholded image, columnwise, from left to right. The first pixel that is on is assumed to indicate the position of LED1. Next the image is again scanned columnwise, but from right to left. The first pixel that is on is assumed to be LED2. The assumption here is that LED1 will always be to the left of LED2. Consider the home position to be the case when, in both camera's view, LED1 is to the left of LED2, and the LEDs are horizontal. This home position corresponds to the user looking straight ahead. Given the constraint that LED1 must stay to the left of LED2 in both images, ideally this gives the user $\pm 90°$ of yaw and $\pm 90°$ of pitch movement relative to the home position.

The tracking method described in the previous paragraph requires a large number of pixel comparisons. Since images are digitized at a resolution of 256 x 256 pixels, up to 65,536 comparisons may have to be performed by the V-Node's GSP. To improve the response time of the image processing, a second tracking method is employed which requires fewer pixel comparisons. Instead of scanning the entire image for an LED, only a small region is scanned around the position where the LED was last spotted. The assumption is that an LED will not move very far from one frame to the next. A 40 x 40 pixel region is scanned around each LED, resulting in a maximum of 3200 comparisons. If an LED is not within its region, the system reverts to the full-screen scanning method to locate it.

The VRS head tracking subsystem has a throughput of fifteen updates per second.

5. SOUND SPATIALIZATION SYSTEM

Together the graphics system, the head-tracking system, and the user form a feedback system. Visual images respond to input from the user who may, in return, respond to the visual images. By adding to this system a second stimulus for the user, we can strengthen the user's perception of the virtual environment. Since sound can be manipulated by computer without much difficulty, it is a practical choice for a second stimulus. A system that can simulate a sound emanating from an arbitrary point in space is called a *sound spatialization system.*

This chapter describes a means of sound **spatialization using intensity** and provides a description of the **audio hardware** used to implement it.

5.1. Spatialization Using Intensity

There are a number of cues that allow us to determine the direction and distance of a sound. One of the most prominent distance cues is intensity. We are able to judge the distance of a familiar sound, such as a car horn, by how loud it is. Even in the case of an unfamiliar sound, we are still able to determine relative distances.

Looking at Figure 5.1, we can make a sound emanate from point A by playing it through a loudspeaker at that position. Without moving the speaker, it is possible to make the same sound seem to originate from point B using the intensity and distance relationship:

$$I \propto \frac{1}{d^2}$$

So if point B were twice the distance from the listener as point A, we would have to play the sound with one-fourth the intensity.

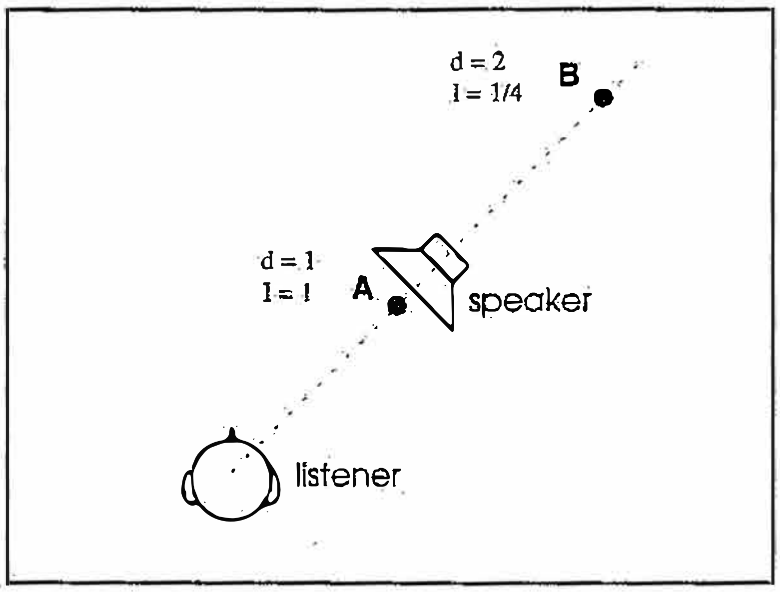


Figure 5.1. Controlling perceived distance through intensity.

Intensity is also a directional cue. Looking at Figure 5.1 again, it is apparent that the right ear of the listener, which is "facing" the speaker, will sense a greater intensity than the left ear. Our ability to determine the direction of a sound is due in part to this *interaural intensity difference*. Given the two-speaker setup in Figure 5.2, we should be able to apply this principle to the problem of positioning a virtual sound at an arbitrary point between the speakers. The method is to play the same sound in both speakers and vary the gain of each to control the interaural intensity difference. For example, to create a virtual sound that originates from directly between the speakers (point *C*), we would set each speaker to an equal gain factor. To place a virtual sound closer to speaker *1* (point *D*), we would increase the gain of speaker *1* and decrease the gain of speaker *2*. This is called intensity panning.

The distance and direction intensity cues can be combined to position a virtual sound at any point in the horizontal plane of the listener. Moore [5] derives the following general purpose intensity-localization rule:

$$G_n(\theta, d) = \begin{cases} \frac{d_n}{d}\cos(\theta - \theta_n) & \text{if } |\theta - \theta_n| < 90^\circ \\ 0 & \text{otherwise} \end{cases}$$

where $G_n(\theta,d)$ is the amplitude gain for channel n for a virtual sound source located at a distance d and an angle θ from the listener, and θ_n and d_n are the angle and distance between the speaker n and the listener. This equation requires a minimum of four speakers, placed at 90° intervals, to cover the full 360° around the listener.

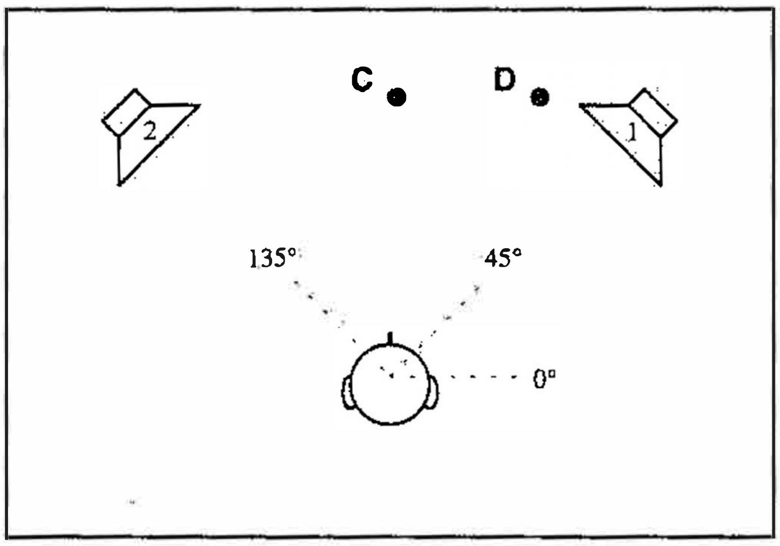


Figure 5.2. Controlling perceived direction through intensity.

Using intensity panning alone for sound spatialization has its limitations. Such a system can only create a "weak" virtual sound between two speakers. Nevertheless, intensity panning is simple to implement and provides a rough clue as to the position of a virtual sound (for example, it is easy to discern between a virtual sound coming from the left and one coming from the rear).

5.2. Audio Hardware

The VRS audio hardware implements a four-speaker intensity panning system. It accepts up to four mono-audio sources. The system can process each audio source to produce a virtual sound channel. The audio hardware consists of an off-the-shelf DSP evaluation board made by Analog Devices (ADSP-21020 EZ-LAB™ Evaluation Board, hereby referred to as "the EVB"). The EVB includes a floating-point DSP; 32K-words of program memory and 32K-words of data memory; a 16-bit, stereo audio coder-decoder; and a two-channel, 8-bit, combined analog to digital and digital to analog converter.

The host system is responsible for calculating the distance and direction of the virtual sound source in relation to the listener. This data must be conveyed to the DSP, which is running the sound spatialization program. Since each virtual sound has to be updated only once per pass through the graphics pipeline, and since a virtual sound's position can be specified in only four bytes, a simple serial link would be well suited for communication between the host and the DSP. Unfortunately, the serial port of the EVB was not designed to be used while the DSP is running code. Therefore, it was necessary to design a custom interface card for the host system that connects to an expansion port of the EVB.

The 21020 EVB interface card is an 8-bit ISA card that plugs into the bus of the host machine. A connector at the back of the card links it to the expansion port of the EVB. The interface hardware consists of 1K-byte of dual-ported RAM. One side of the dual-ported RAM is mapped into the memory space of the host CPU, while the other side is mapped into the memory space of the DSP on the EVB. Each processor can access the RAM independently, with the constraint that they do not access the same location simultaneously. In addition, there is a special memory location in the dual-ported RAM used for sending an interrupt from the host CPU to the DSP.

For each virtual sound source, the host CPU places a two-dimensional vector in the dual-ported RAM. The vector specifies both the direction and distance of the virtual sound on the listener's horizontal plane. After the host CPU has loaded all the vectors, it sends an interrupt to the DSP. When the DSP receives the interrupt, it copies the vectors from the dual-ported RAM to its local memory.

The main loop of the sound localization code, which runs on the DSP, operates at a rate of 44.1 kHz, and currently supports only one virtual sound source. The following tasks are performed once per loop:

- Input a digital sample from the A/D converter
- For each of the four channels, calculate gain based on the virtual sound vector
- For each channel, scale the input sample by the calculated gain
- For each channel, output the scaled sample

6. HEAD-MOUNTED DISPLAY

The goal of a virtual reality display is simple. We want to replace the user's vision with computer synthesized images. Current display technology does not provide an ideal solution to this problem. A popular approach is to attach a display to the user's head so that he will see the computer-generated images regardless of what direction his head is turned to. This is called a *head-mounted display* (HMD). Ideally, we want an HMD to be like a pair of sunglasses: lightweight, unobtrusive, cordless, one-size-fits-all, and completely covering the user's field of view. There are many obstacles in the way of this ideal. Small (one-inch diagonal or less), high resolution, full color display screens are not yet commercially available. Attempting to use larger displays, especially for stereoscopic HMDs, results in a need for complicated optics. Liquid crystal displays (LCDs), which are often used in HMDs, require a backlight. Backlights add significant weight, size, and power consumption to an HMD.

For the head-mounted display of the VRS, a design was chosen which could be easily built using commonly available components. This chapter discusses the **display screen**, **optics**, and **mechanics** of the VRS HMD.

6.1. Display Screen

The critical component of an HMD is its display screen. Once a display is chosen, decisions can then be made concerning stereo-vs.-mono, optics, and packaging options. In my search for a display, I had the following minimum requirements:

- *color output* – helps compensate for lack of geometrical detail in synthesized images.
- *analog RGB input* – eliminates the need for converting the video source into a composite signal which would require additional hardware and degrade image quality.

- *built-in backlight* – required since there is no ambient light within an HMD.

Given these basic constraints, it is desirable to use the smallest, highest-resolution display available. Among a severely limited number of choices, a Sharp LQ4RA01 LCD module was chosen for the project. This four-inch-diagonal display has a resolution of 160 x 234, weighs 170 g, and refreshes at a rate of 30 Hz. It requires an analog RGB signal that adheres to NTSC timing specifications, which the G-Node can be programmed to generate. An external DC to AC converter, also produced by Sharp, is required to power the LCD's backlight. In addition, two external potentiometers are used to adjust the brightness and contrast of the display. A block diagram of the display system is shown in Figure 6.1.

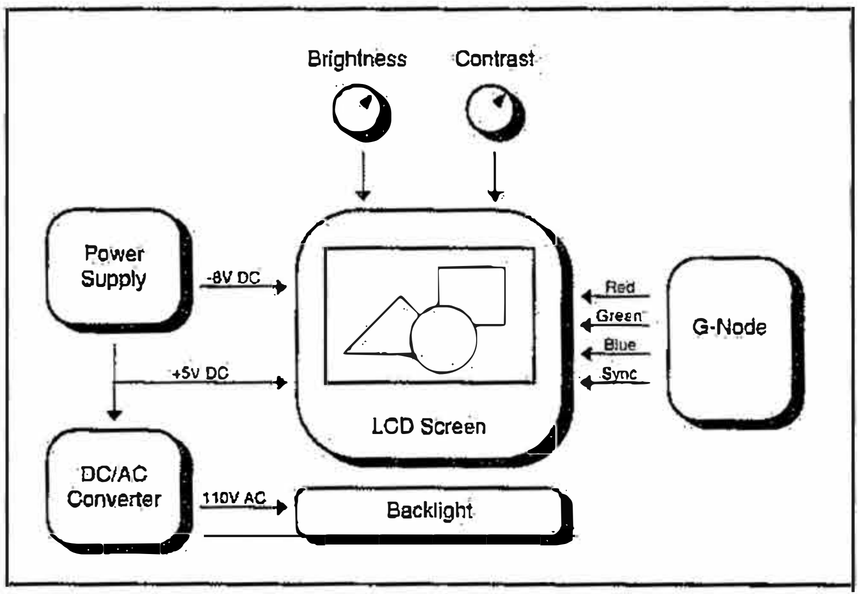


Figure 6.1. Display system block diagram.

6.2. Optics

Ideally an HMD should have two display screens. By presenting a separate image to each eye, a virtual reality system may generate stereoscopic cues that we use for depth perception. Unfortunately, due to the size and weight of the display screen used in this project, it was not practical to build a dual-screen HMD. It may seem that a single-screen HMD would be relatively simple in design compared to a stereo HMD, but this is not the case. The problem is that when we have both eyes looking at the same screen, they must converge on the image. Normally we want to bring the screen very close (less than six inches) to the user's eyes so that the image covers a large percentage of his field of view. As a consequence our user will soon get a headache from trying to maintain convergence on a screen that is so close to his eyes. In the design of a monoscopic HMD, there is a tradeoff between visual comfort and field of view.

By including some optics in an HMD, it is possible to improve its visual characteristics. Heavy or bulky lenses are undesirable for an HMD design. This project uses a Fresnel lens. In addition to being almost paper-thin, a Fresnel lens makes objects appear larger and farther away, thereby reducing eye fatigue. The magnification that the lens provides can be used either to improve the HMD's field of view or to allow the display screen to be placed at a greater distance from the user's eyes. The drawback of a Fresnel lens is that it is not a "perfect lens" and, therefore, distorts the image. This distortion manifests itself as a slight spherical aberration.

For the particular Fresnel lens used in this project, the display screen is in best focus at a distance of six inches from the user's eyes. If the LCD module were placed six inches from the user's head, then, due to the module's weight, a counterbalance would be required. This would increase the overall weight of the HMD. The solution is to place the LCD module near to the head, facing the floor, and use a mirror to reflect the image into the eyes (Figure 6.2). The mirror is positioned so that all points on the display screen's surface appear to be six inches from the user's eyes. Because of this reflection, the image that the user sees is inverted around the y-axis. The graphics display system must take this inversion into account when generating an image.

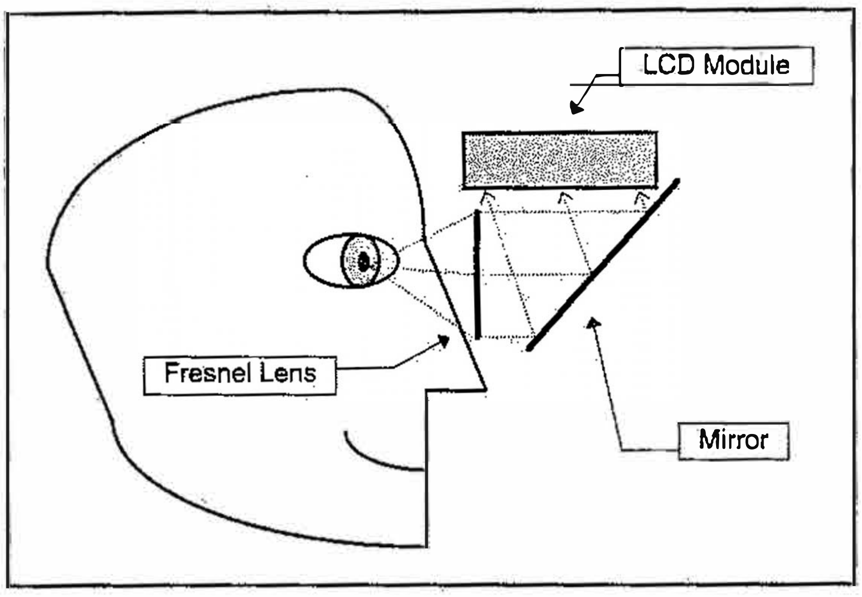


Figure 6.2. The optical system of the HMD.

6.3. Mechanics

The HMD is built around a Crews brand plastic helmet that has a clear shield to protect the face. This open-top helmet has a structure similar to a welder's helmet. A latching knob in the rear of the helmet is used to adjust the diameter of the headband. The shield of the helmet is easily removed and is not used in the design.

The LCD module is mounted under the front of the helmet using L-brackets. A glossy black plastic plate, used as a mirror, is attached to the front edge of the LCD module. A Fresnel lens extends from the rear edge of the LCD module to the bottom of the plastic plate. Black antistatic plastic is used to shroud the entire face of the helmet, serving to block outside light.

Two connectors are mounted on the outside of the helmet. One connector accepts a 5-pin DIN plug that feeds power from an external supply. The other connector accepts a DB-9 plug that feeds video from the G-Node board. Two circuit modules are mounted on the inside-front of the helmet. One module is the Sharp DC/AC converter for the LCD backlight. The other module contains support circuitry for the LCD module, including two thumb-screw potentiometers for brightness and contrast control. Figures 6.3 and 6.4 show the front and top views, respectively, of the HMD.

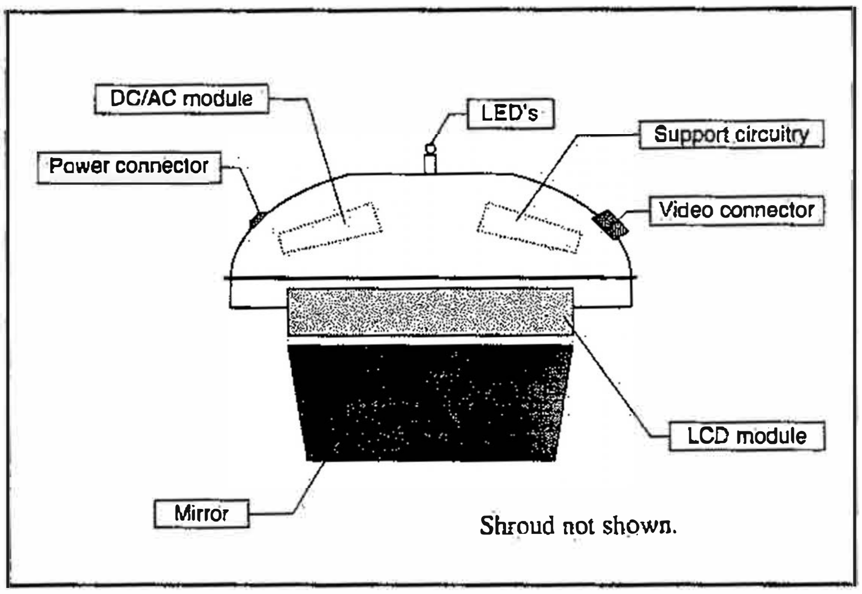


Figure 6.3. HMD design, front view.

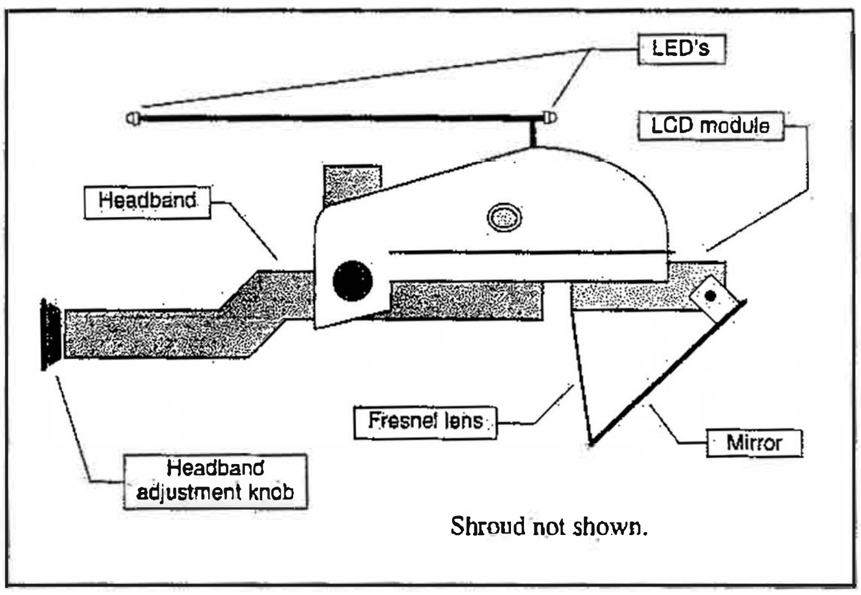


Figure 6.4. HMD design, side view.

7. APPLICATION SOFTWARE

This chapter describes the operation of the application software that runs on the host CPU. The application software defines the interaction between the user and the objects in the virtual world. The application can be considered to be separate from the graphics pipeline, although in the current implementation these two software pieces are combined into a single program.

The current application running on the VRS allows the user to explore an animated landscape. The objects appearing in the landscape are loaded from text definition files at run-time. Each text file defines a single object by listing a three-dimensional position, a diameter, and a color for each sphere making up that object.

The application program is implemented in C++, an object-oriented language. Each object in the virtual world is represented by an instance of a class called *CppObj*, which consists of the file name of that object's text definition file, an array of spheres, and a transformation matrix. When a *CppObj* instance is defined, the file name is passed to the constructor. Currently, the following methods are available in the *CppObj* class:

- load( ) – Parses the text definition file and loads the data into the sphere array. It is only called when the object is first created.
- moveTo(*x*, *y*, *z*) – Translates the object to the absolute position (x, y, z) in world coordinates.
- bounce(*v*) – Causes the object to bounce along the y-dimension, given initial velocity v, assuming a gravity acceleration of 9.8 m/s^2.
- spin(*T*) – Causes the object to spin around its own y-axis with a period of T.
- becomeSound(*n*) – Causes the object to emit sound from virtual channel n.

All motion in the virtual world is based on the real-time clock of the host computer, as opposed to the frame number. This ensures the fluid motion of objects even when the frame rate varies.

8. SUGGESTIONS FOR FUTURE WORK

Much of the potential of the VRS has yet to be tapped. There are a number of improvements that could be made, some of which have already been started by students in the Advanced Digital Systems Laboratory. These include:

- writing a more interactive application program. In the current application, the individual objects of the virtual world do not respond to the actions of the user.
- replacing the current audio system, which consists of the EVB and an interface expansion card, with a single expansion card. This card would have a DSP, memory, and four channels of 16-bit audio input and output.
- enhancing the sound spatialization software. The number of virtual sound channels could easily be expanded beyond the current limit of one. Also, Moore [5] discusses a number of spatialization methods that, combined with intensity panning, could provide a more convincing two-dimensional sound.
- adding advanced input devices, such as arm tracking, to the system. For example, we would like the user to be able to reach out and grab objects in the virtual world.
- moving more of the graphics pipeline stages to the G-Node. As discussed in Section 3.3, there are some stages currently performed by the host CPU that the G-Node could handle more efficiently. This would increase the graphics performance of the VRS.
- enhancing the lighting model of the sphere-based graphics. Instead of always drawing the highlight in the same position on the spheres, the highlight position could be calculated based on the position of the user and a virtual light source. This idea could then be extended to allow for multiple light sources.

LIST OF REFERENCES

[1] H. von Foerster, *Observing Systems.* Seaside, CA: Intersystems Publications, 1981.

[2] K. E. Drexler, G. Pergamit, and C. Peterson, *Unbounding the Future: The Nanotechnology Revolution.* New York, NY: Morrow, 1991.

[3] D. Hearn and M. P. Baker, *Computer Graphics.* Englewood Cliffs, NJ: Prentice-Hall, 1986.

[4] D. F. DeMenthon and L. S. Davis, "Model-Based Object Pose in 25 Lines of Code," *Proceedings of the Image Understanding Workshop*, Jan. 1992, pp. 753-761.

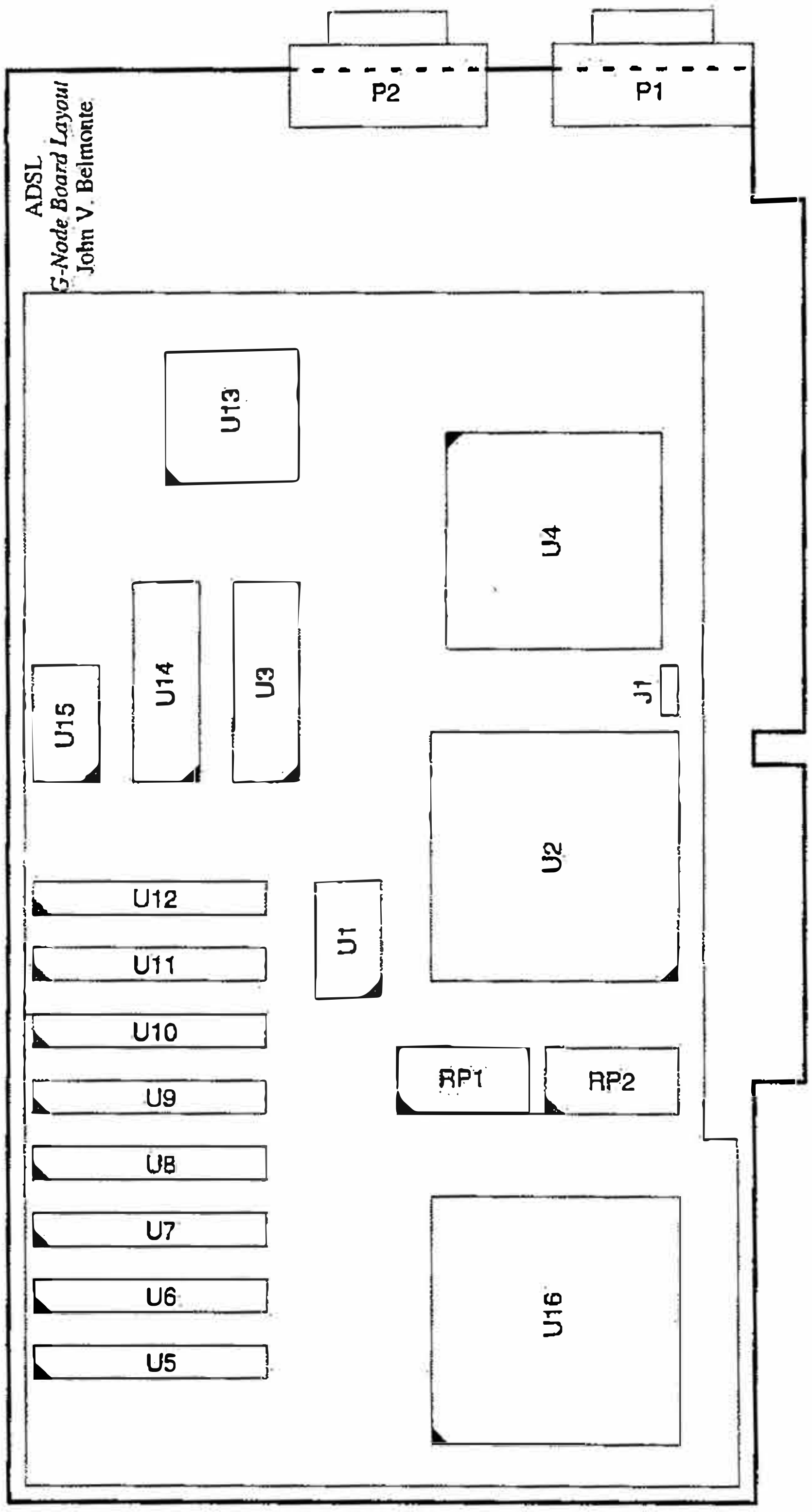
[5] F. R. Moore, *Elements of Computer Music.* Englewood Cliffs, NJ: Prentice-Hall, 1990.

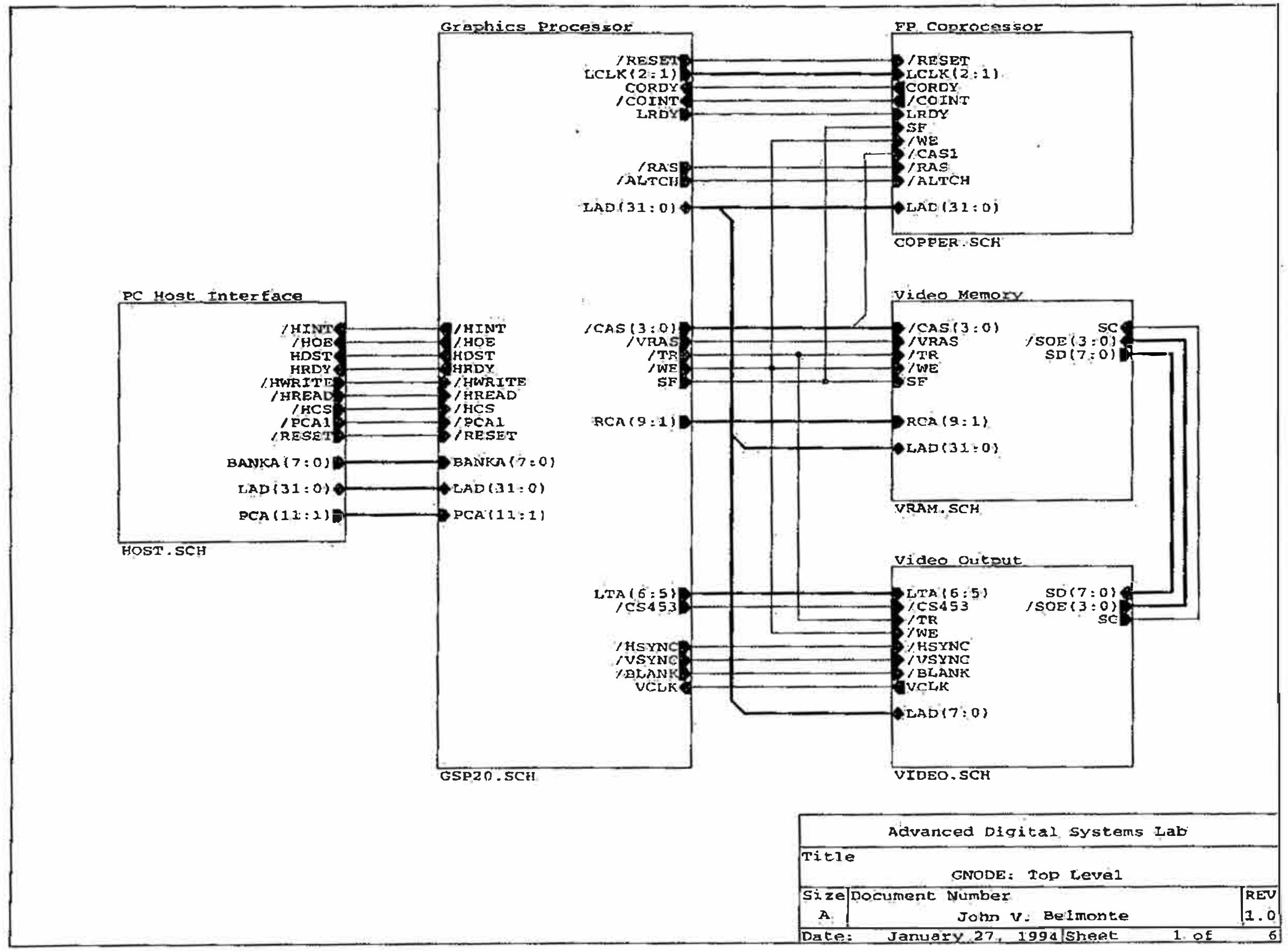
APPENDIX A. G-NODE DOCUMENTATION

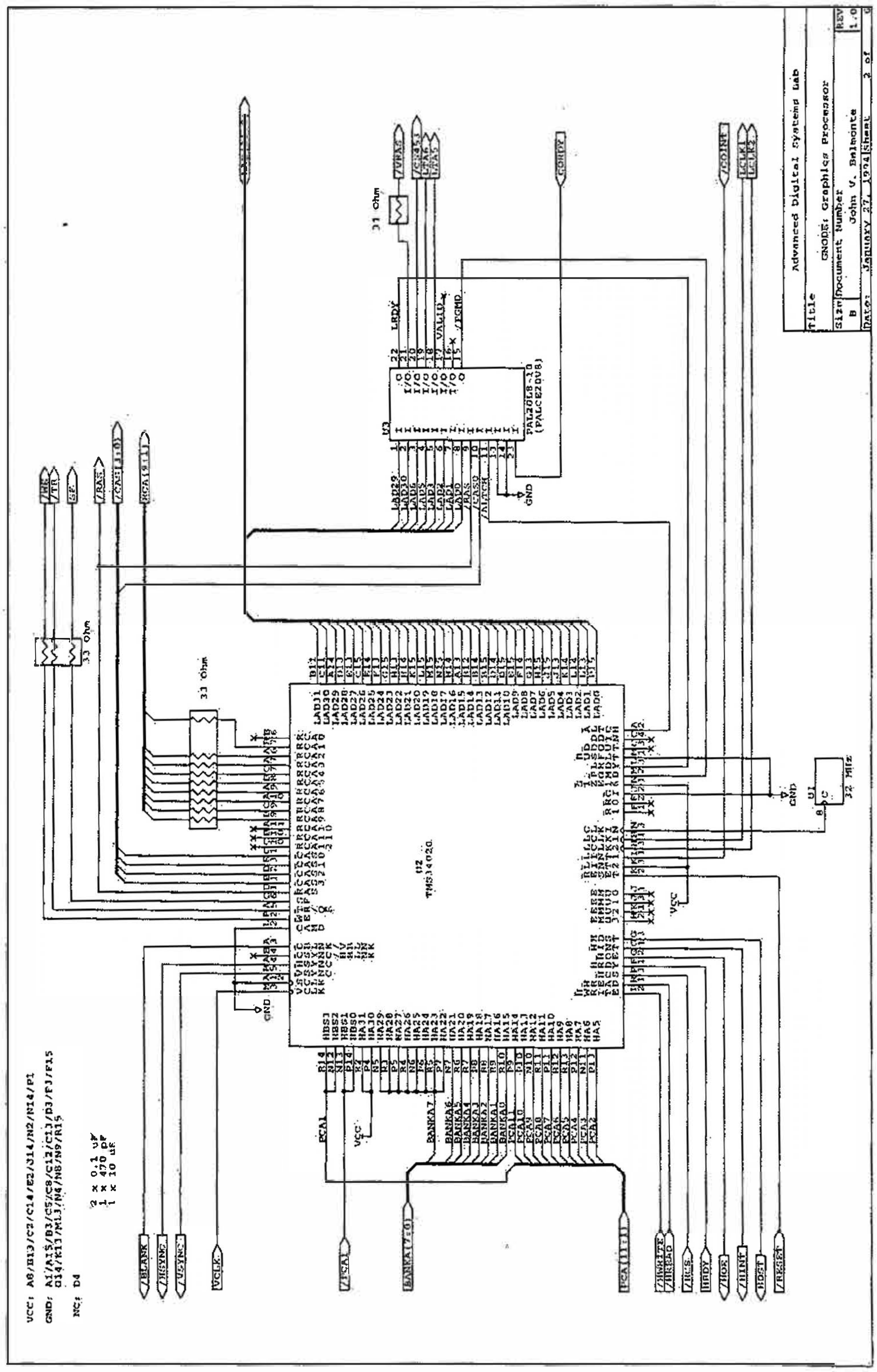
A.1. Schematics and Programmable Logic

The following items describing the G-Node hardware are included in this section:

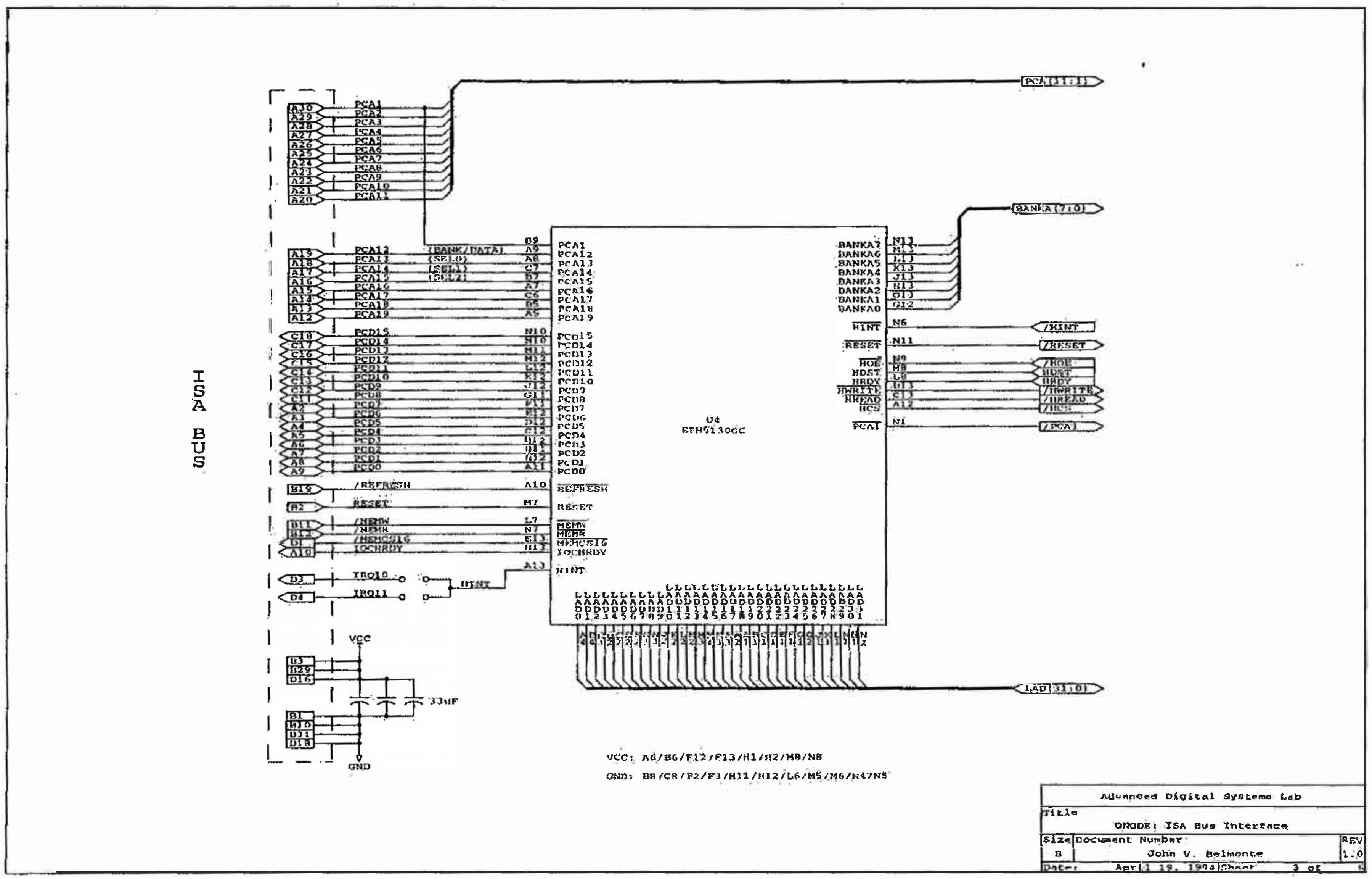
- Circuit board layout diagram (p. 38)
- Circuit board schematic diagrams (pp. 39-44)
- Text file describing programmable logic device (PLD) number U3 (p. 45)
- Schematic diagram describing PLD number U14 (p. 46)
- Schematic diagrams and text file describing PLD number U4 (pp. 47-50)

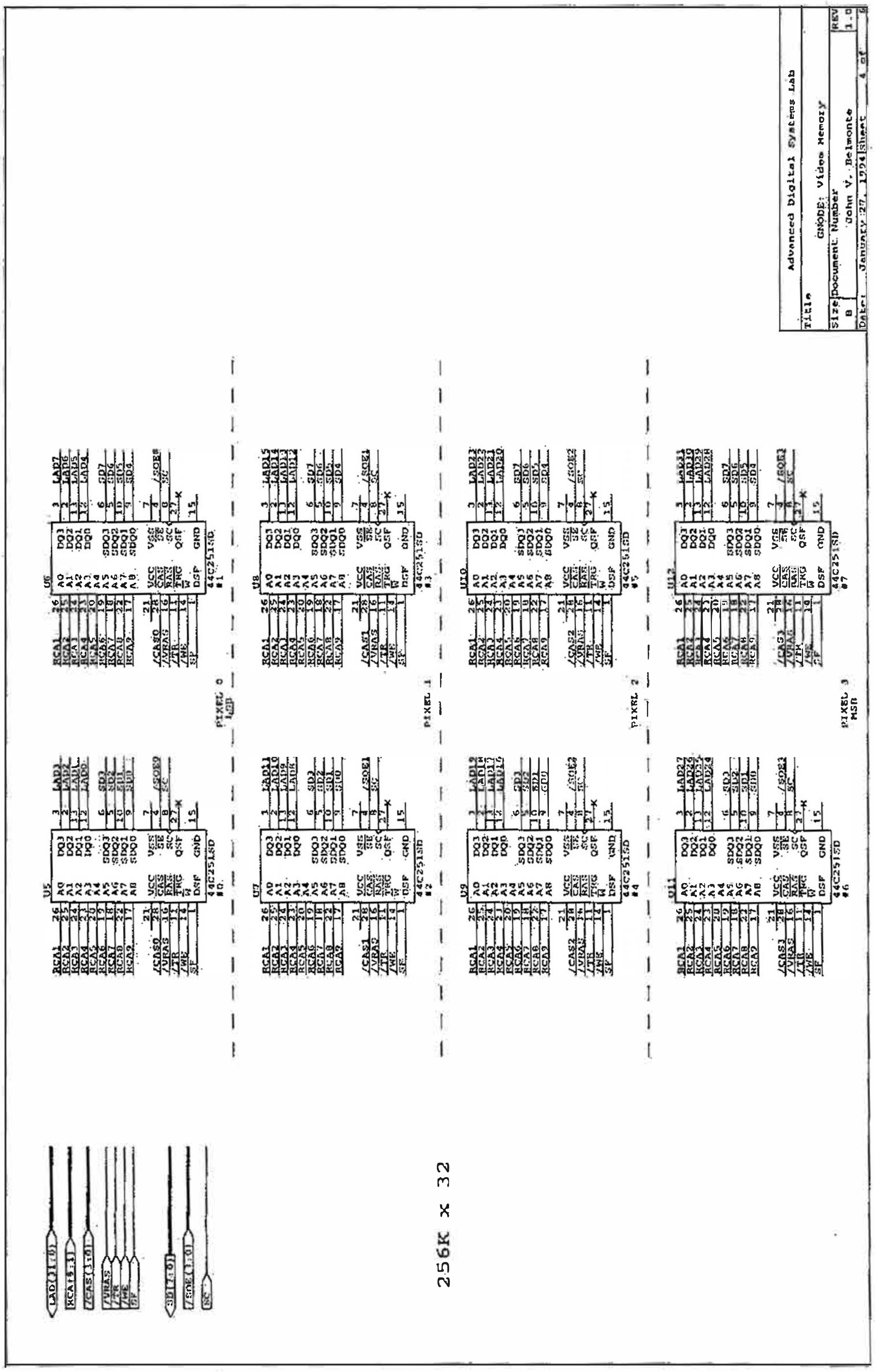

ADSL
G-Node Board Layout
John V. Belmonte
P2
P1
U13
U4
U15
U14
U3
J1
U2
U12
U11
U10
U9
U8
U7
U6
U5
U1
RP1
RP2
U16


PC Host Interface
/HINT
/HOE
HDST
HRDY
/HWRITE
/HREAD
/HCS
/PCA1
/RESET
BANKA(7:0)
LAD(31:0)
PCA(11:1)
HOST.SCH
Graphics Processor
/RESET
LCLK(2:1)
CORDY
/COINT
LRDY
/RAS
/ALTCH
LAD(31:0)
/HINT
/HOE
HDST
HRDY
/HWRITE
/HREAD
/HCS
/PCA1
/RESET
BANKA(7:0)
LAD(31:0)
PCA(11:1)
/CAS(3:0)
/VRAS
/TR
/WE
SF
RCA(9:1)
LTA(6:5)
/CS453
/HSYNC
/VSYNC
/BLANK
VCLK
GSP20.SCH
FP Coprocessor
/RESET
LCLK(2:1)
CORDY
/COINT
LRDY
SF
/WE
/CAS1
/RAS
/ALTCH
LAD(31:0)
COPPER.SCH
Video Memory
/CAS(3:0)
/VRAS
/TR
/WE
SF
RCA(9:1)
LAD(31:0)
SC
/SOE(3:0)
SD(7:0)
VRAM.SCH
Video Output
LTA(6:5)
/CS453
/TR
/WE
/HSYNC
/VSYNC
/BLANK
VCLK
LAD(7:0)
SD(7:0)
/SOE(3:0)
SC
VIDEO.SCH
Advanced Digital Systems Lab
Title
GNODE: Top Level
Size
A
Document Number
John V. Belmonte
REV
1.0
Date: January 27, 1994
Sheet 1 of 6

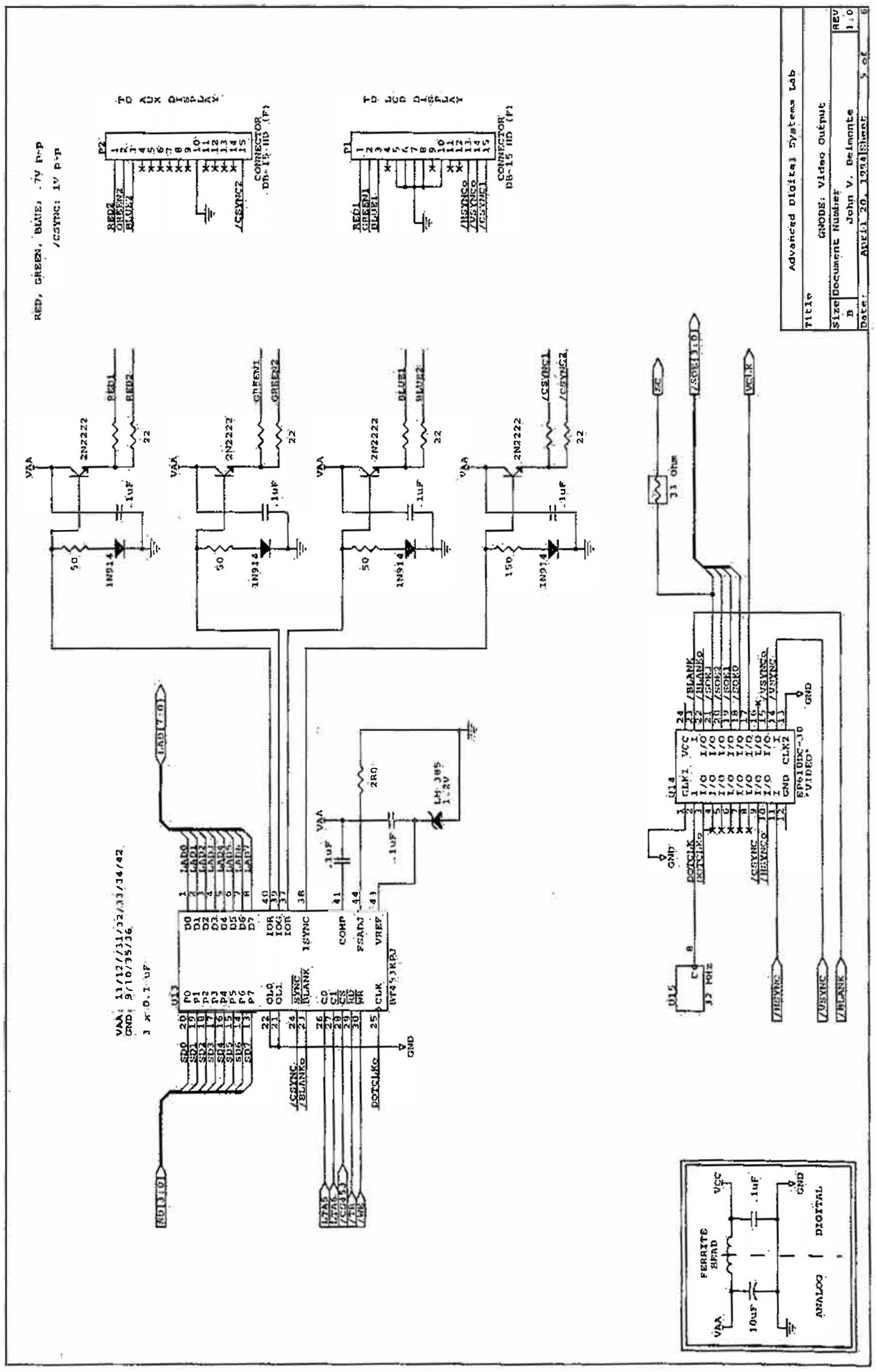


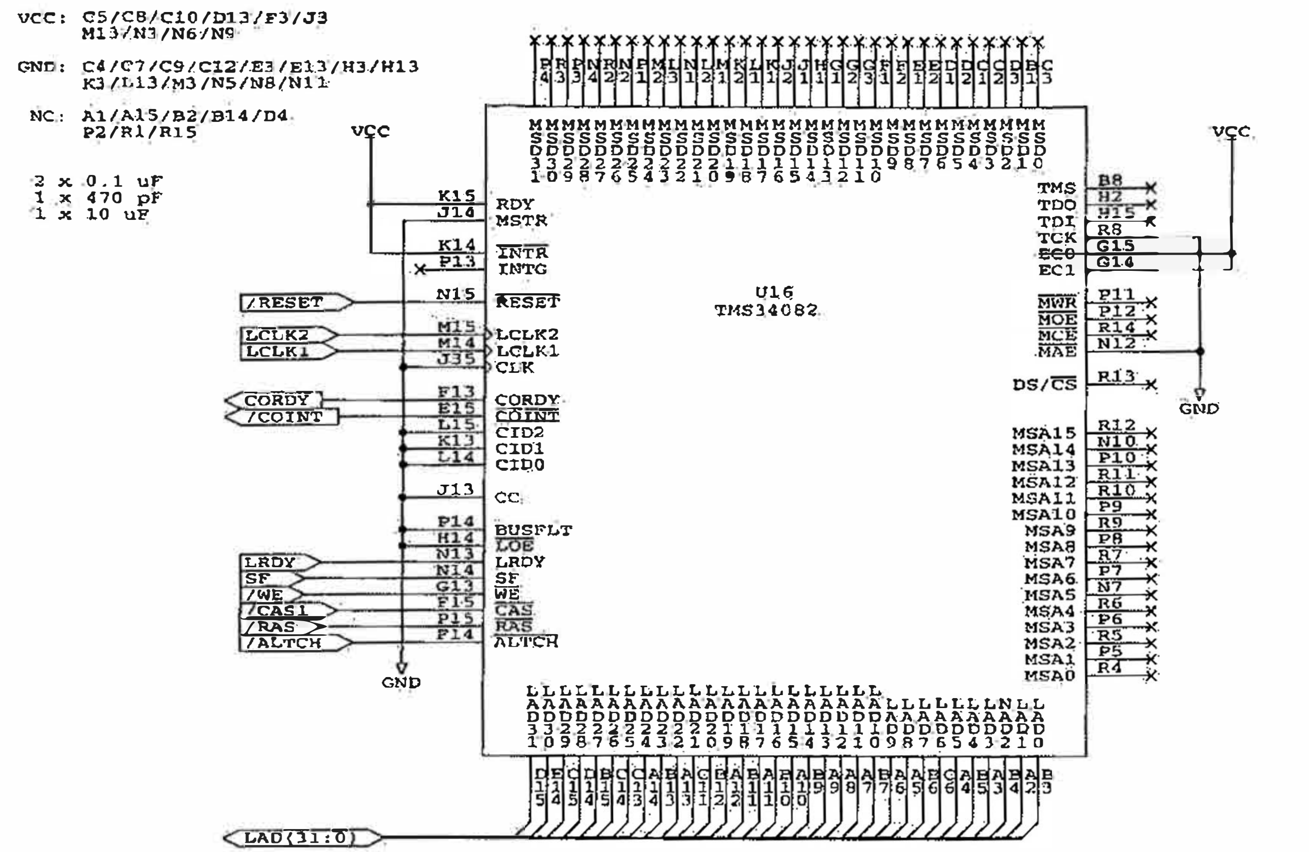
Advanced Digital Systems Lab
Title
GNODE: Graphics Processor
Size B
Document Number
John V. Belmonte
REV 1.0
Date: January 27, 1994
Sheet 2 of
U2
TMS34020
U3
PAL20L8-10
(PALCE20V8)
U1
32 MHz
33 Ohm
VCC: A8/B13/C2/C14/E2/J14/N2/N14/P1
GND: A1/A15/B3/C5/C8/C12/C13/D3/F3/F15
G14/K13/M13/N4/N8/N9/R15
NC: D4
2 x 0.1 uF
1 x 470 pF
1 x 10 uF
/BLANK
/HSYNC
/VSYNC
VCLK
/PCA1
BANKA(7:0)
PCA(11:1)
/HWRITE
/HREAD
/HCS
HRDY
/HOE
/HINT
HOST
/RESET
LCLK1
LCLK2
/COINT
CONDY
/VRAS
GND


ISA BUS
U4
EPM5130GC
VCC: A6/B6/F12/F13/H1/H2/M8/N8
GND: B8/C8/F2/F3/H11/H12/L6/M5/M6/N4/N5
PCA[31:1]
BANKA[7:0]
LAD[31:0]
33uF
VCC
GND
Advanced Digital Systems Lab
Title
GNODE: ISA Bus Interface
Size Document Number REV
B John V. Belmonte 1.0
Date: April 19, 1994 Sheet 3 of



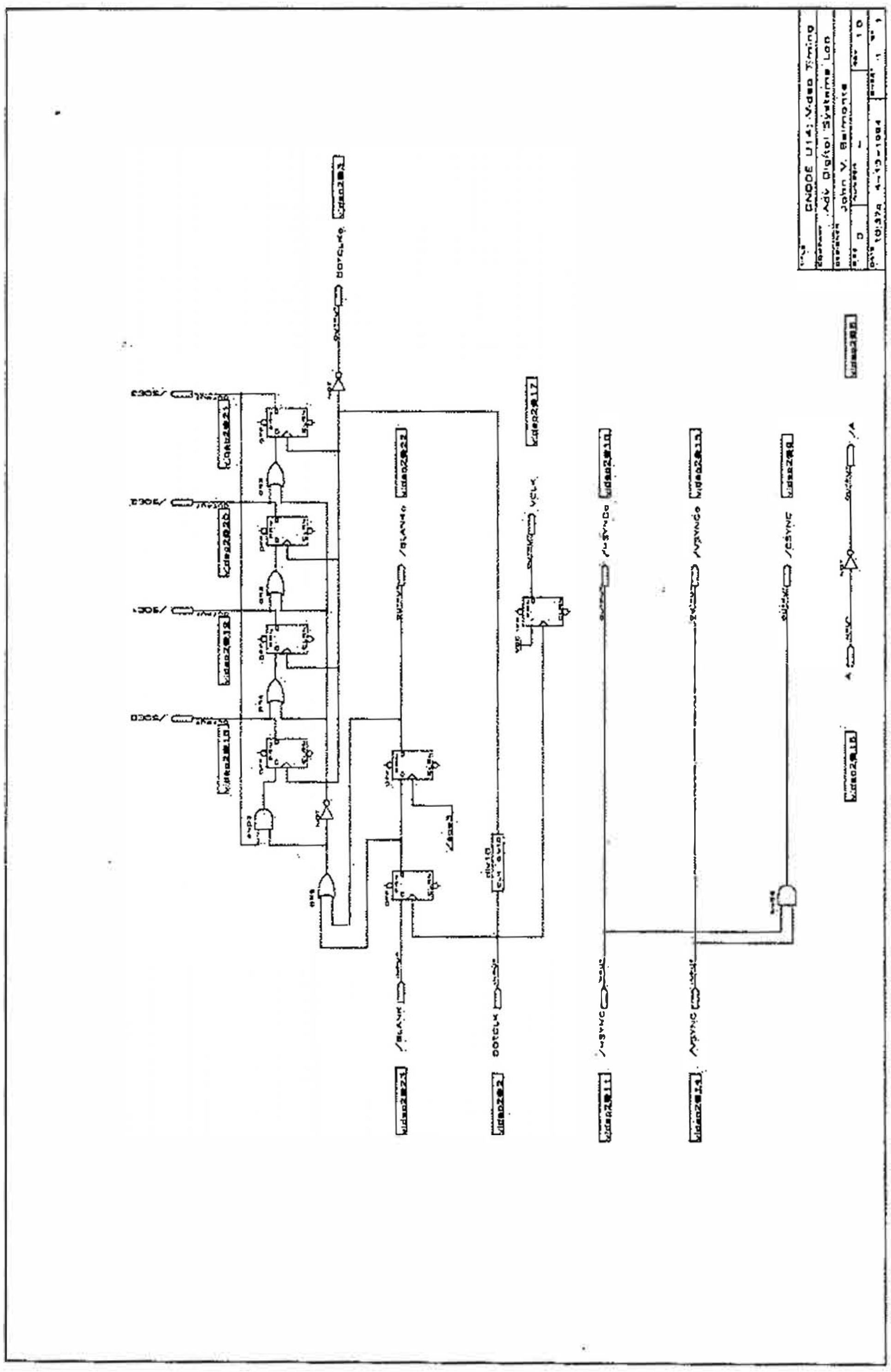
256K x 32
PIXEL 0 LSB
PIXEL 1
PIXEL 2
PIXEL 3 MSB
Advanced Digital Systems Lab
Title
GNODE: Video Memory
Size Document Number
B
John V. Belmonte
REV 1.0
Date: January 27, 1994 Sheet 4 of

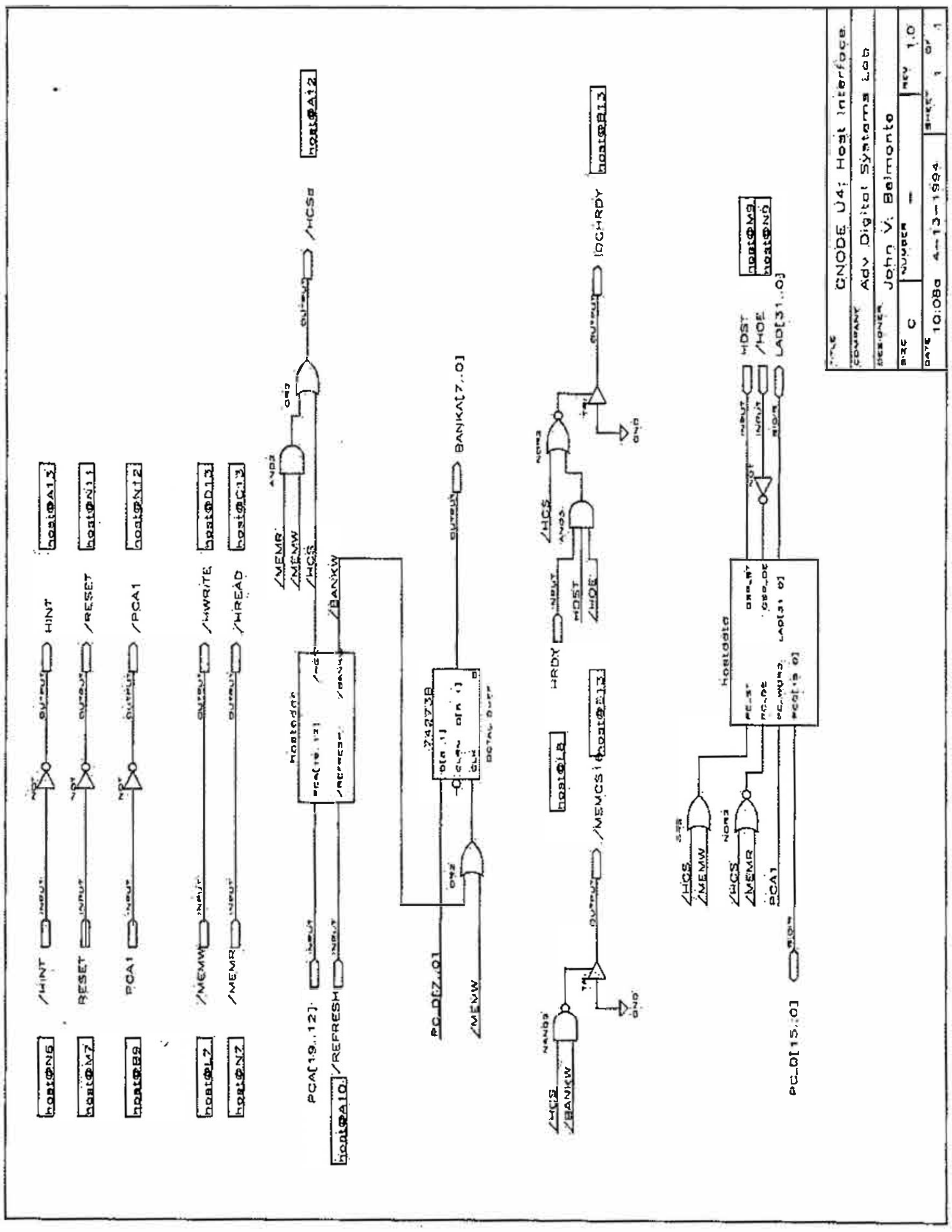

Advanced Digital Systems Lab
Title
GNODE: Video Output
Size
B
Document Number
John V. Delmonte
REV
1.0
Date:
April 20, 1994
Sheet 5 of 6
RED, GREEN, BLUE: .7V p-p
/CSYNC: 1V p-p
TO ADX DISPLAY
TO VGA DISPLAY
CONNECTOR DB-15 HD (F)
2N2222
1N914
FERRITE BEAD
ANALOG
DIGITAL



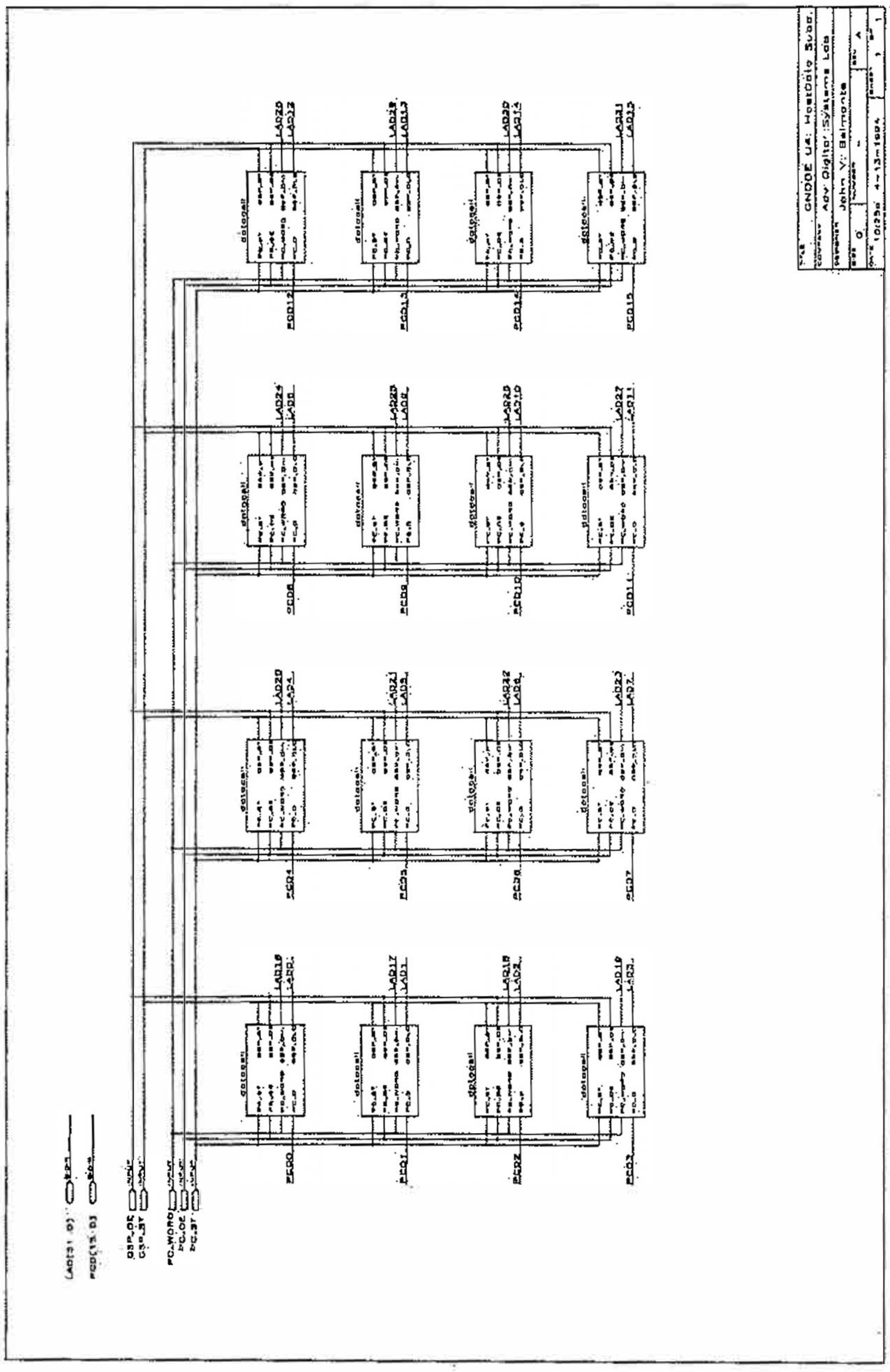
Advanced Digital Systems Lab		
Title: GNODE: Floating Point Coprocessor		
Size: A	Document Number: John V. Belmonte	REV: 1.0
Date: January 27, 1994	Sheet 6 of 6	

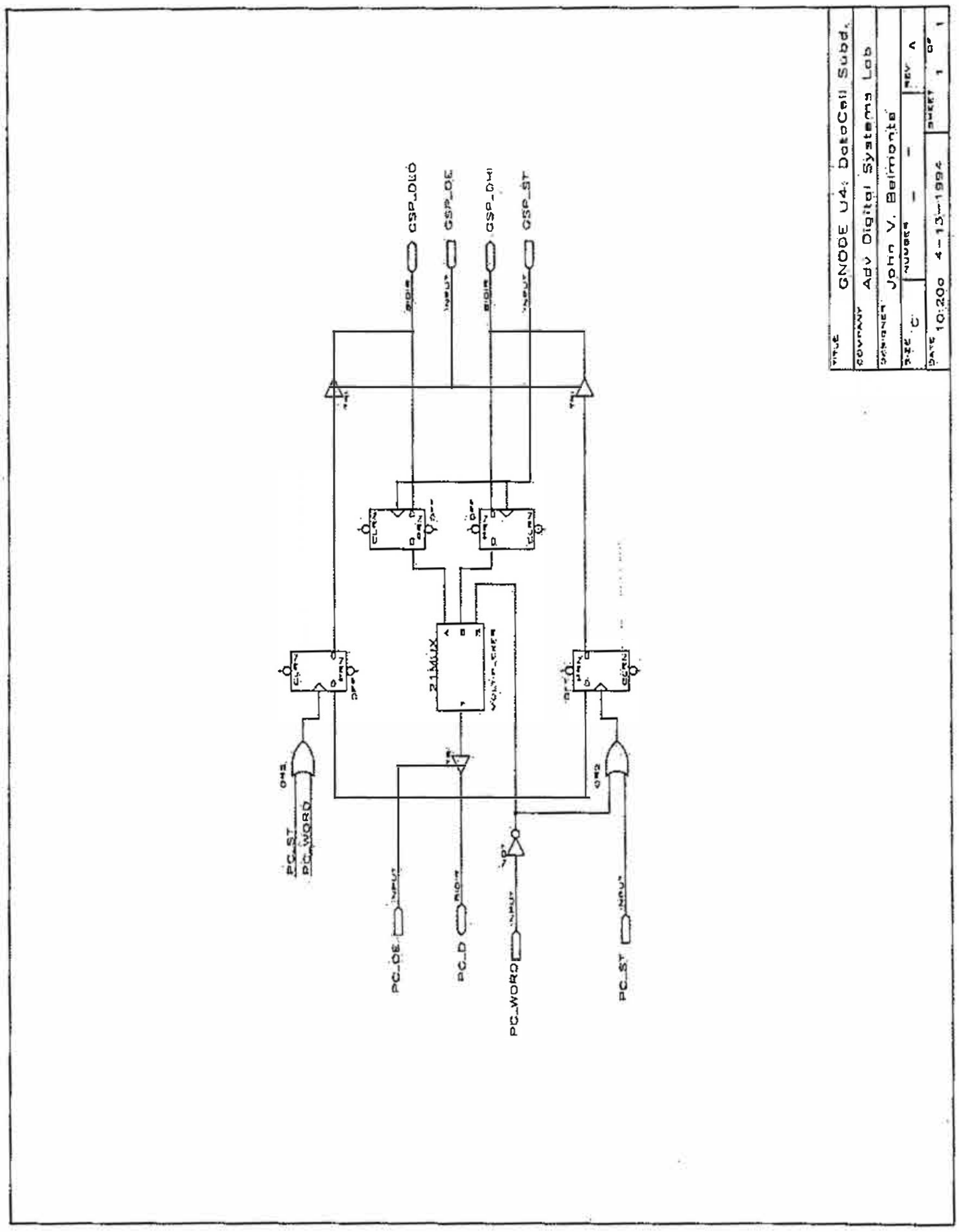
```
;PALASM Design Description

;------------------------------------ Declaration Segment ------------
TITLE    GNODE Local Address Decode, U3
PATTERN
REVISION 0.5
AUTHOR   John V. Belmonte
COMPANY  ADSL
DATE     10/29/92

CHIP  _decode  PALCE20V8

;---------------------------------- PIN Declarations ---------------
PIN  1          LAD29
PIN  2          LAD30
PIN  3          LAD6
PIN  4          LAD5
PIN  5          LAD3
PIN  6          LAD2
PIN  7          LAD1
PIN  8          LAD0
PIN  9          /RAS
PIN  10         /CAS0
PIN  11         /ALTCH
PIN  12         GND
PIN  15         /PGMD
PIN  16         /RATE
PIN  17         /VALID
PIN  18         LTA5
PIN  19         LTA6
PIN  20         /CS453
PIN  21         /VRAS
PIN  22         LRDY
PIN  23         CORDY
PIN  24         VCC

STRING   VRAM      '( LAD30 *  LAD29)'
STRING   RAMDAC    '(/LAD30 * /LAD29)'
STRING   PULSE     '(/LAD30 *  LAD29)'
STRING   REFRESH   '(/LAD3 * /LAD2 *  LAD1 *  LAD0)'
STRING   COLOR     '(/LAD3 *  LAD2 *  LAD1 *  LAD0)'

; note: 34020 I/O registers are at (LAD30 * /LAD29)

;---------------------------------- Boolean Equation Segment ------
EQUATIONS

VALID = ALTCH * /RAS +
        VALID * ( VRAM * /VRAS +
                  REFRESH * /VRAS +
                  COLOR * /VRAS +
                  RAMDAC * /REFRESH * /COLOR * /CS453 )

VRAS = VALID * VRAM * RAS +
       VALID * REFRESH * RAS +
       VALID * COLOR * RAS +
       VRAS * RAS

CS453 = VALID * RAMDAC * /REFRESH * /COLOR * RAS +
        CS453 * RAS

RATE  = VALID * PULSE  * /REFRESH * /COLOR * RAS +
        RATE * RAS

/LTA6 = /(LAD6 * /ALTCH +
          LTA6 *  ALTCH  )

/LTA5 = /(LAD5 * /ALTCH +
          LTA5 *  ALTCH  )

PGMD = VRAS
LRDY = CORDY

; /CAS0 can be used for wait state generation

; During VRAM Color-register load cycle, the 34020 puts out an all-zero
; address.  Since VRAM is in high memory the 'COLOR' status code must be used.
```


CNODE U14: Video Timing
Adv Digital Systems Lab
John V. Belmonte
/BLANK
DOTCLK
/VSYNC
/HSYNC
VCLK
/CSYNC
DOTCLK0


GNODE U4: Host Interface
Adv Digital Systems Lab
John V. Belmonte
10:08a 4-13-1994

```
% HOSTADDR subdesign of G-Node U4 Host Interface PLD %
% Advanced Digital Systems Lab                        %
% John V. Belmonte                                    %

SUBDESIGN HOSTADDR
(
    PCA[19..12]         :INPUT;
    /REFRESH            :INPUT;
    /HCS, /BANKW        :OUTPUT;
)

VARIABLE
    /PCA17,/PCA[15..12] :NODE;
    HCS, BANKW          :NODE;

BEGIN
    /HCS = !HCS;
    /BANKW = !BANKW;
    /PCA17 = !PCA17;
    /PCA[15..12] = !PCA[15..12];

    HCS =   /REFRESH &                                  % $D0XXX    %
            PCA19 & PCA18 & /PCA17 & PCA16 &            % 1101      %
            /PCA15 & /PCA14 & /PCA13 & /PCA12;          % 0000      %

    BANKW = /REFRESH &                                  % $D1XXX    %
            PCA19 & PCA18 & /PCA17 & PCA16 &            % 1101      %
            /PCA15 & /PCA14 & /PCA13 & PCA12;           % 0001      %
END;
```


GNDOE U4: HostData Subs.
Adv Digital Systems Lab
John V. Belmonte
4-13-1994



GNODE U4: DataCell Subd.
Adv Digital Systems Lab
John V. Belmonte
10:200 4-13-1994
GSP_DLO
GSP_DE
GSP_DHI
GSP_ST
21MUX
PC_ST
PC_WORD
PC_DE
PC_D
PC_WORD
PC_ST

A.2. Software

The following items describing VRS software running on the G-Node are included in this section:

- C language source code which implements the G-Node's graphics pipeline stages (*comq.c*, pp. 52-53)

- C language header which defines the communication structure between the host and the G-Node (*com.h*, p. 54)

```
/* comq.c
   Communication interface between host computer and VNODE
   John Belmonte
   2/10/93

   ==============================================================

   2/10  This one adds a software zbuffer.
         Added headroom meter.
   2/11  Now a negative diameter means no highlight.
   2/12  Color 16 is "transparent"
         Added object count display.
         Added primitive xy clipping based on center of sphere
   2/13  Added Z_HOLD to fix bug caused by host changing Z's during qsort()
         Added diameter clamping.
   2/14  Added horizon.
   2/15  Improved clipping
   2/24  Removed headroom meter -- useless
         Added timer bar and game_over()
*/


#include <stdlib.h>   /* qsort() */
#include <gspreg.h>
#include <gsptypes.h>
#include <gspglobs.h>
#include "vr.h"
#include "com.h"
#include "colors.h"

typedef enum { FIELDSIZE = 1 } BIT;

/* prototypes */
int ltoa( long n, char *buffer );
void draw_point( short x, short y );
void fill_oval( short w, short h, short xleft, short ytop );
void draw_line( short x1, short y1, short x2, short y2 );
void draw_oval( short w, short h, short xleft, short ytop );
short cpw( short x, short y );
PTR set_vector( unsigned short trapnum, PTR gptr );
ulong peek_breg( short breg );
void poke_breg( short breg, ulong val );


/* globals */
int disp=0;     /* current display page */
int draw=1;     /* current draw page    */
int facReady;


void horizon( ulong topcolor, ulong bottomcolor, unsigned long y ) {
   ulong fcolor, bcolor;

   get_colors(&fcolor, &bcolor);

   /* top area */
   poke_breg( COLOR1, topcolor );
   asm( " rpix B9 " );                            /* replicate value COLOR1 */
   poke_breg( DYDX, (y << 16) + 160 );            /* x, y area              */
   poke_breg( DADDR, peek_breg( OFFSET ) );       /* start address          */
   asm( " vlcol " );
   asm( " vfill L " );        /* fill area */

   /* bottom area */
   poke_breg( COLOR1, bottomcolor );
   asm( " rpix B9 " );
   poke_breg( DYDX, ((234-y) << 16) + 160 );
   poke_breg( DADDR, peek_breg( DADDR ) + 768 );
   asm( " vlcol " );
   asm( " vfill L " );

   set_colors( fcolor, bcolor );
}

/* Display Interrupt service routine */
void c_int10() {

   horizon( BLACK, BROWN, *(COMBASE + HORIZON) );

   *(ushort *)0x0 = 0;  /* pulse frame rate */

   *(BIT *)DIE = 0;   /* disable display interrupt */
   *(BIT *)DIP = 0;   /* clear display interrupt pending */
   facReady = TRUE;
}


/* initialize the communication structure */
void init_com() {

   *(COMBASE + OBJ_COUNT) = 0;
   *(COMBASE + HORIZON) = 117;
   *(COMBASE + TIMEOUT) = 1;
}


void init_graph() {
   int index;

   set_config( MODE, 1 );

   set_vector( 10, (PTR)c_int10 );

   /* set IE bit in ST register */
   asm(" GETST A8 ");
   asm(" ORI   (1<<21), A8 ");
   asm(" PUTST A8" );

   flipAndClear( disp, draw );
   while( !facReady ) {};
   flipAndClear( disp, draw );
   while( !facReady ) {};

}


void flipAndClear(short display, short draw) {

   facReady = FALSE;
   *(ushort *)DPYINT = *(ushort *)VSBLNK20;

   /* Schedule page flip to occur at bottom of screen. */
   poke_breg(OFFSET, page[draw].BaseAddr);
   *(ulong *)DPYST20 = page[display].DpyStart;

   *(BIT *)DIP = 0;  /* clear display interrupt pending */
   *(BIT *)DIE = 1;  /* enable display interrupt */
}


void render( word * object ) {
   int x, y, z, dia, color, highlight;

   x = *(object + X_POS);
   y = *(object + Y_POS);
   dia = (short)*(object + DIA);
   color = *(object + COLOR);

   highlight = (dia > 0 );
   dia = abs(dia);
   if( dia > 4*160 ) {
      dia = 4*150;
      highlight = FALSE;
   }

   set_fcolor( color );

   if( color != 16 ) {
      set_fcolor( color );
```

```
        fill_oval( dia >> 1, dia, x-(dia >> 2), y-(dia >> 1) );
    } else { /* "transparent */
        set_fcolor( LIGHT_GRAY );
        draw_oval( dia >> 1, dia, x-(dia >> 2), y-(dia >> 1) );
    }

    if( highlight ) {
        set_fcolor( WHITE );
        fill_oval( dia >> 3, dia >> 2,
                   x - (dia >> 4) + (dia >> 3), y - (dia >> 2) - (dia >> 3) );
    }

                                 /* note:  - (x >> 4) + (x >> 3) = (x >> 4) */
}


int sort_function( const short **obj1, const short **obj2 ) {

    if( *(*obj1 + Z_HOLD) >  *(*obj2 + Z_HOLD) ) return (-1);
    if( *(*obj1 + Z_HOLD) <  *(*obj2 + Z_HOLD) ) return ( 1);
    if( *(*obj1 + Z_HOLD) == *(*obj2 + Z_HOLD) ) return ( 0);
}


void game_over() {
    static char s1[] = "TIME IS UP!";
    int width, left;
    int x, y;
    static int color = 0;

    width = text_width( s1 );
    left = 80 - width/2;

    set_fcolor( BLACK );
    fill_rect( width + 25, 75, left - 12, 50 - 12 );

    set_fcolor( color++ );
    text_out( left, 50, s1 );

    for( y = 0; y < 14; y++ )
        for( x = 0; x < width; x++ )
            put_pixel( get_pixel(left + x, y+50), left + width-x, y+75 );

}


word * zarray[500];  /* max 500 objects! */

main() {
    int i;
    vword *object, *maxobject;
    vword *object_base;
    int oc;
    int index;
    short x, y, dia;
    char s[10];
    int cpwval;
    int timeleft;

    init_com();
    init_graph();

    object_base = COMBASE + OBJ_START;

    while( 1 ) {

      oc = *(COMBASE + OBJ_COUNT);

      if( oc > 0 ) {
        maxobject = object_base + OBJ_SIZE*oc;

        index = 0;
        for( object = object_base; object < maxobject; object += OBJ_SIZE ) {

          /* to clip: don't put object in sort array */

            x = (short)*(object + X_POS);
            y = (short)*(object + Y_POS);
            dia = ((short)*(object + DIA ));
            dia = abs(dia);

            *(object + Z_HOLD) = *(object + Z_POS);

            cpwval = cpw(x, y);

            if( cpwval == 0 ) {
              zarray[index++] = (word *)object;
            } else {
              x += (cpwval & 1) ? dia >> 2 : 0;
              x -= (cpwval & 2) ? dia >> 2 : 0;
              y += (cpwval & 4) ? dia >> 1 : 0;
              y -= (cpwval & 8) ? dia >> 1 : 0;
              if( cpw(x, y) == 0 )
                zarray[index++] = (word *)object;
            }
        }

        itoa( index, s );

        if( index > 0 )
            qsort(zarray, index, 4, sort_function);

        while( !facReady ) {};

        for( ; index > 0; )
            render( zarray[--index] );


        /* object count */
        set_fcolor( LIGHT_GRAY );
        text_out( 0, 220, s );

        /* timer bar */
        timeleft = *(COMBASE + TIMEOUT);
        set_fcolor( LIGHT_GREEN );
        draw_line( 0, 233, timeleft, 233 );
        draw_line( 0, 232, timeleft, 232 );
        set_fcolor( LIGHT_RED );
        draw_line( timeleft, 233, 159, 233 );
        draw_line( timeleft, 232, 159, 232 );
        set_fcolor( BLACK );
        draw_line( 0, 231, 159, 231 );

        if( timeleft == 0 ) game_over();    /* display timeout message */

        flipAndClear(disp ^= 1, draw ^= 1);
      }
    }
}
```

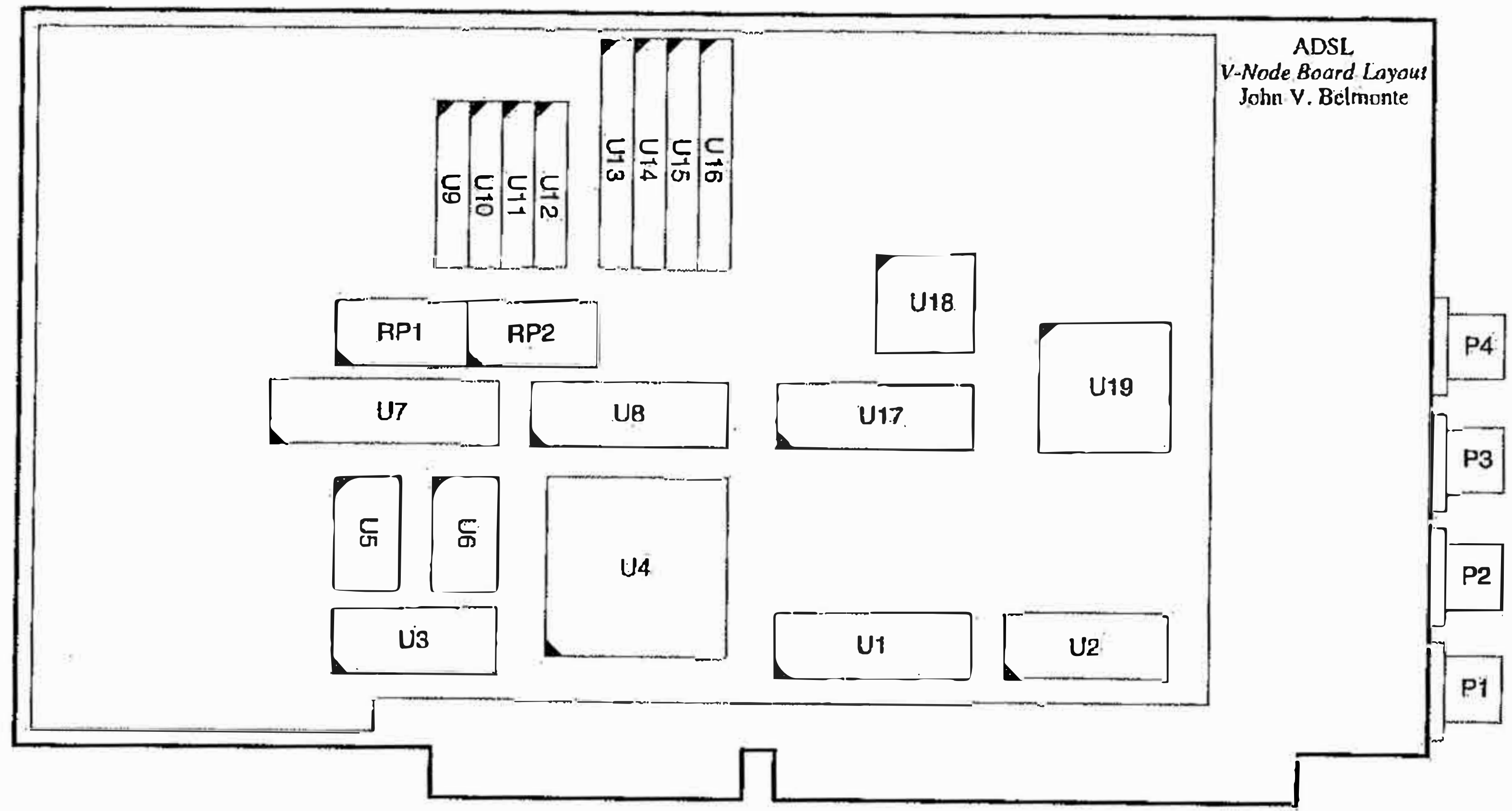
```
/* com.h
   34020 SIDE
   Defines communication interface
   John Belmonte
   2/9/93
*/


#define  COMBASE   ((vword *)0xFFE00000)   /* this is where it all begins */
#define  COMEND    ((vword *)0xFFF00000)   /* 128K for now                */


/* All remaining values are  *
 * in units of 16-BIT WORDS! */


/********************** base structure  ******************************/

#define  OBJ_COUNT   0              /* number of objects in the list */
#define  HORIZON     1              /* y position of horizon         */
#define  TIMEOUT     2              /* user time left                */
#define  OBJ_START   4              /* start of object list          */


/********************** object structure ******************************/

#define  OBJ_SIZE    16             /* this should be power of 2     */

/*  offsets */
#define  X_POS       0
#define  Y_POS       1
#define  Z_POS       2
#define  DIA         3
#define  COLOR       4
#define  Z_HOLD      5
```

APPENDIX B. V-NODE DOCUMENTATION

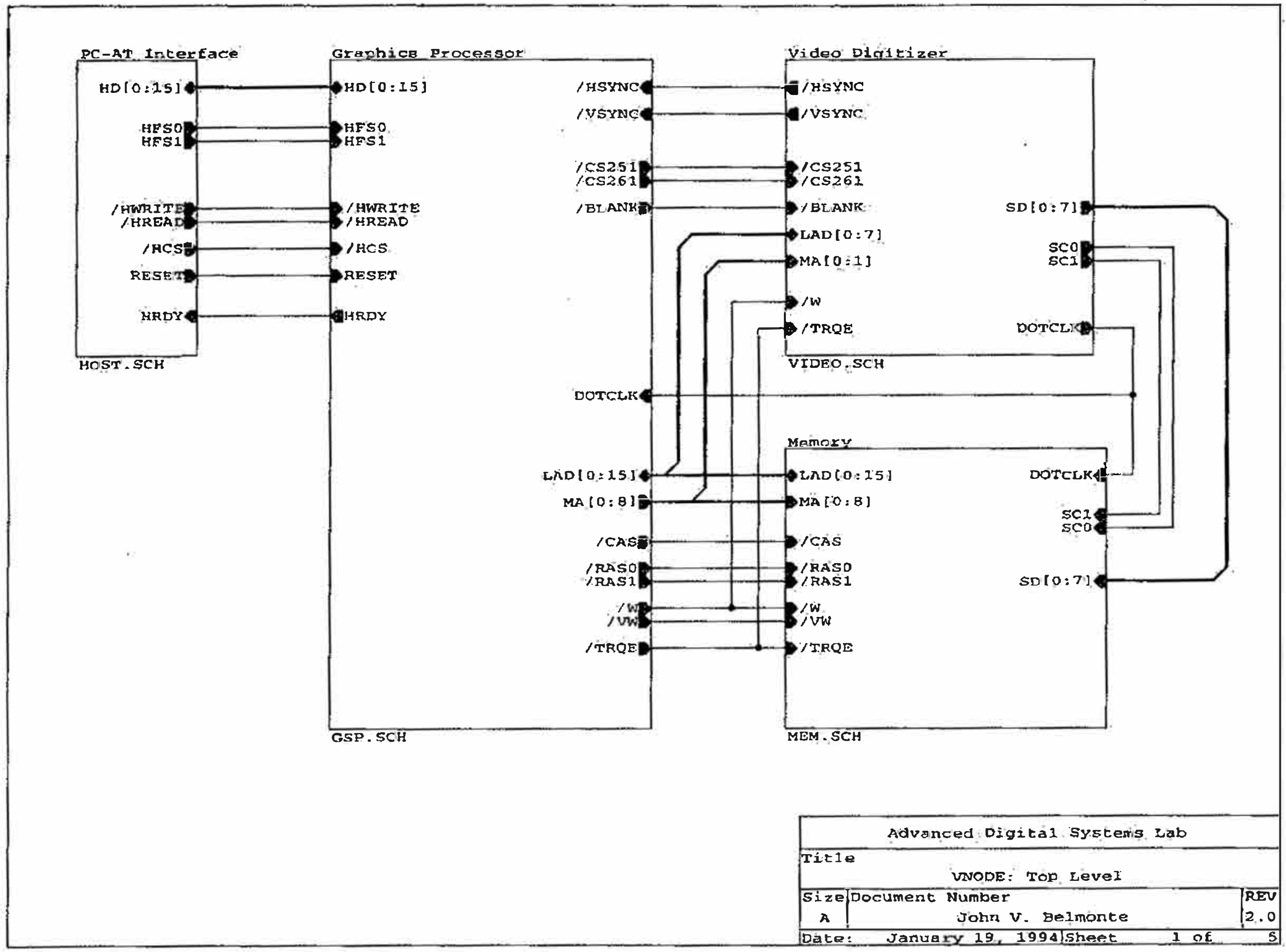
B.1. Schematics and Programmable Logic

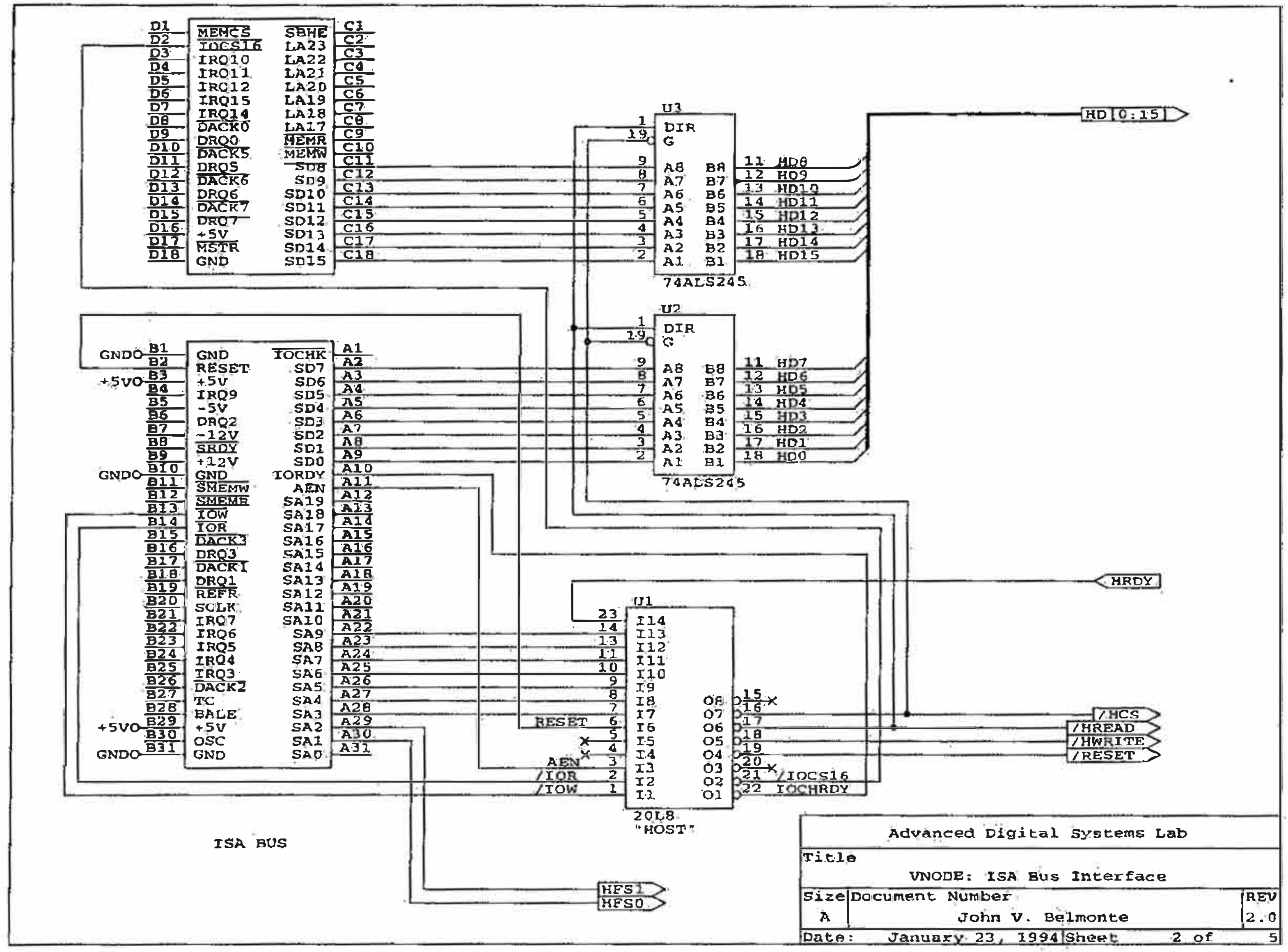
The following items describing the V-Node hardware are included in this section:

- Circuit board layout diagram (p. 56)
- Circuit board schematic diagrams (pp. 57-61)
- Text file describing PLD number U1 (p. 62)
- Text file describing PLD number U8 (p. 63)
- Text file describing PLD number U17 (p. 64)
- Schematic diagram describing PLD number U7 (p. 65)

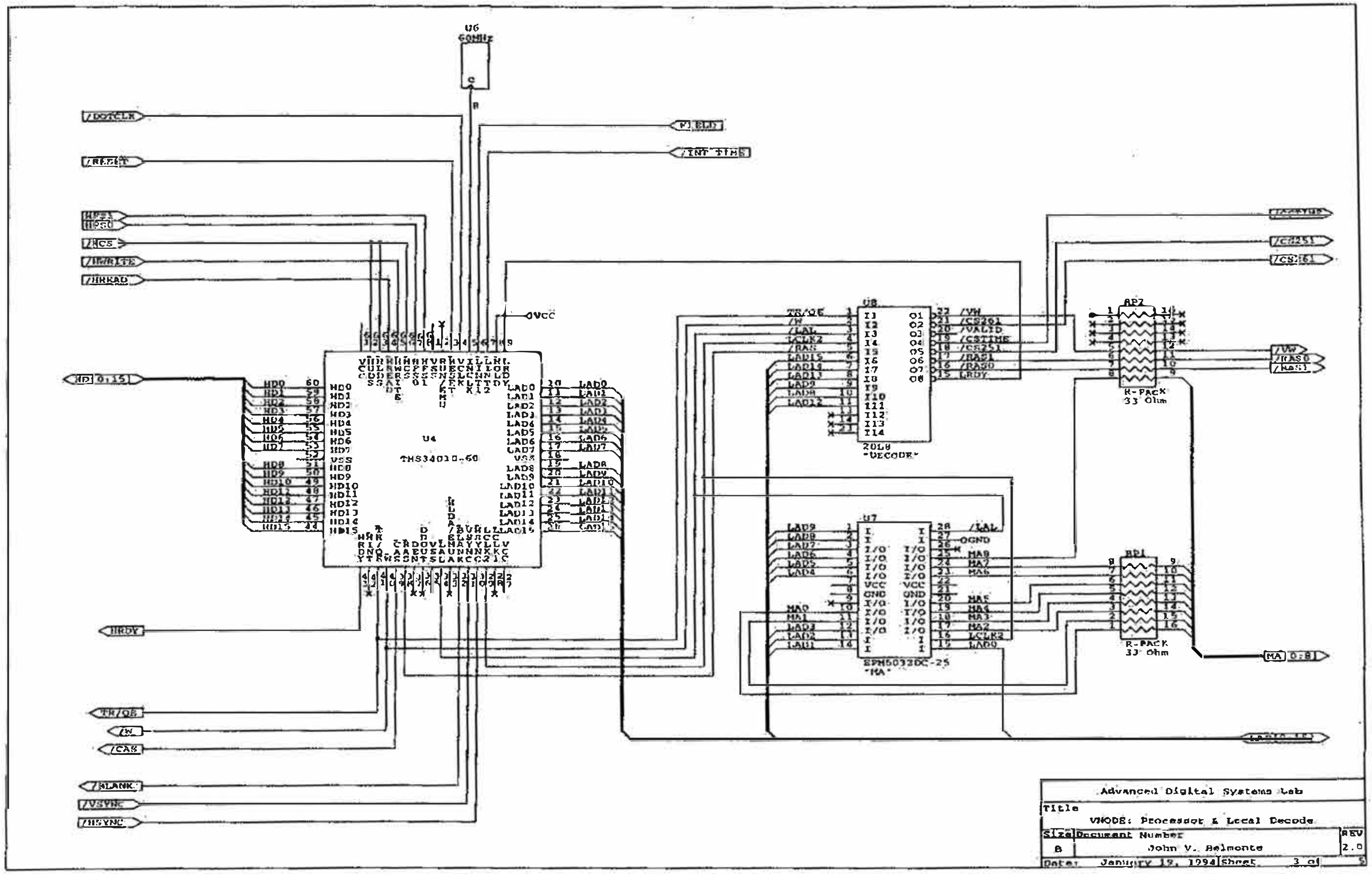


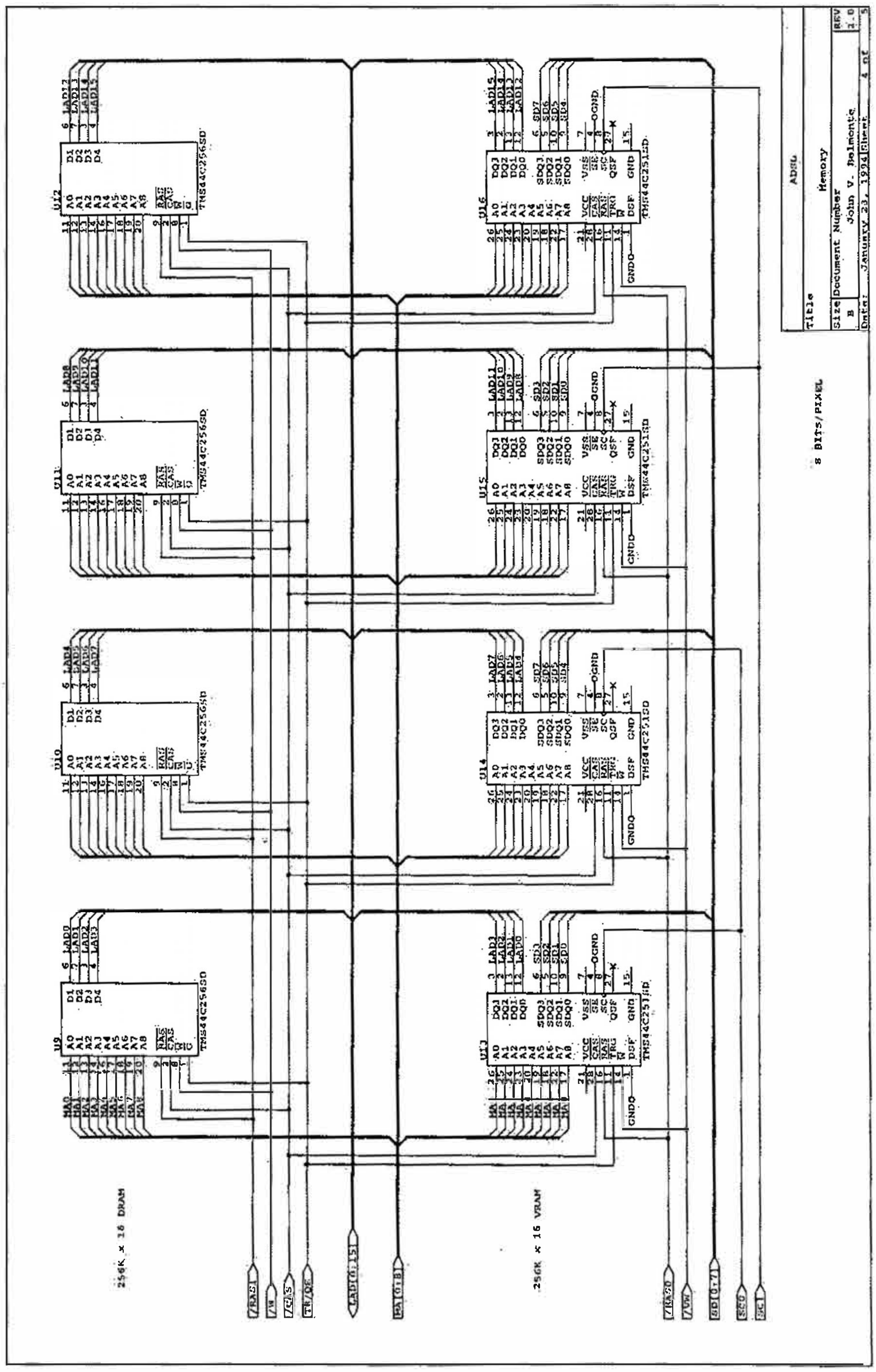
ADSL
V-Node Board Layout
John V. Belmonte
U9
U10
U11
U12
U13
U14
U15
U16
U18
RP1
RP2
U19
U7
U8
U17
U5
U6
U4
U3
U1
U2
P4
P3
P2
P1

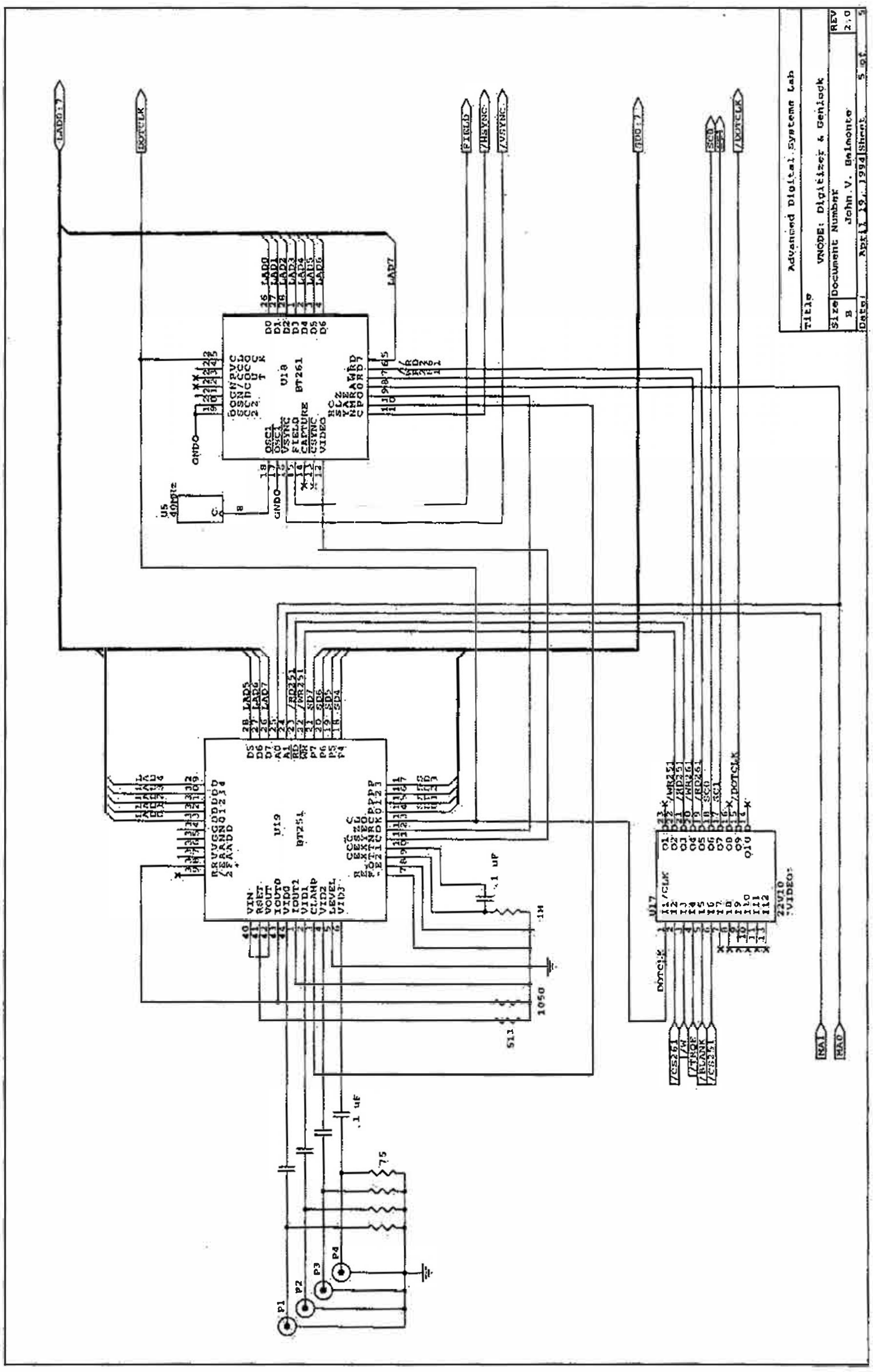

PC-AT Interface
HD[0:15]
HFS0
HFS1
/HWRITE
/HREAD
/HCS
RESET
HRDY
HOST.SCH
Graphics Processor
HD[0:15]
HFS0
HFS1
/HWRITE
/HREAD
/HCS
RESET
HRDY
/HSYNC
/VSYNC
/CS251
/CS261
/BLANK
DOTCLK
LAD[0:15]
MA[0:8]
/CAS
/RAS0
/RAS1
/W
/VW
/TRQE
GSP.SCH
Video Digitizer
/HSYNC
/VSYNC
/CS251
/CS261
/BLANK
LAD[0:7]
MA[0:1]
/W
/TRQE
SD[0:7]
SC0
SC1
DOTCLK
VIDEO.SCH
Memory
LAD[0:15]
MA[0:8]
/CAS
/RAS0
/RAS1
/W
/VW
/TRQE
DOTCLK
SC1
SC0
SD[0:7]
MEM.SCH
Advanced Digital Systems Lab
Title
VNODE: Top Level
Size
A
Document Number
John V. Belmonte
REV
2.0
Date: January 19, 1994
Sheet 1 of 5



ISA BUS
U3
74ALS245
U2
74ALS245
U1
20L8
"HOST"
HD[0:15]
HRDY
/HCS
/HREAD
/HWRITE
/RESET
HFS1
HFS0
RESET
AEN
/IOR
/IOW
/IOCS16
IOCHRDY
Advanced Digital Systems Lab
Title
VNODE: ISA Bus Interface
Size Document Number
A
John V. Belmonte
REV
2.0
Date: January 23, 1994 Sheet 2 of 5


U6
60MHz
/DOTCLK
/RESET
/HCS
/HWRITE
/HREAD
HD[0:15]
VCC
U4
TMS34010-60
HRDY
TR/QE
/W
/CAS
/BLANK
/VSYNC
/HSYNC
U8
20L8
"DECODE"
U7
EPM5032DC-25
"MA"
RP2
R-PACK
33 Ohm
RP1
R-PACK
33 Ohm
/VW
/RAS0
/RAS1
MA[0:8]
Advanced Digital Systems Lab
Title
VNODE: Processor & Local Decode
Size
B
Document Number
John V. Belmonte
REV
2.0
Date: January 19, 1994
Sheet 3 of 5

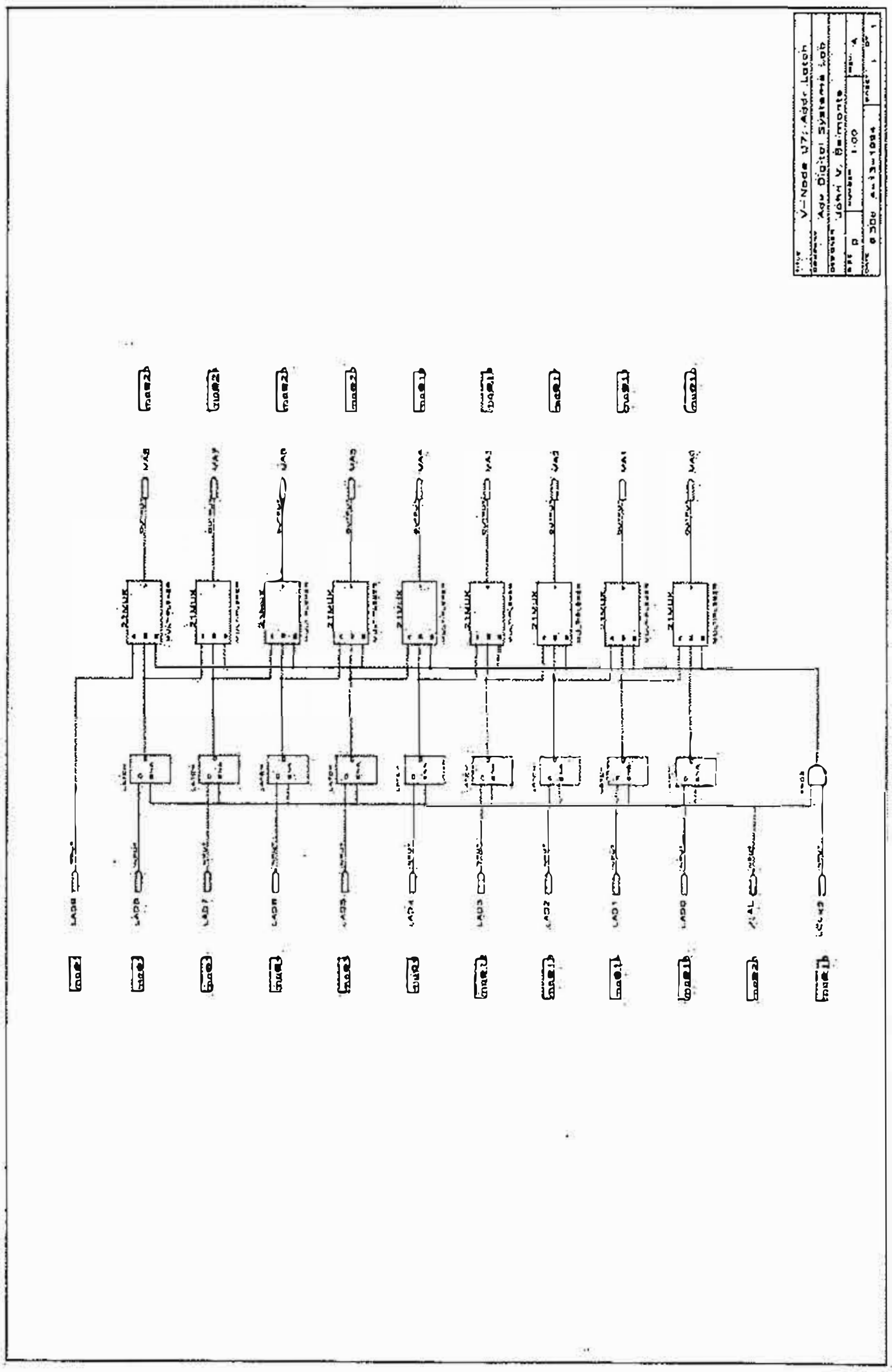

ADSL
Title
Memory
Size
Document Number
B
John V. Belmonte
Date:
January 23, 1994
Sheet
REV
8 BITS/PIXEL
256K x 16 DRAM
256K x 16 VRAM
TMS44C256SD
TMS44C251SD
U9
U10
U11
U12
U13
U14
U15
U16


Advanced Digital Systems Lab
VNODE: Digitizer & Genlock
John V. Belmonte
April 19, 1994
REV 2.0

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    VNODE, U1
PATTERN
REVISION 1.0
AUTHOR   John V. Belmonte
COMPANY
DATE     12/14/91

CHIP  _u1  PAL20L8

;---------------------------------- PIN Declarations ---------------
PIN  1          /IOW                                          ;
PIN  2          /IOR                                          ;
PIN  3          AEN                                           ;
PIN  6          RESETin                                       ;
PIN  7          A3                                            ;
PIN  8          A4                                            ;
PIN  9          A5                                            ;
PIN  10         A6                                            ;
PIN  11         A7                                            ;
PIN  12         GND                                           ;
PIN  13         A8                                            ;
PIN  14         A9                                            ;
PIN  16         /HCS                             COMBINATORIAL ;
PIN  17         /HREAD                           COMBINATORIAL ;
PIN  18         /HWRITE                          COMBINATORIAL ;
PIN  19         /RESET                           COMBINATORIAL ;
PIN  21         /IOCS16                          COMBINATORIAL ;
PIN  22         IOCHRDY                          COMBINATORIAL ;
PIN  23         HRDY                             COMBINATORIAL ;
PIN  24         VCC                                           ;

; DECODE:
;               $300    HSTADRL
;               $302    HSTADRH
;               $304    HSTDATA
;               $306    HSTCTL

;----------------------------------- Boolean Equation Segment ------
EQUATIONS

HCS = /AEN*A9*A8*/A7*/A6*/A5*/A4*/A3*(IOR+IOW)

/IOCHRDY = 1
IOCHRDY.TRST = /HRDY        ; enable when host not ready

IOCS16 = 1
IOCS16.TRST = /AEN*A9*A8*/A7*/A6*/A5*/A4*/A3    ; enable when selected

HWRITE = IOW
HREAD = IOR

RESET = RESETin

;----------------------------------- Simulation Segment ------------
SIMULATION

;-------------------------------------------------------------------
```

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    VNODE. U8
PATTERN
REVISION 1.0
AUTHOR   John V. Belmonte
COMPANY
DATE     1/22/92. revised 9/3/92

CHIP  _u8  PAL20L8

;---------------------------------- PIN Declarations ---------------
PIN  1          /TR
PIN  2          /W
PIN  3          /LAL
PIN  4          LCLK2
PIN  5          /RAS
PIN  6          /RF       ; LAD15
PIN  7          LA26      ; LAD14
PIN  8          LA25      ; LAD13
PIN  11         LA24      ; LAD12
PIN  12         GND
PIN  15         LRDY                               COMBINATORIAL
PIN  16         /RAS0                              COMBINATORIAL
PIN  17         /RAS1                              COMBINATORIAL
PIN  18         /CS251                             COMBINATORIAL
PIN  19         /CSTIME                            COMBINATORIAL
PIN  20         /VALID                             COMBINATORIAL
PIN  21         /CS261                             COMBINATORIAL
PIN  22         /VW                                COMBINATORIAL
PIN  24         VCC

;----------------------------------- Boolean Equation Segment ------
EQUATIONS

VALID = (/RAS * LCLK2) +                    ; SIGNALS ROW ADDRESS ON LAD
        (VALID * /RAS)

RAS0 = (VALID * RF * RAS) +                 ; ACTIVATE FOR REFRESH
       (VALID * /LA26 * /LA25 * /LA24 * RAS) +
                                            ; ACTIVATE FOR 000
       (RAS0 * RAS)                         ; KEEP ACTIVE UNTIL /RAS=1


RAS1 = (VALID * RF * RAS) +                 ; ACTIVATE FOR REFRESH
       (VALID * LA26 * LA25 * LA24 * RAS) +
                                            ; ACTIVATE FOR 111
       (RAS1 * RAS)                         ; KEEP ACTIVE UNTIL /RAS=1


CS251 = (VALID * RAS * /RF * /LA26 * LA25 * /LA24) +
                                                  ; ACTIVATE FOR 010
        (CS251 * (RAS + LAL))                     ; KEEP ACTIVE UNTIL RAS=LAL=0


CS261 = (VALID * RAS * /RF * LA26 * /LA25 * /LA24) +
                                                  ; ACTIVATE FOR 100
        (CS261 * (RAS + LAL))                     ; KEEP ACTIVE UNTIL RAS=LAL=0


CSTIME = (VALID * RAS * /RF * LA26 * LA25 * /LA24 ) +
                                                  ; ACTIVATE FOR 110
         (CSTIME * (RAS + LAL))                   ; UNTIL RAS=LAL=0


VW = /( /W * (LAL + /TR) )                        ; WRITE LINE FOR VRAM

/LRDY = /TR * /W * CSTIME                         ; ADD WAIT-STATE FOR TIMER CHIP
```

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    VMODE. U17
PATTERN
REVISION 1.0
AUTHOR   John V. Belmonte
COMPANY
DATE     2/2/92

CHIP  _u17  PAL22V10

;---------------------------------- PIN Declarations ---------------
PIN  1          DOTCLK
PIN  2          /CS261
PIN  3          /W
PIN  4          /TRQE
PIN  5          /BLANK
PIN  6          /CS251
PIN  12         GND
PIN  15         /DOTCLKo                          COMBINATORIAL ;
PIN  16         Y                                 REGISTERED;
PIN  17         SC1                               COMBINATORIAL ;
PIN  18         SC0                               COMBINATORIAL ;
PIN  19         /RD261                            COMBINATORIAL ;
PIN  20         /WR261                            COMBINATORIAL ;
PIN  21         /RD251                            COMBINATORIAL ;
PIN  22         /WR251                            COMBINATORIAL ;
PIN  24         VCC

;----------------------------------- Boolean Equation Segment ------
EQUATIONS

RD251 = CS251 * TRQE
WR251 = CS251 * W

RD261 = CS261 * TRQE
WR261 = CS261 * W

Y = /Y * /BLANK

SC0 = /Y * /BLANK
SC1 =  Y * /BLANK

DOTCLKo = DOTCLK

;----------------------------------- Simulation Segment ------------
SIMULATION

;-------------------------------------------------------------------
```


V-Node U7: Addr. Latch
Adv Digital Systems Lab
John V. Belmonte

B.2. Software

The following items describing VRS software running on the V-Node are included in this section:

- C language source code which initializes the video control registers of the GSP (*video.c*, p. 67)
- C language source code which initializes the video A/D converter (*bt251.c*, p. 68)
- C language source code which initializes the video genlock (*bt261.c*, pp. 69-70)
- C language source code which initializes the A/D's RAM lookup table (*th.c*, p. 71)
- C language source code which implements the LED tracking (*find2t.c*, pp. 72-73)

```
/* video.c
   This program initializes the video control registers of the J4010.
   version 2
   10/1/92


   Video Timing Registers
   ----------------------
   DPYADR    (counter)
   DPYCTL    ok
   DPYINT    no problem
   DPYSTRT   ok
   DPYTAP    ok
   HCOUNT    (counter)
   HEBLNK    ok
   HESYNC    good
   HSBLNK    ok
   HTOTAL    good
   VCOUNT    (counter)
   VEBLNK    ?????
   VESYNC    good
   VSBLNK    ????
   VTOTAL    good

*/

#include <stdlib.h>
#include "gspreg.h"

typedef unsigned short ushort;

#define WRITE( addr, val )  *(volatile ushort *)addr = val
#define READ( addr )   *(volatile ushort *)addr


main() {
   WRITE( DPYSTRT  , 0xFFFC ); /* LCSTRT = 00 (1 line per refresh) */
   WRITE( DPYTAP   , 0x0000 );

   /* horizontal */
   WRITE( HSBLNK   , 423-14 ); /* 9  before HSYNC */
   WRITE( HEBLNK   ,     55 ); /* 60 after HSYNC */
   WRITE( HTOTAL   , 0xFFFF ); /* must be greater than external HSYNC */
   WRITE( HESYNC   ,    232 ); /* should be average of HEBLNK and HSBLNK */

   /* vertical */
   WRITE( VSBLNK   ,  262-2 );
   WRITE( VEBLNK   ,      8 );
   WRITE( VESYNC   ,    130 ); /* should be average of VEBLNK and VSBLNK */
   WRITE( VTOTAL   , 0xFFFF ); /* must be greater than external VSYNC */

   WRITE( DPYCTL   , 0xD020 ); /* Display Control Register
                                  15    ENV      1
                                  14    NIL      1
                                  13    DXV      0
                                  12    SRE      1
                                  11    SRT      0
                                  10    ORG      0
                                  9:2   DUDATE   00001000  (inc 1024x8)
                                  1     Reserved 0
                                  0     HDS      0
                               */

   WRITE( INTENB, READ( INTENB ) & 0xFBFF );    /* disable DPY interrupt */
}
```

```c
/* bt251.c
   This 34010 program initializes the BT251 digitizer chip.
*/

#include <stdlib.h>
#include "gspreg.h"

typedef volatile unsigned short vus;

#define BT251_ADDR (*(vus*)0x02000000)
#define BT251_RAM  (*(vus*)0x02000010)
#define BT251_REG  (*(vus*)0x02000020)


main() {
   int i;

   BT251_ADDR = 0x00;    /* Command Register                                  */
   BT251_REG  = 0x00;    /* D7,D6 = input select  \_ 0 = VID0, 5 = VID1      */
                         /* D5,D4 = sync select   /  A = VID2, F = VID3      */
                         /* D3,D2 = 00,  50 mV sync slicing level            */
                         /*         01,  75 mV                               */
                         /*         10, 100 mV                               */
                         /*         11, 125 mV                               */
                         /* D1,D0 = 00                                       */

   BT251_ADDR = 0x01;    /* IOUT0 (REF+)      */
   BT251_REG  = 0xD0;    /* D7:D2 = current   */
                         /* D1,D0 = 00        */
                         /* 1.2V = $F8        */
                         /* 1.0V = $D0        */
                         /*  .8V = $A4        */

   BT251_ADDR = 0x02;    /* IOUT1 (REF-)      */
   BT251_REG  = 0x40;    /* D7:D2 = current   */
                         /* D1,D0 = 00        */
                         /*  .3V = $40        */

   BT251_ADDR = 0x00;

   for( i = 0; i < 256; i++ )     /* fill RAM using autoincrement */
       BT251_RAM = i;
}
```

```
/* bt261.c
   This 34010 program initializes the BT261 genlock chip.

   version 2
   10/2/92


   notes:
          CLAMP      OK (clamp during horizontal sync)
          ZERO       OK (zero during back porch)
          NOISEGAT   OK
          VSYNC      OK
          OSC        OK
          HSYNC      OK (4.7us)
          FIELD      ?
          HCOUNT     OK

          command registers
          ----------------- CR0 = 0x58 --------------------  good
          CR07-06  01    Reset counter upon recovered HSYNC
          CR05     d     Capture strobe - unused
          CR04-03  11    Slicing level = 125mV
          CR02-00  000   TTL OSC1 is clock input
          ----------------- CR1 = 0x82 --------------------
          CR17     1     Interlaced input
          CR16     0     Drive CLOCK output
          CR15     0     Drive CSYNC output
          CR14     0     Drive VSYNC output
          CR13     0     Drive HSYNC output
          CR12     d     Reset status bits
          CR11     1     Use internaly generated HSYNC
          CR10     d     enable phase limiting
          ----------------- CR2 = 0x88 --------------------
          CR27-24  1000  Phase lock pixel count
          CR23     1     stop pixel clock at HCOUNT
          CR22     d     lock override
          CR21-20  00    Pixel clock derived from OSC input
          ----------------- CR3 = 0x08 --------------------
          CR37-30  d     Phase lock line count
*/

#include <stdlib.h>

#define BT261_ADDR 0x04000000
#define BT261_DATA 0x04000010

#define CR0 0x00
#define CR1 0x01
#define CR2 0x02
#define CR3 0x03
#define VSYNC 0x04
#define OSCLO 0x05
#define OSCHI 0x06
#define STATUS 0x07
#define HSYNCGLO 0x08
#define HSYNCGHI 0x09
#define HSYNCSLO 0x0A
#define HSYNCSHI 0x0B
#define CLAMPGLO 0x0C
#define CLAMPGHI 0x0D
#define CLAMPSLO 0x0E
#define CLAMPSHI 0x0F
#define ZEROGLO  0x10
#define ZEROGHI  0x11
#define ZEROSLO  0x12
#define ZEROSHI  0x13
#define FIELDGLO 0x14
#define FIELDGHI 0x15
#define FIELDSLO 0x16
#define FIELDSHI 0x17
#define NOISEGLO 0x18
#define NOISEGHI 0x19
#define NOISESLO 0x1A
#define NOISESHI 0x1B
```

```
#define HCOUNTLO 0x1C
#define HCOUNTHI 0x1D


typedef unsigned short ushort;

#define WRITE( addr, val )  *(volatile ushort *)addr = val
#define READ( addr )  *(volatile ushort *)addr


main() {

   /* command registers          (must be initialized first) */
   WRITE( BT261_ADDR, CR0 );
   WRITE( BT261_DATA, 0xD8 ); /* 58 */
   WRITE( BT261_ADDR, CR1 );
   WRITE( BT261_DATA, 0x82 );
   WRITE( BT261_ADDR, CR2 );
   WRITE( BT261_DATA, 0x88 );


   /* oscillator */
   WRITE( BT261_ADDR, OSCLO );   /* derive 6.66MHz dotclock from 40MHz */
   WRITE( BT261_DATA, 0x06 );
   WRITE( BT261_ADDR, OSCHI );
   WRITE( BT261_DATA, 0x06 );


   /* HCOUNT                     (must be initialized early) */
   WRITE( BT261_ADDR, HCOUNTLO );
   WRITE( BT261_DATA, 0xFF );
   WRITE( BT261_ADDR, HCOUNTHI );
   WRITE( BT261_DATA, 0xFF );


   /* Noise Gate                 (must be initialized early) */
   WRITE( BT261_ADDR, NOISEGLO );
   WRITE( BT261_DATA,  195 );    /* NOISEG = HCOUNT/2 - 2.5uS */
   WRITE( BT261_ADDR, NOISEGHI );
   WRITE( BT261_DATA,    0 );
   WRITE( BT261_ADDR, NOISESLO );
   WRITE( BT261_DATA,    8 );    /* NOISES = >HCOUNT/2 */
   WRITE( BT261_ADDR, NOISESHI );
   WRITE( BT261_DATA,    1 );


   /* VSYNC */
   WRITE( BT261_ADDR, VSYNC );
   WRITE( BT261_DATA,  106 );      /* VSYNC = HCOUNT/4 */


   /* HSYNC */
   WRITE( BT261_ADDR, HSYNCSLO );
   WRITE( BT261_DATA,    0 );      /* negate HSYNC immediately upon CSYNC */
   WRITE( BT261_ADDR, HSYNCSHI );
   WRITE( BT261_DATA,    0 );
   WRITE( BT261_ADDR, HSYNCGLO );
   WRITE( BT261_DATA,   31 );      /* HSYNC length = 4.7 uS */
   WRITE( BT261_ADDR, HSYNCGHI );
   WRITE( BT261_DATA,    0 );


   /* clamp */
   WRITE( BT261_ADDR, CLAMPGLO );
   WRITE( BT261_DATA,    5 );      /* CLAMP duration 2uS */
   WRITE( BT261_ADDR, CLAMPGHI );
   WRITE( BT261_DATA,    0 );
   WRITE( BT261_ADDR, CLAMPSLO );
   WRITE( BT261_DATA,   18 );
   WRITE( BT261_ADDR, CLAMPSHI );
   WRITE( BT261_DATA,    0 );

   /* zero */
   WRITE( BT261_ADDR, ZEROGLO );
   WRITE( BT261_DATA,  0x95 );
```

```
    WRITE( BT261_ADDR, ZEROGHI );
    WRITE( BT261_DATA,    1 );
    WRITE( BT261_ADDR, ZEROSLO );
    WRITE( BT261_DATA,  0x97 );        /* ZERO duration 2 DOTCLOCK's */
    WRITE( BT261_ADDR, ZEROSHI );
    WRITE( BT261_DATA,    1 );

    /* field */
    WRITE( BT261_ADDR, FIELDGLO );
    WRITE( BT261_DATA,   89 );        /* field at 1/4 HCOUNT - 2.5us   */
    WRITE( BT261_ADDR, FIELDGHI );
    WRITE( BT261_DATA,    0 );
    WRITE( BT261_ADDR, FIELDSLO );
    WRITE( BT261_DATA, 0x2C );        /* stop at 3/4 HCOUNT - 2.5us    */
    WRITE( BT261_ADDR, FIELDSHI );
    WRITE( BT261_DATA,    1 );

}
```

```
#include <stdlib.h>
#include "gspreg.h"

#define BT251_ADDR 0x02000000
#define BT251_RAM  0x02000010
#define BT251_REG  0x02000020

typedef unsigned short ushort;

#define WRITE( addr, val )  *(ushort *)addr = val
#define READ( addr )  *(ushort *)addr

#define TH 255

main() {
   int i;

   WRITE( BT251_ADDR, 0x00 );

   for( i = 0; i < 256; i++ )     /* Fill RAM using auto          t */
      if( i < TH ) WRITE( BT251_RAM, 0 );
         else WRITE( BT251_RAM, i );                  incremen
}
```

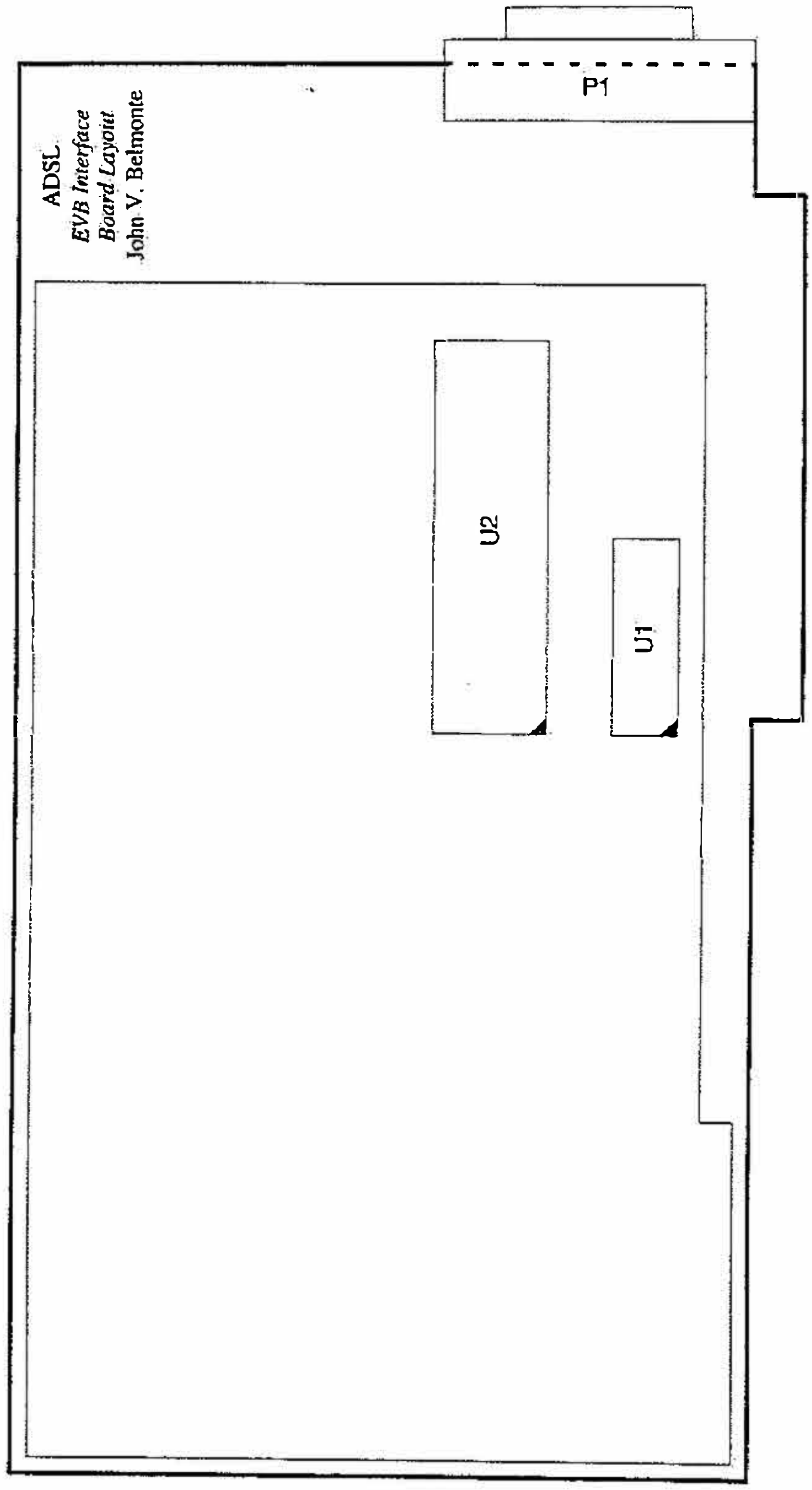
```c
/* find2t.c
   version 3
   12/4/92

   Tracks 2 LED's in the frame buffer.
   Assumes that there is a left LED (X1) and a right LED (X2) and that they
      do not cross boundaries.
   Uses windowing.
   Uses FIELD interrupt to ensure stable field for processing.
   Special stuff for light-tweezer input.
*/

#include <stdlib.h>
#include "gspreg.h"

#define TRUE 1
#define FALSE 0

#define FRAME_START ((10L * 1024L + 2) << 3)    /* start of image */
#define XMAX 340      /* max value send to host              */
#define YMAX 242
#define LEDRAD 12     /* radius of LED in image              */
#define XCOR 4        /* approx. offsets into center of LED  */
#define YCOR 4
#define BOXRAD 20


typedef unsigned char uchar;
typedef unsigned short ushort;
typedef unsigned int uint;

#define DPYINT_VECTOR   0xFFFFFEA0

/* status info starts at FFFF FF00 */
#define LOST        (*(int*)0xFFFFFF00)   /* number of times lock was lost */

/* coordiantes start at FFFF FF80 */
#define X1          (*(short*)0xFFFFFF80)
#define Y1          (*(short*)0xFFFFFF90)
#define X2          (*(short*)0xFFFFFFA0)
#define Y2          (*(short*)0xFFFFFFB0)

#define MAX( x, y ) ( ((x) > (y)) ? (x) : (y) )
#define MIN( x, y ) ( ((x) < (y)) ? (x) : (y) )

#define WRITE( addr, val )     *(ushort *)addr = val
#define READ( addr )           *(ushort *)addr
#define WRITEL( addr, val )    *(uint * )addr = val


grab_frame() {

   WRITE( DPYCTL, READ( DPYCTL ) | 0x1000 );     /* enable refresh       */


   WRITE( DPYINT, 261 );
   WRITE( INTPEND,READ(INTPEND) &0xFBFF);
   while( (READ(INTPEND)& 0x0400) == 0 ) {};

   WRITE( INTPEND, READ(INTPEND)& 0xFBFF);

   while( (READ(INTPEND)&0x0400)==0) {};

   WRITE( DPYCTL, READ( DPYCTL ) & 0xEFFF );     /* disable refresh      */
}


void scan( void ) {
   static int last = FALSE;
   const int xhalf = XMAX/2;
   const int yhalf = YMAX/2;
   int temp;

   if( !last ) {
      X1 = xhalf;
      Y1 = yhalf;

      last = search_box_lr( &X1, &Y1, xhalf, yhalf );  /* find left LED */

      temp = (XMAX - X1) / 2;
      X2 = X1 + temp;
      Y2 = yhalf;

      last = last && search_box_rl( &X2, &Y2, temp, yhalf );  /* find right LED */
      if( last ) WRITE( HSTCTLL, READ( HSTCTLL ) | 0x80 );
      LOST++;

   } else {

      last = search_box_ud( &X1, &Y1, BOXRAD, BOXRAD ) && search_box_ud( &X2, &Y2,
BOXRAD, BOXRAD );
      if( (X1 == X2) && (Y1 == Y2 ) ) last = FALSE;
      if( last ) WRITE( HSTCTLL, READ( HSTCTLL ) | 0x80 );
   }
}


int search_box_ud( short *xcur, short *ycur, short xrad, short yrad ) {
   int x1, x2, y1, y2;
   int x, y;
   unsigned long addr, majaddr;

   x1 = MAX( *xcur - xrad, 0 );           /* clip box against frame */
   x2 = MIN( *xcur + xrad, XMAX );
   y1 = MAX( *ycur - yrad, 0 );
   y2 = MIN( *ycur + yrad, YMAX );

   majaddr = FRAME_START + (((y1 << 10) + x1) << 3);

   for( y = y1; y < y2; y++ ) {

      addr = majaddr;

      for( x = x1; x < x2; x++ ) {
         if( (*(uchar*)addr) != 0 ) goto found_ud;
         addr += 8;
      }

      majaddr += 1024 << 3;
   }

   return( FALSE );  /* LED not found in box */

  found_ud:
   *xcur = x + 1;
   *ycur = y + YCOR;
   return( TRUE );
}


int search_box_lr( short *xcur, short *ycur, short xrad, short yrad ) {
   int x1, x2, y1, y2;
   int x, y;
   unsigned long addr, majaddr;

   x1 = MAX( *xcur - xrad, 0 );           /* clip box against frame */
   x2 = MIN( *xcur + xrad, XMAX );
   y1 = MAX( *ycur - yrad, 0 );
   y2 = MIN( *ycur + yrad, YMAX );


   majaddr = FRAME_START + (((y1 << 10) + x1) << 3);

   for( x = x1; x < x2; x++ ) {

      addr = majaddr;

      for( y = y1; y < y2; y++ ) {
```

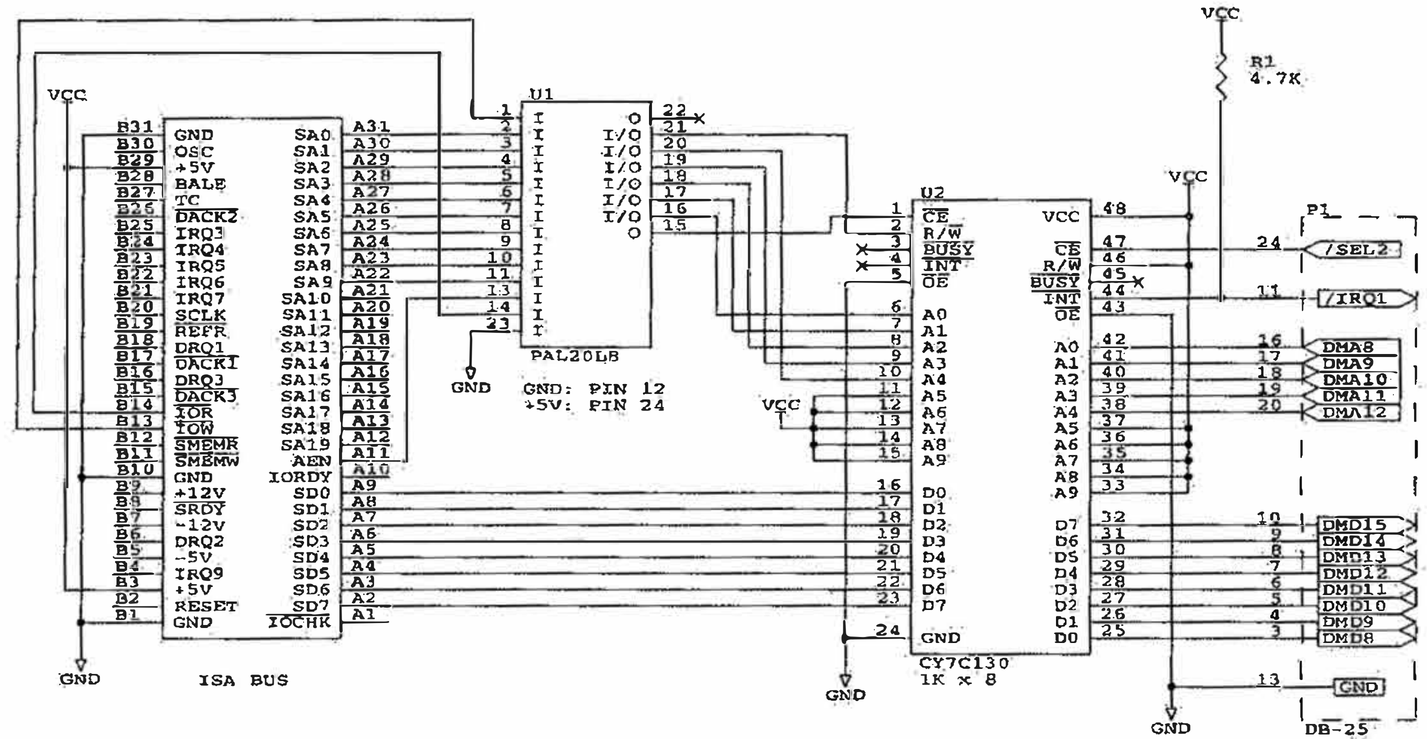
```
            if( (*(uchar*)addr) != 0 ) goto found_lr;
            addr += 1024 << 3;
        }

        majaddr += 8;
    }

    return( FALSE );  /* LED not found in box */

   found_lr:
    *xcur = x + XCOR;
    *ycur = y + 1;
    return( TRUE );
}


int search_box_rl( short *xcur, short *ycur, short xrad, short yrad ) {
    int x1, x2, y1, y2;
    int x, y;
    unsigned long addr, majaddr;

    x1 = MAX( *xcur - xrad, 0 );          /* clip box against frame */
    x2 = MIN( *xcur + xrad, XMAX );
    y1 = MAX( *ycur - yrad, 0 );
    y2 = MIN( *ycur + yrad, YMAX );


    majaddr = FRAME_START + (((y1 << 10) + x2-1) << 3);

    for( x = x2-1; x >= x1; x-- ) {

        addr = majaddr;

        for( y = y1; y < y2; y++ ) {
            if( (*(uchar*)addr) != 0 ) goto found_rl;
            addr += 1024 << 3;
        }

        majaddr -= 8;
    }

    return( FALSE );  /* LED not found in box */

   found_rl:
    *xcur = x - XCOR;
    *ycur = y + 1;
    return( TRUE );
}


main() {

    WRITE( LOST, 0 );

    while( TRUE ) {

/*      while( (READ( HSTCTLL ) & 0x80) == 0x80 ) {}; */
        grab_frame();
        scan();
    }

}
```

APPENDIX C. AUDIO SYSTEM DOCUMENTATION

C.1. Schematics and Programmable Logic

The following items describing the audio hardware are included in this section:


ADSL
EVB Interface
Board Layout
John V. Belmonte
P1
U2
U1



Advanced Digital Systems Lab		
Title: AD21020 EVB INTERFACE		
Size: A	Document Number: John V. Belmonte	REV: 1.0
Date: April 12, 1994	Sheet 1 of 1	

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    ADINT, U1
PATTERN
REVISION 0.5
AUTHOR   John V. Belmonte
COMPANY  ADSL
DATE     2/22/93

CHIP  _u1  PALCE22V10

;---------------------------------- PIN Declarations ---------------
PIN  1          /IOW
PIN  2          A0
PIN  3          A1
PIN  4          A2
PIN  5          A3
PIN  6          A4
PIN  7          A5
PIN  8          A6
PIN  9          A7
PIN  10         A8
PIN  11         A9
PIN  12         GND
PIN  13         AEN
PIN  14         /IOR   ; temp
PIN  15         /CE
PIN  16         A0out
PIN  17         A1out
PIN  18         A2out
PIN  19         A3out
PIN  20         A4out
PIN  21         R_W    ; temp
PIN  24         VCC

;----------------------------------- Boolean Equation Segment ------
EQUATIONS

CE = /AEN * A9 * A8 * /A7 * /A6 * /A5 * A4 * (IOW + IOR)   ; DECODE $310-31F

A0out = A0
A1out = A1
A2out = A2
A3out = A3
A4out = VCC

R_W = IOR
```

C.2. Software

The following item describing VRS software running on the ADSP-21020 evaluation board is included in this section:

- C language source code which implements the sound spatialization algorithm (*vrsnd.c*, pp. 79-80)

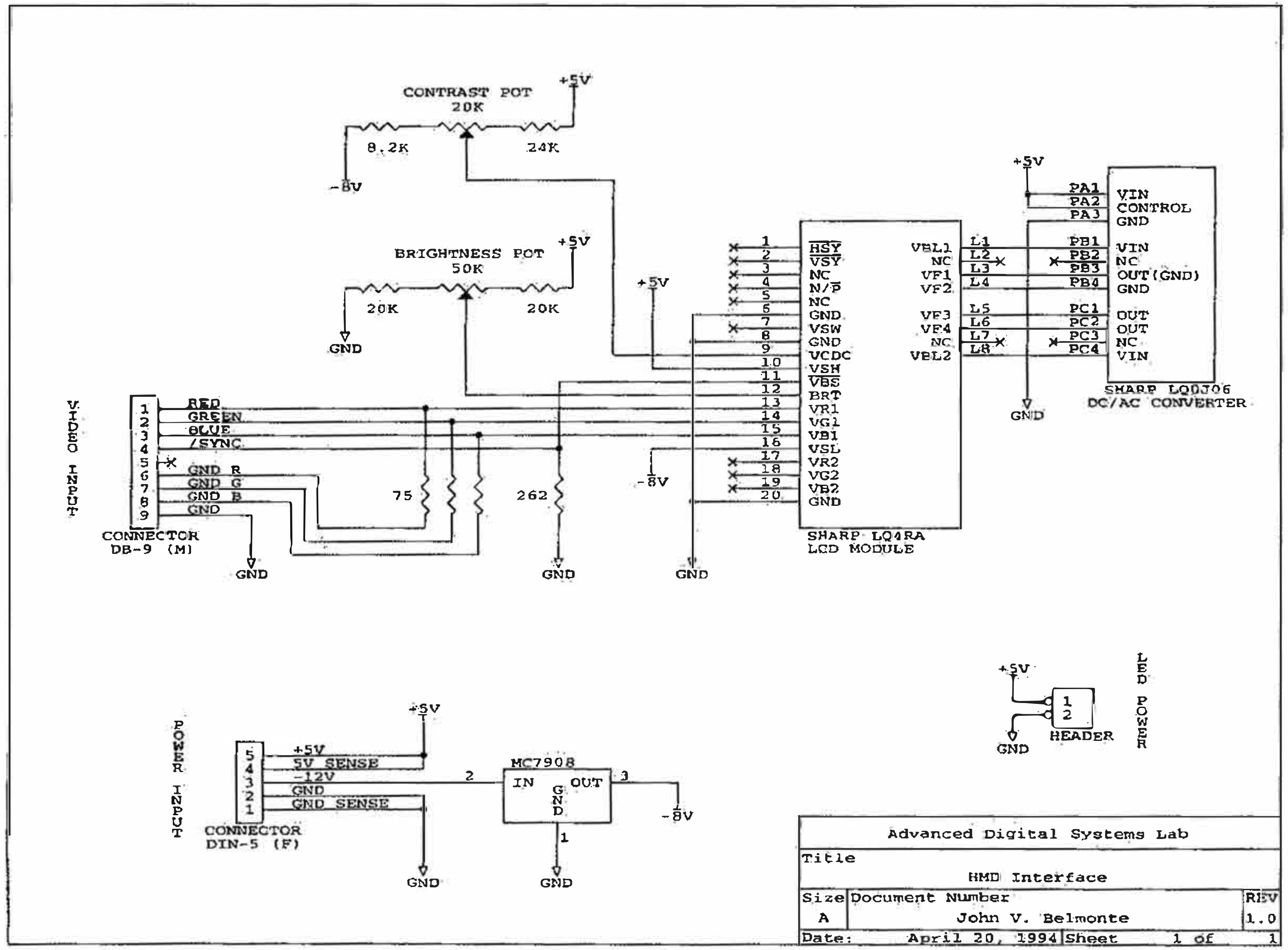
```
/* vrsnd.c   for AD21020 EVB
   Raymund Angara
   2/23/93        implemented reading x,y coordinates off ibm bus interface


                  +--+---------------------+--+
                  |S2|                     |S1|
                  +--+                     +--+
                  |           90°             |
                  |                           |
                  |     180°    *     0°      |
                  |                           |
                  |           270°            |
                  +--+                     +--+
                  |S3|                     |S4|
                  +--+---------------------+--+

*/

#include <21020.h>
#include <signal.h>
#include "def21020.h"
#include "ports8.h"
#include <stdio.h>
#include <math.h>

#define CD

/* coordinates of 4 speakers in room */
float speakers[4][2] = { {  11, 11.0 },    /* S1 */
                         { -11, 11.0 },    /* S2 */
                         { -11,-11.0 },    /* S3 */
                         {  11,-11.0 }, }; /* S4 */

/* G[4]=gain factor multiplied to each of the 4 channels using panning */
float G[4];

/* objx, objy are global coordinates read off from the ibm interface */
int objx,objy;


/* interrupt routine that reads x, and y values off the ibm interface  */
void  readcoord() {
    int templ,temp2;

    temp1 = (XCOORD & 0xff); /* need to mask out only important 8 bits */
    temp2 = (templ <<24 );   /* since we have 32-bit data, need to sign extend */

    objx  = (temp2 >>24);


    templ = (YCOORD & 0xff);
    temp2 = (templ <<24);
    objy  = (temp2 >>24);

    INT_CLR = INT_CLR;

  }

/* interrupt routine that writes values to the speaker channels */
void pass() {
  int val;

  val = in_audio_1;

#ifdef CD
  out_audio_l = val * G[1];                          /* S2 */
  out_audio_r = val * G[0];                          /* S1 */
  out_audio_a = (unsigned int)(val * G[3]) ^ 0x80000000;  /* S4 */
  out_audio_b = (unsigned int)(val * G[2]) ^ 0x80000000;  /* S3 */
#endif

#ifdef SIN
  out_audio_l = (int)0x7fff0000*sin(sinangle);                    /* S2 */
  out_audio_r = (int)0x7fff0000*sin(sinangle);                    /* S1 */
  out_audio_a =(int)(0x7f000000*sin(sinangle) ) ^ 0x80000000;     /* S4 */
  out_audio_b =(int)(0x7f000000*sin(sinangle) ) ^ 0x80000000;     /* S3 */
#endif

}


/* initialization of 21020 chip */
void setup() {

#pragma inline
  bit set mode2 0x8;       /* set IRQ3E : irq3 is edge triggered */
#pragma inline

  interrupt(SIG_IRQ3,pass);

#pragma inline
  idle;                    /* need this idle - don't know why    */
#pragma inline

  control_0 = 0;           /* line input  , no gain, no attenuation */
  control_1 = 0x00060000;  /* sampling rate= 44.1 KHz */


  interrupt(SIG_IRQ1, readcoord);
}


/* compute intensity panning gain factor for channel n with
   loudspeaker at distance distn and angle thetan (in radians)
   from listener located at (0,0) for a virtual sound source
   located at position (x,y)
*/

float gain(float x, float y, float thetan, float distn){
  static double right_angle, threepiov2;
  double diff;
  float  obangle;
  static first = 1;

  if ( first ) {
    first = 0;
    right_angle = 2.0 * atan(1.0);
    threepiov2 = 3.0 * right_angle;
  }
  obangle = atan2( (double)y,(double)x);
  if (obangle <0)
    obangle = obangle + 8.0 *atan(1.0);   /* this makes all angles positive */


  diff = thetan - obangle;

  if( (fabs(diff) > right_angle) && (fabs(diff) < threepiov2) )
    return(0);                    /* if difference between speaker angle and */
  if (diff <0)                    /* object angle is not within 180 deg. is set gain
to 0 */
    diff = diff + 8.0 * atan(1.0);/* make angle positive */

  return( cos(diff)*distn/sqrt( (double) x*x + y*y) );
}


main() {
  float spkrangle;
  int i;
  float theta[4], dist[4];
```

```
    setup();

    for (i=0;i<4;i++) {
       dist[i] = sqrt( speakers[i][0]*speakers[i][0] + speakers[i][1]*speakers[i][1]);

       spkrangle = atan2( speakers[i][1], speakers[i][0] );
       if (spkrangle < 0 )
          spkrangle = spkrangle + 8.0 * atan(1.0); /* make angle positive */
       theta[i] = spkrangle;
    }

    while (1) {

        for (i=0;i<4;i++)
           G[i]=gain((float)objx,(float)objy,theta[i],dist[i]);

    }
}
```

APPENDIX D. HEAD-MOUNTED DISPLAY DOCUMENTATION

The following item describing the HMD hardware is included in this section:

- Circuit board schematic diagram (p. 82)


CONTRAST POT
20K
8.2K
24K
+5V
-8V
BRIGHTNESS POT
50K
20K
20K
GND
VIDEO INPUT
RED
GREEN
BLUE
/SYNC
GND R
GND G
GND B
GND
CONNECTOR
DB-9 (M)
75
262
HSY
VSY
NC
N/P
GND
VSW
VCDC
VSH
VBS
BRT
VR1
VG1
VB1
VSL
VR2
VG2
VB2
VBL1
VF1
VF2
VF3
VF4
VBL2
SHARP LQ4RA
LCD MODULE
PA1
PA2
PA3
PB1
PB2
PB3
PB4
PC1
PC2
PC3
PC4
VIN
CONTROL
OUT(GND)
OUT
SHARP LQ0J06
DC/AC CONVERTER
LED POWER
HEADER
POWER INPUT
+5V
5V SENSE
-12V
GND SENSE
CONNECTOR
DIN-5 (F)
MC7908
IN
OUT
GND
Advanced Digital Systems Lab
Title
HMD Interface
Size
A
Document Number
John V. Belmonte
REV
1.0
Date: April 20, 1994
Sheet 1 of 1

APPENDIX E. HOST SYSTEM DOCUMENTATION

The following items describing VRS software running on the host CPU are included in this section:

- C++ language source code containing the application program and the host CPU's graphics pipeline stages (*v11.cpp*, pp. 84-90)
- C++ language header which defines the communication structure between the host and the G-Node (*com.h*, p. 91)
- Object definition file for miscellaneous scenery (*input.def*, p. 92)
- Object definition file for the musical note (*note.def*, p. 93)
- Object definition file for the pinwheel (*pinwheel.def*, p. 94)
- Object definition file for the cootie bug (*jencoot.def*, p. 95)

```
// C
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <time.h>
#include <ctype.h>
#include <string.h>

// C++
#include <iostream.h>
#include <timer.h>

// Mine
#include "com.h"

#define TRUE  1
#define FALSE 0

#define PI 3.14159265359

/* Defines for head tracking */

#define HSTADRL1 0x300
#define HSTADRH1 0x302
#define HSTDATA1 0x304
#define HSTCTRL1 0x306

#define HSTADRL2 0x308
#define HSTADRH2 0x30A
#define HSTDATA2 0x30C
#define HSTCTRL2 0x30E

#define X1 0xFFFFFF80
#define Y1 0xFFFFFF90
#define X2 0xFFFFFFA0
#define Y2 0xFFFFFFB0

#define XMAX 340
#define YMAX 240


/* defines for sound interface */

#define X_CORD     0x310
#define Y_CORD     0x311
#define SEND_INT   0x31f

#define FLIP


/* joystick */

#define CHANNEL_0       0x0040          /* system tick */
#define CHANNEL_2       0x0042          /* speaker tone */
#define COMMAND_REG     0x0043
#define WRITE_CH0       0x0036
#define WRITE_CH2       0x00b6
#define READ_SPECIAL    0x00c2
#define PORT_B          0x0061          /* speaker on/off control */
#define XTAL            1193000L
#define TIMER_MODES     0x000e
#define TIMER_MODE      0x0002
#define TIMER_OUT       0x0080
#define TIMER_PERIOD    0x0000ffffL
#define JS_PORT         0x201
#define JS_TIMEOUT      32000
#define JS_READ         inp (JS_PORT)

typedef unsigned short  Ushort;
typedef unsigned int    Uint;
typedef unsigned long   Ulong;

struct stick {
        Ushort  a[4];
        Ushort  b[4];
};

typedef struct stick    STICK;


/************************************** --> UNITS are centimeters */

/* structures */
typedef struct { float p[4];
                 float hp[4];
                 float dia;
                 int color;
               } object;

/* globals */
int endprog = FALSE;
float focal = 1.6;                  // 16mm focal length
float wx0 = -1, wx1 = 1;            // 1cm x 1cm view plane
float wy0 = -1, wy1 = 1;
float znear = -1; // , zfar = -1000;
float TM[4][4];
int npoints = 0;                    // number of sphere objects in world
object objList[200];
const unsigned JPORT = 0x201;       // joystick port
float jx, jy;
int sound_objects[4] = { -1, -1, -1, -1 };   // object ID of sound sources
float vrp[3] = { 0, 180, 1000 };     // viewer's position
float torsoAngle = 0;                // angle of user's torso in radians
double dtime;                       // delta time
double hold_time;


/* for TMS34010 Digitizer Boards */

int gspread1( unsigned long addr ) {

        outport( HSTADRL1, addr & 0xFFFF );
        outport( HSTADRH1, (addr >> 16) & 0xFFFF );
        return inport( HSTDATA1 );
}

void gspwrite1( unsigned long addr, unsigned int value ) {

        outport( HSTADRL1, addr & 0xFFFF );
        outport( HSTADRH1, (addr >> 16) & 0xFFFF );
        outport( HSTDATA1, value );
}


int gspread2( unsigned long addr ) {

        outport( HSTADRL2, addr & 0xFFFF );
        outport( HSTADRH2, (addr >> 16) & 0xFFFF );
        return inport( HSTDATA2 );
}

void gspwrite2( unsigned long addr, unsigned int value ) {

        outport( HSTADRL2, addr & 0xFFFF );
        outport( HSTADRH2, (addr >> 16) & 0xFFFF );
        outport( HSTDATA2, value );
}


/* for TMS34020 Display Board */

unsigned int gsp20read( unsigned long addr ) {
    unsigned int val;

    poke( 0xD100, 0x0000, addr >> 15 );              /* set bank     */
    val = peek( 0xD000, (addr >> 3) & 0xOFFE1 );     /* read value   */
    return( val );
```

```
}

void gsp20write( unsigned long addr, unsigned int value ) {

    poke( 0xD100, 0x0000, addr >> 15 );              /* set bank      */
    poke( 0xD000, (addr >> 3) & 0xOFFE1, value );    /* write value   */
}


void sphere20( int id, int x, int y, int z, int dia, int color ) {
    unsigned long object;

    object = COMBASE + OBJ_START + OBJ_SIZE * id;

#ifdef FLIP
    gsp20write( object + X_POS, 160 - x );
#else
    gsp20write( object + X_POS, x );
#endif
    gsp20write( object + Y_POS, y );
    gsp20write( object + Z_POS, (unsigned int)z );
    gsp20write( object + DIA, dia );
    gsp20write( object + COLOR, color );
}


void sphereShut( int id ) {  // set diameter to zero so sphere is not shown
    unsigned long object;

    object = COMBASE + OBJ_START + OBJ_SIZE * id;
    gsp20write( object + DIA, 0 );
}


/* joystick routines */

int jbut1() { return( !(inportb(JPORT) & 1<<4) ); }

int jbut2() { return( !(inportb(JPORT) & 1<<5) ); }


/* sets jx and jy globals, range -5 to 5 */
/*
void jxy() {
    unsigned xcount = 0;
    unsigned ycount = 0;
    unsigned char val;

    outportb(JPORT, 0);   // start joystick timer

    while( (inportb(JPORT) & 0x3) != 0 ) {
        val = inportb(JPORT);

        xcount += (val >> 0) & 1;
        ycount += (val >> 1) & 1;
    }

    jx = xcount/38 - 5;
    jy = ycount/37 - 5;
}
*/

static Ushort near
jdisable (void)
{
        Ushort  flags;

        _asm {
                pushf
                pop     flags
                cli
        }
        return (flags);
}

static void near
jenable (Ushort flags)
{
        _asm {
                push    flags
                popf
        }
}

static Uint near
get_timer (void)                /* get fastest timer available */
{
        Ushort          t, status, flags;

        do {
                flags = jdisable ();
                outp (COMMAND_REG, READ_SPECIAL);
                status = (Ushort) inp (CHANNEL_0);        /* get status */
                t  = (Ushort)inp (CHANNEL_0);             /* low byte */
                t += (Ushort)(inp (CHANNEL_0) << 8);      /* high byte */
                jenable (flags);
        } while (0 == t);

        return ((Uint)(Ushort)-((status & TIMER_MODES) == TIMER_MODE*2 ? t
                : ((t>>1) + (Ushort)((status&TIMER_OUT)<<8))));
}

#define READING (mode ? get_timer () : JS_TIMEOUT-i)


/*  mode: 1 = timer
          0 = counter

    mask: 3 = joystick A
          c = joystick B
*/

/* Joystick routines from Eyal Lebedinsky (eyal@ise.canberra.edu.au) */

static int near
readjoy (STICK *s, int mode, int mask, int nread, int delay)
{
        register int    i;
        register Uint   m;
        unsigned int    t, x1, y1, x2, y2, minx1, miny1, minx2, miny2;
        int             js, tt, ntimes;

        minx1 = miny1 = minx2 = miny2 = 0xffffU; /* avoid compiler warning */
        memset (&->a, 0, sizeof (&->a));

        for (ntimes = 0;;) {
                i = JS_TIMEOUT;
                t = READING;
                x1 = y1 = x2 = y2 = t;
                outp (JS_PORT, 0);              /* set trigger */
                for (m = mask; m;) {
                        while (!(~JS_READ & m) && --i)
                                ;
                        if (!i)
                                break;
                        tt = READING;
                        js = ~JS_READ & m;
                        if (js & 0x01) {
                                x1 = tt;
                                m &= ~0x01;
                        }
                        if (js & 0x02) {
                                y1 = tt;
                                m &= ~0x02;
                        }
                        if (js & 0x04) {
                                x2 = tt;
                                m &= ~0x04;
                        }
```

```
			if (js & 0x08) {
				y2 = tt;
				m &= ~0x08;
			}
		}
		if (minx1 > (x1 -= t))
			minx1 = x1;
		if (miny1 > (y1 -= t))
			miny1 = y1;
		if (minx2 > (x2 -= t))
			minx2 = x2;
		if (miny2 > (y2 -= t))
			miny2 = y2;

		if (++ntimes >= nread)	/* read more? */
			break;

		if (0 != (i = delay)) {		/* delay? */
			tt = 1234;
			for (i *= 10; i-- > 0;)
				tt *= 19;
		}
	}

	js = m | ~mask;
	s->a[0] = (js & 0x01) ? 0 : minx1;	/* analog 1 */
	s->a[1] = (js & 0x02) ? 0 : miny1;	/* analog 2 */
	s->a[2] = (js & 0x04) ? 0 : minx2;	/* analog 3 */
	s->a[3] = (js & 0x08) ? 0 : miny2;	/* analog 4 */

	js = ~JS_READ;
	s->b[0] = !!(js & 0x10);		/* button 1 */
	s->b[1] = !!(js & 0x20);		/* button 2 */
	s->b[2] = !!(js & 0x40);		/* button 3 */
	s->b[3] = !!(js & 0x80);		/* button 4 */

	return (m);
}

#define rj( a )  readjoy( a, 1, 3, 2, 0 )

static int jxmin, jxmax;
static int jymin, jymax;
static int jxc, jyc;

void calibrate( void ) {
    STICK joy[1];

    if( rj(joy) ) {
        printf( "No joystick present.\n" );
        exit(1);
    }

    printf( "Joystick calibration...\n" );
    do {
        rj( joy );
    } while( (joy->b[0] == 1) || (joy->b[1] == 1) );

    printf( "    Center joystick and hit a button\n" );
    do {
        rj( joy );
        jxc = joy->a[0];
        jyc = joy->a[1];
    } while( (joy->b[0] == 0) && (joy->b[1] == 0) );

    do {
        rj( joy );
    } while( (joy->b[0] == 1) || (joy->b[1] == 1) );

    printf( "    Move joystick to upper-left and hit a button\n" );
    do {
        rj( joy );
        jxmin = joy->a[0];
        jymin = joy->a[1];
    } while( (joy->b[0] == 0) && (joy->b[1] == 0) );

    do {
        rj( joy );
    } while( (joy->b[0] == 1) || (joy->b[1] == 1) );

    printf( "    Move joystick to lower-right and hit a button\n" );
    do {
        rj( joy );
        jxmax = joy->a[0];
        jymax = joy->a[1];
    } while( (joy->b[0] == 0) && (joy->b[1] == 0) );

    printf( "%d-%d-%d : %d-%d-%d\n", jxmin, jxc, jxmax, jymin, jyc, jymax );
}

void jxy( void ) {
    STICK joy[1];
    int x, y;

    rj( joy );

    x = (int)joy->a[0] - jxc;
    if( x >= 0 ) x = (10*x + ((jxmax-jxc) >> 1))/(jxmax-jxc);
        else x = (10*x - ((jxc-jxmin) >> 1))/(jxc-jxmin);

    y = (int)joy->a[1] - jyc;
    if( y >= 0 ) y = (10*y + ((jymax-jyc) >> 1))/(jymax-jyc);
        else y = (10*y - ((jyc-jymin) >> 1))/(jyc-jymin);

    jx = (float)x/10;
    jy = (float)y/10;

    printf( "%4.1f %4.1f %ld %ld ** ", jx, jy, joy->b[0], joy->b[1] );
}

double vrTime() {
    static Timer timer;
    double time;

    timer.stop();
    time = timer.time();
    timer.start();

    return( time );
}

void freq( int first ) {
    static double start;
    static unsigned long count;

    if( first ) {
        count = 0;
        start = vrTime();
        first = FALSE;
        return;
    }

    count++;

//    cout << (int)(count/(vrTime() - start));
    printf( "%4d", (int)(count/(vrTime() - start)) );
}

void vnorm( float *v, float *val ) {
    float mag;

    mag = sqrt( v[0]*v[0] + v[1]*v[1] + v[2]*v[2] );

    if( mag == 0 ) mag = .000001;
```

```
    val[0] = v[0]/mag;
    val[1] = v[1]/mag;
    val[2] = v[2]/mag;
}

void vcross( float *v1, float *v2, float *val ) {

    val[0] = (v1[1] * v2[2]) - (v2[1] * v1[2]);
    val[1] = (v1[2] * v2[0]) - (v2[2] * v1[0]);
    val[2] = (v1[0] * v2[1]) - (v2[0] * v1[1]);
}

void add11( float *v1, float *v2, float *ans ) {
    int i;

    for( i = 0; i < 3; i++ )
        ans[i] = v1[i] + v2[i];
}

void sub11( float *v1, float *v2, float *ans ) {
    int i;

    for( i = 0; i < 3; i++ )
        ans[i] = v1[i] - v2[i];
}

void mul01( float s, float *v, float *ans ) {
    int i;

    for( i = 0; i < 3; i++ )
        ans[i] = s * v[i];
}

void mul44( float m1[4][4], float m2[4][4], float val[4][4] ) {
    int i, j, k;

    for( i = 0; i < 4; i++ ) {
        for( j = 0; j < 4; j++ ) {
            val[i][j] = 0;
            for( k = 0; k < 4; k++ ) {
                val[i][j] += m1[i][k] * m2[k][j];
            }
        }
    }
}

void mul14( float *v, float m[4][4], float *val ) {
    int j, k;

    for( j = 0; j < 4; j++ ) {
        val[j] = 0;
        for( k = 0; k < 4; k++ ) {
            val[j] += v[k] * m[k][j];
        }
    }
}

void init4( float m[4][4] ) {

    m[0][1] = m[0][2] = m[0][3] = 0;
    m[1][0] = m[1][2] = m[1][3] = 0;
    m[2][0] = m[2][1] = m[2][3] = 0;
    m[3][0] = m[3][1] = m[3][2] = 0;

    m[0][0] = m[1][1] = m[2][2] = m[3][3] = 1;
}

void show() {
    int i;
    float newp[4];
    float x, y, z, dia;

    for( i = 0; i < npoints; i++ ) {

        mul14( objList[i].p, TM, newp );

        if( newp[3] <= 0 ) {
            sphereShut( i );
            continue;
        }

        x = newp[0]/newp[3];
        y = newp[1]/newp[3];
        z = newp[2];

        /*
        if( z > znear ) {
            sphereShut( i );
            continue;
        }
        */

        dia = - 2.0 * objList[i].dia * focal / z;

        x *= 100.0;
        y *= 100.0;
        dia *= 100.0;

        if( !jbut2() )
            sphere20( i, x+80.5, 117.5-2*y, z, dia, objList[i].color );
            else sphere20( i, x+80.5, 117.5-2*y, z, dia, 16 ); // "wireframe"
    }
}

int parse( char* filename ) {
    FILE * infile;
    char cline[80];
    int x, y, z, dia, color;
    char scanstring[] = "%d %d %d %d %d";
    object *obj;
    int pointsread = 0;

    infile = fopen( filename, "rt" );

    while( TRUE ) {
        if( fgets(cline, 80, infile) == NULL ) break;    // end of file
        if( sscanf( cline, scanstring, &x, &y, &z, &dia, &color ) != 5 ) continue;
        obj = &(objList[npoints++]);
        obj->hp[0] = obj->p[0] = x;
        obj->hp[1] = obj->p[1] = y;
        obj->hp[2] = obj->p[2] = z;
        obj->hp[3] = obj->p[3] = 1;
        obj->dia = dia;
        obj->color = color;
        pointsread++;
    }

    cout << filename << " read: " << pointsread << " spheres.\n";

    return( pointsread );
}

float headVect[3] = { 0, 0, 1 };
void interrupt (* old_irq10)(...);
void interrupt (* old_irq11)(...);

/* get info from HT1 and update headVect[] */
void interrupt ht1(...) {
    int hx1, hy1, hx2, hy2;
    int hx3, hy3, hx4, hy4;
```

```
    outportb( 0xA0, 0x20 );   // signal end of int to interrupt controller

            hx1 = XMAX - gspread1( X1 );
            hy1 = gspread1( Y1 );
            hx2 = XMAX - gspread1( X2 );
            hy2 = gspread1( Y2 );

    if( hx1 > hx2 ) {
        headVect[0] = hy1 - hy2;
        headVect[2] = hx1 - hx2;
    } else {
        headVect[0] = hy2 - hy1;
        headVect[2] = hx2 - hx1;
    }

            outport( HSTCTRL1, inport( HSTCTRL1 ) & 0xFF7F );  // clear INTOUT

    old_irq10();
}

void interrupt ht2(...) {
    int hx3, hy3, hx4, hy4;

    outportb( 0xA0, 0x20 );   // signal end of int to interrupt controller

    hx3 = XMAX - gspread2( X1 );
    hy3 = gspread2( Y1 );
    hx4 = XMAX - gspread2( X2 );
    hy4 = gspread2( Y2 );

    if( hx3 > hx4 ) {
        headVect[1] = hy4 - hy3;
    } else {
        headVect[1] = hy3 - hy4;
    }

            outport( HSTCTRL2, inport( HSTCTRL2 ) & 0xFF7F );  /* clear INTOUT */

    old_irq11();
}


#define  IRQ10  0x72
#define  IRQ11  0x73

void set_irqs() {

    /* save the old interrupt vectors */
    old_irq10 = getvect(IRQ10);
    old_irq11 = getvect(IRQ11);

    setvect( IRQ10, ht1 );
    setvect( IRQ11, ht2 );

            outport( HSTCTRL1, inport( HSTCTRL1 ) & 0xFF7F );  /* clear INTOUTS */
            outport( HSTCTRL2, inport( HSTCTRL2 ) & 0xFF7F );

    outportb( 0xA1, inportb( 0xA1 ) & 0xFB );    // enable IRQ 10
    outportb( 0xA1, inportb( 0xA1 ) & 0xF7 );    // enable IRQ 11
}

void sound2d( float x, float y ) {

    float temp3[3] = { 0, 0, 0 };

#ifdef FLIP
    temp3[0] = x;
#else
    temp3[0] = x;
#endif
    temp3[2] = -y;

    float mag = sqrt( x*x + y*y );
    vnorm( temp3, temp3 );

    if( mag > 10000 ) {
        mul01( 127, temp3, temp3 );
    } else {
        mul01( mag/91 +18, temp3, temp3 );
    }

    outportb( X_CORD, temp3[0] );
    outportb( Y_CORD, temp3[2] );
    outportb( SEND_INT, 0 );
}


void updateSounds() {
    object* tobj;
    float temp3[3];
    float soundV[3];

    for( int i = 0; i < 1; i++ ) {    // currently only 1 sound source

      if( sound_objects[i] == -1 ) continue;

      tobj = &(objList[sound_objects[i]]);

      sub11( tobj->p, vrp, temp3 );

      soundV[0] = -temp3[0]*cos(-torsoAngle) - temp3[2]*sin(-torsoAngle);
      soundV[1] = temp3[1];
      soundV[2] = temp3[2]*cos(-torsoAngle) - temp3[0]*sin(-torsoAngle);

      sound2d( soundV[0], soundV[2] );
    }
}

void check_time( float time, int first ) {
    static float start;
    static int timeout;
    int timeleft;
    char c;

    if( first ) {
        start = time;
        timeout = 1;
    }

    timeleft = timeout - (time - start);

    if( timeleft < 0 ) timeleft = 0;

    gsp20write( COMBASE + TIMEOUT, (float)timeleft/timeout*159 );

    if( timeleft == 0 ) {
        c = getch();
        if( isdigit( c ) ) timeout = (c-48) * 60;
        if( c == 'q' ) endprog = TRUE;
        start = vrTime();
        freq( TRUE );
    }
}

class CppObj {
    int startObj;
    int numObjs;
    char filename[15];
    float trans[4];

    public:

    CppObj( char *s ) {
        strcpy( filename, s );
    }
```

```
void load() {
    startObj = npoints;
    numObjs = parse( filename );
}

void becomeSound( int soundNum ) {
    sound_objects[soundNum] = startObj;
}

void moveTo( float x, float y, float z ) {

    trans[0] = x;
    trans[1] = y;
    trans[2] = z;
    trans[3] = 0;

    for( int i = 0; i < numObjs; i++ ) {
        add11( objList[startObj+i].p, trans, objList[startObj+i].p );
        add11( objList[startObj+i].hp, trans, objList[startObj+i].hp );
    }
}

void bounce( float vo ) {
    const a = -980;  // gravity
    static int first = TRUE;
    static float t0;
    float ltime;

    if( first ) {
        t0 = hold_time;
        first = FALSE;
    }

    ltime = hold_time - t0;

    int x = a*ltime*ltime + vo*ltime;

    if( x < 0 ) first = TRUE;   // do it again

    for( int i = 0; i < numObjs; i++ ) {
        objList[startObj+i].p[1] = objList[startObj+i].hp[1] + x;
    }
}

/*
void randBounce( ) {
    const a = -980;  // gravity
    static int first = TRUE;
    static float t0;
    float ltime;

    if( first ) {
        t0 = hold_time;
        first = FALSE;
    }

    ltime = hold_time - t0;

    int x = a*ltime*ltime + vo*ltime;

    if( x < 0 ) first = TRUE;   // do it again

    for( int i = 0; i < numObjs; i++ ) {
        objList[startObj+i].p[1] = objList[startObj+i].hp[1] + x;
    }
}
*/

void spin( float period ) {
    float angle;
    float *temp;
    float temp3[3];

    angle = hold_time*2.0*PI/period;

    for( int i = 0; i < numObjs; i++ ) {
        temp = objList[startObj+i].p;
        sub11( objList[startObj+i].hp, trans, temp );

        temp3[0] = -temp[0]*cos(-angle) - temp[2]*sin(-angle);
        temp3[1] = temp[1];
        temp3[2] = temp[2]*cos(-angle) - temp[0]*sin(-angle);

        add11( temp3, trans, temp );
    }
  }
};


void main() {
    int i, j;
    float a,b,c,d;
    double oldtime, time;
    float joyAngle;
    float temp3[3];
    float soundV[3];
    float horizon;
    float headVectH[3];
    object *tobj;

    float torsoVect[3] = { 0, 0, 1 };
    float N[3] = { 0, 0, 1 };  // view Normal vector
    float V[3] = { 0, 1, 0 };  // View up vector
    float cop[3] = { 0, 0, focal };       /* cop[2] here is focal length */

    float n[3], u[3], v[3];

    float T[4][4];
    float RR[4][4];
    float ST[4][4];
    float S[4][4];
    float TEMP[4][4];
    float PRE[4][4];

    clrscr();

    calibrate();

    set_irqs();

    /* calc parameters for ST matrix */
    a = cop[0] - (wx0 + wx1)/2;
    b = cop[1] - (wy0 + wy1)/2;
    c = a/cop[2];
    d = b/cop[2];

    init4( ST );

    ST[2][0] = -c;
    ST[2][1] = -d;
    ST[3][0] = a;
    ST[3][1] = b;

    /* setup S matrix */
    init4( S );

    S[2][0] = -cop[0]/cop[2];
    S[2][1] = -cop[1]/cop[2];
    S[2][3] = -1/cop[2];

    /* pre-calc ST * S matrix */
    mul44( ST, S, PRE );

    parse( "input" );      // parse world definition file

    /* the incredible musical note */
    CppObj note( "note.def" );
    note.load();
    note.becomeSound(0);
    note.moveTo( 2000, 0, 2000 );
```

```
/* super pinwheel by jaa */
CppObj pinwheel( "pinwheel.def" );
pinwheel.load();
pinwheel.moveTo( -2000, 1500, -2000 );


CppObj jencoot( "jencoot.def" );
jencoot.load();
jencoot.moveTo( -2000, 0, -2000 );

freq( TRUE );
check_time( 0, TRUE );

gsp20write( COMBASE + OBJ_COUNT, npoints );

/***************************  LOOP  ********************************/

while( 1 ) {

    oldtime = time;
    time = vrTime();
    hold_time = time;
    dtime = time - oldtime;

    if( jbut1() ) {  // go home
        torsoAngle = 0;
        vrp[0] = 0;
        vrp[2] = 1000;
    }

    jxy();

    mul01( 1000.0*jy*dtime, torsoVect, temp3 );
    add11( vrp, temp3, vrp );

    joyAngle = (jx < 0) ? -1.5*dtime*jx*jx : 1.5*dtime*jx*jx;

    torsoAngle += joyAngle;
    if( torsoAngle < (-PI) ) torsoAngle += 2*PI;
    if( torsoAngle >   PI  ) torsoAngle -= 2*PI;

    torsoVect[0] = sin( torsoAngle );
    torsoVect[1] = 0;
    torsoVect[2] = cos( torsoAngle );

    vnorm( headVect, headVectH );   // need local copy

    N[0] = - headVectH[0]*cos(-torsoAngle) - headVectH[2]*sin(-torsoAngle);
    N[1] =   headVectH[1];
    N[2] = + headVectH[2]*cos(-torsoAngle) - headVectH[0]*sin(-torsoAngle);

    horizon = - N[1] * 320 + 119;   // mystery parameters
    if( horizon < 1 ) horizon = 1;
    if( horizon > 234 ) horizon = 234;
    gsp20write( COMBASE + HORIZON, horizon );

    note.bounce( 1000 );
    pinwheel.spin( 3 );

    updateSounds();

    check_time( time, FALSE );

    /* find u, v, n */
    vnorm( N, n );          // n = N / |N|

    vcross( N, V, u );
    vnorm( u, u );          // u = N x V / |N x V|

    vcross( u, n, v );   // v = u x n

    /* setup T and RR */
    init4( T );
    init4( RR );

    for( i = 0; i < 3; i++ ) {
        T[3][i] = -vrp[i];
        RR[i][0] = u[i];
        RR[i][1] = v[i];
        RR[i][2] = n[i];
    }

    /* matrix concatenation: (T * RR * ST * S) */
    /*                       (T * RR * PRE )   */
    mul44( T, RR, TEMP );
    mul44( TEMP, PRE, TM );

    /* TM now contains the complete transformation matrix */
    /* Divide resulting coordinates by (-z/cop3 + 1)      */
    /* Clip against wx0, wy0, wx1, wy1, zfar, znear       */

    delay( 1 );
    show();

    freq( FALSE );
//    printf( " | %d %d %+2d %+2d", jbut1(), jbut2(), jx, jy );
    cout << "\r";

    if( kbhit() ) {
        switch( getch() ) {
            case 27: check_time( time, TRUE );
                     break;

            case 'q': endprog = TRUE;
                      break;
        }
    }

    if( endprog ) break;
}

outportb( 0xA1, inportb( 0xA1 ) | 0xC );  // disable IRQ 10, 11

setvect( IRQ10, old_irq10 );
setvect( IRQ11, old_irq11 );
}
```

```
/* com.h
   IBM SIDE
   Defines communication interface
   John Belmonte
   2/3/93
*/

#define COMBASE   0xFFE00000       /* this is where it all begins */
#define COMEND    0xFFF00000       /* 128K for now                */


/* All remaining values are  *
 * in units of 16-BIT WORDS! */

/********************** base structure ****************************/

#define OBJ_COUNT   0x00           /* number of objects in the list */
#define HORIZON     0x10           /* y position of horizon         */
#define TIMEOUT     0x20           /* user time left                */
#define OBJ_START   0x40           /* start of object list          */

/********************** object structure ***************************/

#define OBJ_SIZE    0x100          /* this should be power of 2     */

/*  offsets */
#define X_POS       0x00
#define Y_POS       0x10
#define Z_POS       0x20
#define DIA         0x30
#define COLOR       0x40
//      Z_HOLD      0x50
```

```
; BLACK.........0   RED...........4   DARK_GRAY......8   LIGHT_RED......12
; BLUE..........1   MAGENTA.......5   LIGHT_BLUE.....9   LIGHT_MAGENTA..13
; GREEN.........2   BROWN.........6   LIGHT_GREEN....10  YELLOW.........14
; CYAN..........3   LIGHT_GRAY....7   LIGHT_CYAN.....11  WHITE..........15

;     X        Y        Z       DIA     COLOR
;   ----     ----     ----     ----     -----

; YIN YANG
      0      300        0       80        8          ; eye
      0      370        0       60        7          ; head...
     50      350        0       60        7
     70      300        0       60        7
     50      250        0       60        7
      0      230        0       60        7
    -50      350        0       60        7
    -70      300        0       60        7
    -50      250        0       60        7
      0      100        0       80        7          ; eye
      0       30        0       60        8          ; head...
     50       50        0       60        8
     70      100        0       60        8
     50      150        0       60        8
      0      170        0       60        8
    -50       50        0       60        8
    -70      100        0       60        8
    -50      150        0       60        8

; COOTIE FACE 1
    400       80        0     -160        2            ;head
    370      110       75       20        1            ;eyes
    430      110       75       20        1            ;
    400       80      100       40        4            ;nose
    380      160       50       30        5            ;antennae
    420      160       50       30        5            ;
    370      180       60       30        5            ;
    430      180       60       30        5            ;
    360      190       75       30        5            ;
    440      190       75       30        5            ;

; COOTIE FACE 2
   -400       80        0     -160        2            ;head
   -430      110      -75       20        1            ;eyes
   -370      110      -75       20        1            ;
   -400       80     -100       40        4            ;nose
   -420      160      -50       30        5            ;antennae
   -380      160      -50       30        5            ;
   -430      180      -60       30        5            ;
   -370      180      -60       30        5            ;
   -440      190      -75       30        5            ;
   -360      190      -75       30        5            ;

; Night Sun
  10000     4000    15000    -1000        1
```

```
; BLACK..........0   RED............4   DARK_GRAY......8   LIGHT_RED......12
; BLUE...........1   MAGENTA........5   LIGHT_BLUE.....9   LIGHT_MAGENTA..13
; GREEN..........2   BROWN..........6   LIGHT_GREEN...10   YELLOW.........14
; CYAN...........3   LIGHT_GRAY.....7   LIGHT_CYAN....11   WHITE..........15

;   X       Y       Z       DIA     COLOR
;  ----    ----    ----    ----    -----

    0      100      0      220      3
   75      200      0       70      3
   75      250      0       70      3
   75      300      0       70      3
   75      350      0       70      3
   75      400      0       70      3
   75      450      0       70      3
   75      500      0       70      3
   75      550      0       70      3
  125      475      0       60      3
  175      450      0       50      3
```

```
; BLACK..........0   RED...........4   DARK_GRAY.....8   LIGHT_RED.....12
; BLUE...........1   MAGENTA.......5   LIGHT_BLUE....9   LIGHT_MAGENTA..13
; GREEN..........2   BROWN.........6   LIGHT_GREEN...10  YELLOW........14
; CYAN...........3   LIGHT_GRAY....7   LIGHT_CYAN....11  WHITE.........15

;     X        Y       Z        DIA     COLOR
;   ----     ----    ----     ----    -----

      0        0       0        160     1
      0        0       160      180     5
      160      0       80       180     5
      160      0       -80      180     5
      0        0       -160     180     5
     -160      0       -80      180     5
     -160      0       80       180     5
     140       0       220      120     5
     280       0       0        120     10
     140       0       -220     120     5
     -140      0       -220     120     10
     -280      0       0        120     5
     -140      0       220      120     10
     240       0       260      80      5
     320       0       -120     80      14
     80        0       -320     80      5
     -240      0       -260     80      14
     -340      0       80       80      5
     -80       0       320      80      14
     320       0       220      80      5
     300       0       -200     80      9
     0         0       -360     80      5
     -300      0       -220     80      9
     -360      0       160      80      5
     0         0       360      80      9
     -320      0       220      40      10
     260       0       -260     40      3
     -60       0       -360     40      10
     -360      0       -160     40      3
     360       0       160      40      10
     60        0       360      40      3
```

```
; BLACK..........0  RED............4  DARK_GRAY......8  LIGHT_RED......12
; BLUE...........1  MAGENTA........5  LIGHT_BLUE.....9  LIGHT_MAGENTA..13
; GREEN..........2  BROWN..........6  LIGHT_GREEN....10 YELLOW.........14
; CYAN...........3  LIGHT_GRAY.....7  LIGHT_CYAN.....11 WHITE..........15

;    X      Y      Z      DIA     COLOR
;  ----   ----   ----    ----    -----

     0      0      0      100      2      ; body
    80      0      0      100      2
   -80      0      0      100      2

    80    -40     40       20     14       ; legs
    80    -40    -40       20     14
    80    -55     50       20     14
    80    -55    -50       20     14
    80    -75     50       20     14
    80    -75    -50       20     14

     0    -40     40       20     14
     0    -40    -40       20     14
     0    -55     50       20     14
     0    -55    -50       20     14
     0    -75     50       20     14
     0    -75    -50       20     14

   -80    -40     40       20     14
   -80    -40    -40       20     14
   -80    -55     50       20     14
   -80    -55    -50       20     14
   -80    -75     50       20     14
   -80    -75    -50       20     14

   160      0      0      100      1       ; head

   220     20     10       20     14        ; eyes
   220     20    -10       20     14

   160     60     10       20     14        ; antenae
   160     60    -10       20     14
   160     80     20       20     14
   160     80    -20       20     14
   160     90     30       20     14
   160     90    -30       20     14
```